FH Bielefeld
University of
Applied Sciences

Campus Minden

Webbasierte Anwendungen
SS 2015

Java Servlets

Dozentin: Grit Behrens
mailto:grit.behrens@fh-bielefeld.de

Studiengang Informatik Fachbereich Technik

Lehrinhaltsübersicht der Vorlesungen zu WBA

1. Einführung in WBA
2. Wiederholung: Grundlagen des WWW, HTML und HTTP
3. Clientseitige Implementierungstechnologien: Javascript, DOM, Ajax, (Java-Applet)
4. **Serverseitige Implementierungstechnologien:** JSP, **Java-Servlet**
5. Java-WebFramework: Java Server Faces

Serverseitige Implementierungstechnologien (II): Java Servlets

1. **Einführung**
2. Eigenschaften
3. Beispiel „Hello Servlet“
4. Lebenszyklus von Java - Servlets
5. Anfragebearbeitung mit und ohne Templateverfahren
6. Graphiken mit Java – Servlets
7. Sessiontracking
8. Der Servlet – Container
9. Webbezogene und Geschäftsbezogene Servlets
10. Einsatzgebiete der Webkomponenten
11. Vergleich JSP zu Java - Servlet
12. Fazits zu JSP ‘s und Java - Servlets
13. Zusammenfassung
14. Literatur
15. Ausblick

Begriff: „Servlet“

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

?

Einführung in Java Servlets

Def: Servlets *sind* ***Servererweiterungen*** *(serverseitige Java-Komponenten) und sind hierarchisch eine Ebene unter den JSP-Seiten. Sie werden vom Servlet-Container ausgeführt und* ***antworten*** *auf ganz bestimmte* ***Anfragen eines Clients*** *mit einem dedizierten dynamisch erzeugten Inhalt.*
Sie sind als ***Web-Komponenten*** *geschäftsdatenbezoger oder Web-bezogener Natur.*

Einsatzgebiete:

1. **Besondere Eignung zum Versenden und Erzeugen von Binärdaten wie :**
   - Bilder
   - gepackte Daten und Archive
   - Messdaten (z.B. GPS-Positionen, CAN-Bus Daten und andere Parameter von mobilen Anwendungen)
2. **Generierung von textuellen Inhalten mit Templates**
3. **Umsetzen von Geschäftsprozessen und Geschäftslogik**

Serverseitige Implementierungstechnologien (II): Java Servlets

1. Einführung
2. **Eigenschaften**
3. Beispiel „Hello Servlet“
4. Lebenszyklus von Java - Servlets
5. Anfragebearbeitung mit und ohne Templateverfahren
6. Graphiken mit Java – Servlets
7. Sessiontracking
8. Der Servlet – Container
9. Webbezogene und Geschäftsbezogene Servlets
10. Einsatzgebiete der Webkomponenten
11. Vergleich JSP zu Java - Servlet
12. Fazits zu JSP ‘s und Java - Servlets
13. Zusammenfassung
14. Literatur
15. Ausblick

Eigenschaften von Java Servlets

Portabilität:
Vollständige Implementierung in Java erzielt Portabilität bei ***verschiedenen Webservern*** als auch ***Betriebssystemen.***

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, z.B. Session-Tracking.

Serverseitige Implementierungstechnologien (II): Java Servlets

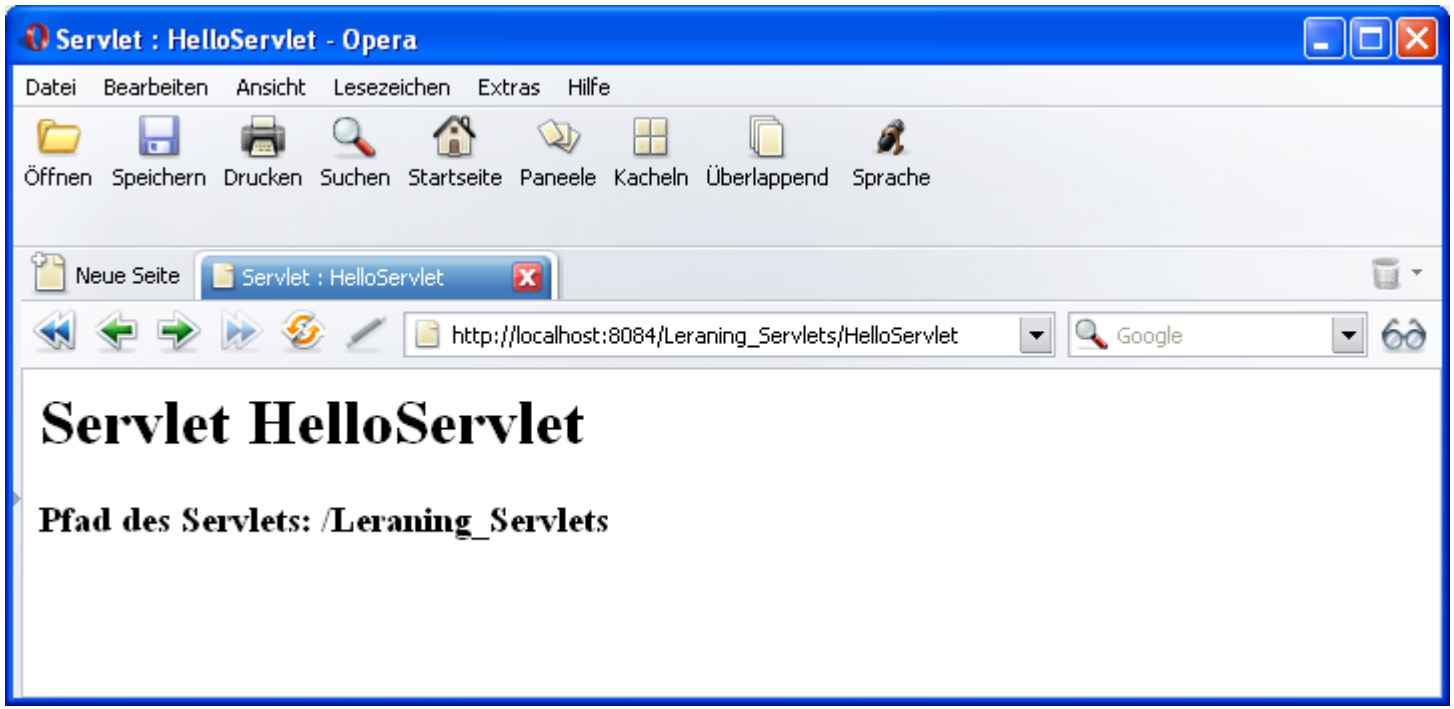
1. Einführung
2. Eigenschaften
3. **Beispiel „Hello Servlet“**
4. Lebenszyklus von Java - Servlets
5. Anfragebearbeitung mit und ohne Templateverfahren
6. Graphiken mit Java – Servlets
7. Sessiontracking
8. Der Servlet – Container
9. Webbezogene und Geschäftsbezogene Servlets
10. Einsatzgebiete der Webkomponenten
11. Vergleich JSP zu Java - Servlet
12. Fazits zu JSP ‘s und Java - Servlets
13. Zusammenfassung
14. Literatur
15. Ausblick

Beispiel Java-Servlet: HelloServlet (Schablone aus Netbeans generiert)

```
public class HelloServlet extends HttpServlet {

/** Processes requests for both HTTP <code>GET</code> and
        <code>POST</code> methods.
    * @param request servlet request
    * @param response servlet response
    */

protected void processRequest(HttpServletRequest request,
        HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet : HelloServlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet  HelloServlet </h1>");
        out.println("<h3>Pfad des Servlets: " +
        request.getContextPath () + "</h3>");
        out.println("</body>");
        out.println("</html>");
        out.close();
    }
```

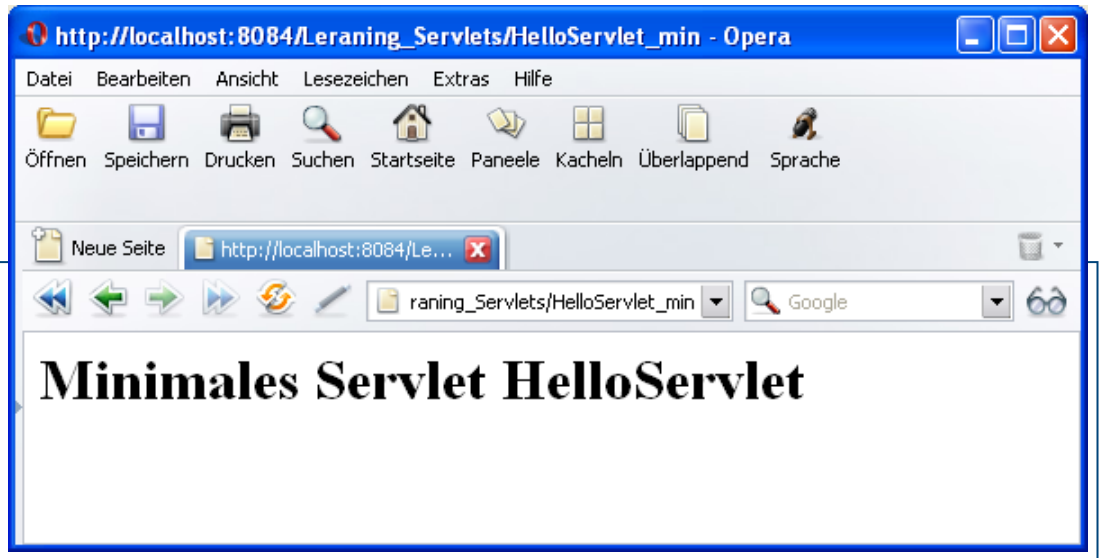
```
/** Handles the HTTP <code>GET</code> method.
    * @param request servlet request
    * @param response servlet response
    */
protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Handles the HTTP <code>POST</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doPost(HttpServletRequest request,
        HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

}
```

Ausgabe HelloServlet



Aufruf mit http://localhost:8084/Leraning_Servlets/HelloServlet
- direktes Ansprechen des Servlets (Präsentationsebene)

Beispiel Java-Servlet: HelloServlet (minimalistische Form)




```
import java.io.*;
import java.net.*;

import javax.servlet.*;
import javax.servlet.http.*;


public class HelloServlet_min extends HttpServlet {

   protected void doGet(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      out.println("<h1>Minimales Servlet  HelloServlet </h1>");
      out.close();
   }
}
```

Serverseitige Implementierungstechnologien (II): Java Servlets

1. Einführung
2. Eigenschaften
3. Beispiel „Hello Servlet“
4. **Lebenszyklus von Java - Servlets**
5. Anfragebearbeitung mit und ohne Templateverfahren
6. Graphiken mit Java – Servlets
7. Sessiontracking
8. Der Servlet – Container
9. Webbezogene und Geschäftsbezogene Servlets
10. Einsatzgebiete der Webkomponenten
11. Vergleich JSP zu Java - Servlet
12. Fazits zu JSP ‘s und Java - Servlets
13. Zusammenfassung
14. Literatur
15. Ausblick

