Prof. Dr. Jörg Desel

Kurs 01796

Web-Programmierung

LESEPROBE

Fakultät für
Mathematik und Informatik

FernUniversität in Hagen

Leseprobe - Web-Programmierung (01796)

Kursinhalte

Kurseinheit 3

Java EE: Servlets und JavaServer Pages

Die vorliegende dritte Kurseinheit beginnt in Kapitel 7 mit einem Überblick über die *Java Platform, Enterprise Edition (Java EE)*, die eine Standardarchitektur für Java-basierte Anwendungen spezifiziert. Der Kurs bezieht sich auf Version 7 der Java EE Spezifikation und setzt diese für die Entwicklung von Web-Anwendungen ein. Die Verzeichnisstruktur der Web-Schicht von Java EE schließt das Kapitel ab.

Der Rest dieser Kurseinheit ist den Konzepten *Servlet* und *JavaServer Pages (JSP)* gewidmet. Kapitel 8 beschreibt neben den Aufgaben eines Servlet einen typischen Request-Ablauf bei Verwendung von Servlets. Kapitel 9 beginnt mit einer Motivation des JSP-Konzepts und einem typischen Request-Ablauf mit einem Servlet und einer JSP-Seite. Anschließend geht es um Techniken zur Behandlung von dynamischen Inhalten in einer JSP durch *Scripting-Elemente*, die *Expression Language* und die *JavaServer Pages Standard Tag Library*. Kapitel 8 und 9 werden jeweils abgerundet durch ein ausführliches Beispiel.

Kapitel 10 schließt die Kurseinheit mit Hinweisen zur Verwendung von Servlets und JSP-Seiten im Sinne der Grundsätze des Software Engineerings ab.

Letzte Aktualisierung: Juli 2015

7. Einstieg

7.1. Java EE im Überblick

Die *Java Platform, Enterprise Edition (Java EE)* ist die Spezifikation einer Standardarchitektur für Java-basierte Anwendungsprogramme, die von der Firma Sun Microsystems herausgegeben wurde und seit der Übernahme von Sun Microsystems durch die Firma Oracle im Jahr 2010 von dieser weiterentwickelt wird. In diesem Kurs werden wir Java EE nur für die Entwicklung von Web-Anwendungen einsetzen. Dabei sei erwähnt, dass viele Technologien von Java EE auch für Java Desktop-Anwendungen genutzt werden können und nicht zwangsweise auf ein Web-Umfeld angewiesen sind.

Unter dem Begriff „Spezifikation“ (aus dem mittellateinischen: specificatio = Auflistung, Verzeichnis) versteht man im Allgemeinen die „Gesamtheit von Vorgaben, nach denen etwas produziert wird “[1]. Im Software Engineering ist eine Spezifikation mit einer Anforderungsdefinition gleichzusetzen. Sie gibt Informationen darüber, was ein Modul, eine Software oder eine Architektur für Anforderungen (Eigenschaften) erfüllen muss, ohne konkrete Auskünfte darüber zu geben, *wie* diese Anforderungen in einer späteren Phase des Softwareentwicklungsprozesses umgesetzt und implementiert werden.

Die Spezifikation Java EE umfasst 39 Dokumente (einzelne Dokumente besitzen mehr als 500 Seiten), die in sechs Kategorien eingeteilt sind[2]. Im Unterschied zu Anforderungsdokumenten, die keine konkrete Implementierung adressieren, ist die Java EE-Spezifikation auf die Programmiersprache Java ausgerichtet und enthält sehr viele Code-Vorgaben und Code-Beispiele. Eine Code-Vorgabe kann z.B. ein Code-Ausschnitt sein, der eine Java-Schnittstelle definiert, die mit ihrer (in der Spezifiation noch offenen) Implementierung eine bestimmte Funktionalität zur Verfügung stellen soll.

Die Architektur von Java EE orientiert sich an einer Schichtenarchitektur. Grundlegend werden drei Schichten unterschieden:

- der Client-Computer
- der Java EE-Server
- der Datenbank-Server

[1] Quelle www.duden.de, Stichwort: Spezifikation

[2] Diese Kategorien sind: Java EE Platform, Web Application Technologies, Enterprise Application Technologies, Web Services Technologies, Management and Security Technologies und Java EE-bezogene Spezifikationen in Java SE, siehe: http://www.oracle.com/technetwork/java/javaee/tech/index.html

Bei einem Web-Anwendungsprogramm verwenden Benutzer einen Web-Browser, der auf ihrem Rechner installiert ist (Client). Der Web-Browser stellt über das Internet Anfragen (Requests) an den Java EE-Server und erhält Antworten (Responses), dies können z.B. HTML-Seiten sein. Bei Bedarf greift der Java EE-Server auf einen Datenbank-Server zu, um dort gespeicherte Daten zu laden, diese ggf. zu bearbeiten und in seine Antwort an den Client einfließen zu lassen.

