BigMemory Go Integrations

Innovation Release

Version 4.3.5

April 2018

TERRACOTTA

This document applies to BigMemory Go Version 4.3.5 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: BMG-INT-435-20180417

Table of Contents

1 Using BigMemory Go with Hibernate

About Using BigMemory Go with Hibernate

BigMemory Go easily integrates with the Hibernate Object/Relational persistence and query service. To configure BigMemory Go for Hibernate:

- Download and install BigMemory Go in your project as described in "Downloading and Installing BigMemory Go for Hibernate" on page 6.
- Configure BigMemory Go as a cache provider in your project's Hibernate configuration as described in "Configuring BigMemory Go as the Second-Level Cache Provider" on page 7.
- Enable second-level caching in your project's Hibernate configuration as described in "Enabling Second-Level Cache and Query Cache Settings" on page 8.
- Configure Hibernate caching for each entity, collection, or query that you want to cache as described in "Configuring Hibernate Entities to use Second-Level Caching" on page 9.
- Configure the `ehcache.xml` file for each entity, collection, or query configured for caching as described in "Configuring ehcache.xml Settings" on page 10.

For additional information about cache configuration in Hibernate, see the Hibernate product documentation at "http://www.hibernate.org/".

Downloading and Installing BigMemory Go for Hibernate

The Hibernate provider is in the ehcache-ee module provided in the BigMemory Go kit.

Building with Maven

Dependency versions vary with the specific kit you intend to use. Each kit is guaranteed to contain compatible artifacts, so find the artifact versions you need by downloading a kit. Configure or add the following repository to your build (pom.xml):

```
<repository>
   <id>terracotta-releases</id>
   <url>http://www.terracotta.org/download/reflector/releases</url>
   <releases><enabled>true</enabled></releases>
   <snapshots><enabled>false</enabled></snapshots>
</repository>
```

Configure or add the defined by the following dependency to your build (pom.xml):

```
<dependency>
  <groupId>net.sf.ehcache</groupId>
  <artifactId>ehcache-ee</artifactId>
  <version>${ehcacheVersion}</version>
</dependency>
<dependency>
```

```
  <groupId>org.terracotta.bigmemory</groupId>
  <artifactId>bigmemory</artifactId>
  <version>${bigmemoryVersion}</version>
</dependency>
```

For the Hibernate-Ehcache integration, add the following dependency:

```
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-ehcache</artifactId>
  <version>${hibernateVersion}</version>
</dependency>
```

For example, the Hibernate-Ehcache integration dependency for Hibernate 4.0.0 is:

```
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-ehcache</artifactId>
  <version>4.0.0</version>
</dependency>
```

Note: Some versions of hibernate-ehcache might have a dependency on a specific version of Ehcache. Check the hibernate-ehcache POM.

Configuring BigMemory Go as the Second-Level Cache Provider

To configure BigMemory Go as a Hibernate second-level cache, set the region factory property to one of the following in the Hibernate configuration. The Hibernate configuration is specified either by hibernate.cfg.xml, hibernate.properties or Spring. The format shown below is for hibernate.cfg.xml.

Hibernate 3.3 (and later 3.x versions)

For instance creation, use:

```
<property name="hibernate.cache.region.factory_class">
        net.sf.ehcache.hibernate.EhCacheRegionFactory</property>
```

To force Hibernate to use a singleton of Ehcache CacheManager, use:

```
<property name="hibernate.cache.region.factory_class">
        net.sf.ehcache.hibernate.SingletonEhCacheRegionFactory</property>
```

Hibernate 4.x

For Hibernate 4, use `org.hibernate.cache.ehcache.EhCacheRegionFactory` instead of `net.sf.ehcache.hibernate.EhCacheRegionFactory`, or `org.hibernate.cache.ehcache.SingletonEhCacheRegionFactory` instead of `net.sf.ehcache.hibernate.SingletonEhCacheRegionFactory`.

Enabling Second-Level Cache and Query Cache Settings

In addition to configuring the second-level cache provider setting, you will need to turn on the second-level cache (by default it is configured to off - 'false' - by Hibernate). To do this, set the following property in your Hibernate config:

```
<property name="hibernate.cache.use_second_level_cache">true</property>
```

You might also want to turn on the Hibernate query cache. To do this, set the following property in your Hibernate config:

```
<property name="hibernate.cache.use_query_cache">true</property>
```

Setting the ConfigurationResourceName Property

You can optionally set the `ConfigurationResourceName` property to specify the location of the Ehcache configuration file to use with the given Hibernate instance and cache provider/region-factory. The resource is searched for in the root of the classpath. It is used to support multiple CacheManagers in the same VM. It tells Hibernate which configuration to use. An example might be "ehcache-2.xml."

