CHAPTER 12

Discovery Management

Clients and services both need to find lookup services. Previously, we looked at the code that was common to both clients and services in both unicast and broadcast discovery. Parts of that code have been used in many of this book's examples since. This chapter discusses some utility classes that make it easier to deal with lookup services by encapsulating this type of code into common utility classes and providing a good interface to them.

Finding Lookup Locators

Both services and clients need to find lookup locators. Services will register with these locators, and clients will query them for suitable services. Finding these lookup locators involves three components:

- A list of lookup locators for unicast discovery
- A list of groups for lookup locators using multicast discovery
- Listeners whose methods are invoked when a service locator is found

Chapter 4 considered the cases of a single unicast lookup service or a set of multicast lookup services. There are also mechanisms to handle a set of unicast lookup services *and* a set of multicast lookup services. Three interfaces are involved:

- `DiscoveryManagement` looks after discovery events.
- `DiscoveryGroupManagement` looks after groups and multicast search.
- `DiscoveryLocatorManagement` looks after unicast discovery.

Different classes may implement different combinations of these three interfaces. The `LookupDiscovery` class discussed in Chapter 4 uses `DiscoveryGroupManagement` and `DiscoveryManagement`. It performs a multicast search, informing its listeners when lookup services are discovered. The `LookupLocatorDiscovery` class is discussed later in this chapter. It performs a similar task for unicast discovery. It implements the two interfaces `DiscoveryLocatorManagement` and `DiscoveryManagement`. Another class discussed later is `LookupDiscoveryManager`, which handles both unicast and broadcast discovery, and so implements all three interfaces. With these three cases covered, it is unlikely that you will need to implement these interfaces yourself.

The `DiscoveryManagement` interface is as follows:

```
package net.jini.discovery;
public interface DiscoveryManagement {
    public void addDiscoveryListener(DiscoveryListener l);
    public void removeDiscoveryListener(DiscoveryListener l);
    public ServiceRegistrar[] getRegistrars();
    public void discard(ServiceRegistrar proxy);
    public void terminate();
}
```

The `addDiscoveryListener()` method is the most important method, as it allows a listener object to be informed whenever a new lookup service is discovered.

The `DiscoveryGroupManagement` interface is shown next:

```
package net.jini.discovery;

public interface DiscoveryGroupManagement {

    public static final String[] ALL_GROUPS = null;
    public static final String[] NO_GROUPS = new String[0];

    public String[] getGroups();
    public void addGroups(String[] groups) throws IOException;
    public void setGroups(String[] groups) throws IOException;
    public void removeGroups(String[] groups);
}
```

The most important of these methods is `setGroups()`. If the groups have initially been set to `NO_GROUPS`, no multicast search is performed. If it is later changed by `setGroups()`, then this initiates a search. Similarly, `addGroups()` will also initiate a search. (This is why they may throw remote exceptions.)

The third interface is `DiscoveryLocatorManagement`:

```
package net.jini.discovery;
public interface DiscoveryLocatorManagement {
    public LookupLocator[] getLocators();
    public void addLocators(LookupLocator[] locators);
    public void setLocators(LookupLocator[] locators);
    public void removeLocators(LookupLocator[] locators);
}
```

An implementation will generally set the locators in its own constructor, so these methods will probably only be useful if you need to change the set of unicast addresses for the lookup services.

LookupLocatorDiscovery

In Chapter 4, the section on finding a lookup service at a known address looked only at a single address. If lookup services at multiple addresses are required, then a naive solution would be to put the code from that chapter into a loop. The LookupLocatorDiscovery class offers a more satisfactory solution by providing the same event handling method as in the multicast case; that is, you supply a list of addresses, and when a lookup service is found at one of these address, a listener object is informed.

The LookupLocatorDiscovery class is specified as follows:

```
package net.jini.discovery;
public class LookupLocatorDiscovery implements DiscoveryManagement,
                                               DiscoveryLocatorManagement {
    public LookupLocatorDiscovery(LookupLocator[] locators);
    public LookupLocatorDiscovery(LookupLocator[] locators,
                                  Configuration config);
    public LookupLocator[] getDiscoveredLocators();
    public LookupLocator[] get UndiscoveredLocators();
}
```

An additional constructor has been added in Jini 2.0.

