# Frequently Asked Questions about JBoss Cache

Release 2.1.0 Alegrias
November 2007

**Authors:**

ManikSurtani(manik@jboss.org)

BenWang(ben.wang@jboss.com)

BelaBan(bela@jboss.com)

ScottMarlow(smarlow@novell.com)

GalderZamarreño(galder.zamarreno@jboss.com)

# Table of Contents

# 1

# General Information

## 1.1. What is JBoss Cache?

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 [http://java.sun.com/products/jta/] compliant transaction manager and make any cache interaction transactional. Note that the cache can also be run without any replication; this is the local mode.

JBoss Cache comes in two flavours: Core and Pojo versions. The core library (using the `org.jboss.cache.Cache` interface ) is the underlying library that organises data in a tree-like structure and handles all locking, passivation, eviction and replication characteristics of data in the cache. The pojo library (using the `org.jboss.cache.pojo.PojoCache` interface) is built atop the core library and allows introspection of objects in the cache providing transparent coherence by using JBoss AOP. Note that the Pojo version of JBoss Cache (referred to as PojoCache) comes with a separate set of documentation (user guide, FAQ, etc.) available on the JBoss Cache documentation site [http://labs.jboss.com/portal/jbosscache/docs/index.html] .

JBoss Cache is made available in one of four different packages:

- `jboss-cache-core`
  contains the core Cache library for users who do not wish to use the additional functionality offered by PojoCache.

- `jboss-cache-pojo`
  contains the core Cache library as well as the PojoCache extensions and dependencies.

- `jboss-cache-all`
  contains all of the above, including unit tests and source code.

- `jboss-cache-core-JDK140`
  contains a JDK 1.4 compatible version of the core Cache library. Note that PojoCache is only available for JDK 5.0.

## 1.2. Who are the JBoss Cache developers?

JBoss Cache has an active community of developers and contributors. The project was founded by Bela Ban and is currently led by Manik Surtani. Jason Greene is the lead for the PojoCache

subsystem, and other contributors both past and present include Ben Wang, Harald Gliebe, Brian Stansberry, Galder Zamarreno and Elias Ross.

1.3. What is the license for JBoss Cache?

JBoss Cache is licensed under LGPL [http://www.gnu.org/licenses/lgpl.html] .

1.4. Where can I download JBoss Cache?

The JBoss Cache product download page [http://www.jboss.com/products/jbosscache/downloads] has prebuilt binaries as well as source distributions. You can also grab snapshots from the JBoss CVS repository (see this wiki page [http://wiki.jboss.org/wiki/Wiki.jsp?page=CVSRepository] ) - the module name is **JBossCache**

1.5. How do I build JBoss Cache from CVS sources?

To build, do `sh build.sh jar` . This will produce `jboss-cache.jar` and `pojocache.jar` in the `dist/lib` directory. Note that you will need to use JDK 5 to build the distribution.

1.6. Which versions of the JDK are supported by JBoss Cache?

JBoss Cache is baselined on Java 5.0 and this is the platform on which JBoss Cache is most thoroughly tested. If, for whatever reason you have to use Java 1.4, you could build a retroweaved version of the core cache library that is Java 1.4 compatible, using the simple instructions on this wiki page on building and running JBoss Cache on Java 1.4. [http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossCacheHabaneroJava1.4] . Note that Red Hat Inc. does not offer commercial support for retroweaved binaries at this stage.

Java 6 should work as well, and we haven't heard of any specific problems of JBoss Cache run under Java 6.

1.7. How do I know the version of JBoss Cache that I am using?

`java -jar jbosscache.jar` will spit out version details.

1.8. Can I run JBoss Cache outside of JBoss Application Server?

Of course! Even though JBoss Cache comes integrated with JBoss Application Server as an MBean service, it can also be run standalone, in any Java EE server such as BEA WebLogic, IBM Websphere or Tomcat. It can also run in a standalone Java process, completely outside of an application server. See the user guide for more details.

1.9. How can I migrate my application and configuration from using JBoss Cache 1.x to 2.x?

Look at this wiki page [http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossCache200Migration] for help.

1.10. Where can I report bugs or problems?

Please report any bugs or problems to JBoss Cache User Forum [http://www.jboss.org/index.html?module=bb&op=viewforum&f=157] .

# 2

# JBoss Cache - Core

2.1. How do I deploy JBoss Cache as a MBean service?

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.

2.2. How do I know if my JBoss Cache MBean has been deployed?

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.

