# [C#编码规范](http://www.cnblogs.com/wulinfeng/archive/2012/08/31/2664720.html)

目录

[C#编码规范 1](#_Toc490560642)

[1     规范目的 1](#_Toc490560643)

[2     适用范围 2](#_Toc490560644)

[3     代码注释 2](#_Toc490560645)

[3.1    代码注释约定 2](#_Toc490560646)

[3.2     模块头部注释规范 2](#_Toc490560647)

[3.3     方法注释规范 3](#_Toc490560648)

[3.4     代码行注释规范 6](#_Toc490560649)

[3.5     变量注释规范 6](#_Toc490560650)

[3.6     Apimark注释 7](#_Toc490560651)

[4    命名规则 7](#_Toc490560652)

[4.1     命名的基本约定 7](#_Toc490560653)

[4.2     各种标示符类型的命名约定 9](#_Toc490560654)

[4.3     通用命名 10](#_Toc490560655)

[5    其它规范 11](#_Toc490560656)

[5.1     编程风格 11](#_Toc490560657)

[5.2     资源释放 18](#_Toc490560658)

[5.3     错误处理 19](#_Toc490560659)

[5.4     其它 19](#_Toc490560660)

[6    VS常用插件和常用技巧 20](#_Toc490560661)

[6.1     常用扩展插件 20](#_Toc490560662)

[6.2     常用快捷方式和技巧 20](#_Toc490560663)

## 1     规范目的

1. 一个软件的生命周期中，80%的花费在于维护；
2. 几乎没有任何一个软件，在其整个生命周期中，均由最初的开发人员来维护；
3. 编码规范可以改善软件的可读性，可以让程序员尽快而彻底地理解新的代码。为了执行规范，每个软件开发人员必须一致遵守编码规范；
4. 使用统一编码规范的主要原因，是使应用程序的结构和编码风格标准化，以便于阅读和理解这段代码；
5. 好的编码约定可使源代码严谨、可读性强且意义清楚，与其它语言约定相一致，并且尽可能的直观。

## 2     适用范围

1. 本规范主要以C#为开发语言的规范；
2. 由于本规范是为撰写程序而设计，所以适用于一切有关程序撰写的工作事项。对于具体的每个项目，可能需要对之进行裁剪和补存。
3. 适用人员：ERP项目组相关人员；
4. 适用产品：ERP项目组。

## 3     代码注释

### 3.1    代码注释约定

1. 所有的方法和函数都应该以描述这段代码的功能的一段简明注释开始（方法是干什么）。这种描述不应该包括执行过程细节（它是怎么做的），因为这常常是随时间而变的，而且这种描述会导致不必要的注释维护工作，甚至更糟—成为错误的注释。代码本身和必要的嵌入注释将描述实现方法。
2. 当参数的功能不明显且当过程希望参数在一个特定的范围内时，也应描述传递给过程的参数。被过程改变的函数返回值和全局变量，特别是通过引用参数的那些，也必须在每个过程的起始处描述它们。

### 3.2     模块头部注释规范

**以一个物理文件为单元的都需要有模块头部注释规范，例如：C#中的.cs文件**

用于每个模块开头的说明，主要包括：（**粗体字**为必需部分，其余为可选部分）

1. **文件名称(File Name)**： 此文件的名称
2. **功能描述(Description)**：   此模块的功能描述与大概流程说明

### 3.3     方法注释规范

　　1>  C# 提供一种机制，使程序员可以使用含有XML 文本的特殊注释语法为他们的代码编写文档。在源代码文件中，具有某种格式的注释可用于指导某个工具根 据这些注释和它们后面的源代码元素生成XML。具体应用当中，类、接口、属性、方法必须有<Summary>节，另外方法如果有参数及返回值，则必须有<Param>及<Returns>节。示例如下：

　　　　/// <summary>

　　　　/// …

　　　　/// </summary>

　　　　/// <param name=””></param>

　　　　/// <returns></returns>

　　2>  事件不需要头注解，但包含复杂处理时（如：循环/数据库操作/复杂逻辑等），应分割成单一处理函数，事件再调用函数。

　　3>  所有的方法必须在其定义前增加方法注释。

　　4>  方法注释采用 /// 形式自动产生XML标签格式的注释。

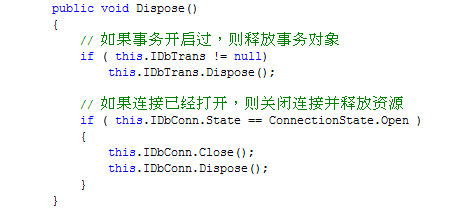
|  |  |  |
| --- | --- | --- |
| **标记** | **说明** | **备注** |
| <c> | 提供了一种将说明中的文本标记为代码的方法 |  |
| <code> | 提供了一种将多行指示为代码的方法 |  |
| <example> | 可以指定使用方法或其他库成员的示例。一般情况下，这将涉及到 <code> 标记的使用。 |  |
| <exception> | 对可从当前编译环境中获取的异常的引用。 |  |
| <include> | 得以引用描述源代码中类型和成员的另一文件中的注释。 |  |
| <list> | 用于定义表或定义列表中的标题行。 |  |
| <para> | 用于诸如<summary>、<remarks> 或 <returns> 等标记内，使您得以将结构添加到文本中。 |  |
| <param> | 应当用于方法声明的注释中，以描述方法的一个参数。 |  |
| <paramref> | 提供了一种指示词为参数的方法。 |  |
| <permission> | 得以将成员的访问记入文档。 |  |
| <remarks> | 用于添加有关某个类型的信息，从而补充由 <summary> 所指定的信息。 |  |
| <returns> | 应当用于方法声明的注释，以描述返回值。 |  |
| <see> | 得以从文本内指定链接。 |  |
| <seealso> | 对可以通过当前编译环境进行调用的成员或字段的引用。 |  |
| <summary> | 应当用于描述类型或类型成员。 |  |
| <value> | 得以描述属性。 |  |

