Java Server Pages in Verbindung mit Servlets im Einsatz

Wir werden in diesem Tutorial eine kleine Web-Anwendung (eine Bibliothek) erstellen, die das Zusammenspiel von JavaServer Pages und Java Servlets veranschaulichen soll. Dabei nutzen wir die JavaServer Pages zur Anzeige der Daten (Präsentation) und die Servlets zur Steuerung und zum Aufrufen der Geschäftslogik der Web-Anwendung.

Im Vorfeld sollte man noch kurz auf die Merkmale der beiden Technologien eingehen.

Allgemeines

Autor:
Sascha Wolski
http://www.lalliluna.de/tutorials.html
Tutorials für Struts, EJB, xdoclet und eclipse.
Datum:
September, 23th 2004
Source code:
http://www.lalliluna.de/assets/tutorials/java-servlets-jsp-tutorial.zip

Die Quellkodes enthalten nicht das Eclipse/MyEclipse Projekt, sondern nur die Quellen. Erstelle ein Projekt und kopiere die Quellen da rein.

PDF Version des Tutorials:

http://www.lalliluna.de/assets/tutorials/java-servlets-jsp-tutorial-de.pdf

Development Tools
Eclipse 3.x

Optional:
MyEclipse plugin 3.8
(Ein kostengünstige, leistungsfähige Erweiterung für die Entwicklung von Web Anwendungen und EJB Applikationen, Testversion gibt es bei MyEclipse.)

Application Server
Jboss 3.2.5
You may use Tomcat here if you like.

Java Servlets

Servlets stellen Java-Programme dar, die auf einem Web-Server laufen. Sie erlauben es Entwicklern dynamische Web-Seiten mit Java zu erstellen.

Ein Servlet hat folgende Aufgaben:

- Daten, die ein Benutzer beispielsweise in ein HTML-Formular auf einer Web-Seite eingegeben hat, werden gelesen und verarbeitet.
- Gegebenenfalls werden weitere Informationen, die mitgesendet werden, ausgewertet. Beispielsweise was für ein Browser oder System verwendet wird, etc.
- Ergebnisse werden anhand der vorliegenden Daten generiert. Es wird Geschäftslogik entweder direkt im Servlet ausgeführt oder eine andere Klasse aufgerufen, welche die Logik enthält oder die z.B. eine Datenbankabfrage durchführt.
- Die Ergebnisse werden formatiert. Erwartet der Browser eine Antwort im HTML-Format, so müssen die Ergebnisse gemäß dem Standard formatiert werden. Es ist mit Servlets auch möglich, Daten in einem beliebigen anderen Format zurückzuliefern (gif, jpeg, doc, etc.).
- Geeignete Antwortparameter werden gesetzt. Bevor das Servlet die Antwort Daten an den Browser zurückschickt, gibt es ihm diverse Parameter mit. In diesen Parametern wird angegeben, was für ein Format zurückgeliefert wird, welche Zwischenspeicherzeiten vom Browser verwendet werden sollen und einige andere Daten mehr.

