

8. Web Application Server

Version 1.1, 10. Okt. 01

8.1 JAVA

8.1.1 Java und der Web Server

Ein zentrales Element für die Anbindung von neuen Internet und e-Business-Anwendungen an existierende OS/390 Strukturen ist die Java Sprache. Java kommt in vielen Ausprägungen; die wichtigsten sind:

- Java
- Java Script
- Java Applet
- Java Servlet
- Java Server Pages
- Java Beans
- Enterprise Java Beans

Alle Java-Ausprägungen (ausgenommen Java Script bedienen sich der Java-Virtuellen Maschine. Die Ablaufumgebung ist in Abb. 8.1.1 wiedergegeben.

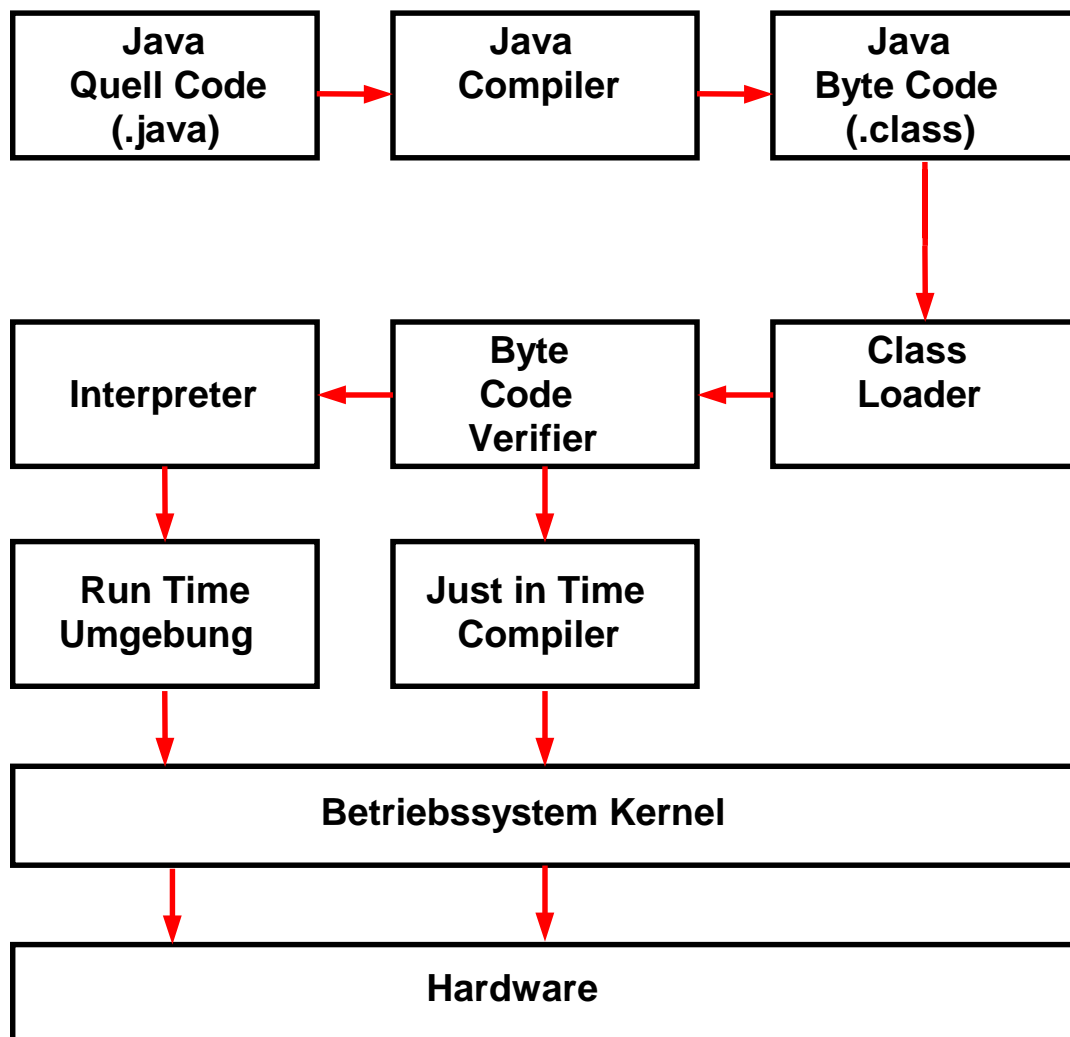


Abb. 8.1.1 :Java Virtuelle Maschine (JVM)

Java hat sich in den letzten Jahren wegen seines erfolgreichen Objekt-Komponentenmodells und wegen seiner Verfügbarkeit auf fast allen Plattformen (write once, run everywhere) durchgesetzt. Der größte Nachteil ist das im Vergleich zu anderen Sprachen geringere Leistungsverhalten.

Hierfür gibt es vier Gründe:

1. Die Objektorientierung von Java bedingt relativ viele, kleine Methoden. Dies führt zu häufigeren Methodenaufrufen, als dies bei anderen Sprachen der Fall ist. Objekt-orientierter Code führt häufig zu tieferen Nesting (Methoden rufen Methoden auf, die Methoden aufrufen). Die Reuseability-Eigenschaften von Objekt-orientierten Code implizieren die Nutzung von generellen Class-Bibliotheken oder existierenden Frameworks zwecks Verbesserung der Produktivität.
2. Die Java-Spezifikation erfordert Runtime Checking zur Gültigkeitsprüfung von Zugriffen auf Objekte und Arrays. Wenn eine Operation ungültig ist, muß eine Exception erzeugt (thrown) werden. Die Umgebung, z.B. lokale Variablen, muß gesichert und dem Exception Handler verfügbar gemacht werden.
3. Synchronisierte Methoden (Blöcke) stellen die Atomizität für eine Menge von Operationen in einer Region sicher. Dies erzeugt Runtime Overhead durch das Locking von Objekten für die Dauer der Ausführung. Generelle Class-Bibliotheken sind in der Regel für die Nutzung in einer Multi-threaded-Umgebung ausgelegt, in der synchronisierte Methoden häufig benutzt werden um Thread-Sicherheit zu garantieren.
4. Die interpretative Ausführung von Byte Code oder die suboptimale Code-Optimierung bei einem Just-in-Time-Compiler.
5. Garbage Collection wird periodisch aufgerufen um den Speicherplatz innerhalb eines Prozesses zu steuern.
6. Java's native Code Page ist Unicode. Dieser erzeugt zusätzlichen Overhead.

8.1.2 Web Browser und Web Server

8.1.2.1 HTTP 1.0

Ein Web Browser greift mit Hilfe des HyperText Transfer Protokoll (HTTP) Protokolls auf einen Web Server zu (Abb. 8.1.2). HTTP ist das ursprüngliche Transport-Protokoll für das World Wide Web. Der Web-Server nimmt Anfragen über TCP/IP Port 80 entgegen, liest die in der URL spezifizierte Seite aus seinem HTML-Speicher aus und sendet sie an den Web Browser zurück.

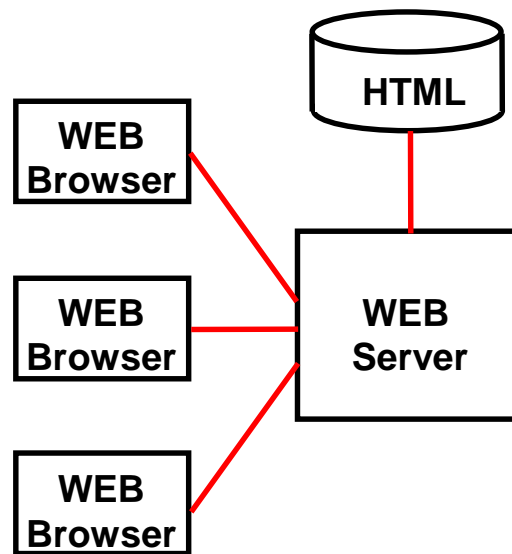


Abb. 8.1.2: Java und der Web Browser

Es existieren mehrere Versionen des HTTP-Protokolls. HTTP 1.0 (spezifiziert in RFC 1945) ist die älteste und derzeit (2001) immer noch am weitesten verbreitete Version.

HTML-Seiten werden zum 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 ein „connectionless“ und ein „stateless“ Protokoll. Ein Web-Zugriff erfolgt in 4 Schritten:

1. Klient öffnet eine Verbindung mit dem (Web) Server, benutzt hierfür TCP (connection oriented).
2. Klient sendet eine "Request" an den Server. Wird z.B. das URL

`http://www.qrx.de/index.html`

eingetragen, so sendet der Klient (Browser) die folgende HTTP-Methode an den Server:

```
GET /index.html HTTP/1.0
User-Agent: Mozilla/4.7 [en] (Win2000; I)
Accept: image/gif, image/jpeg, */*
```

3. Der Server sendet eine HTML-Seite als Antwort (Response).
4. Die TCP-Verbindung wird geschlossen.

Da die Verbindung bereits besteht, genügt für die GET-Methode die relative Adresse. Der Get-Befehl besagt: "Hole mir das File index.html, unter Verwendung des HTTP 1.0-Protokolls. Ich benutze Netscape Version 4.7 das auf Windows 2000 lauffähig ist (englische Version) und akzeptiere gif- und jpeg-Bildformate".

Der Klient kann an den Server-Eingabedaten als Teil der Nachricht übersenden, z.B. durch Eingabe der URL

`http://www.qrx.com/index.html?username=dieter&password=japetus`

Dies erzeugt die HTTP-Nachricht:

```
GET /index.html?username=dieter&password=japetus HTTP/1.0
User-Agent: Mozilla/4.7 [en] (Win2000; I)
Accept: image/gif, image/jpeg, */*
```

Allerdings wird man diese Möglichkeit nicht oft benutzen.

Neben GET kennt das HTTP 1.0-Protokoll sechs weitere Methoden, von denen vier selten benutzt und nicht von allen Browsern implementiert werden. Die wichtigste andere Methode ist POST. Sie wird eingesetzt, um Eingabedaten vom Browser an den HTTP-Server zu senden. POST wird normalerweise durch das Klicken auf den Submit-Knopf einer HTML-Form erzeugt. Dies ist die resultierende HTTP Nachricht:

```
POST /abc.html HTTP/1.0
User-Agent: Mozilla/4.7 [en] (Win2000; I)
Accept: image/gif, image/jpeg, */*
Content-Length: 34
```

```
username=dieter&password=japetus HTTP/1.0
```

Die Response des Web-Servers fängt mit einem dreistelligen Status-Code an. Die wichtigsten Gruppen von Status-Codes sind:

2xx	Das Request des Klienten wurde erfolgreich bearbeitet.
4xx	Das vom Klienten übersandte Request ist inkorrekt formatiert.
5xx	Der Server hat ein Problem, das (gültige) Request des Klienten zu bearbeiten.

Eine HTTP-Response könnte z.B. wie folgt aussehen (der Wert 200 in der ersten Zeile besagt, die Bearbeitung war erfolgreich):

```
HTTP/1.0 200 OK
Date: Sun, 14. Oct. 2001 08:12:41 GMT
Server: Apache/1.2.6
Content-length: xxx
Content-Type: text/html

<HTML><HEAD><TITLE> Hallo </TITLE></HEAD>
<BODY><H1> Hallo </H1>
Hallo Welt !!!
</BODY></HTML>
```

8.1.2.2 Neuere HTTP-Protokolle

HTTP 1.1 (spezifiziert in RFC 2068) definiert zusätzliche Request-Methoden. Eine weitere Eigenschaft sind persistente Verbindungen. Im Gegensatz zu HTTP 1.0 schließt ein HTTP 1.1-Server nicht mehr die Verbindung, nachdem er das Client-Request mit einer Response beantwortet hat (default Einstellung). Die meisten Web-Server unterstützen HTTP 1.1 .

HTTP NG ist eine High Performance-Erweiterung zu HTTP 1.x. Die Entwicklung ist noch nicht abgeschlossen.

S-HTTP ist eine Erweiterung des HTTP-Protokolls. Es erlaubt, einzelne Nachrichten sicher zu übertragen. Nicht alle Browser unterstützen S-HTTP; Secure Socket Layer (SSL) wird häufig als die bessere Alternative angesehen und ist weiter verbreitet.

8.1.3 HTML Forms

HTML-Forms sind ein einfaches Werkzeug, mit dem ein Benutzer Daten mit Hilfe des HTTP-Protokolls an einen Server schicken kann. Eine HTML-Form besteht aus einem Code-Block, der mit dem <FORM> Tag anfängt und mit dem </FORM> Tag aufhört. Eine HTML-Seite kann mehrere Forms

enthalten.

Ein Beispiel für eine einfache Form ist:

```
<HTML>
<HEAD><TITLE>Mailing LIST</TITLE></HEAD>
<BODY>
<FORM METHOD="POST" ACTION="/cgi-bin/login.cgi"
ENCTYPE="application/x-www-form-encoded">
<P>Name: <INPUT TYPE="TEXT" NAME="name" SIZE="25"></P>
<P>Email Address: <INPUT TYPE="TEXT" NAME="email" SIZE="25"></P>
<P><INPUT TYPE="SUBMIT" VALUE="Submit">
</FORM>
</BODY>
```

Der FORM-Tag spezifiziert:

- Die zu benutzende HTTP-Methode. Hier 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-Encodierung der Daten in der FORM. Der Default ist "application/x-www-form-encoded".

8.1.4 Java Script und Java Applets

Von den verschiedenen Java Arten ist JavaScript am einfachsten zu erklären. JavaScript ist eine Scripting Language. Es ist kein Teil von Java, der Java Virtuellen Maschine oder des Java Tool Set. JavaScript ist Objekt-basiert, hat keine Klassen und keine Vererbung.

Ursprünglich war JavaScript als Erweiterung von HTML definiert, für Code, der in ein HTML-Dokument eingebettet, und "on the fly" ausgeführt wird. Beispiel:

```
<script>
function calculate(obj) { .....
.....
}
</script>
```

JavaScript-Programme können auf dem Klienten oder auf dem Server laufen.

