**《软件工程概论B》上机**

**C#编码规范**

**一 、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

**二、适用范围**

1. 本规范主要以C#为开发语言的规范；

2. 由于本规范是为撰写程序而设计，所以适用于一切有关程序撰写的工作事项。对于具体的每个项目，可能需要对之进行裁剪和补存。

3. 适用人员：.net方向开发；

4. 适用产品：以C#编写的程序。

**三、语言规范**

1、代码格式

1.1、要使所有的缩进为一个Tab，即4个空格，使用VS.NET的默认设置。

1.2、在代码中[垂直对齐](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=66636822&ss_c=ssc.citiao.link" \t "_blank)左括号和右括号。

if (x == 0)

{

Response.Write("用户编号必须输入！");

}

不允许以下情况：

if (x == 0) {

Response.Write("用户编号必须输入！");

}

或者：

if (x == 0){ Response.Write("用户编号必须输入！");}

1.3、为了防止在阅读代码时滚动[源代码编辑器](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=142690732&ss_c=ssc.citiao.link" \t "_blank)，每行代码或注释在[1024\*768](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=68525725&ss_c=ssc.citiao.link)的分辨率下尽量不超过一显示屏，如超过则应换行，换行后的代码应该缩进一个Tab。

1.4、当一行被分为几行时，将串联运算符放在每一行的末尾而不是开头，清楚地表示没有后面的行是不完整的。

1.5、每一行上放置的语句避免超过一条，避免写 int i = 0;int j = 1;

1.6、在大多数运算符[之前和之后](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=142255302&ss_c=ssc.citiao.link" \t "_blank)使用空格，这样做时不会改变代码的意图却可以使代码容易阅读。例：

int j = i + k;

而不应写为

    int j=i+k;

1.7、将大的复杂代码节分为较小的、易于理解的模块。

1.8、编写 SQL 语句时，建议（不强求）对于关键字使用全部大写，对于数据库元素（如表、列和视图）使用大小写混合。

1.9、将每个主要的 SQL 子句放在不同的行上，这样更容易阅读和编辑语句，例如：

2、注释规范

注释是为了增加代码的清晰度，保持注释的简洁，不是任何代码都需要注释的，过多的注释反而会影响代码的可读性。

注释不要包括其他的特殊字符，建议先写注释，后写代码，注释和代码一起完成。

如果语句块（比如循环和条件分枝的代码块）代码太长，嵌套太多，则在其结束“｝”要加上注释，标志对应的开始语句。如果分支条件逻辑比较复杂，也要加上注释。

注释规范主要包括：模块（类）注释规范、类属性注释规范、方法注释规范、代码间注释规范。

2.1、模块（类）注释规范

模块开始尽量以以下形式书写模块注释：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \* 文 件 名:

\* 功能描述：

\* [Copyright](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=300434&ss_c=ssc.citiao.link) (c) 2010-2012 \*\*\*\*\*\*\*\*\*公司技术开发部

\* 创 建 人:

\* 日    期:

\* 修 改 人:

\* 修改日期:

\* 修改描述:

\* 版    本:

 \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

注：这里的修改人，是指对整个类进行修改过，需要在这些添加修改记录，且记录累加。

当模块有重大修改或修改他人代码时，必须添加以下注释：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 修 改 人:

\* 日    期:

\* 修改描述：<对此修改的描述>

\* 版    本:

\* 修 改 人:

\* 日    期:

\* 修改描述：<对此修改的描述>

\* 版    本:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

修改时根据需要保留程序员第一次和最近一次修改的代码。具体办法是将原有代码注释。(保留第一次的代码可以反映此部分代码最初要实现的功能及其初始编程思想，保留最新的代码可以反映此代码最近变化要实现的最新功能)

2.2、类属性注释规范

在类的属性必须以以下格式编写属性注释：

/// <summary>

///属性说明

/// </summary>

2.3、方法注释规范

在类的方法声明前必须以以下格式编写注释：

/// </summary>

        ///函数的作用

/// <param name="<参数名称>"><参数说明></param>

/// <对方法[返回值](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=581547&ss_c=ssc.citiao.link" \t "_blank)的说明，该说明必须明确说明返回的值代表什么含义>

        ///<exception>异常类型、异常结果说明（为

#endregion

2.4、代码间注释规范

代码间注释分为[单行注释](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=63796783&ss_c=ssc.citiao.link)和多行注释。

