Memcached 原理和使用详解

作者：heiyeluren(黑夜路人)

博客：http://blog.csdn.net/heiyeshuwu

2009-01

Tech Talk 目录索引

- Memcached介绍
- Memcached安装和使用
- 一些技巧
- Q&A

Memcached介绍：

什么是Memcached?

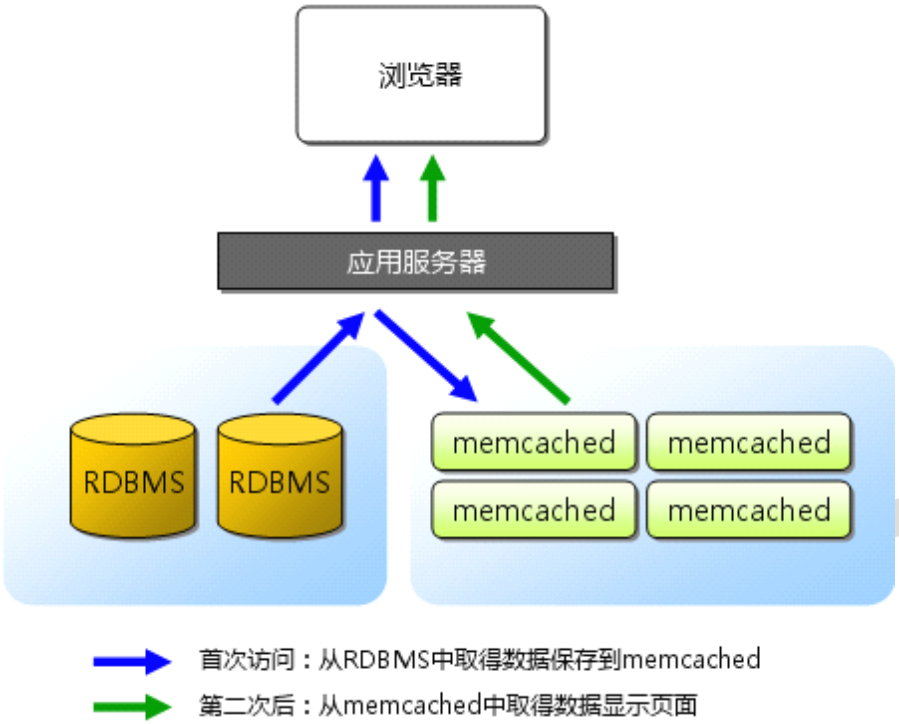
Memcached是国外社区网站 LiveJournal 的开发团队开发的高性能的分布式内存缓存服务器。一般的使用目的是，通过缓存数据库查询结果，减少数据库访问次数，以提高动态Web应用的速度、提高可扩展性。

LiveJournal 团队开发了包括 **Memcached**、**MogileFS**、**Perlbal** 等不错的开源项目。

官方网站：**http://www.danga.com/memcached/**

Memcached介绍

Memcached运行图



Memcached介绍

谁在用Memcached?

国外

国内

Memcached介绍

与Memcached类似的还有什么？

国外

Tokyo Cabinet：http://tokyocabinet.sourceforge.net/index.html（日本mixi.jp公司开发）

国内

MemcacheDB：http://memcachedb.org（新浪开源Team开发）

tmcache: http://heiyeluren.googlecode.com（偶开发的 ^_^）

Memcached介绍

Memcached的主要特点

- 基于C/S架构，协议简单
- 基于libevent的事件处理
- 自主内存存储处理
- 基于客户端的Memcached分布式

Memcached介绍

基于C/S架构，协议简单

```
hed-1.2.0
0]#
0]# memcached -d -m 10  -u root -l 192.168.0.200 -p 12001 -P /tmp/mem.pid -vv
w 268435456
n not implemented
0]# slab class   1: chunk size     80 perslab 13107
  100 perslab 10485
  128 perslab  8192
  160 perslab  6553
```

```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\heiyeluren>telnet 192.168.0.200 12000
```

```
Telnet 192.168.0.200
set abc 0 60 10
1234567890
STORED
get abc
VALUE abc 0 10
1234567890
END
replace abc 0 120 10
abcdefgjhe
STORED
get abc
VALUE abc 0 10
abcdefgjhe
END
delete abc
DELETED
get abc
END
set abc 0 60 10
0987654321
STORED
flush_all
OK
get abc
END
```

Memcached介绍

基于libevent的事件处理

libevent是一套跨平台的事件处理接口的封装，能够兼容包括这些操作系统：Windows/Linux/BSD/Solaris 等操作系统的的事件处理。

包装的接口包括：
poll、select(Windows)、epoll(Linux)、kqueue(BSD)、/dev/pool(Solaris)

