2B
Übung WME
Java Servlets

Martin Lehmann - Martin.Lehmann4@mailbox.tu-dresden.de

Technische Universität Dresden
Fakultät Informatik
Institut für Software- und Multimediatechnik
Lehrstuhl für Multimediatechnik

Gliederung

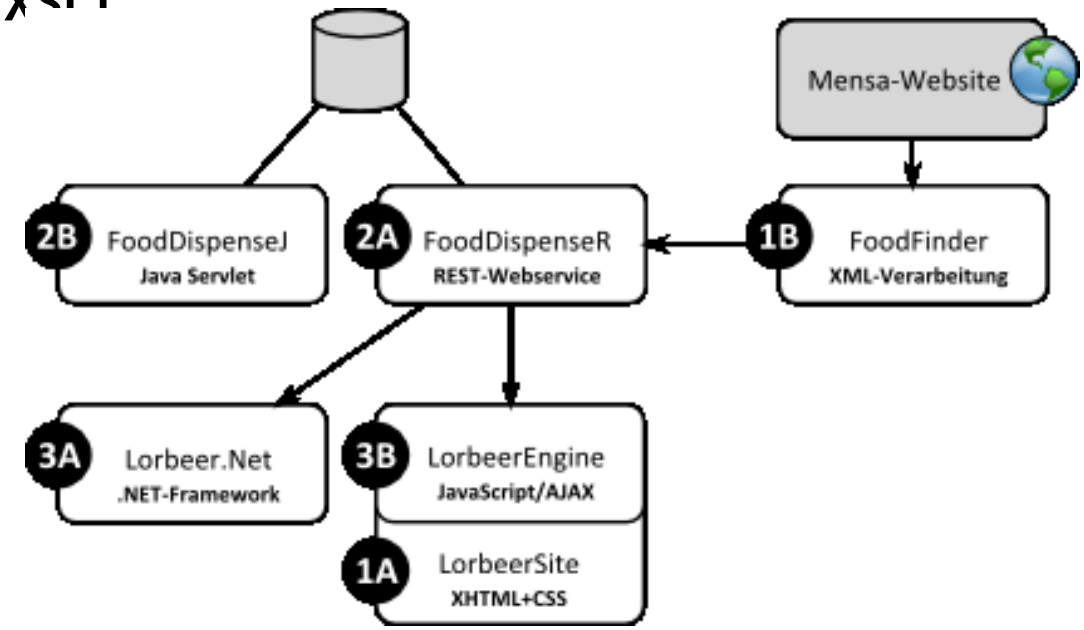
- Organisatorisches
  - Ablauf der Übung
  - Themen
  - Abgabemodalitäten
  - Gruppenzuordnung
- Aufgabenstellung
  - Hinweise zur Umsetzung
- Einstieg ins Thema
  - Servlet Container
  - MVC Paradigma
  - Java Beans, Java Servlets, Java Server Pages
- Links

Organisatorisches: Ablauf der Übung

- Terminplan
  - 17.11.11 Übungseinführung 2B (Pflicht)
  - 24.11.11 Konsultation (Teilnahme freiwillig)
  - 01.12.11 Konsultation (Teilnahme freiwillig)
  - 08.12.11 Konsultation (Teilnahme freiwillig)
  - 15.12.11 Konsultation (Teilnahme freiwillig)
  - 19.12.11 Abgabe der Ergebnisse (bis spätestens 13:00 Uhr)
  - …
  - 01.02.12 Abschlussveranstaltung und Präsentation (Pflicht)