  单行注释：

//<单行注释>（不提倡使用，尽量都使用/\*\*/）

  多行注释：

/\*多行注释1

 \*多行注释2

 \*多行注释3\*/

代码中遇到语句块（if，case……）或实现一个逻辑功能时，必须添加注释。添加的注释应该能够说明此语句块的作用和实现手段（所用算法等等）。

3、命名规范

3.1、类变量命名:类变量命名一律用下划线开头，多个单词组成，第一个字母小写的，其他单词第一个字母大写。

public class Hello

{

private string \_firstName;

private DateTime \_date;

}

3.2、类命名规范:

3.2.1、名字应该能够标识事物的特性；

3.2.2、首字母大写，缩略语可以全部字母大写；

3.2.3、名字可以有两个或三个单词组成，但通常不应多于三个；

3.2.4、在名字中，每个单词的第一个字母大写。例如：IsSuperUser；包含ID的，ID全部大写，如CustomerID；

3.2.5、使用名词或名词短语命名类；

3.2.6、少用缩写，除非是被广泛使用的；

3.2.7、不要使用下划线字符 (\_)。

3.3、局部变量命名：由小写字母开头，其他单词开头字母大写的各单词组成。

int  localVar;

3.4、函数命名：一律由大写字母开头的各个单词组成。

public void GetValue(int type)

{

//…

}

3.5、枚举类型命名:一律由大写字母”E”开头,第一个字母大写的各个单词组。

public enum Esample

{

EMPTY,

A,

B

}

并确保,枚举类型中有个约定的EMPTY元素。

3.6、接口命名规范: 和类命名规范相同，唯一区别是接口在名字前加上"I"前缀。

///

///接口样例

///

public interface ISample

{

///

///获取对象

///

object GetValue();

}

3.7、[抽象类](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=7766757&ss_c=ssc.citiao.link" \t "_blank)的命名:一律由大写字母”A”开头,第一个字母大写的各单词组成。

///

///抽象类样例

}

3.13、方法（Method）、属性（Atribute）命名规范

3.13.1、方法命名应采用首字母大写的方式；

3.13.2、返回值是Bool值的方法和属性以Is开头；

3.13.3、使用返回值是Bool值的方法（属性）的时候，不要将bool值方法（属性）与if语句放在同一行，如应该这样写：

bool valid = IsValid();

if (valid)

{

}

不应这样写：

if (IsValid())

{

}

3.14、类的实例命名规范：类的实例第一个字母小写，其他单词的第一个字母大写。

FileStream fileStream=new FileStream();

**4、异常处理规范**

异常变量命名：针对异常捕获过程中的[Exception](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=7625090&ss_c=ssc.citiao.link)变量命名，统一命名为ex1，[ex2](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=55598959&ss_c=ssc.citiao.link" \t "_blank)……

try

{

// code1

try

{

// code2

}

catch(Exception ex2)

{

//your code

}

}

catch(Exception ex1)

{

//your code

}

如果捕获异常不需要作任何处理，则不需要定义Exception实例。

try

{

//your code

}

catch( Exception )

{

}

不必每个方法都用try-catch。当特定的异常可能发生时才使用。比如，当你写文件时，处理异常FileIOException。

别写太大的 try-catch 模块。如果需要，为每个执行的任务编写单独的 try-catch 模块。 这将帮你找出哪一段代码产生异常，并给用户发出特定的错误消息。

