

# Java Servlets und Java Server Pages



## Java Servlets und Java Server Pages

### Einführung

Servlets

Lebenszyklus

Sessions

JSP - JavaServer Pages

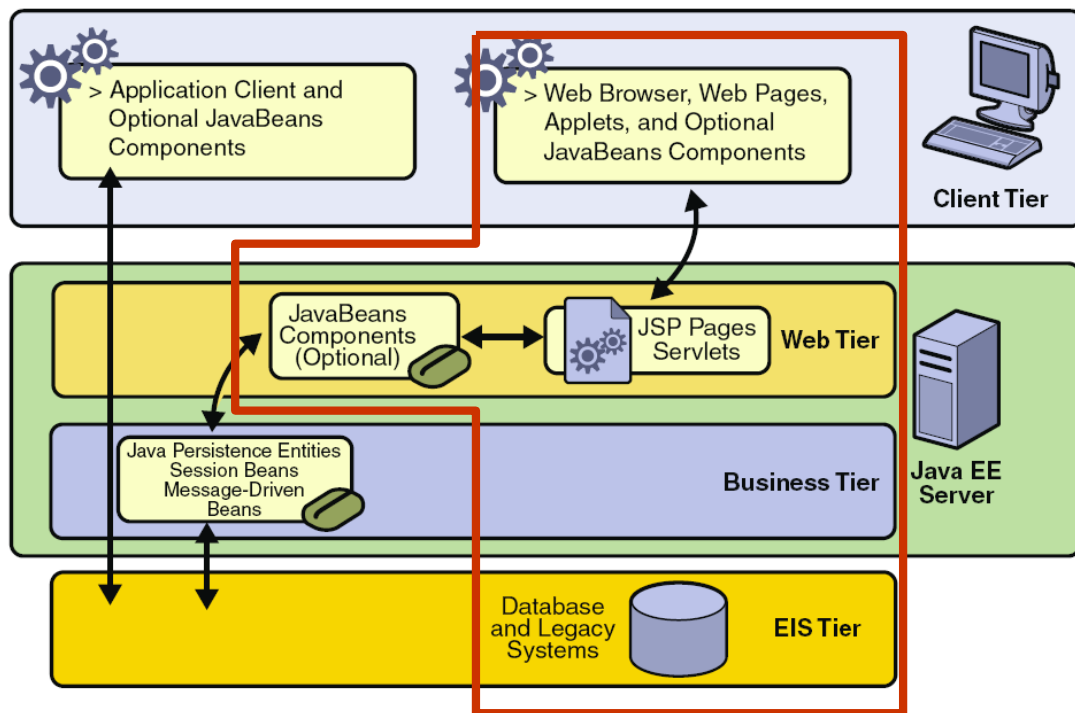
Trennung Logik und Darstellung

Expression Language und Tag Libraries

Zusammenfassung



# Multi-Tier Architecture



## Definition

### SERVLET

=

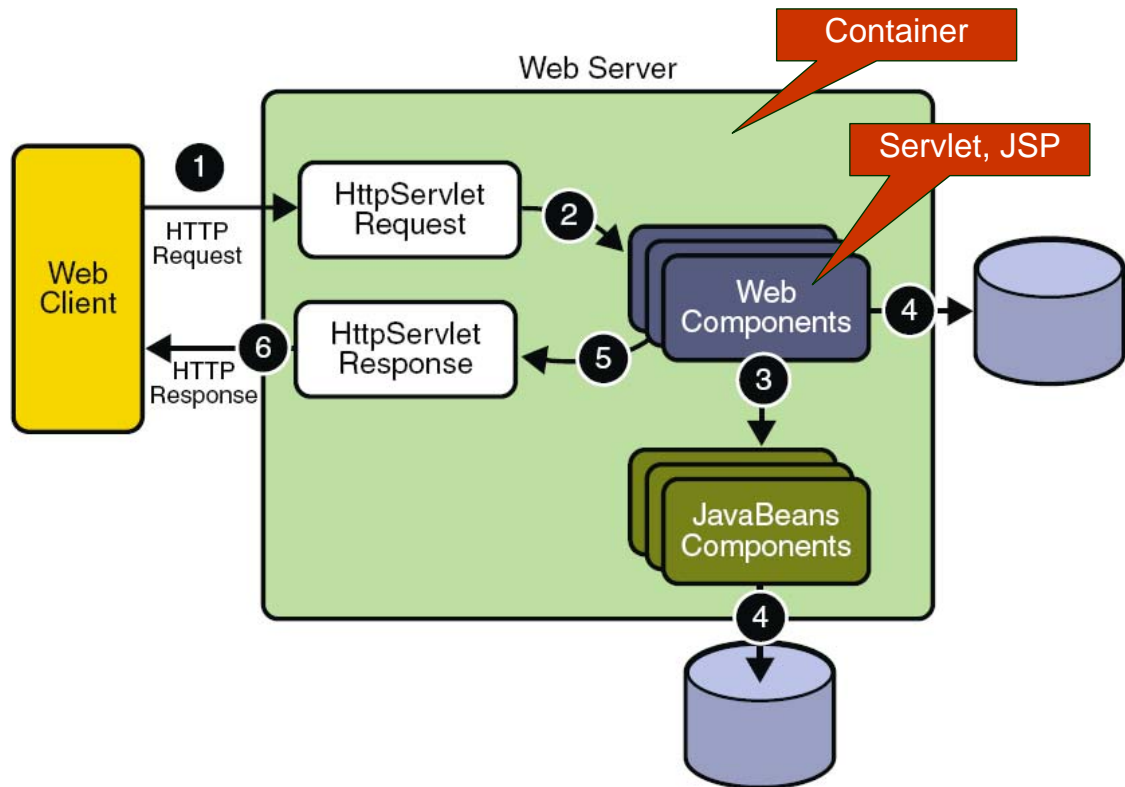
Eine auf Java-Technologie basierte Web-Komponente, die von einem Container verwaltet wird und dynamisch Inhalt generiert.

### SERVLET-ENGINE (Container)

=

Teil eines Web-Servers, der die Netzwerkdienste zum Empfangen von Anfragen und Senden von Antworten bereitstellt und die Servlets über ihren gesamten Lebenszyklus enthält und verwaltet.





## Web-Container und Web-Components

### Web-Container

- Erzeugen und Starten von Servlets
- Lesen und Setzen von Konfigurationsparameter
- Zuordnen von Requests zu Servlets
- Thread-Management
- Zurückschicken der Antwort zum Client
- Verwalten von Objekten (JavaBeans)
- Session Management
- Erkennen von Ereignissen und Aufruf von Listener
- Error-Handling
- Konfiguration von Filterketten
- ...

