◆ ◆ ◆ CHAPTER 11

RMI-IIOP Load Balancing and Failover

This chapter describes using high-availability features for remote EJB references and JNDI objects over RMI-IIOP in GlassFish Server.

- "Overview" on page 155
- "`InitialContext` Load Balancing" on page 156
- "Per-Request Load Balancing (PRLB)" on page 160

Overview

With RMI-IIOP load balancing, IIOP client requests are distributed to different server instances or name servers. The goal is to spread the load evenly across the cluster, thus providing scalability. IIOP load balancing combined with EJB clustering and availability also provides EJB failover.

The following topics are addressed here:

- "General Requirements for Configuring Load Balancing" on page 155
- "Load Balancing Models" on page 156

General Requirements for Configuring Load Balancing

Oracle GlassFish Server provides high availability of remote EJB references and `NameService` objects over RMI-IIOP, provided all the following apply:

- Your deployment has a cluster of at least two instances.
- Java EE applications are deployed to all instances and clusters that participate in load balancing.
- RMI-IIOP client applications are enabled for load balancing.

GlassFish Server supports load balancing for Java applications executing in the Application Client Container (ACC). See "Enabling RMI-IIOP Hardware Load Balancing and Failover" on page 157.

Note – GlassFish Server does not support RMI-IIOP load balancing and failover over secure sockets layer (SSL).

Load Balancing Models

GlassFish Server supports two general models for load balancing:

"InitialContext Load Balancing" on page 156
When a client performs a JNDI lookup for an object, the Naming Service creates a `InitialContext` (IC) object associated with a particular server instance. From then on, all lookup requests made using that IC object are sent to the same server instance. `InitialContext` load balancing can be configured automatically across an entire cluster.

"Per-Request Load Balancing (PRLB)" on page 160
Per Request Load Balancing (PRLB) is a method for load balancing stateless EJBs that enables load-balancing for each request to an EJB instance. PRLB chooses the first node in a cluster to use on each request. PRLB is configured on a per-EJB basis.

`InitialContext` Load Balancing

The following topics are addressed here:

- "InitialContext Summary" on page 156
- "InitialContext Algorithm" on page 157
- "Enabling RMI-IIOP Hardware Load Balancing and Failover" on page 157

`InitialContext` Summary

When `InitialContext` load balancing is used, the client calls the `InitialContext()` method to create a new `InitialContext` (IC) object that is associated with a particular server instance. JNDI lookups are then performed on that IC object, and all lookup requests made using that IC object are sent to the same server instance. All `EJBHome` objects looked up with that `InitialContext` are hosted on the same target server. Any bean references obtained henceforth are also created on the same target host. This effectively provides load balancing, since all clients randomize the list of live target servers when creating `InitialContext` objects. If the target server instance goes down, the lookup or EJB method invocation will failover to another server instance. All objects derived from same `InitialContext` will failover to the same server instance.

IIOP load balancing and failover happens transparently. No special steps are needed during application deployment. IIOP load balancing and failover for the GlassFish Server supports dynamically reconfigured clusters. If the GlassFish Server instance on which the application client is deployed participates in a cluster, the GlassFish Server finds all currently active IIOP endpoints in the cluster automatically. Therefore, you are not required to manually update the list of endpoints if a new instance is added to the cluster or deleted from the cluster. However, a client should have at least two endpoints specified for bootstrapping purposes, in case one of the endpoints has failed.

`InitialContext` Algorithm

GlassFish Server uses a randomization and round-robin algorithm for RMI-IIOP load balancing and failover.

When an RMI-IIOP client first creates a new `InitialContext` object, the list of available GlassFish Server IIOP endpoints is randomized for that client. For that `InitialContext` object, the load balancer directs lookup requests and other `InitialContext` operations to an endpoint on the randomized list. If that endpoint is not available then a different random endpoint in the list is used.

Each time the client subsequently creates a new `InitialContext` object, the endpoint list is rotated so that a different IIOP endpoint is used for `InitialContext` operations. The rotation is randomized, so the rotation is not to the next endpoint in the list, but instead to a random endpoint in the list.

When you obtain or create beans from references obtained by an `InitialContext` object, those beans are created on the GlassFish Server instance serving the IIOP endpoint assigned to the `InitialContext` object. The references to those beans contain the IIOP endpoint addresses of all GlassFish Server instances in the cluster.

The *primary endpoint* is the bean endpoint corresponding to the `InitialContext` endpoint used to look up or create the bean. The other IIOP endpoints in the cluster are designated as *alternate endpoints*. If the bean's primary endpoint becomes unavailable, further requests on that bean fail over to one of the alternate endpoints.

