Conference Notes

By

Juixe

Friday, October 20

Welcome To RubyConf 2006

RubyConf 2006 began with a simple welcome from Rich Kilmer. There was no keynote presentation just a simple welcome speech. The speaker took a quick impromptu survey to see how many of those in attendance today where present at last year's RubyConf, and the year before that, and before that, etc. In the end only four people still had their hands up when the speaker asked who had attended the first RubyConf. The speaker noted, "great to see the growth in Ruby, not just in Rails."

It is interesting to note that only a few female programmers where in attendance; in fact I have only seen two. The speaker made an announcement that "they are flipping a lady's room for us, …, but they haven't told us which, so you have a 50% chance of getting slapped."

I had lunch with John Long, lead developer of Radiant content management system, and fellow Java and Rails developers. From the conversation, we all spoke of Radiant plugin development, Wordpress.com, self-employment, developing online services, and startups. Lunch felt, in terms of content, like presentation only more more personal. The conversations between sessions and in the hall are great. Rubyist hotly debate issues, such as documentation and the solution for the lack of documentation.

The History of Ruby

For this RubyConf 2006 session, the history of Ruby was broken down into five periods that include pre-history, ancient age, middle age, modern age, and contemporary.

The pre-history of Ruby was dated circa 1993 and it is an age of myth and epic legends, which only Matz and his friends know. According to the speaker, Masayoshi Takahashi, Ruby was baptized before it was born, that is to say, before any code had been written. According to Masayoshi-san, a normal name would lead to a normal language; a great name would make a great language. But in a bizarro world, Ruby could have been known as Coral. The name Coral was proposed in a chat session between Matz and an early contributor. But I feel that a rose in any other name would still smell as sweet and that a Ruby in any other name would still bring joy to program.

The ancient age of the Ruby programming language was carbon dated to 1995. At this time, Ruby 0.95 was available to the public. That is Japanese netnews users. At this time the ruby-list ML was launched with the first email: ruby-0.95 test failed.

The third period or middle age of Ruby was when this pure object oriented programming language began to spread in Japan. During this time, 2001-2002, about 20 Ruby related books where published in Japan, a sort of Ruby publishing bubble. In some sense, it seems that this bubble is now brewing here in the states with all the Ruby series books by the major technical

publishers in the works. It almost safe to say that somewhere from 5 to 10 percent of those in attendance has a book deal in the works. The book giveaways at the conference seemed like Ruby authors given books to fellow book authors.

The fourth period, the modern age, began around 2002 when Ruby spread throughout the world. During this period, developer passion drove Ruby's acceptance. The Pragmatic Programmer's Learning Ruby (aka PickAxe) book introduced Ruby to English speaking developers and many online technical sites provided articles on the language.

The firth period, the contemporary age (aka the Rails age), well we all know what happened in this period. Of this period the Masayoshi-san said, "Skip this period."

Rubinius - Hardcore Ruby

This was the second session of RubyConf 2006. I have to be honest, this session went a bit above my head, and from the 'glaze' of the audience, it went above a few of the other attendants. Well, it wasn't so much that it went above my head as it had no code and no demo. This was the second session of the day and I still had not had a 'WTF OMG' moment but this is of no fault of the speakers.

Rubinius is a really hardcore academic, experimental, proof of concept fork of Ruby. A key question asked by Evan Phoenix, the speaker and Seattle.rb memeber, was why would any sane developer do this? What is wrong with the current implementation? According to Evan, he started the Rubinius implementation because he found it to be fun. He also stated that RubyCLR and JRuby are pushing Ruby in new directions. In Rubinius, Evan, wants to provide a simple architecture, a simple core, a simple implementation. Evan stated that Rubinius is "so simple, it's almost naive." Rubinius is a partial port of the Smalltalk implementation provided by the Smalltalk blue book. Evan stated, "Genius stand on the shoulders of giants, not that I am a genius, I am just trying to stand on as many shoulders as possible."

Dynamic Graphics with Ruby

This RubyConf 2006 session provided a quick overview of a few libraries available to create and manipulate images in Ruby. Geoffrey Grosenbach, from the Rails Podcast, provided brief introduction to Scruffy, Gnuplot, MRPlot, PNG library, and Gruff Scene.

