Fachoberschule Wirtschaft - Schwerpunkt Informatik

JavaServer Pages

Christoph Süsens, Matthias Holdorf

01.02.2010

Betreuer
T. Friedrich

Erklärung

Wir versichern, dass wir diese Facharbeit ohne fremde Hilfe selbstständig verfasst haben und nur die angegebenen Quellen und Hilfsmittel benutzt haben. Wörtlich oder dem Sinn nach aus anderen Werken entnommenen Stellen sind unter Angabe der Quellen kenntlich gemacht.

.....	.....
Datum, Unterschrift	Datum, Unterschrift

Vorwort

Diese Facharbeit befasst sich mit dem Thema JavaServer Pages. Die Technik der JavaServer Pages wird zur Programmierung von dynamischen Web-Anwendungen eingesetzt. Nach einer kurzen Einführung in die grundlegenden Kenntnisse, sowie einer Erläuterung von Nutzen und Einsatzgebieten der JavaServer Pages, werden wir uns mit weiterführenden und komplexeren Themen, sowie mit der Erstellung einer Website welche auf JavaServer Pages basiert, auseinandersetzen. Um dem Leser das Thema transparenter zu machen und einen Einstieg zu erleichtern, dient die Beispiel Website der Veranschaulichung.

Das Quellmaterial besteht zu einem großen Teil aus zuverlässigen Internetquellen, sowie einiger themenspezifizierter Literatur.

Einsatzgebiete und Nutzen von JavaServer Pages

„JavaServer Pages bilden eine Möglichkeit, serverseitig Web-Anwendungen zu erstellen. Heutzutage sind diese längst eine Selbstverständlichkeit. In den Anfangstagen des World Wide Web Anfang und Mitte der Neunziger Jahre allerdings waren Websites zunächst rein statische Angelegenheiten. Es gab Seiten, die sich zwar mittels der Hyperlinks untereinander referenzierten, aber es gab weder die Möglichkeit, auf Benutzereingaben zu reagieren, noch dynamische Daten auszugeben, wie bspw. Börsendaten, News etc., ohne jedes Mal die Seiten neu anpacken zu müssen. Angesichts des Erfolges des WWW bereits in diesem frühen Stadium wurden jedoch schnell diverse Technologien geschaffen, um diese Beschränkung zu umgehen. Zerlest und JSPs waren SUN Microsystems Reaktion auf diese Fortentwicklung der Web-Technologien. Diese ermöglichen es nun, mittels Java auf Datenbanken zuzugreifen, Grafiken neu zu berechnen, Nutzereingaben zu verarbeiten, bzw. allgemein unter Ausnutzung von Java eine dynamische Reaktion auf die Nutzer-Anfrage unter Berücksichtigung seiner vorherigen Interaktionen zu generieren. Die Kombination mit Java und mehr noch die Einbettung in die J2EE-Technologien machen die enorme Flexibilität und das enorme Potenzial der Java-basierten Webtechnologien JSP und Servlets aus.“ [1]

[1] http://www.jsptutorial.org/content/introduction

Definition JavaServer Pages

JavaServer Pages ist eine auf JHTML entwickelte Webprogrammiersprache. Sie dient zur einfachen Erzeugung dynamischer HTML & XML Webseiten. Die Ausgaben erfolgen mittels eines Webservers. Weiterhin erlaubt es JSP Java-Code & spezielle JSP-Aktionen in statischen Inhalt einzubetten.

Als Entwurfsmuster dient dabei das sogenannte MVC-Muster (Model-View-Controller).

MVC - realisiert mit JavaServer Pages

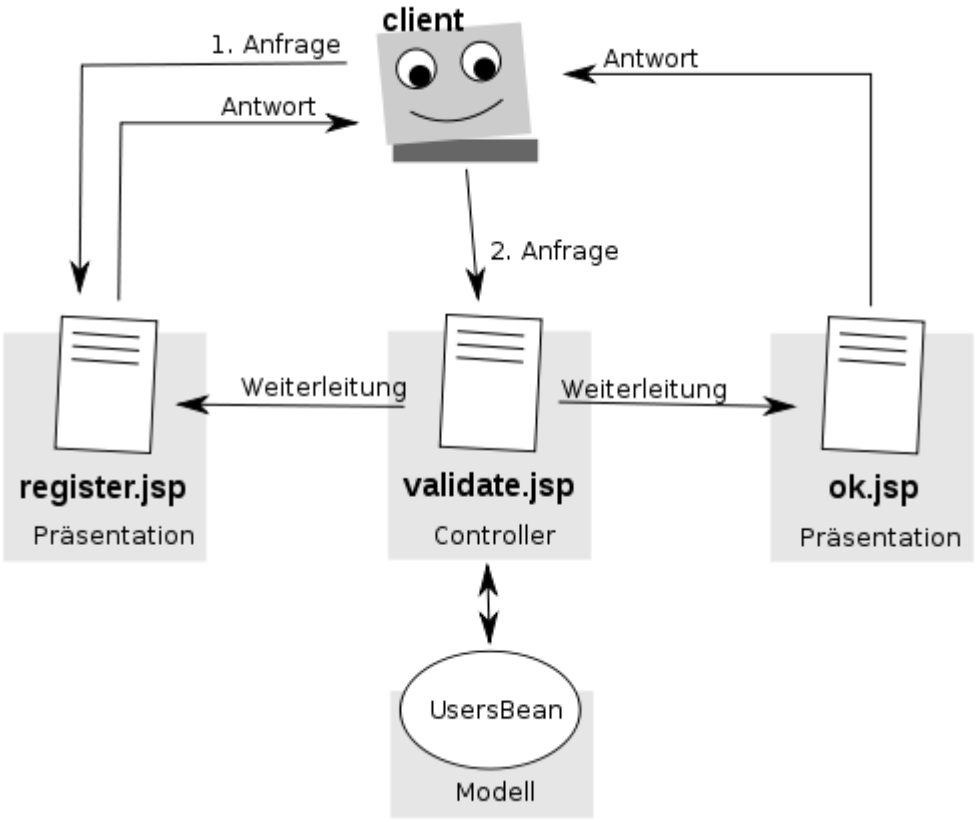

Abbildung 1 – MVC Schaubild

[2]

[2] http://de.wikipedia.org/wiki/Model_View_Controller

Aufruf von Websites

Aufruf einer statischen Website[3]

Zur Vorstellung, wie der Aufruf einer Website funktioniert, eine Abbildung die den Aufruf einer **statischen** Website darstellt.