You can configure RMI-IIOP load balancing and failover to work with applications running in the ACC.

Enabling RMI-IIOP Hardware Load Balancing and Failover

You can enable RMI-IIOP load balancing and failover for applications running in the application client container (ACC). Weighted round-robin load balancing is also supported.

To Enable RMI-IIOP Hardware Load Balancing for the Application Client Container

This procedure provides an overview of the steps necessary to enable RMI-IIOP load balancing and failover with the application client container (ACC). For additional information on the ACC, see "Developing Clients Using the ACC" in *GlassFish Server Open Source Edition 3.1 Application Development Guide*.

Before You Begin The first five steps in this procedure are only necessary if you are enabling RMI-IIOP load balancing on a system other than the DAS. This is common in production environment, but less common in a development environment. For example, a developer who wants to experiment with a cluster and load balancing might create two instances on the same system on which the DAS is running. In such cases, steps 1-5 are unnecessary.

1. **Go to the *install_dir* `/bin` directory.**

2. **Run `package-appclient`.**

   This utility produces an `appclient.jar` file. For more information on `package-appclient`, see `package-appclient`(1M).

3. **Copy the `appclient.jar` file to the machine where you want your client and extract it.**

4. **Edit the `asenv.conf` or `asenv.bat` path variables to refer to the correct directory values on that machine.**

   The file is at *appclient-install-dir* `/config/`.

   For a list of the path variables to update, see `package-appclient`(1M).

5. **If required, make the `appclient` script executable.**

   For example, on UNIX use `chmod 700`.

6. **Find the IIOP listener port number for at least two instances in the cluster.**

   You specify the IIOP listeners as endpoints in Step 7.

   For each instance, obtain the IIOP listener ports as follows:

   a. **Verify that the instances for which you want to determine the IIOP listener port numbers are running.**

      ```
      asadmin> list-instances
      ```

      A list of instances and their status (running, not running) is displayed.

      The instances for which you want to display the IIOP listener ports must be running.

b. For each instance, enter the following command to list the various port numbers used by the instance.

```
asadmin> get servers.server.instance-name.system-property.*.value
```

For example, for an instance name `in1`, you would enter the following command:

```
asadmin> get servers.server.in1.system-property.*.value
```

7 Add at least two `target-server` elements in the `sun-acc.xml` file.

Use the endpoints that you obtained in Step 6.

If the GlassFish Server instance on which the application client is deployed participates in a cluster, the ACC finds all currently active IIOP endpoints in the cluster automatically. However, a client should have at least two endpoints specified for bootstrapping purposes, in case one of the endpoints has failed.

The `target-server` element specifies one or more IIOP endpoints used for load balancing. The `address` attribute is an IPv4 address or host name, and the `port` attribute specifies the port number. See "client-container" in *GlassFish Server Open Source Edition 3.1 Application Deployment Guide*.

As an alternative to using `target-server` elements, you can use the `endpoints` property as follows:

```
jvmarg value = "-Dcom.sun.appserv.iiop.endpoints=host1:port1,host2:port2,..."
```

8 If you require weighted round-robin load balancing, perform the following steps:

a. Set the load-balancing weight of each server instance.

```
asadmin set instance-name.lb-weight=weight
```

b. In the `sun-acc.xml`, set the `com.sun.appserv.iiop.loadbalancingpolicy` property of the ACC to `ic-based-weighted`.

```
...
<client-container send-password="true">
  <property name="com.sun.appserv.iiop.loadbalancingpolicy" \
     value="ic-based-weighed"/>
...
```

9 Deploy your client application with the `--retrieve` option to get the client jar file.

Keep the client jar file on the client machine.

For example:

```
asadmin  --user admin --passwordfile pw.txt deploy --target cluster1 \
--retrieve my_dir myapp.ear
```

10 Run the application client as follows:

```
appclient --client my_dir/myapp.jar
```

Example 11–1 Setting Load-Balancing Weights for RMI-IIOP Weighted Round-Robin Load Balancing

In this example, the load-balancing weights in a cluster of three instances are to be set as shown in the following table.

Instance Name	Load-Balancing Weight
i1	100
i2	200
i3	300

The sequence of commands to set these load balancing weights is as follows:

```
asadmin set i1.lb-weight=100
asadmin set i2.lb-weight=200
asadmin set i3.lb-weight=300
```

Next Steps To test failover, stop one instance in the cluster and see that the application functions normally. You can also have breakpoints (or sleeps) in your client application.