Scruffy is a pure Ruby library that produces attractive and powerful SVG graphs. MRPlot is a pure Ruby plotting library. The PNG is a pure Ruby library put out by the Seattle.rb peeps. I found Gruff Scene a bit interesting. Gruff Scence allows you to define different versions of a set of images which depending on some given state can combine a set of images into a scene that can visualize information. And of course you can always use RMagick, the Ruby interface to ImageMagick.

Now everybody has an opinion, and my opinion is that graphs are not sexy. A bar chart on top a line graph on top of a scatter plot is boring. But my hope is to use the pure Ruby PNG library to create some cellular automata, fractals, and other 'gnarly visualizations', as Professor Rudy Rucker would say.

Life After mkmf

Continuing with my coverage at RubyConf 2006, Kevin Clark presented on his Google Summer of Code project, mkrf. The intent of mkrf is to provide a rake replacement to mkmf. For those that don't know, mkmf is the current "defacto standard for building C extensions to Ruby." At the design level adheres to the kiss principle and will provide a simple and clear OO way to define Ruby extension make files. The work that went into mkrf will make it that much easier to develop Ruby extensions, which hopefully will provide a catalyst for more developers to get involved.

Iron Mongrel - Fuzzin

Zed Shaw, author of Mongrel, got an applause when introduced for this talk at RubyConf 2006. During this presentation, Zed talked about fuzzing. I am new to the ideas of fuzzing and I found this talk extremely interesting. One quick definition of fuzzing is to break a software system by handing it nasty maliciously crafted random input and events. One simple way of fuzzing is to always enter 'yes' on a console application and watch out for errors and exceptions. The basic theory behind fuzzing is that programmers will test using what they thing are valid inputs, but the programmer can't predict user input out in the world. Unit tests are predicatable since a developer wrote the system, knows what to expect, and wrote test based on that. A fuzz test has no idea of your system. According to Zed, fuzzing is a great way to finalized vendor selection of two closed source third party software packages.

Zed has made available a Ruby fuzz gem known as RFuzz. RFuzz is made up of a randomness engine, data collection, and statistical analysis. When fuzzing a software system, you will usually need to do some analysis of the results to verify if a fix actually worked. After this talk I felt that maybe I should try to use RFuzz in JRuby to try to blow up our Java application.

Radiant - Content Management Simplified

John Long, lead developer of Radiant CMS gave an in-depth overview of this Ruby on Rails driven Content Management System. Radiant is a lightweight CMS with a 'pristine interface' and basic elements such as pages, layout, and snippets. According to John, the design philosophy of Radiant is 'simplicity over features.' Basically, Radiant is 'a little bit more than a blogging engine.'

Radiant CMS uses a tag based template engine, unlike rails, which uses ERb in views. Radiant tags look like those provided by Struts taglibs. Radiant

tags provide the same functionality available in rails erb-based views, only that tags are more Dreamweaver/WYSIWYG friendly.

If you want to take a look at Radiant in action, take a look at Ruby homepage. According to John, Radiant is ideal for content oriented sites or small brochure-style sites. John is also a designer so he strives to make Radiant simple and easy to use not only for programmers but also more specifically for web designers. From my observations I have noticed that Rails and Radiant makes it easy for designers to program an application, but there is no tool that helps programmers design a site.

John spent time going over upcoming features in Radiant. The following features are in the works for future version: plugins, replacing behaviors with custom page types, full blogging (comments and tagging) support, search, workflow with roles, and maybe version control.

In the break after John's talk, I downloaded Radiant and got started. I found it easy to use and manage. I plan to use Radiant small sites that I have in the works.

