



**TECHNISCHE
UNIVERSITÄT
WIEN**

**VIENNA
UNIVERSITY OF
TECHNOLOGY**

Web Engineering

Datenbankprogrammierung Teil 3 (Web- DB und Servlet/JSP)

Studienbrief DBPR3

Version 1.2 (Juli 2005)

Jürgen Falb

Inhaltsverzeichnis

0.	Übersicht.....	4
0.1.	Lehrziele.....	4
0.2.	Lehrstoff.....	4
0.3.	Aufgaben, Übungen	4
0.4.	Voraussetzungen	4
0.5.	Literatur.....	5
1.	Java Servlets	6
1.1.	Lebenszyklus eines Java Servlets	6
1.2.	Gemeinsame Nutzung von Information	7
1.3.	Schreiben einer Service Methode	8
1.3.1.	Extrahierung von Informationen aus einem Request	8
1.3.2.	Erstellen einer Antwort	9
1.4.	Zustandsverwaltung in einem Servlet	10
1.4.1.	Zugriff auf eine Session	11
1.4.2.	Zuordnung von Attributen zu einer Session.....	11
1.4.3.	Session Verwaltung.....	11
2.	Java Server Pages (JSP).....	13
2.1.	JSP Architecture.....	13
2.2.	JSP Syntax Grundlagen.....	15
2.2.1.	Direktiven.....	15
2.2.2.	Deklarationen	16
2.2.3.	Ausdrücke.....	16
2.2.4.	Scriptlets.....	17
2.2.5.	Kommentare	17
2.3.	Gültigkeitsbereiche von Objekten.....	17
2.4.	Implizite JSP Objekte.....	18
2.5.	Fehlerbehandlung.....	19

2.6.	Einbindung von Anfragen.....	19
2.7.	Übungsumgebung	20
3.	Lehrzielorientierte Fragen.....	23

0. Übersicht

0.1. Lehrziele

Das primäre Lehrziel der Studieneinheit **Datenbankprogrammierung Teil 3 (Web-DB und Servlet/JSP)** liegt in der Vertiefung von Java Servlets und Java Server Pages, sowie in der Vorbereitung auf die praktische Projektarbeit.

0.2. Lehrstoff

Der Studienbrief gibt einen wiederholenden und vertiefenden Einblick in Java Servlets und JSP. Der Inhalt des Studienbriefes dient auch als Nachschlagewerk für die Projektrealisierung.

Abschnitte, die mit einem (*) gekennzeichnet sind, dienen der freiwilligen Stoffergänzung und werden nicht geprüft.

0.3. Aufgaben, Übungen

Der letzte Abschnitt enthält eine Sammlung lehrzielorientierter Fragen zur Selbstüberprüfung. Die Beantwortung sollte nach Durcharbeiten des Studienbriefes möglich sein. Treten bei einer Frage Probleme auf, so versuchen Sie diese mit ihren Studienkollegen zu lösen. Ist das nicht möglich, können Sie mich per eMail direkt kontaktieren oder Sie stellen die entsprechende Frage in der nächsten Übungsstunde.

0.4. Voraussetzungen

Der vorliegende Kurs setzt Kenntnisse des Software-Engineering und der Netzwerkdienste voraus, sowie Objektorientierte Methoden.

Die Studieneinheit **Datenbankprogrammierung Teil 3 (Web-DB und Servlet/JSP)** baut auf den Studieneinheiten DBPR1 und DBPR2 auf.

0.5. Literatur

- [1] Heuer, A.; Saake, G.: „Datenbanken: Konzepte und Sprachen“, International Thomson Publishing, Bonn, 2.Auflage, 2000.
- [2] Heuer, A.; Saake, G.; Sattler, K: „Datenbanken kompakt“, mitp, Bonn, 2001.
- [3] <http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html> J2EE 1.4 Tutorial
- [4] <http://java.sun.com/products/servlet/download.html> Java Servlet Specification and Implementations
- [5] <http://java.sun.com/products/jsp/download.html> Java Server Pages Specification and Implementations.

1. Java Servlets

Dieses Kapitel stellt vertiefende Informationen zu Java Servlets zur Verfügung. Ein Servlet ist eine Java Klasse die den Funktionsumfang eines Servers erhöht, der nach dem Request/Response Modell arbeitet. Ein typisches Beispiel hierfür ist ein Web Server. An dieser Stelle soll aber darauf hingewiesen werden, dass die Servlet Spezifikation nicht auf das HTTP Protokoll eingeschränkt ist und im Prinzip für jedes Request/Response Protokoll anwendbar ist.

Für die Anwendung in Web Servern bietet die Servlet Spezifikation allerdings spezielle Unterstützung. Weiters stellt die Servlet Spezifikation eine CGI ähnliche Schnittstelle für Java dar.

1.1. Lebenszyklus eines Java Servlets

Der Lebenszyklus eines Java Servlets wird durch den Servlet Container kontrolliert, in welchem das Servlet installiert wurde. Wenn ein Request an ein bestimmtes Servlet weitergeleitet werden soll, führt der Servlet Container die folgenden Schritte durch:

1. Wenn es noch keine Instanz des Servlets gibt,
 - a. lädt der Container die [Servlet](#) Klasse,
 - b. erstellt eine Instanz der Servlet Klasse und
 - c. initialisiert das Servlet durch Aufruf der [init\(\)](#) Methode.
2. Danach wird die [service\(\)](#) Methode des Servlets aufgerufen und ihr ein Request und ein Response Objekt übergeben.

Wird ein Servlet nicht mehr benötigt und soll es entfernt werden, so wird die [destroy\(\)](#) Methode aufgerufen.