When using multiple Hibernate instances, it is recommended to use multiple non-singleton providers or region factories, each with a dedicated Ehcache configuration resource.

```
net.sf.ehcache.configurationResourceName=/name_of_ehcache.xml
```

Setting the Hibernate Cache Provider Programmatically

You can optionally specify the provider programmatically in Hibernate by adding necessary Hibernate property settings to the configuration before creating the SessionFactory:

```
Configuration.setProperty("hibernate.cache.region.factory_class",
                      "net.sf.ehcache.hibernate.EhCacheRegionFactory")
```

For Hibernate 4, use `org.hibernate.cache.ehcache.EhCacheRegionFactory` instead of `net.sf.ehcache.hibernate.EhCacheRegionFactory`.

Putting it all Together

If you are enabling both second-level caching and query caching, then your Hibernate config file should contain the following:

```
<property name="hibernate.cache.use_second_level_cache">true</property>
<property name="hibernate.cache.use_query_cache">true</property>
<property name="hibernate.cache.region.factory_class">
     net.sf.ehcache.hibernate.EhCacheRegionFactory</property>
```

An equivalent Spring configuration file would contain:

```
<prop key="hibernate.cache.use_second_level_cache">true</prop>
<prop key="hibernate.cache.use_query_cache">true</prop>
<prop key="hibernate.cache.region.factory_class">
     net.sf.ehcache.hibernate.EhCacheRegionFactory</prop>
```

For Hibernate 4, use `org.hibernate.cache.ehcache.EhCacheRegionFactory` instead of `net.sf.ehcache.hibernate.EhCacheRegionFactory` in both samples given above.

Configuring Hibernate Entities to use Second-Level Caching

In addition to configuring the Hibernate second-level cache provider, Hibernate must also be configured to enable caching for entities, collections, and queries. For example, to enable cache entries for the domain object com.somecompany.someproject.domain.Country, there would be a mapping file similar to the following:

```
<hibernate-mapping>
<class
name="com.somecompany.someproject.domain.Country"
table="ut_Countries"
dynamic-update="false"
dynamic-insert="false"
>
...
</class>
</hibernate-mapping>
```

To enable caching for this domain object, you add the following element to its mapping entry:

```
<cache usage="read-write|nonstrict-read-write|read-only" />
```

For example:

```
<hibernate-mapping>
<class
name="com.somecompany.someproject.domain.Country"
table="ut_Countries"
dynamic-update="false"
dynamic-insert="false"
>
 <cache usage="read-write" />
...
</class>
</hibernate-mapping>
```

You can also enable caching using the @Cache annotation as shown below.

```
@Entity
@Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
public class Country {
...
}
```

Definition of the Different Cache Strategies

- `read-only` - Caches data that is never updated.
- `nonstrict-read-write` - Caches data that is sometimes updated without ever locking the cache. If concurrent access to an item is possible, this concurrency strategy makes no guarantee that the item returned from the cache is the latest version available in the database. Configure your cache timeout accordingly.

- `read-write` - Caches data that is sometimes updated while maintaining the semantics of "read committed" isolation level. If the database is set to "repeatable read," this concurrency strategy almost maintains the semantics. Repeatable-read isolation is compromised in the case of concurrent writes.

Configuring ehcache.xml Settings

Because the ehcache.xml file has a defaultCache, caches will always be created when required by Hibernate. However you can gain more control over Hibernate caches by configuring each cache based on its name. Doing this is particularly important, because Hibernate caches are populated from databases, and there is potential for them to become very large. You can control the size of a Hibernate cache by capping its `maxEntriesLocalHeap` property and specifying whether to swap to disk beyond that.

Ehcache Settings for Domain Objects

Hibernate bases the names of Domain Object caches on the fully qualified name of Domain Objects. So, for example, a cache for com.somecompany.someproject.domain.Country would be represented by a cache configuration entry in ehcache.xml similar to the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<ehcache>
 <cache
    name="com.somecompany.someproject.domain.Country"
    maxEntriesLocalHeap="10000"
    eternal="false"
    timeToIdleSeconds="300"
    timeToLiveSeconds="600"
    <persistence strategy="localTempSwap"/>
 />
</ehcache>
```

Hibernate CacheConcurrencyStrategy for Domain Objects

The read-write, nonstrict-read-write and read-only policies apply to Domain Objects.

Ehcache Settings for Collections

Hibernate creates collection cache names based on the fully qualified name of the Domain Object followed by "." and the collection field name. For example, a Country domain object has a set of advancedSearchFacilities. The Hibernate doclet for the accessor looks like this:

