Platforma .NET – Wykład 7
Konwersje, pliki i kolekcje oraz dostęp do danych w .NET

Osoba prowadząca wykład, laboratorium i projekt:
dr hab. inż. Marek Sawerwain, prof. UZ

Instytut Sterowania i Systemów Informatycznych
Uniwersytet Zielonogórski

e-mail : M.Sawerwain@issi.uz.zgora.pl
tel. (praca) : 68 328 2321,
pok. 328a A-2,
ul. Prof. Z.Szafrana 2,
65-246 Zielona Góra

Ostatnia kompilacja pliku: Monday 5th June, 2023, t: 23:08

Notatki

Spis treści

Wprowadzenie
- Plan wykładu

Programowanie w C# – Część 3 i 1/2
- Operatory konwersji
- Pliki, strumienie
- Kolekcje

Dostęp do baz danych w .NET
- Architektura ADO.NET
- Model połączeniowy
- Zapytania
- Krótko o narzędziach we Visual Studio

Już za tydzień na wykładzie

Notatki

Plan wykładu – spotkania tydzień po tygodniu

(1) Informacje o wykładzie, pojęcie platformy, podstawowe informacje o platformie .NET
(2) Składowe platformy .NET: CLR, CTS, języki programowania, biblioteki klas, pojęcie podzespołu (ang. assembly)
(3) Programowanie w C# – środowisko VS, MonoDevelop, syntaktyka C#, wyjątki, współpraca z DLL
(4) Programowanie w C# – model obiektowy, typy uogólnione, lambda wyrażenia
(5) Programowanie w C# – aplikacje „okienkowe", programowanie wielowątkowe
(6) Programowanie w F# – podstawy, przetwarzanie danych tekstowych,
(*) "Klasówka I", czyli egzamin cześć pierwsza
(7) Dostęp do baz danych

Notatki

Plan wykładu – tydzień po tygodniu

(8) Język zapytań LINQ, Entity Framework
(9) Obsługa standardu XML
(10) Technologia ASP.NET 1/2
(11) Technologia ASP.NET 2/2
(12) Model widok i kontroler – Model View Controller
(13) Tworzenie usług sieciowych SOAP i WCF (komunikacja sieciowa)
(14) Wykład monograficzny .NET 1
(15) Wykład monograficzny .NET 2
(*) "Klasówka II", czyli egzamin cześć druga

Notatki

Plan wykładu

1. informacje o modelu obiektowy,
   1.1 jawne i niejawne operatory konwersji,
   1.2 pliki i strumienie,
   1.3 kolekcje,
2. dostęp do danych w .NET
   2.1 architektura ADO.NET
   2.2 modele programowania
   2.3 otwarcie połączenia
3. zapytania
   3.1 tworzenie zapytań
   3.2 odczytanie danych
   3.3 krótko o narzędziach dostępnych w Visual Studio

Notatki

Konwersje w modelu obiektowym C#

Język C# wprowadza operatory konwersji pozwalające na konwersję pomiędzy klasami i strukturami. Dostępne są także słowa kluczowe explicit (jawna konwersja, wymagana kiedy konwersja występuje dla typu o większej dziedzinie) oraz **implicit** (niejawna konwersja, może zostać wykonana samodzielnie dla typu o mniejszej dziedzinie, który jest promowany do typu o szerszej dziedzinie):

```
int a = 123;
long b = a;      // niejawna konwersja z int do typu long
int c = (int) b; // jawna konwersja z long do typu int
```

Operatory konwersji posiadają następujące własności:

1. konwersje deklarowane ze słowem **implicit** są realizowane samodzielnie przez kompilator,
2. konwersje deklarowane ze słowem **explicit** muszą zostać wywołane jawnie z operatorem zrzutowania,
3. wszystkie konwersje muszą być deklarowane jako statyczne.

Notatki

Przykład operatora konwersji

Ogólny schemat operatora konwersji:

```
class NazwaKlasy {
    public static explicit operator NazwaKlasy(TYP i) {
        NazwaKlasy t = new NazwaKlasy();

        // zamiana wartości i o typie TYP na
        // typ NazwaKlasy

        return t;
    }
}
```

Przykład operatora jawnej konwersji obiektu prostokąt na obiekt kwadrat:

```
public static explicit operator Square(Rectangle r) {
    Square s = new Square();
    s.Length = r.Height;
    return s;
}
```


Notatki

Konwersje „explicit” i „implicit”

Struktura dla cyfr

```
struct Digit {
    byte value;
    public Digit(byte value) {
        if (value > 9) throw new System.ArgumentException();
        this.value = value;
    }
    public static explicit operator Digit(byte b) {
        Digit d = new Digit(b);
        return d;
    }
    public static implicit operator byte(Digit d) {
        return d.value;
    }
}
```

Jawna konwersja zawężająca typ:

```
byte b = 3; Digit d = (Digit)b;
```


Konwersja niejawna, niebezpieczeństwo utraty danych nie występuje:

Notatki

Pliki, strumienie

Klasy zawarte w Base Class Library oferują wsparcie dla plików oraz strumieni, najważniejsze nie-abstrakcyjne klasy są następujące:

Nazwa klasy	Krótki opis
BinaryReader, BinaryWriter	Klasy dostępu do danych prymitywnych w plikach
BufferedStream	Dostęp buforowany
Directory, DirectoryInfo	Obsługa katalogów
DriveInfo	Informacje o urządzeniu dyskowym
File, FileInfo	Obsługa plików
FileStream	Obsługa plików z dostępem swobodnym
FileSystemWatcher	Monitorowanie zmian we wskazanym pliku
MemoryStream	Strumień o dostępie swobodnym umieszczony w pamięci operacyjnej
Path	Dostęp do pliku/katalogu, w postaci niezależnej od platformy
StreamWriter, StreamReader	strukturalny dostęp do danych
StringWriter, StringReader	strukturalny dostęp do danych zapisanych w ciągu znaków