Für [HTTP Servlets](#) gibt es zu Schritt 2 eine weitere Unterstützung. Bei der Implementierung eines HTTP Servlets muss nicht die `service()` Methode implementiert werden, sondern es werden spezielle Methoden für die einzelnen HTTP Kommandos implementiert. Zwingend implementieren muss der Programmierer die [doGet\(\)](#) und [doPost\(\)](#) Methode. Alle anderen sind optional. Der Ablauf hierzu ist in Abb. 1.1 dargestellt.

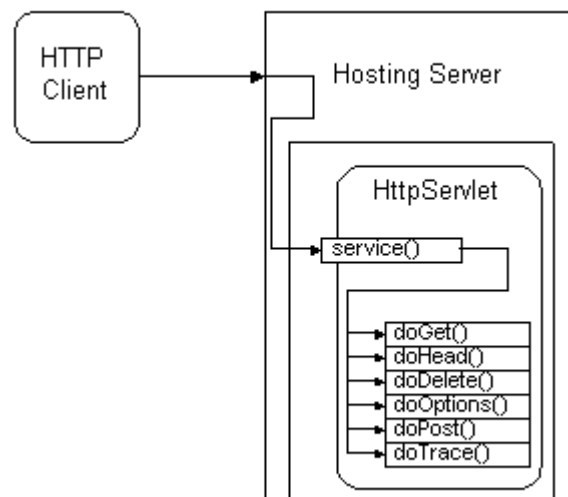


Abb. 1.1: Anfragenbehandlung bei HTTP Servlets

Innerhalb der `service()`, `doGet()` oder `doPost()` Methode wird die komplette Logik ausgeführt, die zum Erstellen der Ausgabe des Servlets notwendig ist. D.h., nach dem Abarbeiten einer dieser Methode wird die generierte Ausgabe an den Browser zurückgeliefert und die Anfrage ist fertig bearbeitet. Innerhalb dieser Methode kann jeder gültige Java Code eingebettet sein.

1.2. Gemeinsame Nutzung von Information

Um Informationen zwischen mehreren Anfragen eines Servlets, oder zwischen mehreren Servlets auszutauschen, können die Informationen (Objekte) mit einem [„Scope Object“](#) verknüpft werden, welches definiert wo eine Referenz auf das Objekt abgespeichert ist und wann Sie entfernt wird. So können zum Beispiel Objekte nur während der Abarbeitung eines Requests oder für die Zeit einer ganzen Sitzung gültig sein. Die Abb. 1.2 stellt die einzelnen Gültigkeitsbereiche dar, mit denen die Objekte verknüpft werden können.

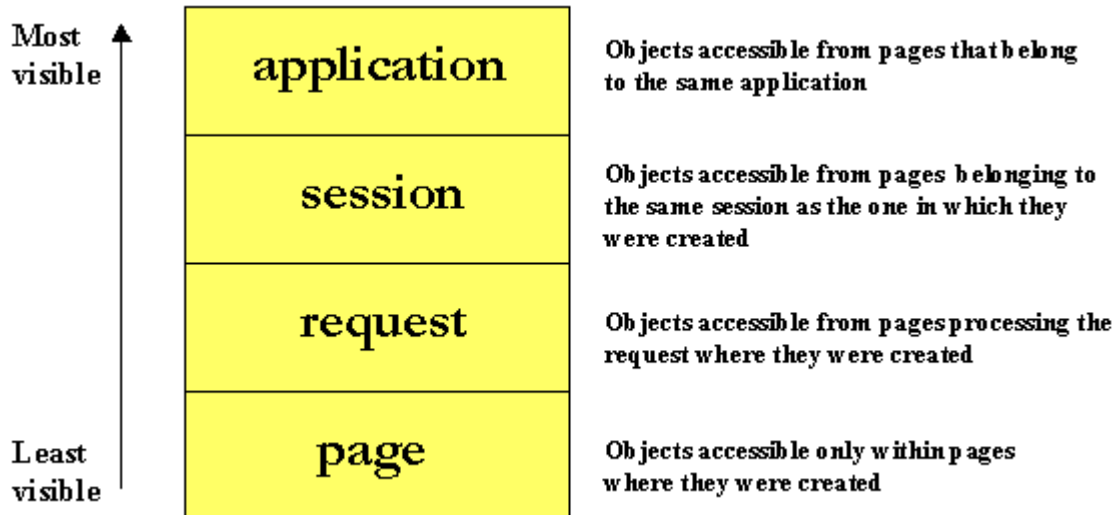


Abb. 1.2: Scope Objekte

1.3. Schreiben einer Service Methode

Als Service Methode wird eine der oben genannten Methoden (`service()`, `doGet()`, `doPost()`, ...) bezeichnet. Die prinzipielle Vorgangsweise bei der Implementierung einer solchen Methode ist wie folgt:

1. Extrahierung von Informationen aus dem Request Objekt.
2. Zugriff auf externe Ressourcen.
3. Eintragen der Ausgabe in das Response Objekt.

Bei HTTP Servlets besteht der letzte Schritt aus weiteren Teilschritten, wobei man sich zuerst einen Output Stream vom Response Objekt besorgt ([getOutputStream\(\)](#)), dann die HTTP Header setzt und zuletzt den Inhalt der Antwort in den Output Stream schreibt.

1.3.1. Extrahierung von Informationen aus einem Request

Ein Request beinhaltet Daten, die zwischen dem Client und dem Servlet ausgetauscht werden. Alle Request Objekte implementieren das [ServletRequest](#) Interface. Dieses beinhaltet die folgenden Informationen:

- Parameter, die zur Übermittlung von Information zwischen Client und Server verwendet werden (z.B. HTML Formulardaten).
- Attribute, die zum Austausch von Informationen zwischen dem Servlet Container und dem Servlet, beziehungsweise zum Austausch zwischen Servlets verwendet werden.

- Informationen über das verwendete Protokoll zwischen Client und Servlet (z.B. HTTP Headerinformationen)
- Informationen, die für die Lokalisierung (Internationalisierung) notwendig sind (z.B. verwendete Sprache, ...)