Im Gegensatz zu JavaScript steht das Java Applet. Letzteres hat volle Java-Funktionalität und läuft typischerweise innerhalb der Java Virtuellen Maschine eines Browsers. Es wird als Teil der HTML-Seite über das HTTP-Protokoll in den Klienten geladen.

Ein Applet kann eingesetzt werden, um für die nachfolgende Kommunikation zwischen Klienten und Server ein anderes Protokoll zu benutzen, z.B. IIOP, RMI oder 3270. Applets werden heute nur sehr behutsam verwendet. Die (bessere) Alternative ist der Einsatz von Server-seitigen Technologien, besonders Servlets und Java Server Pages.

8.1.5 Dynamischer WEB Seiten Inhalt

Für die Wiedergabe von HTML-Seiten existieren zwei Alternativen:

- Der Web Server sendet eine statische Seite aus der HTML-Datenbank an den Web Browser.
- Der Web Server ruft über eine Schnittstelle ein Anwendungsprogramm auf. Dieses kann z.B. Daten aus einer OS/390 DB2-Datenbank verwenden, um eine dynamische HTML-Seite zu erstellen.

CGI und Java Servlets sind die beiden wichtigsten Schnittstellen für das Anwendungsprogramm einer dynamischen HTML-Seite.

CGI-Programme werden häufig in einer Script-Sprache, z.B. PERL (Practical Extraction and Reporting Language) oder Unix Shell Script erstellt, können aber auch in einer beliebigen anderen Sprache, z.B. C/C++, TCL, Unix Shell Scrip, Visual Basic, Java und anderen geschrieben werden.

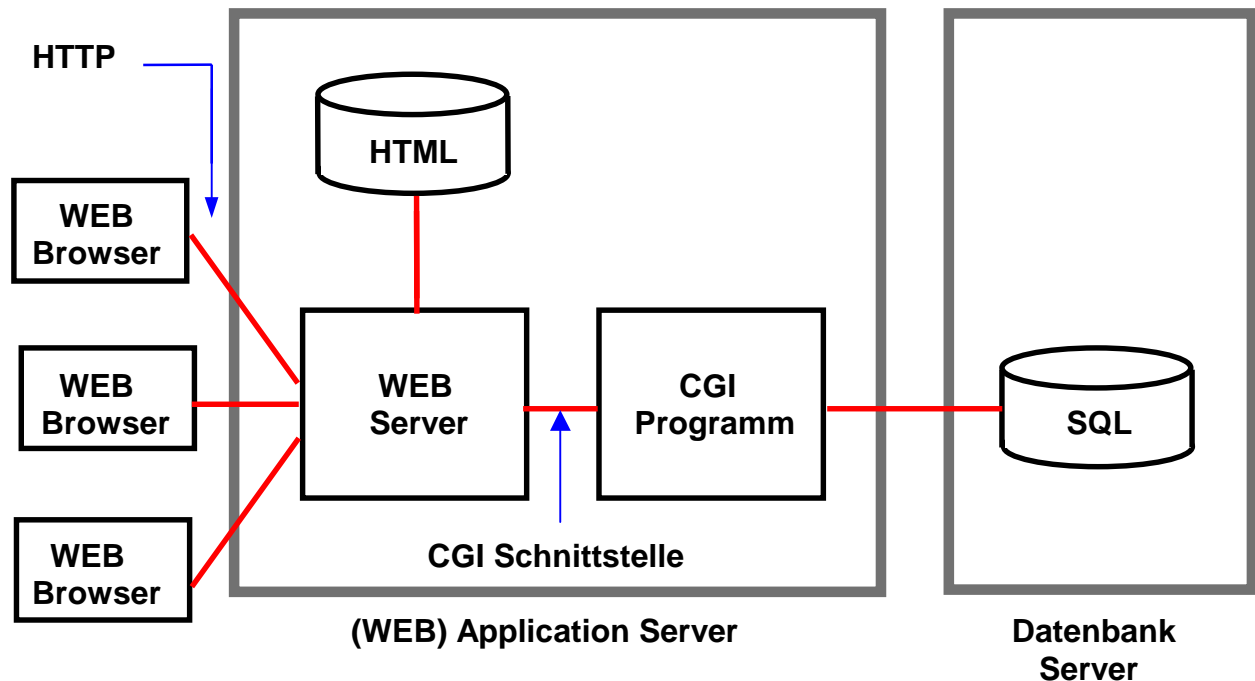


Abb. 8.1.3 Common Gateway Interface (CGI)

Der Web Browser kommuniziert mit dem Web Server über das HyperText Transfer Protocol (HTTP). Der Web Server ruft über die CGI-Schnittstelle ein Anwendungsprogramm auf. Dieses generiert dynamische Seiten in Echtzeit, z.B. indem ein Teil der wiedergegebenen Information aus einer SQL-Datenbank abgefragt wird. Hiermit wird eine dynamische HTML-Seite erstellt. Die Ausgabe geht in der Regel direkt an den Klienten.

8.1.6 Java Servlet

Der moderne Ansatz besteht darin, Java Servlets an Stelle von CGI zu benutzen, siehe Abb. 8.4 . Java Servlets sind normale Java-Klassen, die auf einem Server innerhalb einer standardisierten Laufzeit Umgebung , der "Servlet Engine" oder des „Servlet Containers“, ablaufen. Die Servlet Engine beinhaltet eine normale Java Virtuelle Maschine. Die beiden wichtigsten Vorteile dieses Ansatzes sind:

- Servlets sind vollwertige Java-Programme; sie verfügen über alle Java API's, einschließlich JDBC (Java Data Base Connectivity). Ein Applet kann auf keine Server-seitige Daten zugreifen.
- Im Gegensatz zu CGI erfordert das Java Servlet nur light-weight Context Switches. 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 diesen je nach Bedarf einzelnen konkurrenten Benutzern zuordnen.
- Leistungsfähiges Fehler und Type Checking

Ein Applet kann keine Daten auf eine Klienten- oder Server-Platte schreiben. Ein Servlet ist inder Lage, die ganze Familie von Java API's zu einsetzen, z.B. JDBC für den Zugriff auf eine Relationale Datenbank.

Ein Applet Tag , z.B.

```
<APPLET CODE="HelloApplet.class"></APPLET>
```

in einer HTML-Seite bewirkt das Herunterladen des Applet Byte Code vom Server auf den Klienten, wo der Code ausgeführt wird. Ein Servlet Tag , z.B.

```
<SERVLET CODE="HelloServlet"></SERVLET>
```

in einer HTML-Seite wird auf dem Server ausgeführt

Servlets unterstützen ein Request- und Response-Programmiermodell. Wenn ein Klient ein Request an den Web Server sendet, gibt der Web Server das Request an den Servlet-Container weiter. Das Servlet erzeugt dann ein Response, das der Web Server zurück an den Klienten sendet. Der Klient (normalerweise ein Web Browser) interagiert niemals mit dem Servlet direkt.

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

public
class HalloWeltServlet extends HttpServlet
{
public final static String message = "<html>\n" +
    "<head><title>Hallo Welt</title></head>\n" +
    "<body>\n" +
    "<h1>Hallo Welt</h1>\n" +
    "</body></html>\n";
public void init()
{
System.out.println("In HalloWeltServlet init");
}
public void destroy()
{
System.out.println("In HalloWeltServlet destroy");
}
public void service(ServletRequest req, ServletResponse res)
throws ServletException, IOException
{
PrintWriter out = res.getWriter();
out.println(message);
}
}
```

Abb. 8.1.4 HalloWeltServlet.java

Abb. 8.1.4 zeigt ein einfaches Servlet-Kodierungsbeispiel. Es bewirkt die Ausgabe an den Klienten von einem String mit dem Namen "message". Der String stellt eine normale HTML-Seite dar. Normalerweise ist der String um dynamische Information angereichert, die sich das Servlet z.B. von einer Datenbank geholt hat.

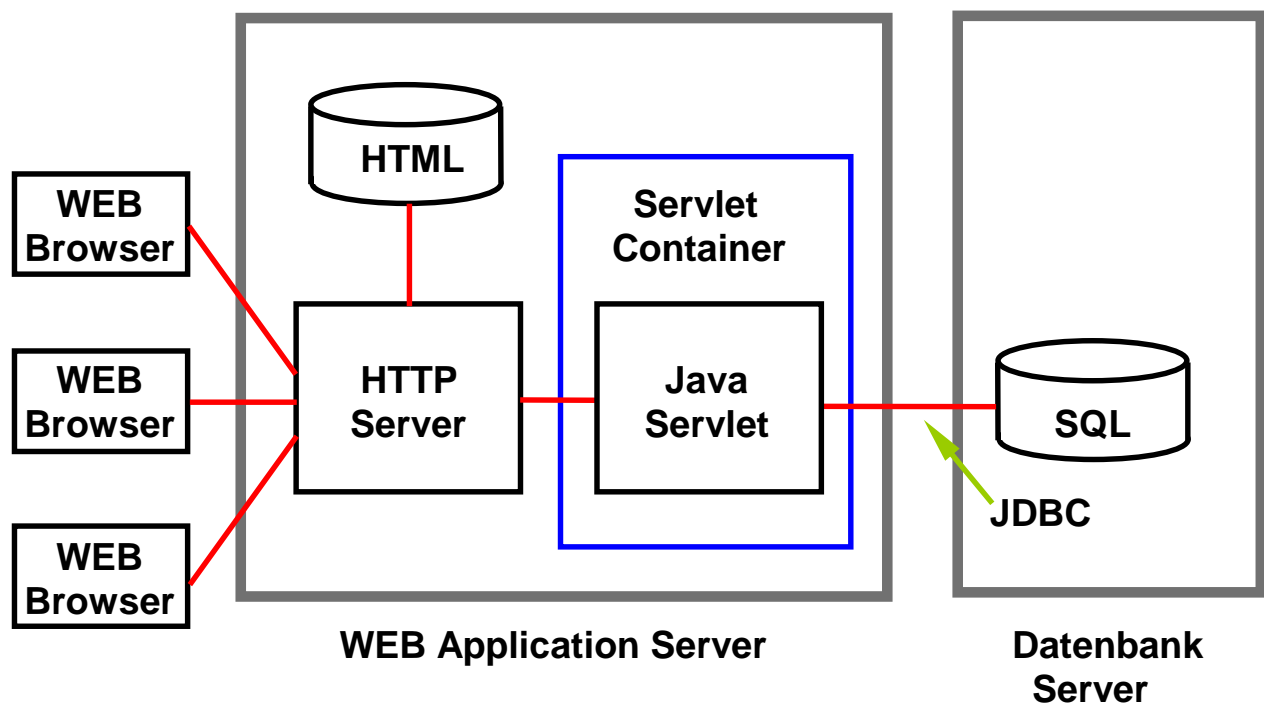


Abb. 8.1.5 Java Servlet und dynamischer WEB-Seiten-Inhalt

Die in Abb. 8.1.5 gezeigte Konfiguration wird als Web Application Server bezeichnet. Der Begriff "Application Server" bezeichnet eine Laufzeitumgebung. Letztere wird von Anwendungen benutzt, die spezifisch für diese Laufzeitumgebung entwickelt wurden. CICS und SAP R/3 sind dafür typische Beispiele. Application Server laufen in der Regel unter einem Standard-Betriebssystem wie z.B. Windows, Linux oder OS/390.

Web Application Server sind Plattformen, auf denen E-Business-Anwendungen installiert werden. Der Zusatz "Application" in einem Web Application Server kennzeichnet die Tatsache, daß der Server nicht mehr nur HTML-Seiten verwaltet, sondern unternehmenskritische Anwendungen ablaufen lassen kann. Ein Web Application Server arbeitet in der Regel mit einer Backend-Komponente zusammen. Diese implementiert entweder ein Datenbanksystem, ein Transaktionsmonitor (CICS, IMS, R/3) oder eine spezifische Backend-Anwendung (siehe Abb. 8.1.5).



Abb. 8.1.5: Zusammenspiel Web Application Server - Back End-System

Die statische HTML Server-Komponente in Abb. 8.1.4 wird als HTTP-Server bezeichnet; in dem hier gezeigten Beispiel besteht der Web Application Server aus zwei Komponenten, dem HTTP-Server und dem Servlet Container.

Der HTTP-Server ist häufig nichts anderes als ein konventioneller Web Server, z.B. Apache, Netscape oder MS IIS. Der Servlet Container wird mit Hilfe von vorhandenen Web Server APIs, wie z.B. der Internet Server API (ISAPI) oder der Netscape Server API (NSAPI) mit dem HTTP-Server verbunden.

Moderne Web Application Server unterstützen neben einem Servlet Container noch einen weiteren Container für die Aufnahme von Java Beans, spezifisch Enterprise Java Beans (siehe Abschnitt 8.3).

Die führenden Web Application Server sind:

BEA WebLogic (Java)
IBM WebSphere (Java)
MS Transaction Server (Visual Basic)

8.1.7 Servlet Container

Servlets laufen in speziellen Servlet-spezifischen Containern, die auch als Servlet Engines bezeichnet werden. Sie verbessern u.a. die Servlet Ausführungszeit. Servlet Containers haben keine Transactions-, Persistence- und Sicherheitseigenschaften. Ein Servlet Container ist ein Programm, das Requests für Servlets und JavaServer 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

Der Servlet Container (Servlet Engine) stellt die Laufzeit-Umgebung für die Servlets dar.

Im Prinzip kann ein Servlet Container als Erweiterung in einen vorhandenen Web Server eingebaut werden. Üblich ist die Bündelung eines Web Servers und eines Servlet Containers zu einem „Web Application Server“. Alle Servlet Container müssen HTTP als das Protokoll für Anfragen und Antworten unterstützen. Häufig unterstützen sie noch weitere Protolle, wie z.B. HTTPS (HTTP über SSL).

In Verbindung mit einem Web Server oder Application Server übersetzt der Servlet Container die Protokolle des Verbindungsnetzwerkes (Internet) in Objekte, die das Servlet versteht. Außerdem ermöglicht es dem Servlet, eine Antwort zu senden. Der Servlet Container verwaltet Servlets während ihres gesamten Lebens-Zyklus.

Ein Container führt alle seine Servlets innerhalb einer einzigen Java Virtuellen Maschine (JVM) aus. Da die Servlets alle in der gleichen JVM laufen, können sie auf gemeinsam genutzte Daten zugreifen, während die JVM private Daten voreinander effektiv schützt. Die Persistenz von Servlets ist in der Zeit zwischen zwei Anfragen möglich; diese beansprucht weit weniger Hauptspeicherplatz als die Implementierung regulärer Prozesse.

Servlet Container haben keine Transaction- und Persistence-Eigenschaften und keine eigenen Sicherheitseigenschaften. Der Servlet Container agiert als ein Request Dispatcher und verwaltet Servlet-Kontexte für die entsprechenden Web-Anwendungen.

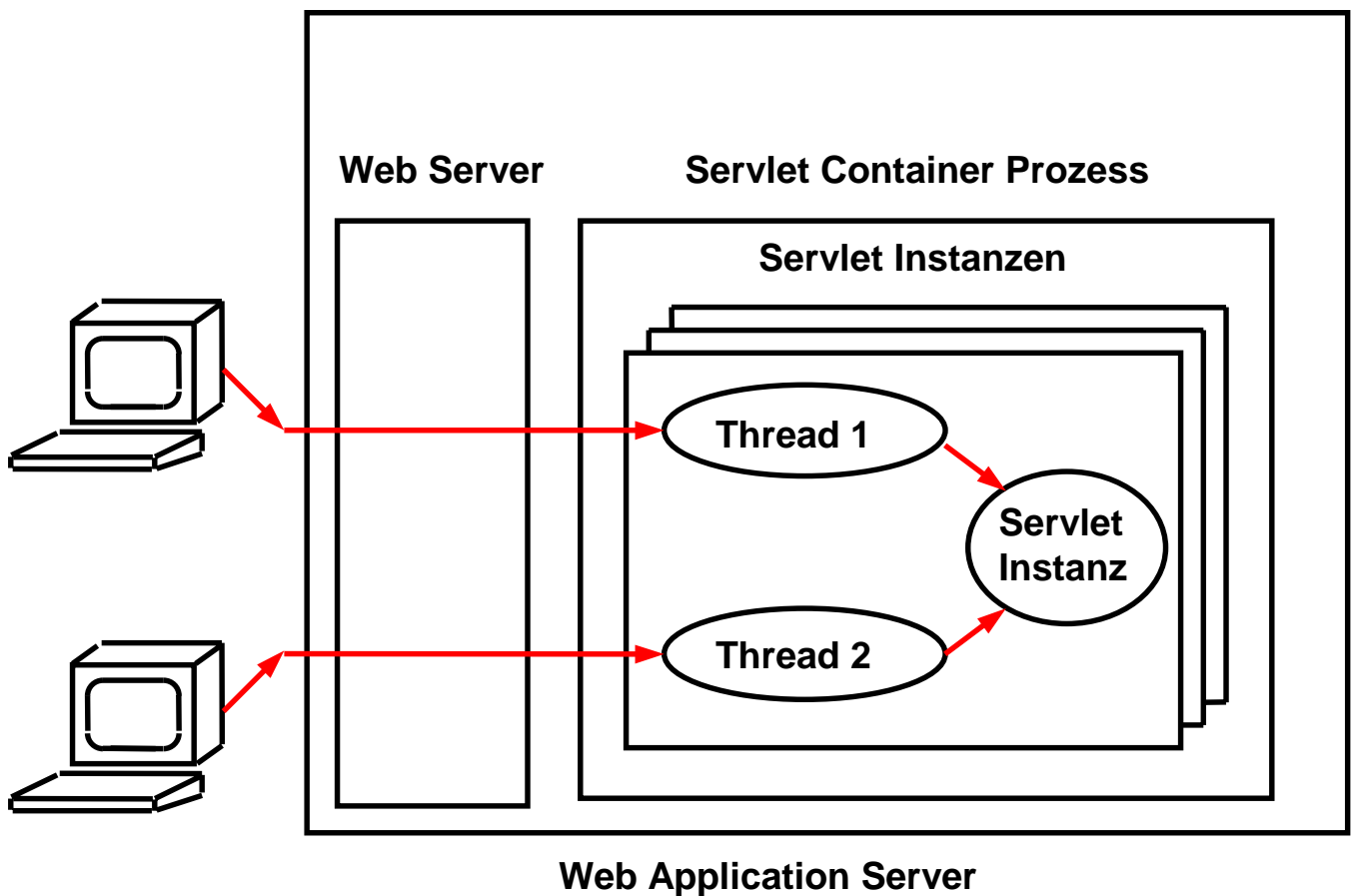


Abb. 8.1.6 Servlet Container

Alle Servlets laufen innerhalb des gleichen Servlet Containers. Die Aufgabe des Container ist es, Servlets zu initialisieren, aufzurufen und bei Bedarf zu beenden (destroy). Jede Servlet-Instanz läuft als ein eigener Thread innerhalb des Servlet Containers.

Es gibt nur eine einzige Instanz des Servlets mit mehrfachen Threads, die geschaffen werden, um mehrere Klienten-Requests gleichzeitig befriedigen zu können. Hieraus resultiert eine effektive Nutzung der Server-Ressourcen. Servlets können dynamisch geladen werden beim erstmaligen Anfordern, oder sie können statisch beim Hochfahren des Containers geladen werden.

8.1.8 Java Server Page (JSP)

Java Server Pages (JSP) sind in der Java Programmiersprache geschrieben. Sie sind eine Erweiterung der Servlet API, und verwenden in Java geschriebene XML - ähnliche Tags und Scriptlets, um die Logik, die den Inhalt der Seite generiert, zu kapseln

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

Abb. 8.1.7 zeigt ein Beispiel für eine einfache Java Server Page.

