Frequently Asked Questions

The Infinispan community

Table of Contents

Welcome to Infinispan's Frequently Asked Questions document. We hope you find the answers to your queries here, however if you don't, we encourage you to connect with the Infinispan community and ask any questions you may have on the Infinispan User Forums.

Chapter 1. Project questions

1.1. What is Infinispan?

Infinispan is an open source data grid platform. It exposes a JSR-107 compatible `Cache` interface (which in turn extends `java.util.Map`) in which you can store objects. While Infinispan can be run in local mode, its real value is in distributed mode where caches cluster together and expose a large memory heap. Distributed mode is more powerful than simple replication since each data entry is spread out only to a fixed number of replicas thus providing resilience to server failures as well as scalability since the work done to store each entry is constant in relation to a cluster size.

So, why would you use it? Infinispan offers:

- *Massive heap* and *high availability* - If you have 100 blade servers, and each node has 2GB of space to dedicate to a replicated cache, you end up with 2 GB of total data. Every server is just a copy. On the other hand, with a distributed grid - assuming you want 1 copy per data item - you get a 100 GB memory backed virtual heap that is efficiently accessible from anywhere in the grid. If a server fails, the grid simply creates new copies of the lost data, and puts them on other servers. Applications looking for ultimate performance are no longer forced to delegate the majority of their data lookups to a large single database server - a bottleneck that exists in over 80% of enterprise applications!
- *Scalability* - Since data is evenly distributed there is essentially no major limit to the size of the grid, except group communication on the network - which is minimised to just discovery of new nodes. All data access patterns use peer-to-peer communication where nodes directly speak to each other, which scales very well. Infinispan does not require entire infrastructure shutdown to allow scaling up or down. Simply add/remove machines to your cluster without incurring any down-time.
- *Data distribution* - Infinispan uses consistent hash algorithm to determine where keys should be located in the cluster. Consistent hashing allows for cheap, fast and above all, deterministic location of keys with no need for further metadata or network traffic. The goal of data distribution is to maintain enough copies of state in the cluster so it can be durable and fault tolerant, but not too many copies to prevent Infinispan from being scalable.
- *Persistence* - Infinispan exposes a CacheStore interface, and several high-performance implementations - including JDBC cache stores, lesystem-based cache stores, Amazon S3 cache stores, etc. CacheStores can be used for "warm starts", or simply to ensure data in the grid survives complete grid restarts, or even to over ow to disk if you really do run out of memory.
- *Language bindings* (PHP, Python, Ruby, C, etc.) - Infinispan offers support for both the popular memcached protocol - with existing clients for almost every popular programming language - as well as an optimised Infinispan-specific protocol called Hot Rod. This means that Infinispan is not just useful to Java. Any major website or application that wants to take advantage of a fast data grid will be able to do so.
- *Management* - When you start thinking about running a grid on several hundred servers, management is no longer an extra, it becomes a necessity. Since version 8.0, Infinispan bundles a management console.
- *Support for Compute Grids* - Infinispan 5 adds the ability to pass a Runnable around the grid.

This allows you to push complex processing towards the server where data is local, and pull back results using a Future. This map/reduce style paradigm is common in applications where a large amount of data is needed to compute relatively small results.

Also see this page on the Infinispan website.

1.2. What would I use Infinispan for?

Most people use Infinispan for one of two reasons. Firstly, as a distributed cache. Putting Infinispan in front of your database, disk-based NoSQL store or any part of your system that is a bottleneck can greatly help improve performance. Often, a simple cache isn't enough - for example if your application is clustered and cache coherency is important to data consistency. A distributed cache can greatly help here.

The other major use-case is as a NoSQL data store. In addition to being in memory, Infinispan can also persist data to a more permanent store. We call this a cache store. Cache stores are pluggable, you can easily write your own, and many already exist for you to use.

A less common use case is adding clusterability and high availability to frameworks. Since Infinispan exposes a distributed data structure, frameworks and libraries that also need to be clustered can easily achieve this by embedding Infinispan and delegating all state management to Infinispan. This way, any framework can easily be clustered by letting Infinispan do all the heavy lifting.

1.3. How is Infinispan related to JBoss Cache?

Certain design ideas and indeed some code have been borrowed from JBoss Cache 3.x, however JBoss Cache is in no way a dependency. Infinispan is a complete, separate and standalone project. Some may consider this a fork, but the people behind Infinispan and JBoss Cache see it as an evolution, since all future effort will be on Infinispan and not JBoss Cache.

1.4. What version of Java does Infinispan need to run? Does Infinispan need an application server to run?

All that is needed is a Java 8 compatible JVM. An application server is *not* a requirement.

1.5. Will there be a POJO Cache replacement in Infinispan?

Yes, and this is called Hibernate OGM .

1.6. How come Infinispan's first release is 4.0.0? This sounds weird!

We didn't want to release Infinispan as a 1.0, as in all fairness it is not a virgin codebase. A lot of the

code, designs and ideas in Infinispan are from JBoss Cache, and has been tried and tested, proven in high stress environments. Infinispan should thus be viewed as a mature and stable platform and not a new, experimental one.

