Webprogrammierung

The HTML Editor

CoffeeCup

Welcome to Amazon.com Books!

One million titles, consistently low prices.

(If you explore just one thing, make it our personal notification service. We think it's very cool!)

Spotlight! -- August 16th

These are the books we love, offered at Amazon.com low prices. The spotlight moves **EVERY** day so please come often.

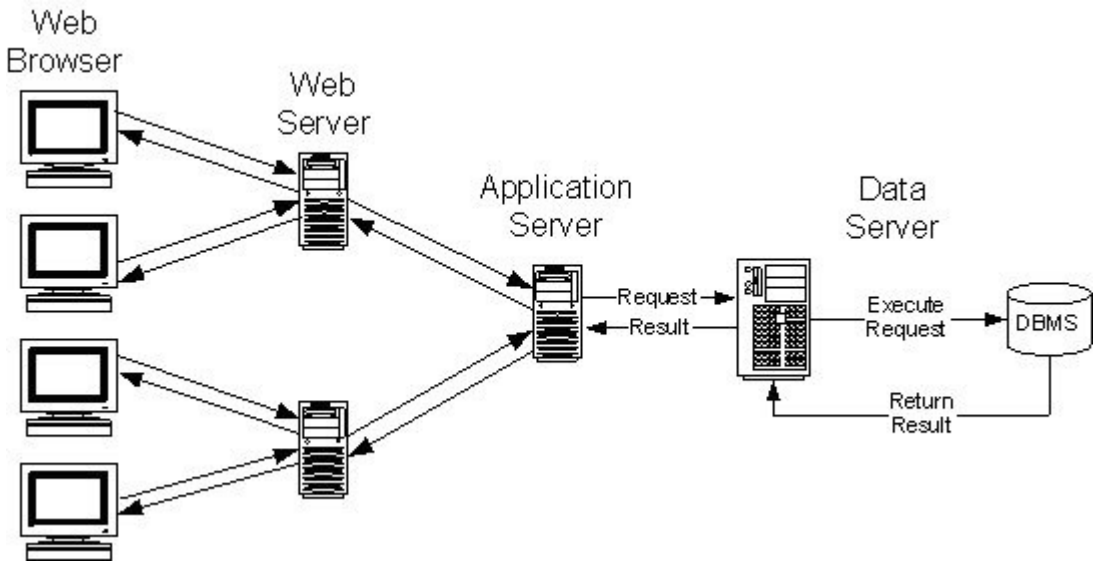
One Million Titles

Search Amazon.com's million title catalog by author, subject, title, keyword, and more... Or take a look at the books we recommend in over 20 categories... Check out our customer reviews and the award winners from the Hugo and Nebula to the Pulitzer and Nobel... and bestsellers are 30% off the publishers list...


razorfish
WILL CODE
HTML For
FOOD
PLEASE
HELP
GOD
BLESS
YOU!

Client/Server-Modell

- Asymmetrisches Modell
- Server stellen Dienste bereit, die von Clients genutzt werden können