Memcached 使用libevent来进行网络并发连接的处理，能够保持在很大并发情况下，仍旧能够保持快速的响应能力。

libevent: http://www.monkey.org/~provos/libevent/

Memcached介绍

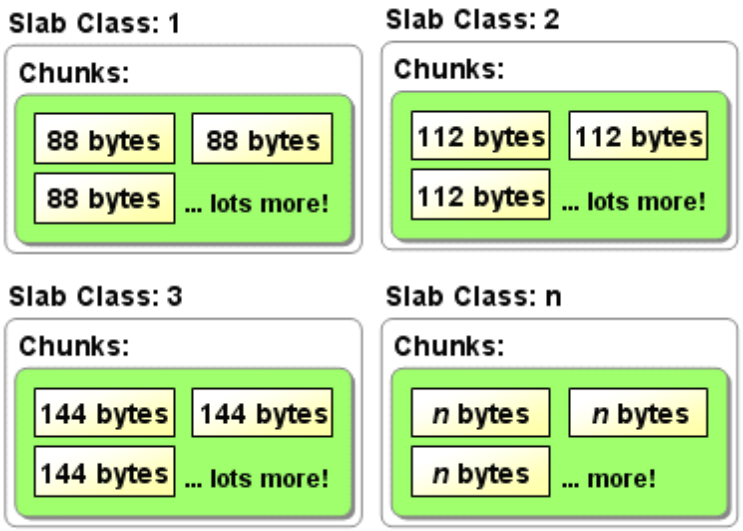
自主的内存存储处理

- 数据存储方式：Slab Allocation
- 数据过期方式：Lazy Expiration + LRU

Memcached介绍

数据存储方式：Slab Allocation

Slab Alloction 构造图



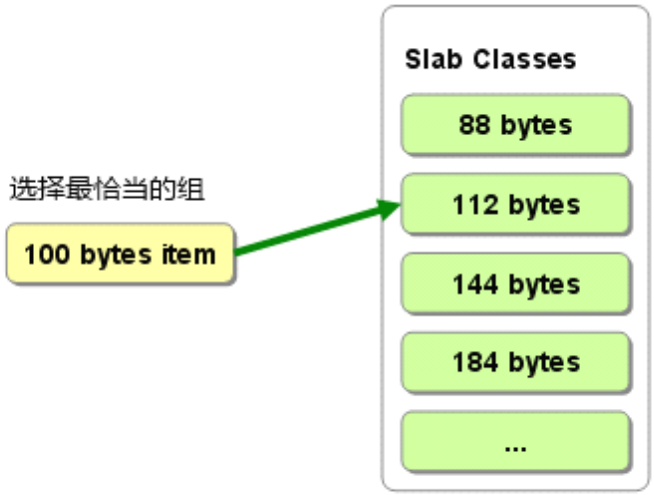
Slab Allocator的基本原理是按照预先规定的大小，将分配的内存分割成特定长度的块，以完全解决内存碎片问题。

Slab Allocation的原理相当简单。 将分配的内存分割成各种尺寸的块（chunk），并把尺寸相同的块分成组（chunk的集合）

Memcached介绍

数据存储方式：Slab Allocation

Slab Classes 分配图



Page：分配给Slab的内存空间，默认是1MB。分配给Slab之后根据slab的大小切分成chunk。

Chunk：用于缓存记录的内存空间。

Slab Class：特定大小的chunk的组。

memcached根据收到的数据的大小，选择最适合数据大小的slab。
memcached中保存着slab内空闲chunk的列表，根据该列表选择chunk，然后将数据缓存于其中。

Memcached介绍：

数据存储方式：Slab Allocation

Slab Alloction 缺点

这个问题就是，由于分配的是特定长度的内存，因此无法有效利用分配的内存。例如，将100字节的数据缓存到128字节的chunk中，剩余的28字节就浪费了。

Memcached介绍:

数据过期方式

- Lazy Expiration

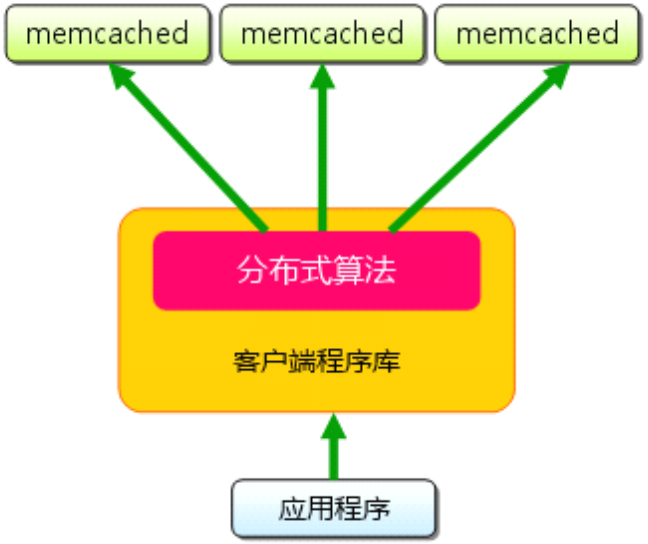
memcached内部不会监视记录是否过期，而是在get时查看记录的时间戳，检查记录是否过期。这种技术被称为lazy（惰性）expiration。因此，memcached不会在过期监视上耗费CPU时间。

- LRU

memcached会优先使用已超时的记录的空间，但即使如此，也会发生追加新记录时空间不足的情况，此时就要使用名为 Least Recently Used（LRU）机制来分配空间。顾名思义，这是删除“最近最少使用”的记录的机制。因此，当memcached的内存空间不足时（无法从slab class 获取到新的空间时），就从最近未被使用的记录中搜索，并将其空间分配给新的记录。从缓存的实用角度来看，该模型十分理想。