1.7. How is this related to JSR 107, the JCACHE specification?

Infinispan core engineers are on the JSR 107 expert group and starting with version 7.0.0, Infinispan provides a certified compatible implementation of version 1.0.0 of the specification.

Have a look at Using Infinispan as a JSR107 (JCache) provider for details.

1.8. Can I use Infinispan with Hibernate?

Yes, you can combine one or more of these integrations in the same application:

- *Using Infinispan as a database replacement*: using Hibernate OGM you can replace the RDBMS and store your entities and relations directly in Infinispan, interacting with it through the well known JPA 2.1 interface, with some limitations in the query capabilities. Hibernate OGM also automates mapping, encoding and decoding of JPA entities to Protobuf. For more details see Hibernate OGM .
- *Caching database access*: Hibernate can cache frequently loaded entities and queries in Infinispan, taking advantage of state of the art eviction algorithms, and clustering if needed but it provides a good performance boost in non-clustered deployments too. See Using Infinispan as JPA/Hibernate Second Level Cache Provider for details on how to do this.
- *Storing Lucene indexes*: When using Hibernate Search to provide full-text capabilities to your Hibernate/JPA enabled application, you need to store an Apache Lucene index separately from the database. You can store the index in Infinispan: this is ideal for clustered applications since it's otherwise tricky to share the index with correct locking on shared file systems, but is an interesting option for non-clustered deployments as well as it can combine the benefits of in-memory performance with reliability and write-through to any CacheStore supported by Infinispan.
- *Using full-text queries on Infinispan*: If you liked the powerful full-text and data mining capabilities of Hibernate Search, but don't need JPA or a database, you can use the indexing and query engine only: the Infinispan Query module reuses Hibernate Search internally, depending on some Hibernate libraries but exposing the Search capabilities only. See Querying Infinispan .
- A *combination of multiple such integrations*: you can use Hibernate OGM as an interface to perform CRUD operations on some Infinispan caches configured for resiliency, while also activating Hibernate 2nd level caching using some different caches configured for high performance read mostly access, and also use Hibernate Search to index your domain model while storing the indexes in Infinispan itself.

Chapter 2. Technical questions

2.1. General questions

2.1.1. What APIs does Infinispan offer?

Infinispan's primary API - `org.infinispan.Cache` - extends `java.util.concurrent.ConcurrentMap` and closely resembles `javax.cache.Cache` from JSR 107. This is the most performant API to use, and should be used for all new projects.

`org.infinispan.tree.TreeCache` is a tree structured API that looks a lot like JBoss Cache's API. Note that the similarities end at the interface though, since internal implementation and representation of the tree is completely different, using a much more efficient flat structure.

TreeCache should be considered as a compatibility API, if you are migrating from JBoss Cache and cannot invest the time in rewriting your application, or your application specifically relies on a tree structure.

2.1.2. Which JVMs (JDKs) does Infinispan work with?

Infinispan is developed and primarily tested against Oracle Java SE 8. It should work with most Java SE 9 implementations, including those from IBM, HP, Apple, Oracle, and OpenJDK. We also build/test against JDK 9.

2.1.3. Does Infinispan store data by value or by reference?

By default, Infinispan stores data by reference. So once clients store some data, clients can still modify entries via original object references. This means that since client references are valid, clients can make changes to entries in the cache using those references, but these modifications are only local and you still need to call one of the cache's put/replace... methods in order for changes to replicate.

Obviously, allowing clients to modify cache contents directly, without any cache invocation, has some risks and that's why Infinispan offers the possibility to store data by value instead. The way store-by-value is enabled is by enabling Infinispan to store data in binary format and forcing it to do these binary transformations eagerly.

The reason Infinispan stores data by-reference instead of by-value is performance. Storing data by reference is quicker than doing it by value because it does not have the penalty of having to transform keys and values into their binary format.

2.1.4. Can I use Infinispan with Groovy? What about Jython, Clojure, JRuby or Scala etc.?

While we haven't extensively tested Infinispan on anything other than Java, there is no reason why it cannot be used in any other environment that sits atop a JVM. We encourage you to try, and we'd love to hear your experiences on using Infinispan from other JVM languages.

2.2. Cache Loader and Cache Store questions

2.2.1. Cache loaders and cache stores - what's the difference?

Please read Persistence for information about the persistence SPI.

2.2.2. Are modifications to asynchronous cache stores coalesced or aggregated?

Modifications are coalesced or aggregated for the interval that the modification processor thread is currently applying. This means that while changes are being queued, if multiple modifications are made to the same key, only the key's last state will be applied, hence reducing the number of calls to the cache store.