Wesentlich sind Konfigurationsdaten in `web.xml`

### Web-Components

- Bearbeitung der Requests
- Zugriff auf und Aufbereitung der Daten
- Erzeugen der Webseiten für die Antwort
- Inkludieren und Weiterleiten an andere Seiten
- ...

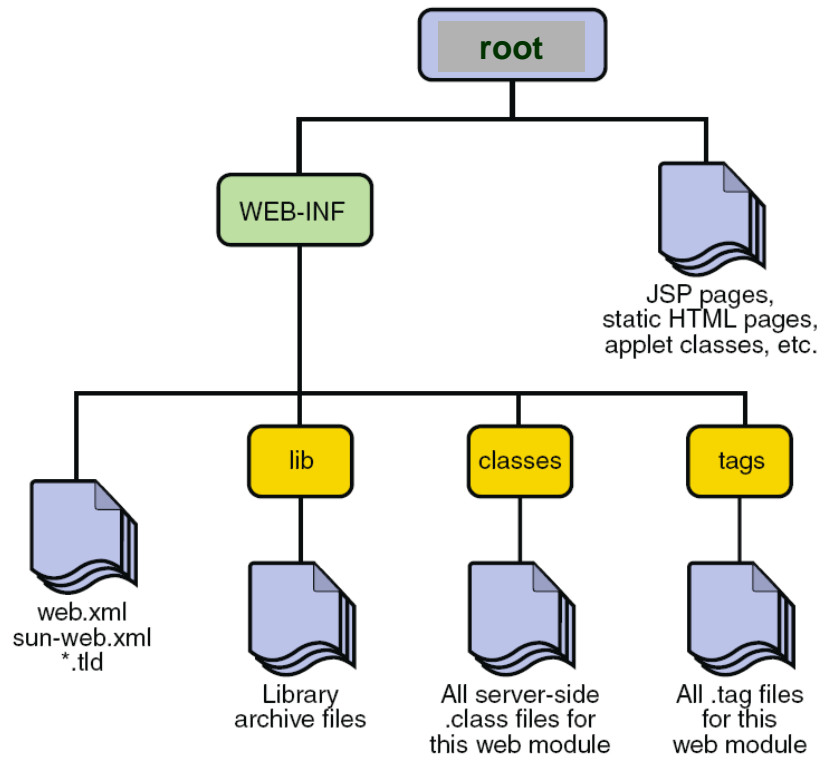
#### Servlets

- Programmierung in Java

#### JSPs

- Kodierung in Template-Files mit `html + Scriptlets + Tags`
- Übersetzung in Servlets





## web.xml

### Konfiguration der Web-Applikation in web.xml

- Initialisierungsparameter
- Servlets
- Listeners
- Diverse Einstellungen

Wesentliche Information für Container

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <!-- Allgemeine Beschreibung -->
  <display-name>Name der Anwendung</display-name>
  <description>Beschreibung der Anwendung</description>
  <context-param>
    <!-- Kontext Parameter für Initialisierung -->
  </context-param>
  <servlet>
    <!-- Konfiguration des Servlets -->
  </servlet>
  <servlet-mapping>
    <!-- Zuordnung der Servlet-Implementierung zu URL -->
  </servlet-mapping>
  ...
</web-app>
```



## HTTP methods

- GET
- HEAD
- POST
- PUT
- DELETE
- TRACE
- CONNECT
- OPTIONS

```
GET /MyServlet.do?name=Max%20Muster&company=MuParamter als Name/Wert-Paare in URL
Host: 140.78.145.103:2020
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

```
POST /MyServlet HTTP/1.1
Host: 140.78.145.103:2020
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 29
name=Max+Muster&company=MuParamter in eigenem Datenbereich
```



# HTML-Formulare

Ermöglichen Benutzereingaben  
Benutzen POST oder GET

```
<html><head><title>HTML Forms</title></head>
<body>
<form action="result.html" method="get">
  <input type="hidden" name="id" value="123"/>
  User Name: <input type="text" name="user"/><br>
  User Password: <input type="password" name="pass"/><br>
  Flash:
  <input type="checkbox" name="flash" checked>Enabled</input><br>
  Age:
  <input type="radio" name="age" value="child">&lt; 18</input>
  <input type="radio" name="age" value="adult" checked>&ge; 18</input><br>
  Role:
  <select name="role">
    <option value="A">Administrator</option>
    <option value="S" selected>Student</option>
  </select><br>
  <input type="submit" value="Submit"/>
</form>
</body></html>
```

User Name:

User Password:

Flash:  Enabled

Age:  < 18  ≥ 18

Role:

HTTP GET Aufruf bei Submit:

```
GET result.html?id=123&user=John+Doe&pass=myspass&flash=on&age=adult&role=S
```



# HTML-Formulare Fortsetzung

`id=123&user=John+Doe&pass=myspass&flash=on&age=adult&role=S`

## Textfelder:

**text**: Text-Eingabefeld

**hidden**: Unsichtbares Textfeld, Information für den Server (z.B.: AppointmentId)

**password**: Text-Eingabefeld bei dem die Eingabe versteckt wird

Sonderzeichen werden codiert übertragen (%HexHex; z.B.: ä=%E4),

Leerzeichen als "+" (z.B.: John+Dow).

## Auswahlfelder:

**checkbox**: Wert wird nur übertragen wenn er gesetzt ist (als name=on, z.B.: flash=on), mit **checked** kann eine Checkbox vorselektiert werden.

**radio**: Wert wird nur übertragen wenn er gesetzt ist (value wenn angegeben sonst "on"), es kann nur ein Wert aus einer Gruppe (gleicher Name) ausgesucht werden, mit **checked** kann eine Vorselektion erfolgen.

**select**: Dropdown-Element, als Wert wird der ausgewählte (value der Option wenn angegeben sonst der Text) übertragen wobei mit **selected** eine Vorauswahl getroffen werden kann (sonst erstes Element).

## Absenden:

**submit**: Erzeugt einen Button, Text kann mit **value** gesetzt werden, wird auch ein **name** gegeben so wird das Name/Wert-Paar mitübertragen (sinnvoll wenn mehrere Buttons in eine Formular).



# Java Servlets und Java Server Pages

Einführung

**Servlets**

Lebenszyklus

Sessions

JSP - JavaServer Pages

Trennung Logik und Darstellung

Expression Language und Tag Libraries

Zusammenfassung



## Generische Klassen: javax.servlet

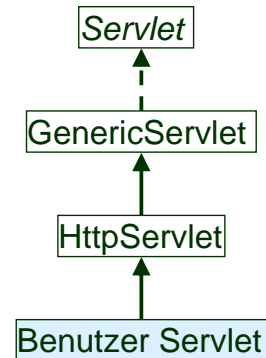
*Servlet* ←----- *GenericServlet**ServletRequest**ServletResponse**ServletContext*

...