Der Java EE-Server ist in zwei weitere Schichten unterteilt: die Web-Schicht und die Schicht der *Geschäftslogik*. An dieser Stelle sollen die Begriffe „Geschäftslogik“ (engl. business logic) und „Geschäftsanwendung“ (engl. business application) erläutert werden, die im Zusammenhang mit Java EE immer wieder auftauchen werden. Java EE wurde als Softwarearchitektur für Geschäftsanwendungen konzipiert. Eine Geschäftsanwendung bezeichnet grob ein Anwendungsprogramm, das ein Unternehmen in seinen Aufgaben unterstützt und im Allgemeinen zu groß und zu komplex für sehr kleine Unternehmen oder Einzelpersonen ist. Die Schicht der Geschäftslogik (engl. business tier) innerhalb des Java EE-Servers beinhaltet die Software-Komponenten, die die Geschäftslogik umsetzen. Eine solche Komponente kann z.B. die Anforderung realisieren, dass vor dem Verkauf eines Produkts die Daten der Kreditkarte des Kunden geprüft werden. Die Geschäftslogik wird im weiteren Kurs auch Anwendungslogik genannt. Die Web-Schicht (engl. web tier) des Java EE-Servers enthält alle Software-Komponenten, die Anfragen von einem Client entgegennehmen, Antworten in Form von Web-Seiten erzeugen und an den Client zurücksenden. Diese Komponenten werden auch Web-Komponenten genannt. Unter einer Software-Komponente versteht man dabei eine funktional abgeschlossene Softwareeinheit, die aus Java-Klassen und -dateien besteht und Teil eines Java EE-Anwendungsprogramms ist. Unterschiedliche Komponenten können miteinander kommunizieren.

Wir wissen nun, dass es sich bei der Java EE-Spezifikation um eine Sammlung von Dokumenten und Vorgaben handelt, jedoch nicht, wie ein Java EE-Anwendungsprogramm ausgeführt wird. Hier kommt der Begriff des Application Server (Anwendungs-Server) ins Spiel. Ein *Application Server* bezeichnet in der Java EE-Welt keinen physischen Rechner, sondern eine Software, die auf einem Server installiert ist und eine Java EE-Spezifikation implementiert. Sie umfasst somit alle durch die Spezifikation gegebenen Anforderungen und ist in der Lage ein entwickeltes Java EE-Anwendungsprogramm auszuführen.

Die Java EE-Spezifikation definiert vier Bereiche, die *Container* genannt werden. Zwei dieser Container sind Teil des Application Server, die anderen beiden Container befinden sich auf dem Client-Rechner. Jeder dieser vier Container basiert auf der Java Standard Edition (Java SE) und stellt eine Laufzeitumgebung für verschiedene Komponenten bereit. Abbildung 7.1 zeigt die Struktur dieser vier Container. Die jeweils in einem Container dargestellten Komponenten werden durch den Container verwaltet.

Wir skizzieren kurz die vier Container und deren Komponenten, genauere Erklärungen folgen später:

- Der *Web Container* ist für die Web-Komponenten *Java Servlet* , die *JavaServer Faces Technologie (JSF)* und die *JavaServer Pages Technologie (JSP)* zuständig. Wie auch bei den anderen Containern sind die Komponenten von den später real verfügbaren Klassen, Objekten und Seiten zu unterscheiden. So verwaltet der Web Container die JSP-Technolgie,

während der Ausführung eines Java-EE Anwendungsprogramms kann es jedoch mehrere auf JSP basierende Internetseiten geben, die durch den Web Container verwaltet werden. Es können auch mehrere Klassen als *Servlet*[3] definiert und beliebig viele Objekte dieser Klassen erzeugt werden. All diese Elemente (aufbauend auf der entsprechenden Technologie) dienen der Bearbeitung von Anfragen, die von einem Client gesendet werden, und der Erzeugung der jeweiligen Antworten in Form von Web-Seiten. Benötigen diese Elemente Funktionalitäten aus der Geschäftslogik, so können Sie auf Elemente des *EJB Containers* zugreifen.

- Der *EJB Container* (EJB steht für *Enterprise Java Beans*) ist für die EJB Technologie zuständig und verwaltet alle auf dieser Technologie basierenden Elemente. Dies sind Klassen, die als Enterprise Java Bean[4] definiert werden, und die von diesen Klassen erzeugten Objekte. EJB-Klassen sind vergleichbar mit Kontrollklassen (s. [SE 1]) und implementieren die Anwendungslogik eines Java EE-Anwendungsprogramms.

- Der *Applet Container* ist für die Verwaltung von *Applets* zuständig. Applets bieten für Web-Anwendungen graphische Oberflächen und können via HTTP mit den Elementen des Web Containers kommunizieren. In ihrer Funktionalität entsprechen sie daher GUI-Anwendungen[5]. Der Applet Container befindet sich auf dem Client-Rechner und besteht aus einem Browser mit zugehörigem Java-Plugin.

