15.–18. 09. 2008
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Gerüchteküche

Enterprise Java Beans 3.1

Werner Eberling

MATHEMA Software GmbH

EJB 3.1 – Aktueller Stand

- JSR-318
- Status: Early Draft Review
- Stand: 21. Februar 2008

Neuerungen in EJB 3.1

- SessionBeans
  - Vereinfachte lokale Sicht
  - Singleton Bean
  - Asynchroner Aufruf
- EJB-Timer
  - Automatische Timer
  - Neue Timer-Ausdrücke
- Vereinfachtes Deployment
- Leichtgewichtiges EJB-Profile

SessionBeans: Vereinfachte lokale Sicht

- Bisherige Bestandteile einer SLSB:
  - Remote Business Interface oder/und
    Local Business Interface oder/und
    WebService Endpoint Interface bzw. WebMethods
  - Implementierenden Klasse

- Jetzt: vereinfachte Variante im ***lokalen*** Fall:
  - Implementierende Klasse

SessionBeans: Vereinfachte lokale Sicht

- Bisher:

```
public interface Hello {
   String hello();
}

@Stateless
public class HelloImpl implements Hello {
   public void hello() {
        return "Hello";
   }
}

public class HelloClient{
   @EJB Hello hello;
   public void doIt() {
        System.out.println(hello.hello());
   }
}
```

SessionBeans: Vereinfachte lokale Sicht

- Mit EJB 3.1:

```
@Stateless
public class HelloImpl{
   public String hello() {
        return "Hello";
   }
}

public class HelloClient{
   @EJB HelloImpl hello;
   public void doIt() {
        System.out.println(hello.hello());
   }
}
```

SessionBeans: Vereinfachte lokale Sicht

- Bean-Klasse darf keine Interfaces implementieren
- Alle(!) public-Methoden werden exportiert
- Client sieht den Typ der Bean-Klasse
- ABER: Guard ist noch immer vorhanden!

Session Beans: Singleton Session Bean

- Eine einzige Session Bean-Instanz pro Applikation
  - ABER: im verteilten Fall ein Singleton pro VM !!!!
- Defacto: Stateful Session Bean mit dem Lifecycle einer Stateless Session Bean
- Neue Annotationen
  - @Singleton
  - @Startup
  - @DependsOn

Session Beans: Singleton Session Bean

- Beispiel

```
@Startup
@Singleton
public class SingletonHelloImpl implements Hello {
   public String hello() {
        return "Hello";
   }
}

@Singleton
@DependsOn("SingletonHelloImpl")
public class SingletonGoodbyeImpl implements Goodbye {
   ...
}
```

Session Beans: Singleton Session Bean

- Parallelität
  - Singletons werden aus Client-Sicht immer parallel genutzt
    - => evtl. Synchronisierung notwendig
      - Container-managed-concurrency
      - Bean-managed-concurrency
  - Neue Annotationen
    - @BeanManagedConcurrency
    - @ContainerManagedConcurrency
    - @ReadOnly
    - @ReadWrite

Session Beans: Singleton Session Bean

- Beispiel (CMC)

```
@Startup
@Singleton
public class SingletonHelloImpl implements Hello {
   long counter;

   @ReadOnly
   public String hello() {
        return "Hello (counter: "+ counter +")";
   }

   @ReadWrite
   public String countingHello() {
        counter++;
        return "Hello (counter: "+ counter +")";
   }
}
```

Session Beans: Singleton Session Bean

- Beispiel (BMC)

```
@Startup
@Singleton
public class SingletonHelloImpl implements Hello {
   volatile long counter;

   public String hello() {
        synchronized(this){
             return "Hello (counter: "+ counter +")";
        }
   }

   public String countingHello() {
        synchronized(this){
             counter++
             return "Hello (counter: "+ counter +")";
        }
   }
}
```

Session Beans: Asynchroner Aufruf

- Session Beans können asynchrone Business-Methoden anbieten
  - Mögliche Rückgabetypen
    - `Void`
    - `Future<V>`
- Für den Client nur indirekt unterscheidbar
- Neue Annotationen
  - @Asynchronous

Session Beans: Asynchroner Aufruf

- Beispiel einer asnychr. BeanMethode

```
public interface AsyncHello {
   Future<String> hello();
}

@Stateless
public class AsyncHelloImpl implements AsyncHello {
   @Asynchronous
   public Future<String> hello() {
        // Irgendwas langwieriges... ,)
        String hello = ...
        return new javax.ejb.AsyncResult<String>(hello);
   }
}
```

Session Beans: Asynchroner Aufruf

- Zustellungsgarantien
  - Transaktional
    - Abarbeitung beginnt nach commit der umgebenden TX
  - Persistent
    - Auftrag zur Abarbeitung / Ergebnis überlebt Server-Crash
  - Konfigurierbar über die Annotation @Asynchronous
- Abbruch der Verarbeitung in Nachhinein möglich
  - `Future<V>.cancel(boolean mayInterruptIfRunning)`
  - `SessionContext.isCancelled()`

Session Beans: Asynchroner Aufruf

- Exceptions
  - Exception wird beim Aufruf von `Future<V>.get()` geworfen
    - `java.lang.concurrent.ExecutionException.getCause()`
  - `Void`-Methoden werfen keine Exceptions
- Transaktionen
  - Keine TX-Klammer über die Ausführung asynch. Methoden
    - Vgl. Transaktionen in MOMs