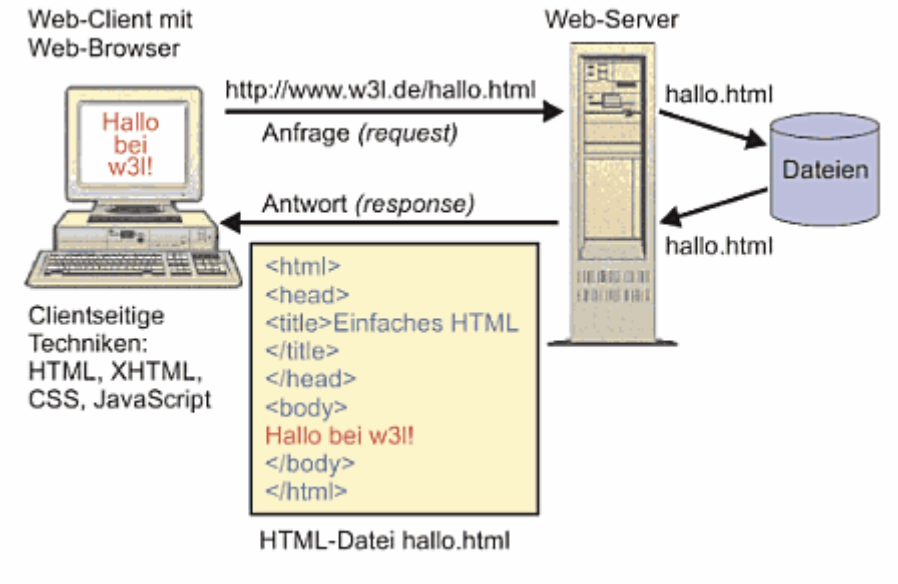

Abbildung 2 – Aufruf einer statischer HTML-Seite

Zuerst wird per Anfrage (Request) an den Webserver, die gewünschte Seite angefordert. Anschließend liest der Webserver die Datei von seiner Festplatte aus und schickt diesen zum Webbrowser, der ihn dann interpretiert anzeigt.

[3] Buch: JavaServer Pages - Dieter Wißmann Seite 5, 6

Aufruf einer dynamischen Website

In der folgenden Abbildung wird, ähnlich wie in Abbildung 1, eine Website vom Webserver angefordert. Bei diesem Szenario handelt es sich allerdings um eine **dynamische** Website.

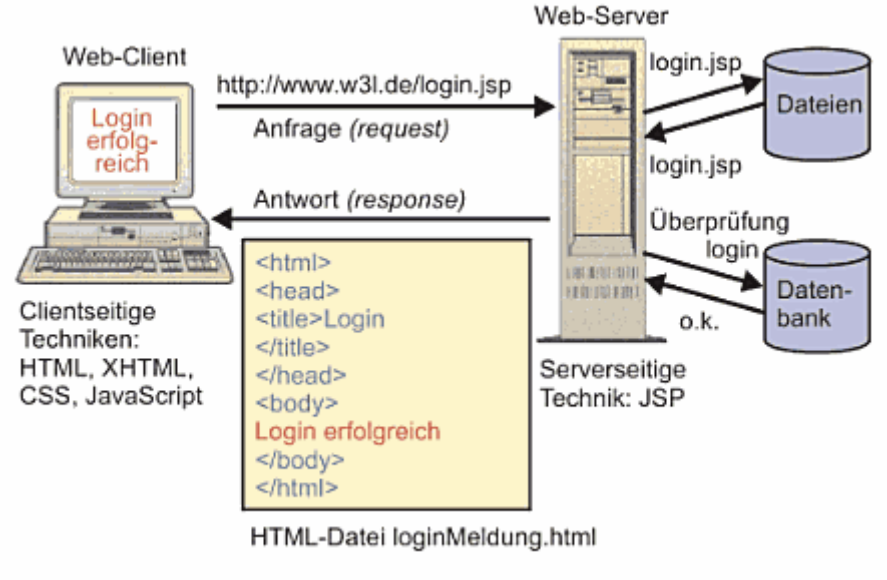

Abbildung 3 - Aufruf einer dynamischen HTML-Seite

Wie in Abbildung 1 wird per Anfrage (Request) vom Webclient eine Website angefordert, hier allerdings mit einem Login. Der Webserver lädt die Datei von seiner Festplatte in seinen Arbeitsspeicher und führt das Programm aus, d. h. er überprüft die Login-Daten über seine Datenbank. Das Ergebnis wird als HTML-Dokument dem Webclient überliefert.

Schnelleinstieg in Java Server Pages

„Hallo-Welt“-Beispiel4

Wie bei Einführungen in Programmiersprachen so üblich, wollen wir auch hier mit einem einfachen Beispiel beginnen.