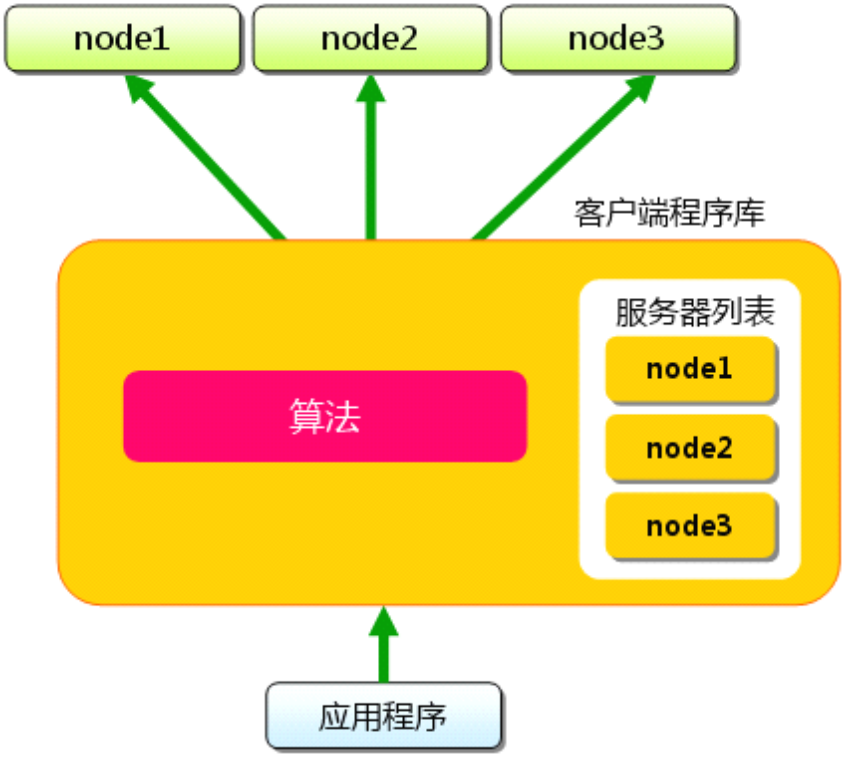
Memcached介绍:

基于客户端的Memcached分布式



Memcached介绍:

基于客户端的Memcached分布式

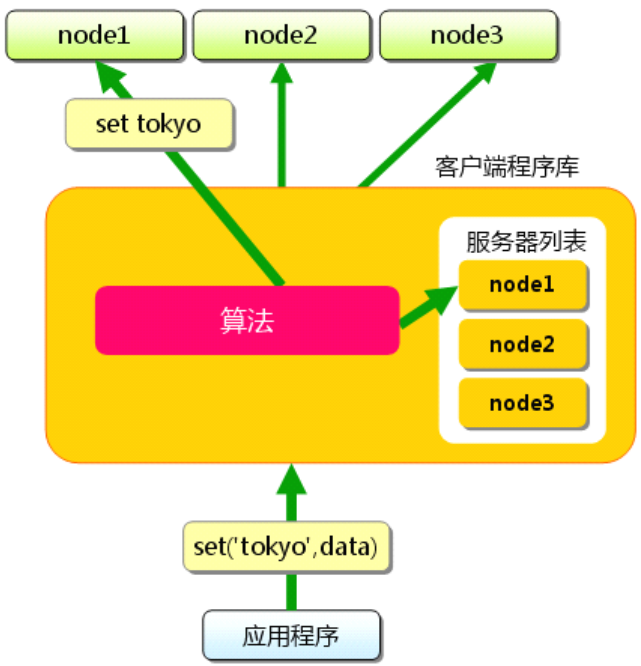



```
//按照Key值，获取一个服务器ID
int getServerId(char *key, int serverTotal) {
    int c, hash = 0;
    while (c = *key++) {
        hash += c;
    }
    return hash % serverTotal;
}


//服务器列表
node[0] => 192.168.0.1:11211
node[1] => 192.168.0.2:11211
node[2] => 192.168.0.3:11211

//获取key是tokyo的节点ID(服务器ID)
int id = getServerId("test", 3);

//得出的结果是1，那么对应的机器就是
node[id] == node[1]
```

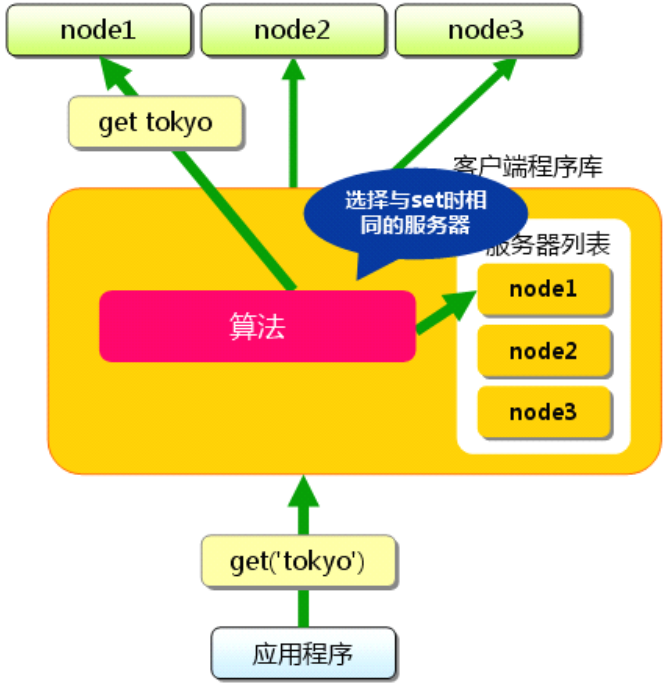
Memcached介绍：

基于客户端的Memcached分布式

写入操作



读取操作



Memcached安装和使用:

- Memcached 安装
- Memcached 与 PHP 结合使用
- Memcached 与 C/C++ 结合使用

Memcached安装和使用：

Memcached 安装

安装步骤：

- 先安装 **libevent**
- 再安装 **Memcached** 主程序

源码下载：**(最新版)**

libevent官网：http://monkey.org/~provos/libevent/
libevent下载：http://monkey.org/~provos/libevent-1.4.9-stable.tar.gz

Memcached官网：http://www.danga.com/memcached
Memcached下载：http://www.danga.com/memcached/dist/memcached-1.2.6.tar.gz

Memcached安装和使用：

Memcached 安装

- 安装 **libevent**

```
# tar zxvf libevent-1.4.9-stable.tar.gz
# cd libevent-1.4.9-stable
# ./configure --prefix=/usr
# make
# make install
```

- 安装 **Memcached**

```
# tar zxvf memcached-1.2.6.tar.gz
# cd memcached-1.2.6
# ./configure --prefix=/usr/local
# make
# make install
```

