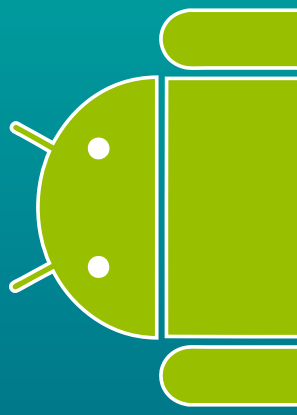
Nr.01 | Frühjahr 2012 | www.ijug.eu

Java aktuell

Das Magazin der Java-Community

Java aktuell

ANDROID in der Praxis



JavaOne 2011
Neuigkeiten und Trends

Oracle Public Cloud
Bereit für Wolke sieben

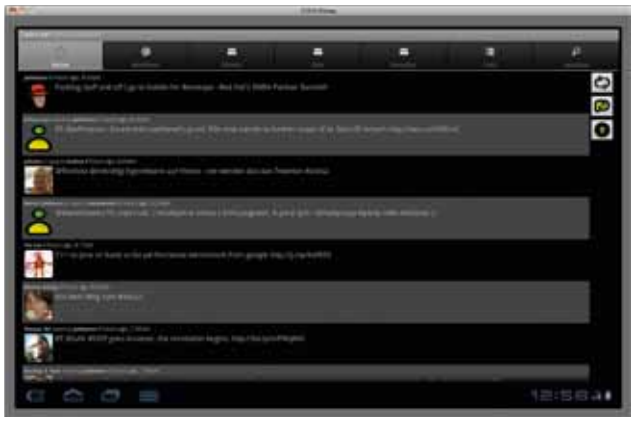
Adobe AIR
Anspruchsvolle Applikationen realisieren

D: 4,90 EUR A: 5,60 EUR CH: 9,80 CHF Benelux: 5,80 EUR ISSN 2191-6977

Sonderdruck

iJUG Verbund

Inhalt



Android-Apps fit für die Zukunft machen, Seite 12

Zusammengesetzte Persistenz-Einheiten

Bernd Müller, Ostfalia – Hochschule für angewandte Wissenschaften, und Harald Wehr, MAN Truck & Bus AG

Die Persistence-Unit ist eines der zentralen Konzepte der Java-Persistence-API. Sie wird in Form der Datei „persistence.xml" verwendet, um dem JPA-Provider notwendige Informationen zur Verwaltung von Entities zur Verfügung zu stellen, etwa die zu verwendende Datenbank inklusive deren Verbindungs- und Authentifizierungsdaten. Seit der Version 2.3 erlaubt der JPA-Provider „EclipseLink" die Verwendung einer sogenannten „Composite-Persistence-Unit". Diese stellt eine logische Persistence-Unit dar, die aus mehreren Teilen, sogenannten „Composite-Member-Persistence-Units" besteht. Die Verwendung erfolgt für den Entwickler zu großen Teilen transparent, geht aber an dieser Stelle auch über den JPA-Standard hinaus.

Eine Instanz des Interface „EntityManagerFactory" repräsentiert zur Laufzeit eine Persistence-Unit. Die Verwendung mehrerer Persistence-Units in einer JPA-Anwendung ist möglich. Diese werden durch mehrere Instanzen des Interface „EntityManagerFactory" unterschieden. Eine Instanz des Interface „EntityManager" repräsentiert einen Persistence-Context, eine Menge von Entity-Instanzen. In Java SE werden „EntityManagerFactory" und „EntityManager" programmatisch erzeugt, in Java EE durch Verwendung der Annotationen „@PersistenceUnit" und „@PersistenceContext" vom Container injiziert.

Durch die Verwendung der EclipseLink-Erweiterung der Composite-Persistence-Units entfällt die explizite, programmatische Verwaltung der EntityManager und EntityManager-Fabriken. Dies übernimmt EclipseLink. Wir entwickeln im Folgenden eine kleine Anwendung zur Verwaltung der Abteilungen einer Firma sowie deren Mitarbeiter und ihrer Adressen, also das typische Department-Employee-Beispiel.

Die Fachlichkeit

Die Fachlichkeit besteht aus den Entities „Mitarbeiter", „Abteilung" und „Adresse", wobei ein Mitarbeiter zu einer Abteilung gehört (n:1) und eine Adresse besitzt (1:1). Wir geben die wichtigen Ausschnitte der Klassen wieder (siehe Listing 1).

Abbildung 1 zeigt das Datenbank-Schema für die drei Entity-Klassen. Wir haben die Zuordnung zu den beiden Composite-Member-Persistence-Units, die im Folgenden eingeführt werden, bereits vorweggenommen.

