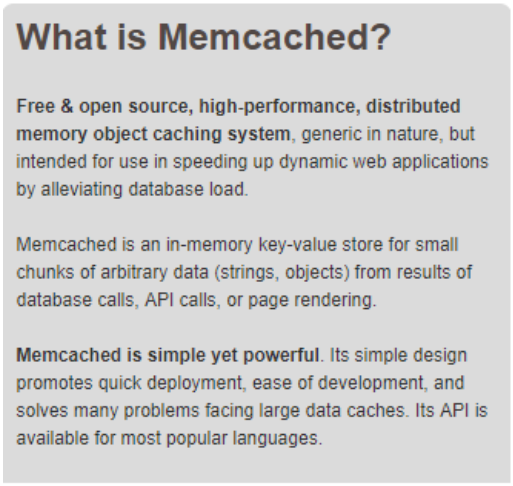
# Memcached教程

> By IT崖柏图
> 2018年03月12日

# 第一章 Memcached 简介

memcached是一种自由并且开放源码, 高性能 ,分布式的内存对象缓存系统,是由livejounal 旗下的 danga 公司开发的老牌 NoSQL 应用.




> 官网 memcached.org

## 1.1 NoSQL是什么？

> NoSQL指的是非关系型数据库,行对于传统的关系型数据库的行列规范,NoSQL的鲜明特点为k-v存储(memcached,redis),或者基于文档存储(mongodb);
>
> 需要注意的是,NoSQL是not only sql的意思,不仅仅是关系型数据库.
>
> 对于memcached而言,数据结构仅仅是简单的键值对类型,但是由于memcached存储介质是计算机的内存,结构简单.使用高性能hash索引(mysql高级会涉及),速度非常快.
>
> 对于mongodb而言,以文档为单位来进行数据存储,文档结构为json对象,文档之间没有必然的联系.

# 第二章 memcached基本使用

## 2.1 linux下编译memcached

### 2.1.1 准备编译环境

在linux编译,需要gcc,make,cmake,autoconf,libtool等工具,以后编译redis等使用.

### 2.1.2 memcache安装

**Centos下安装**

```
yum install memcached
```

**Or**

```
wget https://memcached.org/latest
[you might need to rename the file]
tar -zxf memcached-1.x.x.tar.gz
cd memcached-1.x.x
./configure --prefix=/usr/local/memcached
```

```
make && make test && sudo make install
```

### 2.1.3 下载源码

```
cd  /usr/local/src //切入下载目录

wget https://memcached.org/latest //通过wget下载

tar zxf memcache-1.5.6.tar.gz // 解压

cd memcache-1.5.6 //切入解压后的目录

//编译安装

./configure --prefix=/usr/local/memcached
```

> 注意,在虚拟机下编译的时候,如果系统时间不正确,那么gcc编译过程中可能会检测事件不通过,导致编译一直在等待

```
date -s '20180314 16:53:24'
```

> 检测完成,你会发现遇到一些错误,如下

```
checking for gcc option to accept ISO C99... -std=gnu99
checking sasl/sasl.h usability... yes
checking sasl/sasl.h presence... yes
checking for sasl/sasl.h... yes
checking for gcov... /usr/bin/gcov
checking for main in -lgcov... yes
checking for library containing clock_gettime... -lrt
checking for library containing socket... none required
checking for library containing gethostbyname... none required
checking for libevent directory... configure: error: libevent is required.  You can get it from
http://www.monkey.org/~provos/libevent/

      If it's already installed, specify its path using --with-libevent=/dir/

[root@localhost memcached-1.5.6]#
```

> Memcached depends on libevent. If you're running an OS installed after 2010 the package provided by your OS should be good enough.
> Memcached依赖于libevent库,因此我们要先安装libevent

```
yum install libevent libevent-devel
```

> 安装完毕之后,再次进行编译安装

```
./configure --prefix=/usr/local/memcached

make && make install
```

## 2.2 启动memcached

```
cd /usr/local/memcached/bin/
./memcached
```

直接启动可执行文件,发现系统报错,这是因为memcached不建议使用root用户启动,有可能出现系统漏洞.

