JAVA SERVER PAGES

In diesem Kapitel:

- Einleitung
  - *Vorteile von JSP*
  - *JSP versus ASP*
  - *JSP oder Servlets*?
- JSP Architektur
- JSP Access Modelle
- JSP Syntax Grundlagen
  - *Steuerbefehle / Directives*
    - Seitenformattierung
    - include Directives
  - *Deklarationen*
  - *Ausdrücke*
  - *Scriptlets*
  - *Kommentare*
- Gültigkeitsbereich der Objekte
- Implizite JSP Objekte
- Synchronisation
- Behandlung von Ausnahmen
- Session Management
- Standard Aktionen
  - *Einsatz von JavaBeans Komponenten*
  - *Weiterleiten von Requests*
    - Request Chaining
- Web Sites
- Dokumentation und Spezifikationen

Java Server Pages - Praxis

1.1. Einleitung

Sie können sehr unterschiedliche Techniken einsetzen, um Web Applikationen mit dynamischen Inhalten zu bauen. Hier geht es um die JavaServer Pages (JSP). Diese kann man auf unterschiedlichen Plattformen einsetzen und mit unterschiedlichen Web Servern. JSP kombinieren Server-seitige Technologie mit den GUI Möglichkeiten der HTML Seiten.

JSP Seiten umfassen typischerweise:
- statische HTML / XML Komponenten
- spezielle JSP Tags
- Snipplets - kleine in Java geschriebene Programme.

Sie können JSP Anwendungen mit den üblichen HTML / XML Tools kreieren.

Für Sie ist es wichtig zu wissen, dass die JSP Spezifikation eine Standard Erweiterung des Servlet APIs ist. Sie können also allfällige Erfahrungen mit Servlets weiterverwenden.

Allerdings gibt es signifikante Unterschiede zwischen Servlets und JSPs. Servlets setzen voraus, dass Sie gründliche Kenntnisse von Java und der Web Server Technologie besitzen.

JSP ist für ein breiteres Publikum gedacht. JSPs können auch von Web Designern eingesetzt werden. Ein weiterer Vorteil der JSPs ist die inhärente Separation der Darstellung vom Inhalt, da die Technologie auf wiederverwendbaren Komponenten aufbaut, beispielsweise JavaBeans und Enterprise Java Beans Technologie.

In dieser Kurseinheit lernen Sie diese Technologien kennen, und setzen Tomcat (http://jakarta.apache.org/tomcat/index.html), die JSP Referenzimplementation der Apache Gruppe, als Basis für unsere Beispiele ein.

Mit JSP kann man das Look & Feel von der zugrunde liegenden Businesslogik trennen, so dass dieses Web Server und Plattform-unabhängig wird.

Wir legen den Schwerpunkt auf den Einsatz von JSP für die Entwicklung dynamischer Web Seiten, wobei wir uns zwangsweise auch mit der Syntax und den Komponenten der JSP und der Entwicklungsmethodik befassen.

Sie können JSP auch mit JDBC kombinieren. Darauf gehen wir jedoch hier nicht ein.

1.1.1. Neue Konzepte

Nach dem Durcharbeiten dieses Moduls kennen Sie:
- einige der Vorteile der JSP Technologie
- die JSP Architektur
- den Lebenszyklus einer JSP Seite
- die JSP Syntax und Semantik
- die Bedeutung der JavaBean Komponenten in JSP Seiten.

1.1.2. Lernziele

Nach dem Durcharbeiten dieser Kurseinheit sollten Sie in der Lage sein:
- Session-orientierte Informationen aus JSPs zu verwalten
- zwischen JSP Seiten zu kommunizieren
- einfache Eingabeformulare zu verarbeiten.

1.1.3. Voraussetzungen

Sie sollten die grundlegenden Objekt-orientierten Programmierkonzepte und die Programmiersprache Java kennen. Falls dies nicht der Fall ist, sollten Sie diese Defizite zuerst beseitigen.

In den Übungen müssen Sie einfache Java Programme schreiben und HTML ähnliche Seitenbeschreibungen erstellen.

Hilfreich wären auch Grundkenntnisse über Web Server und Servlets. Zum Thema Servlets existiert auch eine Kurseinheit, die allerdings nicht aktualisiert wurde.

1.1.4. Vorteile von JSP

JSP haben sicher folgende Vorteile im Vergleich zu konventionellen Lösungen:

Separation des statischen vom dynamischen Content :
Falls Sie mit Servlets arbeiten, besteht die Lösung dynamischer Aufgaben für Web Pages darin, dass Sie die Dynamik in die Servlets fix einprogrammieren. Die Kopplung an die statische Darstellung für die Benutzerschnittstelle ist sehr eng. Falls Sie kleinere Änderungen vornehmen wollen, müssen Sie in der Regel das Servlet neu kompilieren. Diese enge Kopplung der Darstellung mit dem Inhalt führt zu eher starren Anwendungen. Mit JSP wird der dynamische Inhalt vom statischen Template getrennt, indem er in JavaBeans eingebettet wird. Diese werden dann mit den JSPs eingesetzt, zusammen mit speziellen Tags und Scriptlets. Änderungen am Template werden automatisch berücksichtigt.

Portabilität - Write once, run anywhere:
Mit JSPs können Sie das Versprechen von Java auch auf Web Seiten übertragen. JSPs können von einem Server auf den andern übertragen werden, ohne geändert werden zu müssen.

Dynamischer Content in unterschiedlichen Formaten:
Das statische Template, in das der JSP Code eingebettet ist, kann aus HTML / DHTML aber auch WML (für WAP) oder XML (für Business-to-Business Applikationen) bestehen.

Empfohlener Web Zugriff für die n-Tier Architektur:
Sun's J2EE Blueprint empfiehlt den Einsatz von JSP und bevorzugt diese Technologie gegenüber jener der Servlets zur Generierung der dynamischen Web Inhalte.

JSP entspricht weitestgehend dem Servlet API:
Falls Sie bereits Servlets kennen, werden Sie kaum Probleme bei der Umstellung haben. JSP entsprechen in etwa High Level Servlets. Sie können so gut wie alles, was Sie mit Servlets erledigen können, auch mit JSPs realisieren.

