Object-relational Mapping (ORM)

Petr Aubrecht

ČVUT, katedra kybernetiky

26. 4. 2007

Obsah

Objekty a databáze

- Typické úložiště dat je relační databáze.
  - entity, relace, transakce
  - návrh modelu je ER diagram

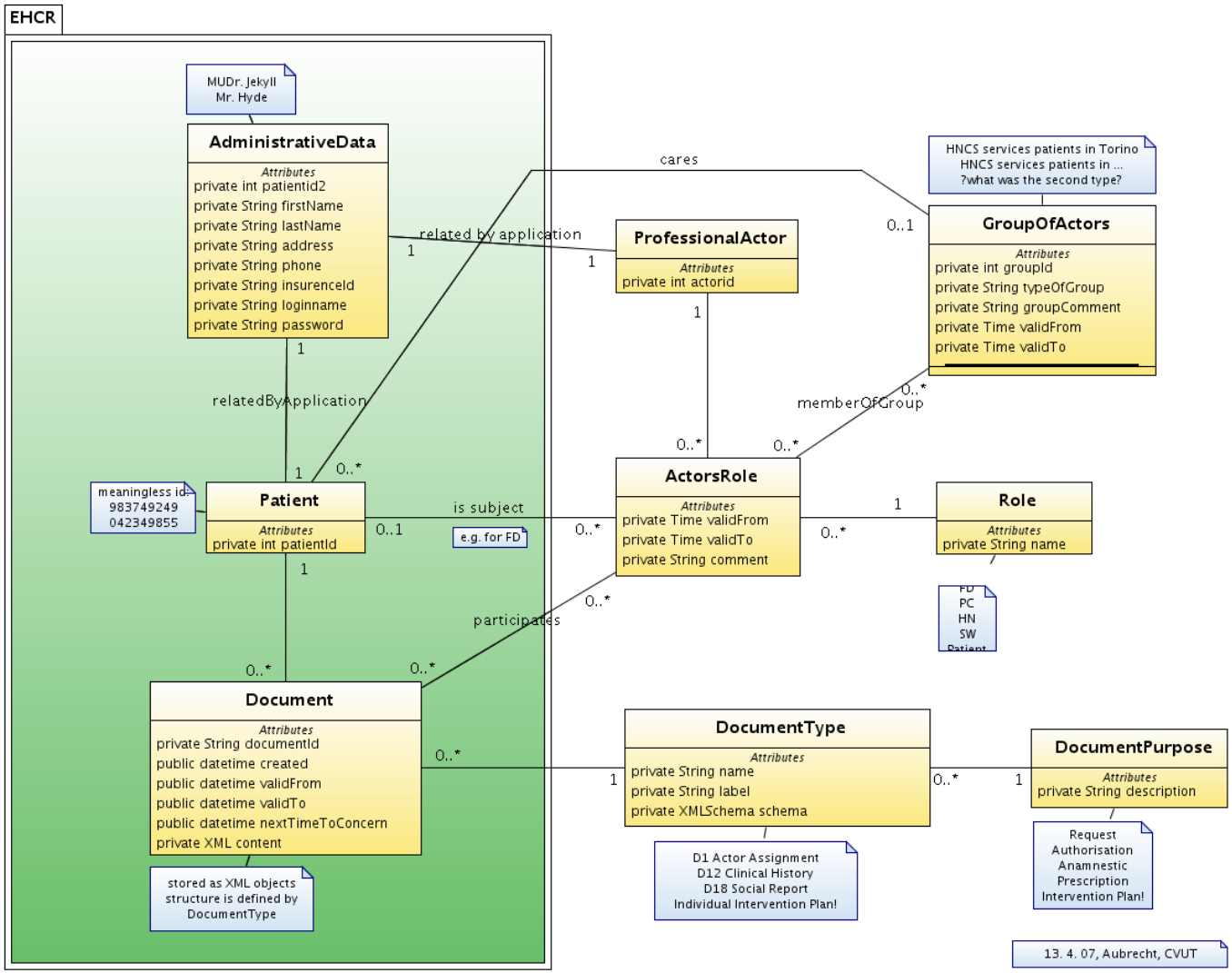
Objekty a databáze

- Typické úložiště dat je relační databáze.
  - entity, relace, transakce
  - návrh modelu je ER diagram
- Typický model software je objektový (OOP).
  - třídy, dědění, instance, ukazatele
  - návrh modelu je class diagram (UML)
  - viz design patterns, MVC atd.