## Spezialisierungen für HTTP: javax.servlet.http

*HttpServlet**HttpServletRequest**HttpServletResponse*

...



## Servlet Interface

**void init(ServletConfig config)**  
**throws ServletException**

Initialisierung des Servlets mit Konfigurationsparameter

**ServletConfig getServletConfig()**

Zugriff auf die Konfigurationsparameter

**void service(ServletRequest req, ServletResponse res)**  
**throws ServletException, IOException**

Führt einen einzelnen Request von einem Client aus und erzeugt die Response.

Für jeden Request wird die Methode in eigenem Thread ausgeführt

Parameter **req** hält alle Informationen über Request;

in Parameter **res** wird die Antwort geschrieben

**String getServletInfo()**

Information über Servlet in Form eines Strings

**void destroy()**

soll Aufräumarbeiten durchführen und zerstört das Servlet



# ServletRequest

`String getParameter(String name)`

Wert (URL-decoded) des ersten Parameters mit dem gegebenen Namen  
null, falls nicht vorhanden

`String[] getParameterValues(String name)`

Array mit Werten (URL-decoded) für jedes Vorkommen des angegebenen  
Parameternamens  
null, falls nicht vorhanden

`Enumeration request.getParameterNames()`

eine Aufzählung (*Enumeration*) aller Parameternamen  
falls keine Parameter übergeben wurden, ist die Enumeration leer

`String getRemoteAddress()`

IP-Adresse des Clients

`boolean isSecure()`

true, wenn der Aufruf über HTTPS erfolgt ist

`void setAttribute(String name, Object value)`

Setzen eines beim Request gespeicherten Attributs mit Namen name

`Object getAttribute(String name)`

Lesen eines beim Request gespeicherten Attributwertes

...

Requests erlauben beliebige Attribute in  
der Form von Name-Value-Paaren



# ServletResponse

`ServletOutputStream getOutputStream()`

Binäre Daten

`PrintWriter getWriter()`

Text

`void reset()`

Löschen des Puffers, der Header und des Statuscodes

`void flushBuffer()`

Schreiben des Puffers (den Header und Statuscode) an den Client

`String getContentType()`

Liefert den aktuell gesetzten MIME Typen (z.B.: text/html, image/jpeg, ...)

`void setContentType(String type)`

Setzt den MIME Typen (z.B.: text/html, image/jpeg, ...)

...

} exklusiv für diese  
Antwort



# HttpServlet

protected Methoden:

```
void doDelete(HttpServletRequest, HttpServletResponse)
void doGet(HttpServletRequest, HttpServletResponse)
void doHead(HttpServletRequest, HttpServletResponse)
void doOptions(HttpServletRequest, HttpServletResponse)
void doPost(HttpServletRequest, HttpServletResponse)
void doPut(HttpServletRequest, HttpServletResponse)
void doTrace(HttpServletRequest, HttpServletResponse)
void service(HttpServletRequest, HttpServletResponse)
```

Leitet HTTP-Anfragen auf die entsprechenden Methoden um.

public Methoden:

```
void service(ServletRequest, ServletResponse)
    Ruft die protected service-Methode auf
```

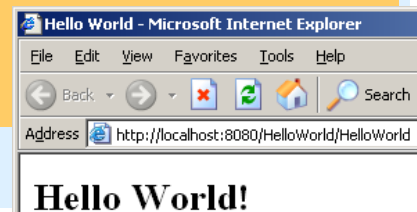


# Beispiel

```
package at.jku.psw2.web;

import java.io.*;
import javax.servlet.ServletException;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {
    public void doGet (HttpServletRequest req,
                      HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println(
            "<html><head><title>Hello World</title></head><body>"
            + "<h2>Hello World!</h2>"
            + "</body></html>"
        );
        out.close();
    }
}
```



## Installieren eines Servlets (web.xml 1/2)

```
<web-app>
...
<!-- Beschreibung der Servlets dieser Web Anwendung.
Jedem Servlet können Parameter mitgegeben werden die über
String value = getServletConfig().getInitParameter(name);
abgefragt werden können.
-->
<ervlet>
  <ervlet-name>Interner Name des Servlets</ervlet-name>
  <description>Beschreibung des Servlets</description>
  <ervlet-class>Voll qualifizierter Klassenname</ervlet-class>
  <init-param>
    <param-name>Name des Parameters</param-name>
    <param-value>Wert des Parameters</param-value>
  </init-param>
  <!--Servlet automatisch mit Tomcat starten!
Zahl gibt die Reihenfolge an, in der die Servlets gestartet werden.
Wenn keine spezielle eihenfolge benötigt wird genügt auch
"<load-on-startup/>" -->
  <load-on-startup>Zahl</load-on-startup>
</ervlet>
```



## Installieren eines Servlets (web.xml 2/2)

```
<!-- Festlegen unter welchem Namen das Servlet erreichbar sein soll. -->
<ervlet-mapping>
  <ervlet-name>Interner Name des Servlets</ervlet-name>
  <url-pattern>Offizieller Name des Servlets</url-pattern>
</ervlet-mapping>
...
</web-app>
```

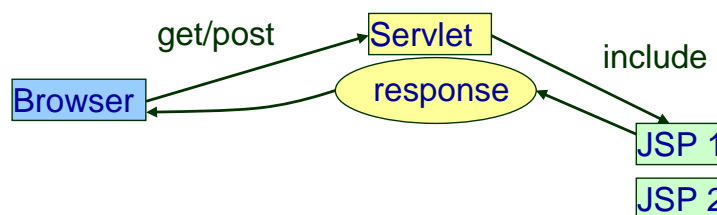


## Beispiel: Installieren eines Servlets (web.xml)

```
<web-app>
  <display-name>PSW2</display-name>
  <description>Lecture PSW2, Institute SSW</description>
  <servlet>
    <description>Simplehello world example</description>
    <display-name>HelloWorld</display-name>
    <servlet-name>HelloWorld</servlet-name>
    <servlet-class>at.jku.ssw.psw2.web.HelloWorld</servlet-class>
    <init-param>
      <description>Default name</description>
      <param-name>name</param-name>
      <param-value>Herbert</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>HelloWorld</servlet-name>
    <url-pattern>/hello</url-pattern>
  </servlet-mapping>
  ...
</web-app>
```



## Weiterleiten an und Inkludieren von Seiten



Anforderung eines `javax.servlet.RequestDispatcher`  
aus `<ServletContext>` oder `<ServletRequet>`

```
<ServletContext>.getRequestDispatcher(String absolutPath)
<ServletContext>.getNamedDispatcher(String name)
<ServletRequet>.getRequestDispatcher(String path)
```