Lebenszyklus von Java Servlets

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

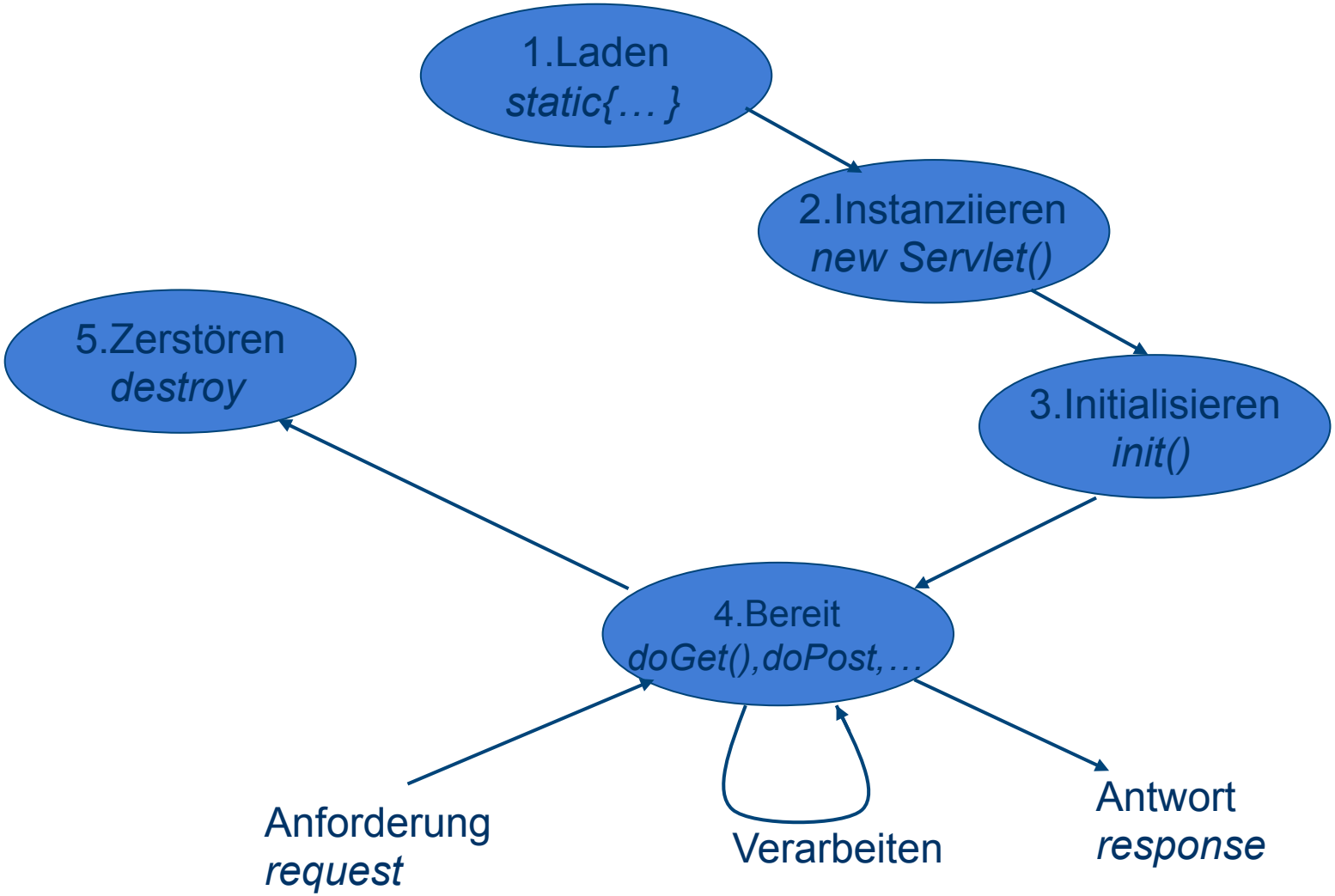
2. **Instanziieren des Servlet-Objektes**
   - parameterloser Konstruktor wird aufgerufen

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

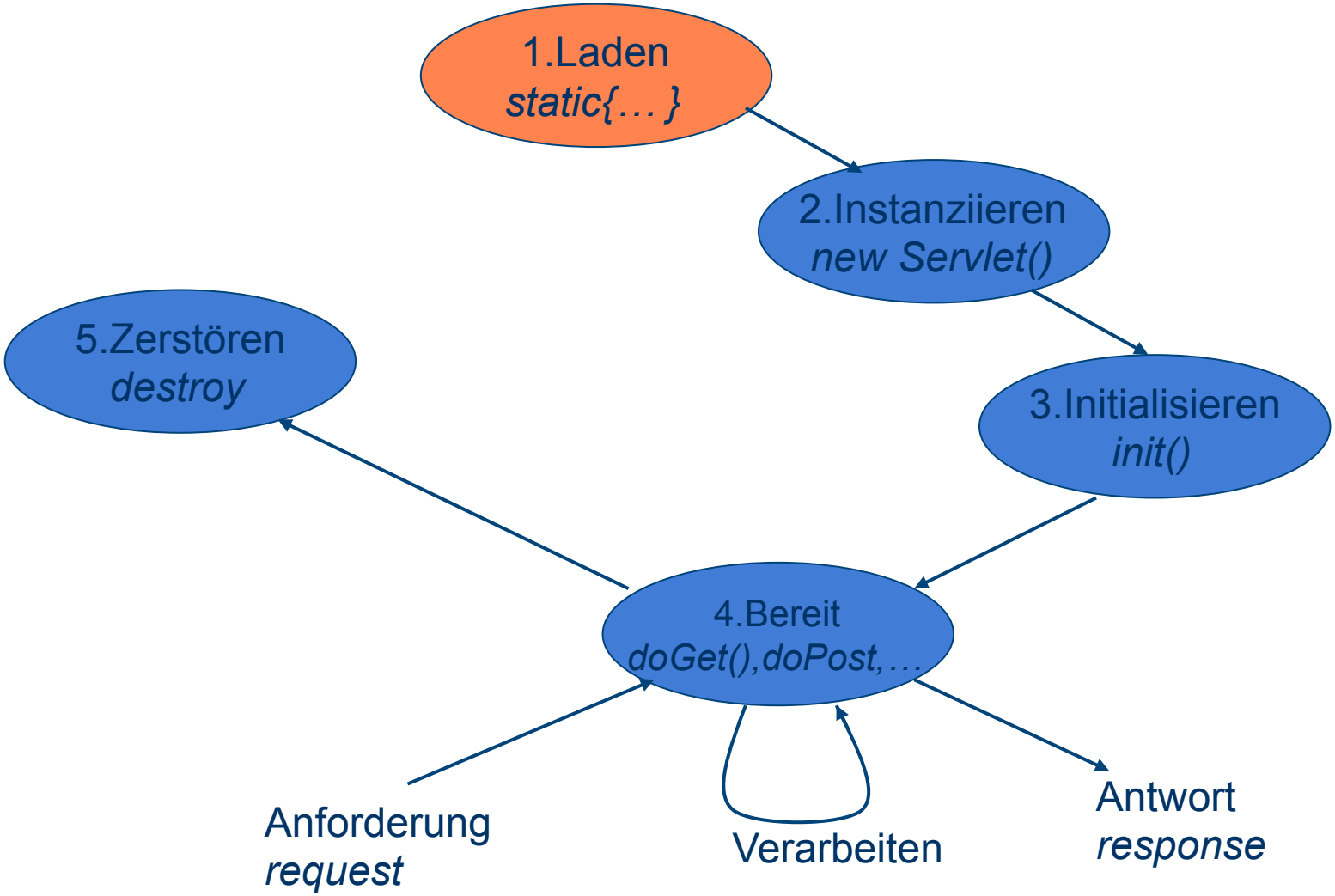
Lebenszyklus von Java Servlets



Java-Servlet – Schablone mit allen Lebenszyklen -

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

public class Schablone extends HttpServlet {

  static{}                          // Laden
  public Schablone(){  }            // Instanziieren
  public void init() {  }           // Initialisieren

  public void doGet(HttpServletRequest request, HttpServletResponse response) {  }
  public void doPost(HttpServletRequest request, HttpServletResponse response) {  }
  public void doHead(HttpServletRequest request, HttpServletResponse response) {  }
  // weitere : doPut(), doDelete()

  public void destroy() {  }

}
```

Servlet laden



Servlet laden

Servlet-Container lädt Servlet

- bei erster Anforderung an das Servlet oder
- beim Starten des Containers, falls in web.xml so konfiguriert
  (Web Application Deployment Descriptor; `<load-on-startup>`)

```
<servlet>
    <servlet-name>CSI</servlet-name>
    <servlet-class>de.becker.csi.CSIServlet</servlet-class>
    <init-param>
      <param-name>mapping</param-name>
      <param-value>/Wap/CSI</param-value>
    </init-param>
    <init-param>
      <param-name>urlEncoder</param-name>
      <param-value>de.becker.csi.servlet.CSIURLEncoder</param-value>
    </init-param>
    <load-on-startup>4</load-on-startup>
  </servlet>
```

Ausschnitt aus einer web.xml

Beispielprogramm: Servlet laden

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

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 des this-Objektes -> Thread bleibt aktiv…
         zaehler += tick;  //…solange das Servlet-Objekt lebt; Erhöhung des Milisekundenzählers
         try {Thread.sleep(tick);}  // Thread schläft „tick“-Milisekunden lang
         catch (InterruptedException ex) {}
      }
   }  //…weiter auf der folgenden Seite
```

