

Java Server Pages & Servlets

Einrichtung der Umgebung

- ❖ apache-tomcat-7.0.59 unter /home/attendee/tools entpacken
- ❖ apache-ant-1.9.4 unter /home/attendee/tools entpacken

Einrichtung Umgebung

- ❖ scott_dump eingespielt über pgAdmin III
- ❖ ANT_HOME in .bashrc auf ant installation setzen
- ❖ catalina-ant.jar, tomcat-util.jar, tomcat-coyote.jar aus dem {tomcat_home}/lib in {ANT_HOME}/lib kopiert
- ❖ tomcat-juli aus {tomcat_home}/bin nach {ANT_HOME}/lib kopiert.

Beispiel tomcat-users.xml

The screenshot shows a terminal window titled "conf - nano - 80x24". The title bar also displays "File: tomcat-users.xml". The main area of the window contains the XML configuration for Tomcat users:

```
-->
<!--
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <user username="tomcat" password="tomcat" roles="tomcat"/>
  <user username="both" password="tomcat" roles="tomcat,role1"/>
  <user username="role1" password="tomcat" roles="role1"/>
-->
<role rolename="manager-gui"/>
<role rolename="manager-script"/>
<user username="tomcat" password="s3cret" roles="manager-gui,manager-script"/>

</tomcat-users>
```

At the bottom of the window, there is a menu bar with various keyboard shortcuts for navigating and editing the file.

Einrichten der Umgebung

- ❖ ant war undeploy deploy im Projekt autovermietung aufrufen.

Einrichten der Umgebung

- ❖ in eclipse neues Java-Projekt anlegen
- ❖ autovermietung_2.zip entpacken und den Verzeichnis-Inhalt in das neue Projekt kopieren.

Was sind Servlets

- ❖ Java-Programme, die in einem besonders präparierten Java-Webserver ausgeführt werden.
- ❖ Java-Programme werden als Klassen vom Webserver geladen und dort auch verwaltet und mit einer besonderen Serlet-Schnittstelle angesprochen.(Servlet-Container)

Die Servlet-API

- ❖ javax.servlet
- ❖ ein Servlet erwartet besondere Oberklasse
- ❖ und realisiert eine doGet-Methode
 - ❖ Methode ruft der Container immer dann auf, wenn der Client eine Standardanfrage über Http sendet.
 - ❖ doGet schreibt in einen besonderen Ausgabestrom.

Servlet-Beispiel

```
public class EasyServlet extends HttpServlet {
    /**
     *
     */
    private static final long serialVersionUID = -2870712050301404984L;

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws IOException {
        PrintWriter out = res.getWriter();
        out.println("<html>");
        out.println("<body>");
        out.println("<h1>Ui, ich bekomme Antwort vom Server, super!!</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

JSP

- ❖ ein Servlet ist eine Java-Klasse, die sich um die Ausgabe des HTMLs kümmert.
- ❖ JSP ist HTML-Seite mit eingebettetem Java-Code
 - ❖ flexibler in der Handhabung(Webdesign)

Beispiel JSP

