Ehcache Replication Guide

Version 2.10.1

October 2015

EHCACHE

This document applies to Ehcache Version 2.10.1 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.

Copyright © 2010-2015 Software AG, Darmstadt, Germany and/or Software AG USA Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at http://softwareag.com/licenses.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at http://softwareag.com/licenses and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software AG Products / Copyright and Trademark Notices of Software AG Products". These documents are part of the product documentation, located at http://softwareag.com/licenses and/or in the root installation directory of the licensed product(s).

Use, reproduction, transfer, publication or disclosure is prohibited except as specifically provided for in your License Agreement with Software AG.

Document ID: EHC-RG-2101-20151015

Table of Contents

Using Replication..**5**
Supported Types of Replication..6
Minimum Configuration for Replication..6
Adding Replication to an Existing Cache..7

Replicated Caching Using RMI..**9**
Using RMI for Replicated Caching..10
Suitable Element Types..11
Configuring the Peer Provider..11
Configuring the CacheManagerPeerListener..12
Configuring Cache Replicators..13
Configuring Bootstrap from a Cache Peer..14
Complete Example..15
Common Issues with RMI Replication..15

Replicated Caching Using JGroups..**17**
Using JGroups for Replicated Caching..18
Suitable Element Types..18
Peer Discovery..18
Configuration..18
Example Configuration using UDP Multicast..19
Example Configuration using TCP Unicast..19
Protocol Considerations..19
Configuring CacheReplicators..19
Complete Sample Configuration..20
Common Issues with JGroups Replication..21

Replicated Caching Using JMS..**23**
Using JMS for Replicated Caching..24
Ehcache Replication and External Publishers..24
Configuration..25
External JMS Publishers..28
Code Samples..29
Using the JMSCacheLoader..31
Configuring Clients for Message Queue Reliability..34
Tested Message Queues..34
Common Issues using JMS-based Replication..35

1 Using Replication

- Supported Types of Replication 6
- Minimum Configuration for Replication 6
- Adding Replication to an Existing Cache 7

Supported Types of Replication

Ehcache provides three mechanisms for replicating a cache across multiple nodes:

RMI Replicated Caching

Ehcache provides replicated caching using RMI. To set up RMI replicated caching, you need to configure the CacheManager with a PeerProvider and a CacheManagerPeerListener. Then for each cache that will be replicated, you need to add one of the RMI cacheEventListener types to propagate messages. You can also optionally configure a cache to bootstrap from other caches in the cluster.

JGroups Replicated Caching

JGroups can be used as the underlying mechanism for the replication operations in Ehcache. JGroups offers a very flexible protocol stack, reliable unicast, and multicast message transmission. To set up replicated caching using JGroups, you need to configure a PeerProviderFactory. For each cache that will be replicated, you then need to add a cacheEventListenerFactory to propagate messages.

JMS Replicated Caching

JMS can also be used as the underlying mechanism for replication operations in Ehcache. The Ehcache jmsreplication module lets organisations with a message queue investment leverage it for caching. It provides replication between cache nodes using a replication topic, pushing of data directly to cache nodes from external topic publishers, and a JMSCacheLoader, which sends cache load requests to a queue.

Minimum Configuration for Replication

The minimum configuration you need to get replicated caching going is:

```
<cacheManagerPeerProviderFactory
   class="net.sf.ehcache.distribution.RMICacheManagerPeerProviderFactory"
   properties="peerDiscovery=automatic,
               multicastGroupAddress=230.0.0.1,
               multicastGroupPort=4446"/>
<cacheManagerPeerListenerFactory
   class="net.sf.ehcache.distribution.RMICacheManagerPeerListenerFactory"/>
```

and then at least one cache declaration with

```
<cacheEventListenerFactory
   class="net.sf.ehcache.distribution.RMICacheReplicatorFactory"/>
```

in it. An example cache is:

```
<cache name="sampleDistributedCache1"
      maxEntriesLocalHeap="10"
      eternal="false"
      timeToIdleSeconds="100"
      timeToLiveSeconds="100">
```

```
   <cacheEventListenerFactory
      class="net.sf.ehcache.distribution.RMICacheReplicatorFactory"/>
</cache>
```

Each peer server can have the same configuration.

Adding Replication to an Existing Cache

The cache event listening works but it does not get plumbed into the peering mechanism. The current API does not have a CacheManager event for cache configuration change. You can however make it work by calling the notifyCacheAdded event.

```
getCache().getCacheManager().
   getCacheManagerEventListenerRegistry().notifyCacheAdded("cacheName");
```

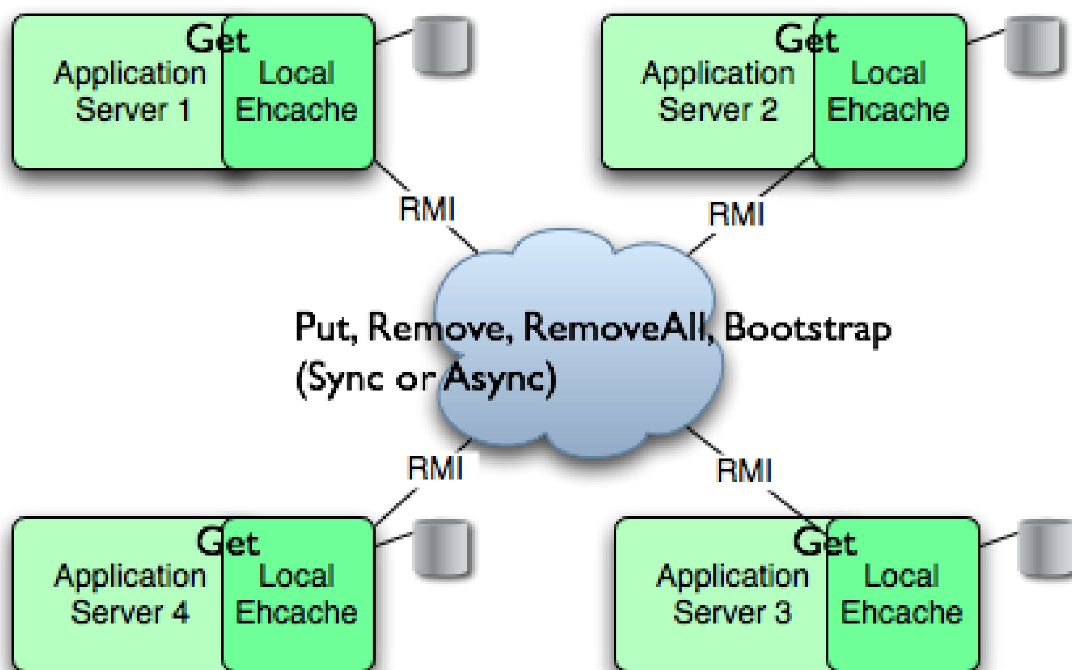
2 Replicated Caching Using RMI

- Using RMI for Replicated Caching 10
- Suitable Element Types 11
- Configuring the Peer Provider 11
- Configuring the CacheManagerPeerListener 12
- Configuring Cache Replicators 13
- Configuring Bootstrap from a Cache Peer 14
- Complete Example 15
- Common Issues with RMI Replication 15

