FH Bielefeld
University of
Applied Sciences

Campus Minden

Webbasierte Anwendungen

SS 2018

Serverseitige Anwendungen

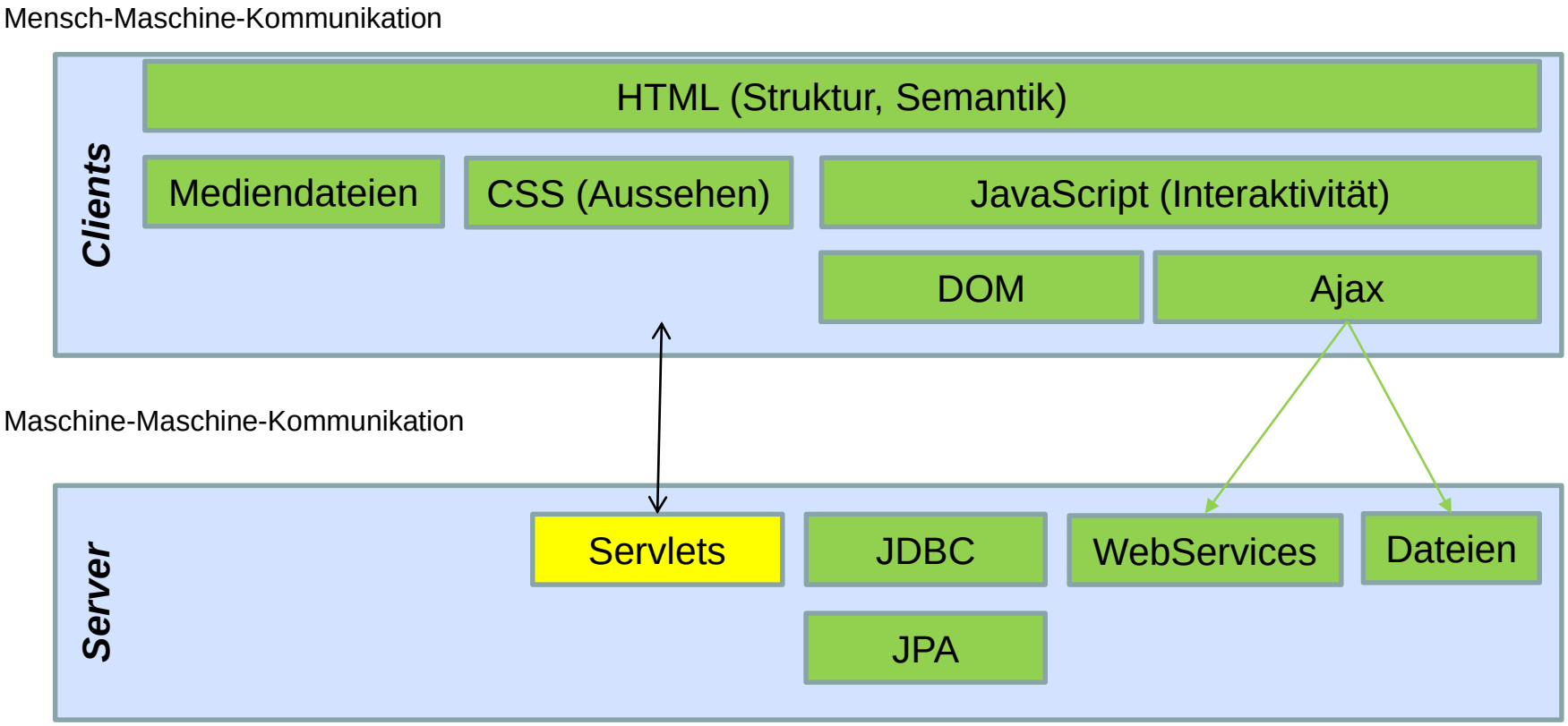
Dozent: B. Sc. Florian Fehring
mailto: florian.fehring@fh-bielefeld.de

Studiengang Informatik

Serverseitige Anwendungen

1. **Kontext und Motivation**
2. Webserver Interfaces
3. Servlets
4. JSP
5. Darüber hinaus
6. Projekt

Problemfelder



Anforderungen

Welche Anforderungen werden als nächstes bearbeitet?

TODO

- Kommunikation untereinander

DONE

- …
- Formular für Kommentare
- Schickes Design für die Seite
- Mediendatein einbinden
- Animationen
- Mehrsprachen-Fähigkeit
- (lokales) Speichern von Artikeln
- Client-Position anzeigen
- Offline-Verwendung ermöglichen
- Inhaltsverzeichnisse
- Formlareingaben in Seite einfügen
- Navigation über Tastaturkürzel
- Externe Inhalte einbinden
- Artikel vom Server einbinden
- Kommentare vom Server
- Medien hochladen
- Kommentare hochladen
- Kommentare speichern

Serverseitige Anwendungen

1. Kontext und Motivation
2. **Webserver Interfaces**
3. Servlets
4. JSP
5. Darüber hinaus
6. Projekt

WebServer Interfaces

Definition: **Ein WebServer Interface ist die Verbindungsstelle zwischen dem WebServer und einer Programmausführung**

Eigenschaften:

- Die Server nehmen Anfragen von Clients entgegen und reichen die Parameter an das Programm weiter
- Die Server starten die Programme und geben ihnen Informationen über die Umgebung mit.
- Die Server nehmen die Antworten der Programme entgegen und verpacken sie in HTTP-Antworten.

Arten:

- Common Gateway Interface (CGI)
- (Java) Servlets

WebServer Interfaces II - CGI

Definition: **Das Common Gateway Interface (CGI) ist ein Standard für den Datenaustausch zwischen Server und nativer Anwendung**

Eigenschaften:

- Ausführung nativer Programme auf dem Server
  - Programme sind plattformabhängig
  - Direkte Ausführung der Programme
- Programme werden als eigener Betriebssystemprozess ausgeführt
  - Benötigte Ressourcen müssen separat geöffnet werden
- Kein Datenaustausch unter Anwendungen
  - Datenaustausch nur über Dateien oder Datenbank
- Keine direkte Manipulation von HTTP-Headern
  - Kein direktes Auslesen oder Setzen von Cookies

Durch die Verwendung von Skriptsprachen besteht eine gewisse Plattformunabhängigkeit. Dafür müssen die Skripte zur Laufzeit geparst werden.

Serverseitige Anwendungen

1. Kontext und Motivation
2. Webserver Interfaces
3. **Servlets**
4. JSP
5. Darüber hinaus
6. Projekt

Servlets

Definition: **Servlets sind Servererweiterungen (serverseitige Java-Komponenten). Sie antworten auf ganz Anfragen eines Clients mit einem dedizierten dynamisch erzeugten Inhalt.**

Eigenschaften:

- Ausführung von Java-Programmen auf dem Server
  - Programme sind plattformunabhängig (sowohl Server- als auch Betriebssystem-bezogen)
  - Ausführung über die JavaVirtualMachine
- Programme werden als JVM-Thread ausgeführt
  - Leichtgewichtiger als Systemprozesse
  - Ressourcen können von mehreren Anwendungen gleichzeitig genutzt werden
- Datenaustausch unter Anwendungen
  - Anwendungen können Daten untereinander austauschen
- Direkte Manipulation von HTTP-Headern
  - Aus Servlets heraus ist eine Manipulation von HTTP-Headern möglich

Beispiele der Vorteile:

- Nutzen eines Datenbank-Verbindungs-Pools für effizienteren Datenbankzugriff
- Direktes Arbeiten mit Cookies und Anfrageparametern wie dem gewünschten Datentyp

Servlets

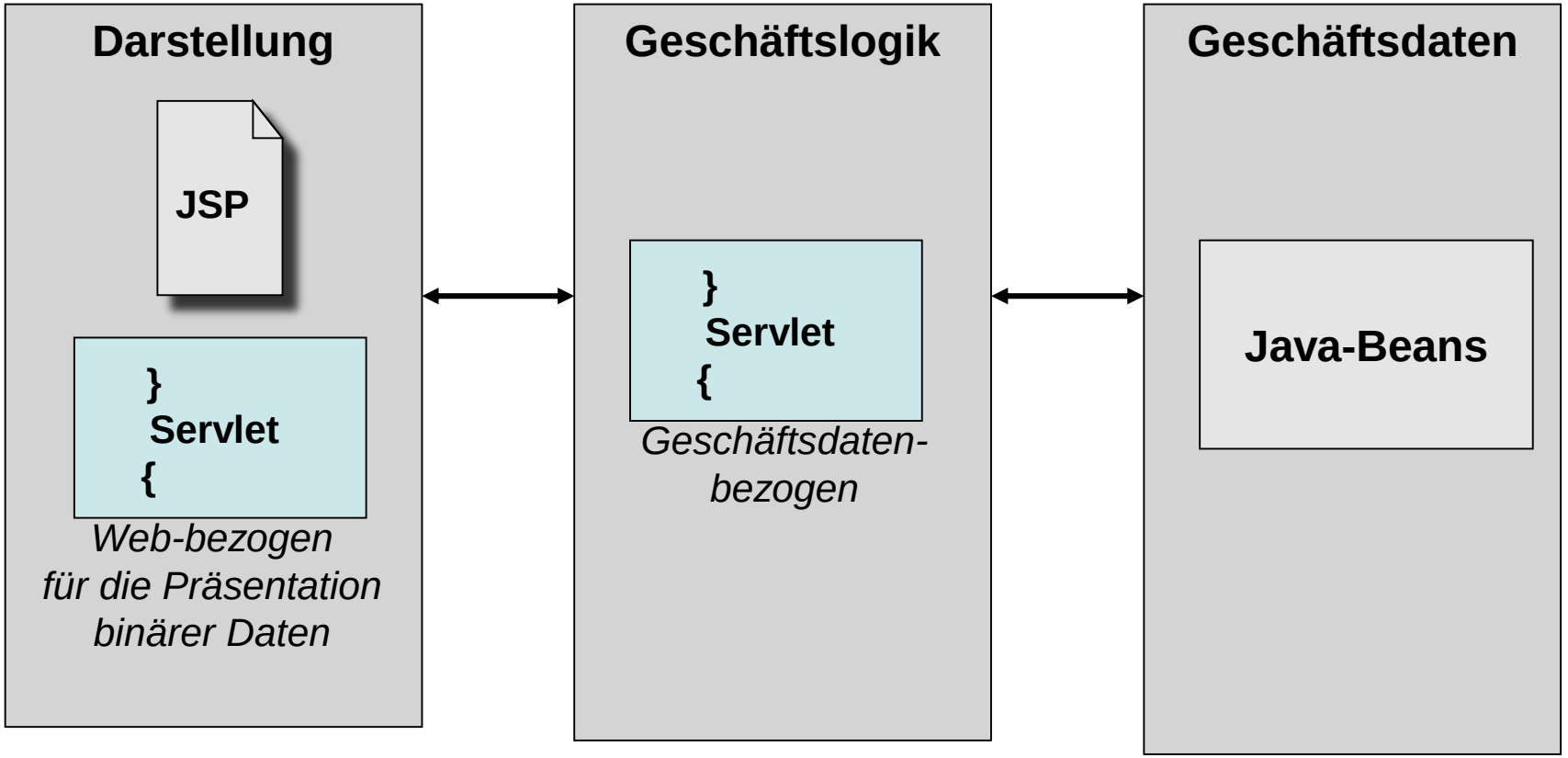
Eigenschaften:

