Databáze v Java aplikacích, JPA – Handout

Martin Ledvinka
martin.ledvinka@fel.cvut.cz

11. listopadu 2015

1 Přístup k databázím v Javě

Jednou z mnoha výhod (a občas nevýhodou) platformy Java je její zaměření na standardizovanost, která umožňuje bezbolestně měnit poskytovatele služeb díky jednotnému rozhraní (teoreticky, v praxi to tak ne vždy funguje). Stejná situace je i u přístupu k databázím.

Databázový přístup v Javě je řešen ve dvou hlavních standardech:

- JDBC
- JPA

Teorii jste měli na přednášce, proto se jimi budeme zabývat jen velmi stručně.

1.0.1 Java Database Connectivity (JDBC)

Nízkoúrovňový standard, který řeší spojení s databázovým strojem a jeho jednotné API. To v praxi znamená, že nám vrstva zvaná JDBC Driver poskytne spojení (`Connection`), přes které můžeme s databází komunikovat pomocí SQL dotazů/příkazů (objekty třídy `Statement` či `PreparedStatement`) podobně, jako z utilit typu PSQL. Rozdíl mezi `Statement` a `PreparedStatement` je ten, že `Statement` bere kompletní string, zatímco `PreparedStatement` umožňuje statement parametrizovat a přepoužít. Pokud chceme podobný dotaz využít vícekrát, jen s jinými hodnotami, je `PreparedStatement` především z hlediska výkonu jasně lepší variantou.

Výstupem SELECT dotazu je v JDBC `ResultSet`, který reprezentuje seznam řádků odpovídajících našemu dotazu. Sloupce seznamu odpovídají buď celé tabulce (v případě použití * v dotazu), či vybraným sloupcům dotazu. Hodnoty z ResultSetu získáváme při iteraci voláním funkcí `get`*`Type`*, kde *`Type`* je jeden z podporovaných typů, např. String, Boolean, Integer apod. Samozřejmě hodnota v daném sloupci musí být mapovatelná na specifikovaný typ. `getBoolean` na sloupec typu INT tedy nebude fungovat.

Tento přístup je pro rozsáhlejší využití v aplikaci poněkud nevhodný. Proto vznikly frameworky poskytující *objektově-relační mapování* (**ORM**).

1.1 ORM

Zjednodušeně řečeno, objektově relační mapování je technika mapování tříd objektových programovacích jazyků (Java, C#) na databázové tabulky. Jednotlivé instance tak odpovídají řádkům tabulky a atributy objektů sloupcům v tabulce. Mapované třídy se v Javě (a v podstatě i obecně) nazývají *entity*.

V minulosti vznikla řada ORM frameworků pro Javu. Hlavním zástupcem je jistě Hibernate. Tyto frameworky byly hnacím motorem pro vývoj standardu JPA.

1.1.1 Java Persistence API (JPA)

JPA je tedy javovský standard pro ORM frameworky. Referenční implementací současné verze – JPA 2.1 – je *Eclipselink*.

Hlavní třídou z JPA, se kterou přijde uživatel do styku, je `EntityManager`. Tato třída reprezentuje *persistentní kontext*. Řádky databáze lze pomocí entity manageru načíst (metoda `find`) jako objekty entity tříd a pracovat s nimi. Stejně tak můžeme nové řádky vkládat (`persist`) a existující mazat (`remove`).

Jako poznámku na okraj uveďme rozdíl mezi metodami `persist` a `merge` v interface `EntityManager`. `persist` slouží výhradně pro vkládání nových záznamů. Při pokusu o persist existujícího objektu dojde k výjimce. `merge` umí též vkládat nové objekty, ale jejím primárním účelem je aktualizace záznamu z *detached* entity (vzpomeňte na entity lifecycle).

Všechny manipulující operace `EntityManager` jsou transakční, tudíž jejich efekty se projeví až po volání *commit* běžící transakce. Jak uvidíme za několik týdnů, frameworky typu Spring či Java EE implementace umožňují transakce ovládat deklarativně, obvykle na úrovni metod (začátek metody = transaction begin, konec = transaction commit). Zatím musíme transakci ovládat programaticky, volání metody `getTransaction`.

Instance `EntityManager`, které samotné nejsou thread-safe, poskytuje objekt typu `EntityManagerFactory`, který vznikne při inicializaci persistentní jednotky (vizte dále).

Spíše než podrobnému popisu JPA, který jsme měli v přednášce, se nyní věnujme některým drobnostem a tipům.

Generated ID Generované identifikátory jsou velmi pohodlnou feature. Je třeba ovšem pamatovat na několik věcí. Za prvé, při vkládání nového objektu do DB nemusí být ID nastaveno okamžitě po volání `persist`. Specifikace říká, že ID musí být nastaveno v entitě při commitu transakce. Pro některé strategie generování ID je ID nastaveno hned při commitu, ale spoléhat na to nemůžeme.

Equals a hashCode Definice vlastních metod `equals` a `hashCode` je běžná praxe. V případě entit je ale třeba mít se na pozoru. Lákavou a logicky vyhlížející možností je použití ID objektu. Problém nastává u entit s generovaným ID, kdy často ID není přítomno. V takové případě `equals` a `hashCode` těžko budou plnit svou funkci. Obdobně je třeba dávat si pozor na použití generovaného ID jako

součást atributů porovnávaných v `equals` a `hashCode`. Po vygenerování ID totiž dojde ke změně hashCode a můžete se dostat do situace, kdy kontejnery založené na hash objektů (např. HashMap, HashSet) budou tvrdit, že neobsahují objekt, který vám vrátily při iteraci. Obecně je tedy třeba důkladně popřemýšlet, jakou strategii implementace `equals` a `hashCode` zvolíme a zda je vůbec vhodné je přepisovat.

