企业信息化服务事业部

C#编程规范

|  |  |  |
| --- | --- | --- |
| 文件状态：  [ ] 草稿  [√] 正式发布  [ ] 正在修改 | 文件编号： |  |
| 当前版本： | V1.0 |
| 作 者： | 杜飞 |
| 审 核 人： |  |
| 审核日期： |  |

**目 录**

[1 概述 1](#_Toc294613845)

[1.1 原则 1](#_Toc294613846)

[1.2 术语定义 1](#_Toc294613847)

[1.2.1 Pascal 大小写 1](#_Toc294613848)

[1.2.2 Camel 大小写 1](#_Toc294613849)

[1.3 文件命名组织 1](#_Toc294613850)

[1.3.1 文件命名 1](#_Toc294613851)

[1.3.2 文件注释 1](#_Toc294613852)

[2 代码外观 2](#_Toc294613853)

[2.1 列宽 2](#_Toc294613854)

[2.2 换行 2](#_Toc294613855)

[2.3 缩进 2](#_Toc294613856)

[2.4 空行 2](#_Toc294613857)

[2.5 空格 3](#_Toc294613858)

[2.6 括号 () 3](#_Toc294613859)

[2.7 花括号 {} 4](#_Toc294613860)

[3 程序注释 4](#_Toc294613861)

[3.1 文档型注释 4](#_Toc294613862)

[3.2 单行注释 5](#_Toc294613863)

[4 申明 5](#_Toc294613864)

[4.1 每行声明数 5](#_Toc294613865)

[4.2 位置 5](#_Toc294613866)

[4.3 类和接口的声明 6](#_Toc294613867)

[4.4 字段的声明 6](#_Toc294613868)

[5 命名规范 7](#_Toc294613869)

[5.1 命名原则 7](#_Toc294613870)

[5.2 大小写规则 7](#_Toc294613871)

[5.3 命名空间 8](#_Toc294613872)

[5.4 类 8](#_Toc294613873)

[5.5 接口 8](#_Toc294613874)

[5.6 枚举 9](#_Toc294613875)

[5.7 参数 9](#_Toc294613876)

[5.8 方法 9](#_Toc294613877)

[5.9 事件 9](#_Toc294613878)

[5.10 常量 10](#_Toc294613879)

[5.11 静态字段 10](#_Toc294613880)

[5.12 控件 11](#_Toc294613881)

[5.12.1 常用Web控件 11](#_Toc294613882)

[5.12.2 常用Form控件 12](#_Toc294613883)

[6 第六章　语句 13](#_Toc294613884)

[6.1 每行一个语句 13](#_Toc294613885)

[6.2 复合语句 13](#_Toc294613886)

[6.3 return 语句 13](#_Toc294613887)

[6.4 if、 if-else、if else-if 语句 14](#_Toc294613888)

[6.5 for、foreach 语句 15](#_Toc294613889)

[6.6 while 语句 15](#_Toc294613890)

[6.7 do - while 语句 15](#_Toc294613891)

[6.8 switch - case 语句 16](#_Toc294613892)

[6.9 try - catch 语句 16](#_Toc294613893)

[6.10 using 块语句 17](#_Toc294613894)

[7 第七章　新特性 17](#_Toc294613895)

[7.1 C# 2.0新增特性(常用的) 17](#_Toc294613896)

[7.2 C# 3.0 新增特性(常用的) 18](#_Toc294613897)

[8 第八章　注意点 19](#_Toc294613898)

[8.1 string str = “” 与 string str = string.Empty 19](#_Toc294613899)

[8.2 +，string.Format()，StringBuilder.Append() 20](#_Toc294613900)

[8.3 将循环次数的判断提到外面 20](#_Toc294613901)

[8.4 简化你的代码，如： 21](#_Toc294613902)

