Funkce repository návrhu systému a důvody pro její existenci.
4IT415 Informační modelování organizací (ZS 2014-2015)

Tomáš Maršálek

December 12, 2014

Repository pattern je návrhový vzor s účelem logického oddělení perzistenční vrstvy systému od aplikační. Do systému přináší nízkou provázanost (loose coupling), což je obecně považováno jako dobrá vlastnost při návrhu. Nízká provázanost umožňuje nahrazení části s minimálním zásahem do jiných částí systému. Proč bychom ale v první řadě uvažovali o nahrazování perzistenční vrstvy?

Prvním důvodem může být nahrazení implementace úložiště. Repository nám dává možnost snadné migrace mezi různými databázemi. Nebo pokud chceme nahradit pouze část úložiště, např. když chceme ukládat obrázky do souborů a ne přímo do databáze, aplikační kód se o téhle změně vůbec nemusí dozvědět, protože k obrázkům přistupuje pouze definovaných metod rozhraní repozitáře.

Změna implementace úložiště je ale většinou uváděna pouze jako ilustrativní využití repository, protože ke změnám implementace v produkčním systému dochází zřídkakdy. Většinou existuje datová základna a aplikace se podle ní přizpůsobují než naopak.

Dalším, a to podstatnějším, důvodem je možnost používat unit testy pro aplikační kód, který je provázaný perzistenčním kódem, a ne jen integrační testy, které testují aplikaci jako celek. Repository nám totiž umožňuje vytvořit takzvanou Mock implementaci perzistenčního rozhraní, díky které je možné jednoduše vytvořit sadu deterministických unit testů. Unit testy se skutečnou databází jsou velmi krkolomné, protože před každým testem je třeba testovací databázi vyprázdnit a deterministicky naplnit daty potřebnými pro konkrétní unit test.

Velkou výhodou oddělení perzistence od aplikace je cachování v paměti, o kterém aplikace nemusí vůbec vědět, protože se vše děje na pozadí v perzistenční vrstvě, která může najednou přistupovat přímo k databázi jako datovému úložišti a mezitím k dočasnému úložišti v paměti nebo k jiné databázi, která slouží jako cache.

Repository pattern není často oblíbený, protože není flexibilní. Rozhraní repository pak může obsahovat metodu pro každou permutaci získání objektu (např. getUserById, getUserByName, getUsersWithDetails, getUsersWithDetailsAndFavouriteColorByAddress). Tento problém spadá do většího problému zvaného object-relational impedance mismatch, který ve zkratce znamená, že je velmi obtížné přesně namapovat objekty z OO světa do relačního modelu. Doposud nikdo nepřišel s řešením, které by bylo vhodné pro všechny případy a stále by mělo vlastnosti nízkého provázání perzistenční a aplikační vrstvy. Problém dostal přezdívku „Vietnam of Computer Science“ v analogii s Vietnamskou válkou, do které se dlouhé roky investovaly neuvěřitelné peníze a výsledek nikde.

Repository je jedním z řešení tohoto problému s vlastností nizkého provázání, avšak za cenu velmi složitého rozhraní, jak bylo zmíněno výše. Jediným uváděným řešením jak se zbavit O/R mapování je odstranit z názvu „O“ nebo „R“, tedy nepoužívat objekty v aplikaci, anebo nepoužívat relační, ale například objektovou nebo grafovou NoSQL databázi. Dalšími navrhovanými řešeními jsou experimentální Event Sourcing (např. Akka persistence ve Scale nebo acid-state v Haskellu).

Řešení, které se v poslední době s rozšířením architektury MVC pro webové aplikace stalo populární, je takzvané ORM - Object Relational Mapper, neboli nástroj, který provede konverzi dat z tabulek relační databáze do grafové reprezentace objektů v objektově orientovaném programování. Vývojář se nesetká s neduhy jako u repository, kde musel vytvořit každou permutaci databázového dotazu, ale tvoří databázové dotazy přímo v kódu podle potřeby. Nevytváří je ale v jazyku závislém na implementaci databáze, ale v konstrukcích nezávislých na implementaci. Alespoň tohle byl původní záměr a bohužel skutečnost je jiná. ORM přináší pouze výhodu v tom, že se v aplikační vrstvě nemusí používat přímo databázový dotazovací jazyk, ale jeho varianta, která odpovídá konstrukcím programovacího jazyka aplikační vrstvy. Nedocílíme nízkého provázání, nelze vytvářet mocky objekty (implementace ORM nástrojů je hutná a vytvořit mock, který by splňoval jejich rozhraní je velmi obtížné) a cachování dat je obtížné, protože v některých výjimečných případech musíme obejít ORM vrstvu a přistupovat přímo do perzistenční.

Každé řešení má své výhody a nevýhody a proto nachází uplatnění na jiných pozicích. ORM získalo v poslední době ve webových frameworcích místo mezi aplikací a perzistencí i přes veliký tábor jeho odpůrců. Repository pattern se často využívá při návrhu API, který v poslední době silně konverguje k jednomu řešení - REST (Representational state transfer pro protokol HTTP).

Praktické rady při použití repository patternu.

Přestože je repository pattern velmi neflexibilní s přibývajicími požadavky na variaci dotazů

na data, můžeme alespoň částečně přidat flexibilitu dotazu, aniž bychom narušili únik abstrakce skrze vrstvy nebo volné provázání. Rozhraní repozitáře by mělo být schopné alespoň stránkovat a řadit výsledky.

Jiným návrhem pro návrh repository je takzvané CQRS (Command Query Responsibility Segregation), které jednoduše odděluje zápisy do perzistence a čtení do oddělených komponent. To umožňuje použít dva ruzné oddělené objektové modely nebo náhledy na stejná data, které jsou vhodnější pro čtení nebo zápis než jednotný model. Výhodou nízkého provázání čtení a zápisu je flexibilita při nasazení aplikace na distribuovaný systém - např. čtecí repositář může obsluhovat 10 výpočetních jednotek a pro zápis můžou být pouze 2.

Cílem eseje bylo přiblížit účel repository patternu, jeho výhody a nevýhody a porovnání s ostatními řešeními problému objektově relačního mapování. Dále jsem zmínil postavení repository patternu v dnešní době ve skutečných aplikacích a rady, které vývojová komunita po letech praxe nalezla. Esej se dotýká širokého rozsahu témat spojených s timto návrhovým vzorem a pro plné pochopení problematiky doporučuji detailnější literaturu, která je většinou ve formě blogových příspěvků a zejména diskusí pod nimi.