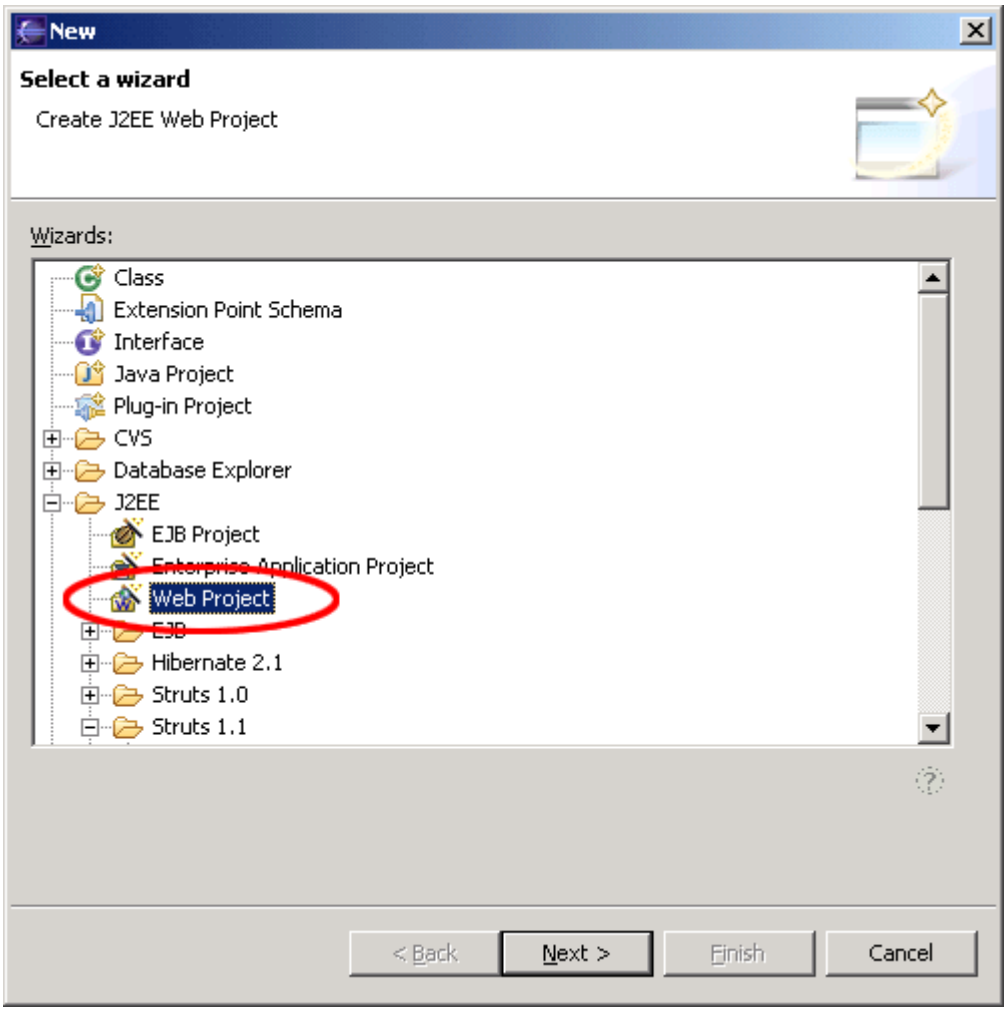
- Zurücksenden des Dokuments an den Browser in dem Format, das in den Antwortparametern angekündigt wurde.

Java Server Pages (JSP)

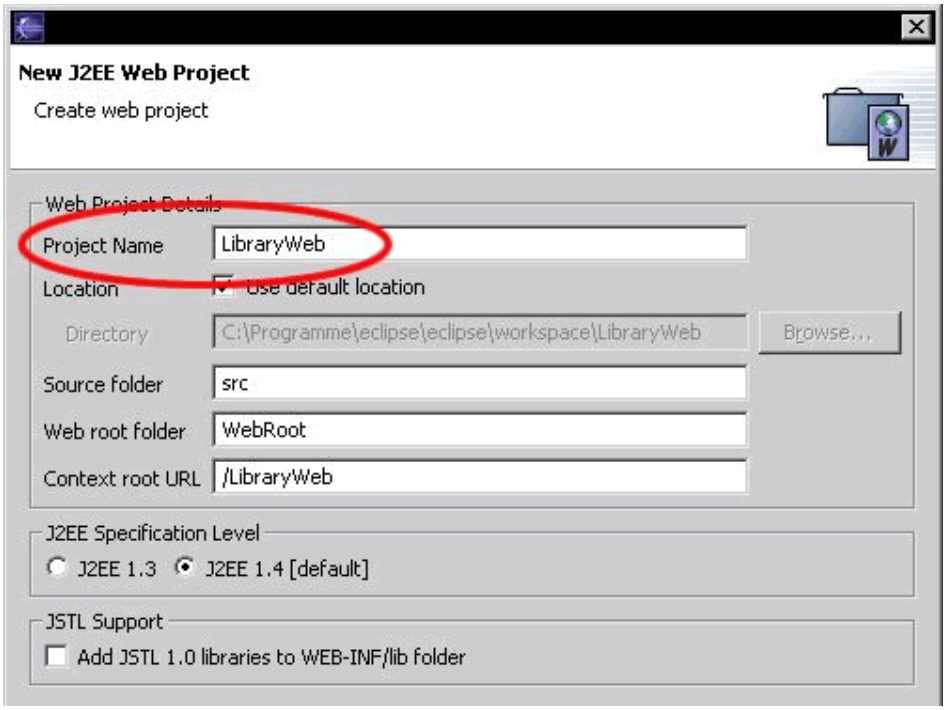
JavaServer Pages (JSP) sind Text-Dokumente, die reinen HTML-Dateien sehr ähnlich sind. Jedoch ist in JSP neben dem HTML-Code zusätzlicher Java-Code vorhanden. JavaServer Pages erlauben die Vermischung von regulärem, statischem HTML mit dynamisch generierten Inhalten durch Java Code. Anders als bei Servlets, bei denen HTML in Java-Code eingebettet wird, ist bei JSP der Java-Code in das HTML-Dokument eingefügt.

