# **[Memcache](http://www.cnblogs.com/mfryf/p/3791408.html) 入门**

自己以前笔记,和大家分享

一、Memcache概述  
出现的原因:随着数据量的增大,访问的集中,使得数据库服务器的负担加重,数据库响应恶化,网站显示延迟等  
  
memcache:是高性能的分布式内存缓存服务器.通过缓存数据库的查询结果,减少数据库的访问次数,以提高web应用的速度,提高可扩展性.缓存方式是将缓存结果存储在内存中,通过内存来维护一个hash表.  
Memcache是一个c/s软件,默认间通过端口为11211  
二、Memcache工作原理  
memcached是以守护程序方式运行于一个或多个服务器中，随时会接收客户端的连接和操作。  
  
原理:  
第一次:web应用访问数据库,将查询的结果显示到应用的页面,并将其查询结果放入memcache中缓存  
第二次:web访问mecache服务器,如果有数据,直接显示到web应用,否则查询数据库,显示,在进行缓存到memcahe   
  
三、为什么要在WEB中使用Memcache  
原因:数据量的增大,访问的集中,使得数据库服务器的负担加重,数据库响应恶化,网站显示延迟等,  
解决方法:这时就需要减少服务器的压力,减少数据库检索次数,可以建立数据库和web应用的中间缓存层来处理  
memcache作为高速运行的分布式内存缓存服务器,具有以下几点,完全满足需求:  
1 本身是开源的,占用资源小,协议简单的软件,将数据库和web之间的数据缓存,减少数据库的检索次数,减少数据库的i/o  
2 基于livevent的时间处理,因为libevent库将linux,bsd,solaris等这些操作系统上的kqueue等时间处理功能功能封装成统一接口,面对连接数增加,也能在linux,bsd,solaris等操作系统上发挥其高性能(i/o).  
3 存储方式:内置于内存存储方式,存取的效率高,执行的速度快  
4 memcache不互相通信的分布式:同个客户端使得key有规律的封装,实现memcache实现分布式,采用多台cached服务器,增加缓存的横向延伸  
四、安装Memcache服务器（Linux和Window上分别安装）  
Linux下  
1 安装libevent时  
./configure –with-libevent=/usr  
Make && make install  
2 安装memcached  
./configure –with-libevent=/usr  
Make && make install  
3 启动Memcahced –d –m 128 –l 192.168.1.111 –p 11211 –u root  
停止: kill `cat /tmp/memcached.pid`;  
Killall memcached  
Windows下  
Memcahced.exe -d install [uninstall]   
Memcached.exe –d -m 50 –l 127.0.0.1 -p 11211 start  
五、Memcached服务器的管理（启动）  
Linux下启动memcached   
# /usr/local/bin/memcached -d -m 2048 -u root -l 192.168.1.20 -p 12111 -c 1024 -P /tmp/memcached.pid   
参数说明：   
-d 启动为守护进程   
-m <num> 分配给Memcached使用的内存数量，单位是MB，默认为64MB   
-u <username> 运行Memcached的用户，仅当作为root运行时   
-l <ip\_addr> 监听的服务器IP地址，默认为环境变量INDRR\_ANY的值   
-p <num> 设置Memcached监听的端口，最好是1024以上的端口   
-c <num> 设置最大并发连接数，默认为1024   
-P <file> 设置保存Memcached的pid文件，与-d选择同时使用   
Windows下安装. 然后开始 memcached -d start  
memcached的基本设置：  
-p 监听的端口   
-l 连接的IP地址, 默认是本机   
-d start 启动memcached服务   
-d restart 重起memcached服务   
-d stop|shutdown 关闭正在运行的memcached服务   
-d install 安装memcached服务   
-d uninstall 卸载memcached服务   
-u 以的身份运行 (仅在以root运行的时候有效)   
-m 最大内存使用，单位MB。默认64MB ，最大好像2G  
-M 内存耗尽时返回错误，而不是删除项   
-c 最大同时连接数，默认是1024   
-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  
六、操作Memcached (命令行方式telnet作为客户端)  
Command  
Description  
Example  
get  
Reads a value  
get mykey  
set  
Set a key unconditionally  
set mykey 0 60 5  
add  
Add a new key  
add newkey 0 60 5  
replace  
Overwrite existing key  
replace key 0 60 5  
append  
Append data to existing key  
append key 0 60 15  
prepend  
Prepend data to existing key  
prepend key 0 60 15  
incr  
Increments numerical key value by given number  
incr mykey 2  
decr  
Decrements numerical key value by given number  
decr mykey 5  
delete  
Deletes an existing key  
delete mykey  
flush\_all  
Invalidate specific items immediately  
flush\_all  
Invalidate all items in n seconds  
flush\_all 900  
stats  
Prints general statistics  
Stats  
Prints memory statistics  
stats slabs  
Prints memory statistics  
stats malloc  