```
<%@ page import="java.text.SimpleDateFormat" %>
<html>
    <head>
        <title>Index Page</title>
    </head>
    <body>
        <html><body>
            <% java.util.Date date = new java.util.Date();
               SimpleDateFormat simpleDateFormat = new SimpleDateFormat("dd.MM.yyyy");
            %>
            <center><h1><%=simpleDateFormat.format(date)%></h1></center>
            <center><h1>Willkommen</h1></center>
        </body>
    </html>
```

JSP-Compiler

- ❖ SP-Skripte werden vom Server automatisch in Servlets übersetzt.
- ❖ Server weiß JSP von normalen HTML-Seiten zu unterscheiden
- ❖ transformiert mithilfe eines JSP-Übersetzers aus der JSP ein Servlet und ruft die bekannten Servlet-Methoden auf
- ❖ danach benutzt der Servlet-Container direkt die übersetzte Klasse

Servlet-Container

- ❖ Container, der alle Servlets verwaltet
 - ❖ Einbettung in Webserver oder Applikationserver
- ❖ Container leitet Anfragen an Container weiter
- ❖ verwaltet Lebenszyklus eines Servlets
- ❖ spricht genau das Servlet an, das der Benutzer nutzen wollte, und übergibt Datenströme zur Ein- und Ausgabe.
- ❖ über Eingabekanal optional Formulardaten.

Servlet-fähige Webcontainer

- ❖ Apache tomcat
 - ❖ <http://tomcat.apache.org/>
- ❖ Jetty
 - ❖ <http://jetty.mortbay.org/>
- ❖ GlassFish
 - ❖ <https://glassfish.dev.java.net/>

installierter tomcat

- ❖ tomcat 7
- ❖ /home/attendee/tools/tomcat(TOMCAT_HOME)

Verzeichnisse des tomcat

Ordner	Beschreibung
<i>bin</i>	Ordner mit Batch-Skripten zum Starten/Beenden des Servers
<i>conf</i>	Konfigurationsdateien
<i>lib</i>	Jar-Dateien von Tomcat und für eigene Webapplikationen
<i>logs</i>	Logging-Dateien
<i>temp</i>	Ordner für temporäre Dateien
<i>webapps</i>	Webapplikationen
<i>work</i>	Servlets, die aus JSPs generiert wurden

Starten und Beenden

- ❖ {TOMCAT_HOME}/bin/startup.sh
- ❖ {TOMCAT_HOME}/bin/shutdown.sh

<http://localhost:8080/>

Konfiguration

- ❖ {TOMCAT_HOME}/conf
 - ❖ server.xml -> z.B. Anpassungen des Ports
- ❖ tomcat-users.xml

Beispiel tomcat-users.xml

The screenshot shows a terminal window titled "conf - nano - 80x24". The title bar also displays "File: tomcat-users.xml". The main area of the window contains the XML configuration for Tomcat users:

```
-->
<!--
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <user username="tomcat" password="tomcat" roles="tomcat"/>
  <user username="both" password="tomcat" roles="tomcat,role1"/>
  <user username="role1" password="tomcat" roles="role1"/>
-->
<role rolename="manager-gui"/>
<role rolename="manager-script"/>
<user username="tomcat" password="s3cret" roles="manager-gui,manager-script"/>

</tomcat-users>
```

At the bottom of the window, there is a menu bar with various keyboard shortcuts for navigating and editing the file.

Übung

- ❖ Uebung1

Webapplikationen

- ❖ definiert logische Struktur der Elemente, die zu einer Webanwendung gehören
 - ❖ statische Webseiten, Bilder, JSPs, Servlets, externe Bibliotheken etc.
- ❖ pro Webapp ein eigenes Verzeichnis

WEB-INF

- ❖ {TOMCAT_HOME}/webapps/examples/WEB-INF

WEB-INF

- ❖ hier stehen Objekte, die der Webserver nicht nach außen freigibt, etwa Servlet-Klassen.
- ❖ web.xml
 - ❖ Deployment-Descriptor
- ❖ classes
 - ❖ übersetzte Java-Klassen
- ❖ lib
 - ❖ sonstige externe Bibliotheken

Es muss nicht immer webapps sein...

- ❖ Eintrag server.xml

```
<Context path="/beispiel"  
        docBase="/Users/schulung/easysample"  
        reloadable="true" />
```

Servlets

```
import java.io.IOException;

public class EasyServlet extends HttpServlet {
    /**
     *
     */
    private static final long serialVersionUID = -2870712050301404984L;

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws IOException {
        PrintWriter out = res.getWriter();
        out.println("<html>");
        out.println("<body>");
        out.println("<h1>Ui, ich bekomme Antwort vom Server, super!!</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

