

## Übung Netzbasierte Informationssysteme

### Termin 2: Web-basierte Informationssysteme

**Prof. Dr. Adrian Paschke**

Arbeitsgruppe Corporate Semantic Web (AG-CSW)  
Institut für Informatik, Freie Universität Berlin  
[paschke@inf.fu-berlin.de](mailto:paschke@inf.fu-berlin.de)  
<http://www.inf.fu-berlin.de/groups/ag-csw/>



# Agenda

---

- Feedback Übungsblatt 1
- Offizielle Anmeldung zur Vorlesung und Übung
- Themen des heutigen Übungsblatts:
  - Java Servlets - extended
  - Java Server Pages (JSPs)
- Übungsblatt 2

# Feedback Übungsblatt 1

---

- **Ziele des 1. Übungsblattes waren:**
  - Kennenlernen der Entwicklungsumgebung
  - Bestandteile eines Servlets
  - Deployen eines Servlets
  - Entwickeln eigener Servlets mit und ohne statische Seite
  - Anwendung eines ANT-Scripts
  
- **Wie war`s?**
  - Gut mit Umgebung zurechtgefunden?
  - Basics von Java Servlets verstanden?
  - Deployment?
  
- **Probleme/Anregungen?**

# Anmeldung zur Vorlesung und Übung

---

- **Explizite Anmeldung zu Prüfungsleistungen**
  - mit Anmeldung muss Prüfung abgelegt werden
  - Nichtteilnahme gilt dann als nicht bestandener Prüfungsversuch
- Jetzt zusätzliche Anmeldung mittels Unterschrift
- **Bitte überprüfen Sie auch die Richtigkeit Ihre Daten**
  - Falsches bitte durchstreichen und leserlich daneben/darüber verbessern
  - Bei Fragen bitte nach diesem Termin zu mir kommen

# Java Servlets - extended

---

- Java Servlets - extended
  - Parameter mittels web.xml setzen
  - Ergebnisweiterleitung
  - Requestweiterleitung incl. Datenübergabe
- Parametersetzung
  - Lebenszyklus eines Servlets init(), ... , destroy()
  - Während Initialisierung können z.B. Datenbankverbindungen aufgebaut werden, ...
  - Parameter dafür können entweder direkt in der init() Methode hard-encoded werden oder ausgelagert werden
  - Eine Art der ausgelagerten Parametersetzung: im Deployment Descriptor web.xml

# Parametersetzung mittels web.xml

```
<web-app>
  <servlet>
    <servlet-name>ServletName</servlet-name>
    <servletclass>mypackage.servletClassName</servetclass>
    <init-param>
      <param-name>DBUser</param-name>
      <param-value>APaschke</param-value>
    </init-param>
    <init-param>
      <param-name>DBPwd</param-name>
      <param-value>FooBar</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>ServletName</servlet-name>
    <url-pattern>/myServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

web.xml

# Parametersetzung mittels web.xml

```
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
...

private String username;
private String userPwd;

public class exampleServlet extends HttpServlet{
    ...

    public void init() throws ServletException {
        userName = getInitParameter("DBUser");
        userPwd = getInitParameter("DBPwd");
    }
    ...
}
```

**servlet**

# Ergebnisweiterleitung auf eine andere Seite

```
...
public class exampleServlet extends HttpServlet{

    ...

    public void doPost() {HttpServletRequest request, ...
        ...
        //redirect client to the error page
        if(...){
            response.sendRedirect("/demo/error.html");
            return;
        }
    }
}
```

**servlet**

- Idealerweise sollte wegen Übersichtlichkeit in der doGet()/doPost()-Methode nur der Kontrollfluss gesteuert werden (Methodenaufrufe)
- Berechnungen und Generierung der html-response sollten in Helper-Methoden ausgegliedert werden

# Requests weiterleiten und Daten übergeben

---

- Innerhalb eines `HttpServletRequest` können Attribute gesetzt und abgerufen werden
  - `Object getAttribute (java.lang.String name)`
  - `void setAttribute (java.lang.String name, java.lang.Object o)`
- Da der Parameter vom Typ `Object` ist, funktioniert das nicht nur mit Strings sondern auch mit selbst erstellten Beanklassen
- Redirect auf anderes Servlet oder JSP erfolgt über ein Hilfsobjekt vom Typ `RequestDispatcher`
- Bei der Weiterleitung werden `request` und `response` übergeben, damit auch die gesetzten Attribute

# Requests weiterleiten und Daten übergeben

---

```
...
public void doPost() { HttpServletRequest request, ...
    ...
    MyBean bean = new MyBean();

    // bean füllen...