- Leistungsfähigkeit
  - Es stehen alle Leistungsmerkmale und Bibliotheken der Sprache Java zur Verfügung.
- Effizienz
  - Nach dem Laden verbleibt ein Servlet - Objekt im Speicher des Webservers, sein Zustand erhalten. Das ermöglicht über Sessionhandling ein Ausgleichen des zustandslosen HTTP-Protokolls.
- Sicherheit
  - Robustheit resultiert aus Sprachimplementierung und Fehlerbehandlung auf Seiten des Webservers, um Serverabstürze zu verhindern.
  - Webserver stellt Java Security Manager zur Verfügung.
- Einfachheit
  - Servlet-API ist einfach und übersichtlich und enthält Features, die die Entwicklung vereinfachen

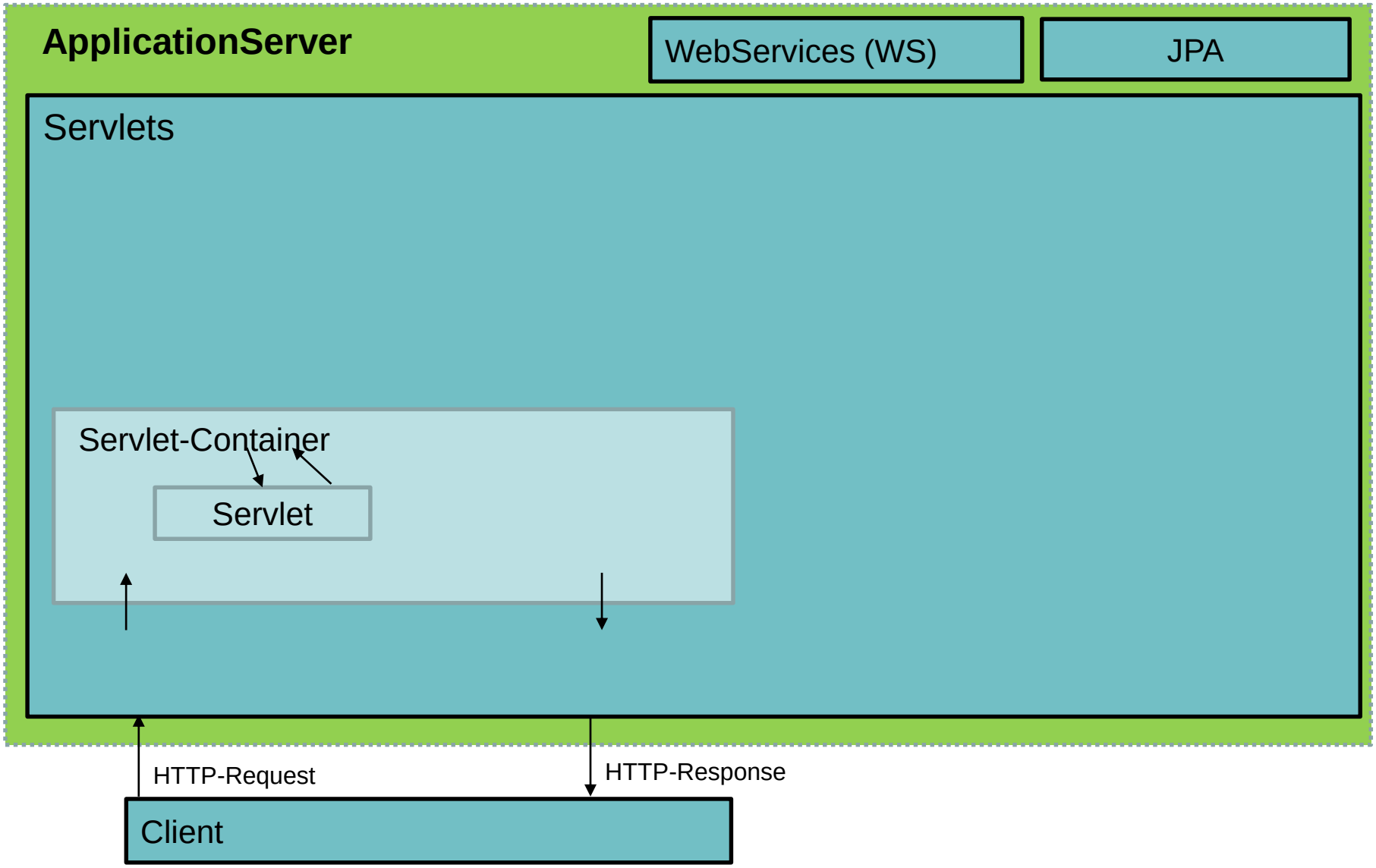
Der Begriff „Servlet“ (engl.) wird nicht übersetzt. Er ist eine Wortkreation aus den Begriffen „Server“ und „Applet“, (serverseitiges Applet) und bedeutet: „*kleine Serveranwendung*“

Servlets II – Bezug

Bezug von Servlets zum Web oder zu den Geschäftsdaten



Servlets II - ApplicationServer



Servlets III – Servlet-Container

Definition: **Servlets werden in einem Servlet-Container ausgeführt**

Aufgaben des Servlet-Containers:

- Kommunikation zwischen Server und Servlet
- Lifecycle – Management der Komponenten
- Multithreading
- Methodenaufrufe bei Client - Anfragen

Servlets III – Servlet

```
public class ServletName extends HttpServlet { … }
```

Ein Servlet besteht aus Methoden zum initialisieren des Servlets, sowie aus Methoden zur Bearbeitung von HTTP-Requests. Es wird in der web.xml Datei konfiguriert.

Methoden:

`static{}`	statischer Klassenkonstruktor
`ServletName()`	Parameterloser Standard-Konstruktor
`init()`	Initialisierungsmethode
`doGet()`	Methode zur Bearbeitung eines HTTP-GET
`doPost`	Methode zur Bearbeitung eines HTTP-POST
`doHead()`	Methode zur Bearbeitung eines HTTP-HEAD
`doPut()`	Methode zur Bearbeitung eines HTTP-PUT
`doDelete()`	Methode zur Bearbeitung eines HTTP-DELETE
`destroy()`	Destruktor

Servlets III – Servlet

```
public class ActivityLoggerServlet extends HttpServlet {

   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {

        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
           out.println("<!DOCTYPE html>");
            out.println("<html>");
           out.println("<body>");
           out.println("</body>");
            out.println("</html>");
        }
    }
}
```

Servlets III – Servlet

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
    <servlet>
        <servlet-name>ActivityLoggerServlet</servlet-name>
        <servlet-class>ActivityLoggerServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>ActivityLoggerServlet</servlet-name>
        <url-pattern>/ActivityLoggerServlet</url-pattern>
    </servlet-mapping>
</web-app>
```

Servlets IV - Lebenszyklus

1. **Laden der Servletklasse**
   - durch Aufruf des statischen Konstruktors *static*

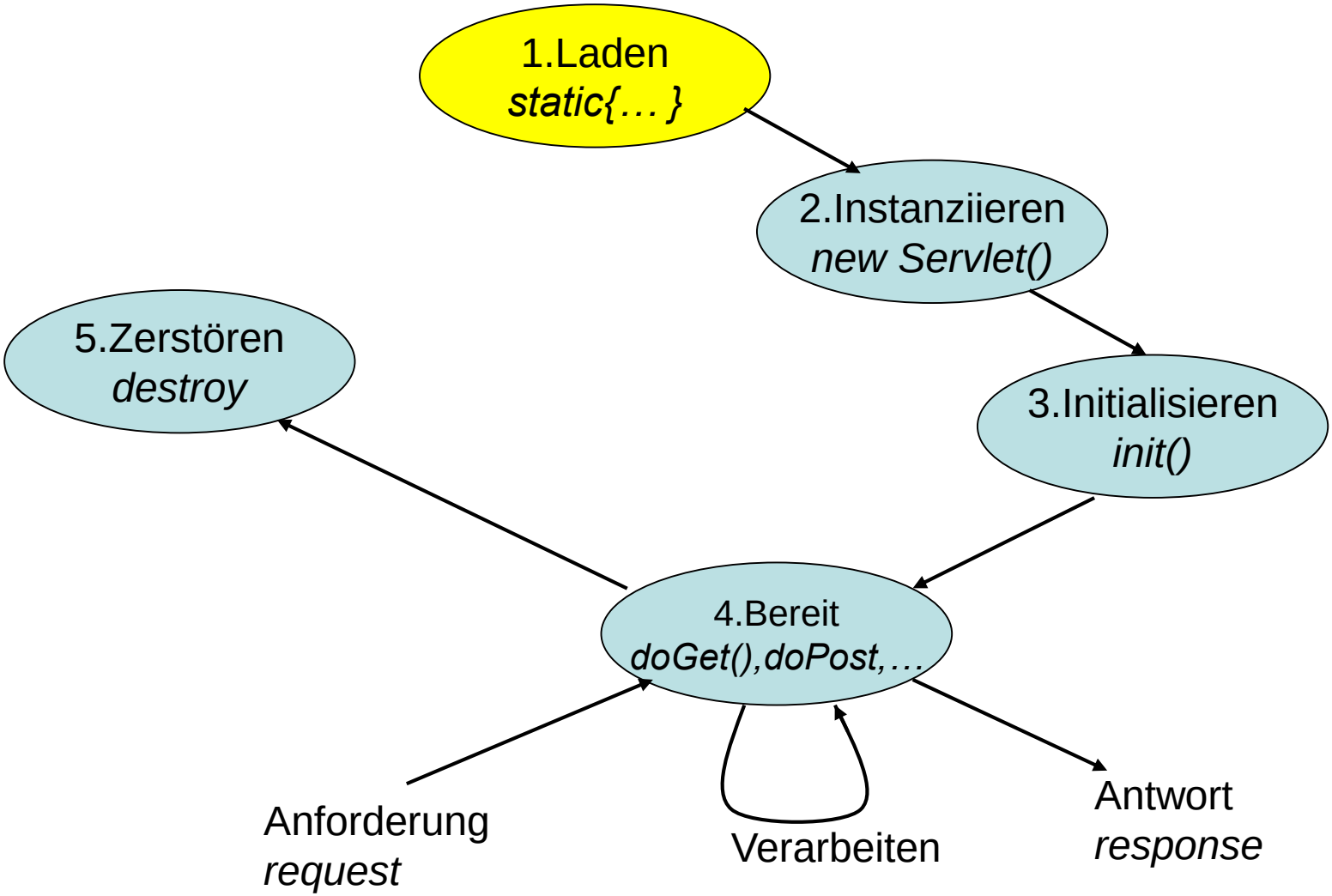
2. **Instanziieren des Servlet-Objektes**
   - parameterloser Konstruktor wird aufgerufen

3. **Initialisieren des Servlet-Objektes**
   - überschriebene Methode *init()* wird aufgerufen

4. **Anforderungen bearbeiten (zentraler Teil)**
   - je nach Anforderungen werden die überschriebenen Methoden *doGet(), doPost(), …* aufgerufen
   - entsprechen den HTTP-Methoden GET u.POST)

5. **Servlet-Objekt entfernen**
   - wird vom Container veranlasst
   - überschriebene Methode *destroy()* wird aufgerufen

Servlets IV – Lebenszyklus 1 - Laden



Servlets IV – Lebenszyklus 1 - Laden

Definition: **Der Servlet-Container lädt ein Servlet bei seiner ersten Anforderung oder beim Starten des Containers und führt den Klassenkonstruktor aus.**

`static { … }`

