Datenbanksysteme SS 2007

Frank Köster
(Oliver Vornberger)

Institut für Informatik
Universität Osnabrück

Kapitel 9b:
Datenbankapplikationen

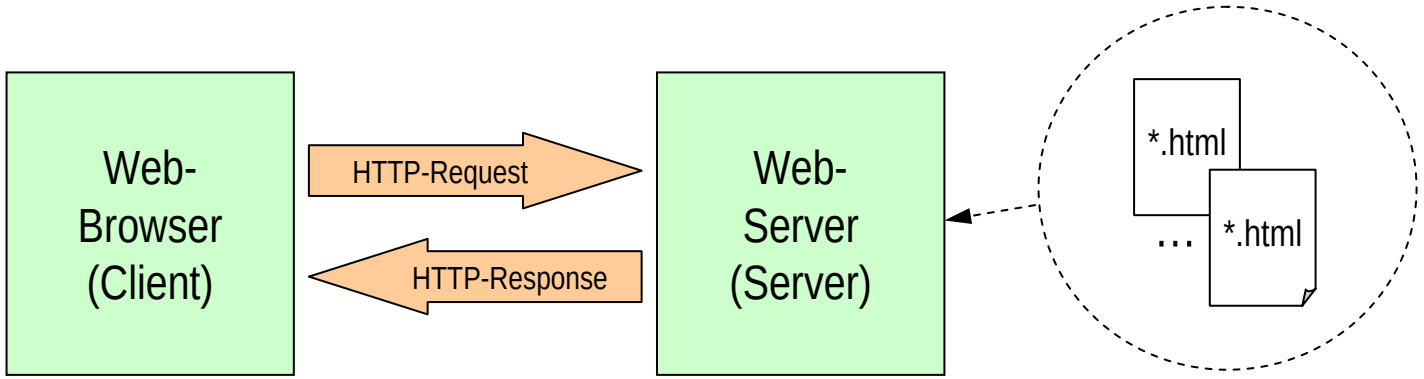
Statische Web-Seiten

Nutzung von HTML – Beispiel:

```
<!DOCTYPE HTML PUBLIC
          "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>&Uuml;bersicht zur heutigen Vorlesung</title>
    </head>
    <body>
        <h1>Gliederung</h1>
        <p>statische Web-Seiten (HTML)</p>
        <p>dynamische Web-Seiten (Servlets, Java Server
             Pages (JSP), Java Beans)</p>
        </p>
    </body>
</html>
```

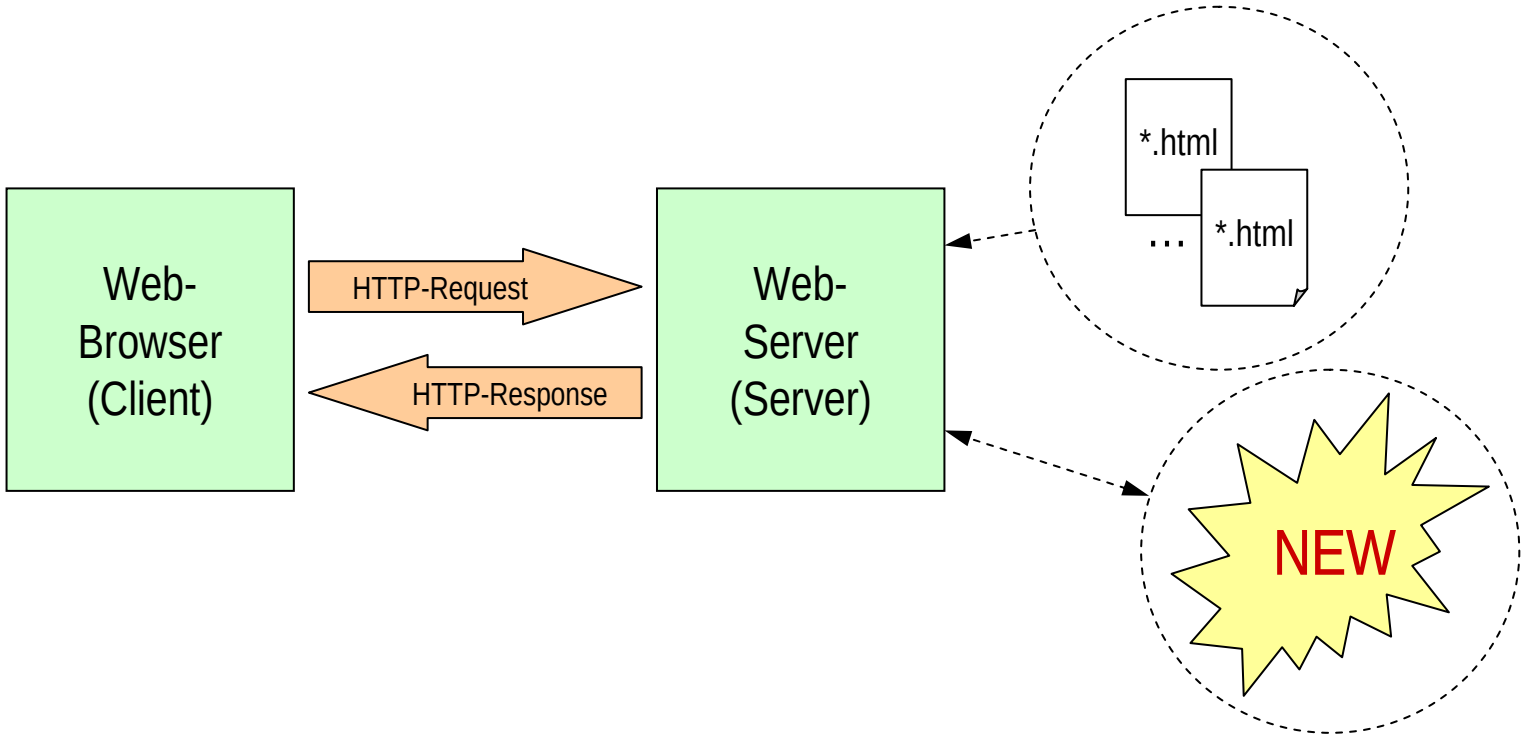

Statische Web-Seiten

Nutzung von HTML – Beispiel (Aufruf)



Statische und dynamische Web-Seiten

Nutzung von statischen und dynamischen Web-Seiten – Beispiel (Aufruf)



Statische und dynamische Web-Seiten

Nutzung von statischen und dynamischen Web-Seiten – Beispiel (Aufruf)