    // bean in den Kontext setzen
    request.setAttribute(„myBean“, bean)

    // Get helper object that can forward the request (and bean) to next servlet or JSP
    // (defined by a String – „name“)

    RequestDispatcher dispatch = request.getRequestDispatcher(name);

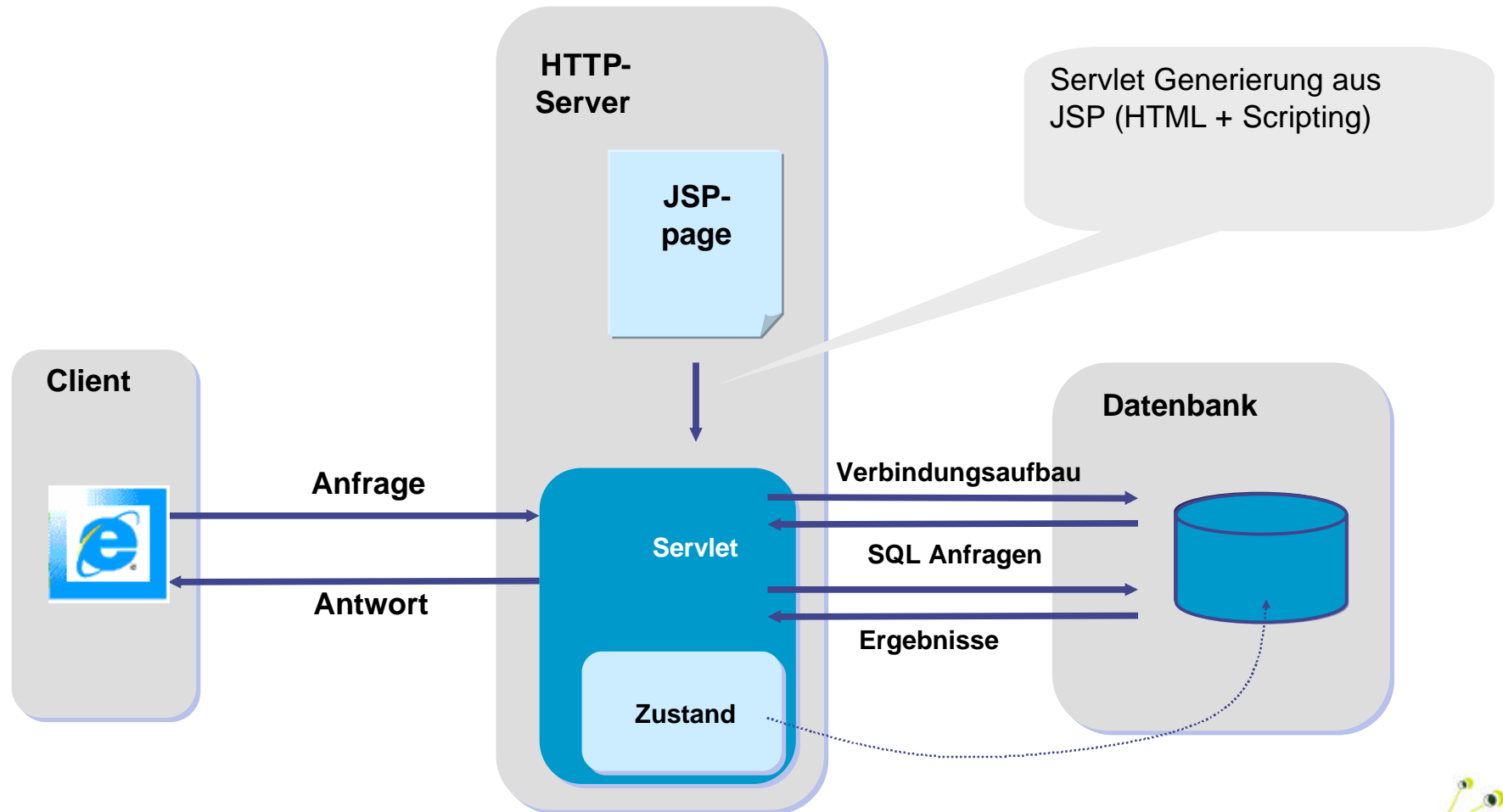
    //forward request
    dispatch.forward(request, response);
}
```

# Java Server Pages (JSPs)

---

- JSPs haben eine andere **Syntax** als Servlets:
  - Servlets produzieren HTML-Code innerhalb von Java-Code
  - JSPs betten Java-Code in HTML-Code ein
- JSPs werden vom Web Container intern in Servlets umgewandelt
- **Vorteile** von JSPs:
  - “Separation of responsibilities” – Logikprogrammierung kann in Java von Programmierern übernommen werden; Design der pages kann von Web Designern übernommen werden
  - Übersichtlichkeit
  - Komponentenartigkeit
- **Struktur** ist normales HTML mit zusätzlichen Basistags:
  - Scriptlets, Expressions, Declarations, Directives

# JSP Ablauf



# Scriptlets

---

- **Scriptlet Tag**
  - `<% code %>`
  - Bettet Java-Code in das JSP-Dokument ein
  - Dieser Code wird bei jedem neuen Aufruf des JSPs ausgeführt