LISTING 1: HELLOWORLD.JSP

```
<html>
        <head><title>Hello World JSP Page.</title></head>
        <body>
                <font size="10"><%="Hello World!" %></font>
        </body>
</html>
```

Dieser Quellcode erzeugt die folgende Ausgabe:

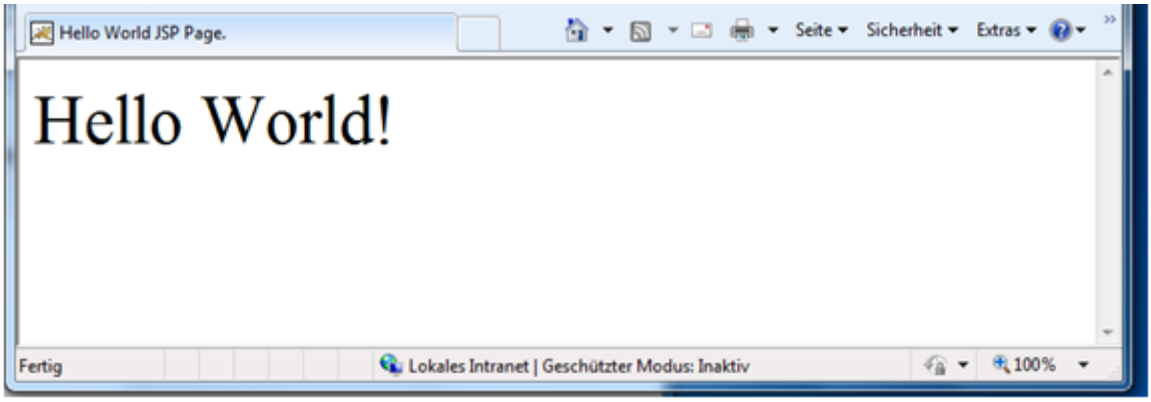

Abbildung 4 – „Hello World!“ Beispiel

[4] http://www.roseindia.net/jsp/Hello_World.shtml

3.2 Die Anatomie einer JSP-Seite

Eine Java-Server-Page ist eine HTML mit Java-Code-Fragmenten geschriebene Datei. Sie trägt die Dateikennung .jsp und wird erst nach dem Übersetzen, mittels eines Containers, z. B. Tomcat ausführbar. Die Java-Code-Fragmente sorgen dafür, dass die HTML-Datei dynamisch wird. Diese Fragmente werden in der Fachsprache als Scripting-Elemente bezeichnet, welche im nächsten Kapitel genauer beschrieben werden. Der Java-Code wird in <%... %>, sog. Tags geschrieben.[5]

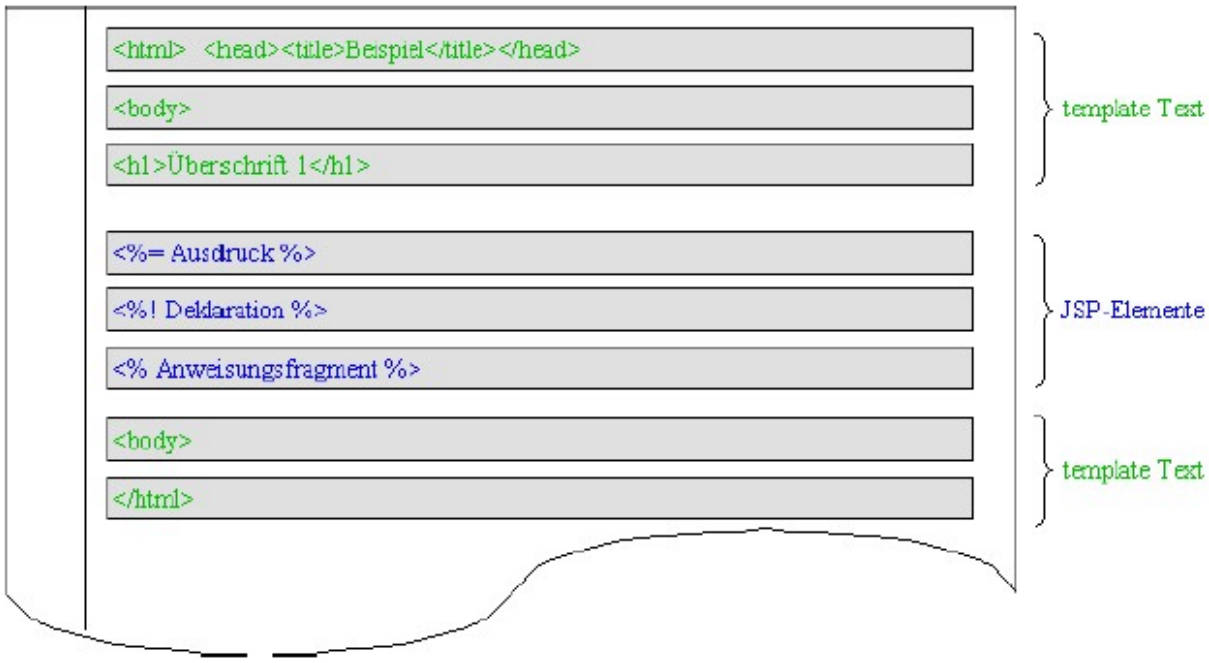

Abbildung 5 - Die Anatomie einer JSP-Seite

[5] Buch: JSP für Einsteiger Seite 13, 47

Die Ausführung einer JSP-Seite

Durch das einbinden von Java-Code, kann die JSP-Datei nicht direkt zum Browser durchgereicht werden. Die Scripting-Elemente müssen zuvor noch von einem Server verarbeitet werden. Bei diesem Vorgang wird die JSP-Datei in ein Servlet umgewandelt.[6]

Übersetzungphase

Sobald eine Anfrage bei dem Server eingeht, fordert dieser die JSP-Datei an, welche beim ersten Zugriff initialisiert wird und zu einem Servlet verarbeitet wird. Es entsteht eine .class Datei. Dieser Vorgang wird als Übersetzungsphase definiert und findet nur einmal statt, es sei denn die JSP-Datei wird aktualisiert, so wiederholt sich der Vorgang aufs Neue. Da es bei der Übersetzung zu längeren Wartezeiten kommen kann, empfiehlt es sich die JSP-Datei vor zu kompilieren.[7]

Um diesen Vorgang sichtbar zu machen, empfiehlt es sich das Verzeichnis

TOMCAT 5.5\WORK\CATALINA\LOCALHOST\SERVLETNAME\ORG\APACHE\JSP

anzugucken, welches beim ersten Ausführen der JSP-Datei entsteht.[8]

