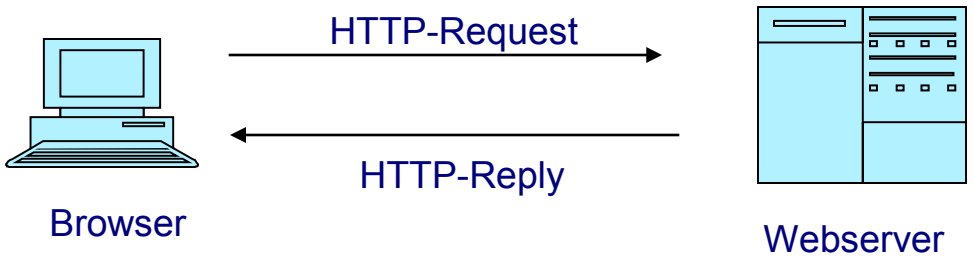
Servlets

Stephan Karrer

Die universelle Client/Server-Architektur: Browser und Webserver

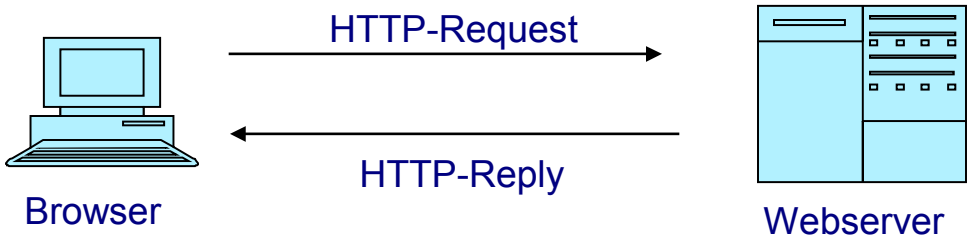
- Universeller Einsatz des HTTP-Protokolls und der Infrastruktur
- Der Webbrowser als universeller Client: Texte, Bilder, Multimedia, ...
- Vielzahl von Webbrowsern mit unterschiedlichen Erweiterungen verfügbar
- Sicherheit und Zustandsverwaltung sind Herausforderungen



- HTML kann statisch vorliegen als auch dynamisch auf der Serverseite generiert werden
- Es gibt Vielzahl von Erweiterungen um Applikationslogik anzubinden

Kommunikation via HTTP

- Standardisiert durch IETF
- Versionen: HTTP/1.0, HTTP/1.1
- Benutzt TCP
- Kommunikationsschema: Request - Response



GET /dbag/dispatch/de/kir/gdb_navigation/home HTTP/1.1

Host: deutsche-boerse.com

User-Agent: Mozilla/5.0 (Windows; U; ...

Accept: text/xml,application/xml, ...

Accept-Language: de-de,de;q=0.8,en-us;q=0.5, ...

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7 ...