```
<html>
  <body>
    <% for (int i = 0; i < 2; i++) { %>
      <p>Hello World!</p>
    <% } %>
  </body>
</html>
```

```
<html>
  <body>
    <p>Hello World!</p>
    <p>Hello World!</p>
  </body>
</html>
```

# Expressions

---

- **Expression Tag**

- `<%= expression %>`
- Bettet einen Java-Ausdruck ein, der jedes Mal neu ausgewertet wird wenn das JSP geladen wird
- Achtung: kein “;” nach dem Java-Ausdruck!
- Sonst Syntax-Fehler

```
<html>
  <body>
    <p><%= Integer.toString( 5 * 5 ) %></p>
  </body>
</html>
```



```
<html>
  <body>
    <p>25</p>
  </body>
</html>
```

# Declaration

---

- **Declaration Tag**
  - `<%! declaration %>`
  - Bettet Java-Deklarationen in JSP ein
  - z.B. Methoden-Deklaration:

```
<html>
  <body>
    <%! public boolean isPositive(int x)
      {
        if (x < 0) {
          return false;
        }
        else {
          return true;
        }
      } %>

  </body>
</html>
```

# Directives

---

- **Directive Tag**

- `<%@ directive %>`
- Directives werden benutzt um spezielle Ausführungshinweise zu der Seite an den JSP-Container zu übertragen

**web.xml :**

```
<taglib>
  <taglib-uri>/taglib/struts-logic</taglib-uri>
  <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
</taglib>
```

**beispiel.jsp:**

```
<%@ taglib uri="/taglib/struts-logic" prefix="logic" %>
```

tld: sog. Tag Library Definition

# URL-encoded Parameter

---

- JSP stellt einen sog. object request zur Verfügung
- dieser speichert Attribute speichert und übergibt sie an den request der JSP-Seite
- Parameter können URL-encoded gesetzt werden:

Beispielaufruf:

`http://localhost/example.jsp?param1=hello&param2=world`

**example.jsp:**

```
<html>
  <body>
    <p><%= request.getParameter("param1") %></p>
    <p><%= request.getParameter("param2") %></p>
  </body>
</html>
```



```
<html>
  <body>
    <p>hello</p>
    <p>world</p>
  </body>
</html>
```

# JSP Action Tags

---

- **JSP Action Tags**

- `<jsp:property />`

- *property* wird in sog. Tag Library Definition files definiert

- JSP Tag reference:

- <http://java.sun.com/products/jsp/syntax/1.2/syntaxref12.ht>

```
<html>
  <body>
    <jsp:useBean id="user" class="mypackage.User" />
    <jsp:setProperty name="user" property="userName" value="MBichler" />
    <p>User Name:&nbsp;  <jsp:getProperty name="user" property="userName" /></p>
  </body>
</html>
```

# JSP und Sessions

---

- **Problem:** HTTP ist zustandslos - jeder neue Request beginnt mit leeren Zustandsraum
- **Lösung** für JSPs: es gibt das **Konzept der Session.**
  - In Sessions lassen sich Attribute speichern und lesen.
  - Session Attribute werden persistent indem sie Cookies nutzen oder request parameter übergeben
  - Das Sessionobject implementiert Methoden des Interfaces `javax.servlet.http.HttpSession`.

Method	Description
<code>getId()</code>	Returns the session ID.
<code>getCreationTime()</code>	Returns the time at which the session was created
<code>isNew()</code>	Returns <code>true</code> if user's browser has not yet confirmed the session id.
<code>invalidate()</code>	Discards the session, releasing any objects stored as attributes.
<code>getAttribute(String key)</code>	Returns an attribute that is stored in the session.
<code>setAttribute(String key, String value)</code>	Stores an attribute in the sessions.

# JSP und Sessions

---

- Attribute können in den Kontext mit folgenden Lebensdauern gesetzt werden:
  - für den HttpRequest - scope = "request"
  - für die Session – scope = "session"
  - für den ServletContext – scope = "application"