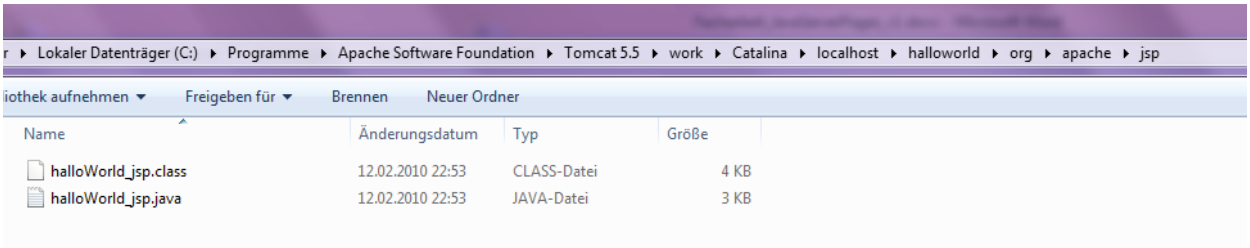

Abbildung 6 – Verzeichnis: Tomcat/…/jsp

[6] Buch: JSP für Einsteiger Seite 47
[7] Buch: JSP für Einsteiger Seite 48
[8] http://www.werthmoeller.de/doc/microhowtos/tomcat/verzeichnisstruktur/

Anfragephase[9]

Desweiteren muss der Server auf jede Anfrage (request) eine Antwort (response) generieren. Dazu führt er die entsprechende class-Datei aus, die das Ergebnis der Übersetzungsphase ist. Die Ausführung dieses Servlets generiert eine entsprechende HTML-Datei, die als Antwort zum Browser geschickt wird. Vorher überprüft der Server allerdings, ob sich die JSP-Datei seit dem letzten Übersetzen geändert hat. Ist dies der Fall, wird erneut die Übersetzungsphase durchlaufen.[10]

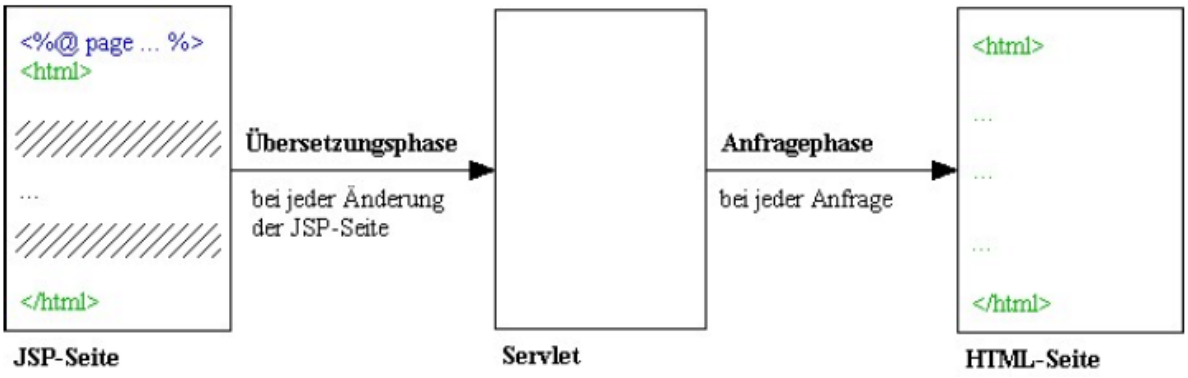

Abbildung 7 - Übersetzungs- und Anfragephase

[9] Buch: JSP für Einsteiger Seite 49
[10] Buch: Servlets & JSP Seite 102

Benötigte Software[11]

Das Java Development Kit (JDK)

Eine Java Entwicklungsumgebung (Netbeans)

Einen Container (Tomcat)

- Stellt die Plattform für JSP dar
- Übersetzt Java Code in Servlets (JSP-Compiler)
- Ordnet die URL dem Servlet zu
- Leitet den Lebenszyklus des Servlet
- Prüft URL Anfragen auf Rechte

Einen HTML Editor (Dreamweaver)

- Zum Erstellen und Testen von JavaServer Pages

Allgemeine Voraussetzung[12]

Der Einsatz von Java-Server-Pages setzt voraus, dass ein JSP fähiger Webserver vorhanden ist. Dieser Container übernimmt die Übersetzungs- und Anfragephase.

[11] Buch: JSP für Einsteiger Seite 8
[12] Buch: Servlets und JSP Seite XXIV

Container - Tomcat

Ein Container, welcher diese Funktionen bereitstellt ist der Apache Tomcat Server. Dieser Entstand durch Sun Microsystems und wurde im Jahre 1999 an die Apache Software Foundation übergeben. Aktuell liegt Version 6.0 vor, welche den Spezifikationen Servlet 2.5 und JSP 2.1 folgt.[13]

Installation Tomcat

Die Installation des Apache Tomcat Servers setzt das Java Development Kit (JDK) voraus. Welches kostenlos von java.sun.com herunter geladen werden kann. Dies resultiert daher, dass Tomcat ein komplett auf Java basierter Webcontainer ist und dringend zum Ausführen und zur Übersetzung der JSP-Dateien benötigt wird. Außerdem muss die Umgebungsvariable „PATH“ um den Wert „JAVA_HOME\bin“ erweitert werden.

