.NET 编码规范

设计人	时间	版本	事件
王海波	2012-5-8	V1.0	初稿

目录

一、 概述

本文档是开发人员在使用.NET 开发时应遵循的编码规范。此规范建立在微软推荐.NET 规范之上，并综合考虑了本部门的特定情况。同时，本文档不仅限于编码规范，亦包括一些值得推荐的最佳编程实践。在实际工作中，应不断吸收新的经验或建议，以便不断完善和改进本文档。

1.1 原则

高质量的代码一般具有一些共性，这些共性可作为我们编写代码时的原则：

- 简洁性：代码应当简洁易懂，逻辑清晰，易于重用。同时应具备必要的注释(关于注释请参考文档后续说明)
- 正确性：应当在完全了解需求之后进行编码，以便代码运行结果与需求相符
- 一致性：代码应该具有一致的编程风格和设计，以保证代码的易读性，从而降低维护和 Bug 修正难度。
- 安全性：编写代码时应适当考虑安全性问题，如避免 SQL 注入攻击等。

1.2 术语

本文档不仅包括了一些必须执行的原则，同时也包括了一些建议与推荐做法。为了进行区分，文档将采用以下术语：

术语	说明	理由
必须	此规范在任何情况下都应该遵守	可防止 bug
不要	此规范不能被使用	可防止 bug
应该	此规范适用于大多数情况	可统一编码风格
不应该	在多数情况下都不要使用，除非有合理的理由	可统一编码风格
可以	可以按照实际情况使用	可用于统一编程风格，但不一定带来益处

二、 命名规范

2.1 命名的基本原则

- **必须**将各种类型、方法、变量、特性和结构选取代表其含义的名称，其名称应当反映其作用

- **不要**在命名中使用下划线、连字号或其他非数字字母的字符
- **不要**使用匈牙利命名法（如 strSql）

2.2 标识符命名约定

标识符	规范	命名结构	示例
类，结构体	Pascal 规范	名词	public class ComplexNumber {...} public struct ComplextStruct {...}
命名空间	Pascal 规范	名词 **不要** 以相同的名称来命名命名空间和其内部的类型。	namespace Microsoft.Sample.Windows7

枚举	Pascal 规范	名词
		必须 以复数名词或名词短语来命名标志枚举，以单数名词或名词短语来命名简单枚举。
			[Flags]
			public enum ConsoleModifiers
			{
			Alt,
			Control
			}

方法	Pascal 规范	动词或动词短语	public void Print （） {...} public void ProcessItem （） {...}
Public 属性、字段	Pascal 规范	名词或形容词 **必须**以集合中项目的复数形式命名该集合，或者单数名词后面跟 “List” 或者 “Collection” 。 **必须**以肯定短语来命名布尔属性，（ CanSeek ，而不是 CantSeek ）。当以 “Is,” “Can,” or “Has” 作布尔属性的前缀有意义时，您也可以这样做。	public string CustomerName public ItemCollection Items public bool CanRead
非 Public 属	Camel 规范或	名词或形容词	private string name

性、字段	_camel 规范	如果使用 '_' 前缀，则应保持代码一致性。	; private string _name;
事件	Pascal 规范	动词或动词短语 **应该**用现在式或过去式来表明事件之前或是之后的概念。 **应该**使用 “Before” 或者 “After” 前缀或后缀来指明事件的先后。	public event WindowClosed public event WindowClosing
委托	Pascal 规范	**必须**为用于事件的委托增加 ‘EventHandler’ 后缀。 **必须**为除了用于事件处理程序之外的委托增加 ‘Callback’ 后缀。 **不要**为委托增加 “Delegate” 后缀。	public delegate WindowClosedEventHandler
接口	Pascal 规范，带有‘I’ 前缀	名词	public interface IDictionary
常量	Pascal 规范用于 Public 常量； Camel 规范用于 Internal 常量； 只有 1 或 2 个字符的缩写需全部字符大写（如：PI）。	名词	public const string MessageText = "A" ; private const string messageText = "B" ; public const double PI =3.14159 ;
参数，变量	Camel 规范	名词	int customerID;
泛型参数	Pascal 规范，带有‘T’ 前缀	名词 **必须**以描述性名称命名泛型参数，除非单字符名称已有足够描述性。 **必须** 以 T 作为描述性类型参数的前缀。 **应该** 使用 T 作为单字符类型参数的名称。	T, TItem, TPolicy
资源	Pascal 规范	名词 **必须** 提供描述性强的标识符。同时，尽可能保持	ArgumentExceptionInvalidName