Memcached安装和使用：

Memcached 运行

- 试运行 **Memcached**

/usr/local/bin/memcached -u hualiangxie

```
root@AD_38_220_sles10:~/memcached/memcached-1.2.6# ls -l /usr/lib/libevent*
lrwxrwxrwx 1 root     21 2008-11-20 20:31 /usr/lib/libevent-1.1.so.1 -> libevent-1.1.so.1.0.2*
-rwxrwxrwx 1 root  28588 2006-06-16 21:38 /usr/lib/libevent-1.1.so.1.0.2*
lrwxrwxrwx 1 root     21 2009-01-05 18:53 /usr/lib/libevent-1.4.so.2 -> libevent-1.4.so.2.1.2*
-rwxr-xr-x 1 root 304885 2009-01-05 18:53 /usr/lib/libevent-1.4.so.2.1.2*
-rw-r--r-- 1 root 388506 2009-01-05 18:53 /usr/lib/libevent.a
lrwxrwxrwx 1 root     26 2009-01-05 18:53 /usr/lib/libevent_core-1.4.so.2 -> libevent_core-1.4.so.2.1.2*
-rwxr-xr-x 1 root 109624 2009-01-05 18:53 /usr/lib/libevent_core-1.4.so.2.1.2*
-rw-r--r-- 1 root 147538 2009-01-05 18:53 /usr/lib/libevent_core.a
-rwxr-xr-x 1 root   1009 2009-01-05 18:53 /usr/lib/libevent_core.la*
lrwxrwxrwx 1 root     26 2009-01-05 18:53 /usr/lib/libevent_core.so -> libevent_core-1.4.so.2.1.2*
lrwxrwxrwx 1 root     27 2009-01-05 18:53 /usr/lib/libevent_extra-1.4.so.2 -> libevent_extra-1.4.so.2.1.2*
-rwxr-xr-x 1 root 244351 2009-01-05 18:53 /usr/lib/libevent_extra-1.4.so.2.1.2*
-rw-r--r-- 1 root 302478 2009-01-05 18:53 /usr/lib/libevent_extra.a
-rwxr-xr-x 1 root   1016 2009-01-05 18:53 /usr/lib/libevent_extra.la*
lrwxrwxrwx 1 root     27 2009-01-05 18:53 /usr/lib/libevent_extra.so -> libevent_extra-1.4.so.2.1.2*
-rwxr-xr-x 1 root    974 2009-01-05 18:53 /usr/lib/libevent.la*
lrwxrwxrwx 1 root     21 2009-01-05 18:53 /usr/lib/libevent.so -> libevent-1.4.so.2.1.2*
root@AD_38_220_sles10:~/memcached/memcached-1.2.6# ls -l /usr/local/bin/memcached*
-rwxr-xr-x 1 root 123871 2009-01-05 18:55 /usr/local/bin/memcached*
-rwxr-xr-x 1 root 132288 2009-01-05 18:55 /usr/local/bin/memcached-debug*
root@AD_38_220_sles10:~/memcached/memcached-1.2.6# /usr/local/bin/memcached -u hualiangxie
```

Memcached安装和使用：

Memcached 运行

查看**Memcached** 帮助信息

/usr/local/bin/memcached -h

```
root@AD_38_220_sles10:~/memcached/memcached-1.2.6# /usr/local/bin/memcached -h
memcached 1.2.6
-p <num>      TCP port number to listen on (default: 11211)
-U <num>      UDP port number to listen on (default: 0, off)
-s <file>     unix socket path to listen on (disables network support)
-a <mask>     access mask for unix socket, in octal (default 0700)
-l <ip_addr>  interface to listen on, default is INDRR_ANY
-d            run as a daemon
-r            maximize core file limit
-u <username> assume identity of <username> (only when run as root)
-m <num>      max memory to use for items in megabytes, default is 64 MB
-M            return error on memory exhausted (rather than removing items)
-c <num>      max simultaneous connections, default is 1024
-k            lock down all paged memory.  Note that there is a
              limit on how much memory you may lock.  Trying to
              allocate more than that would fail, so be sure you
              set the limit correctly for the user you started
              the daemon with (not for -u <username> user;
              under sh this is done with 'ulimit -S -l NUM_KB').
-v            verbose (print errors/warnings while in event loop)
-vv           very verbose (also print client commands/reponses)
-h            print this help and exit
-i            print memcached and libevent license
-b            run a managed instanced (mnemonic: buckets)
-P <file>     save PID in <file>, only used with -d option
-f <factor>   chunk size growth factor, default 1.25
-n <bytes>    minimum space allocated for key+value+flags, default 48
root@AD_38_220_sles10:~/memcached/memcached-1.2.6#
```

Memcached安装和使用：

Memcached 运行

关注基本选项

-p <num>	监听的TCP端口 (缺省: 11211)
-d	以守护进程方式运行Memcached
-u <username>	运行Memcached的账户，非root用户
-m <num>	最大的内存使用, 单位是MB，缺省是 64 MB
-c <num>	软连接数量, 缺省是 1024
-v	输出警告和错误信息
-vv	打印客户端的请求和返回信息
-h	打印帮助信息
-i	打印memcached和libevent的版权信息

运行 **Memcached**

目标：使用11211端口、hualiangxie用户、最大占用512M内存、1024个软连接，输出客户端请求，以守护进程方式运行

```
# /usr/local/bin/memcached -p 11211 -d -u hualiangxie -m 512 -c 1024 -vvv
```