Seite anfordern

```
void include(ServletRequest req, ServletResponse res)
```

Inhalt aus Seite oder Ressource einfügen und Kontrolle behalten

```
void forward(ServletRequest req, ServletResponse res)
```

Kontrolle an Servlet oder JSP weitergeben



## Weiterleiten von Request

Oft notwendig, einen Request an eine andere Seite weiterzuleiten

- Zugriff auf **RequestDispatcher** für Seite über Request
- Aufruf von **forward** zum Weiterleiten

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    ...
    try {
        request.getRequestDispatcher("/hello.jsp").forward(request, response);
    } catch (Exception exc) {
        log(exc.getMessage());
    }
}
```



## Inkludieren von Seiten und Ressourcen

Inkludieren von Inhalt (Ressourcen oder Output von anderen Servlets)

- Zugriff auf **RequestDispatcher** für Seite über Request
- Aufruf von **include** zum Inkludieren des Inhalts

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    ...
    try {
        request.getRequestDispatcher("part.jsp").include(request, response);
    } catch (Exception exc) {
        log(exc.getMessage());
    }
}
```



Einführung

Servlets

Lebenszyklus

Sessions

JSP - JavaServer Pages

Trennung Logik und Darstellung

Expression Language und Tag Libraries

Zusammenfassung



## Lebenszyklus

Engine lädt Servlet-Klasse und erzeugt **ein** Exemplar

alle Variablen werden genau einmal initialisiert

alle Variablen bleiben aktiv, solange das Servlet-Exemplar existiert

Mehrere Web-Browser (=Clients) fordern das Servlet an

Request wird vom Web-Server an Web-Container und von diesem weiter zum entsprechenden Servlet gesandt

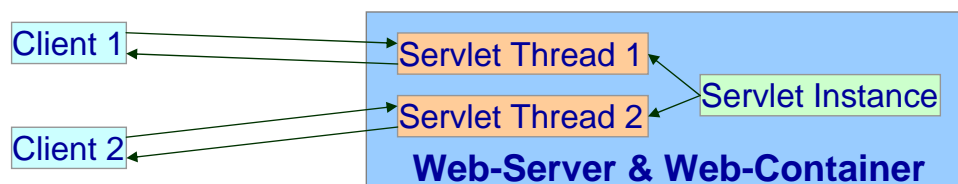
Engine erzeugt pro Request einen Thread

jeder Thread behandelt Request+Response des jeweiligen Clients

jeder Thread hat Zugriff auf die Instanzvariablen des Servlets

Web-Container bestimmt, wann Servlets entladen werden

Servlet aufräumen (z.B. Zustand persistent speichern, Ressourcen freigeben)



# Lebenszyklus

`init(ServletConfig)`

Variablen initialisieren, Ressourcen anfordern

`service(HttpServletRequest, HttpServletResponse)`

`doGet(HttpServletRequest, HttpServletResponse)`

`doPost(HttpServletRequest, HttpServletResponse)`

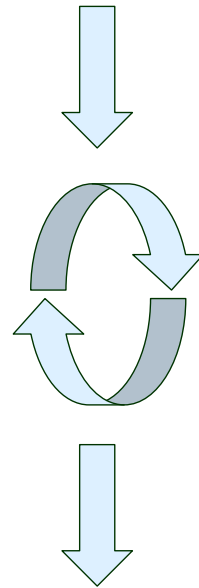
`doPut(HttpServletRequest, HttpServletResponse)`

...

`destroy()`

Zustand speichern

Ressourcen freigeben



# Listeners

Web-Container signalisiert Ereignisse des Servlet-Lebenszyklus  
Listener um auf Ereignisse zu reagieren

Object	Event	Listener Interface and Event Class
Web context	Initialization and destruction	<code>javax.servlet.ServletContextListener</code> and <code>ServletContextEvent</code>
	Attribute added, removed, or replaced	<code>javax.servlet.ServletContextAttributeListener</code> and <code>ServletContextAttributeEvent</code>
Session	Creation, invalidation, activation, passivation, and timeout	<code>javax.servlet.http.HttpSessionListener</code> , <code>javax.servlet.http.HttpSessionActivationListener</code> , and <code>HttpSessionEvent</code>
	Attribute added, removed, or replaced	<code>javax.servlet.http.HttpSessionAttributeListener</code> and <code>HttpSessionBindingEvent</code>
Request	A servlet request has started being processed by web components	<code>javax.servlet.ServletRequestListener</code> and <code>ServletRequestEvent</code>
	Attribute added, removed, or replaced	<code>javax.servlet.ServletRequestAttributeListener</code> and <code>ServletRequestAttributeEvent</code>



# Implementierung von Listener

## Ableiten von Listener-Interface

```
public class MyContextListener implements ServletContextListener {  
    private ServletContext context;  
  
    public ContextListener() {  
    }  
  
    public void contextInitialized(ServletContextEvent evt) {  
        context = evt.getServletContext(); // Zugriff auf Applikationskontext  
        ...  
    }  
  
    public void contextDestroyed(ServletContextEvent evt) {  
        context = evt.getServletContext();  
        ...  
    }  
}
```

Initialisierung der Applikation



# Konfiguration in Listener

## Eintragen der Listener-Klasse in web.xml

Listener wird automatisch angemeldet

(auf Basis des implementierten Listener-Interfaces)

```
<?xml version="1.0" encoding="UTF-8"?>  
<web-app ...>  
    <display-name>ListenerTest</display-name>  
    ...  
    <listener>  
        <listener-class>at.jku.ssw.psw2.web.MyContextListener</listener-class>  
    </listener>  
    ...  
</web-app>
```



# Beispiel: Initialisierung einer Applikation

## Initialisierungsparameter in web.xml

```
<web-app>
  <context-param>
    <param-name>DBDriver</param-name>
    <param-value>org.apache.derby.jdbc.ClientDriver</param-value>
  </context-param>
  ...
  <listener>
    <listener-class>at.jku.ssw.psw2.web.MyContextListener</listener-class>
  </listener>
</web-app>
```

## Initialisieren der Applikation in ServletContextListener

- Zugriff auf Initialisierungsparameter mit `getInitParameter`
- Initialisieren der Applikation (z.B. Datenbank öffnen)
- Setzen von Attributen zur Weitergabe der Datenobjekte

```
public class MyContextListener implements ServletContextListener {
    private DataModel model;
    private ServletContext context;

    public void contextInitialized(ServletContextEvent evt) {
        context = evt.getServletContext();
        String driver = sc.getInitParameter("DBDriver");
        model = ...;
        context.setAttribute("model", model);
    }
}
```



# Initialisierung eines Servlets

## Methode `init()` der Servlet-Klasse

- Zugriff auf Initialisierungsparameter
- Zugriff auf Applikationskontext
- Zugriff auf Attribute
- Setzen von Objektvariablen des Servlets