Erstellen eines neuen Web-Projektes

Klicke mit der rechten Maustaste auf den Package Explorer und wähle `New > Project`. Wähle den Wizard Web Project unter J2EE.



Gib dem Projekt einen geeigneten Namen.



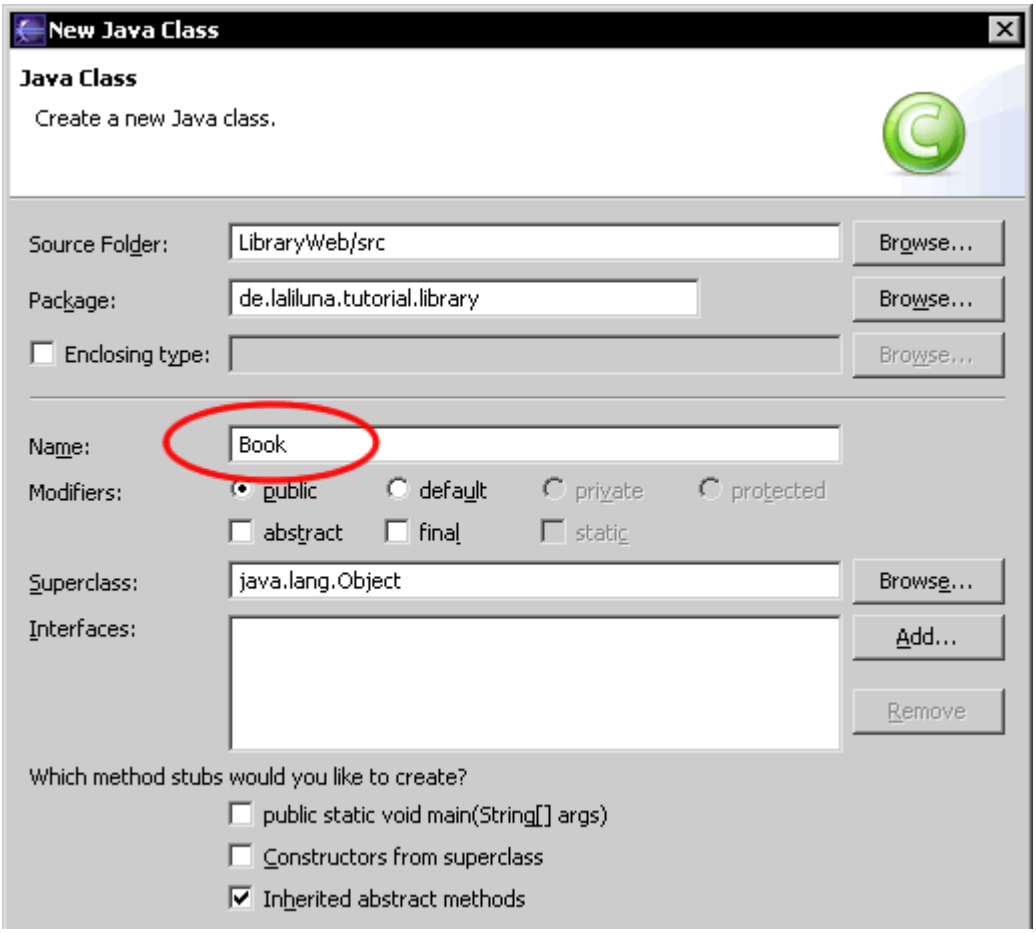
Klasse mit Testdaten

In diesem Beispiel verzichten wir auf die Anbindung einer Datenbank. Wir nutzen dafür eine Klasse die uns einige Testdaten, sowie Methoden zum hinzufügen, aktualisieren und löschen, der Daten bereitstellt. Erstelle dazu ein neues Package `de.laliluna.tutorial.library` unter `src` des Projektes. Damit wir nicht die komplette Klasse erstellen müssen, lade dir das fertige Projekt herunter und kopiere die Datei `SimulatedDB.java` aus dem Verzeichnis `/de/laliluna/tutorial/library` in das Package.

Erstellen der Klasse Book

Da eine Bibliothek Bücher verleiht, benötigen wir eine Klasse die ein solches Buch darstellt.

Erstelle im Package `de.laliluna.tutorial.library` eine neue Klasse mit `New > Class`.



Die Objekt Klasse Book soll folgende private Eigenschaften in Form von Variablen enthalten.

- id
- title
- author
- available

Wir erzeugen für die eben aufgezählte Eigenschaften Getter- und Setter-Methoden, um später auf die privaten Eigenschaften zugreifen zu können.

