FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

07 – Bázové třídy platformy .NET

Obsah přednášky

- BCL
- Správa paměti v .NETu
- IDisposable
- Kolekce
- Streamy

BCL – Base Class Libary

- základní sada knihoven frameworku
- jádro je v mscorlib.dll

Obsahuje

- základní datové typy
- datové struktury
- vykonává I/O
- Informace o načtených typech
- Security
- Text
- Kolekce
- Threading
- Výčet není kompletní!

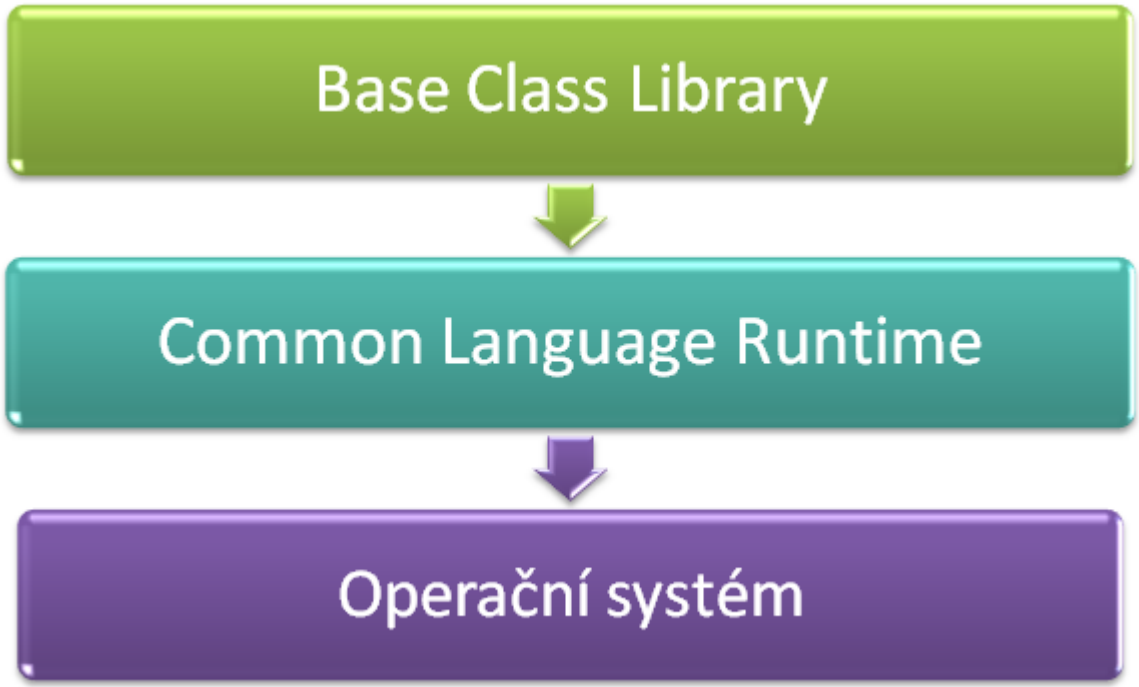
Jak je Base Class Library organizována

- Assemblies
- Organizována do jednotlivých namespaces

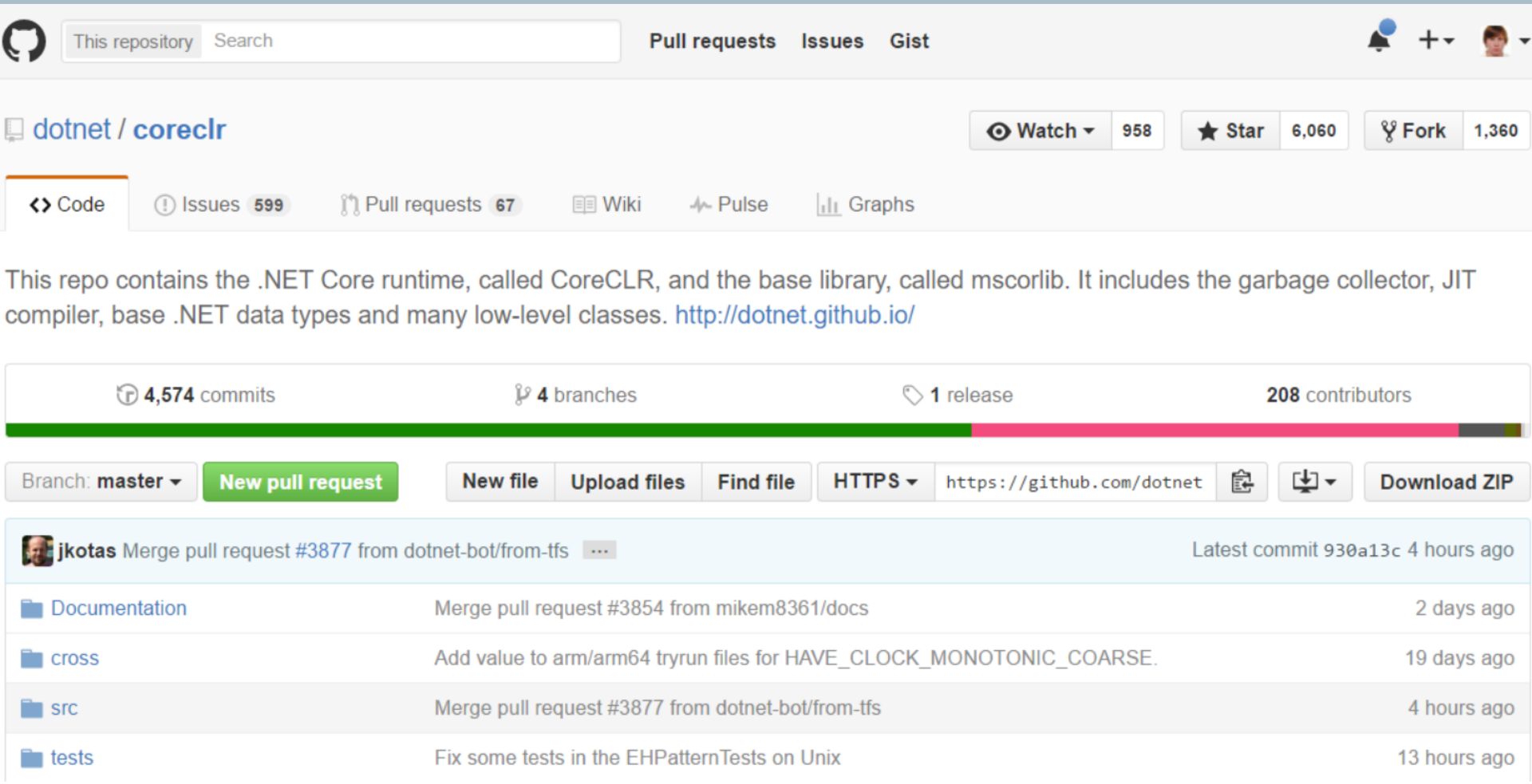
Příklady

- System
- System.Collections
- System.Diagnostics
- System.Globalization
- System.IO
- System.Security
- System.Text
- System.Threading
- Výčet není kompletní!

Směr komunikace



.NET CoreCLR


This repository
Search
Pull requests
Issues
Gist
dotnet / coreclr
Watch 958
Star 6,060
Fork 1,360
Code
Issues 599
Pull requests 67
Wiki
Pulse
Graphs
This repo contains the .NET Core runtime, called CoreCLR, and the base library, called mscorlib. It includes the garbage collector, JIT compiler, base .NET data types and many low-level classes. http://dotnet.github.io/
4,574 commits
4 branches
1 release
208 contributors
Branch: master
New pull request
New file
Upload files
Find file
HTTPS
https://github.com/dotnet
Download ZIP
jkotas Merge pull request #3877 from dotnet-bot/from-tfs
Latest commit 930a13c 4 hours ago
Documentation
Merge pull request #3854 from mikem8361/docs
2 days ago
cross
Add value to arm/arm64 tryrun files for HAVE_CLOCK_MONOTONIC_COARSE.
19 days ago
src
Merge pull request #3877 from dotnet-bot/from-tfs
4 hours ago
tests
Fix some tests in the EHPatternTests on Unix
13 hours ago