2.3. How do I access the JBoss Cache MBean?

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, "jbos
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");

...
}
```

2.4. Can I run multiple JBoss Cache instances on the same VM?

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.

2.5. Can JBoss Cache run as a second level cache inside Hibernate?

Yes. Since Hibernate 3.0 release, you can configure it to use JBoss Cache as a second level cache. For details, see Hibernate documentation, and also see http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossCacheHibernate [http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossCacheHibernate]

2.6. What about using Pojo Cache as a Hibernate cache?

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

2.7. How can I configure JBoss Cache?

You can configure the JBoss Cache through a configuration xml file or programmatically using a `org.jboss.cache.config.Configuration` object, passed in to the `org.jboss.cache.CacheFactory` instance.

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

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.

2.9. What is the difference between the different cache modes?

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.

2.10. How does JBoss Cache's replication mechanism work?

JBoss Cache leverages JGroups [http://www.jgroups.org] 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, unless you use the Buddy Replication features. See the user guide for more details on this.

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

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.

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

At this stage 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.

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

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.

2.14. If I have the need for different configuration properties (e.g., `CacheMode` and `IsolationLevel` ), do I simply need to create multiple `org.jboss.cache.Cache` instances with the appropriate configuration?

Yes. All the above mentioned properties are per cache instance. Therefore you will need separate `org.jboss.cache.Cache` instances.

2.15. Isn't this expensive from a networking standpoint, i.e., needing to create sockets for each `org.jboss.cache.Cache` instance?

Yes, it can be. For such cases it is recommended that you configure your cache using the JGroups Multiplexer, which allows several caches to share a single JGroups channel. Please see the User Guide for details on how to configure the JGroups Multiplexer.

2.16. Does the `ClusterName` configuration element have any relation to the JBoss AS cluster `PartitionName` ?

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.

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

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.

2.18. Does JBoss Cache support cache persistence storage?

Yes. JBoss Cache has a cache loader interface that supports cache persistence. See below for more FAQs on cache loaders.

2.19. Does JBoss Cache support cache passivation/ overflow to a data store?

Yes. JBoss Cache uses the cache loader to support cache passivation/ overflow. See documentation on how to configure and use this feature.

2.20. Is JBoss Cache thread safe?

Yes, it is thread safe.

2.21. Does JBoss Cache support XA (2PC) transactions now?

No, although it is also on our to do list. Our internal implementation does use a procedure similar to 2PC to coordinate a transactions among different instances, but JBoss Cache is not an XA resource.

2.22. Which transaction managers are supported by JBoss Cache?

JBoss Cache supports any TransactionManager that is JTA [http://java.sun.com/products/jta/] compliant such as JBossTM or JBossTS. JBoss Cache ships with a dummy transaction manager ( `org.jboss.cache.transaction.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.

2.23. How do I set up the cache to be transactional?

You either use the default transaction manager that ships with JBoss AS or you have to implement the `org.jboss.cache.transaction.TransactionManagerLookup` interface, and return an instance of your `javax.transaction.TransactionManager` implementation. The configuration property

`TransactionManagerLookupClass` defines the class to be used by the cache to fetch a reference to a transaction manager. It is trivial to implement this interface to support other transaction managers. Once this attribute is specified, the cache will look up the transaction context from this transaction manager.

2.24. How do I control the cache locking level?

JBoss Cache lets you control the cache locking level through the transaction isolation level. This is configured through the attribute `IsolationLevel` . The transaction isolation levels correspond to database 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.

2.25. How does JBoss Cache lock data for concurrent access?

By default JBoss Cache uses pessimistic locking to lock data nodes, based on the isolation level configured. 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.

2.26. How do I enable Optimistic Locking in JBoss Cache?

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.

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

First of all, JBoss Cache 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 JBoss Cache 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".

2.28. Can I use the cache locking level even without a transaction context?

Yes. JBoss Cache 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.

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

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).

2.30. How can I do a mass removal?

If you do a cache.removeNode(Fqn.fromString("/myroot")), it will recursively remove all the entries under "/myroot".

2.31. Can I monitor and manage the JBoss Cache?

Yes, using a JMX console such as the one shipped with JBoss AS or Java 5's `jconsole` utility. See the chapter titled **Management Information** in the JBoss Cache user guide for more details.

2.32. Can I disable JBoss Cache management attributes?

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.

2.33. What happened to jboss-serialization.jar?

As of JBoss Cache 2.0.0, the dependency on JBoss Serialization has been dropped since most of the benefits of JBoss Serialization are available in updated Java 5 VMs. Since JBoss Cache 2.0.0 is baselined on Java 5, there was no need to provide these benefits separately.

2.34. Does JBoss Cache support partitioning?

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.

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

Application-specific classloading is used widely inside a Java EE 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). JBoss Cache 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; JBoss Cache just adds a level of indirection in passing the object.

To solve the first kind of issue JBoss Cache uses a `CacheMarshaller` . 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 `CacheMarshaller` 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 JBoss Cache to handle the classloader issue:

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

public class CacheService
{
private Cache 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);
}
}
```

2.36. Does JBoss Cache currently support pre-event and post-event notification?

Yes. A boolean is passed in to each notification callback identifying whether the callback is before or after the event. See the `org.jboss.cache.CacheListener` interface for details.

