J2EE

Java 2 Enterprise Edition
Ein Überblick

Margrit Höhme
8.12.2004 – ForumF, Köln

J2EE

- **Warum J2EE lernen?**
- Java und das Internet
- J2EE-Konzepte
- Servlets und JSPs
- Web-Anwendungen

Warum J2EE lernen?

- J2EE ist eine Standard-Technologie für Web-Anwendungen
- Java-Programmierung ist meist J2EE-Entwicklung
- Zielgruppen
  - Java-Programmiererinnen
  - angrenzende Berufsbilder, z.B. HTML-Designerinnen, die dynamische Web-Anwendungen entwickeln (möchten)
  - Administratorinnen, die J2EE-Anwendungen verwalten (möchten)

Java ≠ J2EE

- Java = Programmiersprache
- J2EE = Spezifikationen und Richtlinien für die Entwicklung verteilter Komponenten
  - J2EE-Framework ist in Java geschrieben
  - J2EE-Komponenten werden in Java geschrieben
    - Framework = Sammlung von Klassen für bestimmte Aufgaben
    - Komponente = Teil einer Anwendung mit einer bestimmten Aufgabe; besteht aus mehreren Klassen
    - verteilt (engl. *distributed*) = über mehrere Rechner hinweg

Typische Einsatzgebiete

- Web-Anwendungen
  - Shops, Amazon
  - Mail-Sites
  - Ebay
  - Online-Banking
- Middleware
  - Enterprise-Services
  - Anbindung von Datenbanken und Legacy-Systemen

J2EE

- Warum J2EE lernen?
- **Java und das Internet**
- J2EE-Konzepte
- Servlets und JSPs
- Web-Anwendungen

Java und das Internet (I)

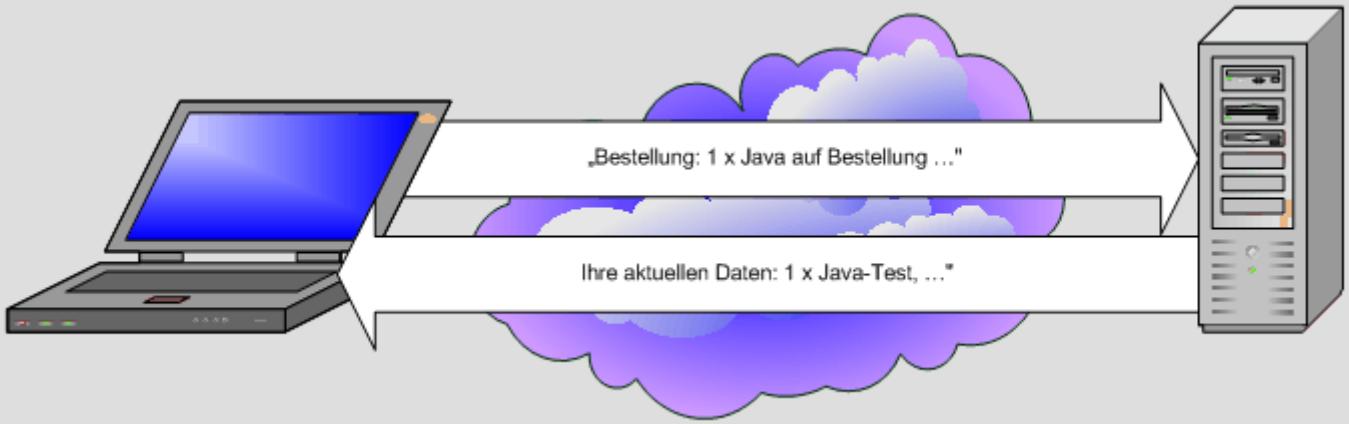
- 1960 Entwicklung und Nutzung von Rechnernetzen für dezentrale Datenhaltung durch das US-Militär (ARPANET)
- 1980 Nutzung von Rechnernetzen für den akademischen Austausch
  TCP/IP
- 1989 URL und HTML (Tim Berners-Lee), HTTP
  Ermöglichung einer kommerziellen Nutzung
- 1992 Erster grafischer Browser

Java und das Internet (II)

1993	CGI-Programmierung
1994	PHP
1995	Java 1.0
1996	SSL (Netscape)
1997	Java-Servlets
1998	Java 2
1999	Java Server Pages J2EE
2003	J2EE-Spezifikation 1.4

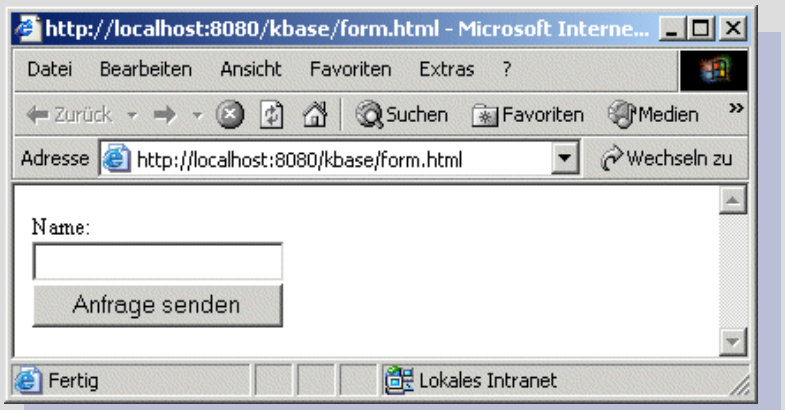
Kommunikationsrichtungen in Web-Anwendungen

- Client: Daten senden
  - Nachricht übermitteln
  - Online-Bestellung
- Server: gesendete Daten auswerten
  - Daten in DB schreiben
  - Versandprozess an-stoßen
- Client: Daten abrufen
  - aktuelle Börsenkurse
  - Produktkatalog
  - Gelbe Seiten
- Server: HTML-Seiten und Grafiken generieren