```

<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
<title>JSP Example 1</title>
<meta http-equiv="Content-Type" content="text/html;
  charset=iso-8859-1">
<meta name="Author" content="Paul Tremblett">
<meta name="GENERATOR" content="Mozilla/4.51 [en] (X11; I;
  Linux 2.2.5-15 i586) [Netscape]">
</head>
<body bgcolor="#FFFFFF">
<p>
<font face="Arial, Helvetica, sans-serif"><b><font size="+2">
JSP Example 1
</font></b></font>
<br>
<br>
<font face = "Arial, Helvetica"><font size="+1">
It is now
<%=
  new java.util.GregorianCalendar(new java.util.SimpleTimeZone
    (-5*60*60*1000,"EDT")).getTime()
  %>
</font>
</body>
</html>

```

Abb. 8.1.7 Einfache Java Server Page

Server-seitige Includes (SSI) erlauben das Einbetten eines Servlet (oder CGI)-Aufrufes innerhalb einer HTML-Seite über die Benutzung eines speziellen `<servlet>` Tags. JSP-Files haben Ähnlichkeit mit Server-seitigen Includes in statischer HTML; beide betten Servlet-Functionalität in eine Web-Seite ein. Jedoch ist in einem Server-seitigem Include ein Servlet mit Hilfe eines speziellen Servlet-Tags implementiert, das den an einer davon getrennten Stelle vorhandenen Servlet-Code aufruft. In einer Java Server Page ist Java Servlet Code (oder anderer Java Code) direkt in die HTML eingebettet.

Mit Hilfe von JSP's kann man den HTML-Code effektiv von der Business Logik der Web Seite trennen. Es ist eine wiederholte Benutzung von Komponenten wie Servlets, Java Beans, Enterprise Beans, und Java-basierten Web-Anwendungen möglich.

Der Trend geht dahin, weniger Applets zu benutzen und nach Möglichkeit alle Logik mit Hilfe von Java Server Pages zu implementieren.

Es existieren zwei Basis-Modelle für die JSP-Nutzung:

1. Ein ASP-like Modell, where a request is made for the JSP page and the page compiled servlet is run with dynamic content filled in.
2. A servlet to JSP model, where the client calls a servlet which generates the dynamic content and then calls a JSP page to be sent in response to the client request. This JSP page gets the dynamic content from the original servlet and then sends the response to the client. This has the benefit of having the creator of dynamic content (the called servlet) be independent of the user of dynamic content (JSP page).

Abb. 8.1.8 zeigt den Ablauf beim Aufruf einer Java Server Page.

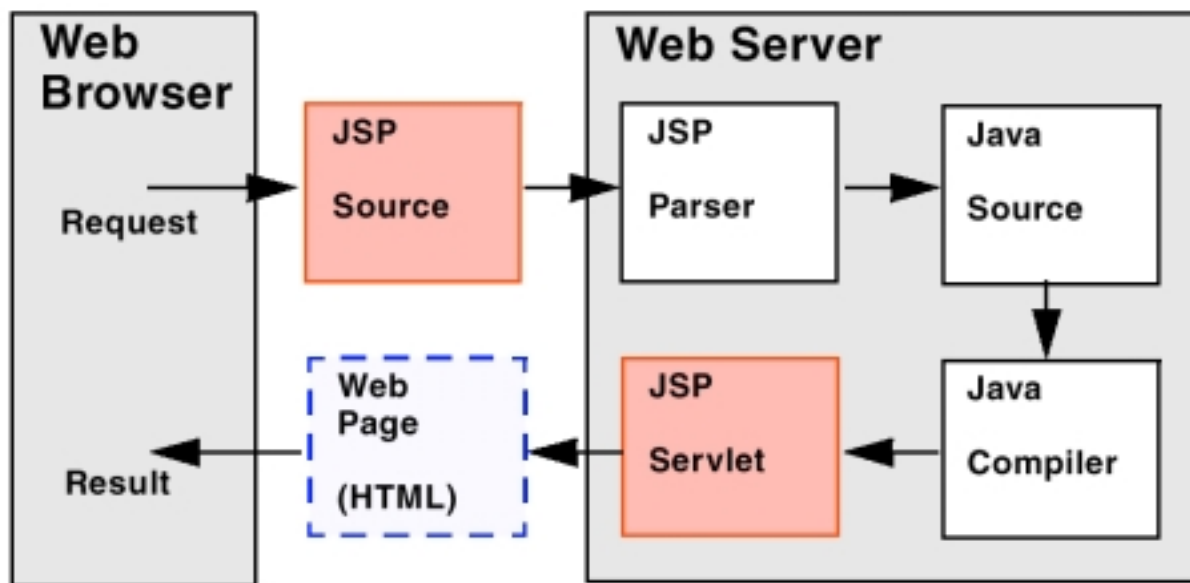


Abb. 8.1.8: Ausführung einer Java Server Page

Beim Aufruf einer JSP werden die folgenden Schritte ausgeführt:

- Der Web Browser sendet eine Request an die JSP-Seite.
- Die JSP Engine parses den Inhalt der JSP File. Sie erstellt temporären Servlet-Quellcode, basierend auf dem Inhalt der JSP.
- Der Servlet-Quellcode wird durch den Java-Compiler in ein Servlet Class File übersetzt.
- Das Servlet wird instantiiert. Die Init- and Service-Methoden des Servlets werden aufgerufen; die Servlet-Logik wird ausgeführt.

Die Kombination von statischem HTML, kombiniert mit den dynamischen Elementen und spezifiziert in der ursprünglichen JSP-Definition, geht an den Web Browser zurück durch den Output Stream des Servlet Response-Objektes.

Abb. 8.1.9 beschreibt das Zusammenspiel von Servlet und JSP für eine Server-seitige Implementierung der Presentation Logic.

Ein Servlet ist ein Java-Programm, das einen Bildschirm-Ausgang in Form einer HTML-Datei produziert. Eine Java Server Page ist eine HTML-Seite mit zusätzlichen JSP Tags. Wird eine JSP-Seite aufgerufen, so wird sie von einem JSP-Übersetzer in ein Servlet übersetzt.

In der Praxis werden Servlets und JSPs von verschiedenen Entwicklern 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.

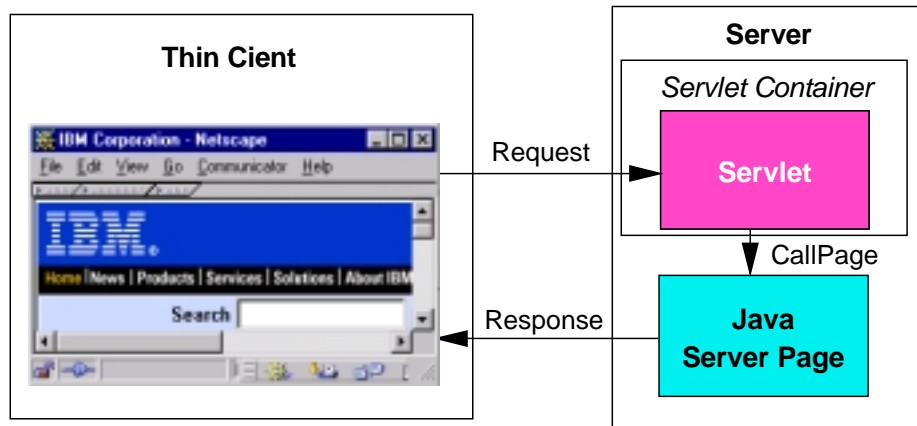


Abb. 8.1.9: Client/Server Szenario mit Servlet und Java Server Page

Es existieren (wie für HTML Seiten) spezielle Werkzeuge für das Erstellen von JSP's, die das Handcoding von HTML-Statements automatisieren.

8.2 Java Beans

8.2.1 Objekte und Komponenten

Eine Client/Server-Anwendung besteht aus Präsentation Logic und Business Logic. Java Server Pages sind geeignet, die Präsentation Logic serverseitig zu implementieren. In einfachen Fällen genügen wenige Zeilen Servlet-Code, um die Business Logic zu implementieren. In komplexeren Situationen kann die Business Logic sehr umfangreich werden. In solchen Fällen bietet es sich an, die objektorientierten Fähigkeiten der Java Sprache einzusetzen, und die Business Logic durch eine Anzahl von Java-Komponenten (Java „Beans“) zu implementieren.

Als Objekt bezeichnen wir eine Menge von Daten, die nur über wohldefinierte Operationen (Methoden) zugänglich sind. Eine Objekt-Klasse (Objekt-Typ) spezifiziert Operationen und Datenstrukturen für gleichartige Objekte. Ein Objekttyp ist die abstrakte Spezifikation der Funktionalität; eine Klasse ist die Implementierung dieser Funktionalität.