		简洁，但是不应该因空间而牺牲可读性。 **应该**仅为命名资源 使用字母数字字符和下划线。	

2.3 用户界面前缀约定：

控件类型	前缀
Button	btn
CheckBox	chk
CheckedListBox	lst
ComboBox	cmb
ContextMenu	mnu
DataGrid	dg
DateTimePicker	dtp
Form	前缀: XXXForm
GroupBox	grp
ImageList	iml
Label	lb

ListBox	lst
ListView	lvw
Menu	mnu
MenuItem	mnu
NotificationIcon	nfy
Panel	pnl
PictureBox	pct
ProgressBar	prg
RadioButton	rad
Splitter	spl
StatusBar	sts
TabControl	tab
TabPage	tab
TextBox	tb
Timer	tmr

TreeView	tvw

三、 基本编程规范

此部分规范主要针对代码风格、格式和结构。

3.1 总原则

- **必须**确保代码的明确性，易读性和可维护性。这亦是编码规范的目标
- **必须**确保代码都使用了本编码规范，以保持风格的一致性
- **应该**限制一行代码的最大长度。过长的代码将会降低易读性
- **不要**引用不必要的程序集。这样可减少项目生成时间，降低出错几率。
- **不要**再同一行放置多条语句，这将影响代码的调试。

3.2 变量

- **必须**在最小的作用域内声明变量，一般是在使用之前声明它们。
- **必须**在声明变量时给出初始值
- **必须**将变量的声明和赋值放于同一行
- **不要**在同一行中声明多个变量。一般一行对应一个变量，这样有利于添加注释，减少歧义

3.3 方法

- 方法的申明和调用原则上应将参数放于一行，如果参数过多，可进行换行，换行时应将返回值和方法名放在一行，推荐以下风格：

示例代码

```
string newS6xml = updateS6Protocol(inputProtocol, callbackProtocol, config, context);
或者：
string newS6xml = updateS6Protocol(inputProtocol,
    callbackProtocol, config, context);
或者：
string newS6xml = updateS6Protocol(inputProtocol,
                                   callbackProtocol,
                                   config,
                                   context);
```

- **必须**根据方法参数的类型进行排序，即 先输入参数，然后 ref 参数，最后 out 参数

3.4 枚举

- **必须**将代表某些值取值集合的类型声明为枚举类型，不要使用硬编码方式。如视频格式应该声明为枚举。
- **必须**为简单枚举提供一个零值枚举值，此举便于对枚举值进行判断(如是否进行初始化)，一般将零值枚举命名为“None"，当然也可根据实际情况定义更好的名称。

示例代码

```
/// <summary>
/// 任务状态
/// </summary>
public enum TaskItemStatus
{
    Unknown = 0, //未知
    Waiting, //等待执行
    Executing, //执行中
    Executed, //已执行
    Error, //错误
    WaitingCallback //等待回调
}
```

- **不应该**为标志枚举定义零值枚举
- 如果枚举为标志，或需要进行位运算，**必须**为其制定 FlagsAttribute 特性。相反，**不要**为简单枚举应用此特性。

示例代码

```
[Flags]
public enum SearchTypes
{
    Title = 1,//标题
    Keywords = 2,//关键字
    Time = 4,//时间
    Address = 8 //地点
}
```

3.5 语句块风格

- **应该**使用空行来分隔语句块。例如，在变量声明和代码之间空一行：

示例代码

```
string mpcmqname = config.MQName;
```

```
string mpcmqip = config.MQIPAddress;

if (string.IsNullOrEmpty(mpcmqname))
{
}
```

- **应该**在方法之间空两行来分隔方法
- **必须**使用 Allman 风格的大括号风格（即大括号应该位于新的一行）。
- 如果条件语句只包括一句代码，也不应该省略大括号，**应该**使用下列写法：

示例代码

```
if (instance == null)
{
    instance = new DefaultScheDuledTaskManager();
}
```

