JBoss Enterprise Application Platform 4.2
JBoss Cache Frequently Asked Questions

---

for Use with JBoss Enterprise Application Platform 4.2
Edition 1.0

Ben Wang
Scott Marlow
Bela Ban
Galder Zamarreño
Manik Surtani

JBoss Enterprise Application Platform 4.2 JBoss Cache Frequently Asked Questions

for Use with JBoss Enterprise Application Platform 4.2
Edition 1.0

Ben Wang

Bela Ban

Manik Surtani

Scott Marlow

Galder Zamarreño

Abstract

This book is a compilation of frequently asked questions about JBoss Cache for use with JBoss Enterprise Application Platform 4.2 and its patch releases.

Table of Contents

Chapter 1. General Information 3

Chapter 2. JBoss Cache - Tree Cache 5

Chapter 3. JBoss Cache - Pojo Cache 14

Chapter 4. Eviction Policies 20

Chapter 5. Cache Loaders 22

Chapter 6. Troubleshooting 24

Revision History 25

Chapter 1. General Information

Please to be building now. :(

Q: What is JBoss Cache?

A: JBoss Cache is a replicated and transactional cache. It is replicated since multiple JBoss Cache instances can be distributed (either within the same JVM or across several JVMs whether they reside on the same machine or on different machines on a network) and data is replicated across the whole group. It is transactional because a user can configure a JTA compliant transaction manager and make the cache operation transactional. Note that the cache can also be run without any replication; this is the local mode.

Currently, JBoss Cache consists of two components: a generic cache (implemented internally as **`org.jboss.cache.TreeCache`**) and a POJO cache (implemented internally as **`org.jboss.cache.aop.PojoCache`**). **`TreeCache`** is a tree-structured cache that provides replication and transaction context, while **`PojoCache`** extends the functionality of **`TreeCache`** but behaves as a true object cache providing transparent and finer-grained object mapping into internal cache.

Q: Who are the JBoss Cache developers?

A: JBossCache has been developed by Bela Ban, Ben Wang, Harald Gliebe, Manik Surtani and Brian Stansberry. Manik is the lead on JBoss Cache and Ben is the lead on PojoCache.

Q: What is the license for JBoss Cache?

A: JBoss Cache is licensed under LGPL .

Q: Where can I download JBoss Cache?

A: The JBoss Cache product download page has prebuilt binaries as well as source distributions. You can also grab snapshots from the JBoss CVS repository (see this wiki page) - the module name is **JBossCache**

Q: How do I build JBoss Cache from CVS sources?

A: To build, do **`sh build.sh jar`** . This will produce **`jboss-cache.jar`** in the **`dist/lib`** directory. Or if you want to build the standalone package, do **`sh build.sh dist`** this will produce **`dist/jboss-cache-dist.zip`** Note that you will need to use JDK 5.0 to build the distribution. You can still use the binaries you build with J2SE 1.4.x though.

Q: Which JVMs are supported by JBoss Cache?

A: JBoss Cache has been tested and supported on J2SE 1.4.x and JDK 5.0. On jboss-3.2 CVS tree, it also compiles on JDK1.3, but there is no official support for this version and using this is not recommended.

Q: From JBoss Cache 1.3.0 onwards, there is a new directory `lib-50` , what is it?

A: From JBoss Cache 1.3.0 onwards, we support the use of Java 5 annotations, used by PojoCache.

As a result, there are **`jboss-aop-jdk50.jar`** and **`jboss-cache-jdk50.jar`** that are needed to work with the Java 5 annotations. You will need to replace **`jboss-aop.jar`** and **`jboss-cache.jar`** in the lib directory with the **`-jdk50`** versions if you intend to use PojoCache, Java 5 and annotations.

Q: How do I know the version of JBoss Cache that I am using?

A: Since release 1.2, you can check the jar version by running: **`java -jar jboss-cache.jar org.jboss.cache.Version`** .

Q: Can I run JBoss Cache outside of JBoss Application Server?

A: Of course! JBoss Cache comes in two flavors:

- Integrated with JBoss Application Server as an MBean service.
- Standalone, that can run in any Java EE server such as BEA WebLogic or IBM Websphere. Of course, it can also run in a standalone Java process (i.e., outside Java EE context).

Q: Where can I report bugs or problems?

A: Please report any bugs or problems to JBoss Cache User Forum .

Chapter 2. JBoss Cache - Tree Cache

Q: How do I deploy JBoss Cache as a MBean service?

A: To deploy JBoss Cache as an MBean inside JBoss, you can copy the configuration xml file over to the **`deploy`** directory (from **`all`** configuration whereby the necessary jars are present). Under the standalone package **`etc/META-INF`** directory , there are example configuration files for different cache modes that can be used to deploy JBoss Cache as well.

Q: How do I know if my JBoss Cache MBean has been deployed?

A: To verify that your JBoss Cache MBean is deployed correctly, you can first check the log output under the command console. Next you can verify it from JBoss JMX console. Look for **`jboss.cache`** domain.

Q: How do I access the JBoss Cache MBean?