Der folgende Ausschnitt zeigt zum Beispiel die Verwendung von Parametern mit Hilfe der Methode [getParameter\(\)](#) um eine ID aus dem Request zu extrahieren:

```
String bookId = request.getParameter("Add");
if (bookId != null) {
    BookDetails book = bookDB.getBookDetails(bookId);
}
```

1.3.2. Erstellen einer Antwort

Eine Response beinhaltet alle Daten, die zwischen einem Servlet und dem Client ausgetauscht werden sollen. Jede Response implementiert das [ServletResponse](#) Interface. Das Interface implementiert Methoden für die folgenden Aufgaben:

- Beziehen eines Output Streams um Daten an den Client zu senden. Um Textdaten zu senden, steht die Methode [getWriter\(\)](#) zur Verfügung. Möchte man Binärdaten in MIME Kodierung senden, so steht die Methode [getOutputStream\(\)](#) zur Verfügung.
- Angabe des „Content Type“ (z.B. text/html) durch Verwendung der Methode [setContentType\(\)](#).
- Angabe ob die Ausgabe gepuffert werden soll oder nicht mit Hilfe der Methode [setBufferSize\(\)](#). Standardmäßig wird jede in den Output Stream geschriebene Information sofort an den Client gesendet.
- Methoden zur Definition von Lokalisierungsinformation, wie Sprache, Character Encoding u.s.w.

HTTP Response Objekte haben zusätzlich Methoden zur Bearbeitung von HTTP Header Informationen, die in dem Interface [HttpServletResponse](#) definiert sind. Unter anderem gehören hier die folgenden Aufgaben dazu:

- Setzen des Status Code einer Anfrage mit Hilfe von [setStatus\(\)](#)
- Bearbeiten von Cookies.

Das folgende Beispiel generiert eine HTML Seite, die Informationen über ein Buch ausgibt, welche das Servlet über ein BookDetail Objekt bezieht:

```
public class BookDetailsServlet extends HttpServlet {
    public void doGet (HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        // set headers before accessing the Writer
        response.setContentType("text/html");
        response.setBufferSize(8192);
        PrintWriter out = response.getWriter();

        // then write the response
        out.println("<html>" +
            "<head><title>+
            messages.getString("TitleBookDescription")
            +</title></head>");

        // Get the dispatcher; it gets the banner to the user
        RequestDispatcher dispatcher =
            getServletContext().
            getRequestDispatcher("/banner");
        if (dispatcher != null)
            dispatcher.include(request, response);

        //Get the identifier of the book to display
        String bookId = request.getParameter("bookId");
        if (bookId != null) {
            // and the information about the book
            try {
                BookDetails bd =
                    bookDB.getBookDetails(bookId);
                ...
                //Print out the information obtained
                out.println("<h2>" + bd.getTitle() + "</h2>" +
                    ...
            } catch (BookNotFoundException ex) {
                response.resetBuffer();
                throw new ServletException(ex);
            }
        }
        out.println("</body></html>");
        out.close();
    }
}
```

1.4. Zustandsverwaltung in einem Servlet

Viele Anwendungen machen es möglich, dass eine Reihe von Requests miteinander assoziiert werden. Ein Beispiel hierfür ist ein Online Shop, wo der Inhalt eines Warenkorb über mehrere Anfragen hinweg verwaltet werden muss. Um dies zu ermöglichen bietet die Servlet Technology ein API zur Verwaltung von Zuständen in einzelnen Sitzungen (Sessions).

1.4.1. Zugriff auf eine Session

Sessions werden durch ein [HttpSession](#) Objekt repräsentiert. Ein Session Objekt kann über den Aufruf der Methode [getSession\(\)](#) eines Request Objekts bezogen werden. Die Methode liefert die Session zurück, mit der der Request assoziiert ist oder erstellt eine neue Session, wenn der Request keiner Session angehört.

Ein Servlet Container kann mehrere Methoden zur Assoziierung einer Session mit einem Benutzer verwenden. Typischerweise werden dazu entweder Cookies oder URL Encoding verwendet.

Wenn eine Applikation Sessions verwendet, muss sie sicherstellen, dass auch die Session Verwaltung funktioniert, wenn der Client Cookies abdreht. Dazu muss die Applikation (Servlet) für jede URL, die es in einer HTML Seite generiert, die Methode [encodeURL\(\)](#) aufrufen, die automatisch die Session ID an die URL anfügt.

1.4.2. Zuordnung von Attributen zu einer Session

Durch Zuordnung von Attributen zu einer Session. Mithilfe von objektwertigen Attributen können beliebige Informationen in einer Session abgespeichert werden. Das folgende Beispiel zeigt, wie Objekte aus Session Attributen wieder ausgelesen werden können:

```
public class CashierServlet extends HttpServlet {
    public void doGet (HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {

        // Get the user's session and shopping cart
        HttpSession session = request.getSession();
        ShoppingCart cart =
            (ShoppingCart)session.
                getAttribute("cart");
        ...
        // Determine the total price of the user's books
        double total = cart.getTotal();
    }
}
```

1.4.3. Session Verwaltung

Da es über HTTP keine Möglichkeit gibt um eine Session zu verwalten und zu beenden, ist jede Session mit einem Timeout versehen. Läuft eine Session in das Timeout, werden alle Ressourcen freigegeben, die in Attributen abgespeichert sind. Das Timeout kann mit den Methoden [getMaxInactiveInterval\(\)](#) und [setMaxInactiveInterval\(\)](#) gesteuert werden.

Eine Session kann auch explizit durch den Aufruf der Methode [invalidate\(\)](#) gelöscht werden, wenn Sie nicht mehr benötigt wird.

2. Java Server Pages (JSP)

Dieses Kapitel stellt ergänzende Informationen zum Abschnitt Java Server Pages (8.4.2) aus dem Kompaktbuch von Heuer und Saake zur Verfügung.

