Java

Servlets, Java Server Pages (JSP)

Server-API-Ansätze

Ausführbare Programmeinheiten im Web-Server

- Entwickelt um Nachteile des CGI/Server-Side-Scripting zu überwinden
- Programtechnischer Ansatz: Anweisungen stehen nicht als Script-Anweisungen in HTML-Datei, sondern bilden eigenständige Anwendung
- Übersetztes Programm wird über eine definierte Schnittstelle (Container) im Server zur Laufzeit eingebunden
- Benötigen speziellen Server, der den Container bereit stellt

Erweiterungen werden in den Adressraum des Servers geladen

- Werden nur einmal geladen
- Werden in Threads statt Prozessen ausgeführt

Bekannteste Vertreter

- ASP.NET (Microsoft)
- Java Servlets (Oracle)

Was ist ein Servlet?

Servlet

- Eine abgeleitete Java-Klasse, die
- von einem Container verwaltet wird
- dynamische HTML-Inhalte (oder auch anderes) generiert

Servlet-Engine (Container – hier unser Web-Server!)

- Enthält und verwaltet die Servlets über ihren gesamten Lebenszyklus
- Bindet die Servlets über feste vorgegebene Interfaces ein
- Stellt die Dienste zum Empfangen von Anfragen und Senden von Antworten bereit
- Bildet somit die Kommunikationsendpunkte für die Servlets, die so über die http-URL angesprochen worden

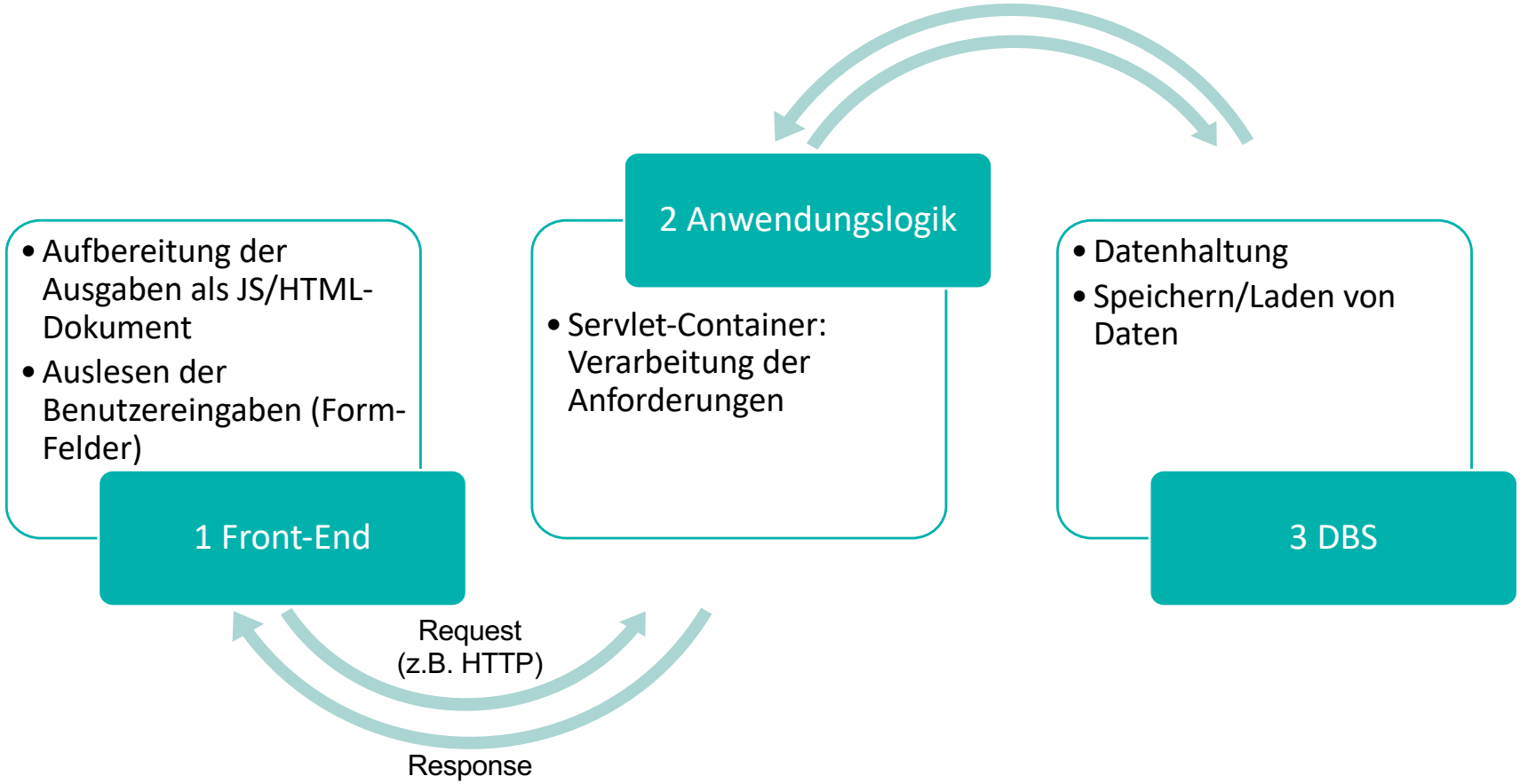
Ein Servlet besitzt keine main-Methode

Programmtechnische Nutzung durch die Servlet-API

- Teil des Software Development Kits der Jakarta Enterprise Edition (JEE)
- javax.servlet: Protokoll-unabhängige Klassen und Interfaces
- javax.servlet.http: http-spezifische Erweiterungen
- Steuerungsmöglichkeiten durch Annotationen