1.1.4.1. Vergleich von JSP mit ASPs

Auf den ersten Blick ähneln sich ASPs und JSPs (ASP = Microsoft Active Server Pages). Die Technologien unterscheiden sich jedoch wesentlich:

	JavaServer Pages	Active Server Pages
Web Server Support	Verfügbar für die gängigen Web Server: Apache, Netscape und Microsoft IIS unterstützen JSP.	Unterstützt den Microsoft IIS und den Personal Web Server. Weitere Web Server werden mittels Produkten von Drittanbietern unterstützt.
Plattform Support	Plattform unabhängig. Läuft auf allen Java-enabled Plattformen.	Unterstützt Win32.
Komponenten Modell	Basiert auf plattformunabhängigen JavaBeans, Enterprise JavaBeans und eigenen Tag Bibliotheken.	Verwendet das Win32-basierte COM Komponenten Modell.
Scripting	Gestattet ein Scripting mit Hilfe von Java oder JavaScript.	Unterstützt VBScript und JScript für das Scripting.
Security	Beachtet das Java Security Modell.	Arbeitet mit der Windows NT Security Architektur.
Datenbank Zugriff	Verwendet JDBC für den Data Access.	Verwendet Active Data Objects (ADOs) für den Data Access.
Anpassbare Tags	JSP ist mittels eigenen Tag Libraries erweiterbar .	Eigene Tag Libraries sind nicht einsetzbar.

1.1.4.2. JSP oder Servlets?

Beide Technologien, Servlets und JSPs haben vieles gemeinsam und können beide zur dynamischen Gestaltung von Wen Seiten eingesetzt werden. Daher kann man sich überlegen, ob man die eine oder die andere Technologie einsetzen kann / soll. Sun's J2EE Blueprint nimmt zu dieser Frage Stellung:

gemäss den Blueprints diesen Servlets ausschliesslich als Web Server Erweiterungen, beispielsweise zur Datenvalidierung oder Kodierung / Dekodierung.

Der "JSP Engine" besteht aus einem speziellen Servlet, welches unter Aufsicht einer Servlet Engine läuft. JSP behandelt aber lediglich Text. Sollten Sie also spezielle Datentypen oder binäre Daten benutzen müssen, sind Sie weiterhin auf Servlets angewiesen.

JSP ist sehr gut geeignet dynamische Web Seiten aufzubauen und an Stelle von Server-Site-Includes.

1.1.5. Übung

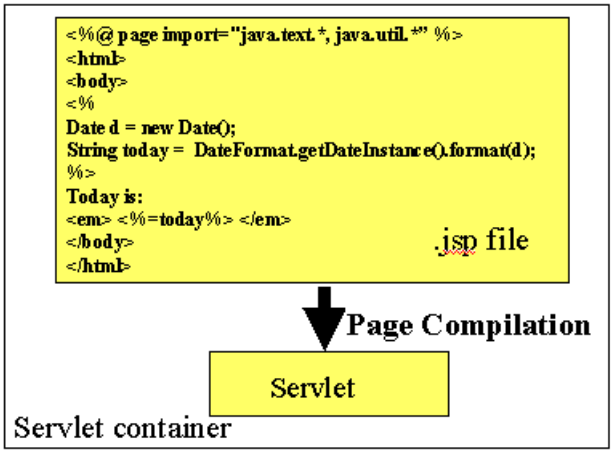
1.1.5.1. Installation und Konfiguration von Tomcat

Arbeiten Sie jetzt die Übung "Installation und Konfiguration von Tomcat" durch. Nach dem Durcharbeiten sollte die Basis für die weiteren Übungen gesetzt sein und Sie über eine lauffähige Tomcat Installation verfügen.

1.1.6. JSP Architektur

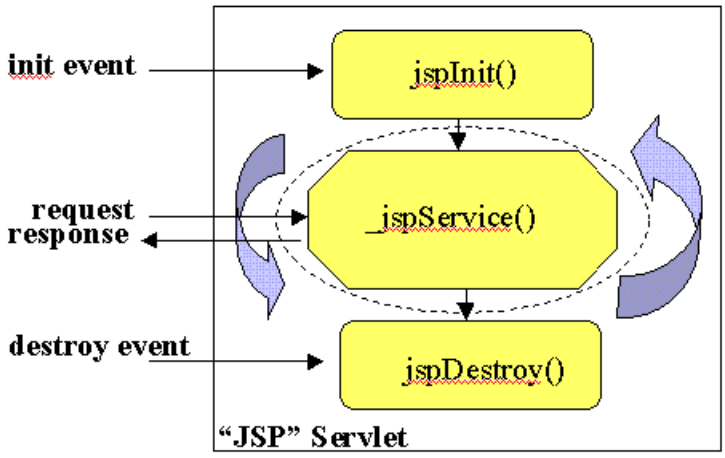
Ziel von JDP ist es, Servlets auf eine deklarative, darstellungszentrierte Art und Weise zu entwickeln, auf der Basis des Servlets APIs. Daher sollte es kaum erstaunen, dass Servlets und JSP vieles gemeinsam haben.

Typischerweise unterscheidet man bei JSPs eine Übersetzungs- und eine Request Processing / Anforderungsverarbeitungs- Phase. Die Übersetzungsphase wird lediglich einmal ausgeführt, ausser die JSP wird verändert. Falls Sie keine Syntaxfehler innerhalb der Seite produzierten, ist das Ergebnis eine JSP Seitenimplementationsklasse, welche das Servlet Interface implementiert. Schematisch sieht dies folgendermassen aus:



Die Übersetzungsphase wird typischerweise durch die JSP Engine ausgeführt, dann wenn die JSP Seite das erste Mal angefordert wird. Sie können aber JSP auch im voraus in eine Class Datei übersetzen. Dies kann sehr sinnvoll sein, da Sie damit die Ladezeit bei der ersten Ausführung reduzieren können. Viele Details betreffend der Übersetzungsphase und der Lokation der Dateien sind abhängig von der Implementation (hier also von der Tomcat Version).