Composite-Member-Persistence-Unit

Die Composite-Member-Persistence-Units werden wie gewöhnliche Persistence-Units verwendet, um Datenquellen zu definieren. In unserem Beispiel sollen die Klasse „Mitarbeiter" in einer, die Klassen „Abteilung" und „Adresse" in einer anderen Composite-Member-Persistence-Unit enthalten sein. Die erste Persistence-Unit für die Klasse „Mitarbeiter" stellt sich wie folgt dar (siehe Listing 2).

Man erkennt den Namen der Persistence-Unit, hier „employee1", die Verwendung einer H2-Datenbank und das Setzen des Properties „eclipselink.composite-unit.member" auf „true". Damit wird gekennzeichnet, dass diese Persistence-Unit nur in einer Composite-Persistence-Unit und nicht stand-alone verwendet werden kann. Der Grund hierfür liegt in den beiden Assoziationen der Klasse „Mitarbeiter": Die Persistence-Unit kann nur mit den entsprechenden Persistence-Units der Assoziationsziele und nicht allein verwendet werden.

Die zweite Composite-Member-Persistence-Unit „employee2" ist analog aufgebaut und verwendet eine PostgreSQL-Datenbank. Da die beiden Klassen „Abteilung" und „Adresse" keine Assoziationsziele in anderen Persistence-Units besitzen, könnte sie auch stand-alone verwendet werden, sodass auf das entsprechende Property verzichtet werden kann (siehe Listing 3).

```
@Entity
public class Mitarbeiter {
  @Id @GeneratedValue
  private Integer id;
  private String vorname;
  private String nachname;
  @Temporal(TemporalType.DATE)
  private Date geburtsdatum;
  @OneToOne
  private Adresse adresse;
  @ManyToOne
  private Abteilung abteilung;
  ...
}

@Entity
public class Adresse {
  @Id @GeneratedValue
  private Integer id;
  private String strasse;
  private String hausnummer;
  private String plz;
  private String ort;
  ...
}

@Entity
public class Abteilung {
  @Id @GeneratedValue
  private Integer id;
  private String name;
  ...
}
```

Listing 1

Packaging

Beim Erzeugen der Jars wird die erste Composite-Member-Persistence-Unit ent-

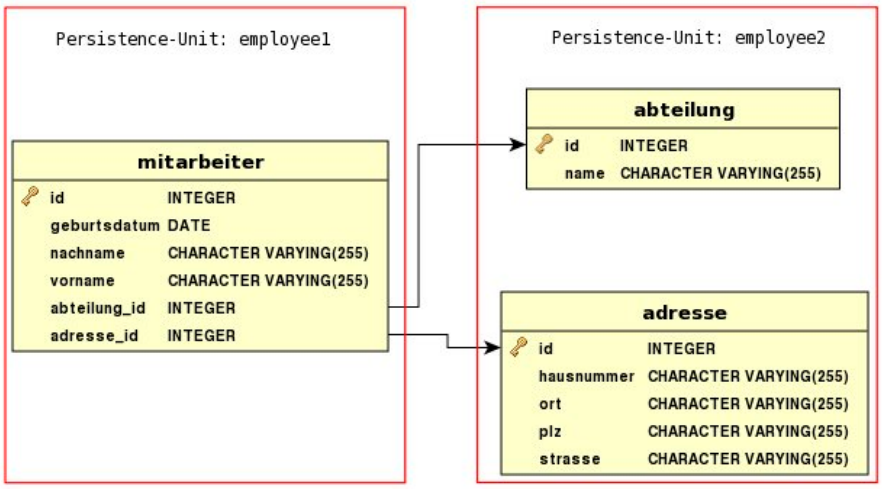


Abbildung 1: Datenbankschema für die Entity-Klassen „Mitarbeiter", „Abteilung" und „Adresse"

```
<persistence ...>
  <persistence-unit name="employee1">
    <exclude-unlisted-classes>false</exclude-unlisted-classes>
    <properties>
      <property name="javax.persistence.jdbc.driver"
              value="org.h2.Driver" />
      <property name="javax.persistence.jdbc.url"
              value="jdbc:h2:tcp://localhost/bank" />
      <property name="javax.persistence.jdbc.user" .../>
      <property name="javax.persistence.jdbc.password" .../>
      <property name="eclipselink.composite-unit.member"
              value="true"/>
    ...
</persistence>
```

Listing 2

```
<persistence ...>
  <persistence-unit name="employee2">
    <exclude-unlisted-classes>false</exclude-unlisted-classes>
    <properties>
      <property name="javax.persistence.jdbc.driver"
              value="org.postgresql.Driver" />
      <property name="javax.persistence.jdbc.url"
              value="jdbc:postgresql://localhost/bank" />
      <property name="javax.persistence.jdbc.user" .../>
      <property name="javax.persistence.jdbc.password" .../>
    ...
</persistence>
```

