

Enterprise JavaBeans 3.0

einfach mächtig – mächtig einfach

Oliver Ihns



Präsentation Java Forum Stuttgart 2005

Track B7, 07.07.2005

Oliver Ihns, Resco GmbH

Über mich...

- Mitglied der EJB 3.0 Expert Group im JCP bei Sun Microsystems
- Angestellt bei der IT-Unternehmensberatung Resco GmbH
- Leiter der Competence Area „Enterprise Application Architecture“
- Herausgeber und Co-Autor des Fachbuches "Enterprise JavaBeans komplett – EJB 2.1"
- Autor diverser Fachartikel
- Themenschwerpunkte
 - ▶ Software-Architekturen und Technologien für *** verteilte Softwaresysteme
 - *** **objekt-orientierte, komponenten-orientierte, service-orientierte**
 - **auf Basis J2EE / EJB, CORBA, MOM, ...**



Oliver Ihns
oliver.ihns@resco.de

Enterprise JavaBeans 3.0

JSR 220 im JCP



J2EE™ 5.0 SDK



Status:



Men at work!

Agenda

EJB 3.0

- Was bis jetzt geschah
- Hauptziele von EJB 3.0
- "Zurück" zu POJO/POJI
- Verwendung von Meta-Annotationen
- Kapselung von Umgebungs- und JNDI-Zugriffen
- Weitere Neuerungen / Erleichterungen
- Session Bean
- Message-driven Bean
- Entity Bean / Persistenz
- Resümee

Agenda

EJB 3.0

- **Was bis jetzt geschah**
- Hauptziele von EJB 3.0
- "Zurück" zu POJO/POJI
- Verwendung von Meta-Annotationen
- Kapselung von Umgebungs- und JNDI-Zugriffen
- Weitere Neuerungen / Erleichterungen
- Session Bean
- Message-driven Bean
- Entity Bean / Persistenz
- Resümee

Was bis jetzt geschah

- 09/2003 ■ EJB 3.0 Expert Group im September 2003 formiert
- 08/2004 ■ Erster öffentlicher Draft (Early Draft) im JCP publiziert
- 09/2004 ■ **JDO (JSR243) geht in EJB 3.0 (JSR220) auf!**
 - ▶ Die führenden JDO-Experten erweitern das EJB 3.0 Team
 - ▶ EJB-Persistenz und JDO werden zu einer einzigen Persistenzspezifikation für J2EE / J2SE konsolidiert
 - ▶ Im Rahmen von EJB 3.0 separate Persistenzspezifikation geplant
- 03/2005 ■ Zweiter öffentlicher Draft (Early Draft) im JCP publiziert
- Q1/2006 ■ Finale Veröffentlichung Q1 / 2006

Zu verfolgen im



<http://www.jcp.org/en/jsr/detail?id=220>

Agenda

EJB 3.0

- Was bis jetzt geschah
- **Hauptziele von EJB 3.0**
- "Zurück" zu POJO/POJI
- Verwendung von Meta-Annotationen
- Kapselung von Umgebungs- und JNDI-Zugriffen
- Weitere Neuerungen / Erleichterungen
- Session Bean
- Message-driven Bean
- Entity Bean / Persistenz
- Resümee

Vereinfachungⁿ

Ease of development

Ease of use



Hauptziele von EJB 3.0

Wichtige Randbedingungen

- EJB 3.0 ist abwärtskompatibel
 - ▶ EJB 2.0/2.1 Komponenten laufen auch in EJB 3.0 Containern
 - ▶ EJB 2.0/2.1 und EJB 3.0 Komponenten lassen sich parallel ausführen

- Implementierungsstile sind wahlfrei
 - ▶ Nutzung der von EJB 3.0 verwendeten Meta-Annotationen ist optional
 - ▶ Nutzung des komplexeren EJB 2.0/2.1 Mechanismus ist optional
 - **EJB 2.0/2.1 APIs werden ebenfalls vorsichtig vereinfacht und erweitert**