Tomcat generiert für die JSP Seite Java Code. Dieser erweitert die `HttpJspBase` Klasse, welche ihrerseits das Servlet Interface implementiert.Die Service Klasse `_jspService()` enthält den Inhalt der JSP Seite, die Sie spezifiziert haben. Obschon `_jspService()` nicht überschrieben werden kann, können Sie Events initialisieren und zerstören, indem Sie die Methoden `jspInit()` und `jspDestroy()` in Ihren JSP Seiten implementieren.



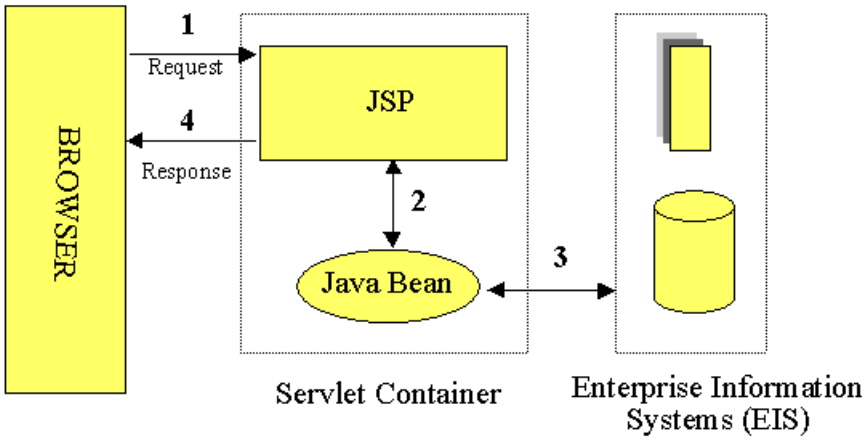
Nach dem Laden der Class Datei in den Servlet Container ist die `_jspService()` Methode für die Beantwortung der Client Anfragen verantwortlich. Standardmässig wird die Methode durch einen eigenen Thread im Servlet Container repräsentiert. Damit können mehrere Client Anfragen nebenläufig bearbeitet werden.

1.1.7. JSP Zugriffs Modelle

Die JSP Spezifikation definierte zwei unterschiedliche Modelle für den Einsatz der JSP Technologie. Diese waren als Modell 1 und Modell 2 Architekturen bekannt. Der wesentliche Unterschied der beiden Modelle bestand in der Art und Weise oder der Ort, an dem die Aufrufe bearbeitet wurden.

Das folgende Schema zeigt die Modell-1 Architektur:

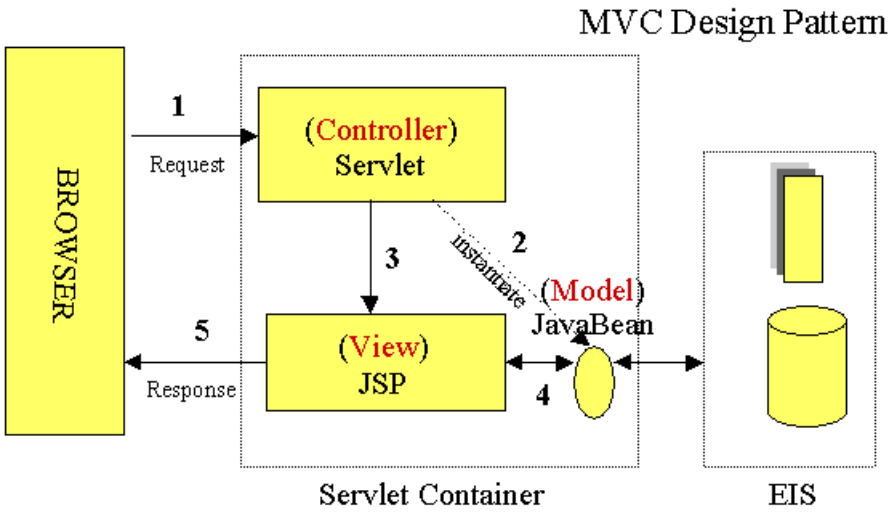
Fehler! Textmarke nicht definiert.



In der Modell 1 Architektur werden eintreffende Anfragen des Web Browsers direkt an die JSP Seite weitergeleitet. Diese ist dann für die Bearbeitung der Seiteninhalte zuständig und antwortet dem Browser. Alle Zugriffe auf irgendwelche Daten werden durch Java Beans ausgeführt, womit die Trennung von Darstellung und Inhalt erreicht wird.

Die Modell 1 Architektur kann in einfachen Anwendungen bestens eingesetzt werden, in komplexeren Implementationen allerdings weniger.Dieses Modell führt in der Regel zu recht umfangreichem eingebettetem Java Code in den JSP Seiten, speziell in den Fällen, in denen umfangreiche Manipulationen benötigt werden. Für Java Programmierer ist dies kein Problem; für JSP Programmierer und Web Page Designer schon! In umfangreichen Projekten führt dies in der Regel zu Problemen. Ein anderer negativer Punkt dieser Architektur ist, dass die JSP Seiten individuell ihren Zustand und die Security überwachen müssen.

Die Modell 2 Architektur unten ist eine serverseitige Implementation des Model / View / Controller Design Patterns.



Fehler! Textmarke nicht definiert.
In diesem Modell wird die Bearbeitung aufgeteilt in eine Präsentations- und eine Front-Komponente. Die Präsentationskomponenten sind JSP Seiten, welche HTML / XML Antworten für den Benutzer, den Browser generieren. Die Frontkomponenten, die Controller, beschäftigen sich nicht mit Präsentationsfragen; sie sind lediglich für alle HTTP Anfragen zuständig. Sie kreieren die benötigten Beans oder Objekte, welche von der Präsentationskomponente benötigt werden. Zudem entscheiden sie, an welche Präsentationskomponenten die Anfrage beantworten muss.Die Frontkomponenten (Controller) können als JSP oder Servlets realisiert werden.

Der Vorteil dieser Architektur besteht darin, dass die Präsentation sich nicht auch noch mit der Bearbeitung und Aufbereitung befassen muss. Die Präsentationskomponente (View) muss lediglich die Objekte und Beans beschaffen, welche der Controller, die Frontkomponente ausgewählt hat und muss den Inhalt extrahieren, der im statischen Template dargestellt werden soll.

Diese klare Aufteilung der Aufgaben führt auch zu klaren Verantwortungen in der Entwicklungsphase (Entwickler befassen sich mit dem Controller, Web Designer mit der View).