Ablauf einer HTTP-Anfrage (3-Tier-Architektur)

- Anmerkung: Es gibt auch nicht http-spezifische Servlets



Servlets
Vergleich der serverseitigen Konzepte

CGI

- Schnittstelle um externe Programme auszuführen
- Jede Anfrage erzeugt einen eigenen Prozess
- Sprach-/System-unabhängiges Konzept

PHP

- Aktive Anweisungen im HTML-Objekt
- Werden beim Einlesen vom HTTP-Server zur Laufzeit interpretiert
- Alle Anfragen besitzen jeweils eine eigene Instanz
- Session-Management-Funktionen

Servlet

- Dauerhafter Prozess der zumeist auch über Anfragen hinweg existent ist (Lebenszyklus wird vom Container verwaltet)
- Mehrere Anfragen interagieren mit der gleichen Instanz

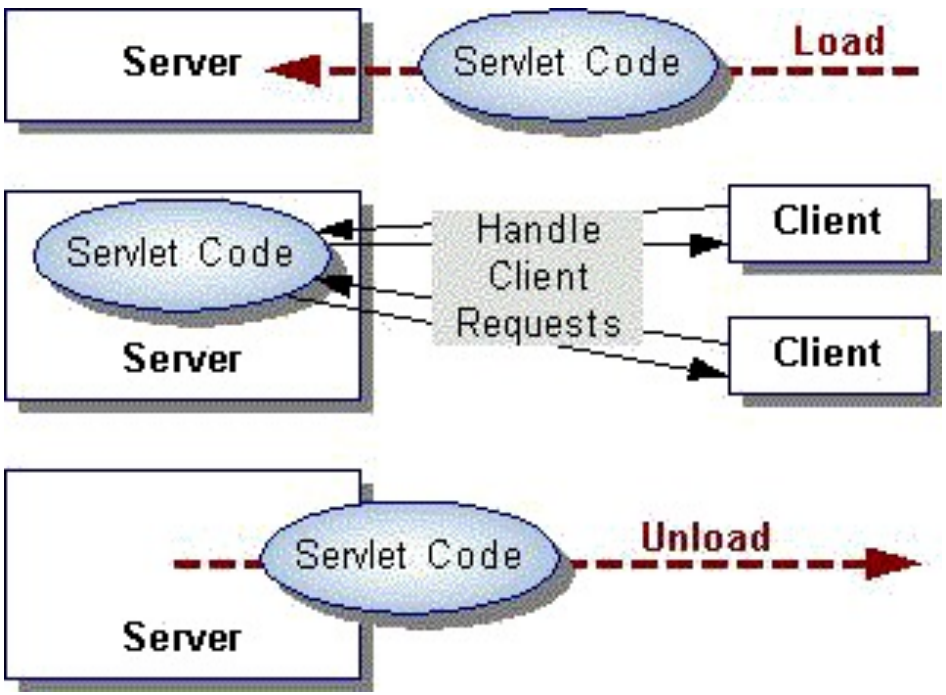
Servlets
Lebenszyklus

Ein Servlet unterliegt einem genau festgelegten Lebenszyklus

1. Laden der Servlet-Klasse und instanziieren (on demand)
2. Initialisieren des Servlet-Objekts
3. Verarbeitung der verschiedenen Anforderungen
4. Entfernen des Servlet-Objekts
5. Entladen der Servlet-Klasse

- Dieser Lebenszyklus wird von einem Container verwaltet und bestimmt somit die programmtechnische Nutzung.
- Sie wird mit den Methoden *init*, *service* und *destroy* realisiert, die überladen werden können/müssen