Using RMI for Replicated Caching

Replicated caching using RMI is desirable because:

- RMI is the default remoting mechanism in Java
- it allows tuning of TCP socket options
- Element keys and values for disk storage must already be Serializable, therefore directly transmittable over RMI without the need for conversion to a third format such as XML
- it can be configured to pass through firewalls



While RMI is a point-to-point protocol, which can generate a lot of network traffic, Ehcache manages this through batching of communications for the asynchronous replicator.

To set up replicated caching with RMI you need to configure the CacheManager with:

- a PeerProvider
- a CacheManagerPeerListener

For each cache that will be replicated, you then need to add one of the RMI cacheEventListener types to propagate messages. You can also optionally configure a cache to bootstrap from other caches in the cluster.

Suitable Element Types

Only Serializable Elements are suitable for replication.

Some operations, such as remove, work off Element keys rather than the full Element itself. In this case the operation will be replicated provided the key is Serializable, even if the Element is not.

Configuring the Peer Provider

Peer Discovery

Ehcache has the notion of a group of caches acting as a replicated cache. Each of the caches is a peer to the others. There is no master cache. How do you know about the other caches that are in your cluster? This problem can be given the name Peer Discovery. Ehcache provides two mechanisms for peer discovery: manual and automatic.

To use one of the built-in peer discovery mechanisms, specify the class attribute of cacheManagerPeerProviderFactory as net.sf.ehcache.distribution.RMICacheManagerPeerProviderFactory in the ehcache.xml configuration file.

Automatic Peer Discovery

Automatic discovery uses TCP multicast to establish and maintain a multicast group. It features minimal configuration and automatic addition to and deletion of members from the group. No a priori knowledge of the servers in the cluster is required. This is recommended as the default option. Peers send heartbeats to the group once per second. If a peer has not been heard of for 5 seconds it is dropped from the group. If a new peer starts sending heartbeats it is admitted to the group.

Any cache within the configuration set up as replicated will be made available for discovery by other peers.

To set automatic peer discovery, specify the properties attribute of cacheManagerPeerProviderFactory as follows:

```
peerDiscovery=automatic
multicastGroupAddress=multicast address | multicast host name
multicastGroupPort=port
timeToLive=0-255 (See below in common problems before setting this)
hostName=the hostname or IP of the interface to be used for
         sending and receiving multicast packets
         (relevant to mulithomed hosts only)
```

Example

Suppose you have two servers in a cluster, server1 and server2. You wish to distribute sampleCache11 and sampleCache12. The configuration required for each server is identical, so the configuration for both server1 and server2 is the following:

```
<cacheManagerPeerProviderFactory
class="net.sf.ehcache.distribution.RMICacheManagerPeerProviderFactory"
properties="peerDiscovery=automatic, multicastGroupAddress=230.0.0.1,
multicastGroupPort=4446, timeToLive=32"/>
```

Manual Peer Discovery

Manual peer configuration requires the IP address and port of each listener to be known. Peers cannot be added or removed at runtime. Manual peer discovery is recommended where there are technical difficulties using multicast, such as a router between servers in a cluster that does not propagate multicast datagrams. You can also use it to set up one way replications of data, by having server2 know about server1 but not vice versa.

To set manual peer discovery, specify the properties attribute of cacheManagerPeerProviderFactory as follows:

```
peerDiscovery=manual
rmiUrls=//server:port/cacheName, ...
```

The rmiUrls is a list of the cache peers of the server being configured. Do not include the server being configured in the list.

Example

Suppose you have two servers in a cluster, server1 and server2. You wish to distribute sampleCache11 and sampleCache12. The following is the configuration required for server1:

```
<cacheManagerPeerProviderFactory
class="net.sf.ehcache.distribution.RMICacheManagerPeerProviderFactory"
properties="peerDiscovery=manual,
rmiUrls=//server2:40001/sampleCache11|//server2:40001/sampleCache12"/>
```

The following is the configuration required for server2:

```
<cacheManagerPeerProviderFactory
class="net.sf.ehcache.distribution.RMICacheManagerPeerProviderFactory"
properties="peerDiscovery=manual,
rmiUrls=//server1:40001/sampleCache11|//server1:40001/sampleCache12"/>
```

Configuring the CacheManagerPeerListener

A CacheManagerPeerListener listens for messages from peers to the current CacheManager.

You configure the CacheManagerPeerListener by specifiying a CacheManagerPeerListenerFactory which is used to create the CacheManagerPeerListener using the plugin mechanism.

The attributes of cacheManagerPeerListenerFactory are:

- class - a fully qualified factory class name
- properties - comma separated properties having meaning only to the factory.

Ehcache comes with a built-in RMI-based distribution system. The listener component is RMICacheManagerPeerListener which is configured using RMICacheManagerPeerListenerFactory. It is configured as per the following example:

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

Valid properties are:

- hostName (optional) - the hostName of the host the listener is running on. Specify where the host is multihomed and you want to control the interface over which cluster messages are received. The hostname is checked for reachability during CacheManager initialisation. If the hostName is unreachable, the CacheManager will refuse to start and an CacheException will be thrown indicating connection was refused. If unspecified, the hostname will use InetAddress.getLocalHost().getHostAddress(), which corresponds to the default host network interface. Warning: Explicitly setting this to localhost refers to the local loopback of 127.0.0.1, which is not network visible and will cause no replications to be received from remote hosts. You should only use this setting when multiple CacheManagers are on the same machine.
- port (mandatory) - the port the listener listens on.
- socketTimeoutMillis (optional) - the number of seconds client sockets will wait when sending messages to this listener until they give up. By default this is 2000ms.