Ein weiterer Vorteil ist, dass man genau eine Schnittstelle serverseitig hat, an die man die Requests senden muss. Damit wird die Wartung vereinfacht, die Sicherheit kontrollierbarer.

1.1.8. JSP Syntax Grundlagen

Die Syntax von JSP ist recht einfach und besteht aus *Direktiven / Directives*, *Skriptelementen* und *Standardaktionen*.

1.1.8.1. Directives / Direktiven

JSP Direktiven sind Messages an die JSP Engine. Sie liefern kein direkt ersichtliches Ergebnis, geben aber der JSP Engine Anweisungen, was mit dem rest der Page zu geschehen hat. Direktiven werden in spezielle Tags eingeschlossen: <%@ ... %>.

Die zwei üblichen Direktiven sind : `page` und `inlude`. Zusätzlich können Sie eigene Tags definieren (custom tag libraries).

1.1.8.1.1. Page Directive

Die page Direktive finden Sie typischerweise ganz oben auf den JSP Seiten. Es sind beliebig viele page Direktiven auf einer JSP Seite erlaubt, einzig das Attribut / Wert Paar muss eindeutig sein. Unerkannte Attribute oder Werte führen zu Übersetzungsfehlern.

Beispiel:

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

stellt die in den importierten Packages deklarierten Datentypen und Klassen für das Skripting in der JSP zur Verfügung und setzt den Page-Buffer auf 16K.

1.1.8.1.2. Include Directive

Mit der inlude Direktive kann man Seiten überschaubarer gestalten, zerlegen. Beispielsweise lassen sich damit einfach Kopf- und Fusszeilen definieren und Standardisieren.

Die so einbezogenen Seiten oder Seitenelemente können dabei statisch oder generelle JSP Inhalte sein.

Beispiel:

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

stellt eine standardisierte Copyright Meldung zur Verfügung, die in einer separaten HTML Seite (`copyright.html`) definiert wird.

1.1.8.2. Deklarationen

JSP Deklarationen gestattet es Ihnen Variablen zu definieren, welche Informationen speichern können, die für die gesamte Seite gültig und lesbar sind, falls nötig. Der Vorteil dieser Deklarationen ist die einfache Möglichkeit solche zu definieren. Der Nachteil ist aber eine schlechte Wartbarkeit. Daher sollten Sie logikintensive Konstrukte besser in JavaBeans verschieben und damit die Wiederverwendbarkeit und Wartbarkeit drastisch verbessern.

Deklarationen werden in **<%!..%>** Tags definiert.

Zu beachten ist, dass alle Variablendeklarationen mit einem Semikolon angeschlossen werden müssen, da es sich um Java Deklarationen handelt.

Beispiel:

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

Sie können aber auch Methoden deklarieren. Beispielsweise könnten Sie das Initialisierungsevent einer JSP deklarieren:

```
<%! public void jspInit() {
        //hier folgt der Initialisierungscode
    }
%>
```

1.1.8.3. Expressions / Ausdrücke

Die Ergebnisse in JSPs werden nach der Evaluation in Zeichenketten umgewandelt und können damit auch ausgegeben werden. Ausdrücke werden typischerweise eingesetzt, um einfache Variablenwerte oder Rückgabewerte von Beans auszugeben. JSP Ausdrücke beginnen mit `<%= . . .%>` und umfassen keine Semikolons.

Beispiele:

```
<%= jspVariable %>
<%= jspBean.getName() %>
```

1.1.8.4. Scriptlets

JSP Codefragmente oder Scriptlets werden in `<% . . . %>` Tags eingebettet.. Dieser Java Code wird immer dann ausgeführt, wenn eine Anfrage mit Hilfe einer JSP Seite beantwortet wird. Sie können beliebigen Java Code in einem Scriptlet verwenden, auch mehrzeilig.

Beispiel:

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

Die Ausgabe ist "Hello World" unterschiedlich gross geschrieben (als Header 1 ... 4)

1.1.8.5. Comments / Kommentare

Sie können jederzeit einfache HTML Kommentare in Ihre JSP Seite einfügen, mit
`<!--Kommentar //-->`

Diese sind alerdings für den Client im Browser sichtbar, falls der Client Benutzer sich den Quellcode ansieht.

Falls Sie den Kommentar mit JSP Tags versehen, wird er zu einem serverseitigen Kommentar und damit für den Client nicht mehrerkennbar:

Beispiel:

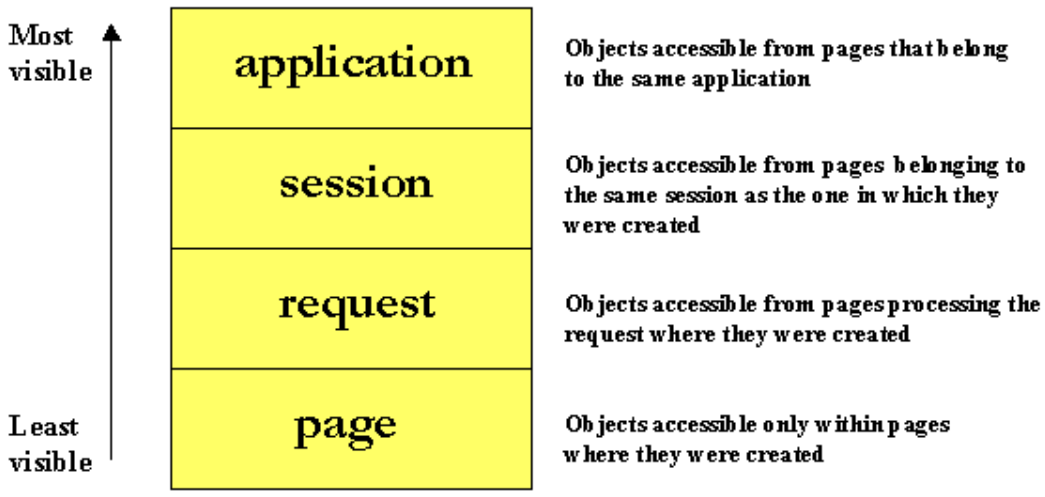
```
<%-- comment for server side only --%>
```

Mit Kommentaren können Sie auch selektiv Scriptlets oder Tags ausblenden, beispielsweise zum Testen der JSPs.

