# 代码规范

## 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代码注释

* 1. **代码注释约定**

1. 所有的函数都应该以描述这段代码功能的一段简明注释开始。这种描述不应该包括执行过程细节，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，甚至可能成为错误的注释。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。

**2.2模块头部注释规范**

用于每个模块开头的说明，主要包括：

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

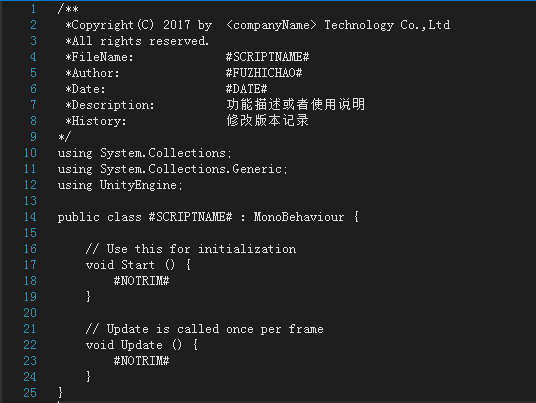
3. 作者(Author)：

4. 日期(Date)：

5. 修改记录(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。

快速方法：

1). 修改Unity3D创建脚本模板文件修改为：



Unity3D脚本模板文件目录为Unity\Editor\Data\Resources\ScriptTemplates\81-C# Script-NewBehaviourScript.cs.txt。具体路径根据Unity安装目录确定。

2). 修改Visual Studio脚本模板文件，文件路径：D:\Program Files (x86)\Microsoft Visual Studio 14.0\Common7\IDE\ItemTemplatesCache\CSharp\Code\2052\Class\Class.cs。具体路径根据安装目录确定。

**2.3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。

5.   修改任何方法，必须要添加修改记录的注释。

　　　　示例图如下：

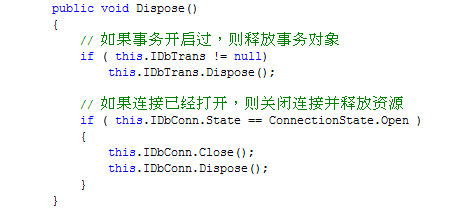


**2.4代码行注释规范**

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

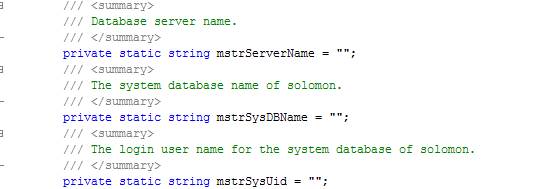
2.  注释从新行增加，与代码开始处左对齐

3.  双斜线与注释之间以空格分开，示例图如下所示：

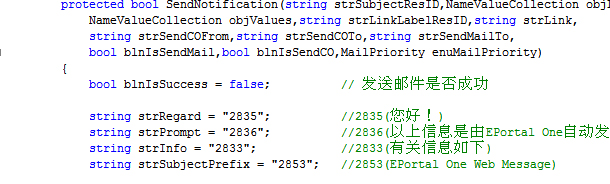


**2.5变量注释规范**

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命名规则

**3.1命名的基本约定**

1.  要使用可以准确说明变量/字段/方法/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）变量就可以命名为 “i”。

2.  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。

3.  下表描述了不同类型标识符的大小写规则：

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

4.  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）；

5.  避免使用长名字（最好不超过 15 个字母）。

6.  避免使用相似或者仅在大小写上有区别的名字。

**3.2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. 类命名**

1. 类的名字要用名词；

2. 避免使用单词的缩写，除非它的缩写已经广为人知，如HTTP。

3. 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>

　 4.  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. 不同功能的类：

数据类：以Data结尾，比如CarData；

逻辑类：以Logic结尾，比如BattleLogic；

管理类：以Manager结尾，比如UIPanelsManager；

面板类：以Panel结尾，比如ShopPanel；

控制类：以Control结尾，比如PlayerControl；

工具类：以Util结尾，比如JsonUtil；

**4. 接口命名**

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；

**5. 方法命名**

1. 不能使用单词缩写，常见缩略词除外，单词首字母需大写；

2.   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；

3.  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**

　　1.  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)   类的公有变量采用加“m”前缀，例如mWorkerName；