Notatki

Przykład odczytu informacji o plikach – 1/2

Odczytanie listy plików o rozszerzeniu .jpg ze wskazanego katalogu:

```
using System.IO;

DirectoryInfo dir = new DirectoryInfo(@"C:\Windows\Web\Wallpaper");
FileInfo[] imageFiles = dir.GetFiles("*.jpg", SearchOption.AllDirectories);

Console.WriteLine("Found {0} *.jpg files\n", imageFiles.Length);
foreach (FileInfo f in imageFiles) {
    ...
    Console.WriteLine("File name: {0}", f.Name);
    ...
}
```


Notatki

Przykład odczytu informacji o plikach – 2/2

Tworzenie i kasowanie katalogu:

```
DirectoryInfo dir = new DirectoryInfo(".");
dir.CreateSubdirectory("MyFolder");
DirectoryInfo myDataFolder = dir.CreateSubdirectory(@"MyFolder2\Data");
...
try  {
    Directory.Delete(@"MyFolder");
    Directory.Delete(@"MyFolder2", true);
}
catch (IOException e) {
    Console.WriteLine(e.Message);
}
```


Notatki

Obsługa plików

Łatwy zapis i odczyt zawartości tablicy

```
string[] DataTable = {
    "elem1", "elem2", "elem3", "elem4"};
File.WriteAllLines( @"file.txt", DataTable );
....
foreach (string task in File.ReadAllLines(@"file.txt")) {
    Console.WriteLine("{0}", task);
}
```

Podstawowe metody i własności w abstrakcyjnej klasie Stream:

- CanRead, CanWrite, CanSeek,
- Close(), Flush(), Length,
- Postion, Seek(),
- Read(), ReadByte(), Write(), WriteBytes().

Najważniejsze metody i własności abstrakcyjnych klas bazowych TextWriter, TextReader, BinaryWriter, BinaryReader:

- Peek(), PeekChar()
- ReadBlock(), ReadLine(), ReadToEnd()
- WriteLine(), NewLine
- ReadXXXX(), gdzie XXXX reprezentuje nazwę typu np.: Int32

Notatki

Proste przykłady

Zapis danych do pliku binarnego:

```
FileInfo f = new FileInfo("BinFile.dat");
using(BinaryWriter bw = new BinaryWriter(f.OpenWrite())) {
    double aDouble = 1234.67; int anInt = 34567;
    string aString = "A,B,C";

    bw.Write(aDouble);
    bw.Write(anInt);
    bw.Write(aString);
}
```

Monitorowanie katalogu:

```
FileSystemWatcher watcher = new FileSystemWatcher();
watcher.Path = @".\katalog";
watcher.NotifyFilter = NotifyFilters.LastAccess
 | NotifyFilters.LastWrite
 | NotifyFilters.FileName | NotifyFilters.DirectoryName;

watcher.Changed += new FileSystemEventHandler(OnChanged);
watcher.Created += new FileSystemEventHandler(OnChanged);
watcher.Deleted += new FileSystemEventHandler(OnChanged);
watcher.Renamed += new RenamedEventHandler(OnRenamed);
```


Notatki

Kolekcje

Co warto wiedzieć o typach kolekcji w C#:

- kolekcje są reprezentowane przez klasy zdefiniowane w przestrzeniach System.Collections (tzw. typy wyspecjalizowane) oraz System.Collections.Generic (typu ogólne bez specjalizacji),
- większość kolekcji jest oparta o interfejsy ICollection, IComparer, IEnumerable, IList, IDictionary, IDictionaryEnumerator oraz uogólnioną odmianę tych interfejsów,
- uogólnione kolekcje charakteryzują się podniesionym bezpieczeństwem typów oraz w niektórych przypadkach wyższą wydajnością, w szczególności jeśli przechowywane są typy wartościowe.

Stosowanie kolekcji uogólnionych pozwala na szybsze implementowanie potrzebnej funkcjonalności, bowiem nie jest konieczne tworzenie nowej klasy z klasy bazowej dla danej kolekcji i implementacja odpowiednich wersji metod dla stosowanego typu. Wydajność typów uogólnionych również jest wyższa niż w przypadku odpowiedników wyspecjalizowanych ze względu na typy oparte o wartości ze względu na to że nie są tworzone pomocnicze obiekty za pomocą techniki „pudełkowania".

Notatki

Dostępne typy kolekcji

Poniższe typy są odpowiednikami istniejących kolekcji:

- `List <T>` – klasa ogólna które jest odpowiednikiem klasy ArrayList,
- `Dictionary <TKey, TValue>`, `ConcurrentDictionary <TKey, TValue>` – klasy są odpowiednikami typu Hashtable.
- `Collection <T>` – to klasa uogólniona odnoszącą się do klasy CollectionBase, klasa `Collection <T>` może być użyta w roli klasy bazowej, choć nie zawiera metod abstrakcyjnych,
- `ReadOnlyCollection <T>` – to klasa będąca odpowiednikiem ReadOnlyCollectionBase, klasa `ReadOnlyCollection <T>` nie jest klasą abstrakcyjną, posiada także konstruktor co powoduje iż jest łatwiejsza w stosowaniu z połączeniu z `List <T>` w roli kolekcji tylko do odczytu,
- `Queue <T>`, `ConcurrentQueue <T>`, `Stack <T>`, `ConcurrentStack<T>`, `SortedList <TKey, TValue>` – wymieniowe klasy są bezpośrednimi odpowiednikami klas specjalizowanych.