2.2.3. What does the passivation flag do?

Passivation is a mode of storing entries in the cache store *only when* they are evicted from memory. The benefit of this approach is to prevent a lot of expensive writes to the cache store if an entry is hot (frequently used) and hence *not* evicted from memory. The reverse process, known as *activation*, occurs when a thread attempts to access an entry which is *not* in memory but is in the store (i.e., a *passivated* entry). Activation involves loading the entry into memory, and then *removing* it from the cache store. With passivation enabled, the cache uses the cache store as an overflow tank, akin to swapping memory pages to disk in virtual memory implementations in operating systems.

If passivation is disabled, the cache store behaves as a write-through (or write-behind if asynchronous) cache, where all entries in memory are also maintained in the cache store. The effect of this is that the cache store will always contain a superset of what is in memory.

2.2.4. What if I get IOException "Unsupported protocol version 48" with JdbcStringBasedCacheStore?

You have probably set your data column type to `VARCHAR`, `CLOB` or something similar, but it should be `BLOB/VARBINARY`. Even though it's called `JdbcStringBasedCacheStore`, only the keys are required to be strings; the values can be anything, so they need to be stored in a binary column. See the setDataColumnType javadoc for more details.

2.2.5. Is there any way I can boost cache store's performance?

If, for put operations, you don't need the previous values existing in the cache/store then the following optimisation can be made:

```
cache.getAdvancedCache().withFlags(Flag.SKIP_CACHE_LOAD).put(key, value);
```

Note that in this case the value returned by `cache.put()` is not reliable. This optimization skips a cache store read and can have very significant performance improvement effects.

For more information, check out our Performance Guide.

More flags are described at Per-Invocation Flags

2.3. How to speed up Infinispan?

Have a look at our Performance Guide.

2.4. Locking and Transaction questions

2.4.1. Does Infinispan support distributed eager locking?

Yes it does. By default, transactions are optimistic, and locks are only acquired during the prepare phase. However, Infinispan can be configured to lock cache keys eagerly, by using the pessimistic locking mode:

```
ConfigurationBuilder builder = new ConfigurationBuilder();
builder.transaction().lockingMode(LockingMode.PESSIMISTIC);
```

With pessimistic locking, Infinispan will implicitly acquire locks when a transaction modifies one or more keys:

```
tm.begin()
cache.put(K,V)    // acquire cluster-wide lock on K
cache.put(K2,V2)  // acquire cluster-wide lock on K2
cache.put(K,V5)   // no-op, we already own cluster wide lock for K
tm.commit()       // releases locks
```

2.4.2. How does Infinispan support explicit eager locking?

When the cache is configured with pessimistic locking, the `lock(K…)` method allows cache users to explicitly lock set of cache keys eagerly during a transaction. Lock call attempts to lock specified cache keys on the proper lock owners and it either succeeds or fails. All locks are released during commit or rollback phase.

```
tm.begin()
cache.getAdvancedCache().lock(K)  // acquire cluster-wide lock on K
cache.put(K,V5)                   // guaranteed to succeed
tm.commit()                       // releases locks
```

2.4.3. What isolation levels does Infinispan support?

Infinispan only supports the isolation levels **READ_COMMITTED** and **REPEATABLE_READ**. Note that exact definition of these levels may differ from traditional database definitions.

The default isolation mode is **READ_COMMITTED**. We consider **READ_COMMITTED** to be good enough for most applications and hence its use as a default.

2.4.4. When using Atomikos transaction manager, distributed caches are not distributing data, what is the problem?

For efficiency reasons, Atomikos transaction manager commits transactions in a separate thread to the thread making the cache operations and until 4.2.1.CR1, Infinispan had problems with this type of scenarios and resulted on distributed caches not sending data to other nodes (see ISPN-927 for more details). Please note that replicated, invalidated or local caches would work fine. It's only distributed caches that would suffer this problem.

There're two ways to get around this issue, either:

1. Upgrade to Infinispan 4.2.1.CR2 or higher where the issue has been fixed.
2. If using Infinispan 4.2.1.CR1 or earlier, configure Atomikos so that `com.atomikos.icatch.threaded_2pc` is set to false . This results in commits happening in the same thread that made the cache operations.

2.5. Eviction and Expiration questions

2.5.1. Expiration does not work, what is the problem?

Multiple cache operations such as `put()` can take a lifespan as parameter which defines the time when the entry should be expired. If you have no eviction configured and and you let this time expire, it can look as Infinispan has not removed the entry. For example, the JMX stats such as number of entries might not updated or the persistent store associated with Infinispan might still contain the entry. To understand what's happening, it's important to note that Infinispan has marked the entry as expired but has not actually removed it. Removal of *expired* entries happens in one of 2 ways:

1. You try and do a get() or containsKey() for that entry. The entry is then detected as expired and is removed.
2. You have enabled eviction and an eviction thread wakes up periodically and purges expired entries.

If you have not enabled (2), or your eviction thread wakeup interval is large and you probe jconsole before the eviction thread kicks in, you will still see the expired entry. You can be assured that if you tried to *retrieve* the entry via a get() or containsKey() though, you won't see the entry (and the entry will be removed).

