Infinispan and JBoss AS 7

In this book, we have covered in detail the configuration of native Infinispan servers. The Infinispan API is also embedded into the latest release of JBoss Application Server (7.1 at the time of writing). In this appendix, we will cover the following topics:

- At first we will shortly introduce shortly the new application server platform
- Next we will show how you can configure and develop applications using Infinispan API on a JBoss AS 7 server.

The new modular application server

JBoss AS has changed a lot with the latest distribution. The new application server has improved in many areas, including lower memory footprint, lightning fast startup, true classloading isolation (between built-in modules and modules delivered by developers), and excellent management of resources with the addition of domain controllers.

We will shortly illustrate how Infinispan platform fits into this new picture. Should you need to know all the core details of the AS 7 architecture, you might consider looking for a copy of *JBoss AS 7 Configuration, Deployment and Administration* (`http://www.packtpub.com/jboss-as-7-configuration-deployment-administration/book`), authored by me a couple of months ago.

In a nutshell, JBoss AS 7 is composed of a set of modules that provide the basic server functionalities. The configuration of modules is not spread in a set of single XML files anymore, but it is centralized into a single file. Thus, every configuration file holds a single server configuration. A server configuration can be in turn based on a set of standalone servers or domain servers. The main difference between standalone servers and domain servers encompasses the management area; as a matter of fact, domain-based servers can be managed from a centralized point (the domain controller) while, on the other hand, standalone servers are independent server units, each one managing its own configuration.

The Infinispan configuration in AS 7

The Infinispan subsystem is included in several built-in AS 7 configurations. Each one of these configurations, in turn, covers a different use case:

Configuration	Use case
`standalone.xml`	Used by non-clustered applications requiring Infinispan API and running on standalone JBoss AS instances.
`standalone-ha.xml` / `standalone-full-ha.xml`	Used by clustered applications requiring Infinispan API and running on standalone JBoss AS instances.
`domain.xml`	Used by single nodes or clustered applications requiring Infinispan API and running on a domain of JBoss AS instances.

So, as you can see, Infinispan can be used both to achieve clustered services and as a data grid, just as we have shown in the earlier examples of this book.

The configuration of Infinispan is, however, slightly different from the native platform configuration that we have learnt. As a matter of fact, when used as a module of the AS 7 architecture, Infinispan needs to comply with the core server configuration. So, let's see what the Infinispan subsystem configuration looks like:

```
<subsystem
  xmlns="urn:jboss:domain:infinispan:1.1"
  default-cache-container="cluster">

  <cache-container
    name="cluster" aliases="ha-partition"
    default-cache="default">
    <transport lock-timeout="60000"/>
    <replicated-cache name="default"
      mode="SYNC" batching="true">
      <locking isolation="REPEATABLE_READ"/>
    </replicated-cache>
  </cache-container>

  <cache-container name="web" aliases="standard-session-cache"
default-cache="repl">
    <transport lock-timeout="60000"/>
    <replicated-cache name="repl" mode="ASYNC" batching="true">
      <file-store/>
    </replicated-cache>
    <replicated-cache name="sso" mode="SYNC" batching="true"/>
```

```
      <distributed-cache name="dist" mode="ASYNC" batching="true">
        <file-store/>
      </distributed-cache>
    </cache-container>

    <cache-container name="ejb"
      aliases="sfsb sfsb-cache"
      default-cache="repl">
      <transport lock-timeout="60000"/>
      <replicated-cache name="repl"
        mode="ASYNC"
        batching="true">
        <eviction strategy="LRU"/>
        <file-store/>
      </replicated-cache>

      <replicated-cache name="remote-connector-client-mappings"
        mode="SYNC"
        batching="true"/>
      <distributed-cache name="dist" mode="ASYNC" batching="true">
        <eviction strategy="LRU"/>
        <file-store/>
         </distributed-cache>
    </cache-container>

    <cache-container name="hibernate" default-cache="local-query">
      <transport lock-timeout="60000"/>
      <local-cache name="local-query">
        <transaction mode="NONE"/>
        <eviction strategy="LRU" max-entries="10000"/>
        <expiration max-idle="100000"/>
      </local-cache>
      <invalidation-cache name="entity" mode="SYNC">
        <transaction mode="NON_XA"/>
        <eviction strategy="LRU" max-entries="10000"/>
        <expiration max-idle="100000"/>
      </invalidation-cache>
      <replicated-cache name="timestamps" mode="ASYNC">
        <transaction mode="NONE"/>
        <eviction strategy="NONE"/>
      </replicated-cache>
    </cache-container>

  </subsystem>
```