Uwaga

W programach tworzonych przy użyciu wersji 4.0 platformy .NET, należy koniecznie stosować typy z przestrzeni **System.Collections.Concurrent**, jeśli dana kolekcja jest wykorzystywana przez wiele różnych wątków (operacje dodawania i usuwania elementów).

Notatki

Szczególne typy kolekcji

Istnieją typy ogólne nie posiadające odpowiedników w typach wyspecjalizowanych. Należą do nich miedzy innymi:

- `LinkedList <T>` – klasa ogólnego przeznaczenia, operacja dodawania i usuwania jest realizowana przy użyciu O(1) operacji,
- `SortedDictionary <TKey, TValue>` – uporządkowany słownik, operacje dodawania, usuwania są realizowane przy użyciu O(log n) operacji, (jest to lepsze rozwiązanie niż posortowana lista: `SortedList <TKey, TValue>`),
- `KeyedCollection <TKey, TItem>` – typ stanowiący hybrydę typów słownikowych oraz listowych, każdy element jest opatrzony kluczem,
- `BlockingCollection <T>` – typ dostarczający dostęp do elementów z funkcjonalnością blokowania oraz ograniczania np.: liczby przechowywanych elementów,
- `ConcurrentBag <T>` – typ pozwala na szybkie usuwanie oraz dodawanie elementów bez określonego porządku.

Notatki

Kolekcja z łańcucha znaków

Przykład implementacji

```
Tokens f = new Tokens("Zdanie rozdzielone spacjami oraz myślnikami.",
                                                    new char[] {' ','-'});
foreach (string item in f) {
    System.Console.WriteLine(item);
}
```

Utworzenie tablicy elementów oraz utworzenie obiektu „enumeratora”:

```
Tokens(string source, char[] delimiters) {
    elements = source.Split(delimiters);
}

public IEnumerator GetEnumerator() {
    return new TokenEnumerator(this);
}
```


Notatki

Kolekcja z łańcucha znaków

Klasa enumeratora:

```
private class TokenEnumerator : IEnumerator {

    private int position = -1; private Tokens t;

    public TokenEnumerator(Tokens t) { this.t = t; }

    public bool MoveNext() {
        if (position < t.elements.Length - 1) {
            position++;
            return true;
        } else {
            return false;
        }
    }
    public void Reset() {
        position = -1;
    }
    public object Current {
        get {
            return t.elements[position];
        }
    }
}
```


Notatki

Równoległość w kolekcjach

Przykład wykorzystując współbieżny stos:

```
using System;
using System.Collections.Concurrent;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;

        static void Main () {
            int errorCount = 0;
            ConcurrentStack<int> cs = new ConcurrentStack<int>();

            cs.PushRange(new int[] { 1, 2, 3, 4, 5, 6, 7 });
            cs.PushRange(new int[] { 8, 9, 10 });
            cs.PushRange(new int[] { 11, 12, 13, 14 });
            cs.PushRange(new int[] { 15, 16, 17, 18, 19, 20 });
            cs.PushRange(new int[] { 21, 22 });
            cs.PushRange(new int[] { 23, 24, 25, 26, 27, 28, 29, 30 });
            ...
        }
}
```


Notatki

Równoległość w kolekcjach

Odczyt danych ze współbieżnego stosu:

```
Parallel.For(0, 10, i =>
  {
    int[] range = new int[3];
    if (cs.TryPopRange(range) != 3) {
        Console.WriteLine("Błąd w TryPopRange");
        Interlocked.Increment(ref errorCount);
    }

    if (!range.Skip(1).SequenceEqual(
                range.Take(range.Length - 1).Select(x => x - 1))) {
        Console.WriteLine("Błąd zakresu: range[0]={0}, range[1]={1}",
                range[0], range[1]);
        Interlocked.Increment(ref errorCount);
                }
  });
```


Notatki

Serwer baz danych

Platforma .NET jest zdolna do współpracy z dowolną współczesną relacyjną bazą danych, min. z następującymi bazami:

- Oracle DB,
- IBM DB2,
- MS SQL Server.

Możliwa jest również współpraca z darmowymi bazami danych jak MySQL, PostgreSQL, Firebird, SQLite. Jednakże, najbardziej posunięta integracja została wykonana dla bazy danych Microsoft SQL Server (Visual Studio współpracuje bezpośrednio z tą bazą, a także z bazą Oracle). Obecnie dostępne są trzy główne rodziny tego systemu baz danych:

1. Datacenter, Enterprise, Standard,
2. Developer, Workgroup, Web,
3. Compact, Express.

Notatki

Architektura ADO.NET

Technologia ADO.NET obsługuje warstwę dostępu do danych:

- obiekty DataSet, Connection, Command, DataAdapter i ich współdziałanie,
- praca w trybie połączonym DataReader,
- praca w trybie odłączonym: DataSet (buforowanie zmian w lokalnej bazie),
- parametryzowanie źródła danych,
- obsługa procedur „składowanych" w bazie danych,
- obsługa transakcji,
- wsparcie dla XML.

Ważne elementy to LINQ, ADO.NET Entity Framework, WCF Data Services (dawniej ADO.NET Data Services), XML oraz ADO.NET.