2.6. Cache Manager questions

2.6.1. Can I create caches using different cache modes using the same cache manager?

Yes. You can create caches using different cache modes, both synchronous and asynchronous,

using the same cache manager.

2.6.2. Can transactions span different Cache instances from the same cache manager?

Yes. Each cache behaves as a separate, standalone JTA resource. Internally though, components may be shared as an optimization but this in no way affects how the caches interact with a JTA manager.

2.6.3. How does multi-tenancy work?

Multi-tenancy is achieved by namespacing. A single Infinispan cluster can have several named caches (attached to the same CacheManager), and different named caches can have duplicate keys. So this is, in effect, multi-tenancy for your key/value store.

2.6.4. Infinispan allows me to create several Caches from a single CacheManager. Are there any reasons to create separate CacheManagers?

As far as possible, internal components are shared between Cache instances. Notably, RPC and networking components are shared. If you need caches that have different network characteristics - such as one cache using TCP while another uses UDP - we recommend you create these using different cache managers.

2.7. Cache Mode questions

2.7.1. What is the difference between a replicated cache and a distributed cache?

Distribution is a new cache mode in Infinispan, in addition to replication and invalidation. In a replicated cache all nodes in a cluster hold all keys i.e. if a key exists on one node, it will also exist on *all* other nodes. In a distributed cache, a number of copies are maintained to provide redundancy and fault tolerance, however this is typically far fewer than the number of nodes in the cluster. A distributed cache provides a far greater degree of scalability than a replicated cache.

A distributed cache is also able to transparently locate keys across a cluster, and provides an L1 cache for fast local read access of state that is stored remotely. You can read more in the relevant User Guide chapter.

2.7.2. Does DIST support both synchronous and asynchronous communications?

Officially, no. And unofficially, yes. Here's the logic. For certain public API methods to have meaningful return values (i.e., to stick to the interface contracts), if you are using DIST , synchronized communications are necessary. For example, you have 3 caches in a cluster, A, B and C. Key K maps to A and B. On C, you perform an operation that requires a return value e.g., Cache.remove(K) . For this to work, the call needs to be forwarded to A and B *synchronously*, and would have to wait for the result from either A or B to return to the caller. If communications were

asynchronous, the return values cannot be guaranteed to be useful - even though the operation would behave as expected.

Now unofficially, we will add a configuration option to allow you to set your cache mode to DIST *and* use asynchronous communications, but this would be an additional configuration option (perhaps something like break_api_contracts) so that users are aware of what they are getting into.

2.7.3. I notice that when using DIST, the cache does a remote get before a write command. Why is this?

Certain methods, such as Cache.put() , are supposed to return the previous value associated with the specified key according to the java.util.Map contract. If this is performed on an instance that does *not* own the key in question and the key is not in L1 cache, the only way to reliably provide this return value is to do a remote GET before the put. This GET is *always* sync (regardless of whether the cache is configured to be sync or async) since we need to wait for that return value.

Isn't that expensive? How can I optimize this away?

It isn't as expensive as it sounds. A remote GET, although sync, will *not* wait for all responses. It will accept the first valid response and move on, thus making its performance has no relation to cluster size.

If you feel your code has no need for these return values, then this can be disabled completely (by specifying the `<unsafe unreliableReturnValues="true" />` configuration element for a cache-wide setting or the `Flag.SKIP_REMOTE_LOOKUP` for a per-invocation setting). Note that while this will *not* impair cache operations and accurate functioning of all public methods is still maintained. However, it *will* break the java.util.Map interface contract by providing unreliable and inaccurate return values to certain methods, so you would need to be certain that your code does not use these return values for anything useful.

2.7.4. I use a clustered cache. I want the guarantees of synchronous replication with the parallelism of asynchronous replication. What can I do?

Infinispan offers a new async API to provide just this. These async methods return Future which can be queried, causing the thread to block till you get a confirmation that any network calls succeeded. You can read more about it .

2.7.5. What is the L1 cache?

An L1 cache (disabled by default) only exists if you set your cache mode to distribution. An L1 cache prevents unnecessary remote fetching of entries mapped to remote caches by storing them locally for a short time after the first time they are accessed. By default, entries in L1 have a lifespan of 60,000 milliseconds (though you can configure how long L1 entries are cached for). L1 entries are also invalidated when the entry is changed elsewhere in the cluster so you are sure you don't have stale entries cached in L1. Caches with L1 enabled will consult the L1 cache before fetching an entry from a remote cache.

2.7.6. What consistency guarantees do I have with different Asynchronous processing settings ?

There are 3 main configuration settings (modes of usage) that affect the behaviour of Infinispan in terms of Asynchronous processing, summarized in the following table:

Config / Mode of usage	Description
API	Usage of Asynchronous API , i.e. methods of the Cache interface like e.g. putAsync(key, val)
Replication	Configuring a clustered cache to replicate data asychronously. In Infinispan XML configuration this is done by using <sync> or <async> sub-elements under <clustering> element.

