Wątki & Refleksje.

C#

Wydział Matematyki i Informatyki Stosowanej
Ćwiczenia: Michał Mnich

Refleksje.

Refleksja służy do uzyskania informacji o typie w trakcie wykonywania programu. Klasy, które mają dostęp do metadanych działającego programu są zdefiniowane w przestrzeni nazw System.Reflection.

Przestrzeń nazw System.Reflection zawiera klasy, które pozwalają na uzyskanie informacji o aplikacji oraz pozwalają na dynamiczne dodawanie typów, wartości i obiektów do aplikacji.

Wykorzystanie refleksji pozwala na:

- podgląd atrybutów w trakcie wykonywania programu;
- sprawdzenie różnych typów danych w danej bibliotece oraz utworzenie ich instancji;
- wykonanie późnego wiązania do metod i właściwości (późne wiązanie oznacza, że np. docelowa metoda jest poszukiwana w trakcie wykonywania programu. Wiązanie takie ma zwykle wpływ na wydajność. Poszukiwanie takie wymaga dopasowania w trakcie wykonywania programu, oznacza to, że wywołania metod są wolniejsze. Przeciwieństwem jest wczesne wiązanie, tj. docelowa metoda jest znana już w trakcie kompilacji kodu);
- tworzenie nowych typów w trakcie wykonywania programu a następnie wykonywanie różnych zadań przy użyciu tych typów.

Przykładowe użycie:

```
using System;
namespace Refleksja
{
   class Program
   {
      static void Main(string[] args)
      {
         System.Reflection.MemberInfo info = typeof(MyClassToGetAttributeInfo);
         // pobranie listy atrybutów
         object[] attributes = info.GetCustomAttributes(true);
         for (int i = 0; i < attributes.Length; i++)
         {
            // Wypisujemy wszystkie atrybuty
            Console.WriteLine(attributes[i]);
            // Dodatkowo uzyskamy dostęp do opisu naszego atrybutu
            ExampleAttribute ea = (ExampleAttribute)attributes[i];
            Console.WriteLine("Info: {0}", ea.message);
         }
         Console.ReadKey();
         // Wynik działania programu
         // Refleksja.ExampleAttribute
         // Info: Informacja o mojej klasie
      }
   }
   [AttributeUsage(AttributeTargets.All)]
   public class ExampleAttribute : Attribute
   {
      public readonly string message;
      private string topic;
      public ExampleAttribute(string Message)
      {
         this.message = Message;
      }
      public string Topic
      {
         get
         {
            return topic;
         }
         set
         {
            topic = value;
         }
      }
   }
   [ExampleAttribute("Informacja o mojej klasie")]
   class MyClassToGetAttributeInfo
   {
   }
}
```

Więcej informacji znajdą państwo w wykładzie.

roces to program wykonujący. System operacyjny używa procesów do rozdzielania wykonywanych aplikacji. Wątek jest jednostką podstawową, do której system operacyjny przydziela czas procesora. Każdy wątek ma priorytet planowania i utrzymuje zestaw struktur, których system używa do zapisywania kontekstu wątku, gdy wykonywanie wątku jest wstrzymane. Kontekst wątku zawiera wszystkie informacje niezbędne do bezproblemowego wznowienia wykonywania wątku, w tym zestaw rejestrów i stosów procesora CPU. W kontekście procesu można uruchomić wiele wątków. Wszystkie wątki procesu współdzielą swoją wirtualną przestrzeń adresową. Wątek może wykonać dowolną część kodu programu, w tym części, które są aktualnie wykonywane przez inny wątek. Domyślnie program .NET jest uruchamiany z pojedynczym wątkiem, często nazywanym wątkiem podstawowym . Można jednak utworzyć dodatkowe wątki do wykonywania kodu równolegle lub współbieżnie przy użyciu wątku głównego. Te wątki są często nazywane wątkami roboczymi .

Wielowątkowości należy używać, aby zwiększyć czas odpowiedzi aplikacji i wykorzystać wieloprocesorowy lub wielordzeniowy system w celu zwiększenia przepływności aplikacji. Rozważ użycie aplikacji klasycznej, w której wątek główny jest odpowiedzialny za elementy interfejsu użytkownika i reaguje na działania użytkownika. Wątki robocze umożliwiają wykonywanie czasochłonnych operacji, które w przeciwnym razie zajmują wątek podstawowy i sprawiają, że interfejs użytkownika nie odpowiada. Można również użyć dedykowanego wątku do komunikacji sieciowej lub urządzenia, aby zwiększyć wydajność przychodzących komunikatów lub zdarzeń.

Jeśli program wykonuje operacje, które mogą być wykonywane równolegle, łączny czas wykonywania można zmniejszyć, wykonując te operacje w oddzielnych wątkach i uruchamiając program w wieloprocesorowym lub wielordzeniowym systemie. W takim systemie użycie wielowątkowości może zwiększyć przepływność oraz zwiększyć czas odpowiedzi.