```
/**
* Returns the advanced search facilities that should appear for this country.
* @hibernate.set cascade="all" inverse="true"
* @hibernate.collection-key column="COUNTRY_ID"
* @hibernate.collection-one-to-many class="com.wotif.jaguar.domain.AdvancedSearchFacility"
* @hibernate.cache usage="read-write"
*/
public Set getAdvancedSearchFacilities() {
```

```
return advancedSearchFacilities;
}
```

You need an additional cache configured for the set. The ehcache.xml configuration looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<ehcache>
 <cache name="com.somecompany.someproject.domain.Country"
    maxEntriesLocalHeap="50"
    eternal="false"
    timeToLiveSeconds="600"
    <persistence strategy="localTempSwap"/>
/>
 <cache
name="com.somecompany.someproject.domain.Country.advancedSearchFacilities"
    maxEntriesLocalHeap="450"
    eternal="false"
    timeToLiveSeconds="600"
    <persistence strategy="localTempSwap"/>
/>
</ehcache>
```

Hibernate CacheConcurrencyStrategy for Collections

The read-write, nonstrict-read-write and read-only policies apply to Domain Object collections.

Ehcache Settings for Queries

Hibernate allows the caching of query results.

StandardQueryCache

This cache is used if you use a query cache without setting a name. A typical ehcache.xml configuration is:

```
<cache
name="org.hibernate.cache.StandardQueryCache"
maxEntriesLocalHeap="5"
eternal="false"
timeToLiveSeconds="120"
<persistence strategy="localTempSwap"/>
/>
```

UpdateTimestampsCache

Tracks the timestamps of the most recent updates to particular tables. It is important that the cache timeout of the underlying cache implementation is set to a higher value than the timeouts of any of the query caches. Therefore, it is recommend that the underlying cache not be configured for expiry at all. A typical ehcache.xml configuration is:

```
<cache
name="org.hibernate.cache.UpdateTimestampsCache"
maxEntriesLocalHeap="5000"
eternal="true"
<persistence strategy="localTempSwap"/>
/>
```

Named Query Caches

In addition, a QueryCache can be given a specific name in Hibernate using Query.setCacheRegion(String name). The name of the cache in ehcache.xml is then the name given in that method. The name can be whatever you want, but by convention you should use "query." followed by a descriptive name. For example:

```
<cache name="query.AdministrativeAreasPerCountry"
maxEntriesLocalHeap="5"
eternal="false"
timeToLiveSeconds="86400"
<persistence strategy="localTempSwap"/>
/>
```

Using Query Caches

Suppose you have a common query running against the Country Domain. Here is the code to use a query cache with it:

```
public List getStreetTypes(final Country country) throws HibernateException {
final Session session = createSession();
try {
   final Query query = session.createQuery(
    "select st.id, st.name"
   + " from StreetType st "
   + " where st.country.id = :countryId "
   + " order by st.sortOrder desc, st.name");
   query.setLong("countryId", country.getId().longValue());
   query.setCacheable(true);
   query.setCacheRegion("query.StreetTypes");
   return query.list();
} finally {
   session.close();
}
}
```

The `query.setCacheable(true)` line caches the query. The `query.setCacheRegion("query.StreetTypes")` line sets the name of the Query Cache. Alex Miller has a good article on the query cache at "http://tech.puredanger.com/2009/07/10/hibernate-query-cache/".

Hibernate CacheConcurrencyStrategy for Queries

None of the read-write, nonstrict-read-write and read-only policies apply to Domain Objects. Cache policies are not configurable for query cache. They act like a non-locking read only cache.

The Demo Application and Tutorial

A demo application is available that shows you how to use the Hibernate CacheRegionFactory. You can download the application from here: "http://svn.terracotta.org/svn/forge/projects/hibernate-tutorial-web/trunk".

Performance Tips

Session.load

`Session.load` will always try to use the cache.

Session.find and Query.find

`Session.find` does not use the cache for the primary object. Hibernate will try to use the cache for any associated objects. `Session.find` does, however, cause the cache to be populated. `Query.find` works in exactly the same way. Use these where the chance of getting a cache hit is low.

Session.iterate and Query.iterate

`Session.iterate` always uses the cache for the primary object and any associated objects. `Query.iterate` works in exactly the same way. Use these where the chance of getting a cache hit is high.