- HTTP-Request
  - URL

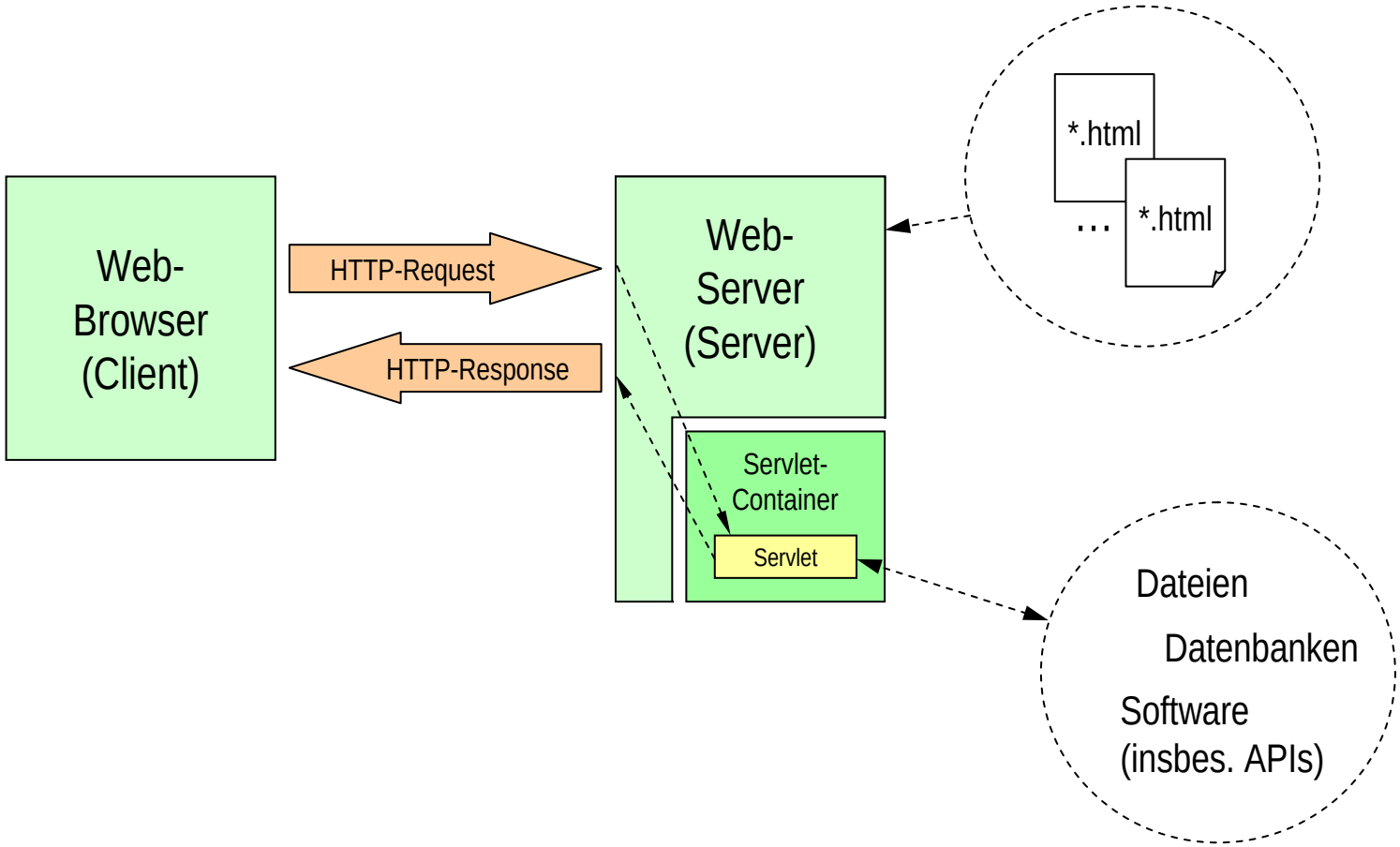
    Schema:http://server:port/pfad/seite?param1=wert1&param2=wert2

    Beispiel: http://localhost:8080/beispiel/addition?x=11&y=12
  - Methode (GET/POST/…)
  - Body (z.B.: Parameter bei POST, Datei-Uploads …)
  - Weitere Informationen (Header)

- HTTP-Response
  - Angeforderte Datei (HTML, PNG, GIF …)
  - Weitere Informationen (Header)

Servlets

Dynamische Web-Seiten mit Hilfe von Servlets



Statische und dynamische Web-Seiten

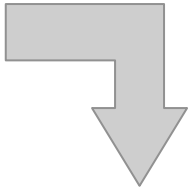
Aufruf eines Servlet

- HTTP-Request (Anfrage)
  - URL-Eingabe im Browser (GET)
  - Klick auf HTML-Link (GET)
  - Abschicken eines HTML-Formulars (GET/POST)
- HTTP-Response (Resultat)
  - Resultat wird durch Browser angezeigt
  - Evtl. ergeben sich automatisch weitere Requests (z.B. Bilder)
- Servlet wird in einem Servlet-Container (serverseitig) ausgeführt – z.B. TOMCAT

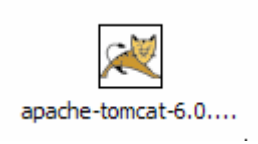
TOMCAT

Download und Installation

- Download
  - http://tomcat.apache.org/
  - apache-tomcat-6.0.13.exe

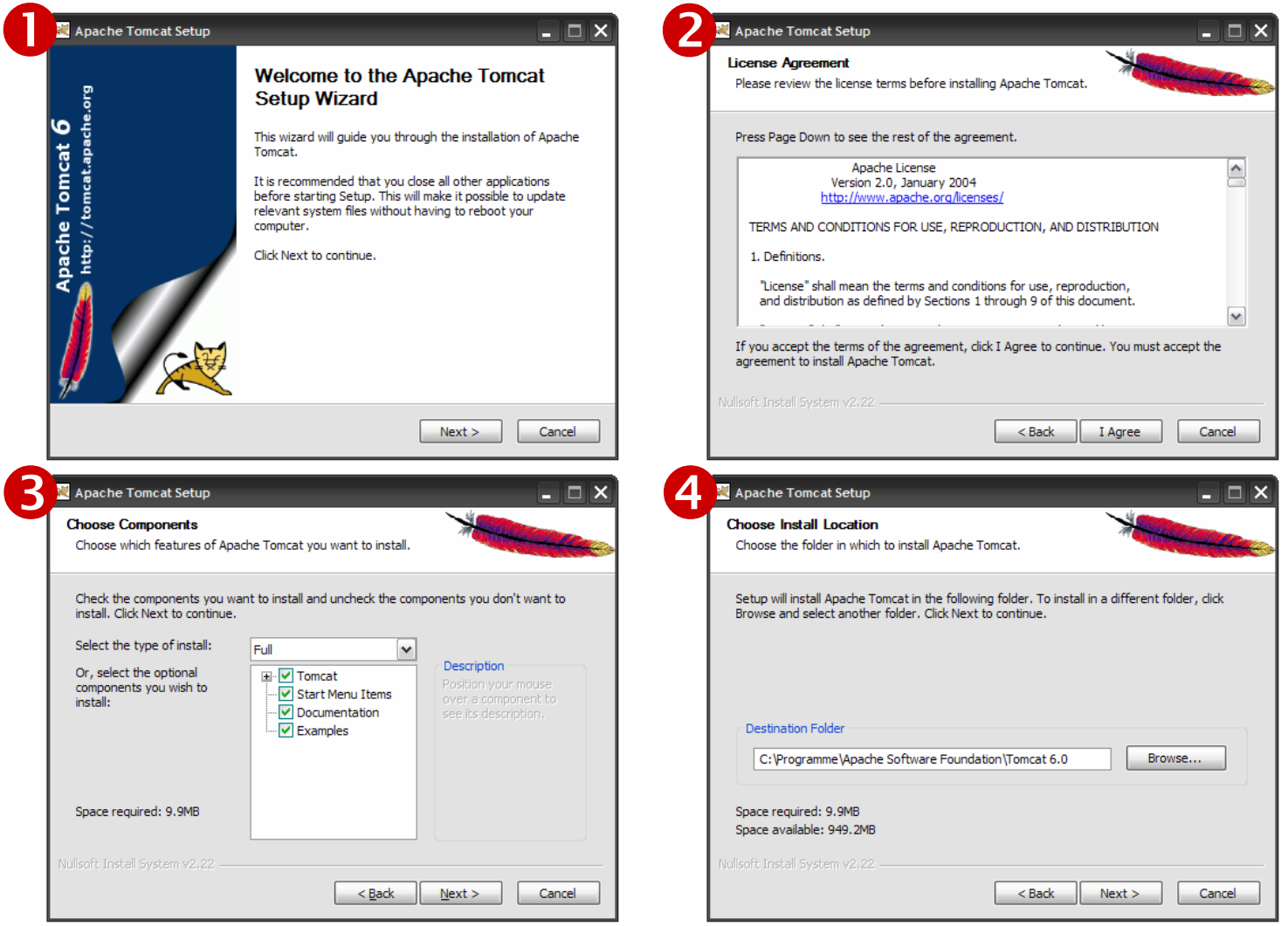


- Installation
  - Ausführen der Setup-Datei:



