Enterprise Application Frameworks Teil 2 - MSP

Roland Hediger

30. Januar 2014

Inhaltsverzeichnis

Teil I.

Theorie

1. Einführung Enterpise Beans

Definition Ein Enterprise Bean ist eine serverseitige Komponente die die Businesslogik einer Applikation Kapselt (Facade).

- Beitet eine externe interface an (Remote)
- Versteckt komplexität der Implementation.

Zusätzliche Services Transaktionsmanagement, Sicherheitsdienste, Concurrency Management.

1.1. EJB Component Architecture

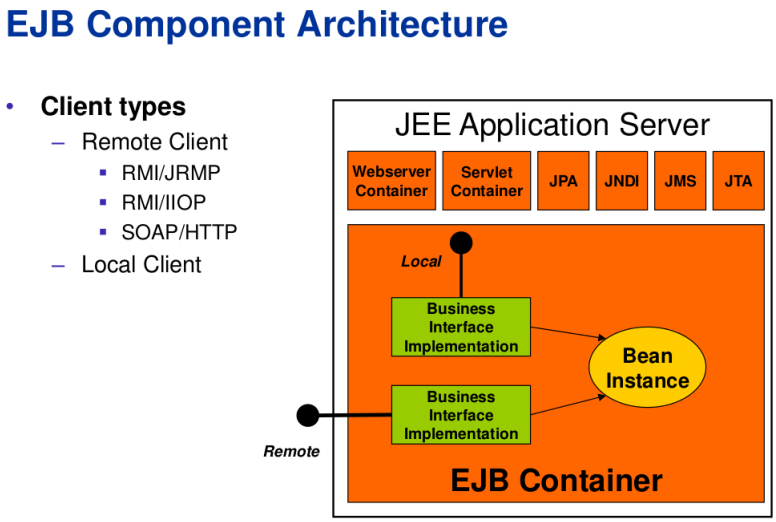

Abbildung 1.1.: figure

1.1.1. Synchrone Enterprise Beans

Stateless Session Bean Bietet Dienste an, kann ein Webservice Implementieren.Ahnlich wie Spring Beans mit `scope=Singleton`

Stateful Session Bean Instanzvariablen representieren eine einzigartige Client-Bean Session. Ähnlich wie Spring Beans mit `scope=Prototype`.

1.1.2. Asynchrone Enterprise Beans

Message Driven Beans Haltet die Daten oder der Zustand eines spezifischen Clients nicht. Kann auf JMS basiert sein.

1.2. Implementation von Enterprise Beans

Enterprise Bean Primäre Artifakt. Annotiertes POJO. Die Annotation geben die Semantik sowohl als die Vorraussetzungen an dem EJB Container.

Enterprise Business Interface Gewöhnliche Interface. Ein EJB[1] kann mehrere "Business Interfaces" implementieren.

[1] Enterprise Java Bean

Remote Interfaces `@Remote` : Entfernte Client kann auf einer anderen JVM laufen. Ort der EJB ist für Client Transparent.

Local Interfaces Die müssen *auf der gleichen JVM wie EJB laufen.* Ort ist für den Client nicht Transparent.

Unterschiede Performanz, Isolation von Paramter.

JNDI Access

Listing 1.1: JNDI

```
Properties props = new Properties();
props.put(Context.INITIAL_CONTEXT_FACTORY,
"org.jboss.naming.remote.client.InitialContextFactory");
props.put(Context.PROVIDER_URL, "remote://localhost:4447");
props.put(Context.SECURITY_PRINCIPAL, "testuser");
props.put(Context.SECURITY_CREDENTIALS, "testpassword");
context = new InitialContext(props);
context.lookup(...)
```

1.3. Stateless Session Beans

Eigenschaften. Bietet ein Dienst an für Clients. Verwaltet keine "Conversational State" mit Clients. Ein Stateless Bean kann **Instanzvariablen beinhalten, aber die sind nur für den Dauer des Aufrufs gültig.** EJB Container kann eine **beliebige Instanz** der Bean an dem Client geben bei einem Methodenaufruf. Client-to-Bean Assoziation ist daher kurz. Es kann jedoch **globale Zustandsinformation** beinhalten wie ein JNDI Context oder JDBC Verbindung. Diese Instanz ist nicht geteilt - 1 Client pro Instanz.

