J2EEKurs

J2EE—Servlets und JavaServer Pages

Peter Thiemann

Universität Freiburg, Germany

Sommercampus J2EEKurs, Freiburg, Germany,
10.-14.10.2005

Servlets
- Einleitung
- Modell
- Lebenszyklus
- Beispiel
- Deployment
- Zustand in Webanwendungen

JavaServer Pages
- Einführung
- Verarbeitung
- Beispiele
- Zustandsverwaltung
- Expression Language

Anwendungsarchitektur

Servlets

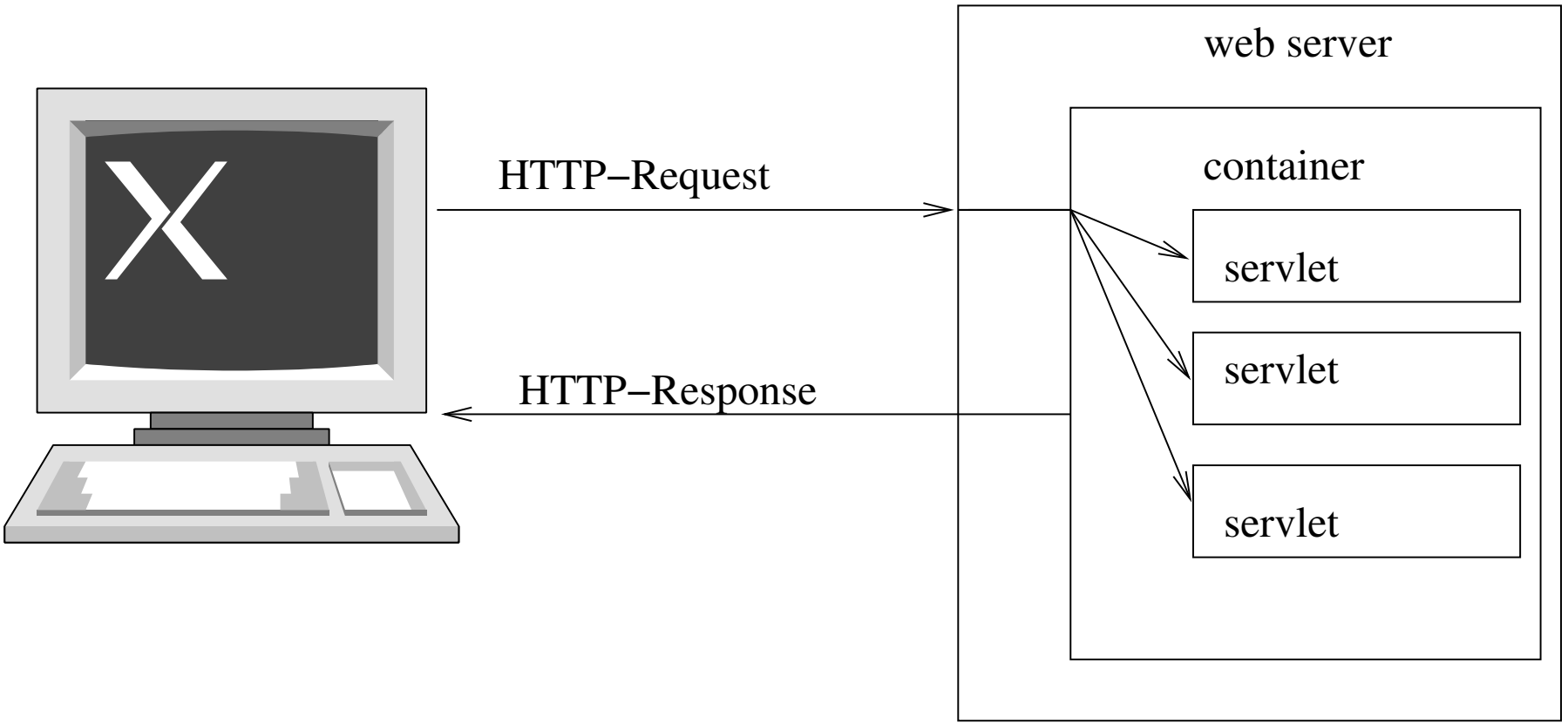
Servlets

- Frontendtechnologie
- programmorientiert
- Präsentationsschicht
- Keine Geschäftslogik

Aus der Servlet Spezifikation, v2.4

A servlet is a JavaTM technology-based Web component, managed by a container, that generates dynamic content. Like other Java technology-based components, servlets are platform-independent Java classes that are compiled to platform-neutral byte code that can be loaded dynamically into and run by a Java technology-enabled Web server. Containers, sometimes called servlet engines, are Web server extensions that provide servlet functionality. Servlets interact with Web clients via a request/response paradigm implemented by the servlet container.

