

# Webanwendungen mit Java und JavaServerPages

ohne JS und ohne Framework

Michael Dienert

21. Januar 2018

## Inhaltsverzeichnis

<b>1</b>	<b>model2: Webanwendungen nach dem MVC-Prinzip</b>	<b>1</b>
<b>2</b>	<b>model2 mit Netbeans und tomcat</b>	<b>1</b>
2.1	Projekt anlegen . . . . .	1
2.2	Controller-Servlet erzeugen . . . . .	1
2.3	View anlegen . . . . .	2
2.4	Weiterleitung durch den Controller . . . . .	2
2.5	Session-Objekt . . . . .	3
2.6	Controller vervollständigen . . . . .	3
2.7	Expression Language . . . . .	4
2.8	Expression Language . . . . .	4
2.9	JSTL . . . . .	5
2.10	forEach mit JSTL . . . . .	5

# 1 model2: Webanwendungen nach dem MVC-Prinzip

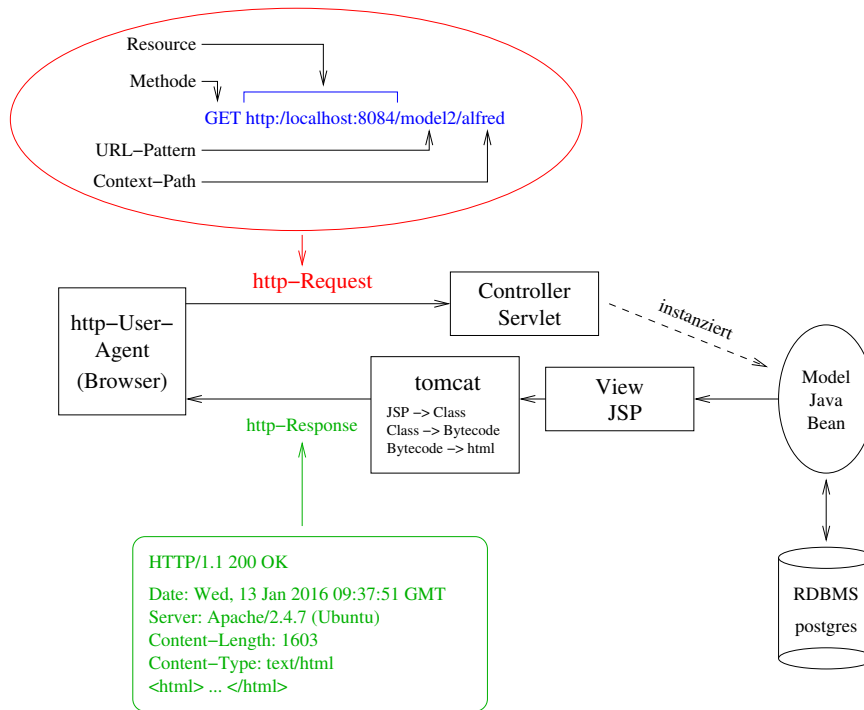


Abbildung 1: Prinzip einer model2-Anwendung

## 2 model2 mit Netbeans und tomcat

### 2.1 Projekt anlegen

- netbeans starten
- File-Menue → New Project → Java Web → Web Application
- Server (tomcat oder glassfish), Java EE - Version und *Kontext-Pfad* wählen. → **Next**
- Hier, für Schulprojekte bitte **kein** Framework wählen.
- Finish!