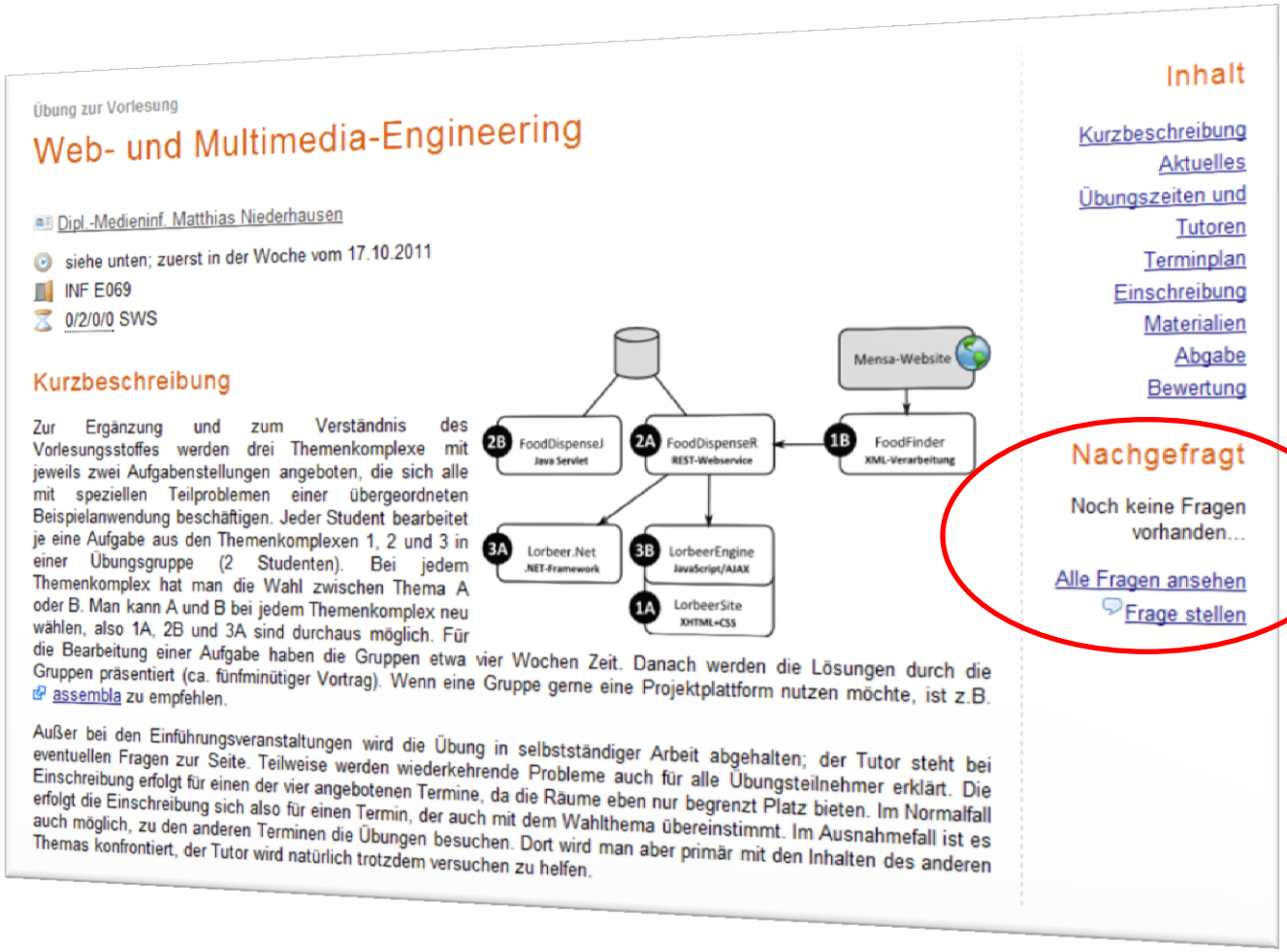
- Details siehe Übungskalender

- Themenkomplex 1
  - 1A) XHTML und CSS
  - 1B) XML-Verarbeitung und XSLT
- Themenkomplex 2
  - 2A) RESTful Webservice
  - 2B) Java Servlets
- Themenkomplex 3
  - 3A) .NET/Silverlight
  - 3B) JavaScript und AJAX



- **Informationen zu den Übungen**

  http://www.mmt.inf.tu-dresden.de/Lehre/Wintersemester_11_12/WME/Uebung/index.xhtml



Organisatorisches: Abgabemodalitäten

- via Upload auf sFTP Server:
  - Servername: serv9.inf.tu-dresden.de
  - Port 22
  - Verzeichnis: /zbv/WME/
  - Authentifizierung mit ZIH-Login

- Abgabe der Lösung
  - Verzeichnis anlegen
    - Ergebnisse_2_B/<Gruppe>_<Matrikel1>_<Matrikel2>
  - Dateien hochladen
    - keine Umlaute in Dateinamen!
  - Zugriffsrechte nicht verändern!