HttpServlet

- ❖ `protected void service(HttpServletRequest req, HttpServletResponse resp)`
 - ❖ Empfängt alle HTTP-Requests und leitet sie aufgrund der unterschiedlichen HTTP-Methoden auf die jeweiligen doXXX()-Methoden weiter.
- ❖ `protected void doDelete(HttpServletRequest req, HttpServletResponse resp)`
 - ❖ Wird von der service()-Methode aufgerufen, wenn ein HTTP-DELETE kommt.
- ❖ `protected void doGet(HttpServletRequest req, HttpServletResponse resp)`
 - ❖ Wird von der service()-Methode aufgerufen, wenn ein HTTP-GET kommt.
- ❖ `protected void doOptions(HttpServletRequest req, HttpServletResponse resp)`
 - ❖ Wird von der service()-Methode aufgerufen, wenn ein HTTP-OPTIONS kommt.

HttpServlet

- ❖ `protected void doPost(HttpServletRequest req, HttpServletResponse resp)`
 - ❖ Wird von der `service()`-Methode aufgerufen, wenn ein HTTP-POST kommt.
- ❖ `Protected void doPut(HttpServletRequest req, HttpServletResponse resp)`
 - ❖ Wird von der `service()`-Methode aufgerufen, wenn ein HTTP-PUT kommt.
- ❖ `protected void doTrace(HttpServletRequest req, HttpServletResponse resp)`
 - ❖ Wird von der `service()`-Methode aufgerufen, wenn ein HTTP-TRACE kommt.

Lebenszyklus

- ❖ Servlet wird am Webserver angesprochen
 - ❖ Container bildet Exemplar und ruft nach Initialisierung service()-Methode auf
 - ❖ diese Methode delegiert an doXXX(), also doGet bei Get-Anfragen.

zwei Parameter(doXXX)

- ❖ HttpServletRequest: Repräsentiert die Anfrage. Es lassen sich zum Beispiel Parameter oder Header erfragen, die der Client zum Server schickt.
- ❖ HttpServletResponse: Repräsentiert das Ergebnis. Es lassen sich zum Beispiel Daten zurück zum Client schicken, genauso Antwort-Header (etwa Content-Type) setzen. Am wichtigsten ist die Methode getWriter(), die uns eine Referenz auf ein Writer-Objekt liefert, damit wir die HTML-Elemente für die Seite abschicken. Für Binärdaten können wir uns auch einen normalen OutputStream besorgen, damit wir zum Beispiel Bilder schicken können.

Servlets compilieren

- ❖ servlet.jar muss eingebunden sein
- ❖ Webapplikation besteht aus einem Verzeichnis WEB-INF
 - ❖ optionale Verzeichnisse classes und lib
 - ❖ übersetzte Klassen müssen in Paketstruktur unter classes

servlet mapping

- ❖ web.xml im WEB-INF-Verzeichnis der Webapplikation
 - ❖ Deployment-Deskriptor zählt die Servlets auf und weist Ihnen Pfade zu.

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_3_0.xsd">

    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>

    <servlet>
        <servlet-name>EasyServlet</servlet-name>
        <servlet-class>servlets.EasyServlet</servlet-class>
    </servlet>

    <servlet>
        <servlet-name>EasyServletContentType</servlet-name>
        <servlet-class>servlets.EasyServletContentType</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>EasyServletContentType</servlet-name>
        <url-pattern>/EasyServletContentType</url-pattern>
    </servlet-mapping>
</web-app>
```

Übung

- ❖ Uebung2.txt

Lebenszyklus eines Servlet

- ❖ Der Container für Servlets registriert eine Anfrage durch den Client und lädt das Servlet in den Speicher.
- ❖ Abarbeitung findet in einem Thread statt, der die Methoden des Servlet-Objekts aufruft
- ❖ drei elementare Methoden werden vorgeschrieben:

Lebenszyklus eines Servlet

- ❖ `void init(ServletConfig config)`
 - ❖ Wird zu Beginn eines Dienstes aufgerufen.
- ❖ `void service(ServletRequest req, ServletResponse res)`
 - ❖ Der Container leitet die Anfrage an das Servlet an diese Stelle.
- ❖ `void destroy()`
 - ❖ Wird am Ende eines Servlets vom Container genau einmal aufgerufen.

Parameter auslesen