Configuring Cache Replicators

Each cache that will be replicated needs to set a cache event listener which then replicates messages to the other CacheManager peers. This is done by adding a cacheEventListenerFactory element to each cache's configuration.

```
<!-- Sample cache named sampleCache2. -->
<cache name="sampleCache2"
  maxEntriesLocalHeap="10"
  eternal="false"
  timeToIdleSeconds="100"
  timeToLiveSeconds="100"
  overflowToDisk="false">
<cacheEventListenerFactory
class="net.sf.ehcache.distribution.RMICacheReplicatorFactory"
properties="replicateAsynchronously=true, replicatePuts=true, replicateUpdates=true,
replicateUpdatesViaCopy=false, replicateRemovals=true "/>
</cache>
```

class - use net.sf.ehcache.distribution.RMICacheReplicatorFactory

The factory recognises the following properties:

- replicatePuts=true | false - whether new elements placed in a cache are replicated to others. Defaults to true.
- replicateUpdates=true | false - whether new elements which override an element already existing with the same key are replicated. Defaults to true.

- replicateRemovals=true - whether element removals are replicated. Defaults to true.
- replicateAsynchronously=true | false - whether replications are asyncrhonous (true) or synchronous (false). Defaults to true.
- replicateUpdatesViaCopy=true | false - whether the new elements are copied to other caches (true), or whether a remove message is sent. Defaults to true.

To reduce typing if you want default behaviour, which is replicate everything in asynchronous mode, you can leave off the RMICacheReplicatorFactory properties as per the following example:

```
<!-- Sample cache named sampleCache4. All missing RMICacheReplicatorFactory properties
    default to true -->
<cache name="sampleCache4"
      maxEntriesLocalHeap="10"
      eternal="true"
      overflowToDisk="false"
      memoryStoreEvictionPolicy="LFU">
   <cacheEventListenerFactory
       class="net.sf.ehcache.distribution.RMICacheReplicatorFactory"/>
</cache>
```

Configuring Bootstrap from a Cache Peer

When a peer comes up, it will be incoherent with other caches. When the bootstrap completes it will be partially coherent. Bootstrap gets the list of keys from a random peer, and then loads those in batches from random peers. If bootstrap fails then the Cache will not start. However if a cache replication operation occurs which is then overwritten by bootstrap there is a chance that the cache could be inconsistent.

Here are some scenarios:

Delete overwritten by bootstrap put:

1. Cache A keys with values: 1, 2, 3, 4, 5
2. Cache B starts bootstrap
3. Cache A removes key 2
4. Cache B removes key 2 and then bootstrap puts it back

Put overwritten by bootstrap put:

1. Cache A keys with values: 1, 2, 3, 4, 5
2. Cache B starts bootstrap
3. Cache A updates the value of key 2
4. Cache B updates the value of key 2 and then bootstrap overwrites it with the old value

The solution is for bootstrap to get a list of keys and write them all before committing transactions.

This could cause synchronous transaction replicates to back up. To solve this problem, commits will be accepted, but not written to the cache until after bootstrap. Coherency is maintained because the cache is not available until bootstrap has completed and the transactions have been completed.

Complete Example

Ehcache's own integration tests provide complete examples of RMI-based replication.

The best example is the integration test for cache replication. You can see it online in the source code example RMICacheReplicatorIT.java at this location: https://fisheye.terracotta.org/browse/Ehcache/branches/ehcache-2.10.1/ehcache-core/src/test/java/net/sf/ehcache/distribution/

The test uses five ehcache.xml files representing five CacheManagers set up to replicate using RMI.

Common Issues with RMI Replication

Tomcat on Windows

Any RMI listener will fail to start on Tomcat, if the installation path has spaces in it. Because the default on Windows is to install Tomcat in "Program Files", this issue will occur by default. The workaround is to remove the spaces in your Tomcat installation path.

Multicast Blocking

The automatic peer discovery process relies on multicast. Multicast can be blocked by routers. Virtualisation technologies like Xen and VMWare may be blocking multicast. If so enable it. You may also need to turn it on in the configuration for your network interface card. An easy way to tell if your multicast is getting through is to use the Ehcache remote debugger and watch for the heartbeat packets to arrive.

Multicast Not Propagating Far Enough or Propagating Too Far

You can control how far the multicast packets propagate by setting the badly misnamed time to live. Using the multicast IP protocol, the timeToLive value indicates the scope or range in which a packet may be forwarded.

By convention:

- 0 is restricted to the same host
- 1 is restricted to the same subnet
- 32 is restricted to the same site
- 64 is restricted to the same region

- 128 is restricted to the same continent
- 255 is unrestricted

The default value in Java is 1, which propagates to the same subnet. Change the timeToLive property to restrict or expand propagation.

3 Replicated Caching Using JGroups

- Using JGroups for Replicated Caching 18
- Suitable Element Types 18
- Peer Discovery 18
- Configuration 18
- Example Configuration using UDP Multicast 19
- Example Configuration using TCP Unicast 19
- Protocol Considerations 19
- Configuring CacheReplicators 19
- Complete Sample Configuration 20
- Common Issues with JGroups Replication 21

Using JGroups for Replicated Caching

JGroups can be used as the underlying mechanism for the replication operations in ehcache. JGroups offers a very flexible protocol stack, reliable unicast and multicast message transmission.

On the down side JGroups can be complex to configure and some protocol stacks have dependencies on others.

To set up replicated caching using JGroups you need to configure a PeerProviderFactory of type JGroupsCacheManagerPeerProviderFactory which is done globally for a CacheManager

For each cache that will be replicated, you then need to add a cacheEventListenerFactory of type JGroupsCacheReplicatorFactory to propagate messages.

Suitable Element Types

Only Serializable Elements are suitable for replication.

Some operations, such as remove, work off Element keys rather than the full Element itself. In this case the operation will be replicated provided the key is Serializable, even if the Element is not.

Peer Discovery

If you use the UDP multicast stack there is no additional configuration. If you use a TCP stack you will need to specify the initial hosts in the cluster.

Configuration

There are two things to configure:

- The JGroupsCacheManagerPeerProviderFactory which is done once per CacheManager and therefore once per ehcache.xml file.
- The JGroupsCacheReplicatorFactory which is added to each cache's configuration.

The main configuration happens in the JGroupsCacheManagerPeerProviderFactory connect sub-property.

A connect property is passed directly to the JGroups channel and therefore all the protocol stacks and options available in JGroups can be set.

If you use the UDP multicast stack there is no additional configuration. If you use a TCP stack you will need to specify the initial hosts in the cluster.