This configuration excerpt resembles the native Infinispan configuration file; however some noticeable differences do exist. The first and most evident difference is that, while a native Infinispan configuration file contains cache configurations for **single** cache managers, the AS 7 Infinispan subsystem configuration defines **multiple** cache managers (labeled as **cache-container**), each one identified by a name.

Additionally, the AS 7 embedded configuration attempts to be more concise, thus it relies largely on default values. The complete set of default values can be found at `https://github.com/jbossas/jboss-as/blob/master/clustering/infinispan/src/main/resources/infinispan-defaults.xml`.

A closer look at the configuration

Let's take as an example, the web cache container that defines an asynchronous replicated cache used by clustered web applications:

```
<cache-container name="web"
    aliases="standard-session-cache" default-cache="repl">
  <transport lock-timeout="60000"/>
   replicated-cache name="repl" mode="ASYNC" batching="true">
    <file-store/>
     </replicated-cache>
 . . . .
</cache-container>
```

As you can see, each cache configuration has a name and an alias that make the role of the cache container understandable. At first, a lock timeout of 60 seconds is defined. Actually, only one cache can be doing state transfer or rehashing at a time; this timeout essentially controls the time to wait to acquire a lock on the distributed lock scenario.

The most interesting section is contained in the `replicated-cache` element—you probably recall defining the cache mode through a separate `<clustering mode="..."/>` attribute on a Infinispan native configuration, as shown here:

```
<clustering mode="replication">
    <async/>
</clustering>
```

When using Infinispan embedded with AS 7, each cache mode uses *its own element*, thus you can define it as follows:

- **replicated-cache**: This element is used by caches that replicate its state across all nodes of the cluster

- **distributed-cache**: This element is used by caches that distribute its state across a set of nodes of the cluster
- **invalidation-cache**: This element is used by caches that simply send an invalidation message to other nodes of the cluster

Another simplification in AS 7 configuration has been introduced to handle file-based cache stores. Because they are used frequently, there are sensible defaults for the class name and the path location of the cache store.

If the default values don't fit with your needs, you can specify the location of the store by using the attributes `<file-store relative-to="..." path="..."/>`. Check the AS 7 documentation (`https://docs.jboss.org/author/display/AS71/Documentation`) for more information about using these attributes.

Another cache configuration that is worth looking at is the **hibernate** cache container, which is related to the hibernate caching of entities and queries:

```
<cache-container name="hibernate" default-cache="local-query">
  <transport lock-timeout="60000"/>
  <local-cache name="local-query">
    <transaction mode="NONE"/>
    <eviction strategy="LRU" max-entries="10000"/>
    <expiration max-idle="100000"/>
  </local-cache>
  <invalidation-cache name="entity" mode="SYNC">
    <transaction mode="NON_XA"/>
    <eviction strategy="LRU" max-entries="10000"/>
    <expiration max-idle="100000"/>
  </invalidation-cache>
  <replicated-cache name="timestamps" mode="ASYNC">
    <transaction mode="NONE"/>
    <eviction strategy="NONE"/>
  </replicated-cache>
</cache-container>
```

In JBoss Application Server 7, Infinispan is the default **second level cache provider**. The purpose of a JPA/Hibernate second-level cache is to store entities/collections recently retrieved from a database, or to maintain results of recent queries. So, part of the aim of the second-level cache is to have data accessible locally rather than having to go to the database to retrieve it every time this is needed.

The first cache (**local-query**) relates to caching queries; by default, query cache is configured so that queries are only cached locally. This cache starts evicting entries when the number of entries exceeds 10,000 units. The query cache entries are also configured to expire after 100 seconds.