```
public class ActivityLoggerServlet extends HttpServlet {

    private static List<String> aList = new ArrayList<>();

    // Step 1: Load
    static { aList.add("load at: " + new Date().getTime());}

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
           out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<body>");
            for(String curAct : aList) {
                out.println(curAct+"<br>");
            }
            out.println("</body>");
            out.println("</html>");
        }
```

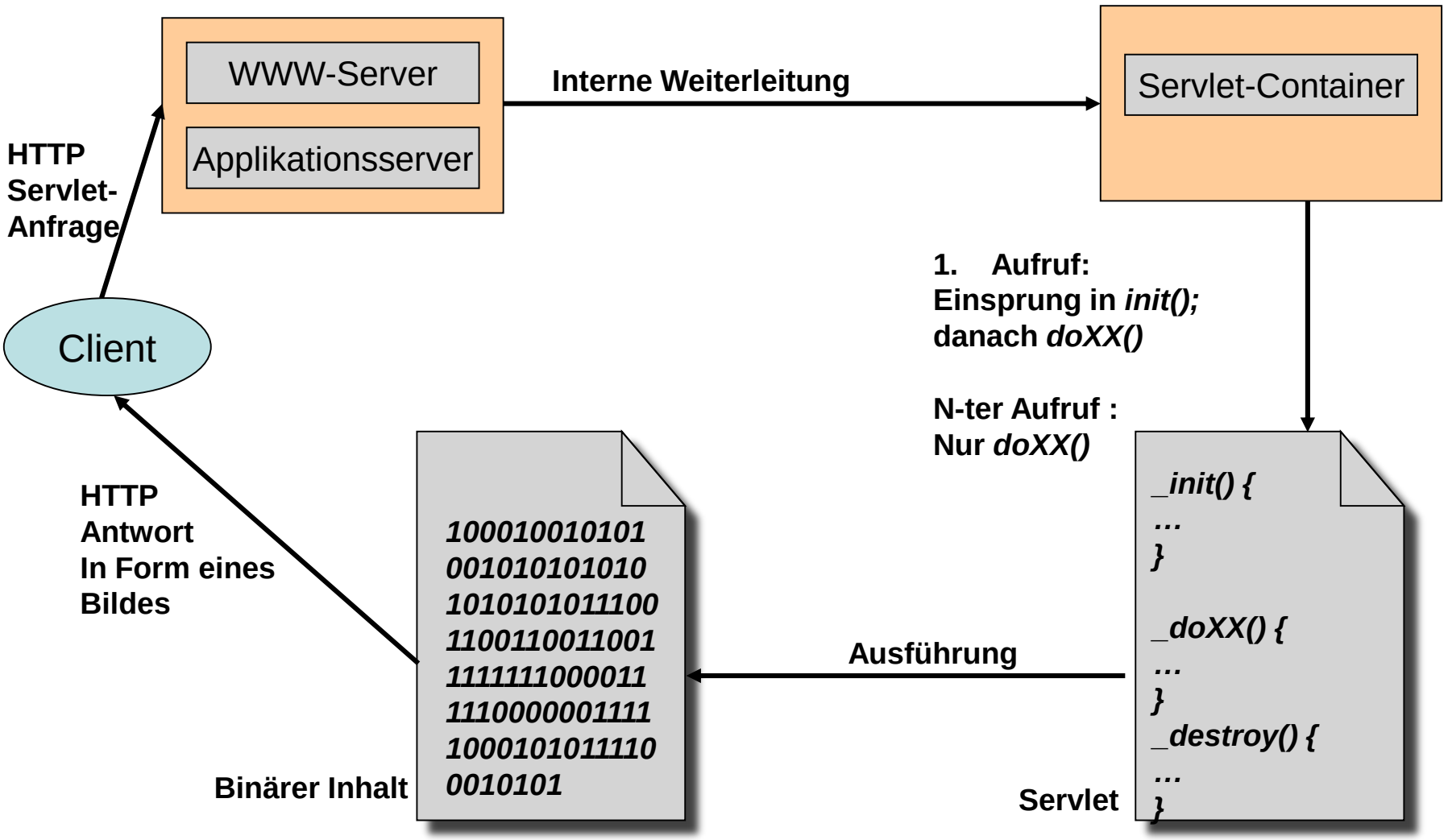
Servlets IV – Lebenszyklus 1 - Laden

Um ein Sevlet bereits beim Start des Servlet-Containers zu starten, muss ein Eintrag in der web.xml hinzugefügt werden.

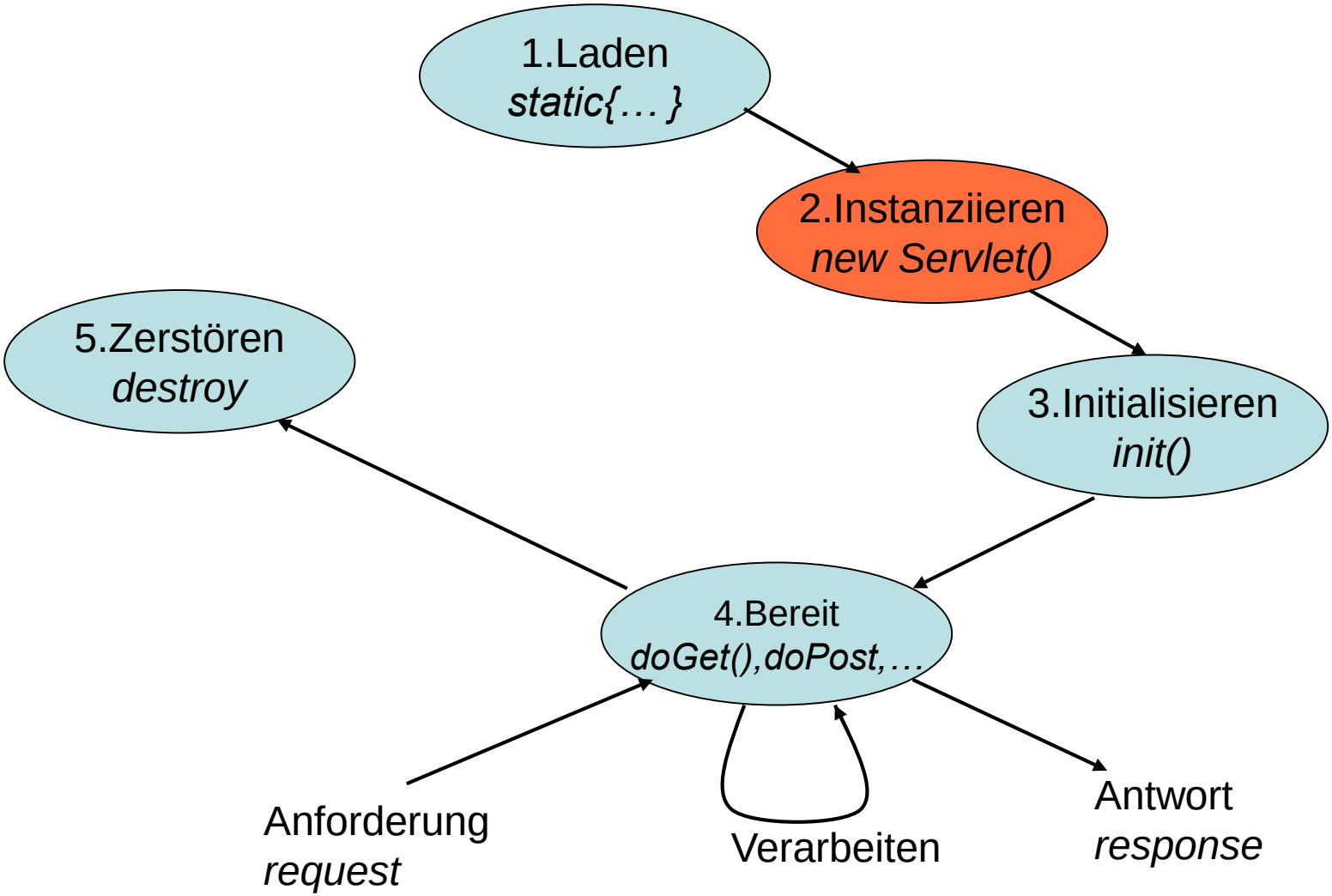
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
    <servlet>
        <servlet-name>ActivityLoggerServlet</servlet-name>
        <servlet-class>ActivityLoggerServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>ActivityLoggerServlet</servlet-name>
        <url-pattern>/ActivityLoggerServlet</url-pattern>
    </servlet-mapping>
    <load-on-startup>4</load-on-startup>
</web-app>
```

Ausschnitt aus einer web.xml

Servlets IV – Lebenszyklus 1 - Laden



Servlets IV – Lebenszyklus 2 - Instanziieren



Servlets IV – Lebenszyklus 2 - Instanziieren

Definition: **Der Servlet-Container instanziiert ein Servlet automatisch, nachdem es geladen wurde. Der Standard-Konstruktor kann überschrieben werden.**

```
public ServletName() { … }
```

