Hibernate Tutorial 13 Hibernate in Web Application (1)

By Gary Mak
hibernatetutorials@metaarchit.com
September 2006

1. Setting up a web application

1.1. Installing Tomcat

Tomcat is an open source application server for running J2EE web applications. You can go to http://tomcat.apache.org/ and download Tomcat 5.5. Install it into a folder say "C:\Tomcat". Note that during the installation, we need to specify the JDK installation path, not the JRE.

1.2. Creating a dynamic web project

To develop a web application, we create a dynamic web project "BookShopWeb" in Eclipse WTP. We need to configure an "Apache Tomcat v5.5" runtime environment for this application. After the wizard has been finished, copy the following jars to the "WebContent/WEB-INF/lib" directory. They will be added to the classpath of our project automatically.

```
${Hibernate_Install_Dir}/hibernate3.jar
${Hibernate_Install_Dir}/lib/antlr.jar
${Hibernate_Install_Dir}/lib/asm.jar
${Hibernate_Install_Dir}/lib/asm-attrs.jars
${Hibernate_Install_Dir}/lib/cglib.jar
${Hibernate_Install_Dir}/lib/commons-collections.jar
${Hibernate_Install_Dir}/lib/commons-logging.jar
${Hibernate_Install_Dir}/lib/dom4j.jar
${Hibernate_Install_Dir}/lib/ehcache.jar
${Hibernate_Install_Dir}/lib/jta.jar
${Hibernate_Install_Dir}/lib/log4j.jar

${Hsqldb_Install_Dir}/lib/hsqldb.jar

${Tomcat_Install_Dir}/webapps/jsp-examples/WEB-INF/lib/standard.jar
${Tomcat_Install_Dir}/webapps/jsp-examples/WEB-INF/lib/jstl.jar
```

For the web application to be developed in this chapter, we only deal with the Book, Publisher and Chapter persistent classes. Let's copy these three classes and the related mappings to our new project. Don't forget to copy the hibernate.cfg.xml, log4j.properties and ehcache.xml as well.

1.3. Configuring connection pool

For our previous examples, we establish a database connection each time when a session is created, and close it at the end of the session. Since creating a physical database connection will be very time consuming, we should better use a connection pool for reusing our connections, especially for a multi-user environment.

As a web application server, Tomcat supports connection pooling. To configure a connection pool in Tomcat, we need to first copy the following JDBC driver to the ${Tomcat_Install_Dir}/common/lib directory.

```
${Hsqldb_Install_Dir}/lib/hsqldb.jar
```

After the first launch of Tomcat from WTP, a new project called “Servers” will be created. We can modify the settings there for our Tomcat runtime. Open the server.xml configuration and locate our context “BookShopWeb”. We create a connection pool for this context by adding the following resource definition inside the <Context> node.

```
<Resource   name="jdbc/BookShopDB"
            type="javax.sql.DataSource"
            driverClassName="org.hsqldb.jdbcDriver"
            url="jdbc:hsqldb:hsql://localhost/BookShopDB"
            username="sa"
            password=""
            maxActive="5"
            maxIdle="2" />
```

Now the connection pool is ready to be used. We modify the hibernate.cfg.xml to use this connection pool instead of creating database connection each time. For the “connection.datasource” property, we provide the JNDI name of the connection pool configured at Tomcat. Also notice that the namespace “java:/comp/env/” should be added to the JNDI name.

```
<hibernate-configuration>
    <session-factory>
        <property name="connection.driver_class">org.hsqldb.jdbcDriver</property>
        <property name="connection.url">jdbc:hsqldb:hsql://localhost/BookShopDB</property>
        <property name="connection.username">sa</property>
        <property name="connection.password"></property>
        <property name="connection.datasource">java:/comp/env/jdbc/BookShopDB</property>
        <property name="dialect">org.hibernate.dialect.HSQLDialect</property>
        ...
    </session-factory>
</hibernate-configuration>
```

2. Developing an online bookshop

In this section, we implement our web-based online bookshop application using the typical MVC (Model-View-Controller) pattern. The purpose of this application is to demonstrate how to use Hibernate in a web environment, so we won't utilize any web frameworks such as Struts, Spring and Tapestry. We just implement it with JSPs / Servlets / Taglibs, which is the standard way of developing J2EE web applications.

2.1. Creating a global session factory

For an application using Hibernate as O/R mapping framework, a global session factory should be created and accessed through a particular interface. Here we use a static variable for storing the session factory. It is initialized in a static block when this class is loaded for the first time.

```
public class HibernateUtil {
    private static final SessionFactory sessionFactory;

    static {
        try {
            Configuration configuration = new Configuration().configure();
            sessionFactory = configuration.buildSessionFactory();
        } catch (Throwable e) {
            e.printStackTrace();
            throw new ExceptionInInitializerError(e);
        }
    }

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

2.2. Listing persistent objects

The first function we implement for our online bookshop is listing all the books available. We create a servlet "BookListServlet" to act as the controller and forward to the view "booklist.jsp" once finished the querying of books.

