Vorlesung
Software aus Komponenten

3. Komponentenmodelle

Prof. Dr. Hans-Gert Gräbe
Wintersemester 2012/13

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Zustandslose Beans

- Implementieren plain old Java Interfaces
- keine Home-Schnittstelle, Laufzeitkontrolle durch Kontext
- Annotation **@Stateless**
- Definierte Callbacks **@PostConstruct** und **@PreDestroy**
- Interzeptor **@AroundInvoke** um eine einzelne Session
- Unspezifizierte Transaktions- und Sicherheitskontexte
- Dependency Injection muss vor dem ersten Aufruf einer Geschäftsmethode erfolgen

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Bean und Interzeptor. Beispiel

```
@Stateless @Interceptors ({MyInterceptor.class})
public class BuchungBean implements Buchung {
   public void buchen(Kunde k, Seminartyp s) { ... }
}


public class MyInterceptor {
   @AroundInvoke
   public Object zeitVerbrauch(InvocationContext ic) throws Exception {
            long time = System.currentTimeMillis();
            try { return ic.proceed(); }
            finally {
                  long total = System.currentTimeMillis() - time;
                  System.out.println("Aufruf von " + ic.getMethod().getName()
                  + " dauerte "+total+" Millisekunden.");
            }
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Zustandsbehaftete Beans

- Implementieren plain old Java Interfaces
- keine Home-Schnittstelle, Laufzeitkontrolle durch Kontext
- Annotation **@Stateful**
- kann Schnittstelle **SessionSynchronization** implementieren
- Definierte Callbacks **@PostConstruct**, **@PreDestroy**, **@PostActivate**, **@PrePassivate**
- Interzeptoren
  - **@afterBegin** – Ausführung zu Beginn jeder Session
  - **@beforeCompletion** – Ausführung von Ende jeder Session
  - **@AroundInvoke** – Einbettung einer Session
- Dependency Injection muss vor dem ersten Aufruf einer Geschäftsmethode erfolgen
- Client bekommt Referenz auf Session direkt über JNDI oder über Dependency Injection
- Annotation **@Remove** für Methode zum Auflösen der Beaninstanz

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Zustandbehaftete Bean. Beispiel

```
@Stateful public class BuchungBean implements Buchung {
    Connection rc;
    @Resource SessionContext sc;

    @PostConstruct @PostActivate public void init() {
            rc = Connection.Open();
    }

    @PreDestroy @PrePassivate public void close() {
            try { rc.close(); }
            catch (CloseException) { /* no op */ }
    }
// ... weiter
```

```
    @Remove public void dispose() { /* was auch immer */ }

    @AroundInvoke public Object monitor(InvocationContext ic) {
            try {
                Object res = ic.proceed();
                if ( (OpResult) res == OpResult.SUCCEED ) {
                    System.out.println("Aufruf war okay");
                }
                return res;
            } catch (Exception ex) { throw ex; }
    }

    // Geschäftsmethode
    public void buchen(Kunde k, Seminartyp s) { ... }
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Zustandbehaftete Bean. Beispiel

Zugriff durch den Client:

```
static public void testBean(Kunde k, Seminartyp s) throws Exception {
    javax.naming.Context ctx = new javax.naming.InitialContext();

    Buchung test = ctx.lookup(Buchung.class.getName());

    // oder Injection, wenn innerhalb einer anderen Bean:
    /* @EJB Buchung test; */

    test.buchen(k,s);          // Test der Geschäftsmethode
    test.dispose();            // Test der Bean dispose-Methode
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Nachrichtengesteuerte Beans

- Implementieren Nachrichten-Listener-Schnittstelle des jeweiligen Nachrichtentyps (bspw. **javax.jms.MessageListener**)
- Annotation **@MessageDriven**
- implementiert nicht notwendig **javax.ejb.MessageDrivenBean**
- kann Schnittstelle **SessionSynchronization** implementieren
- Definierte Lebenszyklus-Event Callbacks **@PostConstruct**, **@PreDestroy**
- Interzeptoren für Listener-Aufrufe
- Dependency Injection muss vor dem ersten Aufruf einer Geschäftsmethode erfolgen

```
@MessageDriven(activateConfig = { ... })
public class ExampleMDB implements javax.jms.MessageListener {
    public void onMessage(Message msg) { ... }
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Bean-Kontext und Umgebung

- Bean spezifiziert ihre Abhängigkeiten durch **Abhängigkeits-annotationen** für Typ, Name und Charakteristika eines benötigten Objekts oder Ressource

```
@Resource (name = "DB", type = "javax.sql.DataSource.class")
```