Servlets
Lebenszyklus



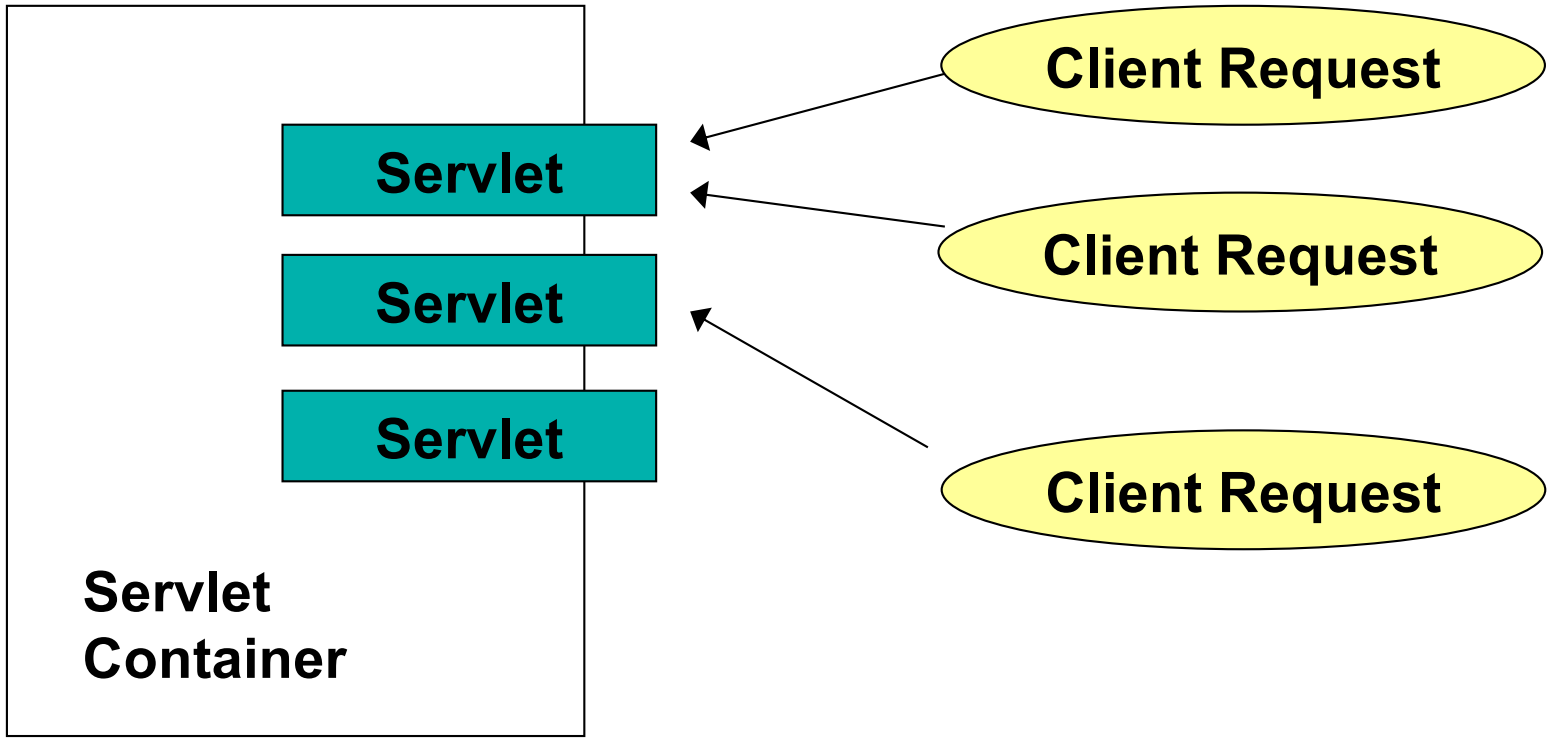
Laden der Servlet-Klasse
Erzeugen einer Instanz
Ausführen der init()-Methode

Prüfen, ob neuere Versionen des *.class-Files vorhanden sind: gegebenenfalls neu laden

Ausführen der Methode service()

Bei Herunterfahren des Servers:
Ausführen der destroy()-Methode

Servlets
Threads



Verschiedene Klienten können Anfragen **zur selben Servlet-Instanz** verschicken. Hierzu wird für **jede Anfrage** ein **eigenständiger Thread** verwendet

Optionen zum Laden und Instanziieren des Objektes

- Beim Start des Servlet-Containers (Web-Server)
- Beim Empfang der ersten Anfrage

Initialisierung

- Hängt ab von der Konfiguration (web.xml oder Annotation): Wenn hier `<load-on-startup> 1`, dann wird dies beim Starten des Containers gemacht (und nicht bei Erstanforderung des Servlets)
- Aufruf der *init*-Methode des Servlets (die überschrieben werden muss oder durch eine Annotation verändert wird)
- Globale Initialisierungsaufgaben, die für alle Anfragen erforderlich sind
  - > Herstellen einer Datenbank- oder Netzwerkverbindung
  - > Einlesen einer Konfigurationsdatei
  - > Starten eines Threads

Client-Anfragen bearbeiten

- Eine Möglichkeit der Anfragebearbeitung ist das Überladen der *service*-Methode
- Hierbei handelt es sich um eine generische Methode, die prinzipiell nicht an HTTP gebunden ist
- Der Programmcode kann auf ein Objekt vom Typ *ServletRequest* zugreifen, wodurch der Zugriff auf alle Daten der Anfrage ermöglicht wird
- Die Methode erzeugt eine generische, nicht HTTP-spezifische Antwort mittels eines Objektes vom Typ *ServletResponse*

Client-Anfragen bearbeiten (Fortsetzung)

