icw
eHEALTH FRAMEWORK

eHealth Framework

How to enable caching for the CodeSystem module

Imprint

InterComponentWare
Altrottstraße 31
69190 Walldorf
Tel.: +49 (0) 6227 385 0
Fax.: +49 (0) 6227 385 199

Document version: preliminary
Document Language: en (US)
Product Name: eHealth Framework
Product Version: 2.10.3
Last Change: 23.03.2010
Editorial Staff: BAS Technology

Notice

The wording in this document applies equally to women and men. The masculine form was selected to ease the comprehensibility and legibility of the text.

Note:

The current version of this document has a draft status and various chapters are still in review.

The document is collaboratively built with the use of the Darwin-Information-Typing-Architecture (DITA) and has therefore a draft status concerning styles and layout. The necessary adaptations are currently also in a developmental stage.

Contents

1 Overview

Purpose

This HOW-TO explains the steps required for setting up the cache for various services, in the *ehf-codesystem* module. The *ehf-codesystem* module uses EHCache internally for caching the results of each method call.

The cache is required to reduce the number of requests to the database, and thereby improve the performance of the application.

Enabling the cache is described in the section Enable the Cache on page 2 This section provides a sample configuration file. The application should be tested to arrive at the optimal values.

If the `CodeAuthoringService` is used in a distributed setup, refer to the section Distributed Caching Options on page 3 It is extremely important to provide the configuration described in the section for deploying the application in a clustered scenario. Presented here are two scenarios:

- Distributed Cache Nodes with Notifications
- Distributed Cache Nodes with Replication

Using the `CodeAuthoringService` in a distributed environment requires that cache removal notifications are distributed across nodes. The *Distributed Cache Nodes with Notifications* scenario setup is the recommended and minimum requirement for using the Code Service caches in a distributed environment.

Attention:

Please be aware that the import processor does not invalidate the cache, and therefore should not be run while the application is still active.

2 Caching for the CodeSystem module

2.1 Enable the Cache

The EHCache requires an XML file for its configuration. The EHCache configuration site explains in detail the various settings available in EHCache.

The *ehf-codesystem* module looks for a file with the name `ehcache-codesystem.xml` in the classpath under the folder META-INF ("classpath:/META-INF/ehcache-codesystem.xml") for the configurations. Omitting this file from the classpath disables the complete caching solution for the *ehf-codesystem* services. By default, the caching is disabled for the *ehf-codesystem* module, since it does not come with a configuration file.

Furthermore, the three Code Services that support caching i.e.:

- CodeFinderService
- CodeValidatorService
- CodeResourceService

need to be configured to provide caching. If a CodeService cache configuration is omitted from the `ehcache-codesystem.xml` then that service does not use caching. To enable a cache for a Code Service, simply name it with the interface name of the service i.e.:

- com.icw.ehf.commons.codesystem.CodeFinderService
- com.icw.ehf.commons.codesystem.CodeValidatorService
- com.icw.ehf.commons.codesystem.CodeResourceService

Sample config file

A sample config file would look like the one below:

```
<?xml version="1.0" encoding="UTF-8"?>
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="ehcache.xsd"
         name="codesystem-cache">

    <defaultCache maxElementsInMemory="10000"
         eternal="false"
         timeToIdleSeconds="120"
         timeToLiveSeconds="120"
         overflowToDisk="false"/>

    <cache name="com.icw.ehf.commons.codesystem.CodeFinderService"
        maxElementsInMemory="10000"
        eternal="true"
        overflowToDisk="false" />

    <cache name="com.icw.ehf.commons.codesystem.CodeResolverService"
        maxElementsInMemory="10000"
        eternal="true"
        overflowToDisk="false" />

    <cache name="com.icw.ehf.commons.codesystem.CodeValidatorService"
        maxElementsInMemory="10000"
        eternal="true"
        overflowToDisk="false" />


</ehcache>
```

The defaultCache defines the mandatory Default Cache configuration. The defaultCache has an implicit name "default" which is a reserved cache name. The *ehf-codesystem*

module does not use the default cache configuration, but the defaultCache is still required by the EHCache.

Configuration parameters for the cache element are as follows.

- **name**: Sets the name of the cache. This is used to identify the cache. It must be unique. For the `CodeValidatorService` cache we have chosen the fully specified class name as its name: "`com.icw.ehf.commons.codesystem.CodeValidatorService`".