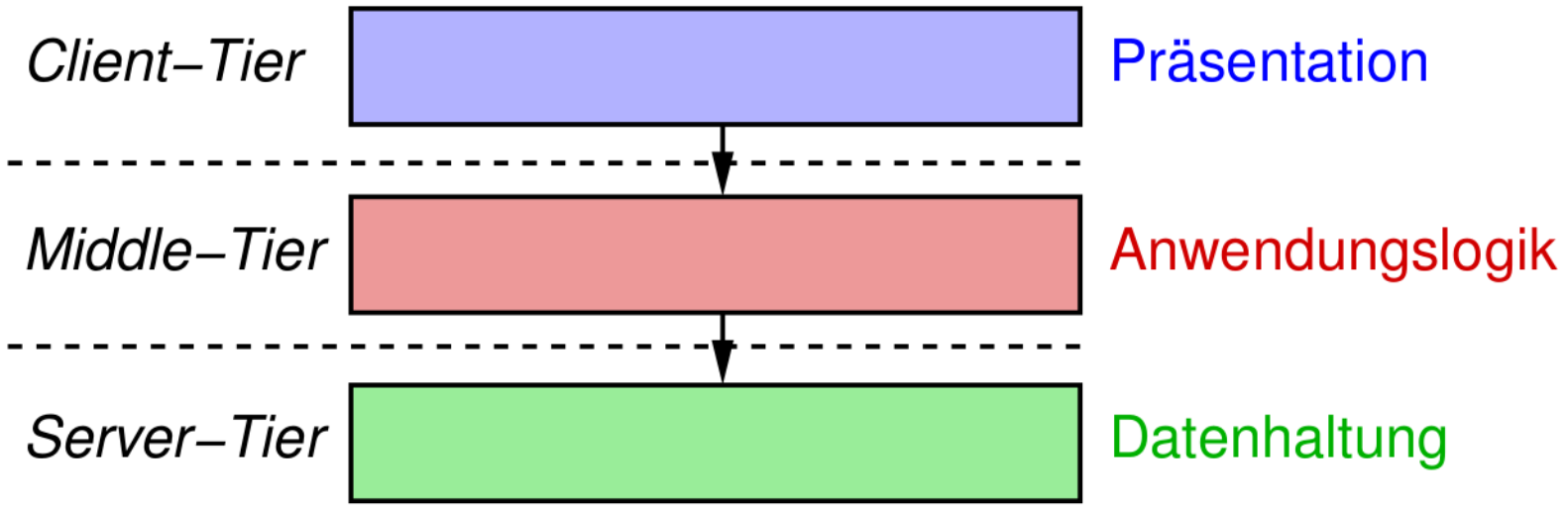


n-Tier-Architekturen

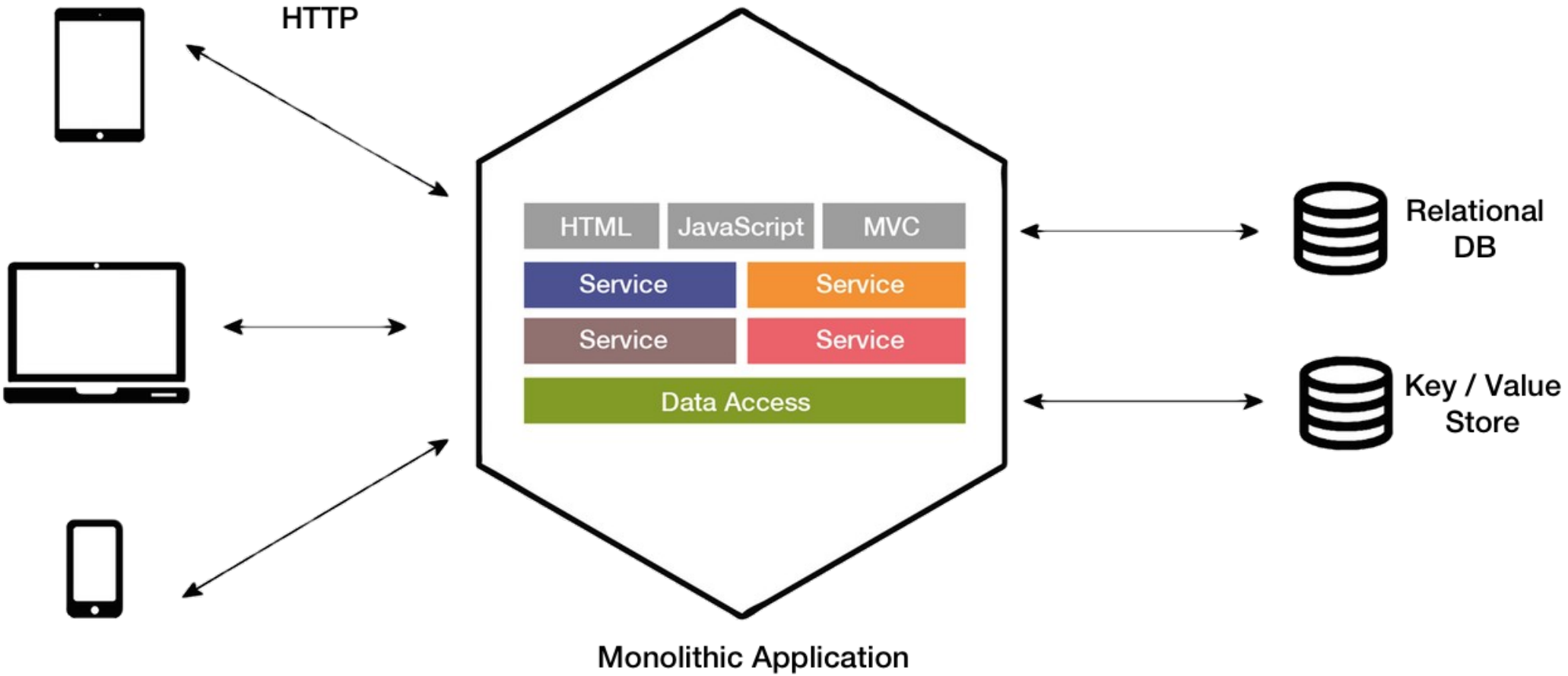
- Verfeinerungen der Client/Server-Architektur
- Modelle zur Verteilung einer Anwendung auf die Knoten einer verteilten Systems
- Vor allem bei Informationssystemen verwendet
- Tier kennzeichnet einen unabhängigen Prozeßraum innerhalb einer verteilten Anwendung