Print higher level allocation statistics  
stats items  
stats detail  
stats sizes  
Resets statistics  
stats reset  
version  
Prints server version.  
version  
verbosity  
Increases log level  
verbosity  
quit  
Terminate telnet session  
quit  
1 Memcache的协议的错误部分主要是三个错误提示之提示指令：  
　　普通错误信息:ERROR\r\n  
　　客户端错误:CLIENT\_ERROR <错误信息>\r\n  
　　服务器端错误:SERVER\_ERROR <错误信息>\r\n  
2 [ 数据保存指令]  
　　数据保存是基本的功能，就是客户端通过命令把数据返回过来，服务器端接收后进行处理。  
　　A 指令格式：<命令> <键> <标记> <有效期> <数据长度>\r\n  
<键key> :就是保存在服务器上唯一的一个表示符  
<标记flag> 一个16位的无符号整形，用来设置服务器端跟客户端一些交互的操作  
<有效期>是数据在服务器上的有效期限，如果是0，则数据永远有效，单位是秒，Memcache服务器端会把一个数据的有效期设置为当前Unix时间+设置的有效时间  
<数据长度>块数据的长度，一般在这个个长度结束以后下一行跟着block data数据内容，发送完数据以后，客户端一般等待服务器端的返回，服务器端的返回：数据保存成功(STORED\r\n),数据保存失败(NOT\_STORED\r\n)，一般是因为服务器端这个数据key已经存在了  
B 主要是三个储存数据的三个命令， set, add, replace  
　　set 命令是保存一个叫做key的数据到服务器上  
　　add 命令是添加一个数据到服务器，但是服务器必须这个key是不存在的，能够保证数据不会被覆盖  
　　replace 命令是替换一个已经存在的数据，如果数据不存在，就是类似set功能  
　　  
3 [ 数据提取命令]  
get指令，格式是：get <键>\*\r\n   
<键>key是是一个不为空的字符串组合，发送这个指令以后，等待服务器的返回。如果服务器端没有任何数据，则是返回：END\r\n,证明没有不存在这个key，没有任何数据，如果存在数据，则返回指定格式：  
　VALUE <键> <标记> <数据长度>\r\n  
　  
　4 [ 数据删除指令]  
　　delete <键> <超时时间>\r\n  
　　<超时时间> - timeout  
　　按照秒为单位，这个是个可选项，如果你没有指定这个值，那么服务器上key数据将马上被删除，如果设置了这个值，那么数据将在超时时间后把数据清除，该项缺省值是0，就是马上被删除,删除数据后，服务器端会返  
　　DELETED\r\n:删除数据成功  
　　NOT\_FOUND\r\n:这个key没有在服务器上找到  
　　  
　　flush\_all指令  
这个指令执行后，服务器上所有缓存的数据都被删除，并且返回  
　　  
5 [其他指令]  
当前所有Memcache服务器运行的状态信息:stats  
如果只是想获取部分项目的信息，可以指定参数，格式：  
　　stats <参数>\r\n:这个指令将只返回指定参数的项目状态信息。  
当前版本信息:version;  
退出:quit  
//统计  
stats items  
stats sizes  
96 1  
stats slabs:机制分配,内存管理信息  
Stats:  
Pid  
memcache服务器的进程ID  
uptime  
服务器已经运行的秒数  
Time  
服务器当前的unix时间戳  
version  
memcache版本  
pointer\_size  
当前操作系统的指针大小（32位系统一般是32bit）  
rusage\_user  
进程的累计用户时间  
rusage\_system  
进程的累计系统时间  
curr\_items  
服务器当前存储的items数量  
Total\_items  
从服务器启动以后存储的items总数量  
Bytes  
当前服务器存储items占用的字节数  
curr\_connections  
当前打开着的连接数  
Total\_connections  
从服务器启动以后曾经打开过的连接数  
connection\_structures  
服务器分配的连接构造数  
cmd\_get  
get命令（获取）总请求次数  
cmd\_set  
set命令（保存）总请求次数  
get\_hits  
总命中次数  
get\_misses  
总未命中次数  
evictions  
为获取空闲内存而删除的items数（分配给memcache的空间用满后需要删除旧的items来得到空间分配给新的items）  
Bytes\_read  
总读取字节数（请求字节数）  
Bytes\_written  
总发送字节数（结果字节数）  
Limit\_maxbytes  
分配给memcache的内存大小（字节）  
threads  
当前线程数  
在php里也可以用getStats()来查看。  
七、如何遍历memcache  
1. <?php  
2. $host='192.168.15.225';  
3. $port=11211;  
4. $mem=new Memcache();  
5. $mem->connect($host,$port);  
6. $items=$mem->getExtendedStats (‘items’);  
7. $items=$items["$host:$port"]['items'];  
8. for($i=0,$len=count($items);$i<$len;$i++){  
9. $number=$items[$i]['number'];  
10. $str=$mem->getExtendedStats ("cachedump",$number,0);  
11. $line=$str["$host:$port"];  
12. if( is\_array($line) && count($line)>0){  
13. foreach($line as $key=>$value){  
14. echo $key.'=>';  
15. print\_r($mem->get($key));  
16. echo "\r\n";  
17. }  
18. }  
19. }  
20. ?>  
八、在PHP程序中使用Memcached  
a 在PHP安装Memcache扩展  
b 在PHP什么地方使用memcache  
一、 数据库读出来的数据（select）使用memcache处理  
  