Listing 1.2: Calculator Example Stateless Bean

```
public interface Calculator {
  double sum (double x, double y);
  double difference (double x, double y);
  double product (double x, double y);
  double quotient (double x, double y);
}
// No interface inheritance - no remote xceptions!

// Implementation

@Stateless
@Remote(value = {Calculator,class})
public class CalculatorBean implements Calculator {
  ...//selbsterkl rend normale implementation.
}

// Client
public class Client {
public static void main(String[] args throws Exception{
 InitialContext ctx = new InitialContext();
Calculator calculator = (Calculator) ctx.lookup(
"calculator/CalculatorBean!calculator.bean.Calculator");
System.out.println("1 + 1 = "+calculator.add(1, 1));
System.out.println("1 - 1 = "+calculator.subtract(1, 1));
}
}
```

Remarks Die Stateless Annotation sagt dem EJB Container dass der CalculatorBean Stateless ist. Es hat folgende Eingabeeigenschaften :

- Name : Names des Beans default : Unqualified Class Name
- Description : Muss in tools angezeigt werden - Beschreibung der Bean.
- mappedName : mapping to a JNDI Name.

@Remote Ist dann business Interface. Value ist dann spezifiziert falls die Annotation auf eine Klasse angewendet ist. Interface nicht unbedingt durch einer Bean Klasse implementiert. Variante : Interface selbst annotieren.

@Local Ist dan ein Local Interface. Funktioniet nicht falls ein Lokale Client auf eine andere VM ist. Mögliche Anwendung - Web Container. **Das gleiche Business Interface kann nicht gleichzeitig Lokal und Remote sein.**

Bean Implementation muss ein Defaultkonstruktor haben.

1.4. Statelss Bean Lifecycle

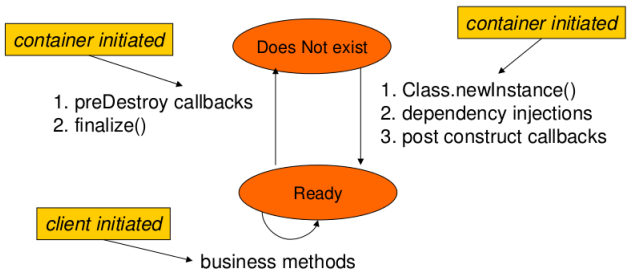

Abbildung 1.2.: figure

1.5. Callback Methods for Stateless Session Beans

@PostConstruct Nur einmal auf jeden Instanz nachdem Dependency Injections fertig sind. Aufgerufen bevor Ausführung der ersten Methode. Nur eine methode kann mit dieser Annotation markiert sein.

@PreDestroy Aufgerufen wenn der Beaninstanz zerstört ist. Sollte alle Resourcen schliessen, ähnlich wie finalize. Nur eine methode kann mit dieser Annotation markiert sein.

Bemerkung Diese Methoden sind in eine *noch nicht spezifizierte* Transaktions und Sicherheitskontext aufgerufen. Annotierte methoden sollen folgende Kritieren erfüllen:

- Keine Parameter, return type void. Sichtbarkeit ist irrelevant.
- Muss keine CheckedExceptions werfen.

1.6. Pooling

Ein Stateless Bean behandelt die Aufrufe von Clients. Kein Conversational State - Keine Abhängigkeit von Client.Teilen von ein Bean zwischen Clients bedeutet weniger Resourcen. **Beans sind Single Threaded**
Clustering: Bean Instanzen können auf mehrere Server sein.
Entfernung der Instanz von einem Stateless Bean ist transparent und von Container behandelt.