- Ein *Application Client* bezeichnet eine klassische Java GUI-Anwendung auf dem Client-Rechner. Der *Application Client Container* umfasst eine Menge von Java-Klassen, Bibliotheken und weitere Dateien, die benötigt werden um eine Java-Anwendung auszuführen. Auch diese Anwendungsprogramme können mit Elementen des Web Containers, des EJB Containers oder direkt mit einer Datenbank kommunizieren.

Für die genaue Beschreibung der einzelnen Bestandteile wird in der Java EE-Spezifikation häufig auf weitere Spezifikationen wie die *Servlet-Spezifikation* [Serv/Spec], die *JSP-Spezifikation* [JSP/Spec], die *Java SE-Spezifikation* [JSE/Spec] oder die *EJB-Spezifikation* [EJB/Spec] verwiesen.

Die Behandlung von persistenten Anwendungsobjekten spezifiziert der *Java Persistence* genannte Teil der Java EE Spezifikation [JPA/Spec]. Ein persistentes Anwendungsobjekt kann z.B. ein Objekt einer Klasse sein, das einen konkreten Kunden im System darstellt. Solche Objekte werden auch *Entities* genannt und sind vergleichbar mit Instanzen von Entitätsklassen (s. [SE 1]). Entities werden wie EJB-Objekte durch den EJB Container verwaltet.

Die wichtigsten Elemente innerhalb des Web Containers sind Servlets und JavaServer Pages. Ein Servlet (s. Kapitel 8) ist eine spezielle Java-Klasse, welche die Abarbeitung einer von einem Client gestellten Anfrage steuert. Fordert die Anfrage Ergebnisse an, die nur mit Wissen der Anwendungslogik geliefert werden können, ruft das Servlet dazu die entsprechenden EJB-Objekte

[3] Eine erste Beschreibung, worum es sich bei einem Servlet handelt, erfolgt etwas später in diesem Abschnitt.

[4] Eine Java Bean ist eine Java-Klasse mit bestimmten Eigenschaften. Es muss ein öffentlicher Standardkonstruktor vorhanden sein, die Klasse muss serialisierbar sein, und es müssen öffentliche Zugriffsmethoden für die Attribute der Klasse vorhanden sein (Getter- und Setter-Methoden).

[5] GUI = Graphical User Interface (graphische Benutzungsoberfläche)

