It 你好 linux 学习文档之 memcached 学习笔记

n

前言

本文是 itnihao 整理来自于网络上各种关于 memcached 的文档，其中有 90% 的内容为网络收集转载而来，非原创作品，由于收集到的网络资料有的无法找到出处，故文中无法做链接，如侵犯到相关人士的权益，请告知我。

请以一种开放的心态分享此文档，如果你觉得本文档不完善，可以继续修改发行此文档，修改后请联系我发布下一版本，谢谢合作。谷歌文档分享项目--自动化运维开源技术文档，地址 http://code.google.com/p/auto-task-pe/downloads/list ，这里汇聚本人写的大多数文档，希望对各位有用，如果你觉得这些文档对你有帮助，可以分享给其他人，让更多的人从中受益。

版本 V1.0
时间 2012-07-31

作者 itnihao
邮箱 itnihao@qq.com

第一章mysql+mencached_大规模 web 应用程序首选

这次是 Fotolog 的经验，传说中比 Flickr 更大的网站，Fotolog 在 21 台服务器上部署了 51 个 memcached 实例，总计有 254G 缓存空间可用，缓存了多达 175G 的内容，这个数量比很多网站的数据库都要大的多，原文是 A Bunch of Great Strategies for Using Memcached and MySQL Better Together，我这里还是选择性的翻译以及按照我的理解补充，感谢 Todd Hoff，总能给我们一些学习的案例，从这里也能看出国外技术的开放态度，不似我们，其实就那么点小九九还藏着掖着，好了，进入正题。

一、关于 memcached

还不知道这个？那你去面试的时候要吃亏了，赶紧去官方网站看一下 http://www.danga.com/memcached/，另外 google 一下用法，硬盘总是太慢，把数据存在内存里面吧，如果你只有一台服务器，推荐用一下 APC(Facebook 在用)或者 eaccelerator 或者 Xcache(国人开发的)，这些产品单机效果更好，如果你需要分布式的缓存方案，那么用 memcached 吧。

二、memcached 如何与 mysql 并肩作战？

- 通过数据库分片来解决数据库写扩展的问题把数据库分片，部署到不同的服务器上，免得只有一个主服务器，写操作成为瓶颈以及可能有的"单点故障"，一般的数据库分片主要是按照业务来分，尽可能的拆分业务，不相干的都独立起来做成服务也好
- 前端 mysql 和一堆 memcached 服务器来应付读的问题应用程序首先从 memcached 中获取数据，获取不到再从数据库中获得并保存在 memcached 中，以前看过一篇文章说好的应用 95%的数据从 memcache 的中获得，3%的数据从 mysql 的 query cache 中获得，剩下 2%才去查表，对比一下你的应用，差距有多远？
- 通过 mysql 复制（master-slave）来解决读的问题
  首先 mysql 数据库通过 master-slave 读写分离，多个 slave 来应对应用程序读的操作。

三、为什么不用 mysql 的 query cache？

我们都知道 mysql 有个 query cache，可以缓存上次查询的结果，可实际上帮不上太多的忙，下面是 mysql query cache 的不足：

- 只能有一个实例，意味着你能存储内容的上限就是你服务器的可用内存，一台服务器能有多少内存？你又能存多少呢？
- 只要有写操作，mysql 的 query cache 就失效只要数据库内容稍有改变，那怕改变的是其他行，mysql 的 query cache 也会失效
-

mysql 的 query cache 只能缓存数据库数据行，意味着其他内容都不行，比如数组，比如对象，而 memcached 理论上可以缓存任何内容，甚至文件^_^

四、Fotolog 的缓存技术

- 非确定性缓存你不确定你要的数据缓存中有没有，你也不知道是不是过期了，于是你就试探性的问 memcached，我要的什么什么数据你那有吗？我可不要过期的数据啊，memcached 告诉你说有并且给你，你就开心了，如果没有呢，你就要从数据库或者别的地方去获取了，这是 memcached 典型的应用。主要应用在：

  1.复杂的数据需要多次读取，你的数据库做了分片处理，从多个数据库中获取数据并组合起来是一个非常大的开销，你大可以把这些数据取出来之后存到 memcached 中

  2.mysql query cache 的一个好的替代方案，这样数据库其他部门改变了，只要自己没改变就没问题（注意数据库更新的问题，后面会提到）

  3.把关系或者列表缓存起来，比如某个栏目下的多篇文章列表

  4.被多个页面调用并且获取起来很慢的数据，或者是更新很慢的数据，比如文章浏览排行榜

  5.如果 cache 的开销超过重新获取的开销，那么不要缓存它吧

  6.标签云和自动建议(类似 google sugest)

  例如：当一个用户上传一个图片，这个用户的好友页面上都要列出这张图片来，那么把它缓存起来吧。

  潜在问题：

  memcached 消耗的主要是服务器内存，对 CPU 消耗很小，所以 Fotolog 把 memcached 部署在他们的应用服务器上（貌似我们也是这样），他们遇到了 CPU 搞到 90%的使用率（怎么会那么高？哪出问题了吧）、内存回收（这是个大问题）等等问题。

- 状态缓存把应用服务的当前状态存在 memcached 中主要应用在：

  1.“昂贵”的操作，开销大的操作

  2.sessions 会话，Flickr 把 session 存在数据库中，个人感觉还是存 memcached 比较“便宜”些，如果 memecached 服务器 down 掉了，那么重新登录吧。

  3.记录用户在线信息(我们也是这样做的)

- 确定性缓存对于某些特定数据库的全部内容，都缓存到 memcached，有一个专门的应用服务来保障你要的数据都在 memcached 中，其他应用服务直接从 memcached 中获取数据而不去取数据库，因为数据库已经全部保存到 memcached 中并保持同步。主要应用在：

  1.读取伸展，所有的读取都从 memcached 中获得，数据库没有负载

  2.”永不过期“(相对的)的数据，比如行政规划数据，变动很小吧

  3.经常调用的内容

  4.用户的认证信息

  5.用户的概要信息

  6.用户的参数设置

7.用户当前常用的媒体文件列表，比如用户的图片

8.用户登录，不走数据库，只走 memcached（个人觉得这个不太好，登录信息还是需要持久化的，用类似 BDB 这样效果也不错）

使用方式：

1.多个专门的缓存池而不是一个大的缓存服务器，多个缓存池保障了高可用性，一个缓存实例挂掉了走其他的缓存实例，所有的缓存实例挂掉了，走数据库（估计数据库抗不住^_^）

2.所有的缓存池都用程序来维护，比如数据库有更新时，程序自动把更新后的内容同步到多个缓存实例中

3.服务器重启之后，缓存要比网站先启动，这就意味着当网站已经启动了，所有的缓存都可用

4.读取的请求可以负载均衡到多个缓存实例中去，高性能，高可靠性

潜在的问题：

1.你需要足够多的内存来存储那么多的数据

2.数据以行记录数据，而 memcached 以对象来存储数据，你的逻辑要把行列的数据转换成缓存对象

3.要维护多个缓存实例非常麻烦,Fotolog 用 Java/Hibernate，他们自己写了个客户端来轮询

4.管理多个缓存实例会增加应用程序的许多开销，但这些开销相对于多个缓存得到的好处来说算不了什么

- 主动缓存数据魔法般的出现在缓存中，当数据库中有更新的时候，缓存立马填充，更新的数据被调用的可能性更高（比如一篇新文章，看的的人当然多），是非确定性缓存的一种变形(原文是 It’s non-deterministic caching with a twist.我觉得这样翻译怪怪的)。主要应用在：

  1.预填充缓存：让 memcached 尽可能的少调用 mysql 如果内容不展现的话。

  2.“预热”缓存：当你需要跨数据中心复制的时候

  使用步骤：

  1.解析数据库更新的二进制日志，发现数据库更新时对 memcached 也进行同样的更新

  2.执行用户自定义函数，设置触发器调用 UDF 更新，具体参考 http://tangent.org/586/Memcached_Functions_for_MySQL.html

  3.使用 BLACKHOLE 策略，传说中 Facebook 也用 mysql 的 Blackhole 存储引擎来填充缓存，写到 Blackhole 的数据复制到缓存中，Facebook 用这来设置数据作废以及跨国界的复制，好处是数据库的复制不走 mysql，这就意味着没有二进制日志以及对 CPU 使用不那么多（啊？难道通过 memcached 存储二进制日志，然后复制到不同的数据库？有经验的同志在这个话题上可以补充。）

- 文件系统缓存把文件直接缓存在 memcached 中，哇，够 BT 的，减轻 NFS 的负担，估计只缓存那些过于热门的图片吧。
- 部分页面内容缓存如果页面的某些部分获取起来非常费劲，以其缓存页面的原始数据还不如把页面的部分内容直接缓存起来直接调用

- 应用程序级别的复制通过 API 来更新缓存，API 的执行细节如下：

1.一个应用把数据写到某个缓存实例，这个缓存实例把内容复制到其他缓存实例（memcached 同步）

2.自动获得缓存池地址以及实例个数

3.同时对多个缓存实例更新

4.如果某个缓存实例 down 掉了，跳到下一个实例，直到更新成功

整个过程非常高效以及低开销

- 其他技巧

1.多节点以应对"单点故障"

2.使用热备技术，当某个节点 down 掉了，另外一台服务自动替换成它的 IP，这样客户端不用更新 memcached 的 IP 地址

3.memcached 可以通过 TCP/UDP 访问，持续连接可以减轻负载，系统设计成可同时承受 1000 个连接

4.不同的应用服务，不同的缓存服务器群

5.检查一下你的数据大小是否匹配你分配的缓存，更多请参考 http://download.tangent.org/talks/Memcached%20Study.pdf

6.不要考虑数据行缓存，缓存复杂的对象

7.不要在你的数据库服务器上跑 memcached，两个都是吃内存的怪兽

8.不要被 TCP 延迟困扰，本地的 TCP/IP 对内存复制是做了优化的

9.尽可能的并行处理数据

10.并不是所有的 memcached 的客户端都是一样的，仔细研究你用的语言所对应的（好像 php 和 memcached 配合的不错）

11.尽可能的是数据过期而不是使数据无效，memcached 可以设定过期时间

12.选择一个好的缓存标识 key，比如更新的时候加上版本号

13.把版本号存储在 memcached 中

作者最后的感言我就不翻译了，貌似 mysql proxy 正在做一个项目，自动同步 mysql 以及 memcached，更多参考
http://jan.kneschke.de/2008/5/18/mysql-proxy-replicating-into-memcache

http://archive.cnblogs.com/a/2010600/

第二章memcached 的安装配置

参考资料

Discuz!的 Memcache 缓存实现：http://www.ccvita.com/261.html
Memcache 分布式部署方案：http://www.ccvita.com/395.html
Memcache 分布式部署方案：http://leven.blog.51cto.com/1675811/362053

1、Memcache 介绍

Memcached 是一种高性能的分布式内存对象缓存系统。通过在内存里维护一个统一的巨大的 hash 表，它能够用来存储各种格式的数据，包括图像、视频、文件以及数据库检索的结果等。简单的说就是将数据调用到内存中，然后从内存中读取，从而大大提高读取速度。

以内存为载体来访问数据，基于 key-value 。

生产环境中，常用来做负载均衡集群中，多个 web server 的 session 共享。还可以做数据库的缓存，比如表的查询结果。Memcached 既能提高访问的速度，同时还减低了数据库的负载。

2、Memcache 原理

memcached 是这样的

1. 接到客户请求后首先查看请求的数据是否在 mem 中存在，如果存在，直接返回给用户
2. 如果不存在，就去查询数据库，把从数据库中获得的数据返回给用户，同时存在 mem 中一份，以后再有用户需要这个数据就直接返回给用户
3. 每次更新了数据库以后 mem 会同时更新数据，保证和数据库的数据一致
4. 服务停止后，缓存中的数据就会丢失

3、Memcache 环境

libevent 服务端软件 http://cloud.github.com/downloads/libevent/libevent/libevent-1.4.14b-stable.tar.gz

memcached 服务端软件

http://memcached.googlecode.com/files/memcached-1.2.8.tar.gz

memcache 客户端软件 http://pecl.php.net/get/memcache-2.2.5.tgz

memcached 服务端需要有 HTTP 环境，对 WEB 客户端提供缓存

memcache 客户端需要有 LAMP 环境

client 客户端测试机

4、Memcached 服务端安装配置

4.1 安装 libevent

```
wget https://github.com/downloads/libevent/libevent/libevent-2.0.19-stable.tar.gz
tar xvf libevent-2.0.19-stable.tar.gz
cd libevent-2.0.19-stable
./configure
make
make install

wget http://memcached.googlecode.com/files/memcached-1.4.13.tar.gz
tar xvf memcached-1.4.13.tar.gz
cd memcached-1.4.13
./configure
make
make install
echo "/usr/local/lib" >> /etc/ld.so.conf
ldconfig
```

4.2 启动 memcached 守护进程

```
/usr/local/bin/memcached  -m 32m -p 11211 -d -u root -P /var/run/memcached.pid -c
1024
```