- Wird die *service*-Methode nicht implementiert, so kann man bei einer Ableitung eines HTTP-Servlets auch die spezifischen Methoden *doGet* und *doPost* ableiten.
- Auch diese Methoden haben Zugriff auf die Daten durch ein spezielles Objekt. Bei einem HTTP-Servlet ist das Objekt vom Typ *HttpServletRequest*
- Analog wird die Antwort durch ein Objekt vom Typ *HttpServletResponse* manipuliert

Anmerkung

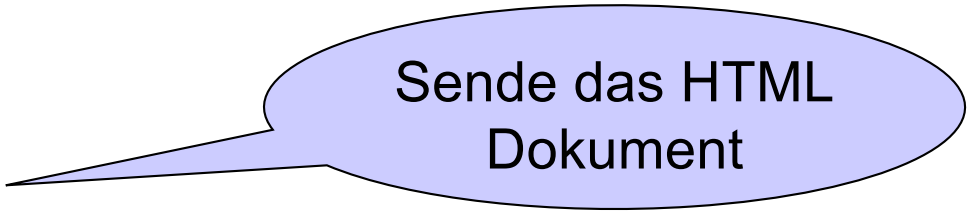
- Da Servlets i.allg. im Kontext des HTTP-Protokolls genutzt werden ist dies die gebräuchliche Form der Implementierung

Servlets
HelloWorld

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

public class HelloWorld extends HttpServlet {

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

      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
```

Servlets
HelloWorld

```
Date d = new Date();

out.println(
    "<html>\n" +
    "<head>\n" +
    "<title>Hello World</title>\n" +
    "</head>\n" +
    "<body>\n" +
    "<h1>Hello World</h1>\n" +
    "Wir haben heute den " + d.toString() +

     "</body>\n" +
    "</html>\n"
);
out.close();
```



Servlet-Klasse wieder entladen

- Der Servlet Container entscheidet, wann die Servlet-Instanz wieder aus dem Speicher entfernt wird
- Vorher wird die Methode *destroy* aufgerufen

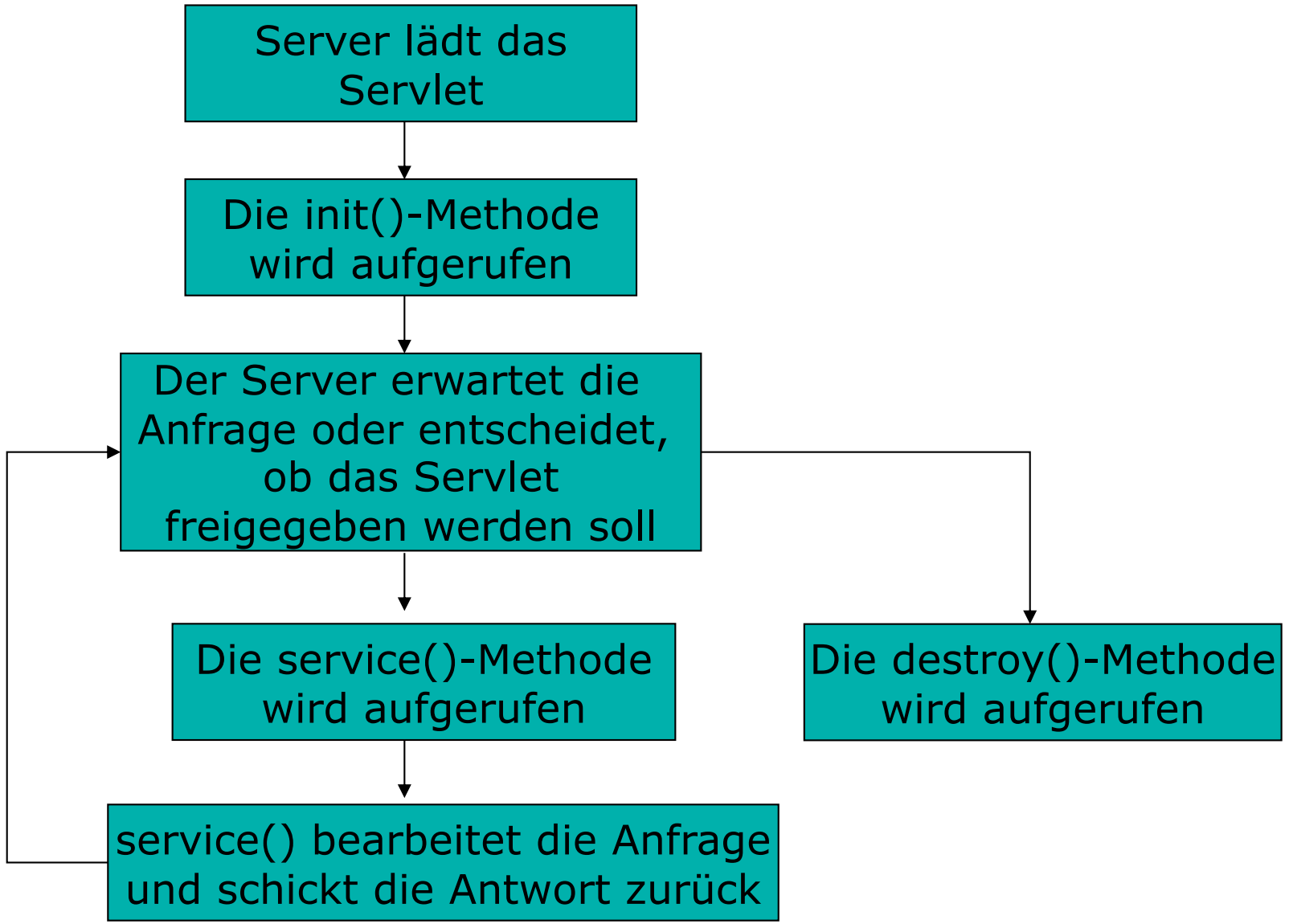
Servlets
Ein einfaches Template für Servlets

