C#编码规范

对于编程中非代码文件（例如xml 等），我们目前的最佳指南是代码风格的一致性。编辑代码文件时，需要保持新增/更改的代码与原文件中的代码风格一致性。对于新建的代码文件，它也应符合本编码规范。好的编码规范，有利于提升代码的可读性。.NET Core 框架项目使用的是 Visual Studio 默认编码规范。

通常，我们编码时，应遵循 Visual Studio 默认编码规范中的以下规则：

**1、接口名、类名、方法名、参数名和变量名**

所有的接口名、类名和方法名都应该使用 **PascalCasing** 风格来定义，对于接口参数、构造函数参数和方法参数，其参数名则使用 **camelCase** 风格。对于方法体内的局部变量，则使用 **camelCase** 风格。

**2、花括号**

使用 **Allman** 样式的花括号，每个花括号在新的一行开始，而且花括号不进行缩进，花括号里面的代码缩进。

while (x == y)

{

something();

somethingelse();}

对于单行语句块，可以不带花括号，但需要把它放在同一行

if (xx == null) xx = new Foo();

一个例外是，允许 using 语句嵌套在另一个 using 语句中，从同一缩进级别的下一行开始，即使嵌套的 using 包含受控块也是如此。

#### 3、空格

使用 4 个空格作为一个缩进，而不是使用 tab。

#### 4、字段

所有的 **internal** 和 **private** 字段使用 **\_camelCase** 风格，即在字段添加 **\_** 前缀，并尽可能使用 **readonly** 。对于静态字段添加 **s\_** 前缀，对于线程静态字段添加 **t\_** 前缀。如果使用了静态的字段而且可以设置 **readonly** ，需要把 **readonly** 放在 **static** 后面（即 **static readonly** 而不是 **readonly static** ）。公开(public)字段应谨慎使用，如果需要公开字段，则应使用 **PascalCasing** 风格，并且不带前缀。

#### 5、限定

除非绝对必要，应避免使用 **this.** 。在扩展方法中，需要使用 **this** 。

#### 6、访问修饰

即使访问可见性是默认的，我们也要始终指定，我们也总是指定可见性（例如， **private string \_foo** 而不是 **string \_foo** ）。访问可见性应该是第一个修饰符（例如， **public abstract** 而不是 **abstract public** ）

#### 7、命名空间

命名空间导入应在命名空间声明之外的文件顶部指定，并且应按字母顺序排序，但 System.\* 命名空间除外，这些命名空间应放置在所有其他名称空间的顶部。

#### 8、空行

随时避免出现多个空白行。例如，类型的成员之间不要有两个空白行。

#### 9、多余空格

避免多余的空格。例如，避免 **if（someVar == 0）...** ，其中点标记为多余空格。如果使用 Visual Studio 辅助检测，通过启用 Visual Studio中“查看空格（Ctrl + R，Ctrl + W）”或“编辑->高级->查看空格”。

#### 10、隐式类型

仅在明显可以知道对象类型时，可以使用 **var** ，如 **var stream = new FileStream(...)** 。在无法明显知道对象类型时，不可以使用 **var** ，如 **var stream = OpenStandardInput()** 。

#### 11、关键字

使用语言的关键字代替 BCL 类型，如使用 **int, string, float** 代替 **Int32，String，Single** 。

12、常量命名

我们使用 **PascalCasing** 风格来命名所有常量局部变量和字段。唯一的例外是互操作代码，其中常量值应与您通过互操作调用的代码的名称和值完全匹配。

#### 13、变量名字符串

如果使用变量名的常量，必须使用 **nameof** 关键字。

#### 14、其它字符

在源代码中包含非 ASCII 字符时，需要使用 Unicode 转义序列（\uXXXX）代替文字字符。文字非 ASCII 字符有时会被工具或编辑器弄乱。

#### 15、条件约定

如果使用条件语句，则遵循以下约定：

if(source == null) throw new ArgumentNullException("source")

if/else if/.../else

if/else if/.../else

#### 16、代码格式工具

我们也可以使用 .NET Codeformatter 工具 来确保代码库随时间推移保持一致的样式，该工具会自动修复代码库以符合上述准则。

#### 17、示例文件

**ObservableLinkedList`1.cs:**

using System;

using System.Collections;

using System.Collections.Generic;

using System.Collections.Specialized;

using System.ComponentModel;

using System.Diagnostics;

using Microsoft.Win32;

namespace System.Collections.Generic

{

public partial class ObservableLinkedList<T> : INotifyCollectionChanged, INotifyPropertyChanged

{

private ObservableLinkedListNode<T> \_head;

private int \_count;

public ObservableLinkedList(IEnumerable<T> items)

{

if (items == null)

throw new ArgumentNullException(nameof(items));

foreach (T item in items)

{

AddLast(item);

}

} public event NotifyCollectionChangedEventHandler CollectionChanged;

public int Count

{

get { return \_count; }

}

public ObservableLinkedListNode AddLast(T value)

{

var newNode = new LinkedListNode<T>(this, value);

InsertNodeBefore(\_head, node);

}

Protected virtual void OnCollectionChanged(NotifyCollectionChangedEventArgs e)

{

NotifyCollectionChangedEventHandler handler = CollectionChanged;

if (handler != null)

{

handler(this, e);

}

}

private void InsertNodeBefore(LinkedListNode<T> node, LinkedListNode<T> newNode)

{

...

}

...

}

}

**ObservableLinkedList`1.ObservableLinkedListNode.cs:**

using System;

namespace System.Collections.Generics

{

partial class ObservableLinkedList<T>

{

public class ObservableLinkedListNode

{

private readonly ObservableLinkedList<T> \_parent;

private readonly T \_value;

internal ObservableLinkedListNode(ObservableLinkedList<T> parent, T value)

{

Debug.Assert(parent != null);

\_parent = parent;

\_value = value;

}

public T Value

{

get { return \_value;

}

}

}

...

}

}