一、MemCache简

session

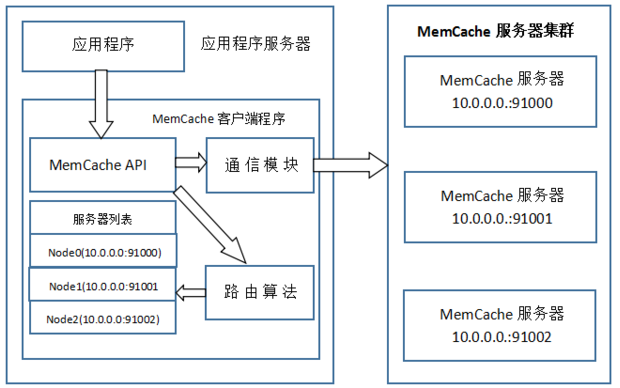
MemCache是一个自由、源码开放、高性能、分布式的分布式内存对象缓存系统，用于动态Web应用以减轻数据库的负载。它通过在内存中缓存数据和对象来减少读取数据库的次数，从而提高了网站访问的速度。 MemCaChe是一个存储键值对的HashMap，在内存中对任意的数据（比如字符串、对象等）所使用的key-value存储，数据可以来自数据库调用、API调用，或者页面渲染的结果。MemCache设计理念就是小而强大，它简单的设计促进了快速部署、易于开发并解决面对大规模的数据缓存的许多难题，而所开放的API使得MemCache能用于Java、C/C++/C#、Perl、Python、PHP、Ruby等大部分流行的程序语言。

另外，说一下为什么会有Memcache和memcached两种名称？其实Memcache是这个项目的名称，而memcached是它服务器端的主程序文件名

MemCache的官方网站为 <http://memcached.org/>

MemCache访问模型

为了加深对memcache的理解，以memcache为代表的分布式缓存，访问模型如下：



特别澄清一个问题，MemCache虽然被称为”分布式缓存”，但是MemCache本身完全不具备分布式的功能，MemCache集群之间不会相互通信（与之形成对比的，比如JBoss Cache，某台服务器有缓存数据更新时，会通知集群中其他机器更新缓存或清除缓存数据），所谓的”分布式”，完全依赖于客户端程序的实现，就像上面这张图的流程一样。

同时基于这张图，理一下MemCache一次写缓存的流程：

1、应用程序输入需要写缓存的数据

2、API将Key输入路由算法模块，路由算法根据Key和MemCache集群服务器列表得到一台服务器编号

3、由服务器编号得到MemCache及其的ip地址和端口号

4、API调用通信模块和指定编号的服务器通信，将数据写入该服务器，完成一次分布式缓存的写操作

读缓存和写缓存一样，只要使用相同的路由算法和服务器列表，只要应用程序查询的是相同的Key，MemCache客户端总是访问相同的客户端去读取数据，只要服务器中还缓存着该数据，就能保证缓存命中。

这种MemCache集群的方式也是从分区容错性的方面考虑的，假如Node2宕机了，那么Node2上面存储的数据都不可用了，此时由于集群中Node0和Node1还存在，下一次请求Node2中存储的Key值的时候，肯定是没有命中的，这时先从数据库中拿到要缓存的数据，然后路由算法模块根据Key值在Node0和Node1中选取一个节点，把对应的数据放进去，这样下一次就又可以走缓存了，这种集群的做法很好，但是缺点是成本比较大。

一致性Hash算法

从上面的图中，可以看出一个很重要的问题，就是对服务器集群的管理，路由算法至关重要，就和负载均衡算法一样，路由算法决定着究竟该访问集群中的哪台服务器，先看一个简单的路由算法。

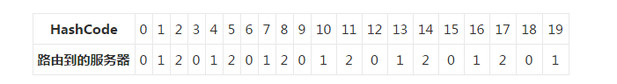
1、余数Hash

简单的路由算法可以使用余数Hash:用服务器数目和缓存数据KEY的hash值相除，余数为服务器列表下标编号，假如某个str对应的HashCode是52、服务器的数目是3，取余数得到1，str对应节点Node1，所以路由算法把str路由到Node1服务器上。由于HashCode随机性比较强，所以使用余数Hash路由算法就可以保证缓存数据在整个MemCache服务器集群中有比较均衡的分布。

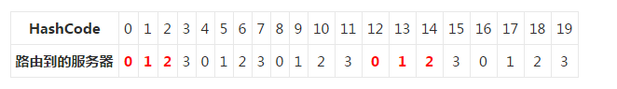
如果不考虑服务器集群的伸缩性，那么余数Hash算法几乎可以满足绝大多数的缓存路由需求，但是当分布式缓存集群需要扩容的时候，就难办了。

就假设MemCache服务器集群由3台变为4台吧，更改服务器列表，仍然使用余数Hash，52对4的余数是0，对应Node0，但是str原来是存在Node1上的，这就导致了缓存没有命中。再举个例子，原来有HashCode为0~19的20个数据，那么：

那么不妨举个例子，原来有HashCode为0~19的20个数据，那么：



现在扩容到4台，加粗标红的表示命中：



如果扩容到20+的台数，只有前三个HashCode对应的Key是命中的，也就是15%。当然现实情况肯定比这个复杂得多，不过足以说明，使用余数Hash的路由算法，在扩容的时候会造成大量的数据无法正确命中（其实不仅仅是无法命中，那些大量的无法命中的数据还在原缓存中在被移除前占据着内存）。在网站业务中，大部分的业务数据度操作请求上事实上是通过缓存获取的，只有少量读操作会访问数据库，因此数据库的负载能力是以有缓存为前提而设计的。当大部分被缓存了的数据因为服务器扩容而不能正确读取时，这些数据访问的压力就落在了数据库的身上，这将大大超过数据库的负载能力，严重的可能会导致数据库宕机。

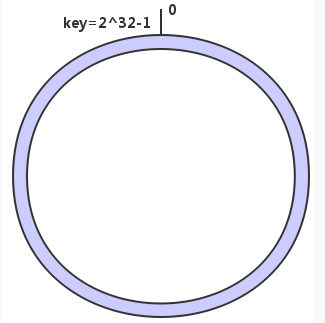
这个问题有解决方案，解决步骤为：

（1）在网站访问量低谷，通常是深夜，技术团队加班，扩容、重启服务器

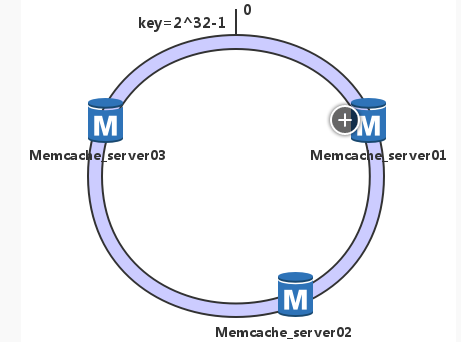
（2）通过模拟请求的方式逐渐预热缓存，使缓存服务器中的数据重新分布

2、一致性Hash算法

一致性Hash算法通过一个叫做一致性Hash环的数据结构实现Key到缓存服务器的Hash映射。简单地说，一致性哈希将整个哈希值空间组织成一个虚拟的圆环（这个环被称为一致性Hash环），如假设某空间哈希函数H的值空间是0~2^32-1（即哈希值是一个32位无符号整形），整个哈希空间如下：

