CHAPTER 11
Join Manager

FINDING A LOOKUP SERVICE INVOLVES a common series of steps, and convenience classes for encapsulating this were considered in the last chapter. Subsequent interaction with the discovered lookup services also involves common steps for services as they register with the lookup services. A join manager encapsulates these additional steps into one convenience class for services.

Jini 1.1 JoinManager

A service needs to locate lookup services and register the service with them. Locating services can be done using the utility classes from Chapter 10. As each lookup service is discovered, it needs to be registered, and the lease needs to be maintained. The `JoinManager` class performs all of these tasks. There are two constructors:

```
public class JoinManager {
     public JoinManager(Object obj,
                        Entry[] attrSets,
                        ServiceIDListener callback,
                        DiscoveryManagement discoverMgr,
                        LeaseRenewalManager leaseMgr)
            throws IOException;

     public JoinManager(Object obj,
                        Entry[] attrSets,
                        ServiceID serviceID,
                        DiscoveryManagement discoverMgr,
                        LeaseRenewalManager leaseMgr)
            throws IOException;
}
```

The first constructor is used when the service is new and does not have a service ID. A `ServiceIDListener` can be added to note and save the ID. The second constructor is used when the service already has an ID. The other parameters are for the service and its entry attributes, a `DiscoveryManagement` object to set groups and unicast locators (typically this will be done using a `LookupDiscoveryManager`), and a lease renewal manager.

The following example uses the `JoinManager` class to register a `FileClassifierImpl`. In the Chapter 8 example of "Uploading a Complete Service" (and other examples in Chapter 9) the server implemented the `DiscoveryListener` interface in order to be informed when new lookup locators were discovered so that the service could be registered with each of them. If you use a join manager, there is no need for a `DiscoveryListener`, since the join manager adds itself as a listener and handles the registration with the lookup service.

```
package joinmgr;

import rmi.FileClassifierImpl;

import net.jini.lookup.JoinManager;
import net.jini.core.lookup.ServiceID;
import net.jini.discovery.LookupDiscovery;
import net.jini.core.lookup.ServiceRegistrar;
import java.rmi.RemoteException;
import net.jini.lookup.ServiceIDListener;
import net.jini.lease.LeaseRenewalManager;
import net.jini.discovery.LookupDiscoveryManager;
import net.jini.discovery.DiscoveryEvent;
import net.jini.discovery.DiscoveryListener;


/**
 * FileClassifierServer.java
 */

public class FileClassifierServer
    implements ServiceIDListener {

    public static void main(String argv[]) {
        new FileClassifierServer();

        // stay around forever
        Object keepAlive = new Object();
        synchronized(keepAlive) {
            try {
                keepAlive.wait();
            } catch(InterruptedException e) {
                // do nothing
            }
        }
    }
```

```
    public FileClassifierServer() {

        JoinManager joinMgr = null;
        try {
            LookupDiscoveryManager mgr =
                new LookupDiscoveryManager(LookupDiscovery.ALL_GROUPS,
                                           null /* unicast locators */,
                                           null /* DiscoveryListener */);
           joinMgr = new JoinManager(new cokeServer.CokeMachine(), // new F

las-
sifierImpl(), /* service */
                                      null /* attr sets */,
                                      this /* ServiceIDListener*/,
                                      mgr /* DiscoveryManagement */,
                                      new LeaseRenewalManager());
        } catch(Exception e) {
            e.printStackTrace();
            System.exit(1);
        }
    }

    public void serviceIDNotify(ServiceID serviceID) {
        // called as a ServiceIDListener
        // Should save the ID to permanent storage
        System.out.println("got service ID " + serviceID.toString());
    }

} // FileClassifierServer
```

There are a number of other methods in `JoinManager` that allow you to modify the state of a service registration.

Jini 1.0 JoinManager

A version of `JoinManager` was present in Jini 1.0. At that time it was in the `com.sun` package and no formal specification was given. Classes in the `com.sun` packages may be changed in later versions of Jini, or may even disappear completely. In moving from Jini 1.0 to Jini 1.1, the `JoinManager` classes were specified and moved to a new package. This section describes the old version for those still using Jini 1.0. When possible, such users should switch to Jini 1.1

There are a number of possible constructors for `JoinManager`. This is the simplest:

```
JoinManager(java.lang.Object obj,
            Entry[] attrSets,
            ServiceIDListener callback,
            LeaseRenewalManager leaseMgr)
```

This constructor specifies the service to be managed and its entry attributes. The `callback` is a listener object that will have its `serviceIDNotify()` method called when a new locator is discovered. This is usually used to find the value of the `ServiceID` assigned by a lookup locator to a service. The `callback` argument can be `null` if the programmer has no interest in saving the `ServiceID`. The `leaseMgr` can also be set to `null` and will then be created as needed.

