Memcached, the Better Memcache Interface

International PHP Conference 2011
Ilia Alshanetsky
@iliiaa
root@foo $: whois ilia

PHP: Core Developer Since 2001
    Release Manager of 4.3, 5.1 and 5.2 branches
    Author of a “few” extensions

Work: CIO at Centah Inc.

Hobbies: Photography, Biking

Dev. Interests: Performance & Security
Memcached

- Interface to Memcached - a distributed, in-memory caching system
- Provides a simple Object Oriented interface
- Offers a built-in session handler
- Purpose built, so lots of nifty features
Memcached
Igbinary serializer

Binary protocol support

FastLz compression

Faster

Buffered writes

Multi-Server interface

Delayed fetches
$mc = new MemCached();

// CONNECT TO MEMCACHE ON LOCAL MACHINE, ON DEFAULT PORT
$mc->addServer('localhost', '11211');

// TRY TO ADD AN ARRAY USING "KEY" FOR 1 DAY
if (!$mc->add('key', array(1, 2, 3), 86400)) {
    // IF ALREADY EXISTS, LET'S REPLACE IT
    if (!$mc->replace('key', array(1, 2, 3), 86400)) {
        die("Critical Error");
    }
}

// LET'S FETCH OUR DATA
if (($data = $mc->get('key')) !== FALSE) {
    // LET'S DELETE IT NOW
    $mc->delete('key'); // RIGHT NOW!
}
$mc->add('key', FALSE);

if (($data = $mc->get('key')) !== FALSE) {
    die("Not Found?"); // not true
    // The value could be FALSE, 0, array(), NULL, ""
}

// The “correct” way!
if (
    (($data = $mc->get('key')) === FALSE)
    &&
    ($mc->getResultCode() != Memcached::RES_SUCCESS)
) {
    die("Not Found");
}
$mc = new MemCached();
// ON LOCAL MACHINE WE CAN CONNECT VIA UNIX SOCKETS FOR BETTER SPEED
$mc->addServer('/var/run/memcached/11211.sock', 0);

// ADD/OR REPLACE, DON'T CARE, JUST GET IT IN THERE
// WITHOUT EXPIRATION PARAMETER, WILL REMAIN IN CACHE "FOREVER"
$mc->set('key1', array(1,2,3));

$key_set = array('key1' => 'foo', 'key1' => array(1,2,3));

// STORE MULTIPLE KEYS AT ONCE FOR 1 HOUR
$mc->setMulti($key_set, 3600);

// GET MULTIPLE KEYS AT ONCE
$data = $mc->getMulti(array_keys($key_set));
/*
 ARRAY(
     'key1' => 'foo'
     'key2' => array(1,2,3)
 )
*/

For multi-(get|set), all ops must succeed for successful return.
Multiple Servers

$mc = new MemCached();

// add multiple servers to the list
// as many servers as you like can be added
$mc->addServers(array(
    array('localhost', 11211, 80), // high-priority 80%
    array('192.168.1.90', 11211, 20)// low-priority 20%
));

// You can also do it one at a time, but this is not recommended
$mc->addServer('localhost', 11211, 80);
$mc->addServer('192.168.1.90', 11211, 20);

// Get a list of servers in the pool
$mc->getServerList();
// array(array('host' => ... , 'port' => ... 'weight' => ...))
Architecture...
Architecture...
The modulo, or “naive approach” used to distribute keys across many servers is pretty simple.

\[
\text{
$server\_idx = \text{crc32('my\_key') \% $num\_servers};$
}
\]

Convert key to an integer and do modulo by the # of servers in the pool.
The modulo, or “naive approach” used to distribute keys across many servers is pretty simple.

```
$server_idx = crc32('my_key') % $num_servers;
```

Convert key to an integer and do modulo by the # of servers in the pool.

