15. Java Enterprise Edition

15.1 Java Virtual Machine

15.1.1 Eigenschaften der Java Virtual Machine

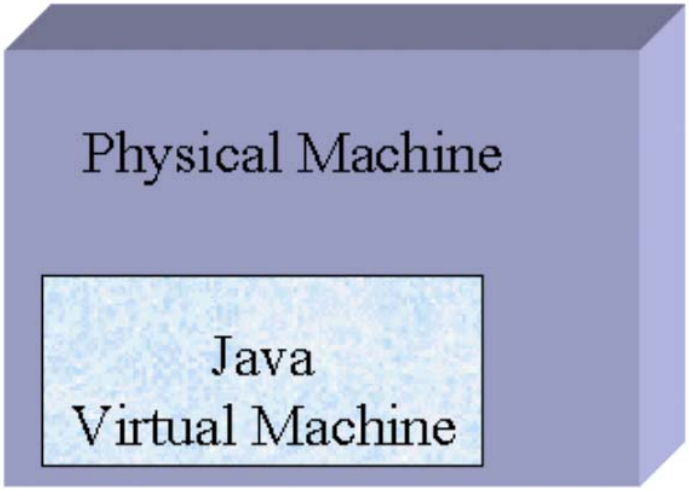


Abb. 15.1.1
Emulation der Java Hardware Architektur

Im Zusammenhang mit Java ist eine neue Hardware Prozessor Architektur entwickelt worden, vergleichbar zur System z, PowerPC oder x86 Architektur. Diese Architektur wird als Java Virtual Machine Architektur (JVMA) bezeichnet.

Wenn Java Quellcode kompiliert wird, entsteht Object Code. Dieser Objekt Code wird als „Byte Code“ bezeichnet.

Das Ausführungen eines Java Programms erfolgt im Prinzip auf speziellen Java-Prozessoren. Dies sind Mikroprozessoren, die Java-Bytecode als Maschinensprache verwenden. Sie wurden jedoch nie in größeren Stückzahlen gebaut. Statt dessen wird die Java Virtual Machine Architektur fast immer auf anderen Rechner Plattformen mit Hilfe einer Java Virtuellen Maschine (JVM) emuliert. Dies ist vergleichbar zur System z Emulation auf einem x86 Rechner mittels Hercules oder zPDT .

Jazelle DBX (Direct Bytecode eXecution) ermöglicht einigen ARM Processor Versionen die Ausführung von Java Bytecode in Hardware als Alternative zur normalen JVM Programm Ausführung. Die Dalvik virtuelle Maschine ist eine inkompatible Alternative zur JVM, die von Android für die Ausführung von Java Code benutzt wird.

Die Java Virtual Machine bietet neben der Platformunabhängigkeit auch einen Gewinn an Sicherheit. Eine JVM überwacht zur Laufzeit die Ausführung des Programms, verhindert also z. B., dass ein Programm über Arraygrenzen hinweg liest oder schreibt. Im speziellen Fall von Java fällt diese Überwachung sehr einfach aus, da Java keine Zeiger unterstützt. Somit werden Pufferüberläufe verhindert, die vor allem bei den in C oder C++ geschriebenen Programmen vorkommen können.

Um die Geschwindigkeit ("performance") der Programmausführung zu erhöhen, setzen die meisten JVM Implementierungen sogenannte JIT-Compiler ein, die unmittelbar während des Programmablaufs den Bytecode „Just In Time“ („Gerade zur rechten Zeit") in Maschinencode übersetzen. Eine Weiterentwicklung dieses Ansatzes, der Hotspot-Optimizer von Sun, behebt teilweise den Geschwindigkeitsnachteil der JVM, der hohe Speicherbedarf bleibt jedoch bestehen. Außerdem hat Java entwurfsbedingt einige Performance Nachteile, vor allem durch die automatische Speicherbereinigung (garbage collection). Einige JVM Implementierungen, zum Beispiel Insignia Jeode oder IBM J9, können diese Nachteile teilweise kompensieren.

BEA Jrockit, Sun Hotspot und IBM J9 sind die derzeitig führenden JVM Implementierungen.

Eine Java Anwendung, welche keine Probleme mit der begrenzten Hauptspeichergröße hat, läuft im 64 Bit Adressiermodus langsamer als im 31 Bit Adressiermodus !

Innerhalb einer JVM können mehrere Prozesse (Threads) ablaufen. Servlets machen beispielsweise hiervon Gebrauch.

Die JVM schottet die Threads vom Betriebssystem ab. Dies hat zur Folge, dass es nicht mehr möglich ist, mit systemeigenen Mitteln Java Prozesse zu kontrollieren. Die JVM stellt aber auch keine eigenen Funktionen zur Prozesskontrolle und -steuerung bereit. Somit können diese auch nicht von außen beendet werden, wenn sie aufgrund eines Fehlers das Gesamtsystem stören. Im Fehlerfall muss die gesamte JVM beendet werden. Viele JVM Implementierungen erlauben deshalb das direkte Abbilden (Mappen) von dedizierten Java-Threads auf Betriebssystem-Prozesse (native Threads).

Heute ist das Mappen von Java-Threads in der Regel die Default-Einstellung, d. h. nur in Ausnahmefällen wie dem Verwenden einer älteren JVM erfolgt das Thread-Management nur durch die JVM und nicht durch das Mapping auf Threads des Betriebssystems.

Die Isolation der Threads innerhalb der JVM untereinander ist unvollständig. Dies ist besonders kritisch, wenn mehrere Transaktionen als threads in der gleichen JVM verarbeitet werden. Hierauf wird später detailliert eingegangen.

15.1.2 Heap und Stack

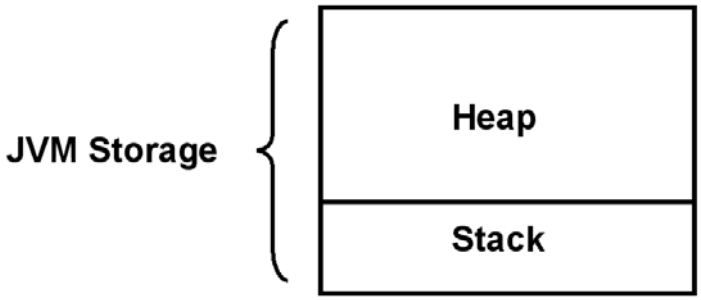


Abb. 15.1.2
Aufteilung in Heap und Stack

Der JVM Interpreter verwaltet einen Speicherbereich, der dem Anwendungsprogrammierer zugänglich ist, und der aus den beiden Teilen Heap und Stack besteht. Der Heap enthält unterschiedliche Arten von Information während der Programmausführung:

- **Lokale Variablen und Stacks für die aktiven Methoden, und**
- **Arrays und Objekte, die während der Programm Ausführung erzeugt werden.**

Der Heap existiert nur einmal pro Instanz der VM.

Der Heap dient zur Speicherung von dynamischen Informationen über konkrete Objekte. Für jedes Objekt sind dies sämtliche Instanzvariablen - sowohl die, die in seiner eigenen Klasse deklariert wurden, als auch die aus sämtlichen Vorfahren dieser Klasse.

Arrays und Objekte können nicht auf dem Stack gespeichert werden. Sie werden in dem Heap gespeichert.

Der `new`-Operator, der vom Anwendungsprogramm aufgerufen wird, um eine Variable auf dem Heap anzulegen, gibt dem Anwendungsprogramm eine Referenz auf die im Heap erzeugte dynamische Variable zurück. An welcher Stelle des Heaps die dynamische Variable angelegt wird, entscheidet nicht der Programmierer, sondern die virtuelle Maschine. Über diese Referenz kann man dann auf die dynamische Variable im Heap zugreifen.

Die Größe des Heaps ist beschränkt. Daher kann es zu einem Überlauf des Heaps kommen, wenn ständig nur Speicher angefordert und nichts zurückgegeben wird. Weiterhin wird mit zunehmendem Gebrauch der Heap zerstückelt, so dass der Fall eintreten kann, dass keine größeren Objekte mehr auf dem Heap angelegt werden können, obwohl in der Summe genügend freier Speicher vorhanden ist, aber eben nicht an einem Stück.

In Java werden Objekte im Heap nicht explizit freigegeben. Es wird vielmehr in unregelmäßigen Abständen durch die virtuelle Maschine der „Garbage Kollektor“ aufgerufen. Der Garbage Collector gibt den Speicherplatz, der nicht mehr referenziert wird, frei. Er ordnet ferner den Speicher neu (Defragmentierung), so dass auf dem Heap wieder größere homogene unbenutzte Speicherbereiche entstehen.

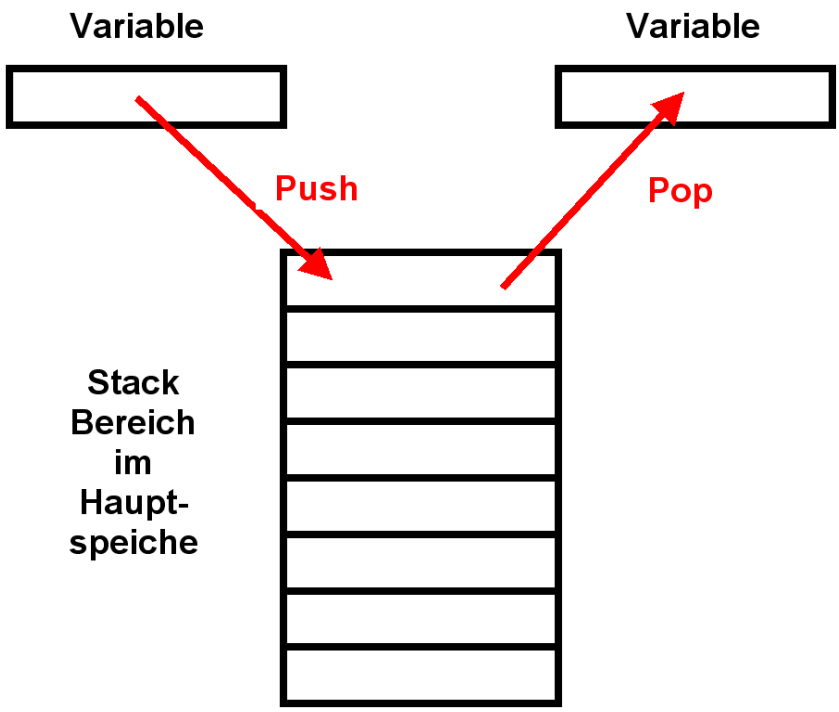


Abb. 15.1.3
Stack Operation

Ein Stack ist ein Container, in dem man nur an der Spitze Elemente ablegen oder von oben wieder wegnehmen kann. Es besteht keine Möglichkeit, ein Element aus der Mitte zu entnehmen oder ein Element in die Mitte einzufügen. Diese Abarbeitungsreihenfolge wird auch als LIF0-Prinzip (last in–first out) bezeichnet. Mit der Methode `push()` kann ein Element auf einen Stack abgelegt werden. Mit der Methode `pop()` kann das oberste Element vom Stack wieder entnommen werden.

Ein Stack kann aus Variablen einfacher Datentypen aufgebaut werden - oder im Falle von Klassen auch aus Referenzen auf Objekte. Sind die Referenzen, die auf einem Stack gespeichert werden, vom Typ Object, so kann jede beliebige Referenz auf dem Stack abgelegt werden.

Der Java-Stack speichert lokale Variablen innerhalb von Methodenaufrufen, sowie Operanden für arithmetische Operationen. Bei jedem Methodenaufruf wird ein Stack-Frame auf den Stack gepuscht, das Platz für die lokalen Variablen dieser Methode und einige Zusatzdaten bietet. Wird die Methode (durch `Return` oder durch eine Exception) verlassen, wird der zugehörige Frame wieder vom Stack genommen.

Die JVM unterhält eine getrennten Stack für jeden Thread.

15.1.3 Class Loader

In Java wird kein ausführbares Programm durch einen Linker erzeugt. Statt dessen werden benötigte Klassen zur Laufzeit durch die JVM nachgeladen. Diese verwendet hierfür eine Klasse `ClassLoader` :

```
public abstract class ClassLoader extends Object
```

Ein `ClassLoader` erhält durch den Aufruf seiner Methode `loadClass()` die Aufforderung, Klassen aus `.class`-Dateien oder `.jar`-Archiven vom lokalen Dateisystem laden. Um diese Dateien zu lokalisieren, wertet der System-ClassLoader die Umgebungsvariable `CLASSPATH` aus.

Ein Java Programm besteht aus einem losen Verbund von einzelnen `.class` Dateien, die bei Bedarf in die virtuelle Maschine geladen werden. Klassen, die eine `main()` Methode haben, können zum Starten einer Anwendung benutzt werden.

Jedes Klassen-Objekt enthält eine Reference zu dem ClassLoader der es definierte. Es existiert ein spezieller Bootstrap-ClassLoader für die Bootstrap-Klassen, und ein Erweiterungs-ClassLoader für die Klassen im lib/ext-Verzeichnis.

Der Java Development Kit (JDK) ist ein Subset des Software Development Kit (SDK). Der JDK ist zuständig für das Schreiben und Ausführen von Java Programmen. Der SDK enthält weitere Software, z.B. Debugger oder Dokumentation.

JDKs unterschiedlicher Hersteller befolgen grundsätzlich alle Java Specifikationen, können aber Funktionen enthalten, die in dem Java Standard nicht festgelegt sind. Beispiele sind Garbage Collection, Compilation Strategien und Optimierungsverfahren. Als Folge können bei der Portierung von Java Programmen Schwierigkeiten auftreten.

15.1.4 Java Runtime Environment

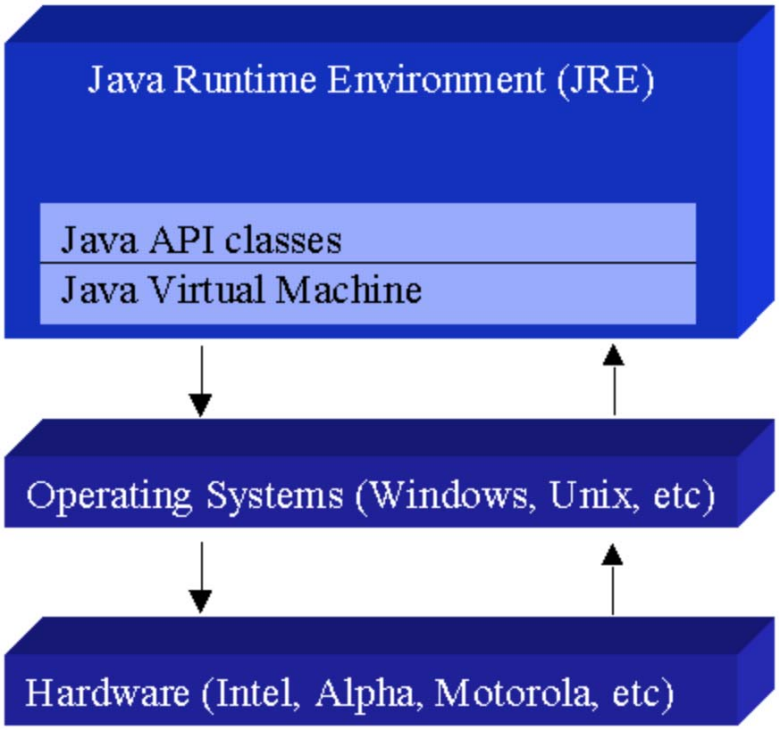


Abb. 15.1.4
Einbettung der JVM in die Java Runtime Umgebung

Die Java Virtual Machine ist Teil eines größeren Systems, dem Java Runtime Environment (JRE). Jedes Betriebssystem und jede Hardware Architektur arbeitet mit einem unterschiedlichen JRE. Das JRE besteht aus einem Satz von Base Klassen, welche eine JVM beinhalten wie auch eine Implementierung der Base Java API. Die JRE Implementierungen auf unterschiedlichen Plattformen ermöglichen die Portabilität von Java Programmen. Java Programme können auf einer bestimmten Plattform nur laufen, wenn dort ein JRE vorhanden ist.

Java arbeitet intern ausschließlich mit Unicode UTF-16 Encoding.

Bei der Default-Codierung werden von der JVM Unicode-Zeichen bis \u00FF so gut wie möglich in die Code-Tabelle des Betriebssystem abgebildet.

Dies muss z.B. beachtet werden, wenn ein Java Programm auf eine DB2 oder IMS Datenbank zugreift.

Die “z/OS Support for Unicode Conversion Services” konvertieren alphanumerische Daten zwischen UTF-8, UTF-16, ASCII und EBCDIC.

Dies kann kompliziert werden, weil die JVM selbst häufig in C/C++ implementiert ist, und auf ASCII Daten zugreift.

15.1.5 JVM unter z/OS und zLinux

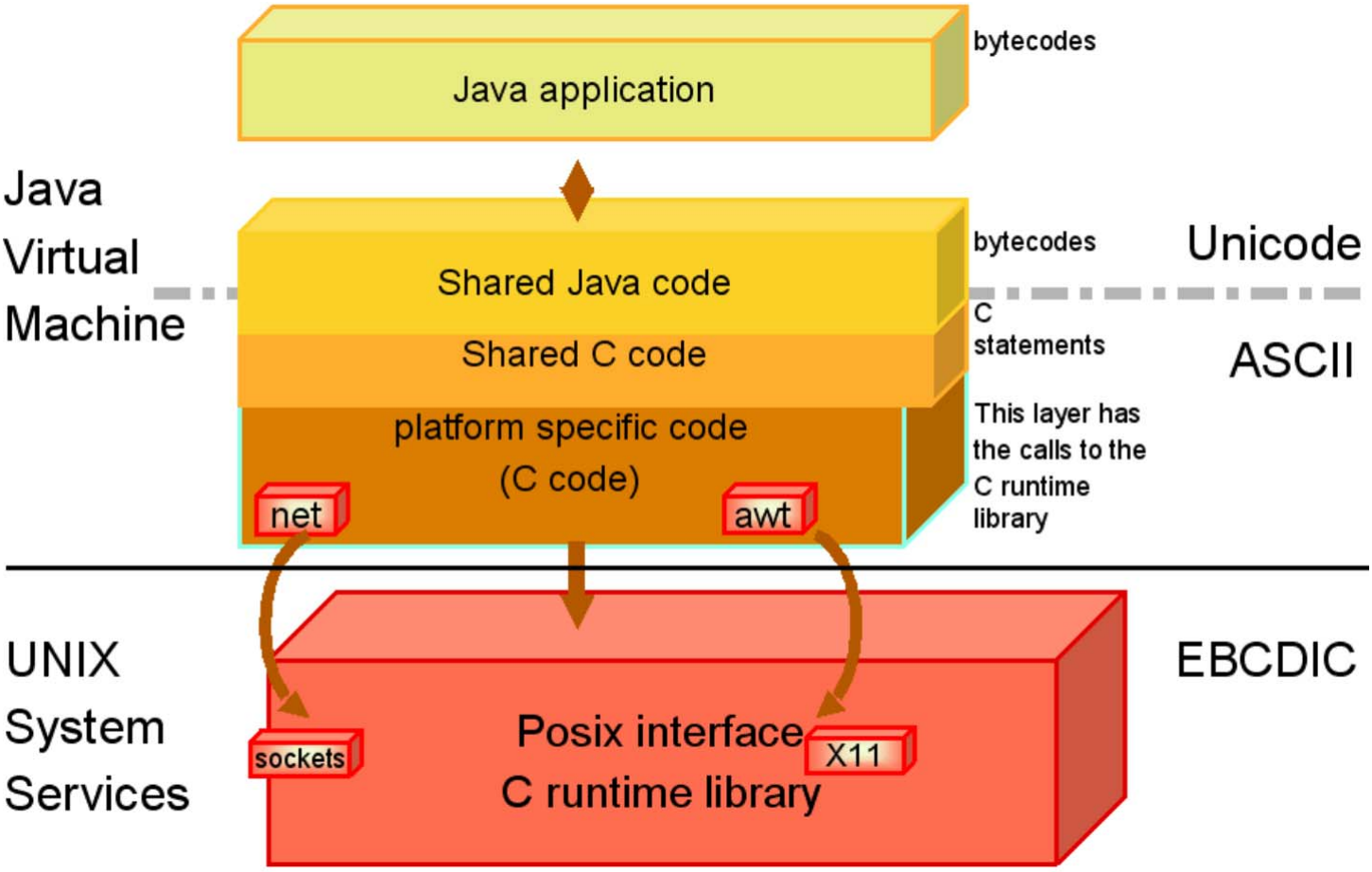


Abb. 15.1.5
Mainframe Implementierung der JVM

Die JVM ist auf den meisten Plattformen teilweise in C/C++ implementiert, so auch unter zLinux. Die zLinux JVM geht davon aus, dass alle Daten in ASCII gespeichert sind; der Zugriff auf EBCDIC Daten erfordert zusätzliche Maßnahmen.