3-Tier-Architektur

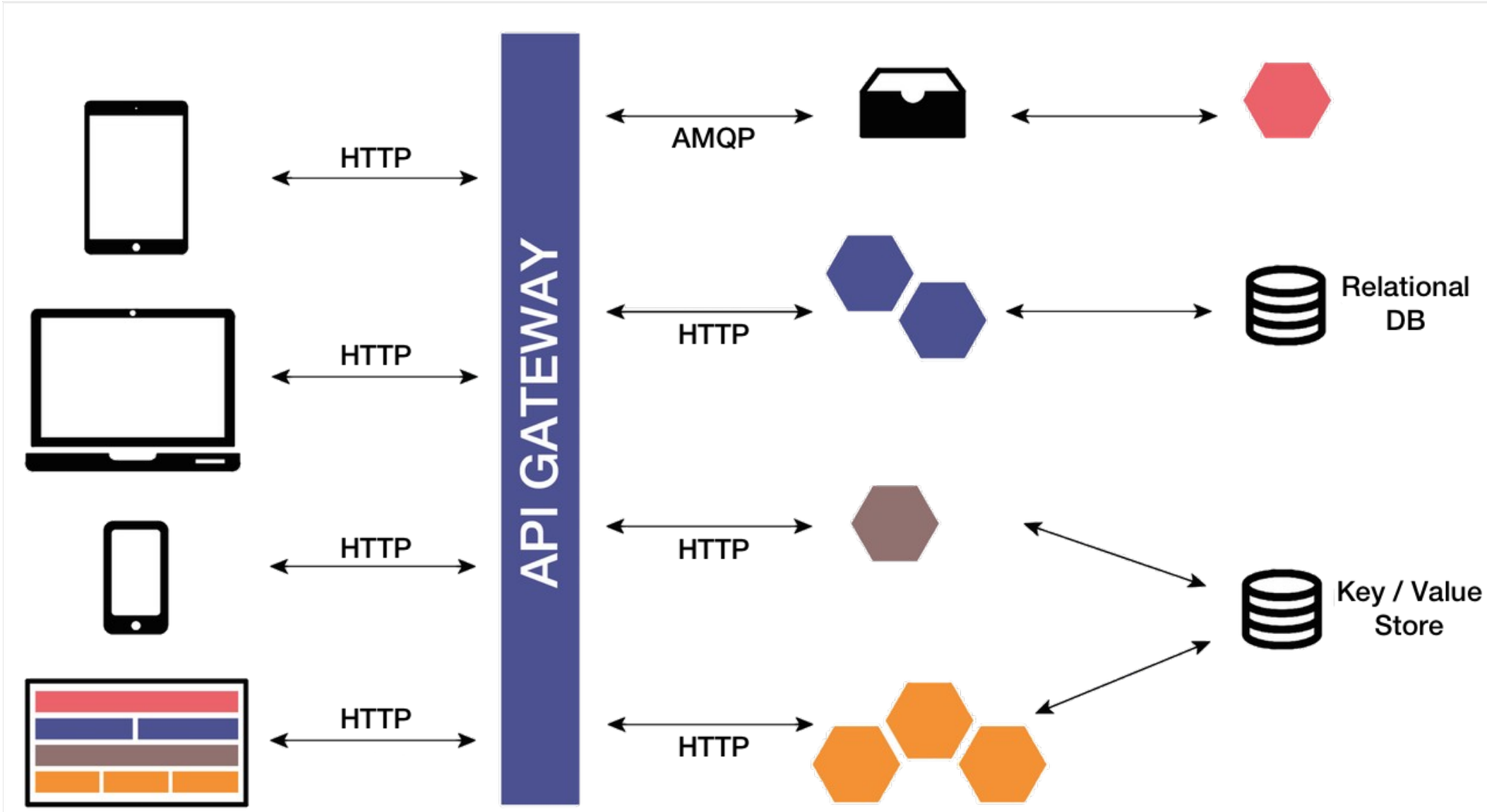
- Standard-Verteilungsmodell für einfache Web-Anwendungen
- Client-Tier : Web-Browser zur Anzeige
- Middle-Tier : Web-Server mit Servlets/JSP...
- Server-Tier : Datenbank-Server