Matz Roundtable

It was great to have Yukihiro 'Matz' Matsumoto, the language designer of Ruby, accessible throughout the RubyConf. I saw him throughout the conference during lunch, breaks, and after hours. During this session he answered questions from the crowd. To get the Q&A session started, Rich Kilmer jokingly asked the first question in behalf of DHH, "my girlfriend wants to know if you have a girlfriend and should I be worried about that?" DHH really didn't ask this question, it sounded like this was an inside joke from RailsConf 2006. Most of the important and serious questions asked by those in attendance where relating to backward compatibility issues in Ruby 1.9 and Ruby 2.0. Matz stated that "if you want a stable Ruby, use 1.8 forever." The key questions that resonated from the audience dealt with heavy implementation details and future features such as continuations, green threads, method lookups, noobie involvement in the C code for Ruby, mailing lists, feature requests/process, testing, and documentation. Regarding questions of feature requests, Matz made it understood that he is open to feature requests but that he would appreciate patches, prototypes, and code along with those feature requests.

Someone in the audience said something I thought it was funny. In effect someone commented that, "one reason I love Ruby is because I tried Python first."

Nick Sieger was also at the conference and he must know shorthand because he transcribed many of the questions and answers. Take a look at Nick's transcripts here:
http://blog.nicksieger.com/articles/2006/10/21/rubyconf-matz-roundtable

Saturday, October 21

Open Classes, Open Companies

Nathaniel Talbott, fearless leader at Terralien, drew some comparisons between Ruby's characteristics and the way you should go about starting and managing a software development outfit. According to Nathaniel, you are saying something when choosing Ruby as your programming language. I have heard other business folks make this same comment, as to say that you are a better developer if you pick a programming language not because its the current trend in job postings but because of your passion for the language. This same statement can be made of Python programmers. Nathaniel also stated that "rails has been a great catalyst" for designers/programmers to start their own company.

Here are Ruby's characteristics and how they should match to your business ethos:

Ruby's succinct expressions are one of its characteristics. According to Nathaniel, succinct is power. Succinct allows you to do more with less and as a small independent company you should strive to be succinct in your code, contracts, process, and mind set.

Reflection was another language characteristic in Ruby which to learn from. Programmers should constantly reflect and introspect on their role and process. A question to ask yourself is, how does your company/business reward or panelize reflection? Relating to reflection, Nathaniel suggests that people should not box themselves in their position or title.

The final characteristic that businesses should ideally learn from Ruby is its open class feature. In Ruby a class is never closed, any Ruby code can reopen a class implementation at any time, in any file. It is Nathaniel's belief that software entities should try to trade security with trust, close door politics with open communication. The speaker reminded the audience that legal agreements and contracts don't build trust. Nathaniel asked that business people ask themselves what is the default for information? Deny or allow?

A final piece of advice that I took away from this session was to put off decisions as long as possible, but not forever. In essence, don't make a final decision prematurely because it will cost you just the same as if you had made the decision too late.

Leveraging Mac OS X From Ruby

Laurent Sansonetti of Apple gave an awesome presentation on RubyOSA. RubyOSA is a Ruby/AppleEvent bridge that allows Ruby programs to interact and manipulate Apple applications in the same fashion that AppleScript can. Here is a code sample from the RubyOSA site:

```
require 'rbosa'
app = OSA.app_with_name('iTunes')
```

```
track = app.current_track
p track # -> #<OSA::Itunes::File_track:0x1495e20>
p track.name # -> "Over The Rainbow"
p track.artist # -> "Keith Jarrett"
p track.duration # -> 362
p track.date_added.to_s # -> "2006-06-30"
p track.enabled? # -> true
```

According to Laurent, Ruby has been shipped with OS X since 10.2, 'Jaguar' which buncled Ruby 1.6.7. Apple packages Ruby as a framework, which is easier for Mac development and allows versioning. In addition to Ruby, OS X includes RubyGems and gems like rake, rails and friends (mongrel, capistrano), libxml2, and sqlite3. Laurent mentioned that he is open to suggestions for adding including additional gems into OS X.

For sometime Apple has allowed Mac developers to programmatically control Mac applications via AppleScript. Now Ruby/Mac developers have that same power via RubyOSA. Laurent gave a powerful demo where he controlled his iTunes from Ruby's irb. He was able to play, pause, and stop a song. He was able to list his play iTunes playlists.