Keep-Alive: 300

Connection: keep-alive

HTTP/1.1 200 OK

Date: Sun, 03 Aug 2008 11:28:10 GMT

Cache-Control: no-store

Pragma: no-cache

Expires: Thu, 01 Jan 1970 00:00:00 GMT

Last-Modified: Sun, 03 Aug 2008 11:28:10 GMT

Content-Type: text/html;charset=UTF-8

Keep-Alive: timeout=5, max=98

Connection: Keep-Alive

Content-Language: de

Das HTTP-Protokoll

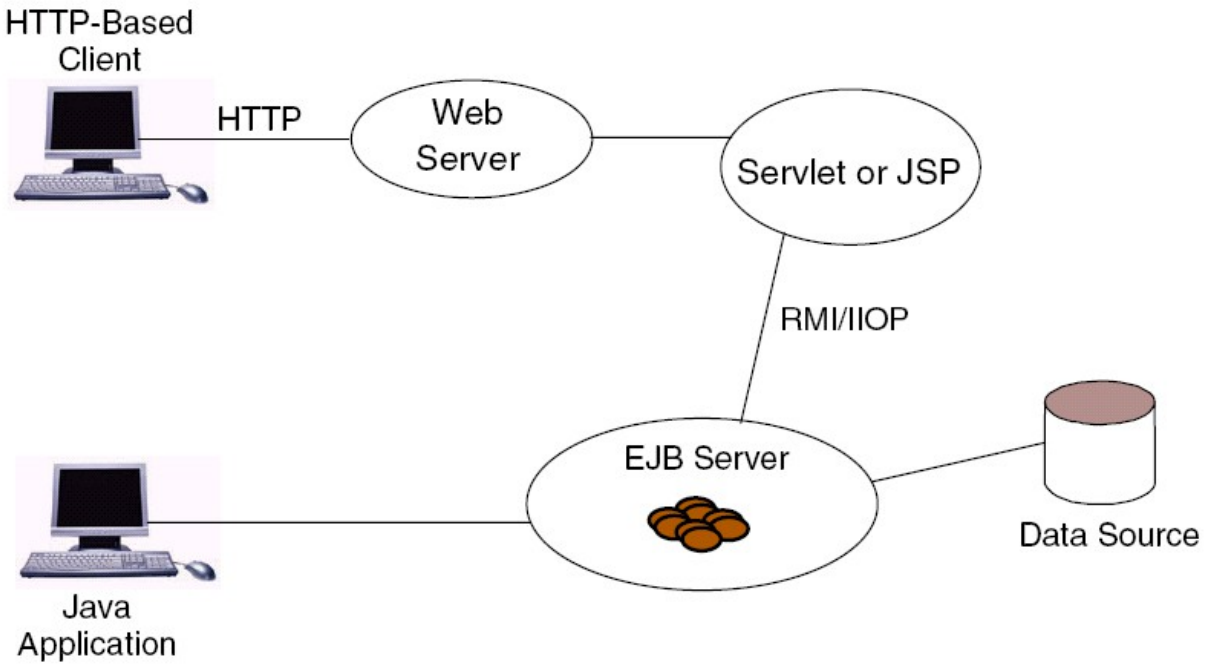
HTTP Methoden	
GET	Ressource-Anforderung
POST	Übertragung von Daten zum Server
HEAD	Server soll nur Header senden
...	

HTTP Stauscodes	
1xx	Zur Information
2xx	Erfolgreiche Operation
3xx	Umleitung
4xx	Fehler auf der Client-Seite
5xx	Fehler auf der Serverseite

Mime Typen		
Typ	Dateiendung	Bedeutung
application/zip	*.zip	ZIP-Archivdateien
image/jpeg	*.jpeg *.jpg *.jpe	JPEG-Dateien
text/html	*.htm *.html *.shtml	HTML-Dateien
text/plain	*.txt	reine Textdateien
...	...	...