Öffne die Klasse Book und füge die Eigenschaften, sowie Getter- und Setter-Methoden ein. Füge noch die unten gezeigten Konstruktoren hinzu.

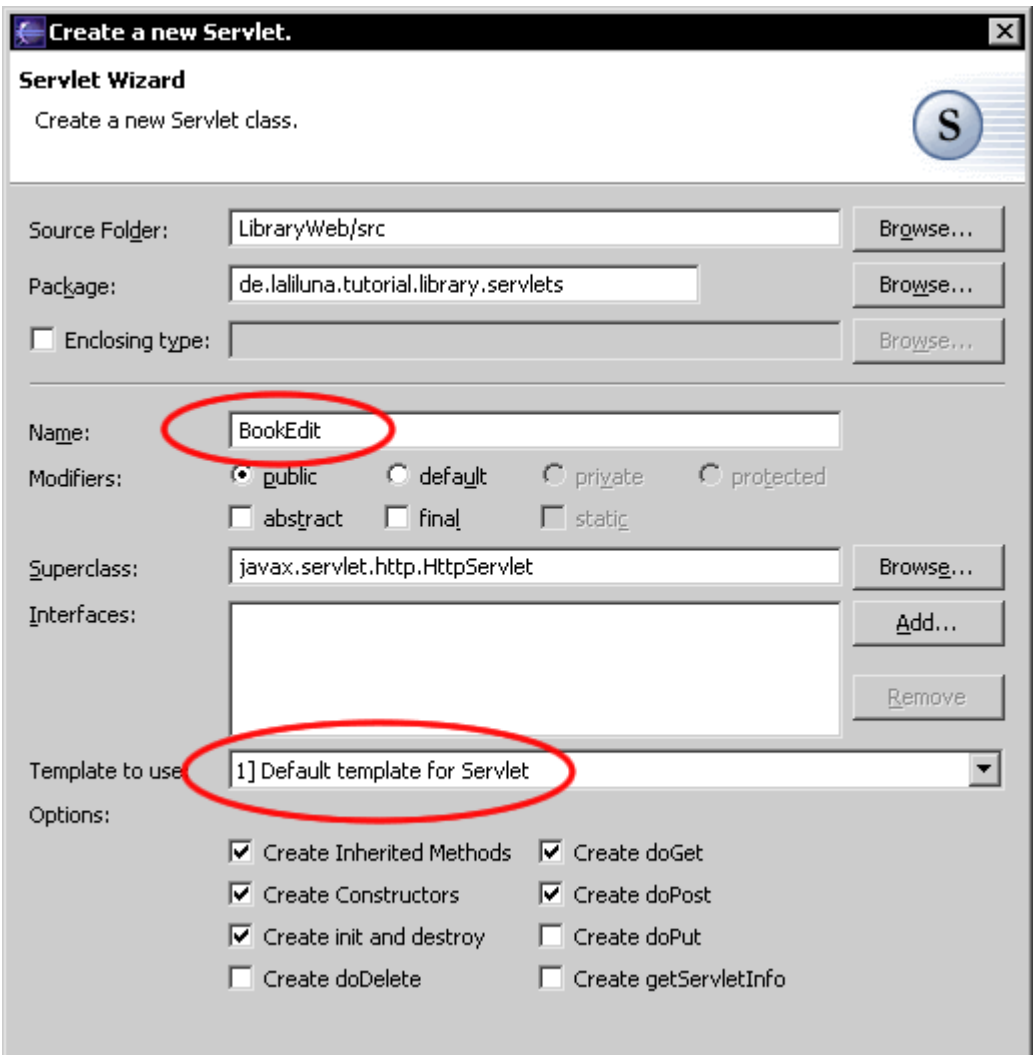
```
public class Book {

    private long id = 0;
    private String title = "";
    private String author = "";
    private boolean available = false;

    // Contructor
    public Book(){}
    //  Contructor to initial the properties
    public Book(long id, String author, String title, boolean available) {
        this.id = id;
        this.author = author;
        this.title = title;
        this.available = available;
    }

    public boolean isAvailable() {
        return available;
    }

    public void setAvailable(boolean available) {
        this.available = available;
    }

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

      public String getAuthor() {
            return author;
      }

      public void setAuthor(String author) {
            this.author = author;
      }

      public String getTitle() {
            return title;
      }

      public void setTitle(String title) {
            this.title = title;
      }
}
```