```
public class BookListServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        SessionFactory factory = HibernateUtil.getSessionFactory();
        Session session = factory.openSession();
```

```
        try {
            Query query = session.createQuery("from Book");
            List books = query.list();
            request.setAttribute("books", books);
        } finally {
            session.close();
        }
        RequestDispatcher dispatcher = request.getRequestDispatcher("booklist.jsp");
        dispatcher.forward(request, response);
    }
}
```

The view “booklist.jsp” is very simple. Its responsibility is to display the books queried by the servlet in a HTML table. At this moment, we only display the simple properties of books. The showing of association properties will be discussed later.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core_rt" %>
<html>
<head>
<title>Book List</title>
</head>
<body>
  <table border="1">
    <th>ISBN</th>
    <th>Name</th>
    <th>Publish Date</th>
    <th>Price</th>
    <c:forEach var="book" items="${books}">
      <tr>
        <td>${book.isbn}</td>
        <td>${book.name}</td>
        <td>${book.publishDate}</td>
        <td>${book.price}</td>
      </tr>
    </c:forEach>
  </table>
</body>
</html>
```

2.3. Updating persistent objects

Next we allow the administrators to update the book details by clicking a hyperlink on the book ISBN. This will trigger another servlet “BookEditServlet” to display a form for editing, passing in the identifier of a book.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core_rt" %>
<html>
<head>
<title>Book List</title>
</head>
<body>
  <table border="1">
    <th>ISBN</th>
    <th>Name</th>
    <th>Publish Date</th>
    <th>Price</th>
    <c:forEach var="book" items="${books}">
      <tr>
        <td><a href="BookEditServlet?bookId=${book.id}">${book.isbn}</a></td>
        <td>${book.name}</td>
        <td>${book.publishDate}</td>
        <td>${book.price}</td>
      </tr>
    </c:forEach>
  </table>
</body>
</html>
```

The doGet() method of “BookEditServlet” will be called when the user clicks the hyperlink on ISBN. We load the book object from database according to the identifier passed in, and then forward to the view “bookedit.jsp” for showing the form.

```
public class BookEditServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String bookId = request.getParameter("bookId");
        SessionFactory factory = HibernateUtil.getSessionFactory();
        Session session = factory.openSession();
        try {
            Book book = (Book) session.get(Book.class, Long.parseLong(bookId));
            request.setAttribute("book", book);
        } finally {
            session.close();
        }
        RequestDispatcher dispatcher = request.getRequestDispatcher("bookedit.jsp");
        dispatcher.forward(request, response);
    }
}
```

The view “bookedit.jsp” shows the detail of a book in a series of form fields.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core_rt" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jstl/fmt_rt" %>
<html>
<head>
<title>Book Edit</title>
</head>
<body>
  <form method="post">
    <table>
      <tr>
        <td>ISBN</td>
        <td><input type="text" name="isbn" value="${book.isbn}"></td>
      </tr>
      <tr>
        <td>Name</td>
        <td><input type="text" name="name" value="${book.name}"></td>
      </tr>
      <tr>
        <td>Publish Date</td>
        <td>
          <input type="text" name="publishDate"
            value="<fmt:formatDate value="${book.publishDate}" pattern="yyyy/MM/dd"/>">
        </td>
      </tr>
      <tr>
        <td>Price</td>
        <td><input type="text" name="price" value="${book.price}"></td>
      </tr>
      <tr>
        <td colspan="2"><input type="submit" value="Submit"></td>
      </tr>
    </table>
    <input type="hidden" name="bookId" value="${book.id}">
  </form>
</body>
</html>
```

When the submit button is clicked, the doPost() method of “BookEditServlet” will be called to handle the form submission. We first load the book object from database and then update its properties according to the request parameters. The user will be redirected to the book list page once finished.

```
public class BookEditServlet extends HttpServlet {
    ...
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
```

```
        String bookId = request.getParameter("bookId");
        String isbn = request.getParameter("isbn");
        String name = request.getParameter("name");
        String publishDate = request.getParameter("publishDate");
        String price = request.getParameter("price");

        SessionFactory factory = HibernateUtil.getSessionFactory();
        Session session = factory.openSession();
        Transaction tx = null;
        try {
            tx = session.beginTransaction();
            Book book = (Book) session.get(Book.class, Long.parseLong(bookId));
            book.setId(Long.parseLong(bookId));
            book.setIsbn(isbn);
            book.setName(name);
            book.setPublishDate(parseDate(publishDate));
            book.setPrice(Integer.parseInt(price));
            session.update(book);
            tx.commit();
        } catch (HibernateException e) {
            if (tx != null) tx.rollback();
            throw e;
        } finally {
            session.close();
        }
        response.sendRedirect("BookListServlet");
    }

    private Date parseDate(String date) {
        try {
            return new SimpleDateFormat("yyyy/MM/dd").parse(date);
        } catch (ParseException e) {
            return null;
        }
    }
}
```

