**1.简介**

memcache模块是一个高效的守护进程，提供用于内存缓存的过程式程序和面向对象的方便的接口，特别是对于设计动态web程序时减少对数据库的访问。

memcache也提供用于通信对话（session\_handler）的处理。

更多Memcache 模块相关信息可以到 <http://www.danga.com/memcached/> 查阅。

**1.1.memcache在php.ini中的配置项列表**

|  |  |  |  |
| --- | --- | --- | --- |
| memcache在php.ini中的配置项列表 | | | |
| **名称** | **默认值** | **是否可变** | **改变日志** |
| memcache.allow\_failover | “1” | PHP\_INI\_ALL | Available since memcache 2.0.2. |
| memcache.max\_failover\_attempts | "20" | PHP\_INI\_ALL | Available since memcache 2.1.0. |
| memcache.chunk\_size | "8192" | PHP\_INI\_ALL | Available since memcache 2.0.2. |
| memcache.default\_port | "11211" | PHP\_INI\_ALL | Available since memcache 2.0.2. |
| memcache.hash\_strategy | "standard" | PHP\_INI\_ALL | Available since memcache 2.2.0. |
| memcache.hash\_function | "crc32" | PHP\_INI\_ALL | Available since memcache 2.2.0. |
| session.save\_handler | "files" | PHP\_INI\_ALL | Supported since memcache 2.1.2 |
| session.save\_path | "" | PHP\_INI\_ALL | Supported since memcache 2.1.2 |

有关 PHP\_INI\_\* 常量进一步的细节与定义参见PHP手册php.ini 配置选项。

**1.2.以下是配置项的简要解释**

*memcache.allow\_failover* Boolean

在错误时是否透明的故障转移到其他服务器上处理（注：故障转移是动词）。

*memcache.max\_failover\_attempts* integer

定义服务器的数量类设置和获取数据，只联合 memcache.allow\_failover 一同使用。

*memcache.chunk\_size* integer

数据将会被分成指定大小（chunk\_size）的块来传输，这个值（chunk\_size）越小，写操作的请求就越多，如果发现其他的无法解释的减速，请试着将这个值增大到32768.

*memcache.default\_port* string

当连接memcache服务器的时候，如果没有指定端口这个默认的tcp端口将被用。

*memcache.hash\_strategy* string

控制在映射 key 到服务器时使用哪种策略。设置这个值一致能使hash 算法始终如一的使用于服务器接受添加或者删除池中变量时将不会被重新映射。设置这个值以标准的结果在旧的策略被使用时。

*memcache.hash\_function* string

控制哪种 hsah 函数被应用于 key映射 到服务器过程中，默认值“crc32”使用 CRC32 算法，而“fnv”则表示使用 FNV-1a 算法。

*session.save\_handler* string

通过设置这个值为memcache来确定使用 memcache 用于通信对话的处理（session handler）。

*session.save\_path* string

定义用于通话存储的各服务器链接的分隔符号，例如：“tcp://host1:11211, tcp://host2:11211”。

每服务器个链接可以包含被接受于该服务器的参数，比较类似使用 Memcache::addServer() 来添加的服务器，例如：“tcp://host1:11211?persistent=1&weight=1&timeout=1& amp;retry\_interval=15”。

**1.3.memcache常量列表**

|  |  |  |
| --- | --- | --- |
| memcache常量列表 | | |
| **名称** | **类型** | **描述** |
| MEMCACHE\_COMPRESSED | integer | 用于调整在使用 Memcache::set(), Memcache::add() 和 Memcache::replace() 几个函数时的压缩比率。 |
| MEMCACHE\_HAVE\_SESSION | integer | 如果通信对话的处理（session handler）被允许使用其值为 1，其他情况值为 0。 |

**2Memcache Functions 函数列表**

**2.1.Memcache::connect**

**2.1.1.说明**

bool Memcache::connect ( string $host [, int $port [, int $timeout ]] )