```
public class ServletDemo1 extends HttpServlet{  
  
    public void doGet(HttpServletRequest request, HttpServletResponse response)  
throws IOException{  
    String name = request.getParameter("name");  
    PrintWriter out = response.getWriter();  
    out.println("<html>");  
    out.println("<body>");  
    out.println("<h1>Hello Servlet</h1>");  
    out.println("</body>");  
    out.println("<h1>" + name + "</h1>");  
    out.println("</html>");  
}  
}
```

Sessions

- ❖ Servlets gehören nicht automatisch zu einer Session
- ❖ mit getSession auf HttpServletRequest kann diese geholt bzw. erzeugt werden.
 - ❖ HttpSession getSession()
 - ❖ Liefert die aktuelle Session, die mit der Anfrage assoziiert ist. Wenn es keine Session gab, wird automatisch eine angelegt.
 - ❖ HttpSession getSession(boolean create)
 - ❖ Wie getSession(), nur dass getSession(false) nicht automatisch eine neue Session anlegt, wenn es keine mit der Sitzung assoziierte gibt.

Weiterleitung

- ❖ RequestDispatcher kann eine Weiterleitung von einem Servlet zum anderen veranlassen

Weiterleitung

```
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class MultipleInclude extends HttpServlet {

    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
        java.io.IOException {

        response.setContentType("text/html");
        java.io.PrintWriter out = response.getWriter();

        out.println("<html>");
        out.println("<head>");
        out.println("<title>Multiple Includes</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Das ist Level 1</h1>");
        out.println("Text von Level 1.");
        RequestDispatcher dispatcher = request.getRequestDispatcher("/Level4");
        dispatcher.include(request, response);
        out.println("</body>");
        out.println("</html>");
        out.close();

    }
}
```

Weiterleitung

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class Level4 extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, java.io.IOException {

        java.io.PrintWriter out = response.getWriter();
        out.println("<h4>Dies ist ein anderes Servlet| doGet</h4>");
    }
}
```

Übung

- ❖ Uebung3.txt

Statischer Template Code

- ❖ im allgemeinen aus Html-Code
- ❖ Datenstrom geht ungefiltert zum Client
 - ❖ Sonderzeichen richtig codieren!!!
 - ❖ nicht ASCII-Zeichen als Entriss schreiben
 - ❖ &XXX;

die erste eigene JSP

```
<%@ page import="java.lang.Math" %>
<html><body>
<h1>Heute programmier ich f&uuml;r den Server!!</h1>
<h2>Und ich erzeuge eine Zufallszahl!!</h2>
<%
    int zufallszahl = (int)(Math.random() * 1000);
%>
Die Zufallszahl lautet: <%=zufallszahl%>!!!!
</body></html>
```

Dynamische Inhalte

- ❖ JSP EL
 - ❖ beginnen mit einem Dollar-Zeichen und enthalten in geschweiften Klammern den Ausdruck für Rechenanweisungen, wie \${1+2}, oder Zugriffe auf Werte von Beans, etwa mit \${person.name}.