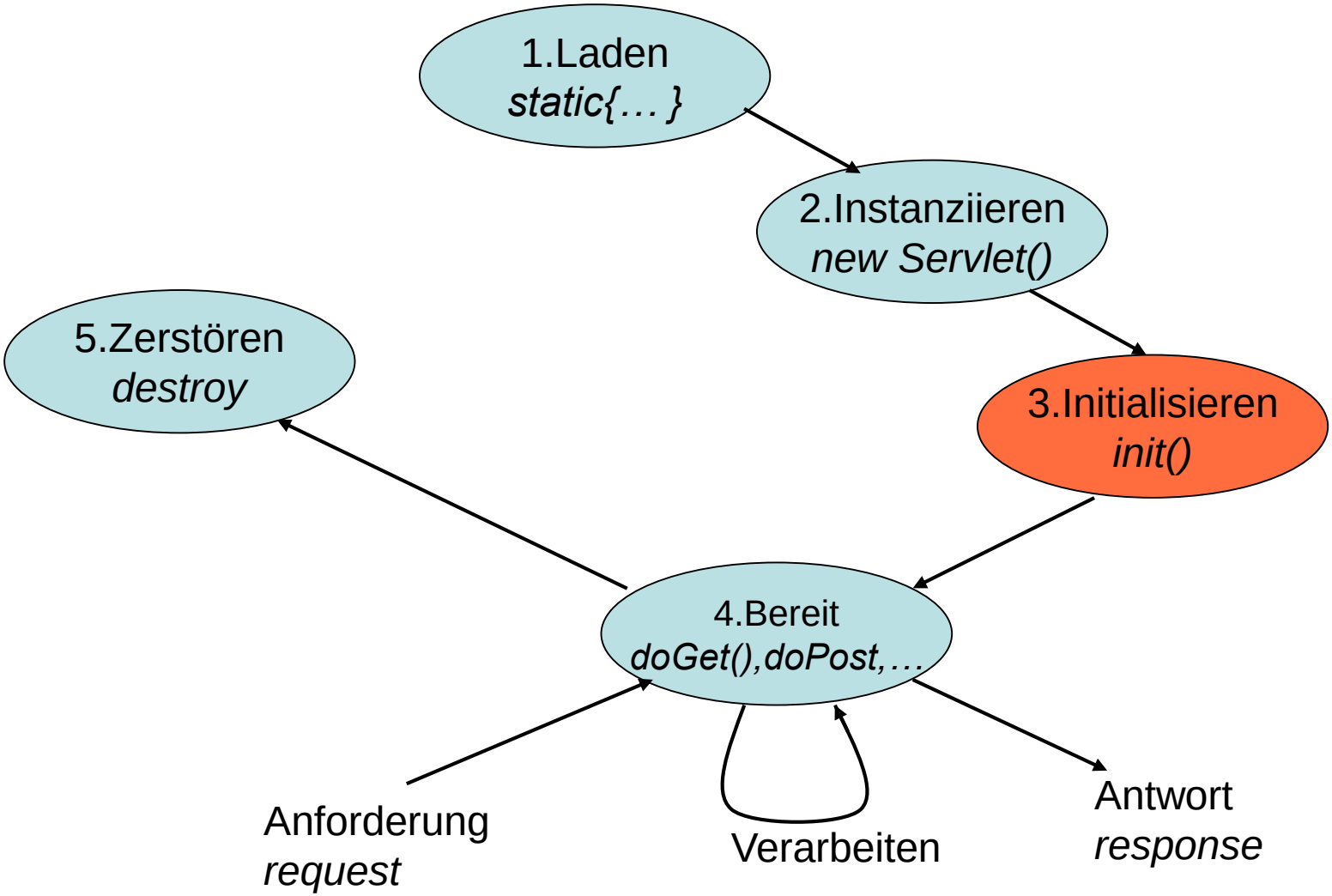
Daraus folgt:

- Zu jeder Servlet-Klasse gibt es eine Instanz
- Jede Anforderung an ein Servlet wird über diese Instanz abgewickelt
- parallele Verarbeitung von Anforderungen an das gleiche Servlet (gleiche Instanz) möglich
- jede Anforderung wird in einem eigenen Thread abgearbeitet wird
  - ➢ Konflikte bei gemeinsamen Ressourcen möglich

Servlets IV – Lebenszyklus 3 - Initialisieren

```
public class ActivityLoggerServlet extends HttpServlet {

    private static List<String> aList = new ArrayList<>();

    // Step 1: Load
    static { aList.add("load at: " + new Date().getTime());}

    // Step 2: Instanziierung
    public ActivityLoggerServlet() {
        aList.add("Instanziierung at: " + new Date().getTime());
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
           out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<body>");
            for(String curAct : aList) {
                out.println(curAct+"<br>");
            }
            out.println("</body>");
            out.println("</html>");
        }
    }
```

Servlets IV – Lebenszyklus 3 - Initialisieren



Servlets IV – Lebenszyklus 3 - Initialisieren

Definition: **Der Servlet-Container initialisiert ein Servlet nach der Initziierung. In der Initialisierung kann auf Eigenschaften des Servlets zugegriffen werden.**

```
public void init(ServletConfig sc) { … }
```

ServletConfig

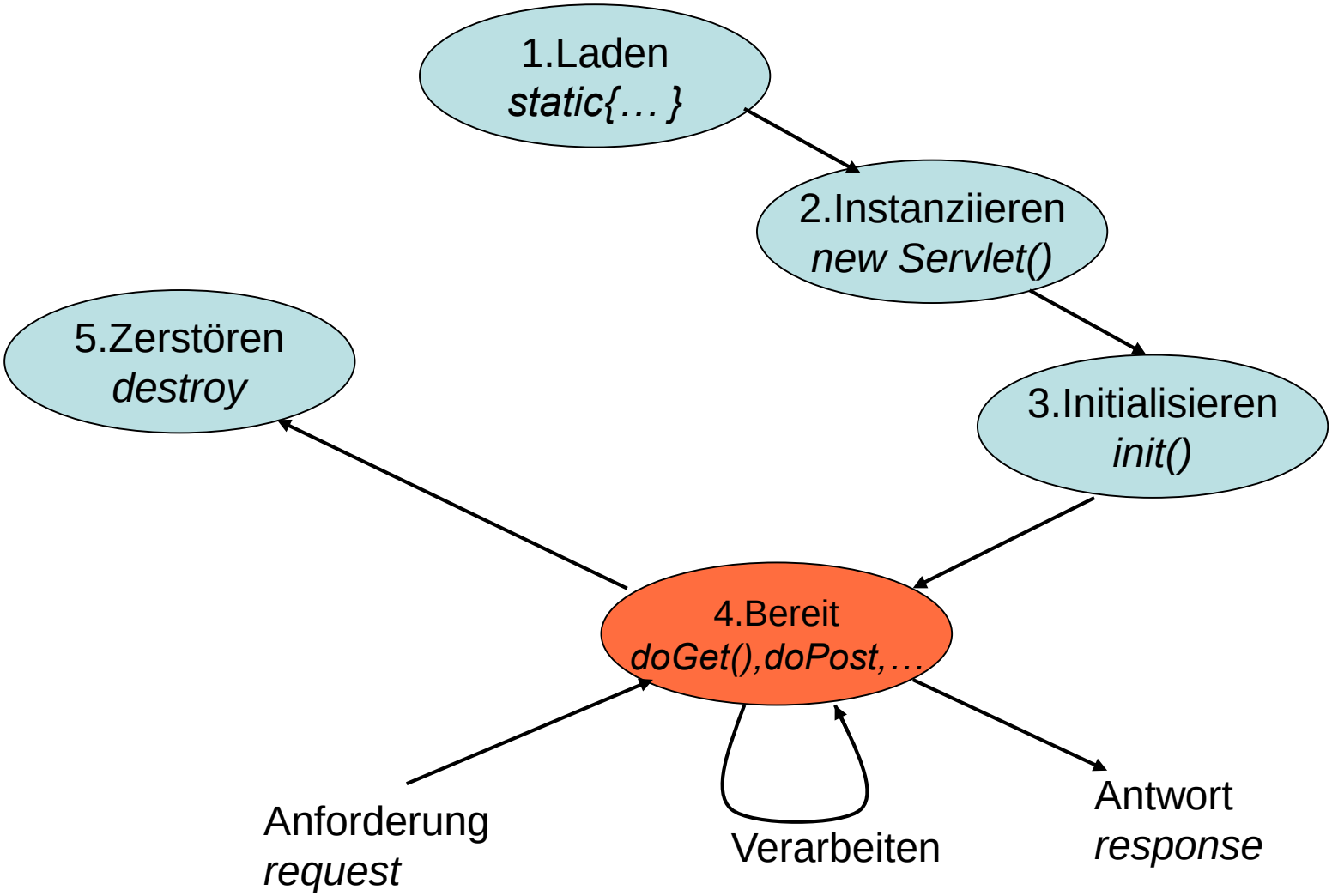
`getServletName()`	liefert den Namen des Servlets lt. web.xml
`getInitParameter()`	liefert Einstellungen aus der web.xml
`getServletContext()`	zugriff auf Zahlreiche Context-Eigenschaften

Ermöglicht es Vorbereitungen für den laufenden Betrieb vorzunehmen.

Servlets IV – Lebenszyklus 3 - Initialisieren

```
public class ActivityLoggerServlet extends HttpServlet {

    private static List<String> aList = new ArrayList<>();

    // Step 1: Load
    static { aList.add("load at: " + new Date().getTime());}

    // Step 2: Instanziierung
    public ActivityLoggerServlet() {
        aList.add("Instanziierung at: " + new Date().getTime());
    }

    // Step 3: Initialisierung
    @Override
    public void init(ServletConfig sc) throws ServletException {
        aList.add(sc.getServletName() + " initalised at: " + new Date().getTime());
    }
}
```

Servlets IV – Lebenszyklus 4 - Bereit



Servlets IV – Lebenszyklus 4 - Bereit

Definition: **Die Phase 4 ist der zentrale Teil des Servlets: Annahme und Bearbeitung der Client-Anfragen finden hier statt.**

```
public void doGet(HttpServletRequest request,
HttpServletResponse response) { … }
```

Es kann für jede der 7 HTTP-Methoden eine Methode implementiert werden.

```
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
       out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<body>");
        for(String curAct : aList) {
            out.println(curAct+"<br>");
        }
        out.println("</body>");
        out.println("</html>");
    }
}
```

Servlets IV – Lebenszyklus 4 - Request

Definition: **Das HttpServletRequest beinhaltet die Informationen der Anfrage.**

```
public void doGet(HttpServletRequest request,
HttpServletResponse response) { … }
```

Methoden

`getParameterNames()`	Namen der Parameter der Anfrage
`getParameter()`	Wert eines Parameters abfragen
`getCookies()`	Liefert die definierten Cookies
`getHeader()`	Zugriff auf HTTP-Header
`getSession()`	enthält Angaben zur Sitzungsverwaltung

