5.– 8. September 2011
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Resümee

EJB 3.1 – Rückblick nach dem ersten Jahr

Werner Eberling

MATHEMA Software GmbH

Agenda

- JEE 6 in der Praxis?
- Bewertung der neuen Features in EJB 3.1
- Bewertung der neue Features in JPA 2.0

JEE 6 in der Praxis

- Wie kommt der neue Standard in der Praxis an?

oder

Neue Features in EJB 3.1

- Standardisierte JNDI-Namen
- SessionBeans
  - Singletons
  - Vereinfachte lokale Sicht
  - Asynchrone Methoden
  - Ressource für RESTful WebServices
- Automatische Timer
- Vereinfachtes Deployment

Standardisierte JNDI-Namen

- JNDI Namen nicht mehr serverspezifisch
  - java:global/[app-name]/{modul-name}/{bean-name}![if-name]

- Definition verschiedener Namensräume
  - java:global
  - java:app
  - java:module

Standardisierte JNDI-Namen

- EJB 3.0

```
// JBOSS
Context ctx = new InitialContext();
ctx.lookup("CampusBean/remote");


// GLASSFISH
Context ctx = new InitialContext();
ctx.lookup("de.mathema.Campus");
```

- EJB 3.1

```
// JBOSS
Context ctx = new InitialContext();
ctx.lookup(
  "java:global/campus/CampusBean");


// GLASSFISH
Context ctx = new InitialContext();
ctx.lookup(
  "java:global/campus/CampusBean");
```

Standardisierte JNDI-Namen

- EJB 3.0

```
// JBOSS
Context ctx = new InitialContext();
ctx.lookup("CampusBean/remote");


// GLASSFISH
Context ctx = new InitialCon
ctx.lookup("de.mathema.Ca
```

- EJB 3.1

```
// JBOSS
Context
ctx.l
                                  ntext();
                                  s/CampusBean");


                                  ISH
                                  t ctx = new InitialContext();
                                  x.lookup(
                                    "java:global/campus/CampusBean");
```

bringt Struktur – aber nicht weltbewegend

Singleton SessionBean

- Nur eine Bean-Instanz pro Server
  - JVM-Singleton
- Einziger multithreaded EJB-Typ
  - Synchronisatzion notwendig (CONTAINER vs. BEAN)
- Bringt neue Design-Möglichkeiten
  - Bsp.: Einfacher Cache
  - Evtl. die schlankere Stateless SessionBean?

Singleton SessionBean

- EJB 3.0

```
@Stateless
public class ZinsCacheImpl
             implements ZinsCache{

 static Map<Integer,Double>
  zinsen = new ConcurrentHashMap();

 public double getZins(int jahre){
  return zinscache.get(jahre);
 }
}
```

- EJB 3.1

```
@Singleton
@ConcurrencyManagement(
  ConcurrencyManagementType.BEAN)
public class ZinsCacheImpl
              implements ZinsCache {

 Map<Integer,Double> zinsen =
            new ConcurrentHashMap();

 public double getZins(int jahre){
  return zinscache.get(jahre);
 }
}
```

Singleton SessionBean

- EJB 3.0

```
@Stateless
public class ZinsCacheImpl
             implements ZinsCache{

 static Map<Integer,Double>
  zinsen = new ConcurrentHashMap

 public double getZins(in
  return zinscache.get
 }
}
```

- EJB 3.1

```
@Singleton
@Concurrency
  Concur                    e.BEAN)
publi                       mpl
                          nts ZinsCache {

                  ,Double> zinsen =
                    new ConcurrentHashMap();

  ublic double getZins(int jahre){
   return zinscache.get(jahre);
  }
}
```

mehr Freiheit im Design – gute Erweiterung

Vereinfachte lokale Sicht

- Lokales Business Interface als technischer Overhead
- Lokal auch Zugriff „direkt“ auf die Implementierung möglich
  - ABER: GUARD noch immer vorhanden
- Gefahr: alle public-Methoden werden zugreifbar!