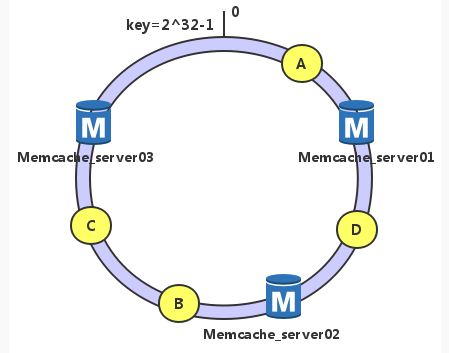


下一步将各个服务器使用H进行一个哈希计算，具体可以使用服务器的IP地址或者主机名作为关键字，这样每台机器能确定其在上面的哈希环上的位置了，并且是按照顺时针排列，这里我们假设三台节点memcache经计算后位置如下



接下来使用相同算法计算出数据的哈希值h,并由此确定数据在此哈希环上的位置

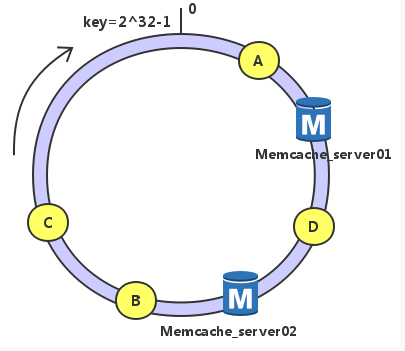
假如我们有数据A、B、C、D、4个对象，经过哈希计算后位置如下：



根据一致性哈希算法，数据A就被绑定到了server01上，D被绑定到了server02上，B、C在server03上，是按照顺时针找最近服务节点方法

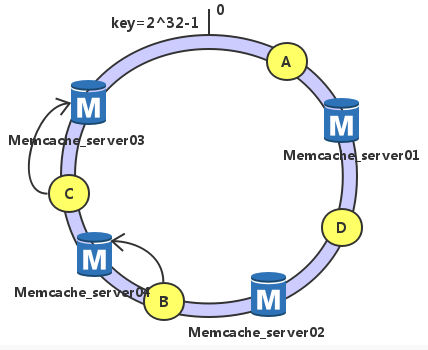
这样得到的哈希环调度方法，有很高的容错性和可扩展性：

假设server03宕机



可以看到此时C、B会受到影响，将B、C被重定位到Server01。一般的，在一致性哈希算法中，如果一台服务器不可用，则受影响的数据仅仅是此服务器到其环空间中前一台服务器（即顺着逆时针方向行走遇到的第一台服务器）之间数据，其它不会受到影响。

考虑另外一种情况，如果我们在系统中增加一台服务器Memcached Server 04：



此时A、D、C不受影响，只有B需要重定位到新的Server04。一般的，在一致性哈希算法中，如果增加一台服务器，则受影响的数据仅仅是新服务器到其环空间中前一台服务器（即顺着逆时针方向行走遇到的第一台服务器）之间数据，其它不会受到影响。

综上所述，一致性哈希算法对于节点的增减都只需重定位环空间中的一小部分数据，具有较好的容错性和可扩展性。

一致性哈希的缺点：在服务节点太少时，容易因为节点分部不均匀而造成数据倾斜问题。我们可以采用增加虚拟节点的方式解决。

更重要的是，集群中缓存服务器节点越多，增加/减少节点带来的影响越小，很好理解。换句话说，随着集群规模的增大，继续命中原有缓存数据的概率会越来越大，虽然仍然有小部分数据缓存在服务器中不能被读到，但是这个比例足够小，即使访问数据库，也不会对数据库造成致命的负载压力。

MemCache实现原理

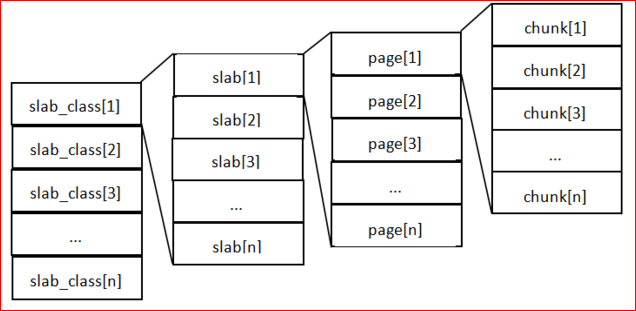
首先要说明一点，MemCache的数据存放在内存中

1、访问数据的速度比传统的关系型数据库要快，因为Oracle、MySQL这些传统的关系型数据库为了保持数据的持久性，数据存放在硬盘中，IO操作速度慢

2、MemCache的数据存放在内存中同时意味着只要MemCache重启了，数据就会消失

3、既然MemCache的数据存放在内存中，那么势必受到机器位数的限制，32位机器最多只能使用2GB的内存空间，64位机器可以认为没有上限

然后我们来看一下MemCache的原理，MemCache最重要的是内存如何分配的，MemCache采用的内存分配方式是固定空间分配，如下图所示：



这张图片里面涉及了slab\_class、slab、page、chunk四个概念，它们之间的关系是：

1、MemCache将内存空间分为一组slab

2、每个slab下又有若干个page，每个page默认是1M，如果一个slab占用100M内存的话，那么这个slab下应该有100个page

3、每个page里面包含一组chunk，chunk是真正存放数据的地方，同一个slab里面的chunk的大小是固定的

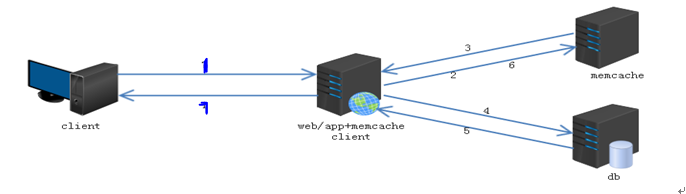
4、有相同大小chunk的slab被组织在一起，称为slab\_class

MemCache内存分配的方式称为allocator（分配运算），slab的数量是有限的，几个、十几个或者几十个，这个和启动参数的配置相关。

MemCache中的value存放的地方是由value的大小决定的，value总是会被存放到与chunk大小最接近的一个slab中，比如slab[1]的chunk大小为80字节、slab[2]的chunk大小为100字节、slab[3]的chunk大小为125字节（相邻slab内的chunk基本以1.25为比例进行增长，MemCache启动时可以用-f指定这个比例），那么过来一个88字节的value，这个value将被放到2号slab中。放slab的时候，首先slab要申请内存，申请内存是以page为单位的，所以在放入第一个数据的时候，无论大小为多少，都会有1M大小的page被分配给该slab。申请到page后，slab会将这个page的内存按chunk的大小进行切分，这样就变成了一个chunk数组，最后从这个chunk数组中选择一个用于存储数据。

