Programmieren II

Dr. Klaus Höppner

Hochschule Darmstadt – Sommersemester 2010

Servlets

MVC-Pattern

Wie kommen Daten von einem Webserver?

In der Praxis existieren verschiedene Wege, wie ein *Server* einem *Client* zu einer Anfrage per *HTTP* ein Resultat liefert:

- Statische Seite (HTML-Dokument)
- Ergebnis eines CGI-Skripts
- Ergebnis von durch den Webserver interpretierten Code (*Server Side Includes* – SSI, *Active Server Pages* – ASP, *PHP*)
- Ergebnis der Verarbeitung durch einen Applikationsserver (*Java Server Pages* – JSP, Java Servlets)

Java-Applikationsserver

Eine Referenzimplementation eines Java-Applikationsservers stellt *Apache Tomcat* dar. Dieser besteht aus

- einem Servlet-Container, in dem Webapplikationen als Servlets ausgeführt werden,
- Compiler und Laufzeitumgebung für *Java Server Pages* (JSP)

Tomcat ist in Java geschrieben und daher auf diversen Plattformen lauffähig. In Produktionsumgebungen wird Tomcat meist mit dem Apache Webserver so kombiniert, dass statische Inhalte vom Webserver und dynamische Inhalte transparent über das *Apache JServ Protocol* (AJP) von Tomcat geliefert werden.

HTTP-Requests

Beim Aufruf einer Seite schickt der Browser einen HTTP-Request an den Server.

Hierbei gibt es zwei Methoden:

GET Hier werden zu übermittelnde Daten (Formulareingaben) innerhalb des URL-Query-Strings übertragen. Vorteil (und gleichzeitig Nachteil): Es sind Bookmarks auf Formular-Daten möglich, Daten sind in der History des Browsers sichtbar.

POST Hier werden Formular-Daten getrennt vom Query-String übertragen. Hierdurch sind die Daten nicht offen sichtbar, es können praktisch beliebig große Datenmengen (z. B. Bilddateien) übertragen werden.

Ausführung von Servlets

Tomcat stellt den Container für die Servlets zur Verfügung. Daher muss für diese beim Aufruf keine eigene Laufzeitumgebung gestartet werden, Servlets sind daher i. A. schneller als CGI-Skripte.

Die Zuordnung von einem URL-Pfad zu einem konkreten Servlet erfolgt anhand der Konfiguration in `web.xml`. Diese definiert Webapplikationen mit einem Servlet-Namen. Diesem Name wird zugeordnet:

- ein Java-Servlet-Klasse
- ein URL-Pfad

Die Zuordnung erfolgt dann durch

URL-Pfad $\rightarrow$ *Servlet-Name* $\rightarrow$ *Servlet-Klasse*

Aufbau eines Servlets

Eigene Servlet-Klassen erben i. A. von `javax.servlet.http.HttpServlet`.
Diese Klasse besitzt zwei wesentliche Methoden, die überladen werden können:

doGet zur Behandlung von GET-Requests,

doPost zur Behandlung von POST-Requests.

Beide Methoden erhalten als Parameter:

HttpRequest req mit Informationen über den Request,

HttpResponse resp zum Füllen mit den Informationen, die zurück geliefert werden.