Organisatorisches: Abgabemodalitäten

- Lösung der Aufgabe
  - zwei Studenten bilden ein Team
    - auf der Einschreibliste eintragen!
  - Durchschleifen wird nicht toleriert
  - Abgabetermin ist verbindlich (19.12.2011 - bis 13Uhr)
    - Abgabeverzeichnisse werden nach Ablauf der Bearbeitungszeit gesperrt
  - jede Gruppe hat eine eigenständige Lösung abzugeben
  - ausgewählte Ergebnisse werden am letzten Übungstermin (01.02.2012) von den Gruppen präsentiert

Aufgabe 2B: *Verarbeitung von HTTP-Requests auf Serverseite*

- Das Portal Lorbeerblatt soll Studenten eine Plattform bieten, auf der sie sich über das aktuelle Mensaessen austauschen können. Die Benutzer sollen Kommentare, Bewertungen und eigene Bilder der verschiedenen Speisen einstellen können. Diese müssen aus den eingehenden HTTP-Requests ausgelesen und gespeichert werden.

Aufgabenstellung im Detail

- Entwicklung des FoodDispenseJ für Lorbeerblatt
  - Serverseitige Verarbeitung der HTTP-Requests durch Java Servlets
  - Funktionen:
    - Hinzufügen eines Kommentars (mit Name und Datum) zu einem Essen
    - Upload eines Bildes zu einem Gericht (Speicherung auf der Festplatte)
    - Debug-Ausgabe der Kommentare und Anzahl gespeicherter Bilder zu einem Gericht
    - Erweiterung der vorhandenen Gerichten durch die genannten Funktionen
  - Der beste Entwurf wird im weiteren Verlauf der Übung weiterverwendet!

Aufgabenstellung im Detail

- Randbedingungen
  - Lauffähige Webanwendung: Die abgegebene Webanwendung soll ohne weitere Konfigurationen unmittelbar lauffähig sein!
  - Java-Objekte, sowie die Servletausgabe müssen reale und korrekte Daten der Mensawebsite beinhalten
    - http://www.studentenwerk-dresden.de/mensen/speiseplan/
  - Lizenzrechte bei verwendeten Fremdinhalten beachten!
- Abgabe
  - Lauffähige Webanwendung (als .war-Datei)
  - gepackte Quellen (z.B. als gepacktes Eclipseprojekt)
  - kurze Ausführungsbeschreibung

Aufgabenstellung im Detail

- Material
  - lorbeerblatt-core.jar
    - enthält vorgefertigte Javaklassen zur Speicherung der geparsten und selektierten Informationen von der Mensawebsite
  - foodfinder.jar
    - Javaklassen aus der Übung 1B, die für das Parsing zuständig sind
    - beinhaltet das XSL-Stylsheet für die Transformation
  - DataBuilder.java
    - ist die Anbindung an den FoodFinder
    - parst mittels der enthaltenden transform.xsl die Mensawebsite und speichert die Ergebnisse in den DAOs
    - Zugriff auf die Daten über MensaDao.instance.*

- Editoren und Entwicklungsumgebungen
  - Notepad++
    - Texteditor mit Syntax-Highlighting
    - klein und schnell
    - zur Ansicht und Bearbeitung der JSPs oder XML-Konfigurationen der Webanwendung
  - Eclipse
    - komplexe Entwicklungsumgebung, ursprünglich für Java
    - für die Umsetzung der Aufgabe unabdingbar
    - bietet die Möglichkeit die Anwendung in ein ApplicationServer zu deployen
    - einfache und schnelle Erstellung von Webprojekten