HTML-Formulare

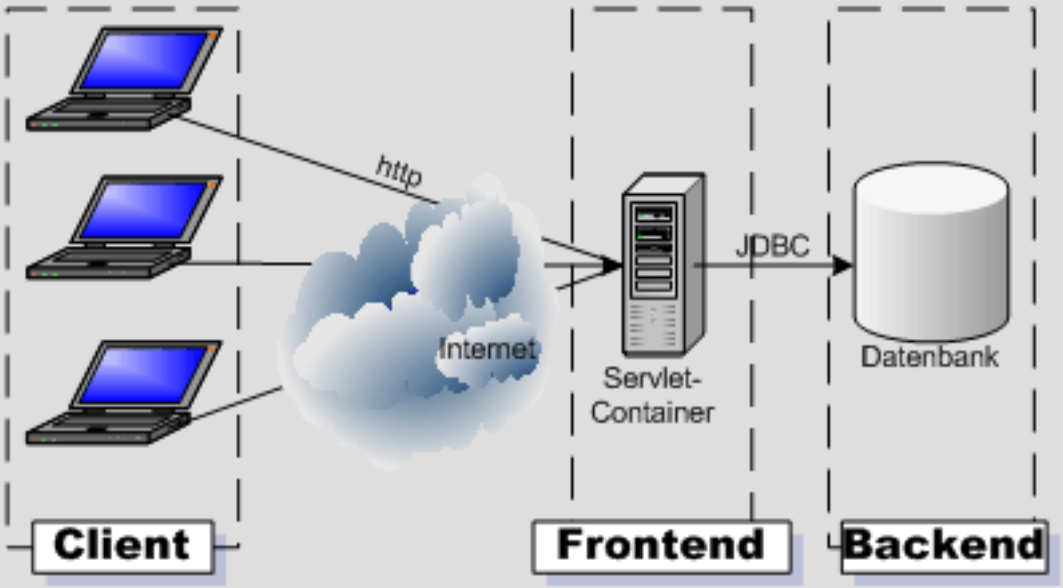
```
<HTML>
<BODY>
<FORM method="GET"
      action="/getservlet">
Name:<BR>
<INPUT type="text"
       name="name"><BR>
<INPUT type="submit">
</FORM>
</BODY>
</HTML>
```



- HTTP-Methode
  - GET: Parameter werden an die URL angehängt
    - Begrenzt auf ca. 240 Zeichen
  - POST: Parameter werden als Content übermittelt
    - unbegrenzte Zeichenzahl
- Action
  - URL des Servlets, das die Formulardaten auswerten soll

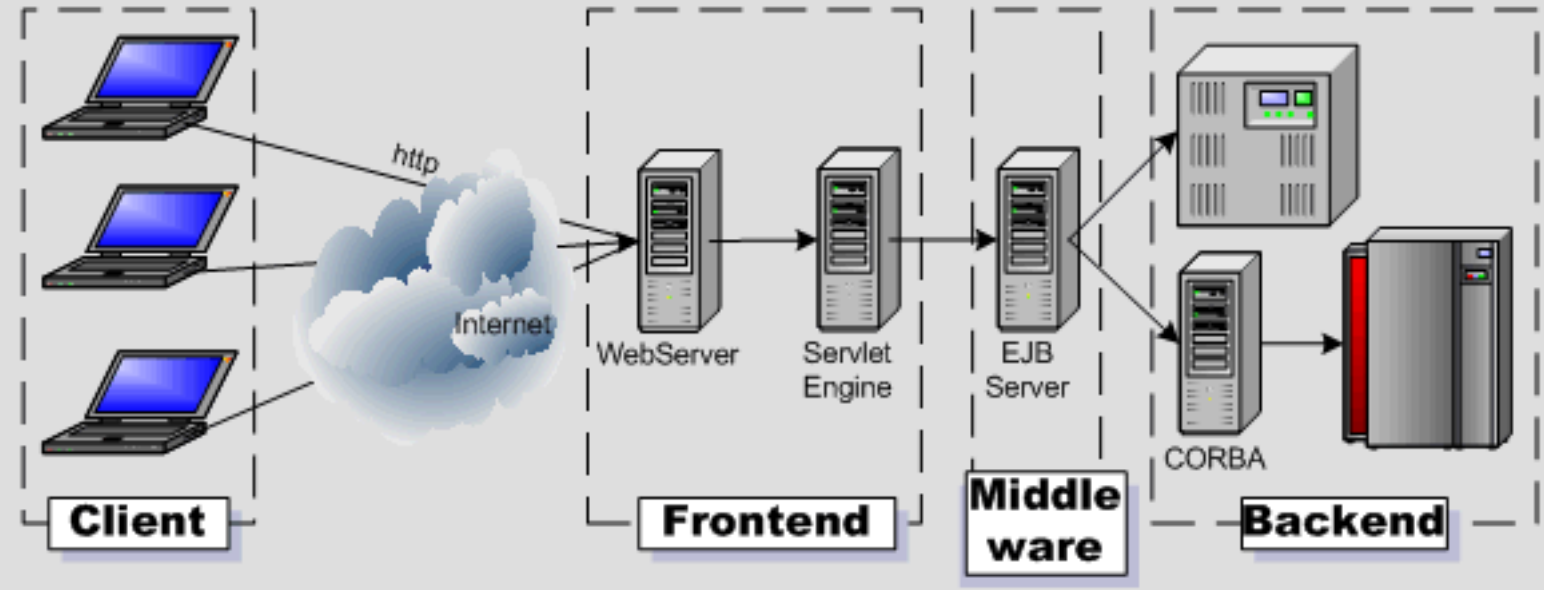
Mehrschichtarchitektur (I)