Hauptziele von EJB 3.0

- Vereinfachung der Struktur von EJB-Komponenten
 - ▶ Bis dato für gewöhnlich fünf (5) Softwareartefakte für eine EJB-Komponente nötig
 - **Erheblicher Entwicklungsaufwand**
 - **Höhere Fehleranfälligkeit bei der Implementierung**
 - **Infrastruktureller Code (will keiner sehen; erst recht nicht implementieren)**
 - ▶ Reduzierung der Fragmente von EJB-Komponenten
 - **Wegfall der Home Interfaces und Callback-Interfaces**
 - ▶ Wegfall von Methoden, die nicht benötigt werden
 - **Beispiel Stateless Session Bean:**
`create()` erzeugt nichts und `remove()` löscht nichts
 - ▶ Implementierung als POJOs/POJIs, statt komplexer Komponenten

Hauptziele von EJB 3.0

- Vereinfachung des Entwicklungsprozesses
 - ▶ Einfache Entwicklung von 'einfachen' Komponenten
 - ▶ Förderung der testgetriebenen Entwicklung
 - **Inversion of Control**
 - **Dependency Injection**
 - ▶ Verwendung von Meta-Annotationen...
 - **eine Codebasis**
 - **Überprüfbarkeit zur Kompilierungszeit**
 - ▶ Default-Verhalten von EJB-Komponenten
 - **Configuration by Exception**
 - **Reduzierung des Programmieraufwands**

Hauptziele von EJB 3.0

- Vereinfachung der Nutzung von EJB-Komponenten
 - ▶ Vereinfachung bestehender APIs
 - **Eliminierung von nicht (mehr) benötigten Methoden**
 - **Eliminierung der.ejb...-Callbackmethoden**
 - **Vereinfachte Aufrufsemantik aus Sicht von Clients**
 - ▶ Reduzierung des Sets der APIs
 - **Wegfall von Schnittstellen, die nicht (mehr) benötigt werden**
 - ▶ Eliminieren Deployment Deskriptoren (zumindest aus Sicht des Programmierers)
 - **zu komplex**
 - **zu aufwändig**
 - **zu fehlerträchtig**
 - **keine Prüfungen zur Kompilierungszeit**

Agenda

EJB 3.0

- Was bis jetzt geschah
- Hauptziele von EJB 3.0
- **"Zurück" zu POJO/POJI**
- Verwendung von Meta-Annotationen
- Kapselung von Umgebungs- und JNDI-Zugriffen
- Weitere Neuerungen / Erleichterungen
- Session Bean
- Message-driven Bean
- Entity Bean / Persistenz
- Resümee

"Zurück" zu POJO/POJI

Beispiel: EJB 2.0/2.1 – Status Quo

Component Interface

```
public interface Reservierer extends EJBObject {  
    public void reserviereFlug(String flugNr, int anzPlaetze, ...) throws RemoteException;  
    public void storniereFlugBuchung(int buchungKey) throws RemoteException;  
}
```

Home Interface

```
public interface ReserviererHome extends EJBHome {  
    Reservierer create() throws CreateException, RemoteException;
```

Bean-Klasse

```
public class ReserviererBean implements SessionBean {  
    private SessionContext ctx;  
    public void setSessionContext(SessionContext ctx) {  
        this.ctx = ctx;  
    }  
    public void ejbCreate () {}  
    public void ejbActivate () {}  
    public void ejbPassivate () {}  
    public void ejbRemove() {}  
  
    public void reserviereFlug (String flugNr, int anzPlaetze, ...) {  
        ...  
    }  
    public void storniereFlugBuchung (int buchungKey) {  
        ...  
    }  
}
```

Deployment Deskriptor

```
...  
<session>  
    <ejb-name>ReserviererEJB</ejb-name>  
    <home>com.example.ReserviererHome</home>  
    <remote>com.example. Reservierer</remote>  
    <ejb-class>com.example. ReserviererBean</ejb-class>  
    <session-type>Stateless</session-type>  
    <transaction-type>Container</transaction-type>  
</session>  
...
```

"Zurück" zu POJO/POJI

Beispiel: EJB 3.0 - POJO + POJI

Component Interface

```
public interface Reservierer {  
    public void reserviereFlug(String flugNr, int anzPlaetze, ...);  
  
    public void storniereFlugBuchung(int buchungKey);  
}
```