Provided that the query is likely to be repeatedly executed on different cluster nodes, and that the entities contained in the cache are mostly read, it can make sense to switch to a replicated cache for your queries:

```
<replicated-cache name="local-query">
    <transaction mode="NONE"/>
    <eviction strategy="LRU" max-entries="10000"/>
    <expiration max-idle="100000"/>
</replicated-cache>
```

The second cache, named **entity**, relates to caching entities and collections. This cache uses synchronous invalidation as the clustering mode, which means that when an entity is updated, the updated cache will send a message to the other members of the cluster telling them that the entity has been modified. Upon receipt of this message, the other nodes will remove this data from their local cache, if it was stored there.

This cache uses the same eviction/expiration settings, that is, the max number of entries is 10,000 and the max idle time before expiration is 100 seconds.

The third cache that is included is named **timestamp-cache**. The timestamp cache keeps track of the last update timestamp for each table (this timestamp is updated for any table modification). Any time the query cache is checked for a query, the timestamp cache is checked for all tables in the query. If the timestamp of the last update on a table is greater than the time the query results were cached, the entry is removed and the lookup is a miss.

By default, the timestamps cache is configured with asynchronous replication as the clustering mode. Local or invalidated cluster modes are not allowed, as all cluster nodes must store all timestamps. As a result, no eviction/expiration is allowed for timestamp caches either.

Defining default elements in the configuration

Careful readers should have noticed one more difference between an Infinispan native configuration and a JBoss AS 7 embedded configuration—recall the native Infinispan configuration:

```
<infinispan>
    <default>
      <jmxStatistics enabled="true"/>
```

```
    </default>

    <namedCache name="transactional">
      <transaction transactionMode="TRANSACTIONAL"/>
    </namedCache>

</infinispan>
```

In native Infinispan, the configuration within the `default` element defines the cache returned by calls to `CacheContainer.getCache()`, while `namedCache` entries inherit the configuration from the default cache.

The semantics of the default cache of a cache container are different in JBoss AS 7 from those in native Infinispan.

```
<subsystem
  xmlns="urn:jboss:domain:infinispan:1.1"
  default-cache-container="cluster">
</subsystem>
```

In JBoss AS 7, all caches defined in the Infinispan subsystem are **named** caches. The `default-cache` attribute identifies which named cache should be returned by calls to `CacheContainer.getCache()`. This lets you easily modify the default cache of a cache container, without having to worry about rearranging configuration property inheritance.

Defining global elements in the configuration

The last difference between native and embedded configuration relates to global settings. Actually, much of the AS 7 global configuration contains references to other JBoss AS services. In JBoss AS 7, these services are auto-injected behind the scenes. This includes things like thread pools, the JGroups transport (also described below), and the MBean server.

For example, as far as it concerns the transport configuration, this is handled by a separate JGroups subsystem:

```
<subsystem
  xmlns="urn:jboss:domain:jgroups:1.1"
  default-stack="udp">
  <stack name="udp">
       <transport
       type="UDP"
       socket-binding="jgroups-udp"
       diagnostics-socket-binding="jgroups-diagnostics"/>
```

```
        <protocol type="PING"/>
            <protocol type="MERGE2"/>
            . . . . .
        </stack>
        <stack name="tcp">
            <transport
           type="TCP"
           socket-binding="jgroups-tcp"
           diagnostics-socket-binding="jgroups-diagnostics"/>
         <protocol type="MPING" socket-binding="jgroups-mping"/>
            <protocol type="MERGE2"/>
            . . . . .
      </stack>
    </subsystem>
```

You can change the default stack for all clustering services by changing the `default-stack` attribute defined in the JGroups subsystem. Alternatively, you can set an individual `cache-container` stack by specifying a `stack` attribute within its transport element. For example:

```
<cache-container name="web" default-cache="repl">
     <transport stack="tcp"/>
  . . . .
</cache-container>
```