二、 在会话控制session中使用  
c 实例  
　　Memcache面向对象的常用接口包括：  
　　Memcache::connect -- 打开一个到Memcache的连接  
　　Memcache::pconnect -- 打开一个到Memcache的长连接  
　　Memcache::close -- 关闭一个Memcache的连接  
　　Memcache::set -- 保存数据到Memcache服务器上  
　　Memcache::get -- 提取一个保存在Memcache服务器上的数据  
　　Memcache::replace -- 替换一个已经存在Memcache服务器上的项目（功能类似Memcache::set）  
　　Memcache::delete -- 从Memcache服务器上删除一个保存的项目  
　　Memcache::flush -- 刷新所有Memcache服务器上保存的项目（类似于删除所有的保存的项目）  
　　Memcache::getStats -- 获取当前Memcache服务器运行的状态  
　　Memcache::addServer -- 分布式服务器添加一个服务器   
  
//创建memcache对象  
$mem=new Memcache;  
  
//连接memcache服务器  
$mem->connect('localhost','11211');  
//长连接memcache服务器  
//$mem->pconnect('localhost','11211');  
/\*添加多个服务器\*/  
//$mem->addServer('url','port');  
//$mem->addServer('www.baidu.com','port');  
//$mem->addServer('192.168.90.112','port');  
  