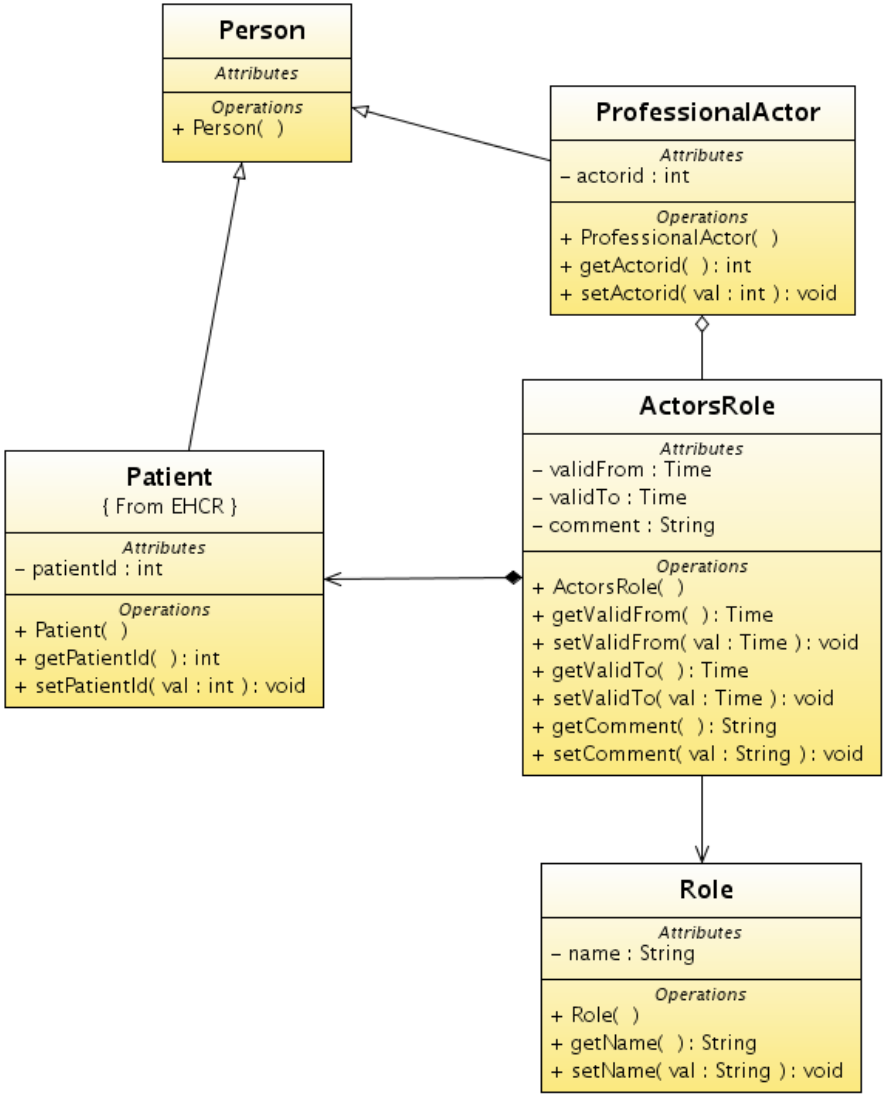
Objekty a databáze

- Typické úložiště dat je relační databáze.
  - entity, relace, transakce
  - návrh modelu je ER diagram
- Typický model software je objektový (OOP).
  - třídy, dědění, instance, ukazatele
  - návrh modelu je class diagram (UML)
  - viz design patterns, MVC atd.
- Entity odpovídají třídám, relace ukazatelům. Existuje přirozené spojení?

Modely – ER Diagram



Modely – Class Diagram



Jak ukládat objekty?

- přímé ukládání objektů

Jak ukládat objekty?

- přímé ukládání objektů
  - serializace (bytestream, XML)

Jak ukládat objekty?

- přímé ukládání objektů
  - serializace (bytestream, XML)
  - objektová databáze

Jak ukládat objekty?

- přímé ukládání objektů
  - serializace (bytestream, XML)
  - objektová databáze – znáte nějakou?

Jak ukládat objekty?

- přímé ukládání objektů
  - serializace (bytestream, XML)
  - objektová databáze – znáte nějakou?
- relační databáze

Jak ukládat objekty?

- přímé ukládání objektů
  - serializace (bytestream, XML)
  - objektová databáze – znáte nějakou?
- relační databáze
  - přímé používání JDBC

Jak ukládat objekty?