如果这个slab中没有chunk可以分配了怎么办，如果MemCache启动没有追加-M（禁止LRU，这种情况下内存不够会报Out Of Memory错误），那么MemCache会把这个slab中最近最少使用的chunk中的数据清理掉，然后放上最新的数据。

Memcache的工作流程：



1、检查客户端的请求数据是否在memcached中，如果有，直接把请求数据返回，不再对数据库进行任何操作，路径操作为①②③⑦。  
2、如果请求的数据不在memcached中，就去查数据库，把从数据库中获取的数据返回给客户端，同时把数据缓存一份到memcached中（memcached客户端不负责，需要程序明确实现），路径操作为①②④⑤⑦⑥。  
3、每次更新数据库的同时更新memcached中的数据，保证一致性。  
4、当分配给memcached内存空间用完之后，会使用LRU（Least Recently Used，最近最少使用）策略加上到期失效策略，失效数据首先被替换，然后再替换掉最近未使用的数据。

**Memcached特征：**  
协议简单:  
   它是基于文本行的协议，直接通过telnet在memcached服务器上可进行存取数据操作

注：文本行的协议：指的是信息以文本传送，一个信息单元传递完毕后要传送换行。比如对于HTTP的GET请求来说，GET /index.html HTTP/1.1是一行，接下去每个头部信息各占一行。一个空行表示整个请求结束

基于libevent事件处理:  
    Libevent是一套利用C开发的程序库，它将BSD系统的kqueue,Linux系统的epoll等事件处理功能封装成一个接口，与传统的select相比，提高了性能。  
内置的内存管理方式:  
    所有数据都保存在内存中，存取数据比硬盘快，当内存满后，通过LRU算法自动删除不使用的缓存，但没有考虑数据的容灾问题，重启服务，所有数据会丢失。  
分布式  
   各个memcached服务器之间互不通信，各自独立存取数据，不共享任何信息。服务器并不具有分布式功能，分布式部署取决于memcache客户端。

Memcache的安装

分为两个过程：memcache服务器端的安装和memcached客户端的安装。

所谓服务器端的安装就是在服务器（一般都是linux系统）上安装Memcache实现数据的存储。

所谓客户端的安装就是指php（或者其他程序，Memcache还有其他不错的api接口提供）去使用服务器端的Memcache提供的数据，需要php添加扩展。

PHP的Memcache

二、centos7.2+nginx+php+memcache+mysql

环境描述：

OS：

[root@www ~]# cat /etc/redhat-release

CentOS Linux release 7.2.1511 (Core)

nginx和php：

nginx-1.10.2.tar.gz

php-5.6.27.tar.gz

ip地址：192.168.31.141/24

memcache:

memcached-1.4.33.tar.gz

ip地址：192.168.31.250/24

mysql:

mysql-5.7.13.tar.gz

ip地址：192.168.31.225/24

1、安装nginx(在192.168.31.141主机操作)

解压zlib

[root@www ~]# tar zxf zlib-1.2.8.tar.gz

说明：不需要编译，只需要解压就行。

解压pcre

[root@www ~]# tar zxf pcre-8.39.tar.gz

说明：不需要编译，只需要解压就行。

[root@www ~]# yum -y install gcc gcc-c++ make libtool openssl openssl-devel

下载nginx的源码包：<http://nginx.org/download>

解压源码包：

[root@www ~]# tar zxf nginx-1.10.2.tar.gz

[root@www ~]# cd nginx-1.10.2/

[root@www ~]# groupadd www #添加www组

[root@www ~]# useradd -g www www -s /sbin/nologin  #创建nginx运行账户www并加入到www组，不允许www用户直接登录系统

[root@www nginx-1.10.2]# ./configure --prefix=/usr/local/nginx1.10 --with-http\_dav\_module --with-http\_stub\_status\_module --with-http\_addition\_module --with-http\_sub\_module --with-http\_flv\_module --with-http\_mp4\_module --with-pcre=/root/pcre-8.39 --with-zlib=/root/zlib-1.2.8 --with-http\_ssl\_module --with-http\_gzip\_static\_module --user=www --group=www

[root@www nginx-1.10.2]# make&& make install

注：

--with-pcre：用来设置pcre的源码目录。  
 --with-zlib：用来设置zlib的源码目录。  
 因为编译nginx需要用到这两个库的源码。

[root@www nginx-1.10.2]# ln -s /usr/local/nginx1.10/sbin/nginx /usr/local/sbin/

[root@www nginx-1.10.2]# nginx -t

启动nginx

[root@www nginx-1.10.2]# nginx

[root@www nginx-1.10.2]# netstat -anpt | grep nginx

tcp 0 0 0.0.0.0:80 0.0.0.0:\* LISTEN 9834/nginx: master

[root@www nginx-1.10.2]# firewall-cmd --permanent --add-port=80/tcp

success

[root@www nginx-1.10.2]# firewall-cmd --reload

success

启动后可以再浏览器中打开页面，会显示nginx默认页面。



2、安装php

安装libmcrypt

[root@www ~]# tar zxf libmcrypt-2.5.7.tar.gz

[root@www ~]# cd libmcrypt-2.5.7/

[root@www libmcrypt-2.5.7]# ./configure --prefix=/usr/local/libmcrypt && make && make install

[root@www ~]# yum -y install libxml2-devel libcurl-devel openssl-devel bzip2-devel

[root@www ~]# tar zxf php-5.6.27.tar.gz

[root@www ~]# cd php-5.6.27/

[root@www php-5.6.27]#./configure --prefix=/usr/local/php5.6 --with-mysql=mysqlnd --with-pdo-mysql=mysqlnd --with-mysqli=mysqlnd --with-openssl --enable-fpm --enable-sockets --enable-sysvshm --enable-mbstring --with-freetype-dir --with-jpeg-dir --with-png-dir --with-zlib --with-libxml-dir=/usr --enable-xml --with-mhash --with-mcrypt=/usr/local/libmcrypt --with-config-file-path=/etc --with-config-file-scan-dir=/etc/php.d --with-bz2 --enable-maintainer-zts

