Mapowanie baz danych LINQ

Magdalena Pluta

Plan prezentacji

- Nowości w C# 3.0
- Czym jest LINQ
- Integracja z SQL
- Wnioski

Nowości w C# 3.0

- Typy anonimowe
- Automatyczne właściwości i inicjowanie obiektów
- Metody rozszerzające
- Wyrażenia lambda

Typy anonimowe

- Słowo kluczowe ***var***
- Musi być od razu zainicjowane
- Nie może być zwracane z funkcji
- Może być zadeklarowane jedynie lokalnie (wewnątrz metody)
- Zamieniane na typ właściwy na etapie kompilacji
- Zapewnia elastyczność języków typowanych dynamicznie
- Zapewnia silne typowanie
- Oszczędza pisania np. zamiast ***IEnumerable<string>*** (19 liter), ***var*** (3 litery)

Typy anonimowe

- Umożliwia tworzenie anonimowych obiektów

```
var imie = "Jan Nowak";
var numer = 42;
var e = 2.71628;
var osoba = new { imie = "Jan Kowalski", alias= "Jan Kowalski"};

string imie2 = "Jan Nowak";
int numer2 = 42;
double e = 2.71628;
```

Typy anonimowe

- Nazwa anonimowego typu jest automatycznie generowana przez kompilator w czasie kompilacji i nie możemy się do niej odwoływać w programie.
- Jeśli w programie podamy dwa inicjatory anonimowego obiektu, które mają takie same nazwy parametrów tego samego typu w tej samej kolejności, to będą to instancje tego samego typu.
- Przykład:

```
var p1 = new { Name = "Lawnmower", Price = 495.00 };
var p2 = new { Name = "Shovel", Price = 26.95 };
p1 = p2;  //OK ponieważ ten sam anonimowy typ
```

Automatyczne właściwości

```
class Pracownik
{
    private string imie;
    private int pensja;

    public string Imie { get { return imie; } set { imie = value; } }
    public string Nazwisko { get; set }
    public int Pensja
    {
        get { return pensja; }
        set
        {
                if(value<0)
                {
                        pensja = 1000;
                }
                else
                        pensja = value;
    }
}
```

Automatyczne właściwości

- ***get*** i ***set*** muszą być deklarowane w parze
- Kod generowany na etapie kompilacji
- W przypadku niestandardowego użycia trzeba deklarować gettery i settery „ręcznie”

Inicjowanie obiektów

```
static void Main(string[] args)
{
    Pracownik pracownik1 = new Pracownik();
    pracownik1.Imie = "Jan";
    pracownik1.Nazwisko = "Nowak";
    pracownik1.stanowisko = new Stanowisko();
    pracownik1. stanowisko.opis= "programista";
    pracownik1. stanowisko.pensja= 1200;

    Pracownik pracownik2 = new Pracownik();
    {
        Imie = "Jan";
        Nazwisko = "Nowak";
                        Stanowisko = new Stanowisko
                        {
                                Opis=„księgowy",
                                Pensja=2000
                        }
    };
}
```

Inicjowanie obiektów

- Należy podać nazwę atrybutu
- Mogą być zagnieżdżone
- Mogą inicjować kolekcje

Metody rozszerzające

- Pozwalają na rozszerzenie istniejących typów o nowe metody
- Metody rozszerzające są deklarowane przez słowo „this” w pierwszym parametrze metody (this string s)
- Metody rozszerzające możemy deklarować tylko w statycznych klasach
- Umożliwiają dostęp tylko do publicznych pól klasy rozszerzanej
- Zwiększa przejrzystość kodu, umożliwia bardziej obiektowe podejście

Metody rozszerzające

- Przykład

```
public static class Extensions
{
    public static int ToInt32 ( this string s)
    {
            return Int32.Parse(s);
    }
}
```

- Użycie metod rozszerzających

```
string s = 123;
int liczba = s.ToInt32() // to samo co Extensions.ToInt32(s)
```