**But what if the number of servers changes?**
The modulo, or “naive approach” used to distribute keys across many servers is pretty simple:

```bash
$server_idx = crc32('my_key') % $num_servers;
```

But what if the number of servers changes?
Consistent Hashing

Select \( N \) random integers for each server and sort them into an array of values \( N \times \text{# of servers} \).
Consistent Hashing

Select $N$ random integers for each server and sort them into an array of values $N \times \text{# of servers}$
Select $N$ random integers for each server and sort them into an array of values $N \times \# \text{ of servers}$

Lookup key’s int hash proximity to randomly picked values in clock-wise manner.
// default, modulo distribution mode
$mem->setOption(
    Memcached::OPT_DISTRIBUTION,
    Memcached::DISTRIBUTION_MODULA
);

// consistent hashing
$mem->setOption(
    Memcached::OPT_DISTRIBUTION,
    Memcached::DISTRIBUTION_CONSISTENT
);
Memcached interface allows you to store certain types of data on specific servers

```php
$mc = new MemCached();
$mc->addServers(...);

// Add data_key with a value of "value" for 10 mins to server identified by "server_key"
$mc->addByKey('server_key', 'data_key', 'value', 600);

// Fetch key from specific server
$mc->getByKey('server_key', 'data_key');

// Add/update key on specific server
$mc->setByKey('server_key', 'data_key', 'value', 600);

// Remove key from specific server
$mc->deleteByKey('server_key', 'data_key');
```
The specific-server interface also supports multi-(get|set)

```php
$mc = new MemCached();
$mc->addServers( ... );

$key_set = array('key1' => "foo", 'key2' => array(1,2,3));

// STORE MULTIPLE KEYS AT ONCE FOR 1 HOUR
$mc->setMultiByKey('server_key', $key_set, 3600);

// GET MULTIPLE KEYS AT ONCE
$data = $mc->getMultiByKey('server_key', array_keys($key_set));
```
Fail-Over Callbacks

```php
$m = new Memcached();
$m->addServer('localhost', 11211);

$data = $m->get('key',
    function (Memcached $memc, $key, &$value) {
        $value = 'RETRIEVE VALUE';
        $memc->add($key, $value);
        return $value;
    }
);
```

Only supported for get() & getByKey() methods
One of the really neat features of Memcached extension is the ability to execute the “fetch” command, but defer the actual data retrieval until later.

Particularly handy when retrieving many keys that won’t be needed until later.
$mc = new MemCached();
$mc->addServer('localhost', '11211');

$mc->getDelayed(array('key'));
// parameter is an array of keys

/* some PHP code that does “stuff” */

// Fetch data one record at a time
while ($data = $mc->fetch()) { ... }

// Fetch all data in one go
$data = $mc->fetchAll();
The delayed result callback allows execution of code upon successful delayed retrieval.
$m = \textbf{new Memcached}();
$m->\textbf{addServer}('localhost', 11211);

$m->\textbf{getDelayed}(
    \textbf{array}('\textbf{footer}', '\textbf{header}'), \textbf{false}, '\textbf{cb}');

\textbf{function cb(Memcached} $m, $data) {
    \textbf{function} cb(Memcached} $m, $data) {
    //\textbf{data} = \textbf{array}('\textbf{key}' => '...', '\textbf{value}' => '...');
    \textbf{function cb(Memcached} $m, $data) {
    //\textbf{data} = \textbf{array}('\textbf{key}' => '...', '\textbf{value}' => '...');
    \textbf{layout::data[\textbf{key}]}($data[\textbf{value}]));
    \textbf{layout::data[\textbf{key}]}($data[\textbf{value}]));

    \textbf{Callback will be called individually for every key}
Compare \& Swap (CAS) is a mechanism for updating a value, unless it was changed by someone else already.

```
$cas = null;

// fetch last_seen value,
// retrieving CAS into $cas by-ref
$value = $mem->get("last_seen", null, $cas);

// update last_seen, unless updated
// by someone else already
$mem->cas($cas, "last_seen", time());
```
$mc = \textbf{new} \text{MemCACHED}();
$mc->addServer('localhost', 11211);

// ADD KEY POSITION IF DOES NOT ALREADY EXIST
if (!$mc->add('key_pos', 1)) {
    // OTHERWISE INCREMENT IT
    $position = $mc->increment('key_pos');
} else {
    $position = 1;
}