Bean-Klasse

```
@Stateless public class ReserviererBean implements Reservierer{  
    public void reserviereFlug (String flugNr, int anzPlaetze, ...) {  
        ...  
    }  
    public void storniereFlugBuchung (int buchungKey) {  
        ...  
    }  
}
```

Home Interface



Deployment Deskriptor



"Zurück" zu POJO/POJI

Beispiel: EJB 3.0 - POJO und generiertes POJI

Component Interface

...wird generiert

```
public interface Reservierer {  
    public void reserviereFlug(String flugNr, int anzPlaetze,...);  
  
    public void storniereFlugBuchung(int buchungKey);  
}
```

Bean-Klasse

```
@Stateless public class ReserviererBean {  
    public void reserviereFlug (String flugNr, int anzPlaetze,...) {  
        ...  
    }  
    public void storniereFlugBuchung (int buchungKey) {  
        ...  
    }  
}
```

Home Interface

Deployment Deskriptor

"Zurück" zu POJO/POJI

Was ist passiert? **W**as haben wir gemacht?

- Forderung nach Home Interface wurde eliminiert
 - ▶ Weniger Implementierungsaufwand
 - ▶ Zugriff gestaltet sich aus Clientsicht einfacher
 - **Weniger Code notwendig**
 - **Kein Cast über `PortableRemoteObject.narrow(...)` mehr notwendig**
- Geschäftsschnittstelle (Business Interface (BI)) ist nun POJI
 - ▶ Bean-Klasse kann POJI implementieren
 - ▶ POJI kann auf Basis der Bean-Klasse (POJO) generiert werden
 - ▶ EJB(Local)Object aus Component Interface und Clientsicht entfernt
 - ▶ RemoteExceptions aus Component Interface und Clientsicht entfernt
- Geschäftsobjekt ist nun POJO
 - ▶ Forderung nach Implementierung der Callback-Methoden eliminiert