Tomcat kann kostenlos, da es sich um ein „open source“ Projekt handelt, von der Homepage tomcat.apache.org geladen werden. Nach dem Ausführen des Setups, steht der Container, welcher gleichzeitig ein vollständiger HTTP-Server ist, startbereit zur Verfügung. Zur Überprüfung ob die Installation erfolgreich war, sollte man http://localhost:8080 aufrufen. Erscheint die Tomcat-Startseite ist die Installation geglückt.[14]

Abbildung 8 – Tomcat Startseite

[13] http://de.wikipedia.org/wiki/Apache_Tomcat
[14] Buch: Servlets und JSP Seite XXIV

Verzeichnisse von Tomcat[15]

Das Tomcat-Verzeichnis enthält einige Unterverzeichnisse:

- bin: enthält Skripte zum Starten und Herunterfahren des Servers
- conf: enthält wichtige Konfigurationsdateien für den Server
- doc: enthält Dokumentation zur Installation und zum Start des Servers, z.B. eine readme und eine faq.
- lib: enthält Module um Tomcat mit anderen Webservern, z.B. Apache, zu verbinden
- logs: enthält log-Dateien des Servers
- src: enthält den Quellcode aller Servlet- und JSP-Klassen.
- webapps: Standardverzeichnis für alle Webapplikationen, die von Tomcat verarbeitet werden, hier sollte das Verzeichnis der zu erstellenden Webapplikation abgelegt werden.
- work: Das Arbeitsverzeichnis des Servers. Es enthält dynamisch erzeugte .java und .class Dateien

Starten und Stoppen von Tomcat

Nach der Installation steht Tomcat als Dienst auf automatischen Start, dies sollt jedoch solange sie Tomcat als Entwicklungsumgebung einsetzten auf manuell gestellt werden.

Im Verzeichnis bin, des Stammverzeichnisses von Tomcat finden sie die tomcat5w.exe mit der Tomcat gestartet und gestoppt werden kann.

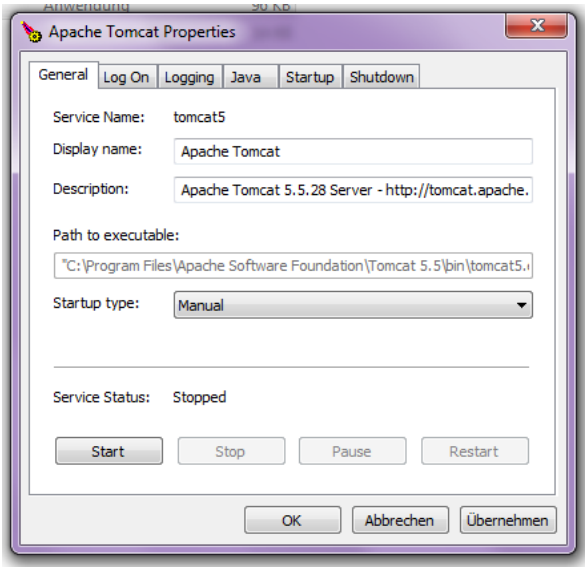

Abbildung 9 – Tomcat

[15] http://www.werthmoeller.de/doc/microhowtos/tomcat/verzeichnisstruktur/

Tomcat Manager[16]

Nach der Installation von Tomcat, steht einem ein Manager zur Verfügung, der die folgenden Funktionen aufweist:

- Veröffentlichen einer neuen Applikation
- Installieren einer neuen Applikation
- Auflistung der veröffentlichten und installierten Applikationen
- Erneutes Laden einer neuen Applikation
- Entfernen einer neuen Applikation
- Auflistung der Eigenschaften von Betriebssystem und der Java Virtual Machine (JVM)
- Auflistung der globalen JNDI Resourcen
- Auflistung der verfügbaren Sicherheits Rollen
- Anzeige von Statistikwerten von Sitzungen
- Start einer bestehenden Applikation
- Stop einer bestehnden Applikation
- Veröffentlichen einer neuen Applikation zurücknehmen

Zum Einloggen wird das Passwort, welches bei der Installation von Tomcat hinterlegt wurde, benötigt. Sollte das Passwort vergessen worden sein, kann es in der tomcat-users.xml Datei, welche sich im Stammverzeichnis unter conf befindet, ausgelesen werden.

[16] http://www.konicsek.de/schwerpunkte/tomcat-konfiguration.htm

Deployment Descriptor (DD)

Der Deployment Descriptor ist eine Konfiguratiosdatei im XML-Format, sie trägt den Namen web.xml und liegt im Verzeichnis WEB-INF, im Wurzelverzeichnis der Webanwendung. Sie dient zur Initialisierung und Installation von Web-Anwendungen. Jede Web-Anwendung muss zwingend eine web.xml Datei besitzen. Mit dem Deployment Descriptor können URL´s zugeordnet werden, Rollen innerhalb der Sicherheitsarchitektur festgelegt werden, Fehlerseiten, Tab-Bibliotheken, Konfigurationsinformationen für die Initialisierung und Zugriffsregeln auf bestimmte J2EE-Server festgelegt werden. [17]

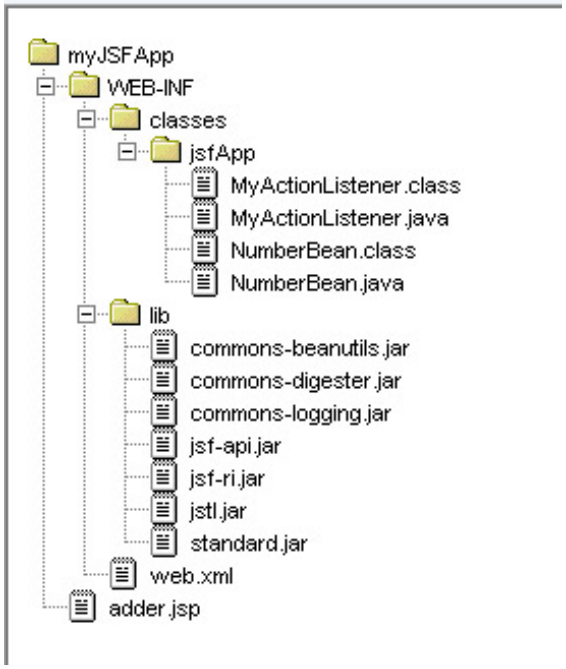

Abbildung 10 – Strukturbaum

Aufbau einer Web-Anwendung WEB-INF mit dem Deployment Descriptor, sowie die Java-Klassen unter WEB-INF/classes und benötigte Bibliotheken unter WEB-INF/lib.[18]

[17] Buch: O´Reilly Servlets & JSP von Kopf bis Fuß Seite 49
[18] http://onjava.com/pub/a/onjava/2003/07/30/jsf_intro.html

Syntax – JavaServer Pages (Scripting Elemente)[19]

Allgemein

JavaServer Pages bestehen aus einem HTML Grundgerüst indem Java-Code eingebettet ist. Diese Code Fragmente sind durch die Markierungen, die sogenannten Tags, zu erkennen. Die Dateiendung der JSPs ist Beispiel**.jsp**.