A: Accessing the JBoss Cache MBean is just like accessing any JBoss MBean. Here is a code snippet:

```
import org.jboss.mx.util.MBeanServerLocator;
import org.jboss.mx.util.MBeanProxyExt;
import org.jboss.cache.TreeCacheMBean;
import javax.management.MBeanServer;
...

MBeanServer server;
TreeCacheMBean cache;

public init() throws Exception
{
try
{
server = MBeanServerLocator.locateJBoss();
cache = (TreeCacheMBean) MBeanProxyExt.create(TreeCacheMBean.class,
"jboss.cache:service=TreeCache",
server);
}
catch (Exception ex)
{
// handle exception
}
}

public void myBusinessMethod()
{
Object value = cache.get("/my/node", "myKey");

HashMap stuff = new HashMap();
stuff.put("key1", "value1");
stuff.put("key2", "value2");
stuff.put("key3", "value3");

cache.put("/my/new/node", stuff);

cache.remove("/my/node");

...
}
```

Q: Can I run JBoss Cache on JBoss AS 3.2.x releases?

A: Yes. The JBoss Cache source code is also up to date on the jboss-3.2 CVS branch. However, only TreeCache is supported there since JBossAop (which PojoCache relies on) is only available in JBoss AS 4.x onwards.

Q: Can I run multiple JBoss Cache instances on the same VM?

A: Yes. There are some scenarios where you may want to run multiple instances of JBoss Cache. For example, you want to run multiple local cache instances with each instance having its own configuration (e.g., different cache policy). In this case, you will need multiple xml configuration files.

Q: Can TreeCache run as a second level cache inside Hibernate?

A: Yes. Since Hibernate 3.0 release, you can configure it to use JBoss Cache (namely, TreeCache)

as a second level cache. For details, see Hibernate documentation, and also see http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossCacheHibernate

Note that since Hibernate 3.0.2 and JBossCache 1.2.2, we have fixed a critical bug that depending on the usage pattern can cause deadlock during query caching.

Q: What about using PojoCache as a Hibernate cache?

A: It is not necessary to use PojoCache for second level cache inside Hibernate because Hibernate manages fine-grained fields in Java objects. So using PojoCache won't provide any advantage.

Q: How can I configure JBoss Cache?

A: You can configure the JBoss Cache through a configuration xml file. Or you can set it programmatically through its get/set methods. Check with the documentation for both examples.

Q: In the configuration xml file, there are tags such as `class` , `MBean` , etc. What are these?

A: These are tags for deploying JBoss Cache as a JBoss MBean service. For consistency, we have kept them in the standalone package as well, specifically, the **`MBean`** tag. If you run in standalone mode, JBoss Cache will ignore these elements.

Q: What is the difference between the different cache modes?

A: JBossCache has five different cache modes, i.e., **`LOCAL`** , **`REPL_SYNC`** , **`REPL_ASYNC`** , **`INVALIDATION_SYNC`** and **`INVALIDATION_ASYNC`** . If you want to run JBoss Cache as a single instance, then you should set the cache mode to **`LOCAL`** so that it won't attempt to replicate anything. If you want to have synchronous replication among different JBoss Cache instances, you set it to **`REPL_SYNC`** . For asynchronous replication, use **`AYSNC_REPL`** . If you do not wish to replicate cached data but simply inform other caches in a cluster that data under specific addresses are now stale and should be evicted from memory, use **`INVALIDATION_SYNC`** or **`INVALIDTAION_ASYNC`** . Synchronous and asynchronous behavior applies to invalidation as well as replication.

Note that **`ASYNC_REPL`** and **`INVALIDATION_ASYNC`** are non-blocking. This can be useful when you want to have another JBoss Cache serving as a mirror or backup and you don't want to wait for confirmation that this mirror has received your messages.

Q: How does JBoss Cache's replication mechanism work?

A: JBoss Cache leverages JGroups as a replication layer. A user can configure the cluster of JBoss Cache instances by sharing the same cluster name (**`cluster name`**). There is also an option of whether to populate the cache data upon starting a new instance in the **`ClusterConfig`** attribute.

Note that once all instances join the same replication group, every replication change is propagated to all participating members. There is no mechanism for sub-partitioning where some replication can be done within only a subset of members. This is on our to do list.

Q: I run a 2 node cluster. If the network dies, do the caches continue to run?

A: Yes, both will continue to run, but depending on your replication mode, all transactions or operations may not complete. If **REPL_SYNC** is used, operations will fail while if **REPL_ASYNC** is used they will succeed. Even if they succeed though, caches will be out of sync.

Q: Can I plug in library X instead of JGroups to handle remote calls and group communications?

A: At this stage (JBoss Cache 1.x) the answer is no. We do have an abstraction layer between the communication suite and JBoss Cache in the pipelines, and this may appear as a feature at some stage in the future.

Q: Does the cache need to replicate to every other instance in the cluster? Isn't this slow if the cluster is large?

A: As of JBoss Cache 1.4.0, replication need not occur to every node in the cluster. This feature - called Buddy Replication - allows each node to pick one or more 'buddies' in the cluster and only replicate to its buddies. This allows a cluster to scale very easily with no extra impact on memory or network traffic with each node added.

See the User Guide for more information on Buddy Replication, and how it can be used to achieve very high scalability.

Q: If I have the need for different TreeCache properties (e.g., CacheMode and IsolationLevel), do I simply need to create multiple TreeCache instances with the appropriate configuration?

A: Yes. All the above mentioned properties are per cache instance. Therefore you will need a separate JBoss Cache instance.

Q: Does the Tree Cache config ClusterName have any relation to the JBoss AS cluster PartitionName ?

A: Yes. They are both JGroups group names. Besides the notion of a channel in JGroups, it also can partition the channel into different group names.