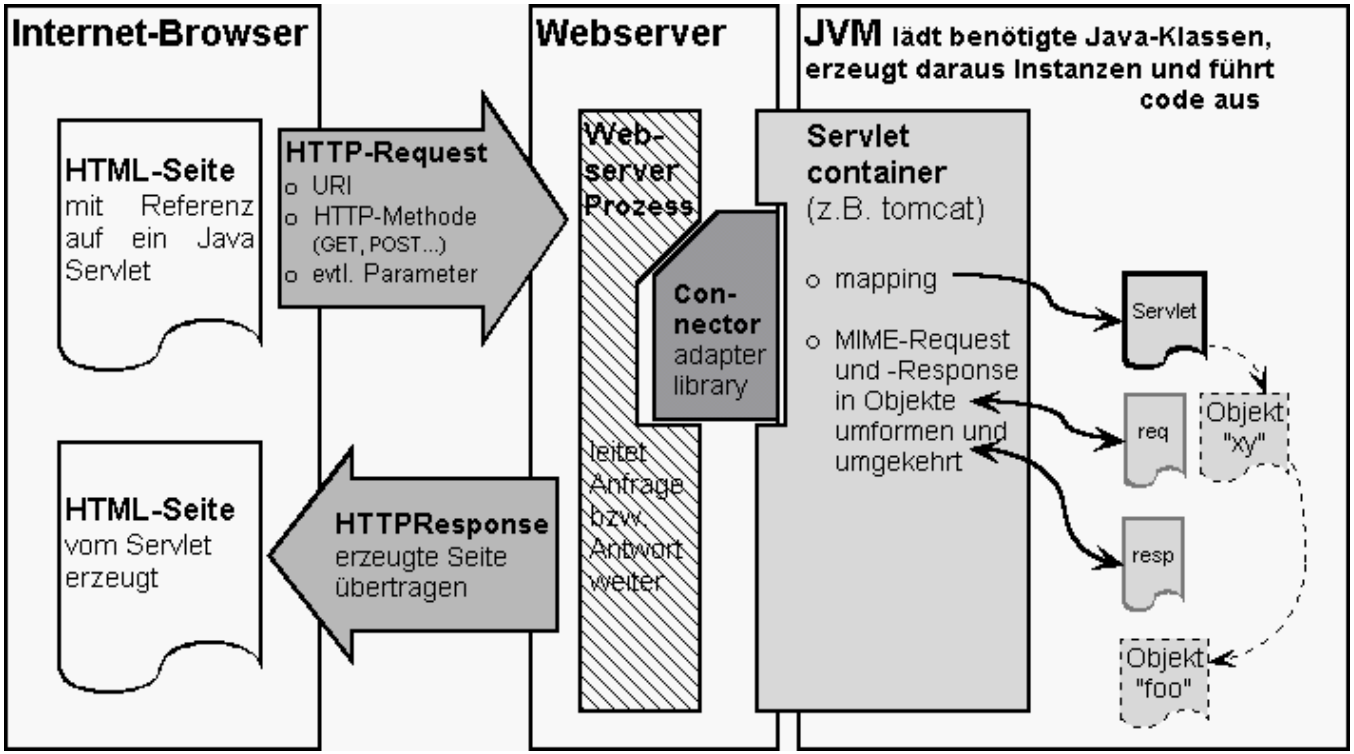
Erstellung einer Applikation in Eclipse

- Dynamic Web Project erstellen
- als Target runtime den Apache Tomcat 7.0 angeben (vorher unter http://tomcat.apache.org/ runterladen)
- unter WebContent/WEB-INF/lib alle benötigten JARs speichern
- unter WebContent/WEB-INF die web.xml erstellen
- in dem Package src werden alle verwendeten Klassen hinterlegt
- Erstellung eines Servlets (erbt von javax.servlet.http.HttpServlet) in einem eigenen Package (Bsp.: de.mmt.lorbeerblatt.controller) und Implementierung der doGet-Methode
- in der web.xml erfolgt das Servlet-Mapping von der URL http://localhost:8080/{Projektname}/CustomServlet auf diese Servletklasse

