Java Servlets

Praktikum aus Softwareentwicklung 2

Abteilung für Bioinformatik JKU Linz

Java Servlets

- Grundlagen
  - HTTP-Protokoll, Request/Response, ...
- Architektur
  - Merkmale, Servlet-Methoden, Lebenszyklus, ...
- Sessions
  - Session Tracking API, Cookies, ...
- Deployment
  - Deployment Descriptor, ...
- Zusammenfassung

Servlets
Grundlagen

- **Web-Browser** realisiert **Benutzerschnittstelle**
  - Erweiterbar durch externe Anwendungen, Plugins, Applets
  - Problem dabei: lokale Installation notwendig; Plattform- und Browser-spezifisch
- **Web-Server** stellt Informationen bereit
  - Dynamische Generierung der Informationen durch CGI-Programme, Server Side Includes, Servlets, JSP, PHP etc.
- **HTML** als Sprache zur Informationsbeschreibung
  - Hypertext, Multimedia
  - Formulare als Mechanismus zur Dateneingabe durch den Benutzer
- **HTTP** als Kommunikationsprotokoll zwischen Server und Browser
  - Zustandslosigkeit des HTTP-Protokolls erfordert Workarounds

Servlets
HTTP-Protokoll

- Definiert in RFC-2068 (`www.ietf.org/rfc.html`)
- Browser *(User-Agent)* sendet **Request**
  - 1. Zeile mit Requestmethode, z.B. `GET` oder `POST`, und *URL-Pfad*
    - `GET`: Download von einer URL; ohne Body; Request-Parameter als Teil des URL sind möglich, z.B. /login?user=xxx
    - `POST`: wie GET, aber Parameter sind im Request-Body kodiert
    - Weiters `PUT`, `DELETE`, `HEAD`, `OPTIONS`, `TRACE`
  - Es folgen *Header*-Zeilen
  - optional einen *Request-Body* (enthält z.B. Formulardaten)
- Server antwortet mit **Response**
  - Statuszeile
  - *Header*-Zeilen, z.B.: `Content-Type`, Cookies, etc.
  - optional ein *Response-Body*, z.B. HTML-Code, GIF, etc.

Servlets

Beispiel: HTTP-Request

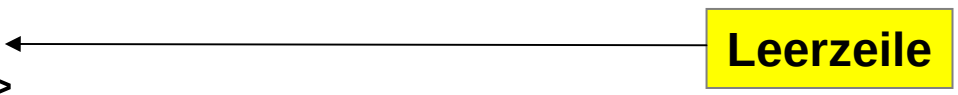
- Erzeugt von MS IE 5 aus der URL **http://www.ifs.uni-linz.ac.at/ifs/index.html**

```
GET /ifs/index.html HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg, application/vnd.ms-excel,
application/msword, application/vnd.ms-powerpoint, */*
Accept-Language: de-at
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01;
Windows NT 5.0)
Host: www.ifs.uni-linz.ac.at
Connection: Keep-Alive
```

Servlets

Beispiel: HTTP-Response

- **HTTP/1.1 200 OK**
  Date: Wed, 21 Mar 2001 14:59:04 GMT
  Server: Apache/1.3.9 (Unix) ApacheJServ/1.1b3
  Keep-Alive: timeout=15, max=100
  Connection: Keep-Alive
  Content-Type: text/html




```
<HTML>
  <HEAD>
    <TITLE>IFS Homepage</TITLE>
  </HEAD>
  <BODY BGCOLOR="#006595"
  ...
```

tiscover
THE TRAVEL NETWORK

Java Servlets

- Grundlagen
  - HTTP-Protokoll, Request/Response, ...
- Architektur
  - Merkmale, Servlet-Methoden, Lebenszyklus, ...
- Sessions
  - Session Tracking API, Cookies, ...
- Deployment
  - Deployment Descriptor, ...
- Zusammenfassung