Wyrażenia lambda

- Lambda wyrażenia jest to nowy sposób pisania anonimowych metod z wnioskowaniem typów oraz możliwością przekształcenia do typów delegowanych i drzewa wyrażeń.Trzeba zadeklarować ich typ
- Przykład

```
listOfFoo.Where(delegate(Foo x) { return x.size>10; } )
```

jest równoważne:

```
listOfFoo.Where (x => x.size>10 );
```

Wyrażenia lambda

- Przykłady

```
x => x + 1                      // domniemany typ, treść
                                // będąca wyrażeniem

x => { return x + 1; }          // typ wywnioskowany,
                                // deklaracja

(int x) => x + 1                // jawny typ, treść będąca
                                // wyrażeniem

(int x) => { return x + 1; }// jawny typ, deklaracja

(x, y) => x * y                 // kilka parametrów

() => Console.WriteLine()  // bez parametrów
```

Wyrażenia lambda

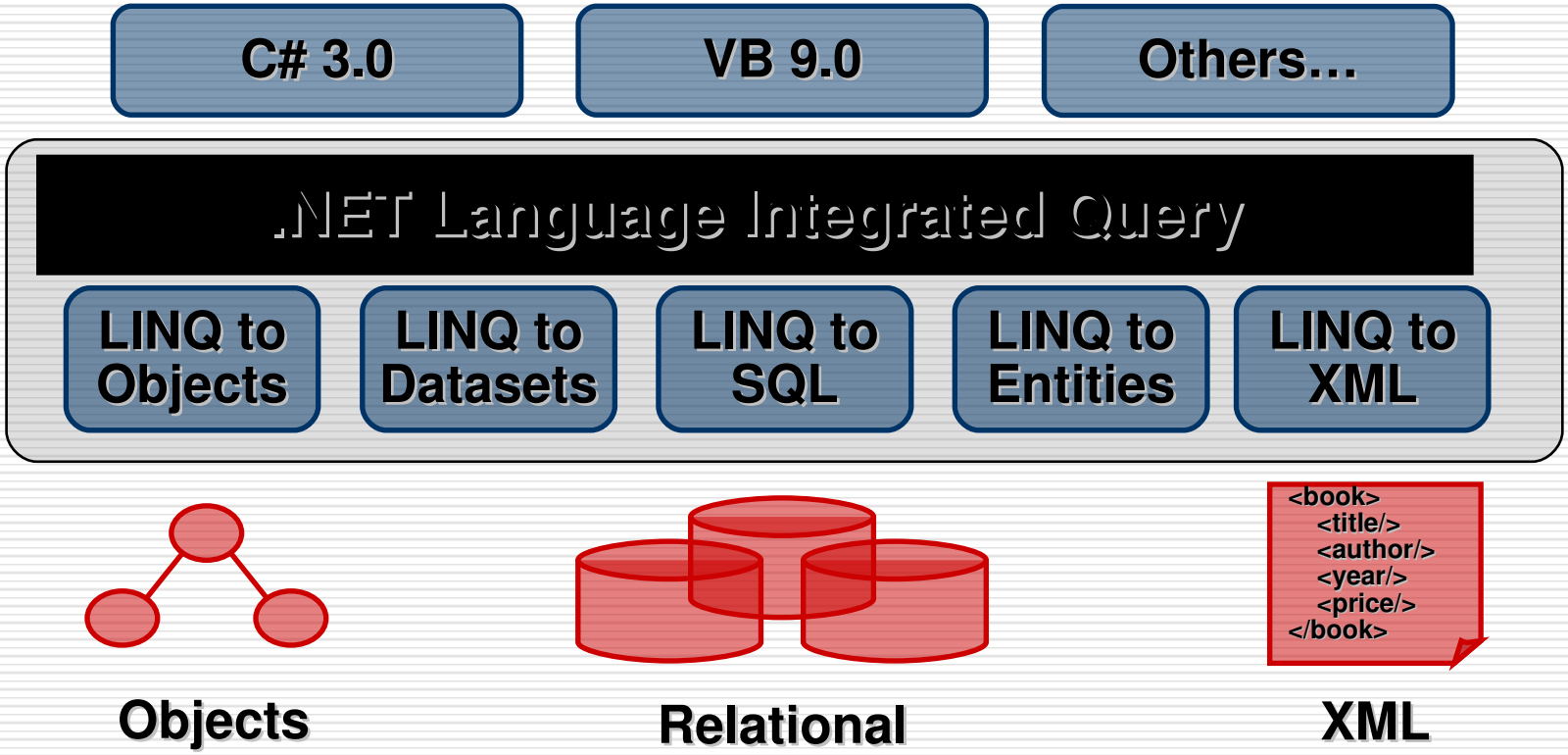
- Ułatwiają pisanie anonimowych funkcji
- Anonimowe metody wymagają podania typu, lambda wyrażenia nie
- Wyrażenie lambda może być deklaracją lub wyrażeniem zaś metoda anonimowa tylko deklaracją
- Składnia: (**argumenty**) => **wyrażenie**
- Maksymalnie 4 argumenty
- Bardzo użyteczne przy zapytaniach LINQ
- Można przekazać jako argument