1.1.9. Objekt Scopes

Bevor wir uns die JSP Syntax und Semantik genauer ansehen, sollten Sie verstehen, wie und in welchem Umfang Java Objekte innerhalb der JSP Seiten sichtbar sind. Objekte können implizit mit JSP Direktiven kreiert werden oder explizit mit Hilfe von Aktionen, seltener direkt mit Hilfe von Programmcode.

Instanziierte Objekten kann ein Scope Attribut hinzugefügt werden, mit dem festgelegt werden kann, wann eine Objektreferenz sichtbar ist und wann sie entfernt wird.



Das obige Diagramm zeigt die Gültigkeitsattribute (application ... page) und deren Definitionen.

1.1.10. JSP Implizite Objekte

Der JSP Container stellt Ihnen der Einfachheit halber sogenannte implizite Objekte zur Verfügung. Diese können Sie in Ihren Scriptlets oder Ausdrücken einsetzen, ohne dass sie zuerst kreiert oder definiert werden müssen. Hinter diesen impliziten Objekten stecken Java Klassen. Diese werden Ihnen mit Hilfe von Wrappern zur Verfügung gestellt. Auch hier geht es um Servlets APIs, die Ihnen zur Verfügung gestellt werden. Die folgenden neun impliziten Objekte geben Ihnen einen Eindruck, was Ihnen so alles zur Verfügung steht:

- `request`:
  repräsentiert den `HttpServletRequest` (`http://java.sun.com/products/servlet/2.2/javadoc/javax/servlet/http/HttpServletRequest.html`) und löst diesen Service aus.
  Scope : Request.

- `response`:
  repräsentiert die `HttpServletResponse`
  Dieses Objekt wird sehr selten eingesetzt.
  Scope : Page

- `pageContext`:
  kapselt implementationsabhängige Teile in einen `PageContext`
  Scope: Page.

- `application`:
  repräsentiert den `ServletContext` den man von einem Servlet Konfigurationsobjekt erhält.
  Scope : Application.

- `out`:
  ein `JspWriter` Objekt, welches in einen Output Stream schreibt.
  Scope: Page.

- `config`:
  repräsentiert das `ServletConfig` Objekt für diese JSP.
  Scope: Page.

- `page`:
  synonym für den "this" Operator, beispielsweise in `HttpJspPage`, wird allerdings von Designern kaum eingesetzt.
  Scope : Page

- `session`:
  eine `HttpSession`.
  Scope: Session

- `exception`:
  ein `Throwable` Objekt, welches von der aktuellen JSP stammt
  Scope: Page

Diese impliziten Objekte sind lediglich innerhalb der vom System generierten `_jspService()` Methode sichtbar, also nicht in eigenen Deklarationen!

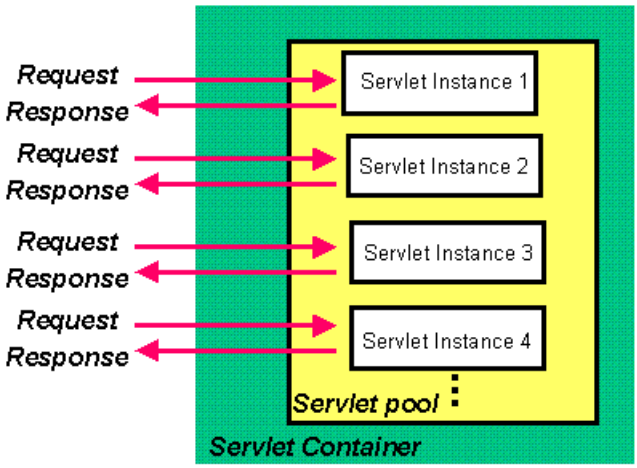
1.1.11. Synchronisationsfragen

Standardmässig sind alle Servicemethoden der JSP Implementationsklassen, welche Client Anfragen erledigen müssen, multithreading fähig. Der JSP Entwickler ist für eine sinnvolle und effiziente Synchronisation zuständig. Es gibt unterschiedliche Techniken, um Ihre Seiten threadfähig zu gestalten.

Die einfachste Technik ist der Einsatz der JSP `page` Direktive:

```
<%@ page isThreadSafe="false" %>
```

Mit dieser Anweisung wird der JSP Container angewiesen, das `SingleThreadModel` Interface anzuwenden. Damit werden mehrere Instanzen des Servlets in den Speicher geladen. Gleichzeitig eintreffende Requests werden auf die vorhandenen Instanzen verteilt, wie das folgende Bild zeigen soll:



Das Scheduling geschieht gemäss Round Robin. Nachteil dieser Methode ist, dass diese nicht skalierbar ist. Falls viele Requests vorliegen, müssen Clients eventuell lange warten, bis sie bedient werden.

Besser wäre eine Methode, in der die Objektzugriffe explizit synchronisiert werden.

Beispiel:

```
<% synchronized (application) {
   SharedObject bsp = (SharedObject)
        application.getAttribute("sharedObject");
        bsp.update(someValue);
        application.setAttribute("sharedObject",bsp);
}
%>
```

1.1.12. Exception Handling

JSP bietet Ihnen eine ziemlich elegante Art und Weise, wie Sie Laufzeitfehler behandeln können. Natürlich können Sie eigene Techniken entwickeln. Aber JSP bietet Ihnen mit der page Direktive eie Möglichkeit, Ausnahmen auf eine "Fehlerbehandlungsseite" umzuleiten.

Beispiel:

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

Diese Direktive informiert den JSP Engine alle Exceptions auf die JSP Fehlerseite `errorHandler.jsp` umzuleiten. Die Seite `errorHandler.jsp` muss sich als Fehlerbehandlungsseite kennzeichnen:

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

Damit kann man in Scriplets auf die Meldungen des Throwable-Objekts zugreifen und anzeigen.

1.1.13. Übung

Exception Handling in JSP

Bearbeiten Sie nun die Übungsaufagen zum Thema Exception Handling. Sie finden auch eine Musterlösung und einige Lösungshinweise im Übungsteil zu dieser Kurseinheit.