Servlets

Entstehung

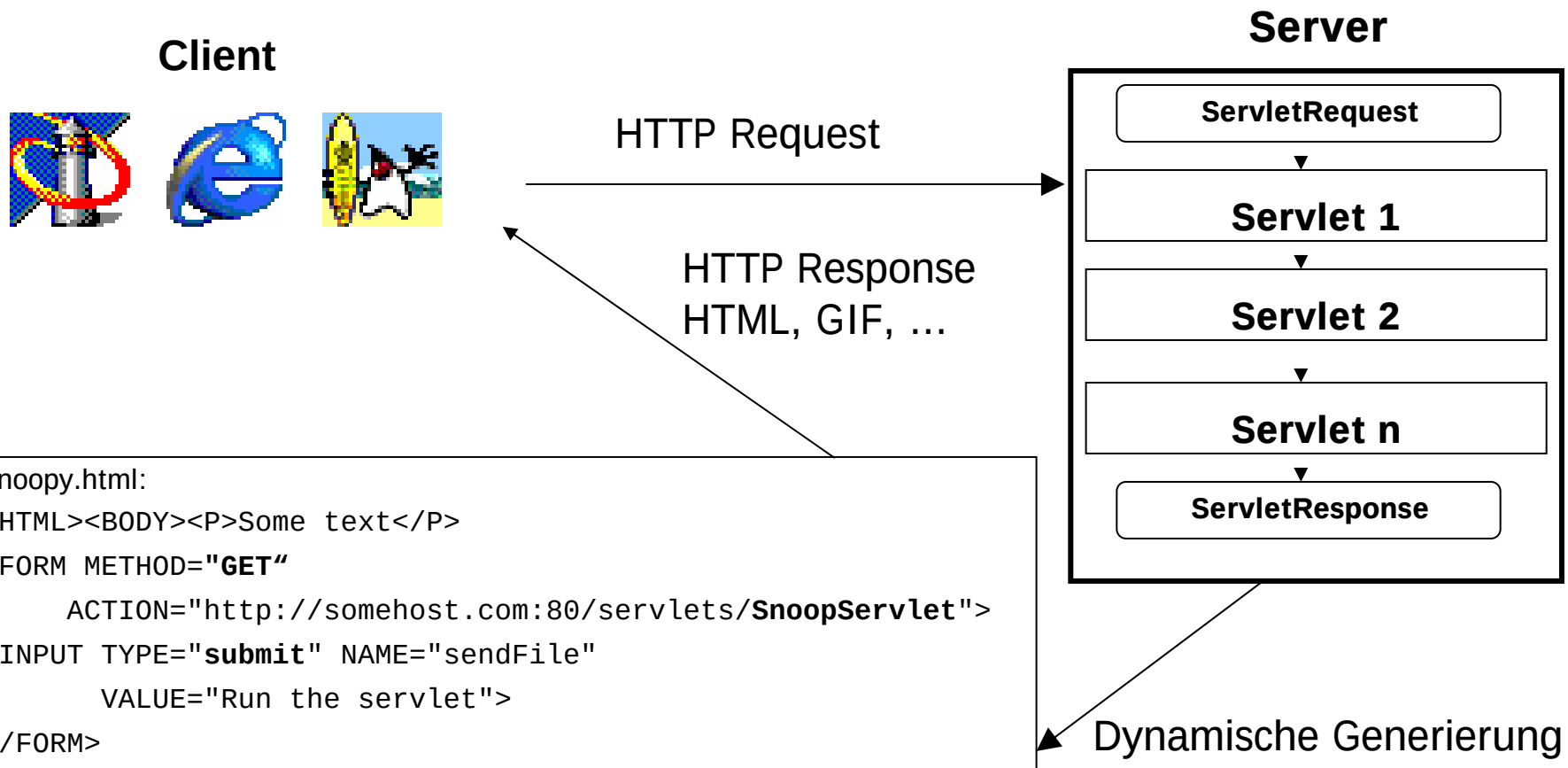
- **CGI** als Mechanismus der Erweiterung der Funktionalität des Web
  - Performanznachteil (-> **FastCGI**)
  - Keine Interaktion mit Webserver, weil eigener Prozess (z.B. kann ein CGI-Prozess nicht in die Logdatei des Servers schreiben)
- 1996 stellt Netscape sogenannte "**Server-side Applets**" vor
- 1996 präsentiert W3C "**resources**" als serverseitige Java Module
- 1997 standardisiert JavaSoft die **Servlet** Technologie
- Ab 1999: Webserver Erweiterungen: PHP...
- Weiters gibt es **proprietäre Erweiterungen** (Server Side Includes, Active Server Pages) von Netscape, Microsoft, etc.
- **Java Server Pages (JSP)** stellen einen Server Side Include (SSI) Mechanismus dar

Servlets

Merkmale

- Servlets sind **Server-seitige** Klassen (Framework), die Applikationen nach dem Request – Response Modell ermöglichen. Primäres Ziel: HTTP-Anwendungen
- **Plattformunabhängigkeit** durch Java Technologie und standardisiertes API
- Wiederverwendung der **Funktionalität** von Java
- **Sicherheitskonzept** von Java
- **Erweiterbarkeit**, z.B. Java Server Pages (`http://java.sun.com/products/jsp/`)
- **URLs**:
  - `http://java.sun.com/products/servlet/`
  - `http://www.novocode.com/doc/servlet-essentials/`