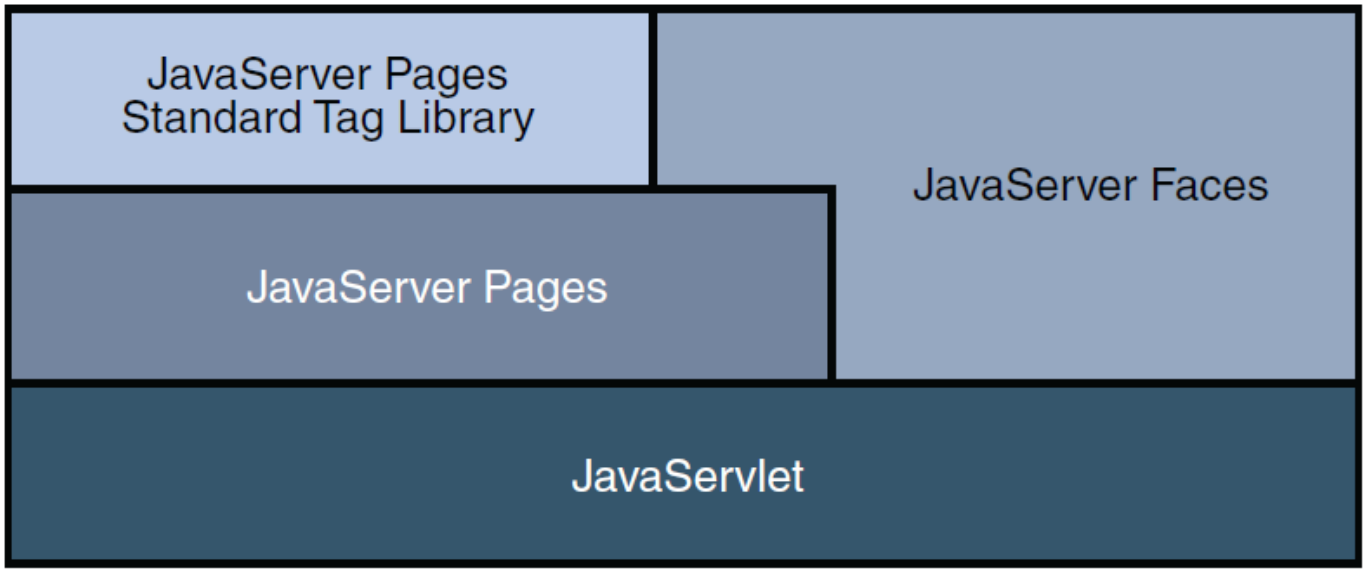
Java auf der Serverseite

- Servlets: Serverseitiges Äquivalent zu Applets
- Java Server Pages: Eigene Skriptsprache zur Ergänzung
- Enterprise Java Beans: Applikationslogik-Bausteine die iinerhalb eines J2EE-Containers laufen



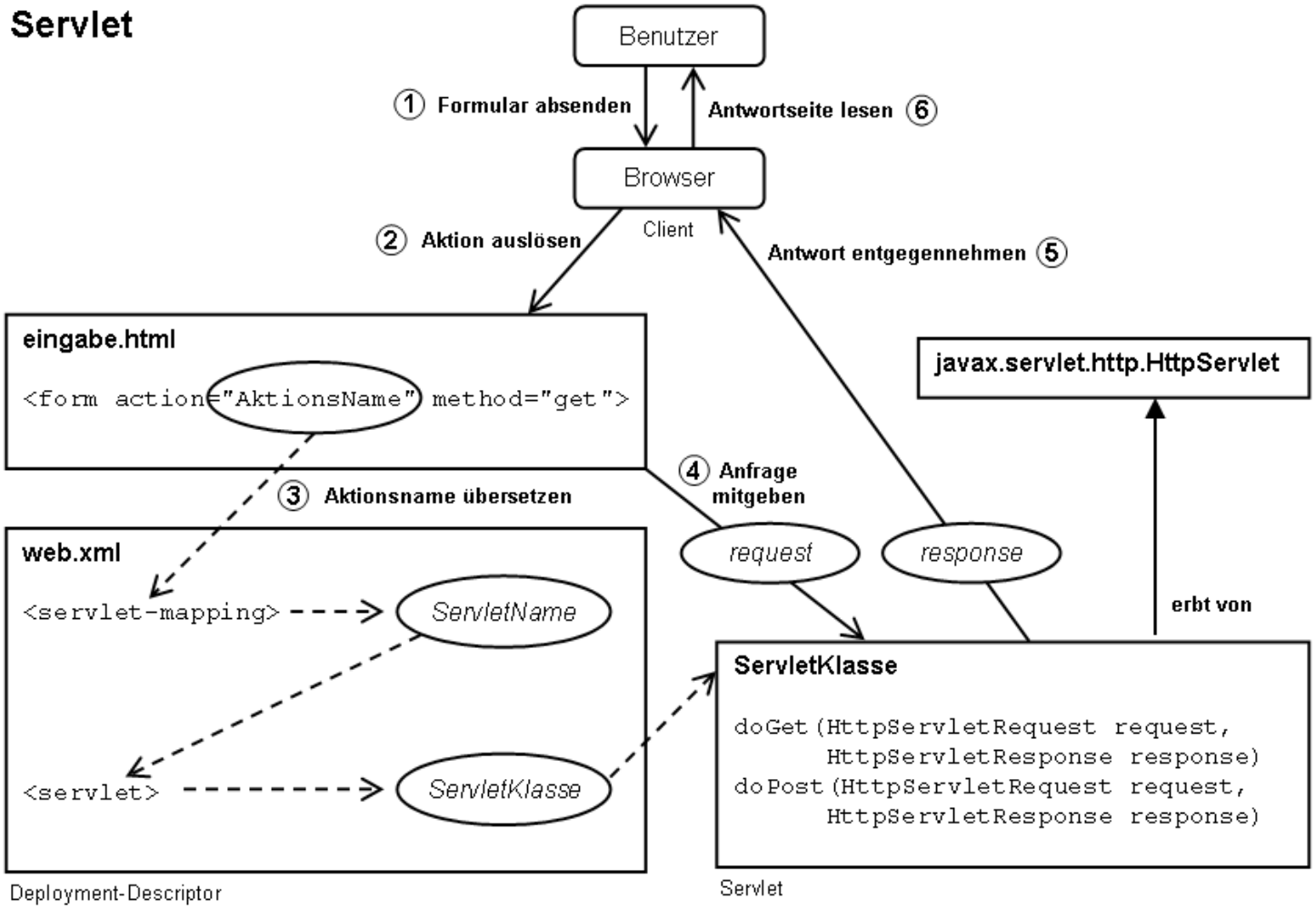
- Wichtige Server-Produkte:
  - Apache: Web-Server
  - Tomcat: Erweiterung des Apache um Servlet-Engine
  - JBoss und Bea Weblogic: Komplette J2EE-Appliationsserver