Laurent topped off his irb/iTunes demo by writing an Ruby/Cocoa UI application that controlled iTunes. Cocoa is a UI framework for building Mac applications, often with Object-C. Laurent described a Ruby/Object-C which allows Ruby developers to create Cocoa applications for the Mac. if you want to get started with prototyping Mac applications with Ruby direct your browser to RubyCocoa. Apple also has a lot of Cocoa reference material at their Cocoa developer site.

Lighting Talks

Because Josh Susser was unavailable for his RubyConf session, the organizers opened the time slot to the audience members, allowing each 5 to 10 minutes for demos or presentations. Here are some of the noteworthy presentations from this time slot.

Someone from NOAA presented on Ruby Queue. Ruby Queue is a tool for building linux clusters. The speaker recommended that Ruby Queue be used in small research teams with between 5 to 30 nodes. Here is a description of Ruby Queue from its website, "ruby queue (rq) is a tool used to create instant linux clusters by managing sqlite databases as nfs mounted priority work queues"

Someone else presented on irb tips and tricks. One great tip was a short script that allows irb to maintain a shell-like history. If you are like most Rubyist, you use the irb console for a lot of prototyping. Make your life a bit easier and trick out your irb. The Tips and Tricks page listed above mentions several other useful tips such as tab completion.

Andre Lewis gave a small and powerful presentation on a Google Maps zoom control he calls GZoom. GZoom allows users to draw a rectangle on a Google map by dragging the cursor. When the cursor is released, Google Maps will zoom into that rectangle.

Another great presentation was by a Seattle.rb member on Hoe. Hoe is a rake helper that helps to generate rdocs, runs tests, and build the project packaging.

I18N, M17N, Unicode And All That

Tim Bray of Sun Microsystems gave an insightful presentation on Internationalization (I18N), Multilingualization (M17N), and unicode. Even Tim who has spent most of his career in unicode support admitted, "for some this is not the most stimulating subject." Tim asked the audience of programmers, "Why do we care about internationalization?" The reason is not so obvious to native English speakers but the answer can be found online. English is no longer the predominant language of the internet. A software project should think about localization (L10N) and internationalization from the onset. According to Tim, it doesn't make much sense to develop and application that does not support I18N, M17N, and L10N.

Tim stated that if you had the following regular expression piece of code, it is probably a bug:

```
/[a-zA-Z]+/
```

Many rails applications have code like this. Perhaps worst yet is that rails heavily depends on Ruby string methods such as capitalize, upcase, downcase, etc. Tim noted that functions not all unicode characters can be capitalize and often some unicode strings will choke one of these functions are applied on them. In addition to capitalize, upcase, and downcase Ruby has I18N issues with the swapcase, match, size, strip, ==, =~, [], eql?, and more…

Tim suggested that to better support internationalization and unicode Ruby programmers should avoid case folding, that is the use of capitalize. According to Tim, case folding routinely provides the wrong answers and should be avoided.

Tim ended this discussion by saying, "I want Ruby to be a good citizen of the world." If you want to take a look through Tim's presentation, it can be found.

Speak My Language

Michael Granger gave a RubyConf 2006 presentation on Natural Language Processing and Natural Language Generation in Ruby. Michael gave some interesting demos where he was able to break a sentence into its grammatical building blocks such as subject, verb, and noun. He was also able to translate a sentence into a more generic version of itself. This session was a personal favorite although I don't have a current need for language processing libraries.

Here is a list of Ruby libraries, which Michael recommends:

Stemmable - an implementation of Porter Stemming Algorithm.

Chronic - a natural language date/time parser written in pure Ruby.

Ruby WordNet - Ruby WordNet is a Ruby interface to the WordNet Lexical Database.

Ruby LinkParser - Ruby port of the perl module Lingua::LinkParser used to determine the structure of a sentence.