Q: When using multiple JGroups based components [cluster-service.xml, treecache (multiple instances)], what is the correct/valid way to configure those components to make sure my multicast addresses don't conflict?

A: There are two parameters to consider: multicast address (plus port) and the group name. At minimum, you will have to run components using a different group name. But whether to run them on the same channel depends upon whether the communication performance is critical for you or not. If it is, then it'd be best to run them on different channels.

Q: Does JBoss Cache currently support cache persistence storage?

A: Yes. Starting with release 1.1, JBoss Cache has a CacheLoader interface that supports cache persistence. See below.

Q: Does JBoss Cache currently support cache passivation/ overflow to a data store?

A: Yes. Starting with release 1.2.4, JBoss Cache uses the CacheLoader to support cache passivation/ overflow. See documentation on how to configure and use this feature.

Q: Is JBoss Cache thread safe?

A: Yes, it is thread safe.

Q: Does JBoss Cache support XA (2PC) transactions now?

A: No, although it is also on our to do list. Our internal implementation does use a similar 2PC procedure to coordinate a transaction among different instances.

Q: Which TransactionManagers are supported by JBoss Cache?

A: JBoss Cache supports any TransactionManager that is JTA compliant such as JBossTM. A user can configure the transaction manager through the configuration xml setting. JBossCache also has a built in dummy transaction manager (**`org.jboss.cache.tm.DummyTransactionManager`**) for testing purposes only. But note that **`DummyTransactionManager`** is not thread safe .i.e., it does not support concurrent transactions. Instead, only one transaction is allowed at a time.

Q: How do I set up the cache to be transactional?

A: You either use the default (JBoss) TransactionManager to run JBossCache inside JBoss, or you have to implement the **`TransactionManagerLookup`** interface, and return an instance of your javax.transaction.TransactionManager. The configuration property **`TransactionManagerLookupClass`** defines the class to be used by the cache to fetch a reference to a TransactionManager. It is trivial to implement this class to support other TransactionManagers. Once this attribute is specified, the cache will look up the transaction context from this transaction manager.

For the client code, here is a snippet to start and commit a transaction:

```
tx = (UserTransaction)new InitialContext(prop).lookup("UserTransaction");
    tree = new TreeCache();
    config = new PropertyConfigurator();
    config.configure(tree, "META-INF/replSync-service.xml");

    tx.begin()
    tree.put(fqn, key, value);
    tx.commit();
```

Q: How do I control the cache locking level?

A: JBossCache lets you control the cache locking level through the transaction isolation level. This is configured through the attribute **`IsolationLevel`** . Currently, JBossCache employs pessimistic locking internally. And the transaction isolation level from the pessimist locking corresponds to JDBC isolation levels, namely, **`NONE`** , **`READ_UNCOMMITTED`** , **`READ_COMMITTED`** , **`REPEATABLE_READ`** , and **`SERIALIZABLE`** . Note that these isolation levels are ignored if optimistic locking is used. For details, please refer to the user manual.

Q: How does JBoss Cache lock data for concurrent access?

A: By default JBoss Cache uses pessimistic locking to lock data nodes, based on the isolation level configured. Since JBoss Cache 1.3.0, we also offer optimistic locking to allow for greater concurrency at the cost of slight processing overhead and performance. See the documentation for a more detailed discussion on concurrency and locking in JBoss Cache.

Q: How do I enable Optimistic Locking in JBoss Cache?

A: Use the XMl attribute **`NodeLockingScheme`** . Note that **`IsolationLevel`** is ignored if **`NodeLockingScheme`** is set to **`OPTIMISTIC`** . Also note that **`NodeLockingScheme`** defaults to **`PESSIMISTIC`** if omitted.

Q: How does the write lock apply to an Fqn node, say, "/org/jboss/test"?

A: First of all, JBossCache has a notion of **`root`** that serves as a starting point for every navigational operation. The default is "/" (since the default separator is "/" for the fqn). The locking then is applied to the node under root, for example "/org" (no locking "/").

Furthermore, let's say when JBossCache needs to apply a write lock on node "/org/jboss/test", it will first try to obtain read lock from the parent nodes recursively (in this example, "/org", and "/org/jboss"). Only when it succeeds then it will try to obtain a write lock on "/org/jboss/test".

Q: Can I use the cache locking level even without a transaction context?

A: Yes. JBossCache controls the individual node locking behavior through the isolation level semantics. This means even if you don't use a transaction, you can specify the lock level via isolation level. You can think of the node locking behavior outside of a transaction as if it is under transaction with **`auto_commit`** on.

Q: With replication (REPL_SYNC/REPL_ASYNC) or invalidation (INVALIDATION_SYNC/INVALIDATION_ASYNC), how often does the cache broadcast messages over the network?

A: If the updates are under transaction, then the broadcasts happen only when the transaction is about to commit (actually during the prepare stage internally). That is, it will be a batch update. However, if the operations are not under transaction context, then each update will trigger replication. Note that this has performance implication if network transport is heavy (it usually is).

Q: How can I do a mass removal?

A: If you do a cache.remove("/root"), it will recursively remove all the entries under "/root".

Q: Can I monitor and manage the JBoss Cache?

A: With JBoss Cache 1.3.0, you can if you are running JBoss Cache within JBoss AS or are using JDK 5.0's **`jconsole`** utility. See the chapter titled **Management Information** in the JBoss Cache user guide for more details.