Listing 3

```
<persistence >
  <persistence-unit name="composite">
    <jar-file>member1.jar</jar-file>
    <jar-file>member2.jar</jar-file>
    <properties>
      <property name="eclipselink.composite-unit"
              value="true" />
    ...
</persistence>
```

Listing 4

```
EntityManagerFactory emf =
    Persistence.createEntityManagerFactory
    („composite“);
```

Listing 5

```
Mitarbeiter mitarbeiter = new Mitarbeiter(...);
Abteilung abteilung = new Abteilung(...);
Adresse adresse = new Adresse(...);
mitarbeiter.setAbteilung(abteilung);
mitarbeiter.setAdresse(adresse);
// em injiziert oder über Fabrik erzeugt:
em.persist(mitarbeiter);
```

Listing 6

sprechend den JPA-Regeln im Verzeichnis „META-INF“ als „persistence.xml“ gepackt. Zusätzlich ist die Anwendungsklasse „Mitarbeiter“ im Jar enthalten. Die zweite Composite-Member-Persistence-Unit wird analog mit den Klassen „Abteilung“ und „Adresse“ in ein weiteres Jar gepackt. Wir verwenden die Namen „member1.jar“ und „member2.jar“. Diese beiden Jars werden in der Composite-Persistence-Unit verwendet. Das Packaging selbst erfolgt sinnvollerweise mit Ant. Eclipse oder Maven können nicht verwendet werden, da das Packaging nicht den Standardregeln entspricht.

Composite-Persistence-Unit

Die Composite-Persistence-Unit referenziert die beiden Jars über das Standard-JPA-Tag „<jar-file>“. Wichtig ist das Property „eclipselink.composite-unit“, das auf „true“ gesetzt werden muss, da es sich um eine zusammengesetzte Persistenzeinheit handelt. Die Composite-Persistence-Unit enthält keine Verbindungsinformationen für eine Datenquelle. Wird eine Datenquelle in einer Composite-Persistence-Unit definiert, so wird dies ignoriert (siehe Listing 4).

Verwendung

Die „EntityManagerFactory“ wird nun, wie in JPA üblich, über die entsprechende Create-Methode erzeugt (siehe Listing 5).

Beim Persistieren von Entities kann völlig transparent mit den drei Entity-Klassen gearbeitet werden. Wenn bei den Annotationen „@ManyToOne“ und „@OneToOne“ der „CascadeType“ entsprechend gesetzt wird, werden mit dem folgenden Code sowohl der Mitarbeiter als auch seine Abteilung und seine Adresse persistiert (siehe Listing 6). Der Mitarbeiter landet in der H2-Datenbank, Abteilung und Adresse in der PostgreSQL-Datenbank. Beim Lesen eines Mitarbeiters mit „em.find()“ werden die beiden Assoziationen korrekt initialisiert, es findet also ein Lesen in beiden Datenbanken statt.

Beschränkungen

Die Verteilung von Entities auf verschiedene Persistenzeinheiten schränkt die Möglichkeiten, die der JPA-Standard bietet, in bestimmten Bereichen stark ein:

- Queries mit Entities verschiedener Composite-Member-Persistence-Units sind nicht erlaubt
- Vererbungshierarchien können nicht auf mehrere Composite-Member-Persistence-Units aufgeteilt werden
- Bei der Verwendung von „@JoinTable“ für 1:n- und n:m-Beziehungen muss die Join-Tabelle in derselben Persistence-Unit wie das Beziehungsziel sein und die Verwendung von „mappedBy“ ist nicht zulässig. Daher können derartige Beziehungen nicht bidirektional sein.

Es gibt noch eine Reihe weiterer Einschränkungen, die wir aber nicht im Einzelnen aufführen. Wir verweisen den Leser auf das EclipseLink-Wiki.

Fazit

Die genannten Beschränkungen reduzieren den allgemeinen Einsatz von Composite-Persistence-Units erheblich, da grundlegende JPA-Mechanismen nicht mehr verwendet werden können. Wir sind dennoch der Meinung, dass sich der Einsatz lohnen kann.

In großen Anwendungen ergibt sich in der Regel ganz automatisch eine Partitionierung von Entity-Klassen, sodass keine Assoziationen und keine gemeinsame Vererbungshierarchie über Partitionsgrenzen hinweg existieren. Die genannten Beschränkungen treffen in diesem Fall dann nicht zu.