　　　 b)   类的私有变量采用首字母小写，例如workerName；

　　　 c)   方法的参数变量采用加“\_”前缀，例如\_workerName；

d) 方法内部的局部变量采用首字母小写，例如workerName。

　　2  尽量要使用短而且具有意义的单词；

　　3  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。

**3.3  组件名称缩写列表**

缩写的基本原则是取组件类名各单词的第一个字母，如果只有一个单词，则去掉其中的元音，留下辅音。缩写全部为小写。

|  |  |  |
| --- | --- | --- |
| 组件类型 | 缩写 | 例子 |
| Label | lbl | lblNote |
| TextBox | txt | txtName |
| Button | btn | btnOK |
| ImageButton | Ib | ibOK |
| CheckBox | cb | cbChoice |
| CheckBoxList | cbl | cblGroup |
| RadioButton | rb | rbChoice |
| RadioButtonList | rbl | rblGroup |
| Image | img | imgBeauty |
| UIPanel | panel | panelTree |
| UIGrid | grid | gridRank |
| UIScrollView | sv | svRank |
| UISprite | spr | sprHead |
| UITexture | tex | texHead |
| GameObject | go | goItem |

## 4其它规范

**4.1 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：**

　　1  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；

　　2  代码缩进使用Tab键实现，最好不要使用空格，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。

　　3  避免方法中有超过5个参数的情况，一般以2,3个为宜。如果超过了，则应使用struct或class来传递多个参数。

　　4  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。（每行代码不要　　　　超过80个字符。）

　　5  程序中不应使用goto语句。

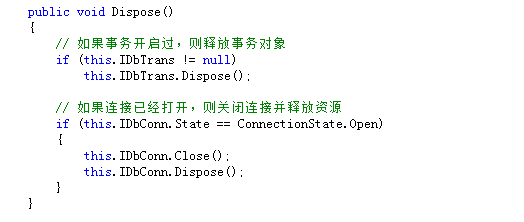
　　6  在switch语句中总是要default子句来显示信息。

　　7  操作符/运算符左右空一个半角空格

8  所有块的{}号分别放置一行，并嵌套对齐，不要放在同一行上 ，自动排版快捷键Ctrl+K、Ctrl+D

**4.2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、流对象，Socket等。



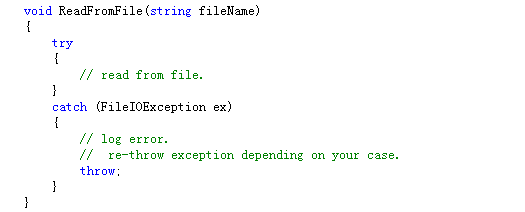
**4.3错误处理**

　　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。

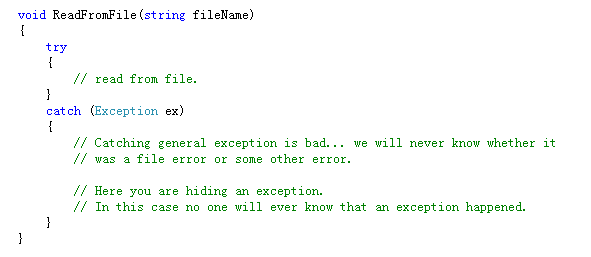
　　2.  发生异常时，给出友好的消息给用户，但要精确记录错误的所有可能细节，包括发生的时间，和相关方法，类名等。

　　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：**



**4.4其它**

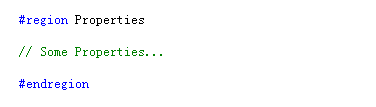
　　1.  一个方法只完成一个任务。不要把多个任务组合到一个方法中，即使那些任务非常小。

　　2.  别在程序中使用固定数值，用常量代替。

　　3.  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。

　　4.  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。

　　5.  在一个类中，所有的属性全部定义在一个属性块中：



**5工程目录结构**

Scenes： 存放所有的场景文件和场景烘焙文件；

Scripts： 存放所有的脚本文件；

Resources： 存放动态加载的资源文件；

Editor： 存放所有的编辑文件；

Plugins： 存放第三方插件；

StreamingAssets： 存放Assetbundle资源；

Shader： 存放Shader文件；

Configs： 存放配置表原文件，如Excel等；

UI： 存放跟UI相关的非动态加载的资源，如图集，字体，Perfab，贴图等；

ArtRes： 存放非动态加载或者打成assetbundle前的原资源文件，如模型资源，动画资源，粒子特效等；