- Die meisten dynamische Anwendungen benötigen Datenhaltungssysteme
- Beispiel für eine einfache Drei-Schicht-Architektur:



Mehrschichtarchitektur (II)

- Beispiel für eine komplexe Fünf-Schicht-Architektur:

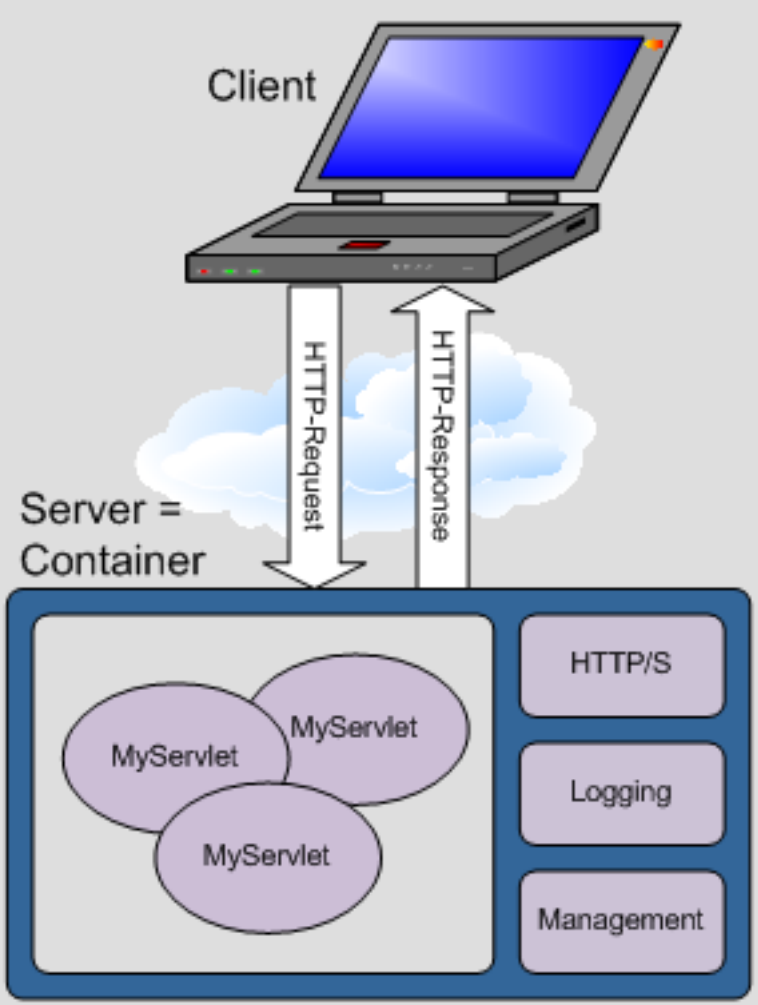


Schicht: *engl.* tier; Mehrschichtarchitektur: multi-tier architecture

J2EE

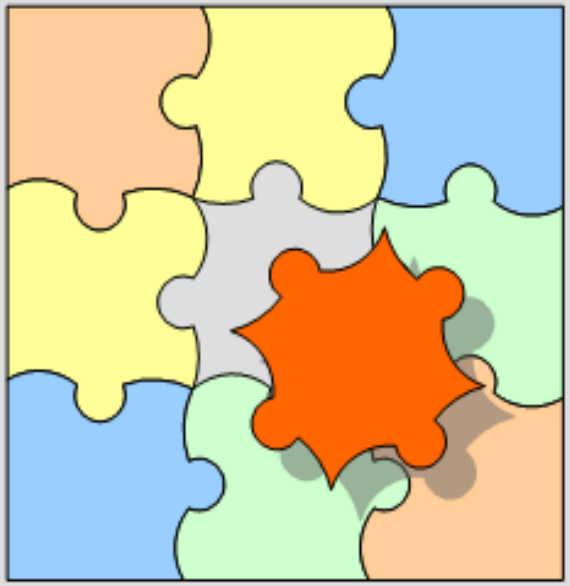
- Warum J2EE lernen?
- Java und das Internet
- **J2EE-Konzepte**
- Servlets und JSPs
- Web-Anwendungen

Begriff des „Containers“



- Laufzeitumgebung für bestimmte Arten von Komponenten
- stellt J2EE-Services bereit, die eine Komponente nutzen kann
  - HTTP/S
  - JDBC (Datenbankanbindung)
  - Sicherheit
  - Transaktionen
  - effiziente Verwaltung von Datenbank-verbindungen (Pooling)
  - ...

