Contents of Doc 19, Djinn Groups & Discovery

References


Jini Discovery and Join Specification 1.0 January 25, 1999
Jini Discovery Utilities Specification 1.0 January 25, 1999

These documents are available at: http://www.sun.com/jini/specs/index.html

Jini Source Code, package com.sun.jini.reggie, classes:

CreateLookup
RegistrarAdminProxy,
RegistrarImpl,
RegistrarProxy,


Djinn Admin & Organization
Djinn Groups

A djinn is a collection of devices, resources and users joined by the Jini software infrastructure. A lookup service provides the central registry of services in the djinn.

Each lookup service belongs to one or more groups. Different lookup services can be long to the same group. A group is defined by its name, which is just a string. Lookup services (djinns) can be found and contacted by using group names.

While any string can be used as the name of a group, one should use DNS-style name. For example:

printers.cs.sdsu.edu
phoneNumbers.sdsu.edu

The use of DNS-style names is recommended to help avoid name conflicts.

There are two special group names:

"none" indicates that the lookup service not in a group
"public" is the default group

When a new lookup service, it is given its initial group names.

The format for startup command line is:

```
java -jar <lookup-server-jarfile> <lookup-client-codebase> 
<lookup-policy-file> <output-log-dir> 
[<lookup-service-group>] [<activation-startup-cmd>] 
[<lookup-VM-cmd-line-flags>]
```

If `<lookup-service-group>` is missing, "public" is used.

To indicated more than one group separate the names by a comma ",,"

```
java   -jar  /opt/jini1_0/lib/reggie.jar
    http://fargo.sdsu.edu:8888/reggie-dl.jar
    /opt/jini1_0/example/lookup/policy
    /tmp/reggie_log
    public,printer.physics.sdsu.edu
```

Djinn Admin

The administration of a lookup service is defined by 5 interfaces. All but RegistrarAdmin are designed for any service.
DestroyAdmin  DiscoveryAdmin  JoinAdmin
RegistrarAdmin  StorageLocationAdmin

com.sun.jini.admin.DestroyAdmin

Destroy the service, if possible, including its persistent storage

Method

destroy()

net.jini.lookup.DiscoveryAdmin

Add/remove service from groups
Change the port the service uses for its lookup locator

Methods

addMemberGroups(java.lang.String[] groups)
getMemberGroups()
removeMemberGroups(java.lang.String[] groups)
setMemberGroups(java.lang.String[] groups)
getUnicastPort()
setUnicastPort(int port)

net.jini.admin.JoinAdmin

The methods in this interface are used to control a service's participation in the join protocol. By providing the relevant information (group, LookupLocator, or attributes) you can have the Lookup service register itself with other lookup services. Actually you can have any service that supports this interface register (join) with any service that supports the join processes. A separate thread manages the join processes for a lookup service. The thread looks for services to join for the life of lookup service.

Methods

addLookupAttributes(Entry[] attrSets)
addLookupGroups(java.lang.String[] groups)
addLookupLocators(LookupLocator[] locators)
getLookupAttributes()
getLookupGroups()
getLookupLocators()
modifyLookupAttributes(Entry[] attrSetTemplates,
removeLookupGroups(java.lang.String[] groups)
removeLookupLocators(LookupLocator[] locators)
setLookupGroups(java.lang.String[] groups)
setLookupLocators(LookupLocator[] locators)

com.sun.jini.admin.StorageLocationAdmin

Change the location of the persistent storage for the service

Methods

getStorageLocation()
setStorageLocation(java.lang.String location)
**com.sun.jini.reggie.RegistrarAdmin**

This interface lookup service:

- Controls lease lengths granted by a lookup service
- Sets criteria for performing incremental/full backs of the service
- Extends: DestroyAdmin, DiscoveryAdmin, JoinAdmin, StorageLocationAdmin,

**minMaxServiceLease**

- Lower bound on the maximum service lease
- Default value = 5 minutes

**minMaxEventLease**

- Lower bound on the maximum event lease
- Default value = 30 minutes

**minRenewalInterval**

- Minimum average interval between lease renewals
- Default value = 100 milliseconds

Maximum leases granted are never less then the lower bound and computed to insure that the time between any two lease renewals is at least minRenewalInterval.

