Seminar „DB-Aspekte des E-Commerce“ – Servlets und JavaServer Pages

Boris Stumm

24. Juni 2001

Zusammenfassung

In dieser Abhandlung werden die beiden Web-Komponenten-Technologien der Java 2 Enterprise Edition (J2EE) behandelt. Servlets und JavaServer Pages werden vorgestellt und an Beispielen erläutert. Weitere Aspekte sind Trennung von Präsentation und Logik sowie Sicherheit. Weiterhin wird das MVC-Pattern vorgestellt und verschiedene Architekturmodelle angesprochen, nach denen J2EE-Anwendungen entwickelt werden.

1 Überblick

Die Java 2 Enterprise Edition Plattform (J2EE) stellt verschiedene Komponenten-, Kommunikations- und Diensttechnologien zur Unterstützung von Multitier-Anwendungen bereit.

Diensttechnologien bieten Zugriff auf andere betriebliche Informationssysteme, und Kommunikationstechnologien ermöglichen die Client-Server-Kommunikation und die Zusammenarbeit von Objekten, die auf verschiedenen Servern existieren. Komponententechnologien werden benutzt, um die eigentliche Anwendung, Benutzerschnittstelle sowie Geschäftslogik,zu erzeugen ([2] Kapitel 2: J2EE Platform Technologies).

Die J2EE Plattform spezifiziert zwei Web-Komponenten, Servlets und JavaServer Pages (JSP) ([2] Kapitel 2.1.2). Sie werden als Benutzerschnittstelle in Web-basierten J2EE-Anwendungen benutzt und unterstützen die dynamische Generierung von Webseiten. Weitere Komponententechnologien sind JavaBeans (siehe 2.3.5), Enterprise JavaBeans, Applets und Application Clients ([2] Kapitel 3 und 5).

In diesem Abschnitt wird ein kurzer Überblick über die Funktionsweise und Anwendung von Servlets und JavaServer Pages gegeben, der zweite Abschnitt befasst sich dann genauer mit der Technologie, und im dritten Abschnitt werden verschiedene Architekturmodelle für J2EE-Applikationen vorgestellt. Im letzten Abschnitt werden wichtige Stichpunkte noch einmal kurz wiederholt.

Weitere Informationen über J2EE sind auf der J2EE-Webseite [1] und in [17], [18] und [19] zu finden.

1.1 Servlets

Servlets sind Java-Klassen, die bestimmten Anforderungen der J2EE Spezifikation genügen müssen, und dann als Erweiterungen in Webserver eingebaut werden können.

Sie sind Plattform- und Protokollunabhängige Server-Erweiterungen, und arbeiten nach dem Anfrage/Antwort („*Request/Response*“) Paradigma [3].

Anfragen können weitergeleitet werden an andere Servlets, beispielsweise zur Lastverteilung, und Antworten von anderen Servlets können in die ei-

gene Antwort eingefügt werden, was die Modularisierung der Anwendung unterstützt.

Wie gesagt sind Servlets protokollunabhängig, aber meist werden sie mit dem HTTP [6] eingesetzt, was durch die im Servlet API definierten Klassen weitere Möglichkeiten wie Sitzungsverwaltung („*Session Management*“) eröffnet. Da HTTP ein zustandsloses Protokoll ist, bietet die Servletspezifikation Möglichkeiten, einen Zustand eines Clients über mehrere Anfragen hinweg zu speichern.

Servlets werden, in Verbindung mit JSPs, zur dynamischen Generierung von Webseiten eingesetzt. Auf die verschiedenen Möglichkeiten dieses Einsatzes wird später noch eingegangen.

1.2 JavaServer Pages

Die JSP-Technologie wurde von Sun entwickelt, um die Erzeugung von Webseiten mit dynamischen Inhalten zu erleichtern [5]. Vereinfacht kann man sagen, das JavaServer Pages HTML-Seiten [7] sind, in die Java-Kode eingebettet ist.

Das Hauptziel war es, die Präsentation von der Erzeugung von Inhalten zu trennen. eine JavaServer Page dient hauptsächlich der Präsentation von Daten, und die Logik wird in JavaBeans und benutzerdefinierte Aktionen („*Tags*“) ausgelagert.

Diese Trennung ist bei größeren Projekten unerläßlich (siehe auch 3.1), da die Präsentation durch andere Personen (evtl. keine Softwareentwickler) als die Logik kreiert wird.

Die JSP-Technologie ist eine Erweiterung der Servlet-Technologie und bietet neue Möglichkeiten, Anwendungen zu entwerfen.

2 Technologie

In diesem Abschnitt werden die beiden Web-Komponenten-Technologien der J2EE behandelt, erst eine kurze Einführung in verschiedene Begriffe und die „Umgebung“ von Servlets und JSP in 2.1; danach folgt die Beschreibung von Servlets und JSPs.

2.1 Umgebung und Begriffe

Web-Komponenten werden in den J2EE-Blueprints [2] als Software-Einheiten, die Antworten zu Anfragen liefern, beschrieben. Wie auch andere Komponenten laufen Web-Komponenten, d. h. Servlets und JSPs, in speziellen Containern, in diesem Zusammenhang als Web-Container bezeichnet.

Container Allgemein ist ein Container ([2] Kapitel 2.1) eine Laufzeitumgebung für Programme. Er stellt verschiedene Dienste zur Verfügung. Neben den Standard-Containerdiensten muß ein Web-Container auch Netzwerkdienste für die Versendung von Anfragen und Antworten anbieten, sowie Zugriff auf die anderen J2EE Dienst- und Kommunikations-APIs; weiterhin noch Funktionalität zum Übersetzen einer JSP in ein Servlet (siehe 2.3).

Ein Web-Container muß mindestens das HTTP Protokoll unterstützen, optional kann er weitere Protokolle, wie zum Beispiel HTTPS (HTTP over SSL, [6] und [9]) um sichere Verbindungen zu ermöglichen, unterstützen.

