Ehcache Operations Guide

Version 2.10.2

April 2016

EHCACHE

This document applies to Ehcache Version 2.10.2 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: EHC-OG-2102-20160415

Table of Contents

1 Tuning Garbage Collection

About Garbage Collection and Ehcache

Applications that use Ehcache can be expected to use large heaps. Some Ehcache applications have heap sizes greater than 6 GB. Ehcache works well at this scale. However, large heaps or long held objects, which is what a cache is composed of, can place strain on the default Garbage Collector (GC). With Ehcache 2.3 and higher, this problem can be solved with BigMemory Go, an in-memory data management product, which provides an additional store outside of the heap. For more information, see the BigMemory Go product documentation.

Detecting Garbage Collection Problems

A full garbage collection event pauses all threads in the JVM. Nothing happens during the pause. If this pause takes more than a few seconds, it will become noticeable.

The clearest way to see if this is happening is to run jstat. The following command will produce a log of garbage collection statistics, updated every ten seconds.

```
jstat -gcutil <pid> 10 1000000
```

The thing to watch for is the Full Garbage Collection Time. The difference between the total time for each reading is the amount of time the system was paused. A jump of more than a few seconds will not be acceptable in most application contexts.

Tuning the Garbage Collector

Tuning suggestion for virtual machines with large heaps using caching:

```
java ... -XX:+DisableExplicitGC -XX:+UseConcMarkSweepGC
-XX:NewSize=<1/4 of total heap size> -XX:SurvivorRatio=16
```

Note that it is better to use `-XX:+DisableExplicitGC`, instead of calling System.gc(). It also helps to use the low pause collector `-XX:+UseConcMarkSweepGC`.

2 Monitoring and Management Using JMX

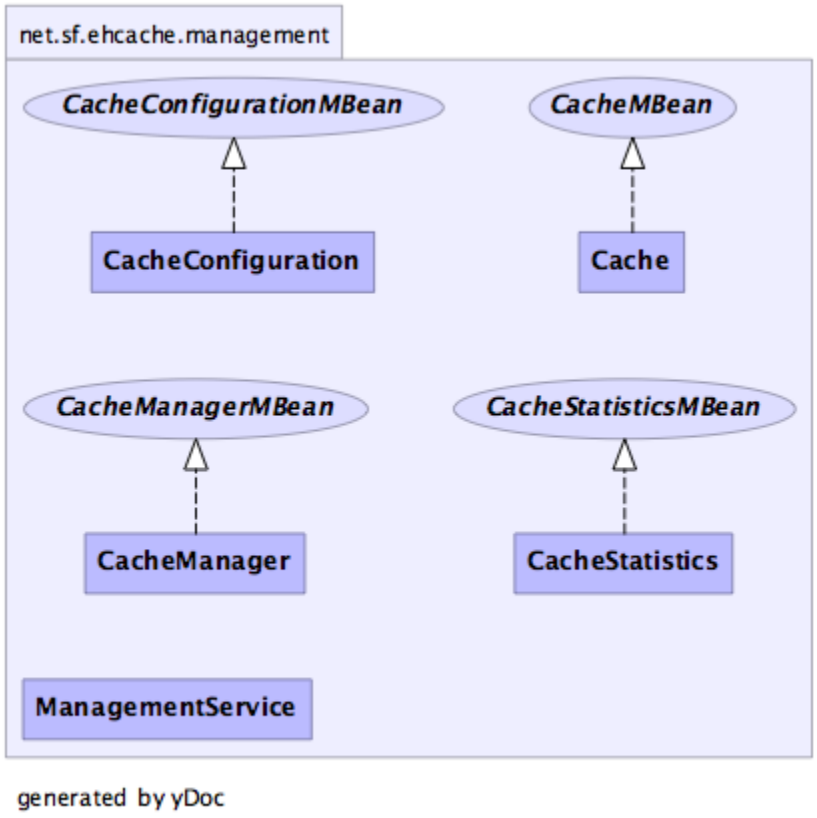
About Using JMX

JMX creates a standard way of instrumenting classes and making them available to a management and monitoring infrastructure.

The net.sf.ehcache.management package contains MBeans and a ManagementService for JMX management of Ehcache . 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.



Note: As an alternative to using JMX, the Terracotta Management Console (TMC) is available for standalone Ehcache. TMC replaces Ehcache Monitor for monitoring, management, and administration. For information about the TMC, see the Terracotta website.

