

J2EE - Java2 Enterprise Edition

Joe Ammann

Version 1.0

4. September 2001

Slides generated with L^AT_EX

1. Übersicht

- J2EE Architektur
- Tiers
- Container Konzept

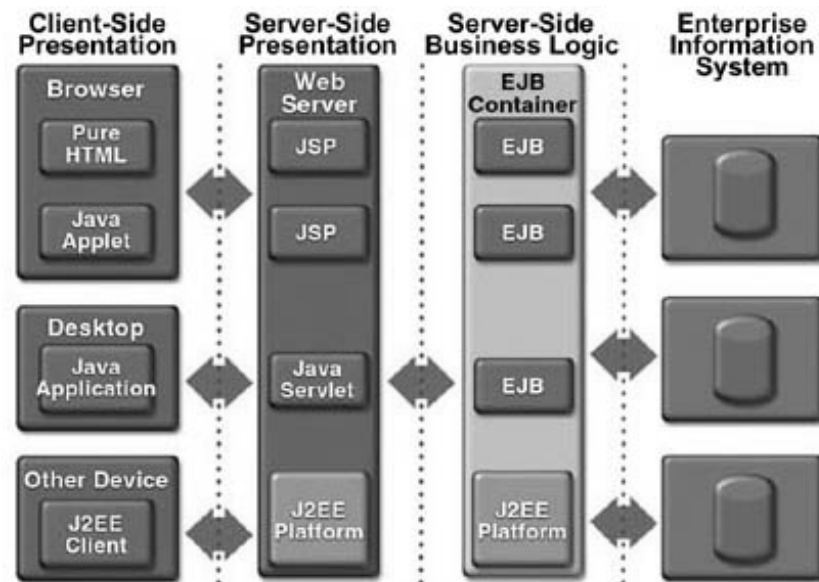
J2EE Architektur – 1.1

- Plattform für Applikationen
- zusammengesetzt aus spezialisierten Komponenten
- Blueprints (best practices) und Beispiel-Applikationen



Tiers – 1.2

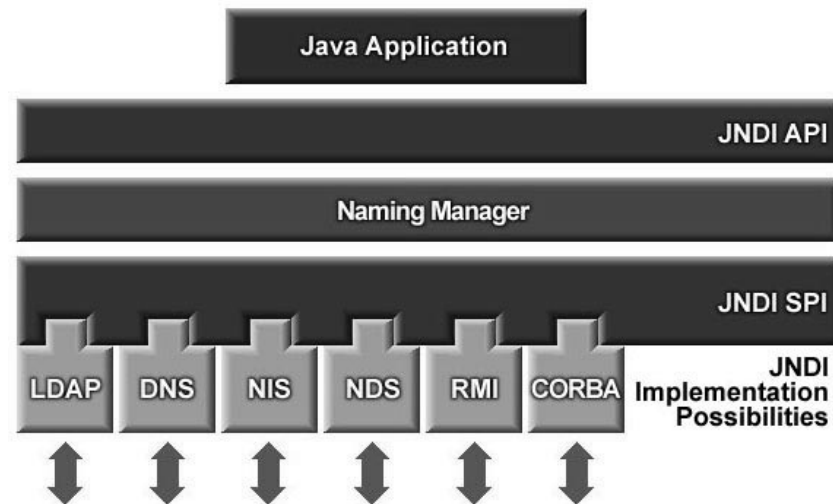
- Client Tier (Browser, Applets und (Client-)Applikationen)
- Web Tier (Servlets und JSP)
- Business Logic Tier (EJB)
- EIS (DB, SAP, Mainframe)



Container Konzept – 1.3

So weit, so gut - was ist “anders” ?

- J2EE definiert standardisierte Container
- Run-Time Umgebungen, die Services zur Verfügung stellen
- API definiert Schnittstelle zur Applikation (Komponente)
- Service Provider Interface (SPI) definiert Schnittstelle zur Implementierung



Kompatibilität

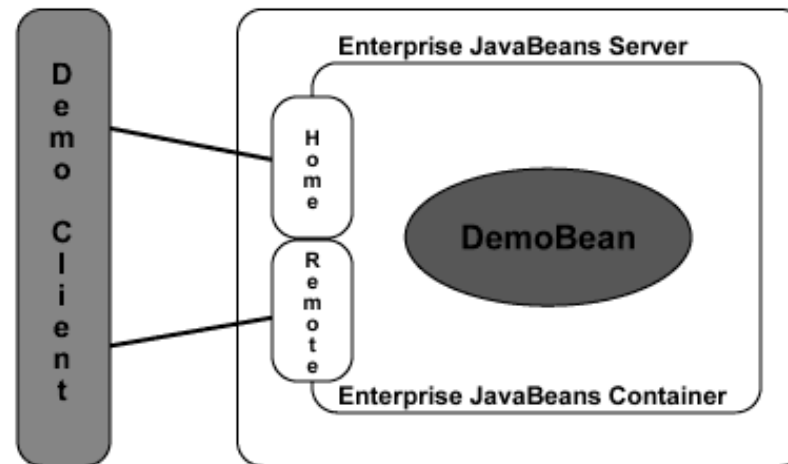
- Theorie: alle Komponenten in jedem J2EE Container lauffähig
- Praxis: Unterschiede vorhanden, aber Portierung mit kalkulierbarem Aufwand möglich
- “make your choice”

