Mainframe Internet Integration

Prof. Dr. Martin Bogdan
Prof. Dr.-Ing. Wilhelm G. Spruth

SS2013

Java Enterprise Edition Teil 3

Enterprise Java Beans

 © copyright W. G. Spruth,

Java Editionen

Komplexität, Umfang

JEE Enterprise Edition

Business Logik, Mainframe

JSE Standard Edition

PC, Workstation

JME Micro Edition

Mobiltelefone, Embedded Devices, Kleingeräte

Java existiert in drei unterschiedlichen Editionen mit einem unterschiedlichen Funktionsumfang:

JME – Java Micro Edition
JSE – Java Standard Edition
JEE – Java Enterprise Edition

Frühere Bezeichnungen: J2ME, J2SE, J2EE

Java Enterprise Edition

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) von Sun 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 Infrastruktur

Die JEE Infrastruktur beinhaltet Applikationsserver 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.

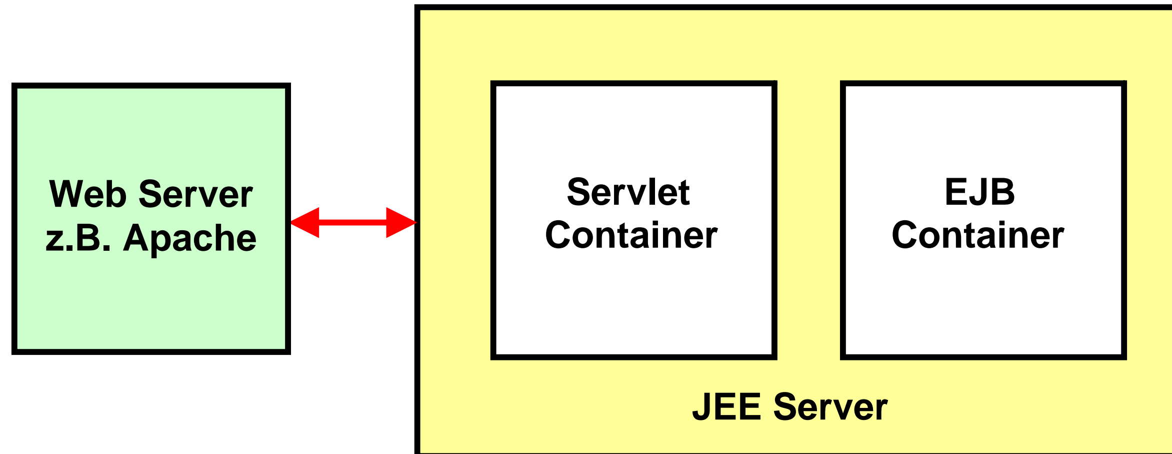
Der JEE Server, beziehungsweise der EJB Container, interagiert mit Systemressourcen des Unternehmens (z.B. Datenbanken) und übernimmt auch die Interaktion mit geografisch verteilten Beans in anderen Servern. Weiterhin kontrolliert er die Ausführung von selbst definierten Transaktionen und handhabt Sicherheitseinstellungen.

JEE Server bieten also typische Middleware–Techniken an, und ermöglichen eine vereinfachte Komponenten-programmierung. 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.

IBM bezeichnet seinen JEE Server als WebSphere Web Application Server (WAS).

