

Objektrelationale Abbildung mit dem Java Persistence API (JPA)

Einführung

Objektrelationale Abbildung (ORM):

Technik der Softwareentwicklung, mit der Objekte eines in einer objektorientierten Programmiersprache geschriebenes Anwendungsprogramm in einer relationalen Datenbank abgelegt (persistiert) werden können.

- standardisierte Schnittstelle für Java ist die Java Persistence API
- **Ziel:** Überführung der Daten aus RDBS in Objekte und umgekehrt
- **Impedance Mismatch**
Unvollständige Abbildung zwischen Relationenmodell und OO Modell aufgrund unterschiedlicher Paradigmen
- verschiedene JPA-Implementierungen: z.B. EclipseLink, Hibernate
- **Kernidee:** alle Eigenschaften die persistiert werden sollen, werden mit Annotationen im Quellcode gekennzeichnet

JPA: Annotationen

Annotationen auf unterschiedlicher Ebene: Klasse, Attribut

Klassenannotation:

@Entity: Klasse wird als Datenobjekt von JPA persistiert

@Table: Mapping auf Tabellenebene (Klassenname ↔ Tabellenname)

@AccessType: Zugriffstyp für Id-Attribute Property oder Field

Attributannotationen:

@Id: kennzeichnet den Primärschlüssel

@Column: Zuordnung Member-Variable ↔ Tabellenattribut

@GeneratedValue: automatisch generierter Wert

@SequenceGenerator: erzeugt Sequenzwert für @GeneratedValue Annotation

Beispiel - Klassendefinition

Klasse zum Zugriff auf Daten der Tabelle Artikel

```
@Entity
@Table(name="Artikel")
public class Artikel implements Serializable
{
    private String anr;
    private String bezeichnung;

    public Artikel() { }

    @Id
    public String getAnr(){
        return this.anr;
    }

    public String getBezeichnung(){
        return this.bezeichnung;
    }

    public void setAnr(String anr_p){
        this.anr=anr_p;
    }

    public void setBezeichnung(String bezeichnung_p){
        this.bezeichnung=bezeichnung_p;
    }
    ...
}
```

Annotation @Id

- Identifikator notwendig zur eindeutigen Referenzierung der Entitäten
- Spezifikation des Primärschlüssels

Strategien zur Generierung von Id-Werten:

- IDENTITY: verwendet die DB-Funktionalität einer selbstinkrementierenden Spalte (AUTO_INCREMENT)
- SEQUENCE: verwendet die DB-Funktionalität einer Sequence, Default-Sequence oder DB-Sequence mittels @SequenceGenerator
- TABLE: Generierung der Id-Werte mittels Table-Generator, Default-Generator oder spezielle Tabelle mittels @TableGenerator
- AUTO: überlässt der JPA-Implementierung die Wahl des Verfahrens (SEQUENCE oder IDENTITY), Default-Einstellung
- Keine gleichzeitige Verwendung von Triggern zum Setzen von Id-Werten

Beispiel:

```
@Id
@SequenceGenerator( name="X_GENERATOR",
                    sequenceName="X_SEQ")
@GeneratedValue(   strategy=GenerationType.SEQUENCE,
                    generator="X_GENERATOR")
```

Annotation @Access.Type: Field

- @Id/@Column-Annotation beim Attribut
- Zugriff direkt auf das Attribut

```
@Entity
@Table(name="Lieferant")
@Access(AccessType.FIELD)
public class Lieferant
{
    @Id
    private String name;

    public String getName()
    {
        return this.name;
    }

    public void setName(String name_p)
    {
        this.name = name_p;
    }
    . . .
}
```

Annotation @Access.Type: Property

- @Id-Annotation bei Getter/Setter
- Zugriff über Methoden

```
@Entity
@Table(name="Artikel")
@Access(AccessType.PROPERTY)
public class Artikel
{
    private String bezeichnung;

    @Id
    public String getBezeichnung()
    {
        return this.bezeichnung;
    }

    public void setBezeichnung(String bezeichnung_p)
    {
        this.name = bezeichnung_p;
    }
    ...
}
```

Zusammengesetzte Schlüssel - Beispiel

- Realisierung z.B. über Annotation `@IdClass`

Beispiel: Tabelle Lieferung mit Spalten (ANr, LNr, Preis) und Primary Key (ANr, LNr) als Entity-Klasse

```
public class LieferungId implements Serializable
{
    private String anr;                //Schlüsselklasse
    private String lnr;                //LieferungId

    public LieferungId(){ }
    public LieferungId(String anr, String lnr){ }
    ...
    public String getANr() { }
    public String getLNr() { }
}

@Entity
@IdClass(LieferungId.Class)
public class Lieferung                //Lieferung mit
{                                       //IdClass-Annotation
    @Id private String anr;
    @Id private String lnr;
    private BigDecimal preis;
    public Lieferung(){ }
    public Lieferung(String anr, String lnr, BigDecimal preis){ }
    ...
}
```


