PV179 – Dotazovanie a modifikácia persistentných dát, Micro ORMs

Martin Macák

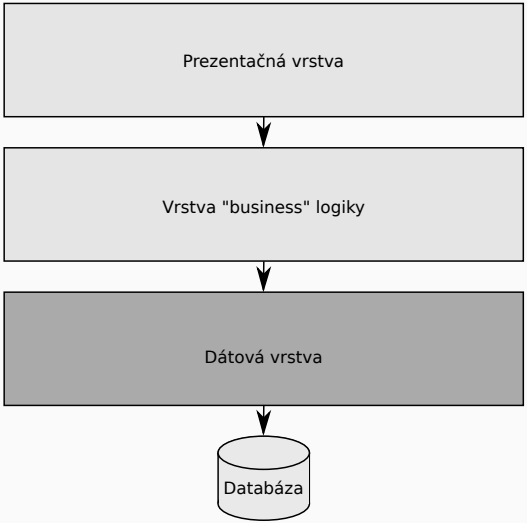
10. 10. 2023

Fakulta informatiky, Masarykova univerzita, Brno

Osnova prednášky

1. Úvod do DAL architektúry
2. Dotazovanie a modifikácia dát
3. Micro ORMs

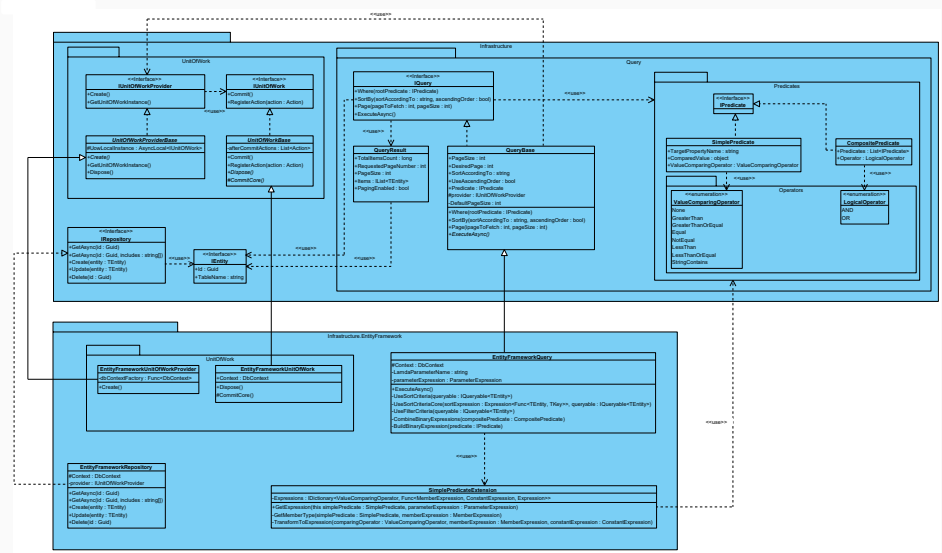
V minulej časti ste videli...



V minulej časti ste videli...

- Entity Framework
  - entity
  - DbContext – **Identity Map**, **Unit of Work**, **Lazy Loading**
  - DbSety – **Repository**

Čo nám ostáva? Príprava infraštruktúry :)



Čo nám ostáva?

- Repository
  - Create, Read, Update, Delete
- Query
  - dotazovanie
  - napr:
    SELECT TOP 10 * FROM Customers
    WHERE Country='USA' AND Age>=18
    ORDER BY Name DESC
- Unit of Work
  - sleduje zmeny v entitách v priebehu danej transakcie

Návrhový vzor Repository

- Create, Read, Update, Delete
- **Prečo robiť vlastný Repository, keď máme DbSet?**
  - Úroveň abstrakcie nad ORM frameworkom
  - Modifikácia dát na jednom mieste

Query

- Reprezentuje dotazovanie.
- Chceme ho nezávislé na ORM frameworku
  - Použité ORM v DAL nemusí vyhovovať po celú dobu.
  - Najčastejšie sa mení kvôli výkonu.
- Naša reprezentácia dotazu by mala podporovať:
  - komplexné WHERE filtrovanie,
  - radenie,
  - stránkovanie.

Návrhový vzor Unit of Work

- sleduje zmeny v entitách v priebehu danej transakcie
- na záver transakcie sa zavolá commit
- **prečo robiť vlastný UoW, keď máme DbContext?**
  - abstrakcia prístupu k dátam cez konkrétne ORM
  - prehľadnejšie transakcie

Dotazovanie a modifikácia dát

1. SQL
   - nevýhody:
     - databázovo závislé
     - žiadna kontrola kompilátorom
2. LINQ to Entities
   - výhody:
     - podobný LINQ to Objects
     - IntelliSense
     - kontrola kompilátorom

Stavy entít

- Added: nie je v DB, sledovaná, má sa pridať
- Unchanged : je v DB, sledovaná, nezmenená
- Modified: je v DB, sledovaná, má sa zmeniť
- Deleted: je v DB, sledovaná, má sa zmazať
- Detached: nie je sledovaná

Nastavenie logovania SQL commandov

- Pre testovacie účely to dáme priamo do DbContextu.
- Potrebujeme package *Microsoft.Extensions.Logging.Console*.