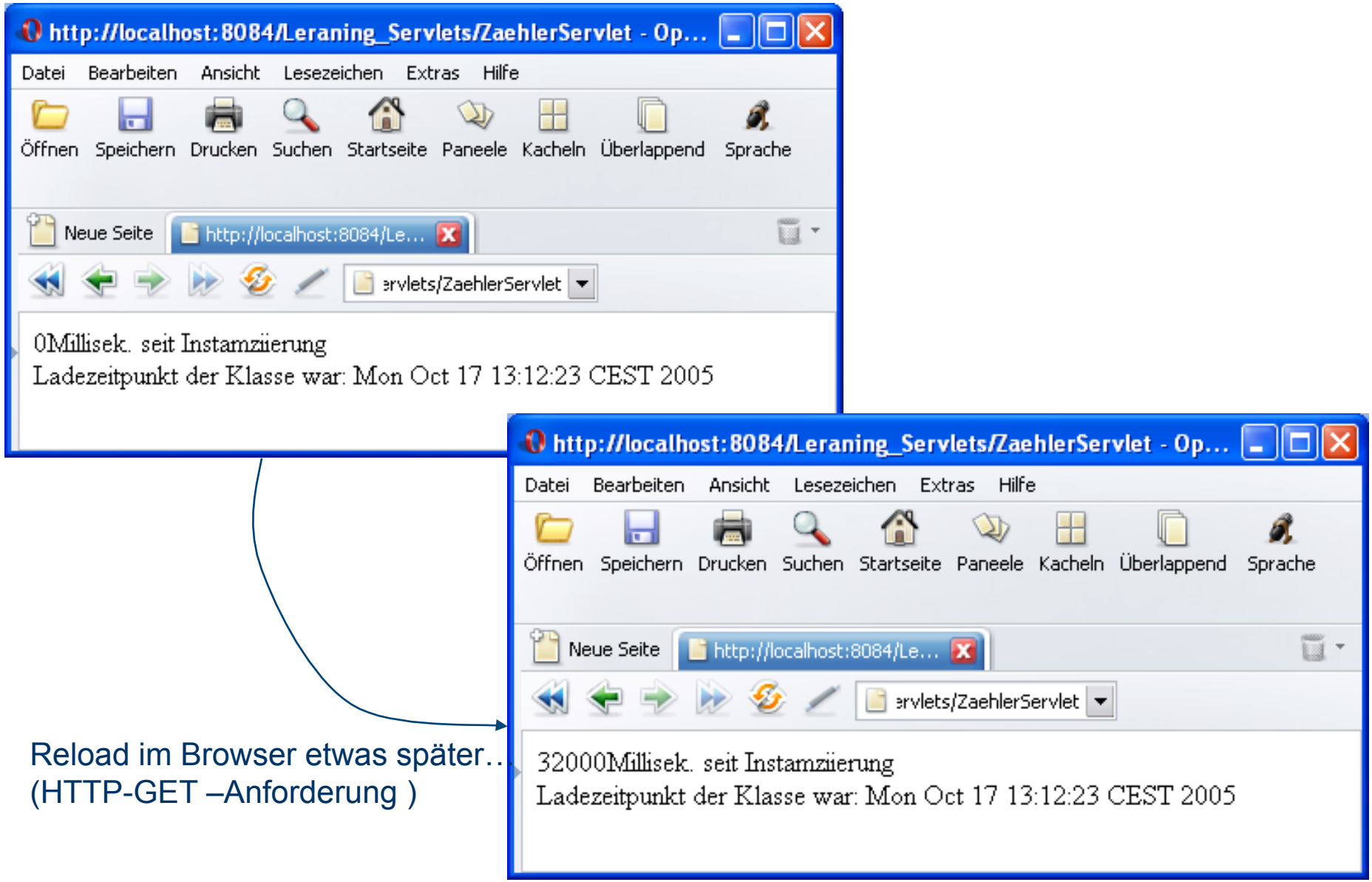
Beispielprogramm: Servlet laden

```
//…weiter von vorheriger Seite

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);
   }
} // doGet wird bei Aufruf des Servlets aus dem Browser aufgerufen
  // gibt Starttermin des Servlets und aktuellen Zählerstand aus.
```

Ausschnitt aus web.xml :

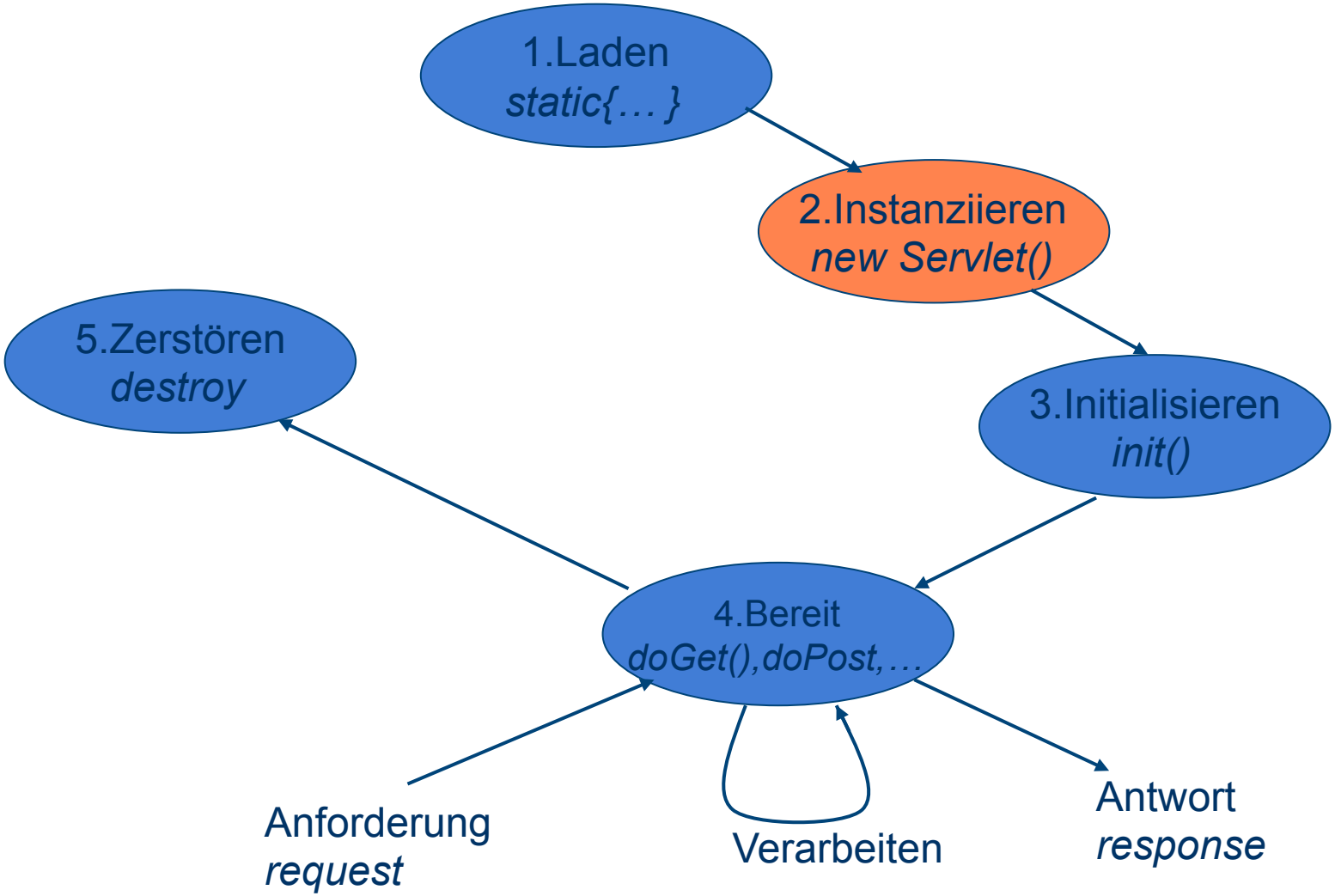
```
<servlet>
  <servlet-name>ZaehlerServlet</servlet-name>  //servlet-name in Netbeans automatisch generiert
  <servlet-class>ZaehlerServlet</servlet-class>    // automatisch generiert
  <init-param>
  <param-name>tick</param-name>     // Initialisierungsparameter wird festgelegt mit Name…
  <param-value>1000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

Beipielprogramm: Servlet laden



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

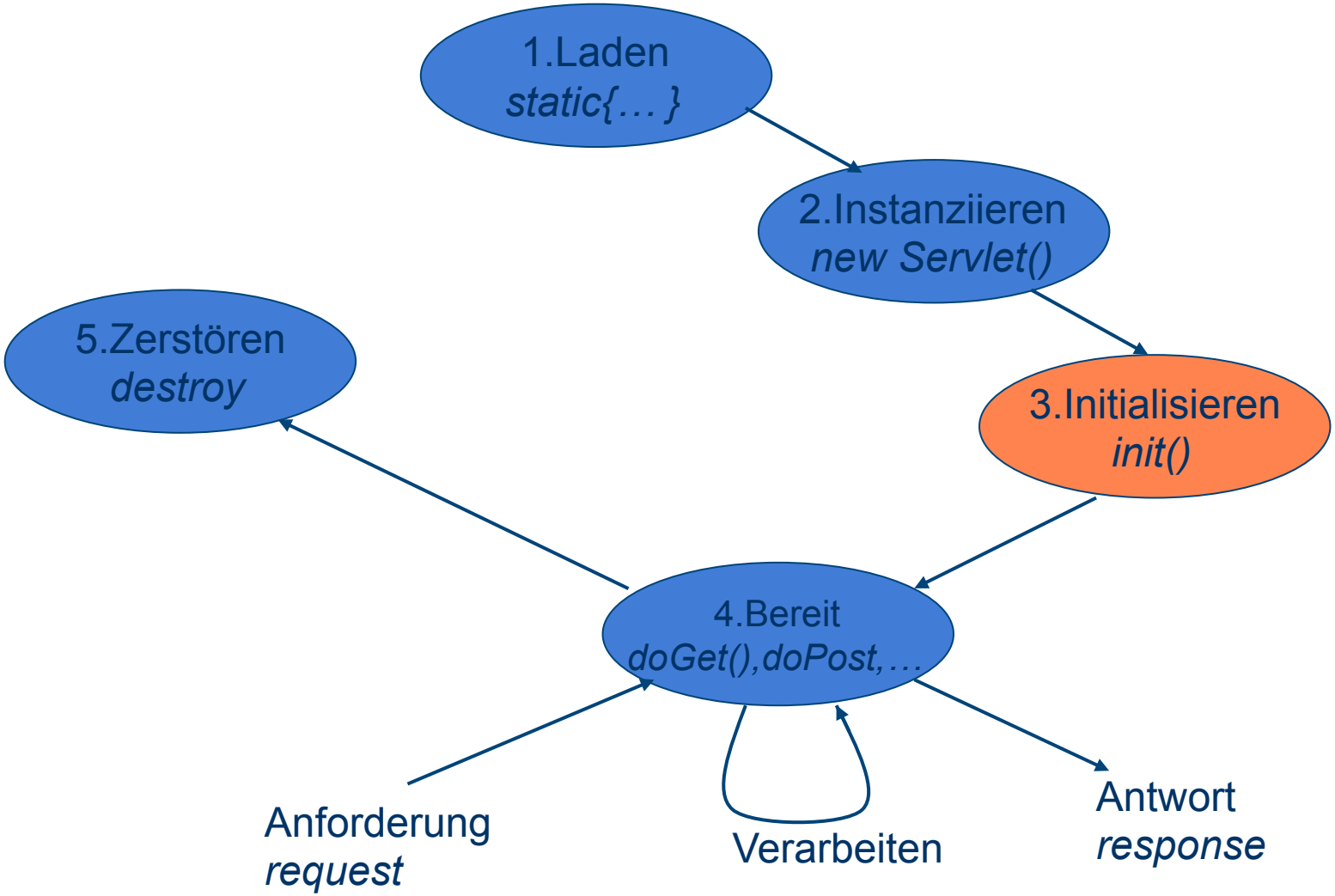
Instanziieren



Lebenszyklus von Java Servlets: 2. Instanziieren

- Nach Laden der Servlet-Klasse wird automatisch instanziiert.
- Standardkonstruktor kann auch überschrieben werden z.B. mit Erstellen einer Datenbankverbindung.
- 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, da jede Anforderung in einem eigenen Thread abgearbeitet wird.-> Konflikte bei gemeinsamen Ressourcen möglich

Lebenszyklus von Java Servlets



Lebenszyklus von Java Servlets: 3. Instanz initialisieren

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

public class InitServlet extends HttpServlet {
   String servletname;
   Date instanziierungsdatum;
   public void init(ServletConfig sc) throws ServletException {   //Aufrufen und Überschreiben der
                                                                   // Funktion init() od. init(ServletConfig)
         super.init(sc);
         instanziierungsdatum = new Date();
         servletname = sc.getServletName();
   }

   protected void doGet(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      out.println("Servlet " + servletname + " geladen am " + instanziierungsdatum);
       out.close();
   }
}
```