if($mem->add('test','this is test',MEMCACHE\_COMPRESSED,3600)){  
echo '添加或修改数据成功<br>';  
}else{  
//输出memcache服务器中的值  
echo $mem->get('test').'<br>';  
}  
if($mem->set('test','lampbrother',MEMCACHE\_COMPRESSED,3600)){  
echo '修改数据成功<br>';  
}  
//存取记录  
$mem->add('kkk','vvvvvv');  
echo $mem->get('kkk').'-----<br>';  
//删除一条记录  
$mem->delete('kkk');  
echo $mem->get('kkk').'-----<br>';  
/\* 存储对象 \*/  
class Person{  
private $name;  
private $age;  
function \_\_construct($name,$age){  
$this->name=$name;  
$this->age=$age;  
}  
}  
if($mem->add('mem\_obj',new Person('张三',23))){  
/\*  
\* 对象的存数方式  
\* O:6:"Person":2:{s:12:"Personname";s:6:"寮犱笁";s:11:"Personage";i:23;}   
\* \*/  
echo '添加数据成功';  
}  
//删除所有记录  
$mem->flush();  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*分割线\*\*\*\*\*\*\*\*\*\*\*/  
echo '=====================<br>';  
echo $mem->get('test').'<br>';  
/\*  
//创建memcache对象  
$memcache=new memcache();   
  
//连接服务器端add(),或者增加新的memcache服务器addServer()  
$mem->pconnect('localhost','11211');  
  
//读取数据  
$data=$memcache->get('key\_v');  
  
//判断是否存在  
if($data){  
//直接使用在memcache端获得资源显示到页面  
}else{  
//不存在时,从数据库去的资源显示页面  
//并将获得结果存入memcache服务器端  
}  
$mem->close();  
注意:  
1 同一个项目安装多次时,key要有前缀来进行区分  
2 一个项目中有多条相同的sql语句,可以使用sql语句key值,同种sql结果保证使用一次数据库服务器,减少数据库服务器压力.防止大小写等不必要的异常错误,进行大小写转换进行md5加密,可以保证32为一致性,同时减少了存储容量.还可以使用字符串函数进行md5加密后截取,存储容量更短  
  