查看端口

```
[root@localhost ~]# netstat -nlput|grep 11211
tcp        0      0 0.0.0.0:11211     0.0.0.0:*   LISTEN      22786/memcached
tcp        0      0 :::11211          :::*        LISTEN      22786/memcached
udp        0      0 0.0.0.0:11211     0.0.0.0:*               22786/memcached
udp        0      0 :::11211          :::*                    22786/memcached
```

```
ps -ef |grep memcache|grep -v grep
root      22786      1  0 Jul29 ?        00:00:00 /usr/local/bin/memcached -m 32m -p
11211 -d -u root -P /var/run/memcached.pid -c 1024
```

```
echo "/usr/local/bin/memcached -m 32m -p 11211 -d -u root -P /var/run/memcached.
pid -c 1024" >>/etc/rc.local
#加入到开机自启动
```

4.3memcache 命令介绍

```
[root@coral tools]# memcache  -h     查看更多帮助
-d  daemon  启动守护进程
-m  memcache     指定最多使用的内存来提供服务(根据生产环境需求设置)
-p  port     指定 TCP 端口(默认就是 TCP 11211)
-u  user     指定启动的用户
-t  代表并发线程数，最好不要超过 CPU 数
-r  代表 maximize core file limit
-c  代表最大可接受并发连接数
-l  代表监听的 IP 地址
-P  指定 pid 文件存放的位置  (也可不指定)
```

4.4 memcache 进程管理

```
-d start         启动 memcache 服务
-d restart       重启 memcache 服务
-d stop |shutdown    停止 memcache 服务
-d install       安装 memcache 服务
-d uninstall     卸载 memcache 服务
```

4.5memcached 图形报表

```
[root@coral tools]# wget http://livebookmark.net/memcachephp/memcachephp.zip
[root@coral tools]# cp memcachephp.zip /var/www/html/
[root@coral tools]# cd /var/www/html/
[root@coral html]# unzip memcachephp.zip
Archive:  memcachephp.zip
  inflating: memcache.php
[root@coral html]# mv memcache.php memcached.php && vim memcached.php
#注意，这么的 php 页面的名称可以自己随便取
$VERSION='$Id: memcache.php,v 1.1.2.3 2008/08/28 18:07:54 mikl Exp $';
```

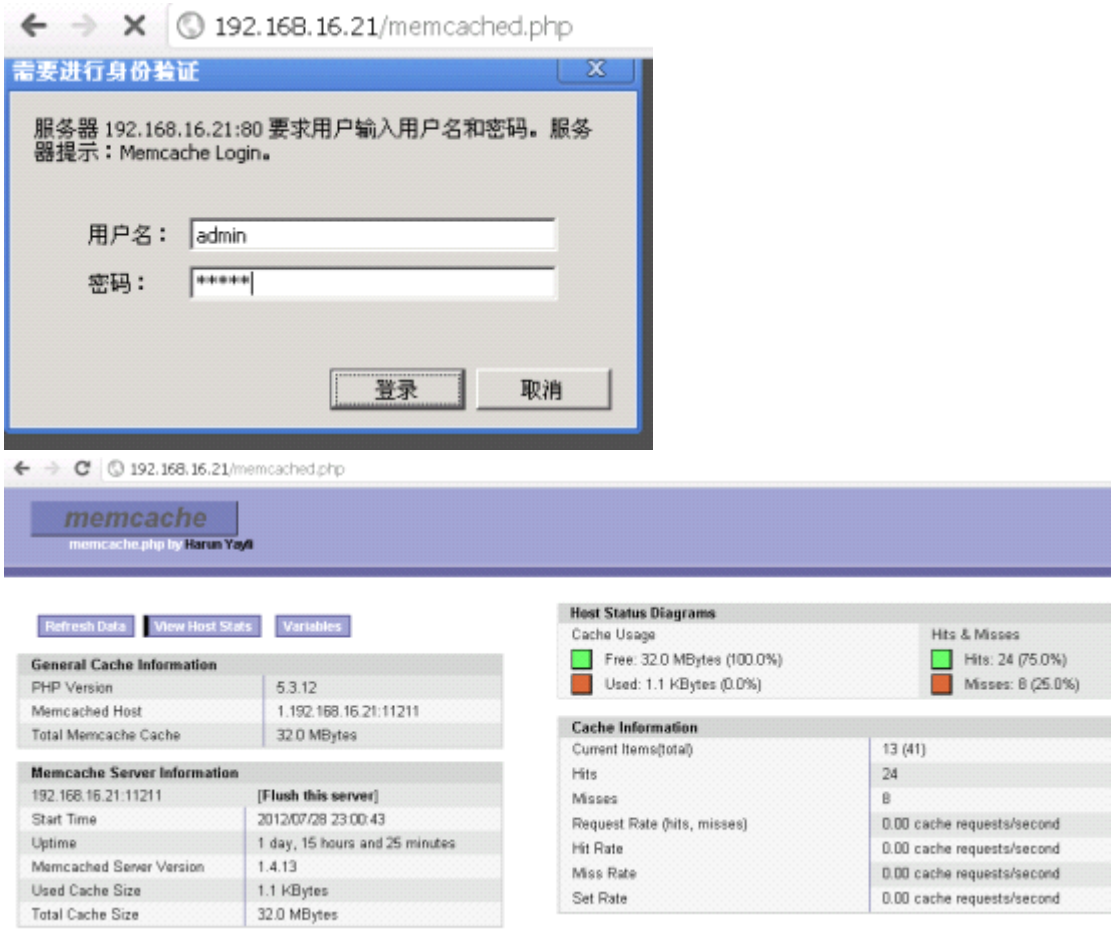
```
define('ADMIN_USERNAME','memcache');     // Admin Username
define('ADMIN_PASSWORD','password');     // Admin Password
define('DATE_FORMAT','Y/m/d H:i:s');
define('GRAPH_SIZE',200);
define('MAX_ITEM_DUMP',50);
$MEMCACHE_SERVERS[] = 'mymemcache-server1:11211'; // add more as an array
$MEMCACHE_SERVERS[] = 'mymemcache-server1:11211'; // add more as an array
```

将以上内容修改成下面内容

```
$VERSION='$Id: memcache.php,v 1.1.2.3 2008/08/28 18:07:54 mikl Exp $';
define('ADMIN_USERNAME','admin');     // Admin Username
define('ADMIN_PASSWORD','admin');     // Admin Password
define('DATE_FORMAT','Y/m/d H:i:s');
define('GRAPH_SIZE',200);
define('MAX_ITEM_DUMP',50);
#$MEMCACHE_SERVERS[] = 'mymemcache-server1:11211'; // add more as an array
#$MEMCACHE_SERVERS[] = 'mymemcache-server1:11211'; // add more as an array
$MEMCACHE_SERVERS[] = '192.168.16.21:11211'; // add more as an array
```

说明：默认用户是 memcache 密码是 password 把它修改掉，然后指定 memcached 服务端的 IP 地址和端口号

5、访问测试



5、php 中 Memcache 插件安装

5.1 安装 php for memcached

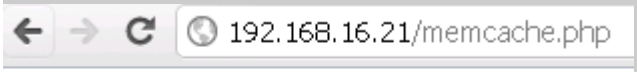
```
wget http://pecl.php.net/get/memcache-2.2.6.tgz
tar xvf memcache-2.2.6.tgz
cd memcache-2.2.6
$(find / -name phpize)
./configure --with-php-config=$(find / -name php-config)
make && make install
#在 php.ini 开启 php 对 memcached 的支持
echo  "extension_dir  =  \"/usr/lib64/extensions/no-debug-non-zts-20090626/\""
>>$(find / -name php.ini)
echo "extension = memcache.so" >>$(find / -name php.ini)
```

重启 web 服务，如采用 fastcgi 的 php，需重启 php-fpm

#php web 页面测试

```
<?php
    $memcache = new Memcache;
 //创建一个 memcache 对象
    $memcache->connect('192.168.16.21',11211) or die ("Could not connect"); //
连接 Memcached 服务器
    $memcache->set('key', 'test_data'); //设置一个变量到内存中，名称是 key 值是 t
est_data
    $get_value = $memcache->get('key'); //从内存中取出 key 的值
    echo $get_value;
?>
```

将此页面放置到 web 目录下面




```
test_data
```

5.2 客户端测试缓存结果

```
[root@client coral]# vim test.sh

#!/bin/bash
for i in `seq 40`
do
 wget --spide http://192.168.16.21/memcached.php
 sleep 1
done
```

脚本说明：每一秒下载一次页面，使用--spide 不是真的下载到本地，测试下载

第三章 php mysql memcache 的应用分析

PHP 缓存相关知识学习笔记

由头

最近，周围的几个同事跳槽升级，弄的自己也心里很痒。关心了一下相关面试笔试的内容，大部分地方对于 PHP 的缓存技术都有过询问。

之前自己做的几个项目，都是简单的 B/S 架构的存取数据库数据的控制面板的咚咚，用户量撑死也就在几百到几天人不等。基本上不用考虑性能和优化的问题。(无论是 Web 的优化，还是数据库的优化)

不过，真正大的公司和 PHP 的项目，都是要考虑这方面的东西的。因此，就揪出这个知识点出来突击一下。

关于 PHP 的缓存

为什么要缓存

为什么需要缓存，缓存的目的就是加速页面的载入，因此使用缓存之后，你的请求将会直接去获取 html 文件，传统的动态页面从数据库中获取数据，然后用程序 将数据显示出来的这个过程就被绕过去了，而这个过程是最消耗系统资源的过程，静态页面传送过程中，系统 I/O 所占用的资源相比之下就可以忽略了。当然，更 传统的其实还是静态页面，因为它出现在动态页面之前。有人说用缓存更新不及时，不够快，所以很不方便，这也是很有道理的，但是仅仅是因为这样就放弃使用缓存，或许不太划算，特别是在你服务器压力比较大的情况下。

什么是缓存

顾名思义，缓存是一块设定了缓冲功能的内存区域，用于提供数据的高速缓冲。无论是针对硬件的 CPU 的缓存，硬盘的缓存。都是起到减少不比要的 I/O 访问，提高数据访问速度的目的。

缓存的形式

字面上，缓存就应该是内存的一个部分。但是，也有其他形式的缓存。

CPU 上的缓存可以有 2 到 3 级 RAM，是现在 CPU 设计中一个重要的组成部分。同样，硬盘上的 Flash RAM 芯片，承担了缓存的作用。

而针对应用软件系统，例如：Memcached 这样的软件，从物理内存中划去了一部分空间，为各种应用程序提供缓存服务。

针对特殊的 Web + DB 的这种 B/S 业务系统，还有类似的生成静态页，减少数据库存取 I/O 操作的缓存方式。

PHP 缓存的具体方法

针对 PHP 这种具体的 B/S 架构的脚本语言，有几种常见的缓存形式。

文件缓存：

文件缓存是 PHP 缓存的十分常见的一种形式。其主要目的是将页面中不经常变更的数据保存在文件中，当有用

户请求访问的时候，直接将数据调出，避免了再次对数据库的请求的 I/O 操作，减少了数据库的负载压力。

一般情况下，PHP 的文件缓存分为两种方法：

第一种，把需要缓存的数据进行处理，形成 PHP 可以直接执行的文件。在需要缓存数据的时候，通过 include 方式引入，并使用。

第二种，把需要的数据通过 serialize 函数序列化后直接保存到文件。在需要使用缓存数据的时候，通过反序列化读入文件内容并复制给需要的变量，然后使用。当然，这里说到的 serialize 函数是 PHP 自带的 serialize()函数。针对该函数的改进问题，后续的章节将会提及。

通过测试发现，第二种也就是 serialize 缓存数据的方式更加高效。

分析原因：
include 方式读取缓存的时候，PHP 需要执行几个过程
1.读取文件
2.解析所 Include 的文件
3.执行，给变量赋值

而 serialize 序列化方式读取缓存的时候：
1.读取数据
2.反序列化数据内容
3.给变量赋值

从以上内容对比的话，可能是由于解析 PHP 文件内的数组需要的时间超过 unserialize 反序列化数组的时间。

测试代码：

CacheTest_IncludeFile.php

```
<?php
$t1 = gettimeofday();
for ($i = 0; $i < 10000; $i++){
    include("CacheTest_IncludeData.php");
}
$t2 = gettimeofday();
echo ($t2['sec'] - $t1['sec']) * 1000 + ($t2['usec'] - $t1['usec']) / 1000 . "\n";
```

CacheTest_IncludeData.php

```
<?php
$testarray = array (
                'mtime' => 0,
                'values' =>
                array (
                        0 =>
                        array (
                                'id' => '3',
                                'title' => 'Title NO.3',
                                'dateline' => '1191244506',
```

```
                                        ),
                                1 =>
                        array (
                                        'id' => '2',
                                        'title' => 'Title NO.2',
                                        'dateline' => '1191244505',
                                        ),
                                2 =>
                                array (
                                        'id' => '1',
                                        'title' => 'Title NO.1',
                                        'dateline' => '1191244504',
                                        ),
                                ),
                        'multipage' => NULL,
                        );
```