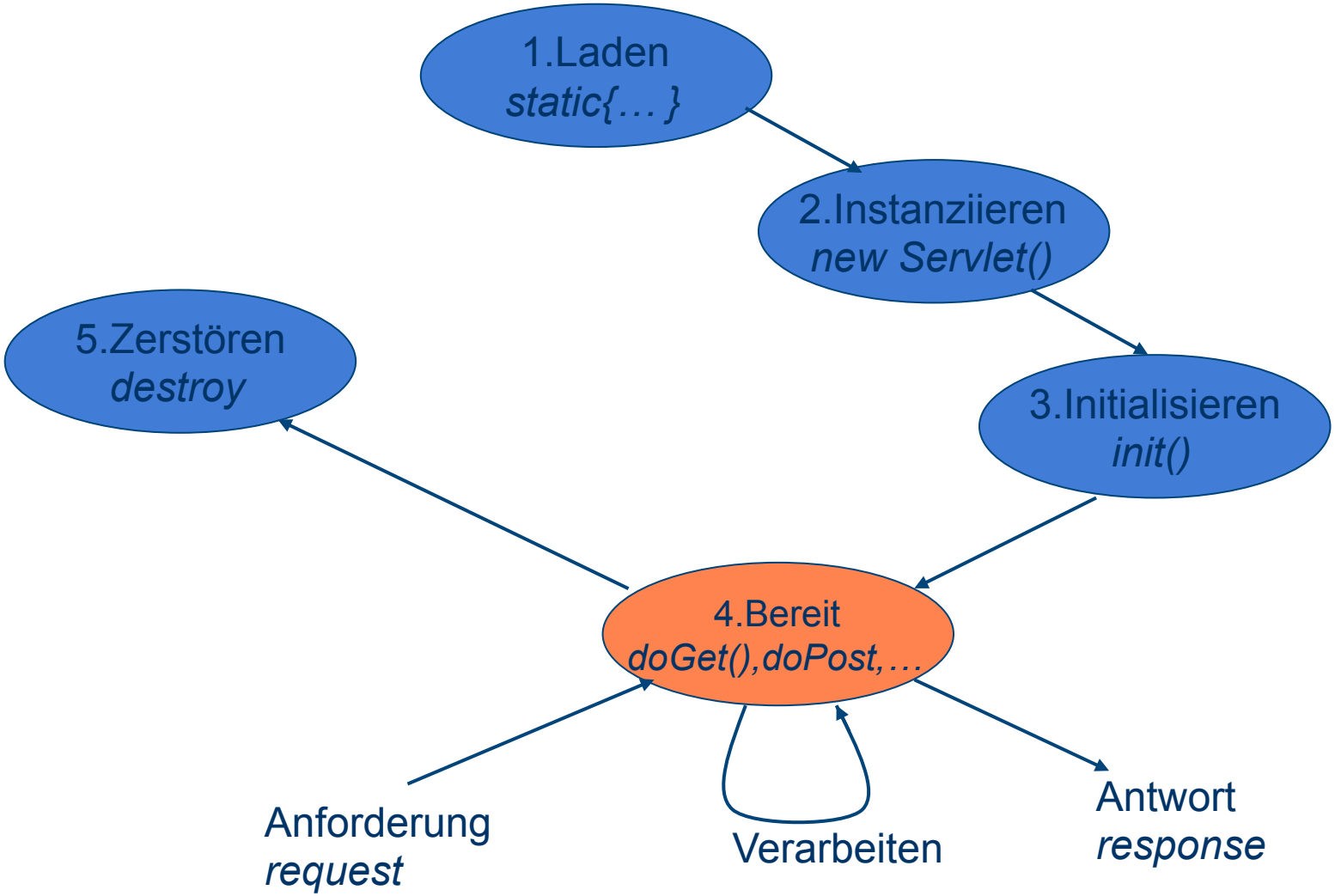
Lebenszyklus von Java Servlets: 3. Instanz initialisieren

Bei jedem weiteren Aufruf der Seite Aus dem Browser bleibt der Ladezeitpunkt erhalten.

In der Methode init() oder init(ServletConfig) lassen sich Vorbereitungsmaßnahmen für den laufenden Betrieb vornehmen z.B.:

- Datenbankverbindung einrichten
- Einlesen von Initialisierungsparametern aus web.xml

Lebenszyklus von Java Servlets
4. Anforderungen bearbeiten



Lebenszyklus von Java Servlets
4. Anforderungen bearbeiten

- zentraler Teil des Servlets: Annahme und Bearbeitung der Client-Anfragen

- HTTP-Protokoll kennt 7 verschiedene Methoden:
  GET, POST, HEAD, PUT, DELETE, OPTIONS, TRACE

- Instanzmethoden der Klasse *HttpServlet():*
  - *doGet()* Dokumentenanforderung
    Daten werden als Querystring mitgesendet
  - *doPost()* Versenden von Daten zum Verarbeiten an den Server
    Daten werden als Formular in HTTP-Entity mitgesendet
  - *doHead()* wie doGet() , ohne Erwarten von Antwortdaten
  - *doPut()* Daten sollen an URL- entsprechenden Ort gespeichert werden
  - *doDelete()* Datei, auf welche die URL weist, soll gelöscht werden
  - *doOptions()* Anfrage nach Anforderungsmethoden oder and. Optionen
  - *doTrace()* zeichnet die Zwischenstationen der Anfrage auf (z.B. Proxy)

Mitgesendete Daten in *doGet()* und *doPost()* können mittels der Methoden ***getParameter(String)*** *und* ***getParameterNames()*** aus dem **HttpServletRequest-Objekt** eingelesen werden.

Lebenszyklus von Java Servlets
4. Anforderungen bearbeiten

- nur *doGet()* und *doPost()* liefern ein Webdokument zurück
- Überschreiben nur einer *doXXX()*-Methode notwendig.
- Bei Freilassen beider Methoden *doGet()* und *doPost()* aber gleicher Funktionalität kann Die Implementierung einmal erfolgen und die jeweils andere Methode direkt aufgerufen werden, Beispiel:

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

public class GetPostServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
/* Implementierung*/
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
   doGet(request, response);
    }
}
```

Lebenszyklus von Java Servlets
4. Anforderungen bearbeiten

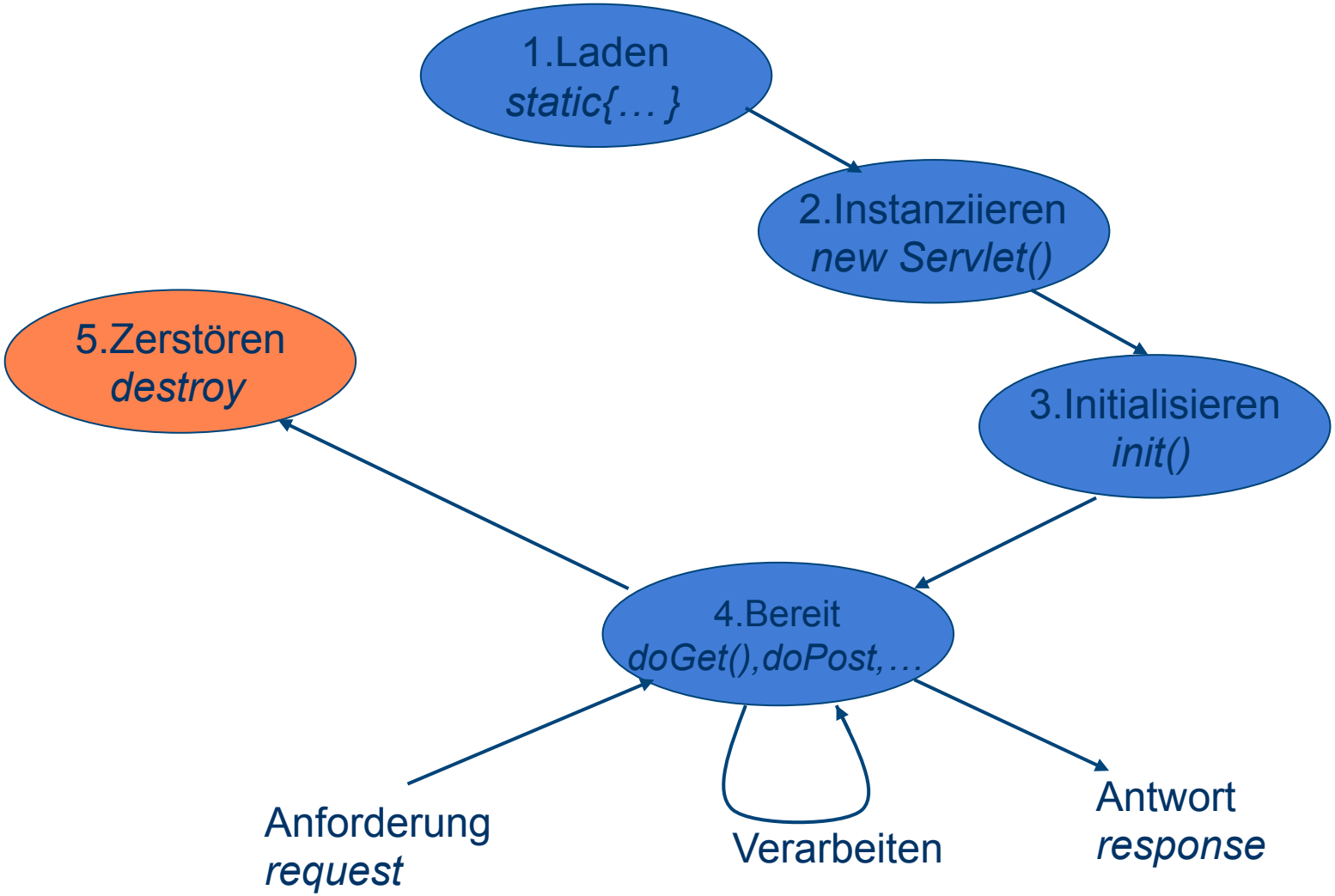
• Lösung in Netbeans mit der Methode *ProcessRequest():*

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

public class HelloServlet extends HttpServlet {

   protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    /*Implementierungen*/
}

   protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

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

Lebenszyklus von Java Servlets
4. Anforderungen bearbeiten

- Methoden *doXxx()* erschließen auch die **Kontexte** der Anforderungen und Antworten*:*
  - ***PrintWriter*-Objekt** übermittelt Antwort
    PrintWriter out = response.getWriter();
    out.print(methode);
  - ***Session*-Objekt** kommt aus Anforderungsobjekt und enthält Angaben zur Sitzungsverwaltung
    HttpSession session = request.getSession();
  - **Servlet-Konfiguration** enthält den Servlet-Kontext und Initialisierungsparameter
    ServletConfig config = this.getServletConfig();
  - **Kontext der WebApplikation (**z.B. Version der Servlet-API)
    ServletContext context = config.getServletContext(); oder
    ServletContext context = this.getServletContext();

Lebenszyklus von Java Servlets: 5. Instanz liquidieren



Lebenszyklus von Java Servlets: 5. Instanz liquidieren

Bei Liquidierung eines Servlets ruft der Container die Methode ***destroy()*** auf.
Diese Methode kann überschrieben werden, mit dem Ziel:

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

Nachfolgendes Beispiel (folgende Seite…) schreibt Zähler für Zugriffe auf Servlet und speichert diese fortlaufend auf der Festplatte.
(Methode *destroy()* wird leider nicht aufgerufen, wenn der Webserver abstürzt, zusätzliche Absicherung notwendig!)

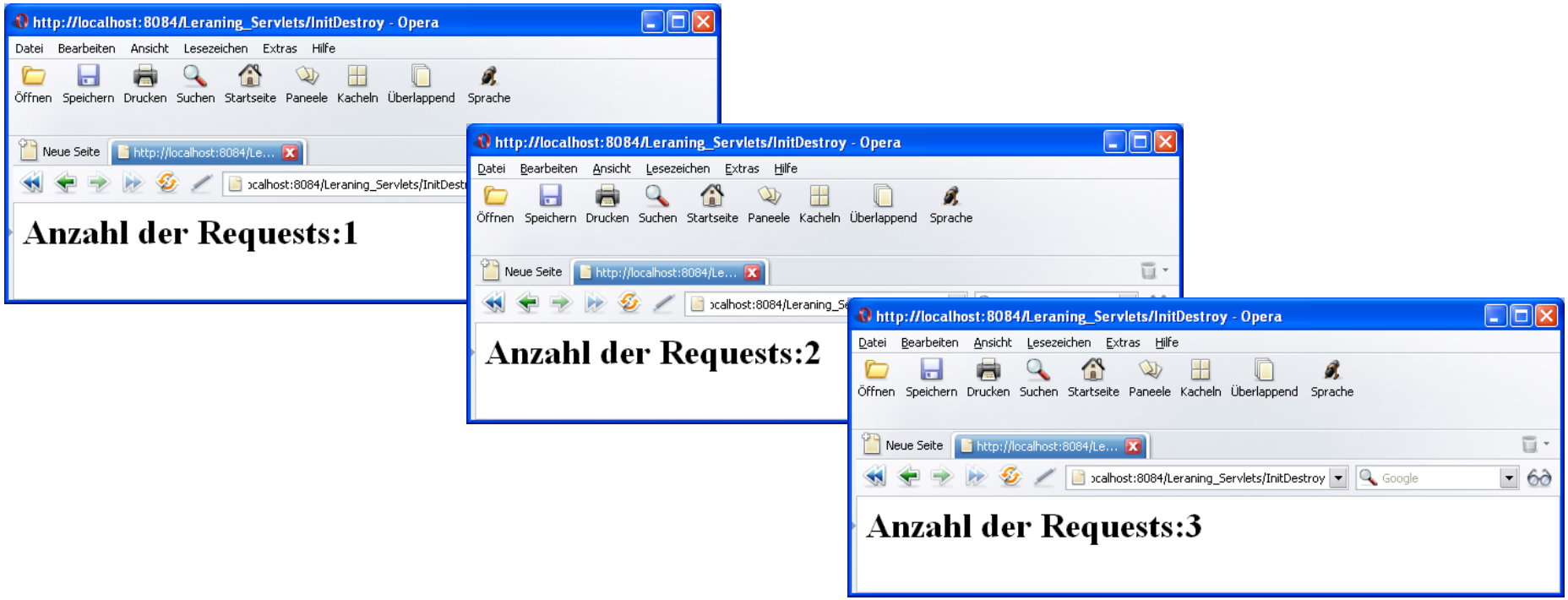