Servlets IV – Lebenszyklus 4 - Request

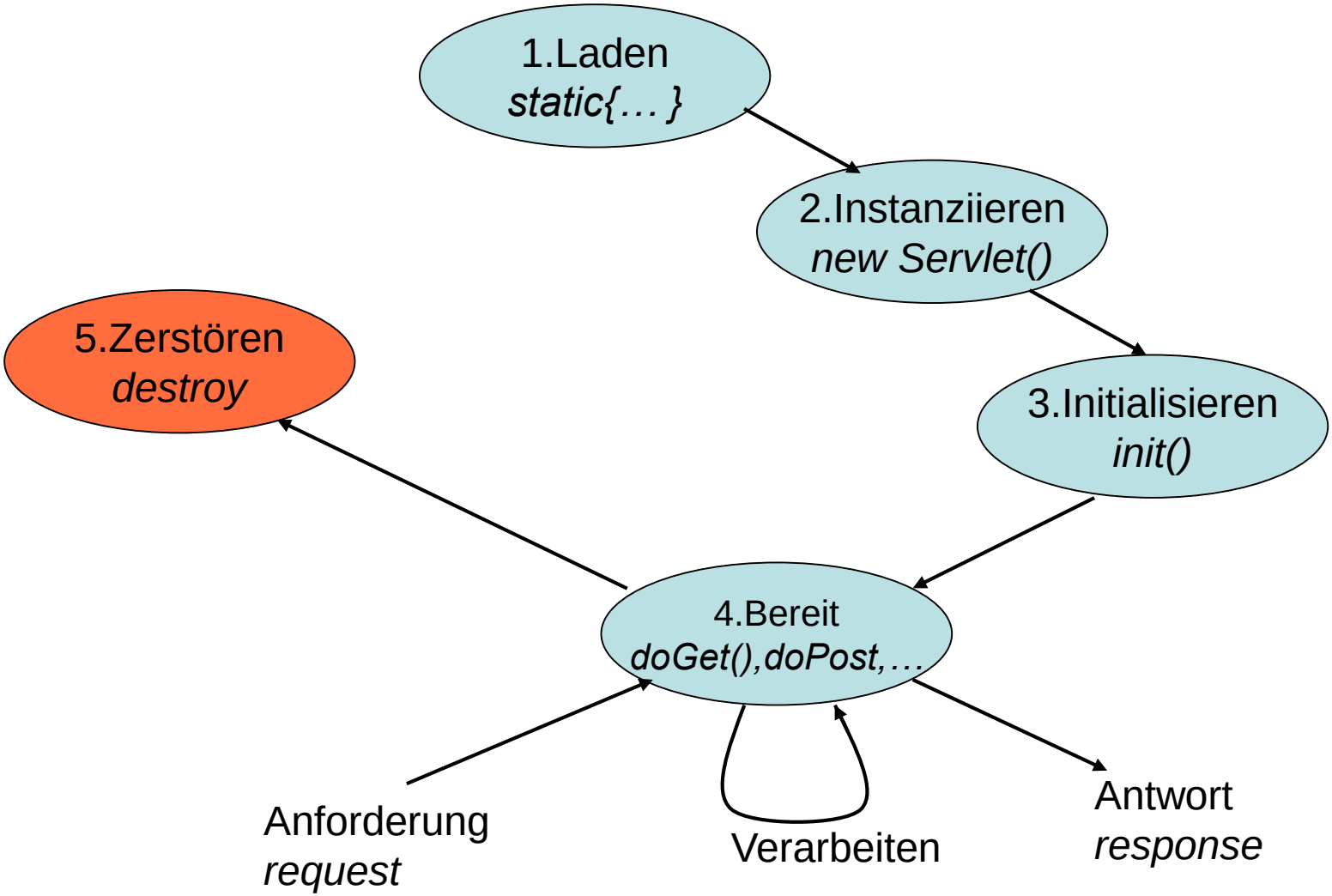
Definition: **Das HttpServletRequest beinhaltet die Informationen der Anfrage.**

```
public void doGet(HttpServletRequest request,
HttpServletResponse response) { … }
```

Methoden

`setContentType()`	Setzt den MIME-Type der Antwort
`getWriter()`	holt den Output-Stream zum schrieben der Antwort
`addCookie()`	Fügt einen Cookie zur Antwort hinzu
`addHeader()`	Fügt einen HTTP-Header zur Antwort hinzu
`sendError()`	sendet einen HTTP-Fehlercode

Servlets IV – Lebenszyklus 5 - Zerstören



Servlets IV – Lebenszyklus 5 - Zerstören

Definition: **Bei Liquidierung eines Servlets ruft der Container die Methode destroy() auf.**

```
public void destroy() { … }
```

Übliche Aufgaben der destroy-Methode:

- Datenbankverbindungen schließen
- Threads deaktivieren
- Sichern von Servletdaten auf Festplatte
- andere Bereinigungsarbeiten

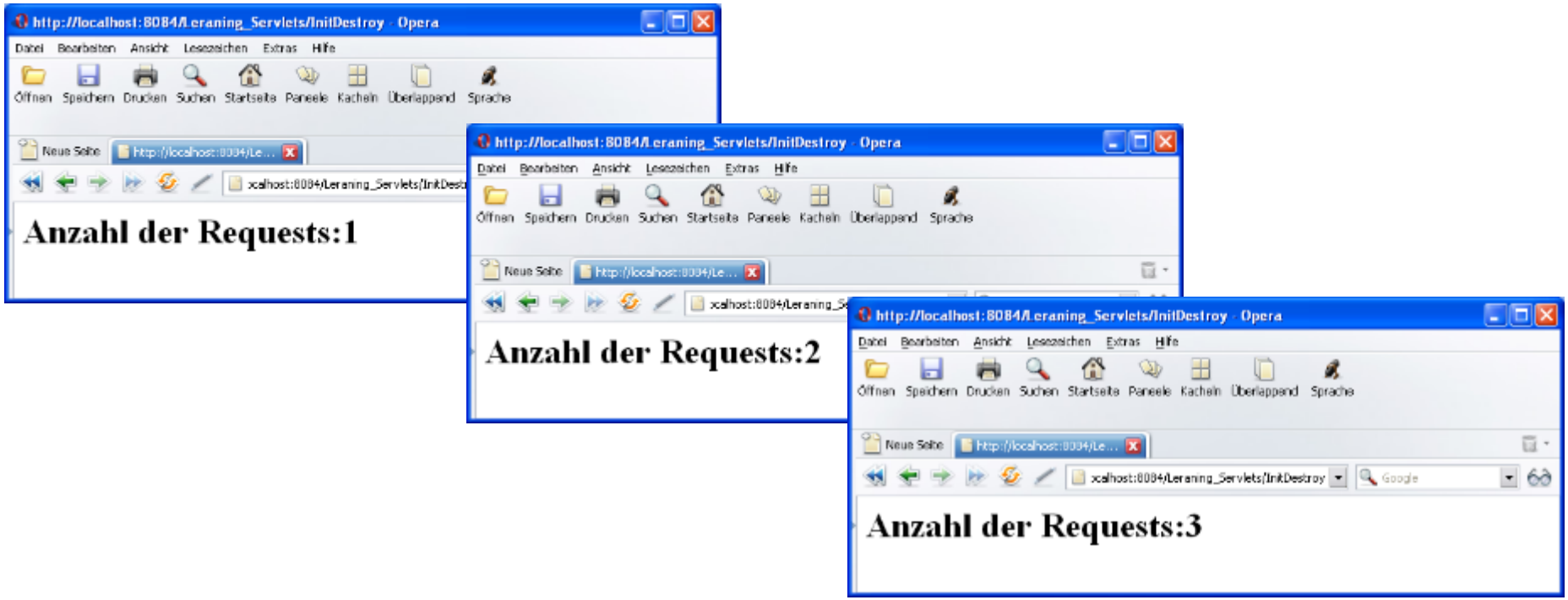
Methode *destroy()* wird leider nicht aufgerufen, wenn der Webserver abstürzt, zusätzliche Absicherung notwendig!

Servlets IV – Lebenszyklus 5 - Zerstören

```
public class InitDestroy extends HttpServlet {
    int zaehler;
    String datei;

    public void init(ServletConfig sc){
        try{
            datei = sc.getServletContext().getRealPath("/")+
         "zaehler.stand";
            FileInputStream f = new FileInputStream(datei);
            zaehler = new DataInputStream(f).readInt();
        } catch (Exception e){ }
    }

    public void destroy() {
        try{
            FileOutputStream f = new FileOutputStream(datei);
            new DataOutputStream(f).writeInt(zaehler);
        }catch (Exception e){ }
    }
    public void doGet(HttpServletRequest request,
         HttpServletResponse response)
         throws ServletException, IOException {
        zaehler++;
        response.setContentType("text/html");
        response.getWriter().println("<h1> Anzahl
        der Requests:" + zaehler + "</h1>");
    }
}
```

- Einlesen des Zählers in *init()*
- Erhöhung des Zählers in jeder Anforderung *doGet()*
- bei jedem *destroy()* wird der Zählerstand in Datei *zaehler.stand* geschrieben (Entladen des Servlets)
- Datei *zaehler.stand* steht im Wurzelverzeichnis der Webapplikation.

Servlets IV – Lebenszyklus 5 - Zerstören



Bei jedem Reload oder Neuanfrage im Browser wird der Zähler um 1 hochgezählt. Auch keine Unterbrechung im Hochzählen durch Stoppen und Starten des Webservers.

Servlets V – Beispiel mit Threads

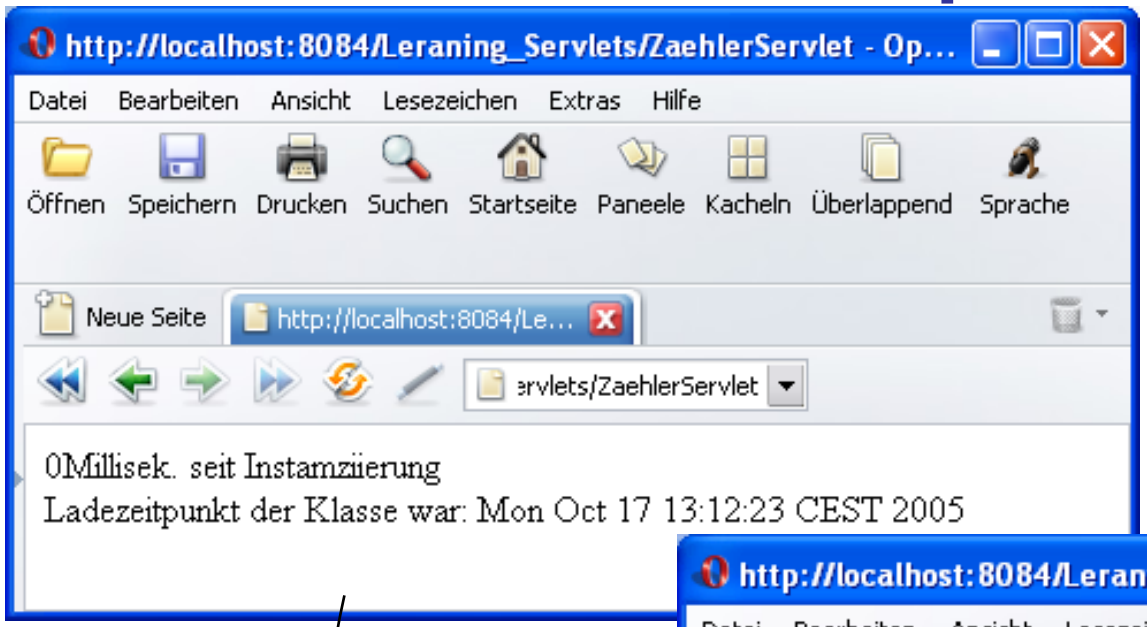
```
public class ZaehlerServlet extends HttpServlet implements Runnable{
    private static Date startzeit;
    static {                    //Klassenkonstruktor hält Datum und Uhrzeit des Ladens fest
        startzeit = new Date();
    }
    private long zaehler;
    private int tick;

    public void init() {       //Container iniziiert Servlet und ruft init-Methode auf
        String temp = getInitParameter("tick"); // aus web.xml eingelesenes Parameter
        tick = (temp==null) ? 1000 : Integer.parseInt(temp);
        new Thread(this).start();  //this-Objekt erhält eigenen Thread, der gestartet wird
    }

    public void run() {
        while (true) { //Endlosschleife in der run-Methode -> Thread bleibt aktiv…
            zaehler += tick;  //…solange das Servlet-Objekt lebt; Erhöhung des Zählers
            try {Thread.sleep(tick);}  // Thread schläft „tick“-Milisekunden lang
            catch (InterruptedException ex) {}
        }
    }

   public synchronized void doGet(HttpServletRequest request, HttpServletResponse response)
                                        throws ServletException, IOException{
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.print(zaehler + "Millisek. seit Instanziierung <BR>");
        out.print("Ladezeitpunkt der Klasse war: " + startzeit);
    }
}
```

