Ehcache Cache Plugin - Reference Documentation

Authors: Burt Beckwith
Version: 1.0.4

Table of Contents

1 Introduction To The Cache Plugin

The Grails Cache Ehcache plugin extends the Cache plugin and uses Ehcache as the storage provider for cached content. It replaces the core plugin's `grailsCacheManager` bean with one that uses an Ehcache backend.

1.1 Change log

Version 1.0.0 - July 4, 2012

Version 1.0.0.M2 - May 12, 2012

2 Usage

Although this plugin uses a different backing store than the core plugin, its usage is the same. You annotate service methods, controller action methods, and taglib closures with the three caching annotations, and configure caches in `Config.groovy` and/or *CacheConfig.groovy artifacts in `grails-app/conf`. See the core plugin docs for general usage information.

Ehcache supports many configuration options and most are available in the cache DSL, including support for distributed caching. See the DSL section for details.

2.1 Cache DSL

The full Ehcache DSL will not be documented because except for some exceptions and extensions, the DSL is nearly identical to the XML format for the `ehcache.xml` file, although of course the syntax is somewhat different. The `ehcache.xml` syntax is described in this well commented example and you can view the XML Schema here. The DSL uses types where appropriate, for example numeric and boolean values; this isn't possible in XML since all attributes must be strings.

General configuration

The configuration options closures (entirely optional) could look like this example:

```
grails{
   cache {
      enabled = true
      ehcache {
         ehcacheXmlLocation = 'classpath:ehcache.xml' // conf/ehcache.xml
         reloadable = false
      }
   }
}
```

- `enabled` determines if the plugin is enabled. If false, then all caching operations are no-ops (as if all cache retrievals are misses).
- `ehcacheXmlLocation` optionally specifies where `ehcache.xml` can be found.
- `reloadable` determines if groovy and ehcache.xml specified configuration can be reloaded at one time (false means configuration is only read at startup).

Caches

Like in the core plugin, you configure a cache with the `cache` call, and set configuration options in a closure, for example

```
grails.cache.config = {
   cache {
      name 'mycache'
      eternal false
      overflowToDisk true
      maxElementsInMemory 10000
      maxElementsOnDisk 10000000
   }
}
```

The `name` attribute is mandatory, and all of the other attributes correspond to the ehcache.xml XML attributes for the `<cache>` tag.

Default cache and cache defaults

Ehcache supports auto-creation of caches and this is enabled in this implementation, so if you call the `getCache(String name)` method on the `CacheManager` (via the `grailsCacheManager` bean) and the cache didn't already exist it would be created and returned. Since it didn't already exist there is no configuration for it, so it uses the settings for the "default cache", for example:

```
grails.cache.config = {
   cache {
      …
   }
defaultCache {
      maxElementsInMemory 10000
      eternal false
      timeToIdleSeconds 120
      timeToLiveSeconds 120
      overflowToDisk true
      maxElementsOnDisk 10000000
      diskPersistent false
      diskExpiryThreadIntervalSeconds 120
      memoryStoreEvictionPolicy 'LRU'
   }
}
```

It is important to specify the `defaultCache` configuration since Ehcache will throw an exception if you attempt to auto-create a cache and there are is no default configuration. You may want this behavior though, since if you are confident that you have specified all known caches then auto-creating a new cache is probably due to a typo or programming error.

Don't confuse `defaultCache` with `defaults`. The latter helps keep your configuration DRY and isn't a feature of Ehcache. This feature lets you specify default values that all caches must have, for example

```
grails.cache.config = {
   cache {
      …
   }
defaults {
      maxElementsInMemory 1000
      eternal false
      overflowToDisk false
      maxElementsOnDisk 0
   }
}
```

These values are set on each cache configuration while the DSL is being parsed, and you can override any value in a `cache` definition, for example where most caches use the default value but a few have custom values.

Note that the special `hibernateQuery` and `hibernateTimestamps` caches (see below) and caches auto-created by Ehcache using the values in `defaultCache` do not use values in the `defaults` configuration. The Hibernate caches don't because they are special-purpose caches and unlikely to share general default settings, and the auto-created caches don't because Ehcache doesn't know about this feature.