Unter z/OS ist es komplizierter. Die JVM ist als Teil der Unix System Services (USS) (Abschnitt 17.1.2) implementiert und enthält aus Kompatibilitätsgründen ebenfalls ASCII Support. Die allermeisten z/OS Daten sind jedoch in EBCDIC gespeichert. Die JVM bemüht sich, bei einer Kommunikation mit JVMs auf anderen Plattformen diesen Unterschied weitgehend unsichtbar zu machen.

Wird die Environment Variable IBM_JAVA_ENABLE_ASCII_FILETAG definiert, und File Encoding für eine von z/OS unterstützte ASCII Code Page spezifiziert, dann werden alle neu angelegten Files mit einem "Coded Character Set Identifier" (CCSID) Tag versehen, der dieser Code Page entspricht. CCSID ist eine 16-bit Ziffer, die das Encoding spezifiziert.

15.1.6 Java Programmausführung

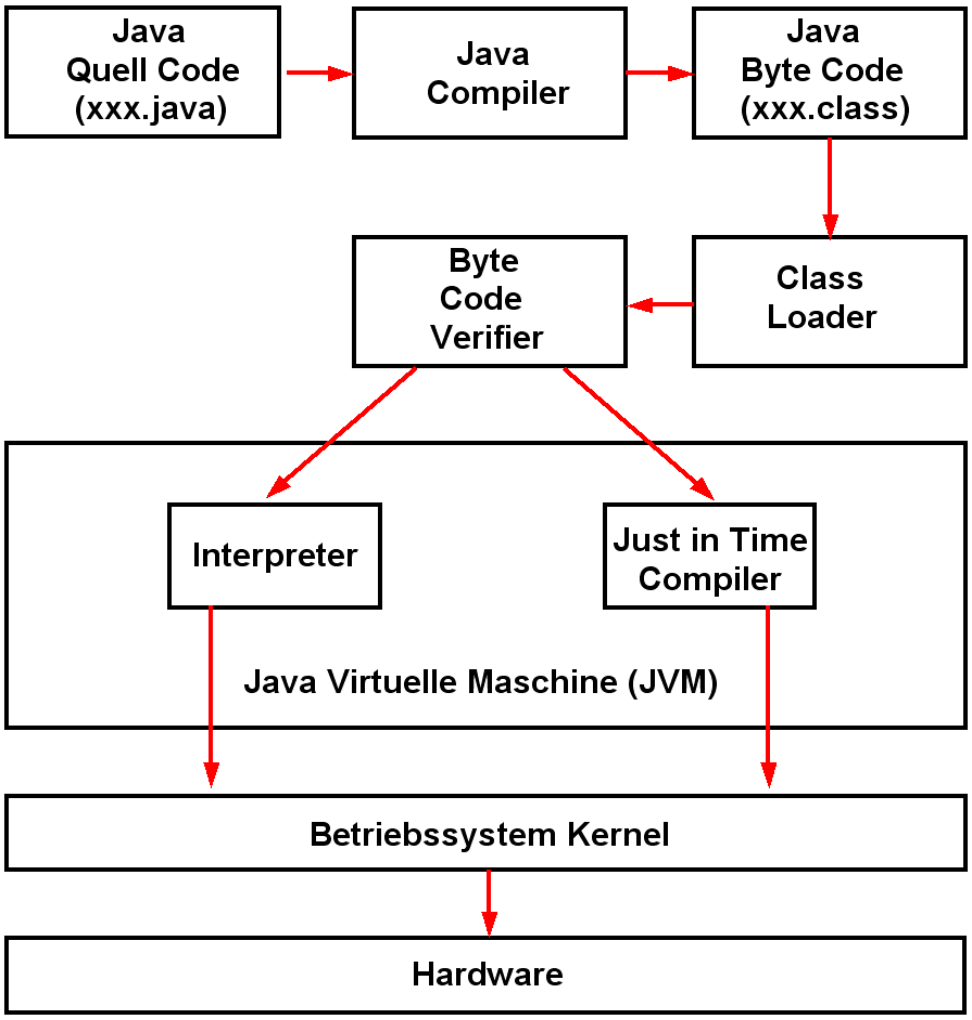


Abb. 15.1.6
Grundsätzliche Java Programm Ausführung

Der Java-Compiler übersetzt Java Quellcode in Java-Objekt-Code (universell als Java Byte Code bezeichnet).

Der kompilierte Code wird mit Hilfe eines Class Loaders und eines Byte-Code Verifiers in die JVM zur Ausführung geladen.

Der Code wird mit Hilfe eines Interpreters oder eines Just-in-Time-Compiler ausgeführt.

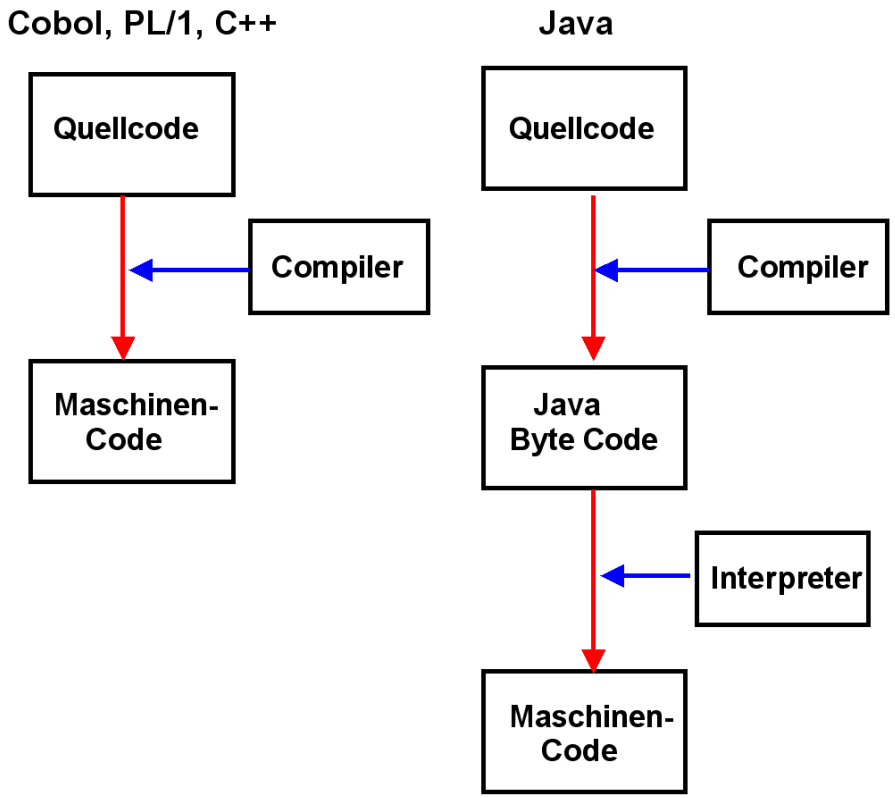


Abb. 15.1.7
Optimising und Just in Time Compiler

Cobol, PL/1 oder C/C++ Quell Code wird durch einen optimierenden Compiler (optimising compiler) in Object Code übersetzt, aus dem nach dem Linking und Loading ausführbarer Maschinencode entsteht.

Bei Java erzeugt der Compiler statt dessen Plattform-unabhängigen Byte Code, der anschließend in einer JVM durch einen Interpreter oder Just-in-Time Compiler ausgeführt wird.

Die JVM selbst ist plattformabhängig. Sie ist typischerweise in C++ implementiert.

Optimizing Compiler werden für Sprachen wie Cobol, PL/1, C++ eingesetzt; sie erzeugen für die Ausführung besonders schnellen Maschinencode. Interpreter und Just in Time Compiler (JIT) sind in der Regel langsamer.

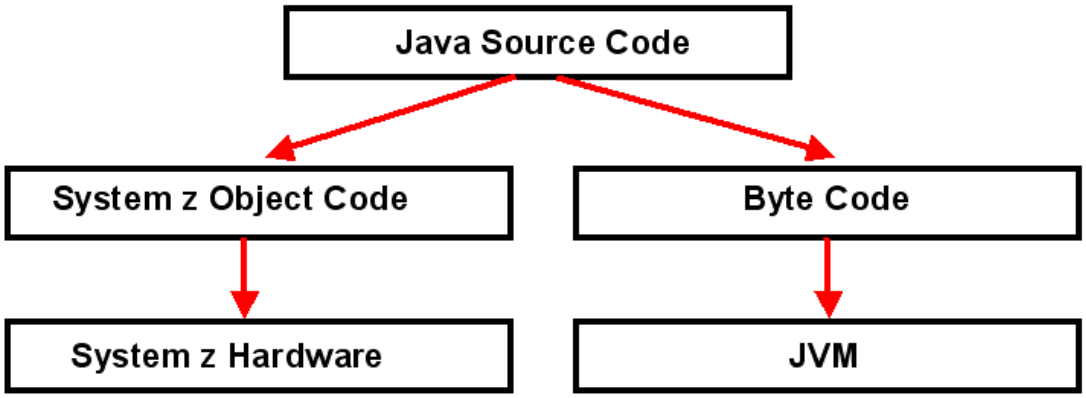


Abb. 15.1.8
z/OS Java Ausführungsalternativen

Unter System z ist die Ausführung sowohl als System z Object Code als auch als Byte Code möglich.

The z/OS High Performance Java Compiler hat die Option, System z Object Code (binaries) an Stelle von Java Byte Code zu generieren. Der System z Object Code kann dann direkt in den Hauptspeicher ohne Benutzung einer JVM gelinked und geladen werden.

Der übersetzte Objekt Code ist nicht mehr portierbar, hat aber eine kürzere Ausführungszeit. Es ist möglich, im Compiler Durchlauf gleichzeitig Byte Code zu erzeugen, der dann auch in anderen Umgebungen ausgeführt werden kann.

15.1.7 Java Anwendungen unter z/OS

Eines der wichtigsten z/OS Funktionen für Java ist die Verfügbarkeit von Record oriented Files (Data Sets) zusätzlich zu unstrukturierten Dateien.

JRIO ist eine Klassenbibliothek, ähnlich zu java.io. Während java.io Byte-orientierte oder feldorientierte Zugriffe auf Dateien ermöglicht, bietet JRIO einen Record-orientierten Zugang. Sie können mit JRIO auf VSAM-Datensätze und non-VSAM-Datensätze (PDS oder sequentiell) und HFS-Dateien zuzugreifen:

- **VSAM-Datensätze (nur KSDS)**
- **Non-VSAM Record orientierte Datensätze**
- **System Katalog**
- **Partitionierted Data Set (PDS) Directory**
- **DDNAME Unterstützung**

In VSAM KSDS können Sie auf Datensätze in Entry Sequence Reihenfolge oder durch primäre eindeutigen Schlüssel zugreifen. JRIO bietet indizierten I/O-Zugriff auf Datensätze innerhalb eines VSAM KSDS mit z/OS nativer Unterstützung.

Die Java Record I/O Funktionalität ist als Teil des IBM JZOS Batch Toolkits verfügbar.

Die meisten Java Anwendungen laufen unter z/OS innerhalb eines Application Servers wie dem CICS Transaction Server oder dem WebSphere Application Server (WAS). Eine Alternative sind Java Stand-alone Anwendungen.

Wir definieren eine Java Stand-alone Anwendung als eine Anwendung, die nicht innerhalb eines Applikationsservers läuft. Ein Java Stand-alone Anwendung verfügt über ein Main-Methode, die ähnlich wie der Main entry Point eines traditionellen COBOL oder PL/1 Lademoduls arbeitet. Die Main Method ist die erste Methode, die ausgeführt wird wenn die Anwendung gestartet wird. Sie kann Parameter enthalten, die der Anwendung übergeben werden.

Ein Java Stand-alone Anwendung läuft in einer eigenen Java Virtual Machine (JVM). Diese muss gestartet werden, ehe die ersten Befehle der Anwendung ausgeführt werden.

Ein Java Stand-alone Anwendung ist autonom und benötigt einen Trigger um gestartet zu werden. Dieser Trigger kann sein:

- **Jemand gibt manuell einen Befehl auf der Unix System Services Command Line ein, um eine JVM mit der Anwendung zu starten.**
- **Ein Job Scheduler startet ein JCL Script, das eine JVM mit der Anwendung startet.**

15.1.8 Trennung zwischen Entwicklung und Produktion

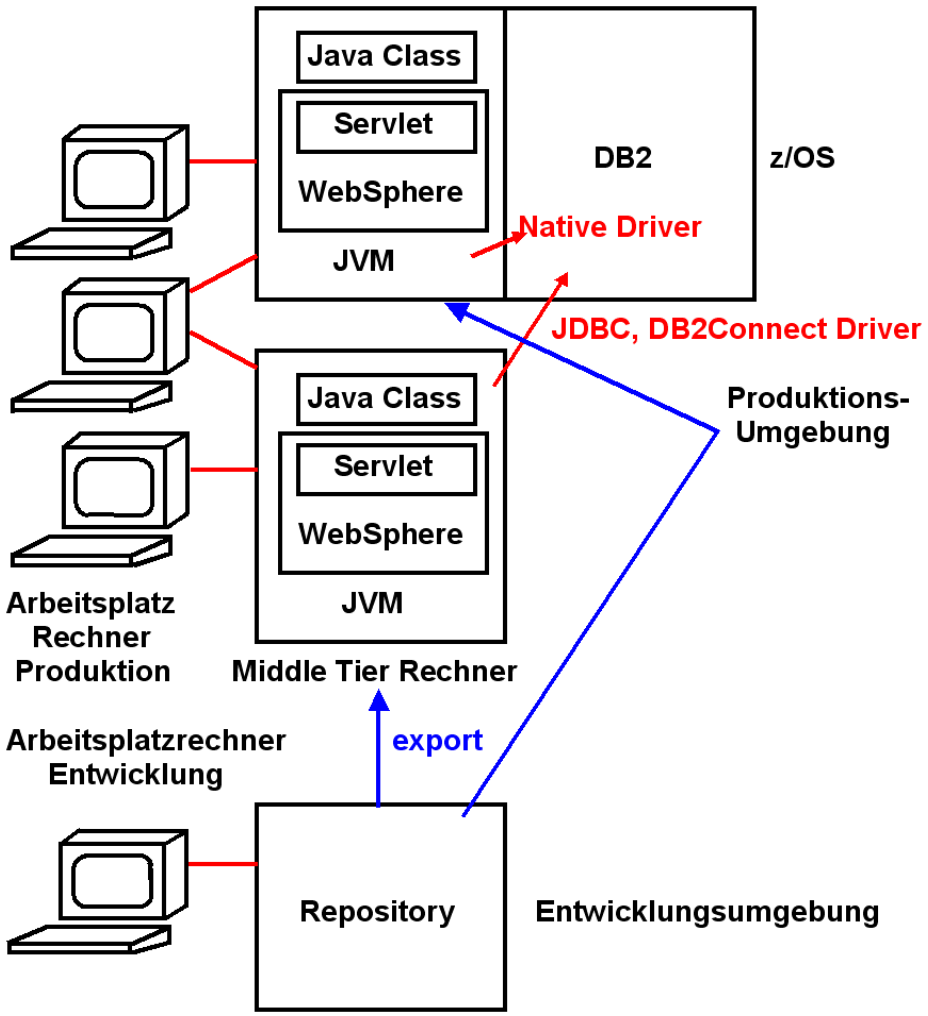


Abb. 15.1.9
Entwicklungsumgebung für neue Anwendungen

Die Entwicklung neuer Java Anwendungen kann mit Hilfe des JDK erfolgen. Sie erfolgt jedoch typischerweise in einer Entwicklungsumgebung (Integrated Development Environment, IDE).

Die Ausführung einer neu entwickelten Java Anwendung erfolgt häufig auf einem Web Application Server. Dieser befindet sich

- **entweder auf einem Middle Tier Server, der mit dem Mainframe verbunden ist, oder**
- **unmittelbar auf dem Mainframe Rechner**

Auf dem Mainframe läuft der Web Application Server entweder unter zLinux oder unter z/OS Unix System Services.

15.1.9 Integrated Development Environment (IDE)

Entwicklungsumgebungen sind Software Produkte, die von viele Herstellern erhältlich sind.

Weit verbreitet sind Eclipse , NetBeans, und IntelliJ IDEA

Eclipse und NetBeans sind Open Source. Eclipse kann heruntergeladen werden von www.eclipse.org . Ca. 350 MByte.

Eclipse (wie auch NetBeans und IntelliJ IDEA) ist sehr leistungsfähig und sehr beliebt, erfordert aber einen nicht unerheblichen Lernaufwand.

Eclipse hat eine sehr offene Architektur, und damit leichte Erweiterungsmöglichkeiten, die als Plug-ins bezeichnet werden.

Das Eclipse WebSphere Rational Application Developer für System z (RDz) Plug-in ist für z/OS Entwicklungen die wichtigste moderne Entwicklungsumgebung. RDz wurde wiederholt von IBM umbenannt: WSED → WDz → RDz, blieb aber im Wesentlichen das gleiche Software-Produkt.

