Object-relational Mapping (ORM)

Petr Aubrecht

ČVUT, katedra kybernetiky

3. 4. 2008

Obsah – update 2008

1. JPA
   - Základy
   - Dotazy
   - Vztahy 1:N
   - CRUD
2. Semestrální práce

Co bude letos jinak

- Vloni se CRUD dělal v jedné třídě
  - jednodušší na pochopení
  - nepříliš vhodný design pattern
  - správně je datová vrstva objektů s daným interfacem, poměrně pracné

Co bude letos jinak

- Vloni se CRUD dělal v jedné třídě
  - jednodušší na pochopení
  - nepříliš vhodný design pattern
  - správně je datová vrstva objektů s daným interfacem, poměrně pracné
- Letos budeme používat JPA
  - NetBeans vygenerují objekty z databáze, bude potřeba drobně doupravit.
  - Většina práce je tak hotova, nemusíte programovat nezáživný CRUD.
  - Programování se může soustředit na práci s objekty a základní práci s databází nechat na JPA.

Jak začít s JPA v NetBeans

- Vygenerujte si databázovou aplikaci a prozkoumejte vygenerované zdrojové kódy.

Jak začít s JPA v NetBeans

- Vygenerujte si databázovou aplikaci a prozkoumejte vygenerované zdrojové kódy.
- META-INF/persistence.xml – definice připojení do databáze
- Entity – definice objektů a mapování do databáze
- Jak dostat z databáze objekty, jak je uložit atd...

Definice připojení do databáze

Example (META-INF/persistence.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns=... xsi:schemaLocation=...>
  <persistence-unit name="misdemoPU" transaction-type="RESO
    <provider>oracle.toplink.essentials.PersistenceProvider
    <class>desktopapplication1.model.Faktura</class>
    <class>desktopapplication1.model.Zakaznik</class>
    ...
    <class>desktopapplication1.model.Sms</class>
    <properties>
      <property name="toplink.jdbc.user" value="misdemo"/>
      <property name="toplink.jdbc.password" value="data"/>
      <property name="toplink.jdbc.url" value="jdbc:postgre
      <property name="toplink.jdbc.driver" value="org.postg
    </properties>
  </persistence-unit>
</persistence>
```

Vygenerovaná entita

Example (Entita)

```
@Entity
@Table(name = "sms")
@NamedQueries({@NamedQuery(name = "Sms.findByIdSms",
  query = "SELECT s FROM Sms s WHERE s.idSms = :idSms")})
public class Sms implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @Column(name = "id_sms", nullable = false)
    private Integer idSms;
    @Column(name = "obsah", nullable = false)
    private String obsah;
    @Column(name = "cislo_od", nullable = false)
    private int cisloOd;
    ...
```

Úprava vygenerovaných entit

- Takto vygenerované entity budou fungovat.
- Nebude však možné využít automatického generování primárních klíčů, které je velmi pohodlné.
- Proto použijeme anotaci @GeneratedValue. TopLink nebere ohled na pojmenování sekvence, proto je potřeba mít ji dobře pojmenovanou – nejsnáze tak, jak ji vygeneruje CREATE TABLE (int ID serial primary key,...)
- Dotazy se v JPA píší v jazyku EJBQL, který je podobný SQL. Pro naše účely stačí intuitivní chápání vygenerovaných dotazů.

Defaultní hodnoty primárního klíče

Example (@GeneratedValue)

```
@Entity
@Table(name = "zakaznik")
@NamedQueries({@NamedQuery(
   name = "Zakaznik.findByIdZakaznik",
   query = "SELECT z FROM Zakaznik z "
           " WHERE z.idZakaznik = :idZakaznik")})
public class Zakaznik implements Serializable {
    //private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY) //!!!
    @Column(name = "id_zakaznik", nullable = false)
    private Integer idZakaznik;
    @Column(name = "jmeno", nullable = false)
    private String jmeno;
      ...
```

Přístup do databáze

Example (Přístup do databáze)