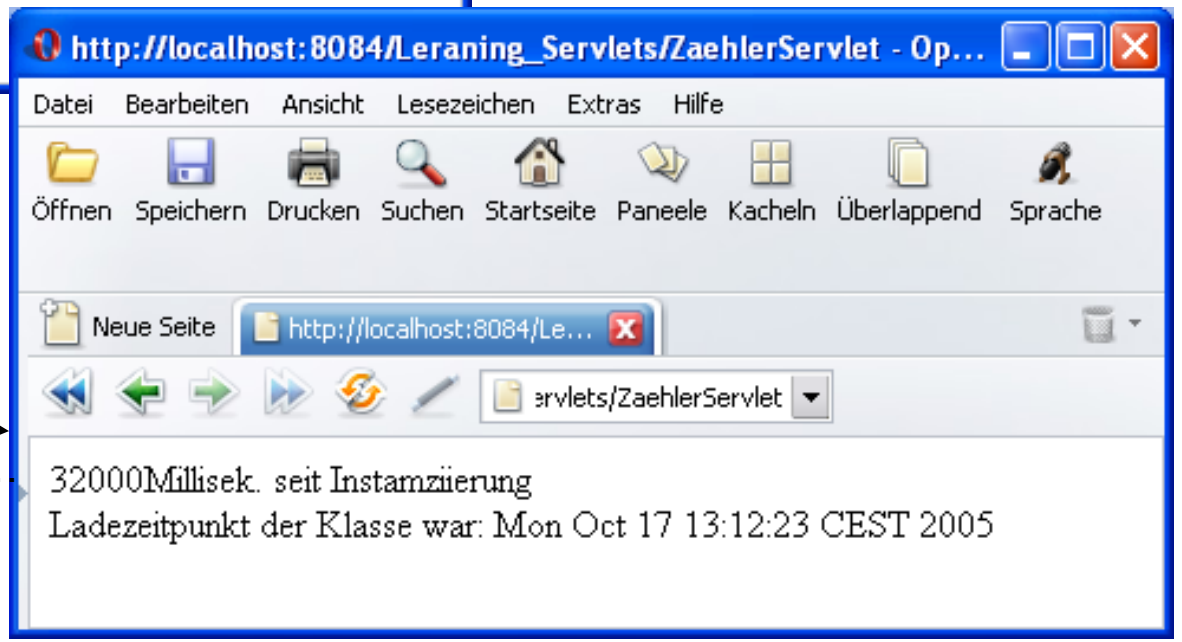
Servlets V – Beispiel mit Threads

Ausschnitt aus web.xml :

```
<servlet>
    <!-- servlet-name in Netbeans automatisch generiert -->
    <servlet-name>ZaehlerServlet</servlet-name>
    <servlet-class>ZaehlerServlet</servlet-class>
    <!-- Initialisierungsparameter wird festgelegt mit Name und Wert -->
    <init-param>
       <param-name>tick</param-name>
       <param-value>1000</param-value>
    </init-param>
    <!-- positiver Wert: Container lädt Servlet bei seinem  Start -->
    <load-on-startup>1</load-on-startup>
</servlet>
```

Servlets V – Beispiel mit Threads





Reload im Browser etwas später…
(HTTP-GET –Anforderung)

Servlets VI – Template Verfahren

Definition: **Beim Template Verfahren werden Teile der Antwort aus einem Template geladen.**

Eigenschaften

- auszugebender Text ist nicht mehr in einer print()-Anweisung, sondern in einer externen Datei oder anderen Textquelle (DB)
- Template wird erst zur Laufzeit eingelesen

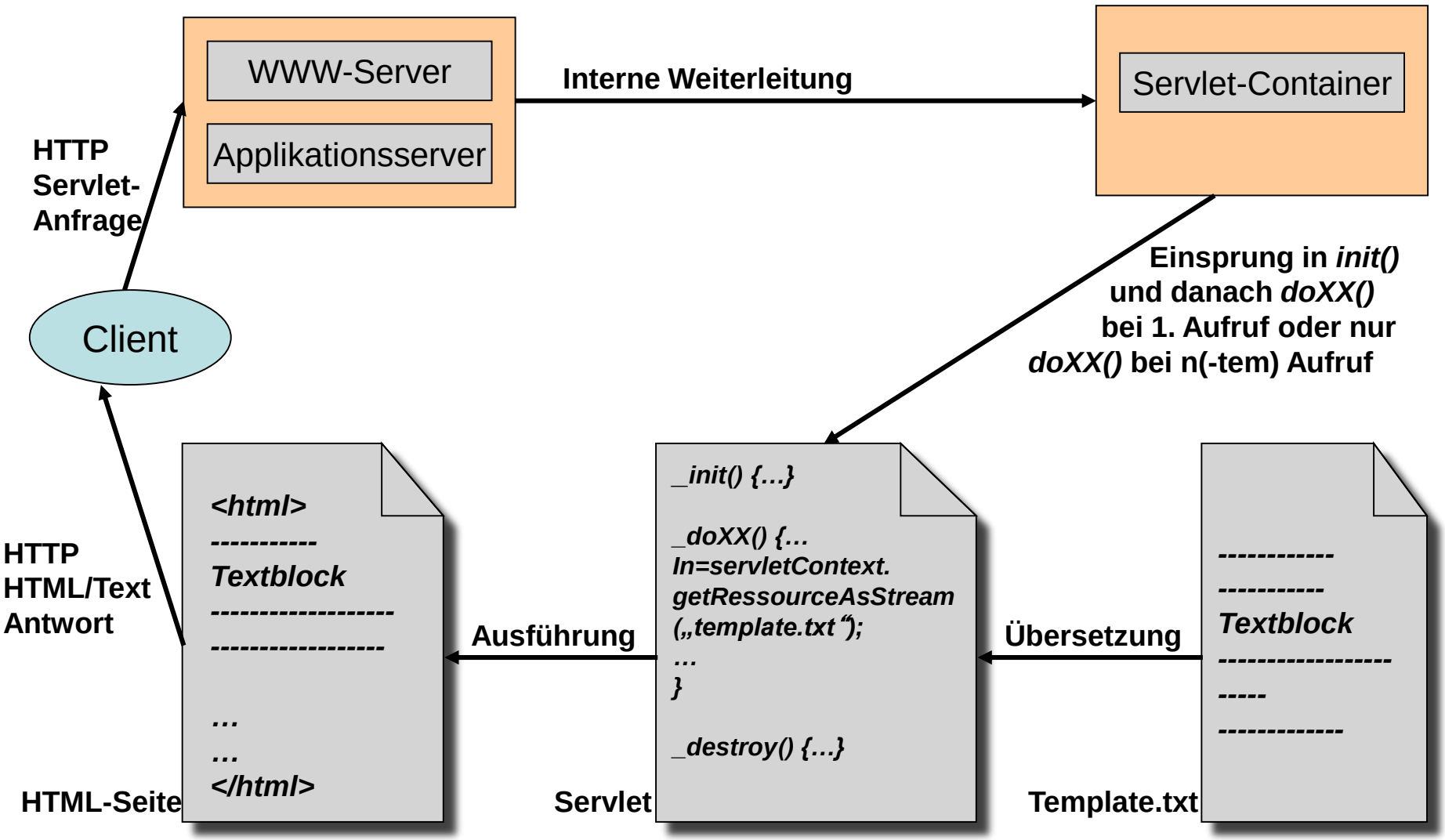
Vorteil

- Gute Trennung zwischen Text und Programmcode

Nachteil

- Performanz des Servers insgesamt wird beeinträchtigt

Servlets VI – Template Verfahren



Servlets VII – Sessions

Definition: **In Servlets können Sessions einfach über ein Session-Objekt verwendet werden.**

```
HttpSession session = request.getSession();
```

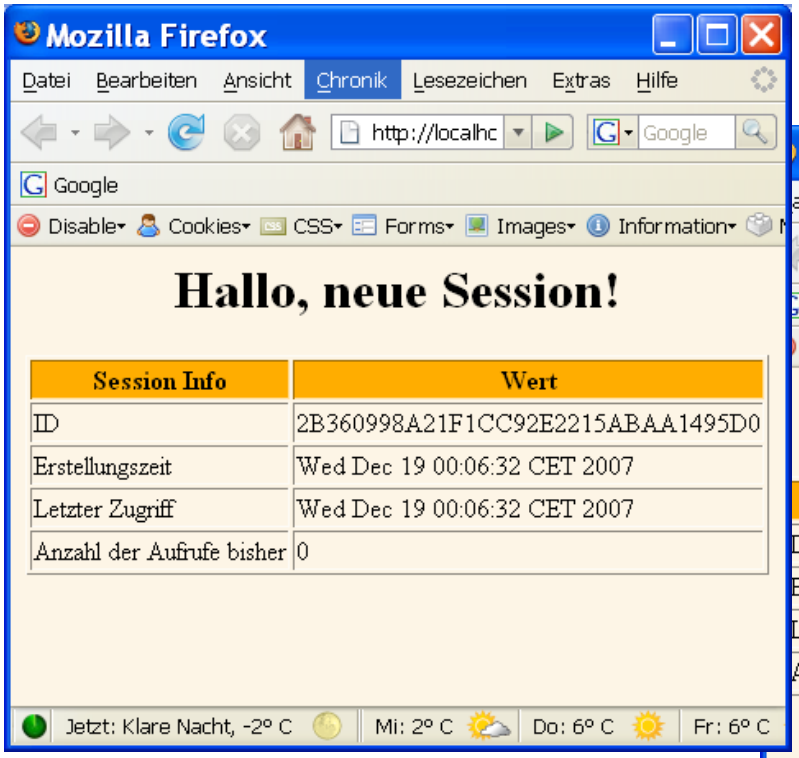
Methoden der Session

`setAttribute(k,v)`	schreibt einen Wert als Attribut in die Session
`getAttribute(k)`	Holt den Wert eines Session-Attributes
`getAttributeNames()`	Liefert eine Liste aller Attribute
`removeAttribute(k)`	Löscht ein Attribut aus der Session
`getId()`	Liefert die ID der Session
`getCreationTime()`	Gibt den Zeitpunkt der Erstellung der Session
`getLastAccessedTime()`	Gibt den Zeitpunkt der letzten Verwendung

