Embedding Infinispan 10.0

Table of Contents

1. Using Infinispan as an embedded cache in Java SE . . . 1
   1.1. Creating a new Infinispan project . . . 1
      1.1.1. Maven users . . . 1
      1.1.2. Ant users . . . 1
   1.2. Running Infinispan on a single node . . . 1
   1.3. Use the default cache . . . 2
   1.4. Use a custom cache . . . 3
2. Using Infinispan as an embedded data grid in Java SE . . . 5
   2.1. Sharing JGroups channels . . . 5
   2.2. Running Infinispan in a cluster . . . 5
      2.2.1. Replicated mode . . . 5
      2.2.2. Distributed mode . . . 6
   2.3. clustered-cache quickstart architecture . . . 6
      2.3.1. Logging changes to the cache . . . 6
      2.3.2. What's going on? . . . 7
   2.4. Configuring the cluster . . . 8
      2.4.1. Tweaking the cluster configuration for your network . . . 8
   2.5. Configuring a replicated data-grid . . . 9
   2.6. Configuring a distributed data-grid . . . 10

Chapter 1. Using Infinispan as an embedded cache in Java SE

Running Infinispan in embedded mode is very easy. First, we'll set up a project, and then we'll run Infinispan, and start adding data.

embedded-cache quickstart

All the code discussed in this tutorial is available in the embedded-cache quickstart.

1.1. Creating a new Infinispan project

The only thing you need to set up Infinispan is add it's dependencies to your project.

1.1.1. Maven users

If you are using Maven (or another build system like Gradle or Ivy which can use Maven dependencies), then this is easy. Just add the following to the `<dependencies>` section of your `pom.xml`:

pom.xml

```
<dependency>
  <groupId>org.infinispan</groupId>
  <artifactId>infinispan-embedded</artifactId>
  <!-- Replace ${version.infinispan} with the
  version of Infinispan that you're using. -->
  <version>${version.infinispan}</version>
</dependency>
```

Which version of Infinispan should I use?

We recommend using the latest stable version of Infinispan. All releases are displayed on the downloads page.

Alternatively, you can use the POM from the quickstart that accompanies this tutorial.

1.1.2. Ant users

If you are using Ant, or another build system which doesn't provide declarative dependency management, then the Infinispan distribution zip contains a lib/ directory. Add the contents of this to the build classpath.

1.2. Running Infinispan on a single node

In order to run Infinispan, we're going to create a `main()` method in the `Quickstart` class. Infinispan comes configured to run out of the box; once you have set up your dependencies, all you need to do

to start using Infinispan is to create a new cache manager and get a handle on the default cache.

Quickstart.java

```
public class Quickstart {

   public static void main(String args[]) throws Exception {
      Cache<Object, Object> c = new DefaultCacheManager().getCache();
   }

}
```

We now need a way to run the main method! To run the Quickstart main class: If you are using Maven:

```
$ mvn compile exec:java
-Dexec.mainClass="org.infinispan.quickstart.embeddedcache.Quickstart"
```

You should see Infinispan start up, and the version in use logged to the console.

Congratulations, you now have Infinispan running as a local cache!

1.3. Use the default cache

Infinispan exposes a Map-like, JSR-107-esque interface for accessing and mutating the data stored in the cache. For example:

DefaultCacheQuickstart.java

```
// Add a entry
cache.put("key", "value");
// Validate the entry is now in the cache
assertEqual(1, cache.size());
assertTrue(cache.containsKey("key"));
// Remove the entry from the cache
Object v = cache.remove("key");
// Validate the entry is no longer in the cache
assertEqual("value", v);
```

Infinispan offers a thread-safe data-structure:

DefaultCacheQuickstart.java

```
// Add an entry with the key "key"
cache.put("key", "value");
// And replace it if missing
cache.putIfAbsent("key", "newValue");
// Validate that the new value was not added
```

By default entries are immortal but you can override this on a per-key basis and provide lifespans.