CacheTest_SerializeFile.php

```
<?php
function read_cache($filename) {
        if(@$fp = fopen($filename, 'r')) {
                        @$data = fread($fp,filesize($cachefile));
                        fclose($fp);
        }
        return $data;
}
$t1 = gettimeofday();
for ($i = 0; $i < 10000; $i++){
        $x = read_cache("CacheTest_SerializeData.php");
        $x_r = unserialize($x);
}
$t2 = gettimeofday();
echo ($t2['sec'] - $t1['sec']) * 1000 + ($t2['usec'] - $t1['usec']) / 1000 . "\n";
```

CacheTest_SerializeData.php

```
a:3:{s:5:"mtime";i:0;s:6:"values";a:3:{i:0;a:3:{s:2:"id";s:1:"3";s:5:"title";s:10:"Title
NO.3";s:8:"dateline";s:10:"1191244506";}i:1;a:3:{s:2:"id";s:1:"2";s:5:"title";s:10:"Title
NO.2";s:8:"dateline";s:10:"1191244505";}i:2;a:3:{s:2:"id";s:1:"1";s:5:"title";s:10:"Title
NO.1";s:8:"dateline";s:10:"1191244504";}}s:9:"multipage";N;}
```

运行结果对比：

```
[root@localhost ~/lab/php/cachetest]# php -e CacheTest_IncludeFile.php
```

495.014
[root@localhost ~/lab/php/cachetest]# php -e CacheTest_SerializeFile.php
235.883

总结分析：
第一种，include 缓存的方式
优点：增加数据的保密性，和安全性，缓存内容不会被外界发现。
缺点：速度相对较慢。
用途：保存禁止系统外部得知的数据，比如 web 系统的设置，甚至 MySQL 信息等的保存
第二种，serialize 序列化缓存的方式
优点：速度较快。
缺点：缓存系统文件路径一旦曝光，缓存内容会泄露。
用途：缓存最新文章，相关文章等不担心外部得知的数据的时候，可以使用这种方式。

备注：
当安装了 ea、apc 等 PHP 内存缓存之后，第一种通过 include 读取缓存的方式速度会高于第二种 serialize 序列化缓存的方式。

本部分参考: http://www.ccvita.com/311.html

Sqlite 缓存：

Sqlite 是嵌入式 SQL 数据库引擎 SQLite（SQLite Embeddable SQL Database Engine）的一个扩展。SQLite 是一个实现嵌入式 SQL 数据库引擎小型 C 语言库（C library），实现了独立的，可嵌入的，零配置的 SQL 数据库引擎。
特性包括：
事务操作是原子，一致，孤立，并且持久的，即使在系统崩溃和电源故障之后。零配置——不需要安装和管理。
实现了绝大多数 SQL92 标准。整个数据库存储在一个单一的文件中。数据库文件可以在不同字节序的机器之间自由地共享。支持 最大可达 2T 的数据库。字符串和 BLOB 类型的大小只受限于可用内存。完整配置的少于 250KB，忽略一些可选特性的少于 150KB。在大多数常见操作上 比流行的客户/服务器数据库引擎更快。简单易于使用的 API。
内建 TCL 绑定。另外提供可用于许多其他语言的绑定。具有良好注释的源代码，代码 95%有较好的注释。
独立：
没有外部依赖。
源代码位于公共域，可用于任何用途。
用 SQLite 连接的程序可以使用 SQL 数据库，但不需要运行一个单独的关系型数据库管理系统进程（separate RDBMS process）。SQLite 不是一个用于连接到大型数据库服务器（big database server）的客户端库（client library），而是非常适合桌面程序和小型网站的数据库服务器。SQLite 直接读写（reads and writes directly）在硬盘上的数据库文件。著名的浏览器 Firefox 的数据库就采用的 Sqlite。Firefox 所记录的配置，书签，历史记录，Cookie 和 Password

等信息，均存储在 Sqlite 中。

在缓存方案中，选择采用 Sqlite 作为方案，主要是看中了它轻量，简单，快速的特点。并且，PHP5 以上版本就内部支持了 Sqlite 的操作，使用起来简单，方便。因此，作为 PHP 和 MySQL 中间的一个有效的缓冲机制，Sqlite 无意是一个很好的选择。由于 Sqlite 的缓存方案不是本文探讨的重点。因此，这里就做上述的简单介绍。

Apc 缓存：

APC，严格的意义上讲应该成为加速器，全称 Alternative PHP Cache，是 PHP 的一个免费公开的优化代码缓存。

它用来提供免费，公开并且强健的架构来缓存和优化 PHP 的中间代码。对应的还有 eAccelerator 和 Zend Optimizer 两个加速器方案。

APC 提供两种缓存功能，即缓存 Opcode(目标文件)，我们称之为 apc_compiler_cache。同时它还提供一些接口用于 PHP 开发人员将用户数据驻留在内存中，我们称之为 apc_user_cache。我们这里主要控讨 php-apc 的配置。

安装 PHP APC:

作为测试环境，我们这里使用的是 CentOS5.4（2.6.18-194.3.1.el5） + Apache2.2.3 + php5.2.10.我们可以去 pecl apc
下载 APC-3.0.19.tgz

```
# tar -xzvf APC-3.0.19.tgz
#cd      APC-3.0.19
# /usr/bin/phpize
# ./configure --enable-apc --enable-mmap --enable-apc-spinlocks --disable-apc-pthreadmutex
#make
#make install
```

注意:我们这里支持 mmap，同时采用 spinlocks 自旋锁。Spinlocks 是 Facebook 推荐使用，同时也是 APC 开发者推荐使用的锁机制。

APC PHP.ini 配置选项详解

```
[APC]
; Alternative PHP Cache   用于缓存和优化 PHP 中间代码

apc.cache_by_default = On
;SYS
;   是否默认对所有文件启用缓冲。
;   若设为 Off 并与以加号开头的 apc.filters 指令一起用，则文件仅在匹配过滤器时才被缓存。

apc.enable_cli = Off
;SYS
;   是否为 CLI 版本启用 APC 功能，仅用于测试和调试目的才打开此指令。
```

```
apc.enabled = On
;   是否启用 APC，如果 APC 被静态编译进 PHP 又想禁用它，这是唯一的办法。

apc.file_update_protection = 2
;SYS
;   当你在一个运行中的服务器上修改文件时，你应当执行原子操作。
;   也就是先写进一个临时文件，然后将该文件重命名(mv)到最终的名字。
;   文本编辑器以及   cp, tar  等程序却并不是这样操作的，从而导致有可能缓冲了残缺的文
件。
;   默认值   2  表示在访问文件时如果发现修改时间距离访问时间小于   2  秒则不做缓冲。
;   那个不幸的访问者可能得到残缺的内容，但是这种坏影响却不会通过缓存扩大化。
;   如果你能确保所有的更新操作都是原子操作，那么可以用   0  关闭此特性。
;   如果你的系统由于大量的 IO  操作导致更新缓慢，你就需要增大此值。

apc.filters =
;SYS
;   一个以逗号分隔的 POSIX 扩展正则表达式列表。
;   如果源文件名与任意一个模式匹配，则该文件不被缓存。
;   注意，用来匹配的文件名是传递给 include/require  的文件名，而不是绝对路径。
;   如果正则表达式的第一个字符是"+"则意味着任何匹配表达式的文件会被缓存，
;   如果第一个字符是"-"则任何匹配项都不会被缓存。"-"是默认值，可以省略掉。

apc.ttl = 0
;SYS
;   缓存条目在缓冲区中允许逗留的秒数。0  表示永不超时。建议值为 7200~36000。
;   设为   0  意味着缓冲区有可能被旧的缓存条目填满，从而导致无法缓存新条目。

apc.user_ttl = 0
;SYS
;   类似于 apc.ttl，只是针对每个用户而言，建议值为 7200~36000。
;   设为   0  意味着缓冲区有可能被旧的缓存条目填满，从而导致无法缓存新条目。

apc.gc_ttl = 3600
;SYS
;   缓存条目在垃圾回收表中能够存在的秒数。
;   此值提供了一个安全措施，即使一个服务器进程在执行缓存的源文件时崩溃，
;   而且该源文件已经被修改，为旧版本分配的内存也不会被回收，直到达到此 TTL 值为止。
;   设为零将禁用此特性。

apc.include_once_override = Off
;SYS
;   关于该指令目前尚无说明文档，参见：http://pecl.php.net/bugs/bug.php?id=8754
;   请保持为 Off，否则可能导致意想不到的结果。
```

```
apc.max_file_size = 1M
;SYS
;   禁止大于此尺寸的文件被缓存。

apc.mmap_file_mask =
;SYS
;   如果使用 - enable-mmap(默认启用)为 APC 编译了 MMAP 支持，
;   这里的值就是传递给 mmap 模块的 mktemp 风格的文件掩码(建议值为"/tmp/apc.XXXXXX")。
;   该掩码用于决定内存映射区域是否要被 file-backed 或者 shared memory backed。
;   对于直接的 file-backed 内存映射，要设置成"/tmp/apc.XXXXXX"的样子(恰好 6 个 X)。
;   要使用 POSIX 风格的 shm_open/mmap 就需要设置成"/apc.shm.XXXXXX"的样子。
;   你还可以设为"/dev/zero"来为匿名映射的内存使用内核的"/dev/zero"接口。
;   不定义此指令则表示强制使用匿名映射。

apc.num_files_hint = 1000
;SYS
; Web 服务器上可能被包含或被请求的不同源文件的大致数量(建议值为 1024~4096)。
;   如果你不能确定，则设为  0 ；此设定主要用于拥有数千个源文件的站点。

apc.optimization = 0
;   优化级别(建议值为  0 ) 。
;   正整数值表示启用优化器，值越高则使用越激进的优化。
;   更高的值可能有非常有限的速度提升，但目前尚在试验中。

apc.report_autofilter = Off
;SYS
;   是否记录所有由于 early/late binding 原因而自动未被缓存的脚本。

apc.shm_segments = 1
;SYS
;   为编译器缓冲区分配的共享内存块数量(建议值为 1)。
;   如果 APC 耗尽了共享内存，并且已将 apc.shm_size 指令设为系统允许的最大值，
;   你可以尝试增大此值。

apc.shm_size = 30
;SYS
;   每个共享内存块的大小(以 MB 为单位，建议值为 128~256)。
;   有些系统(包括大多数 BSD 变种)默认的共享内存块大小非常少。

apc.slam_defense = 0
;SYS(反对使用该指令，建议该用 apc.write_lock 指令)
;   在非常繁忙的服务器上，无论是启动服务还是修改文件，
```

```
;   都可能由于多个进程企图同时缓存一个文件而导致竞争条件。
;   这个指令用于设置进程在处理未被缓存的文件时跳过缓存步骤的百分率。
;   比如设为 75 表示在遇到未被缓存的文件时有 75%的概率不进行缓存，从而减少碰撞几率。
;   鼓励设为   0  来禁用这个特性。

apc.stat = On
;SYS
;   是否启用脚本更新检查。
;   改变这个指令值要非常小心。
;   默认值   On  表示 APC 在每次请求脚本时都检查脚本是否被更新，
;   如果被更新则自动重新编译和缓存编译后的内容。但这样做对性能有不利影响。
;   如果设为   Off  则表示不进行检查，从而使性能得到大幅提高。
;   但是为了使更新的内容生效，你必须重启 Web 服务器。
;   这个指令对于 include/require  的文件同样有效。但是需要注意的是，
;   如果你使用的是相对路径，APC 就必须在每一次 include/require  时都进行检查以定位文
件。
;   而使用绝对路径则可以跳过检查，所以鼓励你使用绝对路径进行 include/require  操作。

apc.user_entries_hint = 100
;SYS
;   类似于 num_files_hint 指令，只是针对每个不同用户而言。
;   如果你不能确定，则设为   0  。

apc.write_lock = On
;SYS
;   是否启用写入锁。
;   在非常繁忙的服务器上，无论是启动服务还是修改文件，
;   都可能由于多个进程企图同时缓存一个文件而导致竞争条件。
;   启用该指令可以避免竞争条件的出现。

apc.rfc1867 = Off
;SYS
;   打开该指令后，对于每个恰好在 file 字段之前含有 APC_UPLOAD_PROGRESS 字段的上
传文件，
; APC  都将自动创建一个 upload_的用户缓存条目(就是 APC_UPLOAD_PROGRESS  字段
值)。
```

APC 的相关函数

```
apc_cache_info          - Retrieves cached information (and meta-data) from APC's data store
apc_clear_cache          - Clears the APC cache
apc_define_constants    - Defines a set of constants for later retrieval and mass-definition
apc_delete              - Removes a stored variable from the cache
apc_fetch                - Fetch a stored variable from the cache
apc_load_constants       - Loads a set of constants from the cache
```

apc_sma_info - Retrieves APC's Shared Memory Allocation information
apc_store - Cache a variable in the data store

apc 的用法比较简单,只有几个函数,列举如下。

apc_cache_info () 返回缓存信息
apc_clear_cache() 清除 apc 缓存内容。默认(无参数)时,只清除系统缓存,要清除用户缓存,需用, user‘参数。
apc_define_constants (string key, array constants [, bool case_sensitive]) 将数组 constants 以常量加入缓存。
apc_load_constants (string Key) 取出常量缓存。
apc_store (string key, mixed var [, int ttl]) 在缓存中保存数据。
apc_fetch (string key) 获得 apc_store 保存的缓存内容
apc_delete (string key) 删除 apc_store 保存的内容。