Persistence Unit

- zentraler Bestandteil der XML-Konfigurationsdatei **persistence.xml**
- enthält globale Einstellungen für das Mapping:
 - Angabe des JPA-Providers
 - Auflistung der zu persistierenden Entity-Klassen (die durch den Entity-Manager verwaltet werden)
 - Spezifikation der Datenbankverbindung :
 - JDBC-Driver (DBMS-abhängig)
 - Parameter für den DB-Zugriff (Driver, URL, User, Password)
- mehrere Persistence Units möglich

Persistence Unit - Beispiel

persistence.xml in EclipseLink:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
  xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
    http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
  <persistence-unit name="project_name">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <class>pkg.Class</class>
    <properties>
      <property name="javax.persistence.jdbc.driver"
        value="oracle.jdbc.OracleDriver" />
      <property name="javax.persistence.jdbc.url"
        value="jdbc:oracle:thin:@server:port:SID" />
      <property name="javax.persistence.jdbc.user"
        value="username" />
      <property name="javax.persistence.jdbc.password"
        value="password" />
    </properties>
  </persistence-unit>
</persistence>
```

Entity Management

- Zentrale Schnittstelle: EntityManager (verwaltet von Java-Anwendung)
- Steuerung der DB-Operationen
- Schnittstelle zur Transaktionsverwaltung
- Klasse: javax.persistence.EntityManager
- je angebundener Datenquelle eigenständige EntityManager-Instanz
- Unterscheidung mit Hilfe der [Persistence Units](#) - Referenz zur Konfiguration der DB-Verbindung

```
EntityManagerFactory emf =  
    Persistence.createEntityManagerFactory("project_name");  
EntityManager em = emf.createEntityManager();  
...  
em.close();  
emf.close();
```

Transaktionsverwaltung

- Manipulationen an den an verwalteten Objektinstanzen müssen von einer JPA-Transaktionen eingehüllt werden
- Expliziter Beginn notwendig
- Expliziter Abschluss (commit, rollback)

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("handel");
EntityManager em = emf.createEntityManager();

EntityManager tx = em.getTransaction();
tx.begin();

...

tx.commit();

em.close();
emf.close();
```

Anlegen von Entitäten

Vorgehensweise:

1. Beginn der Transaktion
2. Aufbau des Objekts (inkl. Setzen aller relevanten Attributwerte)
3. Übergabe des Objekts an EntityManager
4. Abschluss der Transaktion

```
em.getTransaction().begin();
Artikel a = new Artikel("101","SCSI Kabel");
em.persist(a);                                //Einfügen in den Objekt-Cache
em.getTransaction().commit();
```

Vorzeitiges Zurückschreiben in die Datenbank (Beispiel):

```
em.getTransaction().begin();
em.persist(a1);
em.persist(a2);
em.flush();                                //Synchronisation mit der Datenbank
em.persist(a3);
em.getTransaction().commit();
```

Daten-Retrieval von Entitäten

Unterschiedliche Arten der Daten-Abfrage:

- **ID-basiertes Retrieval** unter Nutzung des Entity-Manager
- Daten-Retrieval unter Nutzung der **Java Persistence Query Language** (JPQL)
- **Criteria API** zur objektorientierten Formulierung von Anfragen

Daten-Retrieval: ID-basiert

- Laden von Entitäten anhand ihres Primärschlüssels
- ID: singulares Attribut (@Id), das entweder einen primitiven Datentyp besitzt oder mit Annotation @IdClass assoziiert ist (compound key)
- find()-Methode des EntityManagers
- Mapping automatisch über Attribute mit @Id- Annotation
- Rückgabe: lokalisierte Entity oder NULL

Beispiel:

```
Artikel a = em.find(Artikel.class,101);  
System.out.println(a);
```

Nachteile:

- Entität nur über vollständigen Primärschlüssel identifizierbar
- pro find()-Aufruf nur maximal ein Datensatz selektierbar

Daten-Retrieval: JPQL

- **Java Persistence Query Language**
- SQL-ähnliche Anfragesprache für Daten-Retrieval in JPA-basierten Applikationen
 - SELECT .. FROM .. WHERE, Gruppierung und Funktionen analog zu SQL
- **Umsetzung:**
 - Query-Generator: JPQL-Anfrage -> SQL-Anfrage
 - Entity-Erzeugung: Erstellung eines Objekts je Ergebnis-Tupels im DBS
- **Erzeugung einer Query-Instanz:**
 - `createQuery()`
 - `createNamedQuery()` - mit einem Namen referenzierbare Anfrage
 - `createNativeQuery()` - Einsatz von SQL-Anfragen statt JPQL
- **Ergebnis:**
 - Liste: `getResultList()`
 - Entity: `getSingleResult()`
- **Parameter:**
 - NamedParameter: angesprochen mit Namen, definiert mit Doppelpunkt
 - PositionalParameter: angesprochen über ihre Position, definiert mit Fragezeichen