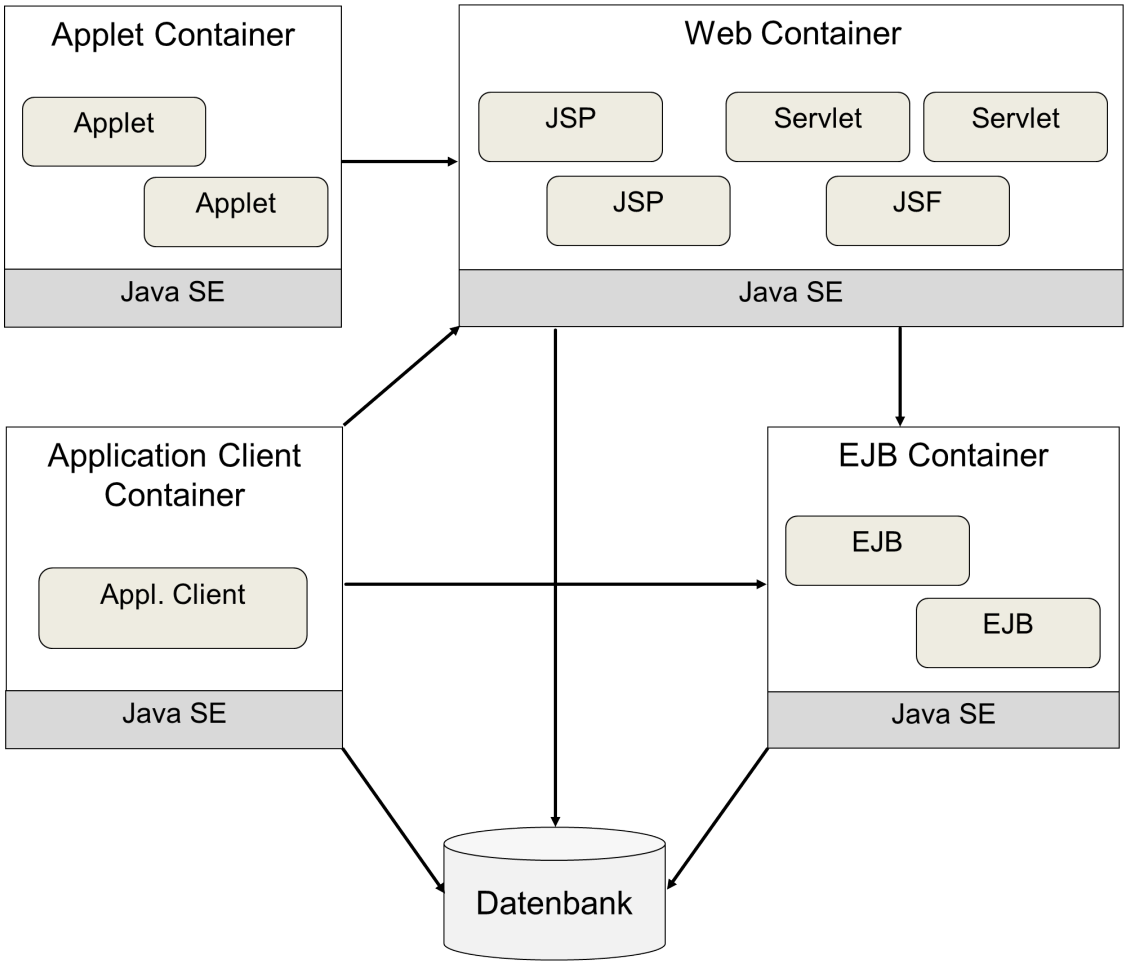

Abbildung 7.1.: Container in Java EE

(auch EJBs genannt) auf. Abschließend erstellt in der Regel eine JSP-Seite (s. Kapitel 9) aus dem dynamisch erzeugten Inhalt und dem statischen Inhalt ein Ausgabedokument, das dann der Web Container an den Client sendet. Der statische Anteil wird im Format des Ausgabedokuments (HTML, XML usw.) angegeben, zur Einbettung des dynamisch erzeugten Inhalts werden verschiedene Techniken angeboten.

Da die Java EE-Spezifikation frei verfügbar ist, gibt es mittlerweile mehrere Implementierungen, sprich Application Server, die teils kommerziell, teils als Open Source angeboten werden. Verschiedene Application Server können in den Punkten, die die Java EE-Spezifikation nicht behandelt oder offen lässt, voneinander abweichen. Dies betrifft insbesondere die Konfiguration der Server. Oracle bietet im Open-Source-Projekt *Glassfish* [GFish] eine Referenzimplementierung unter dem Namen *Oracle GlassFish Server Open Source Edition* (Version 4.0) an, die auch Teil des *Java EE 7 SDK* ist. SDK ist die Abkürzung für Software Development Kit. Ein *Software Development Kit* umfasst eine Menge von Werkzeugen, mit denen ein Anwendungsprogramm entwickelt werden kann, und beinhaltet oftmals auch Dokumentationen, die die Handhabung dieser Werkzeuge und Anwendungen beschreiben. EJB Container und Web Container können

von unterschiedlichen Herstellern stammen. So entwickelt z.B. das *Apache Jakarta Project* den Web Container *Tomcat* [Tom] ständig weiter. Ein anderer, sehr verbreiteter Application Server ist der *JBoss Application Server*, der von der Firma Red Hat entwickelt und betreut wird.

7.2. Verzeichnisstruktur der Java EE-Webschicht

Ein Java EE-Anwendungsprogramm besteht aus einer Vielzahl von Dateien gleichen und unterschiedlichen Typs. Der Dateityp unterscheidet sich z.B. für HTML-Dateien, XML-Dateien und Java-Dateien, und man kann den Dateityp an der jeweiligen Dateinamenserweiterung erkennen. Die Java EE-Spezifikation gibt vor, wie die Verzeichnisstruktur, in die alle benötigen Dateien eingeordnet werden, auszusehen hat. Wir betrachten zunächst lediglich die Verzeichnisstruktur der Web-Schicht, die Teil des Java EE-Servers ist und somit den Server-seitigen Teil der Benutzungsschnittstelle ausmacht. Die Web-Schicht enthält neben Dateien, die Servlet-Klassen oder Hilfsklassen enthalten[6] , JSP-Seiten, JSF-Seiten, sowie statische HTML- und Bilddateien. In Kurseinheit 6 wird die Verzeichnisstruktur der Anwendungslogik als zweite Schicht des Java EE-Servers vorgestellt, die Dateien mit EJB- und Entity-Klassendefinitionen enthält.