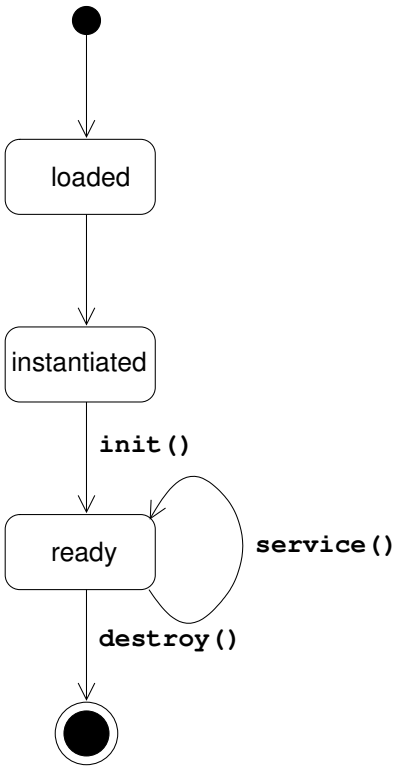
Das Servlet-Modell



Das Servlet-Programmiermodell

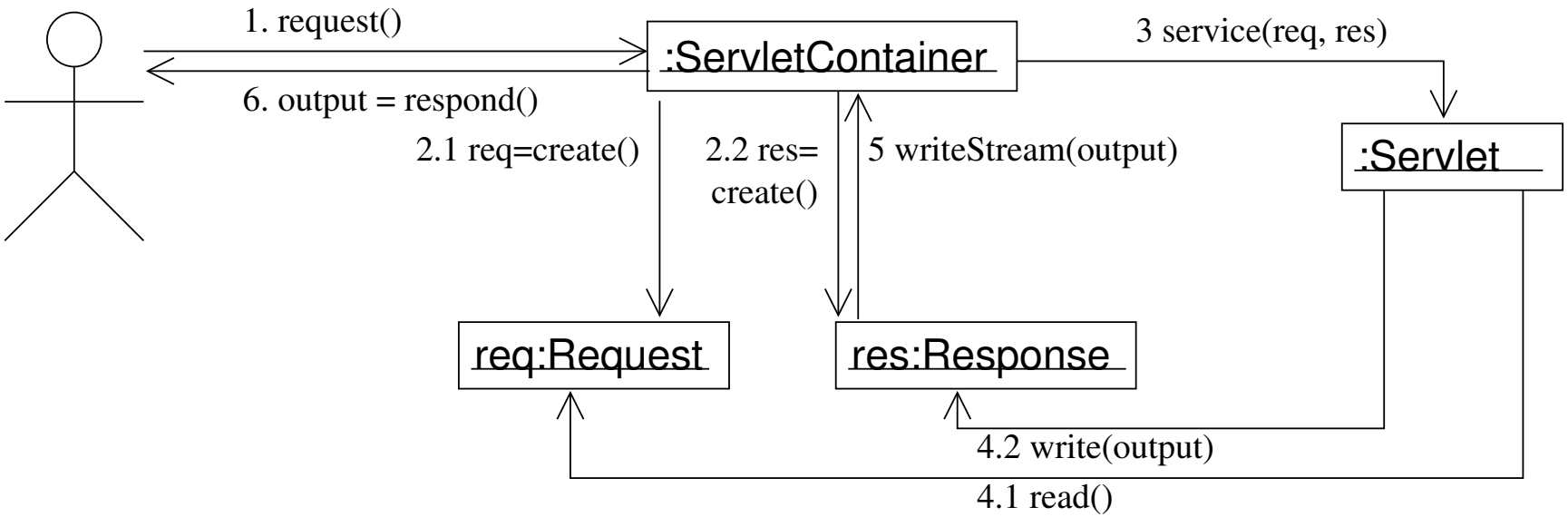
- Subklasse von `javax.servlet.http.HttpServlet`
- `init ()` Initialisierung
- `service (request, response)` Anfrageverarbeitung
  HTTP-Anfrage durch `HttpServletRequest`-Objekt
  HTTP-Antworten durch `HttpServletResponse`-Objekt
  Spezialversionen davon
  - `doGet (request, response)`
    Verarbeitung von `GET`-Anfragen
  - `doPost (request, response)`
    Verarbeitung von `POST`-Anfragen
  - `doHead (request, response)`
    Verarbeitung von `HEAD`-Anfragen
- `destroy ()` Finalisierung