APC 的管理：

到 pecl.php.net 下载 apc 源码包有个 apc.php，copy 到你的 web server 可以访问到的地方，浏览即可访问。

管理界面功能有：

1. Refresh Data
2. View Host Stats
3. System Cache Entries
4. User Cache Entries
5. Version Check

Xcache 缓存：

和 APC(Alternative PHP Cache),eAccelerator(eacc)类似，XCache 是一个开源的 opcode 缓存器/优化器, 这意味着他能够提高您服务器上的 PHP 性能. 他通过把编译 PHP 后的数据缓冲到共享内存从而避免重复的编译过程, 能够直接使用缓冲区已编译的代码从而提高速度. 通常能够提高您的页面生成速率 2 到 5 倍, 降低服务器负载. 需要特别说明的是，这个加速器是国人开发的方案。

与 APC 类似，由于是编译级别的缓存和优化。因此，除配置之外，没有太多可以说道的东西。这里就直接应用 IBM 开发社区中的一篇文章，完成对 XCache 的配置和安装说明。

参考：http://www.ibm.com/developerworks/library/os-php-fastapps1/index.html
三种加速器的性能对比文章：http://www.vpser.net/opt/apc-eaccelerator-xcache.html

Memcached 缓存：

Memcached 是什么？

memcached 是以 LiveJournal 旗下 Danga Interactive 公司的 Brad Fitzpatric 为首开发的一款软件。现在已成为 mixi、 hatena、 Facebook、 Vox、LiveJournal 等众多服务中 提高 Web 应用扩展性的重要因素。

许多 Web 应用都将数据保存到 RDBMS 中，应用服务器从中读取数据并在浏览器中显示。但随着数据量的增大、访问的集中，就会出现 RDBMS 的负担加重、数据库响应恶化、网站显示延迟等重大影响。这时就该 memcached 大显身手了。memcached 是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。

PHP 与 Memcached 的关系

php_memcached 是 php 为 Memcached 提供的 PECL 扩展。由于 Memcached 简单的协议规范，因此，当 Memcached 推出后，就有了 PHP 的接口规范和相关扩展。

在阐述这个问题之前，我们首先要清楚它“不是什么”。很多人把它当作和 SharedMemory 那种形式的存储载体来使用，虽然 memcached 使用了同样的“Key=>Value”方式组织数据，但是它和共享内存、APC 等本地缓存有非常大的区别。Memcached 是分布式的，也就是说它不是本地的。它基于网络连接（当然它也可以使用 localhost）方式完成服务，本身它是一个独立于应用的程序或守护进程（Daemon 方式）。

Memcached 使用 libevent 库实现网络连接服务，理论上可以处理无限多的连接，但是它和 Apache 不同，它更多的时候是面向稳定的持续连接的，所以它实际的并发能力是有限制的。在保守情况下 memcached 的最大同时连接数为 200，这和 Linux 线程能力有关系，这个数值是可以调整的。关于 libevent 可以参考相关文档。 Memcached 内存使用方式也和 APC 不同。APC 是基于共享内存和 MMAP 的，memcached 有自己的内存分配算法和管理方式，它和共享内存没有 关系，也没有共享内存的限制，通常情况下，每个 memcached 进程可以管理 2GB 的内存空间，如果需要更多的空间，可以增加进程数。

Memcached 适合什么场合

在很多时候，memcached 都被滥用了，这当然少不了对它的抱怨。经常在论坛上看见有人发贴，类似于“如何提高效率”，回复是“用 memcached”，至于怎么用，用在哪里，用来干什么一句没有。memcached 不是万能的，它也不是适用在所有场合。

Memcached 是“分布式”的内存对象缓存系统，那么就是说，那些不需要“分布”的，不需要共享的，或者干脆规模小到只有一台服务器的应 用，memcached 不会带来任何好处，相反还会拖慢系统效率，因为网络连接同样需要资源，即使是 UNIX 本地连接也一样。 在我之前的测试数据中显示，memcached 本地读写速度要比直接 PHP 内存数组慢几十倍，而 APC、共享内存方式都和直接数组差不多。可见，如果只是 本地级缓存，使用 memcached 是非常不划算的。

Memcached 在很多时候都是作为数据库前端 cache 使用的。因为它比数据库少了很多 SQL 解析、磁盘操作等开销，而且它是使用内存来管理数 据的，所以它可以提供比直接读取数据库更好的性能，在大型系统中，访问同样的数据是很频繁的，memcached 可以大大降低数据库压力，使系统执行效率 提升。另外，memcached 也经常作为服务器之间数据共享的存储媒介，例如在 SSO 系统中保存系统单点登陆状态的数据就可以保存在 memcached 中，被多个应用共享。

需要注意的是，memcached 使用内存管理数据，所以它是易失的，当服务器重启，或者 memcached 进程中止，数据便会丢失，所以 memcached 不能用来持久保存数据。很多人的

错误理解，memcached 的性能非常好，好到了内存和硬盘的对比程度，其实 memcached 使用 内存并不会得到成百上千的读写速度提高，它的实际瓶颈在于网络连接，它和使用磁盘的数据库系统相比，好处在于它本身非常“轻”，因为没有过多的开销和直接 的读写方式，它可以轻松应付非常大的数据交换量，所以经常会出现两条千兆网络带宽都满负荷了，memcached 进程本身并不占用多少 CPU 资源的情况。

PHP 调用 Memcached 的研发环境的搭建

该章节标题中，我增加了研发环境的搭建，主要原因是，这种搭建方案，在实际生产环境中是并不可取的。不能体现出 Memcached 的分布式，高扩展性等特点。如果是单机版本想提升 Web 系统的效能的缓存方案，建议使用更适合的 APC 等加速器方法。效果要好于采用 Memcached 的方案。

以上内容，基本上描述了 PHP 缓存技术中涉及到的基础的知识点。主要对 Memcache 扩展做了详尽的分析和解释。在整个学习中，主要参考了如下作者的文档，在这里表示感谢。

参考文章：

http://www.ooso.net/archives/306
http://www.ooso.net/archives/428
http://www.ooso.net/archives/475
http://www.ooso.net/archives/479
http://www.ooso.net/archives/524
http://www.ooso.net/archives/558
http://opensource.dynamoid.com/
http://blog.developers.api.sina.com.cn/?p=124
http://www.phpx.com/happy/viewthread.php?tid=138416
http://tech.idv2.com/2008/07/10/memcached-001/#content_2_0
http://framework.zend.com/manual/1.10/zh/zend.cache.backends.html
http://www.okajax.com/a/200904/php_cache.html
http://swingchen.javaeye.com/blog/152800
http://code.sixapart.com/svn/memcached/trunk/server/doc/protocol.txt

第四章 memcached 的协议

Memcache 协议

在 memcache 协议中发送的数据分两种：文本行 和 自由数据。 文本行被用于来自客户端的命令和服务器的回应。自由数据用于客户端从服务器端存取数据时。存储在 memcached 中的数据通过键值来标识。键值是一个文本字符串，对于需要存取这项数据的客户端而言，它必须是唯一的。

协议	Protocol
memcached 的客户端使用TCP链接 与 服务器通讯。（UDP 接口也同样有效，参考后文的 “UDP 协	Clients of memcached communicate with server through TCP connections. (A UDP interface is also

议”）一个运行中的 memcached 服务器监视一些（可设置）端口。客户端连接这些端口，发送命令到服务器，读取回应，最后关闭连接。	available; details are below under “UDP protocol.”) A given running memcached server listens on some (configurable) port; clients connect to that port, send commands to the server, read responses, and eventually close the connection.
结束会话不需要发送任何命令。当不再需 memcached 服务时，要客户端可以在任何时候关闭连接。需要注意的是，鼓励客户端缓存这些连接，而不是每次需要存取数据时都重新打开连接。这是因为 memcached 被特意设计成及时开启很多连接也能够高效的工作（数百个，上千个如果需要的话）。缓存这些连接，可以消除建立连接所带来的开销（/*/相对而言，在服务器端建立一个新连接的准备工作所带来的开销，可以忽略不计。）。	There is no need to send any command to end the session. A client may just close the connection at any moment it no longer needs it. Note, however, that clients are encouraged to cache their connections rather than reopen them every time they need to store or retrieve data. This is because memcached is especially designed to work very efficiently with a very large number (many hundreds, more than a thousand if necessary) of open connections. Caching connections will eliminate the overhead associated with establishing a TCP connection (the overhead of preparing for a new connection on the server side is insignificant compared to this).
在 memcache 协议中发送的数据分两种：文本行 和自由数据。 文本行被用于来自客户端的命令和服务器的回应。自由数据用于客户端从服务器端存取数据时。同样服务器会以字节流的方式传回自由数据。/*/服务器不用关心自由数据的字节顺序。自由数据的特征没有任何限制；但是通过前文提到的文本行，这项数据的接受者（服务器或客户端），便能够精确地获知所发送的数据库的长度。	There are two kinds of data sent in the memcache protocol: text lines and unstructured data. Text lines are used for commands from clients and responses from servers. Unstructured data is sent when a client wants to store or retrieve data. The server will transmit back unstructured data in exactly the same way it received it, as a byte stream. The server doesn’t care about byte order issues in unstructured data and isn’t aware of them. There are no limitations on characters that may appear in unstructured data; however, the reader of such data (either a client or a server) will always know, from a preceding text line, the exact length of the data block being transmitted.
文本行固定以“\r\n”(回车符紧跟一个换行符)结束。自由数据也是同样会以“\r\n”结束，但是 \r(回车符)、	Text lines are always terminated by \r\n. Unstructured data is _also_

\n(换行符)，以及任何其他8位字符，均可出现在数据中。因此，当客户端从服务器取回数据时，必须使用数据区块的长度来确定数据区块的结束位置，而不要依据数据区块末尾的“\r\n”，即使它们固定存在于此。	terminated by \r\n, even though \r, \n or any other 8-bit characters may also appear inside the data. Therefore, when a client retrieves data from a server, it must use the length of the data block (which it will be provided with) to determine where the data block ends, and not the fact that \r\n follows the end of the data block, even though it does.
键值	**Keys**
存储在 memcached 中的数据通过键值来标识。键值是一个文本字符串，对于需要存取这项数据的客户端而言，它必须是唯一的。键值当前的长度限制设定为250字符（当然，客户端通常不会用到这么长的键）；键值中不能使用制表符和其他空白字符（例如空格，换行等）。	Data stored by memcached is identified with the help of a key. A key is a text string which should uniquely identify the data for clients that are interested in storing and retrieving it. Currently the length limit of a key is set at 250 characters (of course, normally clients wouldn’t need to use such long keys); the key must not include control characters or whitespace.
命令	**Commands**
所有命令分为3种类型	There are three types of commands.
存储命令（有3项：’set’、’add’、’repalce’）指示服务器储存一些由键值标识的数据。客户端发送一行命令，后面跟着数据区块；然后，客户端等待接收服务器回传的命令行，指示成功与否。	Storage commands (there are three: “set”, “add” and “replace”) ask the server to store some data identified by a key. The client sends a command line, and then a data block; after that the client expects one line of response, which will indicate success or faulure.
取回命令（只有一项：’get’）指示服务器返回与所给键值相符合的数据（一个请求中右一个或多个键值）。客户端发送一行命令，包括所有请求的键值；服务器每找到一项内容，都会发送回客户端一行关于这项内容的信息，紧跟着是对应的数据区块；直到服务器以一行“END”回应命令结束。	Retrieval commands (there is only one: “get”) ask the server to retrieve data corresponding to a set of keys (one or more keys in one request). The client sends a command line, which includes all the

	requested keys; after that for each item the server finds it sends to the client one response line with information about the item, and one data block with the item's data; this continues until the server finished with the "END" response line.
/*?*/其他的命令都不能携带自由数据。在这些命令中，客户端发送一行命令，然后等待（由命令所决定）一行回应，或最终以一行"END"结束的多行命令。	All other commands don't involve unstructured data. In all of them, the client sends one command line, and expects (depending on the command) either one line of response, or several lines of response ending with "END" on the last line.
一行命令固定以命令名称开始，接着是以空格隔开的参数（如果有参数的话）。命令名称大小写敏感，并且必须小写。	A command line always starts with the name of the command, followed by parameters (if any) delimited by whitespace. Command names are lower-case and are case-sensitive.
一些客户端发送给服务器的命令会包含一些时限（针对内容或客户端请求的操作）。这时，时限的具体内容既可以是 Unix 时间戳（从1970年1月1日开始的秒钟数），或当前时间开始的秒钟数。对后者而言，不能超过 60*60*24*30（30天）；如果超出，服务器将会理解为 Unix 时间戳，而不是从当前时间起的秒偏移。	Some commands involve a client sending some kind of expiration time (relative to an item or to an operation requested by the client) to the server. In all such cases, the actual value sent may either be Unix time (number of seconds since January 1, 1970, as a 32-bit value), or a number of seconds starting from current time. In the latter case, this number of seconds may not exceed 60*60*24*30 (number of seconds in 30 days); if the number sent by a client is larger than that, the server will consider it to be real Unix time value rather than an offset from current time.
错误字串	**Error strings**
每一个由客户端发送的命令，都可能收到来自服务器的错误字串回复。这些错误字串会以三种形式出	Each command sent by a client may be answered with an error string

现：	from the server. These error strings come in three types:
- “ERROR\r\n”	
意味着客户端发送了不存在的命令名称。	means the client sent a nonexistent command name.
- “CLIENT_ERROR <error>\r\n”	
意味着输入的命令行里存在一些客户端错误，例如输入未遵循协议。<error>部分是人类易于理解的错误解说……	means some sort of client error in the input line, i.e. the input doesn’t conform to the protocol in some way. <error> is a human-readable error string.
- “SERVER_ERROR <error>\r\n”	
意味着一些服务器错误，导致命令无法执行。<error>部分是人类易于理解的错误解说。在一些严重的情形下（通常应该不会遇到），服务器将在发送这行错误后关闭连接。这是服务器主动关闭连接的唯一情况。	means some sort of server error prevents the server from carrying out the command. <error> is a human-readable error string. In cases of severe server errors, which make it impossible to continue serving the client (this shouldn’t normally happen), the server will close the connection after sending the error line. This is the only case in which the server closes a connection to a client.
在后面每项命令的描述中，这些错误行不会再特别提到，但是客户端必须考虑到这些它们存在的可能性。	In the descriptions of individual commands below, these error lines are not again specifically mentioned, but clients must allow for their possibility.
存储命令	**Storage commands**
首先，客户端会发送一行像这样的命令：	First, the client sends a command line which looks like this:
<command name> <key> <flags> <exptime> <bytes>\r\n	
- <command name> 是 set, add, 或者 repalce	- <command name> is “set”, “add” or “replace”
• set 意思是 “储存此数据” • add 意思是 “储存此数据，只在服务器*未*保留此键值的数据时” • replace 意思是 “储存此数据，只在服务器*	• “set” means “store this data”. • “add” means “store this data, but only if the server *doesn’t* already hold data for this key”.

