区域卫生研发部研发规范

C#

2018年07月

修改历史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版本号** | **修改人** | **修改日期** | **修改内容** | **审批人** |
| 0.1 | 潘豪格 | 2018/06/07 | 创建 |  |
| 0.2 | 黄灿良 | 2018/07/04 | 初稿 |  |
| 1.00 | 潘豪格 | 2018/07/17 | 魏星星、黄灿良、刘建峰共同评审修订 |  |
| 1.01 | 潘豪格 | 2018/07/24 | 代码优化中增加方法调用层次及请求处理异常日常要求  命名空间增加文件夹相关推荐 |  |

目录

[**1.** **命名规范** 4](#_Toc519669643)

[1.1. 总则 4](#_Toc519669644)

[1.2. 命名空间 4](#_Toc519669645)

[1.3. 文件 4](#_Toc519669646)

[1.4. 类 4](#_Toc519669647)

[1.5. 接口 5](#_Toc519669648)

[1.6. 变量 5](#_Toc519669649)

[1.7. 常量 5](#_Toc519669650)

[1.8. 属性 6](#_Toc519669651)

[1.9. 方法 6](#_Toc519669652)

[1.10. 委托、事件 7](#_Toc519669653)

[1.11. 枚举 7](#_Toc519669654)

[**2.** **注释规范** 7](#_Toc519669655)

[2.1. 类注释规范 7](#_Toc519669656)

[2.2. 属性注释规范 7](#_Toc519669657)

[2.3. 方法注释规范 8](#_Toc519669658)

[2.4. 代码间注释规范 8](#_Toc519669659)

[**3.** **代码规范** 9](#_Toc519669660)

[3.1. 代码格式 9](#_Toc519669661)

[3.2. 代码优化 9](#_Toc519669662)

[**4.** **其它原则** 11](#_Toc519669663)

1. **命名规范**
   1. 总则

* 【强制】无论是变量,类,还是其他对象的命名,都应该采用意义明确的命名,根据命名规范命名，以增加程序的可读性和可维护性。
* 【强制】名称，全部用拼音

正例：病人信息：BingRenXX;

正例：获取病人信息:GetBingRenXX()

* 1. 命名空间
* 【强制】所有命名空间起始部分为：公司名称.系统名称.模块名称

正例：Mediinfo.PAIS.BLL

* 【推荐】工程下创建的文件夹，默认会自动添加文件夹后缀的命名空间，在命名空间中需要删除文件夹后缀；[1.01]
  1. 文件
* 【推荐】文件名命名规范参考类命名规范，一个文件保存一个类，并保持类名与文件名一致；

正例：类BingRenXX对应的文件名应为 BingRenXX.cs

* 1. 类
* 【强制】名称应该能够标识类的特征；
* 【强制】名称前两个中文字全拼+后续中文拼音首字母;若拼音组合一样，则将第三个字改为全拼，若第三个字全拼仍无法区分，则第四个字段改为全拼，依次类推；
* 【强制】拼音首字母大写，其他小写；
* 【推荐】不要使用下划线字符（\_）
* 【推荐】抽象类使用Abstract或Base开头
* 【推荐】异常类用Exception 结尾
* 【推荐】测试类用Test结尾

正例：MenZhenSF（门诊收费类）

* 1. 接口
* 【强制】遵循类名规范，名称前加上“I”前缀

正例：IMenZhenSF（门诊收费接口）

* 1. 变量

使用场景：局部变量、全局变量、参数

* 【强制】名称应该能够标识变量的特征、用途；
* 【强制】名称前两个中文字全拼+后续中文拼音首字母;若拼音组合一样，则将第三个字改为全拼，若第三个字全拼仍无法区分，则第四个字改为全拼，依次类推。
* 【强制】字母全部小写；

正例：个人编号命名为gerenbh

反例：int a,随意命名方式

反例：int ABC,不允许大写

批次信息总数

正例：int picixxzs

反例：int picixxcount,不允许出现拼音+英文的方式

* 1. 常量
* 【强制】名称应该能够标识常量的特征、用途；
* 【强制】名称前两个中文字全拼+后续中文拼音首字母;若拼音组合一样，则将第三个字改为全拼，若第三个字全拼仍无法区分，则第四个字改为全拼，依次类推。
* 【强制】字母全部大写；
* 【强制】以CONST\_开头；
* 【推荐】尽量将全部常量定义在某一个工程中,不要在很多工程下定义常量；