Q: Can I disable JBoss Cache management attributes in JBoss Cache 1.3.0?

A: Yes, you can. Set the **`UseInterceptorMbeans`** configuration attribute to **`false`** (this defaults to

`true`). See the chapter titled **Management Information** in the JBoss Cache user guide for more details.

Q: What is jboss-serialization.jar, introduced in JBoss Cache 1.4.x and do I need this?

A: jboss-serialization.jar is the JBoss Serialization library, which is much more efficient in terms of speed and CPU usage as well as the generated byte stream size than standard Java serialization. This very significantly improves replication performance of custom objects placed in the cache.

From 1.4.x, JBoss Cache relies on this library and it is needed to run JBoss Cache.

Q: Can I disable JBoss Serialization and revert back to standard Java serialization?

A: Yes you can, by passing in the `-Dserialization.jboss=false` environment variable to your JVM.

Q: Does JBoss Cache support partitioning?

A: Not right now. JBoss Cache does not support partitioning that a user can configure to have different set of data residing on different cache instances while still participating as a replication group.

Q: Does JBoss Cache handle the concept of application classloading inside, say, a J2EE container?

A: Application-specific classloading is used widely inside a J2EE container. For example, a web application may require a new classloader to scope a specific version of the user library. However, by default JBoss Cache is agnostic to the classloader. In general, this leads to two kinds of problems:

- Object instance is stored in cache1 and replicated to cache2. As a result, the instance in cache2 is created by the system classloader. The replication may fail if the system classloader on cache2 does not have access to the required class. Even if replication doesn't fail, a user thread in cache2 may not be able to access the object if the user thread is expecting a type defined by the application classloader.
- Object instance is created by thread 1 and will be accessed by thread 2 (with two different classloaders). JBossCache has no notion of the different classloaders involved. As a result, you will have a `ClassCastException` . This is a standard problem in passing an object from one application space to another; JBossCache just adds a level of indirection in passing the object.

To solve the first kind of issue, in JBoss Cache 1.2.4 we introduced the concept of a `TreeCacheMarshaller` . Basically, this allows application code to register a classloader with a portion of the cache tree for use in handling objects replicated to that portion. See the TreeCacheMarshaller section of the user guide for more details.

To solve the second kind of issue, the only solution (that we know of) is to cache "serialized" byte code and only de-serialize it during every object get (and this will be expensive!). That is, during a put operation, the object instance will be serialized and therefore can be deserialized safely by a "foreign" classloader. However, the performance penalty of this approach is quite severe so in general another local in-vm version will need to be used as a "near-line" cache. Note also that each time the serialized bytes are deserialized, a new instance of the object is created.

To help with this kind of handling, JBoss has a utility class called **`MarshalledValue`** that wraps around the serialized object. Here is a code snippet that illustrates how you can create a wrapper around JBossCache to handle the classloader issue:

```
import org.jboss.invocation.MarshalledValue;

    public class CacheService {
    private TreeCache cache_;

    public object get(Fqn fqn, String key) {
    return getUnMarshalledValue(cache_.get(fqn, key));
    }

    public object set(Fqn fqn, String key, Object value) {
    cache_.put(fqn, key, getMarshalledValue(value));
    return value; // only if successful
    }

    ...

    private Object getUnMarshalledValue(object value) {
    // assuming we use the calling thread context classloader
    return ((MarshalledValue)value).get();
    }

    private Object getMarshalledValue(Object value) {
    return new MarshalledValue(value);
    }
    }
```

Q: Does JBoss Cache currently support pre-event and post-event notification?

A: Yes. Starting with release 1.2.4, JBoss Cache has introduced ExtendedTreeCacheListener which takes in consideration pre and post event notification. See documentation for more details. Note that TreeCacheListener and ExtendedTreeCacheListener will be merged into TreeCacheListener in release 1.3.

Q: How do I implement a custom listener to listen to `TreeCache` events?

A: You create a class (myListener) that extends AbstractTreeCacheListener and provide concrete implementation for the node events that you are interested in. Then you add this listener to the TreeCache instance on startup to listen to the events as they occur by calling TreeCache.addTreeCacheListener(myListener).

```
public class MyListener extends AbstractTreeCacheListener
{
...

public void nodeModify(Fqn fqn, boolean pre, boolean isLocal) {
if(log.isTraceEnabled()){
if(pre)
log.trace("Event DataNode about to be modified: " + fqn);
else
log.trace("Event DataNode modified: " + fqn);
}
}

...
}
```

Q: Can I use `useRegionBasedMarshalling` attribute in JBoss Cache in order to get around ClassCastExceptions happening when accessing data in the cache that has just been redeployed?

A: Yes, you can. Originally, **`TreeCache`** Marshalling was designed as a workaround for those replicated caches that upon state transfer did not have access to the classloaders defining the objects in the cache.

On each deployment, JBoss creates a new classloader per the top level deployment artifact, for example an EAR. You also have to bear in mind that a class in an application server is defined not only by the class name but also its classloader. So, assuming that the cache is not deployed as part of your deployment, you could deploy an application and put instances of classes belonging to this deployment inside the cache. If you did a redeployment and try to do a get operation of the data previously put, this would result on a ClassCastException. This is because even though the class names are the same, the class definitions are not. The current classloader is different to the one when the classes were originally put.