连接memcache服务器

**2.1.2.参数**

$host(string)  服务器域名或ip

$port(int)  服务器tcp端口号，默认值是11211

$timeout  连接memcache进程的失效时间，在修改它的默认值1的时候要三思，以免失去所有memcache缓存的优势导致连接变得很慢。

**2.1.3.返回值**

如果成功则返回true，失败则返回false

**2.1.4.范例**

<?php

/\* procedural API \*/

$memcache\_obj = memcache\_connect('memcache\_host', 11211);

/\* OO API \*/

$memcache = new Memcache;

$memcache->connect('memcache\_host', 11211);

?>

**2.2.Memcache::pconnect**

**2.2.1.说明**

bool Memcache::pconnect ( string $host [, int $port [, int $timeout ]] )

以常连接方式连接服务器

**2.2.2.参数**

$host(string)  服务器域名或ip

$port(int)  服务器tcp端口号，默认值是11211

$timeout  连接memcache进程的失效时间，在修改它的默认值1的时候要三思，以免失去所有memcache缓存的优势导致连接变得很慢。

**2.2.3.返回值**

如果成功则返回true，失败则返回false

**2.2.4.范例**

<?php

/\* procedural API \*/

$memcache\_obj = memcache\_pconnect('memcache\_host', 11211);

/\* OO API \*/

$memcache\_obj = new Memcache;

$memcache\_obj->pconnect('memcache\_host', 11211);

?>

**2.3.Memcache::close**

**2.3.1.说明**

bool Memcache::close ( void )

关闭对象 (对常连接不起作用)

**2.3.2.返回值**

如果成功则返回true，失败则返回false

**2.3.3.范例**

<?php

/\* procedural API \*/

$memcache\_obj = memcache\_connect('memcache\_host', 11211);

/\* do something here .. \*/

memcache\_close($memcache\_obj);

/\* OO API \*/

$memcache\_obj = new Memcache;

$memcache\_obj->connect('memcache\_host', 11211);

/\* do something here .. \*/

$memcache\_obj->close();

?>

**2.4.Memcache::addServer**

**2.4.1.说明**

bool Memcache::addServer ( string $host [, int $port [, bool $persistent [, int $weight [, int $timeout [, int $retry\_interval [, bool $status [, callback $failure\_callback ]]]]]]] )

向对象添加一个服务器（注：addServer没有连接到服务器的动作，所以在memcache进程没有启动的时候，执行addServer成功也会返回true）

**2.4.2.参数**

host               服务器域名或 IP

port               端口号，默认为 11211

persistent         是否使用常连接，默认为 TRUE

weight             权重，在多个服务器设置中占的比重

timeout          连接服务器失效的秒数，修改默认值 1 时要三思，有可能失去所有缓存方面的优势导致连接变得很慢

retry\_interval    服务器连接失败时的重试频率，默认是 15 秒一次，如果设置为 -1 将禁止自动重试，当扩展中加载了 dynamically via dl() 时，无论本参数还是常连接设置参数都会失效。

                          每一个失败的服务器在失效前都有独自的生存期，选择后端请求时会被跳过而不服务于请求。一个过期的连接将成功的重新连接或者被标记为失败的连接等待下一次重试。这种效果就是说每一个 web server 的子进程在服务于页面时的重试连接都跟他们自己的重试频率有关。

status             控制服务器是否被标记为 online，设置这个参数为 FALSE 并设置 retry\_interval 为 -1 可以使连接失败的服务器被放到一个描述不响应请求的服务器池子中，对这个服务器的请求将失败，接受设置为失败服务器的设置，默认参数为 TRUE，代表该服务器可以被定义为 online。

failure\_callback   失败时的回调函数，函数的两个参数为失败服务器的 hostname 和 port

**2.4.3.返回值**

成功返回 TRUE，失败返回 FALSE。

注：在测试addServer函数的时候我们主要测试了其参数retry\_interval和status

**2.4.4.范例**

**2.4.4.1.retry\_interval参数的测试**

<?php

$mem = new Memcache;