- Bean-Entwickler annotiert erforderliche Instanzvariablen, die vom Container automatisch nach Setzen von **EJBContext** und vor dem Aufruf der ersten Geschäftsmethode aus dem Kontext instanziiert werden

```
@Stateless public class MySessionBean implements MySession {
   @Resource(name="DB") public DataSource myDB;
   public void getData() {
   // ohne try-catch! Probleme sind nur auf Kontextseite möglich
            Connection c = myDB.getConnection();
   }
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Persistenz

- Entity-Klassen sind durch **@Entity** annotiert und müssen einen **public-Konstruktor ohne Argumente** haben.
  - können abstract oder konkret sowie in Vererbungshierarchien eingebunden sein
  - müssen (in der Regel) Schnittstelle **Serializable** implementieren
- Instanzen einer Entity-Klasse sind leichtgewichtige persistente Objekte: Persistenz auf der Ebene einzelner Attribute
  - Kennzeichnung durch Annotation
  - Persistenz-Anbieter greift auf entsprechende Attribute zu; diese müssen nicht nach außen für andere sichtbar sein
  - Zugriff direkt auf das Attribut oder über Getter-Methode (Default)
- Default: Abbildung auf Datenbank-Tabelle mit demselben Namen
  - Primärschlüssel müssen aus einer Klasse sein, die **equals** und **hashCode** korrekt unterstützt.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Persistenz. Beispiel

```
@Entity @Table(name ="KundenListe")
public class KundenListe implements java.io.Serializable {
    private int id; // Primärschlüssel für die Kundenliste
    private Collection<Kunde> kundenListe;

    @Id(generate = GenerationType.AUTO)
    public int getId() { return id; }
            // spezifiziert, dass dies die Getter-Methode für den Primärschlüssel ist
    public void setId(int id) { this.id = id; }

    @OneToMany public Collection<Kunde> getKunden() { return kundenListe; }
            // Auslesen der Kundenliste
...
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Persistenz (Fortsetzung)

- Persistente Aggregate (Ganzes aus mehreren Teilen) werden über die Annotation **@Embedded** innerhalb einer **@Entity** realisiert
- Eine Entity-Bean kann auf mehrere Datenbank-Tabellen abgebildet werden: **@SecondaryTable**
- Relationen zwischen Entitäten über Annotationen **@OneToOne**, **@OneToMany**, **@ManyToOne**, **@ManyToMany**
  - können unidirektional oder bidirektional sein
- Persistenz kann in Vererbungshierarchien mit vererbt werden: **@Inheritance**

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

EntityManager

- **EntityManager** ist für die Verwaltung des Lebenszyklus von Entity-Beans verantwortlich
- Assoziiert mit dem Persistenzkontext, also einer Menge von Entity-Beans

```
@Stateless public class AuftragsListe {
    @PersistenceContext EntityManager em;
    public void neuerAuftrag(int kundenId, Auftrag neuerAuftrag) {
            Kunde k = (Kunde) em.find("Kunde", kundenId);
            k.getAuftraege().add(neuerAuftrag);
            neuerAuftrag.setKunde(k);
    }
}
```

- EntityManager können vom J2EE-Server zur Verfügung gestellt werden, aber auch von der Applikation → **EntityManagerFactory**

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Weitere Konzepte

- Transaktionskontrolle: entweder via JTA oder über EntityTransaction
- Persistenz-Kontexte: Innerhalb eines solchen Kontexts wird die Eindeutigkeit der Zuordnung Entity – Persistenzobjekt garantiert
- Query API mit eigener, an SQL angelehnter EJB Query Language
- Entity Packaging: Im Deployment-Deskriptor können genauere Informationen über EntityManager gespeichert werden.

Zusammenfassung

- Aufgaben des EJB-Container werden an den Kontext delegiert
- Spezielle Eigenschaften des Ausführungskontexts werden durch Annotationen direkt im Quelltext festgelegt statt als Deployment-Informationen
- Persistenz folgt dem Konzept der Java Data Objects JDO
- Standard fixiert in JSR 220

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Server-Provider (Server-Anbieter)

- stellt Plattform zur Verfügung
- Netzwerkanbindung, Skalierungsfunktion
- Prozess- und Ressourcenmanagement

EJB-Kontext-Provider (Container-Anbieter)

- setzt auf Plattform des Server-Providers auf
- Benutzerverwaltung, Transaktionsmanagement, Persistenz
- Herstellung der Installationswerkzeuge
- Implementierung der EJB-Kontextfunktionalität
- Container- und Server-Provider oft identisch
    - bessere Performance und Wartbarkeit

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Bean-Provider (Komponenten-Entwickler)

- realisiert geforderte Anwendungslogik (Geschäftslogik)
- keine grundlegenden Funktionalitäten (Persistenz, Netzwerk, etc.)
- benötigt Fachwissen über Anwendungsbereich
- Konzentration auf inhaltliche Problemstellung

Application-Assembler (Monteur)

- verbindet Komponenten zu einer Anwendung
- Clients
- Verwendung von Basiskomponenten
- Nutzung wiederverwendbarer Komponenten von Drittanbietern
- oftmals Bean-Provider und Application-Assembler in einem Haus

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Bean-Deployer (Installation)

- installiert Komponenten der Anwendung auf Zielsystem
- verwendet Werkzeuge des Container-Providers
- nutzt Deployment-Deskriptor
- konfiguriert EJBs
- generiert Stummel- und Skelettklassen
- legt Benutzerdatenbank an
- benötigt detailliertes Wissen über Server und Container

Systemadministrator

- verantwortlich für reibungslosen Ablauf
- Benutzerverwaltung

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Java-Basisdienste für Komponenten

Grundlegende Dienste

Java core reflection erlaubt zur Laufzeit

- Inspektion von Klassen und Schnittstellen nach Attributen und Methoden
- Konstruktion neuer Klasseninstanzen und Felder
- Zugriff und Modifikation von Attributen in Verbundobjekten oder Feldern
- Aufruf von Methoden von Objekten und Klassen

- **java.lang.reflect** als eigene Klasse hierfür
  - einige Funktionalität historisch in der (finalen) Klasse **java.lang**.