Memcached安装和使用：

Memcached 运行

检查是否正常启动

 # pa auxxww | grep memcached

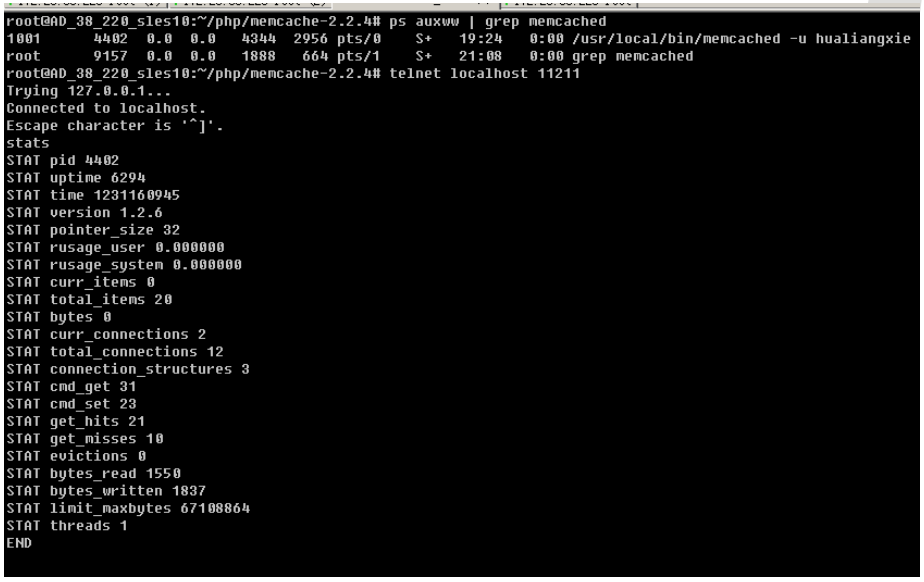
```
1001     4402  0.0  0.0   2296   900 pts/0    S+   19:24   0:00 /usr/local/bin/memcached -u hualiangxie
 root     4547  0.0  0.0   1892   668 pts/3    S+   19:42   0:00 grep memcached
```

 # telnet localhost 11211

```
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
```

stats

```
STAT pid 4402
STAT uptime 1032
STAT time 1231155683
STAT version 1.2.6
STAT pointer_size 32
...
END
```



Memcached安装和使用：

Memcached 基本协议

数据存取

set key1 0 180 3
abc
STORED
add key1 0 180 3
xyz
NOT_STORED
get key1
VALUE key1 0 3
abc
END
replace key1 0 180 3
xyz
STORED
get key1
VALUE key1 0 3
xyz
END
delete key1
DELETED

数字加减

set key2 0 180 4
1234
STORED
incr key2 3
1237
get key2
VALUE key2 0 4
1237
END
decr key2 1
1236
get key2
VALUE key2 0 4
1236
END

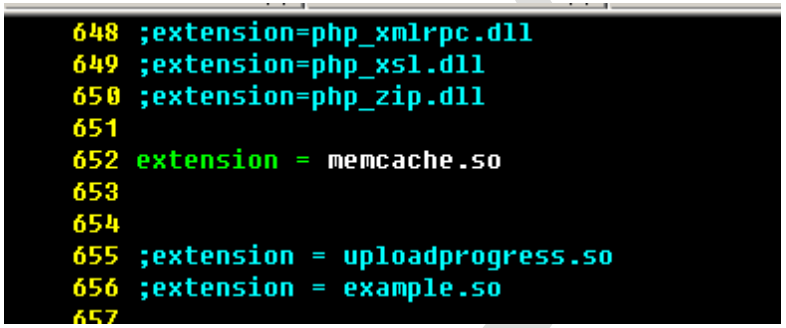
```
root@AD_38_220_sles10:~# telnet localhost 11211
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
set key1 0 180 3
abc
STORED
add key1 0 180 3
xyz
NOT_STORED
get key1
VALUE key1 0 3
abc
END
replace key1 0 180 3
xyz
STORED
get key1
VALUE key1 0 3
xyz
END
delete key1
DELETED
set key2 0 180 4
1234
STORED
incr key2 3
1237
get key2
VALUE key2 0 4
1237
END
decr key2 1
1236
get key2
VALUE key2 0 4
1236
END
quit
Connection closed by foreign host.
```

Memcached安装和使用：

Memcached 和 PHP 结合使用

安装 PHP Memcache 扩展

扩展官网：http://pecl.php.net/package/memcache
扩展下载：http://pecl.php.net/get/memcache-2.2.4.tgz



Memcache扩展安装：
tar zxvf memcache-2.2.4.tgz
cd memcache-2.2.4
/usr/local/php/bin/phpize
./configure --with-php-config=/usr/local/php/bin/php-config
make
make install

配置
ls -l /usr/local/php/lib/php/extensions/no-debug-non-zts-20060613/memcache.so
vim /usr/local/php/lib/php.ini
新增配置内容：
extension_dir = "/usr/local/php/lib/php/extensions/no-debug-non-zts-20060613/"
extension = memcache.so