1.7. EJB Reuse - Dependency Injection

Listing 1.3: JNDI Lookup DI

```
// Service = remote, DAO = local binding
 private MovieDAO movieDAO;
@PostConstruct
private void init(){
try {
InitialContext ctx = new InitialContext();
movieDAO = (MovieDAO) ctx.lookup("...MovieDAO");
}
catch(NamingException e){
throw new RuntimeException(e);
}
}
```

Listing 1.4: Dependency Injection with Annotation

```
@EJB(name="...MovieDAO")
private MovieDAO movieDAO;
```

Parameters:

Name JNDI Name von EJB wenn nichts angegeben ist, ist es aus dem Feld oder property genommen (Container-spezifisch)

beanName Name die gegeben ist in Stateful oder Stateless Annotation

mappedName nichts

description nichts

beanInterface Object.class??

1.7.1. Persistence Context

```
@PersistenceContext(unitName="movie-db")
private EntityManager em;
```

Parameters:

Name Entity Manager Name nicht von DI benutzt.

UnitName Name der Persistence Unit aus persistence.xml

type Transaction/extended.

Properties key value pair.

1.8. Contexts und Dependency Injection (CDI)

```
// @Inject
// Field injection
@Inject
private T t;
//Method injection
private T t;
@Inject
public void setT(T t) { this.t = t; }
// Constructor injection
private T t;
@Inject
public C(T t) { this.t = t; }
```

@Produces Ein Bean kann mit CDI benutzt werden falls es ein Defaultkonstruktor hat oder ein Konstruktor der mit Inject annotiert ist. Produces erlaubt die benutzung von beliebigen klassen (Factory Methods)

```
public class Resources {
@Produces
public Logger produceLog(InjectionPoint
produceLog() {
p) {
Logger.getLogger("MovieRental");
return Logger.getLogger(
}
p.getMember().getDeclaringClass().getName());
}
}
//Default Scope ist DependantScope, neue Instanz per Injection.
```

CDI Extras

Listing 1.5: CDI mit Qualifier

```
    @Inject @MovieService
  Private Service movieService;
3 @MovieService
  public class MovieServiceImp implements Service { ... }

  //They are used on the implementation classes and on the injection points
```

2. Stateful Session Beans (SFSB)

- Behalten Zustandinformation zwischen Methodenaufrufe.
- Bean per Client
- State initializiert durch Client nach Erzeugung.
- Client muss siene Instanz entfernen oder entlassen.

  Container entfernt Bean nach Timeout falls kein Client es nutzt.

  Timeout kann per Annotation definiert werden:

```
@StatefulTimeout(value=5, unit=TimeUnit.SECONDS)
```

- **Conversational State nicht im DB obwohl es sollche Daten zugreifen kann und updaten kann** [1]
- **Stateful Bean kann "Transaction Aware" sein**

2.1. SFSB Examples

```
 @Remote
public interface Cart {
void
initialize(String owner);
void add
void (double amount);
double remove(double amount);
void getTotal();
    addTax();
void close();
}
@Stateful
public class CartBean implements Cart {
private String owner;
private double total;
private boolean taxAdded = false;
public void initialize(String owner) {
if(owner == null) throw new IllegalArgumentException();
this.owner = owner;
}
public double getTotal() {
return total;
}
@Remove public void close() {}public void add (double amount) {
if(owner==null || taxAdded)
throw new IllegalStateException();
total += amount;
}
public void remove(double amount) {
if(owner==null || taxAdded)
throw new IllegalStateException();
total -= amount;
}
public void addTax() {
if(owner==null || taxAdded)
throw new IllegalStateException();
total = total + total * 8.0 / 100;
taxAdded = true;
}
}

public class Client {
public static void main(String[] args) throws Exception
```

[1] wtf

```
String JNDI_NAME = "cart/CartBean!ch.fhnw.....Cart";
InitialContext ctx = new InitialContext();
Cart cart1 = (Cart) ctx.lookup(JNDI_NAME);
Cart cart2 = (Cart) ctx.lookup(JNDI_NAME);
48 cart1.initialize("User1");
cart2.initialize("User2");
cart1.add(100);
cart2.add(50);
System.out.println("Cart1: "+cart1.getTotal());
53 System.out.println("Cart2: "+cart2.getTotal());
cart1.close();
cart2.close();
}
}
```

2.2. SFSB Passivation und Activation

Schwierig zu übersetzen.

Conversational State The conversational state of a stateful session object is defined as the session bean instance field values, plus the transitive closure of the objects from the instance fields reached by following java object references.