Erstellen der Servlets

In unserer Beispielanwendung verwenden wir zwei Servlets, die für das Steuern unserer Web-Anwendung verantwortlich sind.

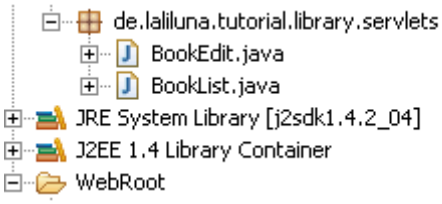
Erstelle dazu ein neues Package `de.laliluna.tutorial.library.servlets` unter `src` unseres Projektes.

Klicke mit der rechten Maustaste auf das Package und wähle `New > Servlet`.



Wiederhole das Ganze für das zweite Servlet. Vergib hier den Namen „BookList“.

In deinem Package sollten sich jetzt die beiden Servlet Klasse befinden.



Bearbeiten der Servlets

BookList.java

Öffne die Servlet Klasse `BookList.java`. Dieses Servlet wird eine Liste aller Bücher auslesen und im Request ablegen. Später wird in der JavaServer Page (JSP) darauf zugegriffen, um die Liste ausgeben.

Ändere die Methode `doGet(...)` wie folgt ab.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
                  throws ServletException, IOException {

            //hole die session des requests
            HttpSession session = request.getSession();

            //initialisiere die Datenbank Klasse
            SimulateDB simulateDB = new SimulateDB();

            //lies alle Bücher aus der Datenbank Klasse
            Collection collection = simulateDB.getAllBooks(session);

            //setze die zurück gegebene Liste der Bücher im Request
            request.setAttribute("books", collection);

            //hole den Request Dispatcher für die JSP Datei
            RequestDispatcher dispatcher = getServletContext().
getRequestDispatcher("/jsp/bookList.jsp");

            //leite auf die JSP Datei zum Anzeigen der Liste weiter
            dispatcher.forward(request, response);
}
```

In der `doPost(...)` Methode rufen wir die `doGet(...)` Methode auf, da bei beiden Anfragen GET oder POST vom Browser, die selbe Logik ausgeführt werden soll.

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
                  throws ServletException, IOException {

            //rufe die doGet(...) Methode auf
            this.doGet(request, response);

}
```

Das Servlet zum auslesen der Bücher unserer Bibliothek ist damit fertig.

BookEdit.java

Im nächsten Schritt wollen wir die Servlet Klasse `BookEdit.java` bearbeiten. Sie soll später das Anlegen, Bearbeiten und Löschen unserer Bücher steuern.