JEE Server

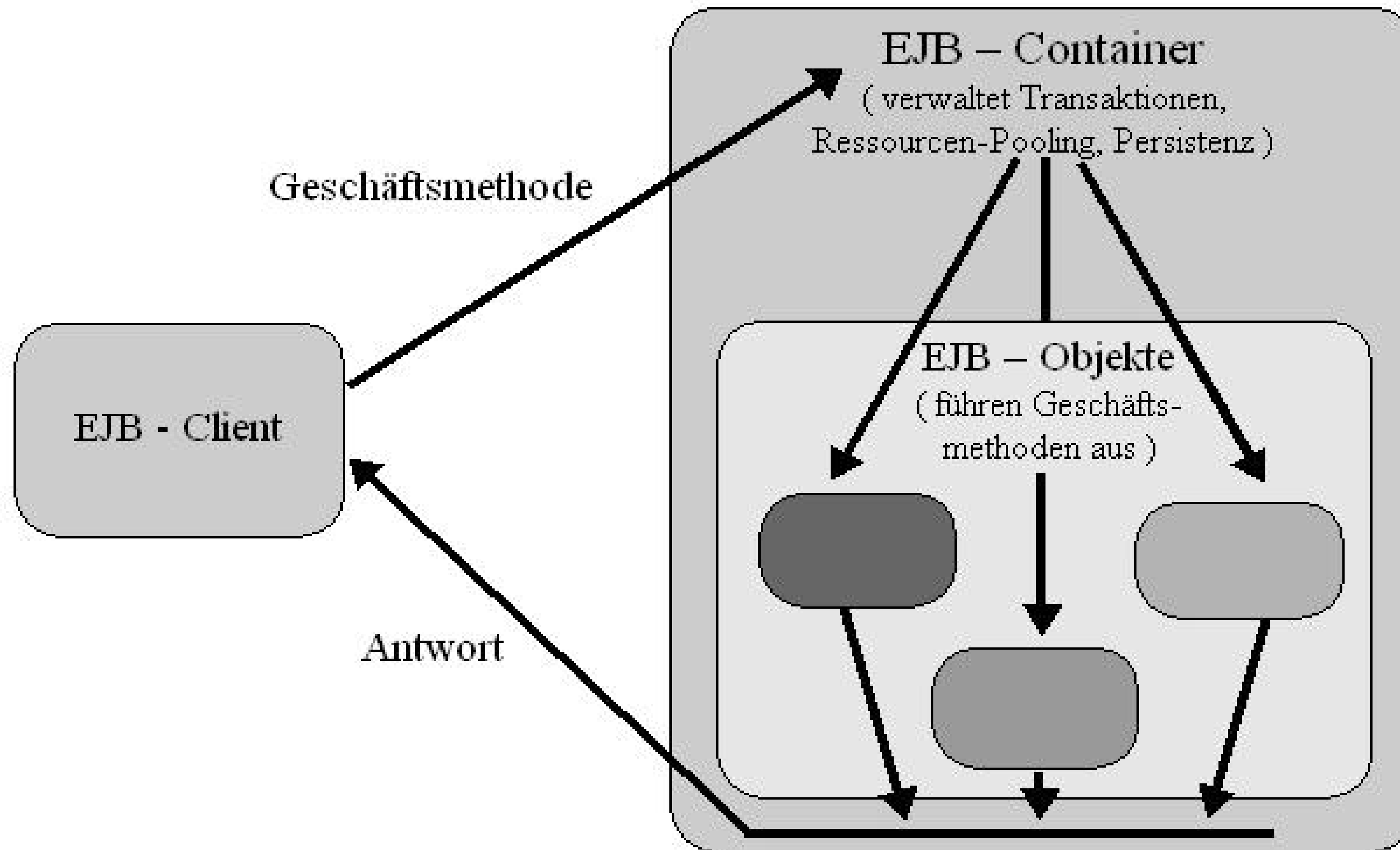


Servlets benötigen für ihre Ausführung eine Servlet Laufzeitumgebung, den Servlet Container. EJBs benötigen für ihre Ausführung eine EJB Laufzeitumgebung, den EJB Container.

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

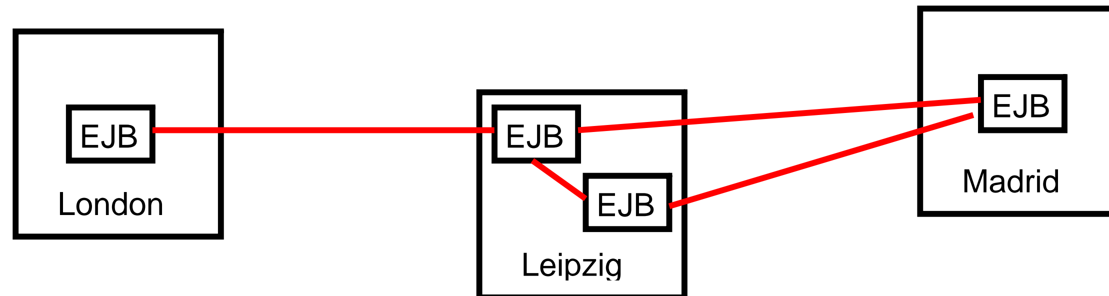
Der Web Server (z.B. Apache oder z/OS http Server) und der JEE Server laufen typischerweise als getrennte Prozesse auf dem gleichen physischen Server. Weil der Begriff „Web Server" mehrfach belegt ist, sollte statt dessen der Begriff http Server benutzt werden.

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



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.



Verteilte Objekte

Verteilte Objekte 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.

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.