  Other services and their corresponding names are listed below:

Service	Name
CodeValidatorService	com.icw.ehf.commons.codesystem.CodeValidatorService
CodeResolverService	com.icw.ehf.commons.codesystem.CodeResolverService
CodeFinderService	com.icw.ehf.commons.codesystem.CodeFinderService

  Table1. Service and Names
- **maxElementsInMemory**: Sets the maximum number of objects that will be held in memory
- **maxElementsOnDisk**: Sets the maximum number of objects that will be maintained in the DiskStore The default value is zero, meaning unlimited.
- **eternal**: Sets whether elements are eternal. If eternal, timeouts are ignored and the element never expires. (Unless it is evicted due to exceeding max elements.)
- **overflowToDisk**: Sets whether elements can overflow to disk when the memory store has reached the maxElementsInMemory limit.

The following attributes and elements are optional.

- **timeToIdleSeconds**: Sets the time to idle for an element before it expires i.e. the maximum amount of time between accesses before an element expires. Is only used if the element is not eternal. A value of 0 means that an Element can idle for infinity. The default value is 0.
- **timeToLiveSeconds**: Sets the time to live for an element before it expires. i.e. the maximum time between creation time and when an element expires. It is only used if the element is not eternal. A value of 0 means that and Element can live for infinity. The default value is 0.
- **diskPersistent**: Whether the disk store persists between restarts of the Virtual Machine. The default value is false.
- **diskExpiryThreadIntervalSeconds**: The number of seconds between runs of the disk expiry thread. The default value is 120 seconds.
- **diskSpoolBufferSizeMB**: This is the size to allocate the DiskStore for a spool buffer. Writes are made to this area and then asynchronously written to disk. The default size is 30MB. Each spool buffer is used only by its cache. If you get OutOfMemory errors consider lowering this value. To improve DiskStore performance consider increasing it.
- **clearOnFlush**: Whether the MemoryStore should be cleared when flush() is called on the cache. By default, this is true i.e. the MemoryStore is cleared.
- **memoryStoreEvictionPolicy**: Policy would be enforced upon reaching the maxElementsInMemory limit. Default policy is Least Recently Used (specified as LRU). Other policies available are: First In First Out (specified as FIFO) and Less Frequently Used (specified as LFU)

An optimal value for these parameters would depend on specific application installations.

2.2 Distributed Caching Options

EHCache supports Distributed Caching through several plugable cache replication mechanisms:

- RMI
- JGroups
- JMS
- Terracotta

The easiest setup approach is to use the `RMI` cache replication mechanism. Currently, the `CodeFinderService` and the `CodeResolverService` do not support serializable shared objects. Only using the `Terracotta` replication mechanism is it possible to replicate cached objects from the Code Services.

Important:

The recommended approach for setting up distributed caching for Code Services is to use the RMI cache replication mechanism. If you are using the `CodeAuthoringService` in a distributed cache environment it is manditory that you configure distributed caching with replicated cache removals. If you desire replicated cached objects by the Code Services, then setup Terracotta.

Please be aware that the settings provided here are provided as example only. There are many different ways to configure a distributed cache. For optimal performance of your caches, care should be taken to determine and provide optimal settings.

2.2.1 Distributed Cache Nodes with Notifications

Usage Scenario
These settings are for situations where distribution of cache removal notifications is required. This is the minimum and recommended requirement when using the `CodeAuthoringService` in a multi-node environment. In this scenario, the caches are isolated except for removal notifications which are replicated between the distributed caches. Therefore, when the `CodeAuthoringService` is invoked on one node, this cache is invalidated and the removals are replicated to the other distributed cache nodes.

Notes:

1. The following example uses RMI to replicate Code Service cache notifications. It also possible to use any of the other cache replication mechanisms to replicated Code Service cache notifications.
2. This example uses localhost configuration so host names and ports may need to be changed. If you are running the nodes on different machines then the host name must be changed. If you are running the nodes on the localhost machine, then you need to ensure unique port numbers in each node.

CAUTION:

It is not possible to replicate puts for cache objects for the `CodeFinderService` and the `CodeResolverService` as these services do not support serializable shared objects.

CAUTION:

In order that caches work correctly with the `CodeAuthoringService` it is required that replication of cache removals is enabled.

Important:

The following example uses a manual configuration for locating other cache nodes. It is also possible to use multicast approaches to configure the `cacheManagerPeerProviderFactory` in order to locate distributed caches.