Java – Web-Technologien



- Servlet- und JSP-Technologie ist Teil der JEE-Spezifikation
  - Aktuell: (Anfang 2011) - Servlet Specification 2.5 (Version 3.0 in Arbeit)
    - Java Server Pages 2.1
- Als Framework für die Entwicklung von Web-Applikationen setzt sich zunehmend JavaServer Faces durch

Datenverarbeitung via Servlets



Ein einfaches Beispiel

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

public class HalloServlet extends GenericServlet {
      public void service (ServletRequest req, ServletResponse res)
                throws ServletException, IOException
      {
         res.setContentType ("text/html");
         PrintWriter output = res.getWriter();
         output.println ("<HTML><BODY>Hallo</BODY></HTML>");
      }
}
```

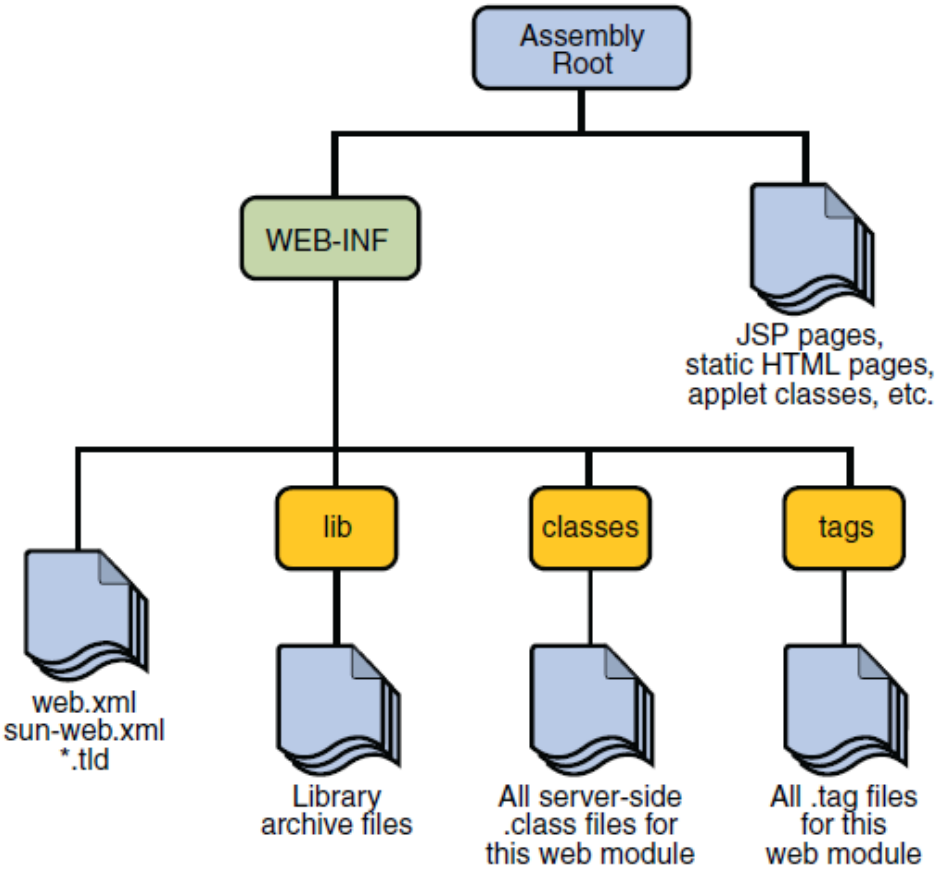
- Die abstrakte Klasse ***GenericServlet,*** implementiert die Interfaces ***Servlet*** und ***ServletConfig,*** ist die allgemeine Basisklasse für Servlets.
- Für eine Implementierung muss die abstrakte Methode ***service()*** implementiert (überschrieben) werden.

Infrastruktur für die Servlet-Verarbeitung

- Wir benötigen einen Web-Server, z.B. Apache
- Wir benötigen einen Servlet-Container, z.B.
  - Tomcat: Open-Source-Referenzimplementierung bzgl. der Servlet 2.4 und JSP 2.0 Spezifikation
  - Sun Java System Web Server (Teil des Appl.Servers GlassFish): Java EE 5 Referenzimplementierung (Servlet 2.5 Spezifikation)
  - Jetty: Ein weiterer freier HTTP-Server und Servlet-Container unter der Apache-Lizenz
  - ...
- Das Zusammenspiel muss konfiguriert werden
- Die Servlets und alle benötigten Ressourcen müssen bereit gestellt (deployed) werden

Struktur einer Web-Applikation (Web-Modul)

- Der Name des Basis-Verzeichnisses (Assembly Root) entspricht in der Regel dem logischen Namen der Web-Applikation.
- Direkt unter dem Basisverzeichnis liegen alle Ressourcen, auf die der Browser Zugriff erhält.
- Unter dem Verzeichnis WEB-INF liegen alle server-seitigen Ressourcen.
- web.xml: Deployment-Deskriptor der Web-Applikation
- Die Struktur kann in ein jar-Archiv gepackt werden und bildet ein sog. Web Archiv (WAR file, .war)



Beispiel: Deployment-Deskriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
        id="WebApp_ID" version="2.5">
  <web-app>
        <display-name>Hallo</display-name>
        <description>Servlet sagt Hallo</description>
        <servlet>
            <servlet-name>HalloServlet</servlet-name>
            <servlet-class>HalloServlet</servlet-class>
        </servlet>
        <servlet-mapping>
            <servlet-name> HalloServlet</servlet-name>
            <url-pattern>/servlet/*</url-pattern>
        </servlet-mapping>
  </web-app>
```