Eclipse wurde ursprünglich für Java Programmierer entwickelt. Spezifisch das RDz Plug-in unterstützt aber auch Entwicklungen in COBOL, PL/I, C/C++ und Assembler.

15.2 Servlets

15.2.1 Statische HTML Seiten

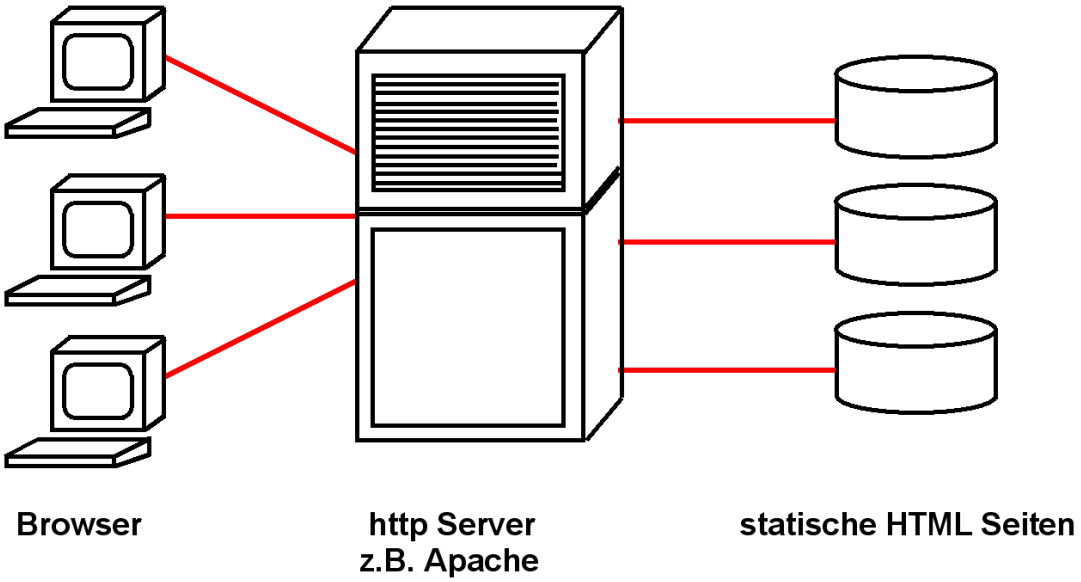


Abb. 15.2.1
Apache ist der am weitesten verbreitete http Server

Statische HTML Seiten sind der einfachste Fall einer Client/Server Anwendung. Der Klient (Browser) ruft eine (statische) HTML Seite auf, die unmodifiziert an den Klienten zurückgespielt wird. Die aus den statischen HTML Seiten bestehende Datensammlung wird lediglich offline geändert (z.B. durch das Hochladen mittels FTP). Der sehr große Vorteil: Es gibt keinerlei Datenintegritätsprobleme.

Es wird angenommen, dass 90% aller Web Zugriffe auf statische HTML Seiten erfolgen.

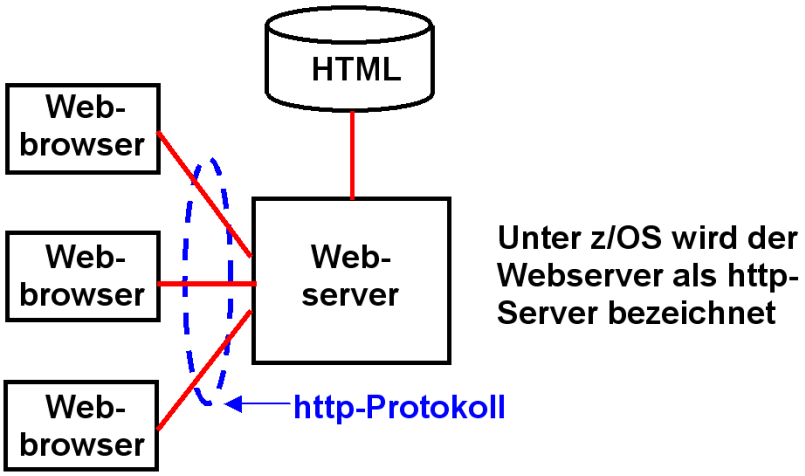


Abb. 15.2.2
http ist eines der am weitesten verbreiteten Internet Protokolle

HTML Seiten werden zum/vom Web Browser mit Hilfe des HTTP Protokolls übertragen. HTTP wird auch als „Web RPC“ betrachtet. Wie der eigentliche RPC ist HTTP zustandslos: Request/Response. Keine Session. HTTP erlaubt die Übertragung selbstbeschreibender Daten. Bei jeder Verbindungsaufnahme müssen Datenformate neu ausgehandelt werden.

HTTP ist eines von vielen Schicht 5 Übertragungsprotokollen. Beispiele für Schicht 5 Protokolle sind: Telnet, FTP, 3270, HTTP, SOAP, IIOP, RMI/JRMP, ...

Ein Web-Server unter z/OS, wie der HTTP Server unter Unix System Services, arbeitet ähnlich wie ein Web-Server auf anderen Plattformen. Der Benutzer sendet eine HTTP-Anfrage an den z/OS HTTP Server, um eine bestimmte Datei zu erhalten. Der HTTP Server ruft die Datei von seinem Datei-Repository ab, und schickt sie an den Benutzer, zusammen mit Informationen über die Datei (z. B. MIME-Typ und Größe) in dem HTTP-Header.

Der z/OS HTTP Server unterscheidet sich von anderen Web-Servern. Da z/OS Dateien im EBCDIC Format codiert sind, müssen z/OS Dokumente zunächst in das ASCII-Format konvertiert werden, ehe sie über das Internet an einen Endbenutzer geschickt werden. Binäre Dokumente und Bilder müssen nicht konvertiert werden.

Der z/OS HTTP Server führt diese Konvertierungen durch und erspart damit dem Programmierer diesen Schritt.

Für das Hochladen von Dokumenten verwendet der Programmierer normalerweise FTP. Dabei gibt der Programmierer ASCII als FTP-Transport-Format an. Der z/OS http Server konvertiert das Format dann automatisch in EBCDIC. Wenn „binary“ als FTP-Transport-Format angegeben wird, wird die Datei nicht konvertiert.

Der zLinux HTTP-Server hat diese Probleme nicht, da zLinux alle Daten im ASCII-Format verarbeitet und in der Regel kein EBCDIC benutzt.

Beim Transfer von Daten zwischen zwei Rechnern kann der Java Programmierer das Encoding der Daten spezifizieren.

Es wird behauptet, dass mehr als 50% aller betriebswirtschaftlich relevanter Daten, auf die über das WWW zugegriffen wird, im EBCDIC Format auf Mainframes gespeichert sind.

15.2.2 HTML Forms

Beim Aufruf von statischen HTML Seiten sendet der Browser lediglich eine URL an den Web Server. Für die Nutzung des Browsers und WWW als Client Server System ist es erforderlich, weitere Daten vom Klienten an den Server zu senden. Dies geschieht mittels „Form Data“. Weiterhin muss der Server eine Anfrage beantworten können. Bei der Nutzung von Java geschieht dies in der Regel mit Hilfe von Servlets und Java Server Pages (JSP).

HTML Forms sind ein einfache Werkzeuge, mit dem ein Benutzer Daten mit Hilfe des HTTP-Protokolls an einen Server schicken kann. HTML Forms erlauben es dem Browser, „Form Data“ an den Server zu senden,

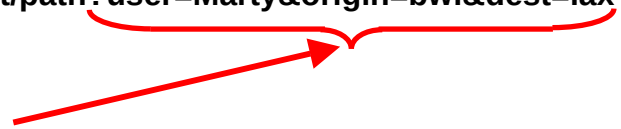
Ein HTML-Form besteht aus einem Code-Block, der mit dem <FORM> Tag anfängt und dem </FORM> Tag aufhört. Eine HTML-Seite kann mehrere Forms enthalten.

Der FORM Tag spezifiziert:

- **Die zu benutzende HTTP-Methode. In den meisten Fällen ist dies POST; die Daten werden innerhalb des Bodys der Nachricht übertragen.**
- **Die Action. Dies ist meistens die URL, es kann aber auch die Action mit ihrem Namen angegeben werden.**
- **Der Typ der MIME-Enkodierung der Daten in der FORM. Der Default ist "application/x-www-form-encoded".**

Wenn Sie jemals eine Web-Suchmaschine verwendet haben, einen on-line Buchladen besuchten, Aktienkurse on-line verfolgten oder ein Quote für den Preis eines Flugtickets abfragten, haben Sie wahrscheinlich eine merkwürdig aussehenden URLs gesehen, wie

Dies ist ein Beispiel für eine GET Anfrage
http://host/path?user=Marty&origin=bwi&dest=lax



Dieser Teil wird als „Form Data“ bezeichnet und ist ein häufig benutztes Verfahren, um Daten von einer gesendeten Webseite an ein Server-seitiges Programm zu übergeben.

Daten zusätzlich zu einer URL sind Form Data. Sie werden in der Regel von einem Browser an einen Webserver in einem von zwei Formaten übertragen.

- **Für GET-Anfragen werden die Form Daten an das Ende der URL nach einem Fragezeichen angebracht, siehe obiges Beispiel.**
- **Für POST-Anfragen werden die Form Daten an den Server in die http Nachricht eingebettet.**

Für alle außer sehr einfachen Anfragen wird die POST-Methode verwendet.

Abb. 15.2.3
Logon Screen, generiert durch eine HTML Form

Die hier dargestellte Wiedergabe auf einer HTML Seite erwartet eine Eingabe seitens des Benutzers in den hiefür vorgesehenen Feldern.

Diese Darstellung wird mit Hilfe von Form Tags innerhalb des HTML Codes programmiert.

Folgend ist eine HTML Seite dargestellt, die vom Server an den Browser gesendet wurde. Nachdem der Benutzer die beiden Felder ausgefüllt hat und den „Submit“ Button aktiviert, wird der Inhalt der beiden Felder, zusammen mit der Action, als Form Data an den Server gesendet.

```
<HTML>
<HEAD><TITLE> Login </TITLE> </HEAD>
<BODY>
<H2>Login to Secure Site</H2>

< FORM METHOD=POST
ACTION="http://abc.de/servlet/HelloWorld.servlet" >

Username: <INPUT TYPE="TEXT" NAME="username"
SIZE="25"><BR>
Password: <INPUT TYPE="PASSWORD"
NAME="password" SIZE="25"><P>

<INPUT TYPE="SUBMIT" VALUE="Submit">
<INPUT TYPE="RESET" VALUE="Clear">
</FORM>

</BODY> </HTML>
```

Abb. 15.2.4
Form Tag zur Erzeugung des Logon Screens

Ein HTML-Form besteht aus einem Code-Block, der mit dem <FORM> Tag anfängt und dem </FORM> Tag aufhört. Eine HTML-Seite kann mehrere Forms enthalten.

Der FORM Tag spezifiziert:

- **Die zu benutzende HTTP-Methode. Hier ist dies POST; die Daten werden innerhalb des Bodys der http Nachricht übertragen.**
- **Die Action. Dies ist meistens die URL, es kann aber auch die Action mit ihrem Namen angegeben werden.**
- **Der Typ der MIME-Enkodierung der Daten in der FORM. Der Default ist "application/x-www-form-encoded".**

Wenn der Server die Nachricht mit den Form Date empfängt, wird das Programm xyz.servlet im Verzeichnis abc.de/servlet aufgerufen, wobei die beiden Variablen username und password übergeben werden.

Multipurpose Internet Mail Extensions (MIME) ist ein Standard, der die Struktur und den Aufbau von E-Mails und anderen Internetnachrichten festlegt.

15.2.3 Dynamischer WEB Seiten Inhalt

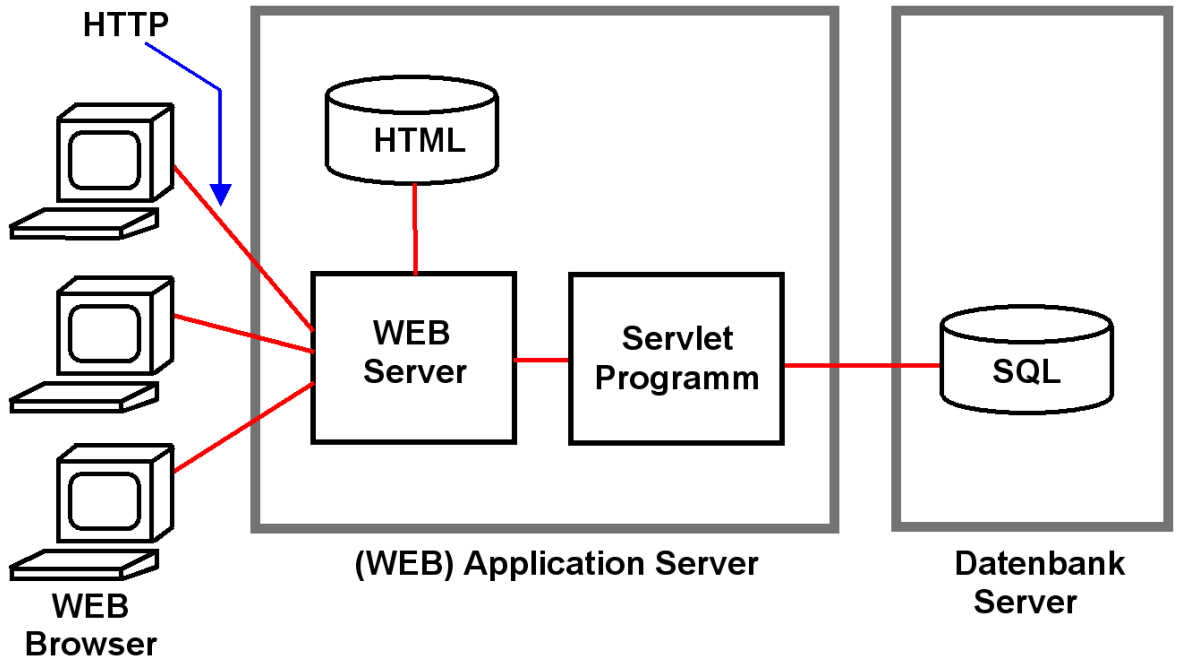


Abb. 15.2.5
Auswertung des Form Tags durch ein Servlet

Damit der Server mit den übertragenen Form Data etwas anfangen kann, enthält die übertragene Nachricht die Angabe eines Programmnamens, der mit dem Parameter ACTION spezifiziert wird. Das Programm wird aufgerufen und die übertragenen Daten werden verarbeitet. Zur Erstellung einer Antwort kann das Programm z.B. Daten aus einer z/OS DB2 Datenbank auslesen, um eine dynamische HTML Seite zu erstellen.

Der HTML Code `<FORM METHOD=POST ACTION="/servlet/HelloWorld">`
ruft ein Java Servlet mit dem Namen HelloWorld auf.

Ein Servlet Tag , z.B. `<SERVLET CODE="HelloServlet"></SERVLET>`
in einer HTML-Seite bewirkt das Ausführen eines Servlets auf dem Server.

Java Servlets sind normale Java-Klassen, die auf einem Server innerhalb einer standardisierten Laufzeit-Umgebung, der "Servlet Engine" oder des "Servlet Containers", ablaufen. Der Servlet Container beinhaltet eine normale Java Virtuelle Maschine.

- **Servlets sind vollwertige Java-Programme; sie verfügen über alle Java APIs, einschließlich JDBC (Java Data Base Connectivity) und SQLJ. Ein Applet kann im Gegensatz dazu auf keine Server-seitigen Daten zugreifen.**
- **Im Gegensatz zu CGI erfordert das Java Servlet nur "Light Weight Context Switches" implementiert über Java Threads. Daraus resultiert ein deutlich besseres Leistungsverhalten.**
- **Da das Servlet im Hauptspeicher verbleibt, können Verbindungen (Connections) zur Datenbank offen gehalten werden. Ein Servlet kann einen gemeinsamen Vorrat an Datenbankverbindungen verwalten, und diese je nach Bedarf einzelnen gleichzeitig ausgeführten Anwendungsprogrammen zuordnen.**
- **Leistungsfähiges Fehler- und Type-Checking.**

15.2.4 Servlet Container

Servlets laufen in einer Servlet-spezifischen Laufzeitumgebung, die auch als Container oder Servlet Engine bezeichnet wird. Der Servlet Container besteht im Wesentlichen aus einer Reihe von spezifischen Java Klassen, die innerhalb der gleichen JVM wie die Servlets untergebracht sind. Eine Servlet Klasse erbt mit Hilfe von „extends HttpServlet“ die Eigenschaften dieser Container Klassen.