Agenda

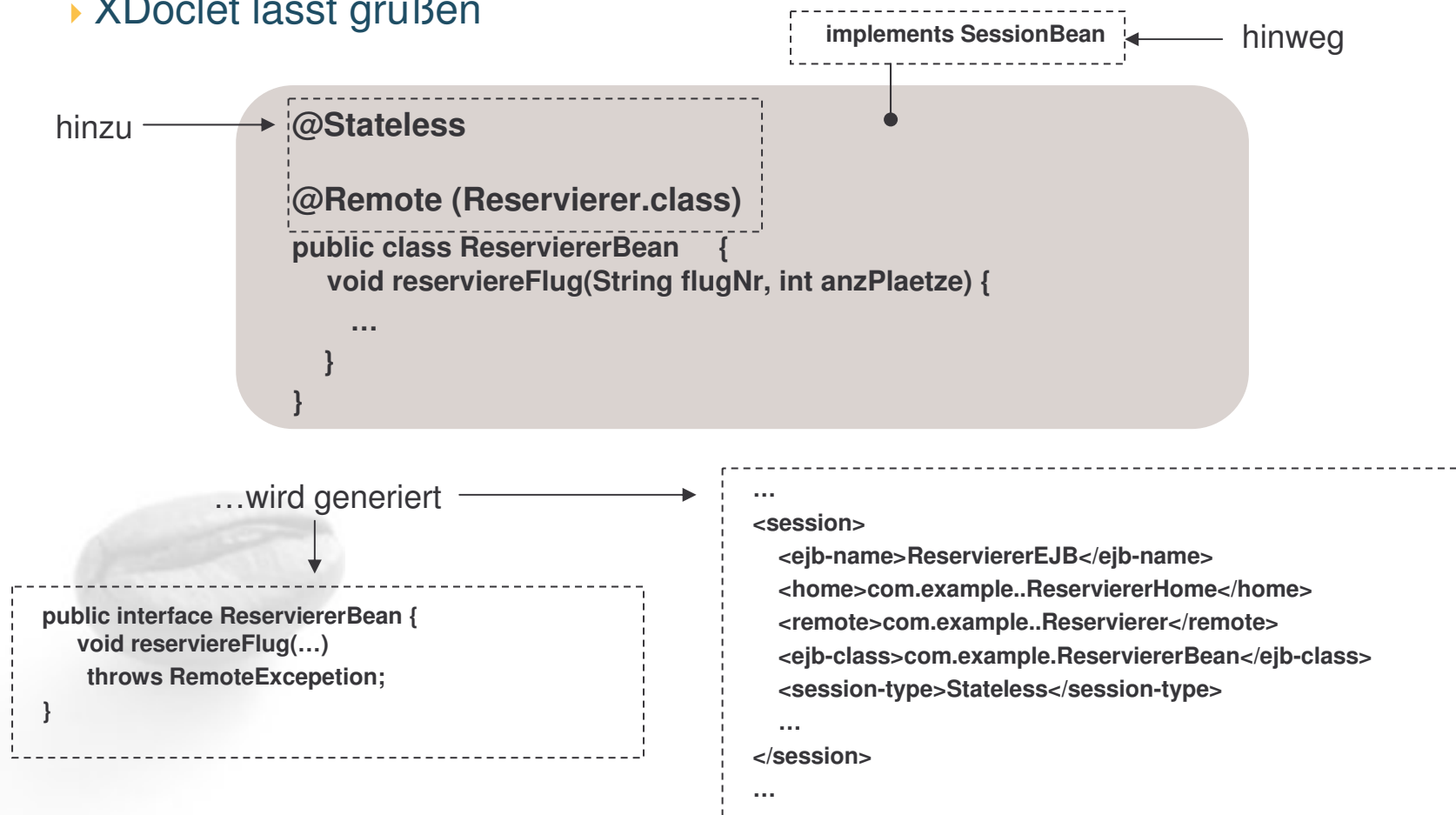
EJB 3.0

- Was bis jetzt geschah
- Hauptziele von EJB 3.0
- "Zurück" zu POJO/POJI
- **Verwendung von Meta-Annotationen**
- Kapselung von Umgebungs- und JNDI-Zugriffen
- Weitere Neuerungen / Erleichterungen
- Session Bean
- Message-driven Bean
- Entity Bean / Persistenz
- Resümee

Meta-Annotationen

■ Verwendung von Meta-Annotationen im Java-Quellcode

- ▶ Siehe JSR 175 (A Metadata Facility for the Java™ Programming Language)
- ▶ XDoclet lässt grüßen



Meta-Annotationen

- Meta-Annotationen ermöglichen der EJB-Technologie...
 - ▶ Generierung von Interfaces
 - ▶ Steuerung des Laufzeitverhaltens von EJB-Komponenten
 - ▶ Eliminierung der Deployment Deskriptoren
 - ▶ Demarkation von Injection-Punkten

- Annotationen können erfolgen für...
 - ▶ Klassen
 - ▶ Methoden
 - ▶ Parameter
 - ▶ Attribute

Agenda

EJB 3.0

- Was bis jetzt geschah
- Hauptziele von EJB 3.0
- "Zurück" zu POJO/POJI
- Verwendung von Meta-Annotationen
- **Kapselung von Umgebungs- und JNDI-Zugriffen**
- Weitere Neuerungen / Erleichterungen
- Session Bean
- Message-driven Bean
- Entity Bean / Persistenz
- Resümee

Kapselung von Umgebungs- und JNDI-Zugriffen

- ...durch Nutzung von Meta-Annotationen
 - ▶ Dependency Annotation
 - ▶ Dependency Injection
- EJB-Container initialisiert annotierte Attribute bzw. Methoden automatisch mit Referenzen auf die gewünschten Ressourcen
 - ▶ Geschieht vor der ersten Ausführung einer Geschäftsmethode
- Vorteil: es muss weniger bis gar kein Code mehr geschrieben werden, um auf Ressourcen zuzugreifen



Kapselung von Umgebungs- und JNDI-Zugriffen

- Vorteil: ermöglicht ENDLICH einfaches Testen von EJB-Komponenten.
 - ▶ J2EE-Applikationsserver als Testumgebung mehr zwingend notwendig
 - ▶ Über **setter**-Methoden werden einer Komponente zur Laufzeit entsprechende Informationen zur Verfügung gestellt (injiziert).
 - ▶ Bekannt als „Inversion of Control“ (IOC)...
 - „**Don't call us – we call you**“ (Hollywood-Prinzip)
 - ▶ ...und „Dependency Injection“ (DI)
- Neue Methode `<Object lookup(String name)>` zum `EJBContext` hinzugefügt
 - ▶ Direkter Zugriff für Beans auf den JNDI-Baum; ohne extra Code