- přímé ukládání objektů
  - serializace (bytestream, XML)
  - objektová databáze – znáte nějakou?
- relační databáze
  - přímé používání JDBC
  - ORM – object-relational mapping

Jak ukládat objekty?

- přímé ukládání objektů
  - serializace (bytestream, XML)
  - objektová databáze – znáte nějakou?
- relační databáze
  - přímé používání JDBC
  - ORM – object-relational mapping
- XML databáze
  - moderní v poslední době
  - kromě XML fanatiků je nikdo nebere vážně, výkon na úrovni parsování textového souboru, vyhledávání pomocí XPath, funguje pouze fulltext search (pro strukturovane dokumenty)

Serializace

- triviálně jednoduché uložení a načtení
  - public class Abc implements Serializable

Example (Serializace)

```
FileOutputStream fos = new FileOutputStream("out.tmp");
ObjectOutputStream oos = new ObjectOutputStream(fos);

oos.writeInt(12345);
oos.writeObject("Today");
oos.writeObject(new Date());

oos.close();
```

Serializace – vlastnosti

- triviálně jednoduché uložení a načtení

Serializace – vlastnosti

- triviálně jednoduché uložení a načtení
- nelze se dotazovat, načítat pouze některé instance

Serializace – vlastnosti

- triviálně jednoduché uložení a načtení
- nelze se dotazovat, načítat pouze některé instance
- nevhodné pro dlouhodobé ukládání (problém s novou verzí objektů)

Serializace – vlastnosti

- triviálně jednoduché uložení a načtení
- nelze se dotazovat, načítat pouze některé instance
- nevhodné pro dlouhodobé ukládání (problém s novou verzí objektů)
- použití
  - krátkodobé uložení
  - komunikace po síti, objektové zprávy (Java RMI, CORBA, JMS)
  - (un)marshaling, změna objektů kvůli rozdílům v architekturách (malý a velký indián)

Objektové databáze

Objektové databáze

- objektové databáze
  - proč: problém rozdělení složitých objektů (normalizace, dekompozice), sql neumí rekurzivní dotazy
  - v 90. letech s nástupem OO jazyků (C++) se hovořilo o konci relačních databází

Objektové databáze

- objektové databáze
  - proč: problém rozdělení složitých objektů (normalizace, dekompozice), sql neumí rekurzivní dotazy
  - v 90. letech s nástupem OO jazyků (C++) se hovořilo o konci relačních databází
- dnešní stav
  - výkon daleko za relačními databázemi
  - neexistence všeobecně přijímaného modelu, dotazovacího jazyka, transakcí, atd.
  - dnes se prakticky nepoužívají ve prospěch relačních DB

Přístup k databázi pomocí JDBC

- Původně procedurální programování, který využíval přímý přístup k databázi. Tento způsob znáte z dřívějších přednášek a ze cvičení.

Přístup k databázi pomocí JDBC

- Původně procedurální programování, který využíval přímý přístup k databázi. Tento způsob znáte z dřívějších přednášek a ze cvičení.

Example (JDBC)

```
Class.forName("org.postgresql.Driver");
Connection conn = DriverManager
    .getConnection("jdbc:postgresql:mis0715",
    "mis0715","pwd");
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery("SELECT * FROM doc");
while(rs.next()) {
    System.out.println(rs.getString("docid"));
}
conn.close();
```

Co všechno je potřeba pro práci s jedinou entitou?

Co všechno je potřeba pro práci s jedinou entitou?

- definovat třídu

Co všechno je potřeba pro práci s jedinou entitou?

- definovat třídu
- vytvořit tabulku CREATE TABLE doc (docid serial primary key...)

Co všechno je potřeba pro práci s jedinou entitou?

- definovat třídu
- vytvořit tabulku CREATE TABLE doc (docid serial primary key...)
- CRUD

Co všechno je potřeba pro práci s jedinou entitou?

- definovat třídu
- vytvořit tabulku CREATE TABLE doc (docid serial primary key...)
- CRUD
  - Create – INSERT INTO doc (created, type) VALUES (now(),'prescr')

Example (JDBC – Create)

```
ResultSet idRs = conn.createStatement().executeQuery(
  "SELECT nextval('doc_docid_seq')");
idRs.next();
int newId = rdRs.getInt(1); setDocId(newId);
PreparedStatement st = conn.prepareStatement(
  "INSERT INTO doc (docid,created,type) VALUES (?,?,?)
st.setInt(1,newId);
st.setDatetime(2,new java.sql.Date(System.currentTime
st.setString(3,"prescr"); // or typeEdit.getText()
st.execute();
```

Co všechno je potřeba pro práci s jedinou entitou?