Beispiel-Servlet

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

public class Servlet1 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req,
            HttpServletResponse resp)
            throws ServletException, IOException {
        resp.setContentType("text/html");
        PrintWriter out = resp.getWriter();
        out.println("<html>");
        out.println("<body>");
        out.println("Hallo Welt!");
        out.println("</body>");
        out.println("</html>");
    }
}
```

Deployment der Klasse

Damit das Servlet benutzt werden kann, muss es in Tomcat platziert und konfiguriert werden.

Hierfür wird im Tomcat-Verzeichnis unterhalb von `webapps` ein Unterverzeichnis `cnam` erstellt, das wiederum die Unterverzeichnisse `WEB-INF` und `WEB-INF/classes` enhält.

In `WEB-INF` wird dann die Mapping-Datei `web.xml`, in `WEB-INF/classes` die Class-Dateien platziert.

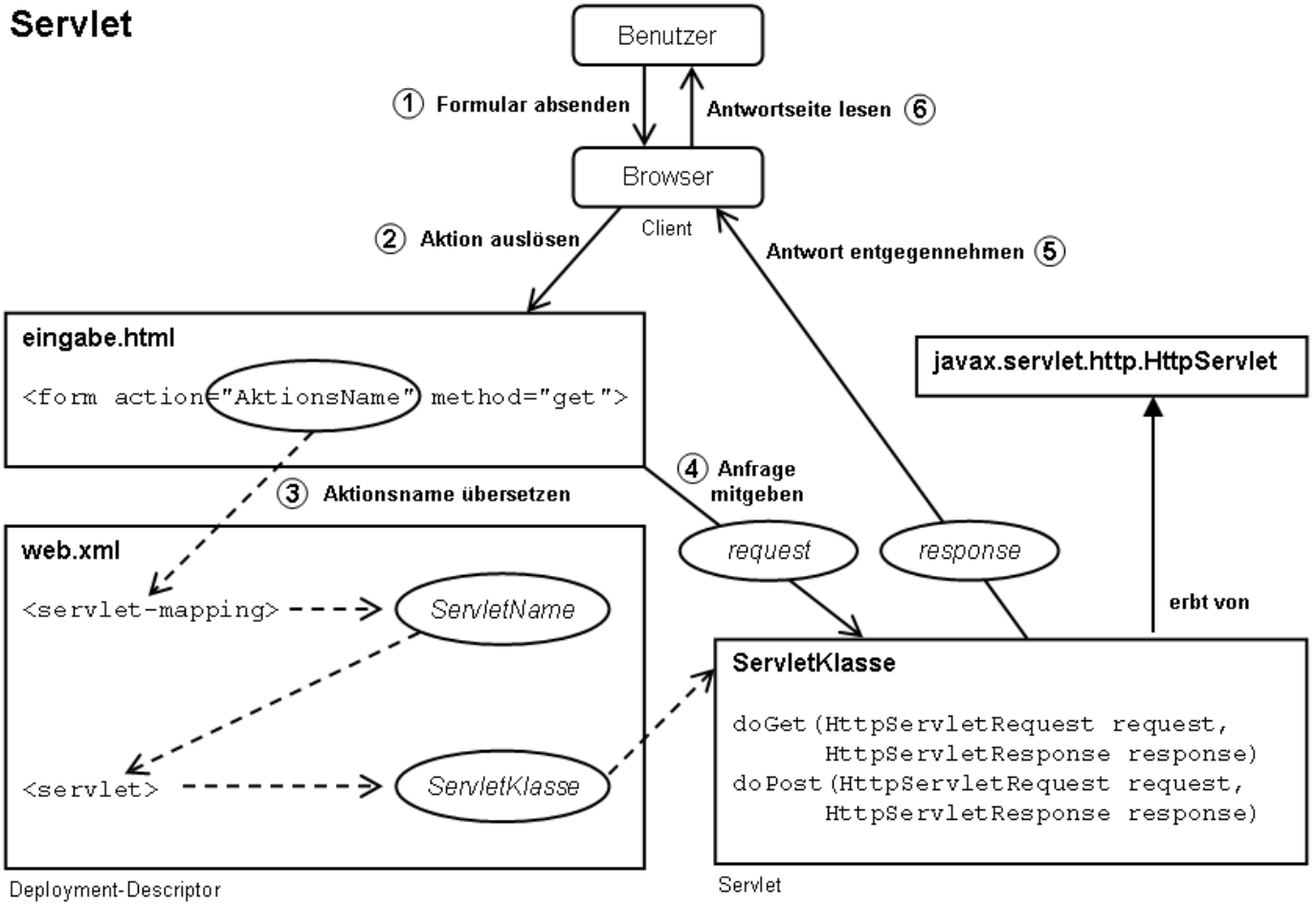
Deployment der Klasse (Forts.)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://jav
    version="2.4">
    <display-name>Hello World Application</display-name>
    <description>Beispiel</description>
    <servlet>
        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>Servlet1</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloServlet</servlet-name>
        <url-pattern>/test1</url-pattern>
    </servlet-mapping>
</web-app>
```

Analyse

- Nach Aufruf von `http://localhost:8080/cnam/test1` wird festgestellt, dass die Pfadkomponente `/test1` (bzgl. des Kontextes `/cnam`) zu dem Servlet-Namen `HelloServlet` gehört.
- Zu diesem Servlet-Namen gehört die Servlet-Klasse `Servlet1`.
- Da es sich um einen GET-Request handelt, wird von diesem Servlet die Methode `doGet` aufgerufen.
- Für die Antwort wird zuerst bekannt gemacht, dass es sich um eine HTML-Seite handelt.
- Dann wird mit `getWriter()` ein PrintWriter für das Response-Objekt geholt.
- In diesen wird dann HTML-Code geschrieben.