Damit der Web Container des Application Servers auf die Dateien der Web-Schicht zugreifen kann, müssen Entwickler darauf achten, die Verzeichnisstruktur für die Web-Schicht auf Ihrem Rechner so zu realisieren, wie durch die Java EE-Spezifikation vorgegeben wird. Bevor das entwickelte Java EE-Anwendungsprogramm, d.h. die realisierte Verzeichnisstruktur dieses Programms mit allen zugehörigen Dateien, auf den Server kopiert wird, wird die gesamte Verzeichnisstruktur in einem *Archiv* zusamengefasst. Ein Archiv ist eine Datei, die als Container für mehrere Dateien fungiert und diese ggf. komprimiert, um den Speicherbedarf zu reduzieren. Ein Java EE-Anwendungsprogramm, d.h. die Verzeichnisstruktur mit den entsprechenden Dateien, kann zu einem *JAR*-Archiv (*Java ARchive*) oder einem *WAR*-Archiv (*Web ARchive*) gepackt werden. Ein WAR-Archiv ist ein JAR-Archiv mit der Restriktion, die für ein Java EE-Anwendungsprogramm benötigte Verzeichnisstruktur einzuhalten. Beide Archivtypen basieren auf dem *ZIP*-Dateiformat (engl. zipper = Reißverschluss), das sich als Archivformat auf Rechnern etabliert hat. Um die verschiedenen Archivtypen voneinander unterscheiden zu können und deren Einsatzzweck zu definieren, ist für JAR-Archive die Dateinamenserweiterung „.jar“ und für WAR-Archive „.war“ vorgesehen. Der Vorteil eines Web Archives besteht in der einfachen Verteilung (Distribution) bzw. Weitergabe eines Anwendungsprogramms, besonders dann, wenn sein Verzeichnis sehr viele Dateien enthält.

Das Installieren, Konfigurieren und Ausführen eines Java EE-Anwendungsprogramms wird auch als *Deployment* bezeichnet. In Tabelle 7.1 ist die Verzeichnisstruktur[7] einer Java EE-Webanwendung angegeben. Es ist darauf zu achten, dass zwischen Groß- und Kleinbuchstaben

[6] Überlicherweise wird für jede Klasse eine Datei erzeugt, die den Namen der enthaltenen Klasse erhält.

[7] Während der Entwicklung wird meist nicht direkt innerhalb des Web-Anwendungsverzeichnisses des Web Container gearbeitet, sondern – meistens mittels einer integrierten Entwicklungsumgebung (kurz IDE für Integrated Development Environment) – eine eigene Verzeichnisstruktur verwendet, die durch die Entwicklungsumgebung in der Phase des Deployment in ein vom Application Server nutzbares JAR- oder WAR-Archiv umgewandelt wird.

unterschieden wird.

Verzeichnis bzw. Datei	Inhalt
`/Name`	Der Name der jeweiligen Web-Anwendung ist zugleich ihr Wurzelverzeichnis. Bei Distribution als WAR tritt der Name nicht in der Verzeichnisstruktur auf, sondern wird als Name für das Archiv herangezogen.
`/Name/WEB-INF/`	In diesem Verzeichnis befinden sich alle Dateien (inkl. der Klassendefinitionen), die nur aus der Anwendung heraus sichtbar sind. Ein Client (Browser) hat auf dieses Verzeichnis sowie seine Unterverzeichnisse im Gegensatz zu einem Servlet keinen direkten Zugriff.
`/Name/WEB-INF/classes/`	Hier sind die übersetzten Java-Klassen abgelegt.
`/Name/WEB-INF/lib/*.jar`	Hier werden Java-Bibliotheken abgelegt, die von der jeweiligen Web-Anwendung genutzt werden.
`/Name/WEB-INF/web.xml`	Dies ist die Konfigurationsdatei (*Deployment Descriptor*) der Web-Anwendung. Sie ist im *XML-Format* verfasst und seit der Java EE Version 5 optional.
`/Name/META-INF/`	Hier werden bei WAR-Archiven (wie bei JAR-Dateien) Metadaten über das Archiv und die darin enthaltenen Dateien gespeichert.
`/Name/*`	Alle anderen Verzeichnisse und Dateien sind direkt über HTTP adressierbar und wie statische Inhalte abrufbar, z.B. eine HTML-Seite.

Tabelle 7.1.: Verzeichnisstruktur einer Java EE-Web-Anwendung

7.3. Konfiguration einer Java EE Web-Anwendung

Eine *Konfiguration* ist eine Menge von Parametern, Eigenschaften und manchmal auch Regeln, die ein Anwendungsprogramm näher beschreiben bzw. die ein Anwendungsprogramm zu befolgen hat. Wird der Code eines Anwendungsprogramms auf einem Server abgelegt, so sollten bestimmte Konfigurationselemente nicht direkt im Programmcode geändert werden, insbesondere wenn sie mehrfach vorkommen. Dies gilt verstärkt, wenn es sich um ein kompiliertes Programm handelt, dessen Programmcode nicht zur Verfügung steht oder nach einer Änderung nicht mit dem kompilierten Programm übereinstimmt.

Oft sollen aber Parameter, Eigenschaften oder das Verhalten eines Anwendungsprogramms zur Laufzeit geändert werden können. All diese Daten werden in einer Konfigurationsdatei festgehalten. In Java EE entspricht der sogenannte *Deployment-Descriptor* einer solchen Konfigurationsdatei. Dies ist eine XML-Datei mit dem Dateinamen „web“ und der Dateinamenserweiterung „.xml“ (web.xml), die Java EE-spezifische Parameter, Eigenschaften und Regeln definiert.

