编码规范说明书

一、 模块头部注释规范

以一个物理文件为单元的都需要有模块头部注释规范，例如：C#中的.cs 文件。用于每个模块开头的说明，主要包括以下几部分：（**粗体字**为必需部分，其余为可选部分）

1. **文件名称(File Name)：** 此文件的名称
2. **功能描述(Description)：** 此模块的功能描述与大概流程说明
3. **数据表(Tables)：** 所用到的数据表，视图，存储过程的说明，如关系比较复杂，则应说明哪些是可擦写的，哪些表为只读的。
4. **作者(Author)：**创建人
5. **日期(Create Date)：**创建时间
6. 参考文档(Reference)(可选)： 该档所对应的分析文档，设计文檔。
7. 引用(Using) (可选) ： 开发的系统中引用其它系统的 Dll、对象时，要列出其对应的出处，是否与系统有关（ 不清楚的可以不写 ），以方便制作安装档。
8. **修改记录(Revision History)：**若档案的所有者改变，则需要有修改人员的名字、修改日期及修改理由。
9. **分割符：********** (前后都要)

示例如下：

```
// ******************************************************************************
//
// 文件名(File Name):           Employee.cs
//
// 数据表(Tables):              nothing
//
// 作者(Author):                武临风
//
// 日期(Create Date):           2012.1.20
//
// 修改记录(Revision History):
//      R1:
//          修改作者:           李雷
//          修改日期:           2012.1.24
//          修改理由:           增加全球化解决方案的公用方法。可以根据默认的登录语言或者特
//                              定的语言获取单个或多个资源信息；可以根据默认的登录语言或者
//                              特定的语言获取单个待替换参数的资源信息。
//      R2:
//          修改作者:           韩梅梅
//          修改日期:           2012.2.14
//          修改理由:           修改文件头部注释格式，增加修改记录部分的格式。
//
// ******************************************************************************

using System.Windows;
using System.Threading;
```

二、 方法注释规范

<1>、类、接口、属性、方法必须有<Summary>节，另外方法如果有参数及返回值，则必须有<Param>及<Returns>节。格式如下：

```
/// <summary>
```

/// …

/// </summary>

/// <param name=””></param>

/// <returns></returns>

示例如下：

```
/// <summary>
/// 使用MD5计算哈希散列
/// </summary>
/// <param name="strSource">需要加密字符串</param>
/// <param name="strSaltCode">盐值</param>
/// <returns>加密后的字符串</returns>
public static string EncryptMD5(string strSource,string strSaltCode)
{
    //md5加密服务对象
    System.Security.Cryptography.MD5CryptoServiceProvider  md5 =
        new System.Security.Cryptography.MD5CryptoServiceProvider();

    byte[] bytValue;
    byte[] bytHash;
```

<2>、公用类库中的公用方法需要在一般方法的注释后添加作者、日期及修改记录信息，统一采用 XML 标签的格式加注。

格式如下：

<Author></Author> 作者

<CreateDate></CreateDate> 建立日期

<RevisionHistory> 修改记录

<ModifyBy></ModifyBy> 修改作者

<ModifyDate></ModifyDate> 修改日期

<ModifyReason></ModifyReason> 修改理由

<ModifyBy></ModifyBy> 修改作者

<ModifyDate></ModifyDate> 修改日期

<ModifyReason></ModifyReason> 修改理由

<ModifyBy></ModifyBy> 修改作者

<ModifyDate></ModifyDate> 修改日期

<ModifyReason></ModifyReason> 修改理由

</RevisionHistory>

<LastModifyDate></LastModifyDate> 最后修改日期

三、 代码行注释规范

格式如下(双斜线)：

// ……

示例如下：

```
public void Dispose()
{
    // 如果事务开启过，则释放事务对象
    if ( this.IDbTrans != null)
        this.IDbTrans.Dispose();

    // 如果连接已经打开，则关闭连接并释放资源
    if ( this.IDbConn.State == ConnectionState.Open )
    {
        this.IDbConn.Close();
        this.IDbConn.Dispose();
    }
}
```