2.4. Creating persistent objects

The function of creating a new book should also be provided. We first create a hyperlink in the book list page for adding a new book. The target servlet is the same as updating but there is no book identifier to pass in.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core_rt" %>
<html>
```

```
<head>
<title>Book List</title>
</head>
<body>
  <table border="1">
    <th>ISBN</th>
    <th>Name</th>
    <th>Publish Date</th>
    <th>Price</th>
    <c:forEach var="book" items="${books}">
      <tr>
        <td>${book.isbn}</td>
        <td>${book.name}</td>
        <td>${book.publishDate}</td>
        <td>${book.price}</td>
      </tr>
    </c:forEach>
  </table>
  <a href="BookEditServlet">Add Book</a>
</body>
</html>
```

The process of adding a new book is very similar to updating an existing book. We can reuse the servlet “BookEditServlet” by checking if the book identifier is null or not. If it is null, the action should be “add”, or else it should be “update”. At last, the session.saveOrUpdate() method is very helpful for distinguishing the correct action.

```
public class BookEditServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String bookId = request.getParameter("bookId");
        if (bookId != null) {
            SessionFactory factory = HibernateUtil.getSessionFactory();
            Session session = factory.openSession();
            try {
                Book book = (Book) session.get(Book.class, Long.parseLong(bookId));
                request.setAttribute("book", book);
            } finally {
                session.close();
            }
        }
        RequestDispatcher dispatcher = request.getRequestDispatcher("bookedit.jsp");
        dispatcher.forward(request, response);
    }
```

```
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String bookId = request.getParameter("bookId");
        String isbn = request.getParameter("isbn");
        String name = request.getParameter("name");
        String publishDate = request.getParameter("publishDate");
        String price = request.getParameter("price");

        SessionFactory factory = HibernateUtil.getSessionFactory();
        Session session = factory.openSession();
        Transaction tx = null;
        try {
            tx = session.beginTransaction();
            Book book = new Book();
            if (!bookId.equals("")) {
                book = (Book) session.get(Book.class, Long.parseLong(bookId));
            }
            book.setIsbn(isbn);
            book.setName(name);
            book.setPublishDate(parseDate(publishDate));
            book.setPrice(Integer.parseInt(price));
            session.saveOrUpdate(book);
            tx.commit();
        } catch (HibernateException e) {
            if (tx != null) tx.rollback();
            throw e;
        } finally {
            session.close();
        }
        response.sendRedirect("BookListServlet");
    }
}
```

2.5. Deleting persistent objects

The last function for book management is allowing the user to delete a book. We add a hyperlink to the last column of each row for deleting the current book.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core_rt" %>
<html>
<head>
<title>Book List</title>
</head>
<body>
  <table border="1">
```

```
    <th>ISBN</th>
    <th>Name</th>
    <th>Publish Date</th>
    <th>Price</th>
    <c:forEach var="book" items="${books}">
      <tr>
        <td>${book.isbn}</td>
        <td>${book.name}</td>
        <td>${book.publishDate}</td>
        <td>${book.price}</td>
        <td><a href="BookDeleteServlet?bookId=${book.id}">Delete</a></td>
      </tr>
    </c:forEach>
  </table>
  <a href="BookEditServlet">Add Book</a>
</body>
</html>
```

To delete a book from database, we must load it through the session first. This is because Hibernate need to handle the cascading of associations. One may worry about the overhead of loading the object prior to deletion. But don't forget that we are caching our book objects in second level caches. So there will not be much impact on the performance.

```
public class BookDeleteServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String bookId = request.getParameter("bookId");
        SessionFactory factory = HibernateUtil.getSessionFactory();
        Session session = factory.openSession();
        Transaction tx = null;
        try {
            tx = session.beginTransaction();
            Book book = (Book) session.get(Book.class, Long.parseLong(bookId));
            session.delete(book);
            tx.commit();
        } catch (HibernateException e) {
            if (tx != null) tx.rollback();
            throw e;
        } finally {
            session.close();
        }
        response.sendRedirect("BookListServlet");
    }
}
```