Vereinfachte lokale Sicht

- EJB 3.0

```
@Stateless
public class ZinsCacheImpl
             implements ZinsCache{

 static Map<Integer,Double>
  zinsen = new ConcurrentHashMap();

 public double getZins(int jahre){
  return zinscache.get(jahre);
 }
}
```

- EJB 3.1

```
@Singleton
@ConcurrencyManagement(
  ConcurrencyManagementType.BEAN)
public class ZinsCacheImpl{

 Map<Integer,Double> zinsen =
            new ConcurrentHashMap();

 public double getZins(int jahre){
  return zinscache.get(jahre);
 }
}
```

Vereinfachte lokale Sicht

- EJB 3.0

```
@Stateless
public class ZinsCacheImpl
             implements ZinsCache{

 static Map<Integer,Double>
  zinsen = new ConcurrentHashMap()

 public double getZins(int
  return zinscache.get(
 }
}
```

- EJB 3.1

```
@Singleton
@Concurre
  Conc                    ype.BEAN)
pu                      heImpl{

                 Double> zinsen =
                   new ConcurrentHashMap();

  blic double getZins(int jahre){
  return zinscache.get(jahre);
 }
}
```

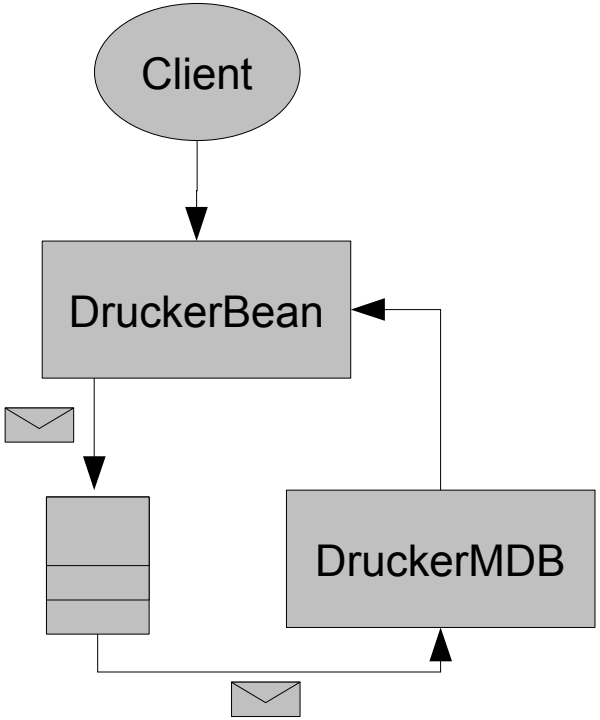
wenige Anwendungsfälle, viele Stolperfallen – unnötig

Asynchrone Bean-Methoden

- Asynchroner Aufruf von SessionBean-Methoden
- Mögliche Rückgabewerte
  - Void
  - Future<T>
- Checked Exceptions nur bei Future<T>

Asynchrone Bean-Methoden

- EJB 3.0



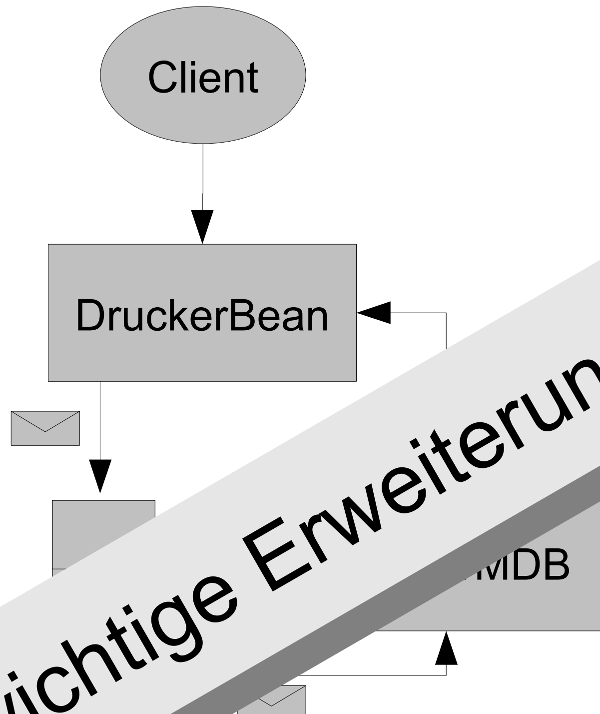
- EJB 3.1

```
@Stateless
public class DruckerImpl
             implements Drucker{

 @Asynchronous
 public void drucke(Job job){
  // ...
 }
}
```