Switching to asynchronous mode in each of these areas causes loss of some consistency guarantees. The known problems are summarised here:

API	Replication	Marshalling	Consistency problems
Sync	Sync	Sync	
Sync	*Async*	Sync	*1* - Cache entry is replicated with a delay or not at all in case of network error. *2* - Node where the operation originated won't be notified about errors that happened on network or on the receiving side.
Sync	*Async*	*Async*	*1, 2 3* - Calling order of sync API method might not be preserved – depends on which operation finishes marshalling first in the asyncExecutor *4* - Replication of put operation can be applied on different nodes in different order – this may result in inconsistent values
Async	Sync	Sync	*3*
Async	*Async*	Sync	*1, 2, 3*
Async	*Async*	*Async*	*1, 2, 3, 4*

2.7.7. Grouping API vs Key Affinity Service

The key affinity (for keys generated with the Key Affinity Service) might be lost during topology changes. E.g. if k1 maps to node N1 and another node is added to the system, k1 can me migrated to N2 (affinity is lost). With grouping API you have the guarantee that the same node (you don't know/control which node) hosts all the data from the same group even after topology changes.

2.8. Listener questions

2.8.1. In a cache entry modified listener, can the modified value be retrieved via Cache.get() when isPre=false?

No, it cannot. Use `CacheEntryModifiedEvent.getValue()` to retrieve the value of the entry that was modified.

2.8.2. When annotating a method with CacheEntryCreated, how do I retrieve the value of the cache entry added?

Use `CacheEntryCreatedEvent.getValue()` to retrieve the value of the entry.

2.8.3. What is the difference between classes in `org.infinispan.notifications.cachelistener.filter` vs `org.infinispan.filter`?

Inside these packages you'll find classes that facilitate filtering and data conversion. The difference is that classes in `org.infinispan.filter` are used for filtering and conversion in multiple areas, such as cache loaders, entry iterators,...etc, whereas classes in `org.infinispan.notifications.cachelistener.filter` are purely used for listener event filtering, and provide more information than similarly named classes in `org.infinispan.filter`. More specifically, remote listener event filtering and conversion require `CacheEventFilter` and `CacheEventConverter` instances located in `org.infinispan.notifications.cachelistener.filter` package to be used.

2.9. IaaS/Cloud Infrastructure questions

2.9.1. How do you make Infinispan send replication traffic over a specific network when you don't know the IP address?

Some cloud providers charge you less for traffic over internal IP addresses compared to public IP addresses, in fact, some cloud providers do not even charge a thing for traffic over the internal network (i.e. GoGrid). In these circumstances, it's really advantageous to configure Infinispan in such way that replication traffic is sent via the internal network. The problem though is that quite often you don't know which internal IP address you'll be assigned (unless you use elastic IPs and dyndns.org), so how do you configure Infinispan to cope with those situations?

JGroups, which is the underlying group communication library to interconnect Infinispan instances, has come up with a way to enable users to bind to a type of address rather than to a specific IP address. So now you can configure `bind_addr` property in JGroups configuration file, or the `-Djgroups.bind_addr` system property to a keyword rather than a dotted decimal or symbolic IP

address:

- GLOBAL : pick a public IP address. You want to avoid this for replication traffic
- SITE_LOCAL : use a private IP address, e.g. 192.168.x.x. This avoids charges for bandwidth from GoGrid, for example
- LINK_LOCAL : use a 169.x.x.x, 254.0.0.0 address. I've never used this, but this would be for traffic only within 1 box
- NON_LOOPBACK : use the first address found on an interface (which is up), which is not a 127.x.x.x address

2.10. Demo questions

2.10.1. When using the GUI Demo, I've just put an entry in the cache with lifespan of -1. Why do I see it as having a lifespan of 60,000?

This is probably a L1 caching event. When you put an entry in the cache, the entry is mapped to specific nodes in a cluster using a consistent hashing algorithm. This means that key K could map on to caches A and B (or however many owners you have configured). If you happen to have done the cache.put(K, V) on cache C , however, K still maps to A and B (and will be added to caches A and B with their proper lifespans), but it will also be put in cache C's L1 cache.

2.11. Logging questions

2.11.1. How can I enable logging?

By default Infinispan uses JBoss Logging 3.0 as logging framework. JBoss Logging acts as a delegator to either JBoss Log Manager, Apache Log4j, Slf4j or JDK Logging. The way it chooses which logging provider to delegate to is by:

1. checking whether the JBoss Log Manager is configured (e.g. Infinispan is running in JBoss Application Server 7) and if it is, using it
2. otherwise, checking if Apache Log4j is in the classpath (JBoss Logging checks if the classes org.apache.log4j.LogManager and org.apache.log4j.Hierarchy are available) and if it is, using it
3. otherwise, checking if LogBack in the classpath (JBoss Logging checks if the class ch.qos.logback.classic.Logger is available) and if it is, using it
4. finally, if none of the above are available, using JDK logging

You can use this log4j2.xml as base for any Infinispan related logging, and you can pass it to your system via system parameter (e.g., `-Dlog4j.configurationFile=file:/path/to/log4j2.xml`).