2.1. JSP Architecture

Im Gegensatz zu Servlets stellen JSPs eine präsentationszentrierte Methode zur Entwicklung von Webanwendungen zur Verfügung. Während in Servlets zum Beispiel HTML Code in Quellcode eingebunden wird, wird in JSP Quellcode in HTML Code eingebunden.

Die JSP Spezifikation selbst ist definiert als eine Standarderweiterung basierend auf der Servlet API. Deshalb ist es auch nicht verwunderlich, dass JSPs und Servlets unterhalb der Oberfläche viel gemeinsam haben.

Der Einsatz von JSPs teilt sich in zwei Phasen, einer Übersetzungsphase und einer Ausführungsphase in der Anfragen behandelt werden. Hierbei wird die Übersetzungsphase nur ein einziges Mal für jede Seite ausgeführt und bei jeder Änderung der Seite selbst. Das Ergebnis der Übersetzungsphase ist eine Implementierungsklasse, die das Servlet API unterstützt, wie in Abb. 2.1 dargestellt.

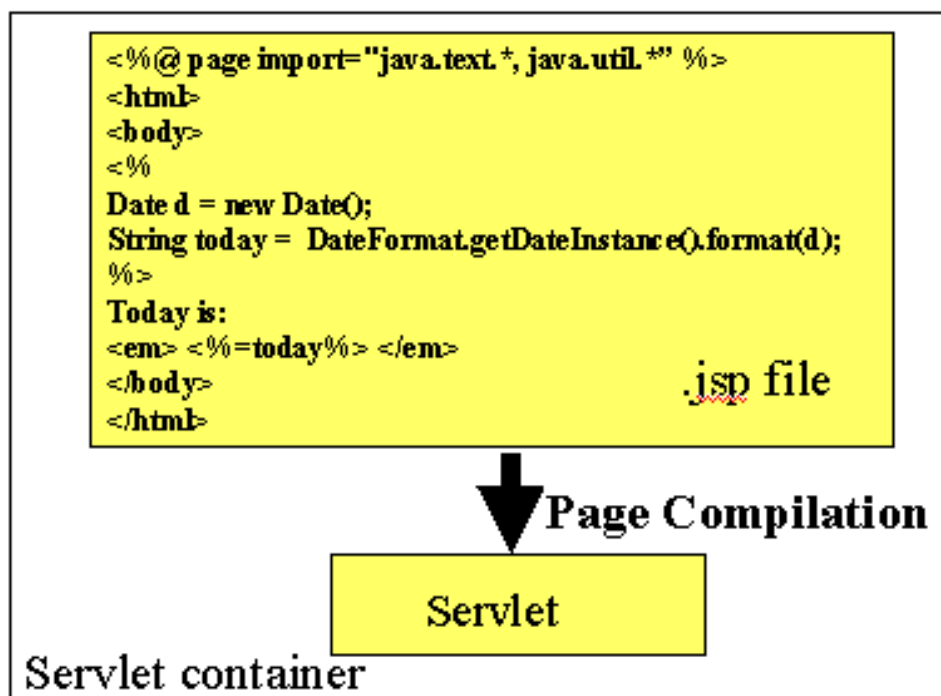


Abb. 2.1: Übersetzungsprozeß einer JSP Seite

Die Übersetzung der JSP Seite wird üblicherweise von der JSP Engine selbst durchgeführt, wenn die erste Anfrage für die Seite eintrifft. Um die Verzögerungen zu vermeiden die dadurch beim ersten Aufruf einer JSP Seite entstehen, können JSP Seiten auch vorkompiliert werden. Diese Einstellungen sind allerdings herstellerspezifisch für die JSP Engine vorzunehmen.

Da die übersetzte Klasse das Servlet API implementiert, entspricht der Lebenszyklus jenem eines Servlets. Um dem Entwickler die Möglichkeit zu geben auf den Initialisierungs- und Zerstörungsprozeß Einfluss zu nehmen, können in JSP Seiten die beiden Methoden [jspInit\(\)](#) und [jspDestroy\(\)](#) überschrieben werden. Abb. 2.2 stellt den Lebenszyklus einer JSP Seite dar.

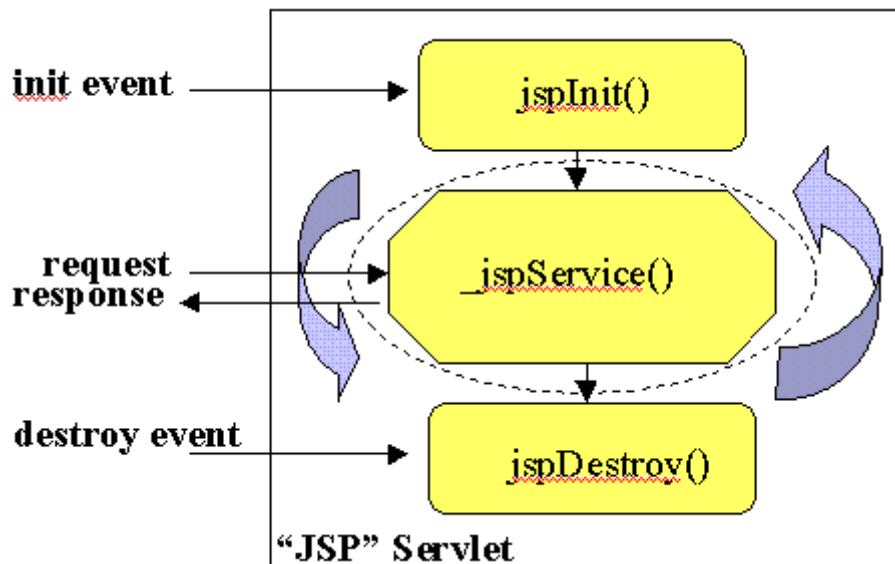


Abb. 2.2: Lebenszyklus einer JSP Seite

Ein typischer Ablauf der Ausführungsphase ist in Abb. 2.3 dargestellt. Zuerst wird eine Anfrage von einem Web Browser an einen Web Server gestellt (Schritt 1). Der Web Server leitet die Anfrage an die JSP Engine weiter. Diese ruft die JSP Seite auf, die der Anfrage zugeordnet ist. Komplexere Logik wird typischerweise in eigene Klassen (sogenannte Java Beans) ausgelagert, die von der JSP Seite aufgerufen werden können (Schritt 2). Das Java Bean führt typischerweise Abfragen an einem Informationssystem beziehungsweise einer Datenbank durch (Schritt 3). Die Ergebnisse des Java Beans können anschließend in die JSP Seite eingebaut werden. Zuletzt wird die Ausgabe der JSP Seite an den Web Browser zurückgeliefert.