Öffne die Klasse `BookEdit.java` und ändere die Methode `doGet(...)` wie folgt ab.
Mit dem Parameter `do` in der `doGet(...)` Methode wir geprüft welche Aktion aufgerufen werden soll. Beim Anlegen eines Buches wird auf die Anzeige JSP weitergeleitet. Wenn dem Parameter `do` der Wert `edit` zugewiesen ist, wird anhand der übergebenen `id` das Buch auslesen und im Request unter dem Namen `book` abgelegt. Beim Löschen wird anhand der übergebenen `id` das Buch entfernt und auf das Servlet BookList, zum Anzeigen der Bücherliste, weitergeleitet.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
                  throws ServletException, IOException {

            //hole die session des requests
            HttpSession session = request.getSession();

            //initialisiere die Datenbank Klasse
            SimulateDB simulateDB = new SimulateDB();
```

```
            //lies den Parameter do aus dem Request
            String action = request.getParameter("do");

            //lies den Parameter id aus dem Request
            long id = 0;
            try{
                  id = Long.parseLong(request.getParameter("id"));
            }catch(NumberFormatException e){}

            //hinzufügen eines Buches
            if(action.equals("add")){

                  //hole den Request Dispatcher für die JSP Datei
                  RequestDispatcher dispatcher = getServletContext().
getRequestDispatcher("/jsp/bookAdd.jsp");

                  //leite auf die JSP Datei zum Anlegen eines Buches weiter
                  dispatcher.forward(request, response);

            //bearbeiten eines Buches
            }else if(action.equals("edit")){

                  //lies das Buch anhand seiner Id aus
                  Book book = simulateDB.loadBookById(id, session);

                  //setze das Buch Objekt im request
                  request.setAttribute("book", book);

                  //hole den Request Dispatcher für die JSP Datei
                  RequestDispatcher dispatcher = getServletContext().
getRequestDispatcher("/jsp/bookEdit.jsp");

                  //leite auf die JSP Datei zum Bearbeiten eines Buches weiter
                  dispatcher.forward(request, response);


            //löschen eines Buches
            }else if(action.equals("delete")){

                  //lösche das Buch anhand der Id
                  simulateDB.deleteBookById(id, session);

                  //Redirect Weiterleitung zum BookList Servlet
                  response.sendRedirect(request.getContextPath() + "/bookList");

            }

}
```

In diesem Servlet wird in der `doPost(...)` Methode **nicht** die `doGet(...)` Methode aufgerufen. Wir unterscheiden hier bei der Anfrage des Browser, ob GET oder POST.

Wird vom Browser eine POST Anfrage an das Servlet gestellt (Abschicken eines Formulars mit `<form method="post">`) sollen die übermittelten Daten, in unserem Falle die Büchereigenschaften (Author, Titel, ...) verarbeitet werden.

Ändere die `doPost(...)` Methode wie folgt ab.
In der Methode werden die übergebenen Eigenschaften aus dem Request gelesen und mit diesen ein neues Objekt Book erstellt. Mit der Methode `saveToDB(...)` unserer Datenbank Klasse, wird das Objekt Book gespeichert. Danach wird auf das Servlet zum Anzeigen der Bücherliste verwiesen.

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

            //hole die session des requests
```

```
            HttpSession session = request.getSession();

            //initialisiere die Datenbank Klasse
            SimulateDB simulateDB = new SimulateDB();

            //lies den Parameter id aus dem Request
            long id = 0;
            try{ id = Long.parseLong(request.getParameter("id")); }
            catch(NumberFormatException e){}

            //lies die Buch-Eigenschaften aus dem Request
            String author = request.getParameter("author");
            String title = request.getParameter("title");
            Boolean available = Boolean.valueOf(request.getParameter
("available"));

            //erstelle ein neues Buch Objekt
            Book book = new Book(id, author, title, available.booleanValue());

            //speichere das Buch Objekt
            simulateDB.saveToDB(book, session);

            //Redirect Weiterleitung zum BookList Servlet
            response.sendRedirect(request.getContextPath() + "/bookList");
}
```

Erstellen der JSP Dateien

Für die Anzeige der Bücherliste und den Dialogen zum Anlegen und Bearbeiten, erstellen wir drei JSP Dateien.

