BigMemory Go Operations Guide

Version 4.3

April 2015

Document ID: BMG-OG-43-20150415

Table of Contents

1 Monitoring and Management Using JMX

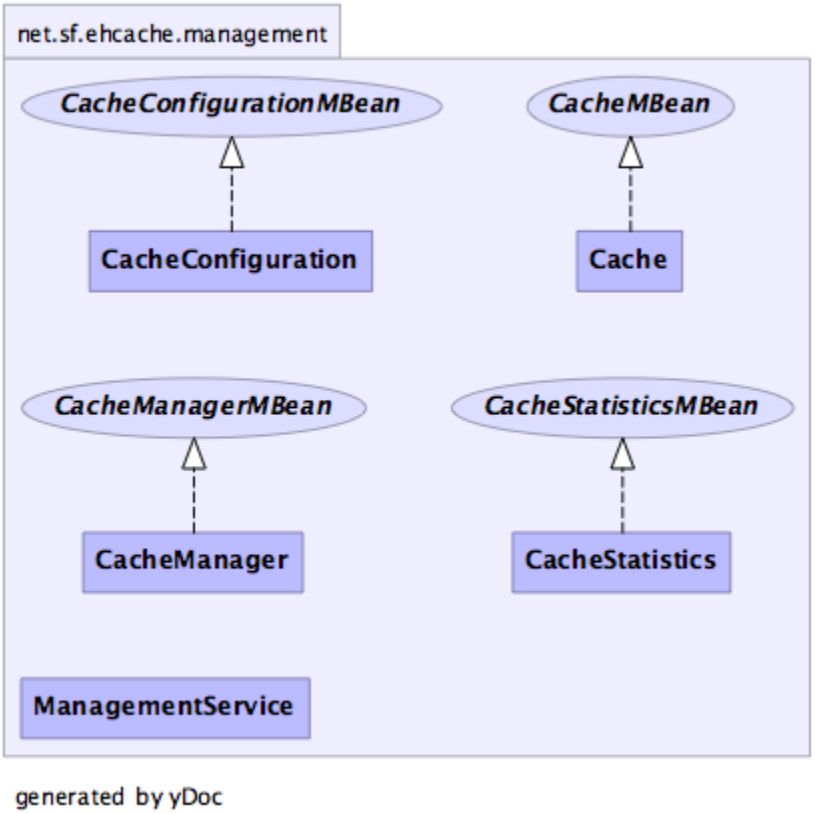
About Using JMX

JMX creates a standard way of instrumenting classes and making them available to a management and monitoring infrastructure. This provides an alternative to the Terracotta Management Server for custom or third-party tools.

The net.sf.ehcache.management package contains MBeans and a ManagementService for JMX management of BigMemory Go . It is in a separate package so that JMX libraries are only required if you want to use it. There is no leakage of JMX dependencies into the core Ehcache package.

Use the net.sf.ehcache.management.ManagementService.registerMBeans(...) static method to register a selection of MBeans to the MBeanServer provided to the method. If you want to monitor Ehcache but not use JMX, use the existing public methods on Cache and CacheStatistics.

The Management package is illustrated in the following image.



MBeans

BigMemory Go supports Standard MBeans. MBeans are available for the following:

- CacheManager
- Cache
- CacheConfiguration
- CacheStatistics

All MBean attributes are available to a local MBeanServer. The CacheManager MBean allows traversal to its collection of Cache MBeans. Each Cache MBean likewise allows traversal to its CacheConfiguration MBean and its CacheStatistics MBean.