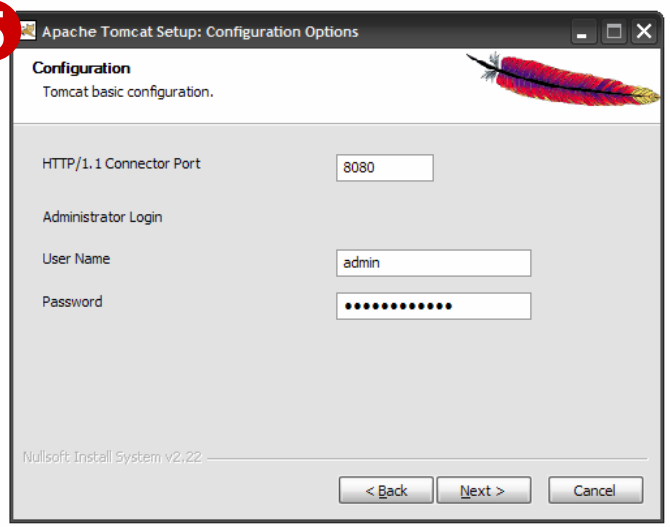
… durchlaufen der Installationsroutine.

Download und Installation

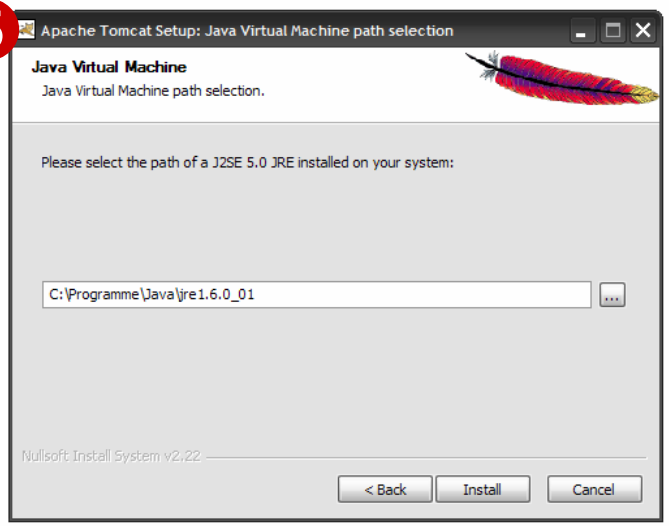


Download und Installation

5

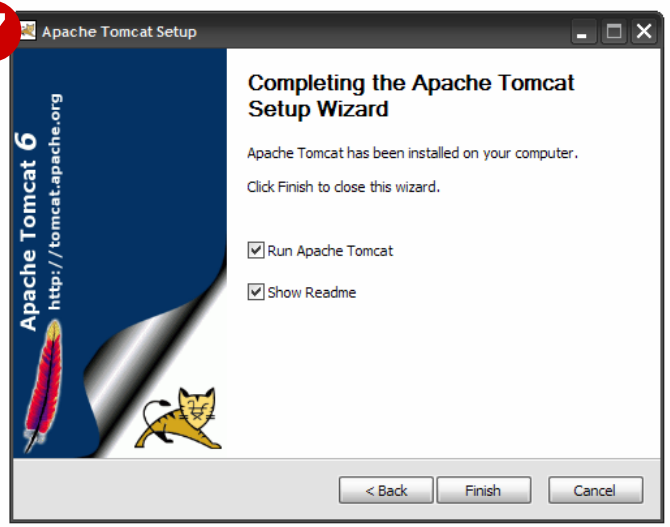


6



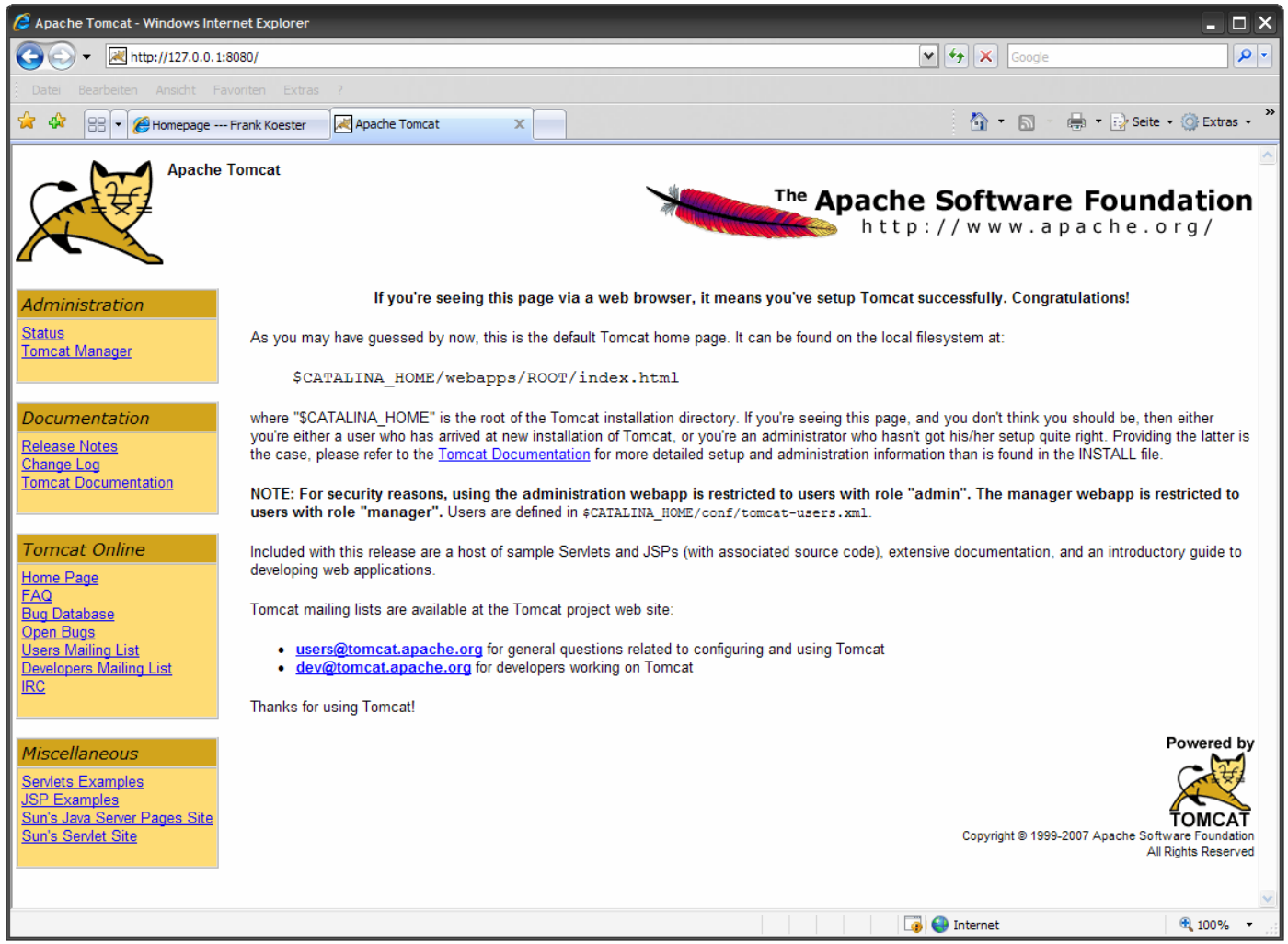
[...]