Erstelle ein neues Verzeichnis `jsp` in `/WebRoot` und lege drei neue JavaServer Pages an.

- bookAdd.jsp
- bookEdit.jsp
- bookList.jsp

bookAdd.jsp

Öffne die erste JSP Datei `bookAdd.jsp`. Die JavaServer Page soll später ein Formular anzeigen, über das der Benutzer ein Buch hinzufügen kann. Diese JSP Datei dent nur zum Aufnehmen neuer Daten und zum Abschicken an das Servelt.

```
<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()
+":"+request.getServerPort()+path+"/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">

    <title>Book add page</title>
```

```
  </head>
  <body>
      <form name="edit" action="bookEdit" method="post">
            <table border="1">
                  <tbody>
                        <tr>
                              <td>Author:</td>
                              <td><input type="text" name="author"
value=""></td>
                              </tr><tr>
                              <td>Title:</td>
                              <td><input type="text" name="title" value=""></td>
                              </tr><tr>
                              <td>Available:</td>
                              <td><input type="checkbox" name="available"
value="true"></td>
                              </tr><tr>
                              <td colspan="2"><input type="submit"
name="btnSave" value="Save"></td>
                        </tr>
                  </tbody>
            </table>
      </form>
  </body>
</html>
```

bookEdit.jsp

Die JavaServer Page `bookEdit.jsp` liest das vom Servlet im Request abgelegte Objekt book aus und setzt die Eigenschaften in den Feldern des Formulars.
In der ersten Zeile wird das Package `de.laliluna.tutorial.library` und alle darunter befindlichen Packages für die JSP Datei bekannt gemacht, eine so genannte JSP-Direktive. Diese ist identisch mit dem import Befehl einer Java Klasse.

Der Quelltext sieht wie folgt aus.

```
<%@ page import="de.laliluna.tutorial.library.*" %>
<%
      //lies das Objekt book aus dem Request
      Book book = (Book)request.getAttribute("book");
%>

<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()
+":"+request.getServerPort()+path+"/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">

    <title>Book edit page</title>
  </head>
  <body>
      <form name="edit" action="bookEdit" method="post">
            <table border="1">
                  <tbody>
                        <tr>
                              <td>Author:</td>
                              <td><input type="text" name="author" value="<%
=book.getAuthor()%>"></td>
                              </tr><tr>
                              <td>Title:</td>
                              <td><input type="text" name="title" value="<%
=book.getTitle()%>"></td>
```

```
                                </tr><tr>
                                <td>Available:</td>
                                <td><input type="checkbox" name="available"
value="true" <% if(book.isAvailable()) out.println("checked"); %>></td>
                                </tr><tr>
                                <td colspan="2"><input type="submit"
name="btnSave" value="Save"></td>
                            </tr>
                    </tbody>
            </table>
            <input type="hidden" name="id" value="<%=book.getId()%>">
      </form>
  </body>
</html>
```

bookList.jsp

In der JSP Datei `bookList.jsp` wird die Liste der vom Servlet ausgelesenden Bücher angezeigt. Es wird eine ArrayList von Büchern aus dem Request ausgelesen und darüber geloopt. In der for Schleife wird mit der Methode println(..) gearbeitet, um die Eigenschaften der Bücher auszugeben.

```
<%@ page language="java" import="java.util.*" %>
<%@ page import="de.laliluna.tutorial.library.*" %>

<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()
+":"+request.getServerPort()+path+"/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">

    <title>Book list page</title>
  </head>
  <body>
      <table border="1">
            <tbody>
            <tr>
                  <td>Author</td>
                  <td>Book name</td>
                  <td>Available</td>
                  <td> </td>
                  <td> </td>
            </tr>
        <%
            //lies die Liste der Bücher aus dem Request
            ArrayList arrayList = (ArrayList)request.getAttribute("books");

            //loope über die Liste von Büchern und gibt die Eigenschaften aus
            for (Iterator iter = arrayList.iterator(); iter.hasNext();) {
                  Book element = (Book) iter.next();

                  out.println("<tr>");
                  out.println("<td>" + element.getAuthor() + "</td>");
                  out.println("<td>" + element.getTitle() + "</td>");
                  if(element.isAvailable())
                        out.println("<td><input type=\"checkbox\"
name=\"available\" value=\"true\" disabled checked></td>");
                  else
                        out.println("<td><input type=\"checkbox\"
name=\"available\" value=\"true\" disabled></td>");

                  out.println("<td><a href=\"bookEdit?do=edit&id=" +
element.getId() + "\">Edit</a></td>");
```

```
                    out.println("<td><a href=\"bookEdit?do=delete&id=" +
element.getId() + "\">Delete</a></td>");
                    out.println("</tr>");
                }
            %>
                </tbody>
        </table>
        <br>
        <a href="bookEdit?do=add">Add a new book</a>
    </body>
</html>
```