Der Servlet-Lebenszyklus



- Laden und Instanzieren durch Servlet-Container
- Mehrfache Instanzierung in unterschiedlichen Threads möglich
- Initialisieren
  Methode `init()`
- Anfrageverarbeitung
  Methode `service(...)`, `doGet(...)`, `doPost(...)`, ...
- Beenden
  Methode `destroy()`

Kollaborationsdiagramm für einen Servlet-Aufruf



Servlet-Beispiel: Echo

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Echo extends HttpServlet {
  public void doGet (HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
      response.setContentType ("text/html; charset=utf-8");
      PrintWriter out = response.getWriter ();
      out.println ("<html>");
      out.println ("<head><title>Echo Results</title></head>");
      out.println ("<body><h3>Echo Results</h3><ul>");
      Enumeration e = request.getParameterNames ();
      while (e.hasMoreElements ()) {
          String name = (String)e.nextElement ();
          String value = request.getParameter (name);
          out.print ("<li>");
          out.println (name + ": " + value);
      }
      out.println ("</ul></body></html>");
  }
}
```

Hinweise

- Jedes Servlet muss eine der do-Methoden (`doGet`, `doPost`, etc) überschreiben.
- Erzeugen der Ausgabe durch Drucken auf `response.getWriter()`.
- Zugriff auf Formulardaten durch `request.getParameter (String name)`.
- Vor Verarbeitung von Formulardaten die Parameter stets auf `null` testen.
- Umleiten einer Anfrage auf ein anderes Servlet `response.sendRedirect (String url)`.

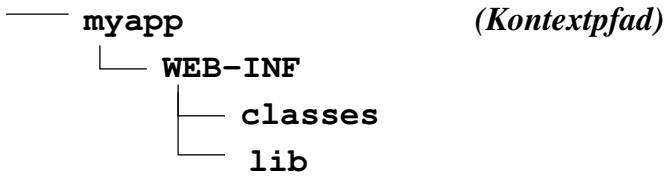
Webanwendung

Eine Webanwendung besteht aus

- einer Menge von Servlets
- Hilfsklassen und Bibliotheken
- XHTML Dokumenten, JavaServer Pages
- Stylesheets
- Bildern, etc.

abgelegt in standardisierter Verzeichnisstruktur

Verzeichnisstruktur einer Webanwendung



- `myapp/` (und alle Unterverzeichnisse ausser `WEB-INF`)
  statische Ressourcen (XHTML Seiten, Bilder, Stylesheets)
- `myapp/WEB-INF/`
  Deployment Descriptor in Datei `web.xml` (s.u.)
- `myapp/WEB-INF/classes/`
  `class`-Dateien von Servlets und Hilfsklassen
  in Verzeichnisstruktur entsprechend den Paketnamen
- `myapp/WEB-INF/lib/`
  `jar`-Dateien im CLASSPATH der Anwendung

Deployment Descriptor

- Erforderlich in jeder Web Anwendung
- Konfigurationsdaten
  - Abbildung von URI-Pfaden auf Ressourcen der Anwendung
  - Initialisierungsparameter; verfügbar über `String getInitParameter (String name)`
  - Fehlerbehandlung
  - Zugriffsbeschränkungen und Authentisierung
  - Registrierung von Listeners und Filters
- deklarativ, getrennt vom Code

Beispiel: Deployment Descriptor