Web-Applikationen Als Web-Applikation wird in [2] eine Sammlung von HTML/XML-Dokumenten, Web-Komponenten und anderen Resourcen, die meist in sogenannten WAR-Dateien (*W*eb *AR*chives) gespeichert sind, bezeichnet. Eine WAR-Datei ist eine gewöhnliche JAR-Datei [10], die außerdem noch einen Web-Deployment-Deskriptor enthält. Dies ist ein XML-Dokument, das verschiedene Abhängigkeiten und Verhaltensweisen der Web-Applikation beschreibt ([2] Kapitel 7.3).

Enterprise JavaBeans (EJBs, [2] Kapitel 2.1.3), eine andere Komponententechnologie der J2EE, werden in eigene Archive gepackt. Eine „vollständige“ Applikation besteht dann aus mehreren Web-Applikationen und evtl. EJB-Modulen.

2.2 Servlets

Ein Servlet ist eine Java-Klasse, die das `javax.servlet.Servlet`[1]-Interface der Servlet-API [11] implementiert. Dieses Interface definiert Methoden, die vom umgebenden Web-Container aufgerufen werden, um mit dem Servlet zu kommunizieren. Die wesentlichen Methoden sind `init()`, `service()` und `destroy()` (siehe Abschnitt 2.2.3 und Abbildung 2).

Die Servlet-API enthält zwei Klassen, die das `Servlet`-Interface implementieren: `GenericServlet` und `HttpServlet`. `GenericServlet` ist eine abstrakte Klasse, eine Subklasse muß hier mindestens noch die `service()`-Methode implementieren, die bei Anfragen vom Web-Container aufgerufen wird. `HttpServlet` ist eine speziell auf das HTTP zugeschnittene Subklasse des `GenericServlet`. Hier sollten Subklassen nicht die `service()`-Methode überschreiben, sondern die verschieden `doXXX`-Methoden, von denen eine für jeden HTTP-Anfragetyp existiert, beispielsweise `doGet()` oder `doPost()`.

2.2.1 Aussehen und Funktionsweise eines Servlets

Bei einer Anfrage an das Servlet wird vom Web-Container die `service()`-Methode aufgerufen, und ein `ServletRequest` Objekt übergeben. Dieses Objekt enthält die Anfrage des Benutzers, auf die das Servlet dann mittels Methoden wie `getAttribute()` und `getParameter()` zugreifen kann.

Das `ServletRequest` Objekt erlaubt auch Zugriff auf verschiedene Meta-Informationen wie Länge, Zeichenkodierung, Typ des Inhalts (MIME-Typ [12]), Sprache und Land (Locale [13]) und Zugriff auf Informationen über den Anfragesteller, beispielsweise IP-Adresse. ?

Die Antwort schreibt das Servlet dann in das ebenfalls übergebene `ServletResponse`-Objekt. Ähnlich dem Anfrageobjekt lassen sich auch hier MIME-Typ und Locale angeben.

Gleichzeitige Anfragen werden nebenläufig bearbeitet. Wenn nötig, können diese synchronisiert werden. Weiterhin kann ein Servlet eine Anfrage weiterleiten an ein anderes Servlet, um beispielsweise eine Lastverteilung auf ?

[1] Zur besseren Lesbarkeit wird im folgenden auf die Angabe des voll qualifizierten Klassennamens verzichtet, falls die Klasse entweder im `javax.servlet`- oder `javax.servlet.http`-Package liegt

mehrere Server vorzunehmen. HTTP-Spezifische Servlets (Subklassen von `HttpServlet`) unterstützen auch Sitzungsverwaltung (siehe 2.2.4)

2.2.2 Beispielprogramm

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

public class MyServlet extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse resp)
                            throws ServletException, IOException {
        HttpSession ses = req.getSession();
        RequestDispatcher disp = req.getRequestDispatcher();

