



---

# Client/Server-Programmierung

WS 2017/2018

Roland Wismüller  
Betriebssysteme / verteilte Systeme  
roland.wismueller@uni-siegen.de  
Tel.: 0271/740-4050, Büro: H-B 8404

Stand: 1. Dezember 2017





---

# Client/Server-Programmierung

WS 2017/2018

## 6 Servlets und JSP

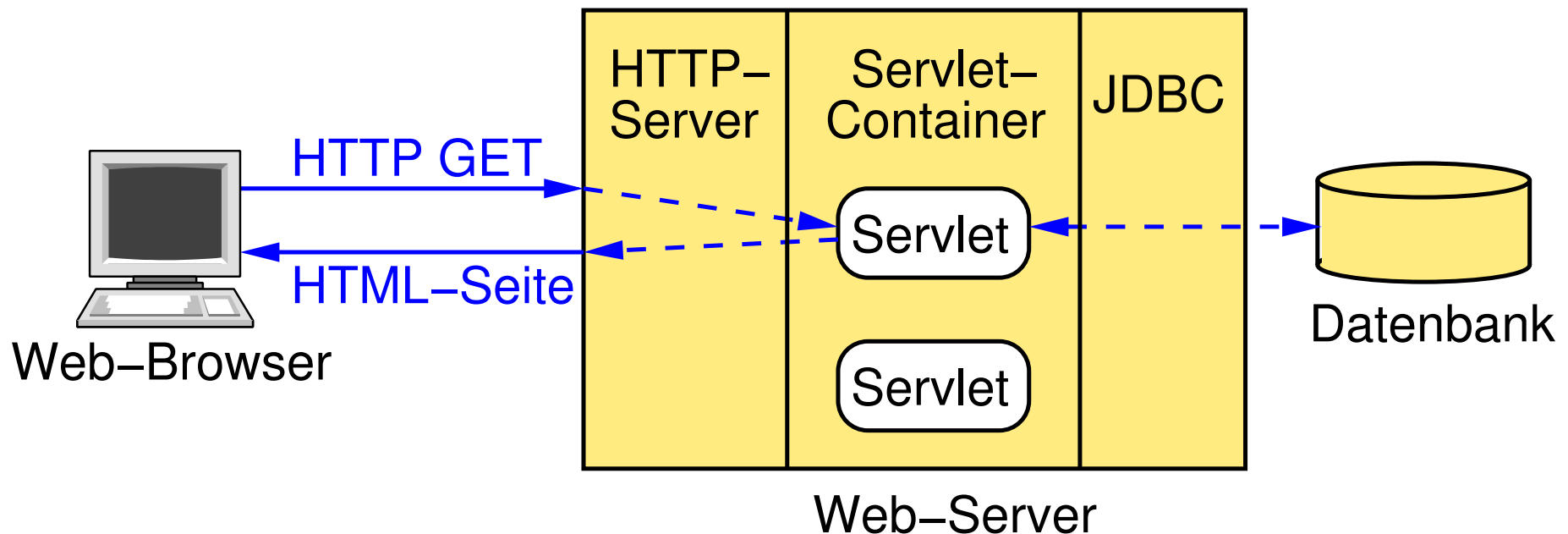


## Inhalt

- ➔ Servlets
- ➔ Java Server Pages (JSP)
  
- ➔ Hunter / Crawford
- ➔ Farley / Crawford / Flanagan, Kap. 5 und 6
- ➔ Langner (Verteilte Anwendungen mit Java), Kap. 6
- ➔ Orfali / Harkey, Kap. 12
- ➔ <http://docs.oracle.com/javaee/5/tutorial/doc/bnadp.html>
- ➔ <http://docs.oracle.com/javaee/6/tutorial/doc/bnafed.html>
- ➔ <http://www.oracle.com/technetwork/java/javaee/tech>

## 6.1 Servlets

- ➔ Java Software-Komponenten zur dynamischen Erweiterung von Web-Servern
  - ➔ Erzeugung dynamischer HTML-Seiten, z.B. aus Datenbank-Inhalten
- ➔ Typische Architektur:





## 6.1.1 Grundlagen

- ➔ Servlets sind Java-Klassen, die innerhalb eines Web-Servers ausgeführt werden
  - ➔ Web-Server muß servlet-fähig sein, d.h. über einen Servlet-Container verfügen (z.B. Tomcat)
  - ➔ Container lädt Servlets bei Bedarf dynamisch nach
- ➔ (HTTP-)Servlets werden (u.a.) über die HTTP-Anfragen GET bzw. POST angesprochen
  - ➔ Servlet bearbeitet die Anfrage und erzeugt eine HTML-Seite
  - ➔ Bearbeitung erfolgt durch eigenen Thread im Adreßraum des Web-Servers
- ➔ (Generische Servlets werden hier nicht behandelt)



### Die HTTP-Methoden GET und POST

- ➔ Teil des HTTP-Protokolls: Browser-Anfragen an den Server
- ➔ Auch verwendet in HTML-Formularen
- ➔ GET-Methode
  - ➔ zum Holen von Dokumenten über eine URL bestimmt
  - ➔ URL kann auch weitere Parameter beinhalten, z.B.
    - ➔ `GET /buy.html?what=shoe&price=50.00 HTTP 1.0`
  - ➔ begrenzte Länge der URL!
- ➔ POST-Methode
  - ➔ zum Senden von Daten an den Web-Server
  - ➔ Parameter werden im Rumpf der HTTP-Anfrage übertragen, sind in der URL nicht sichtbar



### Implementierung von HTTP-Servlets

- ➔ Ableiten einer Klasse von `javax.servlet.http.HttpServlet`
- ➔ I.d.R. Überschreiben einer der Methoden
  - ➔ `void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException`
    - ➔ Behandlung von HTTP-GET-Anfragen
    - ➔ `void doPost(...):` analog für HTTP-POST-Anfragen
- ➔ Bei Bedarf Überschreiben der Methoden
  - ➔ `void init() :` gerufen, wenn Servlet geladen wird
  - ➔ `void destroy() :` gerufen, wenn Servlet entfernt wird
- ➔ Einige weitere Methoden, siehe API-Dokumentation



## 6.1.2 Einschub: Web-Server im Labor

- ➔ Im Labor H-A4111 kann/soll jeder Student einen eigenen Web-Server verwenden
  - ➔ tomcat-Server (für Servlets, Web-Services, ...)
- ➔ Private Installation (i.w. Konfigurationsdateien):
  - ➔ Aufruf des Skripts

```
    /opt/dist/tools/tomcat_install.sh
```

auf einem Rechner im Labor H-A 4111
    - ➔ konfiguriert für jeden Benutzer eigene Ports
  - ➔ Umgebungsvariablen setzen (in \$HOME/.profile):

```
export CATALINA_BASE=$HOME/Soft/apache-tomcat-6.0.18
export CATALINA_HOME=/opt/dist/apache-tomcat-6.0.18
export PATH=$CATALINA_HOME/bin:$PATH
```





### Nutzung von Tomcat

- ➔ Start des tomcat-Servers mit `catalina.sh run`
  - ➔ benutzte Port-Nummer wird beim Start ausgegeben:  
INFO: Initializing Coyote HTTP/1.1 on http-8080
- ➔ Web-Seiten können unter `$CATALINA_BASE/webapps/ROOT` angelegt werden
- ➔ Erreichbar dann unter der URL  
`http://rechnername:port/dateiname`
  - ➔ Im Labor H-A 4111 muß wegen des voreingestellten Web-Proxies immer der vollständige Rechnername (FQDN) angegeben werden
  - ➔ z.B. `bs1ab01.lab.bvs`



### 6.1.3 Beispiel: Hello-World Servlet

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

public class HelloSrv extends HttpServlet {

    private int counter = 0;

    // Wird bei HTTP—Get—Anfrage aufgerufen
    public void doGet(HttpServletRequest request,
                       HttpServletResponse response)
        throws IOException, ServletException {

        counter++;
    }
}
```

## 6.1.3 Beispiel: Hello-World Servlet ...