Model programowania

Dwa podstawowe modele: połączeniowy (connected) – oparty o aktywne połączenia do systemu bazy danych, bezpołączeniowy/rozłączony (disconnected) albo pytanie/odpowiedź, bez wyróżnionych stanów między zapytaniami.

Notatki

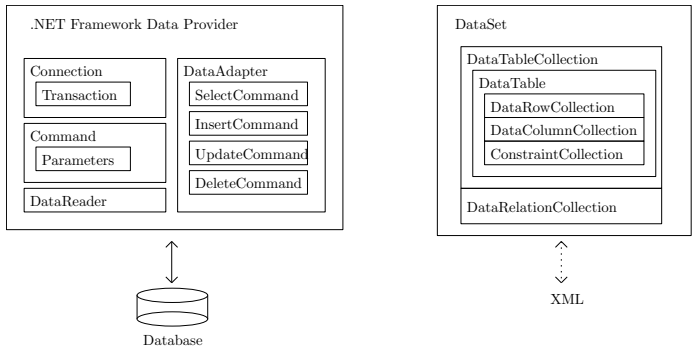
Technologia ADO.NET Entity Framework

Technologia ADO.NET Entity Framework ułatwia tworzenie dostępu do danych poprzez udostępnianie narzędzi do tworzenia modelu aplikacji/danych zamiast bezpośredniego tworzenia relacyjnej bazy danych. Głównym celem jest zmniejszenie ilości kodu jaki należy utrzymywać dla aplikacji korzystającej z dostępu do bazy danych. Aplikacje bazujące na „Entity Framework" zazwyczaj oferują kilka własności przedstawionych poniżej:

- aplikacje mogą pracować w kontekście modelu koncepcyjnego, co oznacza iż ważniejsze są typy, dziedziczenie, złożoność typów oraz relacje pomiędzy typami,
- nie istnieje potrzeba ręcznego kodowania zależności pomiędzy silnikiem baz danych a postacią danych,
- odwzorowanie pomiędzy modelem koncepcyjnym a postacią danych może być modyfikowane bez zmiany kodu aplikacji,
- możliwa jest praca ze spójnym modelem obiektowym który odwzorowuje różne schematy danych implementowane w różnych systemach baz danych,
- wiele modeli koncepcyjnych może być odwzorowanych w jednym schemacie danych,
- zintegrowany język zapytań (LINQ) wspiera weryfikację poprawności zapytań z modelem koncepcyjnym.

Notatki

Architektura ADO.NET



Zadanie obiektu typu „Data Provider"

Zadaniem obiektu dostarczającego dane jest zapewnienie odpowiedniego zbioru klas, do połączenia z bazą danych i utrzymywania połączenia oraz dostarczanie i uaktualniania danych znajdujących się w bazie danych.

Notatki

Dostępne „sterowniki" dostępu do danych w .NET

Podzespoły dostępny bezpośrednio w platformie .NET:

Nazwa	Przestrzeń nazw	Podzespół
OLE DB	System.Data.OleDb	System.Data.dll
Microsoft SQL Server	System.Data.SqlClient	System.Data.dll
Microsoft SQL Server Mobile	System.Data.SqlServerCe	System.Data.SqlServerCe.dll
ODBC	System.Data.Odbc	System.Data.dll

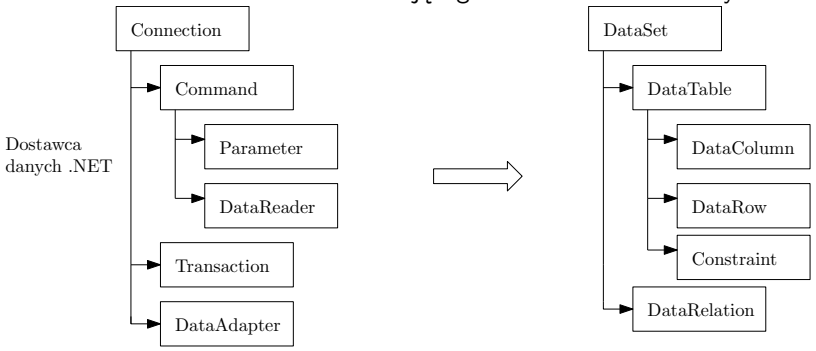
Od wersji 4.0 .NET sterownik dostępu do bazy danych Oracle dostarczany przez firmę Microsoft, został określony jako przestarzały, jednakże firma Oracle, dostarcza własny sterownik dostępu do danych w ramach platformy .NET.

Na stronie `http://www.sqlsummit.com/DataProv.htm` znajdują się adresy wielu innych producentów dostarczających sterowniki dostępu do systemów baz danych. Dostępne są także sterowniki do baz danych OpenSource jak MySQL, Firebird, PostgreSQL oraz SQLite.

Notatki

Najważniejsze obiekty w ADO.NET

Schemat hierarchii obiektu dostarczającego dane oraz zbioru danych:



Notatki

Najważniejsze obiekty w ADO.NET

Typ obiektu	Klasa bazowa	Interfejsy	Opis
Connection	DbConnection	IDbConnection	Obiekt/Klasa odpowiedzialny za nawiązanie połączenia z systemem bazy danych. Obiekty tego typu oferują także wsparcie do zarządzania transakcjami.
Command	DbCommand	IDbCommand	Reprezentuje zapytania SQL oraz procedury wbudowane, udostępniany jest także obiekt DataReader.
DataReader	DbDataReader	IDataReader, IDataRecord	Obiekt dostarcza jednokierunkowy i wyłącznie do odczytu dostęp do danych z kursorem po stronie serwera.
DataAdapter	DbDataAdapter	IDataAdapter, IDbDataAdapter	Odpowiedzialny za transfer danych pomiędzy bazą a aplikacją, udostępnia także mechanizmy współpracy z podstawowymi typami zapytań jak zapytania typu: select, insert, update, delete.
Parameter	DbParameter	IDataParameter, IDbDataParameter	Obiekt stosowany do reprezentacji parametru w trakcie budowy zapytania z parametrem.
Transaction	DbTransaction	IDbTransaction	Obiekt reprezentujący tranzakcję.