检查安装结果
/usr/local/php/bin/php -m
/usr/local/apache2/bin/apachectl restart

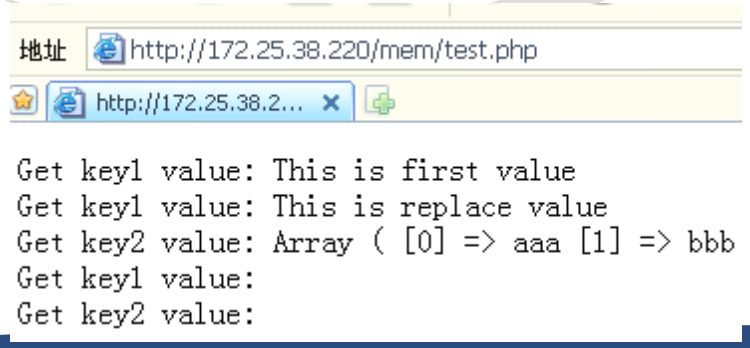
Memcached安装和使用：

Memcached 与 PHP 结合使用

PHP与Memcache结合测试代码

```
<?php
//连接Memcache
$mem = new Memcache;
$mem->connect("localhost", 11211);

//保存数据
$mem->set('key1', 'This is first value', 0, 60);
$val = $mem->get('key1');
echo "Get key1 value: " . $val ."<br>";

//替换数据
$mem->replace('key1', 'This is replace value', 0, 60);
$val = $mem->get('key1');
echo "Get key1 value: " . $val . "<br>";

//保存数组数据
$arr = array('aaa', 'bbb', 'ccc', 'ddd');
$mem->set('key2', $arr, 0, 60);
$val2 = $mem->get('key2');
echo "Get key2 value: ";
print_r($val2);
echo "<br>";
```

```
//删除数据
$mem->delete('key1');
$val = $mem->get('key1');
echo "Get key1 value: " . $val . "<br>";

//清除所有数据
$mem->flush();
$val2 = $mem->get('key2');
echo "Get key2 value: ";
print_r($val2);
echo "<br>";

//关闭连接
$mem->close();
?>
```



Memcached安装和使用：

Memcached 与 PHP 结合使用

PHP与Memcache分布式

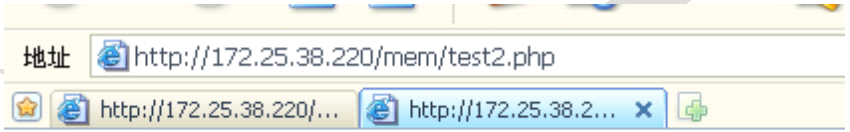
在一台或者多台机器启用一个或者多个进程，这里是在一台机器启用两个进程，使用两个端口：

 # /usr/local/bin/memcached -p 11211 -d -u hualiangxie
 # /usr/local/bin/memcached -p 11212 -d -u hualiangxie

PHP 测试代码

```
<?php
//连接Memcache
$mem = new Memcache;
$mem->addServer("localhost", 11211);
$mem->addServer("localhost", 11212);

//保存数据
$mem->set('key1', 'This is first value', 0, 60);
$val = $mem->get('key1');
echo "Get key1 value: " . $val ."<br>";

//保存数组数据
$arr = array('aaa', 'bbb', 'ccc', 'ddd');
$mem->set('key2', $arr, 0, 60);
$val2 = $mem->get('key2');
echo "Get key2 value: ";
print_r($val2);
echo "<br>";

//删除数据
$mem->delete('key1');
$val = $mem->get('key1');
echo "Get key1 value: " . $val . "<br>";

//关闭连接
$mem->close();
?>
```

注意：实际上Key1保存在11211端口机器，Key2保存在11212端口机器上




```
Get key1 value: This is first value
Get key1 value: This is replace value
Get key2 value: Array ( [0] => aaa [1] => bbb [2] =>
Get key1 value:
```

Memcached安装和使用：

Memcached 和 C/C++ 结合使用

安装 C/C++ Memcached 客户端库：libmemcached

开发库官网：http://tangent.org/552/libmemcached.html
开发库下载：http://download.tangent.org/libmemcached-0.25.tar.gz

libmemcached库安装：

```
# tar zxvf libmemcached-0.25.tar.gz
# cd libmemcached-0.25
# ./configure --prefix=/usr
# make
# make install
```

检查安装结果

```
# ls /usr/lib/libmemcache*          //库文件
# ls /usr/include/libmemcached/*  //头文件
# ls /usr/bin/mem*                  //命令行工具
```

参考 libmenmcached 开发示例代码

```
# man libmemcached_examples
```

Memcached安装和使用：

Memcached 与 C/C++ 结合使用

C/C++与Memcached结合测试代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <libmemcached/memcached.h>

int main(int argc, char *argv[]) {
    memcached_st *memc;
    memcached_return rc;
    memcached_server_st *servers;
    char value[8191];

    //connect server
    memc = memcached_create(NULL);
    servers = memcached_server_list_append(NULL, "localhost",
11211, &rc);
    rc = memcached_server_push(memc, servers);
    memcached_server_free(servers);

    //Save data
    strcpy(value, "This is c first value");
    rc = memcached_set(memc, "key1", 4, value, strlen(value),
(time_t)180, (uint32_t)0);
    if (rc == MEMCACHED_SUCCESS) {
        printf("Save key:key1 data:\"%s\" success.\n", value);
    }

    //Fetch data
    char return_key[MEMCACHED_MAX_KEY];
    size_t return_key_length;
    char *return_value;
    size_t return_value_length;

    char *keys[]= {"key1"};
    size_t key_length[]= {4};
    uint32_t flags;

    rc = memcached_mget(memc, keys, key_length, 1);
    return_value = memcached_fetch(memc, return_key,
&return_key_length, &return_value_length, &flags, &rc);
    if (rc == MEMCACHED_SUCCESS) {
        printf("Fetch key:%s data:%s\n", return_key, return_value);
    }

    //Delete data
    rc = memcached_delete(memc, "key1", 4, (time_t)0);
    if (rc == MEMCACHED_SUCCESS) {
        printf("Delete Key key1 success.\n");
    }