[root@www php-5.6.27]# make&& make install

[root@www php-5.6.27]# cp php.ini-production /etc/php.ini

修改/etc/php.ini文件，将short\_open\_tag修改为on，修改后的内容如下：

short\_open\_tag = On //支持php短标签

创建php-fpm服务启动脚本：

[root@www php-5.6.27]# cp sapi/fpm/init.d.php-fpm /etc/init.d/php-fpm

[root@www php-5.6.27]# chmod +x /etc/init.d/php-fpm

[root@www php-5.6.27]# chkconfig --add php-fpm

[root@www php-5.6.27]# chkconfig php-fpm on

提供php-fpm配置文件并编辑：

#cp /usr/local/php5.6/etc/php-fpm.conf.default /usr/local/php5.6/etc/php-fpm.conf

[root@www php-5.6.27]# vi /usr/local/php5.6/etc/php-fpm.conf

修改内容如下：

pid = run/php-fpm.pid

listen =127.0.0.1:9000

pm.max\_children = 300

pm.start\_servers = 10

pm.min\_spare\_servers = 10

pm.max\_spare\_servers =50

启动php-fpm服务：

[root@phpserver ~]# service php-fpm start

Starting php-fpm done

[root@www php-5.6.27]# netstat -anpt | grep php-fpm

tcp 0 0 127.0.0.1:9000 0.0.0.0:\* LISTEN 10937/php-fpm: mast

3、安装mysql（在192.168.31.225主机操作）

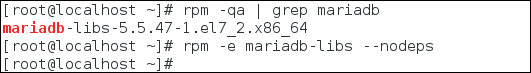
因为centos7.2默认安装了mariadb-libs，所以先要卸载掉

查看是否安装mariadb

#rpm -qa | grep mariadb

卸载mariadb

rpm -e --nodeps mariadb-libs



2、安装依赖包

**注： 相关依赖包的作用**

cmake：由于从MySQL5.5版本开始弃用了常规的configure编译方法，所以需要CMake编译器，用于设置mysql的编译参数。如：安装目录、数据存放目录、字符编码、排序规则等。

Boost #从MySQL 5.7.5开始Boost库是必需的，mysql源码中用到了C++的Boost库，要求必须安装boost1.59.0或以上版本

GCC是Linux下的C语言编译工具，mysql源码编译完全由C和C++编写，要求必须安装GCC

bison：Linux下C/C++语法分析器

ncurses：字符终端处理库

1）安装文件准备

下载cmake-3.5.tar.gz    <http://www.cmake.org/download/>

下载ncurses-5.9.tar.gzftp://ftp.gnu.org/gnu/ncurses/

下载bison-3.0.4.tar.gzhttp://ftp.gnu.org/gnu/bison/

下载mysql-5.7.13.tar.gz

wget http://cdn.mysql.com/Downloads/MySQL-5.7/mysql-5.7.13.tar.gz

下载Boost\_1\_59\_0.tar.gz

wget http://nchc.dl.sourceforge.net/project/boost/boost/1.59.0/boost\_1\_59\_0.tar.gz

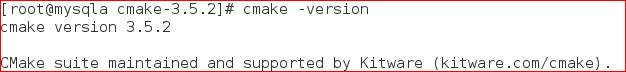
2）安装CMAKE及必要的软件

安装cmake





cmake –version  ---查看cmake版本



**安装ncurses**



安装bison



安装bootst

tar zxf boost\_1\_59\_0.tar.gz

mv boost\_1\_59\_0 /usr/local/boost

3）创建mysql用户和用户组及目录

# groupadd -r mysql && useradd -r -g mysql -s /bin/false -M mysql---新建msyql组和msyql用户禁止登录shell

#mkdir /usr/local/mysql        ---创建目录

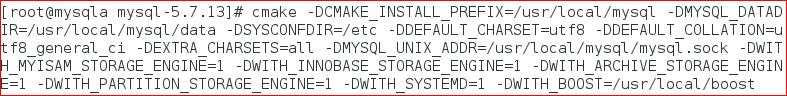
#mkdir /usr/local/mysql/data    ---数据库目录

3、编译安装mysql

解压mysql源码包：



执行cmake命令进行编译前的配置：



开始编译、编译安装：



注1：配置解释：

-DCMAKE\_INSTALL\_PREFIX=/usr/local/mysql[MySQL安装的根目录]-DMYSQL\_DATADIR=/usr/local/mysql /data[MySQL数据库文件存放目录]

-DSYSCONFDIR=/etc [MySQL配置文件所在目录]

-DWITH\_MYISAM\_STORAGE\_ENGINE=1 [添加MYISAM引擎支持]

-DWITH\_INNOBASE\_STORAGE\_ENGINE=1[添加InnoDB引擎支持]

-DWITH\_ARCHIVE\_STORAGE\_ENGINE=1 [添加ARCHIVE引擎支持]

-DMYSQL\_UNIX\_ADDR=/usr/local/mysql /mysql.sock[指定mysql.sock位置]

-DWITH\_PARTITION\_STORAGE\_ENGINE=1[安装支持数据库分区]

-DEXTRA\_CHARSETS=all [使MySQL支持所有的扩展字符]

-DDEFAULT\_CHARSET=utf8[设置MySQL的默认字符集为utf8]-DDEFAULT\_COLLATION=utf8\_general\_ci [设置默认字符集校对规则]

-DWITH-SYSTEMD=1 [可以使用systemd控制mysql服务]

-DWITH\_BOOST=/usr/local/boost [指向boost库所在目录]

更多参数执行[root@localhost mysql-5.7.13]**# cmake . –LH**

**注2：为了加快编译速度可以按下面的方式编译安装**



make -j $(grep processor /proc/cpuinfo | wc –l)

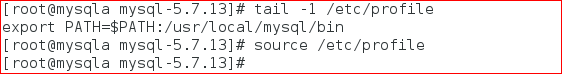
-j参数表示根据CPU核数指定编译时的线程数，可以加快编译速度。默认为1个线程编译。

注3：若要重新运行cmake配置，需要删除CMakeCache.txt文件

# make clean

#rm -f CMakeCache.txt

优化Mysql的执行路径



4、设置权限并初始化MySQL系统授权表

# cd/usr/local/mysql

# **chown -R mysql:mysql .**       ---更改所有者,属组，注意是**mysql .**

**#bin/mysqld --initialize--user=mysql --basedir=/usr/local/mysql --datadir=/usr/local/mysql/data**

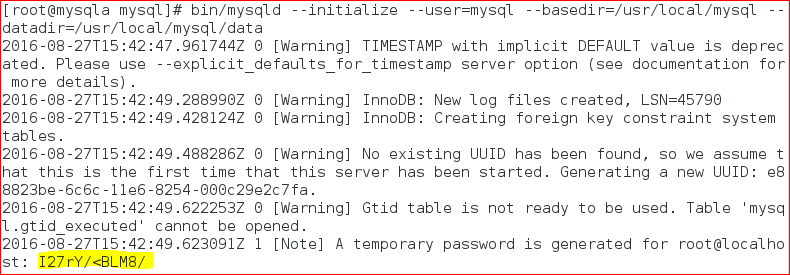
**注1：**以root初始化操作时要加--user=mysql参数，生成一个随机密码（注意保存登录时用）