Java GuI-Klassensammlungen AWT und Swing

- delegierende Ereignisbehandlung
- Datentransfer und Zwischenablage wird unterstützt, drag and drop
- Java 2D rendering, eng damit zusammen Java printing model
- Internationalisierung
- pluggable look and feel, Palette von Standard-Komponenten

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Objektserialisierung

- standardisiertes Kodierungsschema für Serialisierung
- Klasse muss dafür Interface **java.io.Serializable** implementieren
- Objektserialisierung ist sicherheitskritisch
- Mechanismen zu Serialisierung und Deserialisierung ganzer Objekt-Webs
  - nicht zu serialisierende Attribute können als transient markiert werden
    - Bsp: große Cache-Strukturen
  - private Methoden **readObject** und **writeObject** werden statt Default genommen, wenn durch Reflektion gefunden
  - Mehrfachreferenzen auf ein Objekt werden rekonstruiert
- unterstützt einfaches Versionierungsschema:
  - 64-bit-hash-Code (Serial version UID = SUID) wird über die Signatur der Klasse berechnet und kann während **readObject** ausgewertet werden.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Ferne Objekte und RMI

- Auf ferne Objekte kann nie direkt zugegriffen werden, sondern nur über Interface **java.rmi.Remote**. Ressourcenbindung über Namensdienst.
- Remotezugriff kann immer fehlschlagen:
  Exception **java.rmi.RemoteException**
- Parameterübergabe:
  - Referenz, wenn Parameterwert selbst vom Remote-Typ ist
  - Durch Marshalling übergebene Kopie, wenn Parameterwert lokal im aufrufenden Kontext
  - Übergabe nicht serialisierbarer Objekte führt zu Laufzeitausnahme
- Unterstützt verteiltes Garbage Collection
  - durch genaue Buchführung über Remote-Referenzen
  - basiert auf Arbeit [Birrel 1993] über Network Objects
  - eines der bedeutendsten Features von Java RMI
- Konflikt mit Begriff der Objektidentität im Java-Standard
  - Referenzen auf ein fernes Objekt sind Java Referenzen auf das lokale Proxy des fernen Objekts
  - Backcall erzeugt ein lokales Proxy im ObjectHome, neben der eigentlichen Java-Referenz

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Weitere Java-Dienste für Komponenten

JNDI: Java Naming and Directory Service

Aufgabe: Dienste über Namen bzw. Attribute finden

- Interface **Context** macht Namenskontext verfügbar
- Methode **lookup** findet Objekte über ihren Namen
- Interface **DirContext** erweitert **Context** zur Suche über Attributwerte
- Unterstützung von Kontexthierarchien, die rekursiv durchsucht werden.

JMS: Java Messaging Service

Aufgabe: Unterstützung asynchroner datengetriebener Kompositionsmodelle

- Standardisiert Java-Zugriff auf vorhandenes Nachrichtensystem, implementiert keins selbst.

JDBC: Java database connectivity

Aufgabe: Einheitlicher Zugriff auf Datenbanken über entsprechende Treiber

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

JTA: Java Transaction API

JTS: Java Transaction Service

Aufgabe: Unterstützung von Transaktionskonzepten

JCA: J2EE Connector Architecture (seit J2EE 1.3)

- Einheitliches Konzept des Ressourcen-Adapters, über welches externe Ressourcen aus einer EIS (enterprise information structure) in einen J2EE-Applikationsserver eingebunden werden können
- Definition eines entsprechenden **JCA common client interface** (CCI)
- Einsatz innerhalb von Enterprise Application Integration Frameworks

Java und XML: Java unterstützt mit entsprechenden Klassen

- XML-Dokumente (DOM)
- XML-Streaming (SAX)
- XML-Binding (JAXP)
- XML-Messaging (JAXM)
- XML-Processing (JAXP)
- XML-Registries (JAXR)

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Java und CORBA:

- Java wichtigste CORBA-Referenzimplementierung
- Koexistenz in fast allen Applikationsserver-Produkten
- Zugriff auf CORBAservices über Java-spezifische Zugriffs-Schnittstellen sowie weitere Konzepte (POA, Namensdienst) seit Java 1.4
- RMI-over-IIOP als eingeschränkte RMI-Version seit 1999
  - RMI nutzt spezifisches proprietäres Protokoll
  - keine Unterstützung des verteilten Garbage Collection, so dass explizites Lebenszyklus-Management erforderlich ist
    - dafür existieren aber CORBAservices