7



TOMCAT

Startseite



Anlegen einer Web-Applikation mit Servlets

Vorbemerkungen

Servlet-Anwendungen sind Web-Applikationen, die i.Allg. aus mehreren Servlets, sowie statischen HTML-Seiten, JSP-Seiten und JavaBeans(später mehr hierzu) bestehen können.

Um eine solche Web-Applikation portabel und einfach installierbar zu halten, werden alle Daten zu einer Web-Applikation nach einem einheitlichen Schema unterhalb eines speziellen Hauptverzeichnisses abgelegt.

Im Einzelnen verläuft das Anlegen einer Web-Applikation mit Servlets so:

1. Erstellen des Servlet (☺) …
2. Anlegen einer „standardisierten“ Verzeichnisstruktur unter **`<CATALINA_HOME>\webapps\`** (z.B. **`calculator\`**)
3. Vorbereitung der Registrierung durch Anlegen einer Datei **`web.xml`**
4. Start bzw. Neustart des TOMCAT.

Anlegen einer Web-Applikation mit Servlets

Erstellen des Servlet

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

public class CalculatorServlet extends HttpServlet
{
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
              throws ServletException, IOException
    {
        int x = Integer.parseInt(request.getParameter("x"));
        int y = Integer.parseInt(request.getParameter("y"));

        PrintWriter out = response.getWriter();

        out.println("<html><head><title>My Calculator ...</title></head>");
        out.println("<body>");
        out.println("Result: "+x+" + "+y+" = " + (x+y));
        out.println("</body>");
        out.println("</html>");
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
              throws ServletException, IOException
    {
        doGet(request, response);
    }
}
```

Anlegen einer Web-Applikation mit Servlets

Anlegen der Verzeichnisstruktur

Erstellen eines ersten Verzeichnisses unter dem bereits bestehenden Verzeichnis **<CATALINA_HOME>\webapps**(*). Für die im Folgenden betrachtete Web-Applikation ist dies **claculator**

(*) C:\Programme\Apache Software Foundation\Tomcat 6.0\webapps\

Erstellen der folgenden Verzeichnisstruktur unterhalb dieses Verzeichnisses:

…calculator\images
…calculator\WEB-INF
…calculator\WEB-INF\classes
…calculator\WEB-INF\lib
…calculator\WEB-INF\jsp

Die Servlet-Klassen werden ins Verzeichnis **classes** kopiert, die JSP-Seiten ins Verzeichnis **jsp**. Im Verzeichnis **lib** werden i.Allg. zusätzliche Java-Bibliotheken (in Form von jar-Files) zur Verfügung gestellt.

Anlegen einer Web-Applikation mit Servlets

Die Datei `web.xml`

Erstellen der Datei ...**`WEB-INF/web.xml`** mit wenigstens dem folgenden Inhalt (*kursive Kommentare* bitte ignorieren):

```
<?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>
  <servlet>
    <servlet-name>                 Tomcat interne ID des Servlets – muss pro Web-Application eindeutig sein!
      calculator
    </servlet-name>
    <servlet-class>                Name der Servlet-Klasse.
      CalculatorServlet
    </servlet-class>
  </servlet>
  <servlet-mapping>                Verknüpfung Servlet-URL
    <servlet-name>
      calculator                   Servlet ID
    </servlet-name>
    <url-pattern>                  URL Pattern unter dem das Servlet erreichbar ist (Wildcards sind möglich (*))
      /calculator-servlet
    </url-pattern>
  </servlet-mapping>
</web-app>
```


Anlegen einer Web-Applikation mit Servlets

Nutzung

Das Servlet ist unter

`http://`<*Rechnername*>**`:`**<*Port*>**`/`**<*Web-AppName*>**`/`**<*URL-Pattern*>

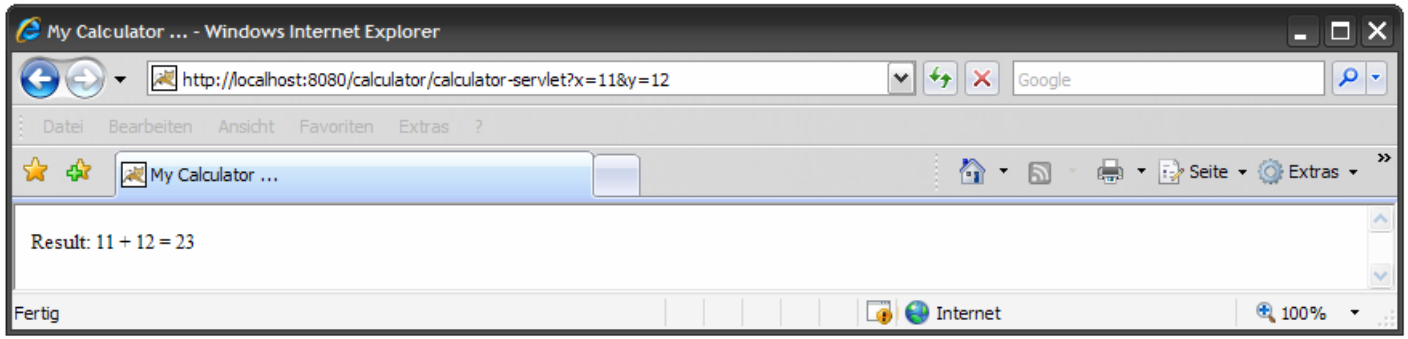
erreichbar.

Für jedes vom Benutzer aufrufbare Servlet muss es entsprechende Einträge in der **`web.xml`**-Datei geben.

Nutzung der Web-Applikation `calculator`

1. (und 2.) Aufrufmöglichkeit – direkter Aufruf

`http://localhost:8080/calculator/calculator-servlet?x=11&y=12`



Aufruf aus HTML-Seite über Link (gleiches Resultat):

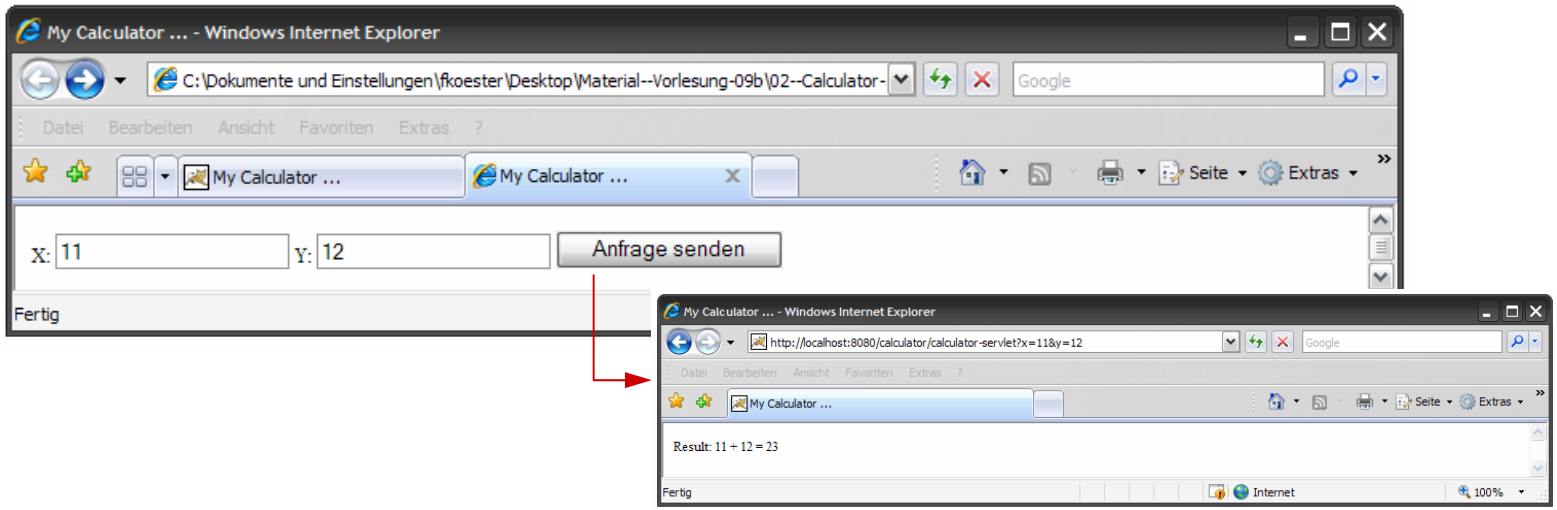
```
<html>
  <head>
    <title>My Calculator ...</title>
  </head>
  <body>
    <a href="http://localhost:8080/calculator/calculator-servlet?x=11&y=12">Berechnung ausführen ...</a>
  </body>
</html>
```

Nutzung der Web-Applikation `calculator`

3. Aufrufmöglichkeit – Formular

Aufruf über HTML-Formular:

```
<html>
  <head>
    <title>Calculator</title>
  </head>
  <body>
    <form action="http://localhost:8080/calculator/calculator-servlet">
      X: <input type="text" name="x">
      Y: <input type="text" name="y">
      <input type="submit">
    </form>
  </body>
</html>
```

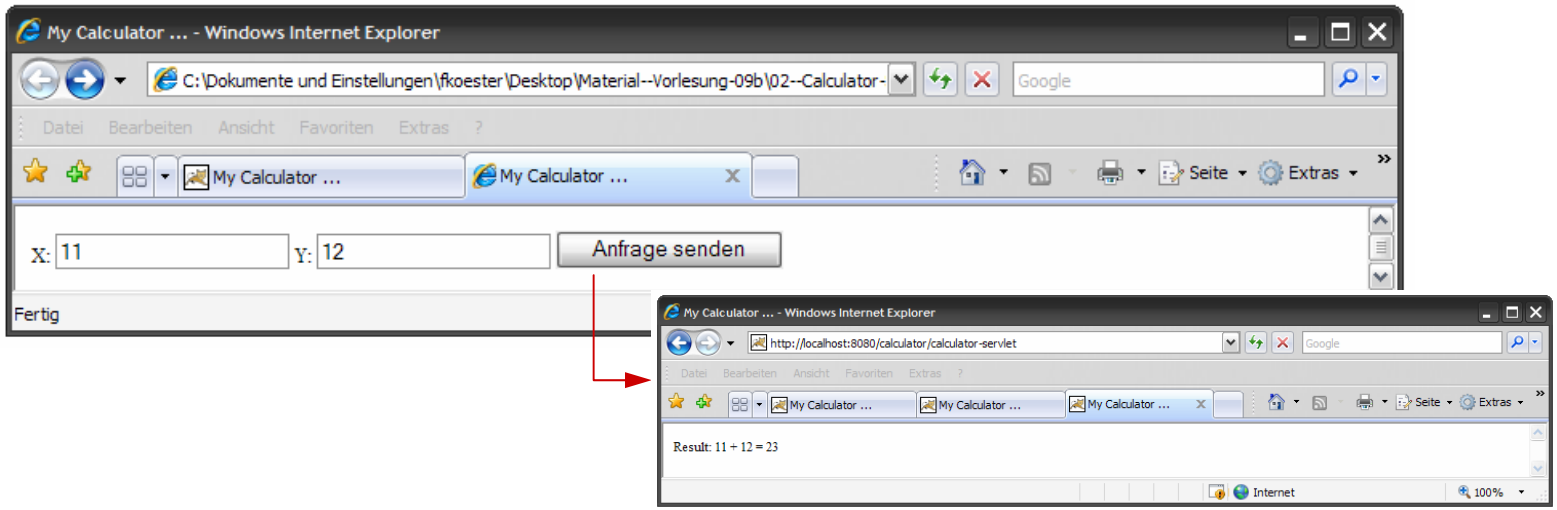


Nutzung der Web-Applikation `calculator`

4. Aufrufmöglichkeit – Formular (method="post")

Aufruf über HTML-Formular (method="post"):

```
<html>
  <head>
    <title>Calculator</title>
  </head>
  <body>
    <form action="http://localhost:8080/calculator/calculator-servlet" method="post">
      X: <input type="text" name="x">
      Y: <input type="text" name="y">
      <input type="submit">
    </form>
  </body>
</html>
```



Nutzung von Servlets – Hinweis

Sessionverwaltung

- Problem
  - Verschiedene Nutzer kommunizieren zur gleichen Zeit mit Server
  - Frage: Welche Requests gehören zu welchem Nutzer?

- Unterstützung bei Problemlösung durch Servlet-Engine

```
// Session holen bzw. erzeugen
HttpSession session = request.getSession()

// Objekt in Session speichern
session.setAttribute("user", aUser);

// Objekt aus Session lesen
User curUser = (User) session.getAttribute("user");
```

Nutzung von Servlets – Hinweis

Sessionverwaltung – verschiedene Implementierungen

- Verschiedene Implementierungen
  - Cookies: Client-seitig wird eine eindeutige Session-ID gespeichert.
  - URL-Rewriting: Session-ID wird in alle URLs kodiert.

- Beachte: Cookies können deaktiviert sein

  ➲ Servlet-Engine wählt automatisch richtige Implementierung.

  ➲ Session-Verwaltung sollte immer auch ohne Cookies funktionieren.