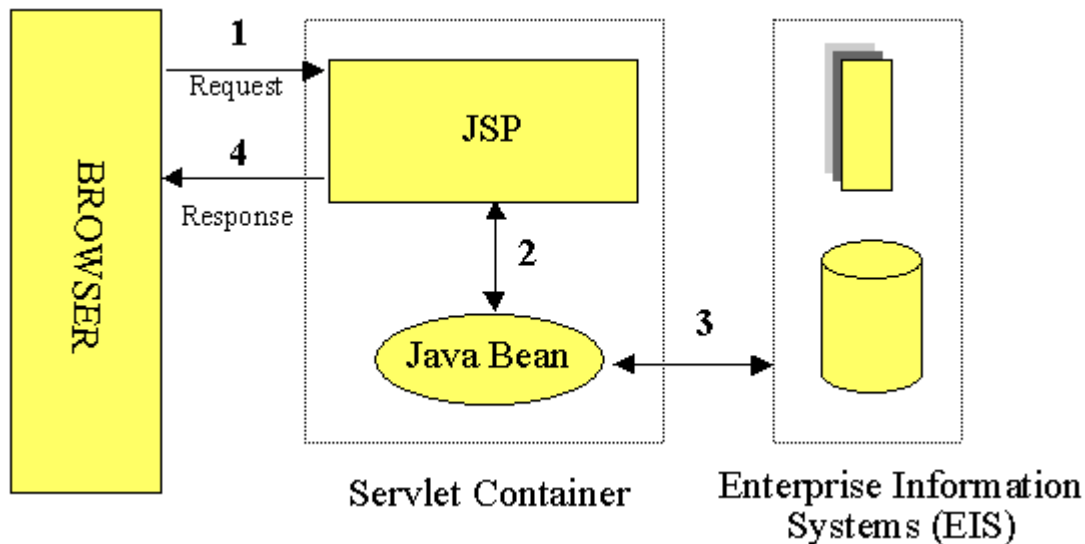


Abb. 2.3: Typischer Ablauf einer Anfrage

Die Struktur die in Abb. 2.3 dargestellt ist, wird auch als „Model 1“ bezeichnet. Weiters gibt es auch ein „Model 2“, welches auf dem Model View Controller Pattern (MVC) aufbaut und eine bessere Trennung zwischen Präsentation und Logik mit sich bringt. Das „Model 2“ soll aber an dieser Stelle nicht näher erläutert werden.

2.2. JSP Syntax Grundlagen

Die JSP Syntax ist relativ einfach und kann in Direktiven, Scripting Elemente und Aktionen eingeteilt werden. Direktiven und Scripting Elemente werden in den folgenden Abschnitten näher erläutert. Mit Aktionen sind einfache Java Befehle gemeint, die in den verschiedenen Elementen ausgeführt werden können.

2.2.1. Direktiven

JSP Direktiven sind Nachrichten an die JSP Engine. Sie produzieren keine direkten sichtbaren Ausgaben, sondern stellen Anweisungen dar, wie mit der restlichen Seite zu Verfahren ist. JSP Direktiven sind immer in ein `<%@ ... %>` Tag eingebettet. Die wichtigsten Direktiven sind die `page` und die `include` Direktive.

Page Direktive

Die [Page Direktive](#) befindet sich typischerweise am Beginn einer JSP Seite. Sie kann beliebig oft vorkommen und beinhaltet Attribut/Wert Paare. Ein typisches Attribut ist das

Importattribut zum Importieren von Java Klassen. Zum Beispiel macht die folgende Direktive

```
<%@ page import="java.util.*, com.foo.*" buffer="16k" %>
```

die Klassen in den beiden Packages verfügbar und setzt den Seitenpuffer auf 16kB.

Include Direktive

Die [Include Direktive](#) erlaubt die Unterteilung des Inhalts in mehrere kleinere Dokumente, wie zum Beispiel die Auslagerung von Kopf- und Fußzeilen. Mit der Include Direktive können diese Dokumente an jeder beliebigen Stelle in die JSP Seite eingebunden werden. Diese Dokumente können sowohl statische HTML Dokumente als auch weitere JSP Seiten sein.

```
<%@ include file="copyright.html" %>
```

Die obige Deklaration inkludiert zum Beispiel den Inhalt der Datei copyright.html an der Stelle in der JSP Seite, wo diese Direktive zu finden ist.

2.2.2. Deklarationen

JSP Deklarationen erlauben die Definition von page-level Variablen und die Definition von Methoden, die in der restlichen JSP Seite genutzt werden können. Deklarationen erlauben somit die Einbindung von beliebig viel Code. Der Nachteil ist allerdings eine schlechtere Wartbarkeit der JSP Seiten. Um logikintensive Aufgaben durchzuführen, empfiehlt es sich diese in ein JavaBean oder in eine Klasse auszulagern.