Servlets
Architektur 1

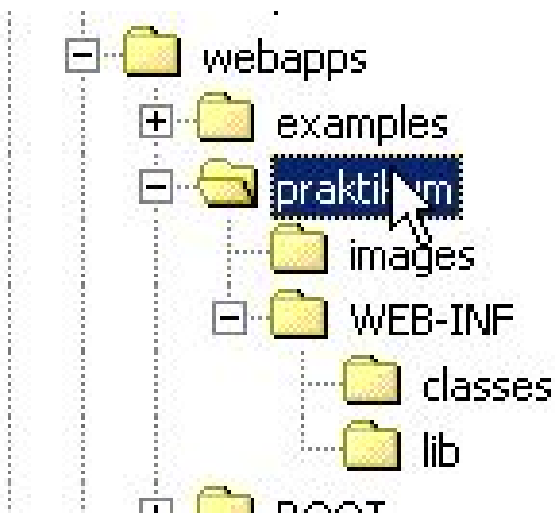


Snoopy.html:

```
<HTML><BODY><P>Some text</P>
<FORM METHOD="GET“
     ACTION="http://somehost.com:80/servlets/SnoopServlet">
<INPUT TYPE="submit" NAME="sendFile"
       VALUE="Run the servlet">
</FORM>
...
```

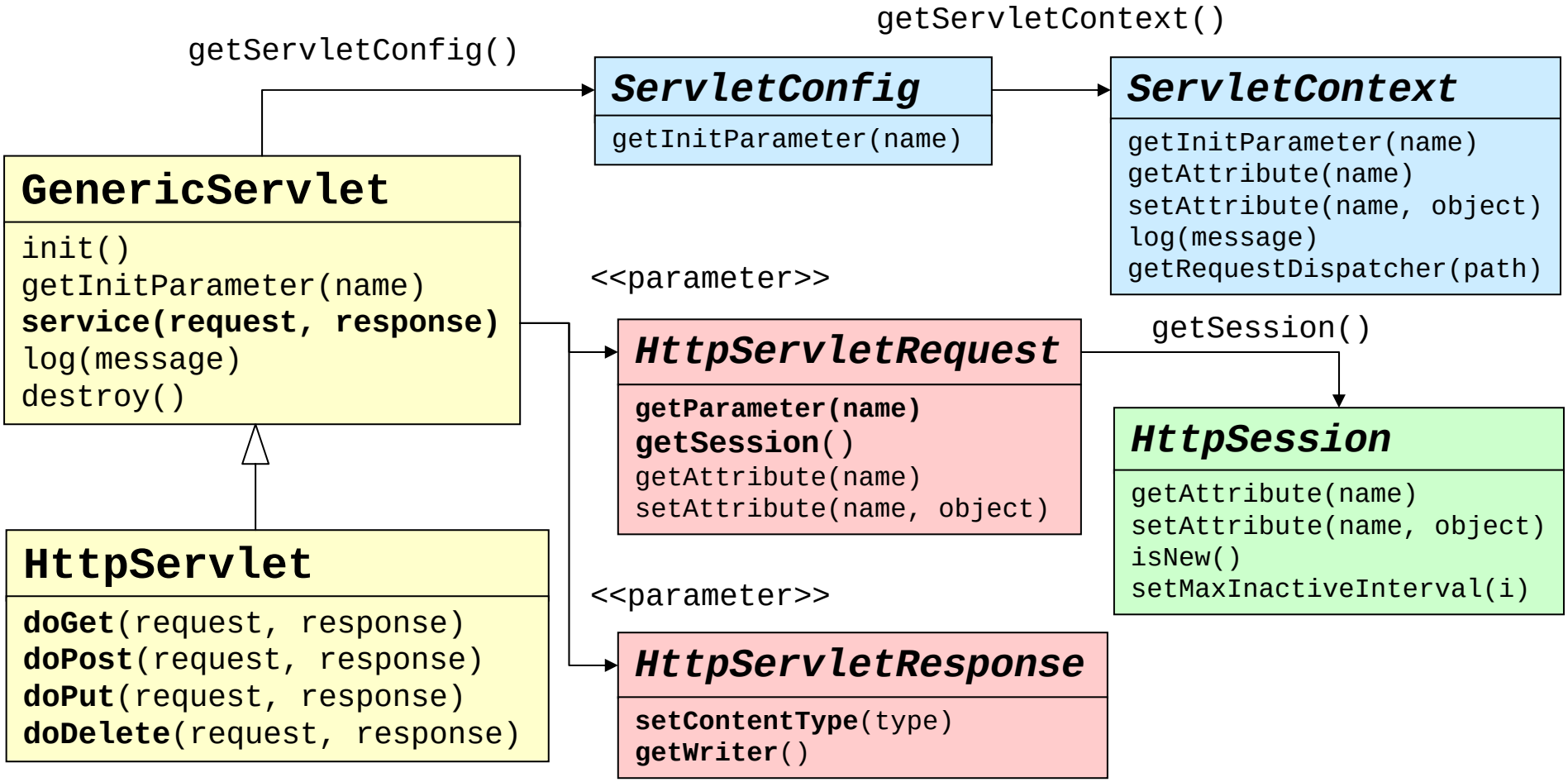
Servlets
Webapplication - <name>.war

- Ermöglicht das schnelle Einsetzen einer (Web)Applikation auf verschiedensten Servlet-Containern.
- Definiert die Verzeichnisstruktur und das Verhalten einer Webapplikation
- Konfigurationsfile: web.xml
- Verzeichnisstruktur
  - RootDir, DocumentRoot
    → praktikum (*.html, images)
  - WEB-INF → web.xml
  - classes → SnoopServlet.class
  - lib → Libraries