MBeans

Ehcache 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

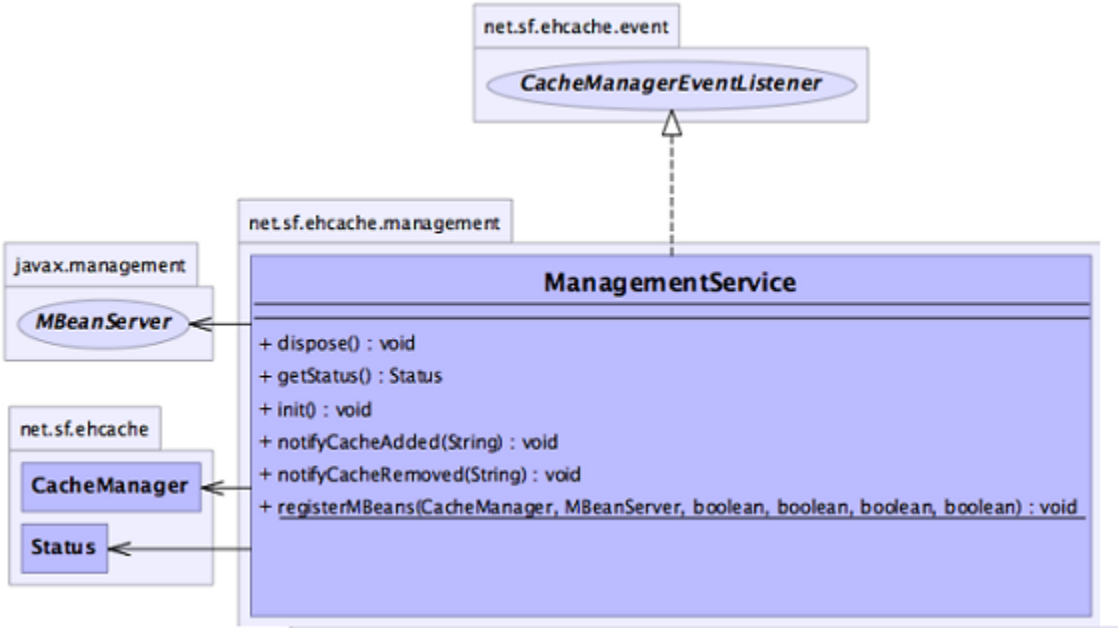
```
"net.sf.ehcache:type=CacheConfiguration,CacheManager=<cacheManagerName>,
   name=<cacheName>"
```

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.
*
```

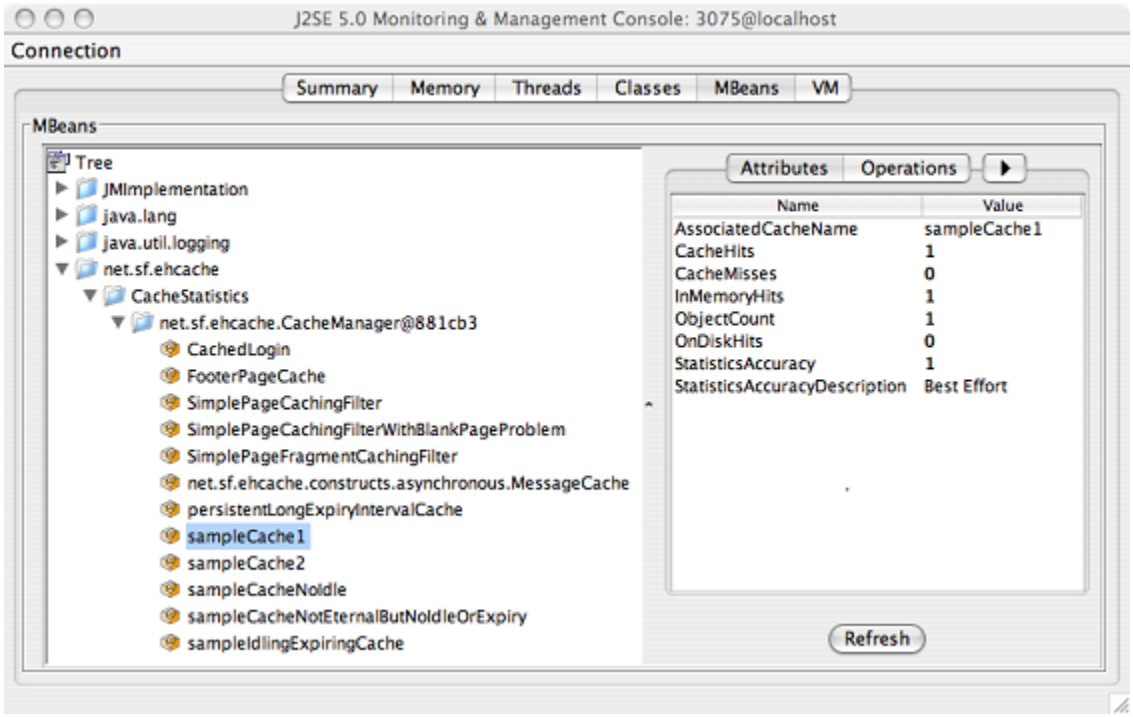
```
* @param cacheManager the CacheManager to listen to
* @param mBeanServer the MBeanServer to register MBeans to
* @param registerCacheManager Whether to register the CacheManager MBean
* @param registerCaches Whether to register the Cache MBeans
* @param registerCacheConfigurations Whether to register 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.