$is\_add = $mem->addServer('localhost', 11211, true, 1, 1, 15, true); // retrt\_interval=15

$is\_set = $mem->set('key1', '中华人民共和国');

?>

上面的例子中如果localhost服务器down掉或是memcache守护进程当掉，执行请求的时候连接服务器失败时算起15秒后会自动重试连接服务器，但是在这15秒内不会去连接这个服务器，就是只要有请求，没15秒就会尝试连接服务器，但是每个服务器连接重试是独立的。比如说我一次添加了两个服务器一个是localhost，一个是172.16.100.60，它们分别是从各自连接失败那个时间算起，只要对各自服务器有请求就会每隔15秒去连接各自的服务器的。

**2.4.4.2.retry\_interval和status结合使用的情况**

<?php

$mem = new Memcache;

$is\_add = $mem->addServer('localhost', 11211, true, 1, 1, -1, false); // retrt\_interval=-1, status=false

$is\_set = $mem->set('key1', '中华人民共和国');

?>

在上面的retrt\_interval=-1, status=false这种情况下，将连接失败的服务器放到一个不响应请求的一个池子中，因此对key分配的算法也就没有影响了，而他是立即返回错误失败还是故障转移还要看memcache.allow\_failover的设置，执行set， add， replace，get等请求的时候都会失败返回false，即使memcache进程运行正常。

**2.4.4.3.status参数的测试**

除了与retry\_interval结合使用，status单独使用的情况会对函数memcache::getServerStatu获得的结果产生影响

无论memcache进程的正常运行还是当掉，status为true的时候getServerStatus的结果都是true，反之则为false

但是在memcache进程正常运行的情况下，对set，add，replace，get等函数都没有影响。

**2.5.Memcache::add**

**2.5.1.说明**

bool Memcache::add ( string $key , mixed $var [, int $flag [, int $expire ]] )   
  
添加一个要缓存的数据如果作为这个缓存的数据的键在服务器上还不存在的情况下,

**2.5.2.参数**

key                缓存数据的键 其长度不能超过250个字符

var                值，整型将直接存储，其他类型将被序列化存储 ，其值最大为1M

flag               是否使用 zlib 压缩 ,当flag=MEMCACHE\_COMPRESSED的时侯，数据很小的时候不会采用zlib压缩，只有数据达到一定大小才对数据进行zlib压缩。（没有具体的测试数据进行压缩的最小值是多少）

expire             过期时间，0 为永不过期，可使用 unix 时间戳格式或距离当前时间的秒数，设为秒数时不能大于 2592000（30 天）

**2.5.3.返回值**

成功返回 TRUE，失败返回 FALSE，如果这个键已经存在，其他方面memcache:;add()的行为与memcache::set相似

**2.5.4.范例**

<?php

$memcache\_obj = memcache\_connect("localhost", 11211);

/\* procedural API \*/

memcache\_add($memcache\_obj, 'var\_key', 'test variable', FALSE, 30);

/\* OO API \*/

$memcache\_obj->add('var\_key', 'test variable', FALSE, 30);

?>

**2.6.Memcache::replace**

**2.6.1.说明**

bool Memcache::replace ( string $key , mixed $var [, int $flag [, int $expire ]] )

替换一个指定 已存在key 的的缓存变量内容

**2.6.2.参数**

key                缓存数据的键， 其长度不能超过250个字符

var                值，整型将直接存储，其他类型将被序列化存储，其值最大为1M

flag               是否使用 zlib 压缩 ,当flag=MEMCACHE\_COMPRESSED的时侯，数据很小的时候不会采用zlib压缩，只有数据达到一定大小才对数据进行zlib压缩。（没有具体的测试数据进行压缩的最小值是多少）

expire             过期时间，0 为永不过期，可使用 unix 时间戳格式或距离当前时间的秒数，设为秒数时不能大于 2592000（30 天）

**2.6.3.返回值**

成功返回 TRUE，失败返回 FALSE。

**2.6.4.范例**

<?php

$memcache\_obj = memcache\_connect('memcache\_host', 11211);