- **Aufbau einer einfachen web.xml:**

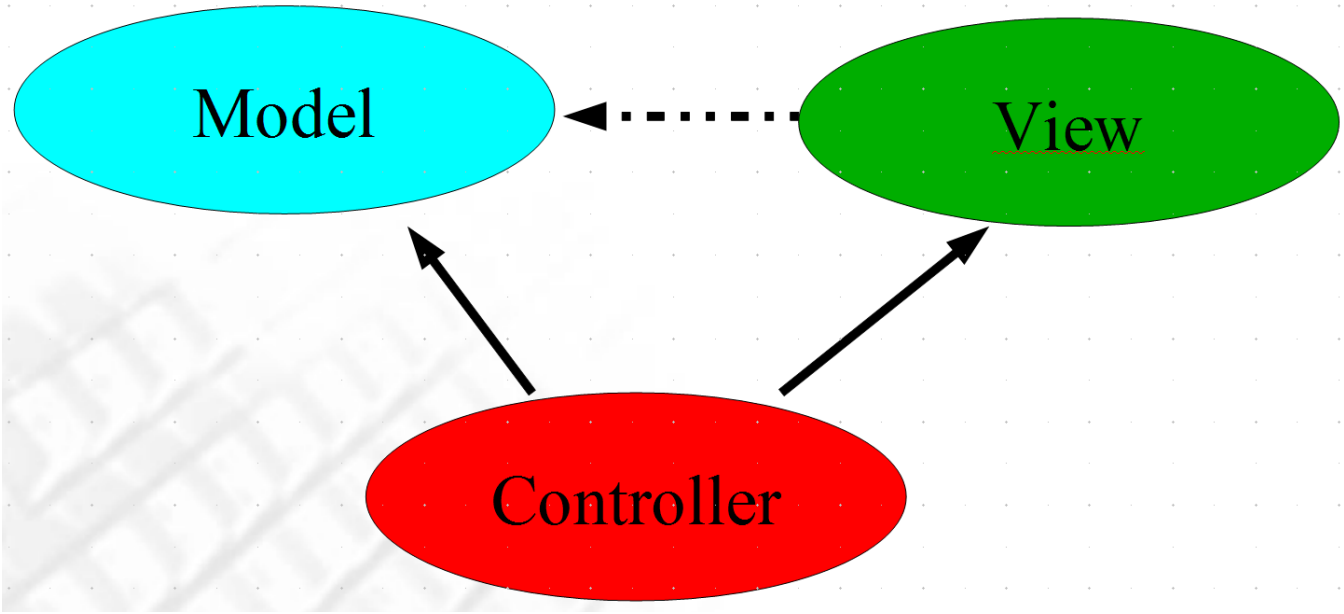
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
            http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
    <display-name>FoodFinder</display-name>
    <servlet>
        <servlet-name>FoodFinder</servlet-name>
        <servlet-class>de.mmt.lorbeerblatt.foodfinder.FoodFinderServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>FoodFinder</servlet-name>
        <url-pattern>/FoodFinder</url-pattern>
    </servlet-mapping>
</web-app>
```

- **Starten der Applikation:**
  - Projekt -> Run As / Run on Server
  - Tomcat v7.0 Server auswählen
  - Server runtime environment setzen (Pfad zum entpackten Tomcat 7 einstellen)
  - nach erfolgreichem Start ist die Anwendung unter http://localhost:8080/{Projektname}/CustomServlet erreichbar



- Servlet Container
  - Multithreading
  - läd & initialisiert Servlets in der Servlet-Engine (Lifecycle-Management)
  - Requestweiterleitung an das entsprechende Servlet

- MVC-Paradigma
  - Architekturmuster zur Strukturierung von Webanwendungen
  - Entkopplung von Anwendungslogik, Darstellung und Manipulation
  - flexibler Programmentwurf erleichtert die spätere Erweiterung und die Wiederverwendbarkeit der einzelnen Komponenten
  - Model:
    - enthält die darzustellenden Daten bzw. den Zustand der Applikation
  - View:
    - ist für die Darstellung der Modelldaten zuständig
  - Controller:
    - verwendet Präsentationen (Views) und leitet auf diese weiter
    - nimmt Benutzeraktionen entgegen und wertet sie aus
    - führt die Datenmanipulation im Modell aus



- Client richtet Request an den Controller
- Controller verarbeitet Request, aktualisiert das Model und leitet an die passende View weiter
- Komponenten der View greifen auf das Model zu

- Java Beans (JB)
  - sind Javaklassen, die einen parameterlosen Konstruktor haben
  - der Zugriff auf Attribute (JB Properties) wird durch getter/setter-Methoden realisiert
  - werden als Modell im MVC-Paradigma bezeichnet

Quelle: http://michelledraws.wordpress.com/2010/02/26/java-bean/

- Java Servlets (JSL)
  - sind serverseitige Java-Komponenten (Programmcode)
  - erzeugen HTML-Code und schreiben diesen in den Response

    → programmzentrierter Ansatz
  - werden in Servlet-Containern (z.B. Tomcat) ausgeführt
  - können mehrere Client-Anfragen parallel bearbeiten
  - Implementierung der JSL-API (java.servlet.*)

  - Beispiel HttpServlet:
    - Aufruf des Servlets mit doGet() und doPost() durch den ServletContainer
    - HttpServletRequest- und HttpServletResponse-Objekte werden übergeben

Beispiel: einfache JSL

Wird HellowServlet aufgerufen, erscheint "Hello" im Browser