    //free
    memcached_free(memc);
    return 0;
}
```

Memcached安装和使用：

Memcached 与 C/C++ 结合使用

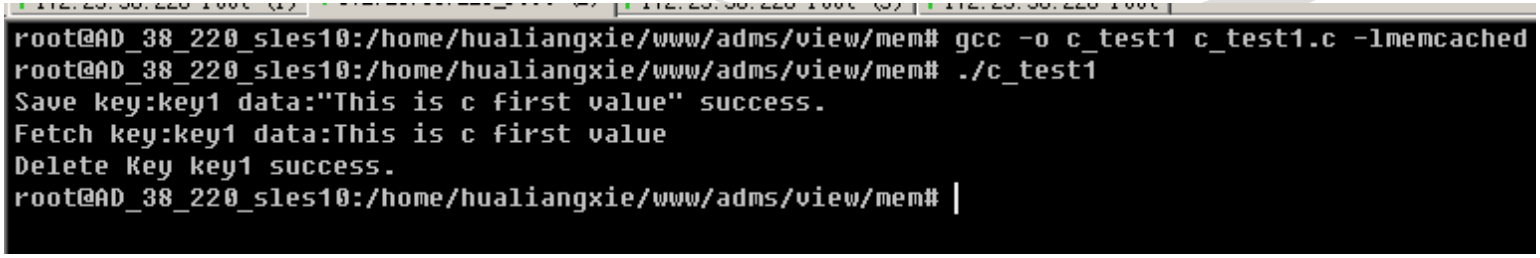
C/C++与Memcached结合测试结果

编译执行以上代码：

```
# gcc -o c_test1 c_test1.c -lmemcached
# ./c_test1
```

输出结果：

```
Save key:key1 data:"This is c first value" success.
Fetch key:key1 data:This is c first value
Delete Key key1 success.
```



Memcached安装和使用：

Memcached 与 C/C++ 结合使用

C/C++与Memcached分布式结合测试代码1

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <libmemcached/memcached.h>

int main(int argc, char *argv[]) {
    memcached_st *memc;
    memcached_return rc;
    memcached_server_st *servers;
    char value[8191];

    //connect multi server
    memc = memcached_create(NULL);
    servers = memcached_server_list_append(NULL, "localhost", 11211, &rc);
    servers = memcached_server_list_append(servers, "localhost", 11212, &rc);
    rc = memcached_server_push(memc, servers);
    memcached_server_free(servers);
```

```
//Save multi data
    size_t i;
    char *keys[]= {"key1", "key2", "key3"};
    size_t key_length[]= {4, 4, 4};
    char *values[] = {"This is c first value", "This is c
second value", "This is c third value"};
    size_t val_length[]= {21, 22, 21};

    for (i=0; i <3; i++) {
        rc = memcached_set(memc, keys[i],
key_length[i], values[i], val_length[i], (time_t)180,
(uint32_t)0);
        if (rc == MEMCACHED_SUCCESS) {
            printf("Save key:%s data:\"%s\" success.\n",
keys[i], values[i]);
        }
    }
```

Memcached安装和使用：

Memcached 与 C/C++ 结合使用

C/C++与Memcached分布式结合测试代码2

```
    //Fetch multi data
    char return_key[MEMCACHED_MAX_KEY];
    size_t return_key_length;
    char *return_value;
    size_t return_value_length;
    uint32_t flags;

    rc = memcached_mget(memc, keys, key_length, 3);
    while ((return_value = memcached_fetch(memc, return_key,
&return_key_length, &return_value_length, &flags, &rc))) {
        if (rc == MEMCACHED_SUCCESS) {
            printf("Fetch key:%s data:%s\n", return_key, return_value);
        }
    }
```

```
    //Delete multi data
    for (i=0; i <3; i++) {
        rc = memcached_set(memc, keys[i], key_length[i],
values[i], val_length[i], (time_t)180, (uint32_t)0);
        rc = memcached_delete(memc, keys[i],
key_length[i], (time_t)0);
        if (rc == MEMCACHED_SUCCESS) {
            printf("Delete %s success\n", keys[i], values[i]);
        }
    }

    //free
    memcached_free(memc);

    return 0;
}
```

Memcached安装和使用：

Memcached 与 C/C++ 结合使用

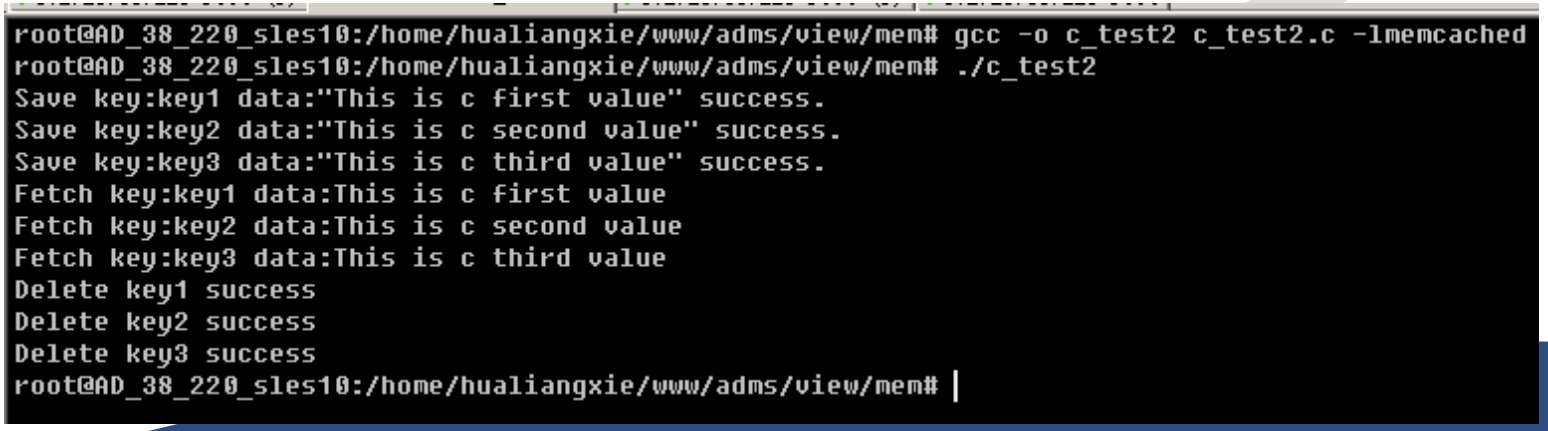
C/C++与Memcached分布式结合测试结果

编译执行以上代码：

```
# gcc -o c_test2 c_test2.c -lmemcached
# ./c_test2
```

输出结果：

```
Save key:key1 data:"This is c first value" success.
Save key:key2 data:"This is c second value" success.
Save key:key3 data:"This is c third value" success.
Fetch key:key1 data:This is c first value
Fetch key:key2 data:This is c second value
Fetch key:key3 data:This is c third value
Delete key1 success
Delete key2 success
Delete key3 success
```