/\* procedural API \*/

memcache\_replace($memcache\_obj, "test\_key", "some variable", FALSE, 30);

/\* OO API \*/

$memcache\_obj->replace("test\_key", "some variable", FALSE, 30);

?>

**2.7.Memcache::set**

**2.7.1.说明**

bool Memcache::set ( string $key , mixed $var [, int $flag [, int $expire ]] )   
  
设置一个指定 key 的缓存变量内容

**2.7.2.参数**

key                缓存数据的键， 其长度不能超过250个字符

var                值，整型将直接存储，其他类型将被序列化存储，其值最大为1M

flag               是否使用 zlib 压缩 ,当flag=MEMCACHE\_COMPRESSED的时侯，数据很小的时候不会采用zlib压缩，只有数据达到一定大小才对数据进行zlib压缩。（没有具体的测试数据进行压缩的最小值是多少）

expire             过期时间，0 为永不过期，可使用 unix 时间戳格式或距离当前时间的秒数，设为秒数时不能大于 2592000（30 天）

**2.7.3.返回值**

成功返回 TRUE，失败返回 FALSE。

**2.7.4.范例**

<?php

/\* procedural API \*/

/\* connect to memcached server \*/

$memcache\_obj = memcache\_connect('memcache\_host', 11211);

/\*

set value of item with key 'var\_key'

using 0 as flag value, compression is not used

expire time is 30 second

\*/

memcache\_set($memcache\_obj, 'var\_key', 'some variable', 0, 30);

echo memcache\_get($memcache\_obj, 'var\_key');

?>

<?php

/\* OO API \*/

$memcache\_obj = new Memcache;

/\* connect to memcached server \*/

$memcache\_obj->connect('memcache\_host', 11211);

/\*

set value of item with key 'var\_key', using on-the-fly compression

expire time is 50 seconds

\*/

$memcache\_obj->set('var\_key', 'some really big variable', MEMCACHE\_COMPRESSED, 50);

echo $memcache\_obj->get('var\_key');

?>

**2.8.Memcache::get**

**2.8.1.说明**

string Memcache::get ( string $key [, int &$flags ] )

array Memcache::get ( array $keys [, array &$flags ] )

获取某个 key 的变量缓存值

**2.8.2.参数**

key                缓存值的键

flags              如果是传址某个变量，获取缓存值被set或是add的flag结果将被存于该变量

**2.8.3.返回值**

返回缓存的指定 key 的变量内容或者是在失败或该变量的值不存在时返回 FALSE

 如果传出的key的数组中的key都不存在，返回的结果是一个空数组，反之则返回key与缓存值相关联的关联数组

**2.8.4.范例**

<?php

/\* procedural API \*/

$memcache\_obj = memcache\_connect('memcache\_host', 11211);

$var = memcache\_get($memcache\_obj, 'some\_key');

/\* OO API \*/

$memcache\_obj = new Memcache;

$memcache\_obj->connect('memcache\_host', 11211);

$var = $memcache\_obj->get('some\_key');

/\*

You also can use array of keys as a parameter.

If such item wasn't found at the server, the result

array simply will not include such key.