### 3.4     代码行注释规范

　　1>  如果处理某一个功能需要很多行代码实现，并且有很多逻辑结构块，类似此种代码应该在代码开始前添加注释，说明此块代码的处理思路及注意事项等

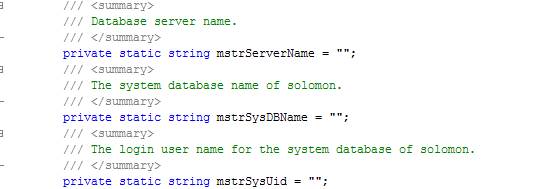
　　2>  注释从新行增加，与代码开始处左对齐

　　3>  双斜线与注释之间以空格分开，示例图如下所示：

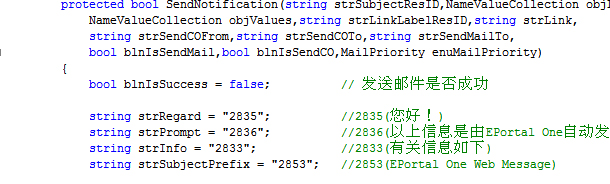


### 3.5     变量注释规范

　　1>  定义变量时需添加变量注释，用以说明变量的用途。

　　2>  Class级变量应以采用 /// 形式自动产生XML标签格式的注释，示例图如下所示：　　                                                                                                

　　3>  方法级的变量注释可以放在变量声明语句的后面，与前后行变量声明的注释左对齐，注释与代码间以Tab隔开。



### 3.6     Apiremark注释

对于方法的注释，我们不需要对参数进行注释，只需要对RequesModel和ResponseModels里的属性进行Apiremark注释

1> ResponseModel和RequesModel 需要添加Apiremark属性

2> webapi接口需要添加Apiremark属性

[Apiremark("商品管理-商品列表", typeof(VwProductPriceGetsResponse))]

## 4    命名规则

### 4.1     命名的基本约定

　　1>  要使用可以准确说明变量/字段/类的完整的英文描述符，如firstName。对一些作用显而易见的变量可以采用简单的命名，如在循环里的递增（减）变量就可以　　　　  被命名为 “i”。

　　2>  要尽量采用项目所涉及领域的术语。

　　3>  要采用大小写混合，提高名字的可读性。为区分一个标识符中的多个单词，把标识符中的每个单词的首字母大写。不采用下划线作分隔字符的写法。

　　　　有两种适合的书写方法，适应于不同类型的标识符：

**PasalCasing：**标识符的第一个单词的字母大写；

**camelCasing：**标识符的第一个单词的字母小写。

　　4>  下表描述了不同类型标识符的大小写规则：

|  |  |  |
| --- | --- | --- |
| **标识符** | **大小写** | **示例** |
| 命名空间 | Pascal | namespace Com.Techstar.ProductionCenter |
| 类型 | Pascal | public class DevsList |
| 接口 | Pascal | public interface ITableModel |
| 方法 | Pascal | public void UpdateData() |
| 属性 | Pascal | Public int Length{…} |
| 事件 | Pascal | public event EventHandler Changed; |
| 私有字段 | Camel | private string fieldName; |
| 非私有字段 | Pascal | public string FieldName； |
| 枚举值 | Pascal | FileMode{Append} |
| 参数 | Camel | public void UpdateData(string fieldName) |
| 局部变量 | Camel | string fieldName; |

　　5>  避免使用缩写，如果一定要使用，就谨慎使用。同时，应该保留一个标准缩写的列表，并且在使用时保持一致。

　　6>  对常见缩略词，两个字母的缩写要采用统一大小写的方式（示例：ioStream，   getIOStream）；多字母缩写采用首字母大写，其他字母小写的方式（示例：　　　　 getHtmlTag）；