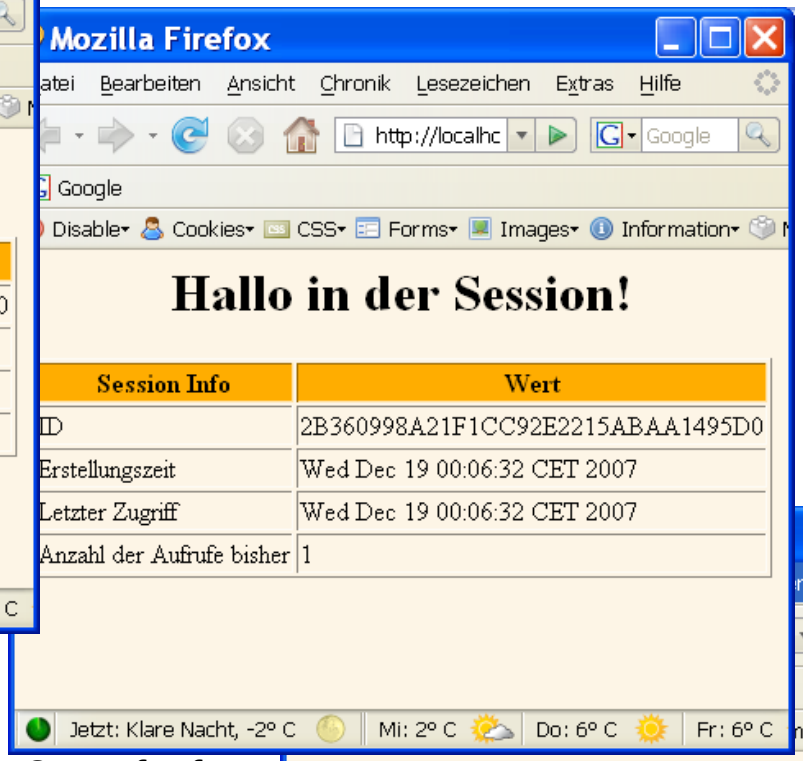
Beispiel Session Tracking
- komplettes Listing-

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

public class SessionTracking extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    HttpSession session = request.getSession(true);
    String heading;
    Integer accessCount
        =(Integer)session.getAttribute("accessCount");
    if (accessCount == null) {
      accessCount = new Integer(0);
      heading = "Hallo, neue Session!";
    } else {
      heading = "Hallo in der Session!";
      accessCount = new Integer(accessCount.intValue() + 1);
    }
    session.setAttribute("accessCount", accessCount);

   // … weiter rechts
```

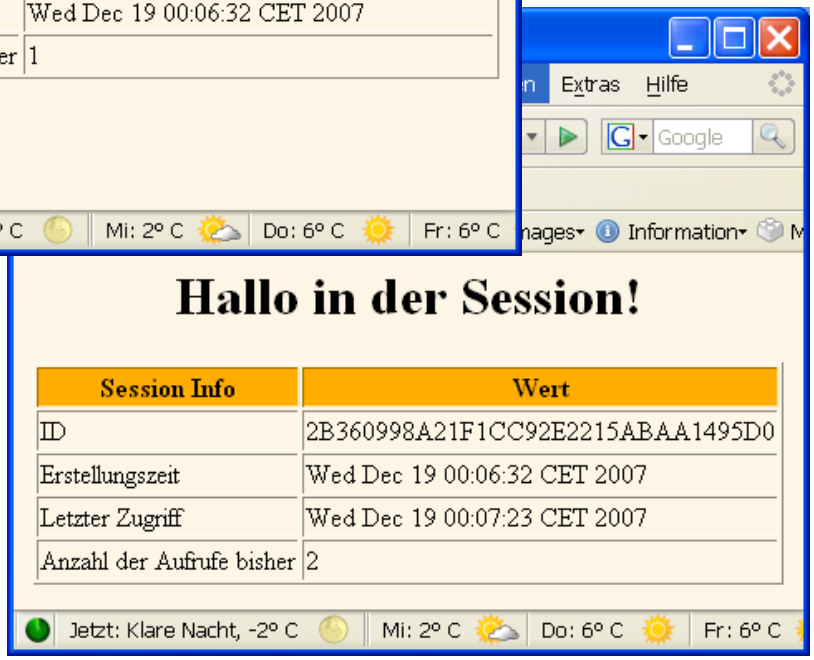
```
/… weiter
out.println(
              "<BODY BGCOLOR=\"#FDF5E6\">\n" +
              "<H1 ALIGN=\"CENTER\">" + heading + "</H1>\n" +
              "<TABLE BORDER=1 ALIGN=\"CENTER\">\n" +
              "<TR BGCOLOR=\"#FFAD00\">\n" +
              "  <TH>Session Info<TH>Wert\n" +
              "<TR>\n" +
              "  <TD>ID\n" +
              "  <TD>" + session.getId() + "\n" +
              "<TR>\n" +
              "  <TD>Erstellungszeit\n" +
              "  <TD>" +
              new Date(session.getCreationTime()) + "\n" +
              "<TR>\n" +
              "  <TD>Letzter Zugriff\n" +
              "  <TD>" +
              new Date(session.getLastAccessedTime()) + "\n" +
              "<TR>\n" +
              "  <TD>Anzahl der Aufrufe bisher\n" +
              "  <TD>" + accessCount + "\n" +
              "</TABLE>\n" +
              "</BODY></HTML>");
  }

  /** Handle GET and POST requests identically. */

  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
      throws ServletException, IOException {
    doGet(request, response);
  }
}
```



1. Aufruf



2. Aufruf



3. Aufruf

Serverseitige Anwendungen

1. Kontext und Motivation
2. Webserver Interfaces
3. Servlets
4. **JSP**
5. Darüber hinaus
6. Projekt

JSP

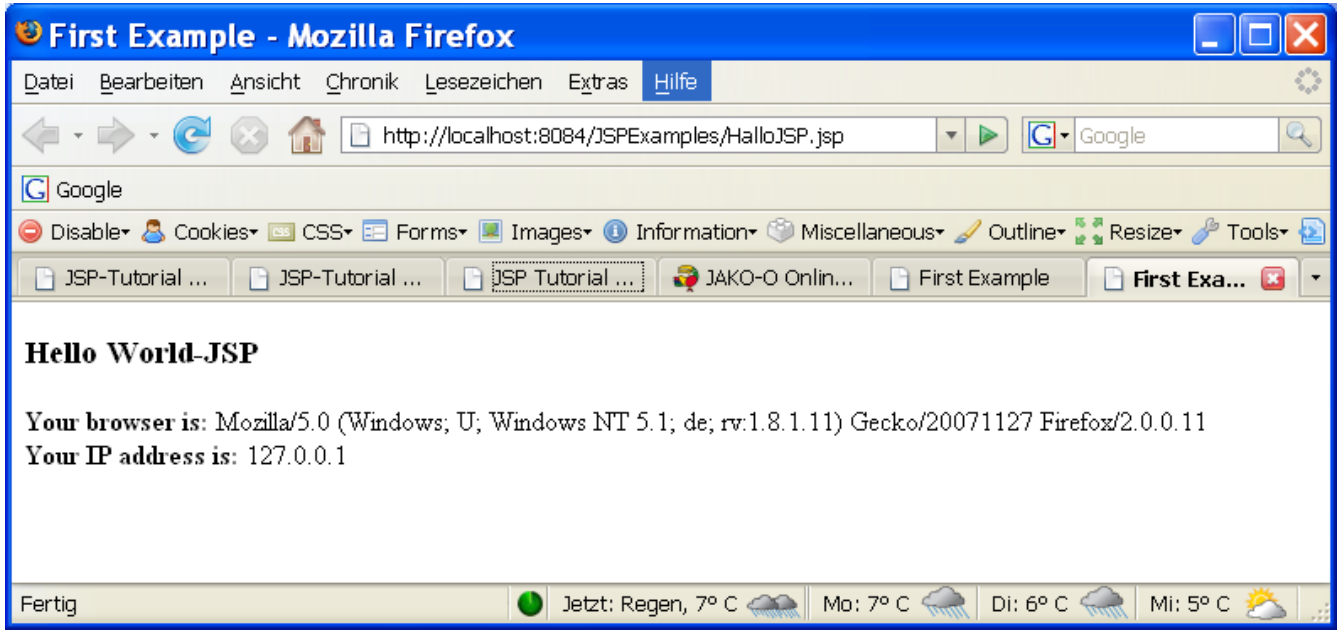
Definition: **Java Server Pages sind eine Websprache zum Entwickeln von Weboberflächen für Java-Anwendungen.**

Eigenschaften

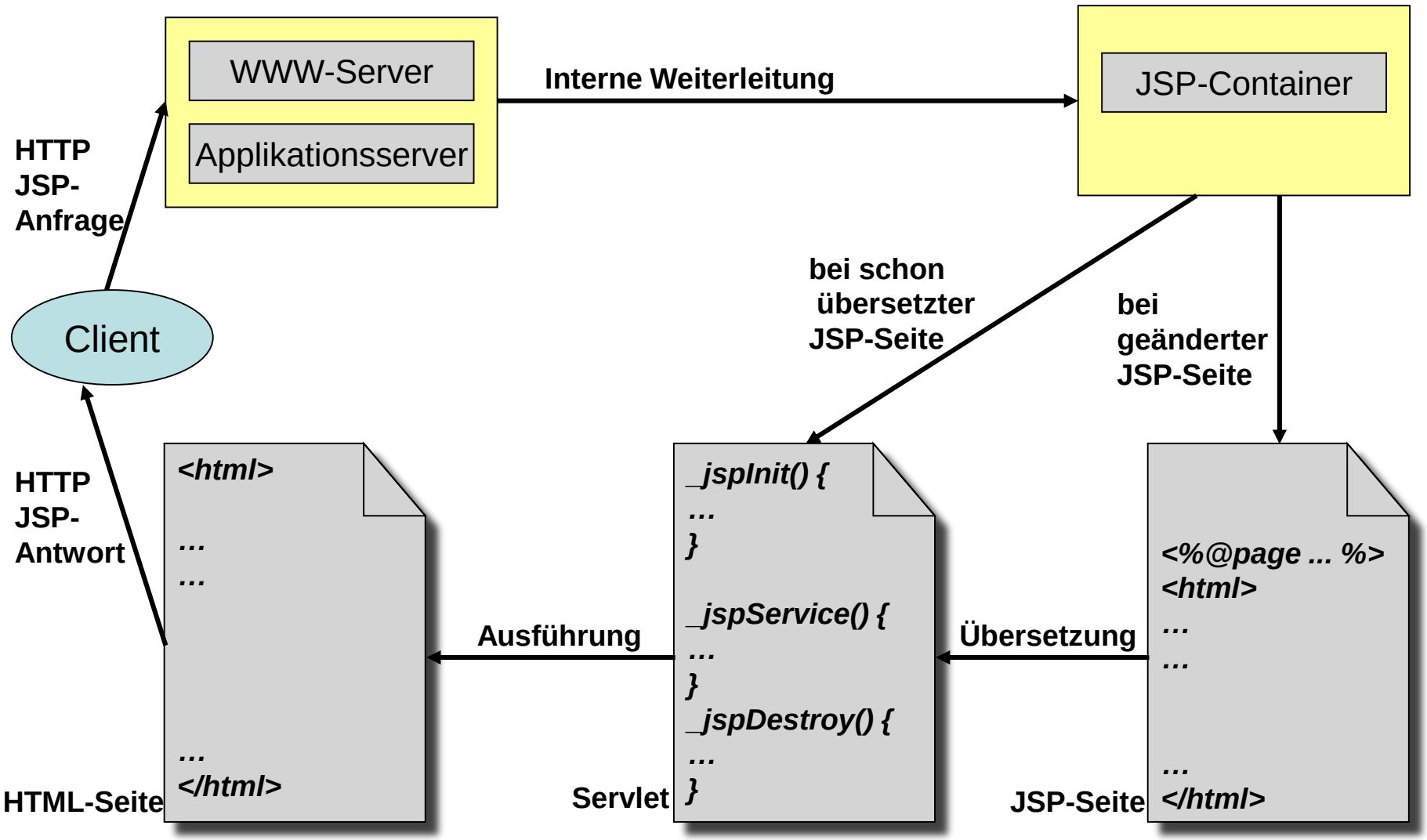
- **Präsentationsschicht** von Webanwendungen
- **Web-Scriptsprache** auf Grundlage der zahlreichen Java - APIs
- Ermöglichen **Integration** von existierenden Systemen ins Web
- Basiert auf der **Java-Servlet-API**
- **Mischen von HTML (XML) mit JSP** in einer Seite möglich
- Ermöglicht konsequente **Aufgabentrennung in Web-Entwicklerteams**
- Daten werden mithilfe von Java - Beans in die Webseiten transportiert
- Zunehmend abgelöst durch JSF (Java Server Faces) und JavaFX

JSP