一些经验和技巧：

Memcached一些特性和限制

- 在 Memcached 中可以保存的item数据量是没有限制的，只有内存足够
- Memcached单进程最大使用内存为2G，要使用更多内存，可以分多个端口开启多个Memcached进程
- 最大30天的数据过期时间, 设置为永久的也会在这个时间过期，常量REALTIME_MAXDELTA 60*60*24*30 控制
- 最大键长为250字节，大于该长度无法存储，常量KEY_MAX_LENGTH 250 控制
- 单个item最大数据是1MB，超过1MB数据不予存储，常量POWER_BLOCK 1048576 进行控制，它是默认的slab大小
- 最大同时连接数是200，通过 conn_init()中的freetotal 进行控制，最大软连接数是1024，通过 settings.maxconns=1024 进行控制
- 跟空间占用相关的参数：settings.factor=1.25, settings.chunk_size=48, 影响slab的数据占用和步进方式

一些经验和技巧：

查看Memcached内部工作状态

访问Memcached：telnet 主机名 端口号
查看总状态：stats
查看某项状态：stats curr_connections

禁止LRU

有些情况下LRU机制反倒会造成麻烦。memcached启动时通过“-M”参数可以禁止LRU，如下所示：

```
$ memcached -M -m 1024
```

启动时必须注意的是，小写的“-m”选项是用来指定最大内存大小的。不指定具体数值则使用默认值64MB。

指定“-M”参数启动后，内存用尽时memcached会返回错误。话说回来，memcached毕竟不是存储器，而是缓存，所以推荐使用LRU。

一些经验和技巧：

Memcached使用线程模式工作

在安装的时候必须打开：./configure --enable-threads

安装完之后，启动的时候看看帮助信息有没有这条：
-t <num>　　number of threads to use, default 4
如果存在该选项，说明已经支持了线程，就可以在启动的时候使用 -t 选项来启动多线程

然后启动的时候必须加上你需要支持的线程数量：
/usr/local/memcache/bin/memcached -t 1024

一些经验和技巧：

调优Slab和内存分配1

memcached在启动时指定 Growth Factor因子（通过-f选项），就可以在某种程度上控制slab之间的差异。默认值为1.25。但是，在该选项出现之前，这个因子曾经固定为2，称为“powers of 2”策略。
让我们用以前的设置，以verbose模式启动memcached试试看：
$ memcached -f 2 -vv

```
slab class   1: chunk size    128 perslab  8192
slab class   2: chunk size    256 perslab  4096
slab class   3: chunk size    512 perslab  2048
slab class   4: chunk size   1024 perslab  1024
slab class   5: chunk size   2048 perslab   512
slab class   6: chunk size   4096 perslab   256
slab class   7: chunk size   8192 perslab   128
slab class   8: chunk size  16384 perslab    64
slab class   9: chunk size  32768 perslab    32
slab class  10: chunk size  65536 perslab    16
slab class  11: chunk size 131072 perslab     8
slab class  12: chunk size 262144 perslab     4
slab class  13: chunk size 524288 perslab     2
```

一些经验和技巧：

调优Slab和内存分配2

可见，从128字节的组开始，组的大小依次增大为原来的2倍。这样设置的问题是，slab之间的差别比较大，有些情况下就相当浪费内存。因此，为尽量减少内存浪费，两年前追加了growth factor这个选项。来看看现在的默认设置（f=1.25）时的输出（篇幅所限，这里只写到第10组）：

```
slab class   1: chunk size     88 perslab 11915
slab class   2: chunk size    112 perslab  9362
slab class   3: chunk size    144 perslab  7281
slab class   4: chunk size    184 perslab  5698
slab class   5: chunk size    232 perslab  4519
slab class   6: chunk size    296 perslab  3542
slab class   7: chunk size    376 perslab  2788
slab class   8: chunk size    472 perslab  2221
slab class   9: chunk size    592 perslab  1771
slab class  10: chunk size    744 perslab  1409
```

可见，组间差距比因子为2时小得多，更适合缓存几百字节的记录。从上面的输出结果来看，可能会觉得有些计算误差，这些误差是为了保持字节数的对齐而故意设置的。

将memcached引入产品，或是直接使用默认值进行部署时，最好是重新计算一下数据的预期平均长度，调整growth factor，以获得最恰当的设置。内存是珍贵的资源，浪费就太可惜了。

一些经验和技巧：

参考文档和延伸阅读

以下为本PPT参考文档，特别是参考了mixi.jp 公司编写的《Memcached全面剖析》

Memcached全面剖析：http://tech.idv2.com/2008/08/17/memcached-pdf/

Memcached 1.2 内存模型分析：http://phpcup.cn/viewthread.php?tid=45

Memcached深度分析：http://funjackyone.javaeye.com/blog/128384

memcached server LRU 深入分析：http://www.javaeye.com/topic/225692

Memcache使用详解: http://blog.csdn.net/heiyeshuwu/archive/2006/11/13/1380838.aspx