WPA - Tvorba doménového modelu pro semestrální práci

Pavel Mička, Bogdan Kostov

Obsah

Semestrální práce (CP2)

- V rámci CP2 musí semestrálka (zejména) splňovat
  - Použití dědičnosti (postačuje i *@MappedSuperclass*)
  - Alespoň jedna z následujících technik: Ordering (*@OrderBy*), Named queries, Cascading, složený klíč, mapování JPQL/native dotazu
  - Business logika
  - Návrh transakcí, rozsahů platností bean, komunikace s okolím
  - Primitivní rozhraní pro testování (jUnit je asi nejvhodnějším prostředkem)

Dědičnost

- Klasická (logická) dědičnost není nejčastějším use-case v informačních systémech
- Dědičnost na úrovni společného identifikátoru (tj. fyzické reprezentace) je téměř nutností (*@MappedSuperclass*)
  - Zajišťuje *DRY princip* (do not repeat yourself) — id a metody equals a hashcode jsou definovány pouze jednou ve společném předkovi
  - Implementuje Serializable, čímž zajišťuje serializovatelnost všech potomků

@MappedSuperclass — příklad

```
@MappedSuperclass
public abstract class AbstractBusinessObject
    implements Serializable {
    @Id
    @GeneratedValue(generator="system-sequence")
    @GenericGenerator(name="system-sequence",
        strategy = "sequence")
    protected Long id;
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    @Override
    public boolean equals(Object obj) {...}
    @Override
    public int hashCode() {...}
}
```

NamedQueries

- Vhodné k rychlému prototypování aplikace
- Pro reálné nasazení značně nevhodné. Použití NamedQuery obvykle vede na tyto problémy :
  - Obrovská redundance kódu
  - voláno EntityManagerem ze servisní vrstvy — porušuje zapouzdření DB logiky
  - Nízká modifikovatelnost/upravitelnost
- Pro použití v semestrální práci silně nedoporučujeme (DB logiku doporučujeme zakrýt pomocí *data access objektů*)

Mapování nativních dotazů

- JPQL neobsahuje např. UNION
- Nativní dotaz je vhodné použít pokud aplikace vyžaduje složitou pro implementaci funkcionalitu, kterou čisté ORM nepodporuje, avšak databáze nativně podporuje.
- Obecně však platí zásada se nativním dotazům maximálně vyhýbat
  - Zvyšuje závislost na implementaci konkrétní DB

Business logika

- Objekt = Data + Operace
  - Závislosti mohou být injektovány i do entit pomocí Springovské anotace *@Configurable*
- Logika (týkající se jednoho objektu) by neměla být separována v servisní vrstvě
  - Vede na transakční skript (programování s objekty)
  - Znemožňuje použití polymorfismu (nahrazuje ho masivním využíváním instanceof operátoru)
  - Špagetizuje a demodularizuje kód
- Pouze logika agregativního charakteru by měla být v servisní vrstvě
- Business logika nikdy nesmí být ve vrstvě řadičů (controllerů)

Rozhraní aplikace

- Rozhraním aplikace by měla být servisní vrstva
- Zajišťuje transakční zpracování a zabezpečení operací (security)
- Okolí aplikace (např. view frameworky) komunikuje striktně skrz ní
- Komunikace probíhá pomocí Data transfer objektů (ne pomocí entit)

Další doporučení

- Máme dobré zkušenosti s ORM frameworkem Hibernate (podporuje i JPA)
- Jako databázi nepoužívejte Derby — použijte takovou DB, kterou byste použili v reálném nasazení (zde můžeme doporučit PostgreSQL)

Dotazy k semestrálkám

- Dotazy k semestrálkám?

Literatura

- Hibernate 4.1 Core — dokumentace

  `http://docs.jboss.org/hibernate/orm/4.1/manual/en-US/html_single/`
- Data access object design

  `http://www.scribd.com/doc/99704289/Data-Access-Object-Design`
- Spring Data

  `http://www.springsource.org/spring-data`
- Service Layer

  `http://martinfowler.com/eaaCatalog/serviceLayer.html`