Der Servlet Container verbessert u.a. die Servlet-Ausführungszeit und stellt dem Programmierer vorgefertigte Strukturen zur Verfügung. Servlet Container haben keine Transaktions-, Persistence- und Sicherheitseigenschaften. Ein Servlet Container ist ein Programm, das Requests für Servlets und Java Server Pages (JSP) behandelt. Der Servlet Container ist verantwortlich für:

- **Erstellung von Servlet-Instanzen,**
- **Initialisierung von Servlets,**
- **Dispatching von Requests,**
- **Verwaltung des Servlet-Kontextes für die Nutzung durch die Web-Anwendungen.**

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

public
class HalloWeltServlet extends HttpServlet                    ← Vererbung der Servlet Klassen
{

   public final static String message = "<html>\n" +
                                "<head><title>Hallo Welt</title></head>\n" +
                                "<body>\n" +
                                "<h1>Hallo Welt</h1>\n" +
                                "</body></html>\n";            } Dies ist eine Java Variable mit dem Namen „message“ .
                                                               } message hat die Form eines html Programms (why not ?) .

   public void init()
   {
      System.out.println("In HalloWeltServlet init");
   }

   public void destroy()
   {
      System.out.println("In HalloWeltServlet destroy");
   }                                                           } Jedes Servlet verfügt über die Methoden init, destroy und service.

   public void service(ServletRequest req, ServletResponse res)
   throws ServletException, IOException                        } Die Methode service ist für die Bearbeitung des Servlet Aufrufs zuständig.
   {
      PrintWriter  out = res.getWriter();                      } Der in message enthaltene HTML Code wird an den Browser gesendet
      out.println(message);
   }
}
```

Abb. 15.2.6
Beispiel: HalloWeltServletJava

15.2.5 Java Server Pages

Java Server Pages (JSP) sind in der Java Programmiersprache geschrieben. Eine JSP ist in Wirklichkeit eine andere Darstellungsform eines Servlets.

JSPs benutzen XML-artige Tags und Scriplets um die Logik zu kapseln, die den Inhalt der Seite generiert.

Alternativ kann die Anwendungslogik woanders liegen, und die Java Server Page greift hierauf mit den Tags und Scriplets zu.

Dies ermöglicht eine Trennung der HTML Seiten-Logik vom Seitenentwurf und der Seitenwiedergabe.

Sie können den folgenden Text in einer Datei mit der .jsp extension im JSP directory speichern und mit einem Browser ansehen:

```
<html>
<head>
<title>JSP Example </title>
</head>
<body>
Hello! The time is now <%= new java.util.Date() %>
</body>
</html>
```

Die Zeichenfolgen <%= und %> schließen Java Epressions ein. Diese werden zur Run Time ausgewertet. Die Klasse new java.util.Date() ist Bestandteil des JDK. (Normalerweise würde hier ein komplexeres Präsentationslogik Programm stehen).

Bei jedem Reload der HTML Seite in den Browser wird die gültige Zeit wiedergegeben.

15.2.6 Interaktion Servlet - JSP

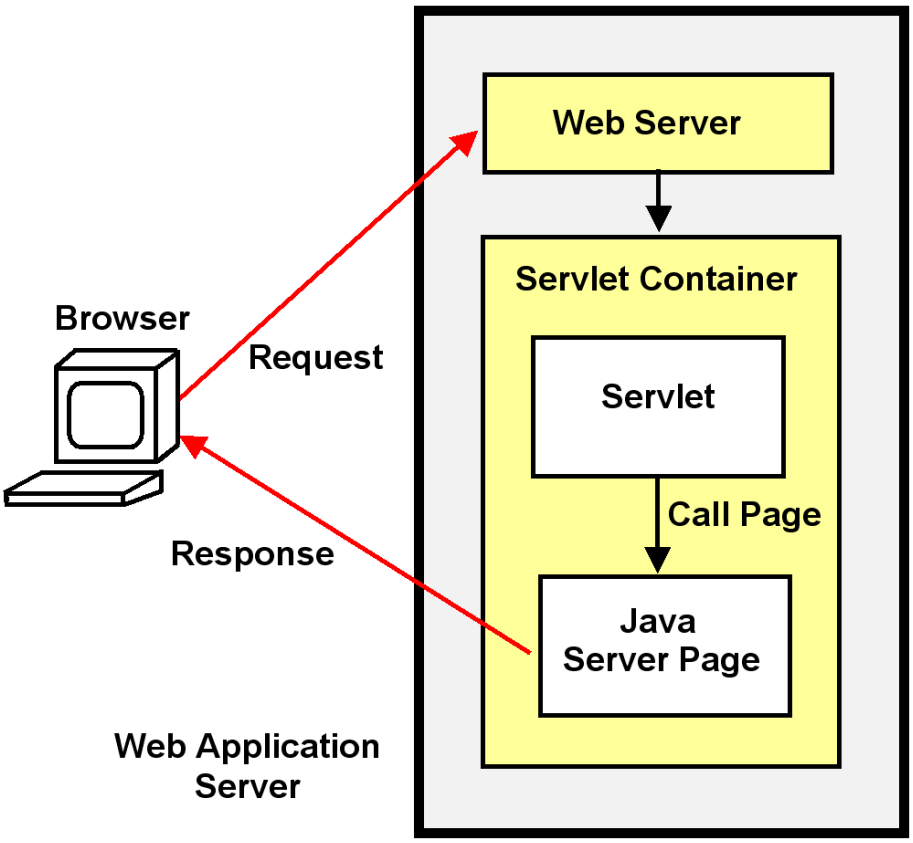


Abb. 15.2.7
Die Kombination Servlet – Java Server Page ist sehr gebräuchlich

In der Praxis ist es eher selten, dass eine JSP direkt aufgerufen wird. Statt dessen wird in der Regel ein Servlet aufgerufen, welches wiederum eine JSP aufruft.

15.2.7 Business- und Präsentationslogik

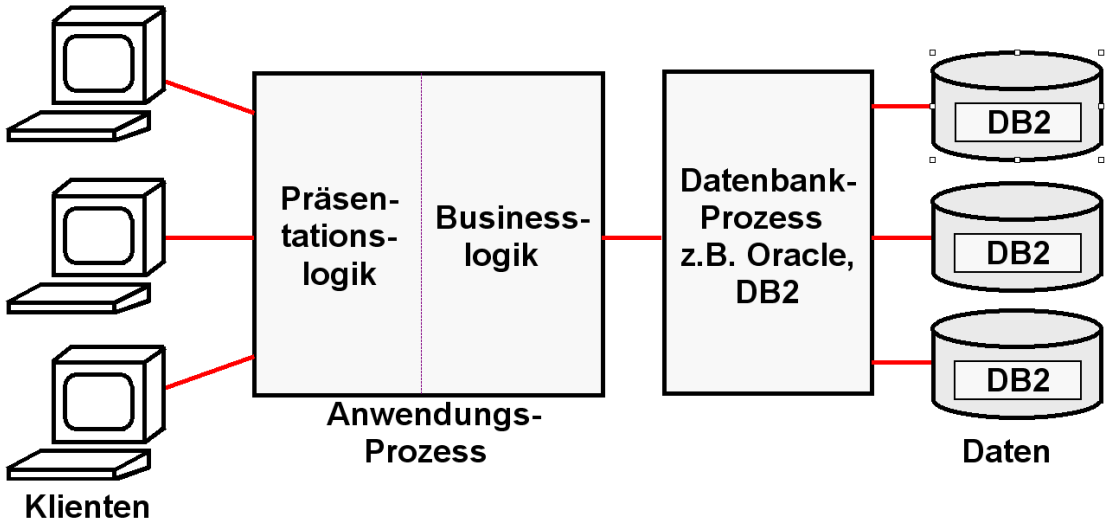


Abb. 15.2.8
Aufteilung einer Anwendung in Business Logik und Präsentationslogik

In einfachen Fällen (z.B. einige hundert Zeilen Code) kann ein Servlet die Business Logik und eine JSP die Präsentationslogik implementieren. In komplexeren Fällen ist es sinnvoll, beides in eine größere Anzahl von Java Klassen zu strukturieren. Hier bietet es sich an, den Großteil der Programmierlogik als getrennte Java Klassen zu implementieren, wobei die Servlets und JSPs lediglich Steuerfunktionen übernehmen.

Hierfür bietet sich ein Java Komponenten Modell an, die Java Beans.

Komponenten sind unabhängige, in sich abgeschlossene, wohl definierte Software Einheiten, die eine spezifische Leistung über standardisierte Schnittstellen bieten. Komponenten lassen sich mit anderen Komponenten zu größeren Einheiten zusammenfügen, die wiederum Komponenten oder eigene Anwendungen sind.

15.2.8 Java Beans

Unter Java Beans versteht man kleine Java-Programme (Klassen) mit festgelegten Konventionen für die Schnittstellen, die eine Wiederverwendung in mehreren Anwendungen (Applikationen, Servlets und Applets) ermöglichen, ähnlich wie bei Unterprogramm-Bibliotheken in anderen Programmiersprachen.

Dies ist vor allem im Hinblick auf das Software-Engineering von komplexen Programmsystemen interessant.

Dafür existiert ein eigenes Beans Development Kit BDK, das man zusätzlich zum JDK installieren kann, und eine Package java.beans, die ab Version 1.1 im JDK enthalten ist,

JavaBeans sind ein Objektorientiertes Java Komponenten Modell. JavaBeans sind Java binary parts. Sie werden häufig für visuelle Komponenten eingesetzt (etwa Buttons und Scrollbalken)

Hauptmerkmale der Java Beans sind:

- **Properties (Eigenschaften, z.B. get und set)**
- **Methoden**
- **Events (Ereignisse)**
- **Namens Konventionen**
- **Introspection (BeanInfo Klasse)**

Für unternehmensweite Anwendungen (Enterprise Applications) fehlen Schlüsseleigenschaften, z.B. Transaktionsdienste, Namensdienste und Sicherheitsdienste. Werden Java Beans hiermit angereichert, spricht man von Enterprise Java Beans.

15.2.9 Nutzung von Java Beans

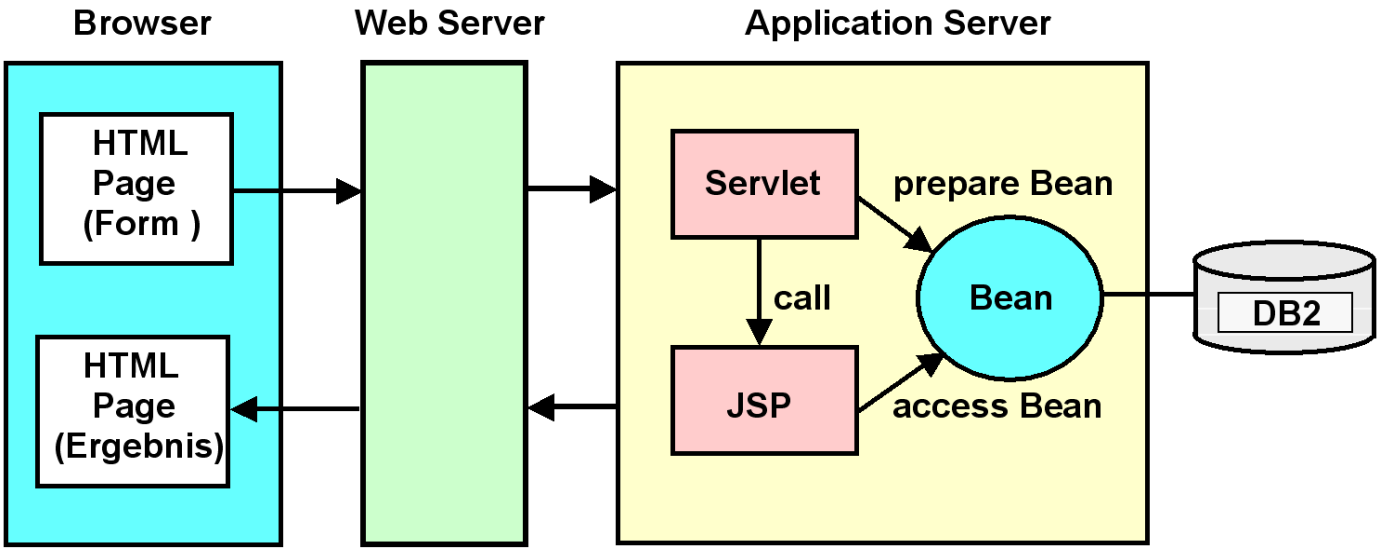


Abb. 15.2.9
Beans werden für die Business Logik eingesetzt

Der Web Server parsed eine HTML Seite und ruft ein Servlet auf.

Ein Servlet ist ein Java Programm, das Bildschirm Output in der Form einer HTML Datei produziert.

Eine JavaServerPage ist eine HTML Seite mit zusätzlichen JSP Tags.

In der Praxis: Servlets und JSP werden von verschiedenen Programmierern erstellt (Model-View-Controller Ansatz). Eine JSP ist zwar eine vollwertige Java Komponente, aber der Java Code Anteil innerhalb der JSP wird in der Regel auf ein Minimum reduziert.

Es existieren (wie für HTML Seiten) spezielle Werkzeuge für das Erstellen von JSPs, die das Hand-coding von HTML Statements automatisieren.

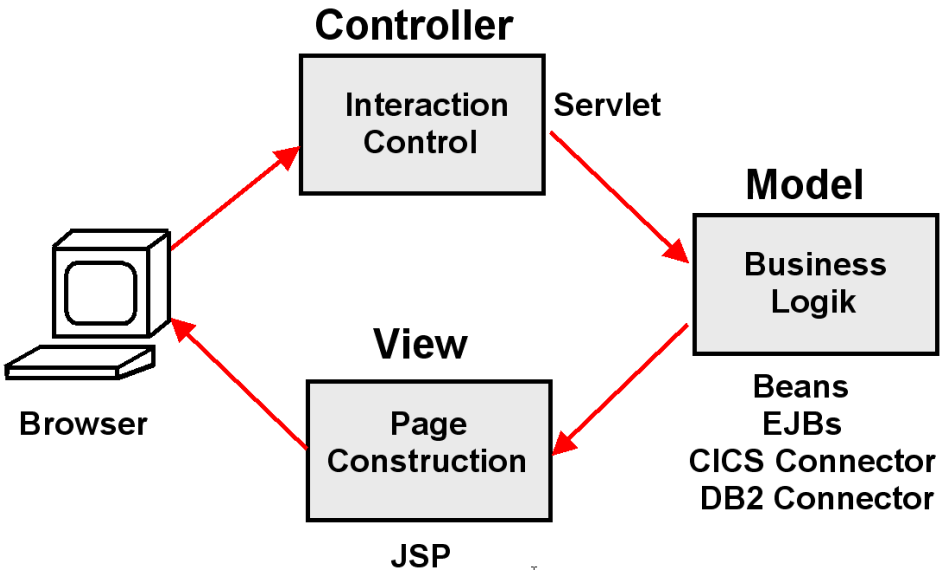


Abb. 15.2.10
MVC ist ein weit verbreitetes Programmiermodell

Als Model/View/Controller Triade (MVC) wird ein zentrisches Programmier Modell bezeichnet. Die gesamte Anwendungslogik (EJB, Servlet, JSP) läuft auf dem Server. Der Klient (thin client) braucht nur einen Browser.

Die Model/View/Controller Triade wird durch die oben dargestellte Kombination von Servlet, Java Beans und JSP implementiert.

Das "Modell" (Business Logic) ist ein Anwendungsobjekt und kapselt die Business Logik. Der "View" ist die Bildschirm Darstellung (Präsentation Logic) dieses Objektes. Der "Controller" definiert, wie die Benutzerschnittstelle auf Benutzereingaben reagiert.

Command- und Data Beans oder Enterprise Java Beans (plus häufig CICS, IMS Programme, oder Stored Procedures) sind das "Modell".

JSP´s und View Beans sind der "View"

Das Servlet ist der "Controller"

MVC entkoppelt Modell und View zur Verbesserung von Flexibilität und Re-Use. Der Entwickler der Browser Darstellung arbeitet nur mit der Java Server Page.

15.2.10 Architektur einer JSP Web Anwendung

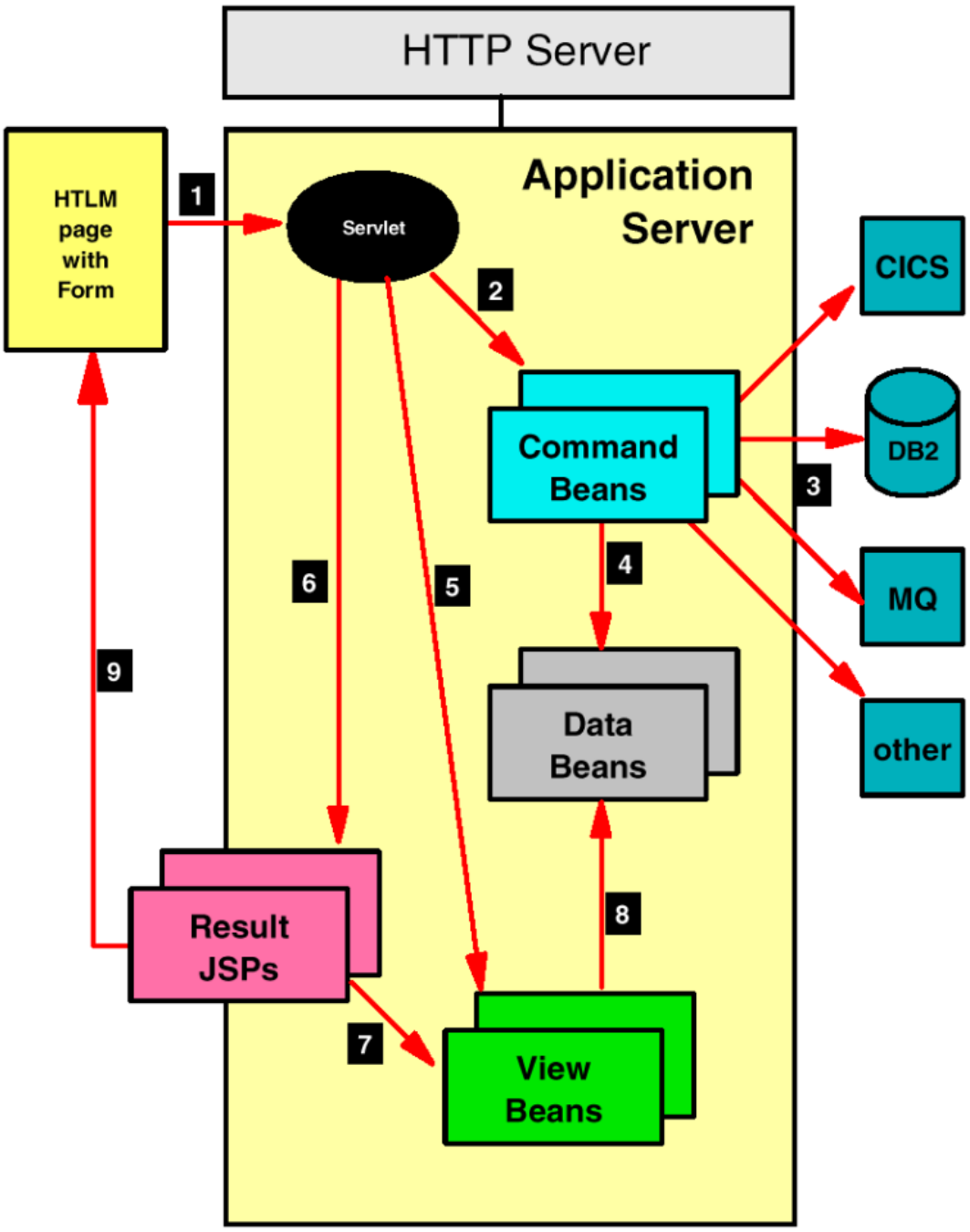


Abb. 15.2.11
Details des Model - View - Controller Ansatzes

1. HTML-Seite, die in einem vorherigen Schritt erstellt wurde, enthält eine oder mehrere HTML Forms, die ein Servlet für die Verarbeitung des nächsten Interaktion aufrufen.

2. Das Servlet erhält die Kontrolle aus der Application Server zur Validierung und Kontrolle des Ablaufes. Es ruft die Command Beans auf, welche die Geschäftslogik ausführen.

3. Command Beans steuern die Verarbeitung der Geschäftslogik. Die Logik kann in Command Beans eingebettet sein. Alternativ kann die Verarbeitung an Back-End oder Enterprise-Systeme delegiert werden, wie relationale Datenbanken, MQSeries, Transaktionen Server (CICS, IMS) usw. Command Beans rufen Datenbank-und Transaktions-Systemen über „Konnektoren" auf..

4. Ergebnisse der Command Beans (oder Back-End-Systeme) werden in Data Beans gespeichert. Data Beans können ein SQL Ergebnis oder eine CICS COMMAREA enthalten. Sie sind das Äquivalent von Unit Records.

5. View Beans definieren, wie Data Beans auf dem Bildschirm dargestellt werden sollen. Ein Servlet initialisiert die View Beans und registriert sie, so dass eine JSPs sie finden kann. View Beans enthalten die gleichen Informationen wie Data Beans, sind aber für die Verwendung durch JSP-Code optimiert.

6. Das Servlet ruft eine JSP zur Ausgabe Verarbeitung und Formatierung in Abhängigkeit von den Ergebnissen der Command Beans auf. JSPs generieren die Ausgabeinformationen für den Browser.

7. JSPs benutzen Tags zur Deklaration der View Beans, sowie um den Zugriff auf alle dynamischen Daten zu erhalten, die in der Ausgabe wiedergegeben werden müssen.

8. View Beans enthalten eine oder mehrere Data Beans und enthalten zugeschnittene Methoden, so dass die JSP Zugriff auf die Daten hat, die in den Data Beans gespeichert sind. Data Beans enthalten nicht notwendigerweise die erforderlichen Methoden, damit eine JSP auf die Daten zugreifen kann.

9. JSP assembliert die Ausgabe und sendet sie als HTML-Seite mit dynamischen Daten zurück an den Browser. In vielen Fällen enthält die Ausgabe wieder Form Tags, damit der Benutzer den Dialog mit der Anwendung fortzusetzen kann.

15.3 Enterprise Java Beans

15.3.1 Java Enterprise Edition

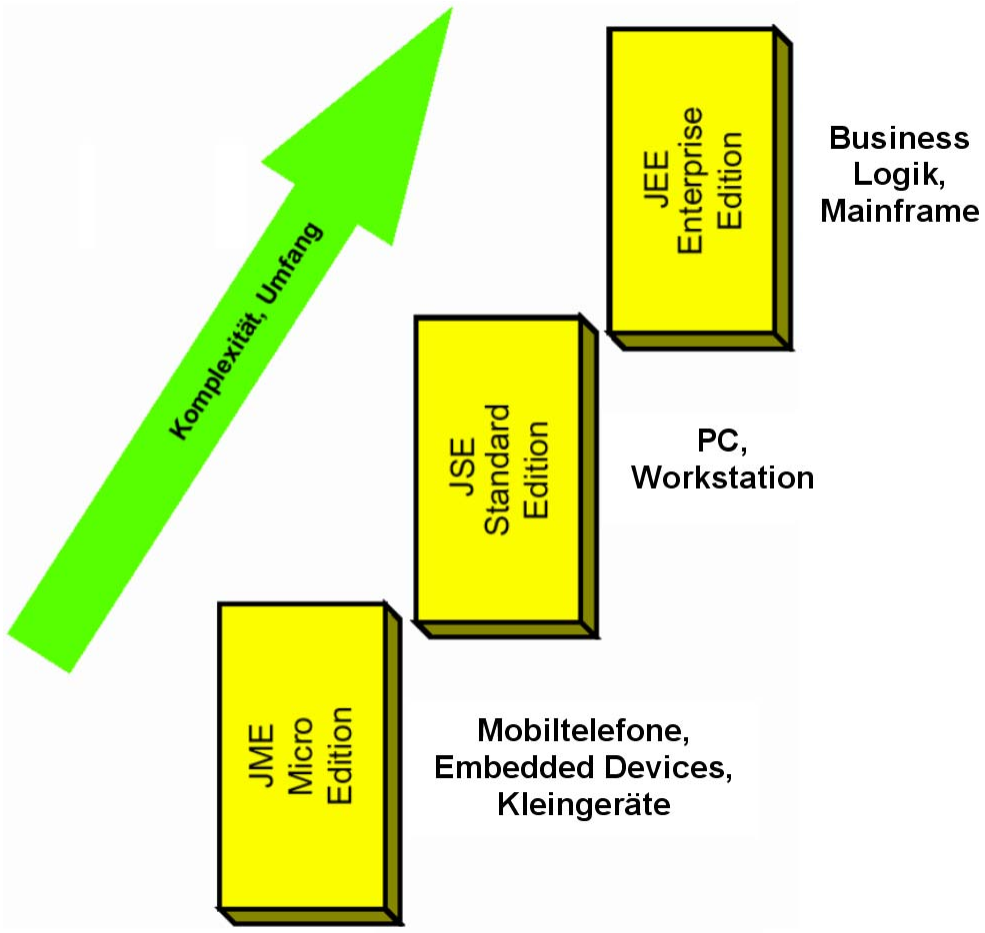


Abb. 15.3.1
Hierarchie der unterschiedlichen Java Editionen

Java existiert in drei unterschiedlichen Editionen mit einem unterschiedlichen Funktionsumfang:

JME – Java Micro Edition
JSE – Java Standard Edition

} **Frühere Bezeichnungen: J2ME, J2SE, J2EE**

Die Java Platform, Enterprise Edition, abgekürzt Java EE (JEE), oder früher J2EE, ist die Spezifikation einer Softwarearchitektur für die transaktionsbasierte Ausführung von in Java programmierten verteilten Geschäftsanwendungen und insbesondere Web-Anwendungen. JEE spezifiert:

- **Enterprise JavaBeans (EJB), die Komponenten der Geschäftsanwendungen,**
- **Infrastruktur zur Ausführung von EJBs.**

JEE unterscheidet sich von JSE vor allem durch die Verfügbarkeit von Enterprise Java Beans (EJB). Enterprise Java Beans sind ein Java basiertes Server Komponentenmodell mit sehr wesentlichen Erweiterungen gegenüber einfachen Java Beans.

Die Java Enterprise Edition (JEE) beinhaltet eine Spezifikation für Verteilte Geschäftsapplikationen. Darin ist neben den Komponenten der Geschäftsapplikation, den Enterprise JavaBeans (EJB), auch die Infrastruktur zu deren Ausführung spezifiziert. Die JEE – Spezifikation zielt auf die Implementierung von mehrschichtigen Client/Server – Anwendungen ab. Mit EJBs werden Geschäftsprozesse und Geschäftskomponenten modelliert (z.B. Kunde, Auftrag, Rechnung).

JEE erschien im März 98. Das Final Release der Version 1.1 der EJB Specifikation erfolgte im Dezember 1999. Aktuell ist die Version 6.0 der JEE Spezifikation, verfügbar seit 2009. JEE Version 7 ist für 2013 vorgesehen.

Die JEE Infrastruktur beinhaltet Applikationserver mit sogenannten Containern. Spezifisch laufen Enterprise Java Beans (EJBs) in einer als EJB Container bezeichneten Laufzeitumgebung, in denen die EJBs ausgeführt werden. Der EJB Container ist anders als, und unabhängig von, dem Servlet Container. Ein Application Server, der zusätzlich zu einem Servlet Container auch einen EJB Container enthält, wird generell als JEE Server oder als Web Application Server bezeichnet.

Der JEE Server, beziehungsweise der EJB Container, interagiert mit Systemressourcen des Unternehmens (z.B. Datenbanken) und übernimmt auch die Interaktion mit geografisch verteilten Beans in anderen Servern. Weiterhin kontrolliert er die Ausführung von selbst definierten Transaktionen und handhabt Sicherheitseinstellungen.

JEE Server bieten also typische Middleware–Techniken an, und ermöglichen eine vereinfachte Komponentenprogrammierung. So sind als Teil der Infrastruktur auch die Kommunikationsformen der EJB mit den Containern und die Kommunikation zwischen den Containern vorgeschrieben. Ebenso ist die Schnittstelle des Servers zu Klienten und zu Ressourcen weit gehend reglementiert.

EJB Programmierer sollen sich größtenteils mit der Lösung der Geschäftsabläufe befassen und wiederverwendbare Komponenten produzieren. Das Idealbild ist, dass diese Komponenten dann in jeglichen nach JEE spezifizierten Servern laufen sollen. Die Wiederverwendbarkeit von Komponenten hat sich in der Praxis bisher aber nur in wenigen Teilbereichen durchsetzen können.

15.3.2 Web Application Server

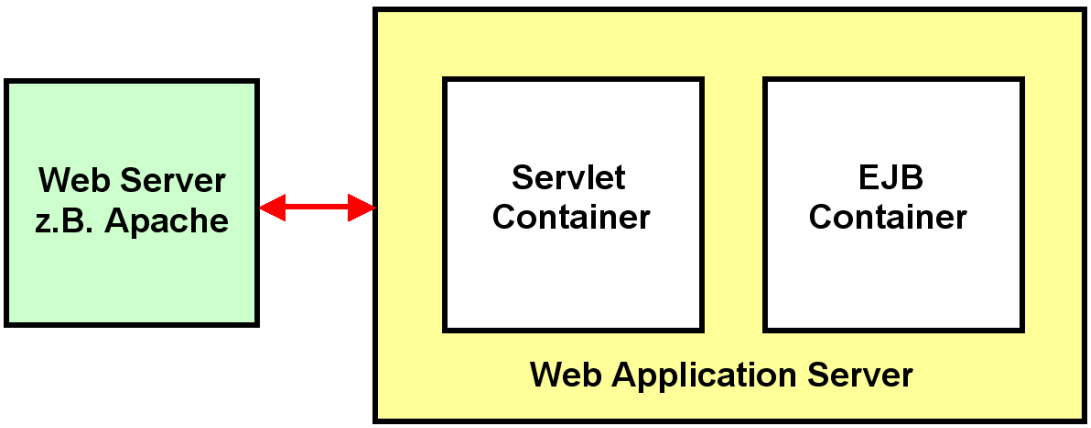


Abb. 15.3.2
Der Web Application Server implementiert den JEE Standard

Servlets benötigen für ihre Ausführung eine Servlet Laufzeitumgebung, den Servlet Container. EJBs benötigen für ihre Ausführung eine EJB Laufzeitumgebung, den EJB Container.