\*/

/\* procedural API \*/

$memcache\_obj = memcache\_connect('memcache\_host', 11211);

$var = memcache\_get($memcache\_obj, Array('some\_key', 'another\_key'));

//如果some\_key，another\_key不存在 $var = array();

//如果some\_key，another\_key存在 $var = array('some\_key'=>'缓存值', 'another\_key'=>'缓存值');

/\* OO API \*/

$memcache\_obj = new Memcache;

$memcache\_obj->connect('memcache\_host', 11211);

$var = $memcache\_obj->get(Array('some\_key', 'second\_key'));

?>

**2.9.Memcache::delete**

**2.9.1.说明**

bool Memcache::delete ( string $key [, int $timeout ] )

删除某一个变量的缓存

**2.9.2.参数**

key          缓存的键 键值不能为null和'’，当它等于前面两个值的时候php会有警告错误。

timeout   删除这项的时间，如果它等于0，这项将被立刻删除反之如果它等于30秒，那么这项被删除在30秒内

**2.9.3.返回值**

成功返回 TRUE，失败返回 FALSE。

2.9.4.范例

<?php

/\* procedural API \*/

$memcache\_obj = memcache\_connect('memcache\_host', 11211);

/\* after 10 seconds item will be deleted by the server \*/

memcache\_delete($memcache\_obj, 'key\_to\_delete', 10);

/\* OO API \*/

$memcache\_obj = new Memcache;

$memcache\_obj->connect('memcache\_host', 11211);

$memcache\_obj->delete('key\_to\_delete', 10);

?>

**2.10.Memcache::flush**

**2.10.1.说明**

bool Memcache::flush ( void )

清空所有缓存内容，不是真的删除缓存的内容，只是使所有变量的缓存过期，使内存中的内容被重写

**2.10.2.返回值**

成功返回 TRUE，失败返回 FALSE。

**2.10.3.范例**

<?php

/\* procedural API \*/

$memcache\_obj = memcache\_connect('memcache\_host', 11211);

memcache\_flush($memcache\_obj);

/\* OO API \*/

$memcache\_obj = new Memcache;

$memcache\_obj->connect('memcache\_host', 11211);

$memcache\_obj->flush();

?>

**2.11.Memcache::getExtendedStats**

**2.11.1.说明**

array Memcache::getExtendedStats ([ string $type [, int $slabid [, int $limit ]]] )

获取所有服务器扩展静态信息

**2.11.2.参数**

type       静态信息类型，有效值包括{reset, malloc, maps, cachedump, slabs, items, sizes}，依照一定规则协议这个可选参数是为了方便开发人员查看不同类别的信息而输入的标题

slabid   用于按指定类型联合设置 cache 堆为有效的片到堆中。缓存堆被被命令绑定到服务器上并被严格的用于调试用途

limit      用于按指定类型联合设置 cache 堆为输入的数字所限制的大小到堆，默认值为 100

**2.11.3.返回值**

返回一个由服务器扩展静态信息二维数组，失败时返回 FALSE

**2.11.4.范例**

<?php

$memcache\_obj = new Memcache;

$memcache\_obj->addServer('memcache\_host', 11211);

$memcache\_obj->addServer('failed\_host', 11211);

$stats = $memcache\_obj->getExtendedStats(); print\_r($stats);

?>

输出结果

Array(

[memcache\_host:11211] => Array(

[pid] => 3756

[uptime] => 603011

[time] => 1133810435

[version] => 1.1.12

[rusage\_user] => 0.451931

[rusage\_system] => 0.634903

[curr\_items] => 2483

[total\_items] => 3079

[bytes] => 2718136

[curr\_connections] => 2

[total\_connections] => 807

[connection\_structures] => 13

[cmd\_get] => 9748

[cmd\_set] => 3096

[get\_hits] => 5976

[get\_misses] => 3772

[bytes\_read] => 3448968

[bytes\_written] => 2318883

[limit\_maxbytes] => 33554432

),

[failed\_host:11211] =>

)

**2.12.Memcache::getStats**

**2.12.1.说明**

array Memcache::getStats ([ string $type [, int $slabid [, int $limit ]]] )

