CHAPTER 13

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 lookup services. A join manager encapsulates these additional steps into one convenience class for services.

Registering Services

A service needs to locate lookup services and register the service with them. Locating services can be done using the utility classes from Chapter 12. As each lookup service is discovered, it then 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;
     public JoinManager(Object obj,
                        Entry[] attrSets,
                        ServiceIDListener callback,
                        DiscoveryManagement discoverMgr,
                        LeaseRenewalManager leaseMgr,
                        Configuration config)
            throws IOException,
                   ConfigurationException;
     public JoinManager(Object obj,
                        Entry[] attrSets,
```

```
                        ServiceID serviceID,
                        DiscoveryManagement discoverMgr,
                        LeaseRenewalManager leaseMgr,
                        Configuration config)
            throws IOException,
                   ConfigurationException;
}
```

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 third and fourth constructors add a `Configuration` parameter to the first and second constructors, respectively.

The following example uses the first constructor to register a `FileClassifierImpl`. There is no need for a `DiscoveryListener`, since the join manager adds itself as a listener and handles the registration with the lookup service. Note that a proxy has to be created using an `Exporter`, and then the proxy is passed as the first parameter to the `JoinManager`.

```
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;
import java.rmi.RMISecurityManager;
import java.rmi.Remote;
import net.jini.config.*;
import net.jini.export.*;
/**
 * FileClassifierServer.java
 */
public class FileClassifierServer
    implements ServiceIDListener {
    // explicit proxy for Jini 2.0
    protected Remote proxy;
    protected FileClassifierImpl impl;
    private static String CONFIG_FILE = "jeri/file_classifier_server.config";

    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() {
        try {
            impl = new FileClassifierImpl();
        } catch(Exception e) {
            System.err.println("New impl: " + e.toString());
            System.exit(1);
        }
        String[] configArgs = new String[] {CONFIG_FILE};
        try {
            // get the configuration (by default a FileConfiguration)
            Configuration config = ConfigurationProvider.getInstance(configArgs);

            // and use this to construct an exporter
            Exporter exporter = (Exporter) config.getEntry( "FileClassifierServer",
                                                            "exporter",
                                                            Exporter.class);
            // export an object of this class
            proxy = exporter.export(impl);
        } catch(Exception e) {
            System.err.println(e.toString());
            e.printStackTrace();
            System.exit(1);
        }
        // install suitable security manager
        System.setSecurityManager(new RMISecurityManager());
        JoinManager joinMgr = null;
        try {
            LookupDiscoveryManager mgr =
                new LookupDiscoveryManager(LookupDiscovery.ALL_GROUPS,
                                           null,  // unicast locators
                                           null); // DiscoveryListener
            joinMgr = new JoinManager(proxy, // service proxy
                                      null,  // attr sets
                                      this,  // ServiceIDListener
                                      mgr,   // DiscoveryManager
                                      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
```

An Ant build, deploy, and run file for this is similar in structure to earlier examples. The changes are to the source and class files. These are `joinmgr.FileClassifierServer.xml`:

```
<!-- files for this project -->
<!-- Source files for the server -->
<property name="src.files"
          value="
                 common/MIMEType.java,
                 common/FileClassifier.java,
                 rmi/RemoteFileClassifier.java,
                 rmi/FileClassifierImpl.java,
                 joinmgr/FileClassifierServer.java
                "/>
<!-- Class files to run the server -->
<property name="class.files"
          value="
                 common/MIMEType.class,
                 common/FileClassifier.class,
                 rmi/RemoteFileClassifier.class,
                 rmi/FileClassifierImpl.class,
                 joinmgr/FileClassifierServer.class
                "/>
<!-- Class files for the client to download -->
<property name="class.files.dl"
          value="
                 common/MIMEType.class,
                 common/FileClassifier.class,
                 rmi/RemoteFileClassifier.class,
                 rmi/FileClassifierImpl.class
                "/>
```

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

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.