- Der Deployment-Deskriptor beschreibt die Parameter für das Zusammenspiel zwischen Container und Servlet.
- Die Struktur ist durch eine entsprechendes Schema (bzw. DTD) vorgegeben.

Beispiel: Nutzung von HttpServlet

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

public class HalloServlet extends HttpServlet {

 protected void doGet(HttpServletRequest req, HttpServletResponse res)
              throws ServletException, IOException  {
    res.getWriter().println("<HTML><BODY>Hallo</BODY></HTML>");
  }
}
```

- Für die Verwendung mit HTTP, insbesondere bei der Verarbeitung von Formulardaten, wird üblicherweise die abstrakte Kindklasse ***HttpServlet*** der Klasse ***GenericServlet*** benutzt.
- Die ***service()***-Methode ruft je nach HTTP-Kommando die entsprechende ***doXXX***-Methode auf.
- Mindestens eine dieser Methoden sollte sinnvollerweise überschrieben werden.

Verarbeitungsmethoden eines Servlets

Für die jeweiligen HTTP-Kommandos (ausser CONNECT) stellt die Klasse **javax.servlet.http.HttpServlet** eine entsprechende Methode zur Verfügung

```
protected void doGet(HttpServletRequest req,
                     HttpServletResponse resp)
protected void doPost(HttpServletRequest req,
                      HttpServletResponse resp)
protected void doHead(HttpServletRequest req,
                      HttpServletResponse resp)
protected void doPut(HttpServletRequest req,
                     HttpServletResponse resp)
protected void doDelete(HttpServletRequest req,
                        HttpServletResponse resp)
protected void doOptions(HttpServletRequest req,
                         HttpServletResponse resp)
protected void doTrace(HttpServletRequest req,
                       HttpServletResponse resp)
```

Ein/Ausgabedaten für ein Servlet

- Über das ***ServletRequest-*** bzw. das ***HttpServletRequest-Objekt*** kann auf folgende Informationen zugegriffen werden:
  - Daten (Parameter) des Clients
  - Attribute des Request
  - Header der HTTP-Anfrage
  - Pfadinformationen bzgl. der Anfrage,
  - Cookies
  - Attribute einer eventuellen SSL-Verbindung
  - Vom Client für eine Antwort bevorzugte Sprache

- Über das ***ServletResponse-*** bzw. das ***HttpServletResponse-Objekt*** werden die Rückgabedaten an den Container geliefert.

Content-Typ