- definovat třídu
- vytvořit tabulku CREATE TABLE doc (docid serial primary key...)
- CRUD
  - Create – INSERT INTO doc (created, type) VALUES (now(),'prescr')
  - Retrieve – SELECT * FROM doc WHERE docid=57

Example (JDBC – Retreive)

```
PreparedStatement st = conn.prepareStatement(
  "SELECT * FROM doc WHERE docid=?");
rs.setInt(1,getDocId());
rs.executeQuery();
rs.next();
setNextTime(rs.getInt("nexttime"));
setType(rs.getString("type"));
```

Co všechno je potřeba pro práci s jedinou entitou?

- definovat třídu
- vytvořit tabulku CREATE TABLE doc (docid serial primary key...)
- CRUD
  - Create – INSERT INTO doc (created, type) VALUES (now(),'prescr')
  - Retrieve – SELECT * FROM doc WHERE docid=57
  - Update – UPDATE doc SET nexttime=now()+365 WHERE docid=75

Example (JDBC – Update)

```
PreparedStatement st = conn.prepareStatement(
  "UPDATE doc SET nexttime=?,type=? WHERE docid=?");
rs.setInt(1,getNextTime());
rs.setInt(2,getType());
rs.setInt(3,getDocId());
rs.execute();
```

Co všechno je potřeba pro práci s jedinou entitou?

- definovat třídu
- vytvořit tabulku CREATE TABLE doc (docid serial primary key...)
- CRUD
  - Create – INSERT INTO doc (created, type) VALUES (now(),'prescr')
  - Retrieve – SELECT * FROM doc WHERE docid=57
  - Update – UPDATE doc SET nexttime=now()+365 WHERE docid=75
  - Delete – DELETE doc WHERE docid=75

Example (JDBC – Update)

```
PreparedStatement st = conn.prepareStatement(
  "DELETE doc WHERE docid=?");
rs.setInt(1,getDocId());
rs.execute();
```

Co všechno je potřeba pro práci s jedinou entitou?

- definovat třídu
- vytvořit tabulku CREATE TABLE doc (docid serial primary key...)
- CRUD
  - Create – INSERT INTO doc (created, type) VALUES (now(),'prescr')
  - Retrieve – SELECT * FROM doc WHERE docid=57
  - Update – UPDATE doc SET nexttime=now()+365 WHERE docid=75
  - Delete – DELETE doc WHERE docid=75
- ...toto pro několik desítek entit

Co všechno je potřeba pro práci s jedinou entitou?

- definovat třídu
- vytvořit tabulku CREATE TABLE doc (docid serial primary key...)
- CRUD
  - Create – INSERT INTO doc (created, type) VALUES (now(),'prescr')
  - Retrieve – SELECT * FROM doc WHERE docid=57
  - Update – UPDATE doc SET nexttime=now()+365 WHERE docid=75
  - Delete – DELETE doc WHERE docid=75
- ...toto pro několik desítek entit
- ...včetně změn během vývoje

Object Relational Mapping

- Pro velké projekty je přímá práce s JDBC příliš náročná, nelze zaručit funkčnost při změně schématu (kromě změny class diagramu a tříd je potřeba změnit všechna místa, kde se s databází pracuje.

Object Relational Mapping

- Pro velké projekty je přímá práce s JDBC příliš náročná, nelze zaručit funkčnost při změně schématu (kromě změny class diagramu a tříd je potřeba změnit všechna místa, kde se s databází pracuje.
- Vzniká logický požadavek na automatické ukládání objektů do databáze: 1 objekt = 1 záznam v tabulce, třída = entita

Object Relational Mapping

- Pro velké projekty je přímá práce s JDBC příliš náročná, nelze zaručit funkčnost při změně schématu (kromě změny class diagramu a tříd je potřeba změnit všechna místa, kde se s databází pracuje.
- Vzniká logický požadavek na automatické ukládání objektů do databáze: 1 objekt = 1 záznam v tabulce, třída = entita
- potřeba vyřešit mapování relací 1:1, 1:N, N:1, M:N jako ukazatele a pole ukazatelů