Notatki

Hierarchia klasa ADO.NET

Interfejsy:

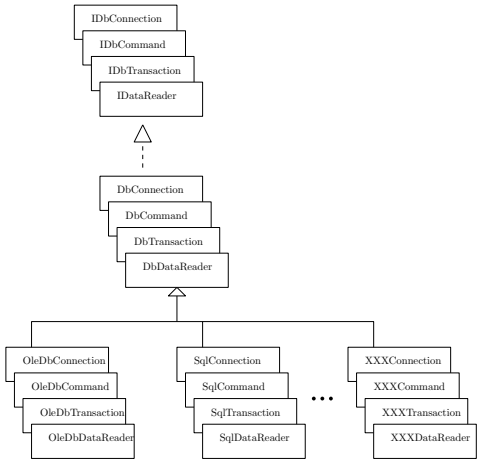
- IDbConnection
- IDbCommand
- IDbTransaction
- IDataReader

Abstrakcyjne klasy bazowe:

- DbConnection
- DbCommand
- DbTransaction
- DbDataReader

Implementacje sterowników:

- OleDB
- SQL
- ...
- inny sterownik.



Notatki

Modele programowania ADO.NET

ADO.NET wspiera dwa różnych modele programowania w kontekście dostępu do danych:

1. model połączeniowy,
   - stosuje się głównie obiekty typu Command oraz DataReader,
   - do odczytu danych stosowany jest DataReader,
   - aktualizacje danych są realizowane przez obiekt Command i odpowiednio przygotowane zapytania,
2. model bezpołączeniowy
   - stosowane są obiekty typu DataSet,
   - Obiekty DataAdapter zarządzają zawartością DataSet, realizują odczyt i zapis danych,
   - obiekty DataSet są niezależne od obiektu dostarczające dane,
   - obiektu DataSet mogą zawierać i przetwarzać dane w postaci XML,
   - obiekt DataSet jest zgodny z RDOM (Relational Document Object Model) dla XML.

Notatki

Nawiązanie połączenia

W .NET istnieje pojecie połączeniowego ciągu znaków, reprezentującego podstawowe informacje o położeniu serwera danych:

```
string strConn = "data source=localhost; " +
    "initial catalog=northwind; integrated security=true";
SqlConnection conn = new SqlConnection(strConn);
```

Platforma .NET oferuje obiekty z rodziny ConnectionStringBuilder ułatwiające tworzenie ciągu połączeniowego:

```
System.Data.SqlClient.SqlConnectionStringBuilder builder =
     new System.Data.SqlClient.SqlConnectionStringBuilder();
          builder["Data Source"] = "(local)";
          builder["integrated Security"] = true;
 builder["Connect Timeout"] = 1000;
          builder["Initial Catalog"] = "MyDataBase";
Console.WriteLine(builder.ConnectionString);
```


Notatki

Zestawienie podstawowych parametrów

Niektóre z parametrów stosowanych w ciągu znaków reprezentującego połączenie:

- Server=nazwa – nazwa serwera danych,
- Connection timeout: dopuszczalny czas uzyskania połączenia,
- Data source=nazwa – nazwa instancji bazy danych SQL Server lub nazwa komputera,
- Initial catalog=nazwa – nazwa bazy danych,
- Integrated security=boolean – gdy podano wartość True połączenie z SQL serwerem na podstawie tożsamości konta utworzonego na maszynie na której uruchomiono serwer,
- User ID=nazwa – nazwa użytkownika,
- Password=hasło – postać hasła .

Notatki

Błędy w połączeniach

Możliwe błędy jakie mogą pojawić się podczas nawiązywania połączenia to min.:

- ciąg opisujący połączenie zawiera błędy,
- nie można odszukać serwera bądź bazy danych,
- logowanie nie udało się,

Inne błędy związane z obsługą danych to np.:

- błąd syntaktyczny w zapytaniu SQL,
- zła nazwa tabeli,
- niepoprawna nazwa pola.

Notatki

Przykładowa obsługa błędów

Obsługa błędów w trakcie nawiązywania połączenia z bazą danych:

```
try {
    SqlConnection conn = new SqlConnection( "...." );
    SqlDataAdapter da = new SqlDataAdapter( "....", conn );
    DataSet ds = new DataSet();
    da.Fill(ds);
}
catch (System.Data.SqlClient.SqlException e) {
    for (int i = 0; i < e.Errors.Count; i++) {
        // ...
    }
}
catch (System.Exception e) {
// inne błędy
}
```


Notatki

Zamykanie niepotrzebnych połączeń – 1/2

Należy zadbać o zamykanie połączenia z bazą danych w momencie, gdy nie jest ono już potrzebne:

```
try {
    ...
    connection.Open();
    ...
}
catch ( SqlException ex ) {
    ...
}
finally {
    ...
    connection.Close ( ) ;
}
```


Notatki

Zamykanie niepotrzebnych połączeń – 2/2

Wykorzystanie słowa kluczowego using:

```
try {
    using (SqlConnection conn = new SqlConnection(source)) {
        ...
        connection.Open();
        ...
    }
}
catch ( SqlException ex ) {
    ...
}
```