typische Internet-Anwendung



Not today, Sonny Jim

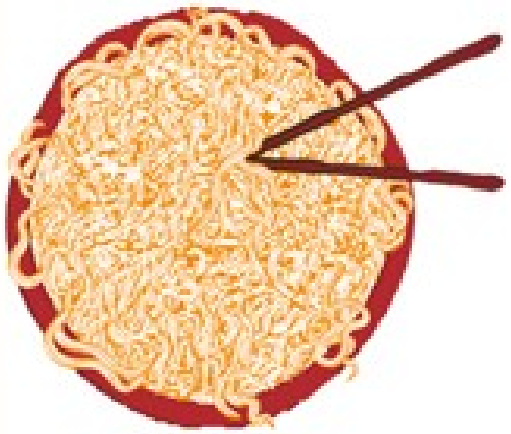



WHY THE FUCK
VIA 9GAG.COM
DOES WWW TAKE LONGER TO SAY THAN
WORLD WIDE WEB

1990s and earlier

Pre-SOA (monolithic)

Tight coupling

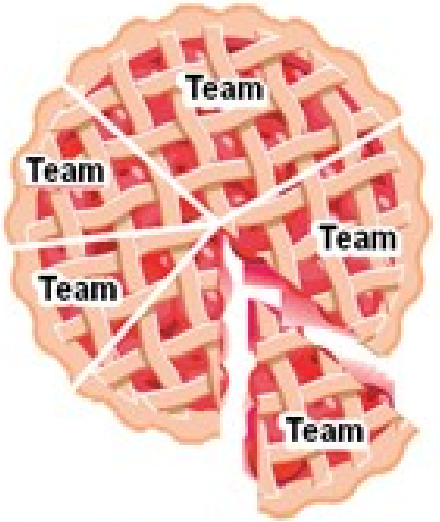


For a monolith to change, all must agree on each change. Each change has unanticipated effects requiring careful testing beforehand.

2000s

Traditional SOA

Looser coupling

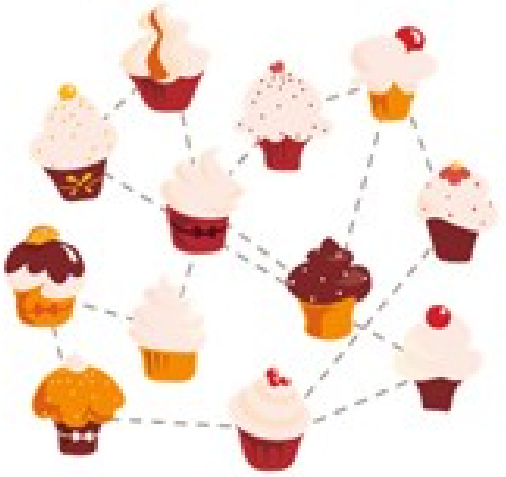


Elements in SOA are developed more autonomously but must be coordinated with others to fit into the overall design.

2010s

Microservices

Decoupled



Developers can create and activate new microservices without prior coordination with others. Their adherence to MSA principles makes continuous delivery of new or modified services possible.

Was ist der Unterschied zwischen einem verteilten System und einem Ein-/Mehrprozessorsystem?

- Mehrprozessorsystem:
  - pseudo-parallel durch time sharing
  - globale Zeit: alle Ereignisse in den Prozessen lassen sich zeitlich eindeutig ordnen
  - globaler Zustand: zur jeder Zeit kann ein eindeutiger Zustand des Systems angegeben werdenzeitlich eindeutig ordnen
- Verteiltes System:
  - echte Parallelität
  - keine globale Zeit
  - kein eindeutiger globaler Zustand

Globale Zeit

- Auf Ein-/Mehrprozessorsystem
  - jedem Ereignis kann (zumindest theoretisch) ein eindeutiger Zeitstempel derselben lokalen Uhr zugeordnet werden
  - bei Mehrprozessorsystemen: Synchronisation am gemeinsamen Speicher
- In verteilten Systemen:
  - viele lokale Uhren (eine pro Knoten)
  - exakte Synchronisation der Uhren nicht möglich
  - Reihenfolge von Ereignissen auf verschiedenen Knoten nicht (immer) eindeutig zu ermitteln

Motivation

- Peer-to-Peer -Anwendung, Prozesse senden sich gegenseitig Aufträge
- wann kann die Anwendung terminieren?
- wenn kein Prozeß mehr einen Auftrag bearbeitet
- Aufträge können noch in Nachrichten unterwegs sein!