```
import javax.servlet.*;
import javax.servlet.http.*;
// other imports
public class TemplateServlet extends HttpServlet {

   public void init() {
      // Wird bei Erstellung des Servlets aufgerufen
   }

   public void destroy() {
      // Wird bei Beendigung des Servlets aufgerufen
   }

...
```

Servlets
Ein einfaches Template für Servlets

```
    public void doGet(HttpServletRequest request,
            HttpServletResponse response)
            throws IOException, ServletException {
        // Abarbeitung einer Get-Anfrage
    }

    public void doPost(HttpServletRequest request,
            HttpServletResponse response)
            throws IOException, ServletException {
        // Abarbeitung einer Post-Anfrage
    }

    // Weitere Methoden
}
```

Servlets
Ablauf



Servlets
Annotationen

Mit der Servlets 3.0-API wurden Annotationen eingeführt:

- @HandlesTypes
- @HttpConstraint
- @HttpMethodConstraint
- @MultipartConfig
- @ServletSecurity
- @WebFilter
- @WebInitParam
- @WebListener
- @WebServlet

Die Annotationen helfen dem Container dabei, das Servlet passend (URL) einzubinden und weitere Funktionalitäten zu erkennen. Sie sollten daher diese Annotationen verwenden

@WebServlet

- Diese Annotation zeichnet die Klasse als Servlet-Klasse aus und ermöglicht beispielsweise die Vergabe eines Namens (auf den sich z.B. Konfigurationen beziehen können)
- Trotzdem muss die Klasse von javax.servlet.http.HttpServlet abgeleitet sein

```
@WebServlet(
    description = "A sample annotated servlet",
    urlPatterns = {"/QuickServlet"}
)
```

```
Kurz: @WebServlet ("/QuickServlet“)
```

Servlets
Annotationen

@WebFilter

- Diese Annotation ermöglicht eine Vorverarbeitung der Anfragen, um z.B. nur authentifzierte oder gar autorisierte Anfragen zu erhalten. Basis hier sind die Methoden

```
@WebFilter("/admin")
public class MyFilter implements Filter {

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
    }

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
            throws IOException, ServletException {
    }

    @Override
    public void destroy() {

    }
}
```

https://www.codejava.net/java-ee/servlet/webfilter-annotation-examples

Der Zugriff auf Umgebungsvariablen wird über das request-Objekt vereinfacht

Hierzu gibt es individuelle Funktionen:

- PATH_INFO request.getPathInfo()
- REMOTE_HOST request.getRemoteHost()
- QUERY_STRING request.getQueryString()

Wie können mehrstufige Operationen/Transaktionen vorgenommen werden?

- Das Servlet muss erkennen, dass der nächste Aufruf der Seite in einem Kontext geschieht
- Lösung wie gehabt: Einsatz von Session-Ids mittels
  - > Cookies
  - > Sessions

```
Cookie myCookie = new Cookie("name", "value");
response.addCookie(myCookie);
```

HttpSession-Objekt

- Das Servlet kann mittels getSession() eine die aktuelle Session erhalten, bzw. eine neue anlegen
- Mit setAttribute kann hier **jederzeit** eigene Key-Value-Paare setzen

```
HttpSession session=request.getSession();
Integer count = new Integer(1);
session.setAttribute("counter",count);

HttpSession session=request.getSession(false);
if session != null
      Integer count2=(Integer)session.getAttribute("counter");