Außerdem kann es aus Effizienz- und Kostengründen sinnvoll sein, sich häufig ändernde Daten, wie normale betriebliche Anwendungsdaten, etwa in Oracle zu halten, während sehr viel umfangreichere, binäre Daten, wie etwa monatliche Abrechnungen, in PDF-Form in dafür geeigneteren Datenbanken abgelegt werden können. Ein weiterer Anwendungsfall wäre, die Ablage von datenschutzrelevanten Informationen in besonders geschützten Security-Zonen beziehungsweise Datenbanken durchzuführen und strikt von weniger sensiblen Daten zu trennen. Auch die Zusammenführung von Informationen aus verschiedenen existierenden Datenbanken beispielsweise für Auswertungen ist mit der EclipseLink-Erweiterung problemlos möglich. Daten aus unterschiedlichen Unternehmensbereichen oder Datenbanken, die fachlich Bezug zueinander haben, können so mit einem übergreifenden Business-Modell schnell zusammengeführt werden. Effiziente SQL-Befehle und deren JPA-Entsprechung bleiben jedoch außen vor, da man mit verschiedenen Datenbanken arbeitet.

Weiterführende Informationen

[1] http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Advanced_JPA_Development/Composite_Persistence_Units

[2] http://wiki.eclipse.org/EclipseLink/Examples/JPA/Composite

Bernd Müller
bernd.mueller@ostfalia.de

Harald Wehr
harald.wehr@gmail.com

Bernd Müller ist seit März 2005 Professor für Software-Technik an der Fakultät Informatik der Hochschule Braunschweig/Wolfenbüttel, nachdem er sieben Jahre Professor für Wirtschaftsinformatik an der Hochschule Harz war.

Harald Wehr ist seit 2005 bei der MAN Gruppe in Salzgitter als Java-Entwickler beschäftigt. Seit 2008 befasst er sich verstärkt mit der SAP HR Anwendungsentwicklung und Beratung. Zusammen mit Bernd Müller verfasste er das Buch „Java-Persistence-API mit Hibernate“. Die Autoren arbeiten derzeit an einer Neuauflage des Buches, das im kommenden Frühjahr beim Hanser-Verlag erscheinen wird.

www.ijug.eu

JETZT ABO BESTELLEN

Sichern Sie sich 4 Ausgaben für 18 EUR

Für Oracle-Anwender und Interessierte gibt es das Java aktuell Abonnement auch mit zusätzlich sechs Ausgaben im Jahr der Fachzeitschrift *DOAG News* und vier Ausgaben im Jahr *Business News* zusammen für 75 EUR. Weitere Informationen unter www.doag.org/shop/

FAXEN SIE DAS AUSGEFÜLLTE FORMULAR AN

0700 11 36 24 39

ODER BESTELLEN SIE ONLINE

go.ijug.eu/go/abo

Interessenverbund der Java User Groups e.V.
Tempelhofer Weg 64
12347 Berlin

Java aktuell

+++ AUSFÜLLEN +++ AUSSCHNEIDEN +++ ABSCHICKEN +++ AUSFÜLLEN +++ AUSSCHNEIDEN +++ ABSCHICKEN +++ AUSFÜLLEN

☐ **Ja**, ich bestelle das Abo Java aktuell – das IJUG-Magazin: 4 Ausgaben zu 18 EUR/Jahr

☐ **Ja**, ich bestelle den kostenfreien Newsletter: Java aktuell – der iJUG-Newsletter

ANSCHRIFT

Name, Vorname

Firma

Abteilung

Straße, Hausnummer

PLZ, Ort

GGF. RECHNUNGSANSCHRIFT

Straße, Hausnummer

PLZ, Ort

E-Mail

Telefonnummer

X

Die allgemeinen Geschäftsbedingungen* erkenne ich an, Datum, Unterschrift

*Allgemeine Geschäftsbedingungen:

Zum Preis von 18 Euro (inkl. MwSt.) pro Kalenderjahr erhalten Sie vier Ausgaben der Zeitschrift "Java aktuell - das iJUG-Magazin" direkt nach Erscheinen per Post zugeschickt. Die Abonnementgebühr wird jeweils im Januar für ein Jahr fällig. Sie erhalten eine entsprechende Rechung. Abonnementverträge, die während eines Jahres beginnen, werden mit 4,90 Euro (inkl. MwSt.) je volles Quartal berechnet. Das Abonnement verlängert sich automatisch um ein weiteres Jahr, wenn es nicht bis zum 31. Oktober eines Jahres schriftlich gekündigt wird. Die Wiederrufsfrist beträgt 14 Tage ab Vertragserklärung in Textform ohne Angabe von Gründen.