Motivation

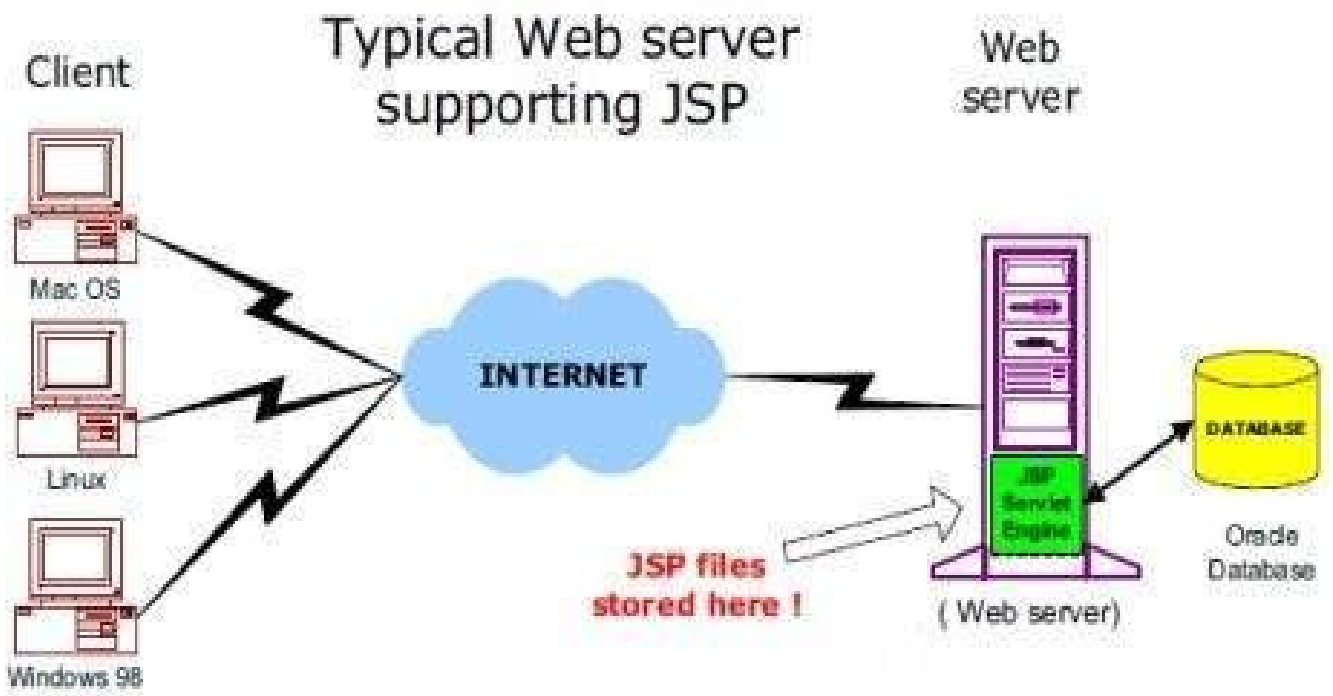
- Wie bestimmt sich der Gesamtzustand eines verteilten Prozeß-systems?
- Summe der Zustände aller Prozesse??
- Zwei Aspekte müssen beachtet werden
- Nachrichten, die noch in Übertragung sind
- Fehlen einer globalen Zeit
  - Zustände der Prozesse beziehen sich immer auf lokale
  - ein Globalzustand zur Zeit *t* kann nicht definiert werden

Middleware

- Middleware ist Schnittstelle zwischen verteilter Anwendung und verteiltem System
- Software glue
- Verbergen der Verteilungsaspekte vor der Anwendung
- Middleware kann auch Zusatzdienste für Anwendungen bieten