### 2.2 Controller-Servlet erzeugen

- Im Projects-Navigator (rechte Spalte): Mouse-Click-Rechts auf Projektname
- Projektname → **New** → **Servlet**

- Namen der Servlet-Klasse und *unbedingt* Package-Namen vergeben → **Next**
- wenn gewünscht: *Add Information to deployment descriptor (web.xml)*
- *Servlet Name* so lassen wie Klassenname, URL-Pattern nach Wunsch.
- Das URL-Pattern wird auf dem Client wie ein Dateiname an den Context-Path gehängt.
- Der Server startet dann das entsprechende Servlet.
- Beispiel-URL:

```
http://localhost:8084/AdressSammler/Controller/index.html
```

- Auszug `web.xml`:

```
<servlet-mapping>
  <servlet-name>Controller</servlet-name>
  <url-pattern>/Controller</url-pattern>
  <url-pattern>/index.html</url-pattern>
</servlet-mapping>
```

- In einer Webanwendung kann es mehrere Controller geben.

## 2.3 View anlegen

- Im Projects-Navigator (rechte Spalte): Mouse-Click-Rechts auf das Verzeichnis **WEB-INF**
- WEB-INF → **New** → **Folder**
- Dem neuen Verzeichnis den Namen **view** geben. → Finish
- Nun werden eine oder mehrere JSP-Seiten in *view* erzeugt:
- Rechtsklick auf *view* → **New** → **other** (ganz unten), dann File-Type **JSP**, Dateiname wählen, Finish
- *tomcat* ist ein in Java geschriebener Webserver, d.h. *tomcat* liefert Dateien aus, die über einen http-Request angefordert werden.
- Direkt ausgeliefert werden aber keinesfalls Dateien, die im Verzeichnis **WEB-INF** stehen, das ist von aussen nicht zugänglich.
- D.h. die Dateien unserer View sind nicht direkt aufrufbar. Ausgeliefert werden können sie nur, über ein sog. *Forwarding* des Controller-Servlets.
- Was angezeigt wird, kann auf diese Weise vom Controller gesteuert werden.

## 2.4 Weiterleitung durch den Controller

- Code für das Weiterleiten auf Seiten innerhalb von WEB-INF/view

```
String url = "/WEB-INF/view/formular.jsp";

ServletContext sc = getServletContext();
RequestDispatcher rd = sc.getRequestDispatcher(url);
rd.forward(request, response);
```

- Je nach User-Aktion, kann man später den Wert des url-Strings beeinflussen und damit auf unterschiedliche Seiten weiterleiten.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP Seite mit Formular</title>
</head>
<body>
<h1>JSP-Seite mit Formular</h1>

<h2>Sage mir Deinen Namen und ich sage Dir, wie Du heisst:</h2>
<form name="form" action="Controller" method="GET">
<table id="tabelle">
<tr>
<td>Vorname:</td>
<td><input name="vorname" type="text" value="${person.vorname}"/></td>
</tr>
<tr>
<td>Nachname:</td>
<td><input name="nachname" type="text" value="${person.nachname}" /></td>
</tr>
</table>
<input type="hidden" name="action" value="REFRESH"/>
<input type="submit" name="submit" value="Hau Wech!"/>
</form>
```

```
<h2>Du heisst ${person.vorname} ${person.nachname}!</h2>

<form name="form" action="Controller" method="GET">
<input type="hidden" name="action" value="CLEAR"/>
<input style="visibility:${sichtbarkeit}"
type="submit"
name="submit"
value="Einträge löschen"/>
</form>

</body>
</html>
```

## 2.5 Session-Objekt

- tomcat verwaltet für jede http-Sitzung eines Users ein sog. *Session-Objekt*
- Über z.B. Cookies oder URL-Rewriting kann tomcat ein Session-Objekt einem Nutzer auch über mehrere Seiten hinweg zuordnen.
- Das session-Objekt kann man sich wie einen Behälter vorstellen, in den man beliebige Objekte unter einem Schlüsselwort ablegen kann (ähnlich einer Hash-Map).
- Daten, die für alle Benutzer zusammen gespeichert werden sollen, können in einer gleichartigen Datenstruktur (ebenfalls ähnlich einer Hash-Map) im ServletContext-Objekt gespeichert werden.

## 2.6 Controller vervollständigen

- Der Quelltext des Controllers wird erweitert:

- Holen des Session-Objekts.
- Speichern beliebiger Daten im Session-Objekt.
- ggfs. Neuanlegen und Speichern eines model-Objekts in der Session.
- Auslesen der QUERY\_STRING oder POST-Daten und Bestimmung der Benutzer-Aktion.
- Fallunterscheidung je nach User-Aktion.
- Auslesen der QUERY\_STRING oder POST-Daten und Zuweisung der Model-Attribute.
- Forwarding auf die nächste JSP-Seite.