\*/  
复制代码   
1. //连接memcache   
2. $m = new Memcache();   
3. $m->connect('localhost', 11211);   
4.   
5. //连接数据库的我就不写了.   
6.   
7. $sql = 'SELECT \* FROM users';   
8. $key = md5($sql); //md5 SQL命令 作为 memcache的唯一标识符   
9. $rows = $m->get($key); //先重memcache获取数据   
10.   
11. if (!$rows) {   
12. //如果$rows为false那么就是没有数据咯, 那么就写入数据   
13. $res = mysql\_query($sql);   
14. $rows = array();   
15. while ($row = mysql\_fetch\_array($res)) {   
16. $rows[] = $row;   
17. }   
18. $m->add($key, $rows); //这里写入重数据库中获取的数据, 可以设置缓存时间, 具体时间设置多少, 根据自己需求吧.   
19. }   
20.   
21. var\_dump($rows); //打印出数据   
22.   
//上面第一次运行程序时, 因为还没有缓存数据, 所以会读取一次数据库, 当再次访问程序时, 就直接重memcache获取了.  
九、Memcache的安全（不让别人访问）  
[ 内网访问]  
最好把两台服务器之间的访问是内网形态的，一般是Web服务器跟Memcache服务器之间。普遍的服务器都是有两块网卡，一块指向互联网，一块指向内网，那么就让Web服务器通过内网的网卡来访问Memcache服务器，我们Memcache的服务器上启动的时候就监听内网的IP地址和端口，内网间的访问能够有效阻止其他非法的访问。  
# memcached -d -m 1024 -u root -l 192.168.0.200 -p 11211 -c 1024 -P /tmp/memcached.pid  
Memcache服务器端设置监听通过内网的192.168.0.200的ip的11211端口，占用1024MB内存，并且允许最大1024个并发连接  
[ 设置防火墙]  
防火墙是简单有效的方式，如果却是两台服务器都是挂在网的，并且需要通过外网IP来访问Memcache的话，那么可以考虑使用防火墙或者代理程序来过滤非法访问。  
一般我们在Linux下可以使用iptables或者FreeBSD下的ipfw来指定一些规则防止一些非法的访问，比如我们可以设置只允许我们的Web服务器来访问我们Memcache服务器，同时阻止其他的访问。  
# iptables -F  
# iptables -P INPUT DROP  
# iptables -A INPUT -p tcp -s 192.168.0.2 --dport 11211 -j ACCEPT  
# iptables -A INPUT -p udp -s 192.168.0.2 --dport 11211 -j ACCEPT  
上面的iptables规则就是只允许192.168.0.2这台Web服务器对Memcache服务器的访问，能够有效的阻止一些非法访问，相应的也可以增加一些其他的规则来加强安全性，这个可以根据自己的需要来做。  
Session()跨域的解决方案:  
1)使用数据库来实现  
2)自己写server端,通过改写session处理函数来请求  
3)使用nfs等跨机存储来保存session  
4)使用memcache来保存  
5)使用zend platform提供的解决方案  
其中的1-4都是通过改用可以跨机的储存机制,再使用session\_set\_save\_handler()来实现  
以下是一些我在使用memcache来实现时的一些记录:  
1)使用类来实现时,各回调函数都定义为静态方法,在类的构造中使用session\_set\_save\_handler注册回调函数, 如:  
session\_set\_save\_handler(  
array('memSession', 'open'),  
array('memSession', 'close'),   
array('memSession', 'read'),   
array('memSession', 'write'),  
array('memSession', 'destroy'),  
array('memSession', 'gc')  
);  
memSession为类名,要使用session,则先new memSession,再session\_start();  
2)生存期和垃圾回收  
memCache的set命令有生存期,即使用set命令添加值时,可加上lifetime,此时间可以作为session的生存期,用户在此时间内没有动作,则会失效,但有动作则不会失效(因为每一个脚本结束时,都会执行write和close,此时lifetime就会被更新了),当然,如果使用cookie传递SID,则控制SESSION生存期可以用:ini\_set('session.cookie\_lifetime',time)来设定,这其实是控制cookie的有效时间,如果session赖以生存的cookie消失了,当然session也就活不了,使用cookie\_lifetime来控制的话,无论有无动作,都将在指定的时间后过时  
gc是指垃圾回收,在session中是指清理过期的session数据,影响的参数有:  
session.gc\_maxlifetime 被视为垃圾前的生存期,超过此时间没有动作,数据会被清走  
注意的是,gc不是每次启动会话都会被执行,而是由session.gc\_probability 和 session.gc\_divisor的比率决定的  
结论:控制SESSION的生存期有几种方法  
一是cookie\_lifttime,这种方式无论有无动作,都会在指定时间内销毁  
二是在read中根椐保存时间控制,此方法在有动作时时间会一直有效  
三设定session.gc\_probability 和 session.gc\_divisor的比率为1(即每次会话都会启用gc),再设定gc.maxlifetime来指定生存期,此方法也是在用户有动作时时间一直有效  
3)回调函数的执行时机  
open 在运行session\_start()时执行  
read 在运行session\_start()时执行,因为在session\_start时,会去read当前session数据并写入$\_SESSION变量  
destroy 在运行session\_destroy()时执行  
close 在脚本执行完成或调用session\_write\_close() 或 session\_destroy()时被执行,即在所有session操作完成后被执行  
gc 执行概率由session.gc\_probability 和 session.gc\_divisor的值决定,时机是在open,read之后,即session\_start会相继执行open,read和gc  
write 此方法在脚本结束和使用session\_write\_close()强制提交SESSION数据时执行  
结论:  
session\_start //执行open(启动会话),read(读取session数据至$\_SESSION),gc(清理垃圾)  
脚本中间所有对$\_SESSION的操作均不会调用这些回调函数  
session\_destroy //执行destroy,销毁当前session(一般是删除相应的记录或文件),相应地,此回调函数销毁的只是session的数据,但此时  
var\_dump一下$\_SESSION变量,仍然有值的,但此值不会在close后被write回去  
session\_write\_close() //执行write和close,保存$\_SESSION至存储,如不手工使用此方法,则会在脚本结束时被自动执行  
清晰了以上信息,将对你清楚了解SESSION的工作原理有很大的帮助...  
4)直接使用memcache作session处理  
在我写了一系列的memcache来保存session的代码后,无意中发现,可以直接在php.ini中设定使用memcache作为session处理,而无须另外编码,方法是:  
修改php.ini中的以下值  
session.save\_handler = memcache  
session.save\_path = 'tcp://host1:11211' #有多个时直接用","分隔即可   
如果只想在特定的应用里使用memcache储存session,可以使用ini\_set的方法对以上两个参数进行设定  
要测试一下是否真正用上了memcache,可以先捕足到使用的PHPSESSID,再作为KEY用memcach去读一下,就清楚了  
这几天做某个产品的时候遇到一个小问题，现象比较诡异  
产品用了两台分布式的memcached服务器  
某一个计数器取回来的数偶尔会不对，最后定位在php memcache client的failover机制上面。  
我们知道，在memcached分布式环境下，某一个key是通过hash计算，分配到某一个memcached上面的  
如果php.ini里面 memcache.allow\_failover = 1的时候，在分布式环境下，某一台memcached出问题的话，会自动到其他的memcached尝试  
就会出现上面的问题，原因如下:  
这个key是hash到服务器A的，但是服务器A正好一瞬间连不上(网络或者其他问题)，PHP就会去另一台服务器B去尝试。  
经过很偶然发生的网络问题和很多次increment操作，有可能两台服务器上面都有这个key，而且值不一样……  
get的时候有可能取到不同的值  
如果对数据一致性要求很严格的话，可以关掉这个参数 memcache.allow\_failover = 0，嗯，问题解  
memcache.allow\_failover 一个布尔值，用于控制当连接出错时 Memcache 扩展是否故障转移到其他服务器上。默认值为 1 (true)。 memcache.max\_failover\_attempts 一个整型值，用于限制连接到持久性数据或检索数据的服务器数目。如果 memcache.allow\_failover 为 false，则将忽略此参数。默认值为 20。 memcache.chunk\_size 一个整型值，用于控制数据传输的大小。默认值为 8192 字节 (8 KB)，但是如果设置为 32768 (32 KB)，则可以获得更好的性能。 memcache.default\_port 另一个整型值，用于设置连接到 Memcache 所使用的 TCP 端口。除非您修改它，否则默认值为无特权的高端口 11211。  
session.save\_handler = memcache  
session.save\_path = "tcp://host:port?persistent=1&weight=2&timeout=2&retry\_interval=15,tcp://host2:port2"  
  