Rewriting the unicast example from Chapter 4 using this utility class makes it look much like the example on multicast discovery from the same chapter. The similarity is that it now uses the same event model for lookup service discovery; the difference is that it uses a set of LookupLocator objects rather than a set of groups.

```
package discoverymgt;
import net.jini.discovery.LookupLocatorDiscovery;
import net.jini.discovery.DiscoveryListener;
import net.jini.discovery.DiscoveryEvent;
import net.jini.core.lookup.ServiceRegistrar;
import net.jini.core.discovery.LookupLocator;
import java.net.MalformedURLException;
import java.rmi.RMISecurityManager;
/**
 * UnicastRegister.java
 */
public class UnicastRegister implements DiscoveryListener {

    static public void main(String argv[]) {
        new UnicastRegister();
        // stay around long enough to receive replies
        try {
            Thread.currentThread().sleep(10000L);
        } catch(java.lang.InterruptedException e) {
            // do nothing
        }
    }
    public UnicastRegister() {
        // install suitable security manager
        System.setSecurityManager(new RMISecurityManager());
        LookupLocatorDiscovery discover = null;
        LookupLocator[] locators = null;
        try {
            locators = new LookupLocator[] {new LookupLocator("jini://localhost")};
        } catch(MalformedURLException e) {
            e.printStackTrace();
            System.exit(1);
        }
        try {
            discover = new LookupLocatorDiscovery(locators);
        } catch(Exception e) {
            System.err.println(e.toString());
            e.printStackTrace();
            System.exit(1);
        }
        discover.addDiscoveryListener(this);
    }

    public void discovered(DiscoveryEvent evt) {
        ServiceRegistrar[] registrars = evt.getRegistrars();
        for (int n = 0; n < registrars.length; n++) {
            ServiceRegistrar registrar = registrars[n];
            // the code takes separate routes from here for client or service
            System.out.println("found a service locator");
          }
    }
    public void discarded(DiscoveryEvent evt) {
    }
} // UnicastRegister
```

LookupDiscoveryManager

An application (client or service) that wants to use a set of lookup services at fixed, known addresses and also wants to use whatever lookup services it can find by multicast can use the `LookupDiscoveryManager` utility class. Most of the methods of this class come from its interfaces:

```
package net.jini.discovery;
public class LookupDiscoveryManager implements DiscoveryManagement,
                                               DiscoveryGroupManagement,
                                               DiscoveryLocatorManagement {
    public LookupDiscoveryManager(String[] groups,
                                  LookupLocator[] locators,
                                  DiscoveryListener listener)
                                      throws IOException;
    public LookupDiscoveryManager(String[] groups,
                                  LookupLocator[] locators,
                                  DiscoveryListener listener,
                                  Configuration config)
                                      throws IOException,
                                             ConfigurationException;
}
```

This class differs from `LookupDiscovery` and `LookupLocatorDiscovery` in that it insists on a `DiscoveryListener` in its constructor. Programs using this class can follow the same event model as the last example:

```
package discoverymgt;
import net.jini.discovery.LookupDiscoveryManager;
import net.jini.discovery.DiscoveryGroupManagement;
import net.jini.discovery.DiscoveryListener;
import net.jini.discovery.DiscoveryEvent;
import net.jini.core.lookup.ServiceRegistrar;
import net.jini.core.discovery.LookupLocator;
import java.net.MalformedURLException;
import java.io.IOException;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
/**
 * AllcastRegister.java
 */
public class AllcastRegister implements DiscoveryListener {

    static public void main(String argv[]) {
        new AllcastRegister();
        // stay around long enough to receive replies
        try {
            Thread.currentThread().sleep(10000L);
        } catch(java.lang.InterruptedException e) {
            // do nothing
        }
    }

    public AllcastRegister() {
        // install suitable security manager
        System.setSecurityManager(new RMISecurityManager());
        LookupDiscoveryManager discover = null;
        LookupLocator[] locators = null;
        try {
            locators = new LookupLocator[] {new LookupLocator("jini://localhost")};
        } catch(MalformedURLException e) {
            e.printStackTrace();
            System.exit(1);
        }
        try {
            discover = new LookupDiscoveryManager
                       (DiscoveryGroupManagement.ALL_GROUPS,
                                                  locators,
                                                  this);
        } catch(IOException e) {
            System.err.println(e.toString());
            e.printStackTrace();
            System.exit(1);
        }
    }

    public void discovered(DiscoveryEvent evt) {
        ServiceRegistrar[] registrars = evt.getRegistrars();
        for (int n = 0; n < registrars.length; n++) {
            ServiceRegistrar registrar = registrars[n];
            try {
                System.out.println("found a service locator at " +
                               registrar.getLocator().getHost());
            } catch(RemoteException e) {
                e.printStackTrace();
                continue;
            }
            // the code takes separate routes from here for client or service
          }
    }
    public void discarded(DiscoveryEvent evt) {
    }
} // AllcastRegister
```

Summary

The `LookupLocatorDiscovery` and `LookupDiscoveryManager` utility classes add to the `LookupDiscovery` class by making it easier to find lookup services using both unicast and broadcast searches.