Seit der Java EE Version 5 gibt es eine weiter Möglichkeit, Konfigurationen vorzunehmen: *Annotationen*. Beide Varianten werden in den nächsten beiden Abschnitten näher beschrieben. Durch die Verwendung von Annotationen wird der Deployment Descriptor in den meisten Fällen obsolet. Werden sowohl Annotationen als auch ein Deployment Descriptor zur Konfiguration einer Web-Anwendung verwendet, werden die durch Annotationen gesetzten Einstellungen durch entsprechende Einstellungen des Deployment Descriptors überschrieben und somit ersetzt.

7.3.1. Der Deployment Descriptor web.xml

Mit dem Deployment Descriptor können die folgenden Bereiche von Web-Anwendungen konfiguriert werden:

- ServletContext-Initialisierungsparameter
- Session-Konfiguration
- Servlet-Deklarationen
- Servlet-Mappings
- Filter-Definitionen
- Filter-Mappings
- Welcome File List
- Fehlerseiten

Der Deployment-Descriptor ist in XML verfasst. Alle Konfigurationen (d.h. die zu der jeweiligen Konfiguration gehörenden Tags) müssen sich zwischen dem Anfangs-Tag `<web-app>` und dem End-Tag `</web-app>` befinden. Zusätzlich befindet sich in der ersten Zeile der XML-Prolog (siehe Kurseinheit 1).

Den nach der Servlet-Spezifikation 3.1 definierten „Rahmen“ der Datei web.xml zeigt Codeausschnitt 7.1.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <web-app xmlns="http://java.sun.com/xml/ns/javaee"
4           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5           xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
6               http://java.sun.com/xml/ns/javaee/web-app_3_1.xsd"
7           version="3.1">
8
9      .
10     .
11     .
12
13 </web-app>
```

Listing 7.1: Aufbau des Deployment Descriptors web.xml

An Stelle der drei Punkte können verschiedene Bereiche konfiguriert werden. Exemplarisch werden die Servlet Mappings, die Servlet Deklarationen und die Welcome File List besprochen.

Das Verzeichnis /WEB-INF sowie seine Unterverzeichnisse sind für Client-Anfragen nicht erreichbar. Servlets[8], die für die Bearbeitung von Benutzeranfragen konzipiert sind, liegen im von außen nicht zugänglichen Verzeichnis /WEB-INF/classes. Für alle Servlets, die direkt von einem Client angesprochen werden sollen, müssen daher im Deployment Descriptor Abbildungsregeln von Anfrage-URLs auf Servlets eingetragen werden. Dafür sind für jedes Servlet zwei Einträge notwendig:

Codeausschnitt 7.2 zeigt den ersten Eintrag (im einfachsten Fall), der ein Servlet deklariert.

```
1 <servlet>
2     <servlet-name>MusterServlet</servlet-name>
3     <servlet-class>servletsammlung1796.MusterServlet</servlet-class>
4 </servlet>
```

Listing 7.2: Servlet-Deklaration im Deployment Descriptor web.xml

Über ¡servlet-name¿ wird dem Servlet ein beliebiger in der Web-Anwendung eindeutiger Name zugewiesen, hier „MusterServlet“. Mit ¡servlet-class¿ wird der Klassenname des Servlets inklusive Paketstruktur angegebene.

Der zweite Eintrag legt die Abbildungsregel fest:

```
1 <servlet-mapping>
2    <servlet-name>MusterServlet</servlet-name>
3    <url-pattern>/start</url-pattern>
4 </servlet-mapping>
```

Listing 7.3: Servlet Mapping Element im Deployment Descriptor `web.xml`

Mit `<servlet-name>` wird das entsprechende Servlet im ersten Eintrag angesprochen. Mit `<url-pattern>` wird die URL (hier: `/start`) angegeben, unter der das Servlet gefunden werden soll. Dabei bezieht sich die Pfadangabe auf das Wurzelverzeichnis der Web-Anwendung. Wenn die Web-Anwendung in dem Verzeichnis „Kurs1796Apps“ liegt und auf dem Host „www.FernUni-Hagen.de“ läuft, dann kann das gerade deklarierte Servlet über

`http://www.FernUni-Hagen.de/Kurs1796Apps/start`

erreicht werden. Im `<url-pattern>`-Eintrag kann als URL auch ein *URL-Muster* wie z.B. *.do angegeben werden. Dieses bedeutet, dass alle URLs der Web-Anwendung, die auf „.do“ enden, von diesem Servlet bearbeitet werden. Seit der Servlet-Spezifikation Version 2.5 dürfen auch mehrere `<url-pattern>`-Einträge in einem `<servlet-mapping>` stehen.

[8] Klasse, die als Servlet definiert wurde

Mit der Konfiguration der *Welcome File List* können Startseiten der Web-Anwendung festgelegt werden, die zurückgeliefert werden, wenn der Client keine explizite Startseite angefordert hat, sonder lediglich die URL `http://.../Webanwendung` angibt. Dann wird vom Web Container automatisch die erste verfügbare Ressource[9] in der Welcome File List ausgewählt. Um z.B. das oben deklarierte Servlet als Startseite festzulegen, genügt der in Codeausschnitt 7.4 dargestellte Eintrag in der Welcome File List[10].