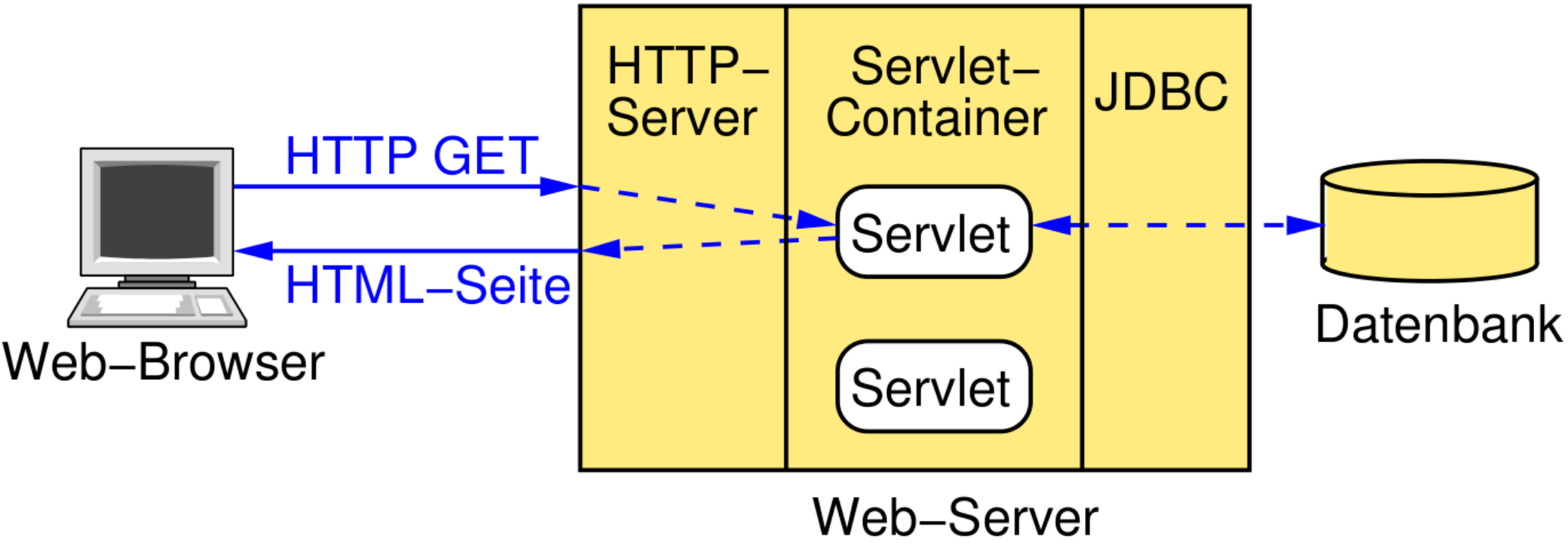
Servlets und JSP

- Servlets
- Java Server Pages (JSP)



Servlets

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



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
- Container lädt Servlets bei Bedarf dynamisch nachontainer
- (HTTP-)Servlets werden über HTTP-Anfragen angesprochen (GET, POST)
- Servlet bearbeitet die Anfrage und erzeugt eine HTML-Seite
- Bearbeitung erfolgt durch eigenen Thread im Adreßraum des Web-Servers

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

  GET /buy.html?what=shoe&price=50.00 HTTP 1.0
- begrenzte Länge der URL

GET und POST

- Teil des HTTP-Protokolls: Browser-Anfragen an den Server
- Auch verwendet in HTML-Formularen

  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

- javax.servlet.http.HttpServlet
- Überschreiben einer der Methoden

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

void doPost(...)
```

Implementierung

- void init() : gerufen, wenn Servlet geladen wird
- void destroy() : gerufen, wenn Servlet entfernt wird
- Einige weitere Methoden
- Tomcat
- Hello World Beispiel

```java
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++;

    // Extrahiert Parameter 'name' aus URL
    String name = request.getParameter("name");
    response.setContentType("text/html");

    // Strom für die erzeugte HTML Page
    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>
<HEAD><TITLE>Hello-World</TITLE></HEAD>
<BODY>
  <P>
    <A HREF="http://localhost:8080/test/hello?name=Cat">
    Say Hello to Cat</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