Aufgabenteilung



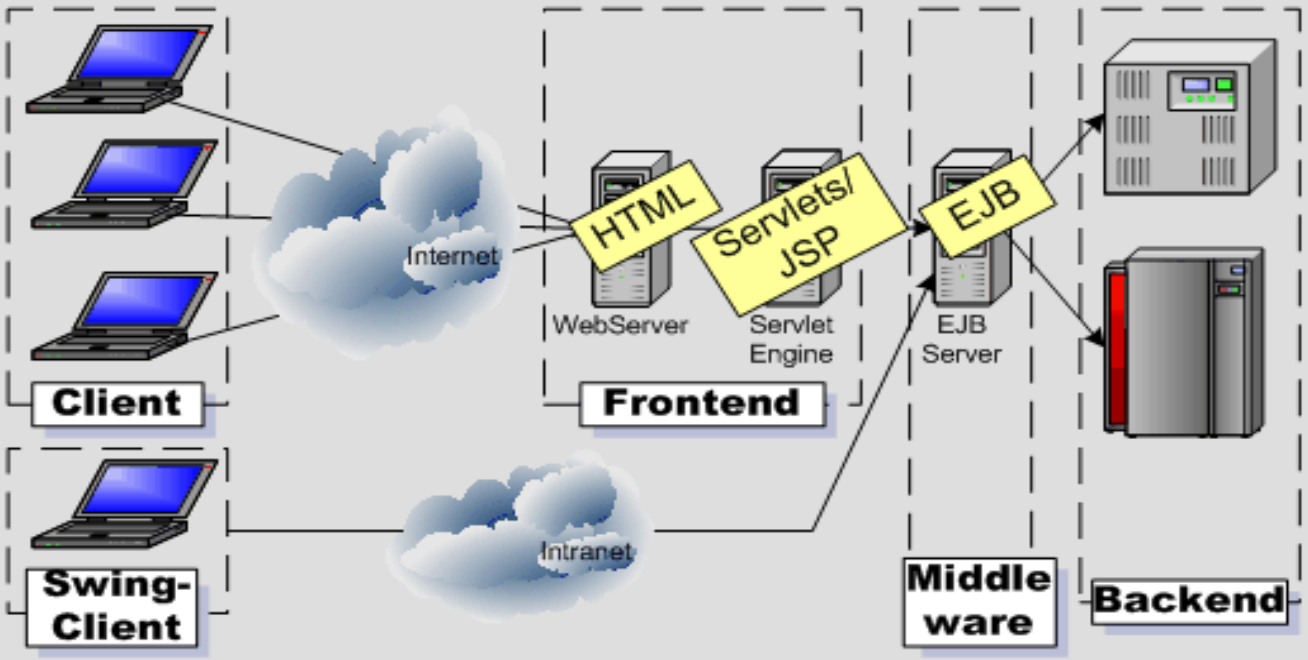
- Container-Provider
  - implementiert technische Belange, die für verschiedene Anwendungsgebiete ähnlich sind, z.B.
    - Kommunikation, Security, Datenbankanbindung, Transaktionen, Pooling, ...
- Komponenten-Provider
  - implementiert fachliche Belange, z.B.
    - Zinsberechnung
    - Buchbestellung
- Ziele der Aufgabenteilung
  - bessere Spezialisierung der Entwickler
  - schnellere Entwicklung

Spezifikationen

- Beschreiben die Schnitt-stellen zwischen Container und Komponente
- in Englisch
- sehr technisch
- als Einstieg nicht geeignet

Servlets vs. EJBs

- J2EE umfaßt Servlets wie auch EJBs
- unterschiedliche Aufgaben
  - Servlets und JSPs ⟶ Frontends
  - EJBs ⟶ Middleware

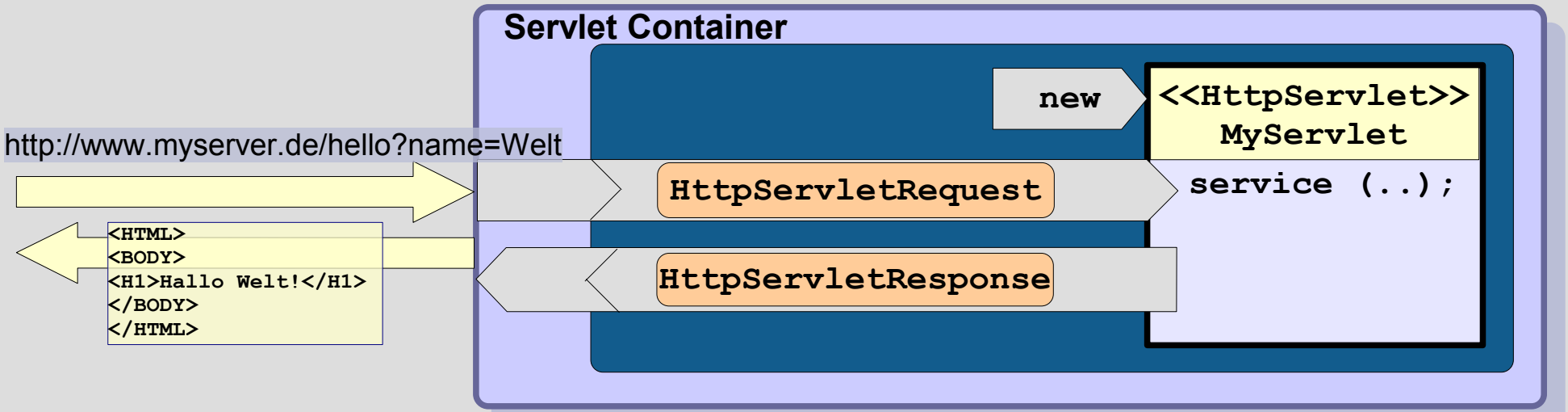


J2EE

- Warum J2EE lernen?
- Java und das Internet
- J2EE-Konzepte
- **Servlets und JSPs**
- Web-Anwendungen

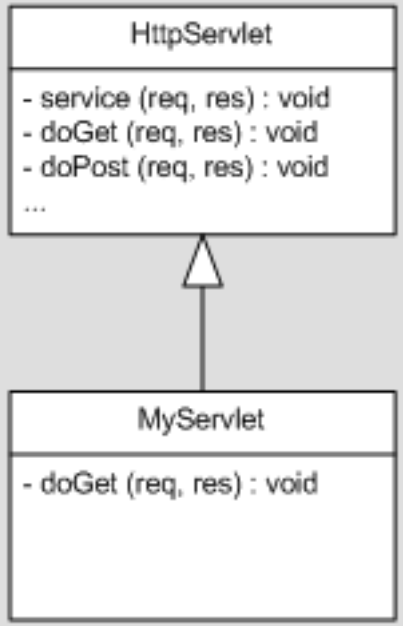
Servlet-Container

- Der Client kommuniziert nur indirekt mit dem Servlet
- Container:
  - erzeugt Servlet-Instanzen (`new MyServlet();`)
  - erzeugt für jeden Aufruf ein Request- und ein Response-Objekt
  - ruft die `service()`-Methode mit dem Request- und Response-Objekt auf
  - erzeugt aus dem Response-Objekt eine HTTP-Response