```
out.println("<a href=\"");
out.println(response.encodeURL("/servlet/abc"));
out.println("\">text</a>");
```

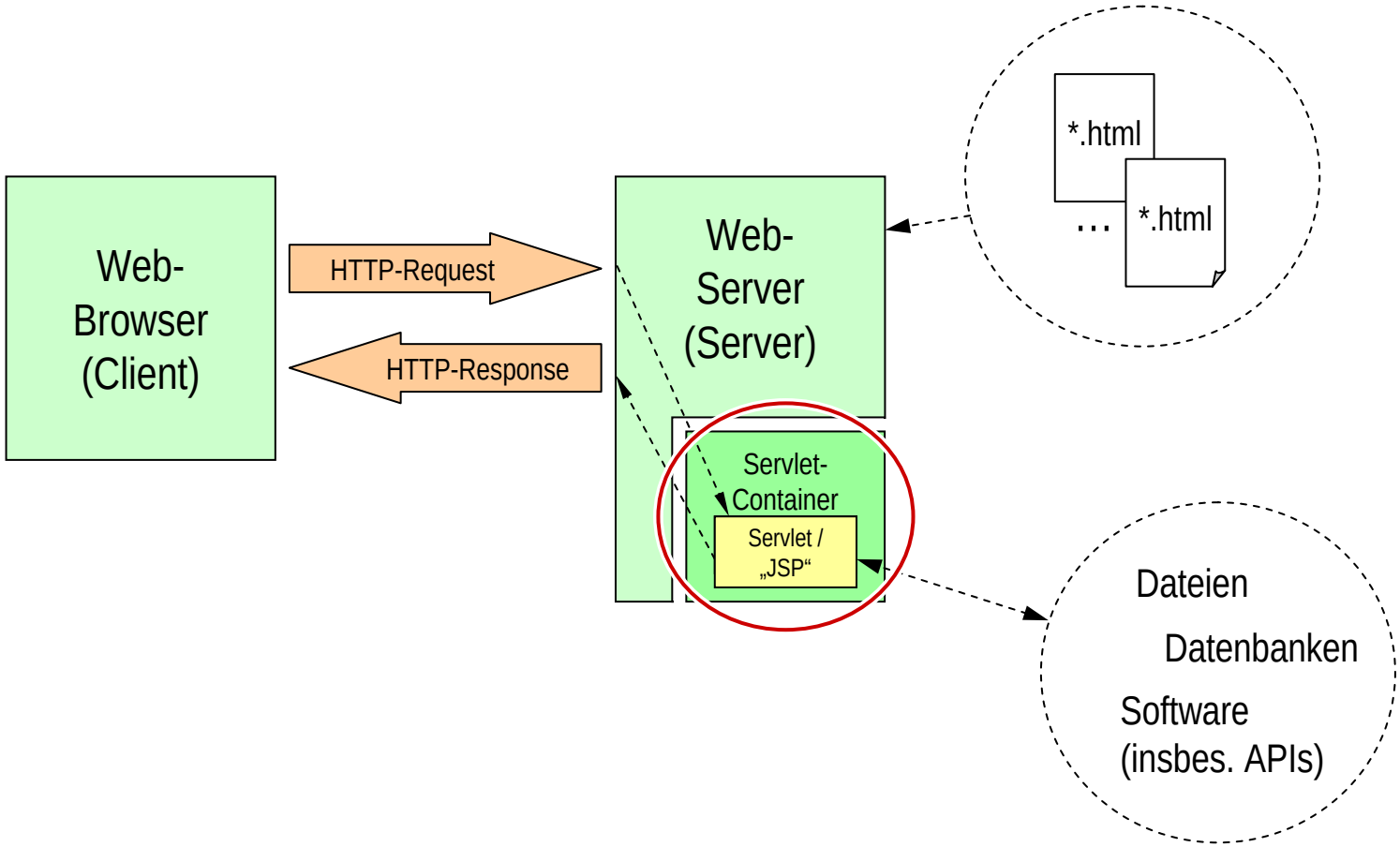

Bewertung von Servlets

Bewertung von Servlets und Motivation von Java Server Pages (JSPs)

- Vorteil von Servlets: Extrem flexibel
- Nachteil von Servlets: HTML ist in Java-Code eingebettet
  - unübersichtlich
  - HTML-Editoren nicht nutzbar (damit z.B. kein Syntax-Highlighting von HTML-Code …)
- Alternative: Java Server Pages

JSPs

Dynamische Web-Seiten mit Hilfe von JSPs



calculator-Beispiel als JSP

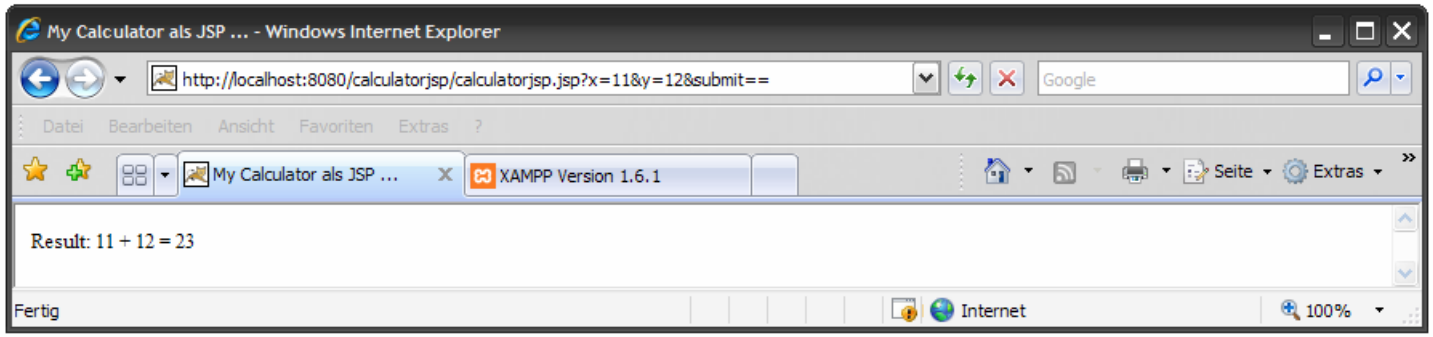
JSP-Realisierung des calculators

```
<html>
<head>
  <title>My Calculator als JSP ...</title>
</head>
<body>
<%@ page import = "java.util.*" %>
<%
  if (request.getParameter("submit") != null) {
    out.println("Result: " + request.getParameter("x") + " + "
                           + request.getParameter("y") + " = ");
    out.println(Integer.parseInt(request.getParameter("x")) +
                Integer.parseInt(request.getParameter("y")));
    out.println("<br>");
  } else {
%>

<H2>Additionsaufgabe:</H2>
<form action="calculatorjsp.jsp" method="post">
  <input type=text name="x" size=10> +
  <input type=text name="y" size=10>
  <input type=submit name="submit" value="=">
</form>

<%
  }    //end if
%>
</body>
</html>
```

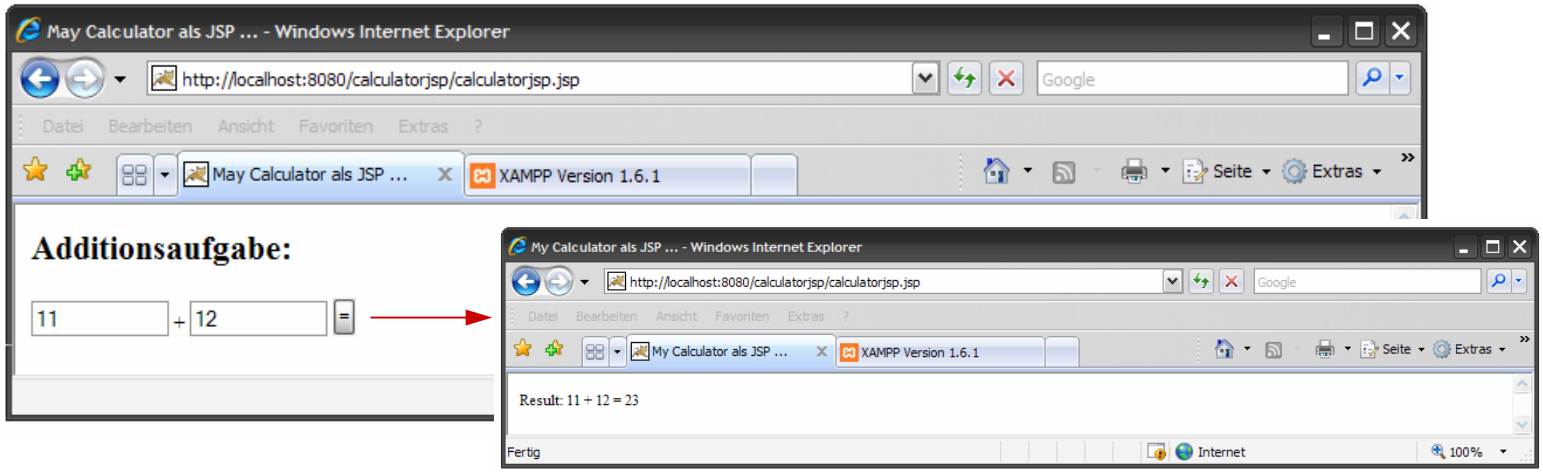

Nutzung der JSP `calculatorjsp.jsp`

1. (und 2.) Aufrufmöglichkeit

`http://localhost:8080/calculatorjsp/calculatorjsp.jsp?x=11&y=12&submit==`



`http://localhost:8080/calculatorjsp/calculatorjsp.jsp`



Syntax zur Implementierung von JSPs (Fragmente)

Programmierung von JSPs

<%= *Java Expression* **%>**

```
2 + 3 = <%= 2 + 3 %>
```

<%-- *Kommentar* **--%>**

```
<%-- JSP Kommentar --%>
<!-- HTML Kommentar -->
```

<%@ *Direktive* **%>**

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

<%! *Funktions- und Variablen-Deklarationen* **%>**