Scriptlets (<% ... %>)

Beliebiger Java Code der mehrere Anweisungen enthalten kann, jede Anweisung wird mit einem Semikolon abgeschlossen. Der Java Code wird syntaxtisch mit „<% ... %>“ eingebunden.

Ausdrücke (<%= ... %>)

Variablen oder Methoden werden direkt in den HTML- oder XML-Ausgabestrom integriert.[20]

Beispiel: **<%= ZAEHLER.GETANZAHL() %>**

Statt: **<% OUT.PRINTLN (ZAEHLER.GETANZAHL()); %>**

Achtung: Niemals mit einem Semikolon abschließen!

Deklarationen (<%! ... %>)

Hier werden Variablen und Methoden deklariert. Dies erfolgt zu Beginn des Quellcodes, außerhalb der Service-Methoden. In Deklarationen erfolgen keine Ausgaben!

[19] http://www.mi.fh-wiesbaden.de/~barth/webanw/vorl/WebAnwPB6.pdf
[20] http://de.wikipedia.org/wiki/Java_Server_Pages#Ausdrücke

Direktiven (<%@ … >)[21]

Eine Direktive dient zum Übermitteln spezieller Seiteninformationen an den JSP-Compiler. Es werden drei Arten unterschieden:

include

Weist den JSP-Compiler an, den vollständigen Inhalt einer externen Datei in die Originaldatei zu kopieren.

<%@ INCLUDE FILE="BEISPIELDATEI.EXT" %>

page

import:

Generiert ein Java-Import-Statement in der Datei.

<%@ PAGE IMPORT="JAVA.UTIL.*" %>

Weiterhin gibt es die folgenden Attribute: *contentType, contentType, isErrorPage, isThreadSafe.* (Nachzulesen im Anhang: „JSP_Syntax_v2.0.pdf“)

taglib

Gibt an, dass eine Taglib verwendet werden soll. Es müssen ein Präfix und eine URL für die Taglib vergeben werden.

<%@ TAGLIB PREFIX="MEINPREFIX" URI="TAGLIB/MEINETAGLIB.TLD" %>

Aktion[22]

JSP-Aktionen sind XML-Tags, welche die eingebaute Funktionalität von Webservern einbinden. Die folgenden Aktionen sind verfügbar:

jsp:include, jsp:param, jsp:forward, jsp:pluginjsp:fallback, jsp:setProperty, jsp:getProperty, jsp:useBean. (Nachzulesen im Anhang: „JSP_Syntax_v2.0.pdf“)

Beispiel: **<JSP: INCLUDE PAGE =“FOO.HTML“/>**

Kommentare

Die Schreibweise: <%-- Kommentar --%>.

Die Kommentar-Funktion „//“ funktioniert nicht, sie würde dem Client als Antwort angezeigt werden. (Ausnahme: In Scriptlet oder Deklaration-Tag)

[21] http://de.wikipedia.org/wiki/Java_Server_Pages#Direktiven
[22] http://de.wikipedia.org/wiki/Java_Server_Pages#Aktion

Beispielprogramm: Gästebuch[23]

Anhand eines Beispiels werden wir die Funktionalität der Java-Server-Pages nun erläutern.

Der Benutzer ruft die „index.html“-Seite auf und füllt das Formular mit Werten. Sendet diese an eine JSP-Datei, welches die Werte prüft, verarbeitet und als „gaestebuch.jsp“ anzeigt.

LISTING 2: INDEX.HTML

Ein Blick auf die index.html:

```
<html>
<head> <title>Mein Gästebuch</title> </head>
<body bgcolor="#FFCC99" text="#3300CC">
<p>Werfen Sie doch einen Blick in mein
 <a href="gaestebuch.jsp">Gästebuch.</a></p>
Hier können Sie Ihren eigenen Eintrag hinzufügen.<br>
<form action="gaestebuch.jsp" method="get" >
  Eintrag <br /> <textarea name="eintrag" cols="55"></textarea>  <br>
  Name* <input type="text" name="name" size="70" value=""> <br>
  E-Mail <input type="text" name="email" size="70"> <br>
  Homepage <input type="text" name="homepage" size="70"
value="http://"> <br>
  <input type="submit"  value="Absenden"> <br>
</form>
</body>
</html>
```

Das Beispiel zeigt die eingebundene JSP-Datei „gaestebuch.jsp“ welche über form action mit der Methode get, die eingegebenen Werte an die JSP-Datei sendet.

LISTING 3: GAESTEBUCH.JSP

Ein Blick auf die „gaestebuch.jsp“:

```
<html>
<head> <title>Mein Gästebuch</title> </head>
<%-- Importieren von Java-Klassenbibliotheken --%>
<%@page import="java.util.*"%>
<%@page import="java.io.*"%>
%@page import="java.text.*"%
```

Der Container wird angewiesen, eine Page-Direktive auszuführen.