Kapselung von Umgebungs- und JNDI-Zugriffen

■ Varianten des Zugriffs

- ▶ `@EJB` → Referenzen auf EJB-Komponenten
- ▶ `@Resource` → Referenzen auf Ressourcen

```
@Stateless public class MySessionBean {  
  
    @Resource(name="myDB") private DataSource customerDB;  
  
    public void myMethod (String myString) {  
        try {  
            Connection conn = customerDB.getConnection();  
            ...  
        }  
        catch (Exception ex) { ...}  
    }  
}
```

```
@Session public class MySessionBean {  
    private DataSource customerDB;  
  
    @Resource private void setCustomerDB(DataSource customerDB) {  
        this.customerDB = customerDB;  
    }  
  
    public void myMethod (String myString) {  
        ...  
        Connection conn = customerDB.getConnection();  
        ...  
    }  
}
```

Dependency Injection (Varianten)

■ Injection von "Singletons" im Container

- ▶ `@Resource javax.ejb.SessionContext ctx;`
- ▶ `@ Resource javax.ejb.MessageDrivenContext ctx;`
- ▶ `@ Resource javax.ejb.TimerService timer;`
- ▶ `@ Resource javax.ejb.UserTransaction ut;`
- ▶ `@ Resource javax.ejb.EntityManager manager;`



Agenda

EJB 3.0

- Was bis jetzt geschah
- Hauptziele von EJB 3.0
- "Zurück" zu POJO/POJI
- Verwendung von Meta-Annotationen
- Kapselung von Umgebungs- und JNDI-Zugriffen
- **Weitere Neuerungen / Erleichterungen**
- Session Bean
- Message-driven Bean
- Entity Bean / Persistenz
- Resümee

Weitere Erleichterungen in EJB 3.0

- "Configuration by Exception"
 - ▶ Spezifiziertes Default-Verhalten von EJB-Komponenten
 - ▶ Ziel: Reduzierung der Entwicklungszeit
 - ▶ Beispiel:
 - **EJB-Komponenten (Session Beans) sind per default „lokal“ zugreifbar**
 - Explizite Steuerung durch Verwendung der Annotationen
@Local und @Remote für das Interface bzw. die Bean-Klasse
 - **Auf Attribute von Entity Beans wird per default über get/set-Methoden zugegriffen**
 - Explizite Steuerung durch Verwendung der Werte der Annotation
@Entity(access=PROPERTY) und @Entity(access=FIELD)

Weitere Erleichterungen in EJB 3.0

- Callback-Annotationen für Lifecycle Events
 - ▶ Vordefinierte Callbacks
 - ▶ Gekennzeichnet durch Annotationen (z.B. @PostConstruct, @PreDestroy)
 - ▶ Session Beans, Message-driven Beans, Entity Beans

- Interzeptor-Methoden
 - ▶ Abfangen von Geschäftsmethodenaufrufen
 - **Ermöglicht die Ausführung von beliebigem Code**
 - Z.B. technische Prüfungen, Security
 - ▶ Session Beans, Message-driven Beans

Agenda

EJB 3.0

- Was bis jetzt geschah
- Hauptziele von EJB 3.0
- "Zurück" zu POJO/POJI
- Verwendung von Meta-Annotationen
- Kapselung von Umgebungs- und JNDI-Zugriffen
- Weitere Erleichterungen
- **Session Bean (Stateless)**
- Message-driven Bean
- Entity Bean / Persistenz
- Resümee

Stateless Session Bean

- Home Interface entfällt
 - ▶ Zugriff auf EJB-Komponente aus Sicht des Clients vereinfacht

```
public static void main(...) throws Exception {  
    InitialContext ctx = new InitialContext();  
    Object o = ctx.lookup("Reservierer");  
    ReserviererHome home =  
        (ReserviererHome) PortableRemoteObject.narrow(o, ReserviererHome.class);  
  
    Reservierer reservierer = home.create();  
}
```