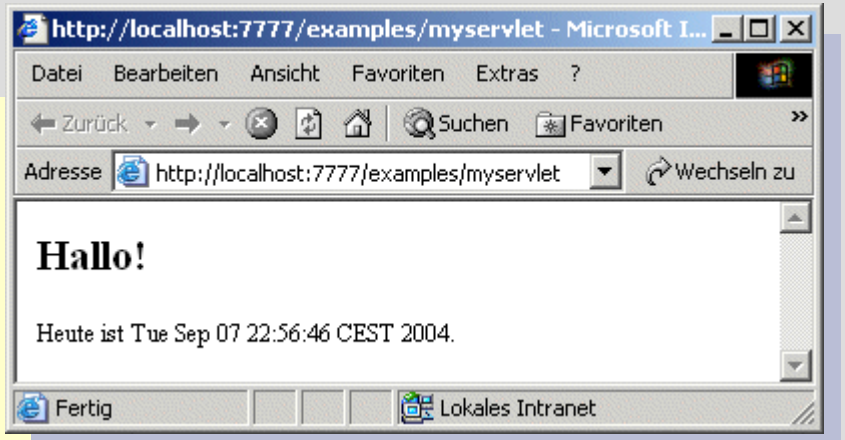
Servlet-Entwicklung

- Servlet ist eine Java-Klasse
- erweitert **`javax.servlet.http.Http-Servlet`**
  - Prinzip der Vererbung: Der Container „sieht“ nur Servlet-Objekte, deren **`service()`**-Methode er ausführt
- überschreibt
  - **`doGet()`**- *und/oder* **`doPost()`**-Methode
  - **`init()`**-Methode, wenn erforderlich
  - die implementieren Methoden müssen grundsätzlich Thread-sicher sein

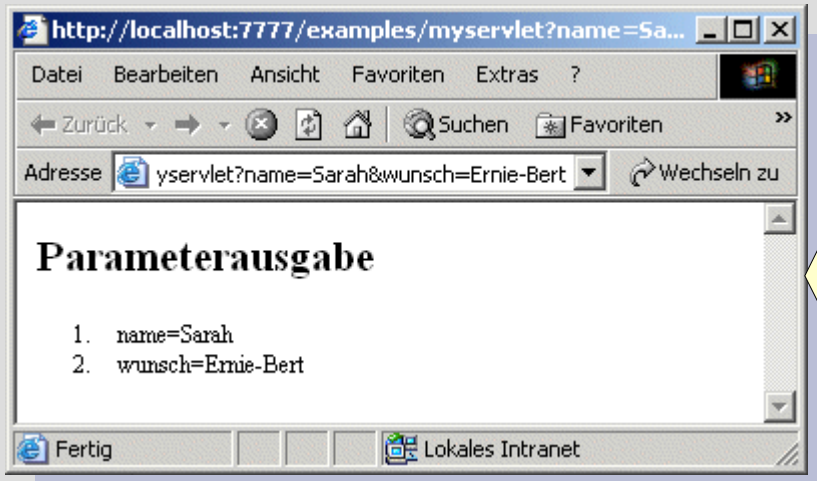


Ein Servlet (I): Response schreiben

```
package de.mhoehme.servlets;

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

import java.util.Date;

public class DateServlet extends HttpServlet {

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

      res.setContentType("text/html");

      PrintWriter out = res.getWriter();
      out.println ("<HTML><BODY><H1>Hallo!</H1>");
      out.println ("<P>Heute ist " +
          new Date()+".</P>");
      out.println ("</BODY></HTML>");
   }
}
```



Servlet (II): Request-Parameter lesen




```
package de.mhoehme.servlets;

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

public class MyServlet extends HttpServlet {

   public void doGet (HttpServletRequest req,
                      HttpServletResponse res)
   throws IOException, ServletException {
      res.setContentType("text/html");

      PrintWriter out = res.getWriter();
      out.println ("<HTML><BODY>");
      out.println ("<H1>Parameterausgabe</H1>");
      out.println ("<OL>");
      Enumeration e = req.getParameterNames();
      while (e.hasMoreElements()) {
         String key = (String) e.nextElement();
         String value = req.getParameter(key);
         out.println ("<LI>"+key+"="+value+"</LI>");
      }
      out.println ("</OL>");
      out.println ("</BODY></HTML>");
   }
}
```

Java Server Pages (JSPs)

- Konsequente Weiterentwicklung des Servlet-Konzepts
- Vorteile gegenüber Servlets:
  - HTML-Code
    - Java-Code kann in Form von sog. Scriptlets eingebettet werden (`<% ... %>`)
    - Funktionalität kann in Form von TagLibs eingebunden werden (z.B. Datums- und Betragformatierung)
  - Bearbeitung in gängigen HTML-Tools möglich
  - HTML-Designer verstehen JSPs :-)
- Der Container generiert aus JSPs Servlets und führt diese aus