　　7>  避免使用长名字（最好不超过 15 个字母）。

　　8>  避免使用相似或者仅在大小写上有区别的名字。

### 4.2     各种标示符类型的命名约定

**1>  程序集命名**

　　实验室名称（Lab）+ 项目名称 + 模块名称（可选），例如：

　　　　中心服务器程序集：Lab.SeverCenter；

　　　　中心服务器业务逻辑程序集：Lab.SeverCenter.Business；

**2>  命名空间命名**

　　采用和程序集命名相同的方式：实验室名称（Lab）+ 项目名称 + 模块名称。 另外，一般情况下建议命名空间和目录结构相同。例如：

　　　　中心服务器：Lab.SeverCenter；

　　　　中心服务器下的用户控件：Lab.SeverCenter.UserControl；

　　　　中心服务器业务逻辑：Lab.SeverCenter.Business；

　　　　中心服务器数据访问：Lab.SeverCenter.Data；

**3>  程序集和DLL**

　　l  大多数情况下，程序集包含全部或部分可重用库，且它包含在单个动态链接库(DLL) 中。

　　l  一个程序集可拆分到多个DLL 中，但这非常少见，在此准则中也没有说明。

　　l  程序集和DLL 是库的物理组织，而命名空间是逻辑组织，其构成应与程序集的组织无关。

　　l  命名空间可以且经常跨越多个程序集。可以考虑如下模式命名DLL：

      　　  <Company>.<Component>.dll

   　　     例：Lab.SeverCenter.dll

**4> 类和接口命名**

　　l  类的名字要用名词；

　　l  避免使用单词的缩写，除非它的缩写已经广为人知，如HTTP。

　　l  接口的名字要以字母I开头。保证对接口的标准实现名字只相差一个“I”前缀，例如对IComponent接口的标准实现为Component；

　　l  泛型类型参数的命名：命名要为T或者以T开头的描述性名字，例如：

　　　　public class List<T>

　　　　public class MyClass<Tsession>

　　l  对同一项目的不同命名空间中的类，命名避免重复。避免引用时的冲突和混淆；

**5> 方法命名**

　　l  第一个单词一般是动词；

　　l  如果方法返回一个成员变量的值，方法名一般为Get+成员变量名，如若返回的值 是bool变量，一般以Is作为前缀。另外，如果必要，考虑用属性来替代方法；

　　l  如果方法修改一个成员变量的值，方法名一般为：Set + 成员变量名。同上，考虑 用属性来替代方法。

**6> 变量命名**

　　l  按照使用范围来分，我们代码中的变量的基本上有以下几种类型，类的公有变量；类的私有变量（受保护同公有）；方法的参数变量；方法内部使用的局部变量。　　　　这些变量的命名规则基本相同，见标识符大小写对照表。区别如下：

　　　　a)   类的公有变量按通常的方式命名，无特殊要求；

　　　　b)   类的私有变量采用两种方式均可：采用加“m”前缀，例如mWorkerName;

　　　　c)   方法的参数变量采用camalString，例如workerName；

　　l  方法内部的局部变量采用camalString，例如workerName。

　　l  不要用\_或&作为第一个字母；

　　l  尽量要使用短而且具有意义的单词；

　　l  单字符的变量名一般只用于生命期非常短暂的变量：i,j,k,m,n一般用于integer；c,d,e 一般用于characters；s用于string

　　l  如果变量是集合，则变量名要用复数。例如表格的行数，命名应为：RowsCount；

　　l  命名组件要采用匈牙利命名法，所有前缀均应遵循同一个组件名称缩写列表

### 4.3     通用命名

**1>  程序集命名**

l  Gets > Reqs > Qries 检索

l  Get > Load > Find 加载

l  Create 创建

l  Update 更新

l  Abandon 作废

l  Audit 审核

## 5    其它规范

### 5.1     编程风格

**1>  变量声明：**

　　为了保持更好的阅读习惯，请不要把多个变量声明写在一行中，即一行只声明一个变量。

　　例如：

　　String strTest1, strTest2;

　　应写成：

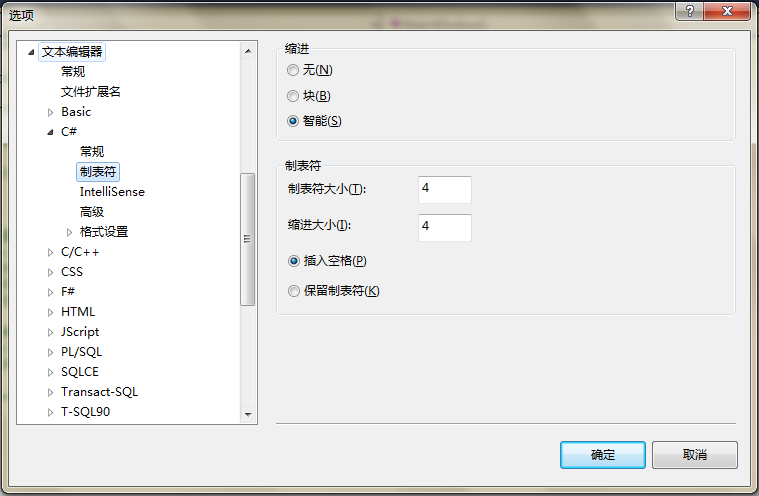
　　String strTest1;

　　String strTest2;