Enumeration e = session.getAttributeNames();
while (e.hasMoreElements()) {
      String name = (String) e.nextElement();
   ... session.getAttribute(name)   }
```

Servlets
Zusammenfassung

Ablauf

1. Erweitern der Klasse-HttpServlet, nutzen von Annotationen
2. Überschreiben der doGet(....) und/oder doPost-Methode
   service()-Methode als Alternative

3. Einlesen der Benutzerparameter mittels HttpServletRequest
   > getParameter("paramName")
4. Erstellen der Antwort mittels HttpServletResponse
   > Content-Type setzen
   > PrintWriter holen
   > HTML-Befehle mittels PrintWriter zum Klienten senden
   > Ggf. weitere Header-Werte setzen, z.B. Cookies
5. Servlets nutzen i.allg. eher Klassenvariablen und verwenden daher nur zu Hilfszwecken Instanzvariablen

Vorteile

- Flexible Anbindung an die Java-Welt
- Threads reduzieren den Ressourcenverbrauch
- Ergeben zusammen mit Enterprise Java Beans Komponenten eine mächtige Entwicklungsbasis

Nachteile

- Abhängig von den Fähigkeiten des http-Servers (kein nativer)
- Verwaltung der verschiedenen Sitzungen muss implementiert werden
- Vermischung von Präsentations- und Anwendungslogik

Servlets eignen sich insbesondere für die Anbindung an komplexe Standardanwendungen

- Anwendungslogik wird dabei oft in Beans Container *) realisiert (JBoss/Tomcat). Mit diesen komplexen Containern lassen sich auch REST-konforme Lösungen über Annotation regeln, z.B. JAX-RS

```
@Path("/hello")
public class HelloService{
   @GET
   @Path("/{param}")
   public Response getMsg(@PathParam("param") String msg) {
      String output = "Jersey say : " + msg;
      return Response.status(200).entity(output).build();
   }
}
```

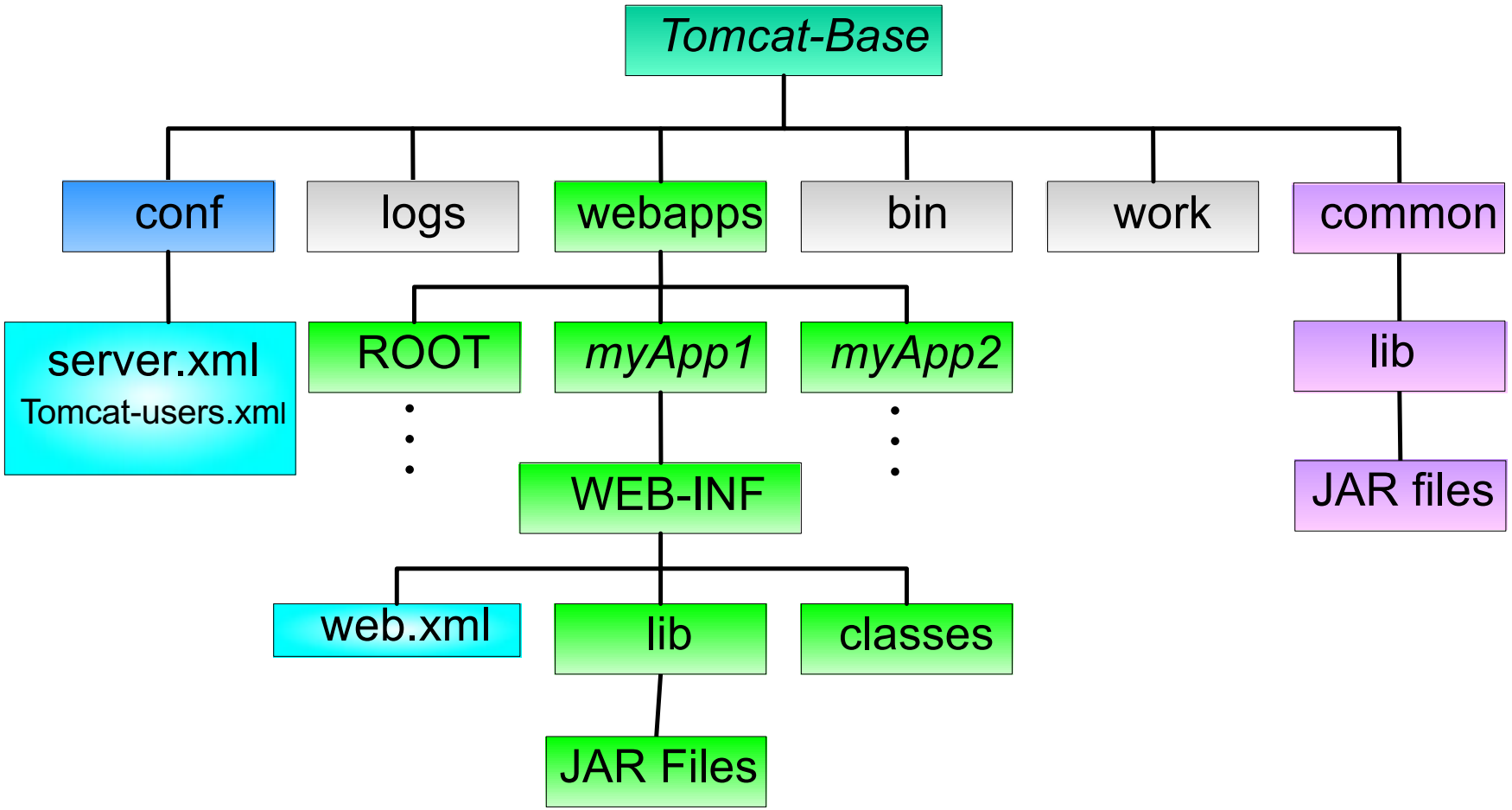
- Präsentationslogik kann besser gelöst werde

*) Enterprise JavaBeans (EJB) sind standardisierte Komponenten innerhalb eines Jakarta-EE-Servers (Jakarta Enterprise Edition). Sie vereinfachen die Entwicklung komplexer mehrschichtiger verteilter Softwaresysteme mittels Java (z.B. bei der GUI oder bei der Anwendungslogik).

Tomcat ist ein Container für Servlet- und Java Server Pages

- Open Source Referenzimplementierung der Apache Software Foundation des Servlet API
- Vollständig in Java geschriebener Web-Server
- Tomcat ist kein Application-Server, vielmehr ein Servlet-Container, der eher leichtgewichtig ist und z.B. keine Enterprise Java Beans unterstützt
- Tomcat alleine kann daher auch kein **JAX-RS**. Hierfür wäre eine Implementierung erforderlich, z.B. **Jersey**

- Die Entwicklung von Tomcat begann bereits 1999 als Nachfolger von Apache JServ
  - > Sun, IBM und Apache gründeten zusammen das Projekt „Jakarta“
  - > Ziel: Implementierung der bis dato nur theoretisch vorhandenen Spezifikation => Referenzimplementierung


Tomcat-Base
conf
logs
webapps
bin
work
common
server.xml
Tomcat-users.xml
ROOT
myApp1
myApp2
lib
WEB-INF
JAR files
web.xml
lib
classes
JAR Files

Java Server Pages (JSP)

Idee

- Umkehren des Servlet-Prinzips: „Java-Code generiert HTML“
- JSP-Seiten sind HTML-Seiten, die auch Java-Code beinhalten
- **Analog zu den behandelten Template-Mechanismen**
- Abgrenzung der Anweisungen durch spezielle Tags (ähnlich zu dem PHP-Prinzip)
  - > Nutzung einer „Skript“-Sprache (analog zu PHP)
  - > Spezielles Verfahren zur Behandlung von JSP-Seiten erforderlich
  - > Auch ohne Java-Anweisungen würde dieses Verfahren angewendet

Java Server Pages sind Servlets

JSP-Seiten werden vom Server automatisch in Servlets übersetzt

- Der Server weiß JSP-Seiten von normalen HTML-Seiten zu unterscheiden
- Er kompiliert mit Hilfe eines JSP-Compilers die Code-Segmente und erstellt ein Servlet
- Das Servlet beinhaltet die HTML-Anweisungen als übliche Ausgabe
- Die globale Datei web.xml gibt die Endung .jsp vor

Anmerkung

- Der Übersetzungsvorgang von JSP in ein Servlet muss dann nur einmal getätigt werden, danach benutzt der Servlet-Container direkt die übersetzte Klasse.

Java Server Pages sind Servlets

Prinzip

- Eine JSP-Seite ist im Prinzip nur eine **Kurzschreibweise** für ein Servlet
- Die Java-Anweisungen werden vom Server automatisch in ein zur Laufzeit erzeugtes Servlet integriert
- Die HTML-Anweisungen der JSP-Datei werden beim Kompilationsschritt einfach an den mit der service-Methode des Servlets erzeugten Ausgabestrom beigefügt

Java Server Pages
Beispiel