JMX Remoting

The Remote API allows connection from a remote JMX Agent to an MBeanServer via an MBeanServerConnection. Only Serializable attributes are available remotely. The following Ehcache MBean attributes are available remotely:

- Limited CacheManager attributes
- Limited Cache attributes
- All CacheConfiguration attributes
- All CacheStatistics attributes

Most attributes use built-in types. To access all attributes, add ehcache.jar to the remote JMX client's classpath. For example:

```
jconsole -J-Djava.class.path=ehcache.jar
```

ObjectName Naming Scheme

CacheManager

```
"net.sf.ehcache:type=CacheManager,name=<CacheManager>"
```

Cache

```
"net.sf.ehcache:type=Cache,CacheManager=<cacheManagerName>,name=<cacheName>"
```

CacheConfiguration

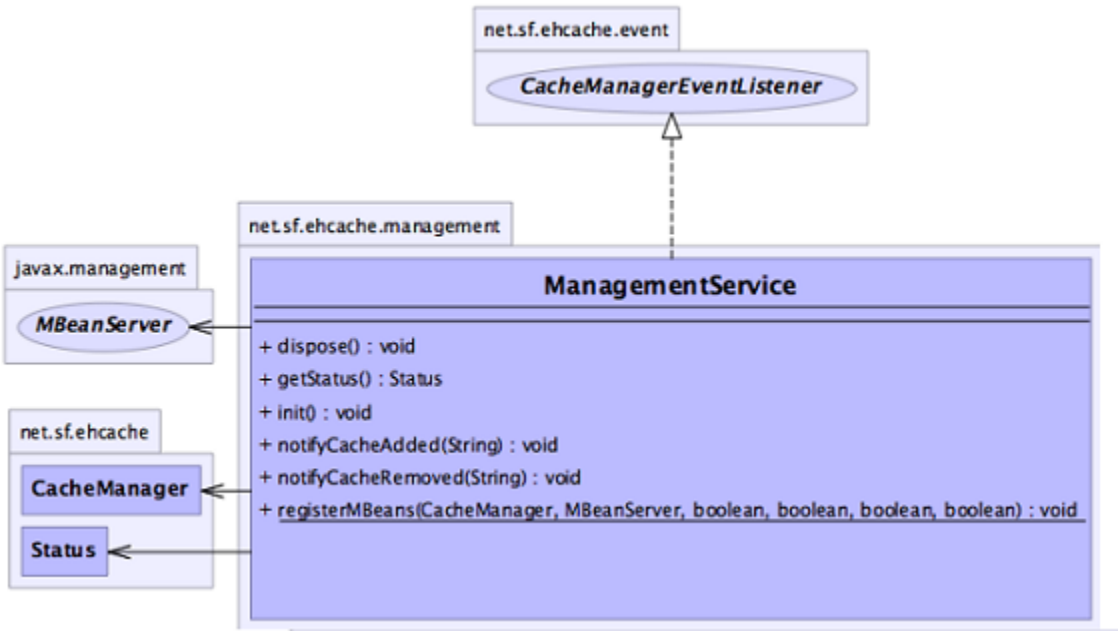
```
"net.sf.ehcache:type=CacheConfiguration,CacheManager=<cacheManagerName>,
   name=<cacheName>"
```

CacheStatistics

```
"net.sf.ehcache:type=CacheStatistics,CacheManager=<cacheManagerName>,
   name=<cacheName>"
```

The Management Service

The ManagementService class is the API entry point.



There is only one method, ManagementService.registerMBeans, which is used to initiate JMX registration of a CacheManager's instrumented MBeans. The ManagementService is a CacheManagerEventListener and is therefore notified of any new Caches added or disposed and updates the MBeanServer appropriately.

Initiated MBeans remain registered in the MBeanServer until the CacheManager shuts down, at which time the MBeans are deregistered. This ensures correct behavior in application servers where applications are deployed and undeployed.

```
/**
* This method causes the selected monitoring options to be be registered
* with the provided MBeanServer for caches in the given CacheManager.
*
* While registering the CacheManager enables traversal to all of the other
* items, this requires programmatic traversal. The other options allow entry
* points closer to an item of interest and are more accessible from JMX
* management tools like JConsole. Moreover CacheManager and Cache are not
* serializable, so remote monitoring is not possible for CacheManager or
* Cache, while CacheStatistics and CacheConfiguration are.
* Finally, CacheManager and Cache enable management operations to be performed.
*
* Once monitoring is enabled caches will automatically added and removed from the
* MBeanServer as they are added and disposed of from the CacheManager. When the
* CacheManager itself shutsdown all registered MBeans will be unregistered.
*
```