Example Configuration using UDP Multicast

Suppose you have two servers in a cluster. You want to replicate sampleCache11 and sampleCache12 and you want to use UDP multicast as the underlying mechanism. The configurations for server1 and server2 are identical and will look like this:

```
<cacheManagerPeerProviderFactory
class="net.sf.ehcache.distribution.jgroups.JGroupsCacheManagerPeerProviderFactory"
properties="connect=UDP(mcast_addr=231.12.21.132;mcast_port=45566;):PING:
MERGE2:FD_SOCK:VERIFY_SUSPECT:pbcast.NAKACK:UNICAST:pbcast.STABLE:FRAG:pbcast.GMS"
propertySeparator="::"
/>
```

Example Configuration using TCP Unicast

The TCP protocol requires the IP address of all servers to be known. They are configured through the TCPPING protocol of JGroups.

Suppose you have two servers: host1 and host2. The configuration is:

```
<cacheManagerPeerProviderFactory
class="net.sf.ehcache.distribution.jgroups.JGroupsCacheManagerPeerProviderFactory"
properties="connect=TCP(start_port=7800):
   TCPPING(initial_hosts=host1[7800],host2[7800];port_range=10;timeout=3000;
   num_initial_members=3;up_thread=true;down_thread=true):
   VERIFY_SUSPECT(timeout=1500;down_thread=false;up_thread=false):
   pbcast.NAKACK(down_thread=true;up_thread=true;gc_lag=100;retransmit_timeout=3000):
   pbcast.GMS(join_timeout=5000;join_retry_timeout=2000;shun=false;
   print_local_addr=false;down_thread=true;up_thread=true)"
propertySeparator="::" />
```

Protocol Considerations

You should read the JGroups documentation to configure the protocols correctly.

See http://www.jgroups.org/javagroupsnew/docs/manual/html_single/index.html.

If using UDP you should at least configure PING, FD_SOCK (Failure detection), VERIFY_SUSPECT, pbcast.NAKACK (Message reliability), pbcast.STABLE (message garbage collection).

Configuring CacheReplicators

Each cache that will be replicated needs to set a cache event listener which then replicates messages to the other CacheManager peers. This is done by adding a cacheEventListenerFactory element to each cache's configuration. The properties are identical to the one used for RMI replication. The listener factory must be of type JGroupsCacheReplicatorFactory.

```
<!-- Sample cache named sampleCache2. -->
<cache name="sampleCache2"
  maxEntriesLocalHeap="10"
  eternal="false"
  timeToIdleSeconds="100"
  timeToLiveSeconds="100"
  overflowToDisk="false">
  <cacheEventListenerFactory
  class="net.sf.ehcache.distribution.jgroups.JGroupsCacheReplicatorFactory"
  properties="replicateAsynchronously=true, replicatePuts=true,
  replicateUpdates=true, replicateUpdatesViaCopy=false, replicateRemovals=true" />
</cache>
```

The configuration options are explained below:

class - use net.sf.ehcache.distribution.jgroups.JGroupsCacheReplicatorFactory

The factory recognizes the following properties:

- replicatePuts=true | false - whether new elements placed in a cache are replicated to others. Defaults to true.
- replicateUpdates=true | false - whether new elements which override an element already existing with the same key are replicated. Defaults to true.
- replicateRemovals=true - whether element removals are replicated. Defaults to true.
- replicateAsynchronously=true | false - whether replications are asyncrhonous (true) or synchronous (false). Defaults to true.
- replicateUpdatesViaCopy=true | false - whether the new elements are copied to other caches (true), or whether a remove message is sent. Defaults to true.
- asynchronousReplicationIntervalMillis default 1000ms Time between updates when replication is asynchronous.

Complete Sample Configuration

A typical complete configuration for one replicated cache configured for UDP will look like:

```
<Ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="../../../main/config/ehcache.xsd">
  <diskStore path="java.io.tmpdir/one"/>
  <cacheManagerPeerProviderFactory class="net.sf.ehcache.distribution.jgroups
    .JGroupsCacheManagerPeerProviderFactory"
    properties="connect=UDP(mcast_addr=231.12.21.132;mcast_port=45566;ip_ttl=32;
    mcast_send_buf_size=150000;mcast_recv_buf_size=80000):
    PING(timeout=2000;num_initial_members=6):
    MERGE2(min_interval=5000;max_interval=10000):
    FD_SOCK:VERIFY_SUSPECT(timeout=1500):
    pbcast.NAKACK(gc_lag=10;retransmit_timeout=3000):
    UNICAST(timeout=5000):
    pbcast.STABLE(desired_avg_gossip=20000):
    FRAG:
    pbcast.GMS(join_timeout=5000;join_retry_timeout=2000;
    shun=false;print_local_addr=true)"
    propertySeparator="::"
  />
```

```
  <cache name="sampleCacheAsync"
    maxEntriesLocalHeap="1000"
    eternal="false"
    timeToIdleSeconds="1000"
    timeToLiveSeconds="1000"
    overflowToDisk="false">
  <cacheEventListenerFactory
    class="net.sf.ehcache.distribution.jgroups.JGroupsCacheReplicatorFactory"
    properties="replicateAsynchronously=true, replicatePuts=true,
      replicateUpdates=true, replicateUpdatesViaCopy=false,
      replicateRemovals=true" />
  </cache>
</ehcache>
```

Common Issues with JGroups Replication

If replication using JGroups doesn't work the way you have it configured, try this configuration which has been extensively tested:

```
<cacheManagerPeerProviderFactory
    class="net.sf.ehcache.distribution.jgroups.JGroupsCacheManagerPeerProviderFactory"/>
<cache name="sampleCacheAsync"
  maxEntriesLocalHeap="1000"
  eternal="false"
  timeToIdleSeconds="1000"
  timeToLiveSeconds="1000"
  overflowToDisk="false">
  <cacheEventListenerFactory
    class="net.sf.ehcache.distribution.jgroups.JGroupsCacheReplicatorFactory"
    properties="replicateAsynchronously=true, replicatePuts=true,
      replicateUpdates=true, replicateUpdatesViaCopy=false,
      replicateRemovals=true" />
</cache>
```

If this fails to replicate, see the example programs in the JGroups documentation at http://www.jgroups.org/manual/html/ch02.html.

Once you have figured out the connection string that works in your network for JGroups, you can directly paste it in the connect property of JGroupsCacheManagerPeerProviderFactory.