Servlets

Architektur 2



Servlets

HTTP Servlet Methoden

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

public class SnoopServlet extends HttpServlet {
 public void doGet(HttpServletRequest req, HttpServletResponse res)
                      throws ServletException, IOException {
   doGetPostRequest(„GET“,req,res);
 }
 public void doPost(HttpServletRequest req, HttpServletResponse res)
                       throws ServletException, IOException  {
    doGetPostRequest(„POST“,req,res);
 }
 public void doGetPostRequest(String method, HttpServletRequest req,
                                  HttpServletResponse res)
                       throws ServletException, IOException  {
   ...
 }
}
```

Servlets
Lebenszyklus

- Methoden, die zur **Steuerung** überschrieben werden können, werden ausschließlich vom Server aufgerufen:

```
void init(ServletConfig config);
void doDelete(HttpServletRequest, HttpServletResponse);
void doGet(HttpServletRequest, HttpServletResponse);
void doPost(HttpServletRequest, HttpServletResponse);
void doPut(HttpServletRequest, HttpServletResponse)
public void destroy();
```

- **Lebenszyklus**:
  - Aufruf von `init()` nach dem **Laden** des Servlets
  - doXXX() bei **Anfragen** von Clients
  - Bei **Beenden** wird `destroy()` aufgerufen

Servlets
Initialisierung

- Methode `init()`
  - Wird vor dem ersten doXXX()-Aufruf ausgeführt
  - throws `UnavailableException`, falls das Servlet nicht benutzbar ist
- Initialisierungsparameter im Deployment Descriptor
  - Für Servlet, z.B: `getInitParameter("forwardURL");`

```
<servlet>
 <init-param>
     <param-name>forwardURL</param-name>
     <param-value>/main</param-value>
 </init-param> …
```

  - Für Applikation: `getServletContext().getInitParameter("name");`

```
<web-app>
   <context-param>
     <param-name>name</param-name>
     <param-value>a_val</param-value>
 </context-param> …
```

Servlets

HttpServletRequest

- Zugriff auf **Header**- bzw. **Body**-Daten des HTTP Request
- HTTP-Parameter (Transparent GET oder POST)
  - `String param = request.getParameter(name);`
  - `String[] params = request.getParameterValues(name);`
  - `Enumeration p = request.getParameterNames();`
  - `String q = request.getQueryString();` (für HTTP GET)
- Session ID: `getRequestedSessionId()` - siehe Session Tracking
- Weitere Header-Daten
  - z.B.: `getRemoteHost()`, `getRemoteUser()`, `getLocale()`,...
- Body-Daten (bei POST, PUT, und DELETE)
  - Entweder **implizit** mit `getParameter()`, oder **explizit**
  - Header-Daten `getContentType()` und `getContentLength()`
  - `getReader()` für Textdaten bzw. `getInputStream()` für Binärdaten

Servlets
HttpServletResponse

- **Header**
  - `response.setContentType("text/html; charset=ISO-8859-1");`
  - Weitere Methoden: `setContentLength(len)`, `addCookie(cookie)`, ...
  - Auch `request.getSession()` kann implizit Header-Daten (Cookie) erzeugen
- **Body** erst *nach* den Header-Daten schreiben
  - `PrintStream out=response.getWriter();` für Textdaten
    - Zeichencodierung entsprechend dem bei `setContentType` angegebenen charset
  - `response.getOutputStream()` für Binärdaten (z.B. Bilder)

Servlets
Generieren von HTML

- Das **Generieren von HTML** kann durch einfache `println` Anweisungen erfolgen. Z.B.

```
out=response.getWriter();
out.println("<HTML><HEAD><TITLE>Hello World</TITLE>");
...
```

  - Online HTML-Tutorial: `http://selfhtml.teamone.de/html/`

- Einen Weg zur **Trennung von HTML und Java**-Code stellt die Verwendung von Templates bzw. Server Side Include-Mechanismen dar, z.B. Java Server Pages

```
<HTML><HEAD><TITLE>Hello World</TITLE></HEAD>
<BODY><TABLE>
<% Iterator i=all.iterator();
   while(i.hasNext()) { %>
<TR><TD> ...
```

Servlets

Beispiel „*doGet()*“