By enabling marshalling, you can control the lifecycle of the data in the cache and if on undeployment, you inactivate the region and unregister the classloader that you'd have registered on deployment, you'd evict the data in the cache locally. That means that in the next deployment, the data won't be in the cache, therefore avoiding the problem. Obviously, using marshalling to get around this problem is only recommended when you have some kind of persistence backing where the data survives, for example using CacheLoaders, or when JBossCache is used as a second level cache in a persistence framework.

To implement this feature, please follow the instructions indicated in the example located in the TreeCacheMarshaller section of the user's guide. It's worth noting that instead of a **`ServletContextListener`**, you could add this code into an **`MBean`** that contained lifecycle methods, such as **`start()`** and **`stop()`**. The key would be for this MBean to depend on the target cache, so that it can operate as long as the cache is up and running.

Chapter 3. JBoss Cache - Pojo Cache

Q: What is PojoCache?

A: PojoCache (currently implemented PojoCache as a sub-class of TreeCache) is a fine-grained field-level replicated and transactional POJO (plain old Java object) cache. By POJO, we mean that the cache: 1) automatically manages object mapping and relationship for a client under both local and replicated cache mode, 2) provides support for inheritance relationship between "aspectized" POJOs. By leveraging the dynamic AOP in JBossAop, it is able to map a complex object into the cache store, preserve and manage the object relationship behind the scene. During replication mode, it performs fine-granularity (i.e., on a per-field basis) update, and thus has the potential to boost cache performance and minimize network traffic.

From a user perspective, once your POJO is managed by the cache, all cache operations are transparent. Therefore, all the usual in-VM POJO method semantics are still preserved, providing ease of use. For example, if a POJO has been put in PojoCache (by calling putObject, for example), then any get/set method will be intercepted by PojoCache to provide the data from the cache.

Q: What's the relationship between PojoCache and TreeCacheAop classes?

A: Since release 1.4, we have created a new class called PojoCache (to better reflect the cache nature). The old implementation TreeCacheAop has been deprecated.

Q: Does PojoCache have all the functional capabilities of TreeCache?

A: Yes. PojoCache extends TreeCache so it has all the same features TreeCache such as cache mode, transaction isolation level, and eviction policy.

Q: What is the difference between TreeCache and PojoCache?

A: Think of PojoCache as a TreeCache on steroids. :-) Seriously, both are cache stores-- one is a generic cache and the other other one POJO Cache. However, while TreeCache only provides pure object reference storage (e.g., **`put(FQN fqn, Object key, Object value)`**), PojoCache goes beyond that and performs fine-grained field level replication object mapping and relationship management for a user behind the scenes. As a result, if you have complex object systems that you would like to cache, you can have PojoCache manage it for you. You simply treat your object systems as they are residing in-memory, e.g., use your regular POJO methods without worrying about cache management. Furthermore, this is true in replication mode as well.

Q: What are the steps to use the PojoCache feature?

A: Starting from release 1.3, depends on the JDK you use, it has slightly different steps. But in general, in order to use PojoCache, you will need to:

- prepare POJO. You can do either via xml declaration or annotation. For annotation, you can use either the JDK1.4 style or JDK50 one (of which is part of JVM spec). If you use JDK14, you will also need a annotation pre-compiler (annoc) to pre-process it.
- instrumentation. You will need to instrument your POJO either at compile-time or load-time. If you do it during compile-time, you use so-called aop pre-compiler (aopc) to do bytecode manipulation. If you do it via load-time, however, you need either a special system class loader or, in JDK50, you can use the javaagent option.

So if you use JDK50, for example, with annotation and load-time instrumentation, then you won't

need any pre-processing step to use PojoCache. For a full example, please refer to the distro examples directory. There are numerous PojoCache examples that uses different options.

Q: Can I run PojoCache in JBoss AS 3.2.x application server?

A: Yes and no. Yes, since JBossAop can also be back-ported to 3.2.x (see JBossAop wiki for details). However, it will take some effort. Therefore, the recommended JBoss version is 4.x to run PojoCache.

Q: Can PojoCache run as a MBean as well?

A: Yes. It is almost the same as TreeCache MBean. The only difference is the object name and the class name. E.g., instead of

```
<mbean code="org.jboss.cache.TreeCache"
                     name="jboss.cache:service=TreeCache">
```

you will have:

```
<mbean code="org.jboss.cache.aop.PojoCache"
                     name="jboss.cache:service=PojoCache">
```

in the xml configuration file.

Q: Can I pre-compile the aop classes such that I don't need to use the system classloader and jboss-aop configuration xml?

A: Yes. The latest versions of JBossCache have a pre-compiler option called **`aopc`** . You can use this option to pre-compile your "aspectized" POJO. Once the classes have been byte code generated, they can be treated as regular class files, i.e., you will not need to include any **`jboss-aop.xml`** that specifies the advisable POJO and to specify the JBossAop system class loader.

For an example of how to use **`aopc`** , please see 1) **`tools`** directory for PojoCacheTasks14.xml and PojoCacheTasks50.xml. Both contain Ant tasks that you can import to your regular project for **`annoc`** and **`aopc`** . In addition, please also check out the **`examples`** directory for concrete examples.

Q: How do I use aopc on multiple module directories?

A: In aopc, you specify the src path for a specific directory. To pre-compile multiple ones, you will need to invoke aopc multiple times.

Q: What's in the `jboss-aop.xml` configuration?