Notatki

Tworzenie puli połączeń

Pula połączeń jest mechanizmem zarządzania aktywnymi połączeniami. Pula połączeń jest wykorzystywana w przypadku połączeń OLE DB oraz SQL Server pozwala na ponowne wykorzystanie połączeń w zależności od kontekstu użytkownika oraz kontekstu bezpieczeństwa. Co oznacza iż pula połączeń przyczynia się do zwiększenia wydajności, skalowalności i bezpieczeństwa aplikacji. Niektóre parametry dla puli połączeń są następujące:

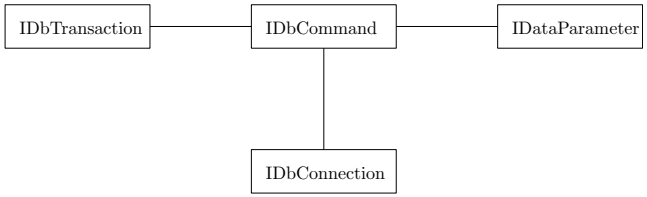
- Connection Lifetime – czas podtrzymywanie połączenia (zero maksymalny czas),
- Connection Reset – określa czy połączenie jest resetowane kiedy jest usuwanie z puli,
- Max Pool Size – maksymalna ilość połączeń w puli,
- Min Pool Size – minimalna ilość połączeń w puli,
- Pooling – True/False – utrzymywanie puli,

```
cnNorthwind.ConnectionString = "Integrated Security=True;" +
    "Initial Catalog=Northwind;" +
    "Data Source=London;" +
    "Pooling=True;" +
    "Min Pool Size=5;" +
    "Connection Lifetime=120;";
```


Notatki

Zapytania

W modelu połączeniowym głównym obiektem reprezentującym zapytania są obiekty typu Command.



Ogólnie o obiekcie Command:

- obiekty Command zawierają wyrażenia SQL lub odnoszą się do procedur wbudowanych,
- wymagają obiektu reprezentującego połączenie,
- mogą posiadać parametry,
- mogą wykorzystywać transakcje.

Notatki

Metody obiektu Command

W przypadku posługiwaniu się obiektami Command mogą zachodzić następujące sytuacje:

1. zapytanie nie zwraca wierszy,
   - wywołanie metody ExecuteNonQuery, zwraca ilość zmodyfikowanych wierszy,
   - zarządzenie bazą danych, wyrażenia DDL, DCL takie jak CREATE, ALTER, DROP, GRANT, DENY, REVOKE
   - modyfikacja danych w bazie danych, INSERT, UPDATE, DELETE
2. zapytanie zwraca pojedynczą wielkość,
   - wywołanie metody ExecuteScalar, metoda ta zwraca wielkość typu Object,
3. zapytanie zwraca wiersze
   - wywołanie mteody ExecuteReader, jako rezultat otrzymuje się obiekt typu DataReader (obiekt tylko do ukierunkowanego odczytu strumienia wierszy),
4. obiekt Commmand zwraca obiekt typu XmlReader
   - ExecuteXmlReader, dostępne tylko dla serwer SQL firmy Microsoft.

Notatki

Interface IDbCommand

Interfejs opisujący zawartość obiektu reprezentującego polecenie do wykonania:

```
public interface IDbCommand : IDisposable {
    void Cancel();
    IDbDataParameter CreateParameter();
    int ExecuteNonQuery();
    IDataReader ExecuteReader();
    IDataReader ExecuteReader(CommandBehavior behavior);
    object ExecuteScalar();
    void Prepare();
    string CommandText                  {get; set;}
    int CommandTimeout                  {get; set;}
    CommandType CommandType             {get; set;}
    IDbConnection Connection            {get; set;}
    IDataParameterCollection Parameters {get; }
    IDbTransaction Transaction          {get; set;}
    UpdateRowSource UpdatedRowSource    {get; set;}
}
```

Ważne elementy Connection, CommandType (Text, StoredProcedure), CommandText (Text, StoredProcedure), Parameters (parametry dla wyrażeń SQL i procedur wbudowanych)

Notatki

Zapytanie bez rezultatu

Usunięcie osoby o wskazanym indeksie:

```
string sqlConnectString = "...." ;
string sqlDelete = "DELETE FROM ExecuteQueryNoResultSet WHERE Id = 2";

SqlConnection connection =
    new SqlConnection(sqlConnectString);

SqlCommand command = new SqlCommand(sqlDelete, connection);
connection.Open( );
...
int rowsAffected = command.ExecuteNonQuery( );
...
connection.Close( );
```


Notatki

Przykład liczba osób

Nawiązanie połączenia i odczytanie liczby osób w tabeli osoby:

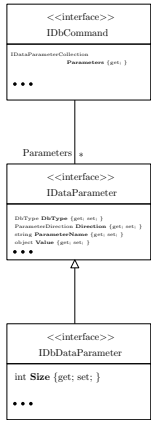
```
SqlConnection con = new SqlConnection(
"Server=localhost; Database=Pubs; Integrated Security=SSPI" );

SqlCommand cmd = new SqlCommand(
"SELECT COUNT( * ) FROM Persons", con );

con.Open();

Console.WriteLine( cmd.ExecuteScalar() );

con.Close();
```

Notatki

Zapytania parametryzowane

Polecenie/Zapytanie SQL może zostać wyposarzone w dodatkowe parametry

- parametry są zapisane we własności IDataParameterCollection Parameters get;,
- obiekt parametru zawiera min.:
  - nazwę
  - wartość
  - typ
  - kierunek (input, output, InputOutput, ReturnValue).