        if (ses.getAttribute("name")==null) {
            disp.forward(resp.encodeURL("/login");
        }
        else {
            // ...
            resp.getWriter().println("Hallo "+ses.getAttribute("name"));
            // ...
            disp.include("/template/banner.html");
        }
    }
}
```

Abbildung 1: Ein Beispiel für ein einfaches Servlet

Das Servlet im Beispiel erbt von `HttpServlet`, wodurch ein `HttpSession` Objekt verfügbar wird. Wenn in dem Session-Objekt noch kein Attribut mit Namen „name“ vorhanden ist, wird die Anfrage mittels des `RequestDispatchers` weitergeleitet an ein anderes Servlet. Wenn sich der Benutzer schon angemeldet hat, d. h. seinen Namen angegeben hat, wird der Hauptteil des Servlets ausgeführt, der in diesem Beispiel nur eine Begrüßungsformel ausgibt und ein Banner in die Ausgabe einfügt.

2.2.3 Lebenszyklus

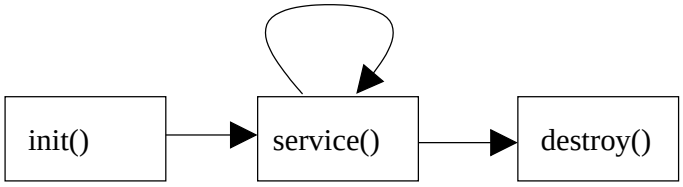

Abbildung 2: Lebenszyklus eines Servlets

Der Lebenszyklus von Servlets kann in drei Phasen unterteilt werden: Initalisierung, Anfragebearbeitung, und Beendigung.

Servlets werden dynamisch geladen, mittels der in Java benutzten Klassen-Lademechanismen. Das erfolgt dynamisch zur Laufzeit, wenn das Servlet zum ersten mal gebraucht wird, allerdings steht es dem Hersteller des Web-Containers frei, auch andere Möglichkeiten anzubieten, z. B. Laden beim Start des Servers. Nach dem Laden bestimmen im Wesentlichen drei Methoden den Lebenszyklus des Servlets, `init()` zur Initalisierung, `service()` zur Anfragebearbeitung und `destroy()` zur Beendigung des Dienstes des Servlets. All diese Methoden werden vom Web-Container aufgerufen.

Die genannten Methoden im Detail:

- `init()`: Diese Methode wird genau einmal bei Aktivierung des Servlets vom Web-Container aufgerufen. Nach dem Aufruf dieser Methode kann das Servlet Anfragen beantworten.

  Vergleichsweise teure Aktionen, wie der Verbindungsaufbau zu einer Datenbank, sollten hier stattfinden, da sie sonst bei jeder Anfrage neu ausgeführt werden müßten. Es muß abgewägt werden, ob es besser ist, eine Resource möglicherweise für die komplette Lebenszeit des Servlets anzufordern und evtl. für andere zu blockieren, oder eine Resource bei Bedarf anzufordern und danach wieder freizugeben.

- `service()`: Bei jeder Anfrage wird diese Methode aufgerufen. Hier geschieht die eigentliche Bearbeitung der Anfrage.

- `destroy()`: Nach dem Aufruf von `destroy()` wird die `service()`-Methode nicht mehr aufgerufen. Hauptsächlich werden hier Resourcen,

die in `init()` angefordert wurden wieder freigegeben. Auch persistente Daten, die vom Servlet verändert wurden, müssen spätestens jetzt abgeglichen und aktualisiert werden.

2.2.4 HTTP-Spezifische Servlets

Da Servlets meist mit dem HTTP eingesetzt werden, definiert das Servlet-API eine `HttpServlet`-Klasse sowie weiterer „HTTP-Klassen“, z. B. `HttpServletRequest` und `HttpServletResponse` in dem `javax.servlet.http`-Package, um dem Servlet-Entwickler HTTP-spezifische Funktionen zur Verfügung zu stellen. Mit HTTP-Servlets ist auch eine Sitzungsverwaltung möglich. Die speziellen Eigenschaften eines `HttpServlet`s werden im folgenden aufgezählt.

Cookies Cookies [15] sind eine Möglichkeit, mit HTTP einen Zustand zwischen Anfragen clientseitig zu speichern. Mit HTTP-Servlets ist es möglich, in der Antwort Cookies zu setzen und in der Anfrage die Cookies auszulesen. Um Zustandsinformationen zu speichern bietet das Servlet-API jedoch bessere Möglichkeiten, nämlich die Sitzungsverwaltung.

Sitzungsverwaltung Bei komplexeren Web-Applikationen wird es nötig, das der Benutzer mehrere aufeinander folgende Anfragen stellt, und der Zustand (z. B. ein Einkaufskorb in einem virtuellen Kaufhaus) gespeichert werden muß.

Dafür gibt es die `HttpSession`. Jedem Benutzer wird ein `HttpSession`-Objekt zugeordnet, das serverseitig gespeichert ist. In dieses Objekt können mittels `setAttribute()` und `getAttribute()` beliebige Java-Objekte abgelegt werden.

Weiterhin kann ein Timeout-Wert angegeben werden, damit beispielsweise nach fünf Minuten Inaktivität die Sitzung invalidiert wird.

Um eine `HttpSession` einem Benutzer zuzuordnen, hat jede `HttpSession` eine eindeutige Nummer, die mittels Cookies oder Umschreiben der URL[2] bei dem Benutzer „gespeichert“ wird.

[2] Aus „http://xy.com/page.html“ wird dann „http://xy.com/page.html?sessionid“

2.2.5 Kommunikation mit anderen Komponenten und der Umgebung

Es gibt verschiedene Formen der Kommunikation für ein Servlet. Die Möglichkeiten reichen von der Anfrageweiterleitung und Ausgabeeinbettung anderer Servlets über den Zugriff auf gemeinsame Daten in einer Web-Applikation bis hin zur Kommunikation mit der Umgebung außerhalb der Web-Applikation.

RequestDispatcher Ein Servlet kann eine Anfrage an ein anderes Servlet (oder eine JSP) weiterleiten und wahlweise dessen Ausgabe in die eigene Ausgabe einbetten, oder auch die Ausgabe komplett dem anderen Servlet überlassen. Dazu gibt es den `RequestDispatcher` mit den Methoden `include()` und `forward()`.

ServletContext Über den `ServletContext`, der für jede Web-Applikation, die aus mehreren Servlets bestehen kann, existiert, können die Servlets dieser Applikation Daten untereinander austauschen. Siehe auch 2.2.6.