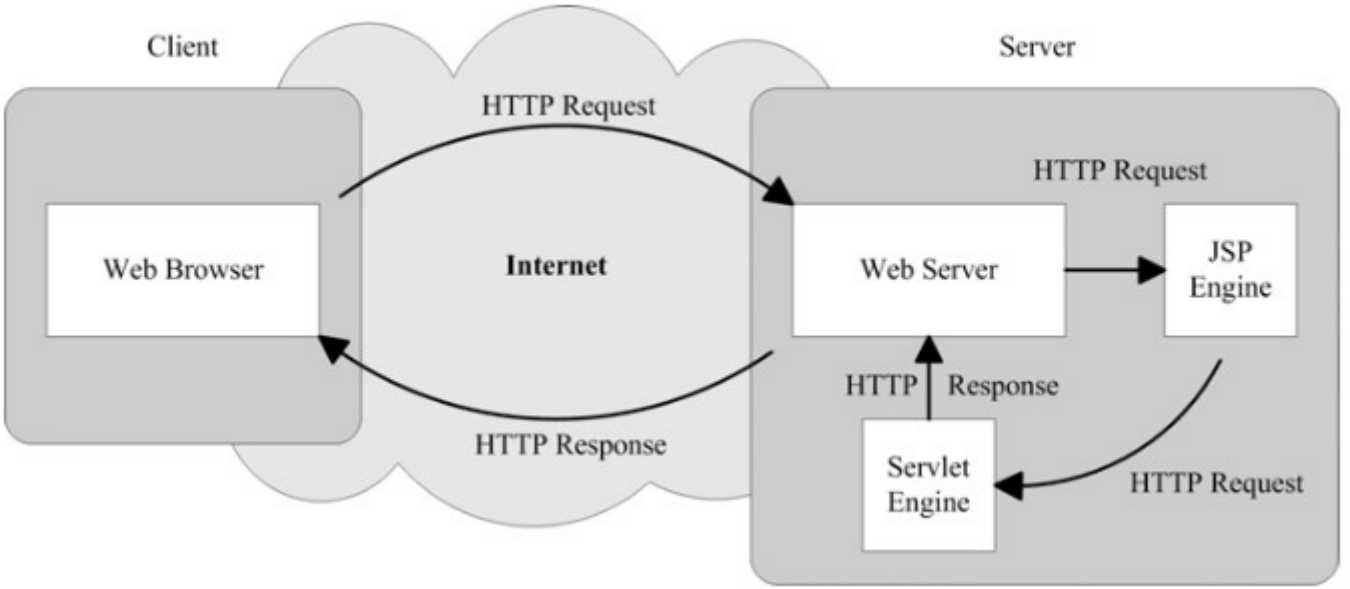
- Übersetzen des Serlvets
- javac -cp $CATALINA_HOME/lib/servlet-api.jar: HelloSrv.java
- Erstellen eines Deployment-Deskriptors unter WEB-INF/web.xml
- cp HelloSrv.class WEB-INF/classes
- jar -cvf test.war WEB-INF
- cp test.war $CATALINA_BASE/webapps

web.xml

```
<?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>
```

Deployment mit Tomcat

- Servlet ist nun unter dieser URL ansprechbar:
- http://localhost:8080/test/hello



Lebenszyklus eines Servlets

- Beim Start des Servers oder durch Client-Anfrage:
  - Beim Start des Servers oder durch Client-Anfrage
  - 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
- Bei Entfernung des Servlets aus dem Server
  - Aufruf der Methode destroy()

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
  - HttpSession getSession()
- HttpServletResponse: HTTP-Antwort
  - void setContentType(String type)
    - setzt MIME-Typ der Antwort
  - PrintWriter getWriter()
    - Liefert PrintWriter zum Schreiben der Ausgabe

Sitzungs-Management

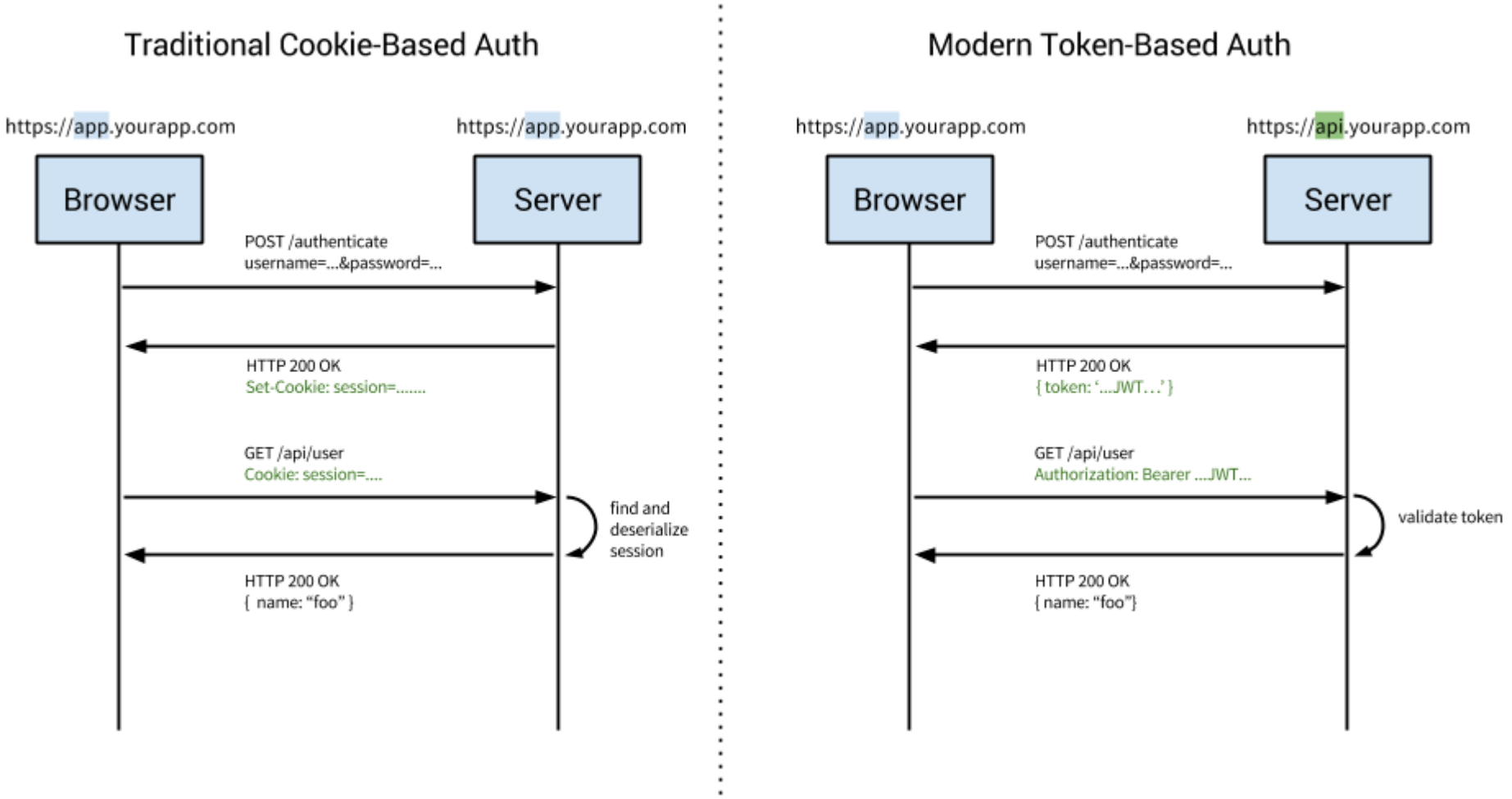
- Methode getSession() erlaubt Management von Client-Sitzungen
  - erzeugt neue Sitzung, falls noch keine existiert
  - liefert Sitzungs-Objekt HttpSession als Ergebnis
- Verfolgung von Sitzungen:
  - Server erzeugt eindeutige Sitzungs-ID
  - Sitzungs-ID wird als Cookie im Client gespeichert
  - Cookie wird bei jeder erneuten Anfrage an Server ubertragen