Notatki

Zapytania parametryzowane

Usuwanie rekordu o wskazanym ID:

```
SqlCommand cmd = new SqlCommand();
cmd.CommandText = "DELETE FROM StudenciTBL WHERE StudentID = @SID";
```

Określenie typu parametru:

```
cmd.Parameters.Add( new SqlParameter("@SID", SqlDbType.BigInt));
```

Ustalenie wartości parametru oraz wykonanie polecenia SQL:

```
cmd.Parameters["@SID"].Value = 1000;
cmd.ExecuteNonQuery();
```


Notatki

Wywołanie procedury wbudowanej

Procedura wbudowana odczytująca imię pupila domowego:

```
GetPetName
@carID int,
@petName char(10) output
AS
SELECT @petName = PetName from Inventory where CarID = @carID
```

Obsługa procedury po stronie języka C#:

```
public string LookUpPetName(int carID) {
string carPetName = string.Empty;

using (SqlCommand cmd = new SqlCommand("GetPetName", this.sqlCn)) {
cmd.CommandType = CommandType.StoredProcedure;

// określenie parametrów
      ...

cmd.ExecuteNonQuery();

// Return output param.
carPetName = (string)cmd.Parameters["@petName"].Value;
}
```


Notatki

Określenie parametrów

Określenie parametrów, w tym typu parametru, czy jest to parametr wejściowy czy też wyjściowy:

```
SqlParameter param = new SqlParameter();
param.ParameterName = "@carID";
param.SqlDbType = SqlDbType.Int;
param.Value = carID;
param.Direction = ParameterDirection.Input;
cmd.Parameters.Add(param);

param = new SqlParameter();
param.ParameterName = "@petName";
param.SqlDbType = SqlDbType.Char;
param.Size = 10;
param.Direction = ParameterDirection.Output;
cmd.Parameters.Add(param);
```


Notatki

Odczyt danych z obiektu DataReader

Klasa DataReader reprezentuje zbiór danych powstały w wyniku realizacji zapytania, własności obiektów typu DataReader są następujące:

- odczyt tylko do odczytu i odczyt jednokierunkowy,
- szybki dostęp do danych
- połączenie z bazą danych
- samodzielnie zarządza połączeniem,
- łatwe zarządzenie otrzymanym zbiorem danych.

```
public interface IDataReader  : IDisposable, IDataRecord {
    void Close();
    DataTable GetSchemaTable();
    bool NextResult();
    bool Read();
    int Depth{get;}
    bool IsClosed{get;}
    int RecordsAffected{get;}
}
```

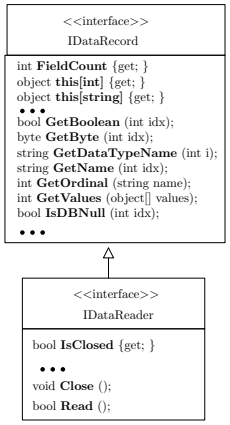
Wszystkie kroki w obsłudze DataReader'a są następujące: utworzenie i otwarcie połączenia do bazy danych, utworzenie obiektu Command, utworzenie obiektu DataReader za pomocą wywołania metody ExecuteReader, przetwarzanie otrzymanych danych, zamknięcie obiektu DataReader, zamknięcie połączenia z bazą danych.

Notatki

Interfejs IDataReader

Interfejs DataReader, czyli obsługa

- odczytanie następnego wiersza: bool Read();,
- dostęp do wartości w kolumnach przy zastosowaniu indekserów: object this[int]get;, object this[string]get;,
- dostęp do kolumn w zależności od typu: bool GetBoolean(int idx);, byte GetByte(int idx);,
- dodatkowe informacje jak np.: nazwy kolumn: string GetDataTypeName(int i);, string GetName(int idx);, int GetOrdinal(string name);.



Notatki

Przykład z DataReader

Odczytanie listy osób:

```
using(SqlDataReader myDataReader = myCommand.ExecuteReader()) {
    while (myDataReader.Read()) {
        Console.WriteLine("-> Imię: {0}, Nazwisko: {1}, Wiek: {2}.",
            myDataReader["FirstName"].ToString(),
            myDataReader["Surname"].ToString(),
            myDataReader["Age"].ToString());
    }
}
```

Podobnie jak wyżej ale bez jawnego podawania nazw pól:

```
while (myDataReader.Read()) {
    for (int i = 0; i < myDataReader.FieldCount; i++) {
        Console.WriteLine("{0} = {1} ",
            myDataReader.GetName(i),
            myDataReader.GetValue(i).ToString());
    }
}
```


Notatki

Zbiór rezultatów

Odczyt zbioru rezulatatów dwóch zapytań typu select:

```
SqlCommand command = new SqlCommand(
        "SELECT CategoryID, CategoryName FROM Categories;" +
        "SELECT EmployeeID, LastName FROM Employees",
    connection);

connection.Open();

SqlDataReader reader = command.ExecuteReader();
while (reader.HasRows) {
    Console.WriteLine("\t{0}\t{1}", reader.GetName(0),
        reader.GetName(1));

    while (reader.Read()) {
        Console.WriteLine("\t{0}\t{1}", reader.GetInt32(0),
            reader.GetString(1));
    }
    reader.NextResult();
}
```