4 Replicated Caching Using JMS

- Using JMS for Replicated Caching 24
- Ehcache Replication and External Publishers 24
- Using the JMSCacheLoader 31
- Configuring Clients for Message Queue Reliability 34
- Tested Message Queues 34
- Common Issues using JMS-based Replication 35

Using JMS for Replicated Caching

As of version 1.6, JMS can be used as the underlying mechanism for the replicated operations in Ehcache with the jmsreplication module.

JMS (Java Message Service) is a mechanism for interacting with message queues. Message queues themselves are a very mature piece of infrastructure used in many enterprise software contexts. Because they are a required part of the Java EE standard, the large enterprise vendors all provide their own implementations. There are also several open source choices including Open MQ and Active MQ. Ehcache is integration tested against both of these.

The Ehcache jmsreplication module lets organizations with a message queue investment leverage it for caching.

It provides:

- Replication between cache nodes using a replication topic, in accordance with Ehcache's standard replication mechanism
- Pushing of data directly to cache nodes from external topic publishers, in any language. This is done by sending the data to the replication topic, where it is automatically picked up by the cache subscribers.
- A JMSCacheLoader, which sends cache load requests to a queue. Either an Ehcache cluster node, or an external queue receiver can respond.

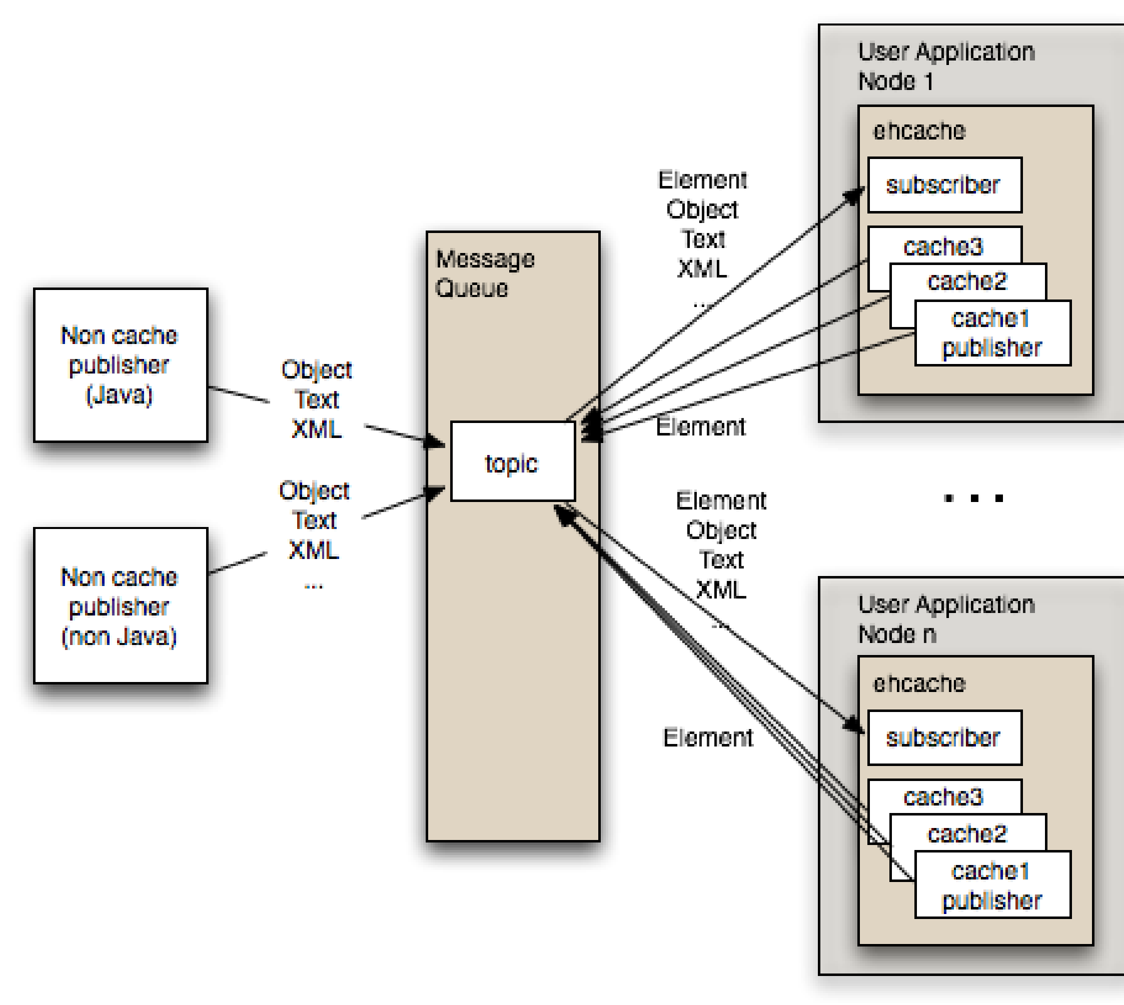
Ehcache Replication and External Publishers

Ehcache replicates using JMS as follows:

- Each cache node subscribes to a predefined topic, configured as the <topicBindingName> in ehcache.xml.
- Each replicated cache publishes cache Elements to that topic. Replication is configured per cache.

To set up replicated caching based on JMS you need to configure a JMSCacheManagerPeerProviderFactory which is done globally for a CacheManager.

For each cache that wishing to replicate, you add a JGroupsCacheReplicatorFactory element to the cache element.



Configuration

Message Queue Configuration

Each cluster needs to use a fixed topic name for replication. Set up a topic using the tools in your message queue. Out of the box, both ActiveMQ and Open MQ support auto creation of destinations, so this step may be optional.

Ehcache Configuration

Configuration is done in the ehcache.xml.

There are two things to configure:

- The JMSCacheManagerPeerProviderFactory which is done once per CacheManager and therefore once per ehcache.xml file.
- The JMSCacheReplicatorFactory which is added to each cache's configuration if you want that cache replicated.

The main configuration happens in the JGroupsCacheManagerPeerProviderFactory connect sub-property. A connect property is passed directly to the JGroups channel and therefore all the protocol stacks and options available in JGroups can be set.