Asynchrone Bean-Methoden

- EJB 3.0



- EJB 3.1

```
@St
            erImpl
            mplements Drucker{

        ronous
    lic void drucke(Job job){
    // ...
  }
}
```

wichtige Erweiterung – macht das Leben einfacher

Ressource für RESTful WebServices

- Eigentlich Teil der JAX-RS-Spezifikation
- Sinnvolle Erweiterung der vorhandenen WebService-Unterstützung
- Konfigurationsaufwand abhängig vom jeweiligen Server

Ressource für RESTful WebServices

```
@Path("Kunde")
public interface KundenManagement {
    @POST
    @Produces("application/xml")
    public Kunde erzeuge(@QueryParam("name") String name,
            @QueryParam("kdnr") int kundenNummer);

    @GET
    @Path("{id}")
    @Produces("application/json")
    public Kunde finde(@PathParam("id") int kundenNummer);

    @DELETE
    @Path("{id}")
    @Produces("application/xml")
    public void loesche(@PathParam("id") int kundenNummer);
}
```

Ressource für RESTful WebServices

```
@Path("Kunde")
public interface KundenManagement {
    @POST
    @Produces("application/xml")
    public Kunde erzeuge(@Query
            @QueryParam("k
    @GET
    @Path("{id}")
    @Produces
    public
                                    Param("id") int kundenNummer);

                 application/xml")
            void loesche(@PathParam("id") int kundenNummer);
```

wichtige Integration in der Welt der REST-Architekturen

Automatische Timer

- Timeout-Methoden seit EJB 2.x bekannt
  - Nur Intervalle definierbar
  - Kompliziert zu aktivieren
- Deklarative Deklaration wiederkehrender Timer möglich
  - Definition per Annotation (Cron-Syntax)
  - Automatische Aktivierung beim Deployment

Automatische Timer

- Timeout-Methoden seit EJB 2.x bekannt
  - Nur Intervalle definierbar
  - Kompliziert zu aktivieren
- Deklarative Deklarati… …nder Timer möglich
  - Definition n… …n-Syntax)
  - Autor… …g beim Deployment

lange benötigte Erweiterung – endlich „brauchbare“ Timer

Vereinfachtes Deployment

- Paketierung in standard Java-Archiv (JAR)
  - Deployment-Deskriptor in Verzeichnis MATA-INF
- Mehrere EJB-JARs können in ein Enterprise-Archiv (EAR) paketiert werden
  - Auch WARs und Java-Bibliotheken möglich
  - Optionaler Applikations-Deskriptor (META-INF/application.xml)
- Paketierung auch als Teil eines Web-Archives (WAR) möglich
  - Vereinfachtes Deployment
  - Unterstützt seit Servlet 3.0 (JEE 6)

Vereinfachtes Deployment

- Paketierung von EJBs innerhalb eines WARs
  - EJB-Klassen direkt im WAR (WEB-INF/classes)
  - EJB-JAR(s) als Library im WAR (WEB-INF/lib)
- Deployment der EJBs erfolgt im Rahmen des WAR-Deployments
  - automatisch
  - weiterhin in den EJB-Container
  - Es gelten die Classloader-Einstellungen des WAR-Classloaders!