Notatki

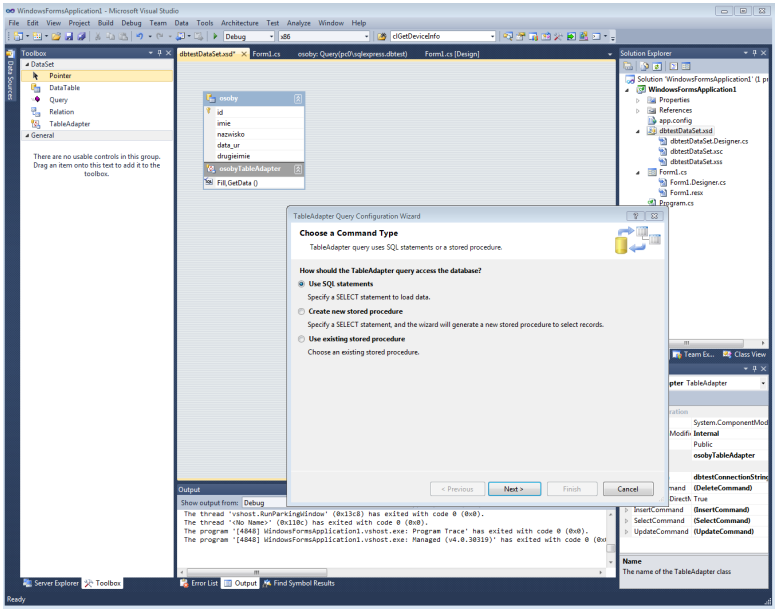
Kontrolki ekranowe

Najważniejsze kontrolki do obsługi baz danych to

- Chart – obsługa wykresów,
- BindingNavigator – interfejs użytkownika do np.: nawigacji w tabeli,
- BindingSource – reprezentacja danych po stronie źródła danych,
- DataGridView – kontrolka ekranowa do wizualizacji danych,
- DataSet – reprezentacja danych po stronie klienta.

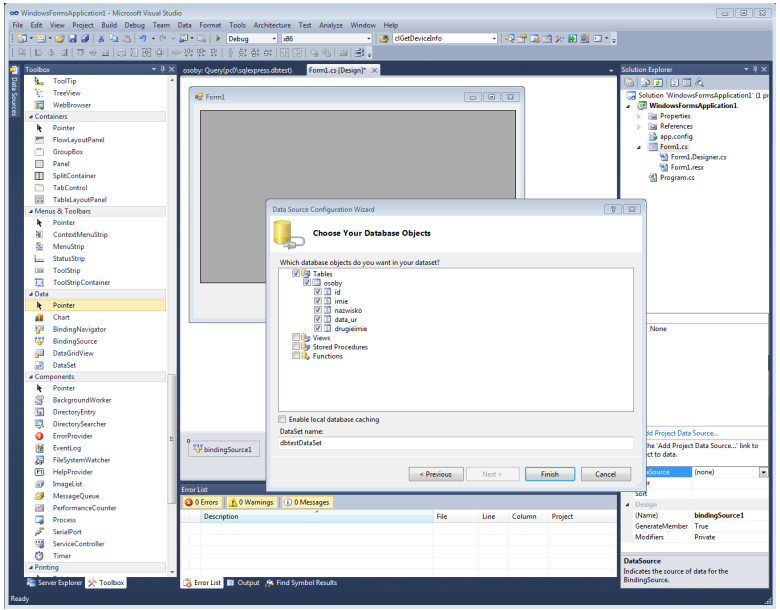
Notatki

Narzędzia dostępne w Visual Studio



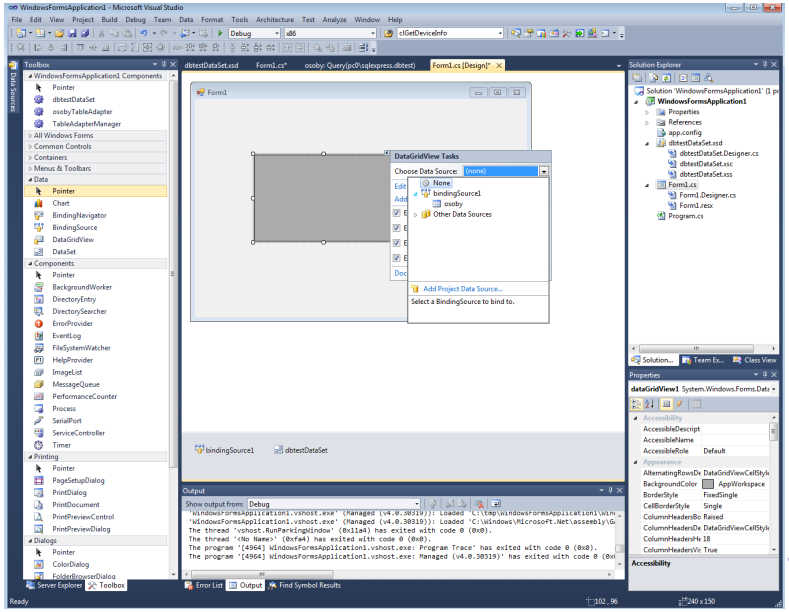
Notatki

Narzędzia dostępne w Visual Studio



Notatki

Narzędzia dostępne w Visual Studio



Notatki

W następnym tygodniu między innymi

1. model bezpołączeniowy i transakcje w ADO.NET,
2. technologia LINQ,
3. zapytania do typów zgodnych z kolekcjami,
4. zapytania do baz danych,
5. zapytania równoległe.

Proponowane tematy prac pisemnych:

1. struktura klas obsługujących dostęp do danych,
2. przeznaczenie oraz zalety ADO.NET Entity Framework,
3. analiza narzędzi dostępnych w pakiecie Visual Studio.

Dziękuje za uwagę!!!

Notatki