OTQP-3-1 编码规范(C#)

V1.0a

**目 录**

[第一章 引言 3](#_Toc289242538)

[第二章 通用风格约定 4](#_Toc289242539)

[花括号的使用 4](#_Toc289242540)

[空格的使用 4](#_Toc289242541)

[缩进的使用 5](#_Toc289242542)

[第三章 命名约定 6](#_Toc289242543)

[第四章 注释 6](#_Toc289242544)

[第五章 文件的组织 6](#_Toc289242545)

[第六章 附录 7](#_Toc289242546)

[参考文献 7](#_Toc289242547)

[修改记录 7](#_Toc289242548)

# 引言

**文档目的**

本文档主要用于采用C#作为开发语言的编码规范，其他编码规范还有java版和php版。

**规范适用**

本规范适用于采用C#语言的软件开发团队。

**规范的表示方法**

我们通过**要**、**考虑**、**避免**、**不要**这些词把规范组织成一条条简单的建议。每条规范都描述了一种好的或者不好的做法，并用统一的方式来表示。对于好的做法，在其前面会用🗸表示，与此对应，不好的做法则用🗴表示。每一条规范的措辞也会明确表示出这条规范的重要性。

“**要…**” 描述的是必须遵循的规范。(必须这一措辞有点太强。虽然有些规范我们的确应该始终遵守。另一方面，在一些非常特殊的情况下，你可能需要违反一些“要…”规范。)

“**考虑…**” 描述的是在一般情况下应该遵守的规范，但如果完全理解规范背后的道理，并有很好的理由不遵守他时，也不要畏惧打破常规。

“**不要…**” 描述的是一些几乎绝对不应该违反的规范。

“**避免…**” 就没有那么绝对，他描述的做法虽然通常并不好，但却存在一些一直的可以违反该规范的情况。

**术语表**

* 产品库

指PMO存档开发团队最终产品和历次升级、BUG修复维护的版本库。

* 开发库

指开发团队为了在开发期间所使用的版本库。

# 通用风格约定

## 花括号的使用

**🗸要** 把左花括号放到新的一行的开始。

if (someExpression)

{

DoSomeThing();

}

**🗸要** 使右花括号与左花括号对齐。

**🗸要** 把右花括号放到新的一行的开始处。

**🗸考虑** 把只有只有一条语句的代码段和左右花括号放到同一行。属性的访问方法经常使用这一风格。

public int Foo

{

get{ return foo; }

set{ foo = value; }

}

**🗸要** 使左、右花括号但独占一行。

**🗴避免** 省略花括号，即使编程语言允许这样做。

如 if (someExpression){ DoSomThing(); }

## 空格的使用

**🗸要** 在左花括号之后和右花括号之前加一空格。

如 if (someExpression){ DoSomThing(); }

**🗴避免** 在左花括号之前加空格。

最好如此：if (someExpression){

可以接受：if (someExpression) {

**🗸要** 在形参的逗号之后加一个空格。

正确：public int Foo(char bar, int x, int y)

错误：public int Foot(char bar,int x,int y)

**🗴避免** 在左小括号之后和右小括号之前加空格。

最好如此：public int Foo(char bar, int x, int y)

可以接受：public int Foo( char bar, int x, int y )

**🗴不要** 再成员的名字和左小括号之前加空格。

正确：Foo()

错误：Foo ()

**🗴不要** 在左中括号之后和右中括号之前加空格。

正确：x = dataArray[index]

错误：x = dataArray[ index ]

**🗴不要** 在控制流程语句之前加空格。

正确：while( x==y )

错误：while ( x==y )

**🗴避免** 在二元操作符之前和之后加空格

最好如此：if( x==y ){ … }

可以接受：if( x == y ){ … }

**🗴不要** 在一元操作符之前或之后加空格

正确：if( !x ){ … }

错误：if( ! x ){ … }

## 缩进的使用

**🗸要** 用制表符(tab)来缩进。

**🗴不要** 用空格缩进。

**🗸要** 对代码块中的内容缩进。

**🗸要** 对case块缩进，尽管没有使用花括号。

如：switch( someExpression )

{

case 0:

DoSomeThing();

break;

case 1:

…

break;

}

# 命名约定

**🗸要** 在命名空间、类和成员用PascalCasing 大小写形式风格，除非是内部字段和私有字段。

**🗸要** 用camelCasing大小写风格命名内部字段和私有字段。

**🗸要** 用camelCasing大小写风格命名似有变量

**🗸要** 用camelCasing大小写风格命名成员的形式参数。

**🗴不要** 使用匈牙利方法来命名变量（也就是，不要再变量中包含变量类型）。

**🗸考虑** 给私有变量增加前缀。

**🗸要** 用有意义的，描述性的词语来命名变量

**🗸要** 使用c#中对应的别名，不要使用.NET框架中的类型名。

例如，要使用int而不是Int32，要使用object而不是Object。

# 注释

应该用注释来描述代码的用意、大致算法以及逻辑流程。最好的情况是，代码编写着之外的人能够通过独自阅读注释来理解函数的目的和行为。虽然注释并不存在一个最低标准，而且一些非常短小的函数根本不需要注释，但对大多数函数来说，通过注释反映程序员的意图和所采用的方法仍然是值得。

**🗴不要** 用注释来描述一些对任何人来说都显而易见的事儿。

**🗴不要** 把注释放到行位，除非注释非常短。

# 文件的组织

**🗴不要** 再一个文件中包含一个以上的公共类型，除非有嵌套类，或者各类型之间的不同之处仅在于泛型参数的数量。

一个文件由多个内部类型是允许的。

**🗸要** 用相同的名字命名原文件和公共类型。

**🗸要** 用相同的层次结构来组织命名空间和文件目录。

**🗸考虑** 根据下面给出的顺序和组别对成员进行分组：

* 所有字段
* 所有构造函数
* 公共及受保护属性
* 方法
* 事件
* 所有显示实现的接口成员
* 内部成员
* 私有成员
* 所有嵌套类型

**🗸要** 把不能公开访问的成员和现实实现的接口成员分别放到自己的#region块中。

**🗸考虑** 在每个组别内根据字幕顺序来组织成员

**🗸考虑** 根据由简单到复杂的顺序来组织成员

**🗸要** 把using指令放到命名空间的生命之外

using System;

namespace Onest.ProjectA

{

…

}

# 附录

## 参考文献

.NET 设计规范——.NET约定、惯用法与模式 人民邮电出版社 2006.7

## 修改记录

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本 | 修订时间 | 修订人 | 修订内容 | 核准 |
| 1.0 | 2010/6/7 | 苑旭 | 起草 |  |
| 1.0a | 2011/3/30 | 苑旭 | 修订格式 |  |