

Manuál pro Spring-based DAO vrstvu (webkiv-db v2.01) aplikací webu Katedry informatiky a výpočetní techniky

Radek Muzika, březen 2008

úpravy a doplňky Přemek Brada

Úvodem

Jádro aplikace je postaveno nad Spring Frameworkem (dále jen SF), ze kterého jsou využity funkce IOC kontejneru a pro přístup do databáze je využita knihovna Spring JDBC.

IOC kontejner – při použití SF v aplikaci jsou veškeré základní objekty, potřebné pro chod aplikace (dao třídy, řídicí třídy,...) odkazovány z IOC kontejneru a jsou nazývány beans. Bean je objekt instanciován, sestaven a řízen IOC kontejnerem. Beans a závislosti mezi nimi jsou definovány pomocí konfiguračních metadat. Konfigurační metadata mohou být buď ve formě xml dokumentu, či od javy 1.5 i jako anotace (v xml je uvedeno, že se konfigurace načte z anotací). V redbsys backendu je využito konfigurace pomocí anotací.

Spring JDBC – usnadňuje práci s databází a odstiňuje programátora od tak nebezpečných úkonů, jako je zavírání všech statements a connections.

Více informací o SF 2.5: <http://static.springframework.org/spring/docs/2.5.x/reference/index.html>

Obsah knihoven

cz.zcu.kiv.db.model – obsahuje doménový model

cz.zcu.kiv.db.dao – obsahuje rozhraní pro DAO třídy (ve verzi 2.01 obsahuje implementace tříd pouze pro některé tabulky datového modelu)

cz.zcu.kiv.db.dao.jdbc – obsahuje implementaci dao rozhraní za použití Spring JDBC knihovny

Implementační DAO třídy jsou vyčleněny mimo do samostatného balíku (dao.jdbc), jelikož je možné provést implementaci i pro jiné způsoby přístupu do databáze, jako např. Hibernate (doménový model je navržen tak, aby vyhovoval i případné implementaci ORM). Tento balík dále obsahuje některé třídy přetěžující doménové objekty. Tyto třídy jsou nezbytné pro implementaci Spring JDBC nad doménovým modelem. Obsahují atributy odpovídající cizím klíčům v databázi a dále jsou zde přetížené některé metody, které zajišťují načtení dodatečných informací svázaných s doménovým objektem z databáze. V implementaci metod je použit „lazy-load“, tj. data jsou načtena z databáze, až když jsou zapotřebí.

Jak použít webkiv-db backend

1) do aplikace přidáme knihovny

- webkiv-db – knihovny redbsys backendu
- spring – základní knihovna SF
- log4j – vyzadovano springem
- mysql-connector – knihovna zprostředkovávající připojení do db

2) nakonfigurujeme web.xml (zadáme cestu ke spring konfiguračnímu souboru)

```
<context-param>
```

```
  <param-name>contextConfigLocation</param-name>
```

```
  <param-value>/WEB-INF/redsys-servlet.xml</param-value>
```

```

</context-param>
<listener>
<listener-class>
    org.springframework.web.context.ContextLoaderListener
</listener-class>
</listener>

```

*Tento krok je svázaný s metodou získání aplikačního kontextu ze servlet kontextu.

3) upravit parametry pro připojení v jdbc.properties

```

jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/redsys
jdbc.username=root
jdbc.password=

```

Použití v JSP

Abychom mohli přistupovat k beans, potřebujeme získat aplikační kontext. Pokud jsme v konfiguraci provedli krok 2, pak získáme kontext ze servlet kontextu zadáním

```

org.springframework.context.ApplicationContext ctx =
WebApplicationContextUtils.getWebApplicationContext(config.getServletContext());

