TomcatHibernate

Tomcat, Hibernate, and Database Connection Pooling

Introduction

This is my understanding concerning how the Tomcat / Hibernate / database environment works. This is based on a post I made to the Tomcat Users mailing list. In this wiki article, I'll work through both Tomcat - managed database connection pooling and Hibernate - managed database connection pooling.

Environments

I use primarily the following two environments when developing concepts and testing ideas.

Component	Version
OS	Fedora 11 32 bit
JDK/JRE	1.6.0_20
Tomcat	6.0.26
MySQL	5.1.41-2
IDE	NetBeans 6.8
Hibernate	3.2.5 ga
OS	Windows/XP Professional SP 4 32 bit
JDK/JRE	1.6.0_20
Tomcat	6.0.26
MySQL	5.1.31
IDE	NetBeans 6.8
Hibernate	3.2.5 ga

In order to work through the two configurations, I've chosen to use the NetBeans Hibernate Tutorial. The original tutorial does not make any use of database pooling. Modifications to the tutorial will be shown below to use either Tomcat's database pooling via JNDI or Hibernate's provided C3P0 database pooling.

Preliminaries

Before proceeding, either work through or download the Netbeans Hibernate Tutorial. This will ensure that the project is set up properly so that modifications can be easily made.

Tomcat JNDI Database Pooling Basics

Tomcat Preparation

Per Tomcat documentation, I've placed mysql-connector-java-5.1.11-bin.jar in Tomcat's lib directory. This makes the JDBC driver available to Tomcat in order to set up database pooling. No Hibernate jars should be placed in Tomcat's lib directory.

Web Application

All Hibernate jars are located in WEB-INF/lib. NetBeans builds the war file correctly, so no jar files have to be copied.

Per Tomcat documentation, a context.xml file in META-INF is created.

```
<?xml version="1.0" encoding="UTF-8"?> <Context antiJARLocking="true" path="/DVDStore">
  <Resource auth="Container"
  driverClassName="com.mysql.jdbc.Driver" maxActive="30" maxIdle="10" maxWait="10000" name="jdbc/sakila"
password="*****"
  type="javax.sql.DataSource" url="jdbc:mysql://localhost/sakila" username="*****"/>
</Context>
```

Obviously, replace the asterisks with the appropriate username and password. Replace the database url with your appropriate database.

In WEB-INF/web.xml you'll need to add the following lines.

```
<resource-ref>
  <description>This is a MySQL database connection</description>
  <res-ref-name>jdbc/sakila</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>
```

Again, this is all boilerplate configuration and is documented on the Tomcat documentation web site. In particular, please note that the **res-ref-name** in web.xml **must match** the **name** attribute of the resource node in the context.xml file.

Hibernate Configuration

So far, this has all been more or less boilerplate Tomcat configuration. The next step is to get Hibernate to talk to Tomcat's database connection pool as opposed to its own connection pool or a direct connection.

After searching a bit on the Internet, I found that the following connection information works.

In hibernate.cfg.xml located in WEB-INF/, put the following.

```
<!-- using container-managed JNDI -->
<propertyname="hibernate.connection.datasource">
   java:comp/env/jdbc/sakila
</property>
```

Note that jdbc/sakila matches the web.xml resource-ref-name, which matches the name attribute in context.xml. java:comp/env/ is the namespace to look up the jdbc reference.

No other connection or pooling information should be present in your hibernate.cfg.xml file. Note, you still need the **database dialect** properties in order for Hibernate to function.

If you are using Netbeans as your IDE, the hibernate.cfg.xml file will be found in <project-name>/src/java by default.

Application Code

One of the nice things about Hibernate is that it shields your application code from database connection particulars. You can change from Tomcat JNDI pooling to Hibernate pooling, to a direct connection all without changing the underlying application code. In particular:

Creating the SessionFactory

```
// SessionFactory from standard hibernate.cfg.xml file
try {
    sessionFactory = new AnnotationConfiguration().configure().buildSessionFactory();
} catch (Throwable ex) {
    // Log the exception.
    log.fatal("Initial SessionFactory creation failed.", ex); throw new ExceptionInInitializerError(ex);
}
```

and getting the SessionFactory

```
public static SessionFactory getSessionFactory() {
    return sessionFactory;
}
```

work with no modifications.

For my application, both of these code snippets live in HibernateUtil.java which NetBeans generates for you. My code also has an mbean for monitoring Hibernate.

You can then call this code from a class that implements ServletContextListener. This will then create the session factory, or throw an exception if this fails at application startup. It's also handy for cleaning up resources when the application is shut down. In my case, I use this to unregister the monitoring mbean.