Eine Objekt-Instanz (Objekt-Exemplar) ist die Ausprägung eines Objektes, das zu einer (vorher definierten) Klasse gehört.

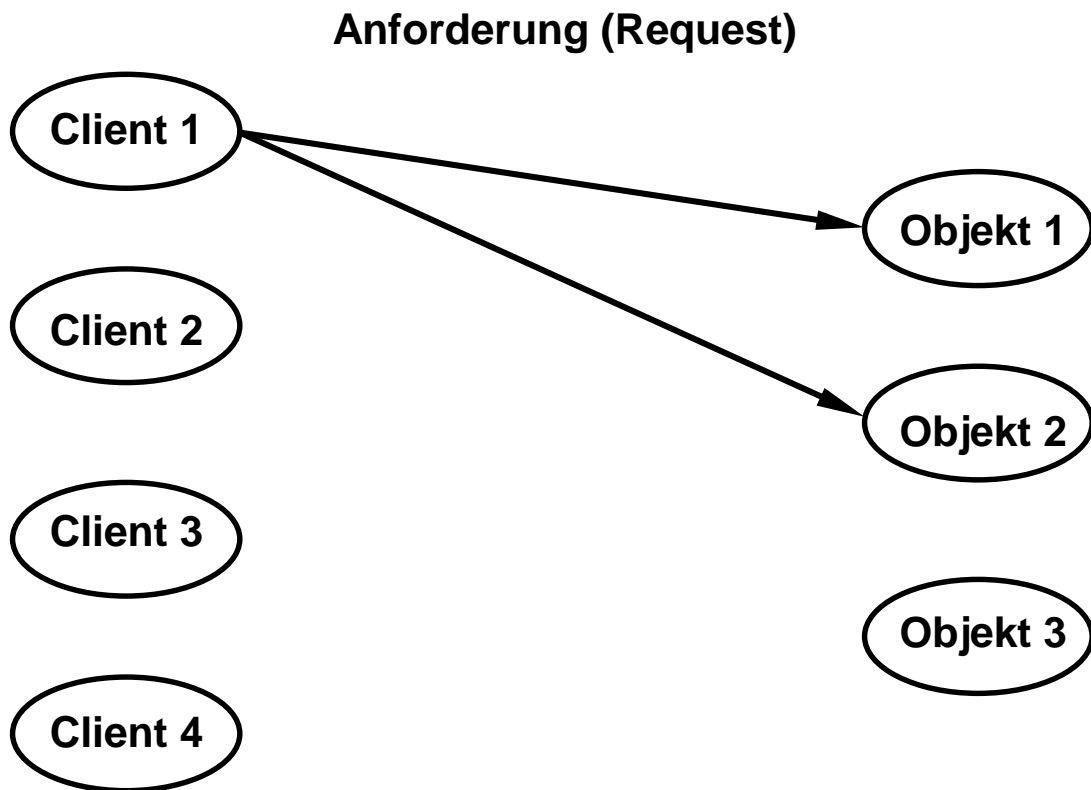


Abb. 8.2.1 Objektorientierte Client/Server-Verarbeitung.

Objekte bieten Operationen an, durch deren Aufruf sie zu bestimmten Aktionen veranlasst werden können. Ein Klient ist eine Software-Einheit, die eine Operation eines Objektes aufruft.

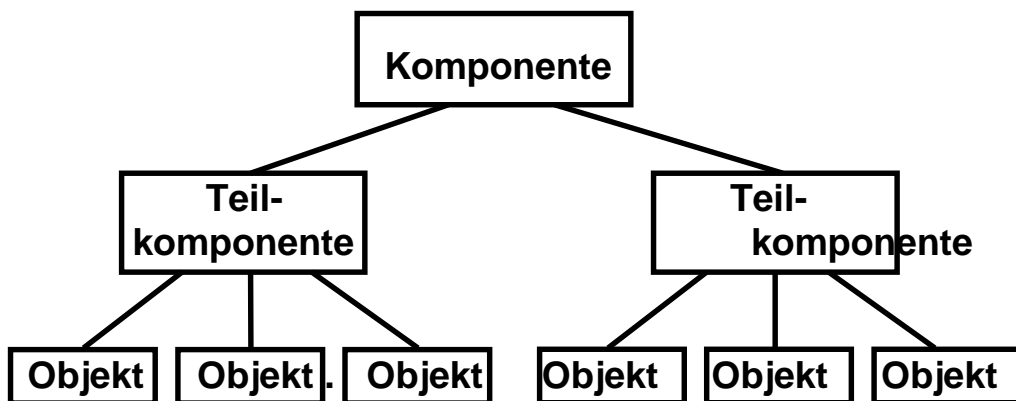


Abb. 8.2.2 Komponenten

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 Komponenten setzen sich typischerweise aus Objekten zusammen, die wiederum Komponenten oder eigene Anwendungen sind.

Komponenten lassen sich durch geeignete Parameterisierung in einer Vielzahl von Entwicklungsumgebungen einsetzen und sind unabhängig von Sprache, Betriebssystem und Hardware.

Eine Komponente hat

- eine Art "Stecker", mit dem sie sich verbinden kann.
- eine Art "Steckdose", welche benutzt wird, um verschiedenen Komponenten die Möglichkeit zu geben, sich "einzustecken".
- die Möglichkeit, Informationen über sich selbst bekannt zu geben.
- eine spezifizierte Menge von Eigenschaften

8.2.2 Java Beans-Komponenten-Modell

Java Beans sind einzelne Softwarekomponenten, aus denen sich schlüsselfertige Java-Anwendungen zusammensetzen lassen. Sie repräsentieren ein objektorientiertes Java-Komponenten-Modell. Das Zusammensetzen der Komponenten erfolgt dabei in der Regel mit visuellen Werkzeugen, also grafischen Editoren. In einem solchen grafischen Editor kann der Benutzer die verschiedenen Komponenten einsetzen, zu einem "großen Ganzen" zusammensetzen und modifizieren. Bei Java Beans handelt es sich häufig um visuelle Komponenten, angefangen bei einzelnen Elementen für grafische User-Interfaces, etwa Buttons und Scrollbalken, bis hin zu komplexen Tabellenkalkulationen oder Textverarbeitungen.

Andere Komponenten, die keine grafischen Aspekte haben (z. B. Module) bzw. den Zugriff auf eine Datenbank ermöglichen, werden meistens nicht durch einfache Beans nachgebildet. Allerdings ist es durchaus möglich, Beans zu entwickeln, die keine grafische Repräsentation haben.

Hauptmerkmale von Java Bean sind:

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

Java Beans haben **Eigenschaften** (Properties), die mit speziellen **Methoden**, den "get"- und "set"-Methoden, ausgelesen und modifiziert werden können. Eine "get"-Methode zu einem Attribut liefert den Wert dieses Attributs zurück und eine "set"-Methode zu einem Attribut setzt den Wert dieses Attributs. So könnte zum Beispiel ein Button Bean (eine Softwarekomponente), das einen Button realisiert, eine Vorder- und eine Hintergrundfarbe und eine Beschriftung als Eigenschaften haben.

Außerdem kann ein Bean mit der Umgebung durch **Ereignisse** kommunizieren, d.h. ein Bean kann von sich aus einen Event produzieren, und die Umgebung, z. B. auch andere Beans können darauf reagieren. Für ein Button Bean ist es sinnvoll, einen Event auszulösen, sobald der Benutzer auf den Button geklickt hat.

Andere Objekte haben dann die Möglichkeit, sich bei dem Bean als sogenannter "Event-Listener" registrieren zu lassen. An alle Objekte, die als "Event-Listener" bei dem Bean registrieren wurden, wird dann dieses Ereignis geschickt. Wenn also ein Objekt auf das Klicken eines Buttons reagieren will, dann wird es sich bei dem Button als "Event-Listener" anmelden und entsprechend informiert werden, sobald das Ereignis eintritt.

Das Java Beans-Komponenten-Modell wird spezifiziert, sobald die Komponenten ihre Eigenschaften, Methoden und Events bekannt geben. Dies geschieht über **Namens-Konventionen**, auch als Java Beans Design Patterns bezeichnet.

Introspection unterscheidet Beans von normalen Java-Klassen. Ein Entwickler kann eine BeanInfo-Klasse definieren, die über Descriptors Bean Introspection Informationen zur Verfügung stellt (teilweise oder ganz).

Praktisch alle Konventionen sind optional; ein Java Bean wird nicht aus einer universellen Basis-

Klasse abgeleitet, die definierte Eigenschaften aufweist..

Das Java Beans-Komponenten-Modell (Teil des SunJDK-Lieferumfangs) unterstützt:

- Sicherheit (benutzt den Java Security Manager)
- Versionsmanagement
- Life-Cycle Management
- Event Notification,
- Configuration und Property Management
- Scripting
- Meta-Daten und Introspection
- Persistenz (über Serialisierung)
- Benutzbarkeit (die "BeanBox" des JDK ist ein Prototyp einer grafischen Umgebung für das Zusammensetzen von Beans)
- Eigeninstallation(über Java Archiv Files)