A: **`jboss-aop.xml`** is needed for POJO instrumentation. In **`jboss-aop.xml`** , you can declare your POJO (e.g., **`Person`**) to be "prepared", a JBossAop term to denote that the object will be "aspectized" by the system. After this declaration, JBossAop will invoke any interceptor that associates with this POJO. PojoCache will dynamically add an **`org.jboss.cache.aop.CacheInterceptor`** to this POJO to perform object mapping and relationship management.

Note that to add your POJO, you should declare all the fields to be "prepared" as in the example.

Q: Can I use annotation instead of the xml declaration?

A: Yes, starting with JBossCache 1.3, you can use annotation to instrument your POJO for both JDK1.4 and 1.5. Check the documentation for details.

Q: What are the pro and con of xml vs. annotation?

A: It really depends on your organization environment, I'd say, since this can be turned into a hot debate. Having said that, I feel strongly that POJO annotation is well suited for PojoCache. This is because once you specify the annotation, you'd probably change it rarely since there is no parameters to tune, for example.

Q: What are the `@org.jboss.cache.aop.annotation.Transient` and `@org.jboss.cache.aop.annotation.Serializable` field level annotations?

A: Starting in 1.4, we also offer two additional field-level annotations. The first one, **`@Transient`** , when applied has the same effect as declaring a field **`transient`** . PojoCache won't put this field under management.

The second one, **`@Serializable`** when applied, will cause PojoCache to treat the field as a Serializable object even when it is **`@PojoCacheable`** .

Q: What about compile-time vs. load-time instrumentation then?

A: Again it depends. But my preference is to do compile-time instrumentation via aopc. I prefer this approach because it is easier to debug (at least at the development stage). In addition, once I generate the new class, there is no more steps needed.

Q: Is it possible to store the same object multiple times but with different Fqn paths? Like /foo/byName and /foo/byId ?

A: Yes, you can use PojoCache to do that. It supports the notion of object reference. PojoCache manages the unique object through association of the dynamic cache interceptor.

Q: Do I need to declare all my objects "prepared" in `jboss-aop.xml` ?

A: Not necessarily. If there is an object that you don't need the cache to manage for you, you can leave it out of the declaration. The cache will treat this object as a "primitive" type. However, the object will need to implement **`Serializable`** interface for replication.

Q: Can the cache aop intercept update via reflection?

A: No. The update via reflection will not be intercepted in JBossAop and therefore PojoCache will not be able to perform the necessary synchronization.

Q: When I declare my POJO to be "aspectized", what happens to the fields with transient, static, and final modifiers?

A: PojoCache currently will ignore the fields with these modifiers. That is, it won't put these fields into

the cache (and thus no replication either).

Q: What are those keys such as `JBoss:internal:class` and `AOPInstance` ?

A: They are for internal use only. Users should ignore these keys and values in the node hashmap.

Q: What about Collection classes? Do I need to declare them "prepared"?

A: No. Since the Collection classes such as **`ArrayList`** are java util classes, aop by default won't instrument these classes. Instead, PojoCache will generate a dynamic class proxy for the Collection classes (upon the **`putObject`** call is invoked). The proxy will delegate the operations to a cache interceptor that implements the actual Collection classes APIs. That is, the system classes won't be invoked when used in PojoCache.

Internally, the cache interceptor implements the APIs by direct interaction with respect to the underlying cache store. Note that this can have implications in performance for certain APIs. For example, both **`ArrayList`** and **`LinkedList`** will have the same implementation. Plan is currently underway to optimize these APIs.

Q: How do I use `List` , `Set` , and `Map` dynamic proxy?

A: PojoCache supports classes extending from **`List`** , **`Set`** , and **`Map`** without users to declare them "aspectized". It is done via a dynamic proxy. Here is a code snippet to use an **`ArrayList`** proxy class.

```
ArrayList list = new ArrayList();
    list.add("first");

    cache.putObject("/list/test", list);
    // Put the list under the aop cache
    list.add("second");
    // Won't work since AOP intercepts the dynamic proxy not the original
POJO.

    ArrayList myList = (List)cache.getObject("/list/test");
    // we are getting a dynamic proxy instead
    myList.add("second");
    // it works now
    myList.add("third");
    myList.remove("third");
```

Q: What is the proper way of assigning two different keys with Collection class object?

A: Let's say you want to assign a **`List`** object under two different names, you will need to use the class proxy to insert the second time to ensure both are managed by the cache. Here is the code snippet.

```
ArrayList list = new ArrayList();
    list.add("first");

    cache.putObject("/list", list);
    // Put the list under the aop cache

    ArrayList myList = (List)cache.getObject("/list");
    // we are getting a dynamic proxy instead
    myList.add("second");
    // it works now

    cache.putObject("/list_alias", myList);
    // Note you will need to use the proxy here!!
    myList.remove("second");
```

Q: OK, so I know I am supposed to use proxy when manipulating the Collection classes once they are managed by the cache. But what happens to Pojos that share the Collection objects, e.g., a `List` instance that is shared by 2 Pojos?

A: Pojos that share Collection instance references will be handled by the cache automatically. That is, when you ask the Cache to manage it, the Cache will dynamically swap out the regular Collection references with the dynamic proxy ones. As a result, it is transparent to the users.

Q: What happens when my "aspectized" POJO has field members that are of Collection class ?