```
//HelloSevelet.java
package buch;
import java.io.*;
import javax.servlet.*;
import javax.servelet.http.*;
public class HelloServelet extends HttpServlet {
  public void doGet(HttpServletRequest req,
                    HttpServletResponse res,
                    throws ServletException, IOException {
      res.setContentType("text/html");
      PrintWriter out=res.getWriter();
      out.println ("<HTML><HEAD>
      out.println (<TITEL>HellowServlet</TITEL>");
      out.println ("</HEAD><BODY>");
      out.println ("<BIG>Hellow</BIG>");
      out.println ("</BODY></HTML>");
} }
```

HttpServlet implementiert die Methoden **doGet()** und doPost(), die beim GET- und POST-Request durch die Laufzeitumgebung aufgerufen werden.

Mit der **setContentType()** Methode wird zuerst der Content-Type im Header für die folgende Respons gesetzt

getWriter() liefert eine Referenz auf die Klasse PrintWriter, zum schreiben von Daten in das **HTTP-Response Objekt**

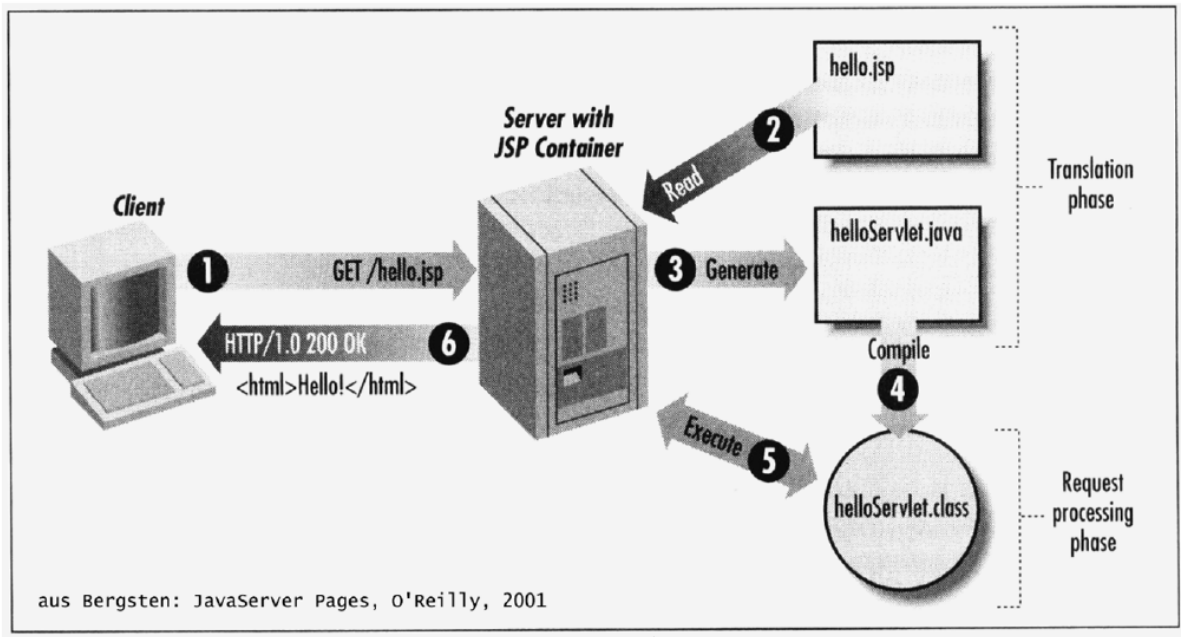
- Java Server Pages (JSP)
  - ermöglichen die Trennung von Präsentation und Anwendung
  - werden als View im MVC-Paradigma bezeichnet
  - sind eine Ausprägung von JSL (JSP-Engine kompiliert die Seiten in ein JSL)
  - Einbettung von Java-Code und JSP-Aktionen in HTML- oder XML-Seiten

    → dokumentenzentrierter Ansatz
  - Zugriff auf Java Beans im Backend

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

<jsp:useBean id="myBean" class="de.mmt.lorbeerblatt.webservice.util.MyBean" scope="request" />
<html>
  <head><title>Hello Application</title></head>
  <body>
    <p>Greetings, it is now <c:out value="<%= new Date().toLocaleString() %>"/></p>

    <jsp:getProperty name="myBean" property="valueOfBean" />

    <%-- Formular wird zu dem CommentController geschickt, welcher die doPost()-Methode überschreiben muss! --%>
    <form action="/doComment" method="post">
        <input type="submit" name="submit" value="comment">
    </form>
  </body>
</html>
```