### Beispiel:

```
<servlet>
  ...
  <init-param>
    <description>Default name</description>
    <param-name>name</param-name>
    <param-value>Herbert</param-value>
  </init-param>
</servlet>
```

```
public class DataModelServlet extends HttpServlet {
    private DataModel model;
    String name;
    ...
    @Override
    public void init() throws ServletException {
        super.init();
        name = getInitParameter("name");
        ServletContext context = getServletContext();
        model = (DataModel)context.getAttribute("model");
    }
}
```



Einführung

Servlets

Lebenszyklus

Sessions

JSP - JavaServer Pages

Trennung Logik und Darstellung

Expression Language und Tag Libraries

Zusammenfassung



## Sitzungen (Sessions)

### Probleme

HTTP ist ein nicht-sitzungsorientiertes Protokoll

Wenn der eBusiness-Kunde einen Artikel in den Einkaufskorb legt, woher weiß der Server, was schon im Einkaufskorb ist?

Wenn der Kunde seine Online-Einkaufstour beendet, woher weiß der Server, welcher Einkaufskorb zu welchem Kunden gehört?

### Lösungen

Cookies

URL mit Parameter

Versteckte HTML-Formular-Felder

### Unterstützung

Servlets bieten ein „Higher Level API“

*HttpSession*



## *HttpSession*

### ***HttpSession* <HttpServletRequest>.getSession()**

Liefert das aktuelle Session-Objekt, oder erzeugt ein neues, wenn noch keines existiert.

### ***HttpSession* <HttpServletRequest>.getSession(**bool c**)**

c = true: neues Session-Objekt anlegen wenn noch keines existiert.

c = false: null liefern wenn noch kein Session-Objekt existiert.

### **Object** **getAttribute(String name)**

Liefert das Objekt mit dem angegebenen Namen oder null falls es nicht existiert.

### **void** **setAttribute(String name, object o)**

Speichert das gegebene Objekt unter dem gegebenen Namen.

Existiert ein Name bereits, wird das Objekt überschrieben

Existiert ein Name bereits und o==null, so wird das Objekt entfernt



## Beispiel: Interaktion mit Sessions [1/4]

```
package at.jku.ssw.swe2;
import java.io.*;
import javax.servlet.ServletException;
import javax.servlet.http.*;

public class OpinionWithSession extends HttpServlet {

    public void doGet(HttpServletRequest req,
                      HttpServletResponse res)
        throws IOException {

        HttpSession ses = req.getSession();
        String last     = (String) ses.getAttribute("opinion");
        String cur      = (String) req.getParameter("opinion");
        ses.setAttribute("opinion", cur);

        PrintWriter out = res.getWriter();
        res.setContentType("text/html");
        out.println("<html>");
        out.println("<head><title>Interaction</title></head>");
        out.println("<body>");
        ...
    }
}
```



## Beispiel: Interaktion mit Sessions [2/4]

```
...
if (last == null && cur == null) {
    out.println("Please enter your opinion:");
}
else if (last == null || last.equals(cur)) {
    out.println("Your opinion is: " + cur);
    out.println("<br /> do you have a new one?");
}
else {
    out.println("Your opinon was: " + last);
    out.println("<br />and is now: " + cur);
    out.println("<br />do you want to change it again?");
}

out.println("<form action=\"Interact\" method=\"GET\">");
out.println("Your opinion: <input type=\"TEXT\"
            name=\"opinion\">");
out.println("<input type=\"SUBMIT\" value=\"Send\">");
out.println("</form></body></html>");
out.close();
}
}
```



## Beispiel: Interaktion mit Sessions [3/4]

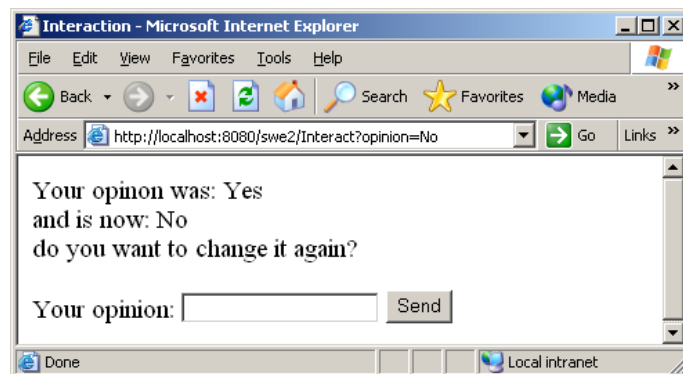
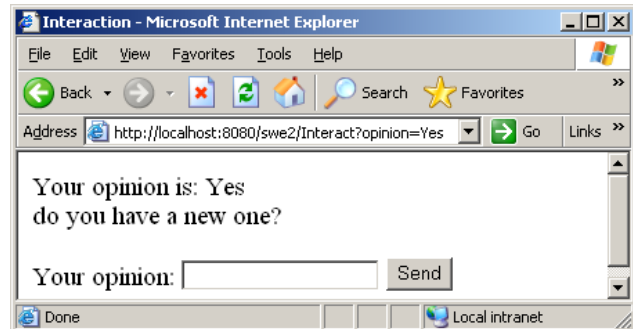
### web.xml Konfiguration

```
<web-app>
...
<servlet>
    <description></description>
    <display-name>OpinionWithSession</display-name>
    <servlet-name>OpinionWithSession</servlet-name>
    <servlet-class>at.jku.ssw.psw2.web.OpinionWithSession</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>OpinionWithSession</servlet-name>
    <url-pattern>/Interact</url-pattern>
</servlet-mapping>

...
</web-app>
```



## Beispiel: Interaktion mit Sessions [4/4]



## Java Servlets und Java Server Pages

Einführung

Servlets

Lebenszyklus

Sessions

JSP - JavaServer Pages

Trennung Logik und Darstellung

Expression Language und Tag Libraries

Zusammenfassung



# JSP vs. Servlet

JSP = JavaServer Pages

JSP-Datei wird vom WebServer einmal in ein Servlet übersetzt

Servlets aber auf einer abstrakteren Ebene programmieren

## JSP

Strikte Trennung zwischen  
Erscheinungsbild (HTML) und Logik  
(Java-Komponenten)

HTML-Code wie gewohnt verwenden.

Veränderung des Aussehens ohne  
Veränderung des dynamischen Inhalts

Änderung der JSP Seite hat  
automatische Übersetzung zur Folge

ASP ähnlich

## Servlets

Logik

HTML Code muss im Quellcode  
ausgegeben werden.