A: When a user puts a POJO into the cache through the call **`putObject`** , it will recursively map the field members into the cache store as well. When the field member is of a Collection class (e.g., List, Set, or Map), PojoCache will first map the collection into cache. Then, it will swap out dynamically the field reference with an corresponding proxy reference.

This is necessary so that an internal update on the field member will be intercepted by the cache.

Q: What are the limitation of Collection classes in PojoCache?

A: Use of Collection class in PojoCache helps you to track fine-grained changes in your collection fields automatically. However, current implementation has the follow limitation that we plan to address soon.

Currently, we only support a limited implementation of Collection classes. That is, we support APIs in List, Set, and Map. However, since the APIs do not stipulate of constraints like NULL key or value, it makes mapping of user instance to our proxy tricky. For example, ArrayList would allow NULL value and some other implementation would not. The Set interface maps to java.util.HashSet implementation. The List interface maps to java.util.ArrayList implementation. The Map interface maps to java.util.HashMap implementation.

Another related issue is the expected performance. For example, the current implementation is ordered, so that makes insert/delete from the Collection slow. Performance between Set, Map and List collections also vary. Adding items to a Set is slower than a List or Map, since Set does not allow duplicate entries.

Q: What are the pros and cons of PojoCache?

A: As mentioned in the reference doc, PojoCache has the following advantages:

- Fine-grained replication and/or persistency. If you use a distributed PojoCache and once your POJO is put in the cache store, there is no need to use another API to trigger your changes. Furthermore, the replication are fine-grained field level. Note this also applies to persistency.
- Fine-grained replication can have potential performance gain if your POJO is big and the changes are fine-grained, e.g., only to some selected fields.
- POJO can posses object relationship, e.g., multiple referenced. Distributed PojoCache will handle this transparently for you.

And here are some cases that you may not want to use PojoCache:

- You use only cache. That is you don't need replication or persistency. Then since everything is operated on the in-memory POJO reference, there is no need for PojoCache.
- You have simple and small POJOs. Your POJO is small in size and also there is no object relationship, then PojoCache possess not clear advantage to plain cache.
- Your application is bounded by memory usage. Because PojoCache need almost twice as much of memory (the original POJO in-memory space and also the additional cache store for the primitive fields), you may not want to use PojoCache.
- Your POJO lifetime is short. That is, you need to create and destroy your POJO often. Then you need to do "pubObject" and "removeObject" often, it will be slow in performance.

Chapter 4. Eviction Policies

Q: Does JBoss Cache support eviction policies?

A: Yes. JBoss Cache currently implements a LRU eviction policy for both TreeCache (**`org.jboss.cache.eviction.LRUPolicy`**) and PojoCache (**`org.jboss.cache.aop.eviction.AopLRUPolicy`**). Users can also plug in their own eviction policy algorithms. See user manual for details. Currently there is user-contributed policy called **`FIFOPolicy`** that evicts the node based on FIFO principle only.

Q: Why can't I use `org.jboss.cache.eviction.LRUPolicy` for PojoCache as well?

A: For PojoCache, you will need to use **`org.jboss.cache.aop.eviction.AopLRUPolicy`**) because AOP has its eviction algorithm, although is LRU but has totally different notion of an "object", for example.

Q: Does JBoss Cache's implemented LRU eviction policy operates in replication mode?

A: Yes and no. :-)

The LRU policy only operates in local mode. That is, nodes are only evicted locally. This may cause the cache contents not to be synchronized temporarily. But when a user tries to obtain the cached contents of an evicted node and finds out that is null (e.g., **`get`** returns null), it should get it from the other data source and re-populate the data in the cache. During this moment, the node content will be propagated and the cache content will be in sync.

However, you still can run eviction policies with cache mode set to either **`REPL_SYNC`** or **`REPL_ASYNC`** . Depending on your use case, you can set multiple cache instances to have their own eviction policy (which are applied locally) or just have selected instances with eviction policies activated.

Also note that, with cache loader option, a locally evicted node can also be persisted to the backend store and a user can retrieve it from the store later on.

Q: Does JBoss Cache support `Region` ?

A: Yes. JBoss Cache has the notion of region where a user can configure the eviction policy parameters (e.g., **`maxNodes`** or **`timeToIdleSeconds`**)

A region in JBoss Cache denotes a portion of tree hierarchy, e.g., a fully qualified name (**`FQN`**). For example, a user can define **`/org/jboss`** and **`/org/foocom`** as two separate regions. But note that you can configure the region programmatically now, i.e., everything has to be configured through the xml file.

Q: What are the `EvictionPolicyConfig` tag parameters for `org.jboss.cache.eviction.LRUPolicy` ?

A: They are:

Table 4.1. Parameters

wakeUpIntervalInSeconds	Interval where the clean up thread wakes to process the sitting queue and sweep away the old data.
region	A area where each eviction policy parameters are specified. Note that it needs a minimum of **`/_default`** region.
maxNodes	Max number of nodes allowed in the eviction queue. 0 means no limit.
timeToLiveInSeconds	Age (in seconds) for the node to be evicted in the queue. 0 denotes no limit.

Q: I have turned on the eviction policy, why do I still get "out of memory" (OOM) exception?

A: OOM can happen when the speed of cache access exceeds the speed of eviction policy handling timer. Eviction policy handler will wake up every **`wakeUpIntervalInSeconds`** seconds to process the eviction event queue. And the queue size is fixed at 20000 now. So when the queue size is full, it will create a backlog and cause OOM to happen unless the eviction timer catches up. To address this problem, in addition to increase the VM heap size, you can also reduce the **`wakeUpIntervaleInSeconds`** so the timer thread processes the queue more frequently.