Czym jest LINQ

- Wymowa „link”
- Część .NET Framework 3.5
- LINQ (ang. Language INtegrated Query czyli Zintegrowany język zapytań)
- Pozwala na wykonywanie zapytań do obiektów implementujących interfejs IEnumerable<T>
- Może działac także na dokumentach XML (XLINQ) oraz bazach danych SQL (DLINQ)
- Składnia zaczerpnięta z SQL

Czym jest LINQ

- LINQ to O/RM (Object Relational Mapping) czyli reprezentowanie danych przez obiekty języka programowania



Potrzeba

Pominięcie kodu przygotowawczego:

- Połączenie

```
SqlConnection connection = new SqlConnection("Data
   Source=MyServer;Initial Catalog=MyDatabase;Integrated
   Security=True");
```

- komenda

```
SqlCommand command = new SqlCommand("SELECT * FROM [dbo].[MyTable]”,
   connection);
```

Potrzeba

Poprawność kodu

- Sprawdzanie składni

```
SqlCommand command = new SqlCommand("SELEECT * FROM [dbo].[MyTable]");
```

- Mocna typowalność

```
SqlDataReader reader = command.ExecuteReader();

while (reader.Read())
{
   MyEntity entity = reader.
}
```

Potrzeba

Różne sposoby pracy na różnych danych

- XML

```
XmlDocument doc = new XmlDocument();
doc.Load(@" c:\Customers.xml");

XmlNodeList resultNodes = doc.SelectNodes("/customers/row/@CompanyName");
foreach (XmlNode aNode in resultNodes)
{ … }
```

- Relacyjne bazy danych

```
SqlConnection sqlConnection = …
```

Potrzeba

Podniesienie poziomu abstrakcji

- Różni dostawcy

```
SqlConnection sqlConnection = new SqlConnection();

MySqlConnection mySqlConnection = new MySqlConnection();
```

- Różna składnia

```
SqlCommand sqlCommand = new SqlCommand("SELECT TOP 10 * FROM [dbo].[MyTable]");

MySqlCommand mySqlCommand = new MySqlCommand("SELECT * FROM MyTable LIMIT 10");
```

Przykład

- Używając standardowych operatorów kwerend w C# 3.0:

```
using System;
using System.Query;
using System.Collections.Generic;
class app {
        static void Main() {
                string[] names = { "Burke", "Connor",
                        "Frank", "Everett", "Albert",
                        "George", "Harris", "David" };
                IEnumerable<string> expr = from s in names where
                                             s.Length == 5
                                           orderby s
                                           select s.ToUpper();
                foreach (string item in expr)
                        Console.WriteLine(item);
        }
}
```

Przykład

- Po przetworzeniu tablicy otrzymujemy:

```
BURKE DAVID FRANK
```

Uwagi

- Inicjalizacja zmiennej przez *query expression*
- Query expression przetwarza jedno lub wiele źródeł danych używając różnych *query operators*
- Wyrażenie w przykładzie używa operatorów:
  - Where
  - OrderBy
  - Select
- Wyrażenie napisane jest w *query syntax*