```

Pokud jsme neprovedli krok 2, pak lze získat kontext ještě dvěma způsoby a to buď zadáním cesty ke konfiguračnímu souboru, nebo umístěním konf.souboru do classpath. Pak by se použilo metod

```

org.springframework.context.ApplicationContext ctx = new
org.springframework.context.support.FileSystemXmlApplicationContext("redsys-
servlet.xml");

org.springframework.context.ApplicationContext ctx = new
org.springframework.context.support.ClassPathXmlApplicationContext("redsys-servlet.xml");

```

Bean z kontextu získáme metodou

```
UdalostDao uDao = (UdalostDao) ctx.getBean("udalostDao");
```

Získáním dao objektu můžeme přistupovat pomocí metod objektu do nakonfigurované databáze.

```

List<Udalost> seznamUdalosti = uDao.getUdalosti();
pageContext.setAttribute("seznamUdalosti", seznamUdalosti);

```

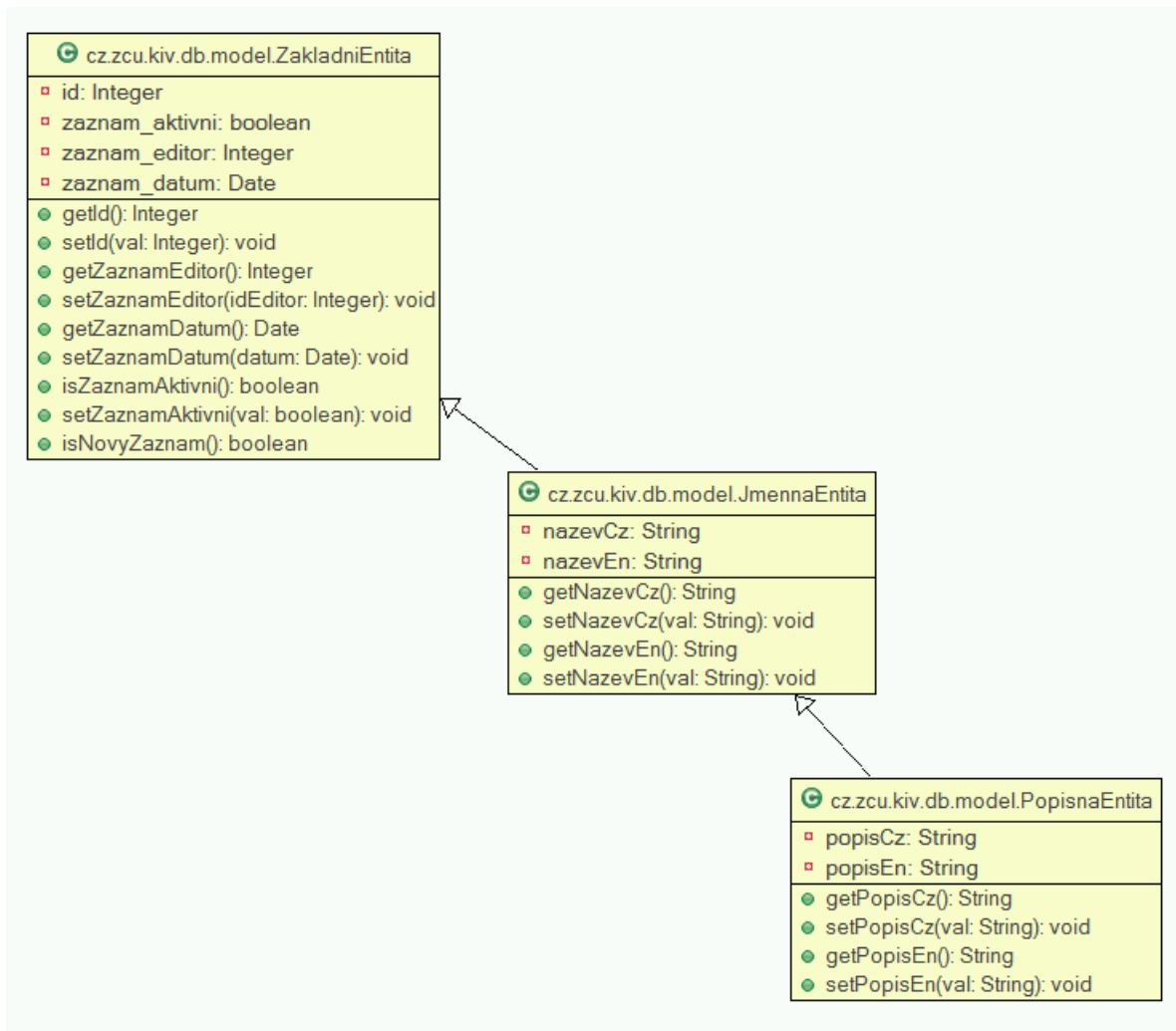
A výsledky zobrazíme pomocí jstl.

```

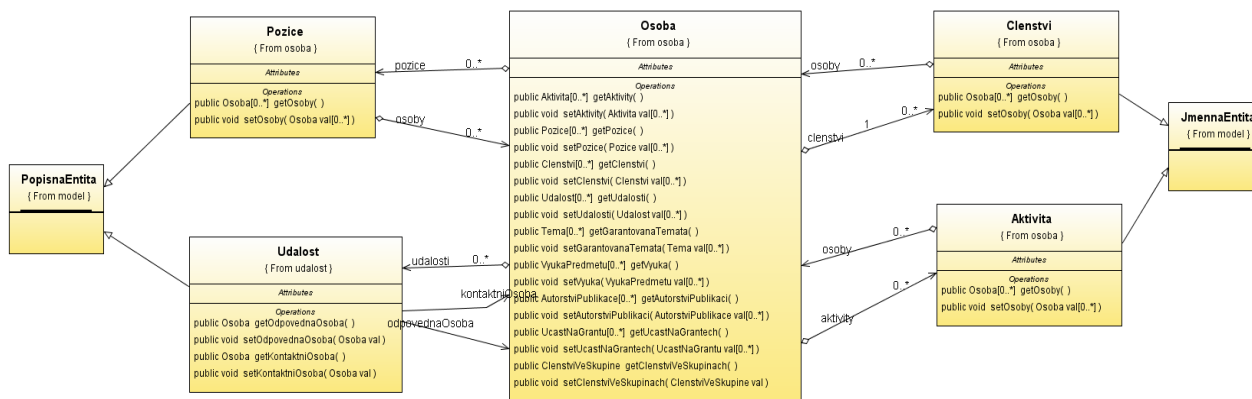
<c:forEach items="{seznamUdalosti}" var="udalost">
    <c:out value="{udalost.nazevCz}" />
</c:forEach>