2.12. Third Party Container questions

2.12.1. Can I use Infinispan on Google App Engine for Java?

Not at this moment. Due to GAE/J restricting classes that can be loaded, and restrictions around use

of threads, Infinispan will not work on GAE/J. However, we do plan to fix this - if you wish to track the progress of Infinispan on GAE/J, have a look at ISPN-57 .

2.12.2. When running on Glassfish or Apache, creating a cache throws an exception saying "Unable to construct a GlobalComponentRegistry", what is it wrong?

It appears that this happens due to some classloading issue. A workaround that is know to work is to call the following before creating the cache manager or container:

```
Thread.currentThread().setContextClassLoader(this.getClass().getClassLoader());
```

2.13. Marshalling and Unmarshalling

2.13.1. Best practices implementing java.io.Externalizable

If you decide to implement Externalizable interface, please make sure that the readExternal() method is thread safe, otherwise you run the risk of potential getting corrupted data and OutOfMemoryException , as seen in this forum post .

2.13.2. Does Infinispan support storing Non-Serializable objects?

See the User Guide's chapter on marshalling for more information.

2.13.3. Do Externalizer implementations need to access internal Externalizer implementations?

No, they don't. Here's an example of what should not be done:

```
public static class ABCMarshallingExternalizer implements AdvancedExternalizer
<ABCMarshalling> {
   @Override
   public void writeObject(ObjectOutput output, ABCMarshalling object) throws
IOException {
      MapExternalizer ma = new MapExternalizer();
      ma.writeObject(output, object.getMap());
   }

   @Override
   public ABCMarshalling readObject(ObjectInput input) throws IOException,
ClassNotFoundException {
      ABCMarshalling hi = new ABCMarshalling();
      MapExternalizer ma = new MapExternalizer();
      hi.setMap((ConcurrentHashMap<Long, Long>) ma.readObject(input));
      return hi;
   }

   ...
}
```

End user externalizers should not need to fiddle with Infinispan internal externalizer classes. Instead, this code should have been written as:

```
public static class ABCMarshallingExternalizer implements AdvancedExternalizer
<ABCMarshalling> {
   @Override
   public void writeObject(ObjectOutput output, ABCMarshalling object) throws
IOException {
      output.writeObject(object.getMap());
   }

   @Override
   public ABCMarshalling readObject(ObjectInput input) throws IOException,
ClassNotFoundException {
      ABCMarshalling hi = new ABCMarshalling();
      hi.setMap((ConcurrentHashMap<Long, Long>) input.readObject());
      return hi;
   }

   ...
}
```

2.13.4. During state transfer, the state receiver logs an EOFException when applying state saying "Read past end of file". Should I worry about this?

It depends on whether the state provider encountered an error or not when generating the state. For example, sometimes the state provider might already be providing state to another node, so

when the node requests the state, the state generator might log:

```
2010-12-09 10:26:21,533 20267 ERROR
[org.infinispan.remoting.transport.jgroups.JGroupsTransport]
(STREAMING_STATE_TRANSFER-sender-1,Infinispan-Cluster,NodeJ-2368:) Caught while
responding to state transfer request
org.infinispan.statetransfer.StateTransferException:
java.util.concurrent.TimeoutException: Could not obtain exclusive processing lock
     at
org.infinispan.statetransfer.StateTransferManagerImpl.generateState(StateTransferManag
erImpl.java:175)
     at
org.infinispan.remoting.InboundInvocationHandlerImpl.generateState(InboundInvocationHa
ndlerImpl.java:119)
     at
org.infinispan.remoting.transport.jgroups.JGroupsTransport.getState(JGroupsTransport.j
ava:586)
     at
org.jgroups.blocks.MessageDispatcher$ProtocolAdapter.handleUpEvent(MessageDispatcher.j
ava:691)
     at
org.jgroups.blocks.MessageDispatcher$ProtocolAdapter.up(MessageDispatcher.java:772)
     at org.jgroups.JChannel.up(JChannel.java:1465)
     at org.jgroups.stack.ProtocolStack.up(ProtocolStack.java:954)
     at org.jgroups.protocols.pbcast.FLUSH.up(FLUSH.java:478)
     at
org.jgroups.protocols.pbcast.STREAMING_STATE_TRANSFER$StateProviderHandler.process(STR
EAMING_STATE_TRANSFER.java:653)
     at
org.jgroups.protocols.pbcast.STREAMING_STATE_TRANSFER$StateProviderThreadSpawner$1.run
(STREAMING_STATE_TRANSFER.java:582)
     at
java.util.concurrent.ThreadPoolExecutor$Worker.runTask(ThreadPoolExecutor.java:886)
     at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:908)
     at java.lang.Thread.run(Thread.java:680)
Caused by: java.util.concurrent.TimeoutException: Could not obtain exclusive
processing lock
     at
org.infinispan.remoting.transport.jgroups.JGroupsDistSync.acquireProcessingLock(JGroup
sDistSync.java:71)
     at
org.infinispan.statetransfer.StateTransferManagerImpl.generateTransactionLog(StateTran
sferManagerImpl.java:202)
     at
org.infinispan.statetransfer.StateTransferManagerImpl.generateState(StateTransferManag
erImpl.java:165)
     ... 12 more
```