Uwagi

- Możliwe jest zapisanie wyrażenie w składni z „.”

```
IEnumerable<string> expr = names
                           .Where(s => s.Length == 5)
                           .OrderBy(s => s)
                           .Select(s => s.ToUpper());
```

- Argumenty operatorów nazywane są *lambda expressions*

Składnia

from *id* **in** *source*
{ **from** *id* **in** *source* | **where** *condition*}
[**orderby** *ordering*, *ordering*, …]
select *expr* | **group** *expr* **by** *key*
[**into** *id query*]

Operatory

filtrowanie	`Where`
projekcja	`Select, SelectMany`
kolejność	`OrderBy, ThenBy`
grupowanie	`GroupBy`
kwalifikatory	`Any, All`
partycje	`Take, Skip, TakeWhile, SkipWhile`
zbiory	`Distinct, Union, Intersect, Except`
elementy	`First, FirstOrDefault, ElementAt`
agregacja	`Count, Sum, Min, Max, Average`
konwersja	`ToArray, ToList, ToDictionary`
rzutowanie	`OfType<T>`

Integracja z SQL (DLinq)

- Możliwość tworzenia zapytań do relacyjnych źródeł
  - MS SQL 2005, 2000
  - MySql
  - PostgreSQL
  - Oracle
- W środowisku projektowym
- Atrybuty [Table] i [Column] definiują typy CLR odpowiadające definicjom SQL Schema

Integracja z SQL (DLinq)

- Przykładowa definicja SQL:

```
create table People (
    Name nvarchar(32) primary key not null,
    Age int not null,
    CanCode bit not null
)
create table Orders (
    OrderID nvarchar(32) primary key not null,
    Customer nvarchar(32) not null,
    Amount int
)
```

Integracja z SQL (DLinq)

- Jedyne czego potrzebujemy to:

```
[Table(Name="People")]
public class Person {
  [Column(DbType="nvarchar(32) not null", Id=true)]
  public string Name;
  [Column]
  public int Age;
  [Column]
  public bool CanCode;
}
[Table(Name="Orders")]
public class Order {
  [Column(DbType="nvarchar(32) not null", Id=true)]
  public string OrderID;
  [Column(DbType="nvarchar(32) not null")]
  public string Customer;
  [Column]
  public int? Amount;
}
```

Integracja z SQL (DLinq)

- Aby móc korzystać z mocy DLinq

```
// tworzymy zapytanie
var query =
    from c in custs
    from o in orders
    where o.Customer == c.Name
    select new {
        c.Name, o.OrderID, o.Amount, c.Age
    };
//  i wywolujemy
foreach (var item in query)
    Console.WriteLine("{0} {1} {2} {3}",
        item.Name, item.OrderID, item.Amount, item.Age);
```

Integracja z SQL (DLinq)

- Przy pomocy Visual Studio
- ***Project*** -> ***Add Class*** -> ***LINQ to SQL Classes***
- Z panelu ***Server Explorer*** przeciągamy Klasy z naszej bazy danych
- W kodzie tworzymy nowy obiekt *NazwaKontekstu****DataContext*** z ewentualnym parametrem połączenia

Wnioski

- Odmienne podejście od schematu mapowań O/R
- Projekt jest we wczesnej fazie rozwoju
- Nowe możliwości operowania na obiektach
- Łatwiejsze połączenie dwóch światów baz danych i obiektów
- Zalety
  - Sprawdzanie typów na etapie kompilacji
  - Zintegrowanie z językiem (IntelliSense, Debuger)
  - Klarowna i przejrzysta składnia
  - Może służyć jako język dedykowany (DSL)
- Wady
  - Szybkość działania

Literatura

- **The LINQ Project - http://msdn.microsoft.com/en-us/netframe**
- **101 LINQ Samples** -http://msdn.microsoft.com/en-us/vcsharp/aa336746.aspx