Począwszy od .NET Framework 4, zalecanym sposobem użycia wielowątkowości jest użycie biblioteki zadań równoległych (TPL) i Parallel LINQ (PLINQ). Zarówno TPL, jak i PLINQ są zależne od ThreadPool wątków. System.Threading.ThreadPoolKlasa udostępnia aplikację .NET z pulą wątków roboczych. Można również użyć wątków puli wątków.

Najstarszym rozwiązaniem jest użycie System.Threading.Thread klasy, która reprezentuje wątek zarządzany. Wiele wątków może potrzebować dostępu do zasobu udostępnionego. Aby zachować zasób w stanie nieuszkodzonym i uniknąć sytuacji wyścigu, należy zsynchronizować z nim dostęp do wątku. Możesz również skoordynować interakcję wielu wątków. Platforma .NET udostępnia szereg typów, których można użyć do synchronizowania dostępu do zasobu udostępnionego lub współdziałania wątku współrzędnych.

Szersze informacje wraz z przykładami znajdują się na wykładzie.

Zadanie 1:

Proszę przy pomocy klasy Ping (z System.Net.NetworkInformation) sprawdzić dostępność serwerów zawartych w pliku ping.txt pllik proszę utworzyć plik na podstawie listy poniżej. Proszę przygotować i zmierzyć czas wykonania zadania realizując je na trzy sposoby: Sekwencyjnie Równolegle (max. 4 wątki) za pomocą AsParallel Równolegle (max. 4 wątki) korzystając z mechanizmu Tasków i wybranej przez siebie metody synchronizacji.

```
Kraj;Adres
Armenia;ftp.am.debian.org
Australia;ftp.au.debian.org
Austria;ftp.at.debian.org
Belgia;ftp.be.debian.org
Białoruś;ftp.by.debian.org
Brazylia;ftp.br.debian.org
Bułgaria;ftp.bg.debian.org
Chile;ftp.cl.debian.org
Chiny;ftp2.cn.debian.org
Chiny;ftp.cn.debian.org
Chorwacja;ftp.hr.debian.org
Czechy;ftp.cz.debian.org
Dania;ftp.dk.debian.org
Estonia;ftp.ee.debian.org
Francja;ftp.fr.debian.org
Grecja;ftp.gr.debian.org
Hiszpania;ftp.es.debian.org
Holandia;ftp.nl.debian.org
Hongkong;ftp.hk.debian.org
Japonia;ftp.jp.debian.org
Kanada;ftp.ca.debian.org
Korea;ftp.kr.debian.org
Litwa;ftp.lt.debian.org
Mołdawia;ftp.md.debian.org
Niemcy;ftp2.de.debian.org
Niemcy;ftp.de.debian.org
Norwegia;ftp.no.debian.org
Nowa Kaledonia;ftp.nc.debian.org
Nowa Zelandia;ftp.nz.debian.org
Polska;ftp.pl.debian.org
Portugalia;ftp.pt.debian.org
Rosja;ftp.ru.debian.org
Salwador;ftp.sv.debian.org
Słowacja;ftp.sk.debian.org
Słowenia;ftp.si.debian.org
Stany Zjednoczone;ftp.us.debian.org
Szwajcaria;ftp.ch.debian.org
Szwecja;ftp.fi.debian.org
Szwecja;ftp.se.debian.org
Tajwan;ftp.tw.debian.org
Turcja;ftp.tr.debian.org
Węgry;ftp.hu.debian.org
Włochy;ftp.it.debian.org
Wielka Brytania;ftp.is.debian.org
Wielka Brytania;ftp.uk.debian.org
```

Zadanie 2:

Proszę przy użyciu refleksji przeprowadzić pełna analizę poniższej klasy.

```
public class Customer
{
   private string _name;
   protected int _age;
   public bool isPreferred;


   public Customer(string name)
   {
      if (string.IsNullOrEmpty(name)) throw new ArgumentNullException("Customer
name!");
      _name = name;
   }

   public string Name
   {
      get
      {
         return _name;
      }
   }
   public string Address { get; set; }
   public int SomeValue { get; set; }

   public int ImportantCalculation()
   {
      return 1000;
   }

   public void ImportantVoidMethod()
   {
   }

      public enum SomeEnumeration
   {
      ValueOne = 1
      , ValueTwo = 2
   }

   public class SomeNestedClass
   {
      private string _someString;
   }
}
```

poprzez analizę rozumiem ćwiczenie, w którym należy wypisać za pomocą refleksji następujące elementy powyższej klasy do każdej prosze przy nazwie danego pola wypisać także jakiego jest ona typu

```
Console.WriteLine("Fields: ");
//lista pól w klasie Pogrupowane względem dostępu
Console.WriteLine("-- Public: ");
//publiczne
Console.WriteLine("-- Non Public: ");
//niepubliczne
//Przykład:
//Type: “string”; name: “_name”

Console.WriteLine("Methods: ");
//Lista metod

Console.WriteLine("Nested types: ");
//typy zagnieżdżone

Console.WriteLine("Properties: ");
//propercje

Console.WriteLine("Members: ");
//Członkowie
```

Zadanie 3:

Za pomocą metody **GetProperty** prosze ustawić wszytkie propercje w klasie **Customer** a następnie prosze je wyswietlić.