JPQL: Beispiele

Beispiel: Einfacher Lesezugriff

```
List<Artikel> al = em.createQuery("SELECT a FROM Artikel a",  
                                Artikel.class).getResultList();
```

Beispiel: Anfrage zum Auslesen der Artikel mit einer bestimmten Bezeichnung

```
List<Artikel> al = em.createQuery("SELECT a FROM Artikel a WHERE a.bezeichnung  
                                LIKE :bezeich", Artikel.class)  
                    .setParameter("bezeich", "Kabel").getResultList();
```

Beispiel: Gleiche Anfrage als Named Query

```
@NamedQuery(name="findAllArtikelByBezeich",  
            query= "SELECT a FROM Artikel a WHERE a.bezeichnung LIKE ?1")
```

Setzen des Parameters 1 (?1) auf den Wert "Kabel"

```
List<Artikel> al = em.createNamedQuery("findAllArtikelByBezeich")  
                    .setParameter(1, "Kabel").getResultList();
```

Aktualisieren und Löschen von Entitäten

Aktualisierung von Entitäten

- Veränderung der relevanten Attributwerte mittels Setter
- Gleichzeitige Änderung des Objekts im Objekt-Cache (Referenz)
- Keine separate Methode notwendig
- Persistierung der Änderung bei Abschluss der Transaktion

Löschen von Entitäten

- Löschen des Objekts aus Objekt-Cache über Entity-Manager
- Persistierung der Löschung bei Abschluss der Transaktion

Beispiele:

```
em.getTransaction().begin();
Artikel a = em.find(Artikel.class, anr);
a.setBezeichnung("SCSI Cable");
em.getTransaction().commit();
```

Update

```
em.getTransaction().begin();
Artikel a = em.find(Artikel.class, anr);
em.remove(a);
em.getTransaction().commit();
```

Löschen

Beziehungen zwischen Entities

- Spezifikation der Beziehungen über Annotationen in Datenobjekten
- Voraussetzung: Definition von Primärschlüsseln
- Unterschiedliche Arten von Beziehungen entsprechend ihrer Kardinalität
 - 1:1: **@OneToOne**
 - 1:N: **@OneToMany, @ManyToOne**
 - M:N: **@ManyToMany** (mit Spezifikation der notwendigen Beziehungstabelle)
- Richtung der Verbindung:
 - Unidirektionale Verbindungen
(Verbindung nur von einer beteiligten Entität deklariert)
 - Bidirektionale Verbindungen
(Verbindung von beiden beteiligten Entitäten deklariert)

Beispiel: Annotationen ManyToOne



Unidirektional:



```
@ManyToOne  
@JoinColumn(name="fk_in_b")  
private A a;
```

Bidirektional:



```
@OneToMany(mappedBy="a")  
private Set<B> bList;
```

```
@ManyToOne  
@JoinColumn(name="fk_in_b")  
private A a;
```

Beispiel: bidirektionale Beziehung

Jeder Artikel ist in mehreren Lieferungen enthalten. Eine Lieferung enthält genau einen Artikel.



```
@Entity
public class Artikel
{
    @Id private String anr;
    private String bezeichnung;

    @OneToMany(mappedBy = "a")
    private List<Lieferung>lieferungen;

    // Getter und Setter
}
```

```
@Entity
@IdClass(LieferungId.Class)
public class Lieferung
{
    @Id private String anr;
    @Id private String lnr;
    private BigDecimal preis;

    @ManyToOne
    @JoinColumn(name = "anr")
    private Artikel a;

    // Getter und Setter
}
```

Beispiel: Arbeiten mit Beziehungen

Kennzeichnung der Beziehung zu den Lieferungen in der Entity-Klasse Artikel

- Automatisches Löschen aller Lieferungen beim Löschen des Artikels
(cascade=ALL)
- Korrespondierendes Datenobjekt
(mappedBy="a")

```
@OneToMany(cascade=ALL, mappedBy="a")
```

- Get-Methode für Lieferungen

```
public Collection<Lieferung> getLieferungen()  
{  
    return lieferungen;  
}
```

- Anlegen einer neuen Lieferung zu einem Artikel

```
Lieferung li = new Lieferung(anr,lnr,preis);  
a.getLieferungen().add(li);  
em.persist(li);
```

JPA-Tutorial

<http://www.java2s.com/Tutorials/Java/JPA/index.htm>