[23] Buch: JSP für Einsteiger Seite 31

```
<%-- Deklaration von Java-Klassen --%>
<%!
//Java-Klasse zur Serialisierung und Deserialisierung
class ObjektDatei
{
      private String Dateiname;
      //Konstruktor
      public ObjektDatei(String Dateiname)
      {
      this.Dateiname = Dateiname;
      }
      //Operationen
      public void speichereObjekt(Object einObjekt)
      {
            try
            {
                  ObjectOutputStream AusgabeDatei =
                  new ObjectOutputStream(new
FileOutputStream(Dateiname));
                  AusgabeDatei.writeObject(einObjekt);
                  AusgabeDatei.close();
            }
            catch (IOException e)
            {System.out.println("Fehler beim Dateispeichern: "+e);}
      }
      public Object leseObjekt()
      {
            Object einObjekt = null;
            try
            {
                  ObjectInputStream EingabeDatei =
                  new ObjectInputStream(new
FileInputStream(Dateiname));
                  einObjekt = EingabeDatei.readObject();
                  EingabeDatei.close();
            }
            catch (IOException e1) {}
            catch (ClassNotFoundException e2) {}
            return einObjekt;
      }
}
```

Anweisung eine Klasse mit dem Namen „ObjectDatei.java“ zu erstellen. Das Konzept der Serialisierung, bzw. Deserialisierung wird hier angewendet. Das bedeutet, ein beliebiges Objekt wird in einen Byte-Strom verwandelt und in einer Datei gespeichert. Bei der Deserialisierung wird die Datei, als Objekt zurück in den Arbeitsspeicher gelesen.

```
//Klasse zur Verwaltung einzelner Gästebucheinträge
class GaesteEintrag implements Serializable
{
      private String name, email, homepage, comment, today;
      //Konstruktor
      GaesteEintrag(String name, String email, String homepage,
      String comment, String today)
      {
      this.name = name;
      this.email = email;
      this.homepage = homepage;
      this.comment = comment;
      this.today = today;
      }
      //get-Operationen
      String getName(){return name;}
      String getEmail(){return email;}
      String getHomepage(){return homepage;}
      String getComment(){return comment;}
      String getToday(){return today;}
}
```

LISTING 4: GAESTEEINTRAG.JAVA

Die Klasse GaesteEintrag.java modelliert die Gästebucheinträge.

```
//Container zur Verwaltung der Gästebucheinträge
//Als Datenstruktur wird ein Vektor verwendet
class GaesteBuch implements Serializable
{
      private Vector einVektor = new Vector();
      //Operationen
      void anfuegen(GaesteEintrag einEintrag)
      {
            einVektor.addElement(einEintrag);
      }
      GaesteEintrag getEintrag(int index)
      {
            return (GaesteEintrag)einVektor.elementAt(index);
      }
      int getAnzahlEintraege()
      {
            return einVektor.size();
      }
}
//Ende der Deklarationen von Klassen
%>
```

Die Klasse GaesteBuch.java dient als Container für die Einträge, alle Einträge werden als ein Objekt im Vektor gespeichert. Damit ist die Deklaration der Klassen beendet.

```
<body bgcolor="#FFCC99" text="#3300CC">

      <%
      //JSP-Skriptlet-Code
      //GaestebuchDatei lesen
      ObjektDatei meinSpeicher = new
      ObjektDatei("c:/temp/Gaestebuchdatei");
      GaesteBuch einGaesteBuch = (GaesteBuch)meinSpeicher.leseObjekt();
      //Wenn Datei noch nicht vorhanden, dann anlegen
      if(einGaesteBuch == null) einGaesteBuch = new GaesteBuch();
```

Der JSP-Skriptlet-Code definiert wo der Ablageort für die Datei definiert ist. Sollte die Datei noch nicht vorhanden sein, wird sie automatisch angelegt. Der Ablageort c:/temp/ muss aber zwingend vorhanden sein!

```
      //Auslesen des HTML-Formulars
      String name = request.getParameter("name");
      String email = request.getParameter("email");
      String homepage = request.getParameter("homepage");
      String comment = request.getParameter("eintrag");
```

Hiermit werden die Gästebuch-Parameter ausgelesen.

```
      //Aktuelles Datum
      Date date = new Date();
      //Deutsches Datumsformat wählen
      DateFormat formatDatumDeutsch =
      DateFormat.getDateInstance(DateFormat.DEFAULT,Locale.GERMANY);
      String date_str = formatDatumDeutsch.format(date);
```

Dann wird automatisch an jedes Objekt das aktuelle Datum angehängt.

```
      //Aktuelle Zeit
      Date time = new Date();
      //Deutsches Zeitformat wählen
      DateFormat formatZeitDeutsch =
      DateFormat.getTimeInstance(DateFormat.DEFAULT,Locale.GERMANY);
      String time_str = formatZeitDeutsch.format(time);
      String today  = date_str + " " + time_str;
```

Automatisch wird an jedes Objekt die aktuelle Zeit angehängt.

```
//Prüfen, ob ein Daten eingegeben wurden
//Hier nur Prüfung auf Name
if (name != null && name.length() > 0)
{
      //Eintrag in Klasse Gästebuch
      einGaesteBuch.anfuegen
      (new GaesteEintrag(name,email,homepage,comment,today));
      //Speichern des Gästebuchs
      meinSpeicher.speichereObjekt(einGaesteBuch);
}
%>
```

Um eine kleine Sicherheit vor gefälschten Einträgen einzubauen, ist es zwingend notwendig, dass ein Name beim Ausfüllen des Gästebuches eingetragen wird.

```
<b>Mein Gästebuch</b><br/>
<hr>
<%
//Anzeige des kompletten Gästebuchs
for (int i=0; i < einGaesteBuch.getAnzahlEintraege(); i++)
      {
            GaesteEintrag einEintrag = einGaesteBuch.getEintrag(i);
%>
Am <%=einEintrag.getToday()%> schrieb:</br>
<%=einEintrag.getName()%></br>
mit der E-Mail: <%=einEintrag.getEmail()%></br>
und der Homepage: <%=einEintrag.getHomepage()%></br>
diesen Eintrag: <%=einEintrag.getComment()%></br>
<hr>
<%}%>
</body>
```

Der letzte Abschnitt ruft das komplette Gästebuch auf.