Lebenszyklus von Java Servlets: 5. Instanz liquidieren

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

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){ }
   }
   … weiter
```

```
   weiter…

   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.

Lebenszyklus von Java Servlets: 5. Instanz liquidieren

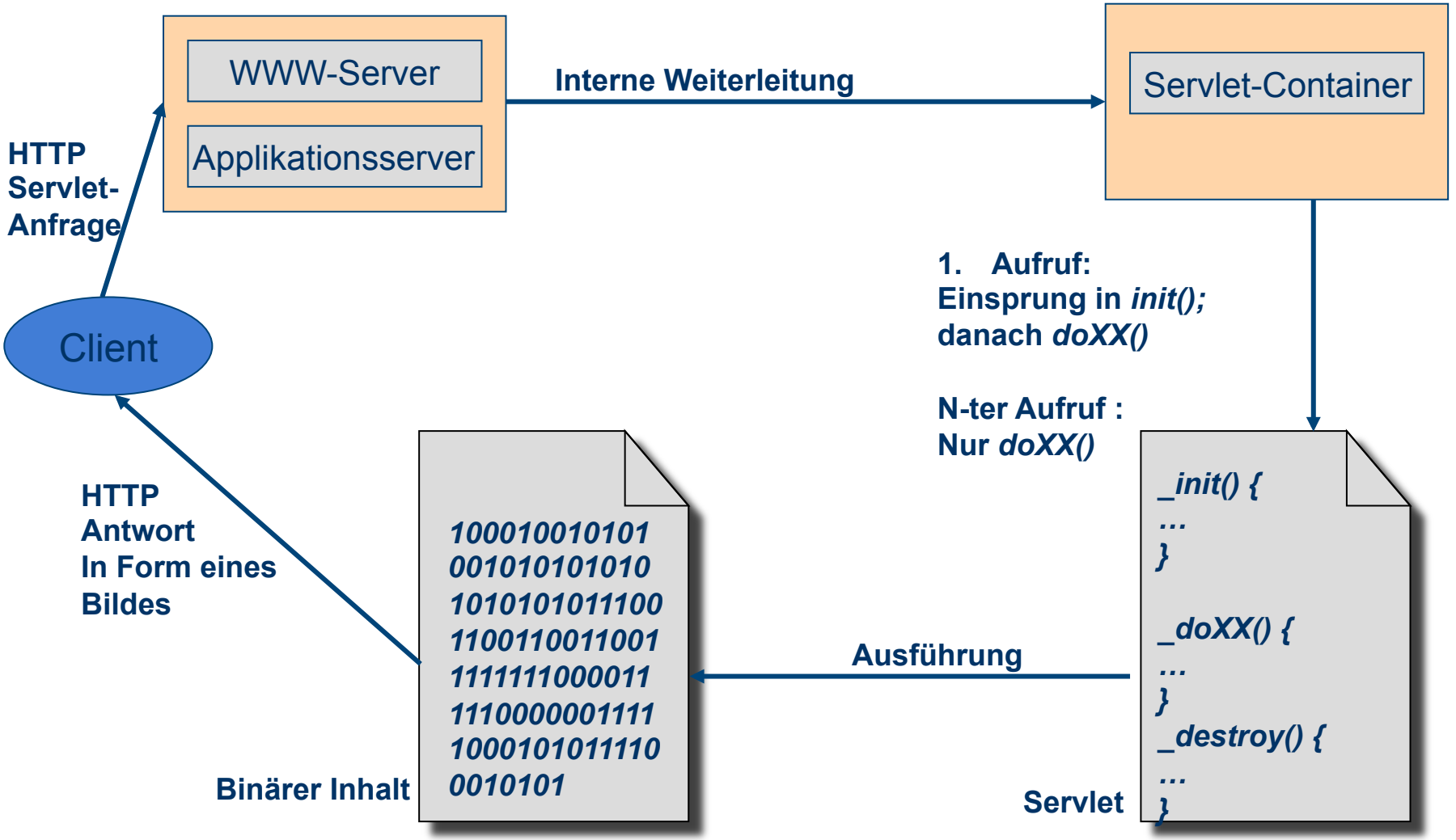


Bei jedem Reload oder Neuanfrage im Browser wird der Zähler um 1 hochgezählt. Auch keine Unterbrechung im Hochzählen durch Stoppen und Starten des Webservers.

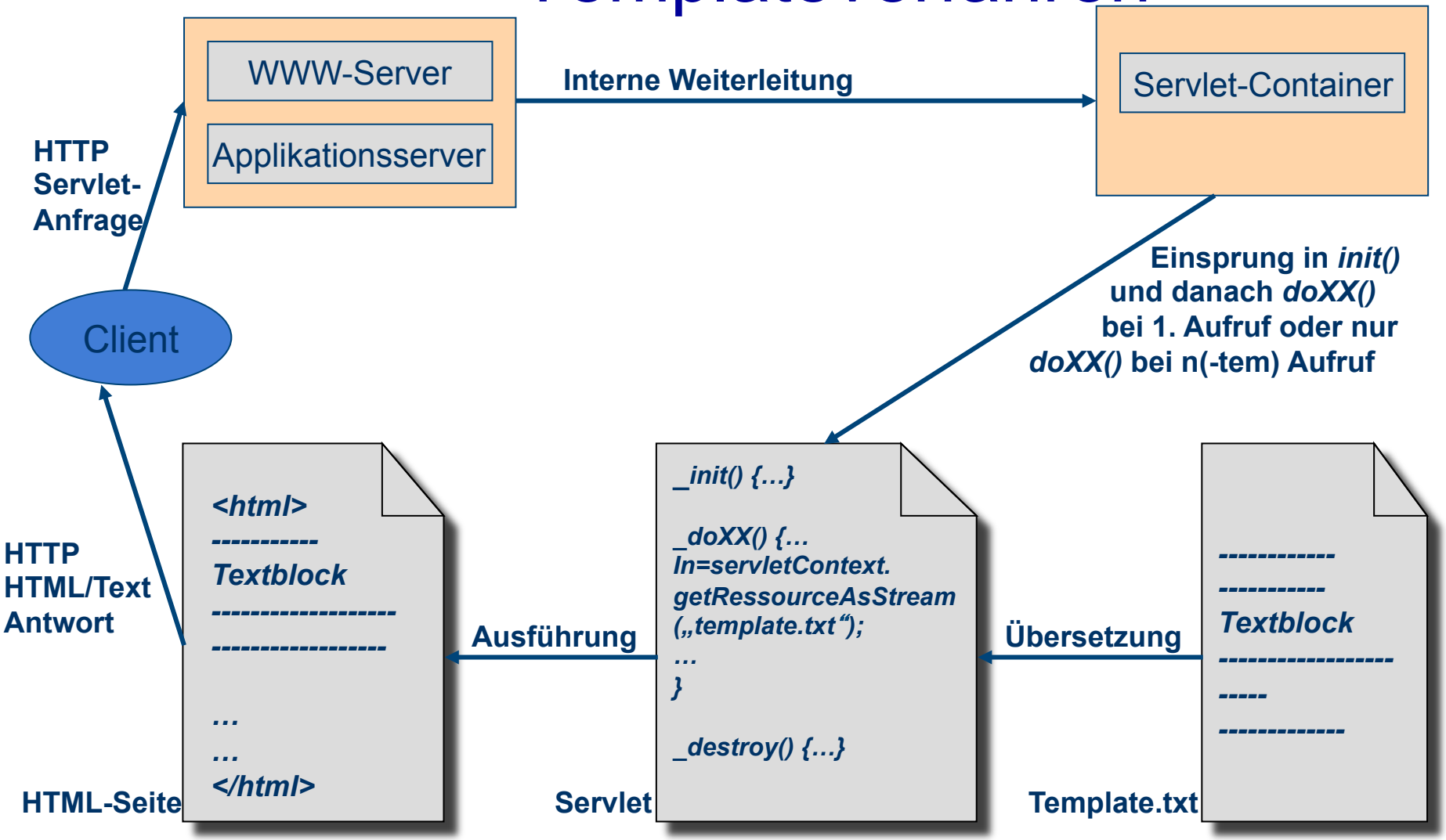
Serverseitige Implementierungstechnologien (II): Java Servlets

1. Einführung
2. Eigenschaften
3. Beispiel „Hello Servlet“
4. Lebenszyklus von Java - Servlets
5. **Anfragebearbeitung mit und ohne Templateverfahren**
6. Graphiken mit Java – Servlets
7. Sessiontracking
8. Der Servlet – Container
9. Webbezogene und Geschäftsbezogene Servlets
10. Einsatzgebiete der Webkomponenten
11. Vergleich JSP zu Java – Servlet
12. Fazits zu JSP ‘s und Java - Servlets
13. Zusammenfassung
14. Literatur
15. Ausblick

Servlets: Anfrage an ein Servlet



Servlets: Anfrage an ein Servlet mit Templateverfahren



Servlets: Anfrage an ein Servlet mit Templateverfahren

- Erzeugung von Textdaten aus einem Servlet durch Templateverfahren (z.B. Header und Footer erzeugen)
- 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 (Mehrsprachigkeit von Webanwendungen)

Nachteil: Performanz des Servers insgesamt wird beeinträchtigt

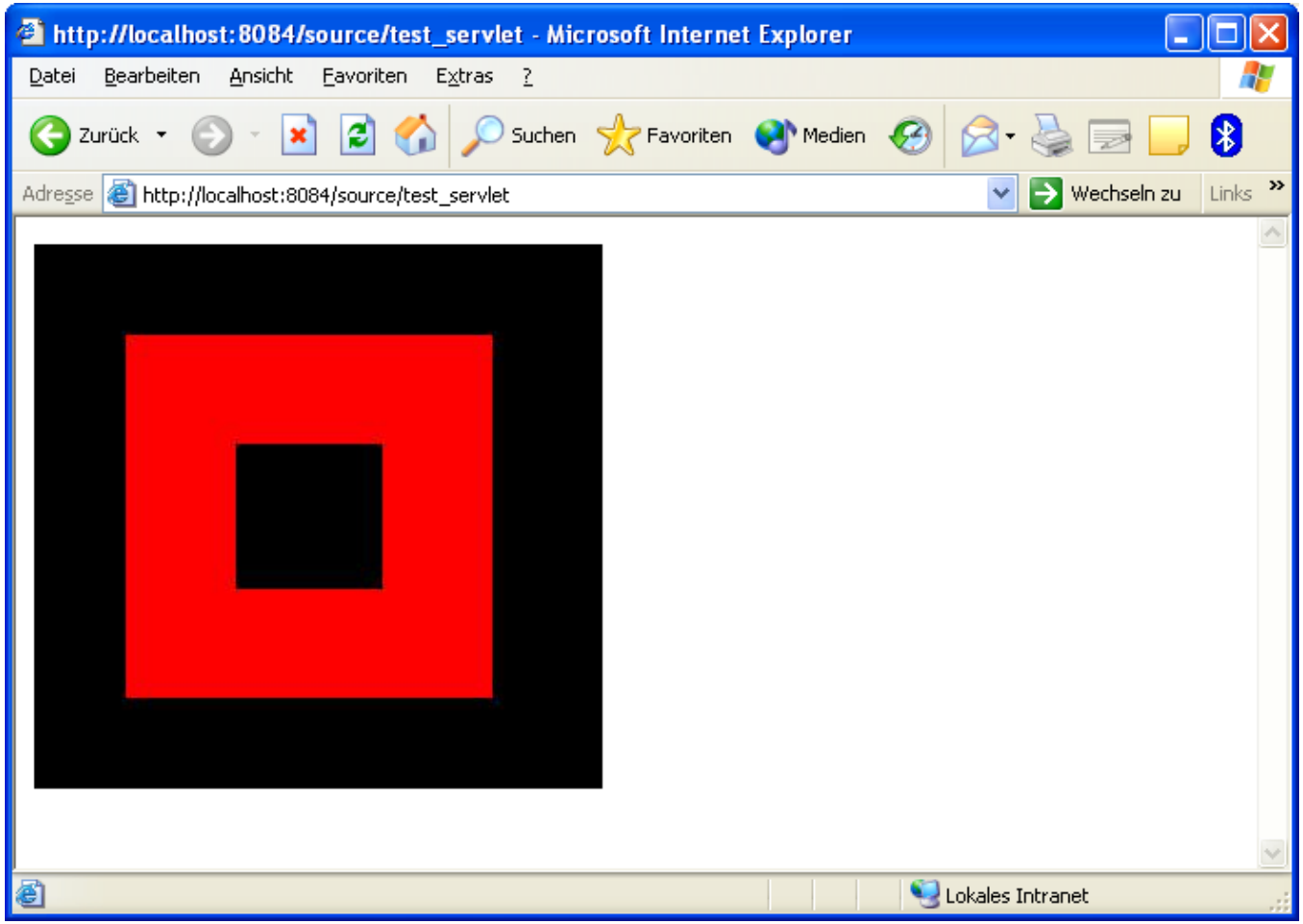
Serverseitige Implementierungstechnologien (II): Java Servlets

1. Einführung
2. Eigenschaften
3. Beispiel „Hello Servlet“
4. Lebenszyklus von Java - Servlets
5. Anfragebearbeitung mit und ohne Templateverfahren
6. **Graphiken mit Java – Servlets**
7. Sessiontracking
8. Der Servlet – Container
9. Webbezogene und Geschäftsbezogene Servlets
10. Einsatzgebiete der Webkomponenten
11. Vergleich JSP zu Java - Servlet
12. Fazits zu JSP ‘s und Java - Servlets
13. Zusammenfassung
14. Literatur
15. Ausblick

Servlet: Graphik-Beispiel