Obsah přednášky

- BCL
- **Správa paměti v .NETu**
- IDisposable
- Kolekce
- Streamy

Správa paměti v .NET – Garbage Collector

- Alokace/dealokace v režii CLR (Common Language Runtime)

- Reprezentováno komponentou Garbage Collector
  - Třída `System.GC`
  - Periodicky odstraňuje objekty bez reference
  - Využívá systém generací
  - Kolekci je možná vynutit – `GC.Collect();`

```
private void AllocateMemory()
{
    //Memory is going to be released when execution
    //leaves the scope of method
    var wastedMemory = new string('X', 100000000);
}
```

- I v .NET je nutné řešit memory-leak (zapomenuté reference)

Správa paměti v .NET - Finalizace

- Ekvivalent destruktorů v jazyce C
- Volá se před smazáním objektu

```
~ClassWithFinalizer()
{
  //Cleanup code
}
```

- Slouží jako „poslední šance“ k uklizení externích zdrojů (viz Dispose), lépe je ale uklidit explicitně!
- Složitý či blokující finalizační kód degraduje výkon
- Pozor na výjimky

- Finalizér je možné potlačit, či naplánovat znovu

```
GC.ReRegisterForFinalize(this);
GC.SuppressFinalize(this);
```

Obsah přednášky

- BCL
- Správa paměti v .NETu
- **IDisposable**
- Kolekce
- Streamy

IDisposable

- Rozhraní pro objekty vyžadující explicitní úklid
- Metoda `void Dispose();`
- Využívá např. `Component, Stream, SqlConnection, …`
- Pokud objekt drží jiné disposable objekty, sám by měl disposable implementovat
- Na volání metody Dispose z kontextu finalizace existuje vzor (viz dále)

IDisposable – implementační vzor pro finalizaci

```
public class DisposableClass : IDisposable
{
    private bool disposed = false;

    0 references
    public void Dispose() // NOT virtual
    {
        Dispose(true);
        GC.SuppressFinalize(this); // Prevent finalizer from running.
    }

    2 references
    protected virtual void Dispose(bool disposing)
    {
        if (!disposed)
        {
            disposed = true;
            if (disposing)
            {
                // Call Dispose() on other objects owned by this instance.
                // You can reference other finalizable objects here.
                Console.WriteLine(" Disposing everything including other finalizable objects");
            }
            // Release unmanaged resources owned by (just) this object.
            Console.WriteLine(" Disposing internals of this object");
        }
    }

    0 references
    ~DisposableClass()
    {
        Dispose(false);
    }
}
```