```
// Extrahiert Parameter 'name' aus URL
```

```
String name = request.getParameter("name");
```

```
response.setContentType("text/html");
```

```
// Ausgabestrom für die erzeugte HTML–Seite
```

```
PrintWriter out = response.getWriter();
```

```
out.println("<html>");
```

```
out.println("<head><title>Hallo World</title></head>");
```

```
out.println("<body><b>" + counter + ". Hello to " + name +  
            "!</b></body>");
```

```
out.println("</html>");
```

```
out.close();
```

```
}
```

```
}
```



### HTML-Seite zum Aufruf des Servlets

```
<HTML>
  <HEAD><TITLE>Hello-World</TITLE></HEAD>
  <BODY>
    <P>
      <A HREF="http://localhost:8080/test/hello?name=Roland">
        Say Hello to Roland</A>
    </P>
    <P> Say Hello to:
      <FORM METHOD="GET"
        ACTION="http://localhost:8080/test/hello">
        <INPUT TYPE="text" NAME="name" SIZE="10">
        <INPUT TYPE="submit" VALUE="Submit">
      </FORM>
    </P>
  </BODY>
</HTML>
```



### *Deployment mit Tomcat-Server*

- ➔ Übersetzen des Servlets
  - ➔ `javac -cp $CATALINA_HOME/lib/servlet-api.jar:. HelloSrv.java`
  - ➔ CLASSPATH nur notwendig, wenn J2EE nicht installiert ist
- ➔ Erstellen eines *Deployment*-Deskriptors unter `WEB-INF/web.xml`
- ➔ Kopieren der `class`-Datei(en) nach `WEB-INF/classes`
  - ➔ `cp HelloSrv.class WEB-INF/classes`
- ➔ Erzeugen eines WAR Archivs
  - ➔ `jar -cvf test.war WEB-INF`
- ➔ Kopieren des WAR Archivs in das Tomcat-Verzeichnis
  - ➔ `cp test.war $CATALINA_BASE/webapps`



### *Deployment-Deskriptor für das Servlet*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC
  "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
  <servlet>
    <servlet-name>HelloWorld</servlet-name>
    <servlet-class>HelloSrv</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>HelloWorld</servlet-name>
    <url-pattern>/hello</url-pattern>
  </servlet-mapping>
</web-app>
```

---

# Client/Server-Programmierung

WS 2017/2018

01.12.2017

Roland Wismüller  
Betriebssysteme / verteilte Systeme  
roland.wismueller@uni-siegen.de  
Tel.: 0271/740-4050, Büro: H-B 8404

Stand: 1. Dezember 2017



### Deployment mit Tomcat-Server ...

- ➔ Servlet ist nun unter dieser URL ansprechbar:
  - ➔ `http://localhost:8080/test/hello`
- ➔ HTML-Datei zum Aufruf des Servlets kann z.B. nach `$CATALINA_BASE/webapps/ROOT/hello.html` kopiert werden
  - ➔ URL dann `http://localhost:8080/hello.html`
- ➔ **Anmerkungen:**
  - ➔ Servlet-Klassen dürfen nicht im CLASSPATH von Tomcat sein!
    - ➔ Tomcat nie im Verzeichnis starten, in dem die Servlet-Klassen liegen!
  - ➔ Tomcat 6.0 kann beim Deployment auch laufen
    - ➔ WAR Archiv wird bei Änderung erneut ausgepackt, Klassen werden neu geladen





### 6.1.4 Lebenszyklus eines Servlets

- ➔ Beim Start des Servers oder durch Client-Anfrage:
  - ➔ Servlet-Klasse wird in Web-Server geladen
  - ➔ **eine** Instanz der Servlet-Klasse wird erzeugt
  - ➔ die `init()`-Methode wird aufgerufen
- ➔ Bei einer HTTP-Anfrage:
  - ➔ Erzeugung eines neuen Threads, der die Methode `doGet()` bzw. `doPost()` ausführt
    - ➔ Implementierung der Methoden muß thread-sicher sein!
- ➔ Bei Entfernung des Servlets aus dem Server
  - ➔ Aufruf der Methode `destroy()`

### 6.1.5 Wichtige Klassen und Methoden

- ➔ `HttpServletRequest`: HTTP-Anfrage
  - ➔ `String getParameter(String name)`
    - ➔ liefert Wert des genannten Anfrage-Parameters
    - ➔ z.B. bei GET `/buy.html?what=shoe` HTTP 1.0
  - ➔ `HttpSession getSession()`
    - ➔ liefert bzw. erzeugt Sitzungs-Objekt (☞ **6.1.6**)
- ➔ `HttpServletResponse`: HTTP-Antwort
  - ➔ `void setContentType(String type)`
    - ➔ setzt MIME-Typ der Antwort (i.d.R. "text/html")
  - ➔ `PrintWriter getWriter()`
    - ➔ liefert `PrintWriter` zum Schreiben der Ausgabe