2.1

```
public static void main(...) throws Exception {  
    InitialContext ctx = new InitialContext();  
    Reservierer reservierer = (Reservierer) ctx.lookup("Reservierer");  
    ... // do something  
}
```

3.0

- ▶ Keine create()-Methode mehr notwendig

Stateless Session Beans besitzen kein Home Interface

Stateful Session Beans können create auf spezielle Init-Methoden verlagern

Stateless Session Bean

- Business-Interface muss vorhanden sein.
 - ▶ Einfaches Java-Interface (POJI)
 - ▶ Kein Erben von EJBObject oder EJBLocalObject
 - ▶ ...damit auch keine RemoteExceptions mehr in den Methoden
- Business-Interface kann generiert werden
- Bean-Klasse muss nicht mehr das Callback-Interface `javax.ejb.SessionBean` implementieren
 - ▶ `ejbRemove()` gibt es nicht mehr
→ ist für eine Stateless Session Bean eine völlig überflüssige Methode
- `@Stateless` annotiert Bean-Klasse als Stateless Session Bean

Stateless Session Bean

■ Beispiel

Component Interface

```
public Interface Reservierer {  
    public void reserviereFlug (String flugNr, int anzPlaetze);  
    public void storniereFlugBuchung (int buchungKey);  
}
```

Bean-Klasse

```
@Stateless public class ReserviererBean implements Reservierer {  
    public void reserviereFlug (String flugNr, int anzPlaetze) {  
        ...  
    }  
    public void storniereFlugBuchung (int buchungKey) {  
        ...  
    }  
}
```

Agenda

EJB 3.0

- Was bis jetzt geschah
- Hauptziele von EJB 3.0
- "Zurück" zu POJO/POJI
- Verwendung von Meta-Annotationen
- Kapselung von Umgebungs- und JNDI-Zugriffen
- Weitere Erleichterungen
- **Session Bean (Stateful)**
- Message-driven Bean
- Entity Bean / Persistenz
- Resümee

Stateful Session Bean

- Kein Home Interface mehr benötigt
 - ▶ Zugriff analog Zugriff auf Stateless Session Bean
 - ▶ Keine create()-Methode mehr obligatorisch existent
 - **Initialisierung der Bean erfolgt durch entsprechende Geschäftsmethode**
 - ...in Diskussion, ob die Kennzeichnung einer solchen Methode über Annotation erfolgt

- Muss ein Business-Interface haben
 - ▶ Einfaches Java-Interface (POJI)
 - ▶ Kein Erben von EJBObject oder EJBLocalObject
 - ▶ ...damit auch keine RemoteExceptions mehr in den Methoden

- Business-Interface kann generiert werden

Stateful Session Bean

- Bean-Klasse muss nicht mehr das Callback-Interface `javax.ejb.SessionBean` implementieren (optional)
- `@Stateful` annotiert Bean-Klasse als Stateful Session Bean
- `remove()`-Methode kann beliebige Methode der Bean-Klasse sein
 - ▶ Annotiert durch `@Remove`



Stateful Session Bean

■ Beispiel

Component Interface

```
public Interface FlugzeugBelegungsManager {  
    public void reserviereFlug (String flugNr, int anzPlaetze);  
    public void storniereFlugBuchung (int buchungKey);  
    @Remove public void closeManager();  
}
```

Bean-Klasse

```
@Stateful public class FlugzeugBelegungsManagerBean implements FlugzeugBelegungsManager {  
    public void reserviereFlug (String flugNr, int anzPlaetze) {  
        ...  
    }  
    public void storniereFlugBuchung (int buchungKey) {  
        ...  
    }  
    @Remove public void closeManager() {  
        // do some cleanup  
    }  
}
```

Agenda

EJB 3.0

- Was bis jetzt geschah
- Hauptziele von EJB 3.0
- "Zurück" zu POJO/POJI
- Verwendung von Meta-Annotationen
- Kapselung von Umgebungs- und JNDI-Zugriffen
- Weitere Erleichterungen
- Session Bean
- **Message-driven Bean**
- Entity Bean / Persistenz
- Resümee

Message-driven Bean

- MDB muss entweder mit `@MessageDriven` Annotation gekennzeichnet werden oder das Callback-Interface `javax.ejb.MessageDrivenBean` implementieren.



