Projektowanie aplikacji bazodanowych w .NET

Wykład 5

Paweł Rajba

Instytut Informatyki

Uniwersytet Wrocławski

Plan wykładu

- CastleProject i wzorzec ActiveRecord
- Wybrane możliwości Castle ActiveRecord
- Inicjalizja i konfiguracja
- Tworzenie klasy encji, dziedziczenie i komponenty
- Pobieranie obiektów

CastleProject i wzorzec ActiveRecord

- CastleProject: http://www.castleproject.org/
- Co wchodzi w skład CastleProject
  - MonoRail – framework MVC
  - ActiveRecord – implementacja wzorca Active Record w oparciu o NHibernate
  - MicroKernel – lekka implementacja wzorca Inversion of Control
  - WindsorContainer – rozszerzona wersja MicroKernel
- Krótka charakterystyka wzorca ActiveRecord
  - Skojarzenie wiersza tabeli z obiektem
  - Metody obiektu działają na jednym wierszu
  - Implementacja operacji CRUD
  - Implementacja logiki biznesowej

Wybrane możliwości Castle ActiveRecord

- Automatyczne generowanie i aktualizowanie schematu bd
- Automatyczna implementacja operacji CRUD
- Mapowanie umożliwia realizację:
  - Kluczy głównych
  - Kolumn wraz z blobami
  - Relacji (wszystkie typy)
  - Sesji i transakcji
  - Leniwego ładowania
- Implementacja logiki biznesowej

Konfiguracja

- W celu skonfigurowania framework'a niezbędne jest zdefiniowane 4 parametrów:
  - `hibernate.connection.driver_class`
  - `hibernate.dialect`
  - `hibernate.connection.provider`
  - `hibernate.connection.connection_string`
- Konfiguracja możemy utworzyć na następujące spoosoby
  - ```
    Hashtable properties = new Hashtable();
    properties.Add( ... )
    InPlaceConfigurationSource src = new InPlaceConfigurationSource();
    src.Add( typeof( ActiveRecordBase ), properties );
    ```
  - `XmlConfigurationSource src = new XmlConfigurationSource( "AR.xml" );`
  - `IConfigurationSource src = ActiveRecordSectionHandler.Instance;`

    (w tym przypadku konfiguracja jest pobierana z pliku konfiguracyjnego aplikacji)
    ```

Inicjalizacja

Możemy ją przeprowadzić na dwa sposoby:

- `ActiveRecordStarter.Initialize( src, typeof( Osoba ) );`
- ```
  ActiveRecordStarter.Initialize(
      Assembly.GetAssembly( typeof( Osoba ) ), src );
  ```

```
ActiveRecordStarter.Initialize(
    Assembly.Load( "FirstExample"), src );
```

W drugim przypadku zaczytywane są wszystkie typy publiczne

- warto mieć assembly tylko dla klas encji – zaczytywanie nie będzie się niepotrzebnie trwać za długo
```

Przykład

- `FirstExample`

Klasy encji

Wprowadzenie

- Klasa staje się klasę utrwalaną poprzez określenie
  - atrybutu *ActiveRecordAttribute*
    - Dodatkowo można podać parametr typu string, który określa nazwę odpowiadającej tabeli
  - oraz dziedziczenia z klasy *ActiveRecordBase*
- Klasa ma schemat:

```
using Castle.ActiveRecord;
[ActiveRecord]
public class Product : ActiveRecordBase
{ ... }
```

Klasy encji

Klucze główne

- Klucz główny określamy poprzez atrybut *PrimaryKeyAttribute*
- Możemy podać szereg parametrów (są analogonami do parametrów z NHibernate'a)
- Klucz złożony realizujemy następująco
  - Tworzymy klasę serializowalną *K*, która będzie kluczem
    - określamy właściwości klucza opatrując je atrybutem *[PropertyAttribute]*
    - nadpisujemy metody *GetHashCode* i *Equals*
  - W klasie właściwej tworzymy właściwość typu *K* i opatrujemy ją atrybutem *[CompositeKey]*

Klasy encji

Proste mapowanie

- Odbywa się poprzez atrybut
  - *PropertyAttribute* dla właściwości,
  - *FieldAttribute* dla pól.
- Możemy podać szereg parametrów (są analogonami do parametrów z NHibernate'a)

Relacje

- Przyjrzymy się relacjom
  - Many-to-one (realizowana przez atrybuty *BelongsTo* i *HasMany*)
  - Many-to-many (realizowana przez atrybut *HasAndBelongsToMany*)
  - One-to-one (realizowana przez atrybut *OneToOne*)

Przykład

- `CompositeKeyExample`
- `OneToOneExample`
- `ManyToOneExample`
- `ManyToManyExample`

Dziedziczenie i komponenty

Dziedziczenie

- Do dyspozycji mamy 3 strategie, które są dostępne w NHibernate
  - Wspólna tabela (z dyskryminatorem)
  - Hierarchia tabel z dziedziczonym kluczem (joined subclass)
  - Tabela na każdą klasę (nie trzeba oprogramowywać)
- Wspólna tabela jest realizowana z wykorzystaniem atrybutów *DiscriminatorColumn*, *DiscriminatorType*, *DiscriminatorValue*
  - *DiscriminatorColumn*, *DiscriminatorType*, *DiscriminatorValue* – w klasie bazowej
  - *DiscriminatorValue* – we wszystkich klasach potomnych

Dziedziczenie i komponenty

Dziedziczenie

- Hierarchia tabel z dziedziczonym kluczem realizowana jest z wykorzystaniem atrybutów *JoinedBase* i *JoinedKey*
  - *JoinedBase* umieszczamy obok atrybutu *ActiveRecord* w klasie bazowej
  - *JoinedKey* umieszczamy zamiast *PrimaryKey* w klasie potomnej

Komponenty

- W terminologii ActiveRecord nazywa się to *nested data*
- Realizowane za pomocą atrybutu *Nested*

Przykład

- `Inheritance1Example`
- `Inheritance2Example`
- `NestedDataExample`

Pobieranie obiektów

- Mamy kilka sposobów pobierania rekordów:
  - Poprzez metody obiektu: Find, FindAll, FindByProperty, itd.
  - Poprzez zapytania HQL i SQL
  - Poprzez określenia kryteriów (*ICritetion*)

Przykład

- `Quering1Example`
- `Quering2Example`