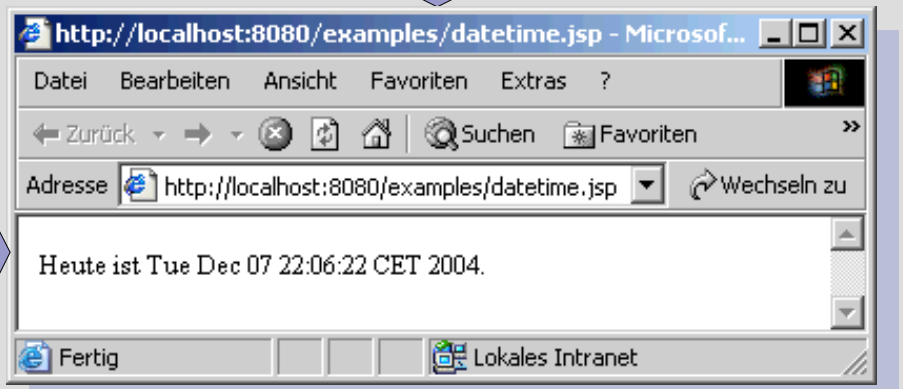
Servlets vs. JSPs

Servlet

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

public class MyServlet extends HttpServlet {

   public void doGet (
      HttpServletRequest req,
      HttpServletResponse res)
   throws IOException {
      res.setContentType ("text/html");

      PrintWriter out = res.getWriter ();
      out.write ("<HTML><BODY><P>");
      out.write ("Heute ist "+
          new Date()+".");
      out.write ("</P></BODY></HTML>");
      out.close();
   }
}
```

Java Server Page

```
<%@ page language="java"
         import="java.util.Date"%>
<HTML>
<BODY>
<P>Heute ist <%=new Date() %>.</P>
</BODY>
</HTML>
```



Servlet und JSP erzeugen dieselbe Ausgabe!

Generiertes Servlet

```
package org.apache.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import org.apache.jasper.runtime.*;
import java.util.Date;

public class test_jsp extends HttpJspBase {


  private static java.util.Vector _jspx_includes;

  public java.util.List getIncludes() {
    return _jspx_includes;
  }

  public void _jspService(HttpServletRequest request,
        HttpServletResponse response)
       throws java.io.IOException, ServletException {

    JspFactory _jspxFactory = null;
    javax.servlet.jsp.PageContext pageContext = null;
    HttpSession session = null;
    ServletContext application = null;
    ServletConfig config = null;
    JspWriter out = null;
    Object page = this;
    JspWriter _jspx_out = null;
```

```
    try {
      _jspxFactory = JspFactory.getDefaultFactory();
      response.setContentType(
             "text/html;charset=ISO-8859-1");
      pageContext = _jspxFactory.getPageContext(
         this, request, response,
         null, true, 8192, true);
      application = pageContext.getServletContext();
      config = pageContext.getServletConfig();
      session = pageContext.getSession();
      out = pageContext.getOut();
      _jspx_out = out;

      out.write("\r\n");
      out.write("<HTML>\r\n");
      out.write("<BODY>\r\n");
      out.write("<H1>Heute ist ");
      out.print(new Date() );
      out.write(".");
      out.write("</H1>\r\n");
      out.write("</BODY>\r\n");
      out.write("</HTML>");
    } catch (Throwable t) {
      out = _jspx_out;
      if (out != null && out.getBufferSize() != 0)
        out.clearBuffer();
      if (pageContext != null)
          pageContext.handlePageException(t);
    } finally {
      if (_jspxFactory != null)
          _jspxFactory.releasePageContext(pageContext);
    }
  }
}
```

Generiert von Tomcat, Version 4.1.18
<TOMCAT_HOME>\work\Standalone\localhost\test\test_jsp.java

Kontrollfluß

Request

ControllerServlet.java

```
import ...;

public class ControllerServlet extends .. {
  public void doPost (req, res) ... {
    Action a = new MyAction ();
    a.process (req, res);
    req.getRequestDispatcher
      ("display.jsp").forward (req,res);
   }
}
```

MyAction.java

```
import ...;

public class MyAction implements Action {
  public void process (req, res) {
    MyBean bean = new MyBean ();
    // Request-Daten im Bean setzen
    new Backend().save(bean);
    res.putAttribute ("daten", bean);
  }
}
```

Backend
save (MyBean)

MyBean
getName()
getStrasse()
getPlz()
getOrt()

display.jsp