四、 变量注释规范

<1>、Class 级变量应以采用 /// 形式自动产生 XML 标签格式的注释。

示例如下：

```
/// <summary>
/// Database server name.
/// </summary>
private static string mstrServerName = "";
/// <summary>
/// The system database name of solomon.
/// </summary>
private static string mstrSysDBName = "";
/// <summary>
/// The login user name for the system database of solomon.
/// </summary>
private static string mstrSysUid = "";
```

<2>、方法级的变量注释可以放在变量声明语句的后面，与前后行变量声明的注释左对齐，注释与代码间以 Tab 隔开。

示例如下：

```
protected bool SendNotification(string strSubjectResID,NameValueCollection objl
    NameValueCollection objValues,string strLinkLabelResID,string strLink,
    string strSendCOFrom,string strSendCOTo,string strSendMailTo,
    bool blnIsSendMail,bool blnIsSendCO,MailPriority enuMailPriority)
{
    bool blnIsSuccess = false;          // 发送邮件是否成功

    string strRegard = "2835";          //2835(您好！)
    string strPrompt = "2836";          //2836(以上信息是由EPortal One自动发
    string strInfo = "2833";            //2833(有关信息如下)
    string strSubjectPrefix = "2853";   //2853(EPortal One Web Message)
```

五、 命名规则

<1>、Package 的命名

Package 的名字应该都是由一个小写单词组成，按照以下格式对包进行命名：

com.wiimedia.项目名称.大类名称.小类名称。

例如：com.wiimedia.mobile.common。

<2>、Class 的命名

Class 的名字必须由大写字母开头而其他字母都小写的单词组成，

例如：DataFile 或 InfoParser。

说明：对同一项目的不同命名空间中的类，命名避免重复。避免引用时的冲突和混淆；

<3>、Class 变量的命名

变量的名字必须用一个小写字母开头。后面的单词用大写字母开头，

例如：debug 或 inputFileSize。

说明：尽量要使用短而且具有意义的单词，如果变量是集合，则变量名要用复数。例如表格的行数，命名应为：RowsCount；

<4>、Static Final 变量的命名

Static Final 变量的名字应该都大写，并且指出完整含义，

例如：MAX_UPLOAD_FILE_SIZE=1024。

<5>、参数命名

参数的名字必须和类变量的命名规范一致。

<6>、数组命名

例如：byte[] buffer; 而不是： byte buffer[];

<7>、接口命名

接口的名字要以字母 I 开头。保证对接口的标准实现名字只相差一个“I”前缀，例如对 IComponent 接口的标准实现为 Component；

<8>、泛型类型参数的命名

泛型类型参数的命名：命名要为 T 或者以 T 开头的描述性名字，

例如：public class List<T>； public class MyClass<Tsession>

<9>、方法的参数

使用有意义的参数命名，如果可能的话，使用和要赋值的字段一样的名字：

例如：setCounter(int size) {

this.size = size;

}

<10>、组件名称缩写规则

缩写的基本原则是取组件类名各单词的第一个字母，如果只有一个单词，则去掉其中的元音，留下辅音。缩写全部为小写。如下：

组件类型	缩写	例子
Label	Lbl	lblNote
TextBox	Txt	txtName
Button	Btn	btnOK

ImageButton	Ib	ibOK
LinkButton	Lb	lbJump
HyperLink	Hl	hlJump
DropDownList	Ddl	ddlList
CheckBox	Cb	cbChoice
CheckBoxList	Cbl	cblGroup
RadioButton	Rb	rbChoice
RadioButtonList	Rbl	rblGroup
Image	Img	imgBeauty
Panel	Pnl	pnlTree
TreeView	Tv	tvUnit
WebComTable	Wct	wctBasic
ImageDateTimeInput	Dti	dtiStart
ComboBox	Cb	cbList
MyImageButton	Mib	mibOK
WebComm.TreeView	Tv	tvUnit
PageBar	Pb	pbMaster

