

Using Memcached with PHP

Introduction

Many common processes utilized in web application development (for example generating large amounts of HTML or fetching data via complex SQL queries) are processor and disk intensive. Serializing and caching the generated data to disk solves the first of those problems, but you still get a small performance hit each time you read the data to disk, plus you have to write routines to automatically purge old cached data.

Memcached solves this problem for you by letting you write frequently accessed data directly to RAM, where it can be fetched almost instantaneously; plus you can tell memcached how long to store the cached data and it will automatically purge stale data without any further intervention from you, the programmer.

Many web sites & applications such as Facebook, LiveJournal, Flickr, Slashdot, WikiPedia/MediaWiki, SourceForge, Digg and Twitter use memcached to enhance their performance.

What is Memcached?

Memcached (Memory Cache Daemon) was developed by the team at LiveJournal to improve performance of their social blogging site by minimizing the impact of the bottleneck caused by reading data directly from the database. Memcached is a server that caches *Name Value Pairs* in memory. The “Name”, or key, is limited to 250 characters, and the “Value” is limited to 1MB in size. Values can consist of data, HTML Fragments, or binary objects; almost any type of data that can be serialized and fits in memcached can be stored.

Drawbacks/Limitations to Memcached

- Memcached is not a persistent data storage mechanism. The data cannot be queried SQL-style or dumped to disk; the only way to access the information in Memcached is retrieve the data from the server via the correct key (or “Name”) for the information stored. There is no fail-over/high availability support; if a memcached server goes down or the service is restarted, the data is gone.
- In many cases there is much less RAM available than disk space, so the amount of memory you can dedicate to memcached is more than likely fairly limited.
- Also-- and this is very important-- there is **NO** authentication mechanism built into memcached. If you're on a shared web host that supports per-user instances of memcached (they do exist; WebFaction is a very good example) or your instance(s) of memcached are listening on IP addresses/ports that are publically accessible, you

most certainly do not want to store any sort of personal or high-security data in memcached.

- Things like resource variables (file handles, etc.) and internal PHP classes cannot be stored using memcached, because they cannot be serialized.

Installation

Installation and configuration of the memcached daemon varies from one flavor of UNIX/Linux to another. Whatever flavor you're using almost certainly has memcached in its package management program; consult Google to find out exactly how to configure and install it as well as how to make sure your PHP installation has the memcached functions available.

If you're running Windows, here are some instructions for setting up memcached on Windows & configuring XAMPP to load the memcached module:

<http://techgurulive.com/2009/09/01/how-to-install-memcache-for-xampp/>

Configuring Memcached

Both Windows and Linux version of the Memcached server use the command line interface to configure the server instance. The three key parameters are:

- p – sets up the port on which the Memcached server instance will be listening (defaults to 11211)
- m – specifies the amount of memory (in MB) to utilize for the server instance (defaults to 64MB)
- d – sets up the Memcached server instance as a daemon

For more information on configuring Memcached, type in `memcached -h`.

When you are configuring memcached, be sure to set it up with adequate memory for your caching needs. When memcached runs out of memory, it will start to eliminate cached data starting with the oldest data, so if you don't have enough memory, you'll be generating cached data much more often than you probably want.

Also, be sure to note what IP address and port the memcached daemon is listening on; you'll need this information to write your PHP code.

Once you've got memcached set up, start the service, and we're ready to go!

Memcached Clients

Memcached clients are available for the following programming platforms:

- Perl
- PHP
- Python
- Ruby
- Java
- .Net/C#
- C
- Postgres
- Chicken

A note on using different clients to populate and access the same data cache: Different clients will use different hashing schemas to hash the keys (or “Name”) for your NameValuePairs (NVPs). If you are planning to populate and/or replenish your cache with a command line client, you need to make sure it utilizes the same hashing schema as your web client or the web client will be unable to access the cached information populated by the command line client.

Getting Started

PHP provides both procedural and object-oriented interfaces to memcached. My examples will use the object-oriented interface. If you're interested in using the procedural interface, consult the PHP manual (<http://www.php.net/memcache>).

We'll assume that our memcached instance is listening on localhost on port 12345.

```
$cache = new Memcache();  
$cache->connect('localhost', 12345);
```

Before you start storing data, you'll want to make sure that you actually were able to connect to the memcached instance.

```
if ($cache->getServerStatus() == 0) die('Could not connect to  
memcached!');
```

You may also at some point want to flush the cache without restarting the service:

```
$cache->flush();
```

Storing Data

The next step is to actually store some data in the memcached instance. Each bit of data you store will be identified by a unique "key." You'll need to be sure that you don't use the same key to store two different bits of data; if you do, the second bit will overwrite the first, as you might expect.

```
$mydata = array(  

```

```
'user_id' => 1234,  
'fullname' => 'John Doe',  
'phone' => '(123) 234-3456'  
);  
$cache->set('my-memcached-key', $mydata, MEMCACHED_COMPRESSED,  
1500);  
$cache->set('uncompressed-data', 'Just a string ...', null, 0);
```

Now you've stored the contents of `$mydata` in the cache!

The third and fourth options in the `set()` method specify whether or not to compress the cached data using `zlib` (use `null` for no compression) and how long in milliseconds to keep the cached data (use `0` for no expiration) respectively. Remember that setting no expiration date doesn't mean that the data will be cached forever ... if `memcached` runs out of memory it will start to remove old data, even data with no expiration date.

Retrieving, Replacing & Deleting Data

Retrieving cached data is very straightforward; you do need to remember to let `memcached` know if the data you are fetching was compressed.

```
echo $cache->get('my-memcached-key', MEMCACHED_COMPRESSED);  
echo $cache->get('uncompressed-data');
```

If you want to replace cached data, you've got two options. First, you can use the `Memcache::replace()` function, which will return `false` and not store your data if the cache key does not exist, or you can use `Memcache::set()`, which will replace the data if it exists, and store the value if it does not exist.

```
$cache->replace('uncompressed-data', 'New Value', null, 1500);  
or  
$cache->set('uncompressed-data', 'New Value', null, 1500);
```

Deleting data is also straightforward ...

```
$cache->delete('my-memcached-key');
```

Incrementing and Decrementing Numeric Values

`Memcached` also provides the `Memcache::increment()` and `Memcache::decrement()` methods for increasing or decreasing the value of cached numeric values by a certain value. These functions will NOT create the value if it does not exist, though.

```
$cache->set('connection-count', 1, null, 60000);
```

```
$cache->increment('connection-count', 1);  
$cache->decrement('connection-count', 1);
```

Useful Websites

Official Memcached site: <http://www.danga.com/memcached/>

Windows Server: <http://www.splinedancer.com/memcached-win32>