Configuring the JMSCacheManagerPeerProviderFactory

Following is the configuration instructions as it appears in the sample ehcache.xml shipped with ehcache:

```
{Configuring JMS replication}.
===========================
The JMS PeerProviderFactory uses JNDI to maintain message queue independence.
Refer to the manual for full configuration examples using ActiveMQ and Open Message Queue.
Valid properties are:
* initialContextFactoryName (mandatory) - the name of the factory used to create
* the message queue initial context.
* providerURL (mandatory) - the JNDI configuration information for the service
* provider to use.
* topicConnectionFactoryBindingName (mandatory) - the JNDI binding name for the
* TopicConnectionFactory
* topicBindingName (mandatory) - the JNDI binding name for the topic name
* securityPrincipalName - the JNDI java.naming.security.principal
* securityCredentials - the JNDI java.naming.security.credentials
* urlPkgPrefixes - the JNDI java.naming.factory.url.pkgs
* userName - the user name to use when creating the TopicConnection to the
* Message Queue
* password - the password to use when creating the TopicConnection to the Message
* Queue
* acknowledgementMode - the JMS Acknowledgement mode for both publisher and
* subscriber.
    The available choices are
      AUTO_ACKNOWLEDGE, DUPS_OK_ACKNOWLEDGE and SESSION_TRANSACTED.
    The default is AUTO_ACKNOWLEDGE.
* listenToTopic - true or false. If false, this cache will send to the JMS topic
* but will not listen for updates.
* Default is true.
```

Example - JMSCacheManagerPeerProviderFactory for Active MQ

This configuration works with Active MQ out of the box.

```
<cacheManagerPeerProviderFactory
       class="net.sf.ehcache.distribution.jms.JMSCacheManagerPeerProviderFactory"
       properties="initialContextFactoryName=ExampleActiveMQInitialContextFactory,
           providerURL=tcp://localhost:61616,
           topicConnectionFactoryBindingName=topicConnectionFactory,
           topicBindingName=ehcache"
       propertySeparator=","
       />
```

You need to provide your own ActiveMQInitialContextFactory for the initialContextFactoryName. An example which should work for most purposes is:

```
public class ExampleActiveMQInitialContextFactory
  extends ActiveMQInitialContextFactory {
    /**
     * {@inheritDoc}
     */
    @Override
    @SuppressWarnings("unchecked")
    public Context getInitialContext(Hashtable environment)
```

```
      throws NamingException
    {
      Map<String, Object> data = new ConcurrentHashMap<String, Object>();
      String factoryBindingName =
        (String)environment.get(JMSCacheManagerPeerProviderFactory
          .TOPIC_CONNECTION_FACTORY_BINDING_NAME);
      try {
        data.put(factoryBindingName, createConnectionFactory(environment));
      } catch (URISyntaxException e) {
        throw new NamingException("Error initialisating ConnectionFactory"
                                  + " with message "
                      + e.getMessage());
      }
      String topicBindingName =
        (String)environment.get(JMSCacheManagerPeerProviderFactory
          .TOPIC_BINDING_NAME);
      data.put(topicBindingName, createTopic(topicBindingName));
      return createContext(environment, data);
    }
}
```

Example - JMSCacheManagerPeerProviderFactory for Open MQ

This configuration works with an out of the box Open MQ.

```
<cacheManagerPeerProviderFactory
  class="net.sf.ehcache.distribution.jms.JMSCacheManagerPeerProviderFactory"
  properties="initialContextFactoryName=
       com.sun.jndi.fscontext.RefFSContextFactory,
           providerURL=file:///tmp,
           topicConnectionFactoryBindingName=MyConnectionFactory,
           topicBindingName=ehcache"
       propertySeparator=","
       />
```

To set up the Open MQ file system initial context to work with this example use the following imqobjmgr commands to create the requires objects in the context.

```
imqobjmgr add -t tf -l 'MyConnectionFactory' -j java.naming.provider.url \
=file:///tmp -j java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory -f
imqobjmgr add -t t -l 'ehcache' -o 'imqDestinationName=EhcacheTopicDest'
-j java.naming.provider.url\
=file:///tmp -j java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory -f
```

Configuring the JMSCacheReplicatorFactory

This is the same as configuring any of the cache replicators. The class should be net.sf.ehcache.distribution.jms.JMSCacheReplicatorFactory.

See the following example:

```
<cache name="sampleCacheAsync"
 maxEntriesLocalHeap="1000"
 eternal="false"
 timeToIdleSeconds="1000"
 timeToLiveSeconds="1000"
 overflowToDisk="false">
  <cacheEventListenerFactory
     class="net.sf.ehcache.distribution.jms.JMSCacheReplicatorFactory"
     properties="replicateAsynchronously=true,
                 replicatePuts=true,
                 replicateUpdates=true,
                 replicateUpdatesViaCopy=true,
```

```
                  replicateRemovals=true,
                  asynchronousReplicationIntervalMillis=1000"
      propertySeparator=","/>
</cache>
```

External JMS Publishers

Anything that can publish to a message queue can also add cache entries to ehcache. These are called non-cache publishers.

Required Message Properties

Publishers need to set up to four String properties on each message: cacheName, action, mimeType and key.

Property	Description
cacheName	A JMS message property which contains the name of the cache to operate on. If no cacheName is set the message will be <ignored>. A warning log message will indicate that the message has been ignored.
action	A JMS message property which contains the action to perform on the cache. Available actions are strings labeled PUT, REMOVE and REMOVE_ALL. If not set no action is performed. A warning log message will indicate that the message has been ignored.
mimeType	A JMS message property which contains the mimeType of the message. Applies to the PUT action. If not set the message is interpreted as follows: ■ ObjectMessage - If it is an net.sf.ehcache.Element, then it is treated as such and stored in the cache. For other objects, a new Element is created using the object in the ObjectMessage as the value and the key property as a key. Because objects are already typed, the mimeType is ignored ■ TextMessage - Stored in the cache as value of MimeTypeByteArray. The mimeType should be specified. If not specified it is stored as type text/plain.

Property	Description
	■ BytesMessage - Stored in the cache as value of MimeTypeByteArray. The mimeType should be specified. If not specified it is stored as type application/octet-stream. Other message types are not supported. To send XML use a TextMessage or BytesMessage and set the mimeType to application/xml. It will be stored in the cache as a value of MimeTypeByteArray. The REMOVE and REMOVE_ALL actions do not require a mimeType property.
key	The key in the cache on which to operate on. The key is of type String. The REMOVE_ALL action does not require a key property. If an ObjectMessage of type net.sf.ehcache.Element is sent, the key is contained in the element. Any key set as a property is ignored. If the key is required but not provided, a warning log message will indicate that the message has been ignored.