Das HttpSession-Objekt

- Identifiziert eindeutig eine Client-Sitzung
- Erlaubt, beliebige Information sitzungs-lokal zu speichern
- Wichtige Methoden:
  - String getId(): liefert Sitzungs-ID
  - boolean isNew()
  - void setAttribute(String name, Object value)
    - Speichern sitzungslokaler Daten unter gegebenem Namen
  - Object getAttribute(String name)
    - Auslesen sitzungslokaler Daten mit gegebenem Namen

```java
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>");
    }

    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();
}
```

Session vs Cookies



Java Server Pages

- einfachere Generierung dynamischer HTML-Seiten
- Probleme von Servlets:
  - Ausgabe von HTML-Code in println()-Anweisungen ist umständlich / fehleranfällig
  - Installation und Deployment von Servlets ist schwierig
- Losungsidee:
  - Einbetten von Java-Servlet-Code in statische HTML-Seiten
  - beim ersten Aufruf der Seite wird über JSP-Compiler auto- matisch ein Servlet erzeugt und in WWW-Server geladen

Spezielle Tags

- Tag für Ausdücke: <%= Ausdruck %>
  - Wert des Java-Ausdrucks erscheint in HTML-Ausgabe
    - <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 ausgef uhrt
- Ausgabe wird in HTML-Ausgabe eingefügt
- Java-Code kann auch mit regulärem HTML-Code gemischt werden

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>
```

Spezielle Tags

- Tag zur Deklaration globaler Variablen: <%! Deklaration %>
- globale Variable = Attribut der erzeugten Servlet-Klasse
- Wert bleibt über alle Aufrufe der JSP-Seite hinweg erhalten

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

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
- JSPWriter für HTML-Ausgabe

Beispiel

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

Beispiel

```
<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;
    }  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>
```

JSP Direktiven

- JSP bietet einige Direktiven, die das Verhalten der JSP-Seite kontrollieren
- Allgemeine Syntax: <%@ Direktive %>
- Wichtige Beispiele:
- <%@ include file="copyright.html" %>
- <%@ page import="javax.rmi.*, javax.naming.*" %>

Servlets

- Servlets sind Java-Klassen zur Bearbeitung von HTML-Anfragen in WWW-Servern
- Erzeugung dynamischer HTML-Seiten
- genau eine Instanz der Servlet-Klasse
- Sitzungs-Management kann/muß explizit programmiert werden
- JSP: Vereinfachter Umgang mit Servlets
  - Mischung von statischem HTML-Code und Servlet-Code
  - Servlet wird dynamisch aus JSP-Seite genieriert