```
javax.persistence.EntityManager entityManager;
javax.persistence.Query query;

entityManager = javax.persistence.Persistence.
    createEntityManagerFactory("misdemoPU").createEntityMana
query = entityManager.createQuery(
    "SELECT z FROM Zakaznik z");
nebo query = entityManager.
    createNamedQuery("Zakaznik.findByIdZakaznik");
nebo query = entityManager.createNativeQuery("");

query.getResultList();
nebo
query.getSingleResult();
nebo
query.executeUpdate();
```

Dotazy

Example (Parametry dotazů)

```
javax.persistence.EntityManager entityManager;
javax.persistence.Query query;

query = entityManager.createQuery(
    "SELECT z FROM Zakaznik z "
    "WHERE z.idZakaznik = :idZakaznik");
query.setParameter("idZakaznik",1234);
Zakaznik z = (Zakaznik)query.getSingleResult();

nebo jednoduseji

Zakaznik z = entityManager.createQuery(
    "SELECT z FROM Zakaznik z "
    "WHERE z.idZakaznik = :idZakaznik"
    ).setParameter("idZakaznik",1234).getSingleResult();
```

Jaký typ dotazů zvolit?

- NamedQuery: uvedení dotazů v definici třídy je dobrý zvyk.
- Jednoduché ad hoc dotazy pomocí EJBQL (Query).
- Složité dotazy (GROUP BY, HAVING ap.) je potřeba psát pomocí NativeQuery.
- Pozor na getSingleResult(), selže v případě více výsledků.

Vztahy mezi entitními typy

Example (Relace 1:N)

```
public class Zakaznik implements Serializable {
    ...
    @OneToMany(cascade = CascadeType.ALL,
     mappedBy = "idZakaznik")
    private Collection<Faktura> fakturaCollection;
    @OneToMany(mappedBy = "idZakaznik")
    private Collection<Telefon> telefonCollection;

public class Faktura implements Serializable {
    ...
    @JoinColumn(name = "id_zakaznik",
     referencedColumnName = "id_zakaznik")
    @ManyToOne
    private Zakaznik idZakaznik;
```

Přechod mezi objekty

Example (Zákazník → Telefon → Hovory)

```
void fillWithCalls(Zakaznik zakaznik) {
  clientsNameLabel.setText(zakaznik.getJmeno());
  hovorList.clear();
  for(Telefon tlf : zakaznik.getTelefonCollection()) {
    for(Hovor hovor : tlf.getHovorCollection()) {
      hovorList.add(hovor);
    }
  }
}
```

Transakce

Example

```
entityManager.getTransaction().begin();
... neco se deje ...
entityManager.getTransaction().commit();
v pripade chyby:
entityManager.getTransaction().rollback();

Vygenerovany kod:
na zacatku:
entityManager.getTransaction().begin();

na save operaci:
if(entityManager.getTransaction().isActive())
    entityManager.getTransaction().commit();
entityManager.getTransaction().begin();
```

Pro připomenutí – CRUD

Example (CRUD)

Create

```
Zakaznik z = new Zakaznik();
entityManager.persist(z);
```

Retrieve

viz queries, vetsinou prochazenim objektu

Update

```
entityManager.persist(z);
```

Delete

```
entityManager.remove(z);
```

Nezapomenout potvrdit transakci!

Co se očekává od semestrálky

- Že se něco naučíte!
  - pokud bude JPA příliš těžké, zůstaňte u JDBC
  - snaha byla usnadnit CRUD, neočekáváme hluboké zkoumání EJBQL

Co se očekává od semestrálky

- Že se něco naučíte!
  - pokud bude JPA příliš těžké, zůstaňte u JDBC
  - snaha byla usnadnit CRUD, neočekáváme hluboké zkoumání EJBQL
- Proč je ve vygenerovaném projektu všechno jinak, než jsme se dosud o SWINGu učili?
  - Protože je postaven na jdesktop.org, které udává tón vývoje desktopových aplikací (poskytuje služby navíc oproti SWINGu, viz background tasky, animace ve status baru, akce definované anotacemi ap.)
  - Netrváme na použití jdesktop, můžete pracovat v čistém SWINGu.
  - http://community.java.net/javadesktop/, https://swingx.dev.java.net/, https://swinglabs.dev.java.net/,...