Viewing Hibernate Statistics

It is possible to access the Hibernate statistics and BigMemory Go statistics using the Java Management Extensions (JMX).

The EhcacheHibernateMBean is the main interface that exposes all the APIs via JMX. It basically extends two interfaces: EhcacheStats and HibernateStats. As the names imply, EhcacheStats contains methods related with Ehcache (and thereby, BigMemory Go) and HibernateStats contains methods related with Hibernate.

Using these APIs, you can see cache hit/miss/put rates, change config element values (e.g., maxElementInMemory, TTL TTI), enable/disable statistics collection, and various other things. For details, see the specific interface.

FAQ

If I use BigMemory Go with my application and with Hibernate for second-level caching, should I try to use the CacheManager created by Hibernate for my app's caches?

While you could share the resource file between the two CacheManagers, a clear separation between the two is recommended. Your application may have a different lifecycle than Hibernate, and in each case your CacheManager "Automatic Resource Control" settings might need to be different.

Should I use the provider in the Hibernate distribution or in BigMemory Go's Ehcache?

Since Hibernate 2.1, Hibernate has included an Ehcache CacheProvider. That provider is periodically synced up with the provider in the Ehcache Core distribution. New features are generally added in to the Ehcache Core provider and then the Hibernate one.

Does BigMemory Go support the transactional strategy?

Yes. It was introduced in Ehcache 2.1.

Why do certain caches sometimes get automatically cleared by Hibernate?

Whenever a Query.executeUpdate() is run, Hibernate invalidates affected cache regions (those corresponding to affected database tables) to ensure that no stale data is cached. This should also happen whenever stored procedures are executed.

For more information, see the Hibernate issue HHH-2224 at : "https://hibernate.atlassian.net/browse/HHH-2224".

How are Hibernate entities keyed?

Hibernate identifies cached entities using an object id. This is normally the primary key of a database row.

Are compound keys supported?

Yes.

I am getting this error message: "An item was expired by the cache while it was locked." What is it?

Soft locks are implemented by replacing a value with a special type that marks the element as locked, thus indicating to other threads to treat it differently than a normal element. This is used in the Hibernate Read/Write strategy to force fall-through to the database during the two-phase commit. We cannot know exactly what should be returned by the cache while the commit is in process (but the database does). If a soft-locked element is evicted by the cache during the two-phase commit, then once the two-phase commit completes, the cache will fail to update (since the soft-locked element was evicted) and the cache entry will be reloaded from the database on the next read of that object. This is non-fatal, but could increase the database load slightly.

In summary the Hibernate messages are not problematic. The underlying cause is that the probabilistic evictor can theoretically evict recently loaded items. You can also use the deterministic evictor to avoid this problem. Specify the `-Dnet.sf.ehcache.use.classic.lru=true` system property to turn on classic LRU, which contains a deterministic evictor.

2 Using BigMemory Go with ColdFusion

About ColdFusion and BigMemory Go

ColdFusion ships with BigMemory Go's Ehcache. The ColdFusion community has actively engaged with Ehcache and put out several blogs. Here are two to get you started. For a short introduction, see "Raymond Camden's blog". For more in-depth analysis, see "14 days of ColdFusion caching", by Aaron West.

Example Integration

To integrate BigMemory Go with ColdFusion, first add the BigMemory Go jars to your web application lib directory.

The following code demonstrates how to call Ehcache from ColdFusion. It will cache a ColdFusion object and set the expiration time to 30 seconds. If you refresh the page many times within 30 seconds, you will see the data from cache. After 30 seconds, you will see a cache miss, then the code will generate a new object and put it in cache again.

```
<CFOBJECT type="JAVA" class="net.sf.ehcache.CacheManager" name="cacheManager">
<cfset cache=cacheManager.getInstance().getCache("MyBookCache")>
<cfset myBookElement=#cache.get("myBook")#>
<cfif IsDefined("myBookElement")>
   <cfoutput>
    myBookElement: #myBookElement#<br />
   </cfoutput>
   <cfif IsStruct(myBookElement.getObjectValue())>
           <strong>Cache Hit</strong><p/>
           <!-- Found the object from cache -->
           <cfset myBook = #myBookElement.getObjectValue()#>
   </cfif>
</cfif>
<cfif IsDefined("myBook")>
<cfelse>
<strong>Cache Miss</strong>
   <!-- object not found in cache, go ahead create it -->
   <cfset myBook = StructNew()>
   <cfset a = StructInsert(myBook, "cacheTime", LSTimeFormat(Now(), 'hh:mm:sstt'), 1)>
   <cfset a = StructInsert(myBook, "title", "EhCache Book", 1)>
   <cfset a = StructInsert(myBook, "author", "Greg Luck", 1)>
   <cfset a = StructInsert(myBook, "ISBN", "ABCD123456", 1)>
   <CFOBJECT type="JAVA" class="net.sf.ehcache.Element" name="myBookElement">
   <cfset myBookElement.init("myBook", myBook)>
   <cfset cache.put(myBookElement)>
</cfif>
<cfoutput>
Cache time: #myBook["cacheTime"]#<br />
Title: #myBook["title"]#<br />
Author: #myBook["author"]#<br />
ISBN: #myBook["ISBN"]#
</cfoutput>
```