Object Relational Mapping

- Pro velké projekty je přímá práce s JDBC příliš náročná, nelze zaručit funkčnost při změně schématu (kromě změny class diagramu a tříd je potřeba změnit všechna místa, kde se s databází pracuje.
- Vzniká logický požadavek na automatické ukládání objektů do databáze: 1 objekt = 1 záznam v tabulce, třída = entita
- potřeba vyřešit mapování relací 1:1, 1:N, N:1, M:N jako ukazatele a pole ukazatelů
- ruční implementace je příliš složitá, lze ji nechat na dostupné knihovně

Object Relational Mapping

- Pro velké projekty je přímá práce s JDBC příliš náročná, nelze zaručit funkčnost při změně schématu (kromě změny class diagramu a tříd je potřeba změnit všechna místa, kde se s databází pracuje.
- Vzniká logický požadavek na automatické ukládání objektů do databáze: 1 objekt = 1 záznam v tabulce, třída = entita
- potřeba vyřešit mapování relací 1:1, 1:N, N:1, M:N jako ukazatele a pole ukazatelů
- ruční implementace je příliš složitá, lze ji nechat na dostupné knihovně
- knihovny přistupují k properties bud přímo (přes data members), nebo pomocí set/get

Existující knihovny pro ORM

- J2EE 1.4, EJB (Enterprise Java Beans) 2.1
  - velmi komplikované (jedna entita odpovídala několika třídám a XML souborům)
  - obecné úložiště
  - řeší například distribuované transakce nad několika oddělenými zdroji dat
  - řeší load balancing (přesun EJB mezi servery v závislosti na zatížení)
  - nebudeme se jím zabývat pro extrémní složitost

Existující knihovny pro ORM

- J2EE 1.4, EJB (Enterprise Java Beans) 2.1
  - velmi komplikované (jedna entita odpovídala několika třídám a XML souborům)
  - obecné úložiště
  - řeší například distribuované transakce nad několika oddělenými zdroji dat
  - řeší load balancing (přesun EJB mezi servery v závislosti na zatížení)
  - nebudeme se jím zabývat pro extrémní složitost
- Hibernate, iBatis
  - jednodušší a přímo navržené pro relační databáze
  - těší se velké oblibě v komunitě vývojářů

Existující knihovny pro ORM

- J2EE 1.4, EJB (Enterprise Java Beans) 2.1
  - velmi komplikované (jedna entita odpovídala několika třídám a XML souborům)
  - obecné úložiště
  - řeší například distribuované transakce nad několika oddělenými zdroji dat
  - řeší load balancing (přesun EJB mezi servery v závislosti na zatížení)
  - nebudeme se jím zabývat pro extrémní složitost
- Hibernate, iBatis
  - jednodušší a přímo navržené pro relační databáze
  - těší se velké oblibě v komunitě vývojářů
- JPA (Java Persistence API)
  - vymyšlen původně pro EJB 3.0
  - pouze pro relační databáze
  - při zjednodušování EJB 2.1 $\rightarrow$ EJB 3.0 se JPA osamostatnilo jako knihovna, takže nyní lze použít i v J2SE (např. OpenJPA)

Existující knihovny pro ORM

- J2EE 1.4, EJB (Enterprise Java Beans) 2.1
  - velmi komplikované (jedna entita odpovídala několika třídám a XML souborům)
  - obecné úložiště
  - řeší například distribuované transakce nad několika oddělenými zdroji dat
  - řeší load balancing (přesun EJB mezi servery v závislosti na zatížení)
  - nebudeme se jím zabývat pro extrémní složitost
- Hibernate, iBatis
  - jednodušší a přímo navržené pro relační databáze
  - těší se velké oblibě v komunitě vývojářů
- JPA (Java Persistence API)
  - vymyšlen původně pro EJB 3.0
  - pouze pro relační databáze
  - při zjednodušování EJB 2.1 $\rightarrow$ EJB 3.0 se JPA osamostatnilo jako knihovna, takže nyní lze použít i v J2SE (např. OpenJPA)
- typicky existuje podpora ze strany IDE, případně nástroje dané knihovny pro automatické generování tříd/konfigurace

Hibernate

- Pracuje s POJO (Plain Old Java Object).

Hibernate

- Pracuje s POJO (Plain Old Java Object).
- Mapování je dáno XML souborem.

Hibernate

- Pracuje s POJO (Plain Old Java Object).
- Mapování je dáno XML souborem.
- Celková konfigurace je v hlavním XML, které popisuje připojení do databáze a seznam souborů s mapováním.