曾*保留此键值的数据时”	• “replace” means “store this data, but only if the server *does* already hold data for this key”.
- <key> 是接下来的客户端所要求储存的数据的键值	- <key> is the key under which the client asks to store the data
- <flags> 是在取回内容时，与数据和发送块一同保存服务器上的任意16位无符号整形（用十进制来书写）。客户端可以用它作为“位域”来存储一些特定的信息；它对服务器是不透明的。	- <flags> is an arbitrary 16-bit unsigned integer (written out in decimal) that the server stores along with the data and sends back when the item is retrieved. Clients may use this as a bit field to store data-specific information; this field is opaque to the server.
- <exptime> 是终止时间。如果为0，该项永不过期(虽然它可能被删除，以便为其他缓存项目腾出位置)。如果非0（Unix 时间戳或当前时刻的秒偏移），到达终止时间后，客户端无法再获得这项内容。	- <exptime> is expiration time. If it’s 0, the item never expires (although it may be deleted from the cache to make place for other items). If it’s non-zero (either Unix time or offset in seconds from current time), it is guaranteed that clients will not be able to retrieve this item after the expiration time arrives (measured by server time).
- <bytes> 是随后的数据区块的字节长度，不包括用于分野的“\r\n”。它可以是0（这时后面跟随一个空的数据区块）。	- <bytes> is the number of bytes in the data block to follow, *not* including the delimiting \r\n. <bytes> may be zero (in which case it’s followed by an empty data block).
在这一行以后，客户端发送数据区块。	After this line, the client sends the data block:
<data block>\r\n	
- <data block> 是大段的8位数据，其长度由前面的命令行中的<bytes>指定。	- <data block> is a chunk of arbitrary 8-bit data of length <bytes> from the previous line.
发送命令行和数据区块以后，客户端等待回复，可能的回复如下：	After sending the command line and the data blockm the client awaits the reply, which may be:
- “STORED\r\n”	

表明成功.	to indicate success.
- “NOT_STORED\r\n”	
表明数据没有被存储，但不是因为发生错误。这通常意味着 add 或 replace 命令的条件不成立，或者，项目已经位列删除队列（参考后文的“delete”命令）。	to indicate the data was not stored, but not because of an error. This normally means that either that the condition for an “add” or a “replace” command wasn’t met, or that the item is in a delete queue (see the “delete” command below).
取回命令	**Retrieval command**
一行取回命令如下：	The retrieval command looks like this:
get <key>*\r\n	
- <key>* 表示一个或多个键值，由空格隔开的字串	- <key>* means one or more key strings separated by whitespace.
这行命令以后，客户端的等待0个或多个项目，每项都会收到一行文本，然后跟着数据区块。所有项目传送完毕后，服务器发送以下字串：	After this command, the client expects zero or more items, each of which is received as a text line followed by a data block. After all the items have been transmitted, the server sends the string
“END\r\n”	
来指示回应完毕。	to indicate the end of response.
服务器用以下形式发送每项内容：	Each item sent by the server looks like this:
VALUE <key> <flags> <bytes>\r\n <data block>\r\n	
- <key> 是所发送的键名	- <key> is the key for the item being sent
- <flags> 是存储命令所设置的记号	- <flags> is the flags value set by the storage command
- <bytes> 是随后数据块的长度，*不包括* 它的界定符“\r\n”	- <bytes> is the length of the data block to follow, *not* including its delimiting \r\n
- <data block> 是发送的数据	- <data block> is the data for this item.
如果在取回请求中发送了一些键名，而服务器没有送回项目列表，这意味着服务器没这些键名（可能因为它们从未被存储，或者为给其他内容腾出空间而被删除，或者到期，或者被已客户端删除）。	If some of the keys appearing in a retrieval request are not sent back by the server in the item list this means that the server does not hold items with such keys (because they were

	never stored, or stored but deleted to make space for more items, or expired, or explicitly deleted by a client).
删除	**Deletion**
命令“delete”允许从外部删除内容：	The command “delete” allows for explicit deletion of items:
delete <key> <time>\r\n	
- <key> 是客户端希望服务器删除的内容的键名	- <key> is the key of the item the client wishes the server to delete
- <time> 是一个单位为秒的时间（或代表直到某一刻的 Unix 时间），在该时间内服务器会拒绝对于此键名的“add”和“replace”命令。此时内容被放入 delete 队列，无法再通过“get”得到该内容，也无法是用“add”和“replace”命令（但是“set”命令可用）。直到指定时间，这些内容被最终从服务器的内存中彻底清除。	- <time> is the amount of time in seconds (or Unix time until which) the client wishes the server to refuse “add” and “replace” commands with this key. For this amount of item, the item is put into a delete queue, which means that it won’t possible to retrieve it by the “get” command, but “add” and “replace” command with this key will also fail (the “set” command will succeed, however). After the time passes, the item is finally deleted from server memory.
<time>参数 是可选的，缺省为0（表示内容会立刻清除，并且随后的存储命令均可用）。	The parameter <time> is optional, and, if absent, defaults to 0 (which means that the item will be deleted immediately and further storage commands with this key will succeed).
此命令有一行回应：	The response line to this command can be one of:
- “DELETED\r\n”	
表示执行成功	to indicate success
- “NOT_FOUND\r\n”	
表示没有找到这项内容	to indicate that the item with this key was not found.
参考随后的“flush_all”命令使所有内容无效	See the “flush_all” command below for immediate invalidation of all existing items.

增加/减少	**Increment/Decrement**
命令 “incr” 和 “decr”被用来修改数据，当一些内容需要 替换、增加 或减少时。这些数据必须是十进制的32位无符号整新。如果不是，则当作0来处理。修改的内容必须存在，当使用“incr”/“decr”命令修改不存在的内容时，不会被当作0处理，而是操作失败。	Commands “incr” and “decr” are used to change data for some item in-place, incrementing or decrementing it. The data for the item is treated as decimal representation of a 32-bit unsigned integer. If the current data value does not conform to such a representation, the commands behave as if the value were 0. Also, the item must already exist for incr/decr to work; these commands won’t pretend that a non-existent key exists with value 0; instead, they will fail.
客户端发送命令行：	The client sends the command line:
incr <key> <value>\r\n 或 decr <key> <value>\r\n	
- <key> 是客户端希望修改的内容的建名	- <key> is the key of the item the client wishes to change
- <value> 是客户端要增加/减少的总数。	- <value> is the amount by which the client wants to increase/decrease the item. It is a decimal representation of a 32-bit unsigned integer.
回复为以下集中情形：	The response will be one of:
- “NOT_FOUND\r\n”	
指示该项内容的值，不存在。	to indicate the item with this value was not found
- <value>\r\n ， <value>是 增加/减少 。	- <value>\r\n , where <value> is the new value of the item’s data, after the increment/decrement operation was carried out.
注意”decr”命令发生下溢：如果客户端尝试减少的结果小于0时，结果会是0。”incr” 命令不会发生溢出。	Note that underflow in the “decr” command is caught: if a client tries to decrease the value below 0, the new value will be 0. Overflow in the “incr” command is not checked.
……	Note also that decrementing a number such that it loses length isn’t

	guaranteed to decrement its returned length. The number MAY be space-padded at the end, but this is purely an implementation optimization, so you also shouldn't rely on that.
状态	**Statistics**
命令"stats" 被用于查询服务器的运行状态和其他内部数据。有两种格式。不带参数的：	The command "stats" is used to query the server about statistics it maintains and other internal data. It has two forms. Without arguments:
stats\r\n	
这会在随后输出各项状态、设定值和文档。另一种格式带有一些参数：	it causes the server to output general-purpose statistics and settings, documented below. In the other form it has some arguments:
stats <args>\r\n	
通过<args>，服务器传回各种内部数据。因为随时可能发生变动，本文不提供参数的种类及其传回数据。	Depending on <args>, various internal data is sent by the server. The kinds of arguments and the data sent are not documented in this vesion of the protocol, and are subject to change for the convenience of memcache developers.
各种状态	**General-purpose statistics**
受到无参数的"stats"命令后，服务器发送多行内容，如下：	Upon receiving the "stats" command without arguments, the server sents a number of lines which look like this:
STAT <name> <value>\r\n	
服务器用以下一行来终止这个清单：	The server terminates this list with the line
END\r\n	
在每行状态中，<name> 是状态的名字，<value> 使状态的数据。 以下清单，是所有的状态名称，数据类型，和数据代表的含义。	In each line of statistics, <name> is the name of this statistic, and <value> is the data. The following is the list of all names sent in response to the "stats" command, together with the type of the value

	sent for this name, and the meaning of the value.
在“类型”一列中，”32u”表示32位无符号整型，”64u”表示64位无符号整型，”32u:32u”表示用冒号隔开的两个32位无符号整型。	In the type column below, “32u” means a 32-bit unsigned integer, “64u” means a 64-bit unsigner integer. ‘32u:32u’ means two 32-but unsigned integers separated by a colon.

名称/Name	类型/Type	含义/Meaning	
pid	32u	服务器进程ID	Process id of this server process
uptime	32u	服务器运行时间，单位秒	Number of seconds this server has been running
time	32u	服务器当前的UNIX时间	current UNIX time according to the server
version	string	服务器的版	Version string of this server

		本号	
rusage_user	32u:32u	该进程累计的用户时间(秒:微妙)	Accumulated user time for this process (seconds:microseconds)
rusage_system	32u:32u	该进程累计的系统时间(秒:微妙)	Accumulated system time for this process (seconds:microseconds)
curr_items	32u	服务器当前存储的内容数量	Current number of items stored by the server
total_items	32u	服务	Total number of items stored by this server

		器启动以来存储过的内容总数	ever since it started
bytes	64u	服务器当前存储内容所占用的字节数	Current number of bytes used by this server to store items
curr_connections	32u	连接数量	Number of open connections
total_connections	32u	服务器运行以来接受	Total number of connections opened since the server started running

		的连接总数	
connection_structures	32u	服务器分配的连接结构的数量	Number of connection structures allocated by the server
cmd_get	32u	取回请求总数	Cumulative number of retrieval requests
cmd_set	32u	存储请求总数	Cumulative number of storage requests
get_hits	32u	请求成功的总次数	Number of keys that have been requested and found present
get_misses	32u	请求失	Number of items that have been requested and not found

		败的总次数	
bytes_read	64u	服务器从网络读取到的总字节数	Total number of bytes read by this server from network
bytes_written	64u	服务器向网络发送的总字节数	Total number of bytes sent by this server to network
limit_maxbytes	32u	服务器在存储时被允许	Number of bytes this server is allowed to use for storage.