```
public void doGetPostRequest (HttpServletRequest req, HttpServletResponse res)
  throws ServletException, IOException {
  PrintWriter out;
  res.setContentType("text/html");
  out = res.getWriter();
  out.println("<html><head><title>Snoop Servlet</title></head>");
  out.println("<body bgcolor=\"#FFFFFF\">");
  out.println("<h1>Requested URL:</h1>");
  out.println (getRequestURL ().toString ());
  log(req.getRequestURL().toString ());
  Enumeration e = req.getHeaderNames();
  while (e.hasMoreElements()) {
    String name = (String)e.nextElement();
    out.println(" " + name + ": " + req.getHeader(name));
  }
  …
  e = req.getParameterNames();
  …
  out.close();
}
```

Servlets

Multithreading / Synchronisierung

- Zugriffe auf den Server von mehreren Clients gleichzeitig möglich; pro Client wird ein eigener Thread gestartet
  - Synchronisierung beim Zugriff auf **gemeinsam verwendete Objekte**, z.B. static-Variablen, ServletContext- und HttpSession-Attribute
- Synchronisierung auf **Servlet-Instanzvariablen**
  - I.A. existiert nur eine Instanz pro Servlet-Klasse, d.h. der Zugriff auf Servlet-Instanzvariablen muss auch synchronisiert werden, z.B.

    ```
    public synchronized void doGet(HttpServletRequest
    req, ...
    ```
  - Besser: Keine Instanzvariablen, sondern mit Objekte in Session oder Request Speichern.

    ```
    req.setAttribute(„com.uni.MyParamBean“,paramBean);
    paramBean = req.getAttribute(„com.uni.MyParamBean“);
    ```
- `SingleThreadModel`

Servlets

Debugging

- Rückgabe von **Fehlermeldungen an den Client** durch die Methode sendError in HttpServletResponse, z.B.

```
public void doGet (HttpServletRequest req,
                   HttpServletResponse res) throws …
  …
  res.sendError(HttpServletResponse.SC_FORBIDDEN);
```

- Mitprotokollieren in der **Logdatei** mit log( ), z.B.

```
… catch(Exception e) { log("oops", e); }
```

  – Eintrag im Logfile der Web-Applikation

```
…
2001-11-03 11:30:53 myservlet: oops
java.lang.NumberFormatException: 4711 not supported
    at MyServlet.doGet(MyServlet.java:50)
    …
```

Servlets
Interaktion zwischen Servlets (via Client)

- Kontrollfluß: URLs
  - Der HTML-Output eines Servlet kann Links auf andere Servlets enthalten, `out.println("<A HREF=/MoreServlet>Weiter</A>");`
  - Die Entscheidung, ob bzw. welches Servlet aufgerufen wird, liegt beim Client
- Datenfluß: Request-Parameter
  - HTTP-Output enthält Daten, die als Request-Parameter an das Ziel-Servlet übermittelt werden
  - Als Teil der URL, z.B.: `out.println("<A HREF=\" /UserDetailServlet?userId=" +user.getId()+ "\">" + user.getName() + "</A>");`
  - Felder in einem Formular, z.B.:
```
out.println("<FORM ACTION=\"/DeleteUserServlet\" METHOD=GET>"+
  "<INPUT TYPE=HIDDEN NAME=userId VALUE=" +user.getId()+ ">"+
  "<INPUT TYPE=SUBMIT VALUE=\"Delete!\"></FORM>");
```
  - Nur String-Daten sind möglich; Alternative: Session-Attribute

Servlets

Interaktion zwischen Servlets (direkt)

- **Kontrollfluß: Delegation und Einbindung**
  Weitergabe einer Client-Anfrage an andere URLs (Servlets, HTML-Seiten, etc.) mittels **`RequestDispatcher`**, z.B.:
  ```
  RequestDispatcher d =
  req.getRequestDispatcher("index.html");
  ```
  - *Delegation* eines Request an eine URL mit `d.`**`forward`**`(request, response);` es dürfen noch keine Response-Daten gesendet worden sein.
  - *Einbindung* einer URL mit `d.`**`include`**`(request, response);` die Ausgabe der URL wird in den Response eingefügt.
- **Datenfluß: Attribute**
  - Attribute eines **ServletRequest**: `request.setAttribute(name, object)` und `object=request.getAttribute(name)`
  - Attribute eines **ServletContext**:
    `Servlet.getServletContext().getAttribute(name)` bzw.
    `Servlet.getServletContext().setAttribute(name, object);`

Servlets

Beispiel: forward() und include()