8.2.3 Model View Controller (MVC) Ansatz

Im einfachsten Fall enthält das Java Servlet die Anwendungslogik. In komplexeren Fällen lohnt es sich, die Anwendung in Komponentenform zu implementieren. Java Beans implementieren das Java-Komponentenmodell.

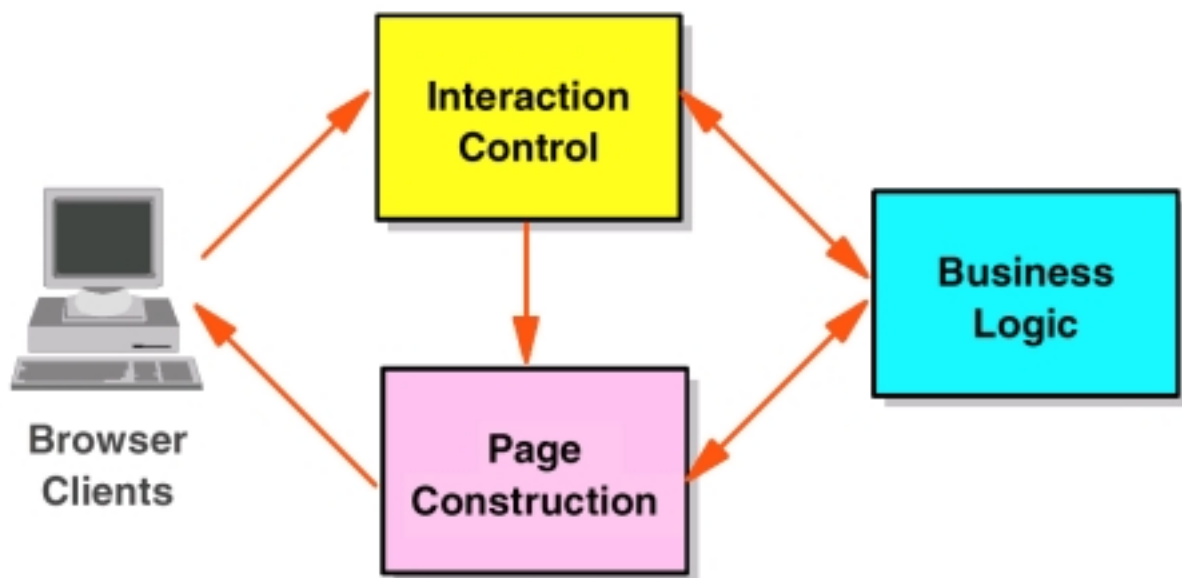


Abb. 8.2.3 Model – View – Controller-Triade

Der Model-View-Controller-Ansatz (auch als Model – View – Controller-Triade bezeichnet) erweitert die gebräuchliche Aufteilung einer Client/Server-Anwendung in Präsentation- und Business-Logik um eine weitere Komponente, den Controller. Der Controller ist für die Ablaufsteuerung zwischen Präsentation- und Business-Logik zuständig. Diese Situation ist in Abb. 8.2.3 dargestellt.

Abb. 8.2.4 zeigt die MVC-Implementierung mit Hilfe von Servlets, Java Server Pages und Java Beans.

Command- und Data-Beans (plus eventuelle CICS- oder MQSeriesProgramme, oder Stored Procedures) sind das „Modell“ (=Business Logik). Der „View“ ist die durch JSP's und View Beans dargestellte Präsentation-Logik. Das Servlet ist der „Controller“. Die Business-Logik besteht aus den mit Hilfe mehrerer Beans implementierten Java-Komponenten (plus evtl zusätzlichen Backend-Komponenten wie CICS oder DB2 Stored Procedures).

Ein entsprechend der MVC-Architektur entworfenes System besteht aus lose gekoppelten Teilen, die über sauber definierte Schnittstellen verbunden sind. Es ist deshalb relativ einfach, Komponenten abzuändern, ohne daß der Rest des Systems dadurch beeinflusst wird.

"Pattern" ist ein im Rahmen der MVC-Architektur häufig auftretender Begriff. Ein Pattern ist die Abstraktion eines konkreten Programms, das in dieser Form in neuen Anwendungen immer wieder auftritt. Man benutzt ein Pattern als ein Skelett für die Erstellung einer neuen Anwendung.

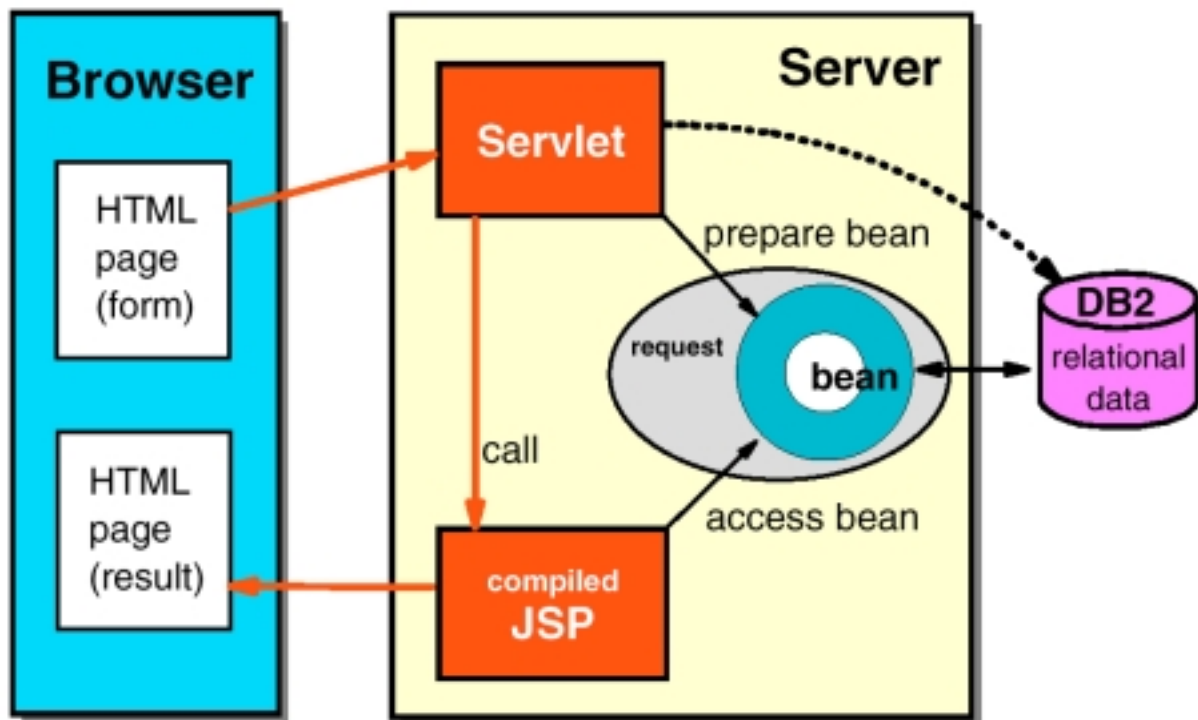


Abb. 8.2.4 Zusammenspiel von Servlet und JSP

Die MVC-Ablaufsteuerung in Abb. 8.2.5 erfolgt in den folgenden Schritten:

1. Eine HTML-Seite (static oder dynamic), in einer vorhergehenden Request/Response-Interaktion erstellt und an den Klienten übertragen, enthält eine oder mehrere Forms, die ein Servlet für die Verarbeitung der nächsten Interaktion aufrufen.
2. Das Servlet übernimmt Validierung und Flußsteuerung. Es ruft Command Beans auf, welche die Business-Logik beinhalten.
3. Command Beans steuern die Verarbeitung der Business-Logik. Die Logik kann in das Command Bean eingebettet sein. Alternativ kann sie an ein Backend -System delegiert werden, z.B. Relationale Datenbanken, Transactionssysteme (CICS, MQSeries, IMS, usw.). Command Beans mögen alternativ eine spezifische Funktion ausführen, oder sie können viele Methoden, jede für eine spezifische Aufgabe, enthalten (task wrappers). Command Beans rufen Datenbanken oder Transaktionssysteme über „Connectoren“ auf.
4. Die Ergebnisse der Verarbeitung durch Command Beans (oder Back-End-Systeme) werden in Data Beans gespeichert. Data Beans können z.B. eine SQL-Zeile oder eine CICS Communication Area enthalten.
5. View Beans enthalten den Vertrag zwischen den (die Ausgabe produzierenden) JSPs und den Data Beans. Letztere enthalten die dynamischen Ausgabedaten. Das Servlet initialisiert die View Beans und registriert sie mit einem Request-Block, über den die JSPs diese finden kann.

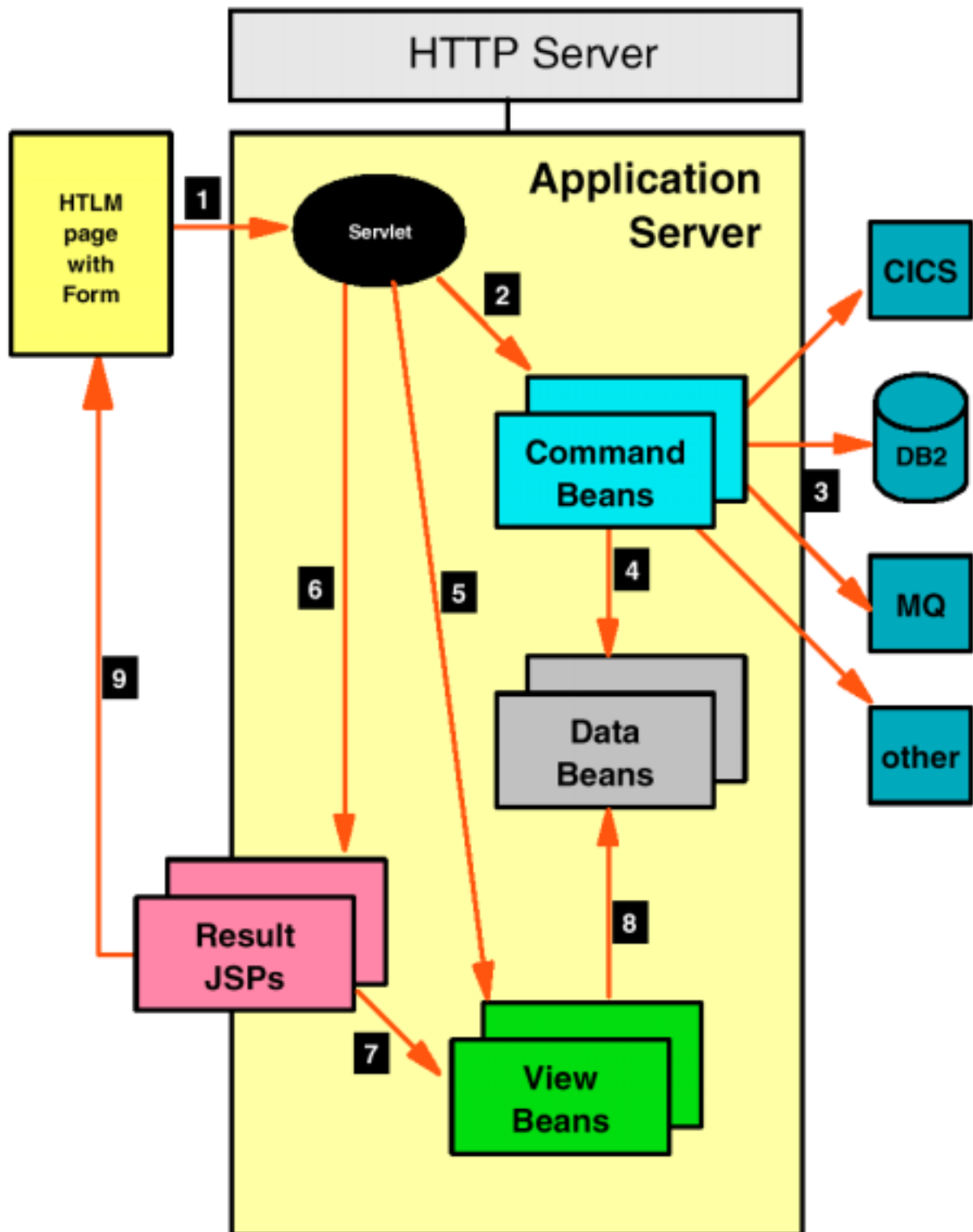


Abb. 8.2.5: Architektur einer JSP-Web-Anwendung

6. Das Servlet ruft eine JSP für die Ausgabe-Verarbeitung und Formatierung auf, abhängig vom Ergebnis der Command Beans. JSPs generieren die Ausgabe für den Browser.

7. JSPs verwenden Tags für die Deklaration der View Beans und retten dynamische Daten.

8. View Beans enthalten ein oder mehrere Data Beans. Sie stellen Methoden zur Verfügung, die es den JSPs ermöglichen, auf die Daten der Data Beans zuzugreifen. Die Data Beans selbst enthalten nicht die hierfür erforderlichen Methoden.

9. Die JSP assembliert die Ausgabe und sendet sie an den Browser in der Form einer HTML-Seite mit dynamischen Daten als Inhalt. In vielen Fällen kann diese Seite wiederum Forms enthalten, die es dem Benutzer ermöglichen, den Dialog mit der Anwendung fortzusetzen.

8.3 Enterprise Java Beans

8.3.1 Dynamischer WEB-Seiten-Inhalt

Java Beans werden meistens benutzt, um innerhalb des Web Browsers, z.B. als Teil eines Applets, graphisch ansprechende Darstellungen wie z.B. Scroll Bars oder Push Buttons auf einer GUI darzustellen. Es sind wiederverwendbare Komponenten, die z.B. mit einem Builder Tool wie Jbuilder von Borland oder VisualAge von IBM manipuliert werden können. Prinzipiell können sie ebenfalls eingesetzt werden, um Servlets oder Business-Logik in Komponentenform zu implementieren. Java Beans implementieren das Java-Komponentenmodell.