**Servlets nur zum Aufbereiten  
der Daten verwenden!**  
**JSP übernimmt die Anzeige!**



# JSP Beispiel

```
<html>
<head><title>JSP 1</title></head>
<body>

<%@ page import="java.util.*, java.text.*" %>

<% int max = 100; // pure Java code! %>
<P> JSP Test - Zähler bis <%= max %> </P>
<% for (int i = 0; i < max; i++) { %>
<%= i %> // Einfügen in html
<% }

String dateString = DateFormat.getDateInstance(DateFormat.SHORT,
Locale.GERMAN).format(new Date());
%>
<P> Heutiges Datum: <%= dateString %> </P>
</body>
</html>
```

Template-  
Text

JSP-  
Element



## Direktiven

### <%@ ... %> - Eigenschaften & Includes

- Erlaubt die Einstellung seitenbezogener Optionen
- Kann mehrmals vorkommen
- Inkludieren von externen Dateien

```
<%@ page
    language="java"
    import="java.util.*, java.text.*"
    contentType="text/html"
    ...
%>
```

```
<%@ include file="eineDatei.inc" %>
```

```
<%@ taglib
    uri="/tagliburi"
    prefix="kurzname"
%>
```



## Skriptelemente:

### <% ... %> - Skriptlet

Beliebiger ausführbarer Java-Code (Anweisungen, Variablendeklaration).

### <%! ... %> - Deklarationen

Variablen- und Methodendeklarationen

### <%= ... %> - Ausdrücke

Ausdruck wird in einen String konvertiert und **in den Seitentext eingefügt**.

Beispiele: <%=i%>, <%=new Date()%>, <%=methode()%>

Achtung: kein abschließendes Semikolon.

## Kommentare:

### <%-- ... --%> - Kommentar

Nur in der JSP Datei enthalten, nicht aber im Ergebnis.

In Skriptelementen normale Java Kommentare (`/* */` und `//`)



`<jsp:declaration> ... <jsp:declaration> ⇔ <%! ... %>`  
`<jsp:expression> ... <jsp:expression> ⇔ <%= ... %>`  
`<jsp:scriptlet> ... <jsp:scriptlet> ⇔ <% ... %>`  
...

## Regeln:

- Einfache Anführungszeichen entsprechen doppelten.
- Leerzeichen bei Zuweisungen von Attributwerten sind nicht erlaubt.
- Alle Tags sind case-sensitive.

## Gesamte Syntax:

JSP\_Syntax\_card12.pdf auf <http://java.sun.com/products/jsp/docs.html>



# Java Servlets und Java Server Pages

Einführung

Servlets

Lebenszyklus

Sessions

JSP - JavaServer Pages

Trennung Logik und Darstellung

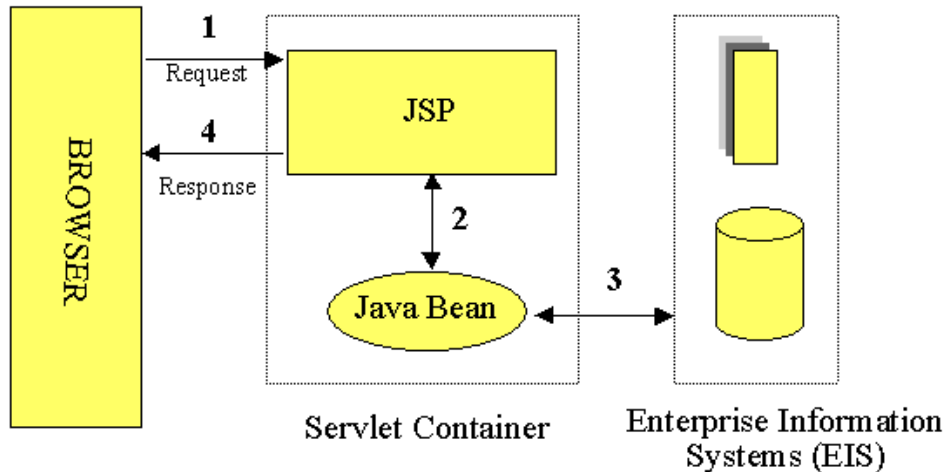
Expression Language und Tag Libraries

Zusammenfassung



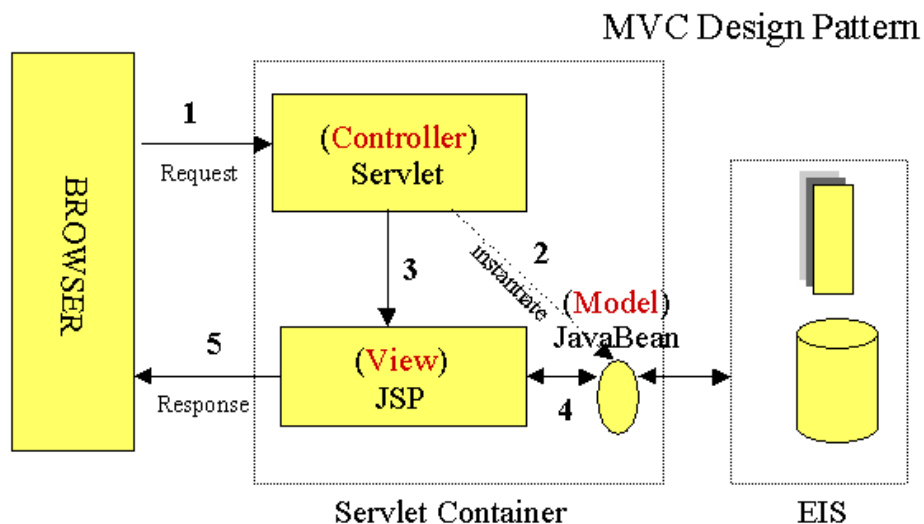
## Model 1 Architecture

- Request werden direkt von der JSP-Seite behandelt
- JSP-Seite übernimmt Steuerung und Generierung der Antwort
- Datenzugriff sollte in Komponenten (JavaBeans) ausgelagert werden



## Model 2 Architecture

- Serverseitige Realisierung des Model/View/Controller Pattern
- Steuerung wird durch ein Servlet übernommen
- Generierung des Outputs wird an JSP-Seiten delegiert
- Modelkomponenten werden von Servlet an JSP-Seiten weitergegeben



# Verwendung von JavaBeans

## JavaBeans-Elemente:

### <jsp:useBean>

JavaBean-Komponente verfügbar machen.

```
<jsp:useBean id="Instanzname" scope="Geltungsbereich"  
class="Klassenname"/>
```

### <jsp:getProperty>

Wert einer JavaBean-Eigenschaft abfragen.

