### C# 编码约定（C# 编程指南）

<https://msdn.microsoft.com/zh-cn/library/ff926074.aspx>

### tab缩进

### 花括号另起一行

switch (foo)

{

//..

}

### 常量命名 全大写

const string FOO\_BAR = "baz";

### 条件语句 有空格

if (true) {

//...

}

### 参数定义无空格

public void SetName(String name) {

// ...

}

while(isTrue) {}

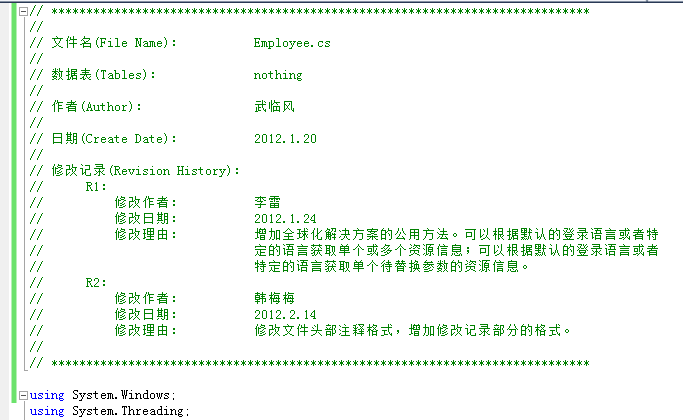
### 模块头部注释规范

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. 数据表(Tables)：             所用到的数据表，视图，存储过程的说明，如关系比较复杂，则应说明哪些是可擦写的，哪些表为只读的。
4. 作者(Author)：
5. 日期(Create Date)：
6. 参考文档(Reference)(可选)：          该档所对应的分析文档，设计文檔。
7. 引用(Using) (可选)﹕           开发的系统中引用其它系统的Dll、对象时，要列出其对应的出处，是否与系统有关﹙不清楚的可以不写﹚，以方便制作安装档。
8. 修改记录(Revision History)：若档案的所有者改变，则需要有修改人员的名字、修改日期及修改理由。
9. 分割符：\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* (前后都要)

示例如下：



### 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。

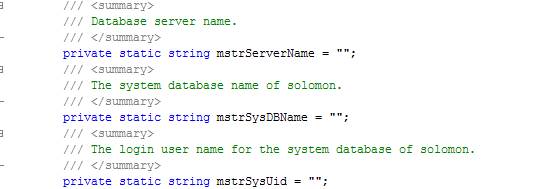
|  |  |  |
| --- | --- | --- |
| 标记 | 说明 | 备注 |
| <c> | 提供了一种将说明中的文本标记为代码的方法 |  |
| <code> | 提供了一种将多行指示为代码的方法 |  |
| <example> | 可以指定使用方法或其他库成员的示例。一般情况下，这将涉及到 <code> 标记的使用。 |  |
| <exception> | 对可从当前编译环境中获取的异常的引用。 |  |
| <include> | 得以引用描述源代码中类型和成员的另一文件中的注释。 |  |
| <list> | 用于定义表或定义列表中的标题行。 |  |
| <para> | 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。 |  |

　　　　示例图如下：

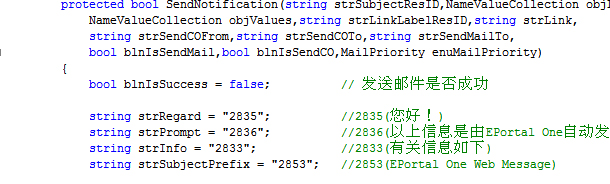


### ****变量注释规范****

　　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。



### 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。