J2EE-Dienst- und Kommunikationstechnologien Darüberhinaus kann ein Servlet mit Hilfe der verschiedenen von J2EE zur Verfügung gestellten Dienst- und Kommunikationstechnologien ([2] Kapitel 2.4 und 2.5) in Kontakt treten mit EJBs, Datenbanken und anderen Resourcen. Näheres hierzu ist auf der J2EE-Webseite [1] zu finden.

2.2.6 Verschiedene Zugriffsebenen

Bei Servlets lassen sich vier Zugriffsebenen unterscheiden:

- **Applikations-Ebene („*Application Scope*“):** Alle Servlets und JSPs der Web-Applikation haben Zugriff auf diesen Ebene. Diese Ebene wird durch das `ServletContext`-Objekt repräsentiert. Hier können globale Daten abgelegt werden, und über diese Ebene können verschiedene Servlets untereinander kommunizieren. Dies gilt nur, wenn die

Web-Applikation nicht verteilt in mehreren JVMs läuft. Bei einer verteilten Applikation muß man um eine solche Zugriffsebene zu realisieren auf externe Resourcen wie Datenbanken zurückgreifen.

- **Anfrage-Ebene („*Request Scope*“):** Daten dieser Ebene sind nur gültig für die aktuelle Anfrage. Das jeweilige `ServletRequest`-Objekt steht für diese Ebene, auch lokale Variablen kann man hier aufführen.
- **Sitzungs-Ebene („*Session Scope*“):** In dem `HttpSession`-Objekt werden sitzungsrelevante Daten gespeichert, für jede Sitzung existiert ein solches Objekt.
- **Seiten-Ebene („*Page Scope*[3]“):** Hiermit sind im Wesentlichen die Instanzvariablen eines Servlets gemeint. Objekte, die für mehrere Anfragen mehrfach genutzt werden, beispielsweise Zugriffszähler u. a., werden in diese Ebene eingeordnet.

2.2.7 Bemerkungen

Servlets werden einmal geladen und können dann viele Anfragen bearbeiten, und sie laufen in einem leichtgewichtigen Prozess („*Thread*“). Das macht sie im Vergleich zu CGI-Skripten [4] weitaus effizienter, da erstens das Laden des CGI-Skriptes bei *jeder* Anfrage nun entfällt, und Prozessumschaltungen bei Threads wesentlich billiger als bei Betriebssystemprozessen, in denen CGI-Skripte laufen, sind. ?

Ein großer Nachteil von Servlets ist die unkomfortable Behandlung von Ausgaben. Eine Antwort eines Servlets ist im Regelfall eine HTML-Seite, die durch Aufrufe von `print()` Methoden in einen Ausgabestream geschrieben werden muß. Es ist nicht möglich, die eigentliche Darstellung der Seite aus dem Servlet herauszulösen. Eine Lösung dieses Problems bieten die JavaServer Pages, die in 2.3 beschrieben werden.

2.3 JavaServer Pages

Mit den JSPs wird es möglich, dynamisch generierte Webseiten fast wie statische Webseiten zu schreiben, als HTML-Dateien, in die Kode zur dynami-

[3]Der Begriff kommt eigentlich von den JSPs, das Prinzip ist aber bei Servlets das Gleiche.

schen Generierung von Daten eingebettet ist.

In einer JavaServer Page sollte so wenig wie möglich Java-Kode stehen und die und die Anwendungslogik in Java-Klassen, z. B. JavaBeans (siehe 2.3.5) verpackt und ausgelagert werden. So wird eine Trennung zwischen Präsentation und Anwendungslogik erreicht.

Wie diese Dinge realisiert werden mit JSP wird im Folgenden beschrieben.

2.3.1 Elemente einer JSP

Eine JavaServer Page besteht aus verschiedenen Teilen:

- *Statischer Text*: Gewöhnlicher Text, normalerweise in HTML formatiert. Oft wird eine JSP fast nur statische Textelemente enthalten, da meist nur wenige Teile einer Seite dynamisch erzeugt werden müssen. Dieser Text wird von dem aus der JSP resultierenden Servlet bei Anfragen ohne Änderungen in den Ausgabestrom geschrieben.
- *JSP-Skriptelemente*: Die verschiedenen Skriptelemente erlauben es, Javakode in die JSP zu integrieren. Es gibt drei Arten von Skriptelementen:
  - *Ausdrücke* in der Form `<%= ausdruck %>`. Das Ergebnis des Ausdrucks wird in den Ausgabestrom geschrieben.
  - *Skripte* enthalten beliebigen Javakode, und werden mit `<% Kode %>` umschlossen. Der Kode wird ins Servlet exakt so übernommen, wie im Skript angegeben, und es müssen keine vollständigen Javaanweisungen sein.
  - *Deklarationen* erlauben es, Methoden und Felder zu definieren, die in dem Servlet als Instanzvariablen und -methoden eingefügt werden; im Gegensatz zu Ausdrücken und Skripten, die zusammen mit dem Statischen anteil den Rumpf der `service()` Methode ausmachen. Deklarationen stehen zwischen `<%!` und `%>`
- *JSP-Direktiven*: Mit Direktiven kann man das Gesamtverhalten von JSPs steuern, beispielsweise können hier mit `<%@ page import="package1.class1" %>` Klassen importiert werden. Weiterhin kann man Fehlerseiten (bei Fehlern wird die

Fehlerseite aufgerufen, und kann dann eine Fehlermeldung ausgeben) angeben, andere Textdateien einfügen (ähnlich dem `#include` in C) oder MIME-Typ der Ausgabe festlegen.

- *Aktionen („Tags“)*: Diese haben eine XML Syntax und haben hauptsächlich folgende Funktionen: Dynamisches Einfügen von Dateien zur Ausführungszeit mit dem `jsp:include` Tag, und Komfortabler Zugriff auf JavaBeans, weiterhin kann man mit `jsp:forward` die Anfrage an eine andere Seite weiterleiten. Weiter unten wird noch näher auf JSP Aktionen eingegangen, im Abschnitt 2.3.4 und 2.3.5 Im Beispiel wird die `jsp:useBean`-Aktion benutzt, um ein JavaBean zu instanziieren.

- *Implizite Objekte*: In jeder JSP gibt es einige implizit vorhandene Objekte, wie `session`, `request` und `response`, die den korrespondierenden Servlet Objekten entsprechen.

2.3.2 Beispiel einer JSP