Vereinfachtes Deployment

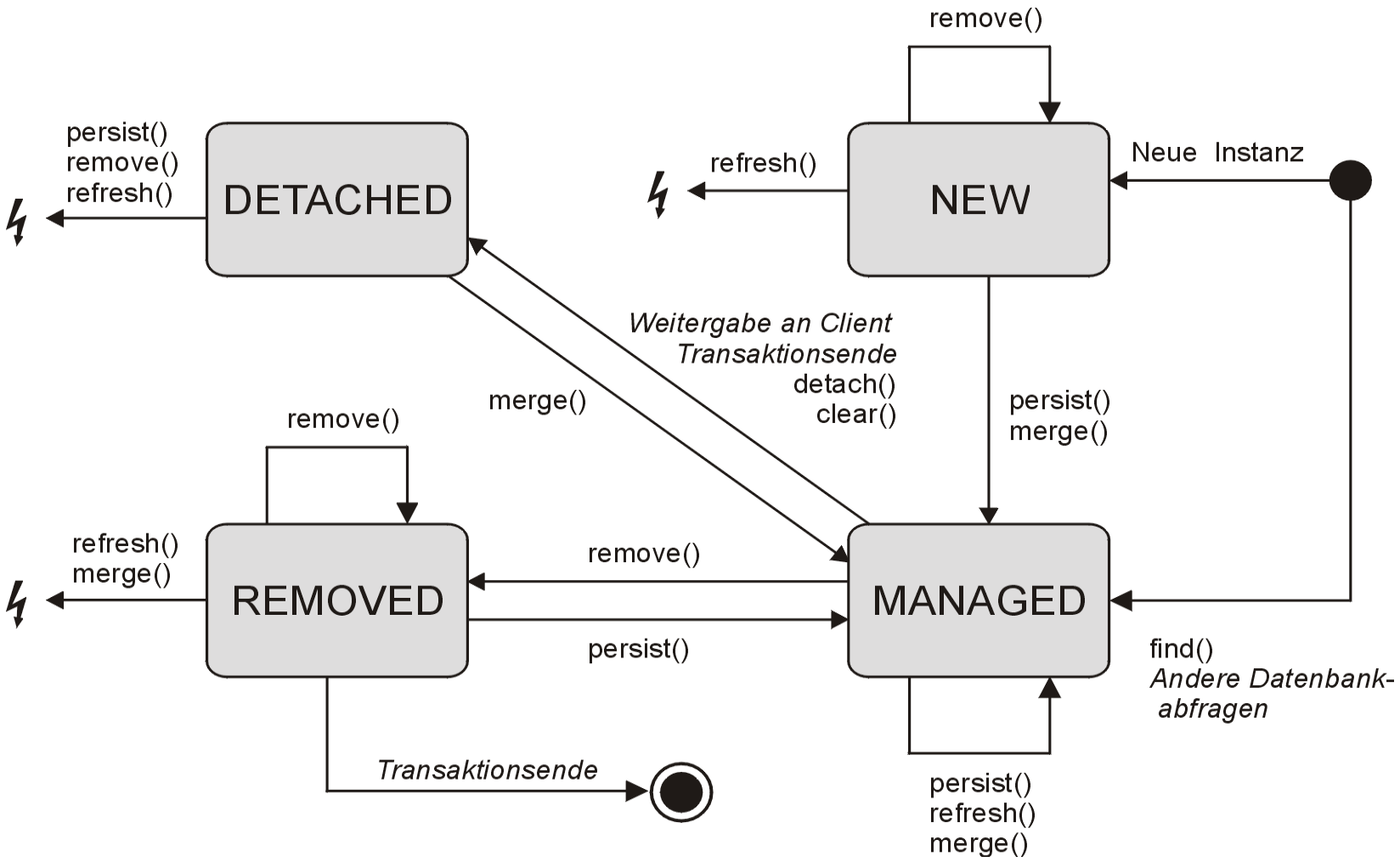
- Paketierung von EJBs innerhalb eines WA
  - EJB-Klassen direkt im WAR (WEB-INF/c
  - EJB-JAR(s) als Library im WAR (W
- Deployment der EJBs
  - automatisch
  - weiterh
  - E

entfernt u.U. unnötigen Overhead – gute Sache

Neue Features in JPA 2.0

- Explizites Detachment
- Criteria Builder
- Bean Validation
- Second-Level Cache
- Und noch 'ne Schippe O/R-Mapping