Schaubild des Ablaufs



Reagieren auf übergebene Parameter

```
protected void doGet(HttpServletRequest req,
        HttpServletResponse resp)
        throws ServletException, IOException {
    resp.setContentType("text/html");
    String name = req.getParameter("name");
    if (name==null) {
        name = "Unbekannter";
    }

    PrintWriter out = resp.getWriter();
    out.println("<html>");
    out.println("<body>");
    out.format("Hallo %s\n", name);
    out.println("</body>");
    out.println("</html>");
}
```

Die Klasse `HttpRequest`

Die Klasse `HttpServletRequest` bietet folgende Methoden:

`String getRequestURI()` liefert den Pfad-Anteil des URI,

`String getQueryString()` liefert den Query-String,

`String getMethod()` liefert die Requestmethode, z. B. `GET`,

`String getHeader(String name)` liefert den Header-Eintrag zu *name*,

`Enumeration getHeaderNames()` liefert eine Aufzählung aller Header-Zeilen

Die Klasse HttpRequest (Forts.)

Die URL `http://localhost:8080/cnam/test2?name=Klaus`
ergibt somit

`req.getHeader("Host")`	`localhost:8080`
`req.getRequestURI()`	`/cnam/test2`
`req.getQueryString()`	`name=Klaus`

Die Klasse `HttpResponse`

Was ein Servlet dem Client als Antwort zurück liefert, wird durch das Objekt der Klasse `HttpResponse` bestimmt, das Parameter von `doGet`/`doPost` ist.

Hier sind die folgende Methoden wichtig:

`void setContentType(String type)` bestimmt den Typ der zurück gesendeten Daten,

`PrintWriter getWriter()` liefert den Writer, in den die Antwortdaten geschrieben werden,

`OutputStream getOutputStream()` dto., aber für binäre Daten.

Arbeiten mit Sessions

Ein Server hat keine Gedächtnis . . . wenn ein Client eine URL anfordert (und hierdurch ein Servlet ausgeführt wird), weiß das Servlet nicht, dass derselbe Client kurz vorher eine andere URL angefordert hat.

Dies führt oft zu Problemen. Beispiel:

- Wenn ein Client eine Datenabfrage machen will, soll zunächst die Berechtigung geprüft werden.
- Hierzu soll jede Anfrage zunächst auf eine Seite zum Einloggen mit Namen und Passwort umgeleitet werden.
- Nach Prüfen der eingegebenen Nutzerdaten soll dann ein Servlet tatsächlich die Daten liefern.

Aber woher was das Servlet, dass der Client sich korrekt angemeldet hat (es sei denn, dieser schickt bei jeder Anfrage die Login-Daten von neuem)?

Die Klasse `HttpSession`

Ein Gedächtnis über verschiedene Request hinweg kann durch Session-Objekte realisiert werden.

Eine Session ist dabei realisiert durch:

- eine Session-ID, die die Session eindeutig identifiziert,
- (meistens) eine temporäre Datei auf dem Server, in der sich Daten gemerkt werden können,
- einen so genannten Cookie im Browser des Clients, in dem die Session-ID enthalten ist.

Die Klasse `HttpSession` (Forts.)

Das Session-Management bei Java-Servlets wird durch die Klasse `HttpSession` realisiert. Für einen HttpRequest kann mit der Methode `getSession()` das Session-Objekt, das zu dem Request gehört, referenziert werden. Falls noch keine Session existiert, wird diese angelegt.

Die Klasse besitzt folgende Methoden:

`String getId()` die Session-Id,

`String getAttribute(String name)` liefert den Wert des Parameters *name* zur aktuellen Session,

`Enumeration getAttributeNames()` liefert ene Aufzählung aller Session-Parameter.

`long getCreationTime()/getLastAccessedTime()` die Zeit des Anlegens bzw. letzten Verwendung der Session in ms seit dem 1.1.1970

Beispiel

```
public void doGet(HttpServletRequest request,
        HttpServletResponse response)
throws IOException, ServletException
{
    response.setContentType("text/html");

    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<body bgcolor=\"white\">");

    HttpSession session = request.getSession();
    out.println("SessionID: " + session.getId());
    out.println("<br>");

    String name = request.getParameter("name");
    if (name != null) {
        session.setAttribute("name", name);
    }
```

Beispiel (Forts.)

```
        out.println("Name: " + session.getAttribute("name"));
        out.println("<br>");

        out.format("<form action=\"%s\" method=\"get\"\n",
                request.getRequestURI());
        out.println("<input type=\"text\" name=\"name\">");
        out.println("<br>");
        out.println("<input type=\"submit\">");
        out.println("</form>");
        out.println("</body>");
        out.println("</html>");
    }
```