正例：const int CONST\_XIANCHENGJGSJ=5 //线程间隔时间

反例：const int CONST\_xcjgsj=5

* 1. 属性
* 【强制】名称应该能够标识属性的特征；
* 【强制】名称前两个中文字全拼+后续中文拼音首字母;若拼音组合一样，则将第三个字改为全拼，若第三个字全拼仍无法区分，则第四个字改为全拼，依次类推；
* 【推荐】字母全部大写；
* 【推荐】不要使用下划线字符（\_）

正例：SHENFENZHM（身份证号码）

* 1. 方法
* 【强制】名称应该能够标识方法的特征、用途；
* 【强制】名称以常见动词开头+前两个中文字全拼+后续中文拼音首字母;若拼音组合一样，则将第三个字改为全拼，若第三个字全拼仍无法区分，则第四个字改为全拼，依次类推；
* 【强制】拼音首字母大写，其他小写；
* 【强制】引用型(ref)、输出(out)参数放在参数列表最后;
* 【推荐】不要使用下划线字符（\_）

常见动词表

|  |  |  |
| --- | --- | --- |
| 动词 | 含义 | 说明 |
| Save | 保存 |  |
| Get | 获取 |  |
| Set | 设置 |  |
| Delete | 删除 |  |
| Update | 更新 |  |
| Check | 检查 |  |
| Is | 是否 |  |
| Cancel | 取消 |  |
| Clear | 清除 |  |
| Trans | 传输 |  |
| Down | 下载 |  |
| Upload | 上传 |  |
| Export | 导出 |  |
| Print | 打印 |  |
| Import | 导入 |  |

正例 ： 获取病人信息GetBingRenXX()

反例： getbingrenxx

* 1. 委托、事件
* 【强制】遵循方法命名规范，并以Handler结尾
  1. 枚举
* 【强制】遵循类名规范，名称前加上“E”前缀

正例：EYaoPinLX 药品类型

1. **注释规范**
   1. 类注释规范

* 【强制】注释需要描述类的功能、依赖和如何使用
* 【强制】不允许使用//来注释，必须使用///

正例：

/// <summary>

/// 类的描述

/// 作者：

/// 创建时间：<格式：YYYY-MM-DD>

/// </summary>

* 1. 属性注释规范
* 【强制】注释必须描述属性的中文含义
* 【强制】值域代码必须在summary中注明相应值域及含义或引用字典标准

正例：

/// <summary>

/// 是否作废(0-正常 1-作废)

/// </summary>

public int SHIFOUZF {get;set;}

正例：

/// <summary>

/// 民族代码(引用GY\_ZD\_JCXMB中GONGNENGBH=’00000’的数据)

/// </summary>

public string MINGZUDM {get;set;}

* 1. 方法注释规范
* 【强制】注释必须描述方法的用途，使用场景；
* 【强制】必须注释各个参数中文含义；
* 【强制】必须描述返回值说明；

正例：

/// <summary>

/// 保存科室信息

/// </summary>

/// <param name="keshixx">科室信息</param>

/// <returns>true:保存成功;false:保存失败</returns>

public bool SaveKeShiXX(GY\_KESHI keshixx)

{

….

}

* 1. 代码间注释规范
* 【强制】单行注释必须以//方式
* 【强制】多行注释必须以/\*\*/

正例：

单行注释：//<单行注释>

多行注释：/\*多行注释1

多行注释2

多行注释3

\*/

* 【强制】全局变量注释，每个全局变量需要写注释
* 【强制】程序流程变化注释，switch, if, while 等条件判断地方必须写注释
* 【强制】关键业务逻辑必须写注释

1. **代码规范**
   1. 代码格式

* 【强制】代码书写完成后必须使用VS默认格式化进行美化；
* 【强制】代码行与行之间不允许连续出现两行及以上空行；

反例：

public string Main() {

string say = "hello";

int flag = 0;

if (flag == 0) {

LogUtils.Info("空行太多");

}

}

* 【推荐】类中的方法放置顺序，按照public,internal,protected,private这样的顺序从上往下放置。并且public中把增删改方法放在最前，查询放在之后;
  1. 代码优化
* 【强制】页面直接访问的方法必须使用try…catch…
* 【强制】事务控制的方法必须使用try…catch…
* 【强制】在操作非托管对象（如流操作）的时候尽量使用using()，这样不论在过程中是否发生异常，对象会在该代码段的最后自动释放占用资源，这样能防止手动漏写相关释放资源的代码，让程序自动回收处理;

正例：

using (StreamReader sr = new StreamReader(@"test.txt", Encoding.UTF8, false))

{

while (!sr.EndOfStream)

{

Console.WriteLine(sr.ReadLine());

}

}

* 【强制】尽可能的考虑到会出现异常的数据情景，多使用条件判断来处理异常，而不是直接try catch;
* 【强制】尽量用StringBuilder，减少字符串相加，提高性能；