### 6.1.6 Sitzungs-Management

- ➔ Methode `getSession()` erlaubt Management von Client-Sitzungen
  - ➔ erzeugt neue Sitzung, falls noch keine existiert
  - ➔ liefert Sitzungs-Objekt `HttpSession` als Ergebnis
- ➔ Verfolgung von Sitzungen:
  - ➔ Server erzeugt eindeutige Sitzungs-ID
  - ➔ Sitzungs-ID wird als *Cookie* im Client gespeichert
  - ➔ *Cookie* wird bei jeder erneuten Anfrage an Server übertragen
  - ➔ (Alternativ kann Sitzungs-ID auch an URLs angefügt werden)



### Das HttpSession-Objekt

- ➔ Identifiziert eindeutig eine Client-Sitzung
- ➔ Erlaubt, beliebige Information sitzungs-lokal zu speichern
- ➔ Wichtige Methoden:
  - ➔ `String getId()`: liefert Sitzungs-ID
  - ➔ `boolean isNew()`: neue Sitzung?
  - ➔ `void setAttribute(String name, Object value)`
    - ➔ Speichern sitzungslokaler Daten unter gegebenem Namen
  - ➔ `Object getAttribute(String name)`
    - ➔ Auslesen sitzungslokaler Daten mit gegebenem Namen
  - ➔ `setMaxInactiveInterval(int interval)`
    - ➔ Einstellen des Sitzungs-*Timeouts*



### Beispiel: Hello-World mit sitzungslokalem Zähler

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

    Integer counter;
    HttpSession session = request.getSession();

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<head><title>Hallo World</title></head>");
    out.println("<body>");

    if (session.isNew()) {
        out.println("<b>Welcome to new session</b><br>");
    }
}
```



### Beispiel: Hello-World mit sitzungslokalem Zähler ...

```
counter = (Integer)session.getAttribute("HelloSession.cntr");
if (counter == null) {
    counter = 1;
}
else {
    counter++;
}
session.setAttribute("HelloSession.cntr", counter);
```

```
String name = request.getParameter("name");
out.println("<b>" + counter + ". Hello to " + name + "!</b>");
out.println("<br>Session ID: " + session.getId() + "</body>");
out.println("</html>");
out.close();
}
```



## 6.2 *Java Server Pages (JSP)*

- ➔ Ziel: einfachere Generierung dynamischer HTML-Seiten
- ➔ Probleme von Servlets:
  - ➔ Ausgabe von HTML-Code in `println()`-Anweisungen ist umständlich / fehleranfällig
    - ➔ der größte Anteil davon ist statisch
  - ➔ Installation und Deployment von Servlets ist schwierig
    - ➔ verglichen mit statischen HTML-Seiten
- ➔ Lösungsidee:
  - ➔ Einbetten von Java-Servlet-Code in statische HTML-Seiten
  - ➔ beim ersten Aufruf der Seite wird über JSP-Compiler automatisch ein Servlet erzeugt und in WWW-Server geladen
  - ➔ spätere Aufrufe nutzen dann direkt das Servlet



### 6.2.1 Spezielle Tags für JSP

- ➔ Tag für Ausdrücke: `<%= Ausdruck %>`
  - ➔ Wert des Java-Ausdrucks erscheint in HTML-Ausgabe
  - ➔ Beispiel:
    - ➔ `<html><body> 17 + 4 = <%= 17+4 %> </body></html>`
    - ➔ Ergebnis: `<html><body> 17 + 4 = 21 </body></html>`
- ➔ Tag für Java-Code: `<% Java-Code %>`
  - ➔ angegebener Java-Code wird in `doGet()` bzw. `doPost()` Methode eines Servlets ausgeführt
  - ➔ Ausgabe wird in HTML-Ausgabe eingefügt
  - ➔ Java-Code kann auch mit regulärem HTML-Code gemischt werden
    - ➔ z.B. für bedingte HTML-Ausgaben





➔ Tag für Java-Code: `<% Java-Code %>` ...

➔ Beispiel:

```
<html>
  <body>
    <%
      java.util.Date date = new java.util.Date();
      if (date.getHours() < 12) {
        %>
          Guten Morgen!
        <% } else { %>
          Guten Tag!
        <% } %>
      Es ist jetzt <%= date.toString() %>.
    </body>
  </html>
```



- ➔ Tag zur Deklaration globaler Variablen: `<%! Deklaration %>`
- ➔ globale Variable  $\hat{=}$  Attribut der erzeugten Servlet-Klasse
- ➔ Wert bleibt über alle Aufrufe der JSP-Seite hinweg erhalten
  - ➔ auch über verschiedene Client-Sitzungen hinweg
- ➔ Beispiel: Hit-Counter