Passivation/Activation

- To efficiently manage the size of its [memory], a session bean container may need to temporarily transfer the state of an idle stateful session bean instance to some form of secondary storage.
- Passivation: Transfer from the working set to secondary storage
- Activation: Transfer back
- Typically, the EJB container uses a LRU algorithm to select a bean for passivation

2.3. SFSB Serializierung

Was kann Serializiert werden?

- Primitives (byte / short / int / long / boolean / float / double)
- A reference to a Serializable object
- A null reference
- A reference to an EJBs business interface
- A reference to a naming context
- A reference to the UserTransaction interface
- A reference to a resource manager
- A reference to a javax.ejb.Timer object

2.3.1. passivation Ausschalten

```
@Stateful(passivationCapable=false)
```

2.4. SFSB Lifecycle

2.4.1. Callback Methoden

@PostConstruct Nach konstruktion nach DependencyInjections, bevor erste Methode ausgeführt ist.

@PreDestroy Nachdem alle Methoden die mit @Remote annotiert worden sind, fertig sind oder nach Timeout von 30 min falls Timeout aktiviert.

@PrePassivate Bevor Container Instanz "passiviert". Assert : Nicht Transiente Felder Serializierbar sind. Beispiel JDBC Conn geschlossen und auf null gesetzt.

@PostActivate Nachdem ein Instanz "reactivated" ist. Felder die im Prepassivate null sind mussen neu gesetzt werden.

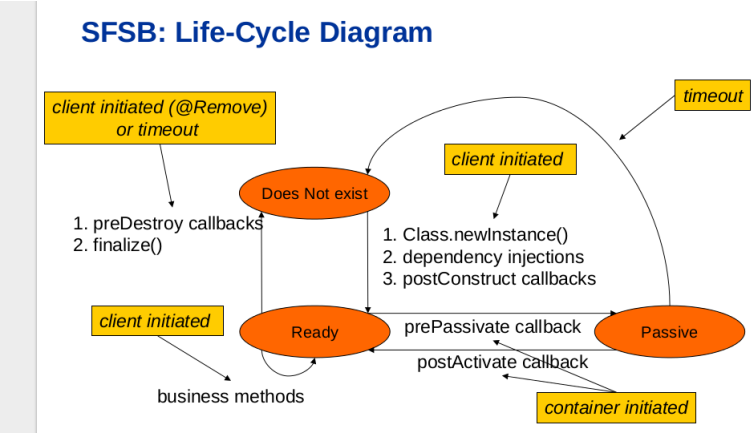

Abbildung 2.1.: figure

2.5. JBOSS Spezifisch : Limited Cache

Listing 2.1: Limited Cache

```
 caches>
<cache name="simple" aliases="NoPassivationCache"/>
<cache name="small" passivation-store-ref="small"/>
</caches>
<passivation-stores>
<file-passivation-store name="small"
idle-timeout="10" idle-timeout-unit="SECONDS"
max-size="2"/>
</passivation-stores>
```

- Falls max-size Beans aktiv sind, kann der Container mit Passivation anfangen.
- Fals ein Bean mehr als idle timeout inaktiv ist, wird er passiviert.

  Instanzen von passivierte Beans werden zerstört.

  Activationinstanzen erzeugt mit new, PostConstruct nicht benutzt hier.

  Zusätzliche init mit @PostActivate.

3. Transaktionen

Rollback Transaktion mit Rollback, dann sind die Veränderungen mit SQL nicht commited. Peristenzkontext wird entfernt, Detached Objekten können ungültige Daten beinhalten aber ist unproblematisch weil sobald die Entitäten neu geladen sind für eine Transaktion, werden die dann die richtigen Daten haben.

- Stateful Session Beans sind **keine Transaktionelle Resourcen** Veränderte Instanzvariablen innerhalb von eine Stateful Bean Sessuin werden nicht wiederhergestellt bei Rollback.
- Stateful Session beans mit CMT können das SessionSyncronization Interface implementieren.