```
protected void doGet(HttpServletRequest req, HttpServletResponse res)
                throws ServletException, IOException  {
        res.setContentType("text/html; charset=ISO-8859-1");
        res.getWriter().println("<HTML><BODY>Hallo</BODY></HTML>");
        String s = res.getContentType();
    }
}
```

- Mittels der Methoden ***get/setContentType()*** kann der Content-Typ und auch das Character-Encoding abgefragt bzw. gesetzt werden.
  - Ein Browser versucht ansonsten das Beste daraus zu machen.
  - Das Setzen muss vor dem Aufruf von ***getWriter()*** erfolgen.
- Der verwendete Zeichensatz kann auch über die Methoden ***get/setCharacterEncoding()*** abgefragt bzw. gesetzt werden.
  - Standardmäßig wird ***ISO-8859-1*** verwendet.
  - Die implizite Vorgabe wird durch das jeweilige ***Locale*** bestimmt.
  - Das Setzen muss ebenfalls vor dem Aufruf von ***getWriter()*** erfolgen.

Interaktion via Formularen

Über Formulare können Benutzereingaben getätigt werden

- Die Eingabedaten werden als Zeichenketten zum Web-Server übertragen

```
<form action="/servlet/TestServlet" method="POST">
  Technologie : <input type="text" name="search" value="java">
        <br><br>
        Region : <select name= "region" size="4" multiple>
        <option value="BA">Bayern</option>
        <option value="HE">Hessen</option>
        <option value="BW">Baden-Würtenberg</option>
        <option value="NRW">Nordrhein-Westfalen</option>
        </select>
        <br><br>
        <input type="submit" value="Search Job">
</form>
```

Zugriff auf Formulardaten

```
public void doPost(HttpServletRequest req,
                   HttpServletResponse res) {
        String searchstring = req.getParameter("search");
        String[] reglist = req.getParameterValues("region");
        Enumeration e = req.getParameterNames();
        Map map = req.getParameterMap();
        //...
}
```

- Das Interface **HTTPServletRequest** stellt entsprechende Zugriffsmethoden zur Verfügung:

Beispiel: Ausgabe aller Parameter

```
public void doPost(HttpServletRequest req,
                   HttpServletResponse res) {
   res.setContentType("text/html");
   PrintWriter out = res.getWriter();
   out.println("<h3>Alle Parameter</h3> <ul>");
   Enumeration names = request.getParameterNames();
   while(names.hasMoreElements()) {
     String param = (String)names.nextElement();
     String[] values = req.getParameterValues(param);
     out.print("<li>"+param+":");
     for(inti=0;i<values.length;i++) {
        out.print("<li>"+values[i]); }
     out.println("</ul>");
   }
}
```

Zugriff auf die Kopfzeilen des HTTP-Pakets

```
public void service(HttpServletRequest req,
                            HttpServletResponse res) {
  Enumeration headers = req.getHeaderNames();
  while (headers.hasMoreElements()) {
    String header = (String) headers.nextElement();
    String value = req.getHeader(header);
    //...
  }  }
```

Das Interface **HttpServletRequest** definiert Zugriffsmethoden:

```
String getHeader(String name)
Enumeration getHeaders(String name)
Enumeration getHeaderNames()
int getIntHeader(String name)
...
```

Zugriff auf Netzwerkparameter

Das Interface **ServletRequest** definiert Zugriffsmethoden:

```
String getLocalAddr()
int getLocalPort()
String getLocalName()
String getRemoteAddr()
int getRemotePort()
String getRemoteHost()
int getServerPort()
String getServerName()
String getScheme()
String getProtocol()
boolean isSecure()
...
```

Setzen der HTTP-Parameter beim Senden

```
boolean containsHeader(String name)
void setDateHeader(String name, long date)
void setHeader(String name, String value)
void setIntHeader(String name, int value)
void addDateHeader(String name, long date)
void addHeader(String name, String value)
void addIntHeader(String name, int value)
```

- Das Interface **HTTPServletRequest** stellt entsprechende Zugriffsmethoden zur Verfügung
- „set“: Überschreibt gegebenenfalls
  „add“: Fügt gegebenenfalls weiteren Wert hinzu

Setzen des HTTP-Statuscode