```
<%@ page import="myservlets.MyBean"%>
<HTML><BODY>
  <jsp:useBean id="daten"
       class="myservlets.MyBean"/>
  <H1>Gespeicherte Daten</H1>
  <P>Name: <%= daten.getName() %></P>
  <P>...</P>
</BODY>
</HTML>
```

Response

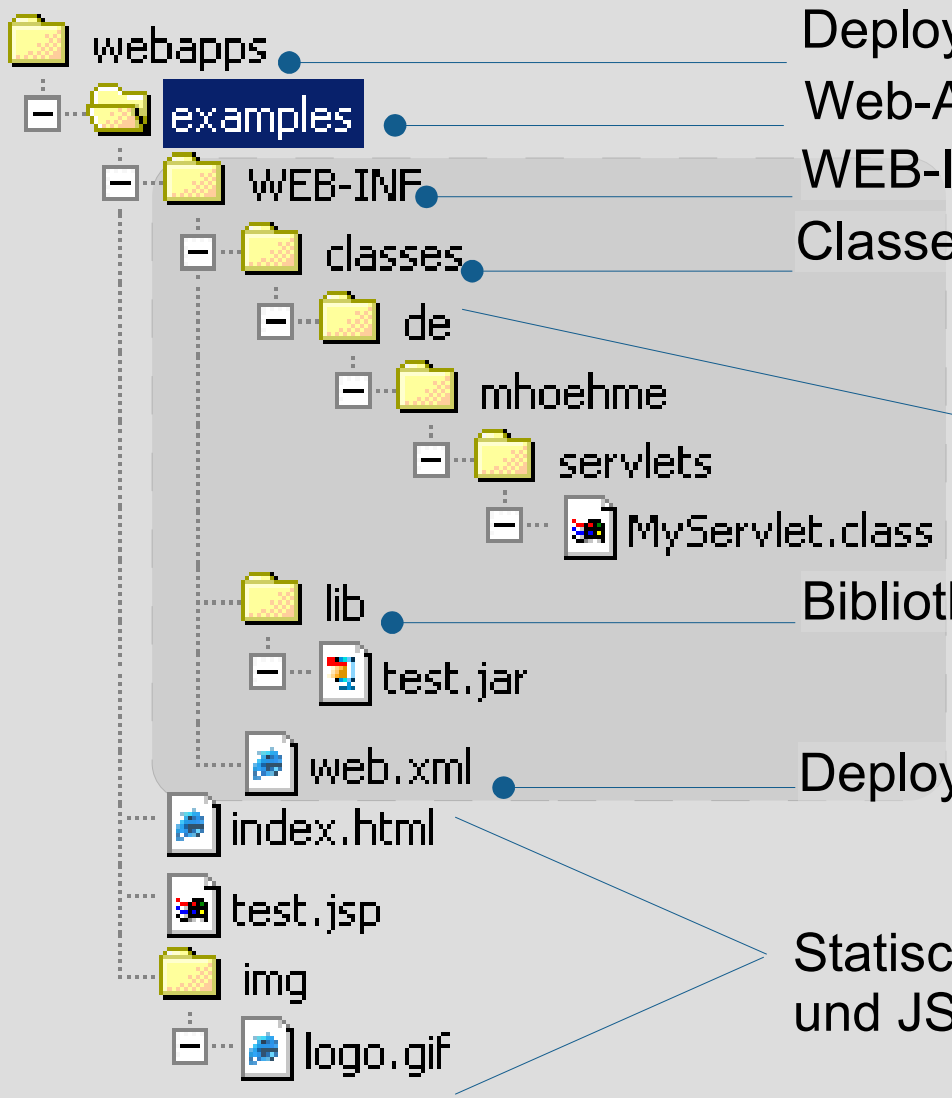
J2EE

- Warum J2EE lernen?
- Entwicklung des Internets
- J2EE-Konzepte
- Servlets und JSPs
- **Web-Anwendungen**

Web-Anwendungen

- bestehen aus vielen Dateien
  - statische Inhalte (HTML-Seiten, Grafiken)
  - dynamische Seiten (JSPs)
  - Servlets und anderen Class-Dateien
  - Bibliotheken (.jar-Dateien)
  - Deployment Descriptor (web.xml)
- vorgegebene Verzeichnis-Struktur
- Deployment im Web-Container

Verzeichnisstruktur



Deployment-Verzeichnis des Containers

Web-Anwendung „examples“

WEB-INF-Verzeichnis

Classes-Verzeichnis

Servlets und andere Klassen, die von Servlets genutzt werden

Bibliotheksverzeichnis

Deployment Descriptor

Statische Inhalte (HTML-Seiten, Grafiken) und JSPs

Web-Archive

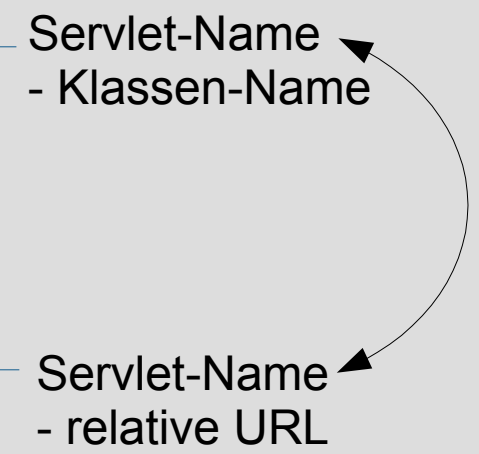
- Web-Applikationen werden in .war-Dateien „gepackt“
  - .war = Web Archive
  - ZIP-Format
  - z.B. examples.war

- Deployment
  - Das .war-Archiv wird in das Deployment-Verzeichnis des Containers kopiert
  - Servlet-Container deployt die Anwendung
    - beim Start des Containers
    - durch Laden der Anwendung mit dem Manager zur Laufzeit (ohne Neustart des Containers)

Deployment Descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<!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>
    <display-name>examples</display-name>
    <description>
        Eine Beispiel-Applikation
    </description>

    <servlet>
       <description>Ein Beispiel-Servlet</description>
       <servlet-name>MyServlet</servlet-name>
       <servlet-class>de.mhoehme.servlets.MySerlvet</servlet-class>
       <init-param>
          <param-name>param</param-name>
          <param-value>Beispiel-Servlet</param-value>
       </init-param>
    </servlet>

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

