C#编码规范

简介： 本篇内容记录了 命名惯例和规范、 ** 良好的编程习惯、** 注释 、 ** 异常处理

一、 命名惯例和规范

注：

Pascal：大小写形式 - 所有单词第一个字母大写，其他字母小写。

驼峰式：大小写形式 - 除了第一个单词，所有单词第一个字母大写，其他字母小写。

1：类名使用 Pascal 大小写形式

```
1 public class HelloWorld{ void SayHello(string name) {  ... }}
```

3：变量和方法参数使用 驼峰式 大小写形式

```
1 Public int totalCount = 0;
```

4：根据类的具体情况进行合理的命名

以Class声明的类，都必须以名词或名词短语命名，体现类的作用。如：

```
1 Class Indicator {}
```

当类只需有一个对象实例（全局对象，比如Application等），必须以Class结尾，如

```
1 Class ScreenClass
```

当类只用于作为其他类的基类，根据情况，以Base结尾：

```
1 Class IndicatorBase
```

5：不要使用匈牙利方法来命名变量 （m_xxxx）

```
1 string m_sName; int nAge;
```

6：用有意义的，描述性的词语来命名变量

7：文件名要和类名匹配

例如，对于类HelloWorld, 相应的文件名应为 HelloWorld.cs

8：接口的名称前加上I

```
1 interface ImyInterface   {……}
```

9： 在私有成员变量前面加上m_。对于m_后面的变量名使用骆驼命名方法：（注意第五条）

```
1 public class SomeClass
2 {
3     private int m_Number;
4 }
```

二、 ** **良好的编程习惯

1：避免使用大文件。如果一个文件里的代码超过300～400行，必须考虑将代码分开到不同类中。

2：避免写太长的方法。一个典型的方法代码在1～50行之间。如果一个方法发代码超过50行，应该考虑将其分解为不同的方法。

3：方法名需能看出它作什么。别使用会引起误解的名字。如果名字一目了然，就无需用文档来解释方法的功能了。

```
1 public class SomeClass   <br>{  <br>    private int m_Number;   <br>}
```

4： 一个方法只完成一个任务。不要把多个任务组合到一个方法中，即使那些任务非常小。

5：别在程序中使用固定数值，用常量代替。

```
Public enum MailType {  Html,  PlainText,  Attachment }
public void SendMail (string message, MailType mailType)
{
        switch ( mailType )
        {
               case MailType.Html:
                    // Do something
                    break;
               case MailType.PlainText:
                    // Do something
                    break;
               case MailType.Attachment:
                    // Do something
                    break;
               default:
                    // Do something
                    break;
         }
}
```

不正确：

```
public void SendMail (string message, string mailType)
         {
                      switch ( mailType )
                      {
                                 case "Html":
                                           // Do something
                                           break;
                                 case "PlainText":
                                           // Do something
                                           break;
                                 case "Attachment":
                                           // Do something
                                           break;
                                 default:
                                           // Do something
                                           break;
                      }
          }
```

9：视情况使用StringBuilder替代String

String对象是不可改变的。每次使用System.String类中的方法之一时，都要在内存中创建一个新的字符串对象，这就需要为该新对象分配新的空间。
在需要对字符串执行重复修改的情况下，与创建新的String对象相关的系统开销可能会非常昂贵。如果要修改字符串而不创建新的对象，则可以使用System.Text.StringBuilder类。

例如，当在一个循环中将许多字符串连接在一起时，使用StringBuilder类可以提升性能。

10：所有的类成员变量应该被声明在类的顶部，并用一个空行把它们和方法以及属性的声明区分开

11：避免在同一个文件中放置多个类

12：一个文件应该只向在一个名称空间内定义类型。避免在一个文件中使用多个名称空间

13：避免写超过5个参数的方法。如果要传递多个参数，使用结构。

14：不要手动去修改任何机器生成的代码

15：只对那些亘古不变的数值使用const关键字，例如一周的天数。

16：避免显式指定枚举的值（不要直接给枚举赋值）

例如：正确

```
1 public enum Color
2 {
3     Red,Green,Blue
4 }
```

不正确

```
1 public enum Color
2 {
3 Red=1,Green=2,Blue=3
4 }
```

```
1 public enum Color:long
2 {
3 Red,Green,Blue
4 }
```

18：对于if语句，总使用一对{}把下面的语句块包含起来，哪怕只有一条语句也是如此。

推荐：

```
1 If(1 == 1)
2 {
3        Console.WriteLine(“正确”);
4 }
```

不推荐：

```
1 If(1 == 1)
2        Console.WriteLine(“正确”);
```

19：避免使用三元条件操作符。

20：避免显示类型转换。使用as关键字安全的转换到另一个类型。

```
1 Dog dog=new GermanShepherd();
2 GermanShepherd shepherd=dog as GermanShepherd;
3 if (shepherd!=null)
4 {…}
```

21：在使用一个对象、数组或代理之前，总要检查其是否为null

22：避免在接口中包含事件。

```
//避免
string name="";
//正确
string name=String.Empty;
```

25：使用条件方法来取代显式进行方法调用排除的代码(**#if ...#endif**)

三、 ** **注释

1： 文件头部注释**

在代码文件的头部进行注释，标注出创始人、创始时间、修改人、修改时间、代码的功能，这在团队开发中必不可少，它们可以使后来维护/修改的同伴在遇到问题时，在第一时间知道他应该向谁去寻求帮助，并且知道这个文件经历了多少次迭代、经历了多少个程序员的手。

示例：

```
/**************************************************************************
 ** 作者： Eunge
 ** 创始时间：2004-6-8
 ** 修改人：Koffer
 ** 修改时间：2004-12-9
 ** 修改人：Ken
 ** 修改时间：2005-01-29
 ** 描述：
 ** 主要用于产品信息的资料录入，…
 **************************************************************************
```

我们甚至可以在这段文件头注释中加入版权信息、文件名、版本信息等。

2：函数、属性、类、公共变量注释**

请使用///三斜线注释，** 这种注释是基于XML的，不仅能导出XML制作帮助文档，而且在各个函数、属性、类等的使用中，编辑环境会自动带出注释，方便你的开发。**以protected，protected Internal，public声明的定义注释请都以这样命名方法。**

例如：

```
{
    /// <summary>
    /// 保存C扫数据
    /// </summary>
    public void SaveCScanData(){}
}
```

3：//注释**

不建议过多的使用此注释。

4：注释说明

注释应该只说明操作的一些前提假设、算法的内部信息等内容。

四、 ** 异常处理

1：不要“捕捉了异常却什么也不做“。如果隐藏了一个异常，你将永远不知道异常到底发生了没有。

2：发生异常时，给出友好的消息给用户，但要精确记录错误的所有可能细节，包括发生的时间，和相关方法，类名等。

3：只捕捉特定的异常，而不是一般的异常。

4：不必在所有方法中捕捉一般异常。不管它，让程序崩溃。这将帮助你在开发周期发现大多数的错误。

5：不必每个方法都用try-catch。当特定的异常可能发生时才使用。比如，当你写文件时，处理异常FileIOException。

6：别写太大的 try-catch 模块。如果需要，为每个执行的任务编写单独的 try-catch 模块。 这将帮你找出哪一段代码产生异常，并给用户发出特定的错误消息

如果应用程序需要，可以编写自己的异常类。自定义异常不应从基类SystemException派生，而要继承于IApplicationException。