MVC-Pattern

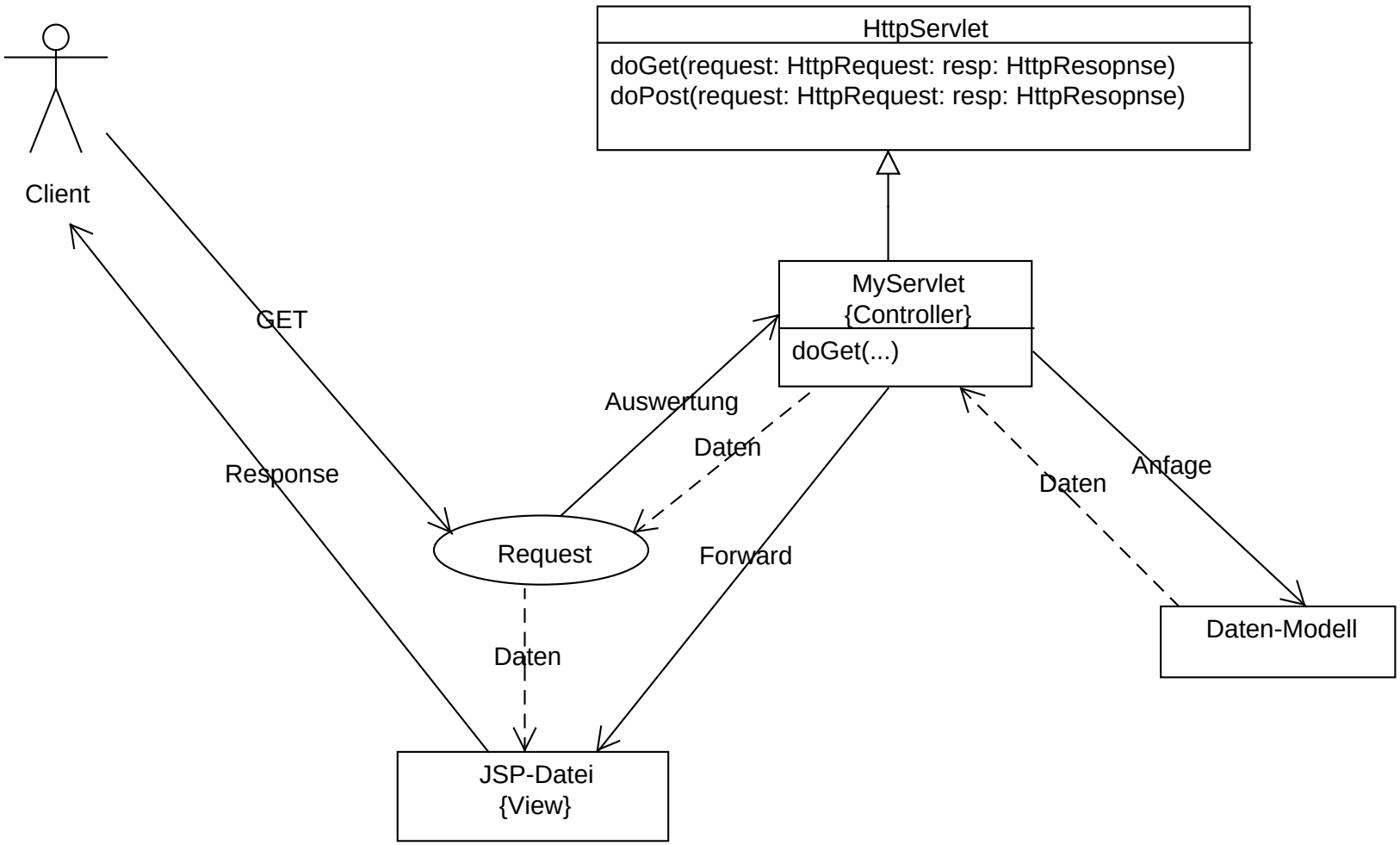
Bei Webanwendungen (wie bei anderen Anwendungen, z. B. GUI) wird oft nach dem Entwurfsmuster *MVC* vorgegangen:

Model Modellschicht, insbesondere das Datenmodell, mit der von der Webanwendung unabhängigen Geschäftslogik,

View Präsentationsschicht, hier zuständig für die Aufbereitung der Daten in HTML,

Controler Steuerungsschicht, die die Anfrage analysiert, an das Datenmodell weiter gibt, das Ergebnis entgegennimmt und an die Präsentationsschicht weiter leitet.

Schaubild


HttpServlet
doGet(request: HttpRequest: resp: HttpResopnse)
doPost(request: HttpRequest: resp: HttpResopnse)
Client
MyServlet
{Controller}
doGet(...)
GET
Response
Auswertung
Daten
Request
Forward
Daten
Anfage
Daten
Daten-Modell
JSP-Datei
{View}

Verteilte Rollen

Schicht	Implementiert als . . .
Model	POJO
	Plain Old Java Objects
	Normale Java-Klassen
View	JSP
	Java Server Pages
	d. h. mit Java-Tags angereichertes HTML, ähnlich PHP
Controler	Servlet