**2>   代码缩进：**

　　l  一致的代码缩进风格，有利于代码的结构层次的表达，使代码更容易阅读和传阅；

　　l  代码缩进使用Tab键实现，不要使用空格，为保证在不同机器上使代码缩进保持一致，特此规定C#的Tab键宽度为4个字符，设定界面如下(工具–选项)：



　　l  避免方法中有超过2个参数的情况。如果超过了，则应使用RequestModel来传递多个参数。

　　l  为了更容易阅读，代码行请不要太长，最好的宽度是屏幕宽度（根据不同的显示分辩率其可见宽度也不同）。请不要超过您正在使用的屏幕宽度。

　　l  程序中不应使用goto语句。

　　l  在switch语句中总是要default子句来显示信息。

　　l  操作符/运算符左右空一个半角空格

　　l  所有块的{}号分别放置一行，并嵌套对齐，不要放在同一行上

**3>    空白：**

　　l  空行将逻辑相关的代码段分隔开，以提高可读性。

　　l  下列情况必须使用一个空行：

　　　　a)  两个方法之间。

　　　　b)  一个方法内的两个逻辑段之间，用以提高可读性。

c)  类声明和接口声明之间。

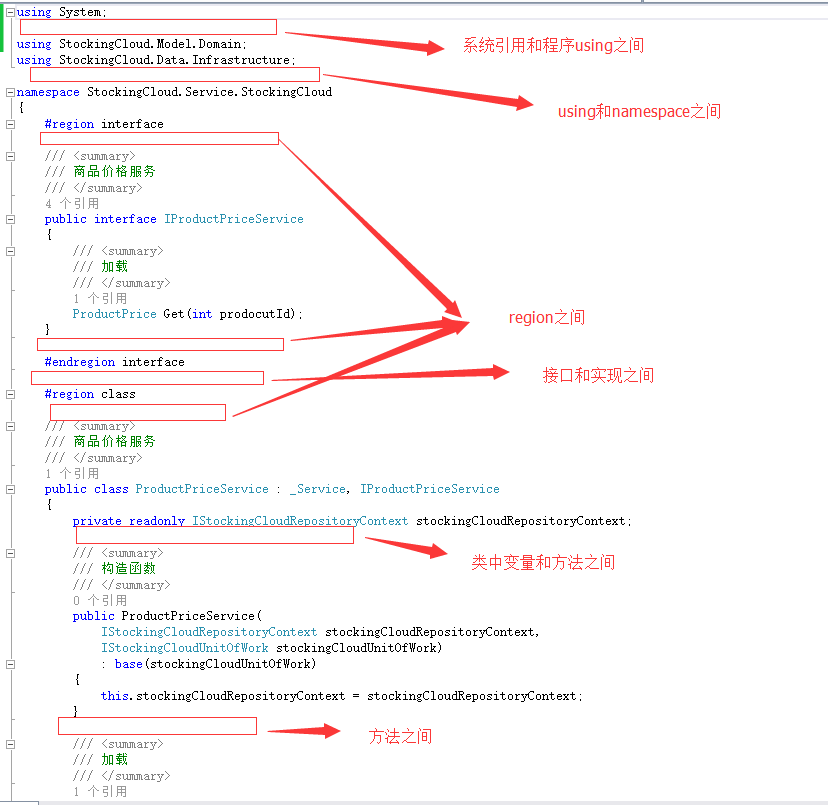
d) region和endregion。

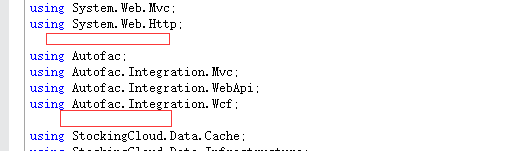
e) using和namespace之间

f)  系统using和第三方using之间，第三方using和本程序集之间

g) 类内的局部变量和方法之间

h) 两个属性之间





l  下列情况必须使用一个空格：

　 a)  空白应该位于参数列表中逗号的后面，如：

　　　　　　void UpdateData(int a, int b)

b)  所有的二元运算符，除了"."，应该使用空格将之与操作数分开。一元操作符和操作数之间不因该加空格，比如：负号("-")、自增("++")和自减("--")。例 如：

　　　　　　　　a += c + d;

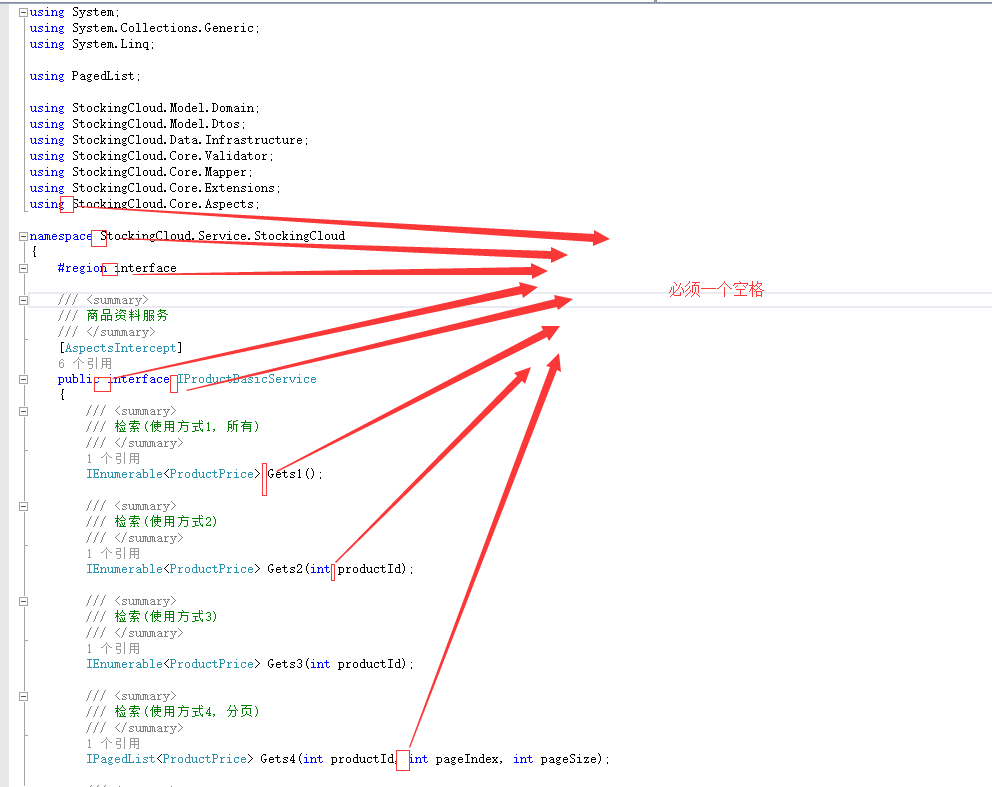
　　　　　　　　d++;