Obsah přednášky

- BCL
- Správa paměti v .NETu
- IDisposable
- **Kolekce**
- Streamy

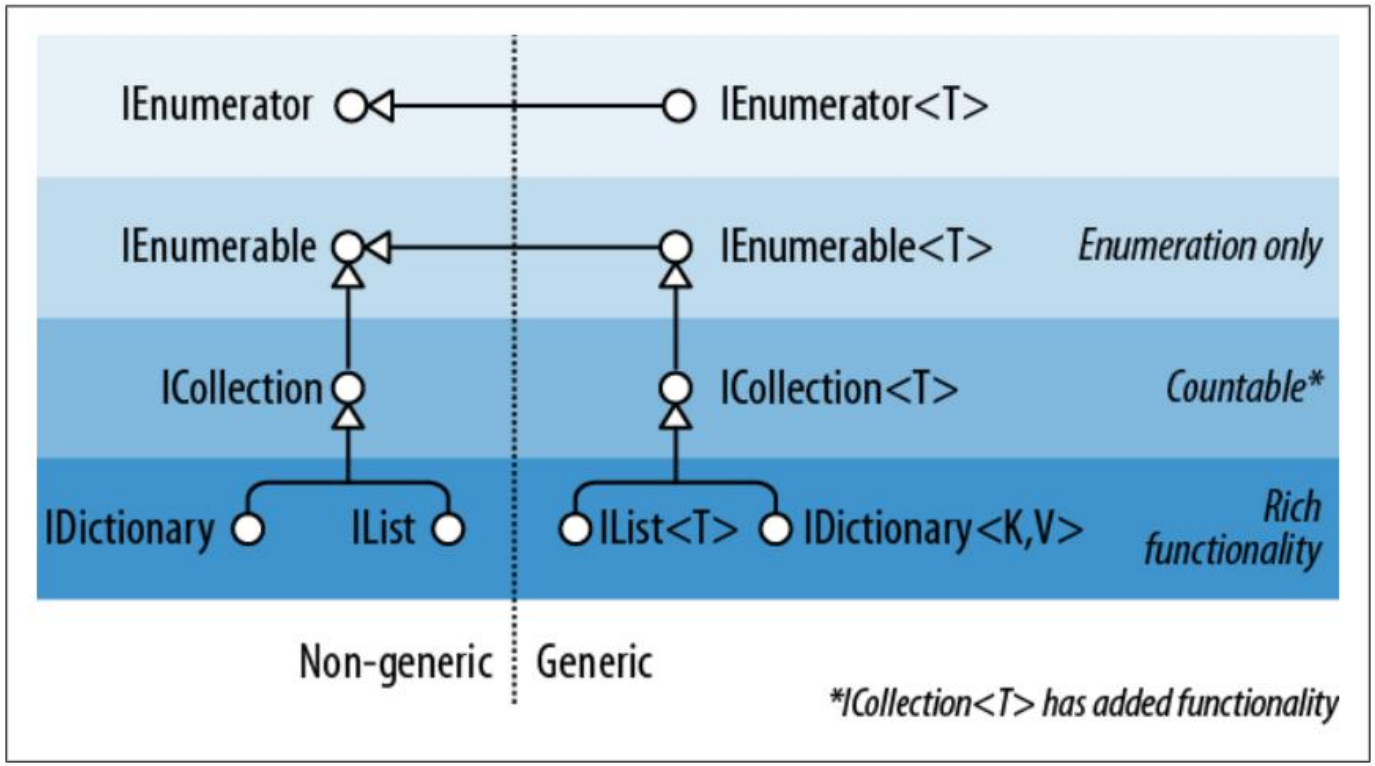
Kolekce - přehled

- **Negenerické**
  - Rozhraní - `IEnumerable, IEnumerator, IComparer, IEqualityComparer, IHashCodeProvider ICollection, IList, IDictionary`
  - Implementace - `BitArray, ArrayList, SortedList, HashTable, Queue, Stack`
  - Speciální – `NameValueCollection, OrderedDictionary, ListDictionary, HybridDictionary`
  - Bázové – `CollectionBase, DictionaryBase`

- **Generické**
  - Rozhranní – `IEnumerable<T>, IEnumerator<T>, IComparer<T>, IEqualityComparer<T> ICollection<T>, IList<T>, IDictionary<TKey,TItem>, IReadOnlyCollection<T>, IReadOnlyDictionary<TKey,TItem>, ISet<T>`
  - Implementace – `List<T>, SortedList<T>, LinkedList<T>, Dictionary<TKey,TItem>, SortedDictionary<TKey,TItem>, HashSet<T>, Queue<T>, Stack<T>`
  - Bázové – `Comparer<T>, Collection<T>, ReadOnlyCollection<T>`

  - Výčet není kompletní!