```
1 <welcome-file-list>
2     <welcome-file>start</welcome-file>
3 </welcome-file-list>
```

Listing 7.4: Welcome File List Element im Deployment Descriptor web.xml

7.3.2. Konfiguration durch Annotationen

Durch *Annotationen* lassen sich in der Programmiersprache Java Meta-Informationen in den Quelltext einbeziehen. Dabei wird zwischen zwei Arten von Annotationen unterschieden, Annotationen, die die Semantik des Programmcodes ändern und Annotationen, die die Semantik nicht ändern (wie z.B. die `@Override` Annotation). Annotation, die die Semantik nicht beeinflussen, beinhalten Informationen über den Programmcode selber. Annotationen werden im Quelltext vor Klassen, Methoden oder Atrributen notiert und durch ein „@“ Zeichen als Annotation gekennzeichnet, auf das der Typ der Annotation folgt. Anschließend kann eine in runden Klammern eingeschlosse Liste von Parametern folgen, die durch Kommata getrennt werden. Jedem angegebenen Parameter kann innerhalb der Annotation ein Wert zugewiesen werden. Dies geschieht, indem zuerst der Parameter bezeichnet und danach, getrennt durch ein Gleichheitszeichen, der Parameterwert angegeben wird.

Durch den *Annotationstyp* wird festlegt, in welchem Kontext die Annotation benutzt werden darf (z.B. nur vor Klassen oder Methoden), welche Parameter (Name und Typ) angegeben werden können oder müssen, wie die Standardwerte für nicht angegebene Parameter lauten etc. Auf der Basis dieser Informationen nimmt der Compiler beim Übersetzen des Quellcodes auch eine Gültigkeitsprüfung vor. Abgesehen von den Standard-Annotationstypen aus dem Java-Paket `java.lang` (wie `Override`) müssen Annotationstypen – genau wie Klassen oder Interfaces – importiert werden, damit sie in einer Java-Datei genutzt werden können.

Die Annotationen werden vom Java-Compiler zusammen mit dem Quellcode verarbeitet, und die darin enthaltenen Meta-Informationen werden mit in die compilierten Java-Klassen eingebunden. Dabei haben diese Meta-Informationen zwar keinen direkten Einfluss auf den annotierten Code, sie werden jedoch zur Laufzeit von anderen Codeteilen (z.B. einem Framework) durch

[9] Eine Ressource ist alles, was über eine URI identifiziert werden kann (KE 1, Abschnitt 2.2).

[10] Im Normalfall genügt ein einziger Eintrag im Welcome File. Mehrere Einträge werden verwendet, falls verschiedene Unterverzeichnisse des Webanwendungs-Stammverzeichnisses per URL angesprochen werden können und in diesen Verzeichnissen jeweils unterschiedliche Startseiten liegen, z.B. „index.html“ in „/A“ und „index.jsp“ in „/B“.

Introspektion[11] ausgelesen. So kann z.B. der Web Container benötigte Meta-Informationen über Servlets direkt aus den Annotationen in Servlet-Klassen auslesen, statt sie aus einer separaten XML-Datei (Konfigurationsdatei) beziehen zu müssen. Indirekt können Annotationen also durchaus Einfluss auf das Verhalten einer Software haben.

Im Rahmen von Java EE gibt es zwei Kategorien von Annotationstypen: vordefinierte und selbstdefinierte Annotationstypen. Ein Application Server, der die Java EE-Spezifikation implementiert, umfasst eine ganze Reihe vordefinierter Annotationstypen, die direkt genutzt werden können und Java EE-spezifische Parameter zur Verfügung stellen. Darüber hinaus können Entwickler eigene Annotationstypen definieren, deren Parameterwerte durch Introspektion ausgelesen werden können. Vordefinierte Annotationen werden meist bei der Kompilierung ausgewertet, selbstdefinierte Annotationen zur Laufzeit eines Anwendungsprogramms.

In Codeausschnitt 7.5 ist ein Beispiel für die Verwendung der `Override`-Annotation aus dem Paket `java.lang` dargestellt.