```
import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.geom.*;
import java.awt.image.BufferedImage;
import com.sun.image.codec.jpeg.*;

import javax.servlet.*;
import javax.servlet.http.*;

public class ImageServlet extends HttpServlet {

protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("image/jpeg");
        OutputStream out = response.getOutputStream();
        JPEGImageEncoder encoder = JPEGCodec.createJPEGEncoder(out);
       BufferedImage bufimg = new BufferedImage(310,300, BufferedImage.TYPE_BYTE_INDEXED);
       Graphics2D graphics = bufimg.createGraphics();

       graphics.setBackground(Color.white);
       graphics.setPaint(Color.red);
       graphics.fill(new Rectangle2D.Double(50, 50, 200, 200));
       graphics.setPaint(Color.black);
       graphics.fill(new Rectangle2D.Double(110, 110, 80, 80));
       encoder.encode(bufimg);
       out.close();
    }
}
```

Ergebnisseite des Beispiels



Serverseitige Implementierungstechnologien (II): Java Servlets

1. Einführung
2. Eigenschaften
3. Beispiel „Hello Servlet“
4. Lebenszyklus von Java - Servlets
5. Anfragebearbeitung mit und ohne Templateverfahren
6. Graphiken mit Java – Servlets
7. **Sessiontracking**
8. Der Servlet – Container
9. Webbezogene und Geschäftsbezogene Servlets
10. Einsatzgebiete der Webkomponenten
11. Vergleich JSP zu Java - Servlet
12. Fazits zu JSP ‘s und Java - Servlets
13. Zusammenfassung
14. Literatur
15. Ausblick

Sessiontracking (Sitzungsverfolgung)

- **Überwindung der Zustandslosigkeit des HTTP-Protokolls**

- **<u>Techniken zur Sitzungsverfolgung:</u>**

  - Cookies `(cookie)`
  - URL-Rewriting
  - versteckte Felder
  - Objekt HttpSession aus Servlet API

1. **<u>Cookies</u>**
   - speichern Kennungen des Clients für den Server
   - Cookie-Objekt (`javax.servlet.http.Cookie`)
     - enthält diese Informationen über den Client
     - wird in den meisten Browsern gespeichert
     - und wird wieder an den Server zur Identifkation des Clients und der Sitzung gesendet
     - Achtung: Speicherung der Cookies im Browser kann fehlerhaft implementiert sein oder vom User abgestellt sein! (weiter ->…)

Sessiontracking (Sitzungsverfolgung)

1. **Cookies (weiter)**

   - `javax.servlet.http.HttpServletRequest` mit Methode `cookie[] request.getcookies()` gibt ein Array zurück, welches alle `cookie` - Objekte enthält, die der Client mit dem Request an den Server sendet
   - `javax.servlet.http.HttpServletResponse` mit der Methode `response.addCookie(Cookie cookie)` fügt dem Response ein Cookie hinzu

2. **URL-Rewriting**

   - Anhängen eines Parameters („Cookie-Parameter“) an die URL zur Sitzungskennzeichnung z.B.: `http://localhost:8084/Learning_Servlets/CookieSetzen;sessionID=xyz123`
   - funktioniert in jedem Browser
   - unschön: diese Parameter sind auch in den Bookmarks enthalten, obwohl zugehörige Sitzungen nicht erhalten bleiben.

Sessiontracking (Sitzungsverfolgung)

3. Versteckte Felder (engl. hidden fields)

- Sessionidentifikation über versteckte Felder in HTML-Formularen mit z.B. **`<INPUT TYPE="HIDDEN" NAME="session" VALUE="...">`** versenden
- Vorteil: keine URL-Anhängsel, keine Cookies

4. Session Objekt der Servlet API

- jede Sitzung ist mit einem Objekt der Klasse **`javax.servlet.http.HttpSession`** verbunden.
- in JSP ‘s gibt es das implizite Objekt **`Session`**
- **`Object getAttribute( String name )`** liefert das mit **`name`** verbundene Objekt; **`null`**, wenn es keine Assoziation gab.
- **`Enumeration getAttributeNames()`** liefert eine Aufzählung aller mit der Sitzung verbundenen Objekte.
- **`void setAttribute( String name, Object value )`** bindet **`name`** mit dem Objekt **`value`** an die Sitzung.
- **`void removeAttribute( String name )`** entfernt das Attribut von der Sitzung

Beispiel zum Sessiontracking

Inhalt des Beispiels:

1) Erstellen eines Sessionobjekts
2) Ausgabe von:
   - Session ID
   - Zeitpunkt des letzten Zugriffs
   - Anzahl der Zugriffe

Zu 1) `HttpSession session = request.getSession(true);`

- Anlegen eines neuen oder Aufruf eines bestehenden Objekts

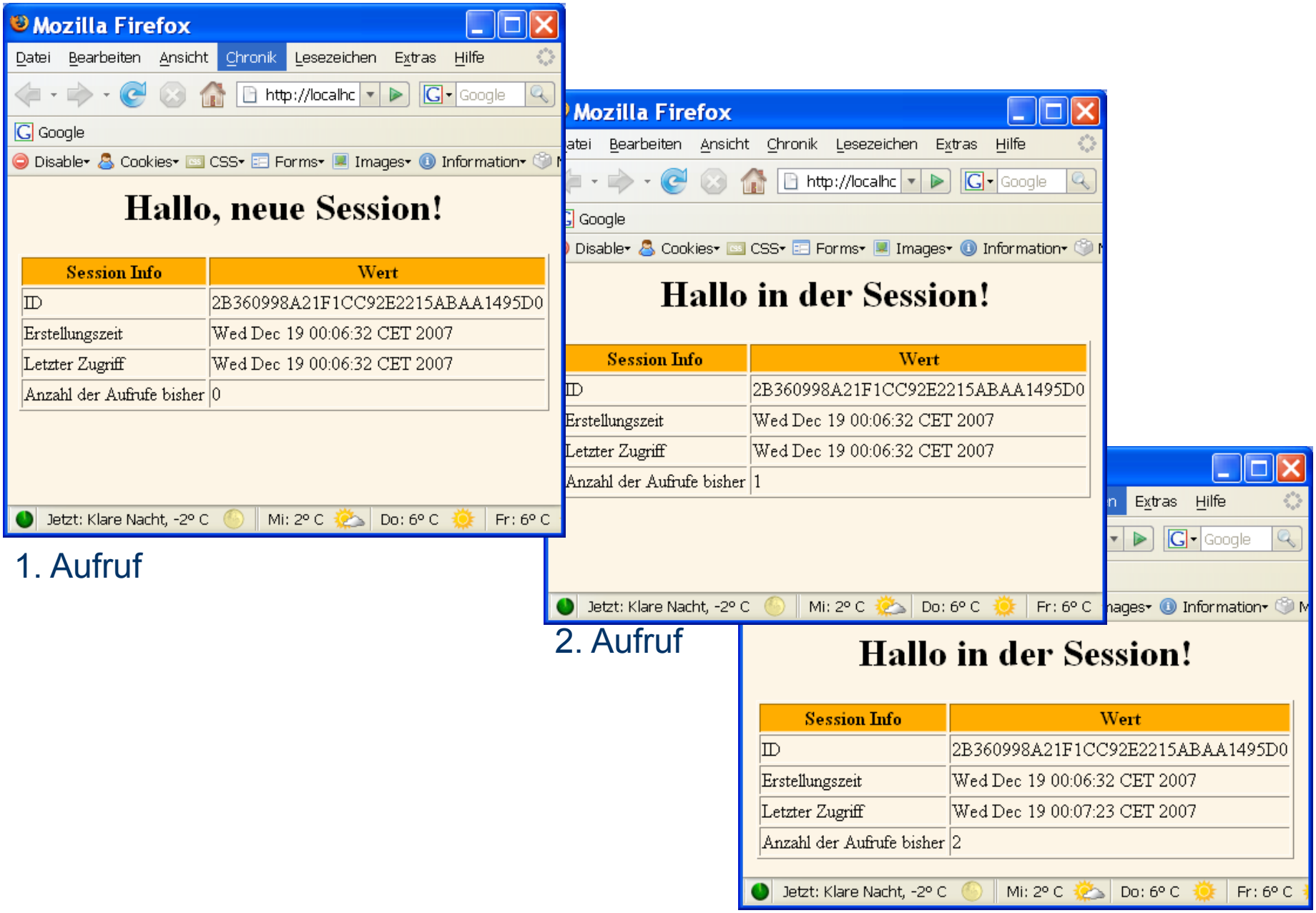
Zu 2) `session.getId()`
`session.getCreationTime()`
`session.getLastAccessedTime()`
Anlegen und Hochzählen eines `accessCount:`

Beispiel zum Sessiontracking

Anlegen und Hochzählen eines accessCount:

```
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);
```

Nutzen der Methoden `getAttribute()` und `setAttribute()`


Mozilla Firefox
Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe
Hallo, neue Session!
Session Info | Wert
ID | 2B360998A21F1CC92E2215ABAA1495D0
Erstellungszeit | Wed Dec 19 00:06:32 CET 2007
Letzter Zugriff | Wed Dec 19 00:06:32 CET 2007
Anzahl der Aufrufe bisher | 0
Jetzt: Klare Nacht, -2° C
Mi: 2° C
Do: 6° C
Fr: 6° C
1. Aufruf
Hallo in der Session!
Session Info | Wert
ID | 2B360998A21F1CC92E2215ABAA1495D0
Erstellungszeit | Wed Dec 19 00:06:32 CET 2007
Letzter Zugriff | Wed Dec 19 00:06:32 CET 2007
Anzahl der Aufrufe bisher | 1
2. Aufruf
Hallo in der Session!
Session Info | Wert
ID | 2B360998A21F1CC92E2215ABAA1495D0
Erstellungszeit | Wed Dec 19 00:06:32 CET 2007
Letzter Zugriff | Wed Dec 19 00:07:23 CET 2007
Anzahl der Aufrufe bisher | 2
3. Aufruf

Beispiel Session Tracking
- komplettes Listing-

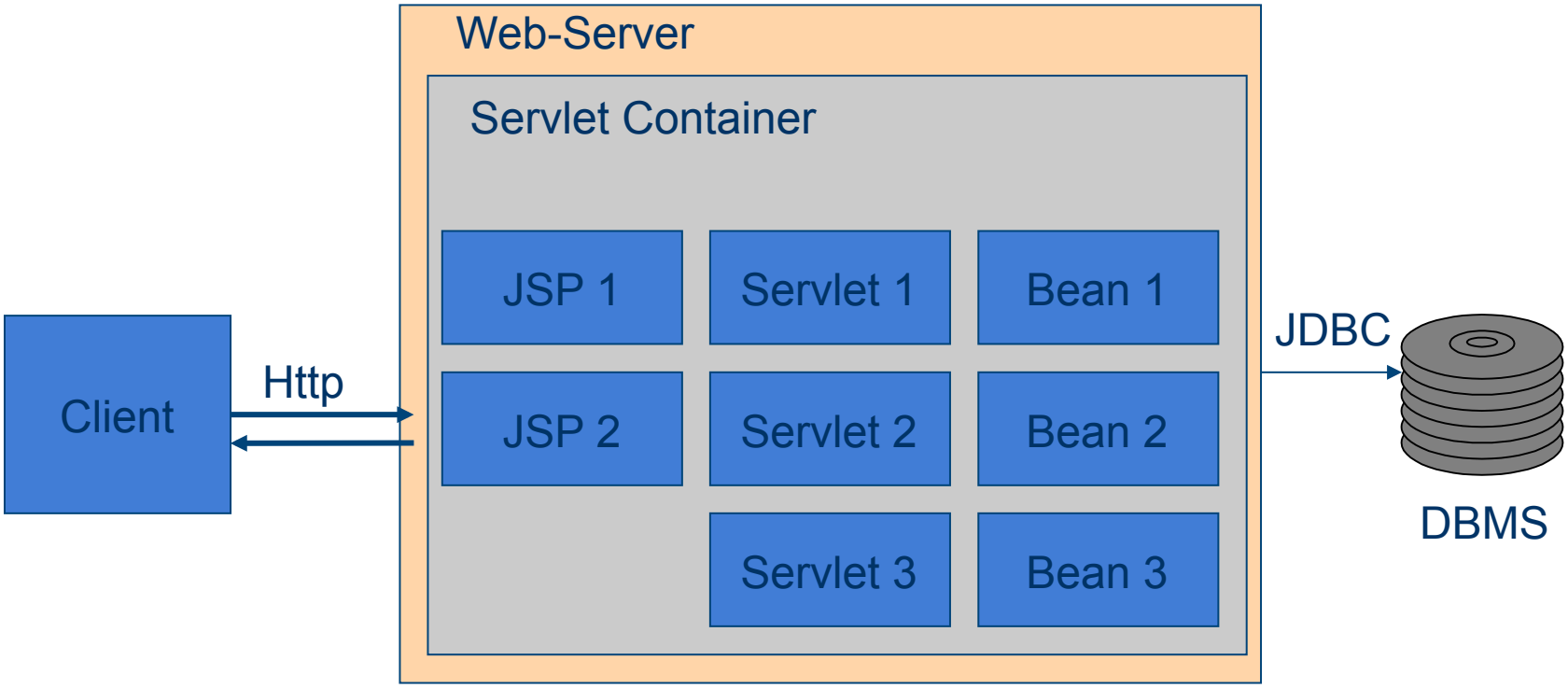
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.net.*;
import java.util.*;

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);
  }
}
```

Serverseitige Implementierungstechnologien (II): Java Servlets

1. Einführung
2. Eigenschaften
3. Beispiel „Hello Servlet“
4. Lebenszyklus von Java - Servlets
5. Anfragebearbeitung mit und ohne Templateverfahren
6. Graphiken mit Java – Servlets
7. Sessiontracking
8. **Der Servlet – Container**
9. Webbezogene und Geschäftsbezogene Servlets
10. Einsatzgebiete der Webkomponenten
11. Vergleich JSP zu Java - Servlet
12. Fazits zu JSP ‘s und Java - Servlets
13. Zusammenfassung
14. Literatur
15. Ausblick

Servlet - Container



Servlets werden von einem Servlet-Container ausgeführt.