```
<%! static final int PI = 3.1415926536;
    double u(double r) { return 2*PI*r; } %>
```

Umfang des Kreises mit Radius 1.0: **<%= u(1.0) %>**

Syntax zur Implementierung von JSPs (Fragmente)

Programmierung von JSPs

<% *beliebiger Java Code („Skriptlet“)* **%>**

```
<% int i = 10;
   for(int j=0; j<i; ++j) { out.println(i); } %>
```

Unterschied zu **<%=** **…** **%>**: Ergebnis wird nicht ausgegeben

Unterschied zu **<%!** **…** **%>**: Variablen sind lokal

Vordefinierte Variablen (Auswahl)

```
ServletRequest request
ServletResponse response
HttpSession session
JspWriter out
…
```

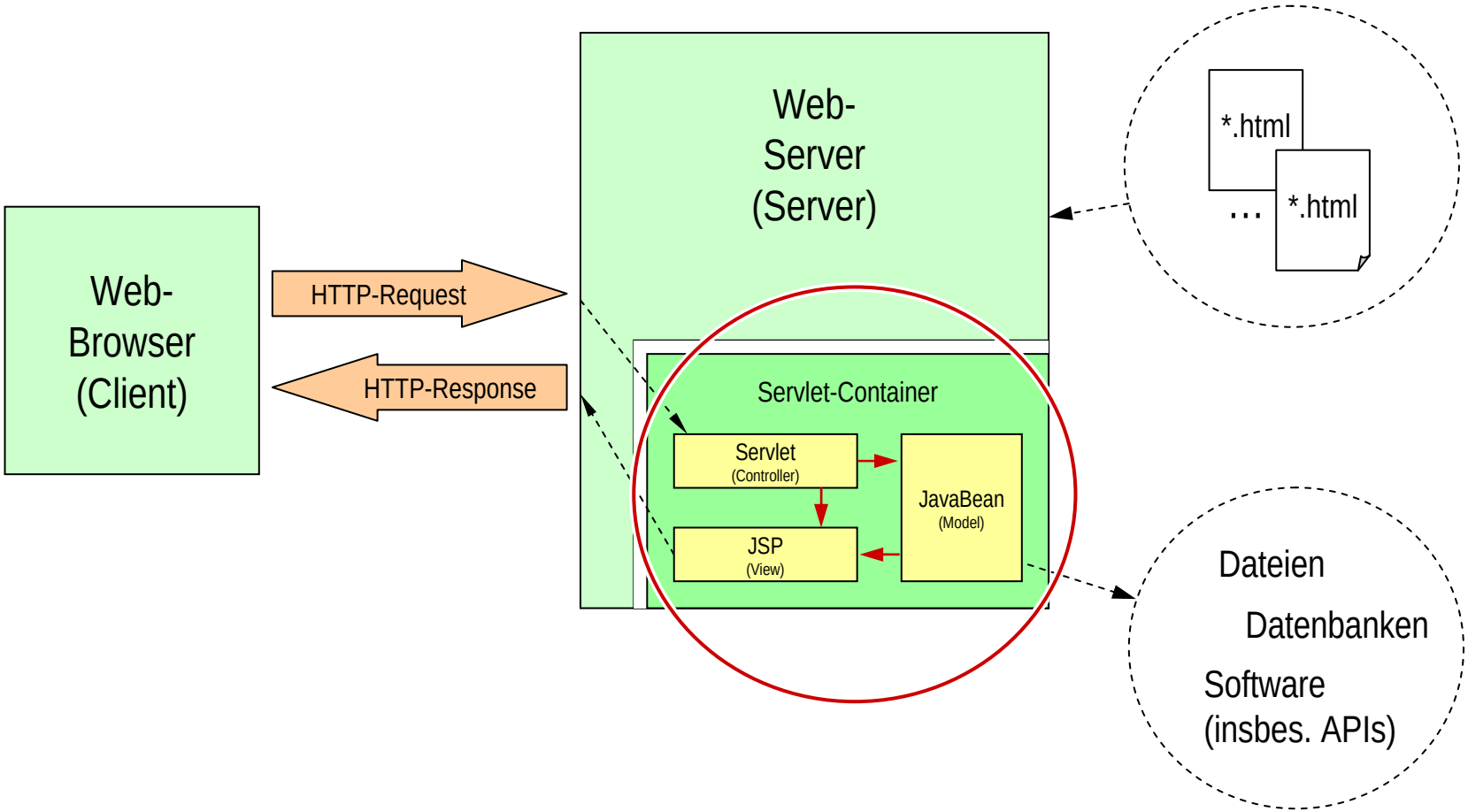

Bewertung von JSPs

Verarbeitung der JSPs am Server und ihr praktischer Wert

- JSPs werden zu Servlets kompiliert
  - *.jsp (JSP-Quellcode) ➲ *.java (Servlet-Quellcode) ➲ *.class
- Erster Schritt zur Trennung von Logik und Layout
  - Servlet: HTML eingebettet in Java
  - JSP: Java eingebettet in HTML
- In der Praxis
  - Servlets (und JavaBeans) für Logik
  - JSPs für Anzeige

Model View Controller (MVC)

Servlets, JSPs und JavaBeans als Grundlage zur MVC-Umsetzung



MVC – Grundlagen

Model – View – Controller

- Model (JavaBeans)
  - Enthalten die eigentliche Anwendungslogik
  - Unabhängig von der Web-Schnittstelle (Aufruf / Ergebnisdarstellung)
    ➲ insbesondere kein Import von **`javax.servlet.*`**
- View (JSPs)
  - Darstellung des Resultats
  - Beachte: Möglichst wenig Java-Code in JSPs integrieren
- Controller (Servlet)
  - Einlesen / Überprüfen der Parameter
  - Aufrufen der Anwendungslogik im Model (JavaBean)
  - Weitergabe des Ergebnisses an passende View

Model (JavaBean)

Calculator-Bean

```
package fk;

public class Calculator
{
    protected int x;
    public void setX(int x) { this.x = x;     }
    public int getX()       { return x;       }

    protected int y;
    public void setY(int y) { this.y = y;     }
    public int getY()       { return y;       }

    public void calculate() { result = x + y; }

    protected int result;
    public int getResult()  { return result;  }
}
```


Controller (Servlet)

ControllerServlet

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

public class ControllerServlet extends HttpServlet
{
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
        throws ServletException, IOException
    {
        int x = Integer.parseInt(request.getParameter("x"));
        int y = Integer.parseInt(request.getParameter("y"));

        fk.Calculator c = new fk.Calculator();
        c.setX(x);
        c.setY(y);
        c.calculate();

        request.setAttribute("calculator", c);
        RequestDispatcher disp =
          request.getRequestDispatcher("/WEB-INF/jsp/view.jsp");
        disp.forward(request, response);
    }
}
```

View (JSP)