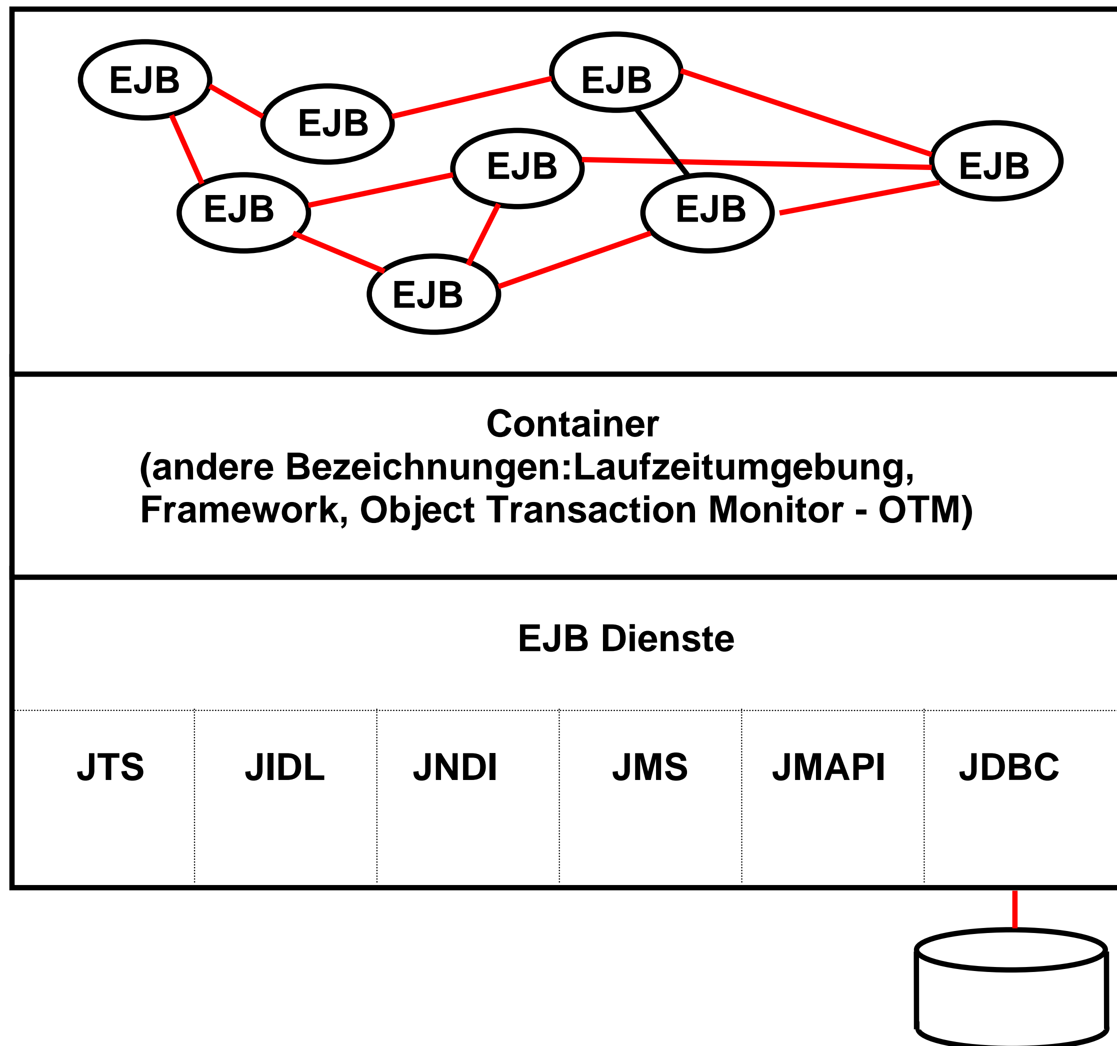
In der Praxis werden Objekte als SQL (oder IMS, ADABAS oder VSAM) Daten gekapselt; der Zugriff erfolgt z.B. über eine JDBC (Java Data Base Connectivity) oder SQLJ Schnittstelle.

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.

Enterprise Java Beans (EJB)



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 and Directory Interface
- JMS Java Messaging Servics
- JDBC Java Data Base Connectivity
- JMAPI Java Management API
- JIDL Java Interface Definition Language

Enterprise Java Beans (EJB)

Enterprise Java Beans sind Java Beans mit erweiterter Funktionalität. Dies sind unter anderem

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

JTS, der Java Transaction Service, ist eine API zum Aufrufen der Funktionen eines Transaction Servers.