3 Using BigMemory Go with Spring

Using Spring 3.1

BigMemory Go's Ehcache supports Spring integration. Spring 3.1 includes an Ehcache implementation. See the "Spring 3.1 JavaDoc".

Spring Framework 3.1 has a generic cache abstraction for transparently applying caching to Spring applications. It has caching support for classes and methods using two annotations:

@Cacheable

Cache a method call. In the following example, the value is the return type, a Manual. The key is extracted from the ISBN argument using the id.

```
@Cacheable(value="manual", key="#isbn.id")
public Manual findManual(ISBN isbn, boolean checkWarehouse)
```

@CacheEvict

Clears the cache when called.

```
@CacheEvict(value = "manuals", allEntries=true)
public void loadManuals(InputStream batch)
```

Spring 2.5 to 3.1

This open source, led by Eric Dalquist, predates the Spring 3.1 project. You can use it with earlier versions of Spring, or you can use it with 3.1.

@Cacheable

As with Spring 3.1, it uses the @Cacheable annotation to cache a method. In this example, calls to findMessage are stored in a cache named "messageCache". The values are of type `Message`. The id for each entry is the `id` argument given.

```
@Cacheable(cacheName = "messageCache")
public Message findMessage(long id)
```

@TriggersRemove

For cache invalidation, there is the @TriggersRemove annotation. In this example, `cache.removeAll()` is called after the method is invoked.

```
@TriggersRemove(cacheName = "messagesCache",
when = When.AFTER_METHOD_INVOCATION, removeAll = true)
public void addMessage(Message message)
```

See "http://blog.goyello.com/2010/07/29/quick-start-with-ehcache-annotations-for-spring/" for a blog post explaining its use and providing further links.

Annotations for Spring Project

To dynamically configure caching of method return values, use the Ehcache Annotations for Spring project at "Ehcache Annotations for Spring project at code.google.com". This project will allow you to configure caching of method calls dynamically. The parameter values of the method are used as a composite key into the cache, caching the return value of the method.

For example, suppose you have a method Dog getDog(String name).

Once caching is added to this method, all calls to the method are cached using the `name` parameter as a key.

So, assume at time t0 the application calls this method with the name equal to "fido". Because "fido" doesn't exist, the method is allowed to run, generating the "fido" Dog object, and returning it. This object is then put into the cache using the key "fido".

Then assume at time t1 the application calls this method with the name equal to "spot". The same process is repeated, and the cache is now populated with the Dog object named "spot."

Finally, at time t2 the application again calls the method with the name "fido". Since "fido" exists in the cache, the "fido" Dog object is returned from the cache instead of calling the method.

To implement this in your application, follow these steps:

Step 1:

Add the jars to your application as listed on the Ehcache Annotations for Spring project at "Ehcache Annotations for Spring project at code.google.com".

Step 2:

Add the Annotation to the methods you want to cache. Let's assume you are using the Dog getDog(String name) method from above:

```
@Cacheable(name="getDog")
Dog getDog(String name)
{
    ....
}
```

Step 3:

Configure Spring. You must add the following to your Spring configuration file in the beans declaration section:

```
<ehcache:annotation-driven cache-manager="ehCacheManager" />
```

More details can be found at:

- Ehcache Annotations for Spring project at code.google.com at "http://code.google.com/p/ehcache-spring-annotations".
- The project getting started page at "http://code.google.com/p/ehcache-spring-annotations/wiki/UsingCacheable".
- The article "Caching Java methods with Spring 3" at "http://www.jeviathon.com/2010/04/caching-java-methods-with-spring-3.html"

4 Using BigMemory Go with JSR107

About BigMemory Go Support for JSR107

Information about BigMemory Go's Ehcache support for JSR107 is available on github at "https://github.com/jsr107/ehcache-jcache".