Other Notes

If you look at your logs, you may see the following warning:

```
INFO SessionFactoryObjectFactory:82 - Not binding factory to JNDI, no JNDI name configured
```

This is due to the fact that Hibernate can bind a session factory to a JNDI name, but none is configured. This is not the JDBC resource. This facility probably exists to make sharing session factories easier, and to clean up calls like the following:

```
Session session = HibernateUtil.getSessionFactory().openSession();
```

Tomcat provides a read-only InitialContext, while Hibernate requires read-write in order to manage multiple session factories. Tomcat is apparently following the specification for unmanaged containers. If you want to bind the session factory to a JNDI object, you'll either have to move to a managed server (Glassfish, JBoss, etc.), or search on the Internet for some posted work-arounds.

The recommendation from the Hibernate documentation is to just leave out the hibernate.session_factory_name property when working with Tomcat to not try binding to JNDI.

Hibernate Managed Database Pooling

I've tried this only briefly, since I always rely on container managed database connections. However, it appears to work cleanly, and may be useful when frequently migrating between servlet containers.

Web Application Configuration

1. Remove the <Resource></Resource> element from the context.xml file. Tomcat is not going to be managing the database connection.
2. Remove the <resource-ref></resource-ref> element from the web.xml file. Tomcat is not going to be managing the database connection.

Jar Files

According to the Tomcat classloader documentation, Tomcat makes jar files available to your application in the following order:

1. Bootstrap classes of your JVM
2. System class loader classses (described above)
3. /WEB-INF/classes of your web application
4. /WEB-INF/lib/*.jar of your web application
5. $CATALINA_HOME/lib
6. $CATALINA_HOME/lib/*.jar

In particular, the following two pieces of information should be noted.

1. The jar files in lib/ are made available to both the application and to Tomcat. Database connection jars are placed here if the application will be relying on Tomcat to manage connection pooling.
2. In general, application-specific jars (such as Hibernate's connection pooling jars) should NOT be placed in the lib/ directory of Tomcat.

Based on the above, the MySQL connection jar will need to be placed in the WEB-INF/lib directory.

Approach Using NetBeans

In order to do this with NetBeans, add the jar file to the project. An easy way to do this on a per-project basis is the following:

1. Right-mouse click on the project name and select "Properties" from the menu.
2. Select the "Libraries" item and the "Compile" tab. Compile-time libraries will be propagated to all other tasks.
3. Click on the "Add Jar/Folder" button, and navigate to the jar file containing the appropriate JDBC driver.
4. When this project is built, the JDBC driver jar will be included in the WAR file in WEB-INF/lib.

Please note that the above method will cause all sorts of problems if you are sharing this project or using SCM. Some more portable ways of adding third party jars include adding the jar as a NetBeans library, creating a library project, or using Maven and placing the dependency in the POM. All of these approaches are beyond the scope of this article.

Do not also copy the JDBC driver jar to $CATALINA_HOME/lib.

Hibernate ships with the C3P0 connection pooling classes, so as long as the Hibernate jars are in WEB-INF/lib directory (which they should be), they should be available.

Hibernate Application Configuration

Since Hibernate will be managing both the connection and the database pooling, Hibernate will have to be configured with that information.

In hibernate.cfg.xml, both connection information and connection pooling information are needed. Here are the snippets of that file.

```
<!-- connection information -->
<property name="hibernate.connection.driver_class">
    com.mysql.jdbc.Driver
</property>
<property name="hibernate.connection.url">
    jdbc:mysql://localhost/sakila
</property>
<property name="hibernate.connection.username">******</property>
<property name="hibernate.connection.password">******</property>

<!-- database pooling information -->
<property name="connection_provider_class">
    org.hibernate.connection.C3P0ConnectionProvider
</property>
<property name="c3p0.minPoolSize">5</property>
<property name="c3p0.timeout">1000</property>
```

There are probably other properties to configure, but that should get things up and running. As always, change the database properties to reflect your particular application.

Summary

Tomcat managed database connections:

- Requires META-INF/context.xml information
- Requires WEB-INF/web.xml information
- Requires a simple JNDI configuration referencing the connection
- Requires the MySQL connectors to be in $CATALINA_HOME/lib
- Requires the Hibernate jars to be in WEB-INF/lib

Hibernate managed database connections:

- Requires NO META-INF/context.xml information
- Requires NO WEB-INF/web.xml information
- Requires connection information to be in hibernate.cfg.xml
- Requires pooling information to be in hibernate.cfg.xml
- Requires MySQL connectors to be in WEB-INF/lib (ideally)
- Requires the Hibernate jars to be in WEB-INF/lib

CategoryFAQ