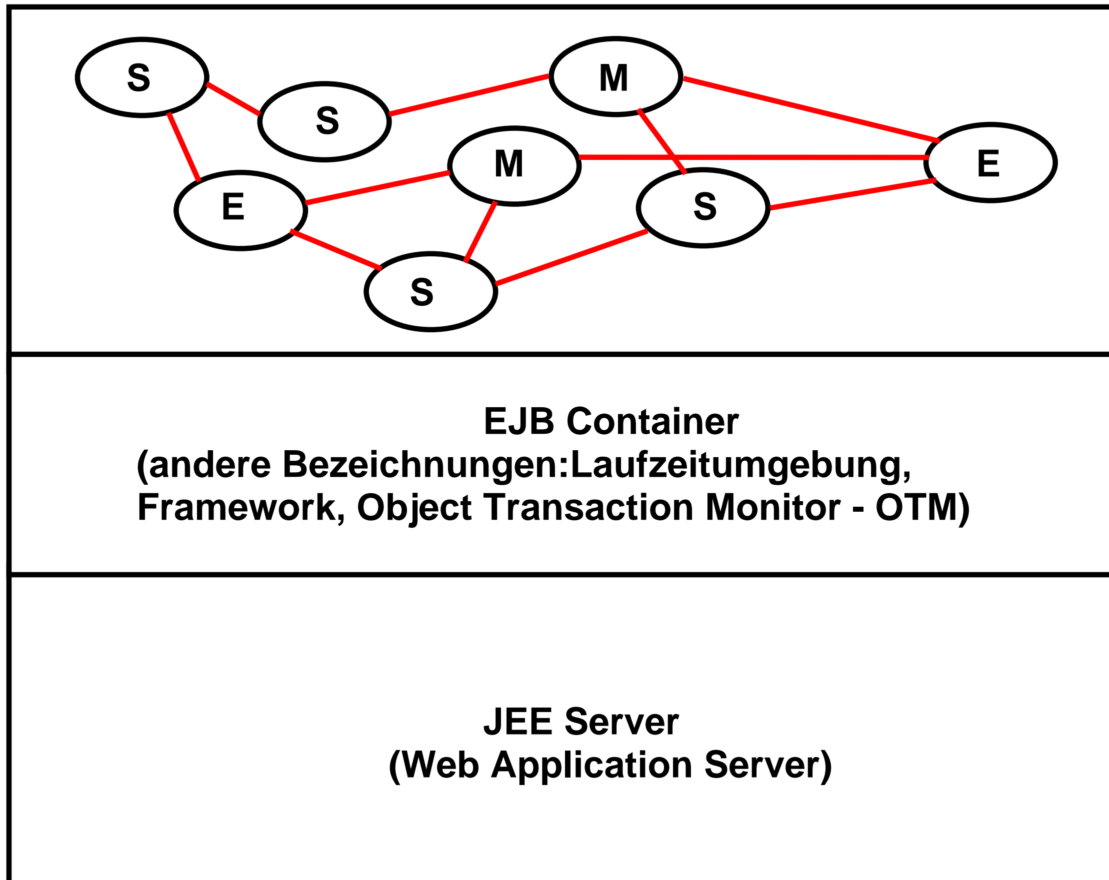
JNDI, die Java Naming and Directory Interface, ist eine API für den Zugriff auf Namens-und Verzeichnisdienste. Zum Beispiel wird sie verwendet, um Datenbanken oder EJBs zu lokalisieren, die von einem Servlet benötigt werden.

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.



S Session Bean (transientes Objekt)
M Message-driven Bean
E Entity Bean (persistentes Objekt)

Drei verschiedene Arten von EJBs

Es existieren drei unterschiedliche Arten von EJBS :

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

- **EJBs sind Komponenten, die eine objektorientierte Sicht auf persistent gespeicherte Entitäten repräsentieren, also eindeutig identifizierbare und über Attribute beschreibare 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.**

- **EJBs sind Komponenten, die auf dem Server ablaufende Geschäftslogik implementieren. Sie modellieren Geschäftsprozesse, wie z.B. eine Überweisung eines Geldbetrages.**