```
public void doPost(HttpServletRequest req,
                   HttpServletResponse res) {
        /* ... */
        res.setStatus(HttpServletResponse.SC_BAD_REQUEST);
}
```

- Der Web-Container löscht den Puffer und setzt den Location-Header, die restlichen Kopfzeilen bleiben unverändert.
- Entsprechende Konstanten für die Statuscodes sind in **HttpServletResponse** definiert.
- In vielen Fällen setzt der Web-Container die Statuscodes automatisch.

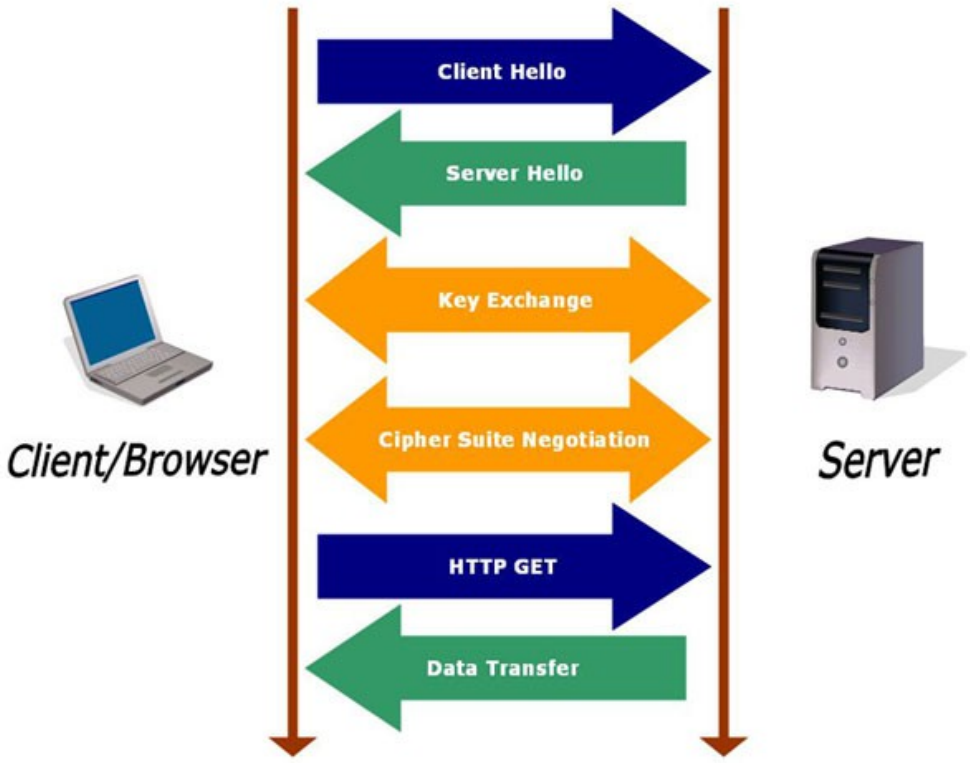
Zustandsverwaltung (Session-Management)

- HTTP ist zustandslos
  - Session-Management und Transaktionsmanagement sind Herausforderungen

Ansätze:

- Verborgene Formularfelder
  - Der Server fügt verborgene Felder in jedes gesendete Formular ein, die dann stets wieder mitgeliefert werden
- URL-Rewriting
  - Die dynamisch erzeugten HTML-Seiten enthalten URLs, die um Session-Informationen erweitert sind.
- Cookies
  - Server kann mit jeder Response-Nachricht Session-Informationen an den Client zur Speicherung übertragen.
  - Bei jedem weiteren Zugriff auf diesen Server überträgt der Client diese Information.
- Eigene Session, z.B. gesicherter Kommmunikationskanal via HTTPS (HTTP over SSL)

Geschützte Kommunikation via SSL bzw. TLS



Quelle: https://www.securetrust.com/resources/how-ssl-works

- SSL bzw. sein Nachfolger TLS werden üblicherweise für eine abgeschirmte Kommunikation eingesetzt.
- Es können verschiedene Verschlüsselungsverfahren ausgehandelt werden.
- Bei den heute üblichen Szenarien (HTTPS) identifiziert sich nur der Server anhand eines Zertifikats.
  - Server benötigt gültiges Zertifikat
  - Client überprüft das Server-Zertifikat
- Vorsicht: nur der Transport ist Ende-zu-Ende verschlüsselt
  - Die Echtheit der Daten und deren Verarbeitung ist nicht geregelt.