其它命令	**Other commands**
“flush_all”命令有一个可选的数字参数。它总是执行成功，服务器会发送“OK\r\n”回应。它的效果是使已经存在的项目立即失效（缺省），或在指定的时间后。此后执行取回命令，将不会有任何内容返回（除非重新存储同样的键名）。flush_all 实际上没有立即释放项目所占用的内存，而是在随后陆续有新的项目被储存时执行。flush_all 效果具体如下：它导致所有更新时间早于 flush_all 所设定时间的项目，在被执行取回命令时命令被忽略。	“flush_all” is a command with an optional numeric argument. It always succeeds, and the server sends “OK\r\n” in response. Its effect is to invalidate all existing items immediately (by default) or after the expiration specified. After invalidation none of the items will be returned in response to a retrieval command (unless it's stored again under the same key *after* flush_all has invalidated the items). flush_all doesn't actually free all the memory taken up by existing items; that will happen gradually as new items are stored. The most precise definition of what flush_all does is the following: it causes all items whose update time is earlier than the time at which flush_all was set to be executed to be ignored for retrieval purposes.
“version”命令没有参数：	“version” is a command with no arguments:
version\r\n	
在回应中，服务器发送：	In response, the server sends
“VERSION <version>\r\n”	
<version> 是服务器的版本字串。	where <version> is the version string for the server.
“quit”命令没有参数：	“quit” is a command with no arguments:
quit\r\n	
接收此命令后，服务器关闭连接。不过，客户端可以在不再需要时，简单地关闭连接就行，并不一定需要发送这个命令。	Upon receiving this command, the server closes the connection. However, the client may also simply close the connection when it no longer needs it, without issuing this command.

UDP 协议	**UDP protocol**
当来自客户端的连接数远大于 TCP 连接的上限时，可以使用基于 UDP 的接口。UDP 接口不能保证传输到位，所以只有在不要求成功的操作中使用；比如被用于一个“get”请求时，会因不当的缓存处理而发生错误或回应有遗失。	For very large installations where the number of clients is high enough that the number of TCP connections causes scaling difficulties, there is also a UDP-based interface. The UDP interface does not provide guaranteed delivery, so should only be used for operations that aren’t required to succeed; typically it is used for “get” requests where a missing or incomplete response can simply be treated as a cache miss.
每个 UDP 数据包都包含一个简单的帧头，数据之后的内容与 TCP 协议的描述类似。在执行所产生的数据流中，请求必须被包含在单独的一个 UDP 数据包中，但是回应可能跨越多个数据包。（只有“get”和“set”请求例外，跨越了多个数据包）	Each UDP datagram contains a simple frame header, followed by data in the same format as the TCP protocol described above. In the current implementation, requests must be contained in a single UDP datagram, but responses may span several datagrams. (The only common requests that would span multiple datagrams are huge multi-key “get” requests and “set” requests, both of which are more suitable to TCP transport for reliability reasons anyway.)
帧头有8字节长，如下（均由16位整数组成，网络字节顺序，高位在前）：	The frame header is 8 bytes long, as follows (all values are 16-bit integers in network byte order, high byte first):
• 0-1 请求 ID • 2-3 序号 • 4-5 该信息的数据包总数 • 6-7 保留位，必须为0	• 0-1 Request ID • 2-3 Sequence number • 4-5 Total number of datagrams in this message • 6-7 Reserved for future use; must be 0
请求 ID 有客户端提供。一般它会是一个从随机基数开始的递增值，不过客户端想用什么样的请求 ID 都可以。服务器的回应会包含一个和请求中的同样的 ID。客户端使用请求 ID 来区分每一个回应。任何一个没有请求 ID 的数据包，可能是之前的请求遭到延迟而造成的，应该被丢弃。	The request ID is supplied by the client. Typically it will be a monotonically increasing value starting from a random seed, but the client is free to use whatever request IDs it likes. The server’s response will contain the same ID as the incoming request. The client uses the request ID to differentiate between responses to outstanding requests if there are several pending from the same server; any datagrams with an unknown request ID are probably delayed responses to an earlier request and should be discarded.

序号的返回是从0到 n-1，n 是该条信息的数据包数量。	The sequence number ranges from 0 to n-1, where n is the total number of datagrams in the message. The client should concatenate the payloads of the datagrams for a given response in sequence number order; the resulting byte stream will contain a complete response in the same format as the TCP protocol (including terminating \r\n sequences).

第五章 Memcached 的统计与监控

使用 telnet 连接 memcached，发送统计命令：stats。

```
shell>telnet 192.168.228.3 9999
        Trying 192.168.228.3...
        Connected to 192.168.228.3.
        Escape character is '^]'.

stats
STAT pid 6995
STAT uptime 87233
STAT time 1222314531
STAT version 1.2.6
STAT pointer_size 32
STAT rusage_user 0.081987
STAT rusage_system 0.246962
STAT curr_items 1000
STAT total_items 3932
STAT bytes 65000
STAT curr_connections 2
STAT total_connections 452
STAT connection_structures 129
STAT cmd_get 7980
STAT cmd_set 3990
STAT get_hits 4975
STAT get_misses 3005
STAT evictions 0
STAT bytes_read 291486
```

```
STAT bytes_written 235479
STAT limit_maxbytes 2147483648
STAT threads 1
END
```

分析统计信息

名称	描述
pid	Memcached 进程 ID
uptime	Memcached 运行时间，单位：秒
time	Memcached 当前的 UNIX 时间
version	Memcached 的版本号
rusage_user	该进程累计的用户时间，单位：秒
rusage_system	该进程累计的系统时间，单位：秒
curr_items	Memcached 当前存储的内容数量
total_items	Memcached 启动以来存储过的内容总数
bytes	Memcached 当前存储内容所占用的字节数
curr_connections	当前连接数量
total_connections	Memcached 运行以来接受的连接总数
connection_structures	Memcached 分配的连接结构的数量
cmd_get	查询请求总数
cmd_set	存储（添加/更新）请求总数
get_hits	查询成功获取数据的总次数
get_misses	查询成功未获取到数据的总次数
bytes_read	Memcached 从网络读取到的总字节数
bytes_written	Memcached 向网络发送的总字节数
limit_maxbytes	Memcached 在存储时被允许使用的字节总数

<正文结束>

监控

最近根据程序员需要在一台服务器上面部署了 memcached 服务，虽然用上了，但是对他还不是非常了解。于是开始收集整理他的相关资料，其中一部分就是对他的监控了。

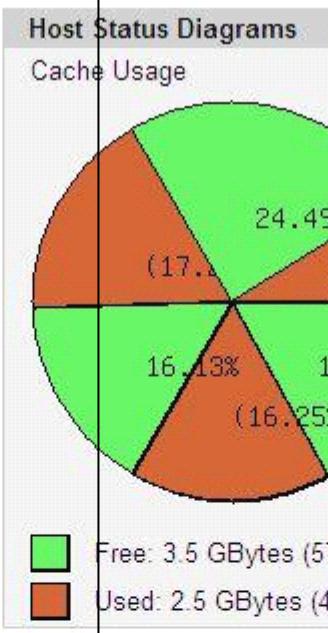
目前我所知道的监控方法大概有以下几种：

一、memcache.php 这个东东算是最简单的了，只要支持 php 环境就能用，把这个文件放到你的网页存放目录下就可以了访问方法 http://ip/memcache.php

Refresh Data | View Host Stats | Variables

General Cache Information	
PHP Version	5.2.10
Memcached Hosts	1. 127.0.0.1:11211 2. 127.0.0.1:11212 3. 127.0.0.1:11213
Total Memcache Cache	6.0 GBytes

Memcache Server Information	
127.0.0.1:11211	[Flush this server]
Start Time	2009/12/12 14:33:19
Uptime	5 days, 44 minutes
Memcached Server Version	1.4.3
Used Cache Size	998.3 MBytes
Total Cache Size	2.0 GBytes
127.0.0.1:11212	[Flush this server]
Start Time	2009/12/12 14:33:27
Uptime	5 days, 43 minutes
Memcached Server Version	1.4.3
Used Cache Size	1.0 GBytes
Total Cache Size	2.0 GBytes
127.0.0.1:11213	[Flush this server]
Start Time	2009/12/12 15:02:57
Uptime	5 days, 14 minutes
Memcached Server Version	1.4.3
Used Cache Size	543.3 MBytes
Total Cache Size	2.0 GBytes



Cache Information
Current Items(total)
Hits
Misses
Request Rate (hits, misse
Hit Rate
Miss Rate
Set Rate

下载地址 http://livebookmark.net/memcachephp/memcachephp.zip

http://blogimg.chinaunix.net/blog/upfile2/081230231118.zip

需要注意的是，使用之前要修改里面的几个选项

vim memcache.php

.....

define('ADMIN_USERNAME','memcache'); // 定义用户名

define('ADMIN_PASSWORD','password'); // 定义密码

.....

$MEMCACHE_SERVERS[] = 'mymemcache-server:11211'; //定义要查看的 ip 和端口

$MEMCACHE_SERVERS[] = 'mymemcache-server2:11212'; //可添加多个

其他内容略

上面的图就是访问时的效果，是不是很直观？

二、利用 memcached 自身的命令来检查

telnet localhost 11211

Trying 127.0.0.1...

Connected to localhost.localdomain (127.0.0.1).

Escape character is '^]'.

stats

STAT pid 24567

STAT uptime 6576

STAT time 1261035123

STAT version 1.4.3

.....

STAT bytes 64035

STAT curr_items 41

STAT total_items 96

STAT evictions 0

END

不过这个方法我觉得不太方便，从网上搜到了另一个好办法，那就是利用 nagios 的 check_tcp

(mixi 的方法)

check_tcp -H localhost -p 11211 -t 5 -E -s 'stats\r\nquit\r\n' -e 'uptime' -M crit 输出结果和上面差

不多

TCP OK - 0.001 second response time on port 11211 [STAT pid 10663

STAT uptime 76444

STAT time 1259641750

STAT version 1.4.2

.....中间省略

STAT bytes 1385408560

STAT curr_items 227799

STAT total_items 5012750

STAT evictions 0

END]|time=0.001142s;;;0.000000;5.000000

这样我们就能在 nagios 里面添加命令来查看他的运行状态了

不过这样还不够，我还需要在 memcached 出现问题的时候通过邮件或者短信来通知我，下面来介绍一个更好的办法

三、Nagios 的 check_memcached

下载地址：

http://search.cpan.org/CPAN/authors/id/Z/ZI/ZIGOROU/Nagios-Plugins-Memcached-0.02.tar.gz

http://cpan.uwinnipeg.ca/cpan/authors/id/Z/ZI/ZIGOROU/Nagios-Plugins-Memcached-0.02.tar.gz

这个脚本是用 perl 编的，所以你要先确保自己的机器里面是否有 perl 环境，不过一般都会默认安装

[root@nodea soft]# which perl

/usr/bin/perl

下载下来后安装

[root@nodea soft]# tar xzvf Nagios-Plugins-Memcached-0.02.tar.gz

[root@nodea soft]# cd Nagios-Plugins-Memcached-0.02

[root@nodea Nagios-Plugins-Memcached-0.02]# perl Makefile.PL

执行后会出现一些提示让你选择，按照自己想法选或者一路回车都能通过

[root@nodea Nagios-Plugins-Memcached-0.02]# make

这时他会下载一些运行时需要的东西

[root@nodea Nagios-Plugins-Memcached-0.02]# make install

默认会把 check_memcached 文件放到/usr/bin/check_memcached

没关系 把他拷贝到 nagios 的 libexec 下

在 commands.cfg 里面加上这么几条（这里我没有把 check_memcached 装在 memcached 服务器上，而是通过 Nagios 的 check_memcached 直接去访问 memcached 服务器的 11211 端口，当然你也可以把他装在 memcached 服务器上利用 check_nrpe 来取他的值）

```
define command {
        command_name check_memcached_11211
            command_line  $USER1$/check_memcached  -H  192.168.1.139:11211  --size-warning  80
--size-critical 90
            }
```

上面这个是来监控 memcached 的内存使用比例

```
define command {
        command_name memcached_response_11211
command_line /usr/local/bin/check_memcached -H 192.168.1.139 -w 300 -c 500
            }
```

这个是用来监控 memcached 是否还有应答

```
define command {
        command_name check_memcached_hit
              command_line  /usr/local/bin/check_memcached  -H  192.168.1.139  --hit-warning  10
--size-critical 5
            }
```

这个就是命中率啦

最后要在 services.cfg 里面加点东西

```
define service{
        host_name               babeltimeWeb1
        service_description     memcached_11211
        check_command           check_memcached_11211
        max_check_attempts      3
        normal_check_interval   3
        retry_check_interval    2
        check_period            24x7
        notification_interval   10
        notification_period     24x7
        notification_options    w,u,c,r
        contact_groups          babelgroup
        }
```

其他的可以按照自己要求添加..

好了，让我们重新启动下 nagios 服务

```
/etc/init.d/nagios restart
```

看看结果

memcached_11211	OK	12-17-2009 16:04:46	1d 20h 42m 52s	1/3	MEMCACHED OK - OK, Size checked: OK
memcached_11212	OK	12-17-2009 16:05:22	1d 20h 42m 16s	1/3	MEMCACHED OK - OK, Size checked: OK
memcached_11213	OK	12-17-2009 16:02:58	1d 20h 41m 40s	1/3	MEMCACHED OK - OK, Size checked: OK

呵呵好了，Nagios 监控 memcached 基本就搞定了。

另外还可以根据 check_tcp -H localhost -p 11211 -t 5 -E -s 'stats\r\nquit\r\n' -e 'uptime' -M crit

的输出结果自己编写脚本来检测 memcached，这里我就不多说了...

还可以利用 check_tcp 的结果结合 CACTI 来制作 memcached 的流量视图，当然 Cacti 也有

专门针对 memcached 的模板(不过我的模板一直抓不到数据...)