This exception is basically saying that the state generator was not able to generate the transaction

log and so the output to which it was writing is closed. In this situation, it's common to see the state receiver log an EOFException , as shown below, when trying to read the transaction log because the sender did not write the transaction log:

```
2010-12-09 10:26:21,535 20269 TRACE [org.infinispan.marshall.VersionAwareMarshaller]
(Incoming-2,Infinispan-Cluster,NodeI-38030:) Log exception reported
java.io.EOFException: Read past end of file
     at
org.jboss.marshalling.AbstractUnmarshaller.eofOnRead(AbstractUnmarshaller.java:184)
     at
org.jboss.marshalling.AbstractUnmarshaller.readUnsignedByteDirect(AbstractUnmarshaller
.java:319)
     at
org.jboss.marshalling.AbstractUnmarshaller.readUnsignedByte(AbstractUnmarshaller.java:
280)
     at
org.jboss.marshalling.river.RiverUnmarshaller.doReadObject(RiverUnmarshaller.java:207)
     at
org.jboss.marshalling.AbstractUnmarshaller.readObject(AbstractUnmarshaller.java:85)
     at
org.infinispan.marshall.jboss.GenericJBossMarshaller.objectFromObjectStream(GenericJBo
ssMarshaller.java:175)
     at
org.infinispan.marshall.VersionAwareMarshaller.objectFromObjectStream(VersionAwareMars
haller.java:184)
     at
org.infinispan.statetransfer.StateTransferManagerImpl.processCommitLog(StateTransferMa
nagerImpl.java:228)
     at
org.infinispan.statetransfer.StateTransferManagerImpl.applyTransactionLog(StateTransfe
rManagerImpl.java:250)
     at
org.infinispan.statetransfer.StateTransferManagerImpl.applyState(StateTransferManagerI
mpl.java:320)
     at
org.infinispan.remoting.InboundInvocationHandlerImpl.applyState(InboundInvocationHandl
erImpl.java:102)
     at
org.infinispan.remoting.transport.jgroups.JGroupsTransport.setState(JGroupsTransport.j
ava:603)
        ...
```

The current logic is for the state receiver to back off in these scenarios and retry after a few seconds. Quite often, after the retry the state generator might have already finished dealing with the other node and hence the state receiver will be able to fully receive the state.

2.13.5. How do I get more information on marshalling and unmarshalling exceptions?

See the section on troubleshooting marshalling in the User Guide.

2.13.6. Why am I getting invalid data passed to readExternal?

If you are using Cache.putAsync() you may find your object is modified after serialization starts, thus corrupting the datastream passed to readExternal . To solve this, make sure you synchronize access to the object.

Read More

You can read more about this issue in this forum thread .

2.14. Tuning questions

2.14.1. When running Infinispan under load, I see RejectedExecutionException, how can I fix it?

Internally Infinispan uses executors to do some processing asynchronously, so the first thing to do is to figure out which of these executors is causing issues. For example, if you see a stacktrace that looks like this, the problem is located in the asyncTransportExecutor :

```
java.util.concurrent.RejectedExecutionException
  at
java.util.concurrent.ThreadPoolExecutor$AbortPolicy.rejectedExecution(ThreadPoolExecut
or.java:1759)
  at java.util.concurrent.ThreadPoolExecutor.reject(ThreadPoolExecutor.java:767)
  at java.util.concurrent.ThreadPoolExecutor.execute(ThreadPoolExecutor.java:658)
  at
java.util.concurrent.AbstractExecutorService.submit(AbstractExecutorService.java:92)
  at
org.infinispan.remoting.transport.jgroups.CommandAwareRpcDispatcher.invokeRemoteComman
ds(CommandAwareRpcDispatcher.java:117)
...
```

To solve this issue, you should try any of these options:

- Increase the maxThreads property in asyncTransportExecutor . At the time of writing, the default value for this particular executor is 25.
- Define your own ExecutorFactory which creates an executor with a bigger queue. You can find more information about different queueing strategies in ThreadPoolExecutor javadoc .
- Disable async marshalling (see the <async ... > element for details). This would mean that an executor is *not* used when replicating, so you will never have a RejectedExecutionException . However this means each put() will take a little longer since marshalling will now happen on the critical path. The RPC is still async though as the thread won't wait for a response from the

recipient (fire-and-forget).

2.15. JNDI questions

2.15.1. Can I bind Cache or CacheManager to JNDI?

Cache or CacheManager can be bound to JNDI, but only to the java: namespace because they are not designed to be exported outside the Java Virtual Machine. In other words, you shouldn't expect that you'll be able to access them remotely by binding them to JNDI and downloading a remote proxy to them because neither Cache nor CacheManager are serializable.