Vyhodnotenie dotazu

- Na strane servera – dáta nie sú v pamäti, filtrovanie prebieha v databáze, typicky práca s **IQueryable**
- Na strane klienta – dáta sú v pamäti, filtrovanie prebieha na klientovi, typicky práca s **IEnumerable**

Operácie v LINQ to Entities

- Select: metódy *ToList*, *First*, *Last*, *Find*, ...
- Insert: metóda *Add*, *AddRange* alebo nastaviť stav *Added*
- Update: zmeniť danú property v tom istom DbContexte, metóda *Update*, *UpdateRange* alebo nastaviť stav *Modified*
- Delete: metóda *Remove*, *RemoveRange* alebo nastaviť stav *Deleted*
- Metódy Add, Update a Remove (+ Range varianty) idú volať aj priamo na DbContexte!

Pozor na nesúvislé spojenie!

- Dáta sú lokálne = no problem.
- Webová aplikácia = nemáme jeden, ten istý DbContext.
  - Pošlem dáta zo servera, DbContext sa ukončí.
  - Keď niečo zmením na klientovi, tak sa vytvorí nový DbContext.
  - Ten netuší, čo ste robili s entitami.
  - Je **vaša zodpovednosť** oznámiť mu stavy entít.

Práca s navigation properties

Predpokladajme, že máme entitu A, ktorá má ako navigačnú property List<B> (je s B vo vzťahu 1 : N).

- Insert
  - ak pridávame do databázy B s nastavenou property typu A, pridá sa stav *Added* na B, ale automaticky aj na A
  - toto nie je vždy to, čo chceme!
- Select
  - pri vrátení entity A bude B štandardne **null**
  - pre získanie B sa využívajú tieto techniky:
    - Eager Loading – vráti všetky A priamo aj so všetkými B (metóda *Include*)
    - Explicit Loading – pre daný objekt A načíta všetky B (metóda *Load*, prípadne *Select*)
    - Lazy Loading – ak sa niekto dotáže napr. na List<B> v danom objekte A, automaticky sa načíta (virtual + EFCore.Proxies + UseLazyLoadingProxies)

Práca s navigation properties

Predpokladajme, že máme entitu A, ktorá má ako navigačnú property List<B> (je s B vo vzťahu 1 : N).

- Update
  - ak nechcem entite B zmeniť property A, ale nejakú inú, nemusím stále načítať aj A
- Delete
  - ak chcem zmazať entitu A, musím načítať aj entitu B, aby jej mohlo zmeniť cudzí kľúč (prípadne ju zmazať)

MicroORMs

- rýchlejšie ORMs
- jednoduché na použitie
- obmedzená funkcionalita

MicroORMs: príklady

- Dapper
- PetaPoco
- Simple.Data
- ...

Raw SQL (db.Pets.First(p => p.Type == "dog")

```
// Raw SQL
using (SqlConnection con = new SqlConnection(connectionString))
{
    con.Open();
    using (SqlCommand command = new SqlCommand(
        "SELECT * FROM Pets WHERE Type='dog'", con))
    {
        using (SqlDataReader reader = command.ExecuteReader())
        {
            while (reader.Read())
            {
                var firstDog = (new Pet
                {
                    Id = reader.GetInt32(0),
                    Name = reader.GetString(1),
                    Type = reader.GetString(2),
                    Age = reader.GetInt32(3)
                });
            }
        }
    }
}
```

MicroORMs ukážky

```
// Dapper
using (var db = new SqlConnection(connectionString))
{
    var firstDog = db.Query<Pet>
        ("SELECT * FROM Pets WHERE Type=@type", new { type = "dog" });
}

// PetaPoco
using(var db = new PetaPoco.Database("PetsConnectionString"))
{
    var firstDog = db.Query<Pet>
        ("SELECT * FROM Pets WHERE Type=@type", new { type = "dog" });
}

//Simple.Data
dynamic sDdb = Simple.Data.Database.OpenConnection(connectionString);

dynamic sDfirstDog = sDdb.Pets.FindByType("dog");
```

Rekapitulácia

1. Návrhové vzory
   - Repository a Unit of Work
2. Dotazovanie
   - Možnosti
   - Stavy entít
   - Vyhodnotenie dotazu
   - LINQ to Entities
   - Navigation properties
3. MicroORMs

macak@mail.muni.cz