We will also externalize the queue size so it will be configurable in the next release.

Chapter 5. Cache Loaders

Q: What is a CacheLoader?

A: A CacheLoader is the connection of JBossCache to a (persistent) data store. The CacheLoader is called by JBossCache to fetch data from a store when that data is not in the cache, and when modifications are made to data in the cache the CacheLoader is called to store those modifications back to the store.

In conjunction with eviction policies, JBossCache with a CacheLoader allows a user to maintain a bounded cache for a large backend datastore. Frequently used data is fetched from the datastore into the cache, and the least used data is evicted, in order to provide fast access to frequently accessed data. This is all configured through XML, and the programmer doesn't have to take care of loading and eviction.

JBossCache currently ships with several CacheLoader implementations, including:

- FileCacheLoader: this implementation uses the file system to store and retrieve data. JBossCache nodes are mapped to directories, subnodes to subdirectories etc. Attributes of a node are mapped to a file **`data`** inside the directory.
- BdbjeCacheLoader: this implementation is based on the Sleepycat Java Edition database, a fast and efficient transactional database. It uses a single file for the entire store. Note that if you use Sleepycat's CacheLoader with JBoss Cache and wish to ship your product, you will have to acquire a commercial license from Sleepycat .
- JDBCCacheLoader: this implementation uses the relational database as the persistent storage.
- ClusteredCacheLoader: this implementation queries the rest of the cluster, treating other servers' in-memory state as a data store.
- And more. See the documentation for more details.

Q: Can writing to CacheLoaders be asynchronous?

A: As of JBossCache 1.2.4, yes. Set the CacheLoaderAsynchronous property to true. See the JBossCache documentation for a more detailed discussion. By default though, all cache loader writes are synchronous and will block.

Q: Can I write my own CacheLoader ?

A: Yes. A CacheLoader is a class implementing **`org.jboss.cache.loader.CacheLoader`** . It is configured via the XML file (see JBossCache and Tutorial documentation).

Q: Does a CacheLoader have to use a persistent store ?

A: No, a CacheLoader could for example fetch (and possibly store) its data from a webdav-capable webserver. Another example is a caching proxy server, which fetches contents from the web. Note that an implementation of CacheLoader may not implement the 'store' functionality in this case, but just the 'load' functionality.

Q: What can I use a CacheLoader for?

A: Some applications:

- HTTP sessions can be persisted (besides being replicated by JBossCache). The

CacheLoader can be configured to be shared, or unshared, meaning that every node in a cluster has its own local store. It is also possible to attach a CacheLoader to just *one* of the nodes.

- Simple persistence for POJOs. Use of JBossCache aop and a local CacheLoader persist POJOs transparently into the store provided by the CacheLoader.
- Highly available replicated and persisted data store. The service is up as long as at least 1 node is running, but even if all nodes are taken offline, when the first node is started again, the data previously saved will still be available (e.g. a shopping cart).
- A caching web proxy (a la Squid): all data are contents of URLs, users access the proxy, and if the URL is not in the cache, the CacheLoader fetches it from the web. This could actually be a replicated and transactional version of Squid.

Q: How do I configure JBossCache with a CacheLoader?

A: Through XML: both the fully-qualified classname of the CacheLoader and its configuration string have to be given. JBossCache will then instantiate a CacheLoader. See JBossCache documentation for details.

Q: Do I have to pay to use Sleepycat's CacheLoader?

A: Not if you use it only for personal use. As soon as you distribute your product with BdbjeCacheLoader, you have to purchase a commercial license from Sleepycat. See details at http://www.sleepycat.com/jeforjbosscache .

Q: Can I use more than one cache loader?

A: As of JBossCache 1.3.0, yes. With the new CacheLoaderConfiguration XML element (see user manual section on cache loaders) you can now describe several cache loaders. The impact is that the cache will look at all of the cache loaders in the order they've been configured, until it finds a valid, non-null element of data. When performing writes, all cache loaders are written to (except if the ignoreModifications element has been set to true for a specific cache loader.

Q: Why do cache loaders go into an inconsistent state when I use transactions, pessimistic locking, and I attempt to read a node after removing it from within the same transaction scope?

A: This is a known bug (see JBCACHE-477 and JBCACHE-352), which have been fixed in JBoss Cache 1.4.0. A very simple workaround if you're using JBoss Cache 1.3.x is to use optimistic locking.

One of the consequences of this bug is that, for example, if you use PojoCache with pojos that have private references to a List and you update and remove someelements of that List within a transaction (when using pessimistic locking and a cache loader), you may see IllegalStateExceptions thrown.

Chapter 6. Troubleshooting

Q: I am having problems getting JBoss Cache to work, where can I get information on troubleshooting?

A: Troubleshooting section can be found in the following wiki link .

Revision History

Revision 1.1-4.402	**Fri Oct 25 2013**	**Rüdiger Landmann**
Rebuild with Publican 4.0.0		
Revision 1.1-4.1	**2013-06-11**	**Misty Stanley-Jones**
Rebuild for updated legal template		
Revision 1.1-4	**2012-07-18**	**Anthony Towns**
Rebuild for Publican 3.0		
Revision 1.1-0	**Thu Jun 03 2010**	**Laura Bailey**
Updated versions for new doc structure.		