Agenda

EJB 3.0

- Was bis jetzt geschah
- Hauptziele von EJB 3.0
- "Zurück" zu POJO/POJI
- Verwendung von Meta-Annotationen
- Kapselung von Umgebungs- und JNDI-Zugriffen
- Weitere Erleichterungen
- Session Bean
- Message-driven Bean
- **Entity Bean / Persistenz**
- Resümee

Entity Bean / Persistenz

- Weiterentwicklung Persistenz
 - ▶ CMP/BMP Nutzung vereinfachen
 - ▶ Leistungsumfang erhöhen
 - ▶ Schwerpunkt CMP
- Ziel: Simplifizierung von Entity Beans → POJOs
- Konkrete Klasse; nicht mehr abstrakt
- @Entity annotiert Bean-Klasse als Entity Bean
- Leichtgewichtiges Domänen-Objekt
- **Entity Beans sind nicht mehr direkt entfernt zugreifbar!**

Entity Bean / Persistenz

- Zugriff erfolgt nur noch über neu eingeführten `javax.ejb.EntityManager`

- **EntityManager**

- ▶ Erzeugen und Löschen von persistenten Entity-Instanzen
- ▶ Finden von Entitäten über Primärschlüssel
- ▶ Abfragen über Entitäten

- **Beispiel**



```
@Stateless public class OrderEntry {
    EntityManager em;

    @Inject public void setEntityManager(EntityManager em) {
        this.em = em;
    }

    public int createOrder(String articleNo, int quantity, ...) {
        Order order = new Order(articleNo, quantity, ...);
        em.create(order);
        return order.getId();
    }

    public Order find(int id) {
        return em.find(Order.class, id);
    }

    public void enterOrder(int custID, Order newOrder) {
        Customer cust = (Customer)em.find("Customer", custID);
        cust.getOrders().add(newOrder);
        newOrder.setCustomer(cust);
    }
}
```

Entity Bean / Persistenz

- Entity Beans beherrschen nun Vererbung und Polymorphie
- Polymorphe EJB QL-Abfragen sind nun zulässig
- Kein Home Interface mehr nötig
- Kein Business Interface mehr nötig (technisch betrachtet ;-)
- JavaBean-Stil
 - ▶ Attribute sind nur per getter/setter-Methoden zugreifbar
 - ▶ Eliminieren der komplexen, aufwändigen und fehleranfälligen Deployment Deskriptoren
 - **Beziehungen etc. werden in den Klassen per Annotationen deklariert**

Entity Bean / Persistenz

- Kein DTO für Transport von Entity Bean-Daten mehr notwendig
 - ▶ Entity Bean selbst wird transferiert (detached object)
 - ▶ Nach Änderungen am 'detached object' können diese mit dem persistenten Zustand zusammengebracht werden (reattached object)



Entity Bean / Persistenz

■ Beispiel

Bean-Klasse

```
@Entity
@Table(name = "PURCHASE_ORDER")
public class Order implements java.io.Serializable {
    private int id;
    private double total;
    private Collection<LinItem> linItems;

    @Id(generate = GenerationType.AUTO) public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public double getTotal() { return total; }
    public void setTotal(double total) { this.total = total;}

    public void addPurchase(String product, int quantity, double price) {
        if (linItems == null) linItems = new ArrayList<LinItem>();
        LinItem item = new LinItem();
        item.setOrder(this);
        ...
    }

    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
    @JoinColumn(name = "order_id")
    public Collection<LinItem> getLinItems() {
        return linItems;
    }

    public void setLinItems(Collection<LinItem> linItems) {
        this.linItems = linItems;
    }
}
```

Weiterentwicklung EJB QL

- Sub Queries

```
SELECT goodCustomer  
FROM Customer goodCustomer  
WHERE goodCustomer.balance < (  
    SELECT avg(c.balance) FROM Customer c)
```

- GroupBy

- Having

```
SELECT c.status, avg(c.filledOrderCount), count(c)  
FROM Customer c  
GROUP BY c.status  
HAVING s.status IN (1, 2)
```