JSP-View

```
<%@ page import="fk.*" %>

<jsp:useBean id="calculator" scope="request"
             type="fk.Calculator"/>

<html>
  <head>
    <title>My Calculator ...</title>
  </head>
  <body>
    Result:
    <jsp:getProperty name="calculator" property="x"/> +
    <jsp:getProperty name="calculator" property="y"/> =
    <jsp:getProperty name="calculator" property="result"/>
  </body>
</html>
```


web.xml

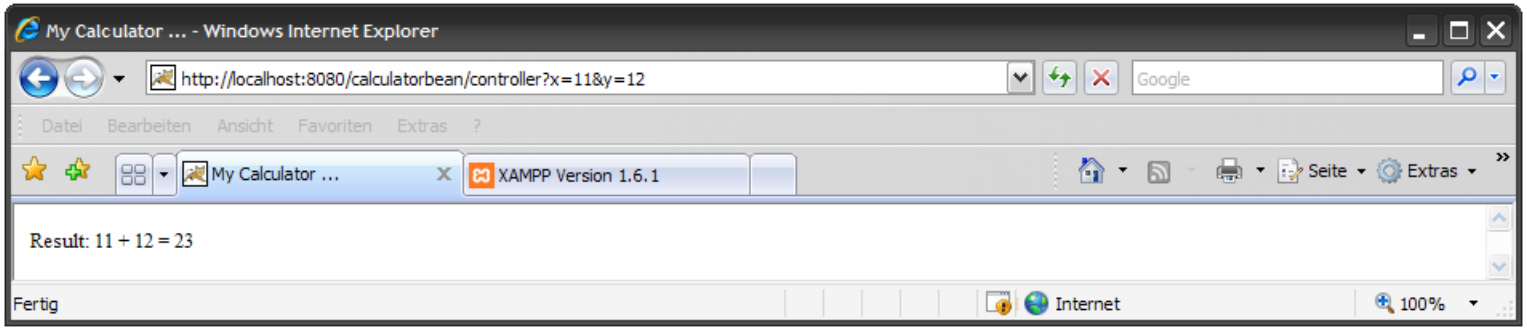
web.xml

```
<?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>

<servlet>
  <servlet-name>controller</servlet-name>
  <servlet-class>ControllerServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>controller</servlet-name>
  <url-pattern>/controller</url-pattern>
</servlet-mapping>

</web-app>
```


…\webapps\

Verzeichnisstruktur … und direkte Nutzung der Web-Applikation

```
webapps\
  images\
  WEB-INF\
    classes\
      fk\
        Calculator.class
      ControllerServlet.class
    jsp\
      view.jsp
    lib\
    web.xml
```



… und wo bleibt die DB?!

Erweiterung der Calculator-Bean (und der Datenbank dbs07) (1/2)

DB-Tabelle speichert die Resultate der **`clatulator`**-Berechnung:

```
create table add_values
    ( ID integer auto_increment not null,
      value integer,
      primary key (ID) );
```

Implementierung des DB-Zugriffs über JDBC …

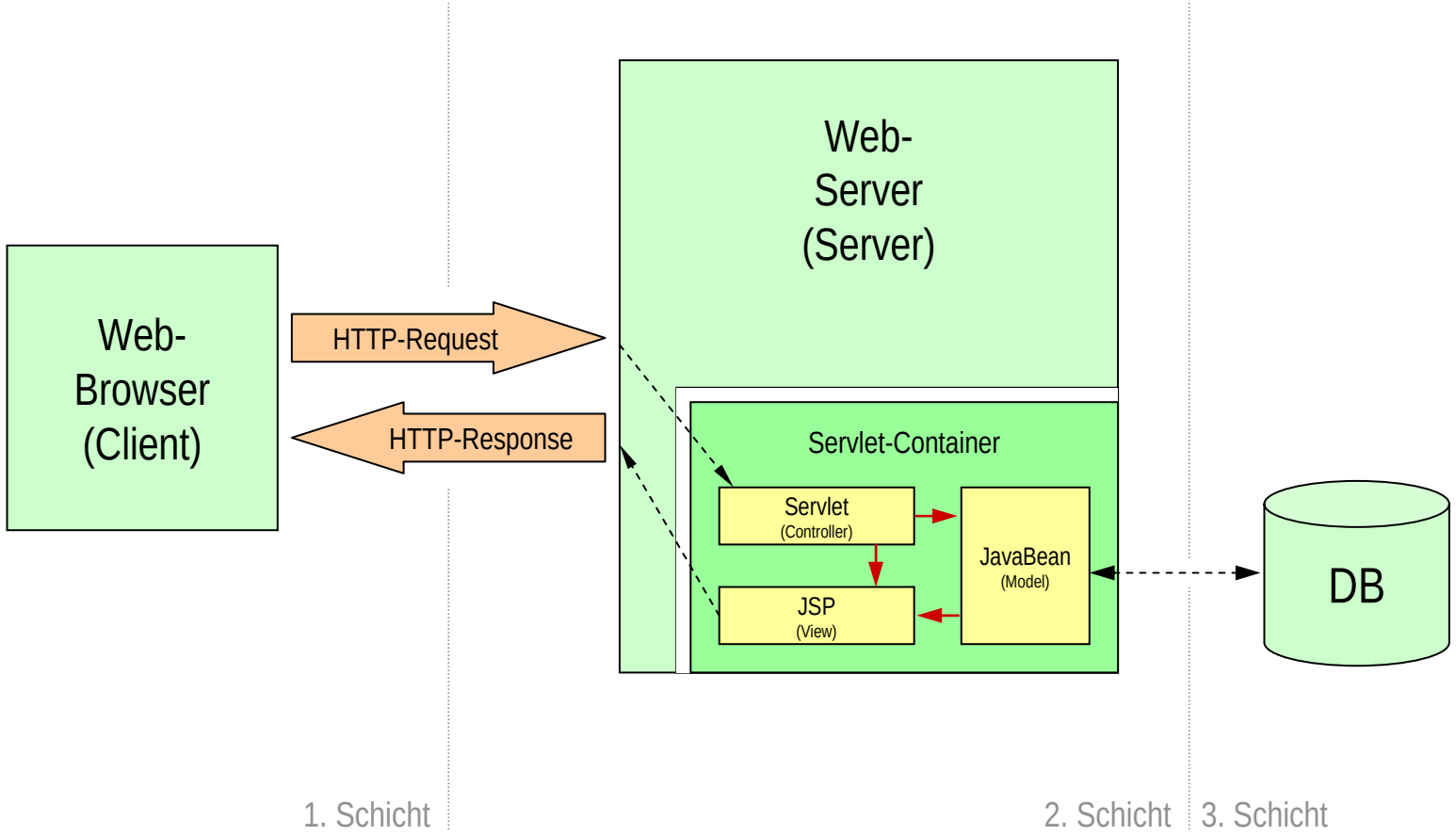
… und wo bleibt die DB?!

Erweiterung der Calculator-Bean (2/2)

```
package fk;
import java.sql.*;                    // Zugriffsmoeglichkeiten auf die Datenbank ...

public class CalculatorDB {
    protected int x; public void setX(int x) { this.x = x; } public int getX() { return x; }
    protected int y; public void setY(int y) { this.y = y; } public int getY() { return y; }

   public void calculate() {
        result = x + y;

        System.out.println ("Laden des MySQL-Treibers (JDBC) ...");
        try {
            Class.forName ("com.mysql.jdbc.Driver").newInstance();
        } catch (Exception e) {
            System.out.println ("Treiber konnte nicht geladen werden!");
              e.printStackTrace();
        }
        System.out.println ("Laden des Treibers erfolgreich!");
        String userURL = "jdbc:mysql://localhost/dbs07";
        try {
              Connection db; Statement  st;
              db = DriverManager.getConnection(userURL,"frank","frank");
              st = db.createStatement();
              String query="insert into add_values (value) values (" + result + ");";
              st.executeUpdate(query);
              st.close(); db.close();
        } catch (SQLException e) {
              System.out.println ("Fehler bei Ausfuehrung!");
              e.printStackTrace();
              System.out.println("SQLException: " + e.getMessage());
              System.out.println("SQLState: " + e.getSQLState());
              System.out.println("VendorError: " + e.getErrorCode());
        }
     }

    protected int result; public int getResult() { return result; }
}
```

Architektur der Web-Applikation mit DB

Servlets, JSPs und JavaBeans als Grundlage zur MVC-Umsetzung



Datenbanksysteme SS 2007

Ende von Kapitel 9b:
Datenbankapplikationen