Tehnici Avansate de Programare

Unit of Work pattern & Dependency Injection

Petru Rebeja, Marius Apetrii

8 Aprilie 2021

Facultatea de Matematică
Universitatea Alexandru Ioan Cuza, Iași

Introducere

Recapitulare

- LINQ — O mulțime de funcții ce permit interogarea uniformă a colecțiilor de date din diverse surse.
- Entity Framework — O platformă care permite interogarea și manipularea datelor folosind paradigma orientat-obiect.
- Repository Pattern — Șablon de proiectare ce decuplează logica aplicației de accesul la date.

Agenda

- Unit of Work pattern
- Dependency Injection

Unit of Work **pattern**

Repository

```
public interface IRepository<TEntity>
{
    IQueryable<TEntity> Query();
    void Insert(TEntity entity);
    void Update(TEntity entity);
    void Delete(TEntity entityId);
    void SaveChanges();
}
```

Repository

- Folosim o instanță de `repository` pentru a efectua operații `CRUD` asupra uneia sau a mai multor entități de același tip.
- Toate modificările sunt păstrate în memorie.
- Metoda `SaveChanges()` este apelată pentru a salva modificările în baza de date.

Scenariu problematic

Vrem să introducem în baza de date un client nou și datele de contact ale acestuia:

```
customerRepository.Insert(new Customer{...});
customerRepository.SaveChanges();

addressRepository.Insert(new CustomerAddress{...});
addressRepository.SaveChanges();
```

Scenariu problematic

Vrem să introducem în baza de date un client nou și datele de contact ale acestuia:

```
customerRepository.Insert(new Customer{...});
customerRepository.SaveChanges();

addressRepository.Insert(new CustomerAddress{...});
addressRepository.SaveChanges();
```

Ce se întâmplă dacă `addressRepository.SaveChanges()` aruncă o excepție?

Scenariu problematic

Ce se întâmplă dacă `addressRepository.SaveChanges()` aruncă o excepție?

- Clientul este salvat în baza de date dar adresa nu.
- Baza de date rămâne într-o stare neconsistentă.
- Pentru a reveni la consistență clientul fără adresă trebuie șters.

Scenariu problematic

- Astfel de scenarii se regăsesc des în cerințele aplicațiilor.
- Atunci când lucrăm cu mai mult de două entități asigurarea consistenței datelor devine un mecanism foarte complex.

Soluția

Pentru a evita astfel de probleme folosim `Unit of Work`.

Unit of Work

`Unit of Work` ține evidența tuturor modificărilor aferente bazei de date efectuate în timpul unei tranzacții logice din aplicație. La finalul tranzacției modificările sunt salvate în baza de date într-un mod care asigură atomicitatea și integritatea datelor.[1]

[1] `https://www.martinfowler.com/eaaCatalog/unitOfWork.html`

Unit of Work Pattern

```
public interface IRepository<TEntity>
{
    IQueryable<TEntity> Query();
    void Insert(TEntity entity);
    void Update(TEntity entity);
    void Delete(TEntity entityId);
}

public interface IUnitOfWork
{
    void Commit();
}
```

- Metoda `SaveChanges()` devine redundantă și trebuie eliminată din interfața `IRepository`.

Exemplu

```
public class RegisterCustomerTransaction
{
    public RegisterCustomerTransaction(
        IRepository<Customer> customerRepository,
        IRepository<CustomerAddress> addressRepository,
        IUnitOfWork unitOfWork)
    {
        ...
    }

    public void Execute(Customer customer, CustomerAddress address)
    {
        customerRepository.Insert(customer);
        addressRepository.Insert(address);
        unitOfWork.Commit();
    }
}
```

Depencency Injection

Definiție

Dependency Injection
`Dependency Injection` este o tehnică prin care un obiect îi furnizează altuia dependențele de care acesta din urmă are nevoie.[2]

[2] `https://en.wikipedia.org/wiki/Dependency_injection`

Definiție

Dependency Injection

`Dependency Injection` este un termen de 25 dolari pentru un concept de 5 cenți. [...] `Dependency Injection` înseamnă să-i dai unei instanțe a unui anumit obiect variabilele de care are nevoie.[3]

[3] `https://www.jamesshore.com/Blog/Dependency-Injection-Demystified.html`

Avantaje

- Separarea responsabilităților[4]:
  - Injectorul construiește dependențele și le injectează în client și
  - Clientul manipulează dependențele pentru obținerea unui anumit scop.
- Reducerea codului de umplutură și a gradului de acuplare.

[4] https://en.wikipedia.org/wiki/Dependency_injection

Avantaje

- Flexibilitatea la schimbări fără necesitatea de recompilare — efectele execuției codului client pot fi modificate prin injectarea unei implementări diferite.
- Facilitează testarea în izolare.

Dezavantaje

- Grad de abstractizare mai mare care crește dificultatea codului.
- Induce o dependență față de o bibliotecă pentru `Dependency Injection`.
- Poate duce la o acuplare mai mare prin utilizarea defectuoasă.

Service Locator

`Service Locator`
Ideea de bază a unui `Service Locator` este să ai la dispoziție un obiect care știe cum să obțină, la cerere, instanțe ale tuturor serviciilor necesare unei aplicații[5].

N.B.: O instanță de serviciu = o dependență.

[5] `https://martinfowler.com/articles/injection.html#UsingAServiceLocator`

Service Locator — exemplu

```
public class RegisterCustomerTransaction
{
    public RegisterCustomerTransaction()
    {
        customerRepository = ServiceLocator
            .GetInstance<IRepository<Customer>>();
        addressRepository = ServiceLocator
            .GetInstance<IRepository<CustomerAddress>>();
        unitOfWork = ServiceLocator.GetInstance<IUnitOfWork>();
    }
}
```

Service Locator

- Este un anti-șablon[6].
- Induce un grad mare de acuplare.
- Ascunde dependențele.

[6] https://blog.ploeh.dk/2010/02/03/ServiceLocatorisanAnti-Pattern/

Încheiere

Recapitulare

Recapitulare

Unit of Work

Recapitulare

Unit of Work

Este un șablon care ne permite să executăm toate modificările aferente bazei de date într-o singură tranzacție.

Recapitulare

Unit of Work

Este un șablon care ne permite să executăm toate modificările aferente bazei de date într-o singură tranzacție.

Dependency Injection

Recapitulare

Unit of Work

Este un șablon care ne permite să executăm toate modificările aferente bazei de date într-o singură tranzacție.

Dependency Injection

Este o modalitate de a-i da unei instanțe variabilele de care aceasta are nevoie separând astfel crearea de instanțe de utilizarea lor.

Vă mulțumesc!

Mulțumesc pentru atenție!