Ein Web Application Server ist ein Software Produkt, welches sowohl einen Servlet Container als auch einen EJB Container enthält. Häufig dient ein Servlet als EJB Client.

Der Web Server (z.B. Apache oder z/OS http Server) und der Web Application Server laufen typischerweise als getrennte Prozesse auf dem gleichen physischen Server. Weil der Begriff „Web Server“ mehrfach belegt ist, sollte statt dessen der Begriff http Server benutzt werden.

WebSphere Web Application Server (IBM), Web Logic (Oracle/Bea) und Netweaver (SAP) sind die am weitesten verbreiteten Web Application Server. Tomcat, JBOSS, GlassFish und Geronimo (Apache Foundation) sind Open Source Software Produkte mit reduzierter Funktionalität.

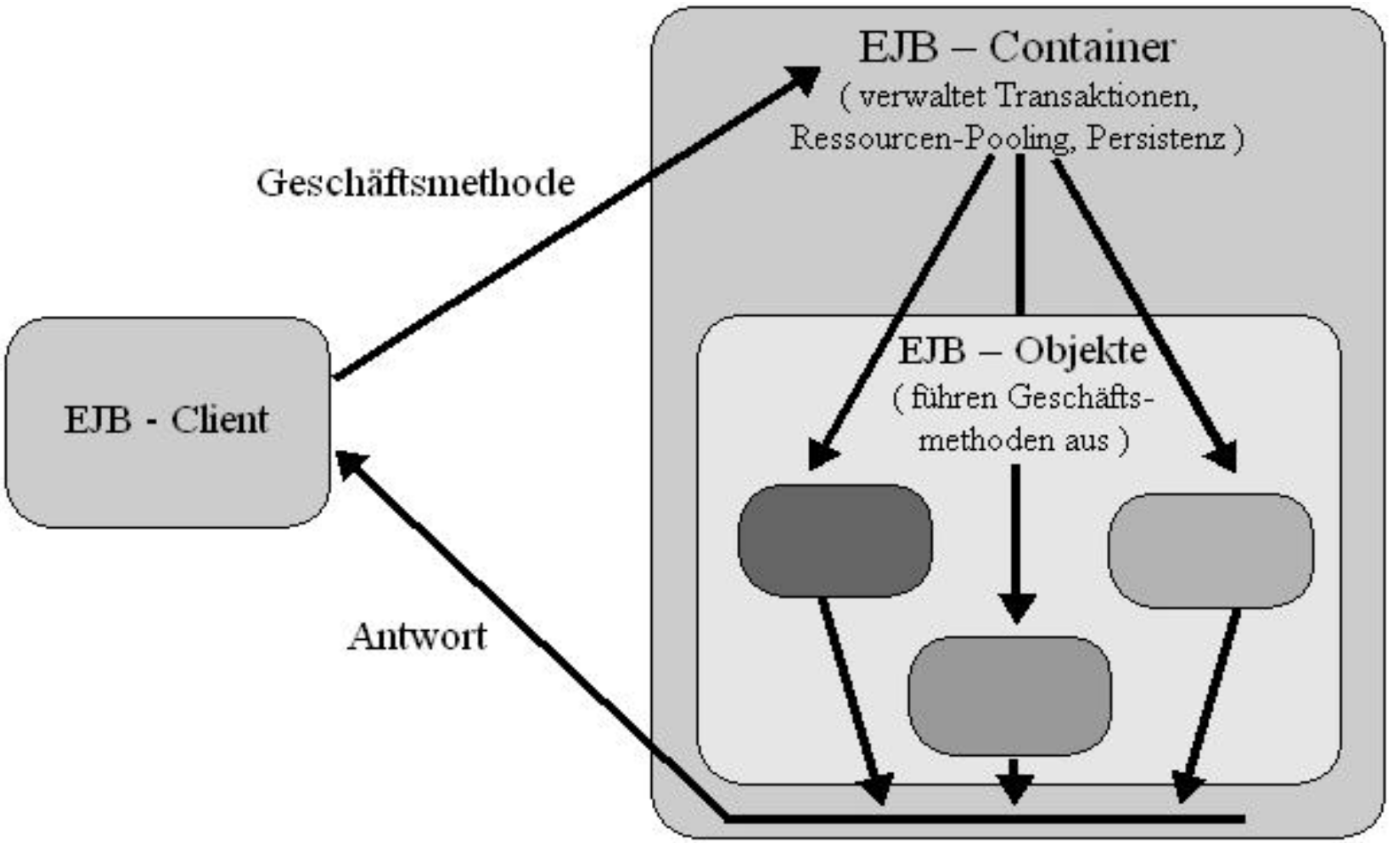


Abb. 15.3.3
Der EJB Container dient als Server in einer Client/Server Konfiguration

Der EJB Klient ist in der großen Mehrzahl der Fälle (aber nicht immer) ein anderes Java Programm außerhalb des EJB Containers. Der EJB Container enthält typischerweise zahlreiche als Enterprise Java Beans implementierten Komponenten, die gemeinsam die Business Logik darstellen.

Präsentationslogik wird in der Regel nicht mit EJBs implementiert.

15.3.3 Distributed Objects

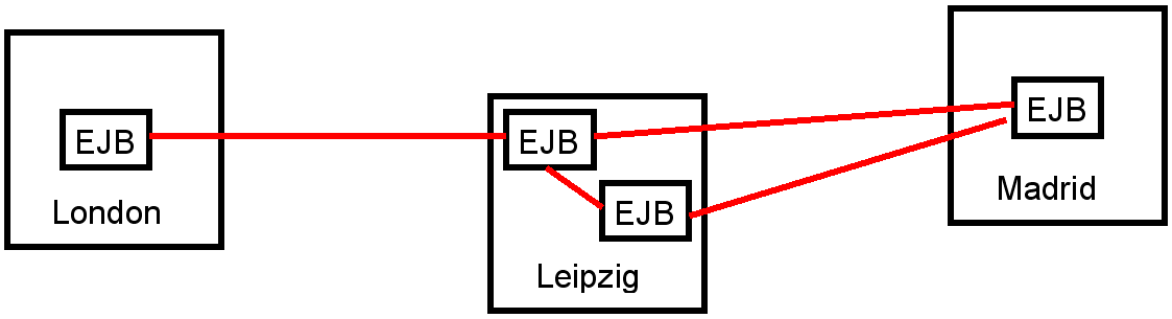


Abb. 15.3.4
EJBs sind grundsätzlich für eine geografisch verteilte Transaktionsverarbeitung ausgelegt.

Distributed Objects sind Enterprise JavaBeans, die sich auf geografisch getrennten Servern untergebracht sind.

Enterprise JavaBeans können sinnvollerweise überall dort eingesetzt werden, wo verteilte Objekte benötigt werden. Ein Beispiel ist eine Online Banking Anwendung: Ein Benutzer sitzt zu Hause und möchte sich mit allen seinen Bankkonten verbinden, gleichgültig wo und bei wem sich diese befinden, und sie alle im Zusammenhang und in einer angenehmen Benutzeroberfläche vorfinden. Die EJB – Komponentenarchitektur ermöglicht es verschiedenen Finanzinstituten, Benutzerkonten als unterschiedliche Implementierungen eines gemeinsamen Interfaces Account zu exportieren.

Da die Account Objekte zugleich EJBs sind, kann man die Transaktionsfähigkeiten des EJB Servers dazu nutzen, die Account Objekte als transaktionale Komponenten zu implementieren. Der Client kann mehrere Kontenoperationen innerhalb einer einzigen Transaktion durchführen und sie dann alle entweder mit einem Commit oder einem Rollback abschließen.

15.3.4 Persistenz

Die permanente Speicherung eines Objektes auf einem Plattenspeicher wird als Persistenz bezeichnet. Konzeptuell können Objekte in einer Objektdatenbank (z.B. POET oder Jasmine) gespeichert werden.

In der Praxis werden Objekte als SQL (oder IMS, ADABAS oder VSAM) Daten gekapselt; der Zugriff erfolgt z.B. über eine JDBC (Java Data Base Connectivity) oder SQLJ Schnittstelle.

Persistente Objekte existieren permanent außerhalb des Gültigkeitsbereichs des Programms, das sie erzeugt hat.

Persistenz wird implementiert, indem der Status (die Attribute) eines Objekts zwischen den einzelnen Programmausführungen gespeichert wird. Wenn das Objekt erneut benötigt wird, wird dieses aus seiner gespeicherten Form wieder hergestellt. Der Herstellungsprozess erzeugt ein neues Objekt, das mit dem ursprünglichen identisch ist.

Bei der Persistenz werden den gespeicherten Daten alle Objektattribute (etwa Klassenname, Feldname und Zugriffs-Modifier) zugeordnet. Ein Benutzer kann sich darauf verlassen, dass persistente Objekte Katastrophen und andere Störfälle überdauern. RAID 5, 6 oder 10 Plattenspeicherstrukturen und weitgehende Sicherheits- und Recovery-Maßnahmen in z/OS und den Enterprise Storage Servern ermöglichen dies.

15.3.5 Dienstleistungen des EJB Containers

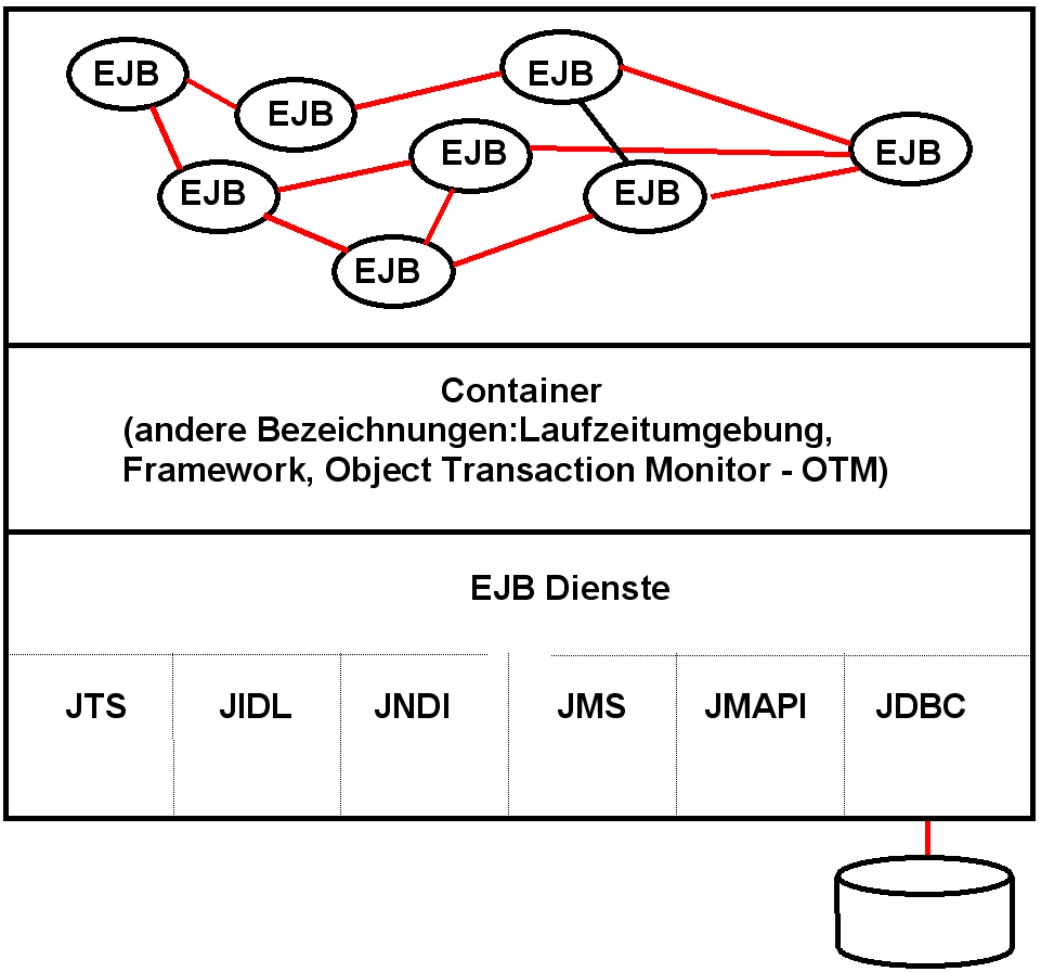


Abb. 15.3.5
Der EJB Container stellt den EJBs eine Reihe von Diensten zur Verfügung

Es existieren drei unterschiedliche Arten von EJBS, die sich gegenseitig aufrufen können :

- **Entity Beans**
- **Session Beans**
- **Message Beans**

Enterprise Java Beans sind Java Beans mit erweiterter Funktionalität. Diese Funktionalität wird von dem EJB Container als EJB-Dienste zur Verfügung gestellt und enthält unter anderem:

- **JTS Java Transaction Service**
- **JNDI Java Naming directory Interface**
- **JMS Java Messaging Service**
- **JDBC Java Data Base Connectivity**
- **JMAPI Java Management API**
- **JIDL Java interface definition language**

JTS, der Java transaction Service stellt die Funktion eines Transaction Servers zur Verfügung. Die Kommunikation zwischen den in Java implementierten Anwendungen und dem JTS geschieht über ein Protokoll namens Java Transaction API (JTA). Siehe Abschnitt 19.1.1.

JNDI, die Java Naming and Directory Interface, ist eine API für den Zugriff auf Namens- und Verzeichnisdienste. Zum Beispiel wird sie verwendet, um Datenbanken oder EJBs zu lokalisieren, die von einem Servlet benötigt werden.

JMS, der Java Message Service, ist eine API zum Aufrufen einer asynchrone Übermittlung von Nachrichten.

JDBC, die Java Database Connectivity API, greift auf Daten in bestehenden Datenbanken über eine gemeinsame Schnittstelle zu.

JMAPI, die Java Management API, definiert den Zugriff auf einen Satz von Diensten zum Verwalten von Java-Ressourcen.

JIDL, die Java Interface Definition Language, ist eine Schnittstelle zu einer Reihe von CORBA Dienstleistungen für verteiltes Rechnen.

EJBs sind Komponenten, die eine objektorientierte Sicht auf persistent gespeicherte Entitäten repräsentieren, also eindeutig identifizierbare und über Attribute beschreibbare Informationseinheiten,. Diese Entitäten sind beispielsweise in einer Datenbank gespeichert. Handelt es sich dabei um eine relationale Datenbank, so korrespondieren Entity Beans z.B. mit jeweils einer Zeile der entsprechenden Datenbanktabelle.

15.3.6 Session und Entity Beans

Session Beans implementieren die eigentliche Business Logik. Man unterscheidet zustandslose (stateless) und zustandsbehaftete (stateful) Session Beans.

Eine zustandsbehaftete Session Bean hat ein eigenes Gedächtnis. Sie kann Informationen aus einem Methodenaufruf speichern, damit sie bei einem späteren Aufruf einer anderen (oder der gleichen) Methode wieder zur Verfügung stehen. Die Zustandsbehaftung wird durch die Vergabe einer eindeutigen ID umgesetzt. Über diese ID können die zustandsbehafteten (stateful) Session Beans unterschieden werden.

Im Gegensatz dazu müssen einer zustandslosen Session Bean bei jedem Aufruf alle Informationen als Parameter übergeben werden, die für die Abarbeitung dieses Aufrufs benötigt werden.

Zustandslose Session Beans können von mehreren Klienten gleichzeitig benutzt werden. Bei den zustandsbehafteten Session Beans muss für jeden Klienten eine eigene Kopie angelegt werden.

Entity Beans repräsentieren für den Klienten eine objektorientierte Sicht auf einen Datensatz. Sie erlauben im Gegensatz zu Session Beans auch Mehrbenutzerbetrieb: Auf eine Instanz einer Entity Beans können gleichzeitig mehrere Benutzer zugreifen. Der Container, in den Entity Beans während der gesamten Lebensdauer eingebettet sind, stellt dazu Mechanismen bereit, um z.B. Sicherheit, Transaktionskonsistenz und Parallelität sicherzustellen.

Da Entity Beans nicht an einen einzelnen Client gebunden sind, endet ihre Lebensdauer nicht nach dem Beenden einer Client - Verbindung.

Im Gegensatz zu Session Beans können bzw. müssen sie sogar nach einem Systemausfall automatisch wiederhergestellt werden. Ihre Existenz ist an das Vorhandensein der mit ihnen verbundenen Daten gebunden.

Eine Entity Bean speichert Daten (ihre Variablen) persistent. Die Erstellung einer neuen Entity Beans erzeugt z.B. automatisch eine neue Zeile in einer Datenbank und fügt die bei der Erstellung mit übergebene Daten der Datenbank hinzu. Wird die EJB gelöscht, wird automatisch der mit dieser EJB verbundene Datensatz aus der Datenbank gelöscht.

15.3.7 Session Fassade Architektur

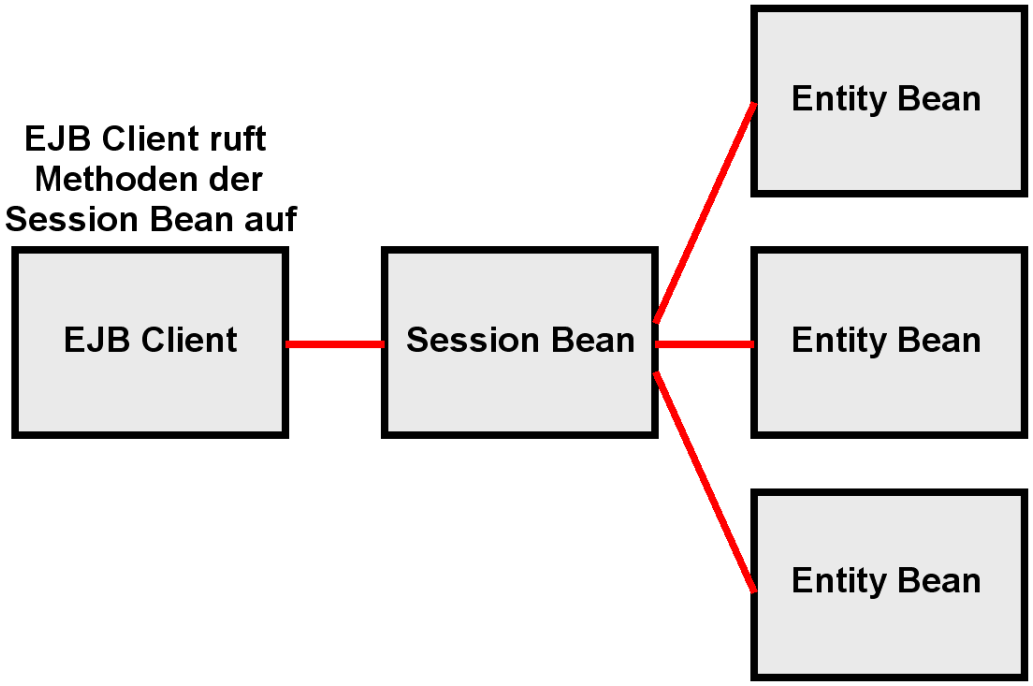


Abb. 15.3.6
Die Session Fassade benutzt Session- und Entity Beans

In der ursprünglichen Konzeption der J2E Architektur werden Session Beans und Entity Beans gemeinsam benutzt, um die Business Logik in Java zu implementieren. Dies geschieht mit Hilfe der „Session Fassade Architektur. Hierbei führen Session Beans Geschäftsprozesse (Business Logik) aus, und manipulieren persistente Objekte (Daten), die als Entity Beans modelliert werden. Dabei ging man davon aus, dass Entity Beans in objektorientierten Datenbanken (z.B. Poet, Jasmine) gespeichert würden. Dies hat sich in der Praxis aber nicht durchgesetzt.

Weitere Probleme mit dem Session Fassade EJB Architektur Modell sind:
- **Entity Beans sind verteilte Objekte, die von beliebigen Klienten aufgerufen werden können**
- **Entity Beans sind transaktionsfähig**

Beides ist bei einer Session Fassade nicht erforderlich und damit unnötiger Overhead.

Manche Anwendungen haben deshalb ausschließlich Session Beans eingesetzt und die Persistenz mit eigenem Code implementiert. Um dies zu erleichtern, stellten einige Hersteller (miteinander inkompatible) Persistence-Frameworks zur Verfügung. Weit verbreitet sind die „Java Data Objects“ (JDO) der Apache Foundation als Ersatz für die Entity Beans. JDOs sind reguläre Java Klassen, die über einen Persistence Encapsulation Mechanismus verfügen.

Aus den JDOs entstand die Java Persistence API (JPA) als Teil des EJB 3.0 Standards. Mit der Einführung des EJB 3.0 Standards wurden Entity Beans deprecated (sind obsolet geworden). Stattdessen sollen Persistent Entities verwendet werden. Eine Persistence Entity ist ein Plain Old Java Object (POJO), das üblicherweise auf eine einzelne Tabelle in der relationalen Datenbank abgebildet wird. Instanzen dieser Klasse entsprechen hierbei den Zeilen der Tabelle. Persistence Entities können je nach Designvorgabe als einfache Datenhaltungs-Klassen (vergleichbar mit einem Struct in C) realisiert werden oder als Business-Objekte inklusive Business-Logik.

15.3.8 Annotation

Als Annotation wird ein Sprachelement bezeichnet, das die Einbindung von Metadaten in den Java Quelltext erlaubt. Dieses Element wurde mit der Version Java5.0 eingeführt (verfügbar seit 2004 als Nachfolger der Java Version 1.4).

Annotationen beginnen mit einem @-Zeichen. Daran schließt sich ihr Name an. Optional kann eine kommagetrennte Parameterliste folgen, die in runden Klammern eingefasst wird. Beispielsweise markiert die Annotation im folgenden Quelltextausschnitt die Klasse A als überholt (deprecated):