```
<html>
  <body>
    <%! int hitCount = 0; %>
    Hit Count: <%= ++hitCount %>
  </body>
</html>
```



### 6.2.2 Vordefinierte Variablen

- ➔ Java-Code in JSP-Seiten kann u.a. folgende vordefinierte Variablen nutzen:
  - ➔ `request`
    - ➔ `HttpServletRequest`-Parameter der Servlet-Methoden `doGet()` bzw. `doPost()`
  - ➔ `response`
    - ➔ `HttpServletResponse`-Parameter
  - ➔ `out` (  $\approx$  `response.getWriter()` )
    - ➔ `JSPWriter` ( $\approx$  `PrintWriter`) für HTML-Ausgabe
  - ➔ `session` ( = `request.getSession()` )
    - ➔ `HttpSession`-Objekt für aktuelle Client-Sitzung



### 6.2.3 Beispiele

➔ Hello-World (funktional identisch mit Beispiel aus **6.1.3**)

```
<html>
  <body>
    <%! int counter = 0; %>
    <%
      counter++;
      String name = request.getParameter("name");
    %>
    <b><%= counter%>. Hello to <%= name%>!</b>
  </body>
</html>
```



- ➔ Hello-World mit sitzungslokalem Zähler  
(funktional identisch mit Servlet-Beispiel aus **6.1.6**)

```
<html>
  <body>
    <%
      Integer counter;

      if (session.isNew()) {
        out.println("<b>Welcome to new session</b><br>");
      }
      counter = (Integer)
        session.getAttribute("HelloSession.cntr");

      if (counter == null) {
        counter = 1;
      }
    %>
  </body>
</html>
```



```
else {
    counter++;
}
session.setAttribute("HelloSession.cntr", counter);
```

```
String name = request.getParameter("name");
out.println("<b>" + counter + ". Hello to "
            + name + "</b>");
out.println("<br>Session ID: " + session.getId());
```

```
%>
```

```
</body>
```

```
</html>
```



### 6.2.4 JSP Direktiven

- ➔ JSP bietet einige Direktiven, die das Verhalten der JSP-Seite kontrollieren
- ➔ Allgemeine Syntax: `<%@ Direktive %>`
- ➔ Wichtige Beispiele:
  - ➔ `<%@ include file="copyright.html" %>`
    - ➔ Einfügen der Datei `copyright.html` in die JSP-Seite
  - ➔ `<%@ page import="javax.rmi.*, javax.naming.*" %>`
    - ➔ Importieren von Java-Paketen (komma-separierte Liste)



### 6.2.5 JSP und Java Beans

➔ JSP-Seiten können auch auf Java Beans zurückgreifen:

➔ `<jsp:useBean id="product"  
                  class="com.company.ProductBean"/>`

➔ Erzeugt **eine** Bean-Instanz für **alle** Aufrufe der JSP-Seite

➔ Referenz in der globalen Variable `product` gespeichert

➔ Abfrage von *Properties* der Bean:

➔ `<jsp:getProperty name="product" property="Price"/>`

➔ entspricht `<%= product.getPrice() %>`

➔ Setzen von *Properties* der Bean:

➔ `<jsp:setProperty name="product" property="Price"  
                  value="12.50"/>`

➔ Vorteil: Modularität, weniger / kein Java-Code in der JSP-Seite





## 6.3 Zusammenfassung

- ➔ Servlets: Java-Klassen zur Bearbeitung von HTML-Anfragen in WWW-Servern
  - ➔ Erzeugung dynamischer HTML-Seiten
  - ➔ Klassen: `HttpServlet`, `HttpServletRequest`, `HttpServletResponse`
- ➔ Im Server: genau eine Instanz der Servlet-Klasse
  - ➔ Sitzungs-Management kann/muß explizit programmiert werden (`HttpSession`)
- ➔ JSP: Vereinfachter Umgang mit Servlets
  - ➔ Mischung von statischem HTML-Code und Servlet-Code
  - ➔ Servlet wird dynamisch aus JSP-Seite generiert