将 MySQL 数据映射到 Memcached 中

出
处:http://www.libing.name/2009/02/06/mysql-map-data-to-memcachedmysql-map-data-to-memcached.html

差不多在一年前，写过一篇文章介绍将 MySQL 数据映射到 Memcached，当时 MySQL 和 Memcached Functions for MySQL 都还不够成熟，时过一年，Memcached Functions for MySQL 升级到了 0.8 版本，而 MySQL 也发布了 GA 版本，加上很多朋友反应前一篇文章中的实现他们因种种原因没能成功，于是便有了这篇文章，就当是上一篇文章的升级版本吧。

测试环境在 Linux 下进行，版本系统为 CentOS5.
以下为相关软件，包括其版本和下载地址：

mysql-5.1.30 下载
memcached-1.2.6 下载
libevent-1.4.7-stable 下载
memcached_functions_mysql-0.8 下载
libmemcached-0.26 下载

编译安装 MySQL,安装因个人细好而定，省略许多与测试无关的编译细节及参数。

```
[root@localhost ~]#tar xzf mysql-5.1.30.tar_001.gz
[root@localhost ~]#cd mysql-5.1.30
[root@localhost ~]#./configure --prefix=/usr/local/mysql51
[root@localhost ~]#make
[root@localhost ~]#make install
[root@localhost ~]#./scripts/mysql_install_db --user=mysql
--skip-name-resolve
[root@localhost ~]#/usr/local/mysql51/bin/mysqld_safe
```

省略列出安装 memcached 和 libevent 的相关命令，具体可按照实际情况安装，测试时我将 libevent 默认安装，memcached 安装于/usr/local/memcached 目录下。
启动 memcached.

```
/usr/local/memcached/bin/memcached -d -m 50 -u root -p 11211
```

编译安装 libmemcache.

```
[root@localhost ~]#tar xzf libmemcached-0.26.tar.gz
[root@localhost ~]#cd libmemcached-0.26
[root@localhost ~]#./configure
--with-memcached=/usr/local/memcached/bin/memcached
[root@localhost ~]# make && make install
```

编译安装 Memcache UDFs for MySQL.

```
[root@localhost ~]# tar xzf
memcached_functions_mysql-0.8.tar.gz
[root@localhost ~]# cd memcached_functions_mysql-0.8
[root@localhost ~]# ./configure
--with-mysql-config=/usr/local/mysql51/bin/mysql_config
[root@localhost ~]# make && make install
```

编译完成后将编译好的库文件复制到 mysql 的插件目录下，以便于加载使用。

```
cp /usr/local/lib/libmemcached_functions_mysql*
/usr/local/mysql51/lib/mysql/plugin/
```

进入 memcached_functions_mysql 的源码目录，在目录下有相关添加 UDF 的 SQL 文件用于初始化。

```
[root@localhost ~]# mysql <sql/install_functions.sql
```

注：如果对这些 UDFs 不熟悉或者不懂，可进行源码目录参看 README，里边有相应的说明。

至此，相关软件的编译和安装完成，进行测试，我们要达到的目的是当 MySQL 有新记录插入时，同时插入到 Memcached 中，当记录更新时同步更新 Memcached 中的记录，删除时同时也删除 Memcached 相关的记录，为此创建三个触发器来实现，如果对 MySQL 的触发程序不熟悉可以参考 MySQL 手册第 21 章，下面 SQL 中的 memcached 为需要操作的表名，SQL 如下：

```
#插入数据时插入 Memcached

create trigger mysqlmmci after insert on memcached for each
row set @tmp = memc_set(NEW.key, NEW.value);

#更新记录时更新 Memcached

create trigger mysqlmmcu after update on memcached for each
row set @tmp = memc_set(NEW.key, NEW.value);

#删除记录时删除 Memcached 相应的记录

create trigger mysqlmmcd before delete on memcached for each
row set @tmp = memc_delete(OLD.key);
```

以下为测试记录，在对 MySQL 操作的同时操作 Memcached 来查看情况，当然你也可以在启动 Memcached 的时候带-vv 参数来查看相关信息.

MySQL 操作相关的记录：

```
[root@localhost ~]#mysql -S /tmp/mysql51.sock
```

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 5.1.30 Source distribution

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

mysql> use test;
Database changed

mysql> create table `memcached` (`key` varchar(10), `value`
varchar(100));
Query OK, 0 rows affected (0.00 sec)

mysql> create trigger mysqlmmci after insert on memcached for
each row set @tmp = memc_set(NEW.key, NEW.value);
Query OK, 0 rows affected (0.00 sec)

mysql> create trigger mysqlmmcu after update on memcached for
each row set @tmp = memc_set(NEW.key, NEW.value);
Query OK, 0 rows affected (0.00 sec)

mysql> create trigger mysqlmmcd before delete on memcached
for each row set @tmp = memc_delete(OLD.key);
Query OK, 0 rows affected (0.00 sec)

mysql> insert into memcached values("keyi",
"valuei"),("keyu","valueu"),("keyd", "valued");
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> update memcached set `value`="update" where
`key`="keyu";
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> delete from memcached where `key`="keyd";
Query OK, 1 row affected (0.00 sec)

mysql> quit
Bye
```

Memcache 查看时的记录：

```
[root@localhost ~]#telnet 127.0.0.1 11211
Trying 127.0.0.1...
Connected to 127.0.0.1.
```

```
Escape character is '^]'.
get keyi
VALUE keyi 0 6
valuei
END
get keyu
VALUE keyu 0 6
valueu
END
get keyd
VALUE keyd 0 6
valued
END
get keyu
VALUE keyu 0 6
update
END
get keyd
END
quit
Connection closed by foreign host.
```

至此，我们基本实现的将 MySQL 的数据同步到 Memcached 中，性能暂时还没有测试，当然上面只是简单的实现的数据映射的功能，如果在实现的生产环境中，则需要考虑名字空间，高可靠性的问题，这些都是可以通过数据库名-表名-关键字的方面能达到 KEY 唯一的目的，而高可靠性则是一个比较大的问题。

第六章 memcached 的启动脚本

```
#!/bin/sh
# memcached:       MemCached Daemon
# chkconfig:       - 90 25
# description:     MemCached Daemon
# Source function library.
. /etc/rc.d/init.d/functions
. /etc/sysconfig/network
#[ ${NETWORKING} = "no" ] && exit 0
#[ -r /etc/sysconfig/dund ] || exit 0
#. /etc/sysconfig/dund
#[ -z "$DUNDARGS" ] && exit 0
start()
  {
```

```
        echo -n $"Starting memcached: "
        daemon $MEMCACHED -u root -d -m 512 -l 127.0.0.1 -p 11211 -c 2048 -P
/var/run/memcached.pid
        echo
 }
stop()
 {
        echo -n $"Shutting down memcached: "
        killproc memcached
        echo
 }
MEMCACHED="/usr/local/memcached/bin/memcached"
[ -f $MEMCACHED ] || exit 1
#See how we were called.
case "$1" in
  start)
        start
        ;;
  stop)
        stop
        ;;
  restart)
        stop
        sleep 3
        start
        ;;
    *)
        echo $"Usage: $0 {start|stop|restart}"
        exit 1
        esac
exit 0
```

```
##################################################
chkconfig   --add memcached
chkconfig   --level 235   memcached   on
chkconfig   --list | grep memcached
```

第七章memcached 变种产品介绍

国内外有很多基于 Memcached 开发的产品，这些产品支持所有 Memcached 的协议，同时

侧重不同的应用场景，可以根据自己的应用需求选择合适的 Memcached 变种。下面分别介绍几种 Memcached 的变种产品。

1. memcachedb

memcachedb 是新浪网基于 Memcached 开发的一个开源项目。通过为 Memcached 增加 Berkeley DB 的持久化存储机制和异步主辅复制机制，使 Memcached 具备了事务恢复能力、持久化能力和分布式复制能力，非常适合需要超高性能读写速度、持久化保存的应用场景，例如，将 memcachedb 应用于新浪博客的管理。如果对 Memcached 有持久化需求，可以考虑使用 memcachedb。

2. repcached

repcached 是日本人开发的基于 Memcached 的一个 patch，实现 Memcached 的复制功能，它支持多个 Memcached 之间相互复制，可以解决 Memcached 的容灾问题。有 cache 容灾需求的可以尝试使用这一功能。

3. memcached_functions_mysql

这个功能相当于 MySQL 的 UDFs（User Defined Functions），在 MySQL 中通过触发器更新 Memcached。这样可以做到把数据写入 MySQL，然后从 Memcached 获取数据，以减轻数据库的压力，同时减少很多开发的工作量。

关于 memcached_functions_mysql 的使用和经验会在下一节进行详细介绍。

4. memcacheQ

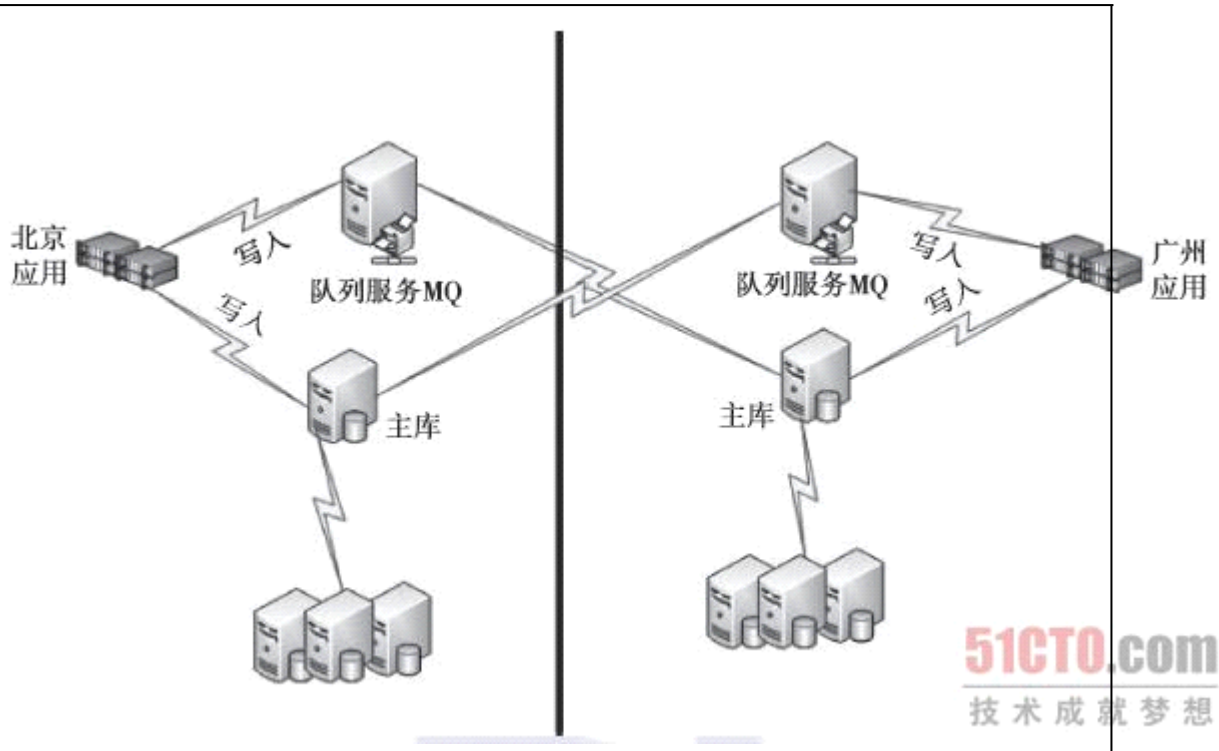
memcacheQ 在 Memcached 的基础上实现了消息队列。下面以 PHP 客户端为例介绍 memcacheQ 实现消息队列的方式。

消息从尾部入栈：memcache_set

消息从头部出栈：memcache_get

memcacheQ 最大的优势是：它是基于 Memcached 开发的，可以通过各种 Memcached 命令对它进行操作。基于 Memcached 开发的应用完全不需要做任何修改。

memcacheQ 应用于数据库的多机房分布式部署环境，数据库分布在各地，各自写各自的数据库，同时写入本地的 memcacheQ，本地的 memcacheQ 通过调度程序（需要自己开发）把数据从 memcacheQ 中读取出来，然后写入其他机房的数据库主库，最终使每个机房的数据库数据一致。如图3-14所示为 memcacheQ 在两个机房的部署情况。



消息队列服务还能使一个有波峰的业务转化成一条直线，这对利用资源非常有好处，只需要准备直线的资源，不需要准备到波峰的资源。Twitter 之前通过 RabbitMQ 来实现消息队列服务，现在改为通过 Kestrel 来实现消息队列服务，类似的消息队列服务产品还有 ActiveMQ 和 ZeroMQ 等。

5 memcached_functions_mysql 的安装应用

UDFs 是 User Defined Functions 的缩写，表示 MySQL 的用户定义函数，应用程序可以利用这些函数从 MySQL 5.0 以上版本的数据库中访问 Memcached 写入或者获取的数据。此外，MySQL 从 5.1 版本开始支持触发器，从而可以在触发器中使用 UDFs 直接更新 Memcached 的内容，这种方式降低了应用程序设计和编写的复杂性。下面简单介绍 UDFs 的安装和使用。

第一部分 下载源码进行安装