```
<%@ page contentType="text/html"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>JSP Beispiel - Hallo Welt!</title>
</head>
<body>

<% out.println("Hello, World!"); %>

</body>
</html>
```

Java Server Pages
Konstrukte

Es gibt neben den HTML-Elementen drei wichtige JSP-Konstrukte:

1. **Scripting-Elemente:** Spezifikation von Java-Fragmenten, die in das erstellte Servlet eingefügt werden
2. **Direktiven:** Sie kontrollieren die Gesamtstruktur des Servlets, z.B. durch Importieren einer Klasse, die in späteren Scripting-Elementen benutzt wird
3. **Aktionen:** Sie starten zusätzliche Funktionalität **zur Laufzeit** und können so die Ausführung des dynamisch erzeugten Servlets beeinflussen und sichern dessen Aktualität

Java Server Pages
Scripting Elemente

In Scripting-Elementen kann implizit auf vorhandene Objekte zugegriffen werden

- **pageContext** – Zugriff auf Objekte in den verschiedenen Gültigkeitsbereichen einer jsp-Seite
- **HttpServletRequest** –response Objekte
- **session** – sitzungsorientierte Anwendungen
- **out** – ermöglicht die Ausgabe

Java Server Pages
Scripting Elemente

Ausdrücke:

- Einfache Java-Anweisungen, die in das Servlet an entsprechender Stelle integriert werden (Ausdruck wird mittels out.print() eingefügt)
- Ermöglichen das Einfügen dynamischer Werte in eine HTML-Datei

Syntax:

- Standard: <%= Java-Ausdruck %>
- Alternative XML-Syntax:

  <jsp:expression> Java-Ausdruck </jsp:expression>
- Unterstützt werden nur einfache Ausdrücke - keine komplexen Strukturen wie z.B. Schleifen oder bedingte Anweisungen