```
/**
 * @deprecated  Die Klasse A wurde mit Version 10.3 durch die Klasse ANeu
ersetzt.
 */
@Deprecated
public class A {}
```

Eingesetzt werden Annotationen unter anderem im Java-EE Umfeld, um Klassen um Informationen zu erweitern, die vor der Einführung von Java5.0 in separaten Dateien hinterlegt werden mussten. Prominente Beispiele sind Home- und Local Interfaces sowie Deployment-Deskriptoren.

15.3.9 Java Naming and Directory Interface

Eine Methode einer Java Klasse wird aufgerufen, indem man ihren Namen spezifiziert. Die Klasse und ihre Methode sind irgendwo mit irgendeiner Adresse gespeichert. Wenn wir einen Server im Internet aufrufen, verwenden wir den Internet Domain Name Service (DNS). Wir rufen einen Server mit einem Namen auf, z.B. leia.informatik.uni-leipzig.de. DNS übersetzt den Namen in eine Internet Adresse, z.B. 139.18.4.30.Für komplexere Vorgänge benutzen wir einen Verzeichnisdienst (directory service) an Stelle eines Namensdienstes (name service) wie DNS. LDAP (Light Weight Directory Access Protocol) ist ein weit verbreiteter Verzeichnisdienst.

Wenn eine Methode einer Java Klasse eine Methode einer zweiten Java Klasse aufruft, muss sie dessen Lokation (Adresse) kennen. Das kann schwierig sein, wenn die zweite Klasse sich irgendwo im Netz befindet.

Lookup (englisch für „Nachschlagen“) ist der Vorgang, mit dem die Adressen von benannten Objekten ermittelt werden.

Java Klassen greifen über eine standardisierte Schnittstelle auf einen Verzeichnisdienst zu, um die Lokation einer entfernten Klasse und deren Methoden zu finden. Diese Schnittstelle ist die Java Naming and Directory Interface (JNDI).

JNDI ist kein Verzeichnisdienst, sondern eine Schnittstelle (API) zu einem Verzeichnisdienst. Die Schnittstelle ist dabei unabhängig von der tatsächlichen Implementierung. Es ist die Aufgabe des Herstellers eines JEE Servers (z.B. IBM WebSphere oder Oracle Weblogic), diesen mit einem Verzeichnisdienst auszurüsten, auf dem mittels JNDI zugegriffen werden kann. In vielen Fällen wird hierfür LDAP benutzt. JNDI ist eine sogenannte Service Provider Interface (SPI), das Herstellern eines Web Application Servers erlaubt, eigene Lösungen in ihren JEE Server einzubinden.

Die API enthält:

- **einen Mechanismus zur Bindung (binding) eines Objekts an einen Namen**
- **Methoden für den Abruf von Informationen anhand eines Namens**
- **ein Ereigniskonzept, über das Klienten über Änderungen informiert werden**
- **spezielle Erweiterungen für LDAP-Funktionalitäten**

In JNDI werden die Namen hierarchisch angeordnet. Namen sind üblicherweise Strings wie „com.mydomain.MyBean“, können aber auch beliebige Objekte sein, die die Schnittstelle javax.naming.Name implementieren. Im Namens- bzw. Verzeichnisdienst ist für jeden Namen entweder das ihm zugeordnete Objekt selbst gespeichert oder eine JNDI-Referenz auf das zugeordnete Objekt. Die Programmierschnittstelle von JNDI („JNDI API“) definiert, wo nach dem Objekt zu suchen ist.

JNDI erlaubt die Nutzung praktisch aller Arten von Namens- und Verzeichnisdiensten, insbesondere von:

- **LDAP**
- **DNS**
- **NIS**
- **CORBA Namensdienst**
- **Dateisystem**

In der Praxis wird JNDI vor allem dazu benutzt, im Rahmen von Java-Enterprise-Anwendungen verteilte Objekte in einem Netzwerk zu registrieren und sie für Remote-Aufrufe (RMI) weiteren Java-Programmteilen zur Verfügung zu stellen.

15.4 Schnittstellen

15.4.1 Struktur des Web Application Servers

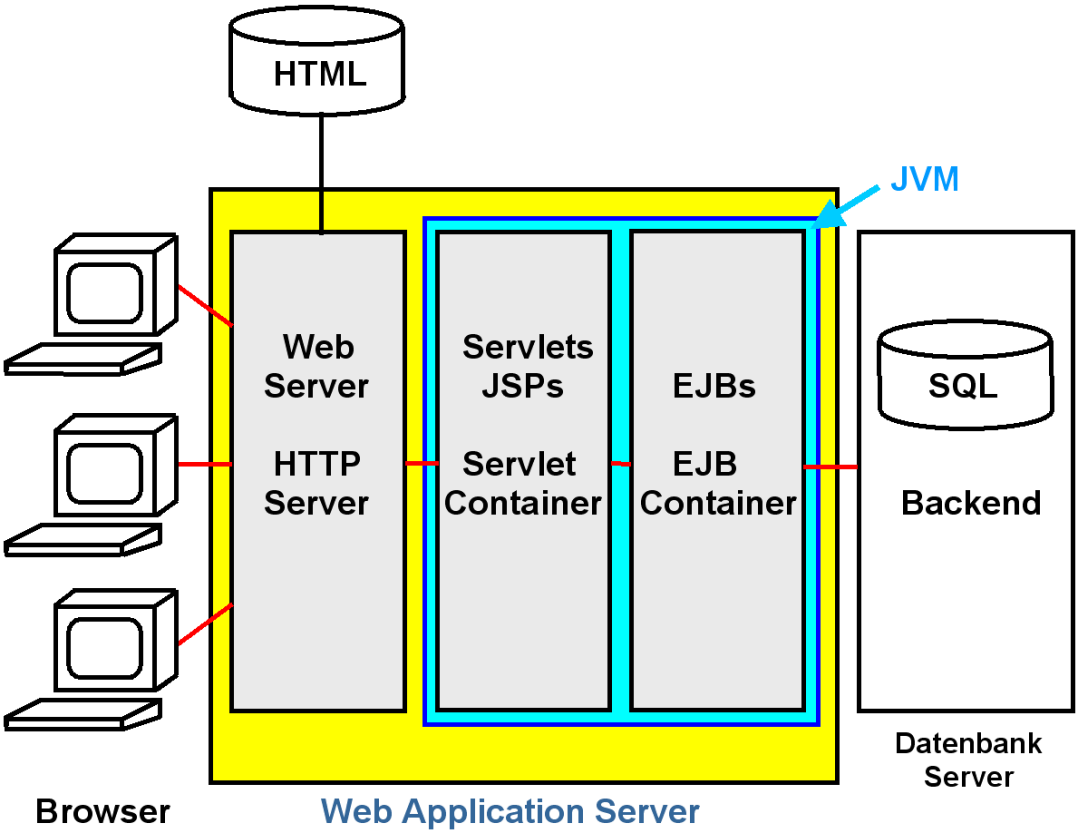


Abb. 15.4.1
Struktur des Web Application Servers

IBM bezeichnet seinen JEE Server als WebSphere Web Application Server (WAS). Der HTTP Server kann ein getrennter Prozess sein.

Servlet Container und EJB Container laufen in einer gemeinsamen Java virtuellen Maschine (JVM). Dies reduziert den Kommunikationsaufwand. Servlet Klassen können EJB Klassen direkt aufrufen. Bei getrennten JVMs erfolgt die entsprechende Kommunikation mit Hilfe des RMI oder RMI/IIOP Protokolls (Kapitel 16).

Der Web Application Server enthält weitere Elemente für Administration und Datenbankzugriff.

Der Web Application Server ist ein Prozess, der normalerweise in seinem eigenen virtuellen Adressenraum läuft. Er besteht aus mehreren Komponenten:

1. **Der Web Server (HTTP Server) ist vielfach Apache.**
2. **Der Java Application Server unterhält u.a. eine Java Virtuelle Maschine**
3. **Der Servlet Container (Servlet Engine) ist eine Java Laufzeit Umgebung (runtime component) für die Ausführung von Servlets und Java Server Pages.**
4. **Der EJB Container ist eine Laufzeit Umgebung für die Ausführung von deployed Enterprise Java Beans.**

Apache ist der am weitesten verbreitete http Server. Apache läuft unter AIX, HP-UX, Linux, Solaris, Windows, und z/OS. Apache wird von einer Open Community von Entwicklern unter der Leitung der Apache Software Foundation kontinuierlich verbessert.

Der IBM HTTP Server basiert auf dem Apache http Server. Das Download ist kostenlos. Der IBM HTTP Server ist ebenfalls ein Teil der IBM WebSphere Application Server Distribution Package.

Neben Apache existieren zahlreiche weitere http Server. Am bekanntesten ist der Microsoft Internet Information Server (IIS), vorgesehen für eine Benutzung mit Microsoft Windows. Dies ist nach Apache der zweitwichtigste http Server.

Der IBM Lotus Domino Server ist ein spezialisiertes Software Product mit einer eigenen Datenbank (Notes Storage Facility). Der Lotus Domino HTTP Server wird gelegentlich auf Mainframes eingesetzt.

Daneben existieren zahlreiche weitere HTTP Server Produkte.

In der Vergangenheit war ein Web Server immer ein Server, der eine URL auswerten und statische HTML Seiten an einen Klienten (Browser) zurückliefern konnte. Bei der Einführung von CGI und Servlets erfolgte eine Ergänzung (Plug-in), um beim Parsen von CGI oder Servlet Tags das entsprechende CGI oder Servlet Programm aufrufen zu können.

Später erfolgte eine Begriffsänderung (Begriffsverwirrung). Man begann komplexe WWW Anwendung unter Nutzung von Servlets, JSPs und EJBs zu erzeugen, sowie viele unterschiedliche derartige Anwendungen auf dem JEE Server zur Verfügung zu stellen. Zu einer vollständigen Anwendung (z.B. Banküberweisung, Hotelreservierung, Bestellung beim Otto Versand) besteht diese neben Servlets, JSPs und EJBs zusätzlich auch aus einer Reihe von statischen HTML Seiten. Man fasst diese anwendungsspezifischen statischen HTML Seiten mit den Servlets und JSPs zu einer Einheit zusammen, und nennt diese Einheit Web Server, Web Engine oder Web Container. Den bisherigen Web Server bezeichnet man zur Unterscheidung als http Server. Der http Server verwaltet alle die HTML Seiten, die nicht einer Anwendung spezifisch zugeordnet werden können.

Leider ist diese Zweideutigkeit bis heute geblieben. Es ist deshalb sinnvoll einen Server für statische HTML Seiten (wie z.B. Apache) grundsätzlich als http Server zu bezeichnen.

15.4.2 Web Server Plug-in

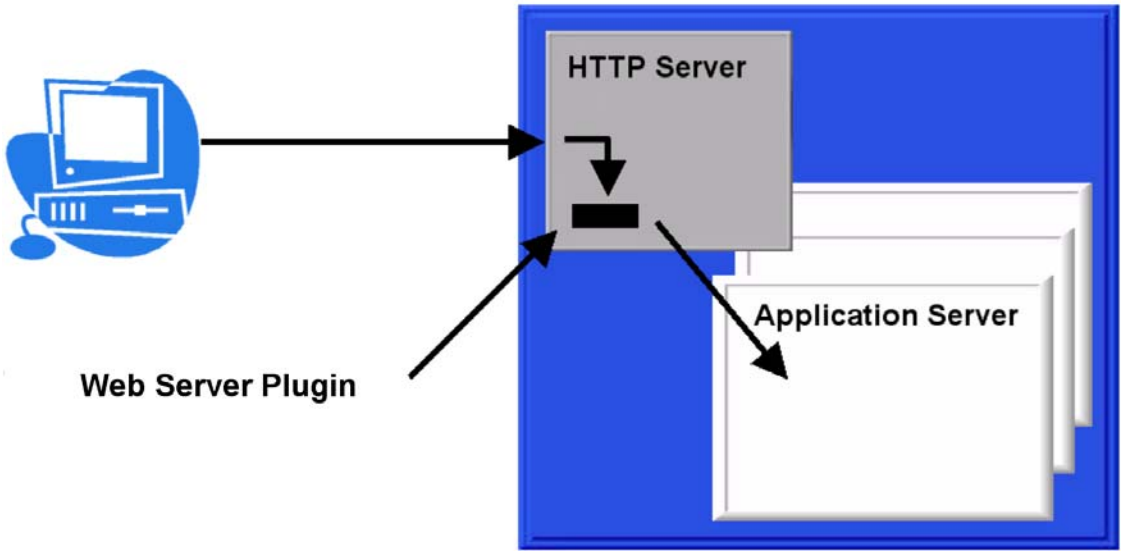


Abb. 15.4.2
Rolle des http Server Plug-ins

Es besteht der Wunsch der Benutzer von JEE Servern wie Weblogic, WebSphere Web Application Server (WAS) und Netweaver, mit unterschiedlichen http Server Produkten (z.B. Apache, IIS) zusammen arbeiten zu können. Deswegen enthält eine JEE Server Distribution mehrere Interface Komponenten, die eine Verbindung mit unterschiedlichen http Servern ermöglichen. Diese Interface Komponenten werden als „Plug-in“ bezeichnet.

Die WebSphere Web Application Server Distribution enthält z.B. Plug-in’s für Apache, den IBM http Server, für IIS und für den Lotus Domino http Server. Bei der Installation wird das Plug-in in den http Server integriert.

Die ursprüngliche WebSphere Application Server Standard Edition für OS/390 (verfügbar in 1999) lief in dem gleichen Adressenraum wie der http-Server.

Der Web Application Server war als Erweiterung zu einem http-Server gedacht, bestehend aus 2 Hauptkomponenten:

- **Ein Plug-in für den Web-Server (http-Server), welcher die Anforderung an den tatsächlichen Application Server weiterreicht, und**
- **Der Application Server selbst.**

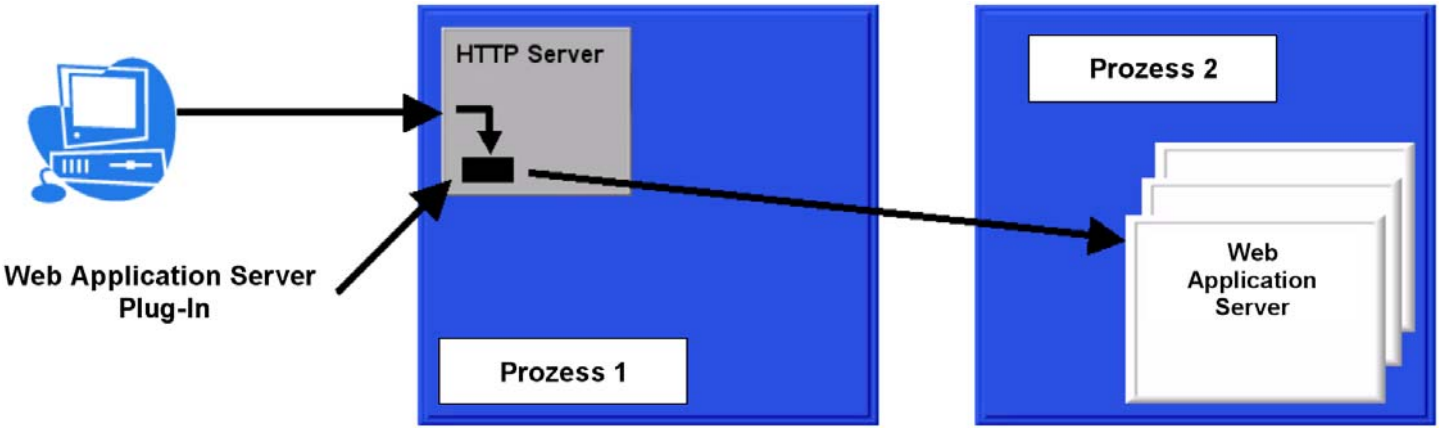


Abb. 15.4.3
Plug-in integriert in den http Server

Heute läuft ein Web Application Server als eigener Prozess, in der Regel auf der gleichen Maschine wie der http-Server (Web-Server). Es könnte auch auf einem anderen Rechner laufen. Es können mehrere Application Server Kopien als unabhängige Prozesse konfiguriert werden.

Das Webserver Plug-in ist ein Teil der Web Application Server Software-Package, wird aber in dem http-Server-Adressraum installiert. Es kann mit dem Application Server über mehrere Protokolle kommunizieren, abhängig von dem Application Server Hersteller. IBM WebSphere WAS, einschließlich der z/OS-Version verwendet HTTP und HTTPS.

Dies ist der Grund dafür:

Ein WebSphere WAS Software Package enthält den IBM HTTP Server-Code, der als Standard-Option installiert ist. Sie kann jedoch konfiguriert werden, einen anderen Web Server zu benutzen, z. B. Microsoft IIS. Damit IIS auf den WebSphere WAS zuzugreifen kann, muss der mit WebSphere mitgelieferte Plug-in in dem IIS-Adressen Raum installiert werden.

Wenn Anforderungen in dem HTTP Server eintreffen, entscheidet die http-Server-Konfigurationsdatei (httpd.conf), ob die Anfrage an den Plug-in code weitergereicht werden soll.

Das Plug-in enthält eine eigene Konfigurationsdatei. Diese entscheidet, an welchen von potentiell mehreren Application Servern die Anforderung weiter gereicht wird.

Abb. 15.4.4 und 15.4.5 zeigen zwei mögliche Web Application Server Konfigurationen.

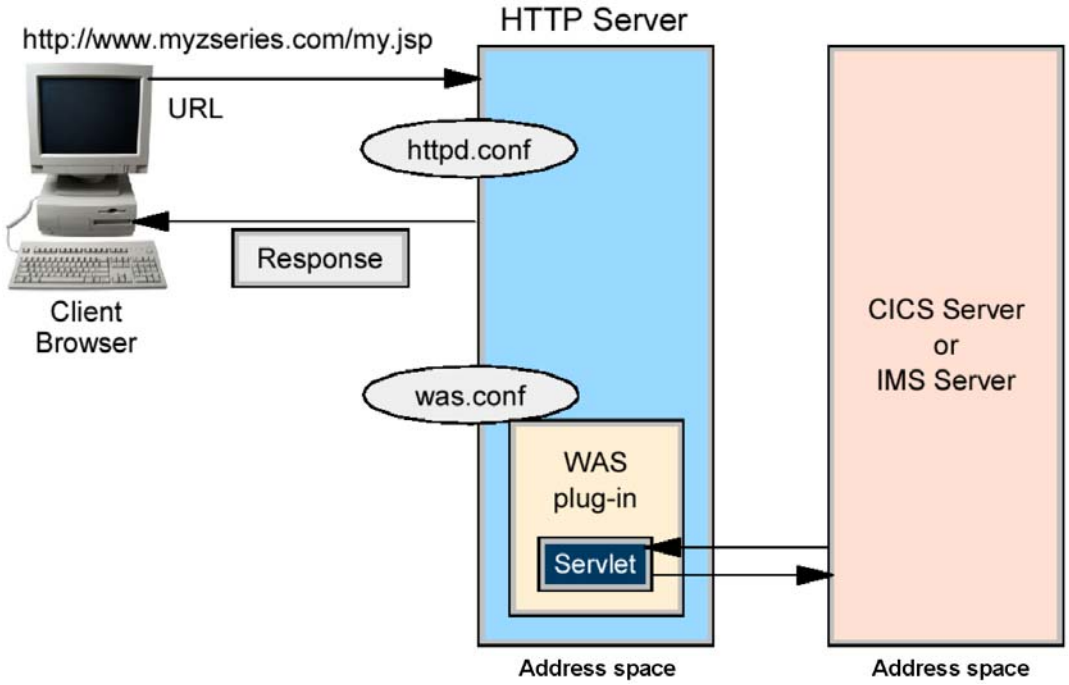


Abb. 15.4.4
Kein EJB Container

Diese Abbildung zeigt eine einfache Konfiguration, in der kein JEE-Server benötigt wird. Der Servlet Container befindet sich in dem gleichen Adressenraum wie der http Server. Das Servlet kann CICS, IMS oder DB2 über JDBC aufrufen. Allerdings wird eine Kodierung von Geschäftslogik innerhalb des Servlets nicht empfohlen.

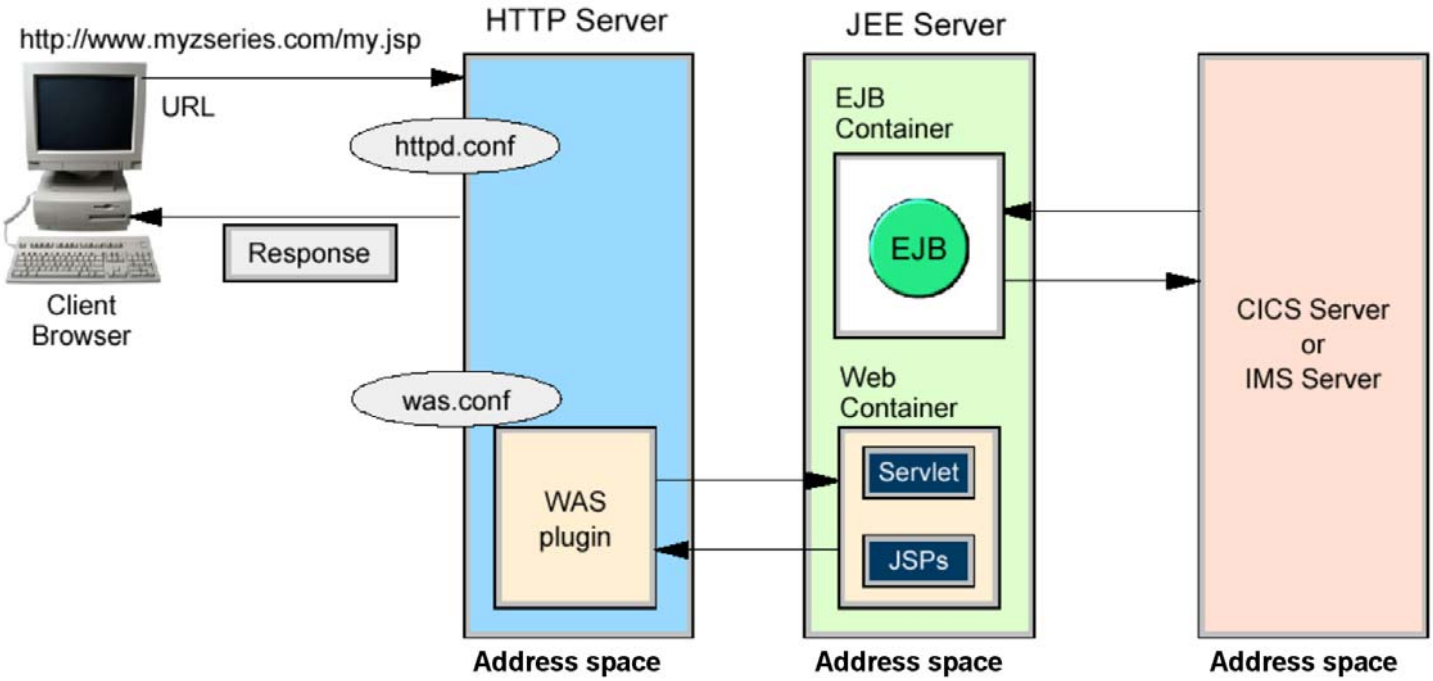


Abb. 15.4.5