To find an example on how to bind Cache or CacheManager to the java: namespace, simply check this unit test case .

2.16. Hibernate 2nd Level Cache questions

2.16.1. Can I use Infinispan as a remote JPA or Hibernate second level cache?

See Remote Infinispan Caching section in Hibernate documentation.

2.16.2. Is it possible to use the Infinispan 2nd level cache outside of a J2EE server, and if so how do I set up the transaction manager lookup?

The User Guide provides details on configuring a transaction manager outside of Java EE. The User Guide also provides details on how to use Atomikos, JTOM and Bitronix.

2.16.3. What are the pitfalls of not using a non-JTA transaction factory such as JDBCTransactionFactory with Hibernate when Infinispan is used as 2nd level cache provider?

The problem is that Hibernate will create a Transaction instance via java.sql.Connection and Infinispan will create a transaction via whatever TransactionManager returned by hibernate.transaction.manager_lookup_class . If hibernate.transaction.manager_lookup_class has not been populated, it will default to the dummy transaction manager.

So, any work on the 2nd level cache will be done under a different transaction to the one used to commit the stuff to the database via Hibernate. In other words, your operations on the database and the 2LC are not treated as a single unit. Risks here include failures to update the 2LC leaving it with stale data while the database committed data correctly.

2.17. Cache Server questions

2.17.1. After running a Hot Rod server for a while, I get a NullPointerException in HotRodEncoder.getTopologyResponse(), how can I get around it?

This is a bug (see ISPN-669) in the Hot Rod code where we didn't specifically set the topology cache to have no eviction and no expiration. So, if someone configured the default cache in the Infinispan configuration file for Hot Rod with expiration or eviction, the topology cache would end up having those capabilities and the topology view could after a while be removed from memory. To get around this issue either:

- Avoid having expiration and eviction on for the default cache.
- Or, make sure you create a namedCache for `___hotRodTopologyCache` with sync replication, state transfer, no expiration and no eviction.

2.17.2. Is there a way to do a Bulk Get on a remote cache?

There's no bulk get operation in Hot Rod, but the Java Hot Rod client has implemented via RemoteCache the getAsync() operation, which returns a org.infinispan.util.concurrent.NotifyingFuture (extends java.util.concurrent.Future). So, if you want to retrieve multiple keys in parallel, just call multiple times *getAsync()* and when you need the values, just call *Future.get()* , or attach a *FutureListener* to the *NotifyingFuture* to get notified when the value is ready.

2.17.3. What is the startServer.sh script used for? What is the startServer.bat script used for?

These scripts were used to start Infinispan server instances in earlier Infinispan versions, but this is not the case any more since the Infinispan Server modules are built into a base Wildfly/EAP instance, allowing all server modules to interact with other base services provided by Wildfly/EAP, e.g. the web container for REST server. Check the dedicated Infinispan Server guide to find out more on how to start it.

2.18. Debugging questions

2.18.1. How can I get Infinispan to show the full byte array? The log only shows partial contents of byte arrays...

Since version 4.1, whenever Infinispan needs to print byte arrays to logs, these are partially printed in order to avoid unnecessarily printing potentially big byte arrays. This happens in situations where either, Infinispan caches have been configured with lazy deserialization, or your running an Memcached or Hot Rod server. So in these cases, only the first 10 positions of the byte array are shown in the logs. If you want Infinispan to show the full byte array in the logs, simply pass the `-Dinfinispan.arrays.debug=true` system property at startup. In the future, this might be controllable at runtime via a JMX call or similar.

Here's an example of log message with a partially displayed byte array:

```
2010-04-14 15:46:09,342 TRACE [ReadCommittedEntry] (HotRodWorker-1-1) Updating entry
(key=CacheKey{data=ByteArray{size=19, hashCode=1b3278a,
array=[107, 45, 116, 101, 115, 116, 82, 101, 112, 108, ..]}}
removed=false valid=true changed=true created=true
value=CacheValue{data=ByteArray{size=19,
array=[118, 45, 116, 101, 115, 116, 82, 101, 112, 108, ..]},
version=281483566645249}]
```

And here's a log message where the full byte array is shown:

```
2010-04-14 15:45:00,723 TRACE [ReadCommittedEntry] (Incoming-2,Infinispan-Cluster,eq-
6834) Updating entry
(key=CacheKey{data=ByteArray{size=19, hashCode=6cc2a4,
array=[107, 45, 116, 101, 115, 116, 82, 101, 112, 108, 105, 99, 97, 116, 101, 100, 80,
117, 116]}}
removed=false valid=true changed=true created=true
value=CacheValue{data=ByteArray{size=19,
array=[118, 45, 116, 101, 115, 116, 82, 101, 112, 108, 105, 99, 97, 116, 101, 100, 80,
117, 116]},
version=281483566645249}]
```

2.19. Clustering Transport questions

2.19.1. How do I retrieve the clustering physical address?

You can retrieve the physical address via `AdvancedCache.getRpcManager().getTransport().getPhysicalAddresses()`