To test load balancing, use multiple clients and see how the load gets distributed among all endpoints.

See Also See "Enabling the High Availability Session Persistence Service" on page 138 for instructions on enabling the session availability service for a cluster or for a Web, EJB, or JMS container running in a cluster.

Per-Request Load Balancing (PRLB)

The following topics are addressed here:

- "PRLB Summary" on page 160
- "Enabling Per-Request Load Balancing" on page 161

PRLB Summary

Per Request Load Balancing (PRLB) is a method for load balancing stateless EJBs that enables load-balancing for each request to an EJB instance. PRLB chooses the first node in a cluster to use on each request. By contrast, `InitialContext` (hardware) load balancing chooses the first node to use when the `InitialContext` is created, and each request thereafter uses the same node unless a failure occurred.

PRLB is enabled by means of the boolean `per-request-load-balancing` property in the `glassfish-ejb-jar.xml` deployment descriptor file for the EJB. If this property is not set, the original load balancing behavior is preserved.

Note – PRLB is only supported for stateless session beans. Using PRLB with any other bean types will result in a deployment error.

Enabling Per-Request Load Balancing

You can enable Per-Request Load Balancing (PRLB) by setting the boolean `per-request-load-balancing` property to `true` in the `glassfish-ejb-jar.xml` deployment descriptor file for the EJB. On the client side, the `initContext.lookup` method is used to access the stateless EJB.

▼ To Enable RMI-IIOP Per-Request Load Balancing for a Stateless EJB

This procedure describes how to enable PRLB for a stateless EJB that is deployed to clustered GlassFish Server instances. This procedure also provides an client-side example for accessing a stateless EJB that uses PRLB.

1 Choose or assemble the EJB that you want to deploy.

In this example, an EJB named `TheGreeter` is used.

For instructions on developing and assembling an EJB for deployment to GlassFish Server, refer to the following documentation:

- Chapter 8, "Using Enterprise JavaBeans Technology," in *GlassFish Server Open Source Edition 3.1 Application Development Guide*
- "EJB Module Deployment Guidelines" in *GlassFish Server Open Source Edition 3.1 Application Deployment Guide*
- "Assembling and Deploying an Application Client Module" in *GlassFish Server Open Source Edition 3.1 Application Deployment Guide*

2 Set the `per-request-load-balancing` property to `true` in the `glassfish-ejb-jar.xml` deployment descriptor file for the EJB.

For more information about the `glassfish-ejb-jar.xml` deployment descriptor file, refer to "The glassfish-ejb-jar.xml File" in *GlassFish Server Open Source Edition 3.1 Application Deployment Guide*

For example, the `glassfish-ejb-jar.xml` file for a sample EJB named `TheGreeter` is listed below.

```
<glassfish-ejb-jar>
  <enterprise-beans>
    <unique-id>1</unique-id>
```

```
    <ejb>
      <ejb-name>TheGreeter</ejb-name>
    <jndi-name>greeter</jndi-name>
    <per-request-load-balancing>true</per-request-load-balancing>
    </ejb>
  </enterprise-beans>
</glassfish-ejb-jar>
```

3 Deploy the EJB.

If the EJB was previously deployed, it must be redployed.

For instructions on deploying EJBs, refer to the following documentation:

- "To Deploy an Application or Module" in *GlassFish Server Open Source Edition 3.1 Application Deployment Guide*
- "To Redeploy an Application or Module" in *GlassFish Server Open Source Edition 3.1 Application Deployment Guide*

4 (Optional) Verify the PRLB configuration by looking for the following `FINE` message in the CORBA log file:

`Setting per-request-load-balancing policyfor EJB` *EJB-name*

5 Configure a client application to access the PRLB-enabled EJB.

For example:

```
public class EJBClient {
    public static void main(String args[]) {
    :
    :
    :
    try {
    // only one lookup

    Object objref = initContext.lookup("test.cluster.loadbalancing.ejb.\
      TestSessionBeanRemote");
        myGreeterRemote = (TestSessionBeanRemote)PortableRemoteObject.narrow\
          (objref,
                                              TestSessionBeanRemote.class);

    } catch (Exception e) {
    :
    }

    for (int i=0; i < 10; i++ ) {
    // method calls in a loop.
        String theMessage = myGreeterRemote.sayHello(Integer.toString(i));
        System.out.println("got"+": " + theMessage);
        }
    }
}
```

See Also See "Enabling the High Availability Session Persistence Service" on page 138 for instructions on enabling the session availability service for a cluster or for a Web, EJB, or JMS container running in a cluster.