```
public class PresentationServlet extends HttpServlet {
   protected void doGet(HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException {
     String item = req.getParameter("item");
     if(item == null) {
       req.setAttibute("exception", new Exception("Item not found"));
       getServletContext()
         .getRequestDispatcher("/servlet/ErrorServlet")
         .forward(req, res);
     } else {
       res.setContentType("text/html");
       PrintWriter out = res.getWriter();
       out.print("<HTML><HEAD><TITLE>Item " + item + "</TITLE>"+
           "</HEAD><BODY>Item " + item + ":<P>");
       getServletContext()
         .getRequestDispatcher("/servlet/ItemServlet?item="+item)
         .include(req, res);
       out.print("</BODY></HTML>"); } } }
```

tiscover
THE TRAVEL NETWORK

Java Servlets

- Grundlagen
  - HTTP-Protokoll, Request/Response, ...
- Architektur
  - Merkmale, Servlet-Methoden, Lebenszyklus, ...
- Sessions
  - Session Tracking API, Cookies, ...
- Deployment
  - Deployment Descriptor, ...
- Zusammenfassung

Servlets

Session Tracking

- Problem: **Zustandslosigkeit** von HTTP
  - es gibt keine "Sessions", d.h., jeder *Request* wird individuell und unabhängig von den anderen betrachtet
  - Es ist z.B. schwer in kontextabhängigen Dokumenten (etwa *Forms*) zu navigieren oder Ergebnisse einer Suche in Teilen zu präsentieren
- Verschiedene Varianten, Zustandsinformation in Anfragen "einzubauen":
  - **(1) Autorisierung** des Benutzers
  - **(2) Versteckte** Felder in Formularen
  - **(3)** Zustandsinformation in **URL kodieren**
  - **(4) Cookies**
  - **(5)** Java Servlet **Session Tracking API**

Servlets

(1) Autorisierung des Benutzers

- Das Servlet wird über eine Username/Passwort **geschützte** Seite geladen. Die **Benutzerinformation** kann dann bei jedem Zugriff **abgefragt** werden und benutzerspezifische Daten z.B. in einer Datenbank abgelegt werden.

```
String un = req.getRemoteUser();
if ((un == null) || (un.equals("")))
     //this should not be the case! Raise an error
else out.println("Welcome "+un+"<BR>");
...
```

- **Vorteile**: einfach zu implementieren, browser- und rechner-unabhängig;
- **Nachteile**: jeder Benutzer muss auf dem Webserver eingetragen sein; einloggen für jeden Request erforderlich

Servlets
(2) Versteckte Felder

- **Versteckte Felder** (diese werden vom Browser nicht angezeigt) enthalten Session Information, z.B.

```
<FORM METHOD="GET" ACTION="http://host/servlet">
     <INPUT TYPE="submit" NAME="sendFile" VALUE="Run">
     <INPUT TYPE="hidden" NAME="item" VALUE="i4711">
     <INPUT TYPE="hidden" NAME="item" VALUE="i4712">
</FORM>
```

- **Verwendung** im Servlet:

```
String []items = req.getParameterValues("item");
for (int k=0; k<items.length; k++)
     out.println("Your cart includes "+items[k]+"<BR>");
...
```

- Man könnte auch nur die **Session ID** als Hidden Field speichern.
- **Vorteile:** browserunabhängig, keine Registrierung erforderlich
- **Nachteil**: fehleranfällig, weil versteckte Felder nicht persistent

Servlets
(3) Kodierung in URL

- Session ID wird im **URL kodiert**, z.B. durch
  - Extra Pfad Information (die das Servlet versteht)
    `http://server:80/servlet/MyServlet/123`
  - Zusätzlichen Parameter
    `http://server:81/servlet/MyServlet?sessionid=123`
  - Eigene Notation
    `http://server:82/servlet/MyServlet;jsessionid=123`
- **Extra Pfad Information** funktioniert mit POST und GET; Nachteil, falls wirklich der Pfad verwendet werden soll;
- Bei Verwendung **zusätzlicher Parameter** sind Namenskonflikte möglich;
- **Eigene Notation** wird nicht notwendigerweise von allen Servern unterstützt (unwahrscheinlich);

Servlets
(4) Cookies

- Ursprünglich von Netscape eingeführt (mittlerweile als **RFC 2109** "HTTP State Management Mechanism" - Standards Track);
- Cookies werden am **Client gespeichert** (wenn der Benutzer das erlaubt);
- **Beschränkungen**
  - 20 Cookies pro Site
  - 300 Cookies insgesamt pro User
  - Größe je Cookie maximal 4096 bytes.