Aufruf von JavaServer Pages

1. Der Client sendet eine GET-Anfrage an den Server.
2. Die *JSP-Engine* liest die JSP-Datei ein
3. Daraus erzeugt sie automatisch ein *Servlet*-Programm (Quellcode)
4. Diese wird kompiliert,
5. instanziiert und ausgeführt
6. Das Ergebnis wird an den Client gesendet



- Bei allen folgenden Anforderungen der Seite existiert das *Servlet* bereits und kann die Anforderung somit direkt beantworten

- JSP-Sprachkonstrukte
  - werden in Skriptlets, JSP-Ausdrücken, JSP-Kommentare und Deklarations- und Interpreter-Anweisungen unterschieden
  - Skriptlets sind Java-Anweisungen in einem HTML-Dokument:

    ```
    <% int variable = 0; out.println("Der Wert der Variable ist: " + variable); %>
    ```

  - Deklarationen dienen der Definition von Variablen und Methoden:

    ```
    <%! int variableMeinerKlasse = 0; %>
    ```

  - Interpreter-Anweisungen erlauben den Import von Java-Packages oder externer Dateien:

    ```
    <%@ page import="java.util.*" %>
    <%@ include file="BeispielDatei.ext" %>
    ```

- JSP-Sprachkonstrukte
  - JSP-Ausdrücke stellen die Kurzform für out.println() dar:

    Die Klassenvariable ist <%= variableMeinerKlasse %>

    Der Benutzername ist <%= request.getParameter("user") %>
  - JSP-Kommentare werden nicht mit kompiliert:

    <%-- Kommentar innerhalb einer JSP --%>
  - JSP-Aktionen:
    - sind in XML notierte, standardisierte JSP-Tags
    - werden in der JSP-Engine bei der JSL-Generierung in Java-Code umgesetzt
    - dienen dem Zugriff auf Skripte und JavaBeans (Objekt erzeugen, Properties auslesen und setzen, ...)

    <jsp:aktion attribut1="wert1" attribut2="wert2" ... />

- JSP-Sprachkonstrukte - Beispiel JSP-Aktionen
  - Importieren der header.jsp in das aktuelle Dokument:

```
<jsp:include page="header.jsp">
    <jsp:param name="extraparam" value="myvalue"/>
</jsp:include>
```

  - Verwendung einer JavaBean mit definierten Scope:

```
<jsp:useBean id="myBean" class="com.foo.MyBean"
scope="request"></jsp:useBean>
```

  - Setzen der Property in der myBean-Klasse:

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

  - Auslesen der Property in der myBean-Klasse:

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

- JSP-Sprachkonstrukte - JSP 2.0
  - JSPX-Dokumente entsprechen wohlgeformte XML-Dokumenten

```
<?xml version="1.0" encoding="utf-8"?>
<jsp:root xmlns="http://www.w3.org/1999/xhtml"
        xmlns:jsp="http://java.sun.com/JSP/Page"
        version="2.0">
    <jsp:output doctype-root-element="html"
        doctype-public="-//W3C//DTD XHTML 1.1//EN"
        doctype-system="http://www.w3c.org/TR/xhtml11/DTD/xhtml11.dtd" />
    <jsp:directive.page contentType="text/html; charset=utf-8" language="java" />
    <html xmlns="http://www.w3.org/1999/xhtml">
       XHTML- und/oder JSP-Elemente
    </html>
</jsp:root>
```

  - Expression Language (EL) ermöglicht den Zugriff auf Variablen, Konstanten oder abstrakte Datenstrukturen:

    ${1+2}, ${PI/2}, ${person.name}

Zum Weiterlesen…

- Tutorien & Co. (Java Servlets, JSP, …)
  - http://www.jsptutorial.org/content/jstl_core
  - http://static.se.uni-hannover.de/lehre/tutorials/JavaEE-HelloWorld-Servlet-with-Eclipse.php
  - http://static.springsource.org/docs/Spring-MVC-step-by-step/
  - http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/
  - http://www.servletworld.com/
- kostenloses Repository
  - http://www.github.com
  - http://www.assembla.com
- Kontakt
  - Martin.Lehmann4@mailbox.tu-dresden.de