Code Samples

These samples use Open MQ as the message queue and use it with out of the box defaults. They are heavily based on Ehcache's own JMS integration tests. See the test source for more details.

Messages should be sent to the topic that Ehcache is listening on. In these samples it is EhcacheTopicDest.

All samples get a Topic Connection using the following method:

```
private TopicConnection getMQConnection() throws JMSException {
  com.sun.messaging.ConnectionFactory factory =
    new com.sun.messaging.ConnectionFactory();
  factory.setProperty(ConnectionConfiguration.imqAddressList, "localhost:7676");
  factory.setProperty(ConnectionConfiguration.imqReconnectEnabled, "true");
  TopicConnection myConnection = factory.createTopicConnection();
  return myConnection;
}
```

PUT a Java Object into an Ehcache Cluster

```
String payload = "this is an object";
TopicConnection connection = getMQConnection();
connection.start();
TopicSession publisherSession =
  connection.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
ObjectMessage message = publisherSession.createObjectMessage(payload);
message.setStringProperty(ACTION_PROPERTY, "PUT");
message.setStringProperty(CACHE_NAME_PROPERTY, "sampleCacheAsync");
//don't set. Should work.
//message.setStringProperty(MIME_TYPE_PROPERTY, null);
//should work. Key should be ignored when sending an element.
message.setStringProperty(KEY_PROPERTY, "1234");
Topic topic = publisherSession.createTopic("EhcacheTopicDest");
TopicPublisher publisher = publisherSession.createPublisher(topic);
publisher.send(message);
connection.stop();
```

Ehcache will create an Element in cache "sampleCacheAsync" with key "1234" and a Java class String value of "this is an object".

PUT XML into an Ehcache Cluster

```
TopicConnection connection = getMQConnection();
connection.start();
TopicSession publisherSession = connection.createTopicSession(false,
  Session.AUTO_ACKNOWLEDGE);
String value = "<?xml version=\"1.0\"?>\n" +
   "<oldjoke>\n" +
   "<burns>Say <quote>goodnight</quote>,\n" +
   "Gracie.</burns>\n" +
   "<allen><quote>Goodnight, \n" +
   "Gracie.</quote></allen>\n" +
   "<applause/>\n" +
   "</oldjoke>";
TextMessage message = publisherSession.createTextMessage(value);
message.setStringProperty(ACTION_PROPERTY, "PUT");
message.setStringProperty(CACHE_NAME_PROPERTY, "sampleCacheAsync");
message.setStringProperty(MIME_TYPE_PROPERTY, "application/xml");
message.setStringProperty(KEY_PROPERTY, "1234");
Topic topic = publisherSession.createTopic("EhcacheTopicDest");
TopicPublisher publisher = publisherSession.createPublisher(topic);
publisher.send(message);
connection.stop();
```

Ehcache will create an Element in cache "sampleCacheAsync" with key "1234" and a value of type MimeTypeByteArray.

On a get from the cache the MimeTypeByteArray will be returned. It is an Ehcache value object from which a mimeType and byte[] can be retrieved. The mimeType will be "application/xml". The byte[] will contain the XML String encoded in bytes, using the platform's default charset.

PUT arbitrary bytes into an Ehcache Cluster

```
byte[] bytes = new byte[]{0x34, (byte) 0xe3, (byte) 0x88};
TopicConnection connection = getMQConnection();
connection.start();
TopicSession publisherSession = connection.createTopicSession(false,
```

```
  Session.AUTO_ACKNOWLEDGE);
BytesMessage message = publisherSession.createBytesMessage();
message.writeBytes(bytes);
message.setStringProperty(ACTION_PROPERTY, "PUT");
message.setStringProperty(CACHE_NAME_PROPERTY, "sampleCacheAsync");
message.setStringProperty(MIME_TYPE_PROPERTY, "application/octet-stream");
message.setStringProperty(KEY_PROPERTY, "1234");
Topic topic = publisherSession.createTopic("EhcacheTopicDest");
TopicPublisher publisher = publisherSession.createPublisher(topic);
publisher.send(message);
```

Ehcache will create an Element in cache "sampleCacheAsync" with key "1234" in and a value of type MimeTypeByteArray.

On a get from the cache the MimeTypeByteArray will be returned. It is an Ehcache value object from which a mimeType and byte[] can be retrieved. The mimeType will be "application/octet-stream". The byte[] will contain the original bytes.

REMOVE

```
TopicConnection connection = getMQConnection();
connection.start();
TopicSession publisherSession = connection.createTopicSession(false,
  Session.AUTO_ACKNOWLEDGE);
ObjectMessage message = publisherSession.createObjectMessage();
message.setStringProperty(ACTION_PROPERTY, "REMOVE");
message.setStringProperty(CACHE_NAME_PROPERTY, "sampleCacheAsync");
message.setStringProperty(KEY_PROPERTY, "1234");
Topic topic = publisherSession.createTopic("EhcacheTopicDest");
TopicPublisher publisher = publisherSession.createPublisher(topic);
publisher.send(message);
```

Ehcache will remove the Element with key "1234" from cache "sampleCacheAsync" from the cluster.

REMOVE ALL

```
TopicConnection connection = getMQConnection();
connection.start();
TopicSession publisherSession = connection.createTopicSession(false,
Session.AUTO_ACKNOWLEDGE);
ObjectMessage message = publisherSession.createObjectMessage();
message.setStringProperty(ACTION_PROPERTY, "REMOVE_ALL");
message.setStringProperty(CACHE_NAME_PROPERTY, "sampleCacheAsync");
Topic topic = publisherSession.createTopic("EhcacheTopicDest");
TopicPublisher publisher = publisherSession.createPublisher(topic);
publisher.send(message);
connection.stop();
```

Ehcache will remove all Elements from cache "sampleCacheAsync" in the cluster.

Using the JMSCacheLoader

The JMSCacheLoader is a CacheLoader which loads objects into the cache by sending requests to a JMS Queue.

The loader places an ObjectMessage of type JMSEventMessage on the getQueue with an Action of type GET.

It is configured with the following String properties, loaderArgument.

The defaultLoaderArgument, or the loaderArgument if specified on the load request. To work with the JMSCacheManagerPeerProvider this should be the name of the cache to load from. For custom responders, it can be anything which has meaning to the responder.

A queue responder will respond to the request. You can either create your own or use the one built-into the JMSCacheManagerPeerProviderFactory, which attempts to load the queue from its cache.

The JMSCacheLoader uses JNDI to maintain message queue independence. Refer to the manual for full configuration examples using ActiveMQ and Open Message Queue.