The "provider" block

All cache DSLs support a `provider` block, it isn't applicable for all DSLs. This is a way to specify provider-specific values, for example

```
grails.cache.config = {
   cache {
      …
   }
provider {
      updateCheck false
      monitoring 'on'
      dynamicConfig false
      defaultTransactionTimeoutInSeconds 30
      maxBytesLocalHeap 10
      maxBytesLocalOffHeap 20
      maxBytesLocalDisk 30
   }
}
```

In the case of Ehcache, these settings end up in the top-level `<ehcache>` tag, e.g.

```
<ehcache defaultTransactionTimeoutInSeconds="30" dynamicConfig="false"
         maxBytesLocalDisk="30" maxBytesLocalHeap="10"
         maxBytesLocalOffHeap="20" monitoring="on" updateCheck="false">
```

Hibernate

To have Hibernate use ehcache, modify DataSource.groovy:

```
hibernate {
   cache.use_second_level_cache = true // set based on whether the second
level cache is desired
   cache.use_query_cache = true // set based on whether the query cache is
desired
   cache.region.factory_class =
'grails.plugin.cache.ehcache.hibernate.BeanEhcacheRegionFactory' // For
Hibernate before 4.0
   cache.region.factory_class =
'grails.plugin.cache.ehcache.hibernate.BeanEhcacheRegionFactory4' // For
Hibernate 4.0 and higher
}
```

Hibernate second-level cache

It is possible to use Ehcache as the Hibernate second-level cache provider and as the provider for the caches managed by this plugin, or use different providers for each. It's more convenient to use Ehcache for both however and the DSL has support for configuring both at the same time.

Hibernate uses two caches, one for its query cache (with name "org.hibernate.cache.StandardQueryCache") and one for tracking the timestamps of the most recent table updates (with name "org.hibernate.cache.UpdateTimestampsCache"). The DSL supports two methods that create reasonable configurations for these caches (but you're free to configure your own). Including a call to the `hibernateQuery()` method will generate this XML:

```
<cache name="org.hibernate.cache.StandardQueryCache"
       maxElementsInMemory="50"
       timeToLiveSeconds="120"
       eternal="false"
       overflowToDisk="true"
       maxElementsOnDisk="0"
/>
```

and a call to the `hibernateTimestamps()` method will generate this XML:

```
<cache name="org.hibernate.cache.UpdateTimestampsCache"
       maxElementsInMemory="5000"
       eternal="true"
       overflowToDisk="false"
       maxElementsOnDisk="0"
/>
```

Hibernate domain class second-level caches

Typically you configure a cache with the `cache` call, but "domain" is an alias for "cache" and can be used when configuring caches for domain classes to make it clear what the cache is used for. Hibernate expects that the cache name is the full class name including the package; you can specify the full class and package as the "name" attribute:

```
grails.cache.config = {
   domain {
      name 'com.yourcompany.yourapp.Person'
      eternal false
      overflowToDisk true
      maxElementsInMemory 10000
      maxElementsOnDisk 10000000
   }
}
```

but you also use the class name to get a compile-time verification that the class is correct:

```
import com.yourcompany.yourapp.Person

…
grails.cache.config = {
   domain {
      name Person
      eternal false
      overflowToDisk true
      maxElementsInMemory 10000
      maxElementsOnDisk 10000000
   }
}
```

When you have configured caching for a mapped collection (for example the `books` collection in an `Author` class) Hibernate expects the cache name to be the containing class full name plus the collection field name. Use the `domainCollection` call with an inner `domain` attribute to tell the DSL what the name of the owning class is, e.g.

```
grails.cache.config = {
   domain {
      name 'com.foo.Author'
      eternal false
      overflowToDisk true
      maxElementsInMemory 10000
      maxElementsOnDisk 10000000
   }
domainCollection {
      name 'books'
      domain 'com.foo.Author'
      eternal true
      overflowToDisk true
      maxElementsInMemory 100
      maxElementsOnDisk 10000
   }
}
```