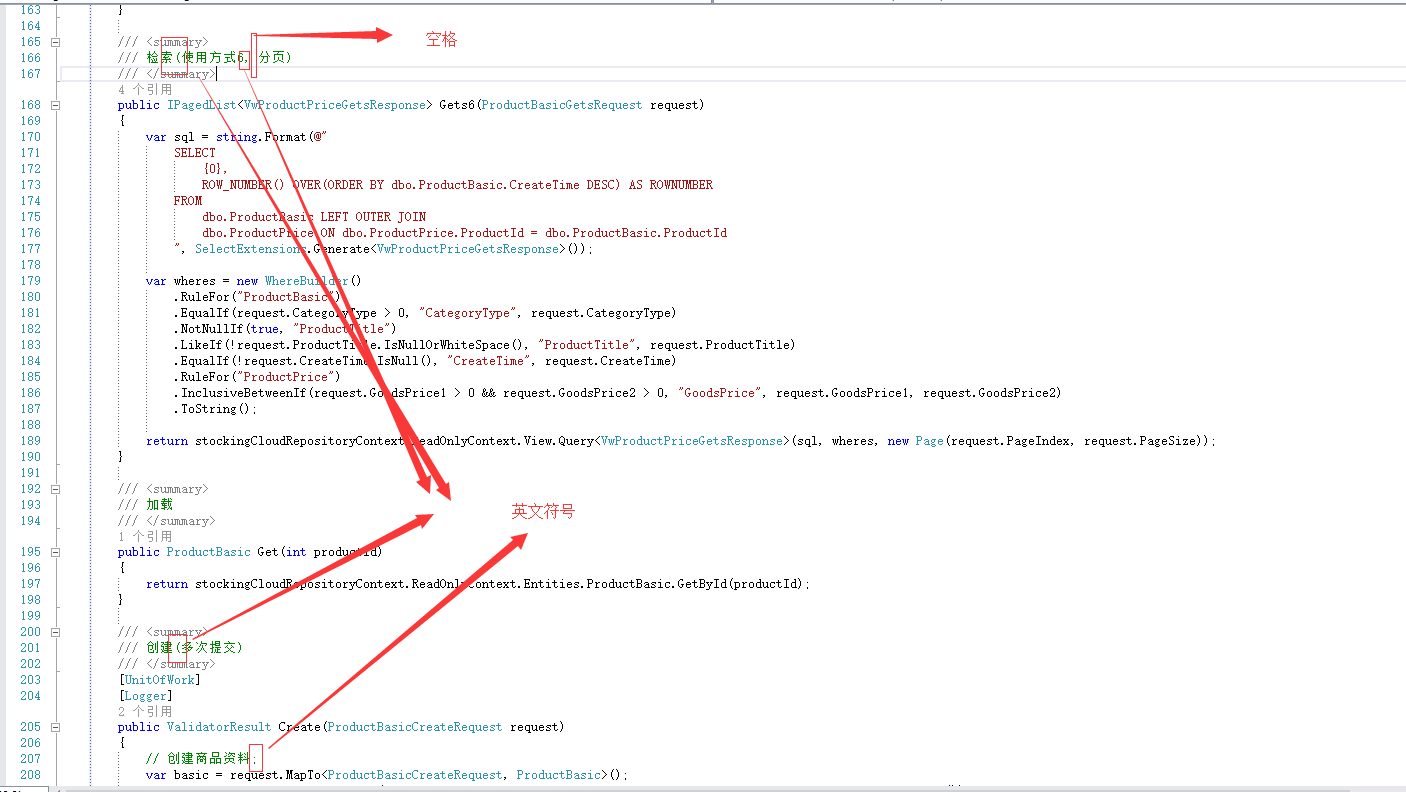
　 c)  for 语句中的表达式应该被空格分开，例如：

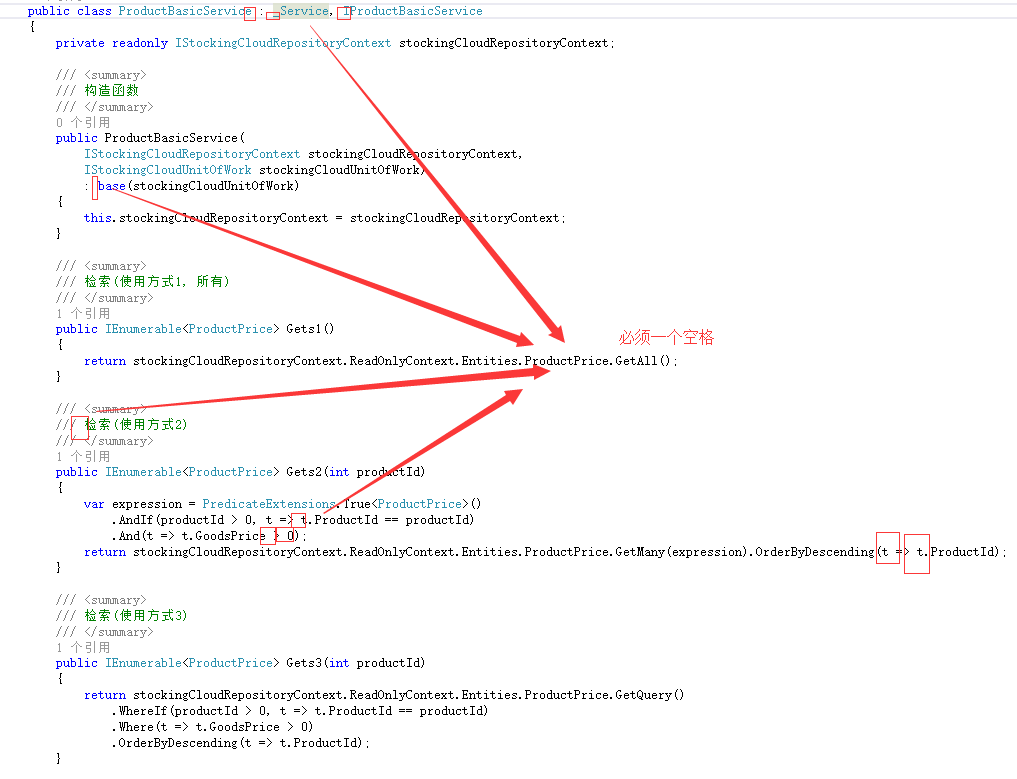
　　　　　　　　for (expr1; expr2; expr3)