```
<HTML>
<%@ page language="java" imports="com.wombat.JSP.*" %>
<jsp:useBean id="clock"
        class="calendar.jspCalendar" scope="session" />

<H1>Welcome</H1>
<P>Today is </P>
<UL>
    <LI>Day: <%= clock.getDayOfMonth() %>
    <LI>Year: <%= clock.getYear() %>
</UL>
<%  Calendar c = Calendar.getInstance();
    if (c.get(Calendar.AM_PM) == Calendar.AM) { %>
        Good Morning
<% } else { %>
        Good Afternoon
<% } %>
<%@ include file="copyright.html" %>
</HTML>
```

Abbildung 3: Beispiel für eine JSP

In diesem Beispiel sieht man die „Vermischung“ von HTML und Javakode. Durch einfache Java-Methodenaufrufe wird das aktuelle Datum eingefügt, und je nach Tageszeit wird eine entsprechende Begrüßungsformel ausgewählt. Weiter zu erwähnen sind die zwei JSP-Direktiven und die `jsp:useBean`-Aktion.

2.3.3 XML statt HTML

Normalerweise erzeugt eine JSP eine HTML Ausgabe, man kann aber auch problemlos XML [8] erzeugen oder sogar die komplette JSP in eine korrekte XML Datei umwandeln, so das z.B. XML Werkzeuge damit umgehen können. Jede Direktive und jedes Element hat ein XML-Äquivalent, so kann man `<% } else { %>` zu `<jsp:scriptlet> } else { </jsp:scriptlet>` umformen; für die statischen Teile muß man ein `CDATA` Element erzeugen.

2.3.4 Benutzerdefinierte Aktionen

Es gibt bei den JavaServer Pages die Möglichkeit, benutzerdefinierte Aktionen (Tags) zu kreieren, die man dann über `präfix:tag` ansprechen kann. Dies ist ein gutes Mittel, um die Präsentation von der Logik zu trennen; die Entwickler kapseln die Logik, z.B. Datenbankzugriffe oder Iterationen, in Tags, und Web-Designer brauchen nur mit der ihnen vertrauten Technologie der Tags zu arbeiten.

2.3.5 JavaBeans

JavaBeans werden von SUN als „The Only Component Architecture for Java Technology“ bezeichnet [14].

JavaServer Pages unterstützen JavaBeans in besonderem Maße durch Aktionen. Mit der `jsp:useBean`-Aktion wird eine neue JavaBean in der angegebenen Zugriffsebene instanziiert beziehungsweise das schon vorhandene Bean-Objekt in der Seite bekanntgemacht.

Um die Arbeitsweise der Aktionen zu verdeutlichen, sei hier ein Beispiel aufgeführt: `<jsp:useBean id="clock" class="calendar.jspCalendar" scope="application" />` entspricht folgendem Java-Kodefragment:

```
<%
    calendar.jspCalendar clock =
            (calendar.jspCalendar)session.getAttribute("clock");
    if (clock == null) {
        clock = new calendar.jspCalendar();
        session.setAttribute("clock", clock);
    }
%>
```

Weiterhin sind noch die die `jsp:getProperty` und `jsp:setProperty`-Aktionen zu nennen, mit denen lesend und schreibend auf die Eigenschaften („*Properties*“) einer JavaBean zugegriffen werden kann.

JavaBeans sind neben den benutzerdefinierten Aktionen eine weitere Möglichkeit, die Applikationslogik von der Präsentation zu trennen.

2.3.6 Zugriffsebenen

Die Zugriffsebenen entsprechen den in 2.2.6 beschriebenen, der Zugriff erfolgt allerdings über implizite Objekte wie `request` und `session`.

2.4 Sicherheit

Sicherheit ist in Web-Applikationen ein sehr wichtiger Punkt, und die J2EE-Plattform nimmt sich dieses Problems auch an. Es gibt die Möglichkeit, in einem Deployement Deskriptor einer Web-Applikation eine Authentifizierungsmethode festzulegen, spezifiziert sind *HTTP Basic Authentication*, *HTTP Form Based Authentication*, und HTTPS Mutual Authentication.

- Basic Authenication: Der Web-Client (Browser) erfragt vom Benutzer Name und Password, und sendet dies an die Web-Komponente. Diese Form der Authentifizierung ist unsicher.
- Form Based Authenication: Hier kann eine spezielle Seite im Deployment Deskriptor angegeben werden, die als Login-Seite benutzt wird. So kann das User-Interface angepasst werden. Auch hier wird das Password unverschlüsselt übertragen, und das macht diese Methode unsicher.

- Mutual Authenication: Client und Server benutzen X.509-Zertifikate um ihre Identität festzustellen. Diese Authentifizierungsmethode ist sicher und läuft über HTTPS.

Es sind auch Mischformen aus Formularbasierter und Gegenseitiger Authentifizierung möglich. Wenn eine der genannten Authentifizierungsmethoden benutzt wird, wird bei einer Anfrage erst geprüft, ob sich der Anfragesteller schon ausgewiesen hat. Ist dies nicht der Fall, wird die Anfrage nicht bearbeitet, bis er sich authentifiziert hat.

2.5 Vergleich von Servlets und JSP

Meist ist der Anteil der dynamischen Inhalte einer Webseite relativ gering, und die Logik, um diese zu Erzeugen recht einfach; in diesem Fall haben JSPs gegenüber Servlets den großen Vorteil das man die dynamischen Inhalte quasi in ein statisches Dokument „einbetten“ kann, ohne an Übersichtlichkeit zu verlieren. Servlets eignen sich eher zur Erzeugung von Binärdaten, oder in Fällen, in denen keine HTML Ausgabe gemacht wird, z.B. bei einer direkten Kommunikation mit einem Applet. In komplexeren Anwendungen wird man eventuell beide Arten von Web-Komponenten benutzen, mehr dazu im dritten Abschnitt .

3 Architekturmodelle

Je nach Art und Komplexität eines Problems gibt es verschiedene Modelle zur Lösung. Das Spektrum reicht von der Benutzung einfacher HTML Seiten ohne jede Dynamik bis hin zum Einsatz von JSP, Servlets und EJBs, die zusammenarbeiten.

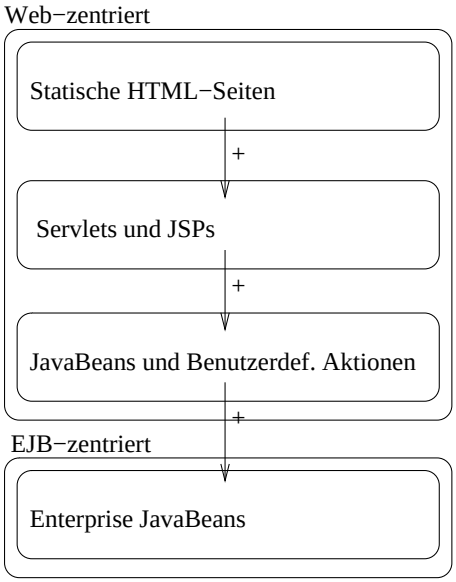

Abbildung 4: Verschiedene Entwurfsmöglichkeiten für Applikationen

Es lassen sich zwei grundlegende Modelle unterscheiden:

1. Das Web-zentrierte Modell. Die Applikation besteht nur aus Web-Komponenten und JavaBeans.

2. Das EJB-zentrierte Modell. Die Geschäftslogik und das Datenmodell werden hier durch EJBs abgebildet, auf die die Web-Komponenten zugreifen.

Die Entscheidung, welches Modell benutzt werden soll, hängt von der Komplexität des Problems ab. Es macht keineswegs immer Sinn, Enterprise JavaBeans zu benutzen.

3.1 Das MVC-Pattern

3.1.1 Beschreibung

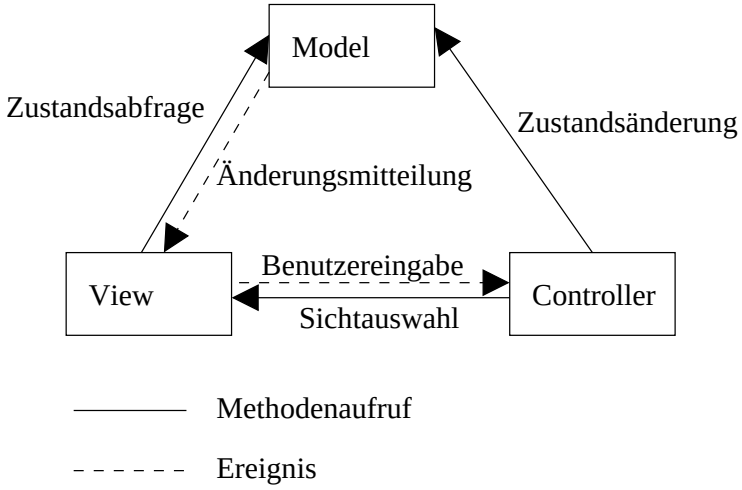

Abbildung 5: Das MVC-Pattern

Allen hier vorgestellten Architekturmodellen liegt das *Model-View-Controller* Pattern zugrunde, welches eine Anwendung in drei Teile partitioniert ([2] Kapitel 1.3.5):

- Modell: Das Modell repräsentiert die Anwendungsdaten und die Geschäftslogik, die auf diesen Daten arbeitet. Oft dient das Modell als Annäherung an einen Prozess der realen Welt. Es informiert die Sichten über Statusänderungen oder bietet Möglichkeiten für die Sichten, den aktuellen Zustand abzufragen.
- Sicht (View): Eine Sicht zeigt den Inhalt des Modells an, möglicherweise nur einen Teil des Gesamtmodells.
- Steuerung (Controller): Der Controller dient als Vermittler zwischen Benutzer und dem Modell. Er übersetzt Benutzereingaben in Aktionen, die vom Modell auszuführen sind. Darüberhinaus wählt er die anzuzeigende Sicht aus.

3.1.2 Anwendung

