Session Clustering with a Database

This feature is only available since release jetty-7.0pre2

Session Clustering using a Database

Jetty can support session clustering by persisting sessions to a shared database. Each jetty instance locally caches sessions for which it has received requests, writing any changes to the session through to the database as the request exits the server. Sessions must obey the Serialization contract, and servlets must call the Session.setAttribute() method to ensure that changes are persisted.

The persistent session mechanism is designed to work in conjunction with a load balancer that supports stickiness. Stickiness can be based on various data items, such as source IP address or characteristics of the session id or a load-balancer specific mechanism. For those load balancers that examine the session id, the Jetty persistent session mechanism appends a node id to the session id which can be used for routing.

It should be noted that in this type of solution, the database can become both a bottleneck and a single point of failure. Jetty takes steps to try to reduce the load on the database (discussed below), but in a heavily loaded environment you may need to investigate other optimisation strategies such as local caching and database replication. You should also consult your database vendor's documentation for information on how to ensure high-availability and fail-over of your database.

Configuration

There are 2 components to session management in Jetty: a session id manager and a session manager. The session id manager's job is to ensure that session ids are unique across all webapps hosted on a jetty instance and thus there can only be one per jetty instance. The session manager's job is to handle the session lifecycle (create/update/invalidate/expire) on behalf of a web application, and thus there is one per web application instance. They also cooperate and collaborate with the org.mortbay.handler.SessionHandler to enable cross-context dispatch.

Configuring the JDBCSessionIdManager

We need to configure an org.mortbay.jetty.servlet.JDBCSessionIdManager instance, either in embedded code or in a jetty.xml file. Here is an example of a jetty.xml setup:

```
<Set name="sessionIdManager">
    <New id="jdbcidmgr"
class="org.mortbay.jetty.servlet.JDBCSession
IdManager">
        <Arg><Ref id="Server"/></Arg>
        <Set name="workerName">fred</Set>
        <Set
name="DatasourceName">javax.sql.DataSource/d
efault</Set>
        <Set
name="scavengeInterval">60</Set>
    </New>
</Set>
<Call name="setAttribute">
      <Arg>jdbcIdMgr</Arg>
      <Arg><Ref id="jdbcidmgr"/></Arg>
</Call>
```

You'll notice that the JDBCSessionIdManager needs access to a database. The above configures it with the name of a javax.sql.DataSource that is defined elsewhere. Consult Jetty Naming Resources for more information on how to configure database access with jetty. If you don't wish to use a DataSource, you can configure jdbc Driver information instead. Here's an example:

```
<Set name="sessionIdManager">
    <New id="jdbcidmgr"
class="org.mortbay.jetty.servlet.JDBCSession
IdManager">
        <Arg><Ref id="Server"/></Arg>
        <Set name="workerName">fred</Set>
        <Call name="setDriverInfo">
          <Arg>com.mysql.jdbc.Driver</Arg>

<Arg>jdbc:mysql://127.0.0.1:3306/sessions?us
er=janb</Arg>
        </Call>
        <Set
name="scavengeInterval">60</Set>
      </New>
</Set>
<Call name="setAttribute">
      <Arg>jdbcIdMgr</Arg>
      <Arg><Ref id="jdbcidmgr"/></Arg>
</Call>
```

As jetty config files are direct mappings of xml to java, it is quite straightforward to see how this would be done in code, but here's an example anyway:

```
Server server = new Server();
    ...
JDBCSessionIdManager idMgr = new
JDBCSessionIdManager(server);
idMgr.setWorkerName("fred");
idMgr.setDriverInfo("com.mysql.jdbc.Driver",
"jdbc:mysql://127.0.0.1:3306/sessions?user=j
anb");
idMgr.setScavengeInterval(60);
server.setSessionIdManager(idMgr);
```

The JDBCSessionIdManager MUST be configured with a **workerName** which is a name unique across the cluster. Typically it relates to the physical node on which the instance is executing. If this name is not unique, your load balancer may fail to distribute your sessions correctly.

You can also configure how often the persistent session mechanism sweeps the database looking for old, expired sessions with the **scavengeInterval** setting. By default, this is set to **60seconds**. We recommend that you do not increase the frequency as you will increase the load on the database with very little gain, as old expired sessions can harmlessly sit in the database.

Configuring a JDBCSessionManager

The way you configure a JDBCSessionManager is a little different depending on whether you're configuring from a context xml file or a jetty-web.xml file, or code. The basic difference is how you get a reference to the Jetty org.mortbay.jetty.server.Jetty instance.

From a context xml file, you reference the Server instance as a Property:

```
<Property name="Server" id="Server">
  <Call id="jdbcIdMgr" name="getAttribute">
    <Arg>jdbcIdMgr</Arg>
  </Call>
</Property>

<Set name="sessionHandler">
   <New
class="org.mortbay.jetty.servlet.SessionHand
ler">
     <Arg>
       <New id="jdbcmgr"
class="org.mortbay.jetty.servlet.JDBCSession
Manager">
         <Set name="idManager">
           <Ref id="jdbcIdMgr"/>
         </Set>
       </New>
     </Arg>
   </New>
</Set>
```

From a WEB-INF/jetty-web.xml file, you can reference the Server instance directly:

```
<Get name="server">
   <Get id="jdbcIdMgr"
name="sessionIdManager"/>
</Get>
<Set name="sessionHandler">
   <New
class="org.mortbay.jetty.servlet.SessionHand
ler">
     <Arg>
       <New
class="org.mortbay.jetty.servlet.JDBCSession
Manager">
         <Set name="idManager">
           <Ref id="jdbcIdMgr"/>
         </Set>
       </New>
     </Arg>
   </New>
</Set>
```

If you're embedding this in code:

```
//assuming you have already set up the
JDBCSessionIdManager as shown earlier
//and have a reference to the Server
instance:

WebAppContext wac = new WebAppContext();
 ... //configure your webapp context
JDBCSessionManager jdbcMgr = new
JDBCSessionManager();
jdbcMgr.setIdManager(server.getSessionIdMana
ger());
wac.setSessionHandler(jdbcMgr);
```