- **必须**对长代码使用#region ....#endregion 进行分块，此举可改善代码的易读性和结构。

3.6 注释

- **应该**使用注释来描述代码的意图，不要让注释流于形式，仅仅是重复代码，而应该描述出代码的目的或注意事项
- **应该**使用”//"注释方式，而不是/**/方式，即使是多行注释也应如此，注释应该与被注释的代码位于同一级
- **不应该**在代码中保留过多的注释，这样会影响代码的可读性。一般情况下，代码本身即是注释，如果代码逻辑复杂，可适当添加注释以便阅读者理解。
- **应该**为每一个重要的类或结构体添加注释，注释的样式应该使用.NET 的类注释模板
- **应该**为重要的方法添加注释，注释的样式应该使用.NET 方法注释模板

四、 文件和结构

本节将描述代码文件的组织结构：

- **不要**在一个代码文件中包括一个以上的 Public 类型，除非他们具有嵌套关系或者泛型关系
- **必须**将类名和文件名匹配，如 MainForm 类应该放在 MainForm.cs 文件里。
- 程序集属性**应该**包括公司名称和版权声明

五、 最佳实践

本节提供一些编码时的最佳实践，这些实践有助于提高代码质量，预防软件缺陷和 bug。

5.1 常量与只读静态变量的区别

编译器对这两种类型的处理方式是不同的，使用时应特别注意。对于常量编译器会将常量字段值嵌入到调用处，这样如果 B 程序集引用了 A 程序集中的常量，实际上在生成的 B 程序集中是嵌入的常量值，这样的结果是如果 A 程序集中的常量值应需求变化而改变，只要 B 程序集没有重新编译，那么 B 程序集中的常量仍然是最原始的值（即常量的编译方式类似于硬编码）。所以，只有在确定值永远不会改变时才使用常量，否则应该使用静态只读变量。

5.2 字符串操作

- **不要**使用"+"操作符来拼接大量字符串，而应该使用 StringBuilder。当然，如果拼接字符串较少就应该使用"+"操作符
- 在比较字符串时，**必须**显式制定比较规则，即是否区分大小写。
- **不要**使用 String.Compare 或 CompareTo 的返回值来检查字符串是否相等，因为这两个方法是用于排序的

5.3 数组与集合

- **应该**在底层函数中使用数组，以便减少内存消耗。对于公开的接口则应该选择集合，不要使用只读数组的字段，因为只读仅限于数组变量本身，而数据里面的项仍然是可以被修改的。这种情况，你应该使用只读集合，如 ReadOnlyCollection<T>
- **不要**返回数组或集合的空引用，而应该返回一个空集合，这样便于调用方简化处理。
- **避免**使用 ArrayList、Queue 等一般的集合类型，因为这些集合中的项都是 object 类型，在项是值类型的情况下，将会产生大量的拆箱和装箱操作。**应该**使用对应的泛型集合，如 List<T> 或 Queue<T>等

5.4 结构

- 尽量避免使用结构体，除非
  - 它逻辑上代表的是一个单值
  - 实例大小小于 16 字节
  - 它是不可变的
  - 它不会引起频繁的装箱或拆箱操作

5.5 类与接口

- **应该**使用继承来表示“is a”的关系，例如"素材是一种新闻实体"，那么 Clip 应该继承于 NewsEntity 类
- **应该**使用接口来表示“can do”的关系，如 IDisposable 表示"对象能被释放"

5.6 字段

- **不要**使用 public 或 protected 的实例字段，字段应该是 private 的，如果需要向外部公开，请使用属性
- **必须**将预定义对象实例定义为 public static readonly 字段
- **必须**将永远不会改变的字段定义为常量字段

5.7 属性

- 如果不允许用户修改属性值，**必须**将属性创建为只读属性
- **不要**使用只写属性，这种属性应该使用方法来代替，一般这种方法名称以 Set 开头
- **必须**为属性提供合理的默认值
- **不应该**从属性访问器中抛出异常，但此规范不适用于索引器，因为索引器中的索引值可能会超出范围，此时可抛出异常

5.8 构造函数

- **应当**减少构造函数的工作量，一般情况下，构造函数主要用于设置数据成员，获得构造函数参数。其余工作量应该延迟，直到必要时。
- **不要**在对象构造函数内部调用虚方法。调用虚方法时，实际调用了继承体系最底层的覆盖（override）方法，而不会考虑定义了该方法的类的构造函数是否已被调用，这样可能导致无法预料的错误

5.9 方法

- **必须**将输出参数(out)放在传值和引用参数之后
- **必须**验证传递给 public、protected 或显式实现的成员方法的实参，如果验证失败应抛出异常