3 Logging

SLFJ Logging

As of 1.7.1, Ehcache 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 Ehcache 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

Ehcache 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.

4 Shutting Down Ehcache

About Shutdown

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

The recommended way to shutdown Ehcache 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 Ehcache is configured for restartability. For more information, see "Configuring Restartability and Persistence" in the *Configuration Guide* for Ehcache.

If using Open Source disk persistence when shut down dirty, then any persistent disk stores will be corrupted. They will be deleted, with a log message, on the next startup.

5 Debugging and Monitoring Replicated Caches

About the Remote Debugger

The ehcache-1.x-remote-debugger.jar can be used to debug replicated cache operations. When started with the same configuration as the cluster, it will join the cluster and then report cluster events for the cache of interest. By providing a window into the cluster it can help to identify the cause of cluster problems.

Packaging

From version 1.5 it is packaged in its own distribution tarball along with a maven module. It is provided as an executable JAR on the Ehcache download page.

Limitations

This version of the debugger has been tested only with the default RMI-based replication.

Using the Debugger

It is invoked as follows:

```
java -jar ehcache-debugger-1.5.0.jar ehcache.xml sampleCache1 path_to_ehcache.xml
[cacheToMonitor]
```

It takes one or two arguments:

- the first argument, which is mandatory, is the Ehcache configuration file e.g. app/config/ehcache.xml. This file should be configured to allow Ehcache to join the cluster. Using one of the existing ehcache.xml files from the other nodes normally is sufficient.
- the second argument, which is optional, is the name of the cache e.g. distributedCache1

If only the first argument is passed, it will print our a list of caches with replication configured from the configuration file, which are then available for monitoring. If the second argument is also provided, the debugger will monitor cache operations received for the given cache. This is done by registering a CacheEventListener which prints out each operation.

Note: Use the Class-Path attribute inside a manifest file to add any additional libraries to the debugger's classpath. Be sure to leave a blank line at the end of the file. Following is an example of Class-Path attribute inside a manifest file (showing two separate JARs added to the classpath):

```
Class-Path: lib/my.jar lib/myLoggerjar
```

Output

When monitoring a cache it prints a list of caches with replication configured, prints notifications as they happen, and periodically prints the cache name, size and total events received. See sample output below which is produced when the RemoteDebuggerTest is run.

```
Caches with replication configured which are available for monitoring are:
  sampleCache19 sampleCache20 sampleCache26 sampleCache42 sampleCache33
  sampleCache51 sampleCache40 sampleCache32 sampleCache18 sampleCache25
  sampleCache9 sampleCache15 sampleCache56 sampleCache31 sampleCache7
  sampleCache12 sampleCache17 sampleCache45 sampleCache41 sampleCache30
  sampleCache13 sampleCache46 sampleCache4 sampleCache36 sampleCache29
  sampleCache50 sampleCache37 sampleCache49 sampleCache48 sampleCache38
  sampleCache6 sampleCache2 sampleCache55 sampleCache16 sampleCache27
  sampleCache11 sampleCache3 sampleCache54 sampleCache28 sampleCache10
  sampleCache8 sampleCache47 sampleCache5 sampleCache53 sampleCache39
  sampleCache23 sampleCache34 sampleCache22 sampleCache44 sampleCache52
  sampleCache24 sampleCache35 sampleCache21 sampleCache43 sampleCache1
  Monitoring cache: sampleCache1
  Cache: sampleCache1 Notifications received: 0 Elements in cache: 0
  Received put notification for element [ key = this is an id, value=this is
  a value, version=1, hitCount=0, CreationTime = 1210656023456,
  LastAccessTime = 0 ]
  Received update notification for element [ key = this is an id, value=this
  is a value, version=1210656025351, hitCount=0, CreationTime =
  1210656024458, LastAccessTime = 0 ]
  Cache: sampleCache1 Notifications received: 2 Elements in cache: 1
  Received remove notification for element this is an id
  Received removeAll notification.
```

Providing more Detailed Logging

If you see nothing happening, but cache operations should be going through, enable trace (LOG4J) or finest (JDK) level logging on net.sf.ehcache.distribution in the logging configuration being used by the debugger. A large volume of log messages will appear. The normal problem is that the CacheManager has not joined the cluster. Look for the list of cache peers.

Yes, but I still have a cluster problem

Check the FAQ, where a lot of commonly reported errors and their solutions are provided. Beyond that, post to the forums or mailing list or contact Ehcache for support.