// ADD REAL VALUE AT THE NEW POSITION
$mc->add('key_value_'. $position, array(1,2,3));

Simplifies cache invalidation and reduces lock contention
Global Namespacing

Global key namespacing allows rapid invalidation of all keys, on major changes, such as a software version upgrade.

```php
$mem->setOption(
    Memcached::OPT_PREFIX_KEY,
    "_" . PHP_VERSION . "_"
);

$mem->set("foo", "bar");
// actual key is _5.3.3-pl11-gentoo_foo

$mem->get("foo");
// gets value from _5.3.3-pl11-gentoo_foo
```
Buffered Writes

When doing many consecutive writes, or writing large data blocks, use buffered writes.

$m->setOption(Memcached::OPT_BUFFER_WRITES, true);

Significant performance increase...
In many cases performance can be gained by compressing large blocks of data. Since in most cases network IO is more expensive than CPU speed + RAM.

```php
$mc = new MemCached();
$mc->addServer('localhost', 11211);
// ENABLE COMPRESSION
$mc->setOption(Memcached::OPT_COMPRESSION, TRUE);
```
Related INI settings (INI_ALL)

Other possible value is zlib

```
memcached.compression_type=fastlz
```

**Minimum compression rate**
```
memcached.compression_factor=1.3
```

**Minimum data size to compress**
```
memcached.compression_threshold=2000
```
If you are using Memcache to store complex data types (arrays & objects), they will need to be converted to strings for the purposes of storage, via serialization.

Memcached can make use of igbinary serializer that works faster (~30%) and produces more compact data set (up-to 45% smaller) than native PHP serializer.

http://github.com/igbinary
Enabling Igbinary

Install Memcached extension with

```bash
--enable-memcached-igbinary
```

```php
$mem = new MemCached();
$mem->addServer('localhost', 11211);

// use Igbinary serializer
$mem->setOption(
    Memcached::OPT_SERIALIZER,
    Memcached::SERIALIZER_IGBINARY
);
```
Utility Methods

$mc = new MemCached();
$mc->addServer('localhost', 11211);

// MEMCACHED STATISTICS GATHERING
$mc->getStats();

Array
(
    [server:port] => Array
        (
            [pid] => 4933
            [uptime] => 786123
            [threads] => 1
            [time] => 1233868010
            [pointer_size] => 32
            [rusage_user_seconds] => 0
            [rusage_user_microseconds] => 140000
            [rusage_system_seconds] => 23
            [rusage_system_microseconds] => 210000
            [curr_items] => 145
            [total_items] => 2374
            [limit_maxbytes] => 67108864
            [curr_connections] => 2
            [total_connections] => 151
            [connection_structures] => 3
            [bytes] => 20345
            [cmd_get] => 213343
            [cmd_set] => 2381
            [get_hits] => 204223
            [get_misses] => 9120
            [evictions] => 0
            [bytes_read] => 9092476
            [bytes_written] => 15420512
            [version] => 1.2.6
        )
)

// CLEAR ALL CACHE ENTRIES
$mc->flush();

// CLEAR ALL CACHE ENTRIES
// IN 10 MINUTES
$mc->flush(600);
Installing Memcached

Download memcached from http://www.memcached.org and compile it.

Download libmemcached from http://tangent.org/552/libmemcached.html and compile it.

pecl install memcached (configure, make, make install)

Enable Memcached from your php.ini file
Installing Memcached

If you want the latest Memcached sources checkout Github:

http://github.com/php-memcached-dev
# Session settings

`session.save_handler` # set to "memcached

`session.save_path` # set to memcache host server:port

`memcached.sess_prefix` # Defaults to memc.sess.key.

# Locking Controls

# Whether to enable session lock, on by default
`memcached.sess_locking`

# Maximum number of microseconds to wait on a lock
`memcached.sess_lock_wait`
Performance

Seconds for 100k runs

- Memcache
- Memcached

get
add
replace

Wednesday, June 1, 2011
Thank You For Listening

Slides will be available at http://ilia.ws

Please give me your feedback http://joind.in/3509