```
Date: <%= new java.util.Date() %>
Host: <%= request.getRemoteHost() %>
```

Java Server Pages
Direktiven

Erneut gibt es drei Arten von Direktiven:

1. **page**: Steuert die Struktur eines Servlets z.B. durch das **Importieren einer Klasse** oder durch das Beeinflussen des Übersetzungsprozesses:

```
<%@ page import="java.sql.*" %>
```

2. **include**: Fügt zum Zeitpunkt der Übersetzung der JSP-Datei eine **weitere JSP-Datei** ein. Die Anweisung sollte dort stehen, wo der Text der Datei stehen soll:

```
<%@ include file="Gut-Erprobtes.jsp" %>
```

   > **Achtung**: Hauptseite und inkludierte Seite haben den gleichen Namensraum!

3. **taglib**: Ermöglicht das **Einführen eigener JSP-Tags**, die dann mittels einer sogenannten Tag-Handler-Klasse ausprogrammierte Programmsequenzen über die übliche HTML-Tag-Notation zugänglich macht

```
<%@ taglib uri="csajsp-taglib.tld" prefix="csajsp" %>
...
    <Title><csajsp: Ein Beispiel /><Title/>
```

Java Server Pages
Aktionen

Include

- Einfügen einer HTML- oder JSP-Seite zur Laufzeit (<%@ include file="..." %> dagegen fügt zu Compilierzeit ein.).

- Die Zielseite kann **dynamisch** ausgewählt werden. Hier im Beispiel hängt die Zielseite von einer Zufallszahl ab.

```
<% int zufall = (int) (Math.random()*100); %>
<jsp:include page =‘ <%= (zufall%2) = = 0 ? "seite1.jsp" :
"seite2.jsp" %> ‘ />
```

Forward

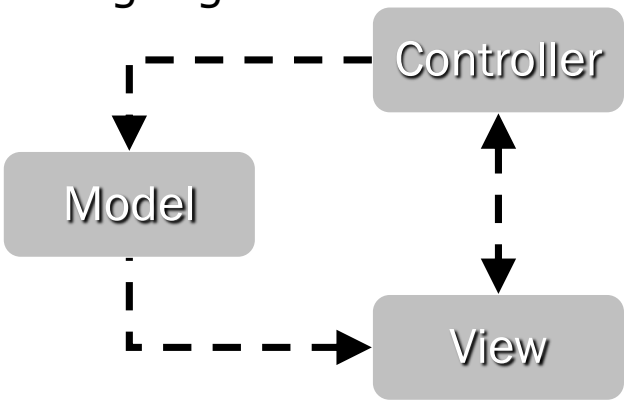
- Anfrage und Antwort werden an eine andere JSP-Seite, anderes Servlet übergeben. Die Steuerung kommt nicht mehr zur gegenwärtigen JSP zurück.

```
<jsp:forward page="subpage.jsp">
  <jsp:param name="forwardedFrom" value="this.jsp"/>
</jsp:forward>
```

Model View Controller

Das Prinzip

- Model-Objekt
  - > definiert die Datenstruktur der Anwendung
  - > speichert die Daten
  - > stellt Methoden zur Änderung der Daten zur Verfügung
  - > **Realisiert manchmal die Geschäftslogik**



- View-Objekt
  - > stellt die Bildschirmrepräsentation dar
  - > Objekt erhält die Daten vom Model
  - > Benutzer führt auf der View die Aktionen aus
  - > die Aktionen werden durch den Controller an das Model weitergeleitet)

- Controller-Objekt
  - > Reaktion und Verarbeitung von Benutzereingaben
  - > Vermittler zwischen Model und View
  - > Beinhaltet oftmals die Geschäftslogik

Model View Controller

Implementierung

- Model
  - > Realisation zumeist mittels Java Beans (nächste Folie)
  - > Java Beans enthalten die eigentliche Programmlogik
  - > Java Beans berechnen Ergebnisse und speichern Zustände
  - > Java Beans sind unabhängig von der Webschnittstelle
- View
  - > Nutzung von JSP-Seiten
  - > Anzeige des Ergebnisses
  - > Ziel: Möglichst wenig Java Code
- Controller
  - > Programmieren eines Servlets
  - > Einlesen und Überprüfen der übergebenen Parameter
  - > Aufrufen der eigentlichen Programmlogik im Model
  - > Weitergabe des Ergebnisses an das passende View

MVC - Enterprise Java Beans aus "Mastering EJB 4.0"

Enterprise Java Beans ... are meant to perform server side operations, such as executing complex algorithms or performing highly transactional business operations. Server components need to run in a highly available (7x24), fault tolerant, transactional, multi-user, secure environment. The application server provides such a server side environment for the enterprise beans...Typically, EJB components can perform any of the following tasks

- Perform business logic
- Access a database
- Integrate with other systems

Model View Controller

Enterprise Java Beans