Distributed Systems – TD7 : JXTA - first steps

Bogdan.Pasca @ens-lyon.fr

13 November 2009

HelloWorld Example

This example illustrates starting and stopping JXSE. The application controls configuration, startup and stopping of the JXTA network using the NetworkManager utility class.

Compile and run the following code :

```
 1  import net.jxta.platform.NetworkManager;
 2  import java.text.MessageFormat;
 3  /**
 4    * A example of starting and stopping JXTA
 5    */
 6  public class HelloWorld {
 7       /**
 8         * Main method
 9         *
10         * @param args none defined
11         */
12       public static void main(String args[]) {
13            NetworkManager manager = null;
14            try {
15                manager = new NetworkManager(NetworkManager.ConfigMode.EDGE, "HelloWorld");
16                System.out.println("Starting␣JXTA");
17                manager.startNetwork();
18                System.out.println("JXTA␣Started");
19            } catch (Exception e) {
20                e.printStackTrace();
21                System.exit(-1);
22            }
23            System.out.println("Waiting␣for␣a␣rendezvous␣connection");
24            boolean connected = manager.waitForRendezvousConnection(12000);
25            System.out.println(MessageFormat.format("Connected␣:{0}", connected));
26            System.out.println("Stopping␣JXTA");
27            manager.stopNetwork();
28       }
29  }
```

For compilation be sure to include in your **classpath** the JXTA jars.

```
javac -classpath ./jxta.jar:./bcprov-jdk14.jar:./javax.servlet.jar:./org.mortbay.jetty.jar:. HelloWorld.java
java -classpath ./jxta.jar:./bcprov-jdk14.jar:./javax.servlet.jar:./org.mortbay.jetty.jar:. HelloWorld
```

Line 15 configures the peer as edge node, with a peer name of *HelloWorld*, this call only creates the default configuration for the JXTA peer.

Why the result ?

```
INFO: Started JXTA Network!
JXTA Started
Waiting for a rendezvous connection
INFO: Start relay client thread
Connected :false
Stopping JXTA
INFO: Stopping JXTA Network!
```

The NetworkManager abstracts the configuration to one of the preset configurations :

Ad-Hoc : for deploying a stand-alone node in an ad-hoc network and connected in a peer-to-peer fashion using multicast, i.e., no rendezvous or relay services are used.

Edge : supports Ad-Hoc behavior and also allows a peer to connect to an infrastructure peer (a rendezvous, relay, or both).

Rendezvous : allows the node to make network bootstrapping services available to others, such as discovery, pipe resolution, etc.

Relay : enables a peer to provide message relaying services to allow peers to traverse firewalls.

Proxy : enables JXME Jxta for J2ME proxy services.

Super : enables the simultaneous functionality of a rendezvous, relay and a proxy node.

Try replacing line 15 with :

```
manager = new NetworkManager(NetworkManager.ConfigMode.ADHOC, "HelloWorld", new File(new File(".cache"), "HelloWorld").toURI());
```

```
INFO: Started JXTA Network!
JXTA Started
Waiting for a rendezvous connection
Connected :true
Stopping JXTA
INFO: Stopping JXTA Network!
```

When the application completes, you can inspect the various files and subdirectories that were created in the `./.cache/HelloWorld` subdirectory :

PlatformConfig : the configuration file created by the auto-configuration tool

cm : the local cache directory ; it contains subdirectories for each group that is discovered. In the hello world example, we should see the jxta-NetGroup and jxta-WorldGroup subdirectories. These subdirectories will contain index files (*.idx) and advertisement store files (advertisements.tbl).

Augment the *Hello World* example so that the details peer and peer-group are also displayed.

```
INFO: Started JXTA Network!
JXTA Started
Peer name       : HelloWorld
Peer Group name: NetPeerGroup
Peer Group ID : urn:jxta:uuid-59616261646162614E5047205032503362 5BC0AEC3604A78A01083C46847105A03
Waiting for a rendezvous connection
Connected :true
Stopping JXTA
```

Remember the *PlatformConfig* file ? This file is created each time the network manager is instantiated. In order to avoid this overhead, We will learn to work with local configurations. This will be useful when we need a specific configuration on an application basis.

We will need a network configurator in order to configure the network :

```
NetworkConfigurator TheConfig;
```

We need to set the network manager configuration to persistent to make sure the Peer ID is not re-created each time the Network Manager is instantiated :

```
manager.setConfigPersistent(true);
```

Next we need to obtain the network configurator or object :

```
TheConfig = manager.getConfigurator();
```

Then we check if a local configuration exists. If it does we load it. Try/catch constructions have been left out for clarity. Be sure to use them in your code.

```
// Does a local peer configuration exist?
if (TheConfig.exists()) {
    // We load it
    File LocalConfig = new File(TheConfig.getHome(), "PlatformConfig");
    TheConfig.load(LocalConfig.toURI());
} else {
    TheConfig.setName(Local_Peer_Name);
    TheConfig.setPrincipal(User_Name);
    TheConfig.setPassword(Password);
    TheConfig.save();
}
```

Augment the *Hello World* example with the capability to read the local configuration.

Next we need to find other peers in the network. On one hand we will have the JXTA-Shell with one peer. On the other we will have an augmented *Hello World* program which sends remote discovery advertisement requests.

In order to be able to send advertisement requests we need to declare and instantiate a discovery service.

```
DiscoveryService discovery;
```

Then we need to start the service :

```
discovery = TheNetPeerGroup.getDiscoveryService();
```

In order to asynchronously receive discovery events, the *HelloWorld* class needs to implement *DiscoveryListener*.

```
public class HelloWorld implements  DiscoveryListener
```

Then we need to implement the `public void discoveryEvent(DiscoveryEvent ev)` method that is asynchronously called when a discovery request reply is received. In order to link the current object to the events associated to the discovery service we need to :

```
discovery.addDiscoveryListener(this);
```

The discovery requests have the form :

```
discovery.getRemoteAdvertisements(// no specific peer (propagate)
null, // Adv type
DiscoveryService.ADV, // Attribute = any
null, // Value = any
null, // one advertisement response is all we are looking for
1, // no query specific listener. we are using a global listener
null);
```

Check that your program works by verifying that only when the shell is started the discoveryEvent(DiscoveryEvent ev) occurs.