```
Cookie co = new Cookie("ID", "Secret Info");
co.setMaxAge(60*60*24*365); //~1 Jahr
res.addCookie(co);  // noch VOR res.getWriter()!
...
Cookie [] cookies = req.getCookies();//in diesem Request!
```

Servlets

(5) Session Tracking API 1/2

- Ab **Java Servlets 2.0**
- Unterstützung **hängt vom verwendeten Server** ab und Client ab; z.B.
  - werden Cookies verwendet, oder
  - URL Kodierung (falls Cookies nicht verfügbar sind)
  - URL-Kodierung mittels **`response.encodeURL(url);`**
- Jeder Benutzer eines Servers hat genau ein Session Objekt. Dieses kann mit `HttpSession session = request.getSession();` abgefragt bzw. beim 1. Aufruf erzeugt werden.
  - Falls mit dem Request eine Session-ID mitgeschickt wurde (mittels Cookie oder URL-Parameter), wird das entsprechende Session-Objekt geliefert
  - Sonst wird ein neues Session-Objekt erzeugt, und die neue Session-ID im Response mitgeschickt
  - Falls der Client Cookies unterstützt, wird automatisch ein Cookie erzeugt
  - Ansonsten kann die Session-ID explizit mit URL-Kodierung übermittelt werden

Servlets

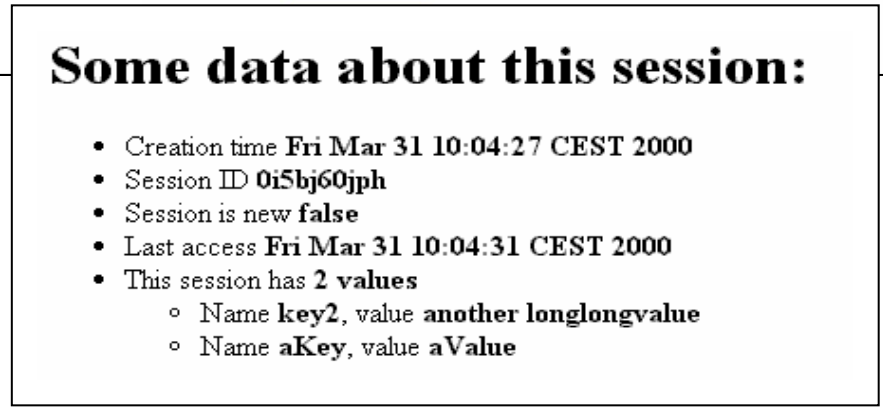
(5) Session Tracking API 2/2

- Im Session-Objekt können **sessionspezifische Daten** gespeichert werden, z.B. Benutzername, temporäre Daten, etc.
  - `HttpSession.`**`setAttribute`**`(String name, Object value);`
  - `Object HttpSession.`**`getAttribute`**`(String name);`
  - `HttpSession.removeAttribute(String name);`
  - `Enumeration HttpSession.getAttributeNames();`
- Eine Session hat eine begrenzte **Gültigkeitsdauer**
  - Explizite Invalidierung, z.B. beim Logout, mit `session.`**`invalidate`**`()`
  - Automatisch nach einer bestimmten Zeit, Default ist 30 min.: `session.setMaxInactiveInterval(int seconds)`
  - Bei einem Request mit einer **abgelaufenen Session-ID** wird ein neues Session-Objekt erzeugt (vgl. `request.getRequestedSessionId()`)
  - Benachrichtigung über das Ende einer Session durch Implementierung des Interface **`HttpSessionBindingListener`**

Servlets

ST Beispiel

- Verwenden von Sessions




```
…
HttpSession session = req.getSession(true);
…
out.println("<LI>Creation time <b>"+new
Date(session.getCreationTime())+"</B>");
out.println("<LI>Session ID <b>"+session.getId()+"</B>");
out.println("<LI>Session is new <b>"+session.isNew()+"</B>");
…
session.setAttribute("aKey", "aValue");
…
Enumeration attrs = session.getAttributeNames();
…
  an=attrs.nextElement().toString();
  out.println("<LI>Name <b>"+an+"</b>, value <b>"+
  session.getAttribute(an)+"</B>");
…
```

tiscover
THE TRAVEL NETWORK

Servlets
Arten von "Servlet-Variablen"

Art der Variable	Sichtbarkeit	Lebensdauer
Attribute eines **ServletContext**	Alle Servlets im ServletContext	Bis zum Ende der Web-Applikation
Attribute einer **Session**	Servlets im ServletContext, die Requests der Session behandeln	Bis zum Ablauf der Session
Instanzvariablen eines **Servlet**	Methoden des Servlet	Servlet.init() bis Servlet.destroy()
Attribute eines **ServletRequest**	Servlets, die den Request behandeln	Bis zum Ende der Request-Behandlung

tiscover
THE TRAVEL NETWORK

Java Servlets

- Grundlagen
  - HTTP-Protokoll, Request/Response, ...
- Architektur
  - Merkmale, Servlet-Methoden, Lebenszyklus, ...
- Sessions
  - Session Tracking API, Cookies, ...
- Deployment
  - Deployment Descriptor, ...
- Zusammenfassung

Servlets

Servlet container: Jakarta Tomcat

- Referenz Implementierung
- Open source: http://jakarta.apache.org/tomcat/index.html
- Aktuelle Version: jakarta-tomcat-5.5.12
- Installation (Windows):
  - EXE-File herunterladen:
    Installation in z.B. **C:\Tomcat 5.5**
  - Benötigt Java 1.5
  - Starten aus **C:\Tomcat 5.5\bin\tomcat5.exe** Erster Aufruf: http://localhost:8080/index.jsp

tiscover THE TRAVEL NETWORK

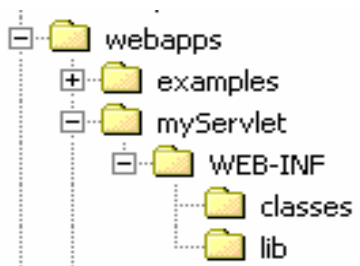
Servlets

Neues Servlet erstellen

- Neues Servlet in Tomcat erstellen:
  - Neues Verzeichnis in c:\jakarta-tomcat\webapps erstellen, z.B.
    myServlet
  - Entsprechende Verzeichnisstruktur anlegen:
  - Datei `web.xml` erstellen
  - Kompilierte Java (Servlet) Klassen in Verzeichnis `webapps\myServlet\classes` kopieren
    (bzw. in entsprechende Unterverzeichnisse,
    wenn Packages verwendet werden)

    Kompilieren eines Servlets:

    ```
    javac –classpath .; C:\Tomcat 5.5\commo
    lib\servlet-api.jar …
    ```



Servlets
Deployment

- Tomcat-Konfigurationsdatei conf/server.xml spezifiziert u.a. "Konnektoren" (Port, Protokoll), sowie die einzelnen **Kontexte**, z.B:

```
... </Context>
      <Context path="/praktikum" docBase="praktikum" debug="0" reloadable="true">
        <Logger className="org.apache.catalina.logger.FileLogger"
                prefix="localhost_praktikum_log." suffix=".txt" timestamp="true"/>
      </Context>
    </Host> ...
```

- Jeder Kontext hat ein **Document Root** Verzeichnis (docBase)
  - Enthält HTML-Dokumente, Bilder, usw.; via URL verfügbar, z.B. `http://localhost:8080/praktikum`
- Spezielles Unterverzeichnis **WEB-INF** (vor Client-Zugriffen geschützt)
  - Unterverzeichnis `classes` für Klassen, insbesonders `HttpServlet`-Klassen
  - Unterverzeichnis `lib` für jar-Archive, z.B. JDBC-Treiber
  - **Web Application Deployment Descriptor** `web.xml`

Servlets

Deployment Deskriptop `web.xml`

- Spezifikation der Servlets, Initialisierungsparameter und URLs (optional)

```
<web-app>
    <servlet>
        <servlet-name>login</servlet-name>
        <servlet-class>LoginServlet</servlet-class>
        <init-param>
            <param-name>foo</param-name>
            <param-value>bar</param-value>
        </init-param>
    </servlet>
    ...
    <servlet-mapping>
        <servlet-name>login</servlet-name>
        <url-pattern>/login</url-pattern>
    </servlet-mapping>
    ...
    <session-config>
        <session-timeout>30</session-timeout>
    </session-config>
</web-app>
```

Servlet-URL:
`http://localhost:8080/test/login`

Servlets
Tomcat starten und beenden

- Tomcat starten:
  - Auf der Kommandozeile in das Verzeichnis c:\jakarta-tomcat\bin wechseln
  - Pfad zum J2SDK 1.5.x Verzeichnis spezifizieren:

```
set JAVA_HOME=c:\j2sdk1.5
```

  - Tomcat starten mit: **startup**
  - Tomcat beenden mit: **shutdown**

- Aufruf des neuen Servlets: `http://localhost:8080/myServlet/MyServlet`

tiscover
THE TRAVEL NETWORK

Java Servlets

- Grundlagen
  - HTTP-Protokoll, Request/Response, ...
- Architektur
  - Merkmale, Servlet-Methoden, Lebenszyklus, ...
- Sessions
  - Session Tracking API, Cookies, ...
- Deployment
  - Deployment Descriptor, ...
- Zusammenfassung

Servlets
Zusammenfassung

- Standardisierter SSI-Mechanismus für HTTP-Server
- Tomcat als Referenzimplementierung der Servlet-Spezifikation
- Servlets werden in einem "Container"(Servlet-Engine) ausgeführt
- Implementierung einer Client/Server Anwendung basierend auf Request/Response-Protokoll
- Session Tracking API zur Simulation von Sessions in HTTP

Servlets
Links

- **HTML** Tutorial: `http://selfhtml.teamone.de/html/`
- **Servlet-URLs**
  - `http://java.sun.com/products/servlet/`
  - `http://www.novocode.com/doc/servlet-essentials/`
- **Servlet-Engines**
  - Apache Tomcat 4.1.x: `http://jakarta.apache.org/tomcat/`
  - Weitere Produkte: `http://java.sun.com/products/servlet/industry.html`
- API und Deployment **Dokumentation**
  - http://jakarta.apache.org/tomcat/tomcat-4.1-doc/index.html
  - http://jakarta.apache.org/tomcat/tomcat-4.1-doc/appdev/index.html

Ende der 4. Übung