Ein Hinweis zur Einbindung in Tomcat:
falls Sie Ihre Lösung testen wollen, bietet es sich an, einfach Ihre Dateien in ein Verzeichnis unterhalb des Examples Verzeichnisses von Tomcat zu kopieren. Dies ist der offizielle Ratschlag gemäss FAQs im Doc Verzeichnis von Tomcat.

Dort finden Sie auch Hinweise, wie Sie ein anderes Verzeichnis einbinden könnten, sofern Sie dies zum Üben wirklich machen wollen.

1.1.14. Session Management

Standardmässig nehmen alle JSP Seiten an HTTP Sessions teil. Das HTTP Objekt `HttpSession` kann innerhalb von Scriptlets benutzt werden, mit Hilfe des impliziten JSP Objekts. Sessions können bestens eingesetzt werden, um Beans und Objekte abzuspeichern, sofern diese in mehreren JSP Seiten und Servlets eingesetzt werden sollen. Session Objekte werden mit einer Session ID eindeutig gekennzeichnet und innerhalb des Browsers als Cookie abgespeichert. Falls der Browser keine Cookies akzeptiert, kann die Session ID auch anders abgespeichert werden, allerdings serverabhängig.

Sie können keine Basisdatentypen in eine Session abspeichern, aber sonst alle gültigen Java Objekte, wobei jedes Objekt eindeutig identifiziert werden muss.

Beispiel:

```
<% Objekt obj = new Objekt();
session.putValue("Objekt",obj);
%>
```

Damit wird eine Instanz der Objekt Klasse innerhalb der JSP Seite und Servlets verfügbar gemacht, beschränkt auf eine und die selbe Session.

Aus einer anderen JSP Seite können Sie folgendermassen auf dieses Objekt zugreifen:

```
<% Objekt meinObj = (Objekt) session.getValue("Objekt");%>
```

Der Aufruf der Methode `session.getValue()` liefert eine Referenz auf ein generisches Objekt. Dieses müssen Sie anschliessend auf den gewünschten Datentyp casten.

Beachten müssen Sie, dass eine JSP nicht unbedingt an einer Session beteiligt sein muss! Sie können mit der `page` Direktive angeben, dass diese JSP Seite zu keiner Session gehört:

```
<%@ page session="false" %>
```

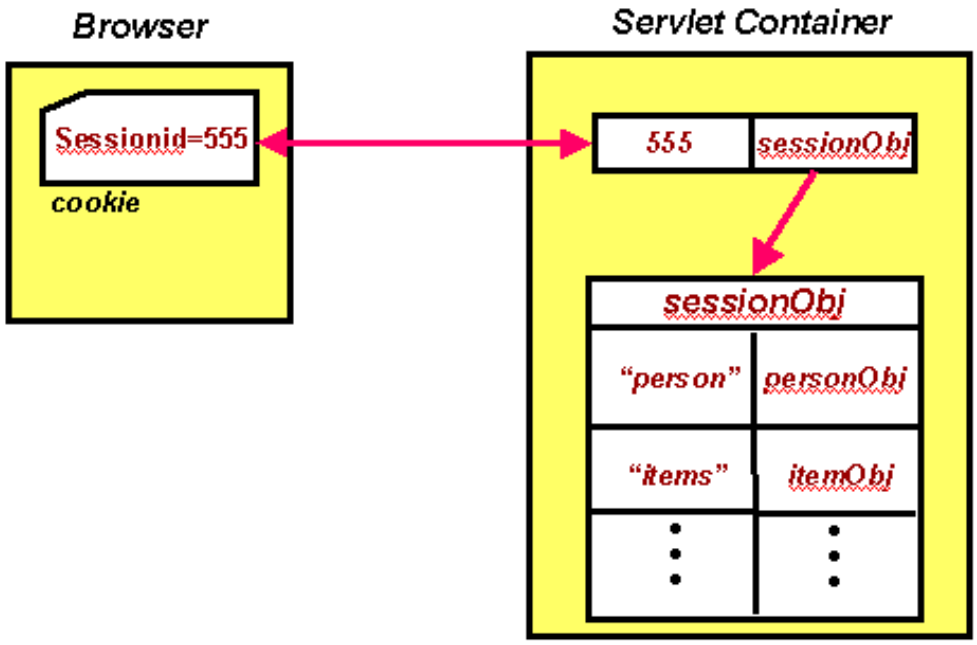
Pro Session können Sie beliebig viele Objekte speichern. Allerdings können viele oder grosse Objekte zu Leistungseinbussen führen und Heap Memory aufbrauchen. Die meisten Server limitieren die Lebensdauer der Session Objekte auf 30 Minuten. Diesen Standardwert können Sie aber leicht überschreiben:

```
sessionObj. setMaxInvalidationInterval(int secs)
```

gestattet es Ihnen eigene Session Lebensdauern zu definieren.

Die folgende Abbildng skizziert diese Situation:

Der Client eröffnet eine Session (`555`). Diese Session ID wird auf dem Server und falls



möglich als Cookie auf dem Client notiert.

Die einzelnen Objekte werden auf dem Heap des Servlet Containers gelegt und bleiben dort, entweder solange wie der Standardwert des Servers angibt, oder solange, wie Sie mit der Methode `setMaxInvalidationInterval(int secs)` spezifiziert haben oder bis die Session beendet wurde.

Sobald die Session beendet wird oder ungültig wird, beispielsweise weil ein Session Timeout geschieht, werden die Live Referenzen des JSP Engine markiert und dem Garbage Collector übergeben.
Fehler! Textmarke nicht definiert.

1.1.15. Standard Aktionen

Aktions / Aktionen gestatten es Ihnen komplexe Aufgaben wie die Instanzierung von Objekten oder die Kommunikation mit serverseitigen Ressourcen (DBMS, JSPs, Servlets, ...) zu realisieren, häufig ohne grossen Java Programmieraufwand.

Sie können zur Erledigung dieser Aufgaben entweder Scriptlets oder eben Action Tags einsetzen. Action Tags sind in der Regel besser zu warten.