如果应用程序需要，可以编写自己的异常类。自定义异常不应从基类SystemException派生，而要继承于IApplicationException。

**5、控件命名规则**：控件命名=控件缩写前缀+变量具体释义。

|  |  |  |
| --- | --- | --- |
| 控件名称 | 前缀 | 例子 |
| Button | btn | btnSubmit |
| CheckBox | chk | chkReadOnly |
| ComboBox | cbo | cboEnglish |
| Command | cmd | cmdDelete |
| [Connection](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=99672510&ss_c=ssc.citiao.link) | con | conOracle |
| [DataAdapter](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=59252236&ss_c=ssc.citiao.link) | dad | dadDiction |
| [DataColumn](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=69152893&ss_c=ssc.citiao.link) | dcol | dcolDiction |
| DataGrid | dgd | dgdTitles |
| [DataReader](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=9177376&ss_c=ssc.citiao.link) | dr | drDiction |
| [DataRelation](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=71493373&ss_c=ssc.citiao.link) | drel | drelDiction |
| DataRow | drow | drowDiction |
| DataSet | ds | dsDiction |
| [DataTable](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=7835293&ss_c=ssc.citiao.link) | dt | dtDiction |
| DataView | dv | dvDiction |
| DateTimePicker | dtp | dtpPublished |
| [DropDownList](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=8750484&ss_c=ssc.citiao.link) | drp | drpOffice |
| Form | frm | frmMain |
| ImageList | ils | ilsAllIcons |
| Label | lbl | lblHelpMessage |
| ListBox | lst | lstPolicyCodes |
| ListView | lv | lvHeadings |
| Menu | mnu | mnuFileOpen |
| MonthCalender | mcd | mcdPeriod |
| Parameter | parm | parmIn |
| PictureBox | pic | picVGA |
| [ProgressBar](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=57159109&ss_c=ssc.citiao.link) | prg | prgLoadFile |
| [RadioButton](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=55808598&ss_c=ssc.citiao.link) | rad | radGender |
| RichTextBox | rtb | rtbReport |
| StatusBar | stab | stabDateTime |
| TabControl | tab | tabOptions |
| TextBox | txt | txtLastName |
| ToolBar | tlb | tlbActions |
| TreeView | trv | trvOrganization |
| [ScriptManager](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=2362711&ss_c=ssc.citiao.link) | smg | smgContent |
| UpdatePanel | upd | updQuery |
| Panel | pl | plQuery |
| GridView | gv | gvShow |
| UpdateProgress | upp | uppUpdate |
| MultiView | mulv | mulvContent |
| View | vie | vieAdd |
| FormView | fvi | fviAdd |
| HiddenField | hf | hfMian |
| ObjectDataSource | ods | odsData |
| DynamicPopulateExtender | dpe | dpeText |
| FilteredTextBoxExtender | ftbe | ftbeNum |

**6、调用规范**

6.1、类内变量和函数：类内函数和变量的调用一定要在前面加上”this.”；

6.2、类常量和类[静态变量](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=786565&ss_c=ssc.citiao.link" \t "_blank)，以及[静态方法](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=10695808&ss_c=ssc.citiao.link)：类内静态元素(元素包括变量和方法)的调用，一定要在前面加上该类的名字；

6.3、系统类的使用：系统类型的使用，.Net中对于基本数值类型,都有相应的引用类型定义，在一般表示的情况下，我们使用小写字母开头的类型，而用到他们方法和数据的时候，则使用引用空间的名称。比如string 有 System.String对应，int 有System.Int32对应。

//注意string和String的使用地方

string strSample = String.Empty;

//注意int和Int32的使用地方

int intSample = 0;

System.Int32.Equals(intSample,3);

6.4、引用类型转换：我们不要使用类型加在前面的转换方式，而使用as，这样当转换不成功时，也不会有异常，但可以根据转换结果进行判断。

object obj;

不要：

ISample sam = (ISample)obj;

而要：

ISample sam = obj as ISample;

if(null == sam)

{ //转换失败

}

**7、DotNet编程提醒**

7.1、循环语句

对一个容器进行循环，如果该容器已经实现了IEnumerable或者IEnumerator接口，那么我们推荐使用foreach…in…语句，例如:ICollection coll中存放了一些ISample对象，那么我们的遍历：

 foreach(ISample samp in coll)

{

//对samp处理

……

}

对于经典的[for循环](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=654572&ss_c=ssc.citiao.link" \t "_blank)：

for(int i=0;i<COLL.COUNT();i++)

{

ISample samp = coll[i] as ISample;

}

我们应该这样写：避免循环的每次都调用coll.Count()方法，而不必要的损失性能。

for(int i=0,size=coll.Count();i< size;++i)

{

ISample samp = coll[i] as ISample;

}

7.2、在需要大量字符串连接的时候,要使用[StringBuilder](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=66238158&ss_c=ssc.citiao.link" \t "_blank)类，而不是使用string+string+string……的方法：

不要：

string strSample = “a”+”b”+”c”+”d”+”e”;

而要：

StringBuilder sb = new StringBuilder();

sb.append(“a”);

sb.append(“b”);

sb.append(“c”);

sb.append(“d”);

sb.append(“e”);

string strSample = sb.ToString();

在需要使用格式化字符串时，使用String.Format而不是使用字符串相加，例如

不要：

string strName = “Shawn”;

string strAge = “23”;

string strSample = “I am ”+strName+”,my age is ”+strAge;

而要：

string strSample =

String.Format(“I am {0},my age is {1}”,strName,strAge);