Ruby Linguistics - A Ruby framework that integrates Ruby WordNet, Ruby LinkParser. Here is some code of Ruby Linguistics in use:

```
require 'linguistics'
Linguistics::use(:en) # extends Array, String, and Numeric

# Pluralization
puts "box".en.plural   # => "boxes"
puts "mouse".en.plural # => "mice"

# Present Participles
"runs".en.present_participle # => "running"
"eats".en.present_participle # => "eating"

# Ordinal Numbers
puts 5.en.ordinal    # => "5th"
puts 2004.en.ordinal # => "2004th"

# Quantification
puts "cow".en.quantify(5)    # => "several cows"
puts "cow".en.quantify(1005) # => "thousands of cows"
```

Matz Keynote

Yukihiro 'Matz' Matsumoto, the language designer of Ruby, gave the keynote speech for RubyConf 2006. The keynote was titled The Return of the Bikeshed or Nuclear Plant in the Backyard. I didn't know what the bike shed and nuclear power plant in the title referred to. I later discovered that is refers to Parkinson's Law which in essence states that the least important the decision the more people have an opinion about it, because most people don't have solutions for hard problems. The title of the keynote set the theme for the night.

According to Matz, Ruby is a programming, scripting, lightweight, and dynamic language. Matz noted the difference between some of these. He stated that the term lightweight language is popular in Japan where as scripting language is detested in the states. Yet Ruby is all of these and more…

Matz then went on to described the four core values of the Agile Manifesto and how they related to language design. The first value is Individuals and Interactions, which so far as language design means a strong focus on developers. The second value is Working Software, which means that the language should encourage readability. Collaboration Over Contracts refers to an expressive language that helps in documentation/communication. And finally, the last Agile value is Responding to Change, a language should

be able to embrace change. With this in values of Agile Manifesto in mind, Matz said that therefore Ruby is an Agile Language.

Matz also described the good, the bad, and the ugly in Ruby. He stated that the good was the language itself, Ruby on Rails, and the people (Martin Fowler said "because Matz is nice."). Matz said that the ugly of Ruby was eval.c and parse.y. The bad was Ruby 2. Ruby 2 has been vaporware since before Perl 6.

Matz then went on to say that Ruby is sometimes perceived as a Fragile Language because of Ruby's bike shed issues. That is to say, the amount of people arguing about bikes shed size issues while leaving complex problem to experts. As bike shed issues, Matz mentioned having the Symbol class extend String, #lines, and removing private and protected. Because of the noise over these bike shed issues people might suggest that Ruby is a Fragile Language.

Earlier in the conference, during his roundtable session, Matz stated that "if you want a stable Ruby, use 1.8." During the keynote Matz said, and I quote, "If I die, keep 1.8 for ever." If Ruby 1.8 is good enough for Matz, it's good enough for me.

First Annual RejectConf

Ryan Davis and Jacob Harris came up with a fantastic idea of holding a RejectConf after the scheduled RubyConf sessions. RejectConf was held Saturday night after Matz keynote and was made up of 5 to 10 minutes talks by rubyist whose session proposal was shot down by the RubyConf selection committee. RejectConf was definitely a high light of the conference and thanks to however brought the three cases of beer!!!

The first reject of the night was Steven Baker who gave a top ten list of why not to use RSpec. It seems that RSpec does not get any respect, in a very Rodney Dangerfield-like way. According to Steven, if you use RSpec that is great, if you don't that is better yet. Here is the first reason why you should not use RSpec, Steven was drunk when he wrote it. The next reason, Documentation is overrated. Moving down the list to the second reason not to use RSpec, tests that don't read anything like English prevent customers from pointing out glaring errors in business logic. And the top reason not to use RSpec, because explaining to new members of your team that you really can test something that doesn't exist is fun.

Another interesting RejectConf was regarding RubyOSA. Earlier in the day there was a similar talk by an Apple engineer descriving how to leverage Mac OS X from Ruby. This talk demonstrated a small RubyOSA program that resized all application windows opened in a Mac. According to the speaker, this hack is useful when moving from a big monitor at work to a small screen at home.

Another great RejectConf talk was by Charles Nutter. Charles is a developer on the JRuby project and is currently working at Sun. Charles gave a 'shock and awe' demo of prototyping a Swing app using JRuby. From the irb console he dynamically created a Swing application. Charles also showed off some early alpha JRuby context help, refactoring support in NetBeans.

According to Charles, some of the NetBeans work to support JRuby is being developed by JavaPosse member Tor Norbye.