```
<?xml version="1.0" encoding="utf-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
         version="2.4">
    <display-name>Echo Your Parameters</display-name>
    <servlet>
        <servlet-name>MyFirstServlet</servlet-name>
        <servlet-class>Echo</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>MyFirstServlet</servlet-name>
        <url-pattern>/echo/*</url-pattern>
    </servlet-mapping>
</web-app>
```

Globaler Zustand

- spezifisch für Anwendung oder Server
- Lebensdauer: persistent, bestimmt durch Anwendung
- Anwendung: Kontextattribute, Datenbank
- Durch `ServletContext getServletContext()`
- Servletcontainer erzeugt **ein** `ServletContext`-Objekt pro Anwendung/context path (URL-Präfix)
- Zustand mit *Attributen* des `ServletContext`
  - `void setAttribute(String name, Object o)`
  - `Object getAttribute(String name)`
- Kontext einer anderen Anwendung: `ServletContext getContext(String uripath)`

Sitzungszustand

- spezifisch für eine Gruppe von Anfragen und Antworten
- Bsp: Einkaufswagen
- Lebensdauer: Sitzung bzw Neustart des Servletcontainers
- Repräsentiert durch `Session`-Objekt
- Klasse `HttpServletRequest`
  Methode `HttpSession getSession (bool create)`
  liefert aktuelles Sitzungsobjekt
- Sitzung lokal zur Anwendung (`ServletContext`)

Flüchtiger Zustand

- spezifisch für eine Anfrage
- Parameter und lokale Variablen in `doGet`, `doPost`, etc

Beispiel mit Zustand: NumberGuess

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

public class NumberGuess extends HttpServlet {
  public void doGet (HttpServletRequest request,
                     HttpServletResponse response)
      throws ServletException, IOException
{
    HttpSession session = request.getSession (true);
    response.setContentType ("text/html; charset=utf-8");
    PrintWriter out = response.getWriter ();
    out.println ("<html><head><title>Number Guess</title></head>");
    out.println ("<body><h3>Number Guess</h3>");
    String guessString = request.getParameter ("guess");
```

Beispiel mit Zustand: NumberGuess/2

```
    if (guessString == null) {
        long n = Math.round (Math.random () * 100);
        session.setAttribute ("SN", new Long (n));
        out.println ("I am thinking of a number from 1-100");
    } else {
        long guess = Long.parseLong (guessString);
        long n = ((Long)session.getAttribute ("SN")).longValue ();
        if (guess == n) {
            out.print ("You got it!");
        } else if (guess > n) {
            out.println ("Try lower");
        } else {
          out.println ("Try higher");
        }
    }
    String uri = request.getRequestURI ();
    out.println ("<form action =\"" + response.encodeURL (uri) + "\" method=\"get\">");
    out.println ("<input type=\"text\" name=\"guess\"/>");
    out.println ("<input type=\"submit\" value=\"Make A Guess\"/>");
    out.println ("</form></body></html>");
  }
}
```

Zusammenfassung Servlets

Plattformunabhängigkeit Servlets können auf jeder Java-Plattform ohne Rekompilierung ausgeführt werden.

Multi-Threading Servlets werden einmal geladen und dann beliebig oft in verschiedenen Threads instanziert (abhängig vom Container).

Effizienz Eine Anfrage kann meist eine existierende Instanz verwenden.

Flexibilität Servlets können beliebige Java-Bibliotheken verwenden.