Session\_保存在memcache中:  
class MemSession {  
private static $handler=null;  
private static $lifetime=null;  
private static $time = null;  
const NS='session\_';  
  
private static function init($handler){  
self::$handler=$handler;  
self::$lifetime=ini\_get('session.gc\_maxlifetime');  
self::$time=time();  
}  
public static function start(Memcache $memcache){  
self::init($memcache);  
session\_set\_save\_handler(  
array(\_\_CLASS\_\_, 'open'),  
array(\_\_CLASS\_\_, 'close'),  
array(\_\_CLASS\_\_, 'read'),  
array(\_\_CLASS\_\_, 'write'),  
array(\_\_CLASS\_\_, 'destroy'),  
array(\_\_CLASS\_\_, 'gc')  
);  
session\_start();  
}  
  
public static function open($path, $name){  
return true;  
}  
public static function close(){  
return true;  
}  
public static function read($PHPSESSID){  
$out=self::$handler->get(self::session\_key($PHPSESSID));  
if($out===false || $out == null)  
return '';  
return $out;  
}  
public static function write($PHPSESSID, $data){  
  
$method=$data ? 'set' : 'replace';  
return self::$handler->$method(self::session\_key($PHPSESSID), $data, MEMCACHE\_COMPRESSED, self::$lifetime);  
}  
public static function destroy($PHPSESSID){  
return self::$handler->delete(self::session\_key($PHPSESSID));  
}  
public static function gc($lifetime){  
return true;  
}  
private static function session\_key($PHPSESSID){  
$session\_key=self::NS.$PHPSESSID;  
return $session\_key;  
}   
}  
$memcache=new Memcache;  
$memcache->connect("localhost", 11211) or die("could not connect!");  
MemSession::start($memcache);  
1 只直接使用session\_strat();  
2 将session.save\_handler=memcache  
session.save\_path = "tcp://host:port?persistent=1&weight=2&timeout=2&retry\_interval=15,tcp://host2:port2"