Perhaps one of the best RejectConf talk was the skit enacted by Adam Keys. I am not going to say much about Adam's skit except that it was a cross been Star Trek and Python, Ruby, and Java language wars. I will say that the Federation were played Rubyists and I think that the Borg were Java developers. Hilarious. See a video of the skit at TopFunky.

Sunday, October 22

Streamlined: A Framework for Data-centric Web Application

Justin Gehtland of Relevance gave a RubyConf session titled Streamlined: A Framework for Data-centric Web Applications. Streamlined is a Ruby on Rails application that is meant to be used in the back-end to manage and administer application data. According to Justin, these back-end administrative views are repetitive and similar, usually with a login, tabular data, and CRUD support. Justin was tired of hacking these administration views for clients so his company developed Streamlined. Right out the box, Streamlined supports authorization, pagination, user preferences, and in a future release it will support role based authorization. Justin mentions that Streamlined goes beyond scaffolding and provides a feature rich ajax powered relationship management. As he stated, "To quote one of our preeminent thinkers of our time, Paris Hilton, 'that's HOT!'" Streamlined uses ajax views by default but this can be easily changed. In most situations all you need is to replace the CSS to get started with Streamlined as it provides the 80% of a clients' requirements. For the remaining 20%, Streamlined allows custom views and has a great declarative DSL for modifying the default behavior.

Speaking of declarative Domain Specific Languages, Justin went on a rant about using Ruby for configuration files. He suggested that when converting libraries from other programming environments we Rubyist should not port their dependence on XML. If possible we should even avoid using yml files for configuration. Justin suggested we use Ruby code in a DSL-like fashion for configuring tools and applications. Justin reminded the audience that the benefit of XML is that it is human readable plain text with markups. Ruby based DSL configuration files are plain text and often read like English.

You Got Your Ruby In My CLR

For the last session of RubyConf 2006, John Lam gave a talk by the name "You got your Ruby in my CLR!" RubyCLR is "a high-performance Ruby to .NET bridge that allows seamless integration of CLR and Ruby objects in the same Win32 process." RubyCLR started as a tradition. John has a three-year-old son and for his first birthday John wanted to give him something other than plastic toys. For his son's first birthday, John wrote a python program. John said that this was a way for him to "hack on family time." For

his son's second birthday, John wanted to write a Ruby program that interoperated with Microsoft/CLR libraries for a flash card like game.

John had some code samples of how of using RubyCLR to develop a Windows Forms application. RubyCLR is open to C# hacks, it allows inline C# code in Ruby files! RubyCLR makes .NET suck less. John also talked about another scenario for using RubyCLR in a .NET environment. He stated that XAML is just too verbose and that with RubyCLR developer can use the XML Builder approach to building Avalon UI applications.

If you want to adopt a technology like RubyCLR, or JRuby for that matter, John suggests that you introduce it in non-threating tasks such as testing or administrative scripting. Another interesting quote by John was when he said "everything in life is easy until you turn on security." He made this comment when asked about how he deals with opening a Ruby class when it has passed the verifier. I am sure he will come up with an interesting solution to issues like security, especially now that John Lam has been hired by Microsoft to continue his work with RubyCLR.

WrapUp

I started to list the session that I liked the most and I soon noticed that I was listing the whole RubyConf schedule! I learned a lot from the all the speakers and from the questions of the audience. I also have to mention that RubyConf is a more intimate conference than say JavaOne. At JavaOne, you are easily lost amongst the 15,000 developers and and marketers in attendance. RubyConf 2006 had over 300 hard-core Rubyist in attendance, and me. Nathanial Talbott, of Terralien, reminded the audience that the first RubyConf, back in 2001, had roughly 30 participants and perhaps one or two of those where paid ruby practitioners.

Oh, I forgot to mention some Rails tips that I received during breakfast of the last day of the conference. I had breakfast with some rails developers from BraveNet. I asked them if they give tell me the key hint or tip that a new rails developers should be aware of. I was told told to never ever use namespaces in models, and that I should seriously think about using namespaces in controllers. Another general rails tip was to use routes to make pretty urls, as opposed to namespace controllers.

RubyConf was a great experience and I can't waint till next year!

Copyright