```
wget
https://github.com/downloads/libevent/libevent/libevent-2.0.19-stable.tar.
gz
tar xvf libevent-2.0.19-stable.tar.gz
cd libevent-2.0.19-stable
./configure
make
make install

wget http://memcached.googlecode.com/files/memcached-1.4.13.tar.gz
tar xvf memcached-1.4.13.tar.gz
```

```
cd memcached-1.4.13
./configure
make
make install

echo "/usr/local/lib" >> /etc/ld.so.conf
ldconfig
/usr/local/bin/memcached  -m 32m -p 11211 -d -u root -P /var/run/memcached.pid
-c 1024

wget
https://launchpad.net/libmemcached/1.0/0.34/+download/libmemcached-0.34.t
ar.gz
tar xvf libmemcached-0.34.tar.gz
cd libmemcached-0.34
./configure --with-memcached=/usr/local/bin/memcached
i#./configure                              --prefix=/usr/local/libmemcached
--with-memcached=/usr/local/bin/memcached
make
make install

#wget
https://launchpad.net/libmemcached/1.0/1.0.8/+download/libmemcached-1.0.8.
tar.gz
#wget
https://launchpad.net/memcached-udfs/trunk/version-1.0/+download/memcache
d_functions_mysql-1.0.tar.gz
wget
https://launchpad.net/memcached-udfs/trunk/1.1/+download/memcached_functi
ons_mysql-1.1.tar.gz
tar xvf memcached_functions_mysql-1.1.tar.gz
cd memcached_functions_mysql-1.1
./configure                --with-mysql=/usr/local/mysql/bin/mysql_config
--libdir=/usr/local/mysql/lib/mysql --with-libmemcached=/usr/local
make
make install
cd utils/ #注意此处的 my.cnf 必须为 osocket=/var/lib/mysql/mysql.sock,否则，运行
脚本会报错  #find  /usr/  -name  libmemcached_functions_mysql.so  ln  -s
/usr/local/mysql/lib/mysql/libmemcached_functions_mysql.so
/usr/local/mysql/lib/plugin/libmemcached_functions_mysql.so   ./install.pl
-s -u=root -p=itnihao
```

第二部分 在 mysql 进行设置 memcached

```
mysql> select name,dl from mysql.func;
```

```
+------------------------------+---------------------------------+
| name                         | dl                              |
+------------------------------+---------------------------------+
| memc_cas_by_key              | libmemcached_functions_mysql.so |
| memc_cas                     | libmemcached_functions_mysql.so |
| memc_servers_set             | libmemcached_functions_mysql.so |
| memc_add                     | libmemcached_functions_mysql.so |
| memc_libmemcached_version    | libmemcached_functions_mysql.so |
| memc_add_by_key              | libmemcached_functions_mysql.so |
| memc_server_count            | libmemcached_functions_mysql.so |
| memc_stat_get_keys           | libmemcached_functions_mysql.so |
| memc_append                  | libmemcached_functions_mysql.so |
| memc_replace_by_key          | libmemcached_functions_mysql.so |
| memc_prepend                 | libmemcached_functions_mysql.so |
| memc_behavior_get            | libmemcached_functions_mysql.so |
| memc_udf_version             | libmemcached_functions_mysql.so |
| memc_set_by_key              | libmemcached_functions_mysql.so |
| memc_get_by_key              | libmemcached_functions_mysql.so |
| memc_increment               | libmemcached_functions_mysql.so |
| memc_behavior_set            | libmemcached_functions_mysql.so |
| memc_stats                   | libmemcached_functions_mysql.so |
| memc_list_distribution_types | libmemcached_functions_mysql.so |
| memc_list_hash_types         | libmemcached_functions_mysql.so |
| memc_append_by_key           | libmemcached_functions_mysql.so |
| memc_servers_behavior_set    | libmemcached_functions_mysql.so |
| memc_replace                 | libmemcached_functions_mysql.so |
| memc_set                     | libmemcached_functions_mysql.so |
| memc_prepend_by_key          | libmemcached_functions_mysql.so |
| memc_get                     | libmemcached_functions_mysql.so |
| memc_list_behaviors          | libmemcached_functions_mysql.so |
| memc_delete                  | libmemcached_functions_mysql.so |
| memc_stat_get_value          | libmemcached_functions_mysql.so |
| memc_decrement               | libmemcached_functions_mysql.so |
| memc_delete_by_key           | libmemcached_functions_mysql.so |
| memc_servers_behavior_get    | libmemcached_functions_mysql.so |
+------------------------------+---------------------------------+
```

第三部分 memcached 的应用实例

```
#对功能进行测试，源码包已经提供测试模块
cat memcached_functions_mysql-1.1/sql/trigger_fun.sql
mysql> use test
drop table if exists urls;
create table urls (
  id int(3) not null,
```

```
  url varchar(64) not null default '',
  primary key (id)
  );


mysql> select memc_servers_set('localhost:11211');
+-------------------------------------+
| memc_servers_set('localhost:11211') |
+-------------------------------------+
|                                   0 |
+-------------------------------------+

mysql> select memc_set('urls:sequence', 0);
+------------------------------+
| memc_set('urls:sequence', 0) |
+------------------------------+
|                            1 |
+------------------------------+

mysql>DELIMITER |

DROP TRIGGER IF EXISTS url_mem_insert |
CREATE TRIGGER url_mem_insert
BEFORE INSERT ON urls
FOR EACH ROW BEGIN
    SET NEW.id= memc_increment('urls:sequence');
    SET @mm= memc_set(concat('urls:',NEW.id), NEW.url);
END |

DROP TRIGGER IF EXISTS url_mem_update |
CREATE TRIGGER url_mem_update
BEFORE UPDATE ON urls
FOR EACH ROW BEGIN
    SET @mm= memc_replace(concat('urls:',OLD.id), NEW.url);
END |

DROP TRIGGER IF EXISTS url_mem_delete |
CREATE TRIGGER url_mem_delete
BEFORE DELETE ON urls
FOR EACH ROW BEGIN
    SET @mm= memc_delete(concat('urls:',OLD.id));
END |

DELIMITER ;
```

```
insert into urls (url) values ('http://google.com');
insert into urls (url) values ('http://lycos.com/');
insert into urls (url) values ('http://tripod.com/');
insert into urls (url) values ('http://microsoft.com/');
insert into urls (url) values ('http://slashdot.org');
insert into urls (url) values ('http://mysql.com');
select * from urls;

select memc_get('urls:1');
select memc_get('urls:2');
select memc_get('urls:3');
select memc_get('urls:4');
select memc_get('urls:5');
select memc_get('urls:6');

update urls set url= 'http://mysql.com/sun' where url = 'http://mysql.com';
select url from urls where url = 'http://mysql.com/sun';
select memc_get('urls:6');

delete from urls where url = 'http://microsoft.com/';
select * from urls where url='http://microsoft.com/';
select memc_get('urls:4');




mysql> select memc_servers_set('192.168.16.21:11211');
+-----------------------------------------+
| memc_servers_set('192.168.16.21:11211') |
+-----------------------------------------+
|                                       0 |
+-----------------------------------------+

mysql> select memc_server_count();
+---------------------+
| memc_server_count() |
+---------------------+
|                   1 |
+---------------------+

mysql> select memc_list_behaviors()\G
*************************** 1. row ***************************
memc_list_behaviors():
```

```
MEMCACHED_SERVER_BEHAVIORS
MEMCACHED_BEHAVIOR_SUPPORT_CAS
MEMCACHED_BEHAVIOR_NO_BLOCK
MEMCACHED_BEHAVIOR_TCP_NODELAY
MEMCACHED_BEHAVIOR_HASH
MEMCACHED_BEHAVIOR_CACHE_LOOKUPS
MEMCACHED_BEHAVIOR_SOCKET_SEND_SIZE
MEMCACHED_BEHAVIOR_SOCKET_RECV_SIZE
MEMCACHED_BEHAVIOR_BUFFER_REQUESTS
MEMCACHED_BEHAVIOR_KETAMA
MEMCACHED_BEHAVIOR_POLL_TIMEOUT
MEMCACHED_BEHAVIOR_RETRY_TIMEOUT
MEMCACHED_BEHAVIOR_DISTRIBUTION
MEMCACHED_BEHAVIOR_BUFFER_REQUESTS
MEMCACHED_BEHAVIOR_USER_DATA
MEMCACHED_BEHAVIOR_SORT_HOSTS
MEMCACHED_BEHAVIOR_VERIFY_KEY
MEMCACHED_BEHAVIOR_CONNECT_TIMEOUT
MEMCACHED_BEHAVIOR_KETAMA_WEIGHTED
MEMCACHED_BEHAVIOR_KETAMA_HASH
MEMCACHED_BEHAVIOR_BINARY_PROTOCOL
MEMCACHED_BEHAVIOR_SND_TIMEOUT
MEMCACHED_BEHAVIOR_RCV_TIMEOUT
MEMCACHED_BEHAVIOR_SERVER_FAILURE_LIMIT
MEMCACHED_BEHAVIOR_IO_MSG_WATERMARK
MEMCACHED_BEHAVIOR_IO_BYTES_WATERMARK

MEMCACHED_HASH_DEFAULT
MEMCACHED_HASH_MD5
MEMCA



mysql>                                                          select
memc_servers_behavior_set('MEMCACHED_BEHAVIOR_NO_BLOCK','1');
+-------------------------------------------------------------+
| memc_servers_behavior_set('MEMCACHED_BEHAVIOR_NO_BLOCK','1') |
+-------------------------------------------------------------+
|                                                           0 |
+-------------------------------------------------------------+

mysql>                                                          select
memc_servers_behavior_set('MEMCACHED_BEHAVIOR_TCP_NODELAY','1');
+---------------------------------------------------------------+
| memc_servers_behavior_set('MEMCACHED_BEHAVIOR_TCP_NODELAY','1') |
```

```
+-------------------------------------------------------------------+
|                                                                 0 |
+-------------------------------------------------------------------+
```

测试

```
#向表 urls 中插入数据，然后查看 memcached 是否对数据执行 set 操作。
mysql> insert into urls (url) values ('http://google.com');
mysql> insert into urls (url) values ('http://lycos.com/');
mysql> insert into urls (url) values ('http://tripod.com/');
mysql> insert into urls (url) values ('http://microsoft.com/');
mysql> insert into urls (url) values ('http://slashdot.org');
mysql> insert into urls (url) values ('http://mysql.com');
mysql> select * from urls;

mysql> select memc_get('urls:8');
+--------------------+
| memc_get('urls:8') |
+--------------------+
| http://google.com  |
+--------------------+
1 row in set (0.00 sec)

mysql> select memc_get('urls:9');
+--------------------+
| memc_get('urls:9') |
+--------------------+
| NULL               |
+--------------------+
1 row in set (0.05 sec)

mysql> select memc_get('urls:10');
+---------------------+
| memc_get('urls:10') |
+---------------------+
| http://tripod.com/  |
+---------------------+
1 row in set (0.00 sec)



#在 memcached 上 telent 查看数据更新
[root@localhost ~]# telnet 192.168.16.21 11211
Trying 192.168.16.21...
Connected to 192.168.16.21.
Escape character is '^]'.
```

```
get urls:8
VALUE urls:8 0 17
http://google.com
END
quit
Connection closed by foreign host.



mysql> update test.urls set url="http://www.baidu.com" where id=8;
Query OK, 0 rows affected, 1 warning (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 1
```

#在 mysql 中更新数据，查看 memcached 中是否会更新呢

```
mysql> select * from urls;
+----+-----------------------+
| id | url                   |
+----+-----------------------+
|  8 | http://www.baidu.com  |
|  9 | http://lycos.com/     |
| 10 | http://tripod.com/    |
| 11 | http://microsoft.com/ |
| 12 | http://slashdot.org   |
| 13 | http://mysql.com      |
+----+-----------------------+
6 rows in set (0.00 sec)
[root@localhost ~]#  telnet 192.168.16.21 11211
Trying 192.168.16.21...
Connected to 192.168.16.21.
Escape character is '^]'.
get urls:8
VALUE urls:8 0 20
http://www.baidu.com
END
quit
Connection closed by foreign host.
```

需要注意的问题使用 libmemcached-0.40以上的版本编译会出现以下问题

```
servers.c: In function ‘memc_servers_set’:
servers.c:121: error: ‘memcached_st’ has no member named ‘hosts’
servers.c:122: error: ‘memcached_st’ has no member named ‘hosts’
servers.c:123: error: ‘memcached_st’ has no member named ‘hosts’
```

源代码此处定义的问题（网上看到有相关修改，不过稳定性如何未知）

```
memcached_version(master_memc);
  if (master_memc->hosts[0].major_version >= 1 &&
      master_memc->hosts[0].minor_version >= 2 &&
      master_memc->hosts[0].micro_version >= 4)
    memcached_behavior_set(master_memc,     MEMCACHED_BEHAVIOR_SUPPORT_CAS,
set);
  memcached_server_list_free(servers);
  pthread_mutex_unlock(&memc_servers_mutex);
  fprintf(stderr, "rc %d\n", rc);
  return ((long long) rc == MEMCACHED_SUCCESS ? 0 : rc);
}
```