注2：MySQL 5.7.6之前的版本执行这个脚本初始化系统数据库

/usr/local/mysql/bin/mysql\_install\_db --user=mysql --basedir=/usr/local/mysql --datadir=/usr/local/mysql/data

# 5.7.6之后版本初始系统数据库脚本（本文使用此方式初始化）

#/usr/local/mysql/bin/mysqld --initialize-insecure--user=mysql --basedir=/usr/local/mysql --datadir=/usr/local/mysql/data



注意：如果使用–initialize参数初始化系统[数据库](http://lib.csdn.net/base/14)之后，会生成root用户的一个临时密码，如上图高亮中所示。

# **chown -Rmysql:mysql .**       ---改所有者，注意是**root .**

5、创建配置文件

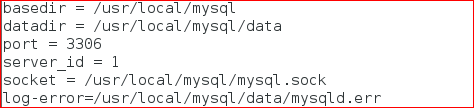
# cd/usr/local/mysql/support-files     ---进入MySQL安装目录支持文件目录

# cp my-default.cnf /etc/my.cnf    ---复制模板为新的配置文件，



修改文件中配置选项，如下图所示，添加如下配置项

#vi /etc/my.cnf

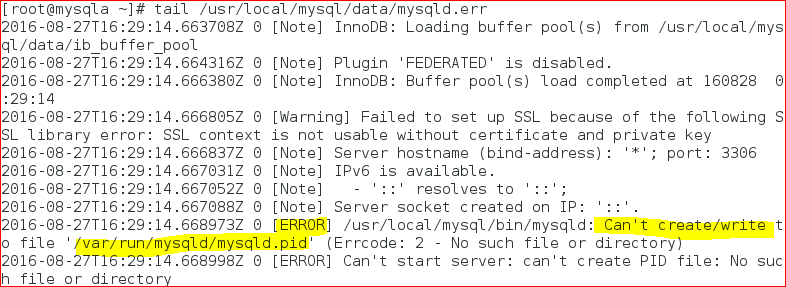


6、配置mysql自动启动





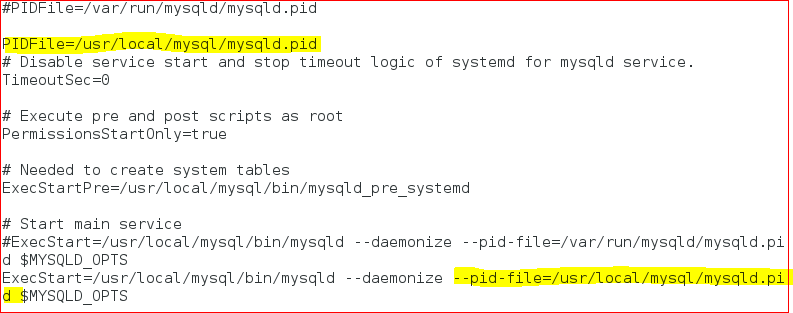
服务启动失败，查看错误日志文件



在mysqld.service，把默认的pid文件指定到了/var/run/mysqld/目录，而并没有事先建立该目录，因此要手动建立该目录并把权限赋给mysql用户。

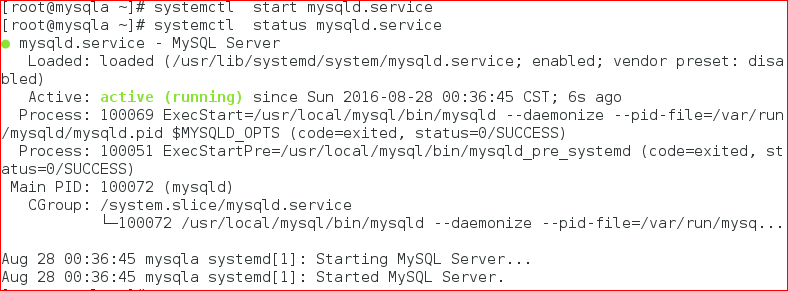


或者修改/usr/lib/system/system/mysqld.service，修改内容如下：



#systemctl daemon-reload

再次启动mysql服务



查看端口号



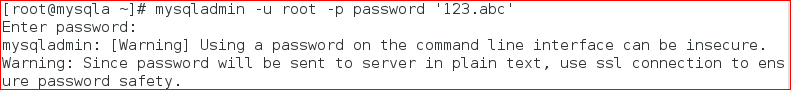
服务启动成功

访问MySQL数据库

# mysql -u root -h 127.0.0.1 -p     ---连接mysql，输入初始化时生成的随机密码



设置数据库管理员用户root的密码



4、安装memcached服务端（在192.168.31.250主机操作）

memcached是基于libevent的事件处理。libevent是个程序库，它将Linux的epoll、BSD类操作系统的kqueue等事件处理功能封装成统一的接口。即使对服务器的连接数增加，也能发挥I/O的性能。 memcached使用这个libevent库，因此能在Linux、BSD、Solaris等操作系统上发挥其高性能。

首先先安装memcached依赖库libevent

[root@memcache ~]# tar zxf libevent-2.0.22-stable.tar.gz

[root@memcache ~]# cd libevent-2.0.22-stable/

[root@memcache libevent-2.0.22-stable]# ./configure

[root@memcache libevent-2.0.22-stable]# make&& make install

安装memcached

[root@memcache ~]# tar zxf memcached-1.4.33.tar.gz

[root@memcache ~]# cd memcached-1.4.33/

[root@memcache memcached-1.4.33]# ./configure --prefix=/usr/local/memcached --with-libevent=/usr/local

[root@memcache memcached-1.4.33]# make&& make install

检测是否成功安装

[root@memcache ~]# ls /usr/local/memcached/bin/memcached

/usr/local/memcached/bin/memcached

通过以上操作就很简单的把memcached服务端编译好了。这时候就可以打开服务端进行工作了。

配置环境变量:  
进入用户宿主目录，编辑.bash\_profile，为系统环境变量LD\_LIBRARY\_PATH增加新的目录，需要增加的内容如下：

[root@memcache ~]# vi ~/.bash\_profile

MEMCACHED\_HOME=/usr/local/memcached

LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:$MEMCACHED\_HOME/lib

[root@memcache ~]# /usr/local/memcached/bin/memcached -d -m 2048 -l 192.168.31.250 -p 11211 -u root -c 10240 -P /usr/local/memcached /memcached.pid