### <jsp:setProperty>

Wert von JavaBean-Eigenschaften festlegen.

```
<jsp:setProperty name="Instanzname" property="*" />
```



# JSP Syntax: JavaBeans

## Syntax:

```
<jsp:useBean  
  id="Bean-Instanzname"  
  scope="page | request | session | application"  
  { class="Qualifizierter Klassenname"  
    [type="Qualifizierter Klassenname"]  
    | beanName="Qualifizierter Klassenname | <%! ... %>"  
    type="Qualifizierter Klassenname"  
    | type="Qualifizierter Klassenname"  
  }  
{</> | > weitere Elemente </jsp:useBean }  
  
<jsp:getProperty name="Bean-Instanzname"  
  property="Property-Name"/>  
  
<jsp:setProperty name="Bean-Instanzname"  
  ( property="*"  
  | property="Property-Name" [param="Parameter-Name"]  
  | property="Property-Name" value="String | <%! ... %>"  
  )  
</>
```

} ... Gruppierung  
| ... Alternative  
[] ... Option



## Beispiel: Verwendung von JavaBean [1/2]

```
<%@ page language="java" contentType="text/html" %>
<html><head><title>JSP with bean</title></head><body>
<jsp:useBean
  id="textBean" scope="page" class="at.jku.ssw.beans.Text"
/>
<jsp:setProperty name="textBean" property="*" />
Letzte Eingabe:
<jsp:getProperty name="textBean" property="text" />
<form name="text" action="usebean.jsp" method="get">
  <input type="text" name="text"/>
  <input type="submit" name="action" value="Senden"/>
</form></body></html>
```

```
package at.jku.ssw.beans;
public class Text {
  private String text;
  public void setText(String text) {
    this.text = text;
  }
  public String getText() { return text; }
}
```



## Beispiel: Verwendung von JavaBean [2/2]

The image displays three overlapping screenshots of a Microsoft Internet Explorer browser window, illustrating the state of a JSP application. Each window shows the address bar with the URL `http://localhost:8080/swe2/Bean/bean.jsp` and the page content.

- The top window shows the initial state: "Letzte Eingabe: null". Below the text is an empty text input field and a "Senden" button.
- The middle window shows the user has entered "Hallo" in the text input field.
- The bottom window shows the user has clicked the "Senden" button, and the page has updated to show "Letzte Eingabe: Hallo".



# JSP Syntax: forward und include

## Elemente für forward und include

### <jsp:include>

Einfügen einer anderen JSP-Seite.

### <jsp:forward>

Kontrolle an eine andere JSP-Seite weitergeben.

### <jsp:param>

Parameterwerte an durch <jsp:include> oder <jsp:forward> verwendete Seiten weitergeben.



# JSP Implizite Objekte

## Objekt

ServletRequest: **request**

ServletResponse: **response**

wird selten benutzt.

PageContext: **pageContext**

HttpSession: **session**

ServletContext: **application**

JspWriter: **out**

ServletConfig: **config**

Object: **page**

wird selten benutzt.

Throwable: **exception**

## Scope

Page

Page

Page

Session

Application

Page

Page

Page

Page



## Beispiel: forward und include [1/3]

Anwendung zum Abfragen eines gültigen Passwortes  
(Anforderung Länge  $\geq 8$  Zeichen)

### pass.jsp:

```
<%@ page language="java" contentType="text/html" %>
<html><head><title>Password</title></head><body>
<jsp:useBean id="pass" scope="request"
  class="at.jku.ssw.psw2.beans.Password"
/>
<% if (pass.isInitialized() && !pass.isValid()) { %>
  Passwort nicht gültig (zu kurz).<br>
<% } %>
Bitte Passwort eingeben (min. 8 Zeichen):
<form name="text" action="validate.jsp" method="post">
  <input type="text" name="text" value="<%=
    (pass.isInitialized()) ? pass.getPassword():"" %>"/>
  <input type="submit" name="action" value="Senden"/>
</form></body></html>
```



## Beispiel: forward und include [2/3]

### validate.jsp:

```
<%@ page language="java" %>

<jsp:useBean id="pass" scope="request"
  class="Beans.Password">
  <jsp:setProperty name="pass" property="password"
    param="text" />
</jsp:useBean>

<%
  if (pass.isValid()) {
    request.getRequestDispatcher("ready.jsp").
      forward(request, response);
  } else {
%>
    <jsp:forward page="pass.jsp"/>
<%
  }
%>
```



## Beispiel: forward und include [3/3]

ready.jsp:

```
<%@ page language="java" contentType="text/html" %>
<html>
<head><title>Password</title></head>
<body>
Danke für das gültige Passwort.
</body>
</html>
```

Bitte Passwort eingeben (min. 8 Zeichen):

Bitte Passwort eingeben (min. 8 Zeichen):

Passwort nicht gültig (zu kurz).

Bitte Passwort eingeben (min. 8 Zeichen):

Passwort nicht gültig (zu kurz).

Bitte Passwort eingeben (min. 8 Zeichen):

Danke für das gültige Passwort.

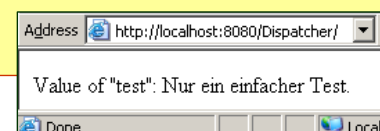


## Beispiel RequestDispatcher

```
public class TestInclude extends HttpServlet {
    public void doGet(HttpServletRequest req,
        HttpServletResponse res) {
        Object o = "Nur ein einfacher Test.";
        res.setContentType("text/html");
        req.setAttribute("test", o);
        try {
            getServletContext()
                .getRequestDispatcher("/jsp/testinclude.jsp")
                .include(req, res);
        } catch (Exception exc) { log(exc.getMessage()); }
    }
}
```

/JSP/testinclude.jsp

```
<html>
<head><title>RequestDispatcher Test</title></head>
<body>
Value of "test": <%= request.getAttribute("test") %>
</body>
</html>
```



Einführung

Servlets

Lebenszyklus

Sessions

JSP - JavaServer Pages

Trennung Logik und Darstellung

Expression Language und Tag Libraries

Zusammenfassung



## Motivation: Expression Language (EL)

```
public class Car {  
    ...  
    public Engine getEngine() {...}  
}
```

```
public class Engine {  
    ...  
    public int getPower() { ... }  
}
```

```
<html>  
...  
Your car got power:  
<%= ((Car) request.getAttribute("car")).getEngine().getPower() %> kW  
...  
</html>
```

Scripting

```
<html>  
...  
<jsp:useBean id="car" class="Car" scope="request" />  
Your car got power:  
<jsp:getProperty name="car" property="engine" /> kW  
...  
</html>
```

Keine Unterstützung für verschachtelten Zugriff

```
<html>  
...  
Your car got power: ${car.engine.power}  
...  
</html>
```

Expression Language (EL)



# Motivation: JSP Standard Tag Library (JSTL)

```
<html>
...
Your favourite cars:
<table>
<% String[] favs = (String[]) request.getAttribute("favcars");
   for (int i = 0; i < favs.length; ++i) {
%>
<tr><td><%= favs[i] %></td></tr>
<% } %>
</table>
...
</html>
```

Scripting

```
<html>
...
Your favourite cars:
<table>
  <c:forEach var="car" items="${favcars}">
    <tr><td>${car}</td></tr>
  </c:forEach>
</table>
...
</html>
```

JSTL



# Expression Language (EL)

Dot Operator

$\${\text{ersterBezeichner} \cdot \text{weitereBezeichner}}$

Erlaubt Zugriff  
auf Properties von  
Properties ...