Hibernate

- Pracuje s POJO (Plain Old Java Object).
- Mapování je dáno XML souborem.
- Celková konfigurace je v hlavním XML, které popisuje připojení do databáze a seznam souborů s mapováním.

Example (hibernate.cfg.xml)

```
<session-factory>
 <property name="connection.url">jdbc:postgresql://lo
 <property name="connection.username">k4care</propert
 <property name="connection.driver_class">org.postgre
 <property name="dialect">org.hibernate.dialect.Postg
 <property name="connection.password">XXXXXXXXXXX</p
 <property name="transaction.factory_class">org.hiber
 <property name="current_session_context_class">threa
 <!-- this will show us all sql statements -->
 <property name="hibernate.show_sql">true</property>
```

Hibernate – řídící soubor

Example (hibernate.cfg.xml (cont.))

```
...
    <!-- mapping files -->
    <mapping resource="net/k4care/storage/model/
hibernate/DocumentHibernate.hbm.xml" />
    <mapping resource="net/k4care/storage/model/
hibernate/AdministrativeDataHibernate.hbm.xml" />
    <mapping resource="net/k4care/storage/model/
hibernate/PatientHibernate.hbm.xml" />
    <mapping resource="net/k4care/storage/model/
hibernate/ProfessionalActorHibernate.hbm.xml" />
  </session-factory>
</hibernate-configuration>
```

Hibernate – Java bean

Example (DocumentHibernate.java)

```
public class DocumentHibernate implements Document {
    private int documentId;
    private String documentType;
    private Calendar created;
    private Calendar validFrom;
    private Calendar validTo;
    private Calendar nextTimeToConcern;
    private String content;
    private PatientHibernate subject;
    private Set<ActorsRole> ActorRoles;
```

Hibernate – mapovací soubor

Example (DocumentHibernate.hbm.xml)

```
<class name="DocumentHibernate" table="document">
  <id name="documentId" column="documentid">
    <generator class="sequence">
      <param name="sequence">document_documentid_seq
    </generator>
  </id>

  <property name="documentType" column="documenttype
  <property name="created" column="created"/>
  <property name="validFrom" column="validfrom"/>
  <property name="validTo" column="validto"/>
  <property name="nextTimeToConcern" column="nexttim
  <property name="content" column="content"/>
...
```

Hibernate – mapovací soubor

Example (DocumentHibernate.hbm.xml)

```
    <many-to-one name="subject"
                 column="patientid"
                 not-null="false"
        />
      <set name="ActorRoles" table="participates">
        <key column="documentid"/>
        <many-to-many column="actorroleid"
          class="ActorRoleHibernate"/>
      </set>
    </class>
</hibernate-mapping>
```

Hibernate – konfigurace

Example (inicializace Hibernate)

```
// Hibernate helper classes
private Configuration cfg = null;
private SessionFactory factory = null;
private Session session = null;
private Transaction tx = null;

public K4CareStorage() {
    cfg = new Configuration();
    cfg.configure(CONFIG_FILE_LOCATION);
    factory = cfg.buildSessionFactory();
    session = factory.openSession();
    session.setFlushMode(FlushMode.AUTO);
```

Hibernate – CRUD

Example (Document **C**RUD)

```
public Document createDocument(String documentType) {
    Transaction txLocal = session.beginTransaction();
    DocumentHibernate object = new DocumentHibernate();
    object.setDocumentType(documentType);
    session.save(object);
    txLocal.commit();
    return object;
}
```

Hibernate – CRUD

Example (Document CRUD)

```
public Document getDocument(int id) {
  DocumentHibernate actor = (DocumentHibernate)session
  return actor;
}

public void saveDocument(Document actor) {
  session.save(actor);
  session.refresh(actor);
}

public void deleteDocument(Document actor) {
  session.delete(actor);
}
```

Hibernate – shrnutí

- Skvělé! Proč nepoužíváme Hibernate?