**Methods**

- getMinMaxEventLease()
- getMinMaxServiceLease()
- getMinRenewalInterval()
- setMinMaxEventLease(long leaseDuration)
- setMinMaxServiceLease(long leaseDuration)
- setMinRenewalInterval(long interval)
- getLogToSnapshotThreshold()
- getSnapshotWeight()
- setLogToSnapshotThreshold(int threshold)
- setSnapshotWeight(float weight)

**net.jini.admin.Administrable**

This interface covers gaining access to implementations of the other administrative interfaces

**Method**

- getAdmin()

Returns administrative object for the service
Lookup Service Administration

The ServiceRegistrar for the lookup service implements:

   Administrable
   RegistrarAdmin

The example on the next slide shows how to interact with the lookup service administration

Administration Example

```java
package whitney.jini.examples.admin;
import com.sun.jini.reggie.RegistrarAdmin;
import java.rmi.RMISecurityManager;
import net.jini.admin.Administrable;
import net.jini.core.discovery.LookupLocator;
import net.jini.core.entry.Entry;
import net.jini.core.lookup.ServiceRegistrar;
import net.jini.core.lookup.ServiceTemplate;
import net.jini.lookup.entry.Name;
public class GroupAdder {
    public static void main (String[] args) throws Exception {
        System.setSecurityManager (new RMISecurityManager ());
        LookupLocator lookup = new LookupLocator ("jini://eli.sdsu.edu");
        ServiceRegistrar registrar = lookup.getRegistrar();

        // The proxy for ServiceRegistrar implements interfaces
        // ServiceRegistrar, Administrable and Serializable
        Administrable adminAccess = (Administrable) registrar;
        RegistrarAdmin lookupAdmin =
            (RegistrarAdmin) adminAccess.getAdmin();

        // Add command line args as new groups
        lookupAdmin.addMemberGroups( args);

        //Verify the change
        String[] serviceGroups = registrar.getGroups();
        System.out.println( "Groups for " + lookup);
        for (int k = 0; k < serviceGroups.length; k++)
            System.out.println( "   " + serviceGroups[k] );
    }
}
```

Djinn Security

Using Java 2 security one can restrict which clients can access a lookup service

There is little control over what a client can do once it connects to the lookup service

Any client that can access a lookup service can:

   Add services to a lookup service
Access the admin methods on a lookup service

Future versions of Jini will add security.

---

**Multicast**

Discovery & Join use UDP multicast packets

There are two important parameters to understand when using UDP multicast packets - TTL and MTU

**Time-to-Live (TTL)**

Each time a multicast datagram goes through a router, the TTL field of the datagram is reduced by one. When a router is overloaded and delays sending the datagram, the router reduces the TTL field by the number of seconds the datagram is delayed in the router. When the TTL field reaches zero, the datagram is discarded and a time exceeded message is sent to the datagram’s source. TTL is sometimes called “hop count”.

The property `net.jini.discovery.ttl` controls the TTL value used in the discovery process.

**Maximum Transfer Unit (MTU)**

Each packet-switching technology has an upper bound on the number of bits in an individual frame. This upper limit is the MTU. Datagrams larger than the MTU are sent using multiple frames. Jini attempts to keep a multicast datagram in one frame. It assumes that the MTU is 512 bytes. To change this value set the `net.jini.discovery.mtu` property.

---

**Discovery**

There are three discovery protocols:

- **Multicast Request Protocol**
  
  Discover lookup services(es) on a LAN via multicast UDP

- **Multicast Announcement Protocol**
  
  Announce a lookup service on a LAN via multicast UDP

- **Unicast Discovery Protocol**
  
  Connect with a lookup service via unicast TCP
Unicast Discovery Protocol

Unicast discovery is done using the LookupLocator.
From doc 18, slide 5 we have the following example of unicast discovery:

```java
public class HelloClient
{
    public static void main (String[] args) throws Exception
    {
        System.setSecurityManager (new RMISecurityManager ());
        LookupLocator lookup = new LookupLocator ("jini://eli.sdsu.edu");
        ServiceRegistrar registrar = lookup.getRegistrar ();
    }
}
```

Multicast Announcement Protocol

Used by lookup services to announce their existence

Will not cover the details. We will focus on the client and multicast request protocol

Multicast Request Protocol

Lookup services listen for multicast UDP requests

Lookup services belong to one or more group. A lookup service looks UDP requests for references for the their groups

Client opens a TCP socket to accept responses from a lookup service

The request contains the groups of interest to the client and information to the lookup service to allow it to contact the client via TCP.

Each Lookup service in the requested group(s) returns a ServiceRegistrar object to the client

```
net.jini.discovery.LookupDiscovery
```

Handles the multicast request protocol for clients

Clients provide a LookupDiscovery object a list of groups. The LookupDiscovery object creates a thread
which operates the multicast request protocol. Each time the LookupDiscovery object finds a lookup service in its list of groups, the LookupDiscovery object informs all DiscoveryListener’s that are registered with the LookupDiscovery object. The list of groups can be modified. The LookupDiscovery object continues to operate until it receives the terminate() message, the JVM exits or the object is garbage collected. In order to use the discovery process the DiscoveryPermission must be set.