Defining thread pools

Another element of the global cache configuration that has been moved into an external subsystem is the thread pool subsystem. Externalizing thread pools in this way has the additional advantage of being able to manage them using native JBoss AS management mechanisms and allows you to share thread pools across cache containers. Here's how you can configure a Infinispan thread pool in JBoss AS 7:

```
<cache-container name="web" default-cache="repl"
listener-executor="infinispan-listener"
eviction-executor="infinispan-eviction"
replication-queue-executor="infinispan-repl-queue"
>
  . . . .
</cache-container>

<subsystem xmlns="urn:jboss:domain:threads:1.0">
  <thread-factory
    name="infinispan-factory"
    priority="1"/>
```

```
    <bounded-queue-thread-pool name="infinispan-transport"/>
      <core-threads count="1"/>
      <queue-length count="100000"/>
      <max-threads count="25"/>
      <thread-factory name="infinispan-factory"/>
    </bounded-queue-thread-pool>
    <bounded-queue-thread-pool name="infinispan-listener"/>
      <core-threads count="1"/>
      <queue-length count="100000"/>
      <max-threads count="1"/>
      <thread-factory name="infinispan-factory"/>
    </bounded-queue-thread-pool>
    <scheduled-thread-pool name="infinispan-eviction"/>
      <max-threads count="1"/>
      <thread-factory name="infinispan-factory"/>
    </scheduled-thread-pool>
    <scheduled-thread-pool name="infinispan-repl-queue"/>
      <max-threads count="1"/>
      <thread-factory name="infinispan-factory"/>
    </scheduled-thread-pool>
  </subsystem>
```

Here, the single thread pools configured in the threads subsystem correspond to the same thread pools that we discussed in the global configuration of native Infinispan, so we will not rehash these concepts here.

Using an Infinispan API from JBoss AS applications

Using Infinispan embedded within AS 7 makes coding your applications even easier. Basically, you can use the Infinispan API just as we have learnt through this book, enhanced by the ability to inject an Infinispan cache into your application using Java EE resource injection.

Let's suppose we want to create a simple version of our Ticket booking system to be used in a web application. Here is the Managed Bean that handles adding resources to our cache:

```
@ManagedBean(name="imanager")
public class InfinispanManager {
  @Resource(lookup="java:jboss/infinispan/container/cluster")
  private org.infinispan.manager.CacheContainer container;
   private org.infinispan.Cache<String, Ticket> cache;
```

```
    private String name;
    private String show;

    ArrayList<Ticket> cacheList;

    @PostConstruct public void start() {
     this.cache = this.container.getCache();
        cacheList = new ArrayList<Ticket>();
    }

    // Getters and Setters methods skipped for brevity

    public void save() {
  Ticket t = new Ticket(name, show);
     cache.put(generateTicketId(), t);
    }
    public void clear() {
  cache.clear();
    }

    public List<Ticket> getCacheList(){
  List<Ticket> dataList =
    new ArrayList<Ticket>();
     dataList.addAll(cache.values());
     return dataList;
    }

    // In real world projects, replace it with a Sequence
//  extracted from the DB
    public String generateTicketId() {
     String uuid =
    UUID.randomUUID().toString();
     return uuid;
    }
}
```

At startup, JBoss AS creates and registers an on-demand service for every Infinispan cache container defined in the Infinispan subsystem. For every cache container, the Infinispan subsystem also creates and registers a JNDI binding service that depends on the associated cache container service.

When the JBoss AS deployer encounters the `@Resource` annotation (see the first highlighted code section), it automatically adds a dependency to the application on the JNDI binding service associated with the specified JNDI name. The effect of this is that your application will include a dependency on the requested cache container. Consequently, the cache container will automatically start on deploy, and stop including all caches) on undeploy, of your application.

Please notice that the lookup attribute of the `@Resource` annotation needs the `jboss-annotations-api_1.1_spec-1.0.0.Final.jar` library in order to compile correctly. Keep it in your mind if you are adding manually the libraries to your project. See this article `http://bit.ly/KO08Nk` for more examples about it.

Subsequently, when `ManagedBean` is instantiated (see method start annotated as `@PostConstruct`), the cache container is stored in the class field cache. This field will be used through the bean to add or remove entries from the cache.

As far as it concerns application deployment, the only thing to note is that, due to JBoss AS's use of modular classloading, Infinispan classes are not available to deployments by default. You need to explicitly tell the AS to import the Infinispan API into your application. This is most easily done by adding the following line to your application's `META-INF/MANIFEST.MF` file:

```
Dependencies: org.infinispan export
```