```
String url = "/error.jsp";
String action = request.getParameter("action");

if (action == null) { action = "NULL"; }

//session holen
HttpSession session = request.getSession();

//attribut fuer sichtbarkeit einiger buttons in der view
session.setAttribute("sichtbar", new String("hidden"));

//model im session-objekt ablegen
Person person;
person = (Person) session.getAttribute("person");

if (person == null) {
    person = new Person();
    session.setAttribute("person", person);
}

//servlet context holen
ServletContext sc = getServletContext();
AdressListe adrListe;
adrListe = (AdressListe) sc.getAttribute("liste");

if (adrListe == null) {
    adrListe = new AdressListe();
    adrListe.setAdressListe(new ArrayList<Person>());
    sc.setAttribute("liste", adrListe);
}
```

```
if (action.equals("REFRESH")) {

    url = "/WEB-INF/view/index.jsp";
    person.setVorname(request.getParameter("vorname"));
    person.setNachname(request.getParameter("nachname"));

    session.setAttribute("sichtbar", new String("visible"));

} else if (action.equals("CHECKOUT")) {

    url = "/WEB-INF/view/confirm.jsp";
    adrListe.getAdressListe().add(person);
    new JaxbMain(adrListe);

} else if (action.equals("NEUSTART")) {

    url = "/WEB-INF/view/index.jsp";
    person = new Person();
    session.setAttribute("sichtbar", new String("hidden"));
    session.setAttribute("person", person);

} else {

    url = "/WEB-INF/view/index.jsp";
}

//ServletContext sc = getServletContext();
RequestDispatcher rd = sc.getRequestDispatcher(url);
rd.forward(request, response);
}
```

### 2.7 Expression Language

- Um Daten aus dem Model oder direkt Attribute des Session-Objekts (s.o.) in der View darzustellen, gibt es eine spezielle Syntax, die *Expression Language*.

- Möchte man z.B. das Attribut `vorname` des Models auf der JSP erscheinen lassen, kann man folgenden Ausdruck in der JSP verwenden: `${person.vorname}`
- `person` ist dabei exakt der Name des Attributs in der Session
- `vorname` muss dabei zur Get-Methode `getVorname` im Model passen.

## 2.8 Expression Language

- Möchte man komplexere Datenstrukturen verwenden, wie z.b. so etwas:

```
${plan.besetzung.musiker.vorname}
```

muss man die verwendeten Klassen der JSP mit einer `import`-Anweisung bekannt machen.

```
<%@ page import="model2muster.Gigplan" %>
```

## 2.9 JSTL

- JSTL steht für Java Server Pages Standard Tag Library
- Möchte man eine JSTL verwenden, muss sie der JSP bekannt gemacht werden:

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

- Zwingend notwendig ist es dabei aber auch, die JSTL zu den Libraries des Netbeans-Projekts hinzuzufügen!!
- Rechtsklick auf `Libraries` -> `Add Library ...` -> `JSTL 1.2.2` wählen.

## 2.10 forEach mit JSTL

- Mit der *Expression Language* können wir *einzelne* Attribute eines Models auf der JSP darstellen.
- Enthält das Model aber eine Sequenz von Daten (Liste, Set usw.), kann diese wie folgt ausgegeben werden:

```
<c:forEach items="${plan.besetzung}" var="musiker">
  ${musiker}<br/>
</c:forEach>
```