Rozhraní


IEnumerator
IEnumerator<T>
IEnumerable
IEnumerable<T>
Enumeration only
ICollection
ICollection<T>
Countable*
IDictionary
IList
IList<T>
IDictionary<K,V>
Rich functionality
Non-generic
Generic
*ICollection<T> has added functionality

Kolekce - Enumerace

- Procházení kolekcí prvek za prvkem na základě `IEnumerable/IEnumerable<T>`
- Umožňuje použít klíčové slovo `foreach`

```
string[] strings = new string[] { "Hello", "World!" };
foreach (string str in strings)
{
  Console.WriteLine(str);
}
```

- Enumeraci je možné ovládat pomocí instance enumerátoru

```
IEnumerator<string> enumerator = strings.GetEnumerator();
string str = enumerator.Current;
bool success = enumerator.MoveNext();
```

- Enumeraci je možné implementovat pomocí `IEnumerator/IEnumerator<T>` nebo jednodušeji pomocí klíčového sousloví `yield return <value>;`

Ukázkové použití enumerátoru

```
string sampleString = "Hello";

// Because string implements IEnumerable, we can call GetEnumerator():
IEnumerator enumerator = sampleString.GetEnumerator();

while (enumerator.MoveNext())
 {
  char sampleChar = (char)enumerator.Current;
  Console.Write(sampleChar + ".");
}

// Output: H.e.l.l.o.
```

Kolekce – Implementace enumerátoru

```
public class MyGenCollection : IEnumerable<int>
 {
    int[] data = { 1, 2, 3 };

    public IEnumerator<int> GetEnumerator()
    {
        foreach (int i in data)
        yield return i;
    }

    // Explicit implementation
    IEnumerator IEnumerable.GetEnumerator()
    {
       // keeps it hidden.
       return GetEnumerator();
     }
}
```

ICollection, IObservableCollection

```
public interface ICollection<T> : IEnumerable<T>, IEnumerable
{
    void Add(T item);
    void Clear();
    bool Contains(T item);
    void CopyTo(T[] array, int arrayIndex);
    bool Remove(T item);
    int Count { get; }
    bool IsReadOnly { get; }
}
```

IObservableCollection

- Analogické INotifyPropertyChange
- Notifikuje mi změnu počtu

Kolekce – praktické ukázky

```
// Example 1 DoBasicCollectionOperations
examples.BasicCollectionOperations.DoBasicCollectionOperations();
Console.ReadKey();

//Example 2 - ArrayList, List<T>, IEnumerable, IEnumerable<T>
examples.ListEnumeration();
Console.ReadKey();

//Example 3 - Enumerator manipulation
examples.UsingEnumerator();
Console.ReadKey();

//Example 4 - Implementation of custom enumeration
examples.CustomEnumerator();
Console.ReadKey();

// ...
```

Obsah přednášky

- BCL
- Správa paměti v .NETu
- IDisposable
- Kolekce
- **Streamy**

Použití

```
using (Stream s = new FileStream("test.txt", FileMode.Create))
{
    Console.WriteLine(s.CanRead); // True
    Console.WriteLine(s.CanWrite); // True
    Console.WriteLine(s.CanSeek); // True
    s.WriteByte(101);
    s.WriteByte(102);
    byte[] block = { 1, 2, 3, 4, 5 };
    s.Write(block, 0, block.Length); // Write block of 5 bytes
    Console.WriteLine(s.Length); // 7
    Console.WriteLine(s.Position); // 7

// ...
```

Compressions streams

```
using (Stream s = File.Create("compressed.bin"))
{
   // DeflateStream == decorator
   using (Stream ds = new DeflateStream(s, CompressionMode.Compress))
   {
      for (byte i = 0; i < 100; i++)
      {
         ds.WriteByte(i);
      }
   }
}
```

Operace se soubory a adresáři

Operace se soubory a adresáři

- BCL obsahuje předpřipravené třídy File, Directory, FileInfo, DirectoryInfo

IsolatedStorage

- Standardizovaná cesta pro ukládání dat desktopových aplikací
- Zabezpečený přístup
- V uživatelské složce
- Unikátní na assembly

Reference

BCL

- .NET Framework Class Library Overview
- .NET Core zdrojové kódy
- Base Class Libraries
- .NET Framework Class Library

Nástroje

- dotPeek Free .NET Decompiler and Assembly Browser
- LINQPad

- Design patterns

- C# in Nutshell