2. Applikationen

- Enterprise Java Beans - EJB
- Servlets
- Java Server Pages - JSP

EJB – 2.1

- Komponenten für Business Logik
- Entity Beans für Datenzugriff (Kunde, Artikel)
- Session Beans für Prozess-Logik (Kunde erfassen, Artikel nachbestellen)
- Message Driven Beans für “Event Behandlung”
- kein GUI, keine Gesamt-Abläufe, keine aktiven Elemente



Servlets – 2.2

- HTTP Requests behandeln
- doGet(), doPost()
- funktional ähnlich wie CGI-Scripts
- automatisches HTTP-Session-Handling (Session Context)
- Mischung zwischen Logik und Präsentation

```
public void doGet(HttpServletRequest req,
                  HttpServletResponse resp) {
    req.getParameter( "name" );
    ...
    resp.getWriter().println( "<HEAD><TITLE>..." );
}
```

JSP – 2.3

- HTML mit embedded Java
- reine Präsentation - keine Logik

```
<P>Today is </P>
```

```
<jsp:useBean id="clock" class="jspCalendar" />
```

```
<UL> <LI>Day: <%= clock.getDayOfMonth() %>
```

```
<LI>Year: <%= clock.getYear() %> </UL>
```

```
<%! int time = Calendar.getInstance().get(Calendar.AM_PM); %>
```

```
<% if (time == Calendar.AM) { %>
```

```
Good Morning
```

```
<% } else { %>
```

```
Good Afternoon
```

```
<% } %>
```

Parametrisierung – 2.4

- EJB's und/oder Servlets und JSP zusammengefasst zu Deployment Units
- Parametrisierung mittels XML-Deskriptoren
- Standard-XML für EJB, Web-Applications
- J2EE-Container-spezifisches XML (z.B. WebLogic, WebSphere) für weitere Parameter (Caching, Performance)

```
<env-entry>
```

```
  <env-entry-name>minAmount</env-entry-name>
```

```
  <env-entry-value>10</env-entry-value>
```

```
</env-entry>
```

```
<resource-ref>
```

```
  <res-ref-name>jms/any-name-for-the-app</res-ref-name>
```

```
  <res-type>javax.jms.QueueConnectionFactory</res-type>
```

```
</resource-ref>
```

3. Services

- Kommunikation
- DB, Connector
- Transaktionen, Naming, Security

Container Services – 3.1

- alle technischen Aspekte
- J2EE Modell forciert die Trennung Applikation/Services
- JNDI (Java Naming and Directory Interface) für jegliche Art von Zugriff auf Services und Komponenten
- konfigurierbar, “Zusammensetzen”

```
// JNDI context von Produkt 1
javax.naming.Context ctx = new InitialContext();

QueueConnectionFactory qcf =
    JMSProd2Class.getQueueConnectionFactory();

ctx.bind("NameToBeUsed", qcf);
```

Kommunikation – 3.2

- Java Message Service (JMS) für asynchrone Kommunikation
- Einbindung in Transaktionen möglich

- CORBA Anbindung (RMI over IIOP)
- J2EE als Java-Implementierung des CORBA Component Model (CORBA 3.x)

- Java Mail

Datenbank und J2EE Connector – 3.3

- JDBC für RDBMS Zugriff
- Applikation definiert “Resource Reference”
- konkrete Abbildung (z.B. Oracle DB XXX, Tabelle FOOBAR) durch Runtime-Umgebung (XML Descriptor)

- J2EE Connector (JCA) für Zugriff auf EIS (Mainframe, SAP, TP System)
- generisches System für jegliche Art von synchroner Request-Reply Interaktion
- Einbindung in Transaktionen und Security möglich

Transaktionen, Security – 3.4

- Java Transaction API für XA-Transaktionen
- Transaktionen explizit im Java Code (bean managed transactions)
- Transaktions-Grenzen deklarativ im XML Descriptor definieren (container managed transactions)

- (einfaches) Security-Modell (rollenbasiert) für Zugriffsschutz
- Authentisierung ist Sache des Containers (meist mittels Username/Password oder HTTPS/X.509 Zertifikaten)
- Principal und Rollen werden durch alle Aufrufe mitgeführt (isCallerInRole())

Mix and Match – 3.5

- durch standardisierte Schnittstellen Mix-and-Match immer besser möglich
- z.B. JMS mittels MQ Series von IBM, JNDI mittels OpenLDAP, JDBC mit PostgreSQL
- Hersteller von J2EE Applikations-Servern “erlauben” immer mehr solche Integrationen