Deklarationen werden immer in das [<%! ... %>](#) Tag eingebettet. Beenden Sie Variablendeklarationen immer mit einem Strichpunkt, da der Inhalt einer Deklaration immer ein gültiges Java Statement sein muss:

```
<%! int i=0; %>
```

Das folgende Beispiel überschreibt die Initialisierungsmethode des JSP Lebenszyklus:

```
<%! public void jspInit() {  
    //some initialization code  
}  
%>
```

2.2.3. Ausdrücke

Ausdrücke in JSP erlauben die Einbindung eines Ergebnisses in die Seite. Dazu wird das Ergebnis des Ausdrucks in einen String umgewandelt und an der entsprechenden Stelle in die Ausgabeseite eingefügt. Ausdrücke werden typischerweise verwendet um einfache Werte von Variablen oder Rückgabewerte von Methodenaufrufen in die Ausgabe zu inkludieren. JSP Ausdrücke sind immer in das [<%= ... %>](#) Tag eingebettet und beinhalten keinen

abschließenden Strichpunkt. (Der Grund besteht darin, dass Ausdrücke in print() Statements in der generierten Klasse inkludiert werden.)

```
<%= fooVariable %>
<%= fooBean.getName() %>
```

2.2.4. Scriptlets

JSP Codefragmente oder Scriptlets sind immer in ein `<% ... %>` Tag eingebettet. Der eingebettete Java Code wird jedesmal ausgeführt, wenn eine Anfrage durch diese JSP Seite bearbeitet wird. Es kann jeder beliebige gültige Java Code mit beliebig vielen Zeilen eingebettet werden. Das folgende Beispiel gibt den String „Hello“ innerhalb der Tags H1, H2, H3 und H4 aus:

```
<% for (int i=1; i<=4; i++) { %>
    <H<%=i%>>Hello</H<%=i%>>
<% } %>
```

2.2.5. Kommentare

Wenn Sie [Kommentare](#) einbetten wollen, die der Benutzer nicht im Quellcode der Ausgabeseite sehen soll, können Sie JSP Kommentare verwenden, wie folgendes Beispiel zeigt:

```
<%-- comment for server side only --%>
```

Mit den JSP Kommentaren können Sie auch selektiv Blöcke von der Kompilierung ausnehmen.

2.3. Gültigkeitsbereiche von Objekten

In JSP ist es wichtig zu wissen welchen Gültigkeitsbereich Java Objekte in einer JSP Seite haben. Objekte können implizit durch JSP Direktiven, explizit durch Aktionen oder in seltenen Fällen direkt durch Scripting Code erzeugt werden. Die erzeugten Objekte können mit einem [„Scope Object“](#) verknüpft werden, welches definiert wo eine Referenz auf das Objekt abgespeichert ist und wann Sie entfernt wird. So können zum Beispiel Objekte nur während der Abarbeitung einer Seite oder für die Zeit einer Sitzung gültig sein.

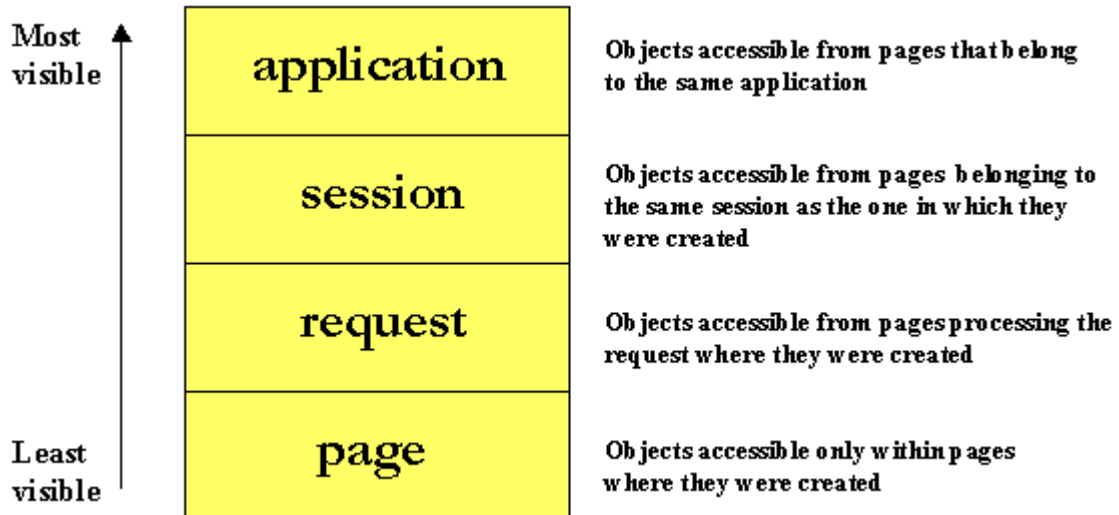


Abb. 2.4: Gültigkeitsbereiche für Objekte in JSP

Die Abb. 2.4 stellt die einzelnen Gültigkeitsbereiche dar, mit denen die Objekte verknüpft werden können.

2.4. Implizite JSP Objekte

Eine Annehmlichkeit in JSP ist die implizite Bereitstellung einiger wichtiger Objekte durch den JSP Container. Diese impliziten Objekte können in Scriplets und Ausdrücken verwendet werden ohne, dass sie vom Programmierer erst erstellt werden müssen. Diese Objekte wrappen typischerweise die Schnittstellen des darunterliegenden Servlet APIs und einiger JSP Klassen. Im Folgenden sind die Objekte beschrieben. In Klammern ist jeweils der Gültigkeitsbereich der Variablen angegeben.

- **request:** repräsentiert einen [HttpServletRequest](#) der die Abarbeitung einer JSP Seite veranlasst hat (Request Scope).
- **response:** repräsentiert die entsprechende [HttpServletResponse](#) die dem Request zugeordnet ist. Dieses Objekt wird eher selten verwendet, da die Ausgabe implizit über den Inhalt der JSP Seite an das response Objekt übergeben wird (Page Scope).
- **pageContext:** kapselt implementierungsabhängige Features einer JSP Engine in einem Objekt vom Typ [PageContext](#) (Page Scope).
- **application:** repräsentiert den [ServletContext](#) der vom Servlet Konfigurationsobjekt erhalten werden kann (Application Scope).