- Explizite Inner-Joins und Outer-Joins

Weiterentwicklung EJB QL

■ Projections

```
SELECT c.id, c.status  
FROM Customer c JOIN c.orders o  
WHERE o.count > 100
```

Liefert ein Array von java.lang.Object-Objekten

```
SELECT new CustomerDetails(c.id, c.status, o.count)  
FROM Customer c JOIN c.orders o  
WHERE o.count > 100
```

Liefert eine Liste von CustomerDetails-Objekten

■ Bulk updates, Bulk deletes

```
DELETE  
FROM Customer c  
WHERE c.status = 'inactive'  
AND c.orders IS EMPTY
```

Weiterentwicklung EJB QL

- Unterstützung für native SQL-Abfragen
- Zusätzliche Funktionen in Where-Klausel
 - ▶ UPPER
 - ▶ LOWER
 - ▶ TRIM
 - ▶ POSITION
 - ▶ CHARACTER_LENGTH
 - ▶ CHAR_LENGTH
 - ▶ BIT_LENGTH
 - ▶ CURRENT_TIME
 - ▶ CURRENT_DATE
 - ▶ CURRENT_TIMESTAMP

Weiterentwicklung EJB QL

■ Dynamische Abfragen, Named-Queries

```
// Erzeugen einer NamedQuery
...
@NamedQuery(
    name="findAllCustomersWithName",
    queryString="SELECT c FROM Customer c WHERE c.name LIKE :custName"
)
...

// Nutzen einer NamedQuery
...
@Inject public EntityManager em;
customers = em.createNamedQuery("findAllCustomersWithName")
    .setParameter("custName", "Smith")
    .listResults();
...
```

■ Polymorphe Abfragen

- ▶ Liefert als Ergebnis nicht nur den spezifizierten Typ, sondern auch alle Subtypen

Agenda

EJB 3.0

- Was bis jetzt geschah
- Hauptziele von EJB 3.0
- "Zurück" zu POJO/POJI
- Verwendung von Meta-Annotationen
- Kapselung von Umgebungs- und JNDI-Zugriffen
- Weitere Erleichterungen
- Session Bean
- Message-driven Bean
- Entity Bean / Persistenz
- **Resümee**

Resümee EJB 3.0

- Statt komplexer Komponenten einfache POJOs / POJIs
 - ▶ Verkürzung der Entwicklungszeit
 - ▶ Wegfall von überflüssigen Interfaces (Home Interface, Component Interface und der Typen-Callback Interfaces)
 - ▶ Bessere und einfachere Modellierbarkeit
 - ▶ Einfache Java-Klassen mit allen Vorteilen von Container-Komponenten
- Statt Deployment Deskriptoren Meta-Annotation im Java-Code
 - ▶ Nur noch eine Codebasis
 - ▶ Erlaubt nun Prüfungen zur Kompilierungszeit
- Inversion of Control / Dependency Injection
- Configuration by Exception
 - ▶ Vereinfachung der Entwicklung (Reduzierung Code und Entwicklungszeit)

Resümee EJB 3.0

- Mit EJB 3.0 wird die Komponententechnologie erheblich einfacher und gleichzeitig mächtiger.
- EJB 3.0 wird die Entwicklung von komponenten-basierten Systemen beschleunigen
 - Konkurrenz .NET
- Das leidige Thema der konkurrierenden Persistenzstrategien innerhalb von J2EE/J2SE wird mit EJB 3.0 endlich behoben
 - es gibt „nur“ noch eine Lösung

Die EJB 3.0 Begleit-Kolumne

*Java*magazin

EJB 3.0 - Kolumne

- Ab Ausgabe 01/05 in jeder Ausgabe des Javamagazins ein Kolumnen-Artikel zu EJB 3.0
- Jeweils die aktuellsten Informationen direkt aus der EJB 3.0 Expert Group
- Beleuchtet in jeder Ausgabe dediziert einen Aspekt der kommenden Version
- Laufzeit: voraussichtlich bis zum finalen Release von EJB 3.0
 - ▶ Q1 / 2006

Ende - Vielen Dank!

Fragen & Antworten
Diskussion

oliver.ihns@resco.de



Weitere Informationen zu EJB 2.1 und EJB 3.0 unter
www.ejbkomplett.de