d) 注释后必须一个空格

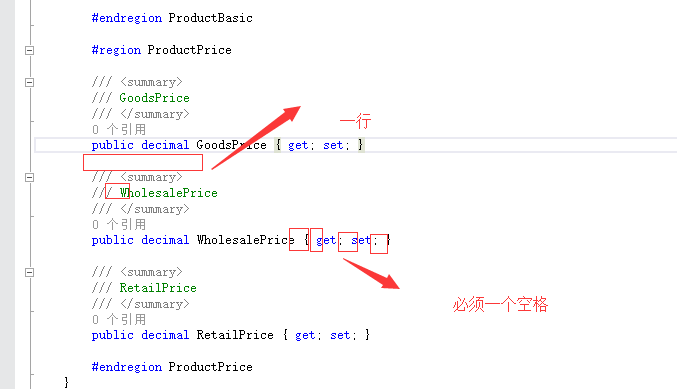
e) 英文逗号后必须添加一个空格





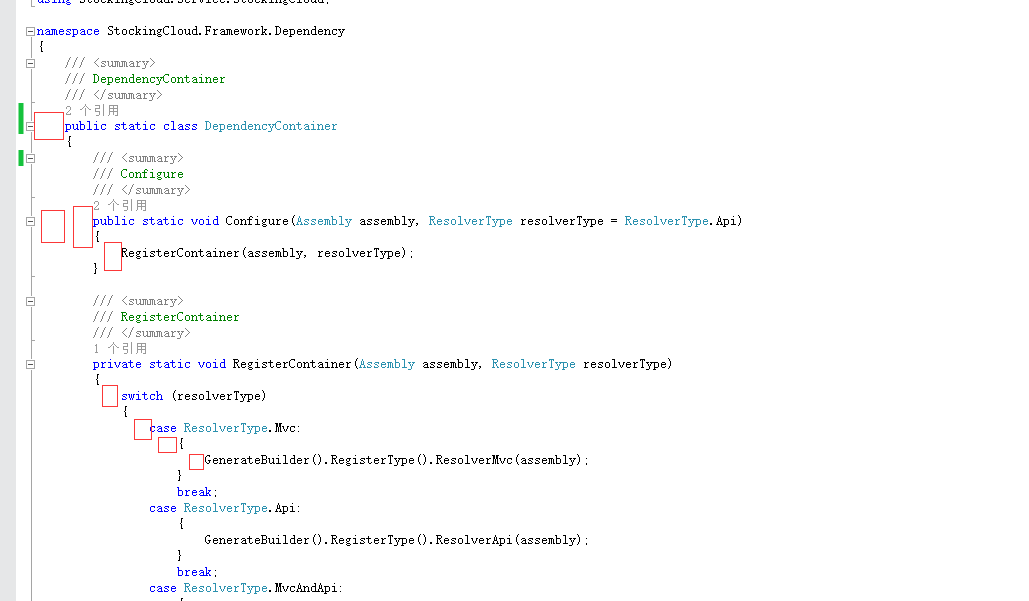


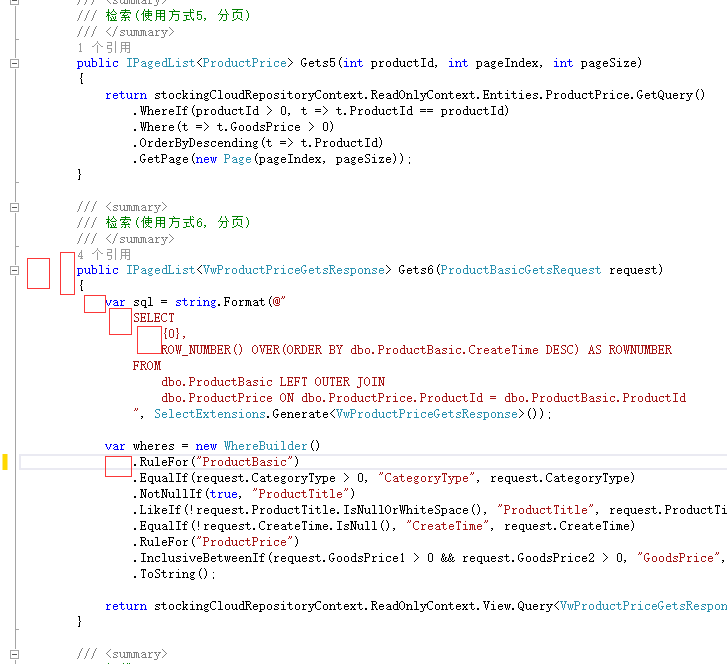




l  下列情况必须使用制表符（tab）：

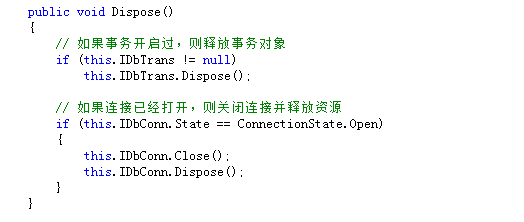
* 1. 类和方法





### 5.2     资源释放

　　所有外部资源都必须显式释放。例如：数据库连接对象、IO对象等。



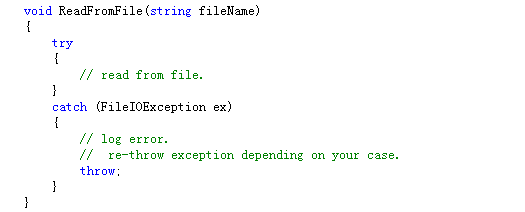
### 5.3     错误处理

　　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。

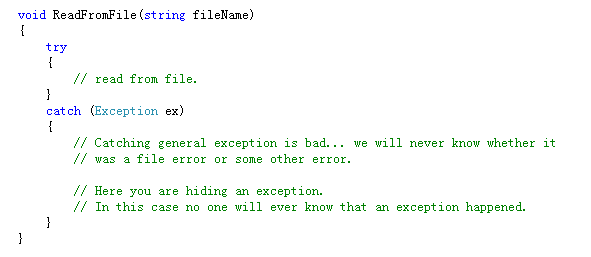
　　2>  发生异常时，给出友好的消息给用户，但要精确记录错误的所有可能细节，包括发生的时间，和相关方法，类名等。

　　3>  只捕捉特定的异常，而不是一般的异常。

**正确做法：**



**错误做法：**



### 5.4     其它

　　1>  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。

