Using ICEfaces in Clustered-Failover Environments

ICEfaces now provides support for failover. Failover, in this case, being defined as the ability for an ICEfaces application to run in a cluster and, when a failure occurs, switch over to redundant node in the cluster. This section outlines the key features required to support failover as well as noting the important areas the developer should keep in mind when creating an application that will be deployed to a cluster.

Session Serialization

An important requirement for supporting failover is session serialization, which is now supported in ICEfaces. Session serialization allows sessions to be persisted and/or copied across nodes in a cluster. When a node fails, these copied sessions can be used by other nodes to continue normal operations. Note that JSF managed beans that are session-scoped, as well as object references inside those beans, need to be properly implemented for this to work. This means that session-scoped beans should implement Serializable and that member variables should also be Serializable or marked as transient.

Reload on Failover

When a failover condition is detected by ICEfaces, the framework will prompt the client to reload the page. This ensures that the required objects related to the user's session are created on the new node and that they are in the proper state to resume normal operations.

Configuring an ICEfaces Application

ICEfaces

There are no application configuration changes that are specific for ICEfaces to support failover. If your application uses Ajax Push to drive updates from the server-side, there are some things you may need to adjust in order to have the Ajax Push functionality failover seamlessly.

- Use the Renderable API rather than the SessionRenderer. This means implementing the Renderable interface on your managed bean. The SessionRenderer API will be supported in a future release.
- The Renderable interface must be implemented on a request-scoped (not session-scoped) bean. Session-scoped beans will be supported in a future release.
- Ensure all logic related to joining render groups is done during bean creation and leaving render groups is done during bean disposal. You can implement the DisposableBean interface to help with this.

When a failover occurs, an ICEfaces application that utilizes Ajax Push will attempt to renegotiate the blocking connection used to notify the client that there are updates. The algorithm used to do this tries to be as robust as possible as the failover to a working node can take some time. If the new node is not yet ready to accept the incoming request, an error can result. The ICEfaces client bridge manages this by making multiple attempts with a cumulative delay between each attempt. The number of attempts and the delay between each attempt are configurable using a context parameter. The context parameter "com.icesoft.faces.serverErrorRetryTimeouts" is used to configure the retry number and delay used for re-establishing the blocking connection after receiving HTTP 500 errors. The parameter takes space separated values, which are treated as the delay between retries and are measured in milliseconds. If the parameter is not specified, the following default is used:

```
<context-param>
   <param-name>com.icesoft.faces.serverErrorRetryTimeouts</param-name>
   <param-value>1000 2000 4000</param-value>
</context-param>
```

This indicates that the bridge should make three attempts with 1000 ms before the first attempt, 2000 ms before the second, and 4000 ms before the third. You can override this by specifying your own number of retries with different delay values. For example, the following settings would set the number of retries to only two, with 1500 ms before it tried the first and 3000 ms before it tried the second:

```
<context-param>
   <param-name>com.icesoft.faces.serverErrorRetryTimeouts</param-name>
   <param-value>1500 3000</param-value>
</context-param>
```

Tomcat

To deploy an application into a Tomcat cluster, you need to add the <distributable/> element to the deployment descriptor (web.xml) of your application.

Sample Application

To illustrate theses points, the ICEfaces bundle includes a version of the TimeZone tutorial application that has been modified to work in a clustered environment. It can gracefully resume operation during a failover from one node to another. The application itself can be found at:

```
[icefaces_dir]/samples/tutorial/timezone-failover
```

To build a working .war file, use your console to change to the directory and type:

```
ant
```

The default build is for Tomcat 6. This version of the TimeZone application has the following characteristics:

- The <distributable/> element has been added to the deployment descriptor (web.xml).
- The TimeZoneWrapper bean now implements Serializable so that an instance can be persisted into the session.
- The state-related information from TimeZoneBean has been removed and put into it's own class, TimeZoneSelections, which is now a JSF-managed, session-scoped bean. This bean now keeps track of the state is automatically injected into the user's session.
- The TimeZoneBean, a request-scoped bean, implements the Renderable interface and manages adding and removing itself from the appropriate render group. This means that when a failover occurs, the incoming request properly runs through the logic to support Ajax Push.

Configuring the Cluster

Supported Environments

The basic configuration for failover is to have a load-balancer handle requests which are proxied to two or more nodes in an application server cluster. Configuring a cluster for ICEfaces is described in more detail in the section Chapter 5 Advanced Topics: Introduction to the Asynchronous HTTP Server. In that section there is documentation for setting up support for several supported application servers. In this release, the currently supported configurations for failover are:

Application Server Clusters
- Tomcat 6.0.18 or higher

Load Balancers
- Apache http 2.2.10 or higher

Configuring the Application Server

For full documentation on setting up a cluster of Tomcat nodes, you should refer to the documentation at the Apache website:

http://tomcat.apache.org/tomcat-6.0-doc/cluster-howto.html

There are options and configuration parameters for customizing the behavior of the cluster so that it suits your environment. A short summary of the minimum requirements is as follows:

To enable a single Tomcat instance for clustering, uncomment the Cluster element in the [tomcat.dir]/conf/server.xml file. You'll need to do this for each instance that will participate as a node in the cluster:

```
<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>
```

To allow Apache to load balance between the nodes in your cluster, add a jvmRoute attribute to the Engine element. Each node should have a unique jvmRoute name.

```
<Engine name="Catalina" defaultHost="localhost" jvmRoute="node1">
```

Configuring the Web Server

You'll need to install the Apache web server and include support for mod_proxy.

```
<IfModule mod_proxy.c>

    ProxyRequests Off

    <Proxy *>
      Order deny,allow
      Allow from all
    </Proxy>

    ProxyPass /application-name balancer://application-server-cluster/application-name
stickysession=JSESSIONID lbmethod=byrequests nofailover=Off
    ProxyPassReverse /application-name ajp://host1-name:host1-port/application-name
    ProxyPassReverse /application-name ajp://host2-name:host2-port/application-name

    <Proxy balancer://application-server-cluster >
      BalancerMember ajp://host1-name:host1-port route=node1
      BalancerMember ajp://host2-name:host2-port route=node2
    </Proxy>

</IfModule>
```

Note: Currently AJP is recommended over HTTP as the load balancing protocol.