Or you can use the more compact inner closure syntax and omit the `domain` attribute:

```
grails.cache.config = {
   domain {
      name 'com.foo.Author'
      eternal false
      overflowToDisk true
      maxElementsInMemory 10000
      maxElementsOnDisk 10000000

collection {
         name 'books'
         eternal true
         overflowToDisk true
         maxElementsInMemory 100
         maxElementsOnDisk 10000
      }
   }
}
```

diskStore

If any cache supports overflowing to disk, you need to configure where to store this data with the `diskStore` call. You can specify an absolute path (but this is likely to be a brittle and non-portable approach):

```
grails.cache.config = {
   …
   diskStore {
      path '/tmp/ehcache'
   }
}
```

You can also call `temp true` to use the value of the `java.io.tmpdir` system property:

```
grails.cache.config = {
   …
   diskStore {
      temp true
   }
}
```

or `home true` to use the value of the `user.home` system property:

```
grails.cache.config = {
   …
   diskStore {
      home true
   }
}
```

or `current true` to use the value of the `user.dir` system property:

```
grails.cache.config = {
   …
   diskStore {
      current true
   }
}
```

Overriding values

You can specify cache configuration in multiple places; the application's `Config.groovy`, the application's *CacheConfig.groovy artifact files in `grails-app/conf`, and in plugins' *CacheConfig.groovy artifact files in their `grails-app/conf` directories. Configurations that are loaded later will replace previously set values and add new ones; this can be configured with the `order` attribute.

So for example a plugin might have this configuration:

```
order = 500

config = {
   cache {
      name 'the_cache'
      eternal false
      overflowToDisk true
      maxElementsInMemory 2
      maxElementsOnDisk 3
   }
}
```

and the application's `Config.groovy` might have this:

```
grails.cache.config = {
   cache {
      name 'the_cache'
      maxElementsInMemory 10000
      maxElementsOnDisk 10000000
   }

cache {
      name 'another_cache'
      …
   }

defaults {
      timeToLiveSeconds 1234
   }
}
```

Since the plugin's `order` is lower than that of the `Config.groovy` (since order is unspecified it defaults to 1000) the resulting configuration will be as if this single configuration existed:

```
grails.cache.config = {
   cache {
      name 'the_cache'
      eternal false
      overflowToDisk true
      maxElementsInMemory 10000
      maxElementsOnDisk 10000000
   }

cache {
      name 'another_cache'
      …
   }

defaults {
      timeToLiveSeconds 1234
   }
}
```

Environments

A `cache` (or `domain`) element can specify one or more environment names to define when the cache is valid; when the DSL is evaluated the XML will only contain caches that are valid for the active environment. You can specify a single string:

```
grails.cache.config = {
   domain {
      name 'com.foo.Other'
      env 'staging'
   }
}
```

or a list of strings

```
grails.cache.config = {
   domain {
      name 'com.foo.Book'
      env(['staging', 'production'])
   }
}
```

and if no environments are specified the cache will be included in all environments.

Distributed caches

The DSL includes support for distributed caches, which are implemented with the `cacheManagerPeerProviderFactory`, `cacheManagerPeerListenerFactory`, `cacheManagerEventListenerFactory`, `bootstrapCacheLoaderFactory`, `cacheExceptionHandlerFactory`, `cacheEventListenerFactory`, `cacheLoaderFactory`, and `cacheExtensionFactory` calls. These support the same attributes as the corresponding XML elements except that the `class` XML attribute is `className` in the DSL.

In addition, there are several shortcuts for common values. These include symbols to use for time to live (TTL) numeric values for the `timeToLive` property in a `cacheManagerPeerProviderFactory` element:

DSL name	value
"host"	0
"subnet"	1
"site"	32
"region"	64
"continent"	128
"unrestricted"	255

symbols to use for `cacheManagerPeerProviderFactory` element `className` attribute:

DSL name	class name
"rmi"	"net.sf.ehcache.distribution.RMICacheManagerPeerProviderFactory"
"jgroups"	"net.sf.ehcache.distribution.jgroups.JGroupsCacheManagerPeerProviderFactory"
"jms"	"net.sf.ehcache.distribution.jms.JMSCacheManagerPeerProviderFactory"

symbols to use for `cacheEventListenerFactory` element `className` attribute:

DSL name	class name
"rmi"	"net.sf.ehcache.distribution.RMICacheReplicatorFactory"
"jgroups"	"net.sf.ehcache.distribution.jgroups.JGroupsCacheReplicatorFactory"
"jms"	"net.sf.ehcache.distribution.jms.JMSCacheReplicatorFactory"

and symbols to use for `bootstrapCacheLoaderFactory` element `className` attribute:

DSL name	class name
"rmi"	"net.sf.ehcache.distribution.RMIBootstrapCacheLoaderFactory"
"jgroups"	"net.sf.ehcache.distribution.jgroups.JGroupsBootstrapCacheLoaderFactory"

Note also that environment support (which is valid in any `cache` (or `domain`) element and any distributed element) is especially important for distributed configuration. Most users will run non-distributed in development and possibly test environments, and only enable it in production or production-like environments. By adding one or more environment values to these elements you can have a single DSL that works in any environment. For example, this Groovy code

```
grails.cache.config = {
   defaults {
      maxElementsInMemory 1000
      eternal false
      overflowToDisk false
      maxElementsOnDisk 0
      cacheEventListenerFactoryName 'cacheEventListenerFactory'
   }

domain {
      name 'com.foo.Book'
   }

cacheManagerPeerProviderFactory {
      env 'production'
      factoryType 'rmi'
      multicastGroupAddress '${ehcacheMulticastGroupAddress}'
      multicastGroupPort '${ehcacheMulticastGroupPort}'
      timeToLive 'subnet'
   }

cacheManagerPeerListenerFactory {
      env 'production'
   }

cacheEventListenerFactory {
      env 'production'
      name 'cacheEventListenerFactory'
      factoryType 'rmi'
      replicateAsynchronously false
   }
}
```

will generate this XML in production:

```
<ehcache ...>

<diskStore path="java.io.tmpdir" />

<defaultCache
          maxElementsInMemory="10000"
          eternal="false"
          timeToIdleSeconds="120"
          timeToLiveSeconds="120"
          overflowToDisk="true"
          maxElementsOnDisk="10000000"
          diskPersistent="false"
          diskExpiryThreadIntervalSeconds="120"
          memoryStoreEvictionPolicy="LRU"
   />

<cacheManagerPeerProviderFactory
      class='net.sf.ehcache.distribution.RMICacheManagerPeerProviderFactory'
      properties="multicastGroupAddress=${ehcacheMulticastGroupAddress},
                  multicastGroupPort=${ehcacheMulticastGroupPort},
                  timeToLive=1,peerDiscovery=automatic"
      propertySeparator=','
   />

<cacheManagerPeerListenerFactory
      class='net.sf.ehcache.distribution.RMICacheManagerPeerListenerFactory'
/>

<cache name="com.foo.Book"
          maxElementsInMemory="1000"
          eternal="false"
          overflowToDisk="false"
          maxElementsOnDisk="0">

<cacheEventListenerFactory
         class='net.sf.ehcache.distribution.RMICacheReplicatorFactory'
         properties="replicateAsynchronously=false"
         propertySeparator=','
      />
   </cache>

</ehcache>
```

(note that ${ehcacheMulticastGroupAddress} and ${ehcacheMulticastGroupPort} are an Ehcache feature that lets you use system property names as variables to be resolved at runtime)

and this XML in development:

```
<ehcache ...>

<diskStore path="java.io.tmpdir" />

<defaultCache
          maxElementsInMemory="10000"
          eternal="false"
          timeToIdleSeconds="120"
          timeToLiveSeconds="120"
          overflowToDisk="true"
          maxElementsOnDisk="10000000"
          diskPersistent="false"
          diskExpiryThreadIntervalSeconds="120"
          memoryStoreEvictionPolicy="LRU"
   />

<cache name="com.foo.Book"
          maxElementsInMemory="1000"
          eternal="false"
          overflowToDisk="false"
          maxElementsOnDisk="0"
   />

</ehcache>
```

Manual Peer Discovery

In some cases it may be necessary to specify peers directly instead of via Multicast (i.e. Amazon EC2), To do this you must specify your rmiUrls via the following dsl:

```
cacheManagerPeerProviderFactory {
      env 'production'
      factoryType 'rmi'

rmiUrl '//server2:40001'
      rmiUrl '//server2:40001'
   }
```

Note: This varies from the standard xml DSL for ehcache (don't need rmiUrls pipe seperated string). The plugin is smart enough to automatically change `peerDiscovery` to 'manual' if rmiUrl is specified.