|  |
| --- |
| C# |
| 代码规范 |

**文档信息：**

|  |  |
| --- | --- |
|  |  |
| 文档名称 | C# 代码规范 |
| 描述 | 该文档描述了在项目开发过程中，开发人员编写代码所应遵循的规范，包括命名规则、代码结构等 |
| 负责人 | 蔡俊斌 |
| 状态 | 最终版 |

**文档变更历史：**

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **时间** | **修改人** | **章节** | **描述** |
|  |  |  |  |
|  |  |  |  |

**文档路径：**

|  |
| --- |
|  |

**审核结果：**

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **审核人** | **意见** | **签名档** |
|  |  |  |
|  |  |  |

**目录**

[1 编程风格 4](#_Toc301981815)

[1.1 统一编程风格的意义 4](#_Toc301981816)

[1.2 变量命名规则 4](#_Toc301981817)

[1.3 函数命名规则 4](#_Toc301981818)

[1.4 类命名规则 4](#_Toc301981819)

[1.5 常见语句书写规则 5](#_Toc301981820)

[注释风格 6](#_Toc301981821)

[2 代码组织 6](#_Toc301981822)

[3 代码优化 7](#_Toc301981823)

[3.1 代码优化的意义 7](#_Toc301981824)

[3.2 函数内的代码优化 7](#_Toc301981825)

[3.3 类内的代码优化 7](#_Toc301981826)

[3.4 类之间的代码优化 8](#_Toc301981827)

[4 调试技巧 8](#_Toc301981828)

[4.1 编译时的错误 8](#_Toc301981829)

[4.2 运行时的错误 8](#_Toc301981830)

[4.3 C#常见问题 9](#_Toc301981831)

# 编程风格

## 统一编程风格的意义

* 增加开发过程代码的强壮性、可读性、易维护性
* 减少有经验和无经验开发人员编程所需的脑力工作
* 为软件的良好维护性打下好的基础
* 在项目范围内统一代码风格
* 通过人为以及自动的方式对最终软件应用质量标准
* 使新的开发人员快速适应项目氛围
* 支持项目资源的复用：允许开发人员从一个项目区域（或子项目团队）移动到另一个，而不需要重新适应新的子项目团队的氛围
* 一个优秀而且职业化的开发团队所必需的素质

## 变量命名规则

* 类的私有变量用小写，公有变量名每个词的首字母大写，常量则全部大写。
* 变量名要能表达其含义或者用其类型的缩写。例如：

SqlCommand cmd = new SqlCommand();

## 函数命名规则

* 函数名用首字母大写的英文单词组合表示（如用动词+名词的方法），其中至少有一个动词
* 应该避免的命名方式
  + - * 和继承来的函数名一样。即使函数的参数不一样，也尽量不要这么做，除非想要重载它
      * 只由一个动词组成，如：Save、Update。改成如：SaveValue、UpdateDataSet则比较好
* 函数参数的命名规则
  + - * 函数参数应该具有自我描述性，应该能够做到见其名而知其意

## 类命名规则

* 类名中不要出现下划线
* 类名首字母大写
* 类名是有意义的英文单词
* 所有定义的借口名一定以大写字母I打头。比如IFly

## 数据库视图命名规则

* 均以大写字母V开头接一个有意义的名词
* 如VUserInfo

## 常见语句书写规则

如下表所示。

|  |  |
| --- | --- |
| 语句 | 提倡的风格 |
| if | if (condition)  {  statements;  }  else  {  statements;  } |
| for | for(initialization; condition; update)  {  statements;  }  禁止使用形如 for(int i = 0;i < 10;i++ )的句子，应该用有意义的常量代替数字10。 |
| foreach | foreach(something in collection)  {  statements;  } |
| switch | switch(…)  {  case ..:  break;  case …:  break;  default:  } |
| while | while(..)  {  statements;  } |
| do-while | do  {  statements;  }  while(condition); |
| try-catch | try  {  statements;  }  catch(Exception e)  {  handle exception;  } |
| 同一代码块内的不同逻辑块之间应空一行 | {  do statement1;  do statement2;  } |
| 函数与函数之间至少空一行，但不超三行 |  |
|  |  |

## 注释风格

* 注释应该正确、简洁、有重点、 无歧义
* 应该写优雅的、可读性良好的代码，而不是为玄妙、晦涩的代码写注释
* 原则上应尽量减少程序体内代码的注释，应该保持代码本身的直接可读性
* 函数的注释，可以只对public或者重要的private函数进行注解
* 如果函数不能一眼看出其功能，或者控制逻辑比较复杂，则应该添加注释
* 均用//形式注释代码

# 代码组织

代码组织是对整个项目的代码进行整理，使之更加有序。实现类似功能的文件应该放在同一个文件夹中或者同一个项目中。例如，可把整个项目分为以下几个层次：

* DataAccess层

这一层的类负责与数据库的连接，以Common层对象为媒介读取、更新、添加、删除数据库对象。为Bussiness层提供数据服务。

* Bussiness Logic 层

如果需要的话可以分为以下两层，也可以合为一层。

* + - * Business Rule层

包含各种商务逻辑和规则。

* + - * Business Façade层

提供给UI层所有的系统接口，这一层抽象出了UI层所需要用到的功能。这一层的类可以通过继承MarshalByRefObject类，支持Remoting，配置到专门的应用程序服务器上。