DefaultCacheQuickstart.java

```
//By default entries are immortal but we can override this on a per-key basis and
provide lifespans.
cache.put("key", "value", 5, SECONDS);
assertTrue(cache.containsKey("key"));
Thread.sleep(10000);
```

to run using maven:

```
$ mvn compile exec:java
-Dexec.mainClass="org.infinispan.quickstart.embeddedcache.DefaultCacheQuickstart"
```

1.4. Use a custom cache

Each cache in Infinispan can offer a different set of features (for example transaction support, different replication modes or support for eviction), and you may want to use different caches for different classes of data in your application. To get a custom cache, you need to register it with the manager first:

CustomCacheQuickstart.java

```
public static void main(String args[]) throws Exception {
   EmbeddedCacheManager manager = new DefaultCacheManager();
   manager.defineConfiguration("custom-cache", new ConfigurationBuilder()
             .eviction().strategy(LIRS).maxEntries(10)
             .build());
   Cache<Object, Object> c = manager.getCache("custom-cache");
}
```

The example above uses Infinispan's fluent configuration, which offers the ability to configure your cache programmatically. However, should you prefer to use XML, then you may. We can create an identical cache to the one created with a programmatic configuration:

To run using maven:

```
$ mvn compile exec:java
-Dexec.mainClass="org.infinispan.quickstart.embeddedcache.CustomCacheQuickstart"
```

infinispan.xml

```
<infinispan
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:config:10.0
http://www.infinispan.org/schemas/infinispan-config-10.0.xsd"
        xmlns="urn:infinispan:config:10.0">

   <cache-container default-cache="default">
       <local-cache name="xml-configured-cache">
          <memory>
            <object size="100"/>
          </memory>
       </local-cache>
   </cache-container>

</infinispan>
```

We then need to load the configuration file, and use the programmatically defined cache:

XmlConfiguredCacheQuickstart.java

```
public static void main(String args[]) throws Exception {
    Cache<Object, Object> c = new DefaultCacheManager("infinispan.xml").getCache("xml-
configured-cache");
}
```

To run using maven:

```
$ mvn compile exec:java
-Dexec.mainClass="org.infinispan.quickstart.embeddedcache.XmlConfiguredCacheQuickstart
"
```

Chapter 2. Using Infinispan as an embedded data grid in Java SE

Clustering Infinispan is simple. Under the covers, Infinispan uses JGroups as a network transport, and JGroups handles all the hard work of forming a cluster.

clustered-cache quickstart

All the code discussed in this tutorial is available in the clustered-cache quickstart.

2.1. Sharing JGroups channels

By default all caches created from a single CacheManager share the same JGroups channel and multiplex RPC messages over it. In this example caches 1, 2 and 3 all use the same JGroups channel.

```
EmbeddedCacheManager cm = new DefaultCacheManager("infinispan.xml");
Cache<Object, Object> replSyncCache = cm.getCache("replSyncCache");
Cache<Object, Object> replAsyncCache = cm.getCache("replAsyncCache");
Cache<Object, Object> invalidationSyncCache = cm.getCache("invalidationSyncCache");
```

2.2. Running Infinispan in a cluster

It is easy set up a clustered cache. This tutorial will show you how to create two nodes in different processes on the same local machine. The quickstart follows the same structure as the embedded-cache quickstart, using Maven to compile the project, and a main method to launch the node.

If you are following along with the quickstarts, you can try the examples out.

The quickstart defines two clustered caches, one in *replication mode* and one *distribution mode*.