Sie können auch das WebSphere Plug-in benutzen, um Servlets remote in einem „Web-Container“ auszuführen. Dies ermöglicht es, Servlets und EJBs in dem gleichen Adressraum auszuführen, so dass keine Remote EJB Aufrufe (über das RMI-Protokoll) benötigt werden.

Web Container ist eine andere Bezeichnung für Servlet Container. Der Name rührt daher, dass in den Web Container auch anwendungsspezifische statische HTML Seiten untergebracht werden können.

15.4.3 Ablaufsteuerung einer JEE Server Anwendung

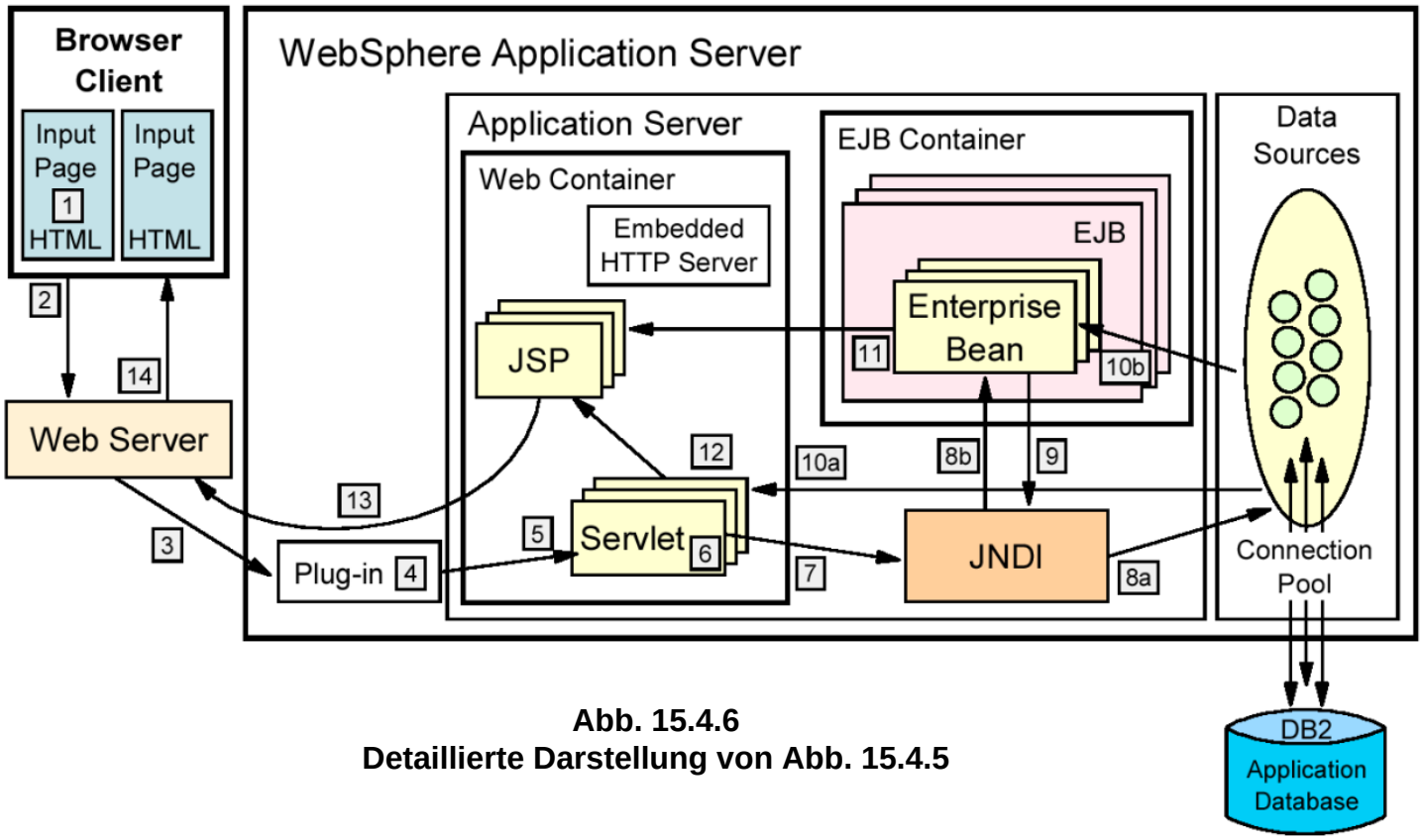


Abb. 15.4.6
Detaillierte Darstellung von Abb. 15.4.5

Der typische Ablauf der Ausführung einer JEE Server Anwendung ist wie folgt:

1. **Ein Web-Client ruft eine URL in seinem Browser auf (Input Page).**
2. **Die Anfrage wird über das Internet an den Web Server (http Server) geroutet.**
3. **Der Web-Server leitet die Anfrage an das Web Server Plug-in weiter.**
4. **Das Web Server Plug-in prüft die URL, und wählt einen Server aus, um die Anfrage zu bearbeiten.**
5. **Ein Stream wird erzeugt. Ein Stream ist eine Verbindung mit einem Web-Container. Es ist möglich, eine Verbindung (Stream) über eine Anzahl von Anfragen aufrecht zu erhalten. Der Web-Container erhält die Anfrage und reicht sie an das richtige Servlet weiter.**
6. **Wenn die Servlet-Klasse nicht geladen ist, lädt der dynamische Klassenlader das Servlet (Servlet init() Methode), und ruft dann die doGet() oder doPost() Methode auf.**
7. **JNDI wird benutzt, um die Lokation entweder der Data Source (Datenbank, VSAM) oder der EJBs zu finden, die von dem Servlet benötigt werden.**
8. **Je nachdem, ob eine Data Source angegeben ist oder eine EJB angefordert wird, directet JNDI das Servlet zu:**
   - **der entsprechende Datenbank. Hierfür wird eine bereits existierende Verbindung von dem Connection Pool zugeordnet.**
   - **dem entsprechenden EJB-Container, welcher dann die EJB instanziiert.**
9. **Wenn die EJB eine SQL-Transaktion beinhaltet, benutzt sie JNDI zum Nachschlagen der Data Source.**

10. Die SQL-Anweisung wird ausgeführt, und die aufgerufenen Daten werden entweder zu dem Servlet oder der EJB zurück gesendet.
11. Data Beans werden erstellt und im Falle einer EJB an eine JSP weiter gereicht.
12. Das Servlet sendet Daten an die JSP.
13. Die JSP generiert eine HTML Seite, die über das Plug-in an den Web-Server weiter gereicht wird.
14. Die Web-Server sendet die Output-Seite (Ausgabe HTML) an den Browser. Diese wird dort zur Input Page für den nächsten Schritt.

15.4.4 Web Application Server als Tranaktionsmonitor Front End

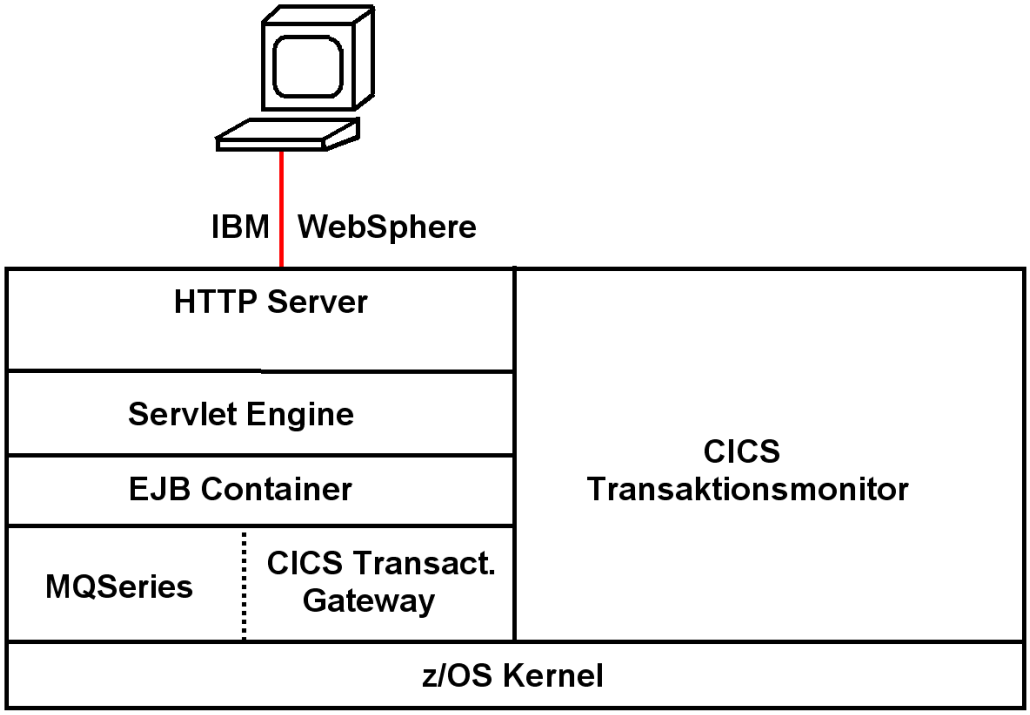


Abb. 15.4.7
WebSphere und CICS laufen unter z/OS in getrennten Adressenräumen

Obwohl EJBs Transaktionseigenschaften haben, und die gesamte Business Logik implementieren könnten, setzt man in vielen Fällen einen traditionellen Transaktionsmonitor wie CICS für den Kern der Business Logik ein.

15.4.5 Message Driven Bean

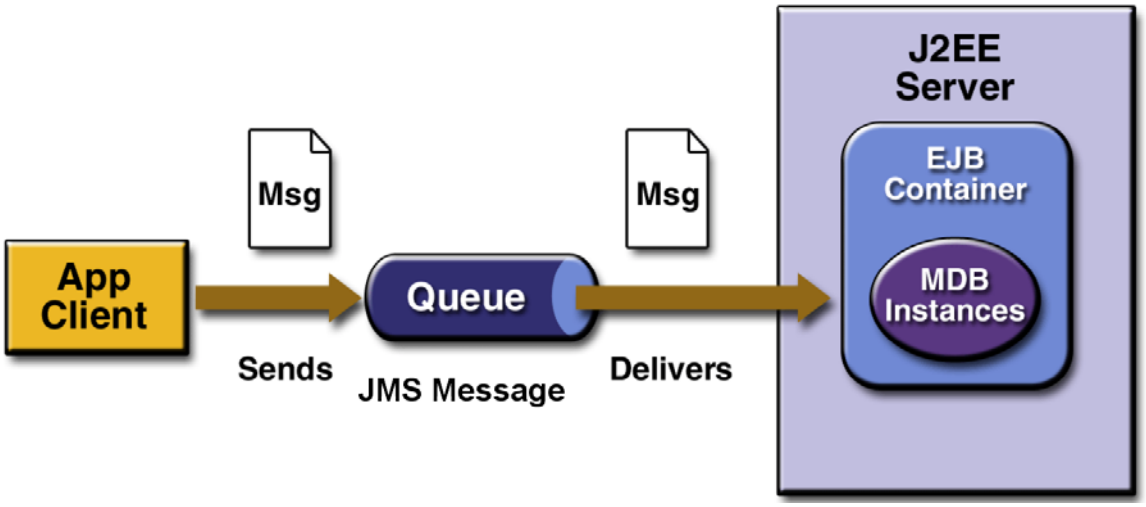


Abb. 15.4.8
WebSphere benutzt MDBs für die Java Implementierung von MBQ

Message based Queuing (MBQ) (Band 1, Abschnitt 10.1.5) ist ein Verfahren zur Program to Program Kommunikation. Es kann verwendet werden, um einen asynchronen RPC implementieren. MQSeries ist ein Beispiel. Programme werden in die Lage versetzt, ohne eine direkte Verbindung Informationen zu senden oder zu empfangen. Programme kommunizieren durch das Hinterlegen von Nachrichten in Message-Queues, und Abrufen von Nachrichten von Message-Queues.

Der JEE Java Message Service (JMS) ist eine Java spezifische Implementierung von MBQ. Es ermöglicht eine Nachrichtenübermittlung durch asynchrone Methodenaufrufe

Der Service, der einen MBQ Nachricht empfängt kann als reguläre Java-Klasse implementiert werden. Alternativ kann der Service als EJB, einer Message Drive Bean (MDB), implementiert werden.

Eine Message Driven Bean (MDB) ist eine spezielle Art einer Enterprise Java Bean (EJB). Eine Message Driven Bean ist ein potentieller Empfänger eines asynchronen Methodenaufrufes. Dieser Methodenaufruf kann von einer beliebigen anderen Java Klasse ausgehen. Die sendende Klasse wartet nicht auf eine Antwort.

Im Gegensatz zu Session Beans und Entitiy Beans erfolgt der Zugriff auf eine MDB nicht über eine Schnittstelle (Interface).

Alle Instanzen eines MDB – Typs sind gleich, da sie nicht direkt für den Client sichtbar sind und keine Zustände annehmen. Daraus resultiert die Möglichkeit für den EJB Container, Pools aus MDB – Instanzen zu bilden, um Skalierbarkeit zu erzeugen.

Der Websphere Web Application Server (WAS) benutzt das MQSeries Produkt, um die MDB Unterstützung zu implementieren. MQSeries existiert in 2 Versionen:

- **Die reguläre MQSeries Version, die alle Sprachen unterstützt. Diese Installation erfordert keinen Websphere Application Server, wird aber dennoch heute als WebSphere MQ bezeichnet.**
- **Eine spezielle MQSeries Version, die MDBs unterstützt und die in WebSphere integriert ist Diese unterhält Queues für die MDBs, sowie eine Listener Komponente, welche eine spezifische MDB benachrichtigt, dass in ihrer Queue eine Nachricht eingetroffen ist, die von der onMessage() Methode der MDB empfangen werden kann.**

Beide Versionen verwenden den gleichen Code. IBM hat daher beschlossen, beide Versionen in WebSphereMQ umzubenennen. Die reguläre MQSeries Version kann (und häufig wird) ohne WebSphere verwendet werden.

Message Driven Beans implementieren eine Enterprise Java Bean Variante einer Message Oriented Middleware wie z.B. MQSeries. Der WebSphere Application Server verwendet MQSeries Software Komponenten für die Implementierung der Message Driven Beans Infrastruktur.

Ein wesentlicher Unterschied zwischen den beiden MQ Varianten besteht darin, dass MDBs an Stelle des MQ Channels und der MQI API Enterprise Java Bean spezifische Konstrukte benutzen.

15.4.6 Message Listener Service

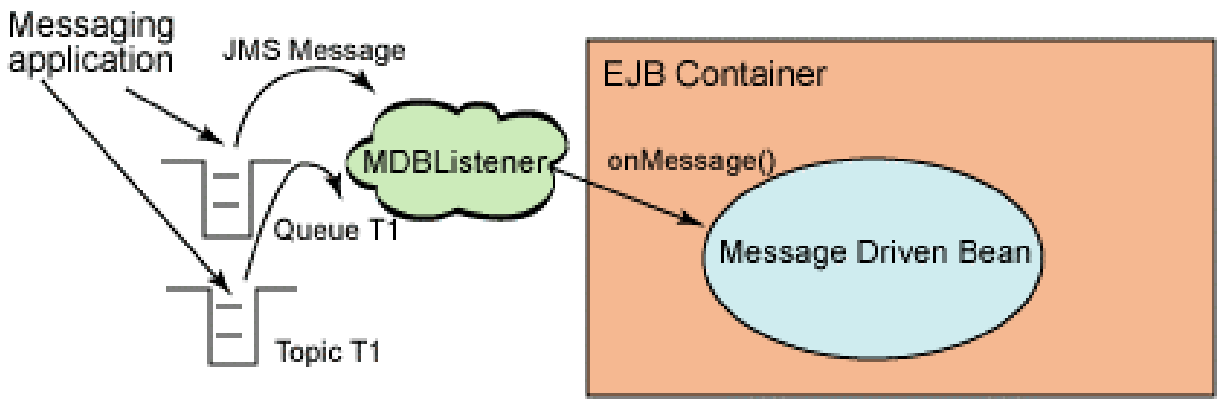


Abb. 5.4.9
Der Message Listener Service ist eine Funktion des Web Application Servers

MDBs hören (listen) ein bestimmtes JMS-Ziel ab. Dies kann zum Beispiel eine Queue sein. Die Message-Client sendet Nachrichten an die jeweilige Queue, auf die die Message Driven Bean zuhört.

Ein MDB-Listener ist ein Service eines WebSphere Application Servers. Der Message Listener Service verwaltet für jede MDB einen Listener. Ein MDB-Listener bewirkt, dass eine MDB (Message-Driven Bean) informiert wird, wenn eine Nachricht an einem bestimmten Ziel z.B. eine Queue angekommen ist . Dem Listener ist ein spezifischer Port zugeordnet. Wenn die Queue eine Nachricht empfängt, ruft der EJB-Container die `onMessage()`-Methode der Message-Driven Bean auf. Die `onMessage()` Methode kann auch als einzige Schnittstelle zu einem MDB betrachtet werden.

Ein Listener-Port ist eine Komponente des Applikationsservers, der ein JMS-Ziel und das Eintreffen von Nachrichten überwacht. MDBs sind an einen bestimmten Listener-Port gebunden. Wenn eine Nachricht an dem Listener-Port ankommt, wird sie an die entsprechende MDB (Message-Driven Bean) weitergeleitet..

In EJB 3.0 wird der MDB-Listener durch die „Aktivierungsspezifikationen“ ersetzt.

15.4.7 Web Application Server Implementierungen

WebSphere (IBM), Web Logic (Oracle) und Netweaver(SAP) sind die mit Abstand führenden Web Application Server Produkte. Nur WebSphere ist verfügbar unter z/OS.

Für einen Vergleich, siehe

http://www.cedix.de/VorlesMirror/Band2/WasCompet01.pdf

und

http://www.cedix.de/VorlesMirror/Band2/WasCompet02.pdf

Jboss, Geronimo und GlassFish (Oracle) sind Opensource-Implementierungen eines Web Application Servers.

Die eigenständige Servlet-Laufzeitumgebung Tomcat (ebenfalls Open Source) ist als Webcontainer (Servlet Container) standardmäßig in JBoss und Geronimo integriert.

JBoss und Geronimo sprechen ihren Transaktionsmanager (bis auf den Import und Export des Transaktionskontextes) allein über die JTS-Schnittstellen an, sodass sich verhältnismäßig einfach beliebige JTS-konforme Transaktionsmanager integrieren lassen.

Jboss, Geronimo und GlassFish haben nicht den Reifezustand und Funktionsumfang von kommerziellen Web Application Servern wie Oracle WebLogic oder IBM WebSphere. JBoss besteht aus knapp 4.300 Klassen und belegt installiert 50 Mbyte auf dem Festplattenspeicher, Websphere besteht aus über 20.000 Klassen und belegt installiert 460 MByte.

15.5 Weiterführende Information

Das Java Cult Buch „Java ist auch eine Insel“ ist kostenlos zum Download verfügbar unter http://jedi.informatik.uni-leipzig.de/de/VorlesMirror/ii/Vorles/JavaInsel.pdf.

Kurzbeschreibung von JEE 7
http://www.heise.de/newsticker/meldung/Oracle-gibt-Java-Enterprise-Edition-7-offiziell-frei-1887250.html

Zum Thema Java Stand-alone Anwendungen existieren zwei Redbooks:

http://www.redbooks.ibm.com/abstracts/sg247177.html
http://www.redbooks.ibm.com/abstracts/sg247291.html

Ein Java Video Tutorial unter Benutzung von Eclipse ist zu finden unter

http://www.youtube.com/watch?v=Ie_-tsHyLXU

Es existieren mehrere Video Tutorials zum Thema JEE, z.B.

http://www.youtube.com/watch?v=dugOUNswU_k

http://www.youtube.com/watch?v=hD2L9AE9sDc

http://www.youtube.com/watch?v=0q-HC3PDX5s

Nicht alles ist positiv mit Java. See:
http://www.inventus.org/posterous/file/2011/12/8168678-083-085_COM.pdf ,
oder Mirror

http://www.cedix.de/VorlesMirror/Band2/JavaWithers.pdf

Eine Dalvik Slide Presentation ist verfügbar unter

https://14b1424d-a-62cb3a1a-s-sites.googlegroups.com/site/io/dalvik-vm-internals/2008-05-29-Presentation-Of-Dalvik-VM-Internals.pdf?attachauth=ANoY7coirObuO3E_oo_ONpPrvNv0FtwVAZbiaqy0_83GabP2Eg40TuIG09FZWN5zabZY3OtMV3w1yjrzKYAp_SYhUDoAevRE7Lbd9meZMb2cBKc-QZH4XzPC8g9OUkMZAiDp908K7X_QnZno21DuDYphAibQ5AGuNBAZs0wCQK7TnjBfPvVM0n4CUOCeosFrXS4S--dUyrIRE6LB3q7VK21QpN0mRfVChLZXN50PNlM--Z6PQEfCWjhYI72lOS8LxMLWK3B9WrSzyANfKOXuJCgQE62HuiXxdw%3D%3D&attredirects=0

oder als Mirror

http://www.cedix.de/VorlesMirror/Band2/DalvikVM.pdf

Ein Video, welches die Android Dalvik virtuelle Maschine beschreibt, ist verfügbar unter

https://sites.google.com/site/io/dalvik-vm-internals/