Die verschiedenen Aufgaben, die im MVC-Pattern definiert werden, sind je nach Modell auf unterschiedliche Komponenten verteilt, bei sehr einfachen Anwendungen wird die Dreiteilung auch nicht immer vollständig vorgenommen.

Es ist üblich, für ein Modell mehrere Sichten und Controller zu haben, die jeweils Zugriff auf unterschiedliche Bereiche eines Modells geben.

3.1.3 Web-zentrierte Applikationen

Die Applikation besteht hier nur aus Web-Komponenten. Im einfachsten Fall werden keine JavaBeans benutzt und die komplette Geschäfts- und Steuerungslogik steckt in den Servlets/JSPs. Solche Applikationen sind jedoch nur schwer wartbar und erweiterbar, und Webseiten (JSPs) müssen von Softwareentwicklern geändert/erstellt werden, weil zuviel Programmkode in den Webseiten steht.

Besser ist es hier, zur Steuerung der Applikation und zur Bearbeitung der Benutzereingaben ein Servlet als „Front Component“ (entspricht im MVC-Pattern dem Controller) einzurichten, und die Geschäftslogik als JavaBeans zu realisieren. Auch Datenbankzugriffe und dergleichen sollten in JavaBeans gekapselt werden. Die Anzeige der Daten wird von JSPs übernommen.

Durch die Verwendung von benutzerdefinierten Tags in JSPs wird es Web-Designern sehr vereinfacht, oder gar erst ermöglicht, die Präsentation der Daten zu übernehmen, da in den JSPs nur noch sehr wenig Logik steckt.

In vielen Fällen wird dieser Ansatz ohne Verwendung von EJBs sinnvoll sein, allerdings muß sich hier der Entwickler selbst um Dinge wie Transaktionsmanagment kümmern, und in komplexeren Anwendungen wird man bald viel Zeit auf solche Dinge verwenden, ohne sich um die eigentliche Applikation kümmern zu können. Dann macht es Sinn, über die Verwendung von EJBs nachzudenken.

In Abbildung 6 ist eine Web-zentrierte Applikation beispielhaft dargestellt. Eine Benutzeranfrage wird in folgenden Schritten abgewickelt:

1. Der Benutzer stellt eine Anfrage. Alle Anfragen werden von einer „Front Component“ entgegengenommen, die auch als Controller fungiert.

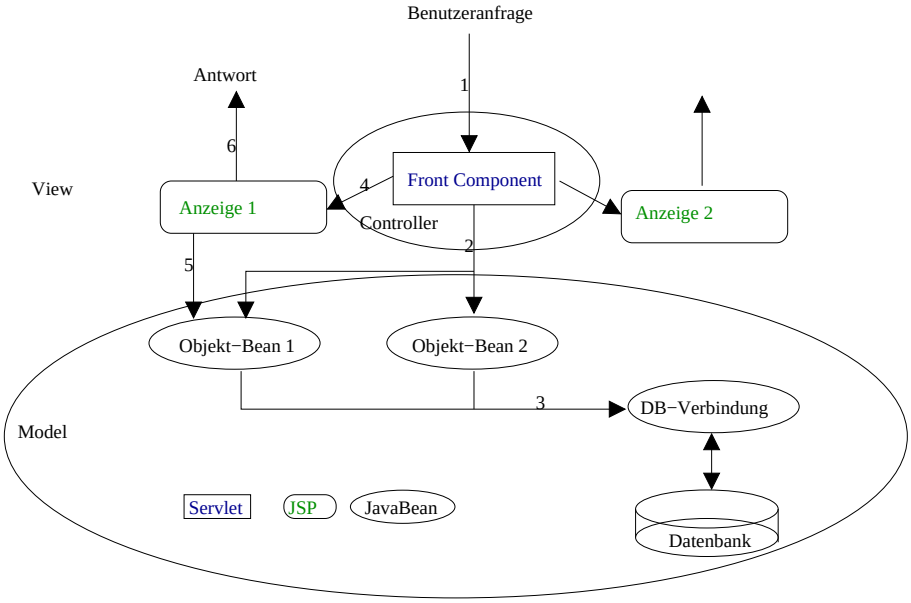

Abbildung 6: Beispiel einer Web-zentrierten Applikation

2. Die Benutzeranfrage wird vom Servlet in entsprechende Aktionen übersetzt, die dann an das Modell weitergeschickt werden. Das Modell wird hier durch die JavaBeans *Objekt-Bean 1* und *Objekt-Bean 2* repräsentiert.

3. Die JavaBeans greifen evtl. auf externe Resourcen zu, in denen die Anwendungsdaten persistent gespeichert sind. Eine JavaBean kapselt hier den Zugriff auf eine Datenbank

4. Nun wählt der Controller eine Sicht aus, die das Ergebnis der vom Benutzer angestoßenen Aktion anzeigen soll (hier die *Anzeige 1*-JSP).

5. Die Sicht holt sich vom Modell die anzuzeigenden Daten.

6. Schließlich wird die Antwort als HTML-Seite an den Benutzer geschickt.

3.1.4 EJB-zentrierte Applikationen

Die Web-Applikation stellt hier nur einen Teil der ganzen Applikation dar. Das eigentliche Daten- und Geschäftsmodell wird mit EJBs abgebildet, und die Web-Applikation übernimmt den View- und evtl. den Controller-Teil im MVC-Pattern.

Das die Web-Komponenten die Inhalte anzeigen, scheint klar, aber die Steuerung des Kontrollflusses kann sowohl in einer Web-Komponente als auch in einer Session Bean angesiedelt sein; möglicherweise wird die Steuerungskomponente auch geteilt.

Der Aufwand, eine EJB-zentrierte Applikation zu entwickeln ist höher als bei Web-zentrierten, bei komplexen Anwendungen lohnt sich das, da z.B. Transaktionen direkt von den EJBs unterstützt werden, und der Entwickler sich somit besser auf die eigentliche Applikation konzentrieren kann.

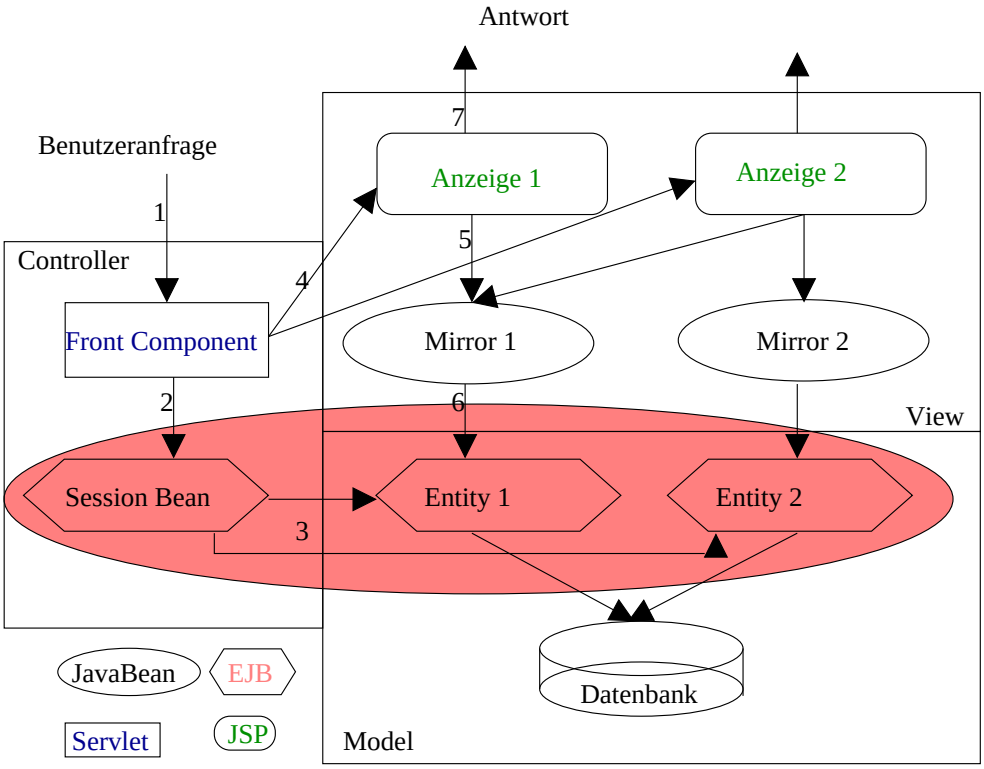

Abbildung 7: Eine EJB-zentrierte Applikation

Eine EJB-zentrierte Applikation sieht man in Abbildung 7. Eine Benutzeranfrage wird hier in folgenden Schritten abgewickelt:

1. Der Benutzer stellt eine Anfrage. Alle Anfragen werden, wie bei einer Web-zentrierten Applikation, von einer „Front Component“ entgegengenommen. Im Beispiel übernimmt die „Front Component“ aber nicht den kompletten Controller-Part.

2. Das Servlet leitet die Anfrage an eine Session-Bean weiter, die als Fassade für die EJB-Applikation dient (siehe 3.2.1). Die Session-Bean nimmt auch Controller-Funktionen wahr.

3. Nun veranlasst die Session-Bean die zuständigen Entity-Beans, die angefragten Aktionen durchzuführen.

4. Nach der Bearbeitung der Anfrage wählt Servlet eine Sicht aus, die das Ergebnis der vom Benutzer angestoßenen Aktion anzeigen soll.

5. Um die Kopplung gering zu halten und möglichst wenig Java-Kode in den JSPs zu haben, erhalten die Sichten (JSPs) die Daten von JavaBeans