```

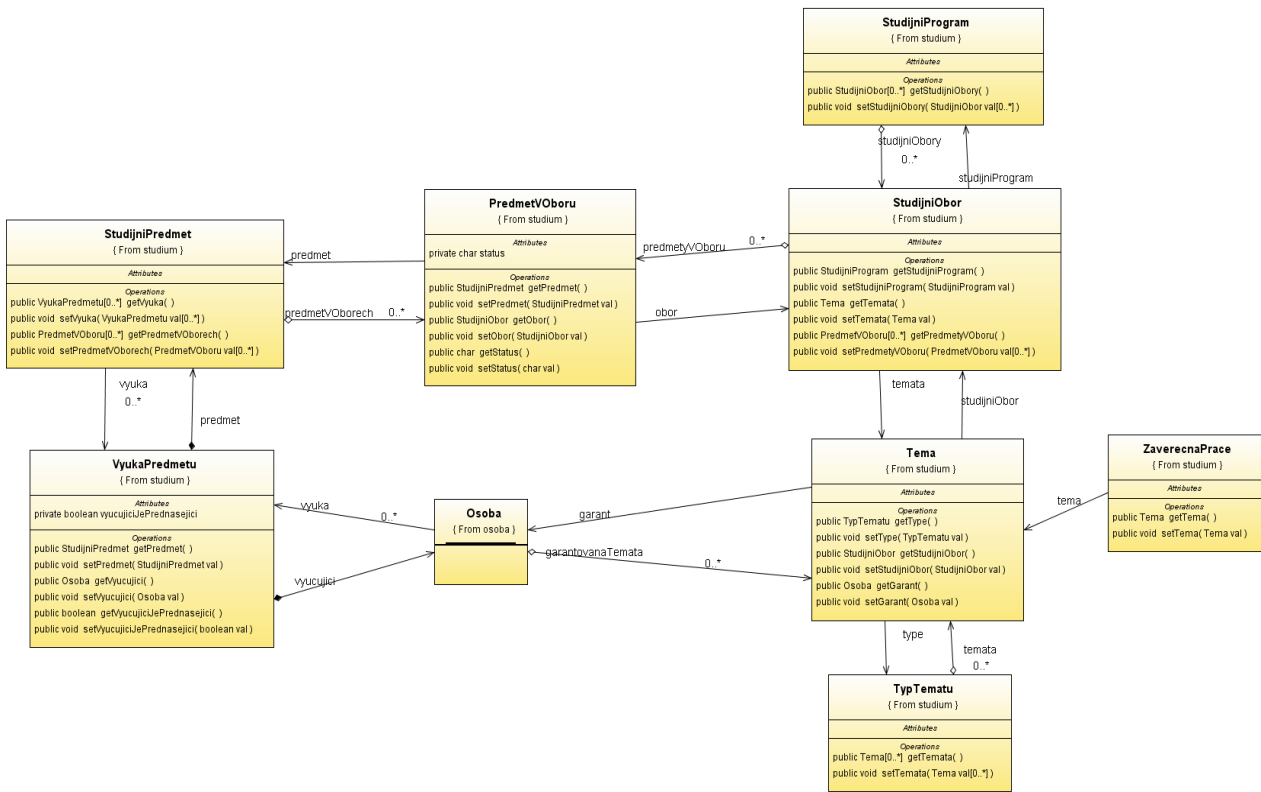
Příloha 1: UML diagramy hlavních tříd



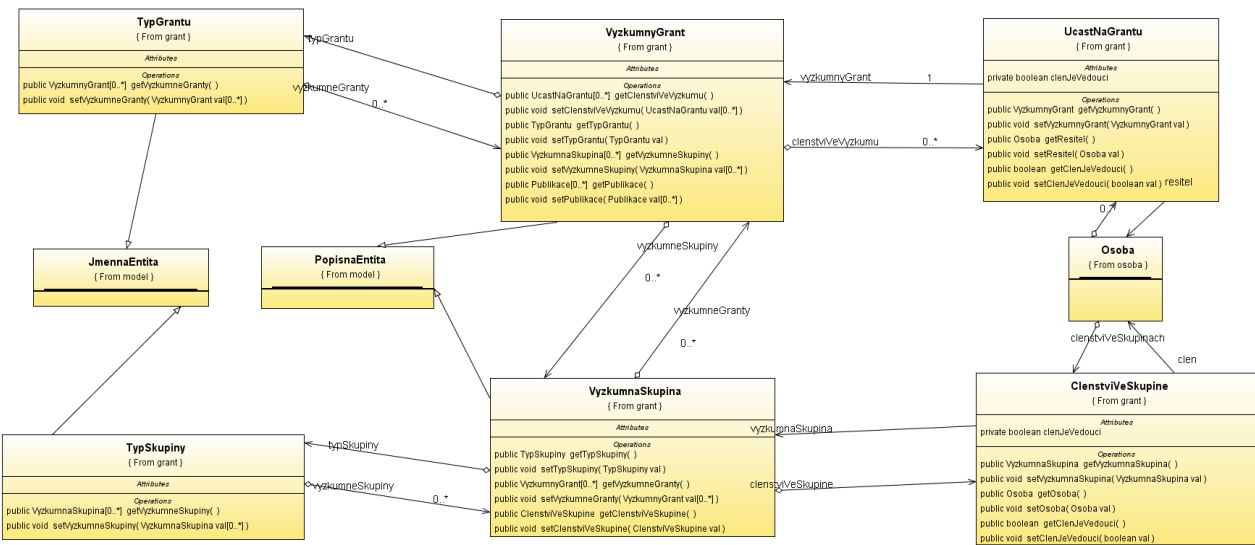
Ilustrace 1: Bázové entity



Ilustrace 2: Doménové třídy Osoby



Ilustrace 3: Doménové třídy Studium



Ilustrace 4: Doménové třídy Vyzkum

<<interface>> UdalostDao { From dao }
Attributes
Operations public Collection getUdalost() public Udalost getUdalost(int id) public Collection getUdalostiProOsobu(Osoba osoba) public int ulozUdalost(Udalost udalost) public void smazUdalost(Udalost udalost) public void deaktivujUdalost(Udalost udalost)

<<interface>> TemaDao { From dao }
Attributes
Operations public Collection getTemata() public Collection getTemataOsoby(int osobald) public Collection getTemataOboru(int oborld) public Collection getTemataProTyp(int typId) public Collection getTypyTemat() public Tema getTema(int id) public TypTematu getTypTematu(int id) public int ulozTema(Tema tema) public int ulozTypTematu(TypTematu typ) public void smazTema(Tema tema) public void smazTypTematu(TypTematu typ) public void deaktivujTema(Tema tema) public void deaktivujTypTematu(TypTematu typ)

<<interface>> StudiumDao { From dao }
Attributes
Operations public Collection getObory() public Collection getPredmety() public Collection getProgramy() public StudijniObor getObor(int id) public StudijniProgram getProgram(int id) public StudijniPredmet getPredmet(int id) public Collection getOboryProgramu(int programId) public Collection getPredmetyOboru(int oborld) public void ulozObor(StudijniObor obor) public void ulozPredmet(StudijniPredmet predmet) public void ulozProgram(StudijniProgram