所以我们切换用户

```
./ memcached -u jyk
```

正常启动memcached服务端,启动之后不能退出,否则进程消失.

memcached有很多启动参数,我们输入`./memcached -h`即可得到,这里仅对几个常用参数进行说明

- -m 手动设置内存缓存系统所使用的内存大小(m)
- -p 手动设置memcached占用的端口号,默认是11211
- -vvv 打印调试信息
- -u 用户身份
- -c 最大连接数 默认是1024
- -f 增长因子

## 2.3 memcached 的连接

> memcached的客户端和服务端的通讯比较简单,和http协议类似,使用基于文本的协议,而不是二进制协议,因此我们可以使用telnet与memcached做交互.

> 如果虚拟机上没有telnet,我们可以直接yum安装

```
yum install telnet
```

> 安装完毕,连接memcached服务器

```
# 格式 telnet host port
telnet localhost 11211
```

> 退出telnet使用quit

## 2.4 memcached 的命令

同其他数据库一样,我们操作memcached无非也是对数据进行增删改查,在此基础之上我们再学习统计相关命令(也是查)

**增加**

语法: add key flag expire length 回车

其中 flag我们接下来的课程会涉及,可以先填0,expire表示数据的有效期,可以填写0表示永久生效,length表示填写的value值的长度

```
add name 0 0 5
white
STORED
```

服务器会显示存储过程

```
NOT FOUND name
>36 STORED
36: going from conn_nread to conn_write
36: going from conn_write to conn_new_cmd
36: going from conn_new_cmd to conn_waiting
36: going from conn_waiting to conn_read
```

我们在客户端同样可以通过get获取相关数据

```
get name
VALUE name 0 5
white
END
```

参数详解

- key 给值去一个独特的名字
- flag 标志,要求为一个正整数
- expire 有效期

- length 缓存的长度(字节为单位)

flag 的意义:

memcached的基本文本协议,传输的东西理解为字符串来存储.但是当我们要存储一个php对象或者数组的时候,该怎么办?

我们需要把对象或者数组序列化为字符串,当然我们取值的时候还要将其反序列化成为对象/数组/json格式等等,这时候,flag的标志意义就出现了

比如说我们定义1就是字符串 2是反转成数组 3是反序列化对象.当我们从memcached取回数据时候,可以根据flag来区分数据类型.

```
$arr = ["name"=>"lisi","age"=>"19"];
print_r(serialize($arr));
```

就是自己定义的区分。 系统并没有区分

expire的意义:

设置缓存的有效性,有3中格式

- 设置描述,从设定开始数,第n秒后失效
- 设置时间戳,到指定的时间戳后失效(相对于设定瞬间的时间戳),比如在团购网站,缓存的某团到中午12:00失效
- 设置0 不自动失效

需要注意的是 当我们的参数大于30天的秒数时,那么不在理解为秒数了,理解为绝对的时间戳

> 有种误会 设置为0,永久有效,这种想法是错误的
>
> 因为memcached在编译的过程中已经指定了一个最长的常量,默认是30天
>
> 可能等不到30天,就会被新数据挤出去

exprie 直接给的是秒数，则最大30*3600*24==========>(当设置expire为31*3600*24时，则存不进去，解决办法如下：)
如果你希望保持时间超过30天 time()+天数*3600*24 即可

**删除**

delete key

删除指定的key;

**替换**

replace key flag expire length

参数和add完全一样,不但读写,可以替换已有的数据.

当我们对一个已有的键重新进行添加的时候,数据会不被存储,我们可以使用replace

**获取**

get key

返回key值

**设置**

set是设置和修改值,和add,replace一样,但是功能是"有则改之,无则加勉"

如果服务器无此键--->增加
如果服务器有此键--->修改

**步进**

语法 incr/decr key num 可以帮助我们增加或者减少值的大小

```
set age 0 0 2
28
STORED
get age
value age 0 2
28
END
```

```
incr age 1
29
incr age 2
31
decr age 2
29
```