```
<html>
    <head><title>First Example</title></head>
    <body>
    <h3>Hello World-JSP</h3>

    Your browser is: <%= request.getHeader("User-Agent") %><br>
    Your IP address is: <%= request.getRemoteAddr() %><br>

    </body>
</html>
```



JSP II - Lebenszyklus



JSP II - Lebenszyklus

Definition: **Jede JSP wird im Verlauf Ihres Lebenszyklus vom Container zunächst in ein *Servlet transformiert*. Zur Ausführung kommen immer nur Servlets.**

1. Servlet Container erhält HTTP-Anfrage
2. Servlet Container identifiziert JSP-Anfrage (URL Mapping)
3. Servlet Container ruft JSP Engine auf
4. JSP Engine prüft, ob für Anfrage bereits eine aktuelle kompilierte JSP existiert
5. Falls nicht, wird die entsprechende JSP-Seite aus Dateisystem geladen und ein Servlet erzeugt
6. Übersetzen des Servlets und Bereitstellen für ClassLoader
   - Initialisierung der JSP mit Methode *jspInit()*
   - Bearbeiten der Anfrage in *jspService()* oder auch*: "jspProcessRequest()"*
   - Vernichten mit *jspDestroy()* (falls aktuellere JSP vorliegt oder Servlets heruntergefahren werden sollen)

Scriptlets

Definition: **Scriptlets sind spezielle Tags, welche die Einbettung von Java-Code erlauben. Durch sie wird die Mächtigkeit der Sprache Java in JSP ‘s genutzt.**

Eigenschaften

- Java kann in JSPs als Code eingebettet werden.
- Java-Komponenten können von JSPs heraus genutzt werden.
- Technische Vorraussetzungen:
  - verwendete Bibliotheken liegen im Classpath der Web-Anwendung
  - benötigte Packages sind importiert (Direktiven).

Vorteil

- einfache Möglichkeit, beliebigen Java-Code in den JSPs zu verwenden

Nachteile

- erzeugter Code schwer lesbar
- gewünschte Trennung von Präsentation und Logik aufgehoben

Scriptlets sollten in modernen großen Webanwendungen nicht eingesetzt werden

Beispiel für Scriptlets : Anmeldung

```
<html>
   <body>
      <% //Benutzerdaten einlesen
      String us = request.getParameter("user");
       String pw = request.getParameter("password");

       // falls Angaben unvollst., Form. noch mal  senden...
       if (us ==null || "".equals (us) || pw == null || "".equals (pw)) {
      %>
      <h1>Anmeldung:</h1>
        <form method="get">
           <label for="user">Benutzername:</label>
           <input type="text" name="user" value="<%out.print( us!=null ? us:
   "");%>">

           <label for="pwd">Passwort</label>
           <input type="password" name="pwd" value="<%out.print( pw!=null? pw:
   "");%> ">
           <input type="submit" value="Anmelden">
        </form>
      <% } else { %>
        <h1>Willkommen <%out.print(us);%> </h1>
      <% } %>
    </body>
</html>
```

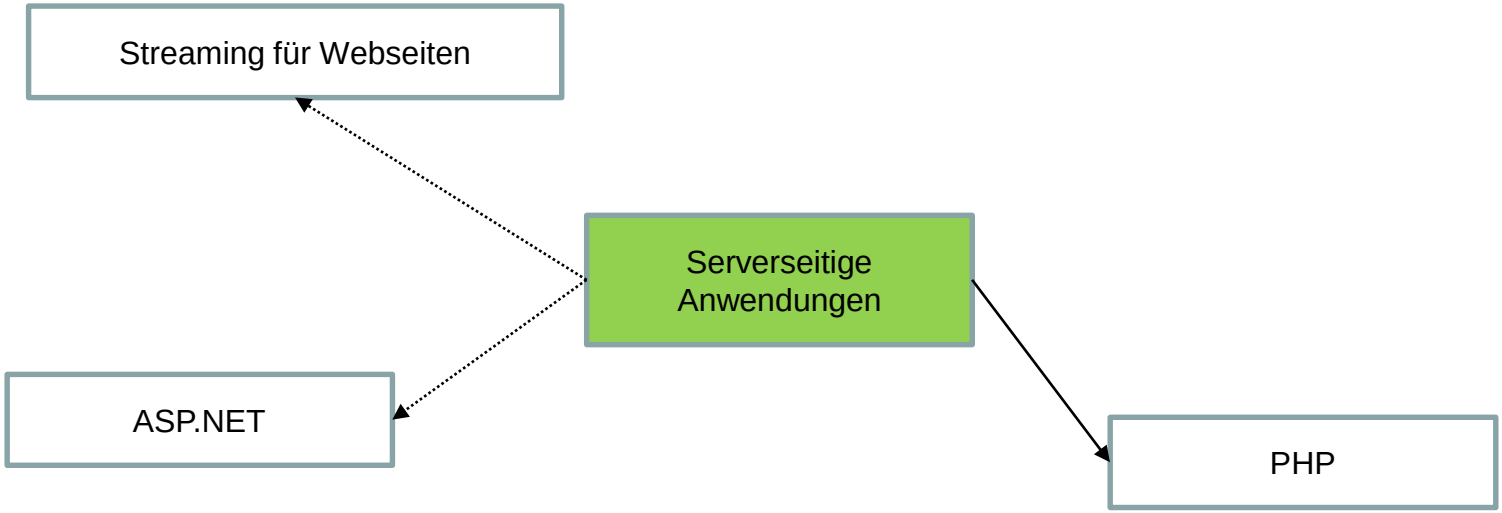
Vergleich JSP - Servlet

Einsatzgebiete :	• dynamische Erzeugung von Textinhalten • Generierung einer Darstellung in HTML, WML, … • Präsentation der Benutzerschnittstelle • Vorverarbeitung eingehender Daten	• dynamische Erzeugung von binären Inhalten • Generierung von textuellen Inhalten mit Templates für Webdarstellungen • Schaffung eines zentralen web-bezogenen Zugangs • Umsetzen der Geschäftsprozesse • Anwenden der Geschäftslogik
Vorteile:	• höhere Abstraktionsebene als Servlets • Einbindung von Tag-Bibliotheken • Deklarativer Stil	• näher am Server (Container) • Darstellung eigener binärer MIME-Typen • Prozeduraler od. objektorientierter Stil
Nachteile:	• starke Mischung von Logik und Darstellung bei größeren Seiten • Mangelnde Skalierbarkeit	• Unübersichtlichkeit bei zu vielen Templates • Reagiert träge zur Laufzeit • ggf. Rollenverteilung von Designer und Entwickler problematisch

Serverseitige Anwendungen

1. Kontext und Motivation
2. Webserver Interfaces
3. Servlets
4. JSP
5. **Darüber hinaus**
6. Projekt

Darüber hinaus



Links:

Apache HTTP-Server - http://httpd.apache.org/
PHP - http://php.net/manual/de/intro-whatis.php
Streaming - https://developer.mozilla.org/en-US/Apps/Fundamentals/Audio_and_video_delivery/Live_streaming_w

Webanwendungen und Sicherheit

Definition:

Sicherheitslücken in Browserimplementierungen können durch JavaScript-Programme ausgenutzt werden z.B.:

- unbemerktes Versenden von Emails
- Auslesen des Browserverlaufs
- Live-Verfolgungen von Internetsitzungen
- Erraten von EBAY-Passwörtern

- Anwender deaktivieren daher manchmal das „Ausführen von JavaScript-Code“ im Browser

- JavaScript-Anwendungen laufen im Browser: Sandbox (abgeriegelte Umgebung ohne Zugriff auf Dateien, Benutzerdaten, BS,..)

Serverseitige Anwendungen

1. Kontext und Motivation
2. Webserver Interfaces
3. Servlets
4. JSP
5. Darüber hinaus
6. **Projekt**

Anforderungen

Welche Anforderungen werden als nächstes bearbeitet?

TODO

- Kommunikation untereinander

DONE

- …
- Formular für Kommentare
- Schickes Design für die Seite
- Mediendatein einbinden
- Animationen
- Mehrsprachen-Fähigkeit
- (lokales) Speichern von Artikeln
- Client-Position anzeigen
- Offline-Verwendung ermöglichen
- Inhaltsverzeichnisse
- Formlareingaben in Seite einfügen
- Navigation über Tastaturkürzel
- Externe Inhalte einbinden
- Artikel vom Server einbinden
- Kommentare vom Server
- Medien hochladen
- Kommentare hochladen
- Kommentare speichern

Literatur: Internet und Netzwerke

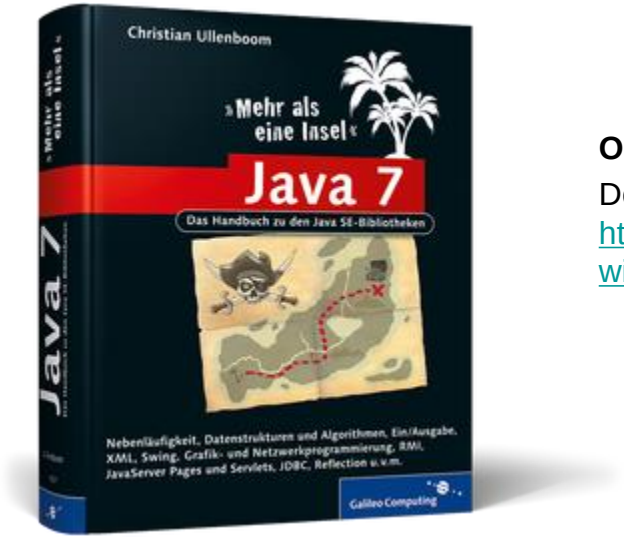
Melzer, Ingo et al. „Service-orientierte Architekturen mit Web Services“ Konzepte – Standards – Praxis 4. Auflage 2010, 381 Seiten, ISBN 978-3-8274-2549-2, Spektrum Akademischer Verlag über Springer Link

Christian Ullenboom: „Java 7 – Mehr als eine Insel

Das Handbuch zu den Java SE-Bibliotheken“

ISBN 978-3-8362-1507-7,

Rheinwerk Verlag 2012



Online-Quellen:

Dokumentation zu Jquery: https://learn.jquery.com/ajax/working-with-jsonp/

Kappel, Gerti & Pröll, Birgit & Reich, Siegfried & Retschitzegger, Werner. (2003). Web Engineering - Die Disziplin zur systematischen Entwicklung von Web-Anwendungen. .