program) public void smazObor(StudijniObor obor) public void smazPredmet(StudijniPredmet predmet) public void smazProgram(StudijniProgram program) public void deaktivujObor(StudijniObor obor) public void deaktivujProgram(StudijniProgram program) public void deaktivujPredmet(StudijniPredmet predmet)

<<interface>> OsobaDao { From dao }
Attributes
Operations public Collection getOsoby() public Collection getClenstvi() public Collection getAktivity() public Collection getPozice() public Collection getAktivityOsoby(int osobald) public Collection getClenstviOsoby(int osobald) public Collection getPoziceOsoby(int osobald) public Collection getOsobyProClenstvi(int clenstvid) public Collection getOsobyProAktivitu(int aktivitald) public Collection getOsobyProRoli(int poziceId) public int ulozitOsobu(Osoba osoba) public int ulozitAktivitu(Aktivita aktivita) public int ulozitClenstvi(Clenstvi clenstvi) public int ulozitPozici(Pozice pozice) public void smazatOsobu(Osoba osoba) public void smazatAktivitu(Aktivita aktivita) public void smazatClenstvi(Clenstvi clenstvi) public void smazatPozici(Pozice pozice) public void pridatAktivituOsobe(Aktivita aktivita, Osoba osoba) public void pridatClenstviOsobe(Clenstvi clenstvi, Osoba osoba) public void pridatPoziciOsobe(Pozice pozice, Osoba osoba) public void deaktivujOsobu(Osoba osoba) public void deaktivujClenstvi(Clenstvi clenstvi) public void deaktivujAktivitu(Aktivita aktivita) public void deaktivujPozici(Pozice pozice)

<<interface>> PublikaceDao { From dao }
Attributes
Operations

<<interface>> GrantDao { From dao }
Attributes
Operations

Ilustrace 5: Rozhraní pro DAO třídy