Damit haben wir die JavaServer Pages für die Anzeige der Daten fertig gestellt.

Default JSP

Damit der Benutzer beim Aufruf des Projektes http://localhost:8080/LibraryWeb/ direkt auf die Anzeige der Bücherliste weitergeleitet wird, erstellen wir im `/WebRoot` Verzeichnis noch eine Datei `index.jsp`.

Ändere den Quelltext der JSP Datei wie folgt ab.

```
<%
        //redirect to the book list servlet
        response.sendRedirect("bookList");
%>
```

Bearbeiten der web.xml

Im nächsten Schritt bearbeiten wir die Konfigurationsdatei unseres Projektes, die `web.xml`. In der web.xml müssen wir nun unsere Servlets bekannt machen und über welche Adresse (URL) ich diese Aufrufen kann.

Öffne die Datei web.xml und füge folgende Konfiguration hinzu.

Mit dem Tag `<servlet-name>` vergeben wir einen Namen für unsere Servlet Klasse. Über diesen Namen identifizieren wir das Servlet in der `web.xml`. `<servlet-class>` definiert die Servlet Klasse. In dem Tag `<servlet-mapping>` weisen wir einem Servlet eine Adresse (URL) zu, über die der Benutzer später das Servlet in seinem Browser aufrufen kann.

Die beiden Servlets werden über /bookList und /bookEdit aufrufbar sein.

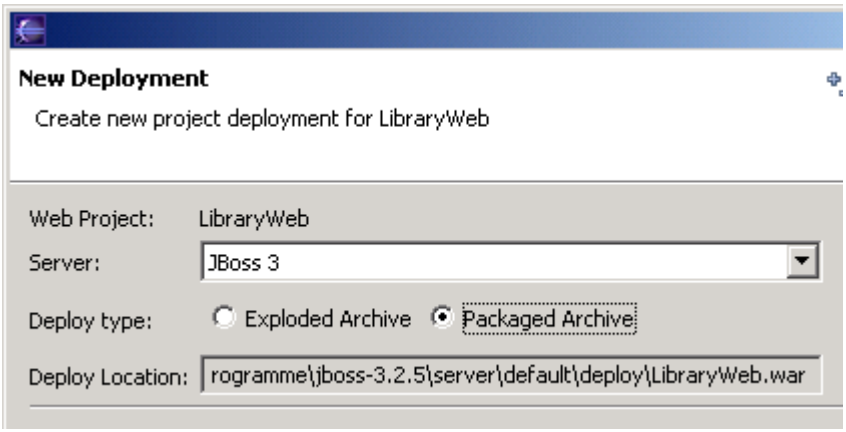
```
<web-app>
<servlet>
        <servlet-name>BookList</servlet-name>
        <servlet-class>de.laliluna.tutorial.library.servlets.BookList</servlet-
class>
</servlet>
<servlet-mapping>
        <servlet-name>BookList</servlet-name>
        <url-pattern>/bookList</url-pattern>
</servlet-mapping>

<servlet>
        <servlet-name>BookEdit</servlet-name>
        <servlet-class>de.laliluna.tutorial.library.servlets.BookEdit</servlet-
class>
</servlet>
<servlet-mapping>
        <servlet-name>BookEdit</servlet-name>
        <url-pattern>/bookEdit</url-pattern>
```

```
</servlet-mapping>
</web-app>
```

Das wars auch schon.

Testen der Anwendung

Deploye das Projekt in deinem Jboss.



Testen danach deine Anwendung http://localhost:8080/LibraryWeb