获取最后添加服务器静态信息

**2.12.2.参数**

type       静态信息类型，有效值包括{reset, malloc, maps, cachedump, slabs, items, sizes}，依照一定规则协议这个可选参数是为了方便开发人员查看不同类别的信息而输入的标题

slabid   用于按指定类型联合设置 cache 堆为有效的片到堆中。缓存堆被被命令绑定到服务器上并被严格的用于调试用途

limit      用于按指定类型联合设置 cache 堆为输入的数字所限制的大小到堆，默认值为 100

**2.12.3.返回值**

返回一个服务器静态信息数组，失败时返回 FALSE

**2.13.Memcache::getServerStatus**

**2.13.1.说明**

int Memcache::getServerStatus ( string $host [, int $port ] )

通过输入的 host 及 port 来获取相应的服务器信息

**2.13.2.参数**

host 服务器域名或 IP

port 端口号，默认为 11211

**2.13.3.返回值**

返回服务器状态，0 为失败，其他情况返回非 0 数字

**2.13.4.范例**

<?php

/\* OO API \*/

$memcache = new Memcache;

$memcache->addServer('memcache\_host', 11211);

echo $memcache->getServerStatus('memcache\_host', 11211);

/\* procedural API \*/

$memcache = memcache\_connect('memcache\_host', 11211);

echo memcache\_get\_server\_status($memcache, 'memcache\_host', 11211);

?>

**2.14.Memcache::getVersion**

**2.14.1.说明**

string Memcache::getVersion ( void )

获取服务器的版本号信息

**2.14.2.返回值**

成功返回服务器的版本号字符串，失败返回 FALSE

**2.14.3.范例**

<?php

/\* OO API \*/

$memcache = new Memcache;

$memcache->connect('memcache\_host', 11211);

echo $memcache->getVersion();

/\* procedural API \*/

$memcache = memcache\_connect('memcache\_host', 11211);

echo memcache\_get\_version($memcache);

?>

**2.15.Memcache::setCompressThreshold**

bool Memcache::setCompressThreshold ( int $threshold [, float $min\_savings ] )

设置压缩极限

**2.15.2.参数**

threshold 设置控制自动压缩的变量长度的最小值

min\_saving 指定的最低压缩比率，值必须介于 0 - 1 之间，默认为 0.2 代表 20% 的压缩比率

**2.15.3.返回值**

成功返回 TRUE，失败返回 FALSE。

**2.15.4.范例**

<?php

/\* OO API \*/

$memcache\_obj = new Memcache;

$memcache\_obj->addServer('memcache\_host', 11211);

$memcache\_obj->setCompressThreshold(20000, 0.2);

/\* procedural API \*/

$memcache\_obj = memcache\_connect('memcache\_host', 11211);

memcache\_set\_compress\_threshold($memcache\_obj, 20000, 0.2);

?>

**2.16.Memcache::setServerParams**

**2.16.1.说明**

bool Memcache::setServerParams ( string $host [, int $port [, int $timeout [, int $retry\_interval [, bool $status [, callback $failure\_callback ]]]]] )

Memcache version 2.1.0 后增加的函数，运行时设置服务器参数

**2.16.2.参数**

host           服务器域名或 IP

port 端口号，默认为 11211

timeout     超时连接失效的秒数，修改默认值 1 时要三思，有可能失去所有缓存方面的优势导致连接变得很慢

retry\_interval     服务器连接失败时的重试频率，默认是 15 秒一次，如果设置为 -1 将禁止自动重试，当扩展中加载了 dynamically via dl() 时，无论本参数还是常连接设置参数都会失效。每一个失败的服务器在失效前都有独自的生存期，选择后端请求时会被跳过而不服务于请求。一个过期的连接将成功的重新连接或者被标记为失败的连接等待下一次重试。这种效果就是说每一个 web server 的子进程在服务于页面时的重试连接都跟他们自己的重试频率有关。

status    控制服务器是否被标记为 online，设置这个参数为 FALSE 并设置 retry\_interval 为 -1 可以使连接失败的服务器被放到一个描述不响应请求的服务器池子中，对这个服务器的请求将失败，接受设置为失败服务器的设置，默认参数为 TRUE，代表该服务器可以被定义为 online。

failure\_callback    失败时的回调函数，函数的两个参数为失败服务器的 hostname 和 port

**2.16.3.返回值**

成功返回 TRUE，失败返回 FALSE。

**2.16.4.范例**

<?php

function \_callback\_memcache\_failure($host, $port)