# 概述

## 原则

（1）方便代码的交流和维护

（2）不影响编码的效率，不与大众习惯冲突。

（3）使代码更美观、阅读更方便

（4）使代码的逻辑更清晰、更易于理解

## 术语定义

### Pascal 大小写

标识符的首字母和后面连接的每个单词的首字母都大写。例如：BackColor。

### Camel 大小写

标识符的首字母小写，而后面连接的单词的首字母都大写。例如： backColor。

## 文件命名组织

### 文件命名

【1】文件命名遵从Pascal命名法，无特殊情况，扩展名小写。

### 文件注释

【1】在每个文件头必须包含以下注释说明

/\* =============================================

\* 创建人：杜飞

\* 创建时间： 2012/11/07

\* 功能描述：费用结果通用表

\* 修改记录： 杜飞 2012-03-01 增加ExcelExport()方法

\* 杜飞 2012-03-02 删除ExcelExport()方法

\* 杜飞 2012-03-03 修改ExcelExport()方法

=============================================\*/

【2】文件功能描述只需简述，具体详情在类的注释中描述。

# 代码外观

## 列宽

代码列宽控制在110字符左右。

## 换行

当表达式超出或即将超出规定的列宽，遵循以下规则进行换行

1、在逗号后换行。

2、 在操作符前换行。

3、规则1优先于规则2

4、等号后缩进

当以上规则会导致代码混乱的时候自己采取更灵活的换行规则。

## 缩进

缩进应该是每行一个Tab(4个空格)，不要在代码中使用Tab字符。

Visual Studio.Net设置：工具->选项->文本编辑器->C#->制表符->插入空格

## 空行

在以下情况下使用一个空行：

1、方法与方法、属性与属性之间。

2、方法中变量声明与语句之间。

3、方法中不同的逻辑块之间。

4、方法中的返回语句与其他的语句之间。

5、属性与方法、属性与字段、方法与字段之间。

## 空格

在以下情况中要使用到空格：

1、关键字和左括符 “(” 应该用空格隔开。如： while (true)

2、多个参数用逗号隔开，每个逗号后都应加一个空格。

3、除了 . 之外，所有的二元操作符都应用空格与它们的操作数隔开。一元操作符、++及--与操作数间不需要空格。如

a += c + d;

a = (a + b) / (c \* d);

while (d++ = s++)

{

n++;

}

PrintSize(“size is “ + size + “\n”);

4、语句中的表达式之间用空格隔开，如：

for (expr1; expr2; expr3)

## 括号 ()

1、 左括号“(” 不要紧靠关键字，中间用一个空格隔开。

2、 左括号“(” 与方法名之间不要添加任何空格。

3、 没有必要的话不要在返回语句中使用()。如

if (condition)

Array.Remove(1)

return 1

## 花括号 {}

1、 左花括号 “{” 放于关键字或方法名的下一行并与之对齐。如

if (condition)

{

}

public int Add(int x, int y)

{

}

2、 左花括号 “{” 要与相应的右花括号 “}”对齐。

3、 通常情况下左花括号 “{”单独成行，不与任何语句并列一行。

4、 if、while、do语句后一定要使用{}，即使{}号中为空或只有一条语句。如：

if (somevalue == 1)

{

somevalue = 2;

}

# 程序注释

## 文档型注释

采用.Net已定义好的Xml标签来标记，在声明接口、类、方法都应该使用该类注释。推荐每个方法都添加Region，并添加描述，方便折叠查找。如

#region 获取案例项目列表

/// <summary>

/// 获取案例项目列表

/// </summary>

/// <param name="teacherId">教师编号</param>

/// <param name="orgId">机构编号</param>

/// <returns>返回案例项目数据列表</returns>

public string GetCases(int teacherId, int orgId)

{

return CaseDAL.GetCases(teacherId, orgId);

}

#endregion

## 单行注释

该类注释用于：

1、方法内的代码注释。如变量的声明、代码或代码段的解释。例：

// 学生数量

private int studentNumber;

# 申明

## 每行声明数

一行只建议声明一个，并按字母顺序排列。如

int level; //推荐

int size; //推荐

int x, y; //不推荐

## 位置

变量建议置于块的开始处，不要总是在第一次使用它们的地方才声明。如

void MyMethod()

{

int int1 = 0; // beginning of method block

if (condition)

{

int int2 = 0; // beginning of "if" block

...

}

}

## 类和接口的声明

1、类和接口中的方法名与其后的左括号间没有任何空格。

2、左花括号 “{” 出现在声明的下行并与之对齐，单独成行。

3、方法间用一个空行隔开。

## 字段的声明

不要使用 public 或 protected 的实例字段，考虑为字段提供 get 和set 属性访问器，而不是使它们成为公共的。例：

public class Control: Component

{

private int handle;

public int Handle

{

get

{

return handle;

}

}

}

# 命名规范

## 命名原则

所有名称必须有一定的意义，如GetStudentInfo()

## 大小写规则

下表汇总了标识符的大小写规则，并提供了不同类型标识符的示例。

|  |  |  |
| --- | --- | --- |
| **标示符** | **大小写方式** | **示例** |
| 类 | Pascal | Student |
| 枚举类型 | Pascal | ErrorLevel |
| 枚举值 | Pascal | FatalError |
| 事件 | Pascal | ValueChange |
| 只读的静态字段 | Pascal | RedValue |
| 接口 | Pascal | IDisposable |
| 方法 | Pascal | InfoGet |
| 文件名 | Pascal | StudentInfoAdd |
| 命名空间 | Pascal | System.Drawing |
| 参数 | Camel | typeName |
| 属性 | Pascal | BackColor |
| 变量 | Camel | backColor |
| 常量 | Pascal | STUDENT\_NUMBER |