3.1. SessionSyncronization Callback Methoden

void afterBegin() Transaktion gerade gestartet.

void beforeCompletion()

- Cached database Updates sollte geschrieben werden.
- Letzte Chance für Rollback.
- Nur aufgerufen falls Transaktion noch nicht für Rollback gesetzt ist.

afterCompletion(boolean commited) Parameter : commited oder rollback? Bean muss möglicherweise seine Zustand manuell zurücksetzen.

3.2. SFSB die SessionSyncronization implementieren

- Falls ein EnterpriseBean diese Interface implementiert können nur die folgende Werte für Transaktionen benutzt werden :

  Required

  Requires_new

  Mandatory
- Diese Einshränkung versichert dass ein Bean immer aufgerufen ist in eine Transaktion sonnst, werden die Callbackmethoden nutzlos.

3.3. Extended Persistence Context

Persistence Context Verwaltet eine Menge von EntityInstanzen. Durch Entitymanager zugreifbar.

```
1 //Applikation verwaltet manager.
   @PersistenceUnit
  EntityManagerFactory emf;
  emf.createEntityManager();

6 //Container verwaltet Manager
  @PersistenceContext
  EntityManager em;
```

Transaction Scoped Entity Manager

```
1  @PersistenceContext(unitName="movieDS",
  type=PersistenceContextType.TRANSACTION)
  EntityManager em;
```

- Entitymanager zustandslos.
- Abhängig von JTA transactions
- PersistenceContext immer mit laufende Transaktion verbunden.

  Wenn eine Operation ausgeführt ist, die PersistenceContext mit dem der Transaktion verbunden ist, wird benutzt.ODER eine neue ist erzeugt und mit Transaktion verbunden.
- Konsequenzen :: Entitymanger muss innerhalb Transaktion zugreifbar sein. Referenz könnte immer das gleiche sein.

SFSB Scoped (Extended Entity Manager)

```
@PersistenceContext(unitName="movieDS",
type=PersistenceContextType.EXTENDED)
EntityManager em;
```

- Persistence Context:

  Erzeugt wenn SFSB Instanziert wird.

  entfernt wenn SFSB entfernt wird.
- Konsequenzen:

  Entities können als "Conversational State" gespreichert werden.

  Veränderungen in Transaktionen sind in der Datenbank gespeichert.

  Keine Probleme mit detached Instanzen (alles ist immer attached)

3.4. Extended Entity Manager Beispiel

```
@Stateful
public class UserManagerImpl implements UserManager {
@PersistenceContext(unitName="movieDS")
// Transaction scoped
EntityManager em;
User user;
public void init(int id) { user = em.find(User.class, id); }
public void rentMovie(int mvoieId) {
rentMovie(user, em.find(Movie.class, movieId));
}
public void setName(String name) {
user.setName(name); // no effect as detached instance is
}
// changed => em.merge is necessary!
@Remove
public void finished() {}
}

@Stateful
public class UserManagerImpl implements UserManager {
@PersistenceContext(unitName="movieDS",
type=PersistenceContextType.EXTENDED)
EntityManager em;
User user;
public void init(int id) { user = em.find(User.class, id); }
public void rentMovie(int mvoieId) {
rentMovie(user, em.find(Movie.class, movieId));
}
public void setName(String name) {
user.setName(name); // saved after commit
}
@Remove
public void finished() {}
}
```

3.5. Fazit

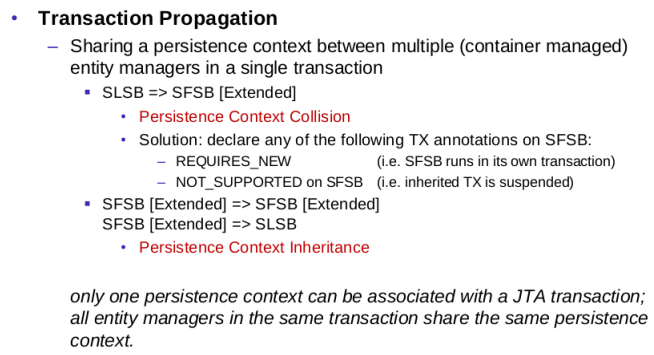

Abbildung 3.1.: figure

Teil II.

Code,Übungen,usw.