# hibernate-memcached简介

A library for using [Memcached](http://www.danga.com/memcached/) as a second level distributed cache in Hibernate.

* Based on the excellent [spymemcached client](http://code.google.com/p/spymemcached/)
* Includes support for the Whalin (danga) memcached client
* Supports entity and query caching.
* See the [Configuration](http://code.google.com/p/hibernate-memcached/wiki/Configuration) page
* See how easy it is to [configure in Grails](http://code.google.com/p/hibernate-memcached/wiki/ConfiguringGrails)

Project home：http://code.google.com/p/hibernate-memcached/

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Configuration *How to configure hibernate-memcached* Adding hibernate-memcached to your application If you are using Maven2 for your application build you can follow the instructions on the [MavenRepository](http://code.google.com/p/hibernate-memcached/wiki/MavenRepository) page. For those that aren't using Maven2 you'll need to collect a few simple jars for yourself. I am going to assume you already have hibenrnate3 setup before you came here.  You'll need...   * hibernate-memcached-(version).jar * [memcached-2.1.jar](http://spymemcached.googlecode.com/files/memcached-2.1.jar) * [spy-2.4.jar](http://bleu.west.spy.net/%7Edustin/repo/spy/jars/spy-2.4.jar) * [commons-codec 1.3](http://repo1.maven.org/maven2/commons-codec/commons-codec/1.3/commons-codec-1.3.jar) * [slf4j-1.5.0.jar](http://repo1.maven.org/maven2/org/slf4j/slf4j-api/1.5.0/slf4j-api-1.5.0.jar) * An slf4j implementation for your preferred logging framework. For log4j you can use [slf4j-log4j12-1.5.0.jar](http://repo1.maven.org/maven2/org/slf4j/slf4j-log4j12/1.5.0/slf4j-log4j12-1.5.0.jar).  Configuring Hibernate The first thing to do is to tell hibernate which cache provider you'd like to use, and to enable the second level cache.   |  |  | | --- | --- | | hibernate.cache.provider\_class | com.googlecode.hibernate.memcached.MemcachedCacheProvider |   Using just the property above you get basic entity caching by default. The hibernate-memcached library supports query caching, but you have to enable query caching separately.   |  |  | | --- | --- | | hibernate.cache.use\_query\_cache | true |   !Now you can start setting up the hibernate-memcached specific properties. The hibernate-memcached properties are divided into two categories; cache-wide settings and cache-region settings. Cache Wide Settings The cache wide settings all have defaults can mostly be overwritten at a cache-region level.   |  |  |  | | --- | --- | --- | | **Property** | **Default** | **Description** | | hibernate.memcached.servers | localhost:11211 | Space delimited list of memcached instances in host:port format | | hibernate.memcached.cacheTimeSeconds | 300 | The default number of seconds items should be cached. Can be overriden at the region level. | | hibernate.memcached.keyStrategy | HashCodeKeyStrategy | Sets the strategy class to to use for generating cache keys. Must provide a class name that implements KeyStrategy | | hibernate.memcached.readBufferSize | DefaultConnectionFactory.DEFAULT\_READ\_BUFFER\_SIZE | The read buffer size for each server connection from this factory | | hibernate.memcached.operationQueueLength | DefaultConnectionFactory.DEFAULT\_OP\_QUEUE\_LEN | Maximum length of the operation queue returned by this connection factory | | hibernate.memcached.operationTimeout | DefaultConnectionFactory.DEFAULT\_OPERATION\_TIMEOUT | Default operation timeout in milliseconds | | hibernate.memcached.hashAlgorithm | HashAlgorithm.KETAMA\_HASH | Which hash algorithm to use when adding items to the cache. Note: the MemcachedClient defaults to using HashAlgorithm.NATIVE\_HASH, while the hibernate-memcached cache defaults to KETAMA\_HASH for "consistent hashing" | | hibernate.memcached.clearSupported | false | Enables support for the MemcachedCache.clear() method for all cache regions. The way clear is implemented for memcached is expensive and adds overhead to all get/set operations. It is not recommended for production use. |  Cache Region Settings Cache region properties are set by giving your cached data a "region name" in hibernate. You can tune the MemcachedCache instance for your region using the following properties. These properties essentially override the cache-wide properties above.   |  |  |  | | --- | --- | --- | | **Property** | **Default** | **Description** | | hibernate.memcached.[region-name].cacheTimeSeconds | none, see hibernate.memcached.cacheTimeSeconds | Set the cache time for this cache region, overriding the cache-wide setting. | | hibernate.memcached.[region-name].keyStrategy | none, see hibernate.memcached.keyStrategy | Overrides the strategy class to to use for generating cache keys in this cache region. Must provide a class name that implements KeyStrategy | | hibernate.memcached.[region-name].clearSupported | none, see hibernate.memcached.clearSupported | Enables clear() operations for this cache region only. Again, the clear operation incurs cost on every get/set operation. | |