It is configured as per the following example:

```
<cacheLoaderFactory class="net.sf.ehcache.distribution.jms.JMSCacheLoaderFactory"
  properties="initialContextFactoryName=com.sun.jndi.fscontext.RefFSContextFactory,
  providerURL=file:///tmp,
  replicationTopicConnectionFactoryBindingName=MyConnectionFactory,
  replicationTopicBindingName=ehcache,
  getQueueConnectionFactoryBindingName=queueConnectionFactory,
  getQueueBindingName=ehcacheGetQueue,
  timeoutMillis=20000
  defaultLoaderArgument=/>
```

Valid properties are:

- initialContextFactoryName (mandatory) - the name of the factory used to create the message queue initial context.
- providerURL (mandatory) - the JNDI configuration information for the service provider to use.
- getQueueConnectionFactoryBindingName (mandatory) - the JNDI binding name for the QueueConnectionFactory
- getQueueBindingName (mandatory) - the JNDI binding name for the queue name used to do make requests.
- defaultLoaderArgument - (optional) - an application specific argument. If not supplied as a cache.load() parameter this default value will be used. The argument is passed in the JMS request as a StringProperty called loaderArgument.
- timeoutMillis - time in milliseconds to wait for a reply.
- securityPrincipalName - the JNDI java.naming.security.principal
- securityCredentials - the JNDI java.naming.security.credentials
- urlPkgPrefixes - the JNDI java.naming.factory.url.pkgs
- userName - the user name to use when creating the TopicConnection to the Message Queue
- password - the password to use when creating the TopicConnection to the Message Queue

- acknowledgementMode - the JMS Acknowledgement mode for both publisher and subscriber. The available choices are AUTO_ACKNOWLEDGE, DUPS_OK_ACKNOWLEDGE and SESSION_TRANSACTED. The default is AUTO_ACKNOWLEDGE.

Example Configuration Using Active MQ

```
<cache name="sampleCacheNorep"
  maxEntriesLocalHeap="1000"
  eternal="false"
  timeToIdleSeconds="1000"
  timeToLiveSeconds="1000"
  overflowToDisk="false">
  <cacheEventListenerFactory
   class="net.sf.ehcache.distribution.jms.JMSCacheReplicatorFactory"
   properties="replicateAsynchronously=false, replicatePuts=false,
   replicateUpdates=false, replicateUpdatesViaCopy=false,
   replicateRemovals=false, loaderArgument=sampleCacheNorep"
   propertySeparator=","/>
<cacheLoaderFactory
  class="net.sf.ehcache.distribution.jms.JMSCacheLoaderFactory"
  properties="initialContextFactoryName=net.sf.ehcache.distribution.jms.
       TestActiveMQInitialContextFactory,
       providerURL=tcp://localhost:61616,
       replicationTopicConnectionFactoryBindingName=topicConnectionFactory,
       getQueueConnectionFactoryBindingName=queueConnectionFactory,
       replicationTopicBindingName=ehcache,
       getQueueBindingName=ehcacheGetQueue,
       timeoutMillis=10000"/>
</cache>
```

Example Configuration Using Open MQ

```
<cache name="sampleCacheNorep"
  maxEntriesLocalHeap="1000"
  eternal="false"
  timeToIdleSeconds="100000"
  timeToLiveSeconds="100000"
  overflowToDisk="false">
  <cacheEventListenerFactory
    class="net.sf.ehcache.distribution.jms.JMSCacheReplicatorFactory"
    properties="replicateAsynchronously=false, replicatePuts=false,
                replicateUpdates=false, replicateUpdatesViaCopy=false,
                replicateRemovals=false"
                propertySeparator=","/>
  <cacheLoaderFactory
    class="net.sf.ehcache.distribution.jms.JMSCacheLoaderFactory"
    properties="initialContextFactoryName=com.sun.jndi.fscontext.RefFSContextFactory,
                providerURL=file:///tmp,
                replicationTopicConnectionFactoryBindingName=MyConnectionFactory,
                replicationTopicBindingName=ehcache,
                getQueueConnectionFactoryBindingName=queueConnectionFactory,
                getQueueBindingName=ehcacheGetQueue,
                timeoutMillis=10000,
                userName=test,
                password=test"/>
</cache>
```

Configuring Clients for Message Queue Reliability

Ehcache replication and cache loading is designed to gracefully degrade if the message queue infrastructure stops. Replicates and loads will fail. But when the message queue comes back, these operations will start up again.

For this to work, the ConnectionFactory used with the specific message queue needs to be configured correctly. For example, with Open MQ, reconnection is configured as follows:

- imqReconnect='true' - without this reconnect will not happen
- imqPingInterval='5' - Consumers will not reconnect until they notice the connection is down. The ping interval
- does this. The default is 30. Set it lower if you want the Ehcache cluster to reform more quickly.
- Finally, unlimited retry attempts are recommended. This is also the default.

For greater reliability consider using a message queue cluster. Most message queues support clustering. The cluster configuration is once again placed in the ConnectionFactory configuration.

Tested Message Queues

Open MQ

This open source message queue is tested in integration tests. It works perfectly.

Active MQ

This open source message queue is tested in integration tests. It works perfectly other than having a problem with temporary reply queues which prevents the use of JMSCacheLoader. JMSCacheLoader is not used during replication.

Oracle AQ

Versions up to and including 0.4 do not work, due to Oracle not supporting the unified API (send) for topics.

JBoss Queue

Works as reported by a user.

Common Issues using JMS-based Replication

Active MQ Temporary Destinatons

ActiveMQ seems to have a bug in at least ActiveMQ 5.1 where it does not cleanup temporary queues, even though they have been deleted. That bug appears to be long standing but was though to have been fixed. See http://issues.apache.org/activemq/browse/AMQ-1255.

The JMSCacheLoader uses temporary reply queues when loading. The Active MQ issue is readily reproduced in Ehcache integration testing. Accordingly, use of the JMSCacheLoader with ActiveMQ is not recommended. Open MQ tests fine.

Active MQ works fine for replication.

WebSphere 5 and 6

WebSphere Application Server prevents MessageListeners, which are not MDBs, from being created in the container.

While this is a general Java EE limitation, most other app servers either are permissive or can be configured to be permissive. WebSphere 4 worked, but 5 and 6 enforce the restriction.

Accordingly, Ehcache together with JMS cannot be used with WebSphere 5 and 6.