2.2.1.1 Example Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="ehcache.xsd"
         name="codesystem-cache">



    <defaultCache maxElementsInMemory="10000"
         eternal="false"
         timeToIdleSeconds="120"
         timeToLiveSeconds="120"
         overflowToDisk="false"/>



      <cacheManagerPeerProviderFactory
           class="net.sf.ehcache.distribution.RMICacheManagerPeerProviderFactory"
           properties="peerDiscovery=manual,
           rmiUrls=//localhost:40002/com.icw.ehf.commons.codesystem.
CodeValidatorService|
                   //localhost:40002/com.icw.ehf.commons.codesystem.
CodeFinderService|
                  //localhost:40002/com.icw.ehf.commons.codesystem.
CodeResolverService" />


      <cacheManagerPeerListenerFactory
            class="net.sf.ehcache.distribution.RMICacheManagerPeerListenerFactory"
            properties="hostName=localhost, port=40001, socketTimeoutMillis=
2000" />


      <!-- The CodeValidatorService supports replication over RMI. This is an
 example
           configuration.
```

```
      -->

      <cache name="com.icw.ehf.commons.codesystem.CodeValidatorService"
            maxElementsInMemory="10000" eternal="true" overflowToDisk="false"
            timeToIdleSeconds="60" timeToLiveSeconds="3600">
            <cacheEventListenerFactory
                class="net.sf.ehcache.distribution.RMICacheReplicatorFactory"
                properties="replicateAsynchronously=true,
                            replicatePuts=true,
                            replicateUpdates=true,
                            replicateUpdatesViaCopy=true,
                            replicateRemovals=true,
                            asynchronousReplicationIntervalMillis=1000" />
            <bootstrapCacheLoaderFactory
                class="net.sf.ehcache.distribution.RMIBootstrapCacheLoaderFactory"
                    properties="bootstrapAsynchronously=true,
                                maximumChunkSizeBytes=5000000" />
       </cache>

       <!-- The CodeFinderService and CodeResolverService do not support
 serializable
            shared objects. Therefore puts and updates are disabled. It is
 neccessary to
            enable removals if using the CodeAuthoringService.
       -->
       <cache name="com.icw.ehf.commons.codesystem.CodeFinderService"
            maxElementsInMemory="10000" eternal="true" overflowToDisk="false"
            timeToIdleSeconds="60" timeToLiveSeconds="3600">
            <cacheEventListenerFactory
                class="net.sf.ehcache.distribution.RMICacheReplicatorFactory"
                properties="replicateAsynchronously=true,
                            replicatePuts=false,
                            replicateUpdates=false,
                            replicateUpdatesViaCopy=false,
                            replicateRemovals=true,
                            asynchronousReplicationIntervalMillis=1000" />
        </cache>


       <cache name="com.icw.ehf.commons.codesystem.CodeResolverService"
           maxElementsInMemory="10000" eternal="true" overflowToDisk="false"
           timeToIdleSeconds="60" timeToLiveSeconds="3600">
           <cacheEventListenerFactory
               class="net.sf.ehcache.distribution.RMICacheReplicatorFactory"
               properties="replicateAsynchronously=true,
                           replicatePuts=false,
                           replicateUpdates=false,
                           replicateUpdatesViaCopy=false,
                           replicateRemovals=true,
                           asynchronousReplicationIntervalMillis=1000" />
       </cache>
</ehcache>
```

2.2.1.2 Description of the Configuration Settings

Caches are managed by a CacheManager. The CacheManager, when using the RMI distributed cache mechanism listens to other CacheManagers using *Peer Providers* and provides notifications using a *Peer Listener*. The individual caches are configured to use a *Replicator* to provide notifications and replications to other nodes.

Peer Providers: These are the peers the cache is listening to for notifications. In the example, the peers are manually configured to listen to the three CodeService caches residing on the *localhost:40002* node.

```
<cacheManagerPeerProviderFactory
          class="net.sf.ehcache.distribution.RMICacheManagerPeerProviderFactory"
```

```
          properties="peerDiscovery=manual,
          rmiUrls=//localhost:40002/com.icw.ehf.commons.codesystem.
CodeValidatorService|
                  //localhost:40002/com.icw.ehf.commons.codesystem.
CodeFinderService|
                 //localhost:40002/com.icw.ehf.commons.codesystem.
CodeResolverService" />
```

Note: It is also possible to use a multicast peer provider listener. Replace the manually configured `cacheManagerPeerProviderFactory` in the example above with:

```
<cacheManagerPeerProviderFactory
           class="net.sf.ehcache.distribution.RMICacheManagerPeerProviderFactory"
           properties="peerDiscovery=automatic,
                       multicastGroupAddress=224.0.2.5,
                       multicastGroupPort=4444, timeToLive=1"/>
```

An appropriate multicast group address and port, that is also unique in the network, should be used for your cluster. The IANA IPv4 Multicast Addresses registry could be used for finding unreserved/unassigned Multicast addresses.

Peer Listener: This is the settings that other cache nodes can listen to for changes e.g.