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

EJBs sind Java Beans mit zusätzlicher Funktionalität, besonders Transaktionseigenschaften (ACID), Persistenz und Sicherheit. Sie stellen eine serverseitige (im Gegensatz zu Applets) Komponenten-Architektur dar, die ein transaktionsorientiertes Programmiermodell für persistente Objekte darstellt. EJB's werden eingesetzt, um die die Backend-Logik zu entwickeln, d.h. diejenige Business Logic die benötigt wird, um von einem Browser intelligent auf zentrale Datenbanken zugreifen zu können.

Ein Enterprise Java Bean ist ein nichtvisuelles Java Bean. Es ist ein Business-Objekt, entweder ein Gegenstand mit hoher Lebensdauer wie ein Bank-Konto, oder eine kurzfristig bestehende Einheit wie ein „Besuch bei der Bank“. Im Gegensatz zu einem einfachen Java Bean existiert ein Enterprise Java Bean innerhalb eines EJB Containers, der wiederum innerhalb eines Web Application Servers läuft.

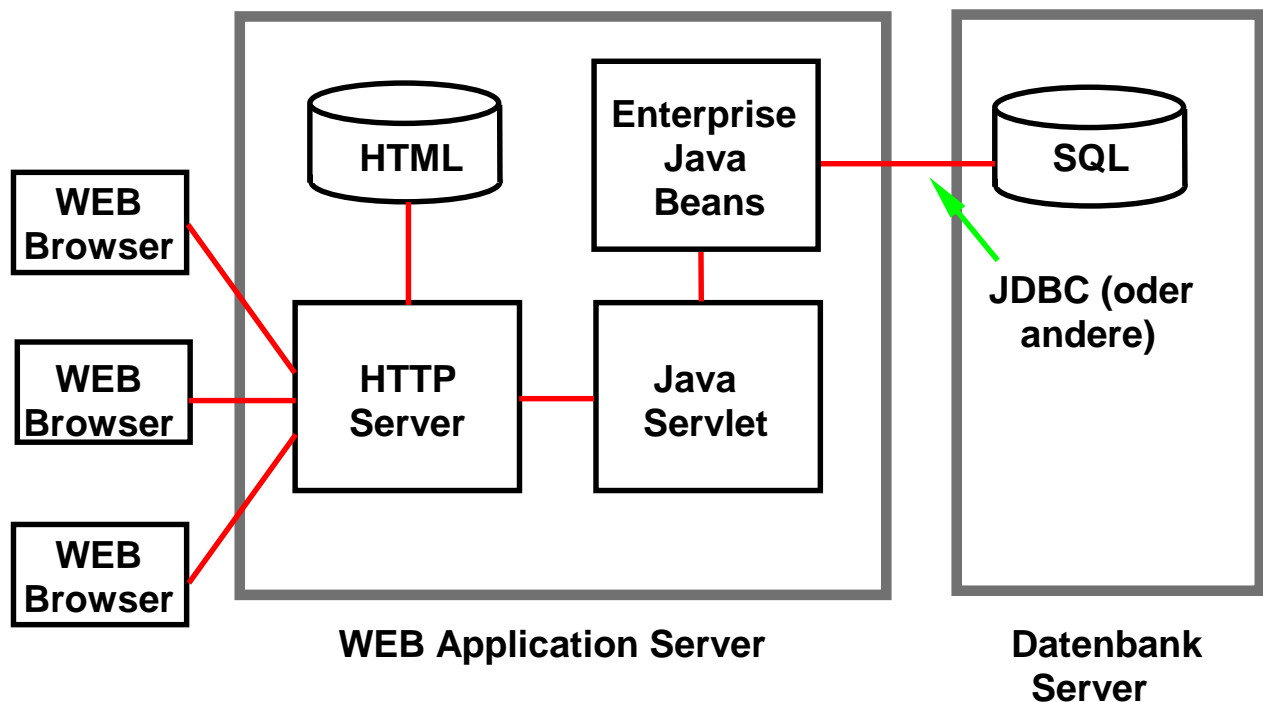


Abb. 8.3.1 Web Application Server

Abb. 8.3.1 zeigt die Struktur eines EJB-fähigen Web Application Servers. Letzterer besteht hauptsächlich aus drei Komponenten:

- Web Server (HTTP Server)

- Servlet Engine (auch als Servlet Container bezeichnet)
- EJB „Container“

Bei der Servlet Engine und dem EJB Container handelt es sich um getrennte Laufzeitumgebungen für Servlets bzw. EJB's. Ein Web Application Server-Produkt hat in der Regel keinen eigenen Web Server, sondern verwendet statt dessen einen regulären Web Server wie z.B. Apache, der als „Plug-In“ in den Web Application Server eingebettet wird.

Web Application Server sind der wichtigste Baustein für die Erstellung von neuen, Java- und Corba-basierten Internet-Anwendungen. Es existiert eine Java-Referenz-Implementierung der Firma Sun. Zahlreiche Firmen stellen Web Application Server als Software-Produkte her. Die wichtigsten und meistverwendeten sind WebLogic (Bea), WebSphere (IBM) und Iplanet (Sun). In Deutschland haben Unternehmen wie SAP, Brokat und Intershop ihre eigenen Web Application Server entwickelt.

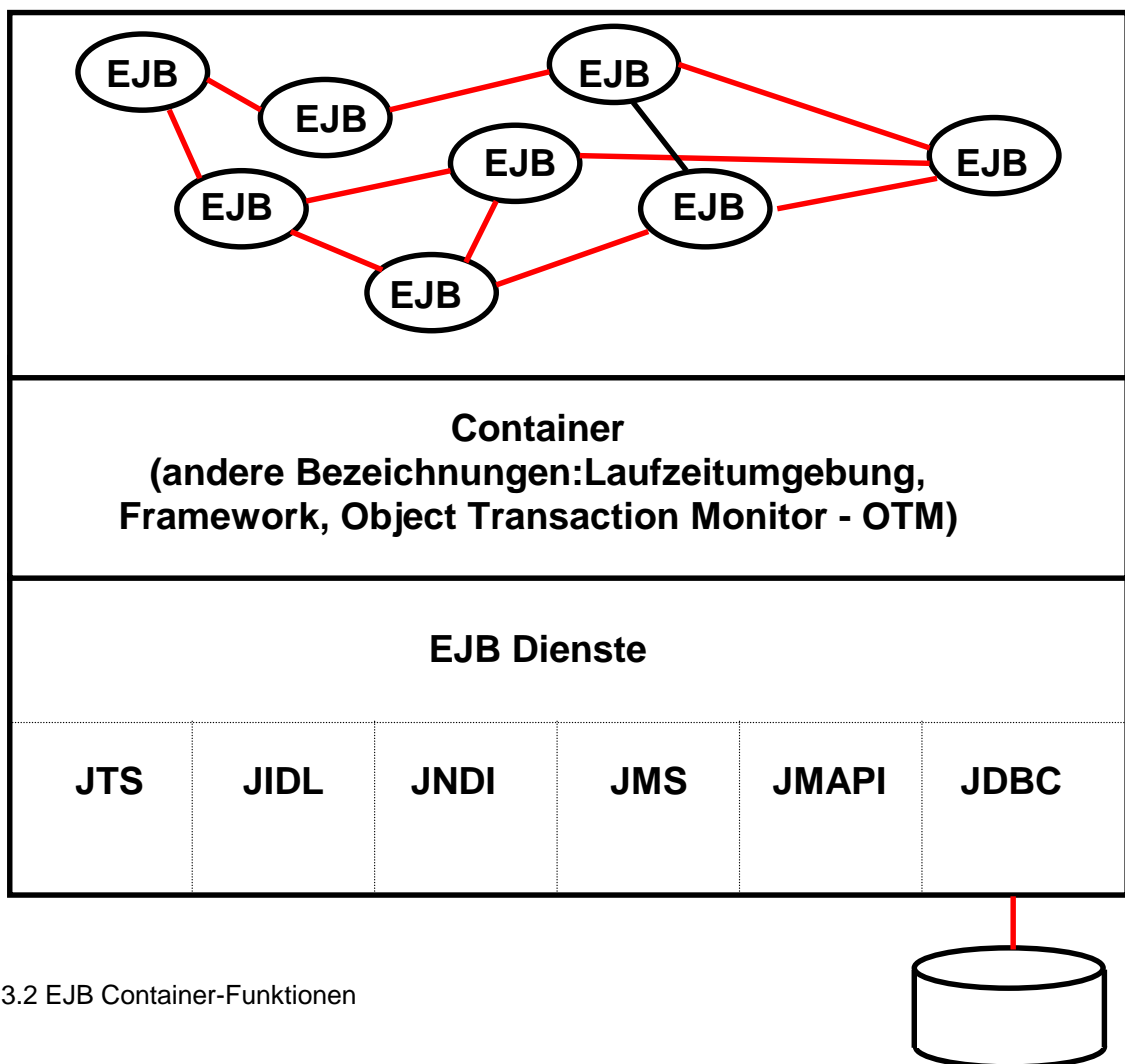


Abb. 8.3.2 EJB Container-Funktionen

Enterprise Java Beans sind Java Beans mit erweiterter Funktionalität. Diese Funktionalität wird von dem EJB Container zur Verfügung gestellt. Hinter den in Abb.: 8.3.2 wiedergegebenen Abkürzungen verbergen sich die folgenden Eigenschaften:

- JTS ist der Java Transaction Service, eine API für die Inanspruchnahme von Transaktionsdiensten.
- JNDI, das Java Naming und Directory Interface, ist eine API für den Zugriff auf Namens- und Directory-Dienste.
- Mit JMS, dem Java Message Service, werden asynchrone Nachrichten ausgetauscht.

- JDBC , das Java Database Connectivity API, greift auf die Daten in existierenden Datenbanken über eine genormte Schnittstelle zu.
- JMAPI bezieht sich auf das Java Management API. Dieses definiert den Zugriff auf Dienste für das Management von Java-Ressourcen.
- JIDL ist die Java Interface Definition Language, eine CORBA-Schnittstelle für das Arbeiten mit verteilten Objekten, die in unterschiedlichen Sprachen implementiert sind.

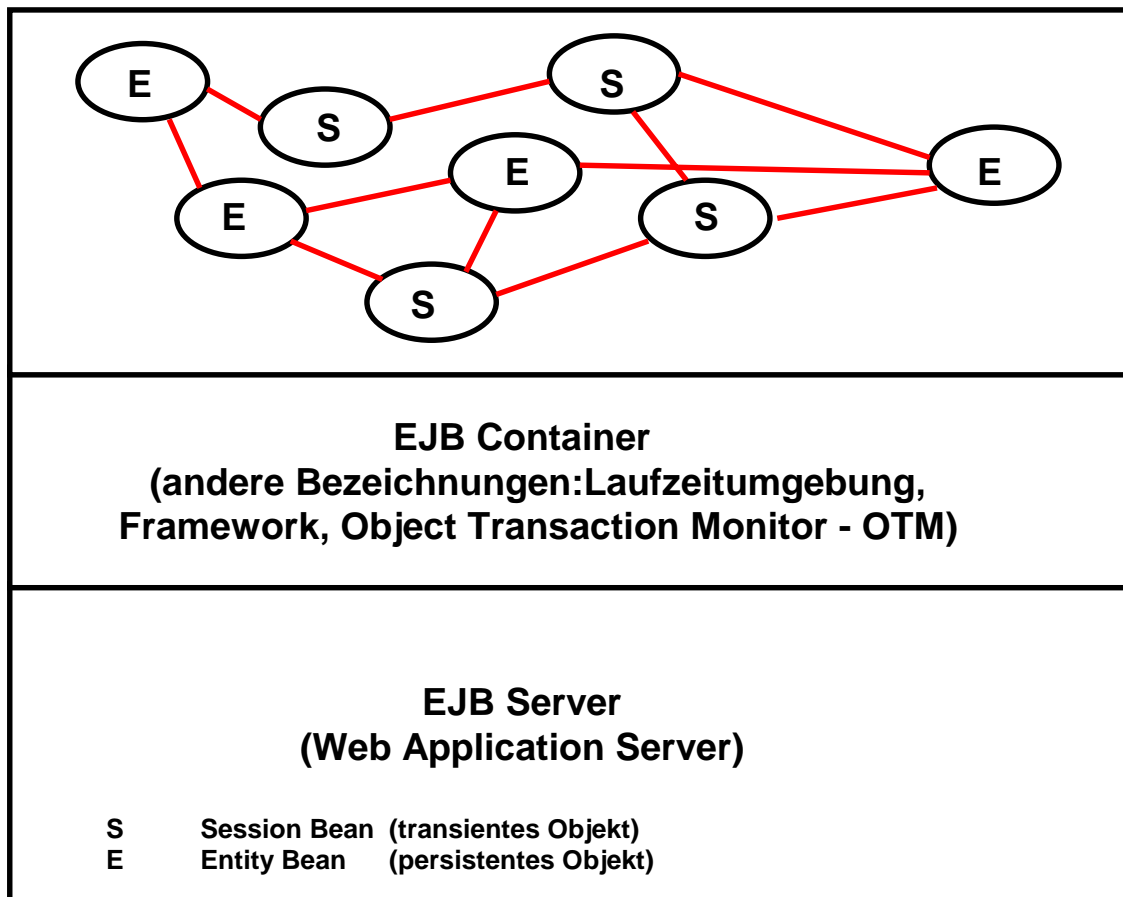


Abb. 8.3.3: Arten von EJBs

Wir unterscheiden zwei Arten von Enterprise Java Beans:

Entity Beans:

- repräsentieren spezifische Daten (z.B. eine Reihe in einer SQL-Datenbank),
- Methoden ermöglichen die Manipulation der Daten, welche das Bean repräsentiert,
- überleben, so lange die Daten in der Datenbank überleben.

Entity Beans benutzen einen Transaktionsmanager, der sicherstellt, daß sich der Zustand des Beans auf eine konsistente Art ändert. Der Zustand eines Entity Bean wird häufig durch eine Zeile in einer relationalen Tabelle dargestellt.

Session Beans:

- können den internen Zustand einer Session speichern, der aber nicht persistent ist,
- sind einem bestimmten Klienten für die Dauer einer Sitzung (Session) zugeordnet. Sie werden bei Beendigung der Sitzung zerstört.

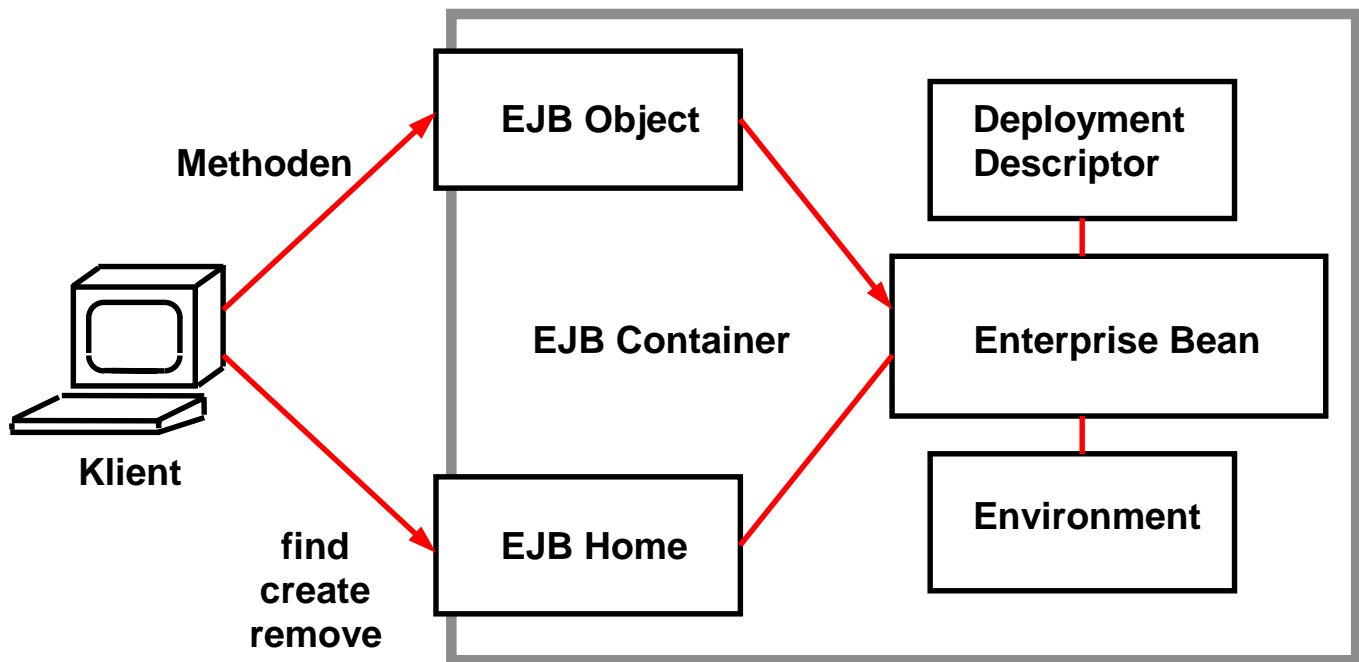


Abb. 8.3.4: EJB-Schnittstellen

Eine Enterprise Java Bean verfügt über zwei Standard Schnittstellen, siehe Abb. 8.3.4 . Die „EJB Home“-Schnittstelle dient der Identifizierung des Beans. Auf die EJB Home-Schnittstelle kann mit JNDI zugegriffen werden. Sie implementiert alle Life-Cycle-Dienste für das Bean.

Die „EJB Object“-Schnittstelle empfängt alle Methodenaufrufe. Sie implementiert die Transaktions-Zustandsverwaltungs, Persistenz- und Sicherheitsdienste für das Bean je nach den Angaben in einem „Deployment Descriptor“. Letzterer ist dem EJB fest zugeordnet. Die Angaben in dem Deployment Descriptor werden typischerweise durch den EJB-Entwickler angelegt. Sie können später durch einen Administrator mit Hilfe eines Tools abgeändert werden.