**Constructor**

`LookupDiscovery(java.lang.String[] groups)`

Construct a new lookup discovery object, set to discover the given set of groups.

**Methods**

- `addDiscoveryListener(DiscoveryListener l)`
- `addGroups(java.lang.String[] newGroups)`
- `discard(ServiceRegistrar reg)`
- `finalize()`
- `getGroups()`
- `removeDiscoveryListener(DiscoveryListener l)`
- `removeGroups(java.lang.String[] oldGroups)`
- `setGroups(java.lang.String[] newGroups)`
- `terminate()`

**net.jini.discovery.DiscoveryPermission**

In order to use the discovery process the DiscoveryPermission must be set in the policy file for both the Lookup service and for the client code. The following the is the format for this permission. See document 7, slide 7-26 for more information on permissions.

```grant
{
  permission net.jini.discovery.DiscoveryPermission "groupName";
};
```

The table below gives the strings that have special meaning in the `groupName` entry. Use multiple permission entries to list more than one group.

<table>
<thead>
<tr>
<th><code>groupName</code></th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&quot;&quot;</code></td>
<td>public group</td>
</tr>
<tr>
<td><code>&quot;*&quot;</code></td>
<td>all groups</td>
</tr>
<tr>
<td><code>&quot;*.sdsu.edu&quot;</code></td>
<td>all groups name ending in &quot;.sdsu.edu&quot;</td>
</tr>
</tbody>
</table>

**Interface net.jini.discovery.DiscoveryListener**

Client code that handles the interacts with lookup services found by a LookupDiscovery object needs to implement this interface. The client code is informed with lookup services are found or dropped. Lookup services can be dropped when a lookup service drops out of a group or when a LookupDiscovery object list of groups is modified. The methods discarded and discovered should return quickly as they block the discovery thread. The docs state these methods should not make remote calls.
Methods

discarded(DiscoveryEvent e)
discovered(DiscoveryEvent e)

net.jini.discovery.DiscoveryEvent

DiscoveryListeners are given DiscoveryEvent objects. The getRegistrar() method of DiscoveryEvent class returns service registrar for the discovered lookup services. The client can now interact with the lookup service.

Method

ServiceRegistrar[] getRegistrars()

Multicast Discovery Example

The following example illustrates the discovery process

It shows all visible lookup services for 15 minutes

While the example does not access any service in the lookup, once it has a ServiceRegistrar object it could access and use services.

Discovery Example Code

package whitney.jini.examples.discoveryJoin;
import net.jini.discovery.LookupDiscovery;
import net.jini.discovery.DiscoveryListener;
import net.jini.core.lookup.ServiceRegistrar;
import java.rmi.RemoteException;
import net.jini.core.lookup.ServiceRegistration;
import java.io.IOException;
import java.rmi.RMISecurityManager;
public class DiscoverAll implements DiscoveryListener{
    private static final int MILLSECS_PER_MINUTE = 1000 * 60;
    public static void main( String[] arguments )
        throws IOException, InterruptedException {
            System.setSecurityManager (new RMISecurityManager ());
            LookupDiscovery findAllLookupServices =
                new LookupDiscovery(LookupDiscovery.ALL_GROUPS );
            findAllLookupServices.addDiscoveryListener( new DiscoverAll() );
            System.out.println( "Start looking");
            // Look for 15 minutes
            Thread.currentThread().sleep( MILLSECS_PER_MINUTE * 15 );
            findAllLookupServices.terminate();
        }
    public void discovered(DiscoveryEvent lookupService ) {
        System.out.println( "A lookup service discovered");
        displayEvent( lookupService);
    }
    public void discarded(DiscoveryEvent lookupService) {
        System.out.println( "A lookup service discarded");
        displayEvent( lookupService);
    }
private void displayEvent(DiscoveryEvent event) {
  try {
    ServiceRegistrar lookupServices[] = event.getRegistrars();
    for (int i = 0; i < lookupServices.length; i++)
      displayService(lookupServices[i]);
  } catch (Exception lookupProblem) {
    lookupProblem.printStackTrace();
  }
}

private void displayService(ServiceRegistrar service)
  throws RemoteException {
  System.out.println("Registrar " + service.getLocator());
  String[] serviceGroups = service.getGroups();
  if (serviceGroups.length > 0) {
    System.out.println("Groups of registrar ");
    for (int k=0; k< serviceGroups.length; k++)
      System.out.println( serviceGroups[k] );
  }
}