```
<cacheManagerPeerListenerFactory
           class="net.sf.ehcache.distribution.RMICacheManagerPeerListenerFactory"
           properties="hostName=localhost, port=40001, socketTimeoutMillis=
2000" />
```

In this example, the peer listener is setup on *localhost:40001*. **Replicators:**Used to configure the cache to replicate data to other caches e.g.

```
<cache name="com.icw.ehf.commons.codesystem.CodeResolverService"
          maxElementsInMemory="10000" eternal="true" overflowToDisk="false"
          timeToIdleSeconds="60" timeToLiveSeconds="3600">
          <cacheEventListenerFactory
              class="net.sf.ehcache.distribution.RMICacheReplicatorFactory"
              properties="replicateAsynchronously=true,
                          replicatePuts=false,
                          replicateUpdates=false,
                          replicateUpdatesViaCopy=false,
                          replicateRemovals=true,
                          asynchronousReplicationIntervalMillis=1000" />
      </cache>
```

Note: `replicateRemovals=true` should be enabled for all distributed caches using RMI. Only `CodeValidatorService` supports the following:

- replicatePuts=true
- replicateUpdates=true
- replicateUpdatesViaCopy=true

Bootstrap Cache Loader: Initialises information from other caches on starup. This is only applicable to the `CodeValidatorService` cache if puts and updates are configured. e.g.

```
<cache name="com.icw.ehf.commons.codesystem.CodeValidatorService"
           maxElementsInMemory="10000" eternal="true" overflowToDisk="false"
           timeToIdleSeconds="60" timeToLiveSeconds="3600">
           <cacheEventListenerFactory
               class="net.sf.ehcache.distribution.RMICacheReplicatorFactory"
               properties="replicateAsynchronously=true,
```

```
                            replicatePuts=true,
                            replicateUpdates=true,
                            replicateUpdatesViaCopy=true,
                            replicateRemovals=true,
                            asynchronousReplicationIntervalMillis=1000" />
            <bootstrapCacheLoaderFactory
                class="net.sf.ehcache.distribution.RMIBootstrapCacheLoaderFactory"
                    properties="bootstrapAsynchronously=true,
                                maximumChunkSizeBytes=5000000" />
      </cache>
```

2.2.2 Distributed Cache Nodes with Replication

Usage Scenario
These settings are for situations where distribution of cache notifications is required and replication of all objects in the caches is desired between nodes.

Notes:

1. In order for this to work, a custom Terracotta setup is required and *Identity* value mode needs to be enabled. Please ensure you have correctly setup Terracotta by following the Terracotta Distributed ehcache documentation available at http://www.terracotta.org/documentation/
2. The Terracotta setup requires the following modules to be installed:
    - `tim-tomcat`
    - `tim-ehcache`

These modules need to be configured in the Terracotta configuration file e.g. `tc-config.xml`.If you are not using Tomcat, use the appropriate TIM module for your applicationserver must be used instead of tim-tomcat.

1. In order for replication of all objects in the CodeSystem service caches to work, the shared classes needs to be added to the tc-config file used by Terracotta.

2.2.2.1 Example Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="ehcache.xsd"
         name="codesystem-cache">

    <defaultCache maxElementsInMemory="10000"
         eternal="false"
         timeToIdleSeconds="120"
         timeToLiveSeconds="120"
         overflowToDisk="false"/>

    <cache name="com.icw.ehf.commons.codesystem.CodeValidatorService"
                maxElementsInMemory="10000"
                eternal="true"
                overflowToDisk="false">
            <terracotta clustered="true" valueMode="identity"/>
    </cache>

    <cache name="com.icw.ehf.commons.codesystem.CodeFinderService"
                maxElementsInMemory="10000"
                eternal="true"
                overflowToDisk="false">
            <terracotta clustered="true" valueMode="identity"/>
```

```
    </cache>

    <cache name="com.icw.ehf.commons.codesystem.CodeResolverService"
               maxElementsInMemory="10000"
               eternal="true"
               overflowToDisk="false">
           <terracotta clustered="true" valueMode="identity"/>
     </cache>
</ehcache>
```

2.2.2.2 Description of the Configuration Settings

The only setting required for Terracotta distributed caches is the following:

```
<terracotta clustered="true" valueMode="identity"/>
```

Important:

Please be aware that *valueMode=Identity* needs to be used to allow replication on `CodeFinderService` and `CodeResolverServer` objects. Identity mode can only be used with a Terracotta custom Distributed Cache installation. For further information, please see the Terracotta documentation