1.1.15.1. Einsatz von JavaBean Komponenten

Das Komponentenmodell der JSP Technologie basiert auf der JavaBeans Komponentenarchitektur. JavaBeans Komponenten sind einfach Java Objekte, welche bestimmte Design / Namensgebungsmuster befolgen. Beans kapseln ihre Eigenschaften, indem sie diese als `private` deklarieren und spezielle getter/setter Methoden deklarieren, welche öffentlich sind und eingesetzt werden können, um Werte zu lesen oder zu setzen.

Bevor Sie Beans in JSPs einsetzen können, müssen Sie diese in der JSP deklarieren und eine Referenz darauf erhalten. Mit dem Tag

```
<jsp:useBean>
```

können Sie versuchen eine Referenz auf eine bestehende Instanz zu erhalten. Diese JavaBean wurde vielleicht instanziert und lagert im Container / der Session, mit einer eindeutigen ID.

Falls keine Referenz auf die JavaBean gefunden wurde, wird eine neue Instanz kreiert:

Beispiel:

```
<jsp:useBean id="user" class="com.MeineFirma.Person" scope="session" />
```

In diesem Beispiel wird eine Instanz der `Person` Klasse aus dem Package `com.MeineFirma` kreiert.

Falls später oder in einer anderen JSP zur selben Session ein Tag auftritt mit einer Referenz auf diese JavaBean, wird wieder auf das bereits bestehende Objekt zugegriffen.

Der Tag `<jsp:useBean>` kann optional auch noch einen Body / Zusatzinformationen enthalten:

```
<jsp:useBean id="user" class="com.MeineFirma.Person"
   scope="session">

<%  user.setDate(DateFormat.getDateInstance(                    ).
    format(new Date()));
  //etc..
%>

</jsp:useBean>
```

Auf die Properties der JavaBeans können Sie mit dem Tag

```
 <jsp:getProperty>
```

zugreifen. Dabei müssen Sie den Namen des Feldes und seien Wert angeben:

```
<jsp:getProperty name="user" property="name" />
```

Das Setzen der Properties geschieht mit dem Tag

```
<jsp:setProperty>
```

Sie können den Namen und Wert spezifizieren und setzen:

```
<jsp:setProperty name="user" property="name" value="ichBins" />
```

oder

```
<jsp:setProperty name="user" property="name"   value="<%=expression %>" />
```

Falls Sie Beans für die Verarbeitung der Formulardaten einsetzen wollen, sollten Sie sich an ein einfaches Design Pattern halten:

benennen Sie die Formularfelder und JavaBean Felder genau gleich. Damit reduzieren Sie die Fehlerwahrscheinlichkeit beträchtlich. Zudem können sie in diesem Fall die Properties generisch angeben:

```
<jsp:setProperty name="user" property="*"/>
```

Diese Technik benutzt *introspection*, mit deren Hilfe Klassen ihre Eigenschaften anzeigen können. Introspektion wird durch die JSP Engine angeboten und beruht auf dem Java Reflection Interface. Falls Sie komplexe Formulare einsetzen wollen, könen Sie mit diesem Pattern sehr viel Zeit / Aufwand sparen.

Falls Sie keine Namensgleichheit haben, müssen Sie explizit Properties mit Parametern matchen / verknüpfen:

```
<jsp:setProperty name="user" property="address"
  param="parameterName" />
```

1.1.16. Übung

Bearbeiten Sie jetzt die zwei Übungen

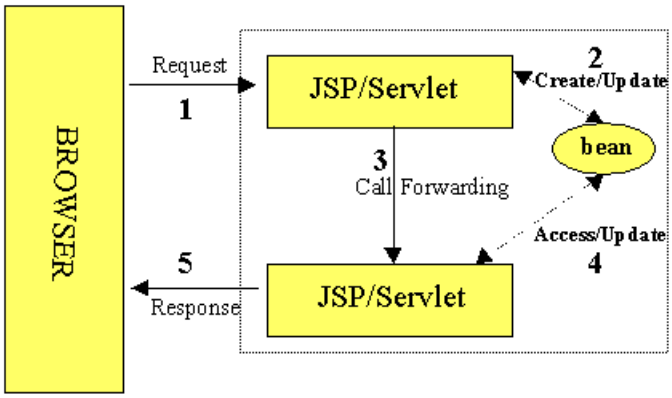
1) JSP Objekt Scopes verstehen
2) Formulare mit JSP verarbeiten.

1.1.16.1. Requests weiterleiten

Mit dem `<jsp:forward>` Tag können Sie einen Request auf irgend eine JSP oder ein Servlet oder auch eine statische HTML Seite aus dem selben Context umleiten. Contexts finden Sie in der dokumentation von Tomcat erklärt. Die verarbeitung der aktuellen Seite wird ab diesem Tag unterbrochen.

```
<jsp:forward page="somePage.jsp" />
```

Die auslösende / aufrufende Seite kann auch nocj JavaBean Parameter an die Umleitung weitergeben. Diese können im Request mitgegeben werden. Die folgende Skizze veranschaulicht diesen Sachverhalt:

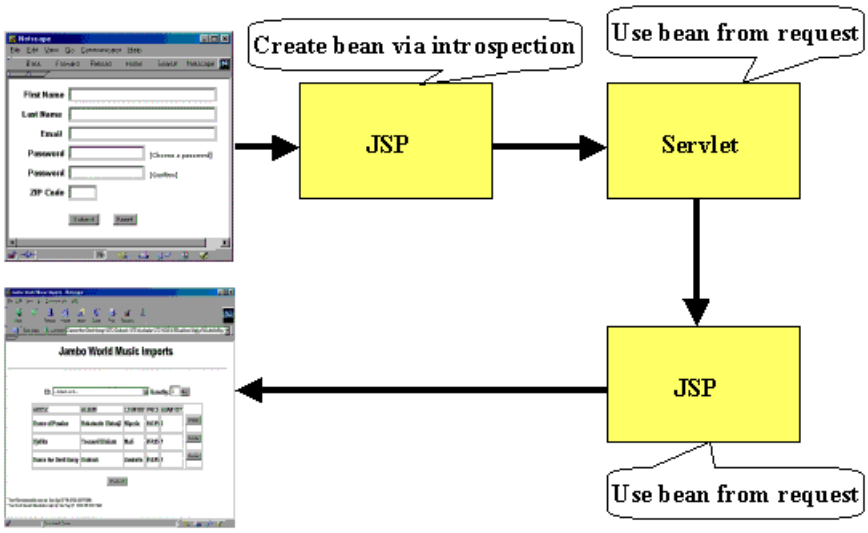


Ein <jsp:forward> Tag kannn auch jsp:param Subelemente enthalten und mit dieser Technik Werte weiterreichen:

```
<jsp:forward pa. ge="<%= somePage %>" >
<jsp:param name="name1" value="value1" />
<jsp:param name="name2" value="value2" />
</jsp:forward>
```