启动参数说明：

-d　　选项是启动一个守护进程。

-m　　分配给Memcache使用的内存数量，单位是MB，默认64MB。

-l　　监听的IP地址。（默认：INADDR\_ANY，所有地址）

-p　　设置Memcache的TCP监听的端口，最好是1024以上的端口。

-u　　运行Memcache的用户，如果当前为root的话，需要使用此参数指定用户。

-c　　选项是最大运行的并发连接数，默认是1024。

-P　　设置保存Memcache的pid文件。

-M 内存耗尽时返回错误，而不是删除项

-f 块大小增长因子，默认是1.25  
-n 最小分配空间，key+value+flags默认是48  
-h 显示帮助

[root@memcache ~]# netstat -anpt |grep memcached

tcp 0 0 192.168.31.250:11211 0.0.0.0:\* LISTEN 12840/memcached

设置防火墙：

[root@memcache ~]# firewall-cmd --permanent --add-port=11211/tcp

success

[root@memcache ~]# firewall-cmd --reload

success

刷新用户环境变量：

[root@memcache ~]# source ~/.bash\_profile

编写memcached服务启停脚本

[root@memcache ~]# vi /etc/init.d/memcached

脚本内容如下：

[root@memcache ~]# cat /etc/init.d/memcached

#!/bin/sh

#

# pidfile: /usr/local/memcached/memcached.pid

# memcached\_home: /usr/local/memcached

# chkconfig: 35 21 79

# description: Start and stop memcached Service

# Source function library

. /etc/rc.d/init.d/functions

RETVAL=0

prog="memcached"

basedir=/usr/local/memcached

cmd=${basedir}/bin/memcached

pidfile="$basedir/${prog}.pid"

#interface to listen on (default: INADDR\_ANY, all addresses)

ipaddr="192.168.31.250"

#listen port

port=11211

#username for memcached

username="root"

#max memory for memcached,default is 64M

max\_memory=2048

#max connections for memcached

max\_simul\_conn=10240

start() {

echo -n $"Starting service: $prog"

$cmd -d -m $max\_memory -u $username -l $ipaddr -p $port -c $max\_simul\_conn -P $pidfile

RETVAL=$?

echo

[ $RETVAL -eq 0 ] && touch /var/lock/subsys/$prog

}

stop() {

echo -n $"Stopping service: $prog "

run\_user=$(whoami)

pidlist=$(ps -ef | grep $run\_user | grep memcached | grep -v grep | awk '{print($2)}')

for pid in $pidlist

do

kill -9 $pid

if [ $? -ne 0 ]; then

return 1

fi

done

RETVAL=$?

echo

[ $RETVAL -eq 0 ] && rm -f /var/lock/subsys/$prog

}

# See how we were called.

case "$1" in

start)

start

;;

stop)

stop

;;

restart)

stop

start

;;

\*)

echo "Usage: $0 {start|stop|restart|status}"

exit 1

esac

exit $RETVAL

设置脚本可被执行：

[root@memcache ~]# chmod +x /etc/init.d/memcached

[root@memcache ~]# chkconfig --add memcached

[root@memcache ~]# chkconfig memcached on

说明：

shell脚本中return的作用

1)终止一个函数.   
2)return命令允许带一个整型参数, 这个整数将作为函数的"退出状态  
码"返回给调用这个函数的脚本, 并且这个整数也被赋值给变量$?.  
3)命令格式：return value

5、配置nginx.conf文件（在nginx主机操作）

配置内容如下：

user www www;

worker\_processes 4;

worker\_cpu\_affinity 0001 0010 0100 1000;

error\_log logs/error.log;

#error\_log logs/error.log notice;

#error\_log logs/error.log info;

pid logs/nginx.pid;

events {

use epoll;

worker\_connections 65535;

multi\_accept on;

}

http {

include mime.types;

default\_type application/octet-stream;

#log\_format main '$remote\_addr - $remote\_user [$time\_local] "$request" '

# '$status $body\_bytes\_sent "$http\_referer" '

# '"$http\_user\_agent" "$http\_x\_forwarded\_for"';

#access\_log logs/access.log main;

sendfile on;

tcp\_nopush on;

keepalive\_timeout 65;

tcp\_nodelay on;

client\_header\_buffer\_size 4k;

open\_file\_cache max=102400 inactive=20s;

open\_file\_cache\_valid 30s;

open\_file\_cache\_min\_uses 1;

client\_header\_timeout 15;

client\_body\_timeout 15;

reset\_timedout\_connection on;

send\_timeout 15;

server\_tokens off;

client\_max\_body\_size 10m;

fastcgi\_connect\_timeout 600;

fastcgi\_send\_timeout 600;

fastcgi\_read\_timeout 600;

fastcgi\_buffer\_size 64k;

fastcgi\_buffers 4 64k;

fastcgi\_busy\_buffers\_size 128k;

fastcgi\_temp\_file\_write\_size 128k;

fastcgi\_temp\_path /usr/local/nginx1.10/nginx\_tmp;

fastcgi\_intercept\_errors on;

fastcgi\_cache\_path /usr/local/nginx1.10/fastcgi\_cache levels=1:2 keys\_zone=cache\_fastcgi:128m inactive=1d max\_size=10g;

gzip on;

gzip\_min\_length 2k;

gzip\_buffers 4 32k;

gzip\_http\_version 1.1;

gzip\_comp\_level 6;

gzip\_types text/plain text/css text/javascript application/json application/javascript application/x-javascript application/xml;

gzip\_vary on;

gzip\_proxied any;

server {

listen 80;

server\_name www.benet.com;

#charset koi8-r;

#access\_log logs/host.access.log main;

location ~\* ^.+\.(jpg|gif|png|swf|flv|wma|wmv|asf|mp3|mmf|zip|rar)$ {

valid\_referers none blocked www.benet.com benet.com;

if ($invalid\_referer) {

#return 302 http://www.benet.com/img/nolink.jpg;

return 404;

break;

}

access\_log off;

}

location / {

root html;

index index.php index.html index.htm;

}

location ~\* \.(ico|jpe?g|gif|png|bmp|swf|flv)$ {

expires 30d;

#log\_not\_found off;

access\_log off;

}

location ~\* \.(js|css)$ {

expires 7d;

log\_not\_found off;

access\_log off;

}

location = /(favicon.ico|roboots.txt) {

access\_log off;

log\_not\_found off;

}

location /status {

stub\_status on;

}

location ~ .\*\.(php|php5)?$ {

root html;

fastcgi\_pass 127.0.0.1:9000;

fastcgi\_index index.php;

include fastcgi.conf;

fastcgi\_cache cache\_fastcgi;

fastcgi\_cache\_valid 200 302 1h;

fastcgi\_cache\_valid 301 1d;

fastcgi\_cache\_valid any 1m;

fastcgi\_cache\_min\_uses 1;

fastcgi\_cache\_use\_stale error timeout invalid\_header http\_500;

fastcgi\_cache\_key http://$host$request\_uri;