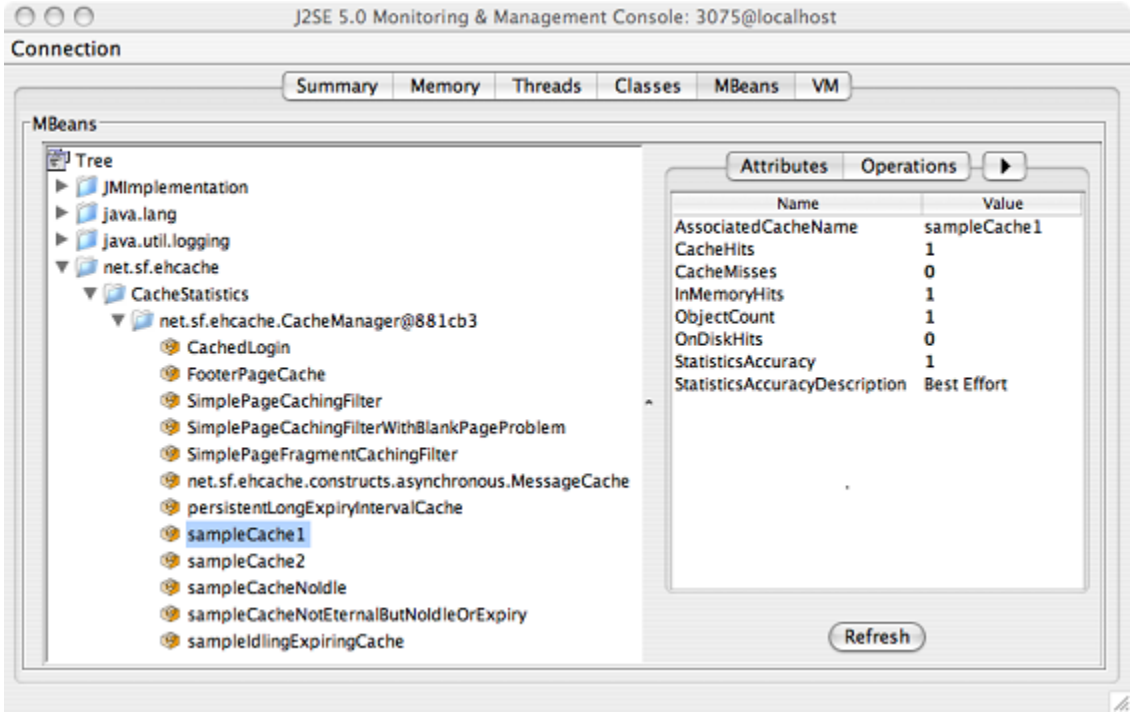
```
* @param cacheManager the CacheManager to listen to
* @param mBeanServer the MBeanServer to register MBeans to
* @param registerCacheManager Whether to register the CacheManager MBean
* @param registerCaches Whether to register the Cache MBeans
* @param registerCacheConfigurations Whether to register the CacheConfiguration
MBeans
* @param registerCacheStatistics Whether to register the CacheStatistics MBeans
*/
public static void registerMBeans(
   net.sf.ehcache.CacheManager cacheManager,
   MBeanServer mBeanServer,
   boolean registerCacheManager,
   boolean registerCaches,
   boolean registerCacheConfigurations,
   boolean registerCacheStatistics) throws CacheException {
```

JConsole Example

This example shows how to register CacheStatistics in the JDK platform MBeanServer, which works with the JConsole management agent.

```
CacheManager manager = new CacheManager();
MBeanServer mBeanServer = ManagementFactory.getPlatformMBeanServer();
ManagementService.registerMBeans(manager, mBeanServer, false, false, false,
true);
```

CacheStatistics MBeans are then registered. The following shows CacheStatistics MBeans in JConsole.



JMX Tutorial

See this online tutorial.

Performance Considerations

Collection of cache statistics is not entirely free of overhead, however, the statistics API switches on/off automatically according to usage. If you need few statistics, you incur little overhead; on the other hand, as you use more statistics, you can incur more. Statistics are off by default.