　　2>  使用C#的特有类型，而不是System命名空间中定义的别名类型。

　　3>  别在程序中使用固定数值，用常量代替。

　　4>  避免使用很多成员变量。声明局部变量，并传递给方法。不要在方法间共享成员变量。如果在几个方法间共享一个成员变量，那就很难知道是哪个方法在什么 时候修改了它的值。

　　5>  别把成员变量声明为 public 或 protected。都声明为 private 而使用 public/protected 的属性

　　6>  不在代码中使用具体的路径和驱动器名。 使用相对路径，并使路径可编程。

　　7>  应用程序启动时作些“自检”并确保所需文件和附件在指定的位置。必要时检查数据库连接。出现任何问题给用户一个友好的提示。

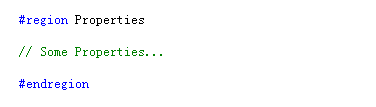
　　8>  如果需要的配置文件找不到，应用程序需能自己创建使用默认值的一份。

　　9>  如果在配置文件中发现错误值，应用程序要抛出错误，给出提示消息告诉用户正确值。

　　10>  DataColumn取其列时要用字段名，不要用索引号。  
　　　　例： 正确 DataColumn[“Name”]  
    　　　　   不好 DataColumn[0]

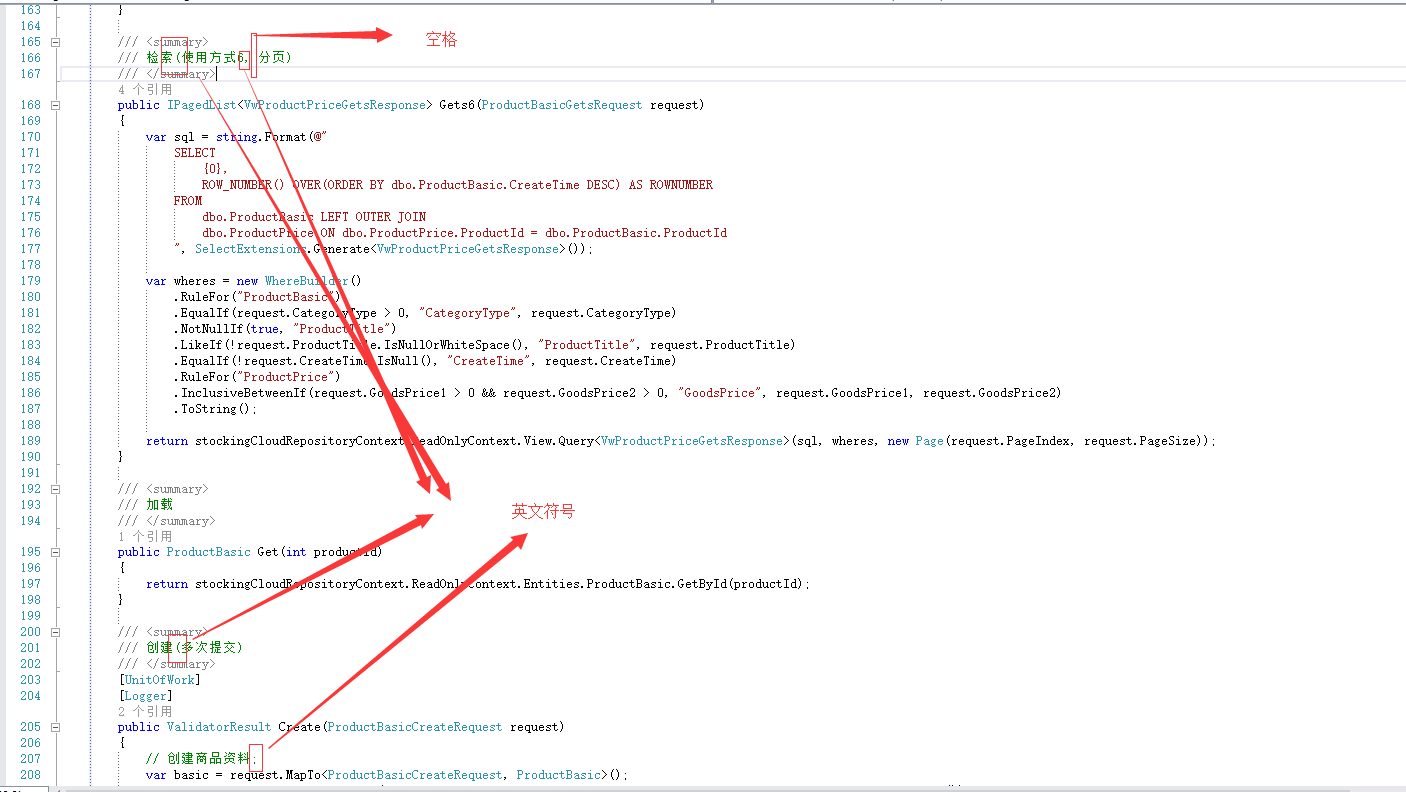
　　11>  在一个类中，字段定义全部统一放在class的头部、所有方法或属性的前面。