- **out:** ein [JspWriter](#) Objekt welches in den Output Stream schreibt (Page Scope).
- **config:** repräsentiert den [ServletConfig](#) für die JSP Seite (Page Scope).
- **page:** Synonym für den "this" Operator und ist vom Typ [HttpJspPage](#). Wird nicht oft von Programmierern verwendet (Page Scope).
- **session:** repräsentiert eine [HttpSession](#) (Session Scope).
- **exception:** beinhaltet das nicht abgefangene [Throwable](#) Objekt, welches den Auslöser für den Aufruf der Fehlerseite darstellt (Page Scope).

Bei diesen impliziten Objekten ist allerdings zu beachten, dass sie nur innerhalb der automatisch generierten `_jspService()` Methode sichtbar sind. Sie sind nicht sichtbar innerhalb von Methoden die selbst in einem Deklarationsabschnitt definiert wurden.

2.5. Fehlerbehandlung

JSP bietet einen Mechanismus zur Behandlung von Laufzeitfehlern. Obwohl man seine eigenen Fehlerbehandlungsroutinen innerhalb einer JSP Seite schreiben kann, ist es nicht immer möglich alle Fehler abzufangen. Durch Verwendung der Page Direktive `errorPage`, ist es möglich eine nicht abgefangene Fehlermeldung an eine JSP Fehlerseite weiterzuleiten.

Zum Beispiel informiert

```
<%@ page isErrorPage="false" errorPage="errorHandler.jsp" %>
```

die JSP Engine, dass sie jede nicht abgefangene Ausnahme an die Fehlerseite „errorHandler.jsp“ weiterleitet. Die Seite „errorHandler.jsp“ muss dann allerdings als Fehlerseite mit Hilfe des `isErrorPage` Attributs gekennzeichnet sein:

```
<%@ page isErrorPage="true" %>
```

Innerhalb der Fehlerseite kann dann auf das implizite `exception` Objekt zugegriffen werden und die Fehlerbehandlung durchgeführt werden.

2.6. Einbindung von Anfragen

Das [<jsp:include>](#) Tag erlaubt die Umleitung der Anfrage (Request) an eine beliebige statische oder dynamische Ressource, die sich im selben [Kontext](#) wie die aufrufende JSP Seite befindet. Die aufrufende JSP Seite kann auch der aufgerufenen JSP Seite Parameter mitübereichen, in dem Sie diese in das [Request](#) Objekt einträgt, wie in Abb. 2.5 dargestellt.

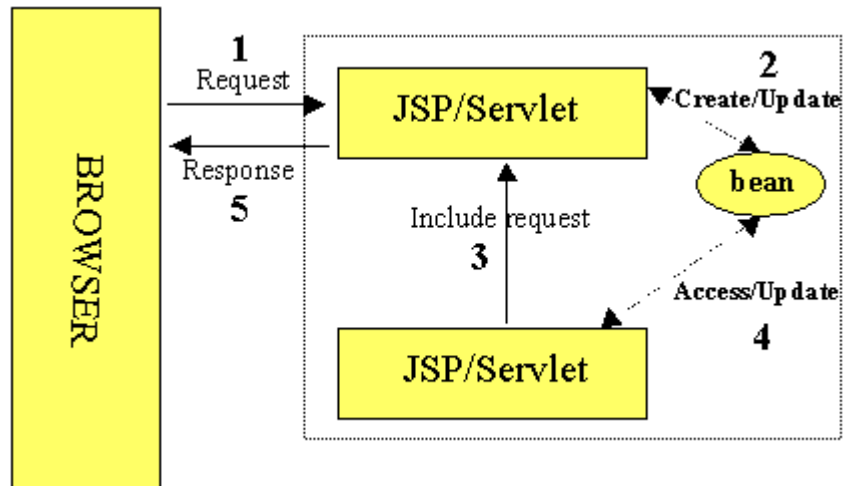


Abb. 2.5: Weiterleitung und Einbindung von Anfragen mit Parameterübergabe

Zum Beispiel erlaubt die Anweisung

```
<jsp:include page="shoppingcart.jsp" flush="true"/>
```

der Seite „shoppingcart.jsp“ nicht nur den Zugriff auf Objekte, die in das Request Objekt eingetragen wurden, sondern auch die Einbindung der generierten Ausgabe in die aufrufende Seite an jener Stelle wo sich das `<jsp:include>` Tag befindet.

Ein Nachteil ist, dass die aufgerufene Seite keine HTTP Header setzen darf, wodurch es zum Beispiel keine Cookies setzen kann.

2.7. Übungsumgebung

Die Übungsumgebung erlaubt unter dem Menüpunkt Verzeichnisverwaltung den Upload von JSP Seiten, die beim ersten Aufruf automatisch kompiliert werden. Allerdings muss in der Übungsumgebung die komplette Logik in der JSP Seite inkludiert werden, da der Upload und die Kompilierung von Java Klassen und somit von Java Servlets nicht unterstützt wird.