JavaServer Pages

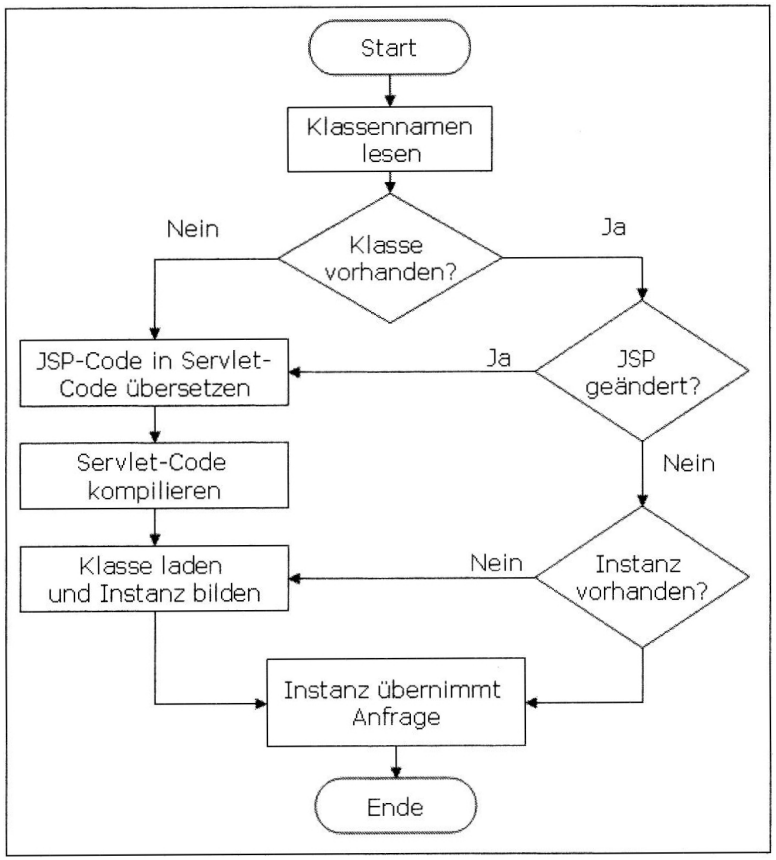
JavaServer Pages

- Frontendtechnologie, . . .
- seitenorientiert (XML/XHTML)
- basierend auf Servlets
- Anspruch:
  - möglichst wenig Java-Code
  - erstellbar mit XHTML-Designwerkzeug

JavaServer Pages (JSP)

- JSP = XML + Java + EL (Expression Language)
- JSPs werden direkt auf Server abgelegt
  - Transparente Übersetzung in Servlets
  - Transparente Übersetzung in Bytecode
- XML-basierte Erweiterung durch *Tag-Bibliotheken*
- Zusätzlich zu Servlets
  - Vereinfachter Zugriff auf Kontext
  - Zustandsverwaltung durch Parameter

Verarbeitung von JSPs



JSP Beispiel

```
<!-- Hallo.jsp -->
<hmtl>
  <head>
    <title>Hallo Welt</title>
  </head>
  <body>
    <p>Hallo Welt, die Uhrzeit ist:
       <% new java.util.Date() %>
    </p>
  </body>
</html>
```

JSP—übersetzt

```
public class abcedfgh extends HttpServlet {
  public void doGet(HttpServletRequest req,
                    HttpServletResponse res)
    throws ServletException, IOException {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.print("<html>"
      + "<head><title>Hallo Welt</title></head>"
      + "<body><p>Hallo Welt, die Uhrzeit ist: ");
    out.print(new java.util.Date());
    out.print("</p></body></html>");
  }
}
```

Beispiel: Hit Counter

```
<% response.addDateHeader("Expires", 0); %>
<html>
  <head><title>JSP</title></head>
  <body><h1>Hello World!</h1>
  <%! int hits = 0; %>
  You are visitor number
  <% synchronized(this) { out.println(++hits); } %>
  since the last time the service was restarted.
  <p> This page was last updated:
  <%= new java.util.Date().toLocaleString() %>
  </p></body>
</html>
```

Kontext eines Servlets

- Automatisch deklarierte Variablen (vgl. Servlets)

```
HttpServletRequest request;
HttpServletResponse response;
HttpSession session;
ServletContext application;
ServletConfig config;
```

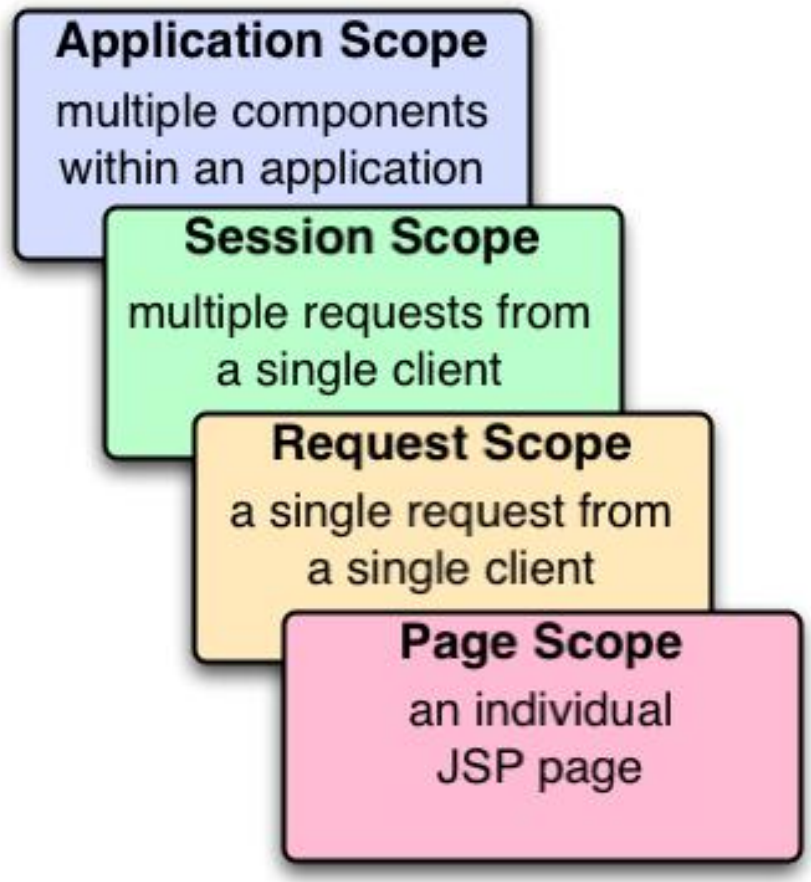
- Ausgabekanal (Exceptions, Pufferung)

```
JspWriter out;
```

