Evaluation of the JINI HelloWorld example

The Documentation about JINI can be accessed from JINI 2.2.2 Documentation[1].

1. Project structure

The JINI HelloWorld example includes the following projects:

- mdeos.jini.root
- HelloWorldJiniRoot
- HelloEventsJiniRoot

1.1. The mdeos.jini.root

mdeos.jini.root is aggregator (maven pom) of a set of projects.

The projects in this set aim to provide common libraries to manage dependencies, initialization procedures.

- mdeos.jini.bootstrapper (Bootstraper)
- mdeos.jini.helper
- mdeos.jini.bom (Bill Of Material)
- mdeos.jini.riverallinone

mdeos.jini.bootstrapper project

The JiniBootstraperMain class starts a set of Jini services:

- ClassServer - HTTP server for codebase management (remote Classpath);
- Reggie – Lookup service or service registry;
- Mahalo – Transaction manager;
- Outrigger – Java spaces (Linda coordination model);

[1] https://river.apache.org/release-doc/2.2.2/

The mdeos.jini.helper project

The **ConfigureJiniFramework** class provide helper methods to be used by client and server/services.

The **setSecurity(String policyFile)** method receives a configuration security policies file (in the examples with ALL permissions **not acceptable in a real system**) and initializes a SecurityManager is necessary.

The **setServerCodebase()** method set value (**NOT acceptable in a real system**) to follow system property

- java.rmi.server.useCodebaseOnly - to false to allow remote code to be loaded and execute
- java.rmi.server.codebase - Automatic set codebase to http:YOUR_IP:8080/classes

The mdeos.jini.bom project

It concentrates the dependencies related to the JINI specific framework. Dependencies are resolved through an indirect mechanism based on this pom project;

The mdeos.jini.riverallinone project

Manages the JINI dependencies based on the Apache River open source and supports the generation of executable JAR files with all the dependencies embedded.

1.2. The HelloWorldJiniRoot project

Includes the three projects with a similar structure already used to validate OSGi Remote Services

- HelloWorld - HelloWorld API
- HelloWorldImpl - HelloWorld Implementation
- HelloWorldClient - HelloWorld Client

The HelloWorld API

The HelloWorld extends Remote since it needs to be exported as an remote reference. It further throws the **RemoteException** when an error occurs during remote method invocation.

HelloWorld interface.

```
public interface HelloWorld extends Remote {
    public String sayHello(String msg) throws RemoteException;
}
```

The HelloWorld Implementation (JINI service)

This project has 3 class

- HelloWorldImpl.java - Implementation of HelloWorld API
- HelloWorldRegistration.java - Registration of HelloWorldImpl proxy
- HelloWorldMain - Application entry point

The HelloWorld implementation(HelloWorldImpl.java).

```
public class HelloWorldImpl implements HelloWorld{

    public HelloWorldImpl() {
        System.out.println("Constructor of HelloWorldImpl()...");
    }

    @Override
    public String sayHello(String msg) {
        System.out.println("In sayHello(): " + msg);
        return msg.toUpperCase();
    }
}
```

The **JrmpExporter** class (Jini Exporter implementation) that generate the remote reference (proxy) with embed information about the location (e.g http path) from which the class is to be loaded.

Service Exporting JrmpExporter (HelloWorldRegistration.java).

```
//CODE OMITTED...
this.helloWorldImpl = new HelloWorldImpl();
this.exporter = new JrmpExporter();
this.helloWorldProxy = (HelloWorld) this.exporter.export(this.helloWorldImpl);
//CODE OMITTED...
```

Service Registration with JoinManager (HelloWorldRegistration.java).

```
//CODE OMITTED...
//Setup LookupDiscoveryManager(look for reggie)
```

```
this.discoveryManager =
    new LookupDiscoveryManager(SERVICE_GROUP, lookupLocators, null);

//Service Registration
this.joinManager =
    new JoinManager(this.helloWorldProxy, entries, this,
        discoveryManager, new LeaseRenewalManager());
//CODE OMITTED...
```

HelloWorldMain.java.

```
//CODE OMITTED...
//COPY EMBEDDED DIRECTORY 'config' to '{user directory}/.jini/'
ConfigureJiniFramework.copyDefaulEmbeddedDirToDefaultFileSystemDir();
//CODE OMITTED...
```

The Client

This project has 3 classes to show diferrent way of dicoverying service

Setting System property.

```
//CODE OMMITED...
ConfigureJiniFramework.setSecurity(ConfigureJiniFramework.JINI_RUNTIME_CONFDIR + "/
HelloWorldClient.policy");
ConfigureJiniFramework.setServerCodebase();
//CODE OMMITED...
```

lookup for JINI services.