Neuer Punkt im Lebenszyklus der Entity



Programmatisches Detachment

- Applikation kann Zeitpunkt für Detachment selbst bestimmen
- Bisherige Intransparenz entfällt

```
public class KundenverwaltungImpl implements Kundenverwaltung{
 @PersistenceContext
 EntityManager em;

 public Kunde getKunde(int kundennummer){
  Kunde ergebnis = em.findById(kundennummer, Kunde.class);
  if (ergebnis != null)
    em.detach(ergebnis);
  return ergebnis;
 }
}
```

Programmatisches Detachment

- Applikation kann Zeitpunkt für Detachment bestimmen
- Bisherige Intransparenz entfällt

```
public class Kundenverwa                    s Kundenverwaltung{
 @PersistenceContext
 EntityManager em

 public Ku                                  ndennummer){
  Kunde                                   dById(kundennummer, Kunde.class);
  if                                      1)
                                          bnis);
                                          ls;
```

wichtiger Freiheitsgrad für Architekturentscheidungen

Criteria API

- API zur Konstruktion syntaktisch korrekter Queries
  - Erkennbarkeit von Fehlern zur Compilezeit
  - Idealer Weise in Verbindung mit Metamodel-API
  - Generiert aus Entityschema
  - Falsche Schemabezüge fallen ebenfalls zur Compilezeit auf
  - Integration in IDEs über Annotation Processor

Criteria API

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<String> abfrage = cb.createQuery(String.class);
Root<Fahrzeug> fahrzeuge = abfrage.from(Fahrzeug.class);
abfrage.select(fahrzeuge.get(Fahrzeug_.kennzeichen));
```

EJB-QL

```
String abfrage = "SELECT f.kennzeichen FROM Fahrzeug f";
```

Criteria API

- API zur Konstruktion syntaktisch korrekter Que
  - Erkennbarkeit von Fehlern zur Compilezeit
  - Idealer Weise in Verbindung mit Metamod
  - Generiert aus Entityschema
  - Falsche Schemabezüge fallen e lezeit auf
  - Integration in IDEs über A sor

gute Intention aber leider viel, viel zu kompliziert

```
CriteriaBuilder                iaBuilder();
CriteriaQuer                   = cb.createQuery(String.class);
Root<Fah                       abfrage.from(Fahrzeug.class);
abfra                          e.get(Fahrzeug_.kennzeichen));
```

Criteria API

```
                age = "SELECT f.kennzeichen FROM Fahrzeug f";
```

EJB-QL

Bean-Validation

- Attribut-Constraints für Entities über Annotationen
  - Automatische Auswertung bei DB-Synchronisation
  - Oder programmatisch
  - Oder z.B. durch GUI-Frameworks

```
@Entity
public class Fahrzeug implements Serializable {

  @Pattern("[A-Z]{1,3}-[A-Z]{1,3} [1-9]+[0-9]{0,3}")
  @Id String kennzeichen;

  String modell;

  @Min(0) @Max(2000000)
  int kilometerstand;
}
```

Bean-Validation

- Attribut-Constraints für Entities über Anno
  - Automatische Auswertung bei DB-Synchro
  - Oder programmatisch
  - Oder z.B. durch GUI-Framewo

```
@Entity
public class F                    s Serializable {

   @Pat                           A-Z]{1,3} [1-9]+[0-9]{0,3}")
   @                              men;

                                  1;