5.10 事件

- 注意事件处理方法中可能会执行任意代码。**考虑**将引发事件的代码放入一个 try-catch 块中，

以避免由事件处理方法中抛出的未处理异常而引起的程序异常终止。

- **不要**在有性能要求的 API 中使用事件。虽然事件易于理解和使用，但是就性能和内存消耗而言，它们不如虚函数。

5.11 方法重载

- **应该**将最长重载函数设为虚函数（为了拓展性考虑）。短重载函数应该调用长重载函数。

示例代码

```
public static string ConvertToFullPath(TransducerContext context, UnifiedContentDefine s6obj,
   string sourceFileName, string format)
{
      return ConvertToFullPath(context, s6obj, sourceFileName, format, null);
}

public static string ConvertToFullPath(TransducerContext context, UnifiedContentDefine s6obj,
   string sourceFileName, string format, FileItemType fileItem)
{
      .....
      return target;
}
```

5.12 静态类

- **应当**合理使用静态类，它一般提供框架内基于对象的核心辅助功能
- 如果类的成员全是静态成员，那么**应该**将类也声明为静态类

5.13 错误和异常

- **应该**通过抛出异常来告知执行失败。异常在框架内是告知错误的主要手段。如果一个成员方法不能成功的如预期般执行，便应该认为是执行失败，并抛出一个异常。一定**不要**返回一个错误代码。
- **应该**抛出最明确，最有意义的异常（继承体系最底层的）。比如，如果传递了一个 null 实参，则抛出 ArgumentNullException，而不是其基类 ArgumentException。抛出 并捕获 System.Exception 异常通常都是没有意义的
- **不要**显式的从 finally 块内抛出异常。从调用方法内隐式的抛出异常是可以接受的。
- 在捕获并重新抛出异常时，**应该**使用 throw，这可以保留异常的调用堆栈。

示例代码

```
try
{
      ....
}
catch (Exception ex)
{
    LogWriter.WriteLog("S6ToMPCTranscode", String.Format("getMpcNotifyMQAddress()异常：{0}",
   ex.ToString()));
    throw;   //不要使用  throw ex 抛出异常
}
```

六、 技术选择

6.1 XML 操作

推荐使用 LINQ to XML，具体使用方法，请参见 MSDN 文档。地址：

http://msdn.microsoft.com/zh-cn/library/bb387098(v=vs.90).aspx

6.2 集合的处理

包括对集合的排序、筛选、投影、分组、聚合运算等都可使用 LINQ to Object 来实现，请参考 MSDN《LINQ 常规编程指南》，地址：http://msdn.microsoft.com/zh-cn/library/bb397912(v=vs.90).aspx

6.3 WebService

推荐使用 WCF 框架，相对于 WebService，WCF 具有更高的安全性、扩展性，同时适用范围更广。请参考 MSDN《基本 WCF 编程》，地址：http://msdn.microsoft.com/zh-cn/library/ms731067(v=vs.90).aspx

6.4 多线程

开启新线程时推荐使用.NET 的线程池，而不是手动创建新的线程。

示例代码

```
public void StartQuery()
{
      //使用线程池
      ThreadPool.QueueUserWorkItem(queryProc);

      //不要手动创建线程
      Thread queryThread = new Thread(queryProc);
      queryThread.IsBackground = true;
      queryThread.Start(null);
}
```

```
private void queryProc(object context)
{
    ……
}
```

多线程下的单例模式，可采用锁定双重检查的方法：

示例代码

```
namespace NSLib.WorkFlow.Engine.Scheduled
{
    public static   class ScheduledTaskManagerFactory
    {
        private static IScheduledTaskManager instance = null;
        private static object locker = new object();

        public static IScheduledTaskManager CurrentScheduledTaskManger
        {
            get
            {
                if (instance == null)
                {
                    lock (locker)
                    {
                        if (instance == null)
                        {
                            instance = new DefaultScheDuledTaskManager();
                        }
                    }
                }
                return instance;
            }
        }
    }
}
```

七、 参考资料

- 微软一站式示例代码库编程规范，网址：http://1code.codeplex.com/
- .NET 类库开发设计准则，地址：http://msdn.microsoft.com/zh-cn/library/ms229042(v=vs.90).aspx
- Microsoft 基本设计准则，地址：http://msdn.microsoft.com/zh-cn/library/dd264947.aspx