A service in JINI is operationalized through a Java class. The HelloWorld example explores the remote access but a JINI service can be accessed by moving its implementation to the client (local access; serializable objects).

1.3. The Common classes to interact with the LookupService (Reggie)

The ServiceDiscoveryManager[2] a util class to simplify discovery of services (can be available at more than one registrar ; fault tolerance).

[2] https://river.apache.org/release-doc/2.2.2/api/net/jini/lookup/ServiceDiscoveryManager.html

The LookupDiscoveryManager[3] is common to clients and service publishers. The publishers use mainly the while service clients use the **ServiceDiscoveryManager** to look for services and obtain references (remote or local, in this case the object itself).

The **JoinManager** a util class to simplify service registration and leasing An associated default LeaseRenewalManager makes the service renewed forever.

2. The configuration files (JINI and Java Security)

The ~ <user>/.jini directory was adopted to support all the configuration needs. The configuration has three main aspects:

- The JINI configuration files, making possible to change behavior at runtime;
  - ~ <user>/.jini/config
- The codebase, providing classes to the classloaders at (remote) client side, and;
  - ** ~ <user>/.jini/www/classes
- The Java security mandatory when remote accesses are used (through Remote Method Invocation - RMI).
  - ~ <user>/.jini/config

To manage the configuration files under an automatic procedure, the POM files of the projects (src/main/resources) includes a plugin, maven-resources-plugin for that purpose.

How to copy class to specific folder with maven plugin (maven-resources-plugin).

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-resources-plugin</artifactId>
    <executions>
        <execution>
            <id>copy-resources</id>
            <!-- CODE OMITTED -->
        </execution>
    </executions>
</plugin>
```

[3] https://river.apache.org/release-doc/2.2.2/api/net/jini/discovery/LookupDiscoveryManager.html

This way, always a project change, all te resources are updated for runtime. The problem is with jar executable files. In these cases there is a need to manually copy the files to the respective places.

3. The Generation of a single executable JAR file

The Maven concept of profile makes possible to generate a different type of artifact. It is, in fact, a parametrized set of specific configurations, in the case of the iesd-1718sv-jini\mdeos.jini.root \mdeos.jini.bootstrapper project, it generates a single executable JAR file.

From the above directory, the directory with the POM file, call the mvn command with the -P switch.

Generate single (Big Jar) from command line.

```
mvn -Pbigjar package
```

See pom.xml in the mdeos.jini.bootstrapper project for more detail.

How to generate single jar with maven.

```
<profiles>
 <profile>
  <id>bigjar</id>
     <!-- CODE OMITTED -->
    </profile>
</profile>
```

4. Remoting

The **codebase**, while powerful, is a critical aspect since the code base need to be set and the directories with the required classes need to hold them (copied by a plugin in the maven POM of the respective project).

- codebase RFC3986 compliant URI path.

5. Multicast issues

The configuration of multicast is commonly a difficult. Considering performance (bandwith restrictions) extra messages beteen peers are not wellcome.

5.1. Windows

Configuration link[4] for the MADCAP service

5.2. Linux

The Linux shell command 'netsh interface ip show joins' shows relevant information for the understanding of Multicast configuration.

```
/sbin/route -n
/sbin/ifconfig enp0s8 multicast
/sbin/route -n
/sbin/route -n add -net 224.0.0.0 netmask 240.0.0.0 dev enp0
sudo ip link set dev enp0s8 multicast on
sudo tcpdump -i enp0s8 ip multicast
```

The Multicast configuration needs further formalization to make deployments more predictable (under the expected behaviors).

6. Reference

- Apache River 2.2.2 Javadoc[5]
- Apache River 2.2.2 Documentation[6].
- Dynamic code downloading using Java™ RMI(Using the java.rmi.server.codebase Property)[7]
- How Codebase Works[8]
- Articles: Distributed Events in Jini Technology[9]
- Professional Jini, FREE sample chapter7 - Jini Distributed Events[10]

[4] https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc776133(v=ws.10)
[5] https://river.apache.org/release-doc/2.2.2/api/overview-summary.html
[6] https://river.apache.org/release-doc/2.2.2/
[7] https://docs.oracle.com/javase/7/docs/technotes/guides/rmi/codebase.html
[8] http://www.kedwards.com/jini/codebase.html
[9] http://www.oracle.com/technetwork/articles/javase/jinievents-140780.html
[10] http://www.javacoffeebreak.com/books/samples/professionaljini/3552_Chap7_idx.pdf