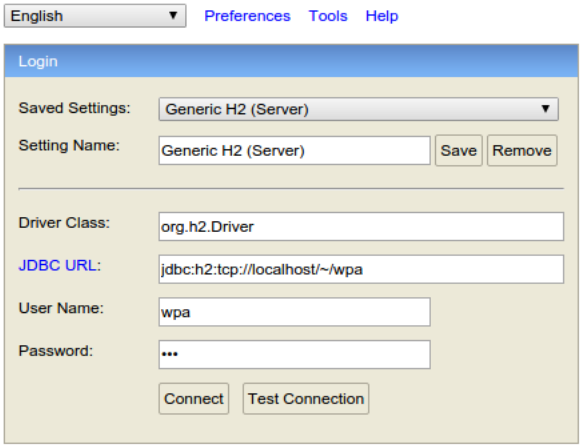
2 Demo aplikace

2.1 Spouštění

Pro běh demo aplikace použijte buď databázi H2 jako na cvičení, nebo nastavte v persistence.xml spojení k PostgreSQL databázi (stačí url, username, heslo a třída driveru, vizte zakomentované nastavení tamtéž). Pokud chcete použít jinou databázi, musíte také přidat odpovídající JDBC driver jako Maven dependency do pom.xml.

Pro H2 je postup následující:

1. `cd cesta/k/rozbalene/h2/bin`,
2. `java -cp *.jar org.h2.tools.Server`,
   - Na Windows musíte místo *.jar použít plné jméno jar souboru,
   - Pokud je obsazený port 8082, na kterém startuje webová konzole H2, můžete nastavit jiný přepínačem -webPort,
3. Otevře se vám prohlížeč s hlavní stránkou H2 konzole, vyberte nastavení Generic H2 (Server) a v JDBC URL nastavte jméno databáze (za posledním lomítkem). Můžete nastavit i username a heslo,
   - Screenshot 1 ukazuje nastavení webové konzole stejně, jako jsme měli na cvičení,
4. Klikněte na *Connect*,
5. V IDE použijte JDBC URL, username a heslo z H2 databáze pro nastavení spojení v persistence.xml,
6. Otevřete a spusťte `cz.cvut.kbss.wpa.jdbc.Main` pro ukázku použití JDBC,
7. Otevřete a spusťte `cz.cvut.kbss.wpa.jdbc.Main` pro ukázku použití JPA,
8. Prostudujte si kód s využití poznámek níže.



Obrázek 1: Nastavení H2 databáze.

2.2 Poznámky k aplikaci

Přidejme pár poznámek k demo aplikaci.

Persistence se inicializuje ve třídě `PersistenceFactory`, která slouží jako poskytovatel objektů typu `EntityManager`. Třída funguje jako jednoduchý singleton, ale není thread safe! Všimněte si, jak je `EntityManagerFactory` propojena s konfigurací persistence unit v persistence.xml.

Třída `BaseEntity` funguje jako parent všech ostatních tříd a definuje identifikátor entit, abychom ho nemuseli v každé entitě opakovat. `Person` pak demonstruje využití dědičnosti v JPA, kdy narozdíl od `BaseEntity` má vlastní DB tabulku a slouží jako plnohodnotná entita. Zkuste si nastavit jiné strategie dědičnosti a sledujte, jak se mění struktura tabulek v DB. `Employee` je nyní podtřídou `Person` a dědí hodnoty jména a příjmení. Dále definuje vztah s třídou `Department`. `Department.addEmployee` ukazuje, jak vhodné je definovat chování entit. Zároveň tu vidíme nutnost nastavení obou stran vztahu `Employee` – `Department`. Zkuste zakomentovat přidání zaměstnance do departmentu a spustit `Main` – department najednou nemá zaměstnatnce, ačkoliv v DB je cizí klíč u zaměstnance prokazatelně nastaven. O mapování sloupců (jména, unikátnost, neprázdnost) jsme už mluvili na cvičení.

Třídy `Student` a `Course` demonstrují další typ vztahů mezi entitami – *ManyToMany*. V tomto případě jen jednosměrný. Ale klidně si zkuste přidat i druhou stranu vztahu ve třídě `Course`. Stačí seznam studentů oanotovat anotací `@ManyToMany` a v atributu `mappedBy` specifikovat jméno odpovídajícího atributu ve třídě `Student`. Explicitní definice join tabulky ve třídy `Student` není nutná. Zkuste ji zakomentovat a podívejte se, jaké jméno zvolí JPA provider. Ještě doplňme, že není nutné pro multi-target vztahy používat jen `List`, stejně dobře by fungovaly `Set` či `Collection`. Nám se to hodí pro úkoly (vizte dále).

2.3 Vyzoušejte si

Jako (nepovinný) úkol si můžete vyzkoušet následující:

- Vylistujte zaměstnance jednoho oddělení pomocí dotazu (můžete použít zaměstnance přidaného ve funkci `addEmployeeToDepartment` nebo si přidejte další):
  - Nejprve nativní SQL dotaz,
  - Pak JPQL dotaz (inspirovat se můžete JPQL dotazem ve funkci `addStudentsAndCourses`),
- Upravte třídu `Student`, aby uspořádávala kurzy podle abecedy (vyzkoušejte tím, že přidáte studentovi více než jeden kurz a načtete ho z DB),
- Přidejte třídu `Phone`, která bude v *OneToOne* vztahu s třídou `Employee`. Zda by měl být vztah jedno či oboustranný necháme na vás.