Dynamische Inhalte

- ❖ Tag-Libraries
 - ❖ Mit Tags ist eine Verarbeitung verbunden, um etwa mit <c:if> Teile der Webseite auszusparen oder einzusetzen.
- ❖ Scripting-Elemente
 - ❖ Java-Programmcode, der direkt in das aus der JSP generierte Servlet wandert ->
 - ❖ Beispiel <%=java.util.Date()%>

Dynamische Inhalte

- ❖ Direktiven
 - ❖ Direktiven steuern die Struktur der Seite, setzen den Content-Type und übernehmen zum Beispiel den Import einer Unterseite. Im Quellcode tauchen Direktiven über die Schreibweise <jsp:directive> auf.
- ❖ Aktionen
 - ❖ Aktionen interagieren zur Laufzeit mit dem Servlet-Container. So lassen sich Komponenten wie JavaBeans oder Webressourcen einbinden. Für jede JSP-Aktion gibt es einzelne Tags, etwa <jsp:include>, <jsp:forward>, <jsp:useBean>.

Formulardaten

- ❖ bei einer HTTP-Anfrage kann der Nutzer Daten mitgeben, z.B. Formulardaten
- ❖ Servlet-Container verarbeitet diese und stellt und stellt sie über das implizite EL-Objekt param einer jsp zur Verfügung
- ❖ Get(? an der Url) und Post(Datenstrom) transparent

HTML-Formulardaten

HTML-Tag	Beschreibung
<input type="text">	Eingabefeld
<input type="submit">	Submit-Button
<input type="password">	Eingabefeld für Passwörter
<input type="checkbox">	Checkbox
<input type="radio">	Radiobutton mit optionaler Gruppe
<textarea>	Mehrzeilige Eingabe
<button>	Schaltflächen
<select> mit <option>	Auswahllisten (Pulldown-Menü)

Beans

- ❖ <jsp:useBean> erwartet:
 - ❖ **id:** Gibt den Namen der Bean an. Der Name muss ein gültiger Java-Bezeichnername sein, denn die Aktion führt zu einer lokalen Variablen im Servlet, auf die ein Scriptlet zugreifen kann.
 - ❖ **class:** Identifiziert mit dem voll qualifizierten Klassennamen die JavaBean.

Beans

```
public class Details {
    public Details() {
    }
    private String username;
    private int age;
    private String password;
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

Beans

```
<html>
  <head>
    <title>
      useBean, getProperty and setProperty example
    </title>
  </head>
  <form action="useBeanDetails.jsp" method="post">
    User Name: <input type="text" name="username"><br>
    User Password: <input type="password" name="password"><br>
    User Age: <input type="text" name="age"><br>
    <input type="submit" value="register">
  </form>
</html>
```

Beans

```
<jsp:useBean id="userinfo" class="servlets.Details"></jsp:useBean>
<!--<jsp:setProperty property="*" name="userinfo"/>-->
<jsp:setProperty property="age" name="userinfo"/>
<jsp:setProperty property="username" name="userinfo"/>
<jsp:setProperty property="password" name="userinfo"/>
You have entered below details:<br>
<jsp:getProperty property="username" name="userinfo"/><br>
<jsp:getProperty property="password" name="userinfo"/><br>
<jsp:getProperty property="age" name="userinfo" /><br>
```

JSTL

- ❖ Kernaufgaben (Iterationen, Fallunterscheidungen)
- ❖ landestypische Formatierungen
- ❖ XML-Verarbeitung (Parsing, Transformationen)
- ❖ Datenbankanbindungen

JSTL

- ❖ müssen im Klassenpfad der Applikation liegen
 - ❖ global im Webcontainer
 - ❖ lokal in der Applikation
- ❖ zu beziehen unter:
 - ❖ <https://jstl.java.net/download.html#>

definierte TagLibs

TagLib	Übliches Präfix	Logischer Name/URI
Core	c	http://java.sun.com/jsp/jstl/core
Formatierung	fmt	http://java.sun.com/jsp/jstl/fmt
Funktionen	fn	http://java.sun.com/jsp/jstl/functions
SQL	sql	http://java.sun.com/jsp/jstl/sql
XML	x	http://java.sun.com/jsp/jstl/xml

Core Tags

- ❖ `<c:out>`: Ausgabe von Werten, mit und ohne Umkodierungen der HTML-Sonderzeichen »`<<`, »`&`«.
- ❖ `<c:remove>`: Löscht Variablen aus einem Gültigkeitsbereich.
- ❖ `<c:if>`, `<c:choose>`, `<c:when>`, `<c:otherwise>`: Realisieren Fallunterscheidungen.
- ❖ `<c:forEach>`, `<c:forTokens>`: Iterieren über Mengen oder Zeichenketten.
- ❖ `<c:catch>`: Fängt Ausnahmen auf.
- ❖ `<c:url>`, `<c:redirect>`, `<c:import>`: für URLs, Umleitungen und Einbettungen

<c:if>

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<jsp:useBean id="datum" class="java.util.Date" />
${datum.time}:
<c:if test="${datum.time mod 2 == 0}">
    Gerade Anzahl Millisekunden.
</c:if>
```