6. Die JavaBeans (*Mirror 1* und *Mirror 2*) spiegeln die Daten der EJBs.

7. Schließlich wird die Antwort als HTML-Seite an den Benutzer geschickt.

3.2 Andere Design-Patterns

Hier werden kurz nützliche Design-Patterns vorgestellt, die in Web-Applikationen oft Anwendung finden [16].

3.2.1 Session Facade

Das Session Facade Pattern ist eine Instanz des Facade Patterns. Die Web-Komponenten greifen nur über eine Session Bean auf die EJBs zu, d.h. auf das möglicherweise sehr komplexe Subsystem der EJBs wird über eine einfache Schnittstelle zugegriffen.

So können z.B. verschiedene Applikationen, die jeweils nur einen Teil des ganzen Geschäftsmodells nutzen können/dürfen, über eine zugeschnittene Schnittstelle mit den EJBs kommunizieren.

3.2.2 Page-by-Page Iterator

In manchen Applikationen gibt es sehr große Ergebnismengen auf Anfragen, beispielsweise in Online-Einkaufszentren, Suchmaschinen oder Datenbankbrowsern. Hier wird es nötig, die Ergebnismenge auf mehreren Seiten anzuzeigen, da entweder der Speicher nicht reicht, um alle Ergebnisse zu halten, oder die Übertragung der kompletten Ergebnissmenge zu lange dauern würde.

Hier bietet sich die Nutzung dieses Patterns, eine Variante des Iterator-Patterns an. Der Benutzer gibt an, wie viele Ergebnisse er auf einmal sehen will/kann, und dementsprechend werden pro Seite nur x Ergebnisse angezeigt, der Benutzer kann vorwärts- und rückwärtsblättern.

Zu beachten ist hierbei, das bei stückweiser Holung der Ergebnissmenge im Schnitt die Netzlast verringert wird, da normalerweise nicht die komplette Ergebnissmenge empfangen wird, jedoch die Anzahl der Serveranfragen steigt.

4 Zusammenfassung

Zusammenfassend wird hier ein Überblick über die Möglichkeiten, die sich durch die Benutzung der J2EE-Plattform und speziell der darin spezifizierten Web-Komponenten-Technologien erschließen gegeben.

J2EE bietet mit Servlets und JavaServer Pages zwei Technologien, die es ermöglichen, dynamische Inhalte in Webseiten zu erzeugen. Die J2EE-Technologien unterscheiden sich von anderen, beispielsweise dem CGI, durch bessere Perfomanz und Portabilität.

Zur Entwicklung von Anwendungen unterschiedlicher Komplexität existieren verschiedene J2EE-Architekturmodelle. Zu unterscheiden sind hier Web-zentrierte Modelle und EJB-zentrierte Modelle, die entweder nur auf die Web-Komponenten-Technologien basieren oder auch Enterprise JavaBeans nutzen. Grundlage aller Modelle bildet jedoch das MVC-Pattern, welches eine Applikation in Modell, Sichten und Steuerung partitioniert.

Literatur

[1] *Java 2 Platform, Enterprise Edition*, http://java.sun.com/j2ee/

[2] *Java 2 Platform, Enterprise Edition Blueprints*,
http://java.sun.com/j2ee/blueprints/index.html
Inhalt: http://java.sun.com/j2ee/blueprints/apmTOC.html

[3] *Java Servlet Whitepaper*,
http://java.sun.com/products/servlet/whitepaper.html

[4] *The Common Gateway Interface*, http://hoohoo.ncsa.uiuc.edu/cgi/

[5] *JavaServer Pages Whitepaper*,
http://java.sun.com/products/jsp/whitepaper.html

[6] *HTTP – Hypertext Transfer Protocol*, http://www.w3.org/Protocols/

[7] *HyperText Markup Language*, http://www.w3.org/MarkUp/

[8] *Extensible Markup Language*, http://www.w3.org/XML/

[9] *SSL (Secure Socket Layer) 3.0 Specification*,
http://www.netscape.com/eng/ssl3/

[10] *Java Archive (JAR) Features*,
http://java.sun.com/j2se/1.3/docs/guide/jar/index.html

[11] *J2EE API Dokumentation*,
http://java.sun.com/j2ee/tutorial/api/index.html

[12] *Multipurpose Internet Mail Extensions*,
http://www.ietf.org/rfc/rfc2045.txt

[13] http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt und
http://www.chemie.fu-berlin.de/diverse/doc/ISO_3166.html

[14] *JavaBeans – The Only Component Architecture for Java Technology*,
http://java.sun.com/products/javabeans/

[15] *Cookie Specification*,
http://portal.research.bell-labs.com/~ dmk/cookie.html

[16] E. Gamma, R. Helm, R. Johnson, J. Vlissidesr: *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995

[17] N. Kassem: *Designing Enterprise Applications with the JavaTM 2 Platform, Enterprise Edition (Blueprints)*, ISBN 0-20-1702770, http://java.sun.com/j2ee/blueprints/aboutthebook.html.

[18] S. Allamaraju et. al.: *Professional Java Server Programming J2EE Edition*, Wrox Press Ltd., Birmingham, UK, 2000

[19] E. Roman: *Mastering Enterprise Java Beans, and the Java 2 Platform, Enterprise Edition*, Wiley Computer Publishing, New York, 1999.