Beispiel für einen einfachen Deployment Descriptor



Mapping = Zuordnung

Frameworks

- The Jakarta Project
  - OpenSource-Software
- Tomcat
  - OpenSource-Web-Container
- Struts
  - De-Facto-Standard für interaktive Web-Anwendungen
    - Handling von Formulardaten
    - Tag-Libraries
    - Request-Dispatching (einfachere Wartung des Page-Flows)
- Ant
  - De-Facto-Standard für Build-Management

Struts

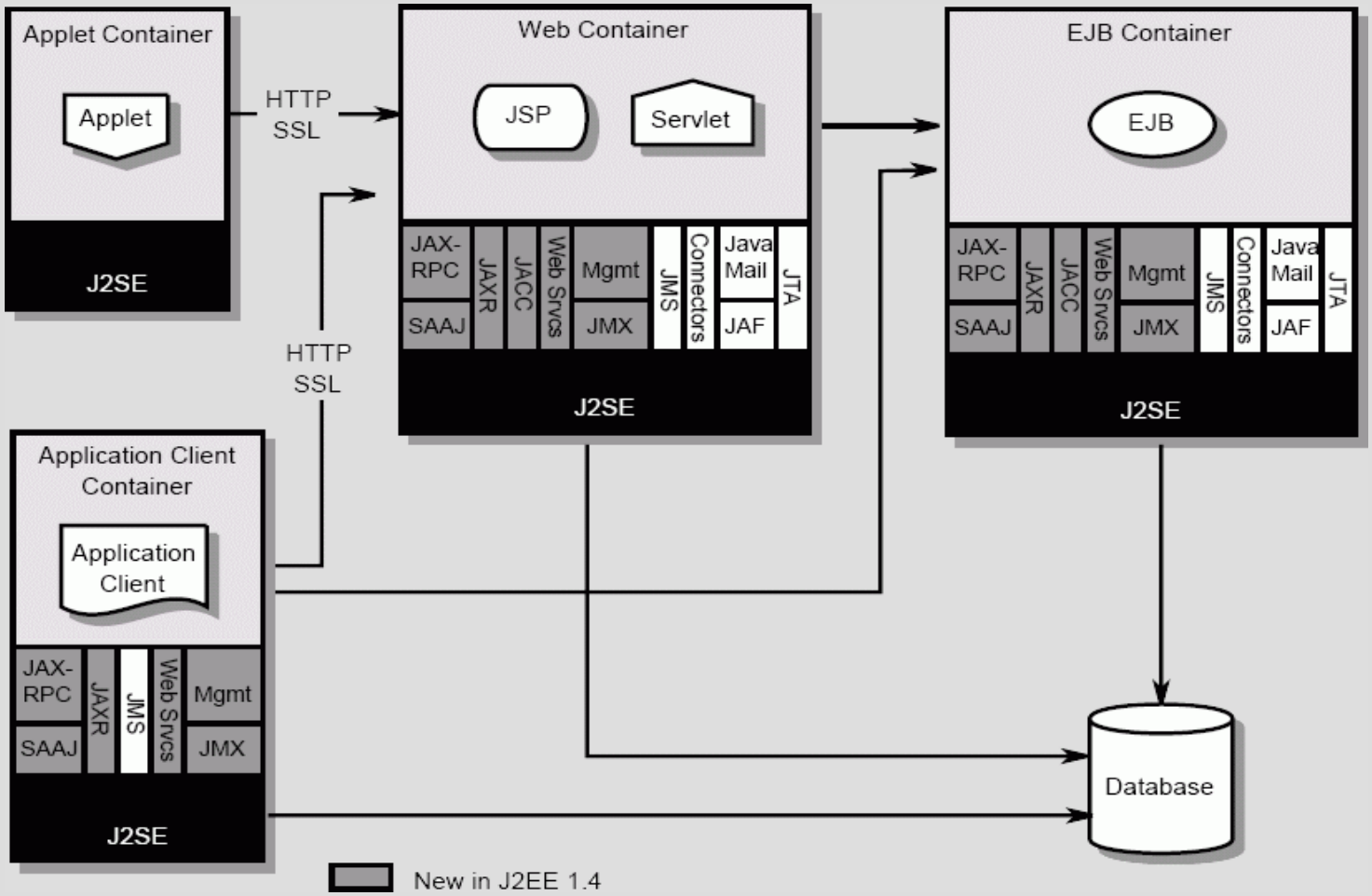
J2EE

- Warum J2EE lernen?
- Java und das Internet
- J2EE-Konzepte
- Servlets und JSPs
- Web-Anwendungen

J2EE

Weiterführendes

J2EE-Services



Quelle: J2EE-Spezifikation 1.4

Software

- J2SDK und J2EE
  - http://java.sun.com
- OpenSource-Software und -Frameworks
  - http://jakarta.apache.org
- Open-Source-Frameworks
  - http://sourceforge.net

Software

- J2EE-Container gibt es für jeden Zweck und jedes Budget

	JSPs	Servlets	EJB	Hinweis	URLs
Tomcat	X	X		Open Source; Referenz-Implementierung	jakarta.apache.org
Resin	X	X		Open Source	www.caucho.org
Allaire's JRun	X	X			www.allaire.com
JBoss			X	Open Source	www.jboss.org
IBM WebSphere	X	X	X		www.ibm.com
Bea Weblogic	(X)	(X)	X	Nicht für Servlets/JSPs optimiert	www.bea.com

Kommentierte Liste: http://www.servlets.com

Geschriebenes

- Jason Hunter with William Crawford, *Java Servlet Programming* (2nd Ed., O'Reilly) – auch in dt. Übersetzung
- Simon Brown et al., *Pro JSP* (Apress 2003)
- Spezifikationen
  - http://java.sun.com
- Tutorials
  - The J2EE 1.4 Tutorial
    - http://java.sun.com/j2ee/1.4/download.html

Online-Resourcen

- Diverse Online-Resourcen
  - http://www.theserverside.com
    - Serverseitige Programmierung, aktuelle Entwicklungen, Meinungen, Diskussionen
  - http://www.javaworld.com
    - Online-Magazin
  - http://saloon.javaranch.com
    - Foren, Zertifizierung
  - http://www.servlets.com
    - Web-Seite mit Übersicht über aktuelle Produkte von Jason Hunter
  - http://www.teamone.de
    - SelfHTML von Stefan Münz

J2EE

- Warum J2EE lernen?
- Entwicklung des Internets
- J2EE-Konzepte
- Servlets und JSPs
- Web-Anwendungen