Zustandsverwaltung (Attribute)

- Attribut = benanntes Objekt
- abgelegt in und zugreifbar durch Laufzeitumgebung
- Vier Zustandsebenen (*scoped attributes*)
  - *page scope*
    `PageContext pageContext;` `getAttribute`, `setAttribute` (alle Ebenen zugreifbar)
    `findAttribute` durchsucht alle Ebenen
  - *request scope*
  - *session scope* (auch über Session-Objekt)
  - *application scope* (auch über Kontext-Objekt)

Funktion der Ebenen



JSP Expression Language

- Ziel: Anwendungen ohne Java Syntax / Kenntnisse
- Syntax ${ *expression* }
- Innerhalb von XML-Fragmenten
- Ersetzt durch Wert des *expression*
- Syntax vergleichbar mit JavaScript (vgl. JSP-Spezifikation)
- Objektoperationen: `${shoppingCart.price}`
  wird transformiert nach

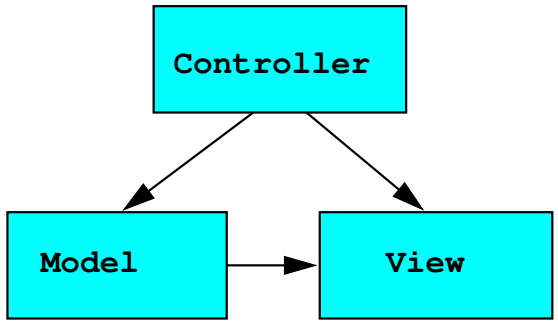
```
pageContext.findAttribute ("shoppingCart")
           .getPrice()
```

Implizite Objekte

- `param` Request Parameter
- `pageContext`
- `cookie`
- Beispiel

```
<html>
  <head><title>Addition</title></head>
  <body bgcolor="${param.color}">
    The sum of ${param.x} and ${param.y} is
    ${param.x+param.y}
  </body>
</html>
```

Model-View-Controller



- Standardmuster für GUI-Anwendungen (Smalltalk)
- **Model** enthält die Daten und Operationen darauf
- **View** enthält die Präsentationsschicht
- **Controller**
  - Interaktion mit dem Klient
  - Veränderung des Modells
  - Auswahl der Views

Model 2 Architektur

- Instanz von MVC für Webanwendungen
- **Controller**
  Servlet als Dispatcher von JSPs
- **View**
  JavaServer Pages
  - nur Präsentationslogik
  - unabhängig von Daten und Geschäftslogik
- **Model**
  Java Klassen
  - Datenhaltung mit EJB *entity beans*
  - Geschäftslogik mit EJB *session beans*

Controller

- Servlet als zentraler Dispatcher
- Konvention für URLs von Einstiegspunkten: *start*.do, *entry1*.do, *entry2*.do, ...

```
<servlet>
  <servlet-name>Controller</servlet-name>
  <servlet-class>Controller</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>Controller</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

Controllerzustand

```
public class Controller extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response) {
    HttpSession session = request.getSession();
    String command = request.getServletPath();
    Model model;
    if (command.equals("/start.do")) {
      model = new Model();
      session.setAttribute("model", model);
      getServletContext()
        .getRequestDispatcher("/start.jsp")
        .forward(request, response);
    } else if (command.equals("/entry1.do")) { // ...
```