{

print "memcache '$host:$port' failed";

}

/\* OO API \*/

$memcache = new Memcache;

// Add the server in offline mode

$memcache->addServer('memcache\_host', 11211, FALSE, 1, 1, -1, FALSE);

// Bring the server back online

$memcache->setServerParams('memcache\_host', 11211, 1, 15, TRUE, '\_callback\_memcache\_failure');

/\* procedural API \*/

$memcache\_obj = memcache\_connect('memcache\_host', 11211);

memcache\_set\_server\_params($memcache\_obj, 'memcache\_host', 11211, 1, 15, TRUE, '\_callback\_memcache\_failure');

?>

**2.17.Memcache::increment**

**2.17.1.说明**

int Memcache::increment ( string $key [, int $value ] )

给指定 key 的缓存变量一个增值，如果该变量不是数字时不会被转化为数字，这个增值将会加到该变量原有的数字之上，变量不存在不会新增变量，对于压缩存储的变量不要使用本函数因为相应的取值方法会失败。

**2.17.2.参数**

key 缓存值的键

var 值，整型将直接存储，其他类型将被序列化存储

**2.17.3.返回值**

成功返回新的变量值，失败返回 FALSE。

**2.17.4.范例**

<?php

/\* procedural API \*/

$memcache\_obj = memcache\_connect('memcache\_host', 11211);

/\* increment counter by 2 \*/

$current\_value = memcache\_increment($memcache\_obj, 'counter', 2);

/\* OO API \*/

$memcache\_obj = new Memcache;

$memcache\_obj->connect('memcache\_host', 11211);

/\* increment counter by 3 \*/

$current\_value = $memcache\_obj->increment('counter', 3);

?>

**2.18.Memcache::decrement**

**2.18.2.说明**

 int Memcache::decrement ( string $key [, int $value ] )

给指定 key 的缓存变量一个递减值，与 increment 操作类似，将在原有变量基础上减去这个值，该项的值将会在转化为数字后减去，新项的值不会小于 0，对于压缩存储的变量不要使用本函数因为相应的取值方法会失败。

**2.18.2.参数**

key 缓存值的键

var 值，整型将直接存储，其他类型将被序列化存储

**2.18.3.返回值**

成功返回新的变量值，失败返回 FALSE。

**2.18.4.范例**

<?php

/\* procedural API \*/

$memcache\_obj = memcache\_connect('memcache\_host', 11211);

/\* decrement item by 2 \*/

$new\_value = memcache\_decrement($memcache\_obj, 'test\_item', 2);

/\* OO API \*/

$memcache\_obj = new Memcache;

$memcache\_obj->connect('memcache\_host', 11211);

/\* decrement item by 3 \*/

$new\_value = $memcache\_obj->decrement('test\_item', 3);

?>

**2.19.memcache\_debug**

**2.19.1.说明**

bool memcache\_debug ( bool $on\_off )

设置 memcache 的调试器是否开启，值为 TRUE 或 FALSE。 受影响于 php 安装时是否使用了 --enable-debug 选项，如果使用了该函数才会返回 TRUE，其他情况将始终返回 FALSE。

**2.19.2.参数**

on\_off 设置调试模式是否开启，TRUE 为开启，FALSE 为关闭

**2.19.3.返回值**

php 安装时如果使使用了 --enable-debug 选项返回 TRUE，否则将返回 FALSE。