## 命名空间

1、命名命名空间一般是使用公司名称，后跟技术名称和可选的功能与设计，如下。

namespace Langchao.Procurement //浪潮公司的采购单管理系统

namespace Langchao.Procurement.DataRules //浪潮公司的采购单管理系统的业务规则模块

2、命名空间使用Pascal大小写。

3、命名空间和类不能使用同样的名字。例如，有一个类被命名为Debug后，就不要再使用Debug作为一个命名空间名。

## 类

1、使用 Pascal 大小写。

2、用名词或名词短语命名类。

3、使用全称避免缩写，除非缩写已是一种公认的约定，如URL、HTML

4、不要使用类型前缀，如在类名称上对类使用 C 前缀。例如，使用类名称 FileStream，而不是 CFileStream。

5 、不要使用下划线字符 (\_)。

## 接口

1. 用名词或名词短语命名接口。例如IComponent 。
2. 使用 Pascal 大小写。

3、少用缩写。

4、给接口名称加上字母 I 前缀，以指示该类型为接口。

5、不要使用下划线字符 (\_)。

## 枚举

1、使用 Pascal 大小写。

2、少用缩写。

3、不要在 Enum 类型名称上使用 Enum 后缀。

## 参数

1、使用描述性参数名称。参数名称应当具有足够的描述性，以便参数的名称及其类型可用于在大多数情况下确定它的含义。

2、使用 Camel 大小写。

3、不要使用保留的参数。保留的参数是专用参数，如果需要，可以在未来的版本中公开它们。

4、不要给参数名称加匈牙利语类型表示法的前缀。

## 方法

1、使用动词或动词短语命名方法。

2、使用 Pascal 大小写。

如：(可以使同一个模块方法排序在一起，便于查找)

EmailSend()

EmailReceive()

EmailCreate()

EmailGetByID(int id)

## 事件

1. 对事件处理程序名称使用 EventHandler 后缀。
2. 指定两个名为 sender 和 e 的参数。sender 参数表示引发事件的对象，

对 e 参数类型使用适当而特定的事件类。

3、用 EventArgs 后缀命名事件参数类。

4、考虑用动词命名事件。

5、使用动名词（动词的“ing”形式）创建表示事件前的概念的事件名称，用过去式表示事件后。例如，可以取消的 Close 事件应当具有 Closing 事件和 Closed 事件。不要使用BeforeXxx/AfterXxx 命名模式。

6、不要在类型的事件声明上使用前缀或者后缀。例如，使用 Close，而不要使用 OnClose。

## 常量

1. 所有单词大写，多个单词之间用 "\_" 隔开。

如： public const string PAGE\_TITLE = "Welcome";

## 静态字段

1、使用名词、名词短语或者名词的缩写命名静态字段。

2、使用 Pascal 大小写。

3、对静态字段名称使用匈牙利语表示法前缀。

4、建议尽可能使用静态属性而不是公共静态字段。

## 控件

### 常用Web控件

|  |  |
| --- | --- |
| **控件名** | **缩写** |
| Label | lbl |
| TextBox | txt |
| Button | btn |
| LinkButton | lbtn |
| ImageButton | ibtn |
| DropDownList | ddl |
| ListBox | lst |
| DataList | dl |
| CheckBox | chk |
| CheckBoxList | chkls |
| RadioButton | rbtn |
| RadioButtonList | rbtnls |
| Image | img |
| Panel | pnl |
| GridView | gv |
| Repeater | rpt |
| Table | tbl |
| AdRotator | ar |
| ValidatorSummary | vs |
| CompareValidator | cv |
| RangeValidator | rv |

### 常用Form控件

像Label这样在Web中也出现的控件其命名规则跟Web中一样，下面仅列出在Web中没有的控件的命名规则，如下图：

|  |  |
| --- | --- |
| **控件名** | **缩写** |
| LinkLabel | llbl |
| MainMenu | mmnu |
| GroupBox | gbx |
| PictureBox | pic |
| DataGrid | dg |
| CheckedListBox | clst |
| ComboBox | cbo |
| ListView | lvw |
| TreeView | tvw |
| TabControl | tctl |
| DateTimePicker | dtp |
| HscrollBar | hsb |
| VscrollBar | vsb |
| Timer | tmr |
| ImageList | ilst |
| ToolBar | tlb |
| StatusBar | stb |
| OpenFileDialog | odlg |
| SaveFileDialog | sdlg |
| ColorDialog | cdlg |

# 第六章　语句

## 每行一个语句

每行最多包含一个语句。如

a++; //推荐

b--; //推荐

a++; b--; //不推荐

## 复合语句

复合语句是指包含"父语句{子语句;子语句;}"的语句，使用复合语句应遵循以下几点：

1、子语句要缩进。

2、左花括号“{” 在复合语句父语句的下一行并与之对齐，单独成行。

3 、即使只有一条子语句要不要省略花括号“ {}”。 如

while (d + = s++)