}

#error\_page 404 /404.html;

# redirect server error pages to the static page /50x.html

#

error\_page 500 502 503 504 /50x.html;

location = /50x.html {

root html;

}

}

}

重启nginx服务

生成一个php测试页

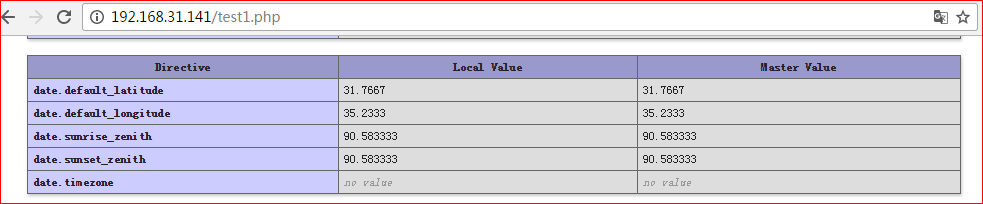
[root@www memcache-3.0.8]# cat /usr/local/nginx1.10/html/test1.php

<?php

phpinfo();

?>

使用浏览器访问test1.php测试页



6、memcache客户端（在php服务器操作）:

memcache分为服务端和客户端。服务端用来存放缓存，客户端用来操作缓存。

安装php扩展库（phpmemcache）。

安装PHP Memcache扩展:

可以使用php自带的pecl安装程序  
# /usr/local/php5.6/bin/pecl install memcache

也可以从源码安装**,**他是生成php的扩展库文件memcache.so。

安装memcache扩展库

[root@www ~]# tar zxf memcache-3.0.8.tgz

[root@www ~]# cd memcache-3.0.8/

[root@www memcache-3.0.8]# /usr/local/php5.6/bin/phpize

[root@wwwmemcache-3.0.8]#./configure --enable-memcache --with-php-config=/usr/local/php5.6/bin/php-config

[root@www memcache-3.0.8]# make&& make install

安装完后会有类似这样的提示：

Installing shared extensions: /usr/local/php5.6/lib/php/extensions/no-debug-zts-20131226/

把这个记住，然后修改php.ini

添加一行

extension=/usr/local/php5.6/lib/php/extensions/no-debug-zts-20131226/memcache.so

重启php-fpm服务

[root@www memcache-3.0.8]# service php-fpm restart

Gracefully shutting down php-fpm .done

Starting php-fpm done

测试：

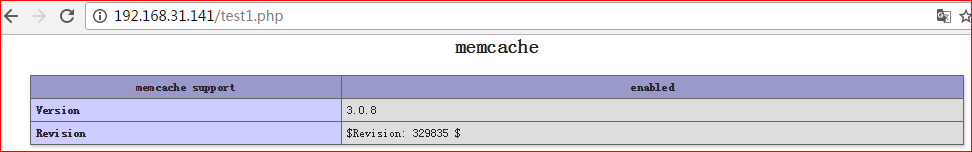
**检查php扩展是否正确安装**

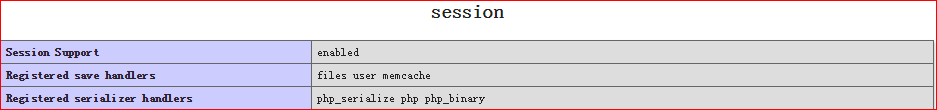
1、[root@www html]# /usr/local/php5.6/bin/php -m

命令行执行php -m 查询结果中是否有memcache项

2、创建phpinfo()页面，查询session项下面的Registered save handlers值中是否有memcache项

浏览器访问test1.php





**测试代码：**

[root@www ~]# cat /usr/local/nginx1.10/html/test2.php

<?php

$memcache = new Memcache;

$memcache->connect('192.168.31.250', 11211) or die ("Could not connect");

$version = $memcache->getVersion();

echo "Server's version: ".$version."<br/>";

$tmp\_object = new stdClass;

$tmp\_object->str\_attr = 'test';

$tmp\_object->int\_attr = 123;

$memcache->set('key', $tmp\_object, false, 10) or die ("Failed to save data at the server");

echo "Store data in the cache (data will expire in 10 seconds)<br/>";

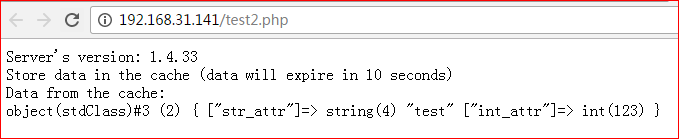
$get\_result = $memcache->get('key');

echo "Data from the cache:<br/>";

var\_dump($get\_result);

?>

浏览器访问test2.php



使用memcache实现session共享

配置php.ini中的Session为memcache方式。

session.save\_handler = memcache

session.save\_path = "tcp://192.168.31.250:11211?persistent=1&weight=1&timeout=1&retry\_interval=15"

注：

**session.save\_handler：**设置session的储存方式为memcache 。默认以文件方式存取session数据，如果想要使用自定义的处理来存取session数据，比如[memcache](http://lib.csdn.net/base/mysql)方式则修为session.save\_handler = memcache

session.save\_path：设置session储存的位置，多台memcache用逗号隔开

使用多个 memcached server 时用逗号”,”隔开，可以带额外的参数”persistent”、”weight”、”timeout”、”retry\_interval”等等，  
类似这样的："tcp://host:port?persistent=1&weight=2,tcp://host2:port2"。

memcache实现session共享也可以在某个一个应用中设置：   
ini\_set("session.save\_handler", "memcache");   
ini\_set("session.save\_path", "[tcp://192.168.0.9](http://192.168.0.9/):11211");

ini\_set()只对当前php页面有效，并且不会去修改php.ini文件本身，也不会影响其他php页面。

测试memcache可用性

重启php-fpm

在web服务器上新建//usr/local/nginx1.10/html/memcache.php文件。内容如

<?php

session\_start();

if (!isset($\_SESSION['session\_time']))

{

$\_SESSION['session\_time'] = time();

}

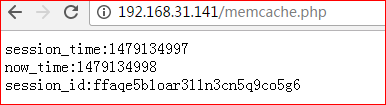
echo "session\_time:".$\_SESSION['session\_time']."<br />";

echo "now\_time:".time()."<br />";

echo "session\_id:".session\_id()."<br />";

?>

访问网址http://192.168.31.141/memcache.php可以查看session\_time是否都是为memcache中的Session，同时可以在不同的服务器上修改不同的标识查看是否为不同的服务器上的。



可以直接用sessionid 去 memcached 里查询一下：

[root@www html]# telnet 192.168.31.250 11211

Trying 192.168.31.250...

Connected to 192.168.31.250.