```
1 public class MeineKlasse extends MeineOberklasse {
2    @Override
3    public boolean methode1(String param1) {
4       ...
5    }
6 }
```

Listing 7.5: Override Annotation

Diese Annotation hat keinerlei Auswirkungen auf das Programmverhalten, sodass sich das Programm mit und ohne die `Override`-Annotation identisch verhält. Durch die Annotation wird der Compiler veranlasst zu überprüfen, ob die Methode `methode1` der Unterklasse `MeineKlasse` die Methode `methode1` der Oberklasse `MeineOberklasse` überschreibt. Sollte es in der Oberklasse keine entsprechende Methode geben, gibt der Compiler beim Übersetzen eine Fehlermeldung aus. Diese Annotation dient also der Vermeidung von Programmierfehlern.

Im Rahmen dieses Kurses wird zur Konfiguration von Servlets lediglich die Annotation `@WebServlet` benötigt und besprochen[12]. Die `@WebServlet` Annotation deklariert eine Java-Klasse als Servlet. Die einzigen Parameter, für die Parameterwerte angegeben werden müssen, sind `value` und `urlPatterns`[13]. Ist kein Parameterwert für `value` angegeben, muss ein Parameterwert für `urlPatterns` angegeben werden und umgekehrt. Es dürfen aber nicht für beide Parameter zugleich Parameterwerte angegeben werden. Beiden Parametern kann als Wert nur eine nichtleere Liste von URLs zugewiesen werden, über die das Servlet angesprochen werden soll. Dabei bezieht sich die Pfadangabe auf das Wurzelverzeichnis der Web-Anwendung. Wenn kein anderer Parameter angegeben wird, sollte der Parameter `value` und ansonsten `urlPatterns` verwendet werden. Des Weiteren lässt sich über den optionalen Parameter `name` dem Servlet ein Name zuweisen.

[11] Introspektion bezeichnet in der Programmierung das Konzept, dass ein Programm seine eigene Struktur kennt und diese ggf. verändern kann.

[12] Eine vollständige Auflistung und Dokumentation aller Servlet-Annotationen findet sich in [Serv/Spec].

[13] Eine vollständige Auflistung und Dokumentation aller Parameter der `@WebServlet` Annotation findet sich unter [JEE/API] im Paket javax.servlet.annotation.

Die Codeausschnitte 7.6[14] und 7.7 stellen jeweils ein Beispiel für eine Klasse `MusterServlet` dar, welches durch die Annotation `@WebServlet` als Servlet deklariert wird. Wie die Servlet-Klasse genau aufgebaut sein muss, wird in Abschnitt 8.4 erläutert.

```
1  package kurs01796.ke3.servletbeispiele
2
3  import java.io.IOException;
4  import javax.servlet.ServletException;
5  import javax.servlet.annotation.WebServlet;
6  import javax.servlet.http.HttpServlet;
7  import javax.servlet.http.HttpServletRequest;
8  import javax.servlet.http.HttpServletResponse;
9
10 @WebServlet(value={"/start","/musterServlet"})
11 public class MusterServlet extends HttpServlet {
12    protected void doGet(HttpServletRequest request, HttpServletResponse
          response) throws ServletException, IOException {
13          ...
14    }
15    ...
16 }
```

Listing 7.6: Servlet Deklaration durch Annotation @WebServlet

```
1  package kurs01796.ke3.servletbeispiele
2
3  import java.io.IOException;
4  import javax.servlet.ServletException;
5  import javax.servlet.annotation.WebServlet;
6  import javax.servlet.http.HttpServlet;
7  import javax.servlet.http.HttpServletRequest;
8  import javax.servlet.http.HttpServletResponse;
9
10 @WebServlet(name="MusterServlet", urlPatterns={"/start","/musterServlet"})
11 public class MusterServlet extends HttpServlet {
12    protected void doGet(HttpServletRequest request, HttpServletResponse
          response) throws ServletException, IOException {
13          ...
14    }
15    ...
16 }
```

Listing 7.7: Servlet Deklaration durch Annotation @WebServlet mit mehreren Parametern

Wenn wieder davon ausgegangen wird, dass die Web-Anwendung in dem Verzeichnis „Kurs1796Apps“ liegt und auf dem Host „www.FernUni-Hagen.de“ läuft, kann das Servlet „MusterServlet“ in beiden Beispielen über die URLs

[14] Der Parameterbezeichner und das Gleicheitszeichen können weggelassen werden, wenn kein weiterer Parameter angegeben wird: `@WebServlet("/start","/musterServlet")`.

`http://www.FernUni-Hagen.de/Kurs1796Apps/start` und
`http://www.FernUni-Hagen.de/Kurs1796Apps/musterServlet`

erreicht werden. Während in Codeausschnitt 7.7 dem Servlet der Name „MusterServlet“ zugewiesen und daher der Parameter `urlPatterns` verwendet wird, wird in Codeausschnitt 7.6 als einziger Parameter `value` verwendet. Das Servlet erhält in Codeausschnitt 7.6 standardmäßig den voll qualifizierten Klassennamen als Namen: `... .servletbeispiele.MusterServlet`.