反例：

strrecodeset += "'FYZJE':'" + FYZJE + "',";

strrecodeset += "'DXFYZJE':'" + DataHelper.GetBig(FYZJE + "") + "',";

strrecodeset += "'YXJE':'" + YXJE + "',";

strrecodeset += "'TFJE':'" + (-TFJE) + "',";

strrecodeset += " yljgdm:'330327065'";

正例：

StringBuilder strrecodeset = new StringBuilder();

strrecodeset.AppendFormat("'FYZJE':'{0}',", FYZJE);

strrecodeset.AppendFormat("'DXFYZJE':'{0}',", FYZJE);

strrecodeset.AppendFormat("'YXJE':'{0}',", YXJE);

strrecodeset.AppendFormat("'TFJE':'{0}',", (-TFJE));

strrecodeset.Append("yljgdm:'330327065'");

* 【推荐】一个方法尽量只做一件事，避免进行重复造轮子，出现一些重复的冗余的代码功能块;
* 【推荐】逻辑判断：一个方法里面不要嵌套太多的逻辑判断（if else或者switch case），嵌套达到三层的判断就可以考虑把其中的一部分独立成新方法调用，或者使用尽快返回的方式;
* 【推荐】尽量缩短变量的存活周期；尽量不使用全局变量；尽量不使用静态变量；
* 【推荐】变量跨度：变量声明定义开始到第一次使用该变量的代码行之间的行距尽可能短；
* 【推荐】占用内存较高的变量建议手工释放；
* 【推荐】声明方法的参数类型时，应尽量指定最弱的类型，最好是接口而不是基类。例如，如果要写一个方法来处理一组数据项，最好是用接口（比如IEnumerable<T>）来声明方法的参数，而不要用强数据类型（比如List<T>）或者更强的接口类型（比如ICollecion<T>或IList<T>）。原因是可传递的参数对象变多。当然，如果方法需要的是一个列表（而非仅仅是可枚举的对象），就应该将参数类型声明为IList<T>。但是，仍然要避免声明为List<T>。声明为IList<T>，调用者可传递实现了IList<T>的其他类型对象。同理，有基类的尽量使用基类。除非有特定原因。相反，返回的时候返回最强的类型;
* 【推荐】当一个方法的参数数量比较多的时候，如果可能，声明一个单独的类进行封装;
* 【推荐】使用as操作符而不是强制类型转换，因为相对于强制类型转换来说，as更加安全，也更加高效

正例：

User user = obj as User;

反例:

User user = (User) obj;

* 【推荐】对于一个方法超出了整个屏幕可以显示的范围，尽量去分割它;,小方法并不仅体现在代码的可读性上，还关系到程序运行时的效率
* 【强制】MVC模式中控制器层需要在异常处理中记录错误日志, 非MVC方式的是否需要在一般处理程序或AJAX请求中异常处理中记录错误日志；[1.01]
* 【推荐】方法调用深度不能大于3层,尽量保持最少的调用层次；[1.01]

反例：

public bool SaveShouFeiXX(ShouFeiXX shoufeixx)

{

…

SaveShouFei1(shoufeixx)

}

public bool SaveShouFei1(ShouFeiXX shoufeixx)

{

…

SaveShouFei2(shoufeixx)

}

public bool SaveShouFei2(ShouFeiXX shoufeixx)

{

…

SaveShouFei3(shoufeixx)

}

public bool SaveShouFei3(ShouFeiXX shoufeixx)

{

…

SaveShouFei4(shoufeixx)

}

public bool SaveShouFei4(ShouFeiXX shoufeixx)

{

…

}

正例：

public bool SaveShouFeiXX(ShouFeiXX shoufeixx)

{

…

SaveShouFei1(shoufeixx)

…

SaveShouFei2(shoufeixx)

…

SaveShouFei3(shoufeixx)

…

SaveShouFei4(shoufeixx)

…

}

public bool SaveShouFei1(ShouFeiXX shoufeixx)

{

…

}

public bool SaveShouFei2(ShouFeiXX shoufeixx)

{

…

}

public bool SaveShouFei3(ShouFeiXX shoufeixx)

{

…

}

public bool SaveShouFei4(ShouFeiXX shoufeixx)

{

…

}

1. **其它原则**

* 【推荐】在保证类型可以完成其工作的前提下，应该尽可能地给类型分配最小的可见性。也就是，仅仅暴露那些需要暴露的。尽量使用较低可见性的类来实现公有接口。可见性越低，能访问你功能的代码越少，以后可能出现的修改也就越少；《Effective C# Second Edition》中原则21