Der Deployment Descriptor legt die Laufzeit-Parameter eines EJB fest. Parameter können statisch zur Übersetzungszeit oder dynamisch zur Laufzeit festgelegt werden. Beispiele für Parameter sind:

- Datenbank-Name
- Verbindung zu Legacy-Anwendungen
- JNDI Namensraum des Containers
- Transaktions-Semantik
- Umgebungseigenschaften

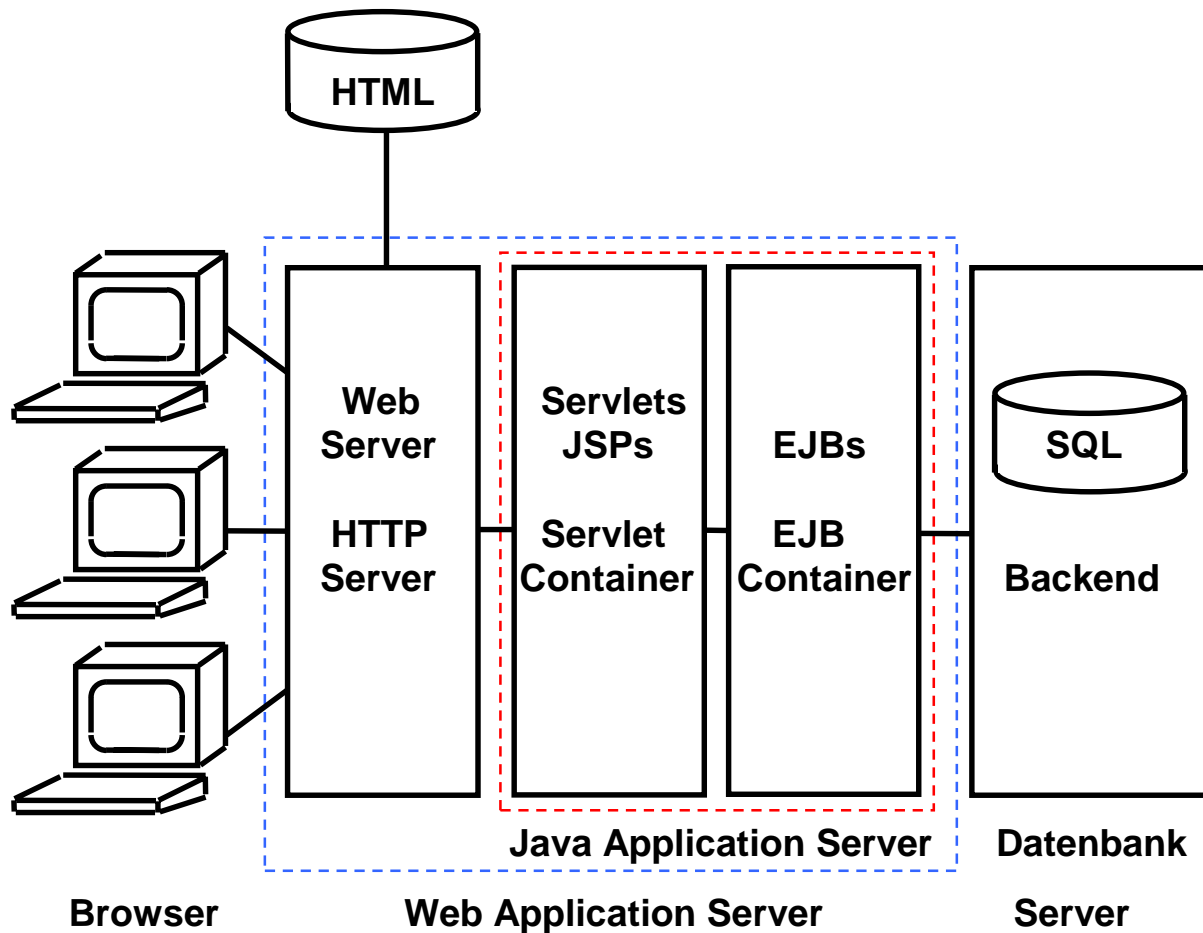


Abb. 8.3.5: Web Application Server

Die Funktionalität von Web-Server (auch als HTTP-Server bezeichnet), Servlet-Container (Servlet Engine) und EJB-Container wird von zahlreichen Herstellern zu einer einzigen Funktionsgruppe zusammengeschlossen und als „Web Application Server“ bezeichnet. Web Application Server werden von einer ganzen Reihe von Herstellern vertrieben. Beispiele sind:

- BEA Weblogic
- Brokat Twister
- IBM WebSphere
- Inprise
- Iona Orbix
- Silverstream
- Sun IPlanet

Hierbei besteht der Web Application Server aus zwei Komponenten, einem eigentlichen Java Application Server, der die Servlet Container- und EJB Container-Funktionalität umfaßt, sowie einem Web Server. Der Web Server kann alleine die Requests nach statischen HTML-Seiten aus seinem lokalen HTML-Speicher befriedigen; er leitet außerdem Server-seitige Includes an den Servlet Container weiter. Die Web Server sind die gleichen Komponenten, die auch freistehend benutzt werden, z.B. Apache, Microsoft IIS oder IBM HTTP-Server. Ein Web Application Server wird in der Regel mit einem vorinstallierten Web Server ausgeliefert. Der vorinstallierte Web Server kann aber leicht gegen einen anderen ausgewechselt werden (plug-in).

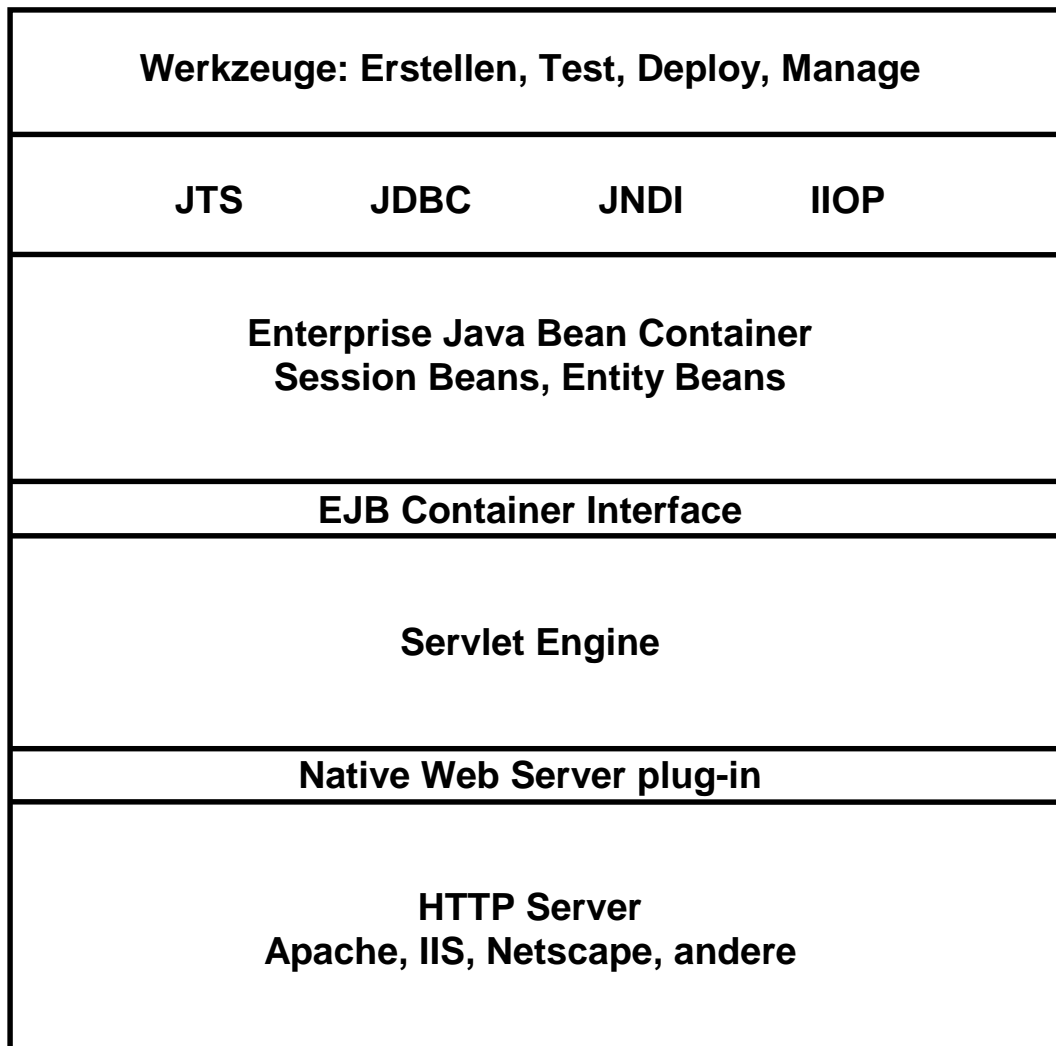


Abb. 8.3.6: Struktur eines Web Application Servers

Abb. 8.3.6 zeigt die interne Struktur eines Web Application Servers. Dies ist die Aufgabenverteilung:

- Der Web Server betreut statische HTML-Seiten, CGI und proprietäre Plug-ins
- Die Servlet Engine behandelt Java Servlet-Anforderungen und dynamische Seitenanforderungen (JSP)
- Die Servlets befriedigen HTTP-Anforderungen, verwalten HTTP Sessions mit dem Klienten, erzeugen Presentation Logic via HTML, bewältigt nicht-transaktionale (Business) Anwendungen
- Session und Entity Beans bearbeiten (Business Logic) Anwendungen mit transaktionaler Integrität
- Die wichtigsten Werkzeuge (Tools) für den WebSphere Web Application Server sind die "VisualAge"-Entwicklungsumgebung und "WebSphere Studio" für die Entwicklung der Presentation Services. Zahlreiche weitere Tools sind von anderen Herstellern verfügbar.

Ein wichtiges Thema für einen Web Application Server ist seine Skalierbarkeit. Ein typischer Windows 2000 Web Server ist in der Lage, 600 statische Zugriffe pro Sekunde oder 100 Java-Zugriffe pro Sekunde auszuführen. In einem mittleren Unternehmen findet man häufig eine Spitzenbelastung von mehreren tausend dynamischen Zugriffen pro Sekunde. Dabei ist das Verhältnis von Spitzen- und Durchschnittsbelastung häufig 5 : 1, auch ein Verhältnis 10 : 1 ist nicht selten.

Beispielsweise kann unmittelbar nach einer Fernsehwerbung für ein e-Commerce-Unternehmen die Belastung dramatisch anwachsen. Wir erwarten weiterhin ein Wachstum der Belastung um einen Faktor 10 innerhalb weniger Jahre.

8.4 Funktionalität und Leistungsverhalten

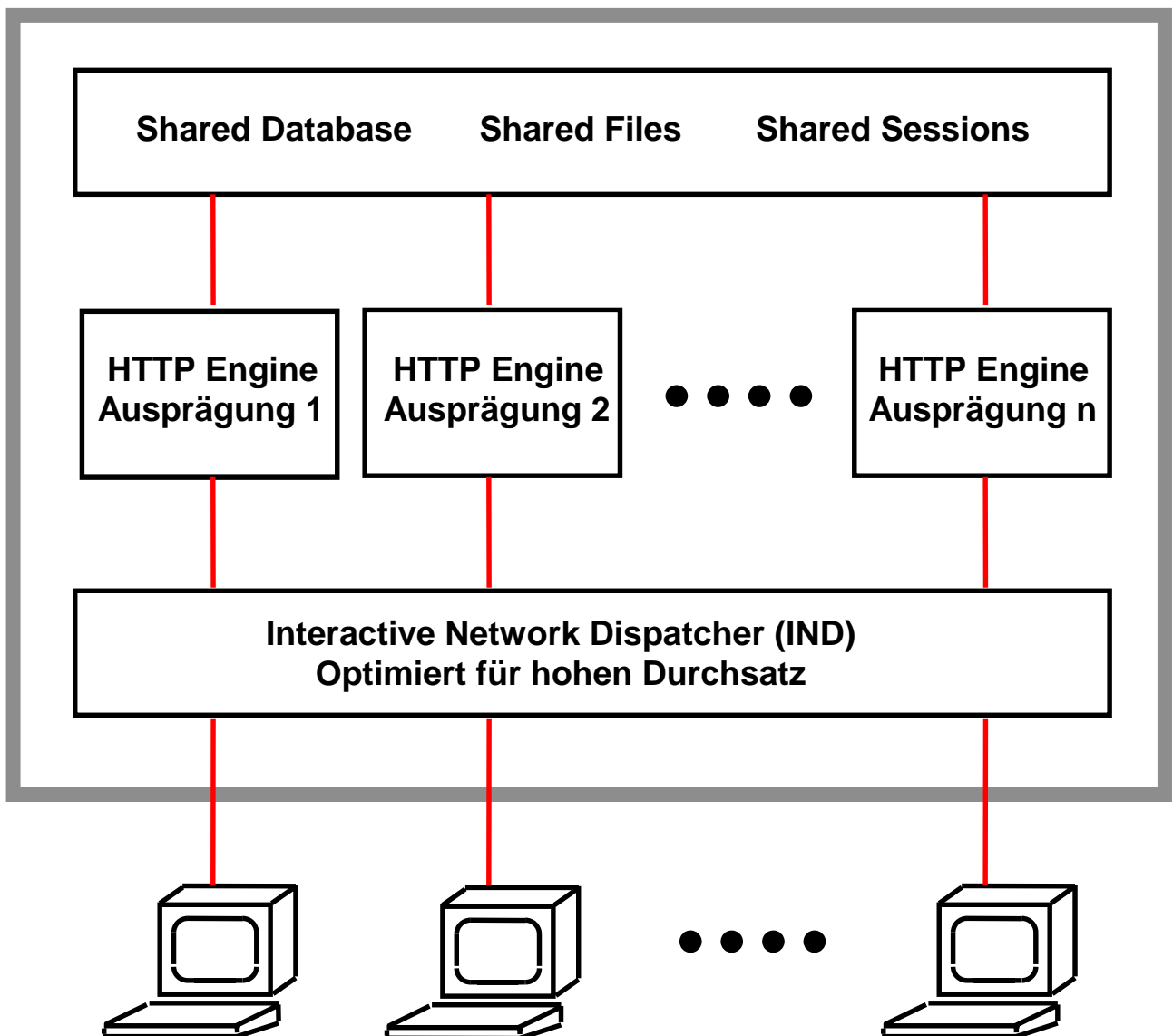


Abb. 8.4.1: Web (HTTP) Server

Der Web Server oder HTTP-Server behandelt Anforderungen für statische Ressourcen, z.B. HTML-Seiten und GIF-Dateien. Charakteristisch ist ein hohes Verkehrsaufkommen und kurzlebige Anforderungen. Die Skalierung erfolgt durch mehrfache Web Server Engines.

Der Interactive Network Dispatcher (auch als „Sprayer“ oder Load Balancer bezeichnet) verteilt die Anforderungen auf die einzelnen Web Engines. Die Aufteilung kann statisch oder dynamisch erfolgen. Der gleichzeitige Zugriff mehrerer HTTP Server auf die gleichen Daten (HTML-Seiten) ist unkritisch, da diese nicht dynamisch modifiziert werden.

Der größte Teil der Zugriffe (z.B. 90 %) ist statisch, und wird vom HTTP-Server alleine abgearbeitet. Nur ein kleiner Teil der Zugriffe benötigt dynamische Daten und bedingt den Aufruf eines Servlets. Allerdings ist der Verarbeitungsaufwand viel größer als bei einem einfachen HTTP-Zugriff. Die Bewältigung dieser Anforderungen erfordert spezielle Maßnahmen.

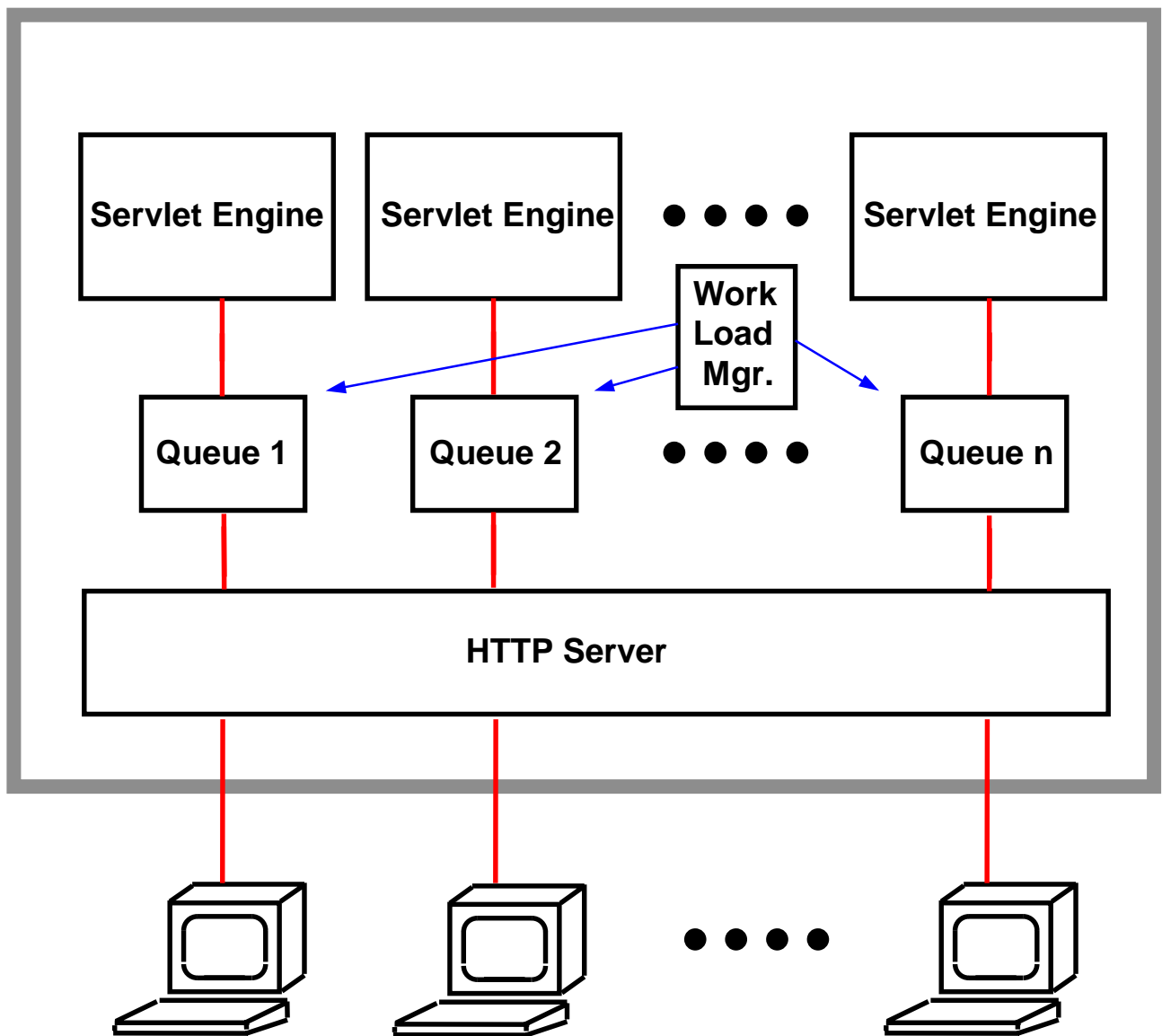


Abb. 8.4.2: Anwendungs-Queues und mehrfache Prozesse

Zur Verbesserung des Leistungsverhaltens laufen mehrere Servlet-Prozesse auf dem Applikations-Server. Anforderungen von dem Web Server gehen (je nach Policy) zu einer von mehreren Queues. Jede Queue wird von mehreren Java-Prozessen bedient.