2.37. How do I implement a custom listener to listen to cache events?

Either implement `org.jboss.cache.CacheListener` or extend `org.jboss.cache.AbstractCacheListener` and override for the events you are interested in. You can then register the listener using the `org.jboss.cache.Cache.addCacheListener()` API.

2.38. 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?

Yes, you can. Originally, cache 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 deactivate 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 JBoss Cache 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 CacheMarshaller 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.

# 3

# Eviction Policies

3.1. Does JBoss Cache support eviction policies?

Yes. JBoss Cache currently supports multiple eviction policies such as LRU, MRU, and FIFO. Users can also plug in their own eviction policy algorithms. See user manual for details.

3.2. Does JBoss Cache's eviction policy operates in replication mode?

Yes and no. :-)

The eviction 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.

3.3. Does JBoss Cache support `Region` ?

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 ( `org.jboss.cache.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.

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

They are:

**Table 3.1. Parameters**

| | |
|---|---|
| eventQueueSize | A fine-tuning parameter where you can configure the size of the eviction notification event queue. Default to 200,000. |
| 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. |

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

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. So when the queue size is full, it will create a backlog and cause out-of-memory exceptions 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.

The eviction queue size is configurable.

# 4

# Cache Loaders

## 4.1. What is a cache loader?

A cache loader is the connection of JBoss Cache to a (persistent) data store. The cache loader is called by JBoss Cache 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 Cache Loader is called to store those modifications back to the store.

In conjunction with eviction policies, JBoss Cache with a cache loader 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.

JBoss Cache currently ships with several cache loader implementations, including:

- `org.jboss.cache.loader.FileCacheLoader` : this implementation uses the file system to store and retrieve data. JBoss Cache nodes are mapped to directories, subnodes to subdirectories etc. Attributes of a node are mapped to a data file inside the directory.
- `org.jboss.cache.loader.BdbjeCacheLoader` : this implementation is based on the Oracle's Berkeley DB Java Edition database, a fast and efficient transactional database. It uses a single file for the entire store. Note that if you use the Berkeley DB cache loader with JBoss Cache and wish to ship your product, you will have to acquire a commercial license from Oracle [http://www.sleepycat.com/jeforjbosscache] .
- `org.jboss.cache.loader.JDBCCacheLoader` : this implementation uses the relational database as the persistent storage.
- And more. See the chapter on cache loaders in the User Guide for more details.

## 4.2. Is the FileCacheLoader recommended for production use?

No, it is not. The FileCacheLoader has some severe limitations which restrict it's use in a production environment, or if used in such an environment, it should be used with due care and sufficient understanding of these limitations.

- Due to the way the FileCacheLoader represents a tree structure on disk (directories and files) traversal is inefficient for deep trees.
- Usage on shared filesystems like NFS, Windows shares, etc. should be avoided as these do not implement proper file locking and can cause data corruption.

- Usage with an isolation level of NONE can cause corrupt writes as multiple threads attempt to write to the same file.
- File systems are inherently not transactional, so when attempting to use your cache in a transactional context, failures when writing to the file (which happens during the commit phase) cannot be recovered.

As a rule of thumb, it is recommended that the FileCacheLoader not be used in a highly concurrent, transactional or stressful environment, and it's use is restricted to testing.

4.3. Can writing to cache loaders be asynchronous?

Yes. Set the `async` attrobute to true. See the JBoss Cache User Guide for a more detailed discussion. By default though, all cache loader writes are synchronous and will block.

4.4. Can I write my own cache loader ?

Yes. A cache loader is a class implementing `org.jboss.cache.loader.CacheLoader` or extending `org.jboss.cache.loader.AbstractCacheLoader` . It is configured via the XML file (see JBoss Cache User Guide).

4.5. Does a cache loader have to use a persistent store ?

No, a cache loader 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.

4.6. Do I have to pay to use Oracle's Berkeley DB CacheLoader?

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 Oracle. See details at http://www.sleepycat.com/jeforjbosscache [http://www.sleepycat.com/jeforjbosscache] .

4.7. Can I use more than one cache loader?

Yes. Within the CacheLoaderConfiguration XML element (see user guide chapter on cache loaders) you can 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.

4.8. Can I migrate a JDBCacheLoader or FileCacheLoader based cache store containing data formatted with JBoss Cache 1.x.x to JBoss Cache 2.0 format?

Yes. See "Transforming Cache Loaders" section within the "Cache Loaders" section located in the JBoss Cache users guide.

# 5

# Troubleshooting

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

Troubleshooting section can be found in the following wiki link [http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossCacheTroubleshooting] .