Hibernate – shrnutí

- Skvělé! Proč nepoužíváme Hibernate?
  - protože nejdříve musíte rozumět základům, tedy JDBC
  - stačí, že teď se učíte používat Javu, NetBeans a PostgreSQL
  - bohužel neudržuje v relaci 1:N obě strany, je potřeba udělat refresh()

Hibernate – shrnutí

- Skvělé! Proč nepoužíváme Hibernate?
  - protože nejdříve musíte rozumět základům, tedy JDBC
  - stačí, že teď se učíte používat Javu, NetBeans a PostgreSQL
  - bohužel neudržuje v relaci 1:N obě strany, je potřeba udělat refresh()
- Výhody

Hibernate – shrnutí

- Skvělé! Proč nepoužíváme Hibernate?
  - protože nejdříve musíte rozumět základům, tedy JDBC
  - stačí, že teď se učíte používat Javu, NetBeans a PostgreSQL
  - bohužel neudržuje v relaci 1:N obě strany, je potřeba udělat refresh()
- Výhody
  - celkem jednoduchá práce s objekty

Hibernate – shrnutí

- Skvělé! Proč nepoužíváme Hibernate?
  - protože nejdříve musíte rozumět základům, tedy JDBC
  - stačí, že teď se učíte používat Javu, NetBeans a PostgreSQL
  - bohužel neudržuje v relaci 1:N obě strany, je potřeba udělat refresh()
- Výhody
  - celkem jednoduchá práce s objekty
  - dovolí změnou konfigurace spustit na jiném SQL serveru

Hibernate – shrnutí

- Skvělé! Proč nepoužíváme Hibernate?
  - protože nejdříve musíte rozumět základům, tedy JDBC
  - stačí, že teď se učíte používat Javu, NetBeans a PostgreSQL
  - bohužel neudržuje v relaci 1:N obě strany, je potřeba udělat refresh()
- Výhody
  - celkem jednoduchá práce s objekty
  - dovolí změnou konfigurace spustit na jiném SQL serveru
  - nezávislé na okolí, je to knihovna

EJB 3.0, resp. JPA

- důraz je kladen na jednoduchost

EJB 3.0, resp. JPA

- důraz je kladen na jednoduchost
- využití Annotation (od Javy 5)

EJB 3.0, resp. JPA

- důraz je kladen na jednoduchost
- využití Annotation (od Javy 5)
- smysluplné defaultní hodnoty (anotují se pouze odchylky)

EJB 3.0, resp. JPA

- důraz je kladen na jednoduchost
- využití Annotation (od Javy 5)
- smysluplné defaultní hodnoty (anotují se pouze odchylky)

Example (Document EJB 3.0)

```
@Entity
public class Document implements Serializable {
    @Id
    private Integer documentid;

    private Date created;
    private Date validfrom;
    private Date validto;
    private Date nexttimetoconcern;
    private String content;
```

EJB 3.0 – relace N:1 a 1:N

Example (Document EJB 3.0 – relace)

```
...
   @JoinColumn(name = "documenttype",
      referencedColumnName = "typename")
   @ManyToOne
   private Documenttype documenttype;

   @OneToMany(cascade = CascadeType.ALL,
      mappedBy = "documentid")
   private Collection<Participates>
      participatesCollection;
```

Document EJB 3.0 plná specifikace

- položky lze detailně popsat, pokud se liší od defaultních hodnot

Example (Document EJB 3.0 Full)

```
@Entity
@Table(name = "document")
public class Document implements Serializable {

    @Id
    @Column(name = "documentid", nullable = false)
    private Integer documentid;

    @Column(name = "created")
    @Temporal(TemporalType.TIMESTAMP)
    private Date created;

    @Column(name = "validfrom")
```

EJB – business logic

- V EJB je logika aplikace v session nebo message-driven beans, tedy nikoliv v entity beans.

EJB – business logic

- V EJB je logika aplikace v session nebo message-driven beans, tedy nikoliv v entity beans.
- Přístup do databáze je řízen nastavením serveru, tedy je mimo vlastní aplikaci.

EJB – business logic

- V EJB je logika aplikace v session nebo message-driven beans, tedy nikoliv v entity beans.
- Přístup do databáze je řízen nastavením serveru, tedy je mimo vlastní aplikaci.
- Přístup do databáze si nevyžádá beana, je jí nastaven (IoC, Inversion of Control, Injection)

Document EJB 3.0 session bean

Example (Stateless session bean ukládá nový Document do db)

```
@Stateless(mappedName="ejb/TestSessionBean")
public class TestSessionBean implements TestSessionRem
  @PersistenceContext
  private EntityManager em;

  public TestSessionBean() {
  }

  public void doAction() {
    Document doc = new Document();
    em.persist(doc);
  }
```

EJB 3.0 CRUD

- Create: doc = new Document(); em.persist(doc);
- Retrieve: doc = em.find(Document.class, 12345); (12345 je prim. klíč)
- Update: em.persist(doc);
- Delete: em.remove(doc);