```
<jsp:useBean scope="request" id="userInfo" class="mypackage.UserBean" />  
  <jsp:setProperty name="userInfo" property="*" />  
</jsp:useBean>
```

- setProperty kann dazu benützt werden, ein spezielles property auf einen bestimmten Wert zu setzen
- property = "\*" transferiert alle Daten eines Formulars in das Bean
- dabei wichtig: Benennungen gleich, alle set-Parameter Strings, Beankonformität einhalten!

# JSPs - Deployment

---

## Verzeichnisstruktur einer Web-Anwendung laut Spezifikation:

- `nameDerAnwendung/`
  - enthält ergänzende Dateien (z.B. HTML, CSS, GIF, **JSP Dateien**), welche sich auch in Unterverzeichnissen befinden können
- `nameDerAnwendung/WEB-INF/`
  - enthält den Deployment Descriptor `web.xml`
- `nameDerAnwendung/WEB-INF/classes/`
  - enthält die `.class` - Dateien der Servlets und aller weiteren benötigten Java-Klassen.
  - Struktur der Unterverzeichnisse innerhalb des `/classes` - Verzeichnisses muss mit der Paketstruktur der Implementierung korrespondieren
- `nameDerAnwendung/WEB-INF/lib/`
  - enthält die benötigten `.jar` - Archive

# JSPs - Deployment

---

**Damit muss man nur seine JSP-Seite in den jeweiligen Kontext von /webapps kopieren**

- Bsp: /webapps/nameDerAnwendung/example.JSP
- Beim Ausführen wird die JSP-Seite in ein Servlet transformiert
- Dieses liegt dann im tomcat/work context
- Falls im JSP andere Java-Klassen verwendet werden, müssen diese (wie auch bei den Servlets) in /nameDerAnwendung/WEB-INF/classes/ kopiert werden
  - Dabei ist wieder auf eventuelle Package-Strukturen zu achten!
  - Im JSP muss dann explizit die benutzte Klasse am Anfang der JSP importiert werden:
    - `<%@ page import= "mypackage.MyClass" %>`

# Hypertext Markup Language

---

- HTML
  - Einfache Auszeichnungssprache
  - „Lingua Franca“ des Webs
  - Besteht aus Inhalten (Text) und Markup (Tags)
  - Web Browser können HTML Seiten interpretieren und anzeigen

- HTML Dokumenten

- Grobstruktur:

- <HTML><HEAD></HEAD><BODY></BODY></HTML>

- Header möglich (mit Versionierungsinformation)

- <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

- In HEAD findet man TITLE (auch META)

- In BODY findet man Inhalt mit Tags:

- Headings <h1>, <h2>, <h3>, <h4>, <h5>, <h6>

- Paragraphs <p> und Line Breaks <br>

- Textformattierung <b>, <u>, <i>, <big>, <small>, <em>

- Anchors (Hyperlinks) <a href=„...“> </a>

- Bilder (Images) hinzufügen <img src=„...“/>

- HTML Tabellen verbessern den Layout

- Beachten:

- Character Entities, z.B. ü -> &uuml;

- HTML Referenzen online

- z.B. <http://www.html-reference.com/>

# Webseiten publizieren

- Ihr Heimbereich
  - Unix: ~/web-home/
  - Windows: \\web\page.mi.fu-berlin.de\web-home\
    - Alle HTML, Bilder usw. gehen in Ordner public\_html
- Sie können anderen Nutzer Ihre Website editieren lassen
  - In Linux:
  - legen Sie in ihrem web-home/ eine leere Datei namens .multiuser an.
  - setzen Sie mittels *Access Control Lists* (ACLs) wie folgt die Rechte:

```
setfacl -R -d -m u:<weiterer username>:rwx ~/web-home/
```

# Bitte beachten!

---

- Es gelten die Nutzungsbedingungen aus dem Nutzerantrag des Rechnerbetriebs
- Weiteres auf

<https://www.mi.fu-berlin.de/wiki/bin/view/Tec/UserPages>

# Web Good Practices

---

- Was sind Web Good Practices?
  - Anweisungen die den Wert des Webs erhöhen
- Wo finde ich sie?
  - „Architecture of the World Wide Web, Volume One“
  - <http://www.w3.org/TR/webarch/>
- Welche Aspekte des Webs umfassen sie?
  - Identifikation (URIs usw.)
  - Interaktion (HTTP usw.)
  - Repräsentation (HTML usw.)

# Literatur

---

- Folien, Web
- Servlet und JSP Online Tutorial:
  - <http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html>
  - <http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/>
  - <http://developer.java.sun.com/developer/onlineTraining/JSPIntro/contents.html>
  - <http://www.galileocomputing.de/openbook/javainse15/>, Chapter 17
- Servlet und JSP Buch:
  - Hall, M.: Core Servlets and Java Server Pages; Sun Microsystems Press/Prentice Hall PTR, 2000.
- Übersichtsartikel:
  - Turau, V.: Techniken zur Realisierung Web-basierter Anwendungen, Informatik Spektrum, 22 (1999) 1, S. 3-12.
- „Architecture of the World Wide Web, Volume One“
  - <http://www.w3.org/TR/webarch/>



**Viel Spaß und Erfolg !**