{

n++;

}

## return 语句

return语句中不使用括号，除非它能使返回值更加清晰。如

return;

return myDisk.size();

## if、 if-else、if else-if 语句

if、 if-else、if else-if 语句使用格式

if (condition)

{

statements;

}

if (condition)

{

statements;

}

else

{

statements;

}

if (condition)

{

statements;

}

else if (condition)

{

statements;

}

else

{

statements;

}

## for、foreach 语句

for 语句使用格式

for (initialization; condition; update)

{

statements;

}

foreach 语句使用格式

foreach (object obj in array)

{

statements;

}

注意 1、在循环过程中不要修改循环计数器。

2、对每个空循环体给出确认性注释。

## while 语句

while 语句使用格式

while (condition)

{

statements;

}

## do - while 语句

do - while 语句使用格式

do

{

statements;

} while (condition);

## switch - case 语句

switch - case 语句使用格式

switch (condition)

{

case 1:

statements;

break;

case 2:

statements;

break;

default:

statements;

break;

}

注意： 1、语句switch中的每个case各占一行。

2、语句switch中的case按字母顺序排列。

3、为所有switch语句提供default分支。

4、所有的非空 case 语句必须用 break; 语句结束。

## try - catch 语句

try - catch 语句使用格式

try

{

statements;

}

catch (ExceptionClass e)

{

statements;

}

finally

{

statements;

}

## using 块语句

using 块语句使用格式

using (object)

{

statements;

}

# 第七章　新特性

## C# 2.0新增特性(常用的)

自动属性: 在C# 1.0中我们定义一个Name属性常用如下语句

private string name;

public string Name

{

get

{

return name;

}

set

{

name = value;

}

}

而在C# 2.0中我们可以简写成如下

public string Name { get; set; }

## C# 3.0 新增特性(常用的)

1、 对象初始化：

在以前的C#版本中声明一个对象并初始化数据时，需要先New一个对象，然后再给其成员赋值：

public class Person

{  
public string Name {get;set;}

public int Age {get;set;}

}

Person person1 = new Person();

person1.Name = “Jack” ;

person1.Age = 25 ;

而在C# 3.0 中只需要如下声明即可:

Person person1= new Person {Name="Jack",Age=25};

2、集合初始化：

与对象初始化语法类似的是集合初始化。这个语法使得我们可以用简单的数组类型来初始化一个泛型容器。

// 初始化一个普通数组

int [] arr = { 0, 1, 2, 3, 4, 5, 6, 7, 8};

 // 初始化一个int型泛型List<>

 List<int> arr = new List<int> { 0, 1, 2, 3, 4, 5, 6, 7, 8};

如果要初始化一个Person集合，可以使用如下声明：

List<Person> people = new List<Person>

{

new Person (){ Name = "Jack", Age=20 },

new Person (){ Name = "Tom", Age=23 },

new Person (){ Name = "Lily", Age=22 }

};

例子: 在我们的数据库访问层中可能会经常出现以下语句：

List<SqlParameter> parames = new List<SqlParameter>();

parames.Add(new SqlParameter("@ImageId", model.ImageID));

parames.Add(new SqlParameter("@ImageTime", model.ImageTime));

现在你可以用集合初始化的特性简写成：

List<SqlParameter> parames = new List<SqlParameter>

{

new SqlParameter("@ImageId", model.ImageID),

new SqlParameter("@ImageTime", model.ImageTime)

};

# 第八章　注意点

## string str = “” 与 string str = string.Empty

string str = “” 会给变量申请一个为空的内存空间，而后者则不会分配内存空间，所以申明一个空字符串时请用string str = string.Empty

## +，string.Format()，StringBuilder.Append()

1、对于少量固定的字符串拼接，如string str= "a" + "b" + "c"，系统会优化成str= String.Concat("a","b","c")，不会新建多个字符串，但如果写成string str="a"; str +="b"; str +="c"，则会创建三个新的字符串。对于大量字符串拼接的情况，应该使用StirngBulider。

2、分析一下.NET 的源代码就不难发现，string.Format 其实调用的也是StringBuilder，只不过Format 调用的是AppendFormat方法。比用“+”的拼接方式高效，并且代码易于阅读。

string str = string.Format("{0}{1}{2}","a","b","c");

3、字符串较少时，可以使用string.Format()代替，但当有大量的字符串操作时，比如在循环体内，则必须使用StringBuilder.Append()。

## 将循环次数的判断提到外面

for (int i = 0; i< arr.Length; i++)

{

…

}

此代码执行时每循环一次都要计算一次数组的长度，影响效率，应将计算数组的长度提到循环外

int len = arr.Length;

for (int i = 0; i< len; i++)

{

…

}

## 简化你的代码，如：

1、

if (a==b)

{

return true;

}

else

{

return false;

}

请直接写成：

return a==b ;