Document EJB 3.0 session bean

Example (Stateless session bean přistupuje přímo do db)

```
@Stateless(mappedName="ejb/TestSessionBean")
public class TestSessionBean implements net.k4care.ses
  @Resource(mappedName="jdbc/k4caredb")
  DataSource dataSource;

  public TestSessionBean() {
  }

  public void doAction() {
    try {
      Logger.getLogger(getClass().getCanonicalName()).
      Connection con = dataSource.getConnection();
      PreparedStatement ps = con.prepareStatement("ins
      ps.setDate(1,new java.sql.Date(System.currentTim
```

EJB – shrnutí

- Skvělé! Proč nepoužíváme EJB?

EJB – shrnutí

- Skvělé! Proč nepoužíváme EJB?
  - složitá technologie, záleží na nastavení, problémy tudíž nelze krokovat
  - vyžaduje server, práva na deploy aplikace
  - vyžaduje znalost JNDI
  - nejlépe postupovat podle knihy (JDBC se lze naučit z JavaDoc)

EJB – shrnutí

- Skvělé! Proč nepoužíváme EJB?
  - složitá technologie, záleží na nastavení, problémy tudíž nelze krokovat
  - vyžaduje server, práva na deploy aplikace
  - vyžaduje znalost JNDI
  - nejlépe postupovat podle knihy (JDBC se lze naučit z JavaDoc)
- Výhody

EJB – shrnutí

- Skvělé! Proč nepoužíváme EJB?
  - složitá technologie, záleží na nastavení, problémy tudíž nelze krokovat
  - vyžaduje server, práva na deploy aplikace
  - vyžaduje znalost JNDI
  - nejlépe postupovat podle knihy (JDBC se lze naučit z JavaDoc)
- Výhody
  - jednoduchá práce (po zaškolení)

EJB – shrnutí

- Skvělé! Proč nepoužíváme EJB?
  - složitá technologie, záleží na nastavení, problémy tudíž nelze krokovat
  - vyžaduje server, práva na deploy aplikace
  - vyžaduje znalost JNDI
  - nejlépe postupovat podle knihy (JDBC se lze naučit z JavaDoc)
- Výhody
  - jednoduchá práce (po zaškolení)
  - automaticky se stará o transakce (při vstupu a výstupu z metody session bean)

EJB – shrnutí

- Skvělé! Proč nepoužíváme EJB?
  - složitá technologie, záleží na nastavení, problémy tudíž nelze krokovat
  - vyžaduje server, práva na deploy aplikace
  - vyžaduje znalost JNDI
  - nejlépe postupovat podle knihy (JDBC se lze naučit z JavaDoc)
- Výhody
  - jednoduchá práce (po zaškolení)
  - automaticky se stará o transakce (při vstupu a výstupu z metody session bean)
  - dříve zmíněné výhody – load balancing, distribuované transakce, promyšlená integrace a další...

Závěr

- Data se zásadně ukládají do relační databáze.

Závěr

- Data se zásadně ukládají do relační databáze.
- Pokud je aplikace malá, stačí přístup pomocí JDBC.

Závěr

- Data se zásadně ukládají do relační databáze.
- Pokud je aplikace malá, stačí přístup pomocí JDBC.
- Pro desítky entit je potřeba použít automatický generátor tříd a popisek.

Závěr

- Data se zásadně ukládají do relační databáze.
- Pokud je aplikace malá, stačí přístup pomocí JDBC.
- Pro desítky entit je potřeba použít automatický generátor tříd a popisek.
- Po uvolnění JPA osobně doporučuji spíše JPA, než Hibernate.

Závěr

- Data se zásadně ukládají do relační databáze.
- Pokud je aplikace malá, stačí přístup pomocí JDBC.
- Pro desítky entit je potřeba použít automatický generátor tříd a popisek.
- Po uvolnění JPA osobně doporučuji spíše JPA, než Hibernate.
- Hibernate poskytuje JPA rozhraní! :-) (příp. Apache OpenJPA)

Závěr

- Data se zásadně ukládají do relační databáze.
- Pokud je aplikace malá, stačí přístup pomocí JDBC.
- Pro desítky entit je potřeba použít automatický generátor tříd a popisek.
- Po uvolnění JPA osobně doporučuji spíše JPA, než Hibernate.
- Hibernate poskytuje JPA rozhraní! :-) (příp. Apache OpenJPA)
- Pro opravdu velké aplikace J2EE, EJB 3.0.