Escape character is '^]'.

get ffaqe5b1oar311n3cn5q9co5g6

VALUE ffaqe5b1oar311n3cn5q9co5g6 0 26

session\_time|i:1479134997;

得到session\_time|i:1479134997;这样的结果，说明session 正常工作

默认memcache会监听11221端口，如果想清空服务器上memecache的缓存，一般使用的是：  
[root@memcache ~]# telnet 192.168.31.250 11211

Trying 192.168.31.250...

Connected to 192.168.31.250.

Escape character is '^]'.

flush\_all

OK

同样也可以使用：  
[root@memcache ~]# echo "flush\_all" | nc 192.168.31.250 11211

OK

使用flush\_all 后并不是删除memcache上的key，而是置为过期

memcache安全配置

因为memcache不进行权限控制，因此需要通过iptables将memcache仅开放个web服务器。

7、测试memcache缓存数据库数据

在Mysql服务器上创建测试表

mysql> create database testdb1;

Query OK, 1 row affected (0.00 sec)

mysql> use testdb1;

Database changed

mysql> create table test1(id int not null auto\_increment,name varchar(20) default null,primary key (id)) engine=innodb auto\_increment=1 default charset=utf8;

Query OK, 0 rows affected (0.03 sec)

mysql> insert into test1(name) values ('tom1'),('tom2'),('tom3'),('tom4'),('tom5');

Query OK, 5 rows affected (0.01 sec)

Records: 5 Duplicates: 0 Warnings: 0

mysql> select \* from test1;

+----+------+

| id | name |

+----+------+

| 1 | tom1 |

| 2 | tom2 |

| 3 | tom3 |

| 4 | tom4 |

| 5 | tom5 |

+----+------+

5 rows in set (0.00 sec)

测试

下面就是测试的工作了，这里有个php脚本，用于测试memcache是否缓存数据成功  
需要为这个脚本添加一个只读的数据库用户，命令格式

mysql> grant select on testdb1.\* to user@'%' identified by '123456';

Query OK, 0 rows affected, 1 warning (0.00 sec)

在web服务器上创建测试脚本内容如下：

[root@www html]# cat /usr/local/nginx1.10/html/test\_db.php

<?php

$memcachehost = '192.168.31.250';

$memcacheport = 11211;

$memcachelife = 60;

$memcache = new Memcache;

$memcache->connect($memcachehost,$memcacheport) or die ("Could not connect");

$query="select \* from test1 limit 10";

$key=md5($query);

if(!$memcache->get($key))

{

$conn=mysql\_connect("192.168.31.225","user","123456");

mysql\_select\_db(testdb1);

$result=mysql\_query($query);

while ($row=mysql\_fetch\_assoc($result))

{

$arr[]=$row;

}

$f = 'mysql';

$memcache->add($key,serialize($arr),0,30);

$data = $arr ;

}

else{

$f = 'memcache';

$data\_mem=$memcache->get($key);

$data = unserialize($data\_mem);

}

echo $f;

echo "<br>";

echo "$key";

echo "<br>";

//print\_r($data);

foreach($data as $a)

{

echo "number is <b><font color=#FF0000>$a[id]</font></b>";

echo "<br>";

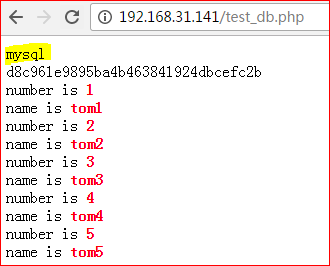
echo "name is <b><font color=#FF0000>$a[name]</font></b>";

echo "<br>";

}

?>

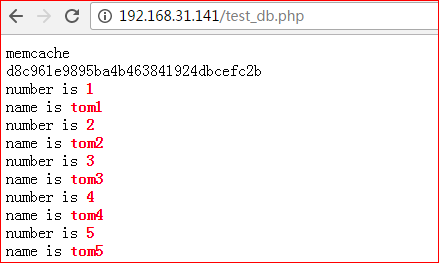
访问页面测试



如果出现mysql表示memcached中没有内容，需要memcached从数据库中取得

再刷新页面，如果有memcache标志表示这次的数据是从memcached中取得的。

memcached有个缓存时间默认是1分钟，过了一分钟后，memcached需要重新从数据库中取得数据



查看 Memcached 缓存情况

我们需要使用 telnet 命令查看

[root@memcache ~]# telnet 192.168.31.250 11211

Trying 192.168.31.250...

Connected to 192.168.31.250.

Escape character is '^]'.

stats

STAT pid 1681 //Memcached 进程的ID

STAT uptime 8429 //进程运行时间

STAT time 1479142306 //当前时间

STAT version 1.4.33 // Memcached 版本

STAT libevent 2.0.22-stable

STAT pointer\_size 64

STAT rusage\_user 1.218430

STAT rusage\_system 1.449512

STAT curr\_connections 5

STAT total\_connections 32

STAT connection\_structures 10

STAT reserved\_fds 20

STAT cmd\_get 25//总共获取数据的次数（等于 get\_hits + get\_misses ）

STAT cmd\_set 19 //总共设置数据的次数

STAT cmd\_flush 4

STAT cmd\_touch 0

STAT get\_hits 15//命中了多少次数据，也就是从 Memcached 缓存中成功获取数据的次数

STAT get\_misses 10//没有命中的次数

STAT get\_expired 3

STAT get\_flushed 1

STAT delete\_misses 0

STAT delete\_hits 0

STAT incr\_misses 2

STAT incr\_hits 2

STAT decr\_misses 0

STAT decr\_hits 0

STAT cas\_misses 0

STAT cas\_hits 0

STAT cas\_badval 0

STAT touch\_hits 0

STAT touch\_misses 0

STAT auth\_cmds 0

STAT auth\_errors 0

STAT bytes\_read 3370

STAT bytes\_written 15710

STAT limit\_maxbytes 2147483648//总的存储大小，默认为 64M

STAT accepting\_conns 1

STAT listen\_disabled\_num 0

STAT time\_in\_listen\_disabled\_us 0

STAT threads 4

STAT conn\_yields 0

STAT hash\_power\_level 16

STAT hash\_bytes 524288

STAT hash\_is\_expanding 0

STAT malloc\_fails 0

STAT log\_worker\_dropped 0

STAT log\_worker\_written 0

STAT log\_watcher\_skipped 0

STAT log\_watcher\_sent 0

STAT bytes 584//当前所用存储大小

STAT curr\_items 3

STAT total\_items 17

STAT expired\_unfetched 2

STAT evicted\_unfetched 0

STAT evictions 0

STAT reclaimed 4

STAT crawler\_reclaimed 0

STAT crawler\_items\_checked 0

STAT lrutail\_reflocked 0

END

命中率= get\_hits/ cmd\_get