Im folgenden ist das Beispiel zum Buch suchen aus dem letzten Studienbrief nochmals in Form einer JSP Seite dargestellt. Sie können das Beispiel auch in der Übungsumgebung uploaden und ausprobieren.

```

<%@page contentType="text/html"%>
<%@page import="java.sql.DriverManager" %>
<%@page import="java.sql.Connection" %>
<%@page import="java.sql.PreparedStatement" %>
<%@page import="java.sql.ResultSet" %>
    
```

```

<%
String name =
request.getParameter("wName")==null?"":request.getParameter("wName");
String keywords =
request.getParameter("wKeywords")==null?"":request.getParameter("wKeywords"
);
String author =
request.getParameter("wAuthor")==null?"":request.getParameter("wAuthor");
%>

<html>
<head><title>Buch suchen</title></head>
<body>
<center><h2>Buch suchen</h2></center><br>
Bitte geben Sie die Suchparameter ein und drücken Sie "Suchen...".<br>
<h5>Hinweis beim Buchnamen reicht ein Bruchstück des Namens (ein % am
Anfang sucht den Ausdruck an irgendeiner Stelle des Titels), <br>
Keywords (z.B. RDB, ER, OODB,..) und Autor müssen exact eingeben
werden.</h5><br>

<form method="POST" action="buch_suchen.jsp">
<table border="0">
  <tr><td>Name</td>
    <td><input name="wName" value="<%= name %>"></td></tr>
  <tr><td>Keywords</td>
    <td><input name="wKeywords" value="<%= keywords %>"></td></tr>
  <tr><td>Author</td>
    <td><input name="wAuthor" value="<%= author %>"></td>
    <td><input type="submit" value="Suchen..."></td></tr>
</table>
</form><br><hr>
<%
  if (name.equals("") && author.equals("") && keywords.equals("")) {
    %>Bitte mehr Parameter eingeben!<%
  }
  else {
    Connection con = null;
    PreparedStatement stmt = null;
    ResultSet result = null;
    try {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      con = DriverManager.getConnection(
        "jdbc:oracle:thin:@195.245.225.70:1521:db",
        "dbsa040501", "dbsa040501");
      stmt = con.prepareStatement("select Titel, Buecher.ISBN"+
        "ISBN,Buecher.Verlagsname Verlagsname,verlagsort, "+
        "buch_exemplare.auflage Auflage, Jahr,Seiten,Preis, "+
        "count(*) Anzahl from Buecher,verlage,Buch_exemplare, "+
        "Buch_versionen "+
        "where Buecher.verlagsname=Verlage.Verlagsname and "+
        "Buecher.ISBN=Buch_Versionen.ISBN and "+
        "Buch_versionen.auflage=buch_exemplare.auflage and "+
        "Buecher.isbn=Buch_Exemplare.isbn "+
        "and (lower(Titel) like '"+name.toLowerCase()+"%') "+
        "and (('"+author+"' is null) or ('"+author.toLowerCase()+
        "' in (select lower(Autor) from Buch_Autor where "+
        "Buch_Autor.isbn=Buecher.isbn))) "+
        "and (('"+keywords+"' is null) or ('"+keywords.toLowerCase()+
        "' in (select lower(Stichwort) from Buch_Stichwort where "+
        "Buch_Stichwort.isbn=Buecher.isbn))) "+
        "group by Titel,Buecher.ISBN,Buecher.Verlagsname, "+

```

```

        "verlagsort,buch_exemplare.auflage,Jahr,Seiten, Preis "+
        "order by titel");
        result = stmt.executeQuery();
    %>
    <table border="1">
        <tr><td><b>Buchname</b></td>
            <td><b>Autoren</b></td><td><b>ISBN</b></td>
            <td><b>Verlag</b></td>
            <td><b>Verlagsort</b></td>
            <td><b>vorhandene<BR>Exemplare</b></td>
            <td><b>Auflage</b></td>
            <td><b>Erscheinungsjahr</b></td>
            <td><b>Seiten</b></td>
            <td><b>Preis/DM</b></td></tr>
    <%
        stmt = con.prepareStatement("select * from buch_autor where isbn = ?");
        while (result.next()) {
    %>
        <tr>
            <td valign="top"><%= result.getString("Titel") %></td>
    <%
        stmt.setString(1,result.getString("ISBN"));
        ResultSet result2 = stmt.executeQuery();
    %><td valign="top"><%
        while (result2.next()) {
            out.println(result2.getString("Autor")+"<br>"); }
        result2.close();
    %>
        </td>
            <td valign="top"><%= result.getString("ISBN") %></td>
            <td valign="top"><%= result.getString("Verlagsname") %></td>
            <td valign="top"><%= result.getString("Verlagsort") %></td>
            <td valign="top"><%= result.getString("Anzahl") %></td>
            <td valign="top"><%= result.getString("Auflage") %></td>
            <td valign="top"><%= result.getString("Jahr") %></td>
            <td valign="top"><%= result.getString("Seiten") %></td>
            <td valign="top"><%= result.getString("Preis") %></td>
        </tr>
    <%
    }
    %>
    </table>
    <%
    }
    catch(Exception e) {
        out.println("Error: "+e.getMessage());
    }
    finally {
        if (result != null) result.close();
        if (stmt != null) stmt.close();
        if (con != null) con.close();
    }
    }
    %>
</body></html>

```

3. Lehrzielorientierte Fragen

1. Was ist das Hypertext Transfer Protokoll und welche Grundeigenschaft besitzt es? Wieso ist diese Eigenschaft ein Problem für Datenbankanwendungen?
2. Welche Anforderungen ergeben sich aus der Sicht der Datenbankanwendung für die Webanbindung?
3. Welche Mängel besitzt ein User-Interface welches nur auf HTML basiert? Wie kann man diese Nachteile umgehen?
4. Erklären Sie das Prinzip des Common Gateway Interfaces.
5. Erläutern Sie den Lebenszyklus eines Java Servlets und beschreiben Sie dessen API. Gehen Sie im speziellen auf http Servlets ein.
6. Welche Techniken können Sie verwenden um von einem Servlet auf eine Datenbank zuzugreifen damit Sie die Ergebnisse in eine Webseite einbauen können? Überlegen Sie, an welchen Stellen Sie am günstigsten die Datenbankverbindung auf- und abbauen.
7. Erklären Sie den Unterschied zwischen JSP und Servlets. Welche Einbettungskonstrukte gibt es in JSP? Erklären Sie diese.
8. Welche Möglichkeiten der Zustandsrealisierung gibt es?