Memcached in Zend Framework

What is Memcached

Taken straight from the Memcached website, “Memcached is a high-performance, distributed memory object caching system”. In other words, Memcached helps you increase application perfomance by caching data in memory. Using Memcached to load data instead of a database or file system can have a major impact on your applications performance. (Insert meaningless benchmark term such as “ten-fold” here.) The best part is these benefits only increase as load increases, so you get increase scalability for free.
A Quick Overview of Zend_Cache

Before continuing down the Memcached path in ZF, it’s important to get a picture of how caching is implemented with Zend_Cache. With Zend_Cache, you must configure a frontend “strategy” (Zend_Cache_Frontend) and a backend “engine” (Zend_Cache_Backend). Each caching object must have a backend and a frontend.

The backend engine is exactly what you think it is: the method that Zend_Cache uses to actually store cached data. Zend Framework currently supports File, SqlLite, XCache, Memcached, Apc, and Zend Platform as backends. There’s another backend called “TwoLevels”, which stores data in either a fast backend or a slow backend depending on priority. (I admittedly am not completely familiar with how this works or why you would use it.)

The frontend strategy determines both how expiration is handled, and how you implement caching. For example, you can implement Zend_Cache_Frontend_File to cache data until a certain file is changed. A real-world example of this is caching an instance of Zend_Config_Xml until the XML file it points to is changed. The most basic frontend strategy is Zend_Cache_Core, which basically says “tell me what to cache, and tell me when it should expire”. Zend_Cache_Core is what we’ll use in our example. Implementing With Zend Framework

The first thing we need to do is setup our caching object. As you guessed, Zend Framework implements Memcached through Zend_Cache_Backend_Memcached. I found Zend_Cache_Core to be the most useful for a first run in my applications, so I suggest you start there as well.

```
// configure caching backend strategy
$oBackend = new Zend_Cache_Backend_Memcached(
 array(
  'servers' => array( array(
   'host' => '127.0.0.1',
   'port' => '11211'
  ) ),
  'compression' => true
) );

// configure caching logger
$oCacheLog =  new Zend_Log();
$oCacheLog->addWriter( new Zend_Log_Writer_Stream( 'file:///tmp/pr-memcache.log' ) );

// configure caching frontend strategy
$oFrontend = new Zend_Cache_Core(
 array(
  'caching' => true,
  'cache_id_prefix' => 'myApp',
  'logging' => true,
  'logger'  => $oCacheLog,
  'write_control' => true,
  'automatic_serialization' => true,
  'ignore_user_abort' => true
 ) );

// build a caching object
$oCache = Zend_Cache::factory( $oFrontend, $oBackend );
```

Here we’re creating a new Zend_Cache_Core (frontend) instance and a Zend_Cache_Backend_Memcached instance which uses a Zend_Log instance for logging. Zend_Cache_Backend_Memcached configuration: serversA comma separated list of servers that this instance should use.compressionDetermines if data should be compressed before it’s written. Zend_Cache_Core configuration: cachingthis enables caching. This is useful if you want to keep the on/off switch for caching out of your logic. In my applications, I’m passing a Zend_Config variable here.cache_id_prefixThis is the value that will be pre-pended to they id (index) you choose for each cached item. This

allows you to use Memcached for multiple applications without worrying about naming collisions.loggingThis enables logging for this backend. You must pass in a logger to the `logger` option, described below.loggerThis should be an instance of Zend_Log that you want to use for logging.write_controlthis performs a consistency check whenever data is written to cache. It’s safer, but slower.automatic_serializationTurning this option on allows you to transparently pass data which is not a string or number. For example, you can pass in an instance of Zend_Xml_Config without serializing it first, and expect an instance back when you read from cache.ignore_user_abortIf this is set, ignore_user_abort will be set to true while cache is being written. This helps prevent data corruption.

Once everything is configured and instantiated, we put it together with Zend_Cache::factory(), which gives us a fully configured caching object. Now, we can use our caching object in our code: $sCacheId = 'LargeDataSet';

if (! $oCache->test($sCacheId)) {

//cache miss, so we have to get the data the hard way

$aDataSet = doExpensiveQuery();
$oCache->save($aDataset, $sCacheId);

} else {

//cache hit, load from memcache. Zoom Zoom.

$aDataSet = $oCache->load($sCacheId);
}

The logic is rather simple. First, we do a test against or memcache server to see if the data exists and is usable. If it’s not, we load the data the way we normally would without caching. If it is available, we just load it from memcache and go about our business.

Note that if you do not enable automatic_serialization like we did above, you can use a much simpler construct. When you’re only storing strings, and Zend_Cache doesn’t do any transformation for you, the data is not tested for integrity (which I assume means that it can be unserialized correctly). //----------
// Simpler construct when automatic_serialization is turned off
//----------
$sCacheId = 'LargeDataSet';

if (! ($aDataSet = $oCache->load($sCacheId))) {

//cache miss, so we have to get the data the hard way

$aDataSet = doExpensiveQuery();
$oCache->save($aDataset, $sCacheId);
}

Notice that you save a few lines of code, and you essentially only have to wrap your existing code in a single if block. Just remember — if it’s not a string or a number, you will have to serialize/deserialize it yourself Conclusion

Implementing Memcached with Zend Framework is incredibly simple. If you have the proper resources and privileges to install Memcached, I can’t think of any reason you would not want to do it. A few additional lines of code can instantly make your application perform better and scale easier.

For more information about caching using Zend_Cache, please visit the Zend Framework documentation on this subject: http://framework.zend.com/manual/en/zend.cache.html

Source: http://ajbrown.org/blog/?p=51