Hierbei muss darauf verwiesen werden, dass dieses Gästebuch aus Software-technischer Sicht keine optimale Lösung ist, da die Funktionalität komplett in der JSP-Datei integriert ist. Besser ist es, die Logik außerhalb der JSP zu definieren und die Klassen dann über die JSP-Datei aufzurufen. (MVC-Entwurfsmuster)

Benutzeroberfläche des Gästebuches

Abbildung 11 – Beispiel Gästebuch

JSP und Java Beans

Je umfangreicher der Java-Code in einer JSP-Datei wird, desto schwieriger wird die Wartung und die Unterscheidung von Darstellung und Logik, Java Beans schaffen da Abhilfe. [24]

Was sind JavaBeans

Java Beans sind laut Sun konfigurierbare Container, welche Variablen speichern und darauf basierende Logik kapseln.

Die wichtigsten Merkmale einer Java-Bean sind:

1. Java Bean müssen einen parameterlosen Konstruktor besitzen, über den Sie erzeugt werden.
2. Java Beans sollten die persistenten Daten in Form von so genannten Attributen verwahren.
3. Die Attribute bestehen aus einer nicht-öffentlichen (private) Instanzvariablen und öffentlichen Zugriffsmethoden (public), welche umgangssprachlich auch Getter und Setter genannt wird. [25]

jsp:setProperty

diese Aktion setzt ein Property in der definierten Java-Bean.

```
<jsp:setProperty name="myBean" property="lastChanged" value="<%= new Date()%>" />
```

jsp:getProperty

diese Aktion holt sich ein Property von der definierten Java-Bean.

```
<jsp:getProperty name="myBean" property="lastChanged" />
```
[26]

[24] Buch: Markt und Technik J2EE Seite 93
[25] Buch: Markt und Technik J2EE Seite 94
[26] http://de.wikipedia.org/wiki/JavaServer_Pages

Einbindung von Beans in JSP-Seiten[27]

jsp:useBean

diese Aktion erstellt oder verwendet eine Java-Bean wieder. Ist eine Bean vom definierten Typ nicht vorhanden, wird sie also initial neu erstellt. Das optionale Attribut scope gibt an, wie lange die Bean zur Verfügung steht, d. h. in welchem Gültigkeitsbereich die Bean hinterlegt wird. Folgende Werte können definiert werden:

request

Attribute sind nur solange verfügbar, wie der Request existiert.

page

Attribute sind nur für die gegenwärtige JSP verfügbar. (Standard)

session

Attribute sind nur solange verfügbar, wie die Benutzer-Session existiert.

application

Attribute sind immer verfügbar.

„Verfügbar“ heißt hier, wie lange bzw. von wo der Entwickler auf die Bean zugreifen kann. Ist die Bean beispielsweise im Request-Scope, so kann der Entwickler mit dieser Bean sowohl innerhalb der aktuellen JSP als auch in nachgelagerten (inkludierten) arbeiten. „Page-Scope“ beschränkt den möglichen Zugriff auf die aktuelle JSP-Seite. „Session“ beschreibt alle Requests einer Benutzersitzung zu dieser Web-Anwendung. „Application“ sind alle Requests aller Benutzer der gleichen Web-Anwendung dieses Web-Servers.

Beispiel:

```
<JSP:USEBEAN ID="MYBEAN" CLASS="COM.FOO.MYBEAN" SCOPE="REQUEST">

</JSP:USEBEAN>
```

[27] http://de.wikipedia.org/wiki/JavaServer_Pages

Schlusswort

Die strikte Trennung von Logik und Präsentation, durch Benutzung des MVC-Entwurfsmusters, sorgt dafür, dass die Zusammenarbeit im Team gefördert wird. Web-Entwickler und Java-Programmierer sind bis zur Verschmelzung der Strukturen unabhängig voneinander, Web-Entwickler müssen die JSP-Syntax lernen und nicht Java. Das Einbinden vorhandener Java-Projekte ist schnell realisiert und die Anwendungen sind plattformunabhängig.

Quellenverzeichnis

Bücher

JavaServer Pages Seite 5, 6

JSP für Einsteiger Seite 13, 47

JSP für Einsteiger Seite 47

JSP für Einsteiger Seite 48

JSP für Einsteiger Seite 49

Servlets & JSP Seite 102

JSP für Einsteiger Seite 8

Servlets und JSP Seite XXIV

Servlets und JSP Seite XXIV

O´Reilly Servlets & JSP von Kopf bis Fuß Seite 49

Markt und Technik J2EE Seite 93

Markt und Technik J2EE Seite 94

JSP für Einsteiger Seite 31

Internet:

http://www.jsptutorial.org/content/introduction - aufgerufen am 05.02.10

http://de.wikipedia.org/wiki/Model_View_Controller - aufgerufen am 05.02.10

http://www.roseindia.net/jsp/Hello_World.shtml - aufgerufen am 05.02.10

http://www.werthmoeller.de/doc/microhowtos/tomcat/verzeichnisstruktur/ - aufgerufen am 05.02.10

http://de.wikipedia.org/wiki/Apache_Tomcat - aufgerufen am 06.02.10

http://www.werthmoeller.de/doc/microhowtos/tomcat/verzeichnisstruktur/ - aufgerufen am 06.02.10

http://www.konicsek.de/schwerpunkte/tomcat-konfiguration.htm - aufgerufen am 06.02.10

http://onjava.com/pub/a/onjava/2003/07/30/jsf_intro.html - aufgerufen am 06.02.10

http://www.mi.fh-wiesbaden.de/~barth/webanw/vorl/WebAnwPB6.pdf - aufgerufen am 07.02.10

http://de.wikipedia.org/wiki/Java_Server_Pages#Ausdrücke - aufgerufen am 07.02.10

http://de.wikipedia.org/wiki/Java_Server_Pages#Direktiven - aufgerufen am 07.02.10

http://de.wikipedia.org/wiki/Java_Server_Pages#Aktion - aufgerufen am 07.02.10

http://de.wikipedia.org/wiki/JavaServer_Pages - aufgerufen am 08.02.10

http://de.wikipedia.org/wiki/JavaServer_Pages - aufgerufen am 08.02.10

Abbildungsverzeichnis

Listings