> 注意 incr/decr操作是把值理解成为32位无符号数值来进行加减操作的,值在[0-2^32-1]范围内

```
秒杀场景

一个人下单要牵涉的数据操作比较多,需要对数据库读取,
写入订单,更改库存,及事物要求,对于传统数据库来说,压力是巨大的.

可以利用memcached的incr/decr功能,在内存存储count库存量,
秒杀1000台,每个人抢单主要在内存操作,速度非常快.
我们在秒杀场景中可以先不去找传统数据库做实务,而是把瞬间的压力转移给memcached.

抢到count<=1000的号的人,得到一个订单号,再去另外一个页面慢慢支付
```

需要注意的是,做优化并不是单个主机的调优,而是对整个集群的管理.

**统计命令**

使用stats命令可以把memcached当前的运行信息统计出来.

stats

```
stats
STAT pid 5889 进程号
STAT uptime 12331 持续运行时间
STAT time 1510038285
STAT version 1.5.2 版本号
STAT libevent 1.4.13-stable libevent 版本号
STAT pointer_size 64
STAT rusage_user 0.751885
STAT rusage_system 0.893864
STAT max_connections 1024 最大连接数
STAT curr_connections 10
STAT total_connections 13
STAT rejected_connections 0
STAT connection_structures 11
STAT reserved_fds 20
*STAT cmd_get 20
STAT cmd_set 2
STAT cmd_flush 0
STAT cmd_touch 0
*STAT get_hits 13
*STAT get_misses 7 /这两个参数,可以算出命中率
STAT get_expired 0
STAT get_flushed 0
STAT delete_misses 0
STAT delete_hits 0
STAT incr_misses 0
STAT incr_hits 0
STAT decr_misses 0
STAT decr_hits 0
//...
STAT limit_maxbytes 67108864
STAT accepting_conns 1
STAT listen_disabled_num 0
STAT time_in_listen_disabled_us 0
STAT threads 4
//....
STAT curr_items 1 当前存储的键个数
STAT total_items 2 总键个数
//...
END
```

缓存有一个重要的概念 :命中率0;

命中率是指(查询到的数据的次数/查询综述)*100%;

如上 13/(13+7)=60%的命中率;

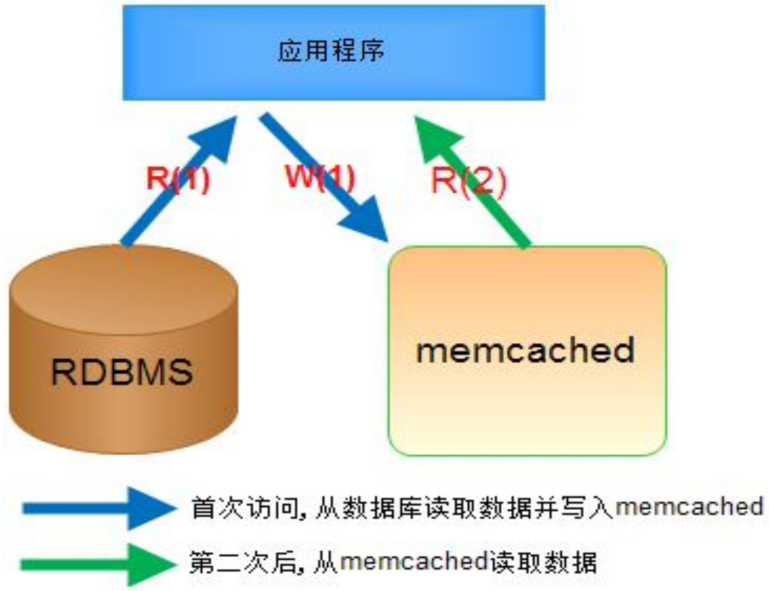
对于缓存而言,命中率能达到百分之60以上已经比较高了(因为100%那是永久存储!缓存总是会失效的)

**清空所有的存储对象**

flush_all 慎用!实际使用中可能导致缓存雪崩现象.(击穿)

# 第三章 编译php及memcached扩展

在通常的场景当中,memcached一般用作缓存数据库查询结果;




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,提高扩展性.