Neuerungen in EJB 3.1

- SessionBeans
  - Vereinfachte lokale Sicht
  - Singleton Bean
  - Asynchroner Aufruf
- EJB-Timer
  - Automatische Timer
  - Neue Timer-Ausdrücke
- Vereinfachtes Deployment
- Leichtgewichtiges EJB-Profile

EJB-Timer: Automatische Timer

- Bisher:
  - Timer mussten explizit registriert und gestartet werden:

```
@Resource
SessionContext ctx

@Timeout
public void prüfeAuftragseingang(Timer timer) {
  ...
}

// Callback alle 60 Sekunden
public void starteTimer() {
 ctx.getTimerService().createTimer(60000, null);
}
```

EJB-Timer: Automatische Timer

- Mit EJB3.1:
  - Container richtet bei Bedarf automatisch einen Timer ein:

```
// An jedem Monatsersten um 3 Uhr
@Schedule(hour="3", dayOfMonth="1")
public void erzeugeAbrechung() {
  ...
}
```

- Neue Annotationen
  - @Schedule
  - @Schedules

EJB-Timer: Automatische Timer

- `javax.ejb.TimedObject` entfällt für automatische Timer
- Die bekannten Timer-Callbacks existieren weiterhin
- Timerausdrücke mit cron-ähnlicher Syntax möglich

EJB-Timer: Neue Timer-Ausdrücke

- Kalender-basierte Timerausdrücke
  - Angeleht an die aus Unix bekannte cron-Syntax:
    - `@Schedule(dayOfWeek="Mon")`
    - `@Schedule(second="0", minute="0", hour="0", dayOfMonth="*", month="*", dayOfWeek="Mon", year="*)`
    - `@Schedule(second="0", minute="0",dayOfWeek="Mon-Fr")`

EJB-Timer: Neue Timer-Ausdrücke

- Explizite Erzeugung von Callbacks über ScheduleExpressions:

```
...
  ScheduleExpression schedule =
     new ScheduleExpression().dayOfWeek("Sat").hour(1);
  Timer timer = timerService.createTimer(schedule, null);
...
```

Neuerungen in EJB 3.1

- SessionBeans
  - Vereinfachte lokale Sicht
  - Singleton Bean
  - Asynchroner Aufruf
- EJB-Timer
  - Automatische Timer
  - Neue Timer-Ausdrücke
- Vereinfachtes Deployment
- Leichtgewichtiges EJB-Profile

Vereinfachtes Deplyoment

- Bisher:
  - Paketierung als EJB-Jar
    - Enthält seit EJB 3.0 (im Minimalfall) nur noch annotierte Java-Klassen
- Ab EJB 3.1:
  - Auch Paketierung innerhalb eines WAR-Files möglich

Vereinfachtes Deplyoment

- Was landet wo?
  - Klassen/Interfaces: WEB-INF/classes
  - Deskriptor: WEB-INF/ejb-jar.xml
- Es gelten die aus dem Webcontainer bekannten Regeln bzgl. Classloading!

Neuerungen in EJB 3.1

- SessionBeans
  - Vereinfachte lokale Sicht
  - Singleton Bean
  - Asynchroner Aufruf
- EJB-Timer
  - Automatische Timer
  - Neue Timer-Ausdrücke
- Vereinfachtes Deployment
- Leichtgewichtiges EJB-Profile

Leichtgewichtiges EJB-Profile

- JEE 6 unterscheidet unterschiedliche Profiles
  - Je nach Szenario unterschiedlicher Funktionsumfang
  - Ermöglicht leichtgewichtigere Container/Server
- EJB 3.1 schafft Unterteilung möglicher Container in
  - Basis-Container (z.B. Im Rahmen eines JEE Web Profiles)
  - JEE Full Profile-Container

Leichtgewichtiges EJB-Profile

- Grundlegende APIs (Basis aller Profile)
  - JDBC
  - RMI-IIOP
  - JNDI
  - JAXP
  - Java IDL
  - JAAS

Leichtgewichtiges EJB-Profile

- APIs innerhalb aller EJB-Container
  - EJB 3.1 (ohne Remote-View und asynchr. SessionBeans!)
  - Java Persistence 2.0
  - JTA 1.1 (nur UserTransaction)
  - Common Annotations 1.0
- APIs innerhalb eines Full-Profile-Containers
  - JMS 1.1
  - JavaMail 1.2
  - JCA 1.5
  - WebService 1.2
  - ...

Leichtgewichtiges EJB-Profile

- Ermöglicht „kleine“ Container
- Quasi „EJB-light”
- ABER: noch immer keine explizite Forderung nach EJB-Betrieb ausserhalb des App'Servers (vgl. JPA)

Neuerungen in EJB 3.1

- SessionBeans
  - Vereinfachte lokale Sicht
  - Singleton Bean
  - Asynchroner Aufruf
- EJB-Timer
  - Automatische Timer
  - Neue Timer-Ausdrücke
- Vereinfachtes Deployment
- Leichtgewichtiges EJB-Profile

Und was ist mit JPA?

- Ist nicht (mehr) Teil der EJB-Spec.
- JSR-317
- Aber irgendwie gehört's ja doch dazu ,) ..

JPA 2.0 - Schlaglichter

- Collections von Embeddables
- Mehrstufige Embeddables
- Orphan Removal
- Explizites Locking
- (2nd Level)-Cache-Interface
- ...

Vielen Dank!

Werner Eberling

MATHEMA Software GmbH