                    0)  @Max(2000000)
                        kilometerstand;
}
```

nahtlose Integration, vielseitig und zukunftsträchtig

Second-Level Cache

- Caching über die Grenze von Transaktionen hinaus
- JPA definiert nicht das Verhalten, sondern nur
  - Deklaratives Ein-/Abschalten
  - Programmatisches Fluten
- Standardisierter Zugang zu einem hersteller-spezifischen Feature ?!

Second-Level Cache

- Caching über die Grenze von Transaktion
- JPA definiert nicht das Verhalte
  - Deklaratives Ein-/Abschalten
  - Programmatisches Flut
- Standardisi einem hersteller-spezifi

risikoreich, im Standard nicht konsequent verankert

Noch mehr O/R-Mapping

- @Access
- @Cacheable
- @CollectionTable
- @ElementCollection
- @OrderColumn
- @MapKey
- @MapKeyClass
- @MapKeyColumn
- @MapKeyEnumerated
- @MapKeyJoinColumn
- @MapKeyJoinColumns
- @MapKeyTemporal
- @MapsId

```
@Embeddable public class Position {
    public int x;
    public int y;
}

@Entity class Vieleck {
    @Id private int id;

    @ElementCollection @OrderColumn
    private Collection<Position> stuetzpunkte;
}
```

Noch mehr O/R-Mapping

- @Access
- @Cacheable
- @CollectionTable
- @ElementCollection
- @OrderColumn
- @MapKey
- @MapKeyCla
- @MapKe
- @M ted
- Column
- yJoinColumns
- apKeyTemporal
- @MapsId

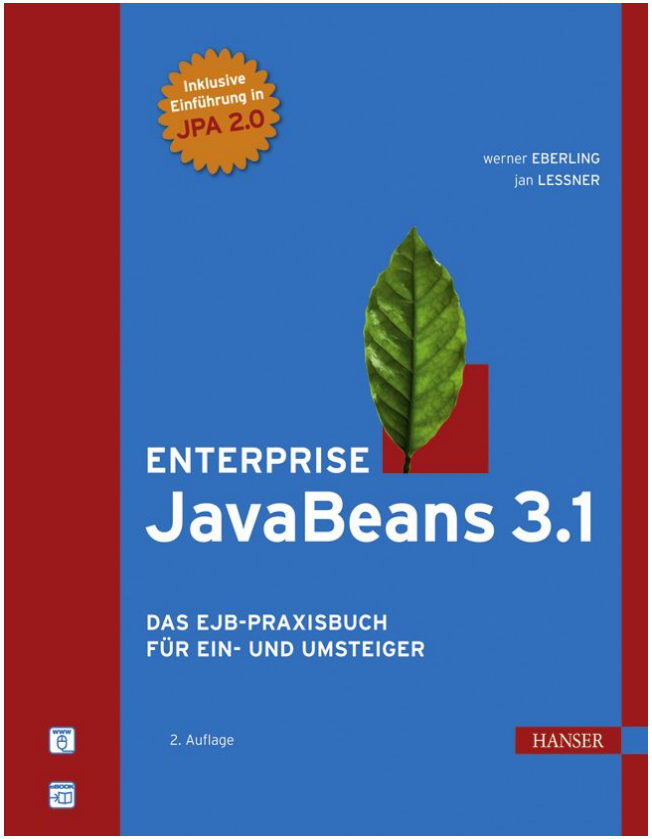
```
@Embeddable public c
    public int x
    public in
}

@Er                {
                nt id;

        tCollection @OrderColumn
        vate Collection<Position> stuetzpunkte;
```

hauptsächlich Lückenschluss, nichts Weltbewegendes

Mehr zum Thema



- 2. Auflage (Februar 2011)
- 364 Seiten zu EJB und JPA
- NEU: Kochrezepte-Kapitel
- ISBN 3-446-42259-5
- Beispiele zum Download

Vielen Dank!

Werner Eberling

MATHEMA Software GmbH