为了使用php能够连接到memcached,我们需要编译php的memcache扩展

1.在pecl下载php拓展库

> http://pecl.php.net/
> 注意,我们添加的是客户端拓展memcache

```
wget http://pecl.php.net/get/memcached-3.0.4.tgz
tar zxvf memcache-3.0.4.tgz
```

此处memcached是PHP的一个扩展

2.生成configure文件

在目录下直接执行

在解压的memcached-3.0.4的目录执行该命令

```
/usr/local/php/bin/phpize
```

phpize会根据你当前的版本和内核生成configure文件

此时生成的configure文件是在memcached-3.0.4文件夹下生成的

3.执行configure,并且make&&make install

> 注意,要填写--with-php-config=PATH参数,其中PATH一般在/usr/local/php/bin/php-config

**检测编译环境**

```
./configure --with-php-config=/usr/local/php/bin/php-config
```

**注意,由于memcached依赖于libmemcached扩展,当执行完上步操作以后会遇到下图错误,所以我们要先编译libmemcached扩展**

```
checking for memcached session support... enabled
checking for memcached igbinary support... disabled
checking for memcached msgpack support... disabled
checking for libmemcached location... configure: error: memcached support requires libmemcached.
 Use --with-libmemcached-dir=<DIR> to specify the prefix where libmemcached headers and library
are located
```

**编译安装libmemcached扩展,注意在这里不要使用yum进行安装,由于yum下载旧版本,导致不支持,所以我们要到官网下载最新的稳定版本**

- `cd /usr/local/src`
- `wget https://launchpad.net/libmemcached/1.0/1.0.18/+download/libmemcached-1.0.18.tar.gz`
- `tar zxf libmemcached-1.0.18.tar.gz`
- `cd ./libmemcached-1.0.18`
- `./configure --prefix=/usr/local/libmemcached`
- `make && make install`
  **再次从新编译安装memcached扩展**
- `cd /usr/local/src/memcache-3.0.8`
- `./configure --with-php-config=/usr/local/php/bin/php-config --with-libmemcached-dir=/usr/local/libmemcached`
- `make && make install`

4.编译完成后只是生成了memcache.so文件,因为我们要在php中使用,还需要配置php.ini开启memcached功能模块

```
vim /usr/local/php/lib/php.ini
```

将刚刚生成的文件添加到模块中

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

重启php 查看phpinfo(),发现memcache已经成功加入到php拓展中.

> 注意官方提供的memcache扩展应用到php7上会报错,我们需要到memcache扩展的github分支目录上寻找
>
> https://github.com/websupport-sk/pecl-memcache/archive/php7.zip

## 缓存实战

```
header('content-type:text/html;charset=utf-8');
$pdo = new PDO('mysql:host=localhost;dbname=test','root','admin123');
$pdo->query('set names utf8');
$sql = 'select * from user';
$st = $pdo->query($sql);
$users = $st->fetchAll(PDO::FETCH_ASSOC);
print_r($users);
```

上述代码是一个典型的php查询mysql的实例,现在我们要做的是将查询出来的结果存入memcache,当我们第二次访问的时候即可利用高性能缓存进行数据读取.
memcache在php端的操作所用的语法和在命令行的大致相同

> http://php.net/memcache

修改之后的代码为

```
header('content-type:text/html;charset=utf-8');
$mem = new Memcached();
$mem->addServer('localhost',11211);
$users = $mem->get('users');
if(!$users){
    $pdo = new PDO('mysql:host=localhost;dbname=test','root','admin123');
    $pdo->query('set names utf8');
    $sql = 'select * from user';
```

```
    $st = $pdo->query($sql);
    $users = $st->fetchAll(PDO::FETCH_ASSOC);
    $mem->add('users',$users,8);
    echo 'from mysql';

}else{
    echo 'from cache';
}
print_r($users);
```

一般情况下,我们在设置缓存失效时间的时候,需要根据实际情况推断,一般热度高并且刷新速度快的字段,比如新闻头条,可以设置3~5分钟,在第一次访问的时候就已经存入memcache,剩下的用户访问全都在请求缓存.

## 后台运行Memcached

```
./bin/memcached -u nobody -m 64 -p 11211 &
```