Session Bean

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

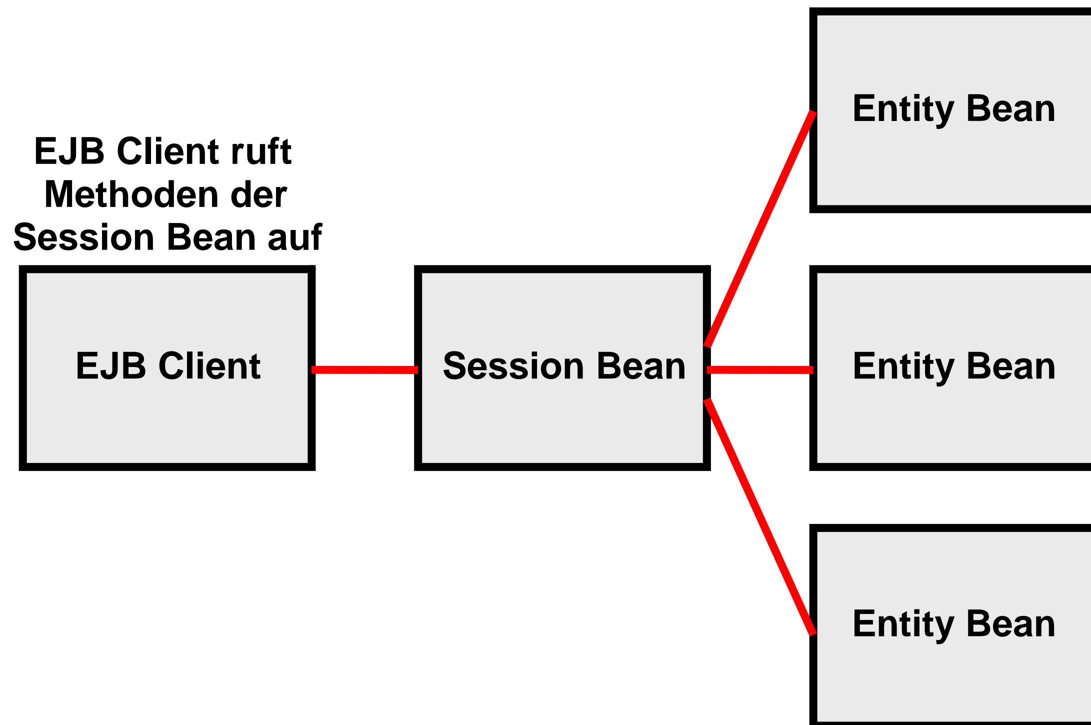
Entity Beans repräsentieren für den Client eine objektorientierte Sicht auf einen Datensatz. Sie erlauben im Gegensatz zu Session Beans auch Mehrbenutzerbetrieb: Auf eine Instanz einer Entity Beans können gleichzeitig mehrere Benutzer zugreifen. Der Container, in den Entity Beans während der gesamten 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. In vielen Fällen entsprechen diese Daten einer Zeile in einer relationalen Datenbank Tabelle.

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.



Session Fassade EJB Architektur

Aufgabenteilung:

- **Session Beans enthalten die Business Logik**
- **Entity Beans speichern Daten**

In dem ursprünglich vorgesehenen Ansatz 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.

Haben Entity Beans eine Zukunft ?

Bei der ursprünglichen Konzeption der J2E Architektur ging man davon aus, dass Entity Beans in objektorienttierten 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 ausschlieslich Session Beans eingesetzt und die Persistenz mit eigenem Code implementiert. Um dies zu erleichten, stellten mehrere 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.

Eine weitere Alternative sind „Konnektoren“. Dies sind Java Klassen, die einen Zugriff auf DB2, VSAM, IMS oder Adabas ermöglichen. Wir gehen hierauf später ein.

Persistenz

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

In der Praxis werden SQL (oder IMS, ADABAS oder VSAM) Daten als Objekte gekapselt; der Zugriff erfolgt z.B. über eine JDBC (Java Data Base Connectivity) SQLJ oder DB2Connect 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 es 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, so dass verhindert wird, dass die Daten versehentlich mit einem falschen Objekttyp abgelegt werden.

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.

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.

JNDI (1)

Eine Methode einer Java Klasse wird aufgerufen, indem man ihren Namen spezifiziert. Die Klasse und ihre Methode sind irgendwo mit irgendeiner Adresse gespeichert.

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 benannten Objekte ermittelt werden.

Wenn wir eine URL im Internet aufrufen, verwenden wir den Internet Domain Name Service (DNS) um den Server und das Verzeichnis zu finden, auf dem sich die gesuchte URL befindet. 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.

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.

JNDI (2)

Die API enthält:

- **einen Mechanismus zur Bindung 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.