This constructor will initiate a search for service locators belonging to the group "public", which is defined by a group value of the empty string `""`. There is no constant for this, and the locators from Sun do not appear to belong to this group, so most applications will need to follow this up immediately with a call to search for locators belonging to any group:

```
JoinManager joinMgr = new JoinManager(obj, null, null, null);
joinMgr.setGroups(LookupDiscovery.ALL_GROUPS);
```

The second constructor is as follows:

```
JoinManager(java.lang.Object obj,
            Entry[] attrSets,
            java.lang.String[] groups,
            LookupLocator[] locators,
            ServiceIDListener callback,
            LeaseRenewalManager leaseMgr)
```

This constructor adds `groups` and `locators`, which allow multicast searches for locators belonging to certain groups, and also unicast lookups for known locators.

A multicast-only search for any groups would have both additional parameters set to `null`:

```
JoinManager joinMgr = new JoinManager(obj, null,
                                      LookupDiscovery.ALL_GROUPS,
                                      null, null, null);
```

On the other hand, a unicast lookup for a single known site would be done like this:

```
LookupLocator[] locators = new LookupLocator[1];
locators[0] new LookupLocator("http://www.all_about_files.com");
JoinManager joinMgr = new JoinManager(obj, null,
                                      LookupDiscovery.NO_GROUPS,
                                      locators, null, null);
```

(This code ignores exception handling.)

For example, uploading the complete service of the `complete` package could be done as follows:

```
package joinmgr;

import complete.FileClassifierImpl;

import com.sun.jini.lookup.JoinManager;
import net.jini.core.lookup.ServiceID;
import com.sun.jini.lookup.ServiceIDListener;
import com.sun.jini.lease.LeaseRenewalManager;
import net.jini.discovery.LookupDiscovery;

/**
 * FileClassifierServer1_0.java
 */

public class FileClassifierServer1_0 implements ServiceIDListener {

    public static void main(String argv[]) {
        new FileClassifierServer1_0();

        // stay around long enough to receive replies
        try {
            Thread.currentThread().sleep(1000000L);
        } catch(java.lang.InterruptedException e) {
            // do nothing
        }

    }

    public FileClassifierServer1_0() {

        JoinManager joinMgr = null;
```

```
        try {
            /* this is one way of doing it
            joinMgr = new JoinManager(new FileClassifierImpl(),
                                      null,
                                      this,
                                      new LeaseRenewalManager());
            joinMgr.setGroups(null);
            */
            /* here is another */
            joinMgr = new JoinManager(new FileClassifierImpl(),
                                      null,
                                      LookupDiscovery.ALL_GROUPS,
                                      null,
                                      this,
                                      new LeaseRenewalManager());
        } catch(Exception e) {
            e.printStackTrace();
            System.exit(1);
        }
    }

    public void serviceIDNotify(ServiceID serviceID) {
        System.out.println("got service ID " + serviceID.toString());
    }

} // FileClassifierServer1_0
```

Getting Information from JoinManager

The `JoinManager` looks unhelpful in supplying information about the lookup locators it finds. However, this information is available by a slightly circuitous route. A service can register a `ServiceIDListener` to the `JoinManager`. This will be invoked whenever a new locator is found by its `serviceIDNotify()` method. A `ServiceID` is not particularly useful, so we just ignore it. However, within the `serviceIDNotify()` method we do know that a new service locator has been found, since that is the only occasion on which it is called.

The *complete set* of service locators can be found with the `JoinManager`'s `getJoinSet()` method, which returns an array of `ServiceRegistrar` objects. We have met this class before: its `getLocator()` method will return a `LookupLocator`, which has information such as the host in `getHost()`. These classes can be put together as follows:

```
protected JoinManager joinmgr;
...
```

```
joinmgr = new JoinManager(service, null,
                                        this, new  eManager());
...

public  void serviceIDNotify(ServiceID serviceID) {
    ServiceRegistrar registrars = joinmgr.getJoinSet();
    for (int n = 0; n < registrar.length; n++) {
        LookupLocator locator = registrars[n].getLocator();
        String hostName = locator.getHost();
        ...
    }
}
```

If you want to find out which is the latest locator to be found, you will have to cache the previous set and find which is new in the array returned. Each call to `getJoinSet()` will return a new array.

Summary

A `JoinManager` can be used by a server to simplify many of the aspects of locating lookup services, registering one or more services, and renewing leases for them.