Implizite Objekte

pageScope  
requestScope  
sessionScope  
applicationScope

param  
paramValues

header  
headerValues

cookie  
initParam

pageContext

Attribute aus

pageScope  
requestScope  
sessionScope  
applicationScope

Schlüssel einer Map

oder

Property

Muss sich an die Java  
Namenskonventionen halten!



# Expression Language (EL)

[] - Operator

Keine Java Namens-einschränkungen

$\${\text{Bezeichner} [\text{Bezeichner}]}$

Map	Schlüssel einer Map
Bean	Property
List	Listen-Index
Array	Array-Index

Beispiele:

$\${\text{cars}[\text{favoriteCars}[0]]}$

Verschachtelt

$\${\text{cars}["\text{Astra}"]}$

Map-Zugriff

$\${\text{cars}[\text{Astra}]}$

Auswerten von Astra & Map-Zugriff  
⇒ Astra muss ein Attribut sein!



# Expression Language Operatoren

Merke, JSP ist die View!

- Arithmetisch
  - Addition: +
  - Subtraktion: -
  - Multiplikation: \*
  - Division: /, div
  - Divisionsrest: %, mod
- Logisch
  - Und: &&, and
  - Oder: ||, or
  - Nicht: !, not
- Vergleichsoperatoren
  - Gleichheit: ==, eq
  - Ungleichheit: !=, ne
  - Kleiner: <, lt
  - Größer: >, gt
  - Kleiner gleich: <=, le
  - Größer gleich: >=, ge
- Literale
  - true
  - false
  - null
  - instanceof: reserviert
  - empty: Zeigt an ob ein Attribut gesetzt ist, z.B.:  $\${\text{not empty cars}}$

**Achtung!**

Keine Schlüsselwörter  
als Bezeichner  
verwenden!



# Java Standard Tag Library (JSTL)

- Nicht Teil der JSP-Spezifikation
- Übernimmt einfache Aufgaben
  - Iteration: `<c:foreach>`
  - Bedingungen: `<c:if>`
  - Alternativen: `<c:choose>`, `<c:when>`, `<c:otherwise>`
  - ...
- Installation (Tomcat)
  - `jstl.jar` und `standard.jar` nach `WEB-INF/lib` kopieren
  - Dateien sind in der Tomcat-Distribution unter `jsp-examples/WEB-INF/lib/` zu finden.
- Mehr Informationen
  - `J2EETutorial.pdf`
  - Beispiel `TicTacToe` von der Praktikumsseite.



## JSTL Beispiel `<c:foreach>`

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

<html>
<body>
  Print all cars:
  <c:forEach var="car" items="{Cars}">
    ${car}<br />
  </c:forEach>

  Print the numbers from 0 to 23 with a step of 5:
  <c:forEach begin="0" end="23" step="5" varStatus="counter">
    ${counter.count}<br />
  </c:forEach>
</body>
...
</html>
```



**Info** JSTL Tags können auch verschachtelt auftreten!



## JSTL Beispiel <c:if>

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

<html>
<body>
  <c:if test="${car eq 'Smart'}">
    Be smart drive Smart!
  </c:if>
  <c:if test="${car eq 'SUV'}">
    Real man drive hard!
  </c:if>
</body>
...
</html>
```



**Info**

Man kann einfache Anführungszeichen in EL benutzen!

Es gibt kein **else** für **<c:if>**!



## JSTL Beispiel <c:choose>, <c:when>, <c:otherwise>

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

<html>
<body>
  <c:choose>
    <c:when test="${car eq 'Smart'}">
      Be smart drive Smart!
    </c:when>
    <c:when test="${car eq 'SUV'}">
      Real man drive hard!
    </c:when>
    <c:otherwise>
      Porsche, all a lady can expect.
    </c:otherwise>
  </c:choose>
</body>
...
</html>
```



**Info**

**<c:choose>** funktioniert ähnlich einem switch-Statement; switch ~ choose, case ~ when und default ~ otherwise.



## JSTL Beispiel <c:url>, <c:param>

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

<html>
<body>
  Please visit our
  <a href="<c:url value='exhibition.do'>
    <c:param name='color' value='${customer.favouritColor}' />
  </c:url"> car exhibition</a>
  to see your next vehicle!

  <a href="<c:url value='logout.jsp' />" >Logout</a>
</body>
</html>
```



**Info** Ermöglicht Webanwendungen ohne Cookies!



## Zusammenfassung

- Expression Language
  - Ermöglicht einfachen Propertyzugriff über mehrere Ebenen
  - Weiters Zugriff auf
    - Abbildungen (Maps)
    - Listen
    - Arrays
  - Anwendung
    - Dot-Operator, einfach, kurz, für einfache Zugriffe
    - []-Operator, mächtiger, für komplexere Zugriffe wo man mit der Java-Namenskonvention an Grenzen stößt.
- JSTL
  - Tags für Standardaufgaben: Iteration, Option, Alternativen, ...
- Wer mit EL und JSTL nicht auskommt
  - Sollte sich Überlegen, ob ZU VIEL Logik in der JSP-Seite ist!
  - Muss sich eventuell mit „custom tag handlers“ beschäftigen
    - Siehe Beispiel TicTacToe von der Praktikumsseite.
- Buchtipp: Bryan Basham, Kathy Sierra & Bert Bates, Head First Servlets & JSP, O'Reilly, 2004



Einführung

Servlets

Lebenszyklus

Sessions

JSP - JavaServer Pages

Trennung Logik und Darstellung

Expression Language und Tag Libraries

Zusammenfassung



## Zusammenfassung

- Dynamische Webseiten
- Sitzungen
- JSP ~ ASP/ASP.NET
- JSP: Trennung von Aussehen und Implementierung
- JSP/Servlet vs. Applet
  - Ausführung am Server, Ergebnis: HTML Seite
  - Applets: Ausführung am Client, Ergebnis: interaktives Programm
- JSP ist abstraktes Servlet



## J2EE installieren

<http://java.sun.com/javaee/downloads/index.jsp>

## WTP Eclipse Plug-in installieren

<http://www.eclipse.org/webtools/>

## Apache Tomcat installieren

<http://tomcat.apache.org/>