* UI层

只调用Bussiness层和SystemFramework层的接口，实现用户界面。包括：

* + - * WinUI
      * WebUI
      * WebService（并不是用户界面，但是逻辑上属于这一层）

# 代码优化

## 代码优化的意义

* 仅仅对符合功能说明书的要求、能正确运行的代码进行优化是有意义的
* 代码优化能减少冗余代码的数量，用更少的代码来实现同样的功能
* 提高代码的内聚程度，减少耦合程度
* 对代码的抽象能提高代码的重用度，对今后其他项目的进度有非常重要的意义

## 函数内的代码优化

* 去掉从来没有用到过的参数
* 始终进行参数检验。不要认为只有我才会调用这个函数，我能够保证参数的有效性。事实上很多运行错误就是没有对参数进行检验。对于传入了非法值的函数调用，可以返回一个对调用无意义的值（如：null，-1），或者干脆抛出一个异常
* 函数的参数不宜过多，如果实在是太多，可以考虑将这些参数封装在一个类中，然后将这个类的某个实例作为参数传入函数
* 如果函数从来不会修改某个参数的值，则应该尽量将参数声明为const
* 如果函数中用到的类成员变量或者其他全局变量可以用传入参数的方式代替，则用参数代替，这样可以减少该函数和外界的关系，提高内聚
* 一个单一的函数的代码量不宜过多。如果实在很多，则可以把它切分成小的函数，例如长的switch语句是最容易切分的
* 单个函数中尽量避免相同的代码，可以用条件语句或者抽取出来作为函数的方法消除这些冗余
* 及时释放不必要的资源。比如数据库用完了要关闭连接，将文件读取完了也要关闭输入流。

## 类内的代码优化

* 只有类对外的接口才声明为public
* 在类的成员函数中如果存在着相同的代码，则将其抽取成为private的成员函数，以减少代码的冗余，保持在一个类中没有相同的两份代码的副本
* 尽量减少成员函数之间的依赖，特别是对成员变量值的依赖
* 禁止使用public的成员变量

## 类之间的代码优化

* 类应该是一个实体，具有自己的数据和对这些数据的操作
* 把界面操作和数据处理分离在两个类中是比较好的做法
* 对于不同类之间有相同代码的情况，有以下几种处理方法：
  + - * 将相同的代码抽象出来作为父类，其他的类从中继承，由此来共享代码
      * 将相同的代码抽象出来作为一个新类，其他类中声明一个该类的变量，由此来共享代码

这两种方法各有利弊，前种方法比较适于当共享代码在调用之前必须做特殊的初始化，而这些初始化可能很难用函数调用来完成，这时父类的初始化代码中可以加入一个虚拟函数，所有的子类都重载该函数，做特定的初始化；后种方法可以封装得很彻底，只暴露出对外的接口，和其他类的耦合程度比较小

* 任何重复的代码都可以抽取出来，不仅仅是对数据进行处理的代码，界面代码同样可以抽取出来
* 如果许多类都有做类似事情的函数，名称相同、内部具体的操作不同，这时候可以将这些函数提取出来作为一个接口。其他类都从中继承，然后根据自己的要求来实现之

# 调试技巧

## 编译时的错误

* 始终在“输出”窗口中看程序编译的输出，“任务列表”窗口中经常会遗留以前编译后留下来的消息
* 认真查看编译输出的错误消息，掌握正确的错误地点和信息
* 当碰到莫名其妙的编译时的错误应

1. 重新编译整个项目或者解决方案。
2. 关闭Visual Studio.NET，然后再打开。
3. 重新启动计算机。
4. 保证编译出来的程序不在运行中或者所有的输出文件的属性都是可写的。

## 运行时的错误

* 首先要读取异常信息，猜测大概的发生地和发生原因
* 仔细读发生异常处源代码
* 在相应处设置断点，然后单步运行
* 如果还是找不出错误，可以请同事帮忙。当着同事的面讲解自己的源代码，旁观者看得最清
* 配置问题和数据库中数据的错误也会导致运行时的错误

## C#常见问题

* C#中控件的消息处理是立即的。也就是说，如果对某个控件的某个消息写了消息处理函数，然后假如当程序中某处的代码A引发了该消息时，程序流程会立即跳转到该消息的消息函数中去，如果这时消息函数中发生异常，即使代码A处于异常块中，该异常也无法捕获。所以如果出现在给控件的某个属性赋值后发生异常的情况，则请找一下是否已经对该控件的该属性写了消息函数（别忘了在父类也许会有），如果有的话，则应在这个消息处理函数中也加上断点
* 注意集成环境中窗体设计器的副作用。对于处在InitializeComponent中的代码，如果需要做修改，尽量先将其搬到函数外面来，否则，不能保证修改过的代码不被集成环境改回来或者删掉
* C#中很多异常都是由于强制转换产生的，所以对强制转换一定要放在异常处理块中