2.2.1. Replicated mode

To run the example in replication mode, we need to launch two nodes from different consoles. For the first node:

```
$ mvn exec:java -Djava.net.preferIPv4Stack=true -Djgroups.bind_addr=127.0.0.1
-Dexec.mainClass="org.infinispan.quickstart.clusteredcache.Node" -Dexec.args="A"
```

And for the second node:

```
$ mvn exec:java -Djava.net.preferIPv4Stack=true -Djgroups.bind_addr=127.0.0.1
-Dexec.mainClass="org.infinispan.quickstart.clusteredcache.Node" -Dexec.args="B"
```

Note: You need to set `-Djava.net.preferIPv4Stack=true` because the JGroups configuration uses IPv4

multicast address. Normally you should not need `-Djgroups.bind_addr=127.0.0.1`, but many wireless routers do not relay IP multicast by default.

Each node will insert or update an entry every second, and it will log any changes.

2.2.2. Distributed mode

To run the example in distribution mode and see how entries are replicated to only two nodes, we need to launch three nodes from different consoles. For the first node:

```
$ mvn compile exec:java -Djava.net.preferIPv4Stack=true
-Dexec.mainClass="org.infinispan.quickstart.clusteredcache.Node" -Dexec.args="-d A"
```

For the second node:

```
$ mvn compile exec:java -Djava.net.preferIPv4Stack=true
-Dexec.mainClass="org.infinispan.quickstart.clusteredcache.Node" -Dexec.args="-d B"
```

And for the third node:

```
$ mvn compile exec:java -Djava.net.preferIPv4Stack=true
-Dexec.mainClass="org.infinispan.quickstart.clusteredcache.Node" -Dexec.args="-d C"
```

The same as in replication mode, each node will insert or update an entry every second, and it will log any changes. But unlike in replication mode, not every node will see every modification.

You can also see that each node holds a different set of entries by pressing Enter.

2.3. clustered-cache quickstart architecture

2.3.1. Logging changes to the cache

An easy way to see what is going on with your cache is to log mutated entries. An Infinispan listener is notified of any mutations:

```
import org.infinispan.notifications.Listener;
import org.infinispan.notifications.cachelistener.annotation.*;
import org.infinispan.notifications.cachelistener.event.*;
import org.jboss.logging.Logger;

@Listener
public class LoggingListener {

   private BasicLogger log = BasicLogFactory.getLog(LoggingListener.class);

   @CacheEntryCreated
   public void observeAdd(CacheEntryCreatedEvent<String, String> event) {
      if (event.isPre())
         return;
      log.infof("Cache entry %s = %s added in cache %s", event.getKey(), event
.getValue(), event.getCache());
   }

   @CacheEntryModified
   public void observeUpdate(CacheEntryModifiedEvent<String, String> event) {
      if (event.isPre())
         return;
      log.infof("Cache entry %s = %s modified in cache %s", event.getKey(), event
.getValue(), event.getCache());
   }

   @CacheEntryRemoved
   public void observeRemove(CacheEntryRemovedEvent<String, String> event) {
      if (event.isPre())
         return;
      log.infof("Cache entry %s removed in cache %s", event.getKey(), event.getCache(
));
   }
}
```

Listeners methods are declared using annotations, and receive a payload which contains metadata about the notification. Listeners are notified of any changes. Here, the listeners simply log any entries added, modified, or removed.

2.3.2. What's going on?

The example allows you to start two or more nodes, each of which are started in a separate process. The node code is very simple, each node starts up, prints the local cache contents, registers a listener that logs any changes, and starts storing entries of the form `key-<counter> = <local address>-counter`.

State transfer

Infinispan automatically replicates the cache contents from the existing members to joining members. This can be controlled in two ways:

- If you don't want the `getCache()` call to block until the entire cache is transferred, you can configure `clustering.stateTransfer.awaitInitialTransfer = false`. Note that `cache.get(key)` will still return the correct value, even before the state transfer is finished.
- If it's fast enough to re-create the cache entries from another source, you can disable state transfer completely, by configuring `clustering.stateTransfer.fetchInMemoryState = false`.