　　12>  在一个类中，所有的属性全部定义在一个属性块中：(说明目前属性不多，不用这样处理，endregion 后面必须与region保持一直的备注，如下endregion 就缺少 Properties)



13> 所有单行注释结尾都需要用英文的分号(;),

14> 所有注释中逗号(,),句号(.)以及括号(())都必须用英文的,并且逗号后面要有个空格

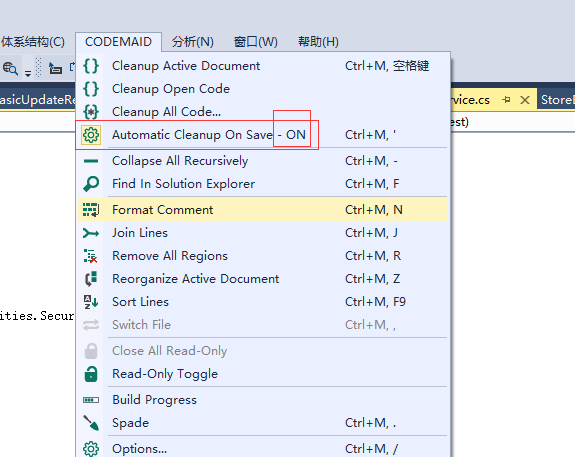


## 6    VS常用插件和常用技巧

### 6.1     常用扩展插件

1> Indent Guides

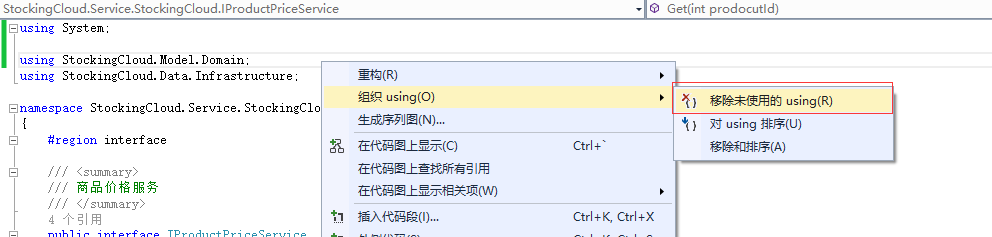
2> Codemaid 可以保存时自动格式化代码



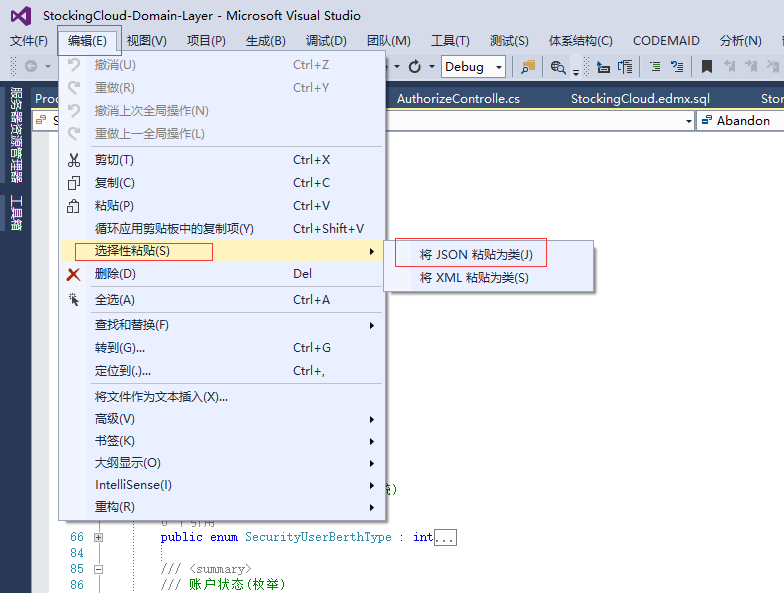
### 6.2     常用快捷方式和技巧

1> 快捷键Ctrl + K , Ctrl + D(编排整个文档)来使代码对齐

2> 移除未使用的using



3> 选择性粘贴，把json转换成类



Version：1.0.0

Date：2017.8.14