2 Logging

SLFJ Logging

BigMemory Go uses the SLF4J logging facade, so you can plug in your own logging framework. The following information pertains to Ehcache logging. For information about SLF4J in general, refer to the SLF4J website.

With SLF4J, users must choose a concrete logging implementation at deploy time. The options include Maven and the download kit.

Concrete Logging Implementation use in Maven

The maven dependency declarations are reproduced here for convenience. Add *one* of these to your Maven POM.

```
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-jdk14</artifactId>
    <version>1.5.8</version>
</dependency>
 <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>1.5.8</version>
</dependency>
 <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-simple</artifactId>
    <version>1.5.8</version>
</dependency>
```

Concrete Logging Implementation use in the Download Kit

The slf4j-api jar is in the kit along with the BigMemory Go jars so that, if the app does not already use SLF4J, you have everything you need. Additional concrete logging implementations can be downloaded from SLF4J website.

Recommended Logging Levels

BigMemory Go seeks to trade off informing production-support developers of important messages and cluttering the log. ERROR messages should not occur in normal production and indicate that action should be taken.

WARN messages generally indicate a configuration change should be made or an unusual event has occurred. DEBUG and TRACE messages are for development use. All DEBUG level statements are surrounded with a guard so that no performance cost is incurred unless the logging level is set. Setting the logging level to DEBUG should provide more information on the source of any problems. Many logging systems enable a logging level change to be made without restarting the application.

3 Shutting Down Ehcache

About Shutdown

BigMemory Go is shut down through the Ehcache API. Note that Hibernate automatically shuts down its Ehcache CacheManager.

The recommended way to shutdown BigMemory Go is:

- To call CacheManager.shutdown()
- In a web app, register the Ehcache ShutdownListener

Though not recommended, you can also register a JVM shutdown hook.

ServletContextListener

Ehcache provides a ServletContextListener that shuts down the CacheManager. Use this to shut down Ehcache automatically, when the web application is shut down. To receive notification events, this class must be configured in the deployment descriptor for the web application. To do so, add the following to web.xml in your web application:

```
<listener>
  <listener-class>
      net.sf.ehcache.constructs.web.ShutdownListener</listener-class>
</listener>
```

The Shutdown Hook

The CacheManager can optionally register a shutdown hook. To do so, set the system property `net.sf.ehcache.enableShutdownHook=true`. This will shut down the CacheManager when it detects the Virtual Machine shutting down and it is not already shut down.

Use the shutdown hook when the CacheManager is not already being shutdown by a framework you are using, or by your application.

Note: Shutdown hooks are inherently problematic. The JVM is shutting down, so sometimes things that can never be null are. Ehcache guards against as many of these as it can, but the shutdown hook should be the last option to use.

The shutdown hook is on CacheManager. It simply calls the shutdown method. The sequence of events is:

- Call dispose for each registered CacheManager event listener.
- Call dispose for each Cache.

Each Cache will:

- Shutdown the MemoryStore. The MemoryStore will flush to the DiskStore.

- Shutdown the DiskStore. If the DiskStore is persistent ("localRestartable"), it will write the entries and index to disk.
- Shutdown each registered CacheEventListener.
- Set the Cache status to shutdown, preventing any further operations on it.
- Set the CacheManager status to shutdown, preventing any further operations on it.

The shutdown hook runs when:

- A program exists normally. For example, when System.exit() is called, or when the last non-daemon thread exits.
- The Virtual Machine is terminated, e.g., CTRL-C. This corresponds to `kill -SIGTERM pid` or `kill -15 pid` on Unix systems.

The shutdown hook will not run when:

- The Virtual Machine aborts.
- A SIGKILL signal is sent to the Virtual Machine process on Unix systems, e.g., `kill -SIGKILL pid` or `kill -9 pid`.
- A `TerminateProcess` call is sent to the process on Windows systems.

Dirty Shutdown

If Ehcache is shutdown dirty, all in-memory data will be retained if BigMemory Go is configured for restartability. For more information, see "Configuring Fast Restart" in the *Configuration Guide* for BigMemory Go.