2.4. Configuring the cluster

First, we need to ensure that the cache manager is cluster aware. Infinispan provides a default configuration for a clustered cache manager:

```
GlobalConfigurationBuilder.getClusteredDefault().build()
```

2.4.1. Tweaking the cluster configuration for your network

Depending on your network setup, you may need to tweak your JGroups set up. JGroups is configured via an XML file; the file to use can be specified via the GlobalConfiguration:

```
DefaultCacheManager cacheManager = new DefaultCacheManager(
      GlobalConfigurationBuilder.defaultClusteredBuilder()
            .transport().nodeName(nodeName).addProperty("configurationFile",
"jgroups.xml")
            .build()
);
```

The JGroups documentation provides extensive advice on getting JGroups working on your network. If you are new to configuring JGroups, you may get a little lost, so you might want to try tweaking these configuration parameters:

- Using the system property `-Djgroups.bind_addr=127.0.0.1` causes JGroups to bind only to your loopback interface, meaning any firewall you may have configured won't get in the way. Very useful for testing a cluster where all nodes are on one machine.

You can also configure the JGroups configuration to use in Infinispan's XML configuration:

```
<infinispan>
  <jgroups>
     <!-- Add custom JGroups stacks in external files. -->
     <stack-file name="prod-tcp" path="prod-jgroups-tcp.xml"/>
  </jgroups>
  <cache-container default-cache="replicatedCache">
    <!-- Add custom JGroups stacks to clustered caches. -->
    <transport stack="prod-tcp" />
    <replicated-cache name="replicatedCache"/>
  </cache-container>
  ...
</infinispan>
```

2.5. Configuring a replicated data-grid

In replicated mode, Infinispan will store every entry on every node in the grid. This offers high durability and availability of data, but means the storage capacity is limited by the available heap space on the node with least memory. The cache should be configured to work in replication mode (either synchronous or asynchronous), and can otherwise be configured as normal. For example, if you want to configure the cache programmatically:

```
cacheManager.defineConfiguration("repl", new ConfigurationBuilder()
      .clustering()
      .cacheMode(CacheMode.REPL_SYNC)
      .build()
);
```

You can configure an identical cache using XML:

infinispan-replication.xml

```
<infinispan>
  <jgroups/>
  <cache-container default-cache="repl">
     <transport/>
     <replicated-cache name="repl" mode="SYNC" />
  </cache-container>
</infinispan>
```

along with

```
private static EmbeddedCacheManager createCacheManagerFromXml() throws IOException {
   return new DefaultCacheManager("infinispan-replication.xml");
}
```

2.6. Configuring a distributed data-grid

In distributed mode, Infinispan will store every entry on a subset of the nodes in the grid (the parameter numOwners controls how many owners each entry will have). Compared to replication, distribution offers increased storage capacity, but with increased latency to access data from non-owner nodes, and durability (data may be lost if all the owners are stopped in a short time interval). Adjusting the number of owners allows you to obtain the trade off between space, durability, and latency.

Infinispan also offers a *topology aware consistent hash* which will ensure that the owners of entries are located in different data centers, racks, or physical machines, to offer improved durability in case of node crashes or network outages.

The cache should be configured to work in distributed mode (either synchronous or asynchronous), and can otherwise be configured as normal. For example, if you want to configure the cache programmatically:

```
cacheManager.defineConfiguration("dist", new ConfigurationBuilder()
      .clustering()
      .cacheMode(CacheMode.DIST_SYNC)
      .hash().numOwners(2)
      .build()
);
```

You can configure an identical cache using XML:

infinispan-distribution.xml:

```
<infinispan>
  <jgroups/>
  <cache-container default-cache="repl">
    <transport/>
    <distributed-cache owners="2" mode="SYNC" />
  </cache-container>
</infinispan>
```

along with

```
private static EmbeddedCacheManager createCacheManagerFromXml() throws IOException {
   return new DefaultCacheManager("infinispan-distribution.xml");
}
```