Dies ist in Abb. 8.4.2 dargestellt. Im einfachen Fall laufen die unterschiedlichen Servlet Engines auf dem gleichen physischem Rechner. Wenn ein Server ausfällt, können die anderen weiter arbeiten. Garbage Collection erfolgt nur durch einen Server in einem gegebenen Zeitpunkt.

In größeren Installationen besteht der physische Server aus einem Symmetric Multiprocessor (SMP) oder einem Cluster von SMPs.

Innerhalb eines SMP oder eines Clusters von SMP wird die Aufteilung der Service Requests von einem "Work Load Manager" (WLM) vorgenommen. WebSphere verfügt zu diesem Zweck über eine eigene WLM-Komponente. Alternativ kann die WLM-Komponente des zugrunde liegenden Betriebssystems in Anspruch genommen werden, z.B. beim OS/390 Sysplex. Die Aufteilung der Service Requests auf die einzelnen Servlet Engines wird durch Policies bestimmt. Die WLM Policy bestimmt:

- URLs, die von der Queue bedient werden

- Anzahl der Prozesse für diese Queue
- Sicherheitsumgebung

Der System Administrator legt die Anzahl und die Policies jeder Queue fest.

Das Zuweisen der Requests muß den normalen Konsistenzanforderungen entsprechen. Der Work Load Manager muß Sitzungs- und Transaktionsaffinität sicherstellen. Normalerweise bedeutet diese Forderung, daß alle Requests innerhalb einer Sitzung zu dem gleichen physischen Rechner weitergeleitet werden müssen.

Um den Zustand einer Sitzung im Fehlerfall zu erhalten, kann es erforderlich sein, ihn persistent auf einer Platte zu speichern. Hiermit kann die Fortführung einer Sitzung auf einem anderen physischen Rechner im Fehlerfall erreicht werden. Dies kann erforderlich sein, um die Transaktionseigenschaften einer Datenbankänderung sicherzustellen.

Eine weitere Leistungsverbesserung wird mit Hilfe des Seitenfragment-Cache erreicht. Es ist üblich, innerhalb von Proxy und Edge Web Servern ganze Seiten im Cache abzulegen. Der Seitenfragment-Cache ermöglicht es, an Stelle von ganzen Seiten nur Teile einer Seite zwischenspeichern.

Das Zwischenspeichern von Seitenfragmenten ist in B2C (On-line shopping)-Anwendungen besonders interessant. Es ermöglicht eine flexible Personalisierung von Seiten. Seitenpersonalisierung führt zu einer großen Anzahl unterschiedlicher HTML-Seiten, die sich nur in wenigen Merkmalen voneinander unterscheiden. Der Seitenfragment-Cache ermöglicht es, Teile einer Seite zwischenspeichern, die nicht personalisiert werden.

Portale sind eine weitere Internet-Anwendung, die im großen Umfang mit personalisierten HTML-Seiten arbeitet. Ein Portal ist eine Web Site mit den folgenden Eigenschaften:

- Daten Aggregation von verschiedenen Quellen
- Content Management
- Intelligente Suche und Data Mining
- Zugriff auf Web-Anwendungen
- Personalisierung
- Benutzer-Administration mit anonymer Benutzer-Unterstützung
- Zugriffskontrolle mit einem Single Sign-on
- Konfigurierbare Benachrichtigungsmechanismen (Agenten)
- Integrierter Desktop mit Unterstützung für Echtzeit-Nachrichten

Im Laufe der Zeit haben sich unterschiedliche Typen von Portalen entwickelt. Persönliche Portale stellen Einzelpersonen allgemeine Information und/oder Produktivitätswerkzeuge zur Verfügung. Community Portale bieten Informationen für eine spezifische Gruppe von Personen an. Firmen Portale sind Community Portale für die Mitarbeiter eines Unternehmens. Inter-Enterprise Portale ermöglichen anderen Firmen einen Zugriff auf Daten und Dienstleistungen desjenigen Unternehmens, das dieses Portal unterhält.

8.5 Distributed Objects

Bei der Entwicklung des RMI-Konzeptes hat sich SUN stark an CORBA orientiert, einem aufwendigeren System für verteilte Anwendungen. Allerdings ist das RMI-Konzept sehr primitiv. So fehlt die in CORBA bekannte Ortstransparenz, d.h. ein Client muß innerhalb des RMI-Konzeptes wissen, auf welchem Rechner sich das Server-Objekt befindet. Der RMI-Namensdienst ist einfach und nicht persistent (RMI Registry). Es existiert kein zentraler Objekt-Broker, der diese Aufgabe in CORBA für alle Objekte übernimmt.

Außerdem können mittels RMI nur Java-Objekte die Methoden anderer Java-Objekte aufrufen. Es fehlt bei RMI ein Mechanismus, d.h. ein Interface unabhängig von der verwendeten Programmiersprache zu beschreiben, es sei denn, man betrachtet Java selbst als Sprache zur Definition einer Art Interface und benutzt die in Java vorhandenen Möglichkeiten zum Aufrufen von Native-Methoden, die zum Beispiel in C programmiert werden.

Durch das Weglassen dieser zentralen CORBA-Features ist RMI ein sehr einfacher Weg, verteilte Anwendungen zu realisieren

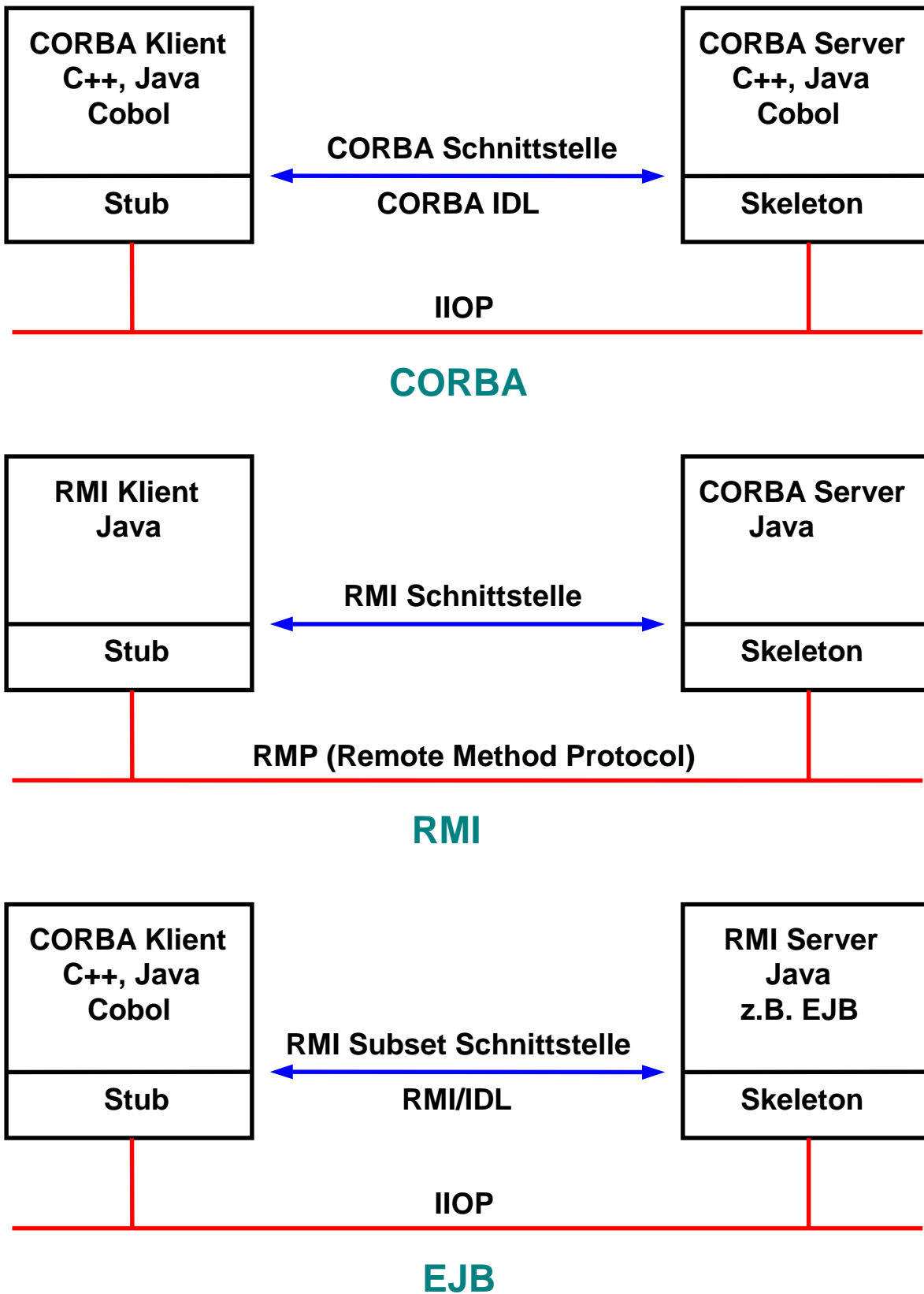


Abb. 8.6.1: Vergleich CORBA - RMI