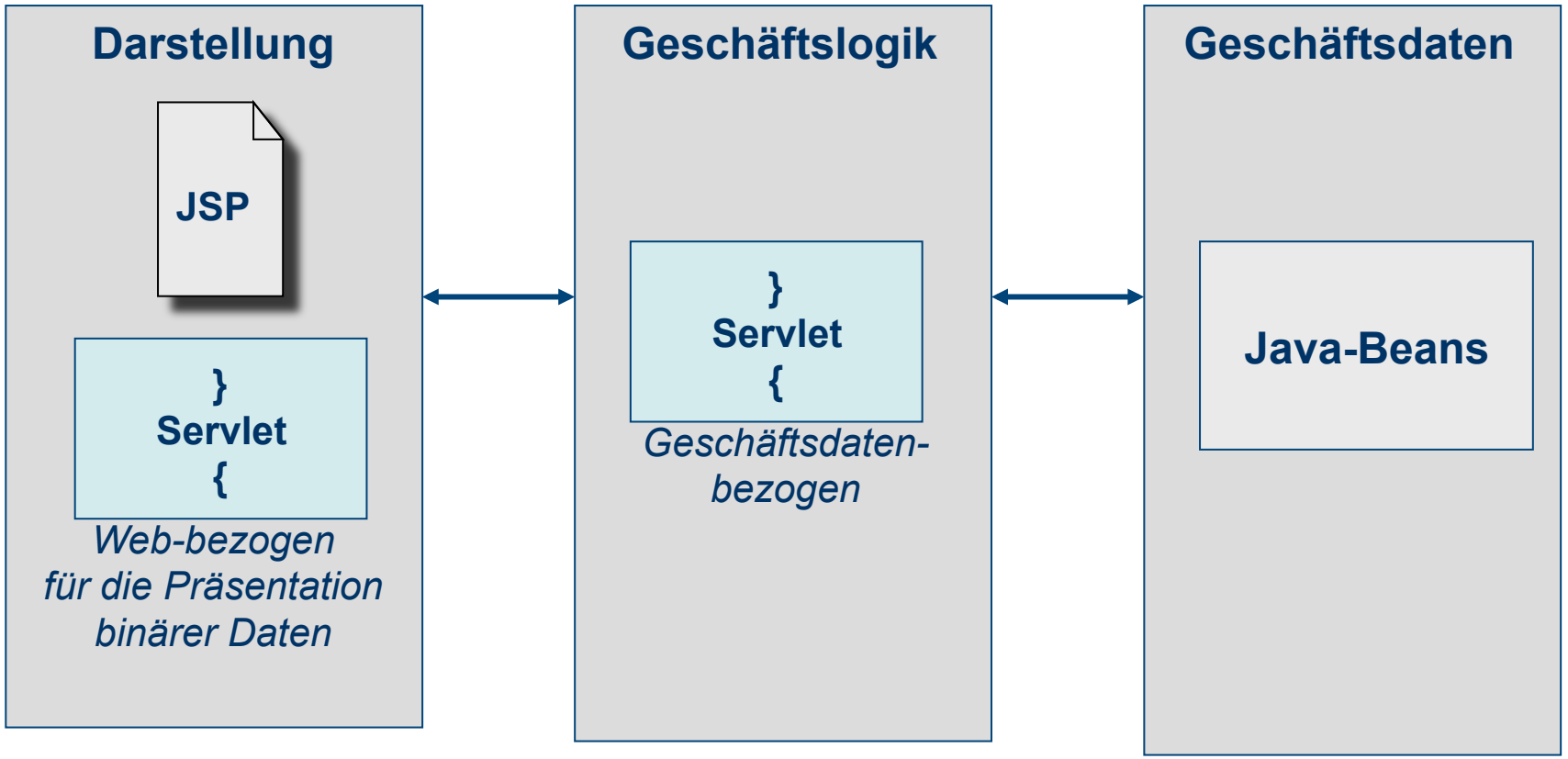
Aufgaben des Servlet - Containers

- Kommunikation mit den Servlet – Komponenten
- Lifecycle – Management der Komponenten
- Multithreading
- Methodenaufrufe bei Client - Anfragen

Serverseitige Implementierungstechnologien (II): Java Servlets

1. Einführung
2. Eigenschaften
3. Beispiel „Hello Servlet“
4. Lebenszyklus von Java - Servlets
5. Anfragebearbeitung mit und ohne Templateverfahren
6. Graphiken mit Java – Servlets
7. Sessiontracking
8. Der Servlet – Container
9. **Webbezogene und Geschäftsbezogene Servlets**
10. Einsatzgebiete der Webkomponenten
11. Vergleich JSP zu Java - Servlet
12. Fazits zu JSP ‘s und Java - Servlets
13. Zusammenfassung
14. Literatur
15. Ausblick

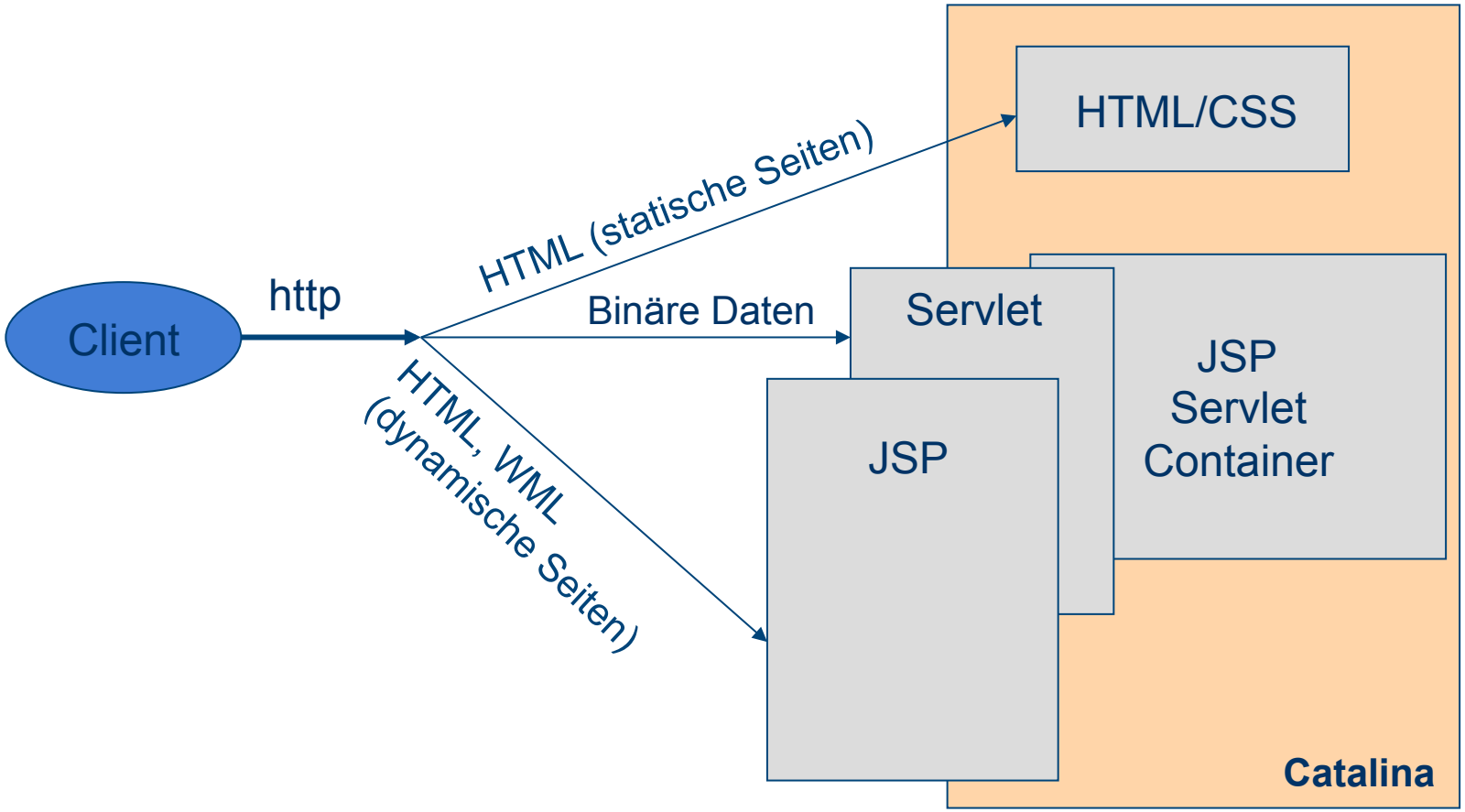
Bezug von Servlets zum Web oder zu den Geschäftsdaten



Serverseitige Implementierungstechnologien (II): Java Servlets

1. Einführung
2. Eigenschaften
3. Beispiel „Hello Servlet“
4. Lebenszyklus von Java - Servlets
5. Anfragebearbeitung mit und ohne Templateverfahren
6. Graphiken mit Java – Servlets
7. Sessiontracking
8. Der Servlet – Container
9. Webbezogene und Geschäftsbezogene Servlets
10. **Einsatzgebiete der Webkomponenten**
11. Vergleich JSP zu Java – Servlet
12. Fazits zu JSP ‘s und Java - Servlets
13. Zusammenfassung
14. Literatur
15. Ausblick

Einsatzgebiete der Web – Komponenten (serverseitig)



Serverseitige Implementierungstechnologien (II): Java Servlets

1. Einführung
2. Eigenschaften
3. Beispiel „Hello Servlet“
4. Lebenszyklus von Java - Servlets
5. Anfragebearbeitung mit und ohne Templateverfahren
6. Graphiken mit Java – Servlets
7. Sessiontracking
8. Der Servlet – Container
9. Webbezogene und Geschäftsbezogene Servlets
10. Einsatzgebiete der Webkomponenten
11. **Vergleich JSP zu Java - Servlet**
12. Fazits zu JSP ‘s und Java - Servlets
13. Zusammenfassung
14. Literatur
15. Ausblick

Vergleich **JSP** / **Servlet**

	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 Implementierungstechnologien (II): Java Servlets

1. Einführung
2. Eigenschaften
3. Beispiel „Hello Servlet“
4. Lebenszyklus von Java - Servlets
5. Anfragebearbeitung mit und ohne Templateverfahren
6. Graphiken mit Java – Servlets
7. Sessiontracking
8. Der Servlet – Container
9. Webbezogene und Geschäftsbezogene Servlets
10. Einsatzgebiete der Webkomponenten
11. Vergleich JSP zu Java - Servlet
12. **Fazits zu JSP ‘s und Java - Servlets**
13. Zusammenfassung
14. Literatur
15. Ausblick

Fazit zu JSP ‘s und Java - Servlets

- **gute Performance und Skalierbarkeit**
  - Einsatz bei großen und sehr großen Projekten
  - Nutzen der java - typischen Spracheigenschaften
- **umfassende standardisierte Bibliothek**
  - u.a. Integration von Legacysystemen möglich
  - Entwicklung eigener Bibliotheken
- **Portabilität**
  - über Betriebssystem-Plattformen über WebContainer / Applikationssserver hinweg
- **hoher Standardisierungsgrad**
  - kaum Abhängigkeiten von einem Anbieter
  - kommerzielle Applikationsserver: Bea Weblogic, IBM WebSpere, Oracle Application Server, SAP Web Application Server
  - Open-Source-Implementierungen: Apache TomCat, JBoss
- **klare Architektur und Aufgabenteilung möglich** durch Kombination von Java Servlets, JSP ‘s, und JavaBeans -> Trennung von Präsentation, Applikationssteuerung und Businesslogik

Literatur für Java Servlets

- Heiko Wöhr „Webtechnologien“ , dpunkt.verlag, Heidelberg 2004
- Coy und Bornemann: „Java & Webapplikationen“, SPC TEIA Lehrbuch Verlag 2002
- Wolfgang Dehnhard „Java und Datenbanken“ Anwendungsprogrammierung mit JDBC, Servlets und JSP
- Sun – Tutorial : http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets.html
- http://www.galileocomputing.de/openbook/javainsel4/

Zusammenfassung

- Kennenlernen der Funktionsweise und des Lebenszyklus von Java – Servlets an Beispielen
- Sessiontracking mit Beispiel
- Rollen von Servlets und JSP ‘s in Webapplikationen
- Unterschiede und Gemeinsamkeiten sowie Fazits aus den JSP ‘s und den Servlets