<c:forEach>

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<jsp:useBean id="format"
              class="java.text.SimpleDateFormat" />
<c:forEach var="w"
              items="${format.dateFormatSymbols.weekdays}" >
    ${w}
</c:forEach|
```

Scripting Elemente

- ❖ **JSP-Scriptlets** `<% .. %>`:
 - ❖ In die eingebetteten Java-Stücke lassen sich Anweisungen einsetzen, die in einer normalen Methode gültig sind.
- ❖ **JSP-Ausdrücke** `<%= .. %>`:
 - ❖ Das Ergebnis des Ausdrucks wird in die Seite eingebaut.
- ❖ **JSP-Deklarationen** `<%! .. %>`:
 - ❖ Diese Umgebung deklariert Variablen, Methoden und innere Klassen.

Beispiele

JSP Scriptlets

JSP
Deklarationen

```
<%@ page import="java.lang.Math" %>
<html><body>
<h1>Heute programmier ich f&uuml;r den Server!!</h1>
<h2>Und ich erzeuge eine Zufallszahl!!</h2>
<%
    int zufallszahl = (int)(Math.random() * 1000);
%>
<%!
    public int quadrat( int x ) {
        return x * x;
    }
%>
Die Zufallszahl lautet: <%=zufallszahl%>!!!!
Die Quadratzahl: <%=quadrat(9)%>
</body></html>
```

JSP Ausdrücke

Implizite Objekte

Implizites Objekt	Benutzt, um ...	Typ
request	Anfragen zu verarbeiten und Eingabewerte wie Parameter zu lesen.	HttpServletRequest
response	etwas an den Client zu übermitteln, wie beispielsweise Header.	HttpServletResponse
out	in den Ausgabestrom zu schreiben.	JspWriter
application	Daten aller Anwendungen zu speichern.	ServletContext
session	Sitzungsinformationen zu speichern.	HttpSession
pageContext	Kontextdaten für eine Seite zu speichern.	PageContext
Page	ein Exemplar des Servlets anzusprechen.	Object (this)



Anfragen sind erst einmal...

- ❖ **zustandslos aber:**

Sitzungsverfolgung

- ❖ Cookies
- ❖ URL-Rewriting
- ❖ versteckte Felder

Sitzungen in JSPs

- ❖ bei erster Verbindung des Browsers zum Server, erstellt der Container eine Sitzung und sendet standardmäßig ein Cookie zurück.
- ❖ Austausch dieses Cookies

Cookie

- ❖ enthält eine ID, die mit einem Assoziativspeicher verbunden ist
- ❖ Assoziativspeicher können Schlüssel / Wert Paare haben.

Assoziativspeicher

Setzen des Attributes user auf die Session

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<% session.setAttribute( "user", "Carsten" ); %>
Der User in der Session was: ${sessionScope.user}.<br>
<c:set var="url" value="www.lufthansa.de" scope="session" />
Die URL lautet ${sessionScope.url}
```

Alternativ über die JSTL