1.1.16.1.1. Request Chaining / Das Verketten von Requests

Request Chaining ist ein mächtiges Feature und kann beispielsweise eingesetzt werden, um JSPs und Servlets effizient zu mischen / zu kombinieren. Die folgende Skizze zeigt grob wie dies geschehen könnte:



Formaler:

```
<jsp:useBean id="fBean" class="meinpackage.FormBean"  scope="request"/>
<jsp:setProperty name="fBean" property="*" />
<jsp:forward page="/servlet/JSP2Servlet" />
```

Und so könnte es weitergehen beim Servlet:

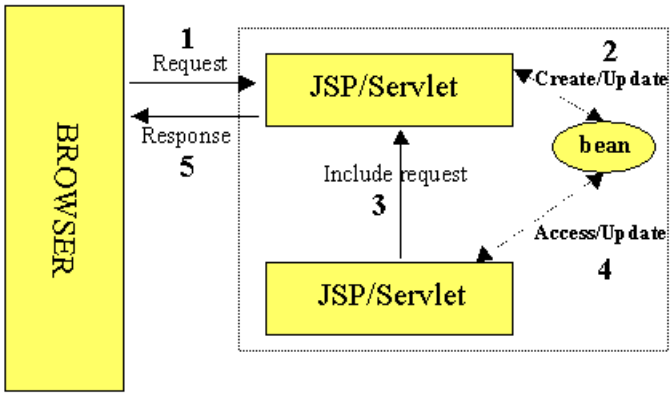
```
public void doPost (HttpServletRequest request,
                    HttpServletResponse response) {
  try {
    FormBean f = (FormBean) request.getAttribute
                                          ("fBean");
    f.setName("Mogambo");
    // do whatever else necessary
    getServletConfig().getServletContext().
      getRequestDispatcher("/jsp/Bean2.jsp").
        forward(request, response);
  } catch (Exception ex) {
    . . .
  }
}
```

Und von dort geht es weiter zur zweiten JSP

```
<html>
<body>
<jsp:useBean id="fBean" class="meinpackage.FormBean"
   scope="request"/>
<jsp:getProperty name="fBean" property="name" />
</body>
</html>
```

1.1.16.2. Einbeziehen von Requests

Das <jsp:include> Tag kann auch eingesetzt werden, um Request zu einer statischen oder dynamischen Ressource weiterzuleiten, welche zum selben Context gehört, wie die aufrufende JSP.




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

Ihrer Phantasie sind keine Grenzen gesetzt. Diese letzten Abschnitte sollten Ihnen lediglich zeigen, dass JSPs flexibel kombiniert werden können. Ob Sie dies auch brauchen, bleibt Ihnen überlassen.

1.1.17. Web Sites

Die folgenden Web Sites enthalten die wichtigsten Informationen zum Thema JSP:

- Sun Microsystems, JSP Home Page
  http://java.sun.com/products/jsp/

- JSP-INTEREST Mailing List Archive
  http://archives.java.sun.com/archives/jsp-interest.html

1.1.18. Dokumentation und Spezifikationen

Auch diese finden Sie bei Sun:

- JSP 1.1 Specification
  http://java.sun.com/products/jsp/download.html

- Sun Microsystems, Inc. Java 2 Enterprise Edition (J2EE) Home page
  http://java.sun.com/j2ee/

- Tomcat Project
  http://java.sun.com/products/jsp/tomcat/

- JSP Technical Resources
  http://java.sun.com/products/jsp/technical.html

- Java Servlet API
  http://java.sun.com/products/servlet/

- JSP Whitepaper
  http://java.sun.com/products/jsp/whitepaper.html

- JSP Syntax Card
  http://java.sun.com/products/jsp/syntax.html

Ein letzter Hinweis:
JSPs werden als Teil der J2EE betrachtet. Das J2EE SDK wird / bietet bereits einfache Konstrukte an, um JSPs zu entwickeln. Aber ich nehme an, Sie haben gesehen, dass man mit dem JBuilder genausogut JSPs entwickeln kann.

JAVA SERVER PAGES - PRAXIS 1

1.1. EINLEITUNG 1
1.1.1. Neue Konzepte 2
1.1.2. Lernziele 2
1.1.3. Voraussetzungen 2
1.1.4. Vorteile von JSP 3
1.1.4.1. Vergleich von JSP mit ASPs 4
1.1.4.2. JSP oder Servlets? 5
1.1.5. Übung 5
1.1.5.1. Installation und Konfiguration von Tomcat 5
1.1.6. JSP Architektur 6
1.1.7. JSP Zugriffs Modelle 7
1.1.8. JSP Syntax Grundlagen 9
1.1.8.1. Directives / Direktiven 9
1.1.8.1.1. Page Directive 9
1.1.8.1.2. Include Directive 9
1.1.8.2. Deklarationen 10
1.1.8.3. Expressions / Ausdrücke 10
1.1.8.4. Scriptlets 10
1.1.8.5. Comments / Kommentare 11
1.1.9. Objekt Scopes 11
1.1.10. JSP Implizite Objekte 12
1.1.11. Synchronisationsfragen 13
1.1.12. Exception Handling 14
1.1.13. Übung 14
1.1.14. Session Management 15
1.1.15. Standard Aktionen 17
1.1.15.1. Einsatz von JavaBean Komponenten 17
1.1.16. Übung 18
1.1.16.1. Requests weiterleiten 19
1.1.16.1.1. Request Chaining / Das Verketten von Requests 20
1.1.16.2. Einbeziehen von Requests 21
1.1.17. Web Sites 22
1.1.18. Dokumentation und Spezifikationen 22