六、 代码格式

<1>、文件头声明

源文件的头部需要功能说明描述。该段定义在 package 之上，例如：

```
/**********************************************************
 * copyright@2009 youngfar Co. Ltd.
 * All right reserved.
 * 功能描述
 **********************************************************/
```

<2>、Using 顺序

按以下顺序：

a)、.net 标准包

b)、.net 扩展包

c)、使用的外部库的包（例如 Microsoft.Practices.EnterpriseLibrary.Data）

d)、使用的项目的公共包

e)、使用的模块的其他包

每一类 Using 后面加一个换行。

<3>、类头声明

在类的头部需要功能说明，作者，创建时间，版本进行描述。该段定义在类声明之上，例如：

```
/**
 * 功能描述
 *
 * @author sometimes
 * @since 2009.04.01
 * @version 1.0.0
 * */
```

注：如若非本人修改需要标注，修改人，修改原因，修改时间，修改函数名称；若为本人修改只需按要求升级版本号即可。

例如：

```
/**
 * 功能描述
 *
 * @author sometimes
 * @since 2009.04.01
 * @version 1.0.1
 * @modify 1.0.1 by gongbin since 2009.04.01 method = test() 修改样例
* */
```

<4>、代码长度

a)、对于每一个函数建议尽可能控制其代码长度为 53 行左右，超过 53 行的代码要重新考虑将其拆分为两个或两个以上的函数。函数拆分规则应该一不破坏原有算法为基础，同时拆分出来的部分应该是可以重复利用的。对于在多个模块或者窗体中都要用到的重复性代码，完全可以将起独立成为一个具备公用性质的函数，放置于一个公用模块中。

b)、页宽

页宽应该设置为 80 字符。源代码一般不会超过这个宽度，并导致无法完整显示，但这一设置也可以灵活调整. 在任何情况下，超长的语句应该在一个逗号或者一个操作符后折行. 一条语句折行后，应该比原来的语句再缩进 2 个字符.

c)、行数

一般的集成编程环境下，每屏大概只能显示不超过 50 行的程序，所以这个函数大

概要 5-6 屏显示，在某些环境下要 8 屏左右才能显示完。这样一来，无论是读程序还是修改程序，都会有困难。因此建议把完成比较独立功能的程序块抽出，单独成为一个函数。把完成相同或相近功能的程序块抽出，独立为一个子函数。可以发现，越是上层的函数越简单，就是调用几个子函数，越是底层的函数完成的越是具体的工作。这是好程序的一个标志。这样，我们就可以在较上层函数里容易控制整个程序的逻辑，而在底层的函数里专注于某方面的功能的实现了。

<5>、函数命名

通常，函数的命名也是以能表达函数的动作意义为原则的，一般是由动词打头，然后跟上表示动作对象的名词，各单词的首字母应该大写。另外，还有一些函数命名的通用规则。如取数，则用 get 打头，然后跟上要取的对象的名字；设置数，则用 set 打头，然后跟上要设的对象的名字；而对象中为了响应消息进行动作的函数，可以命名为 on 打头，然后是相应的消息的名称；进行主动动作的函数，可以命名为 do 打头，然后是相应的动作名称。

<6>、函数注释

系统自动生成的函数，如鼠标动作响应函数等，不必太多的注释和解释；

对于自行编写的函数，若是系统关键函数，则必须在函数实现部分的上方标明该函数的信息，格式如下：

```
/**********************************************************
* 功能描述
* @param 参数类型 参数含义
* @return 返回值类型 返回值含义
* @ throws 异常类型
**********************************************************/
```

注：对于参数是容器的地方，1.4 版本以下的 JDK 因为没有泛型的支持，所以应该在注释中加上容器中存的是什么对象。

如： @param nameList List<String>

七、 其他规范

<1>、ASPX 页面编码规范。

a)、Request、session、application

web 应该有 request session application 的概念不要这三个不分，不要把所有的数据都存储在 session 中。而且 session 应该做一个 session 容器内，方便管理与监控,不要把 session 乱存。

b)、文件命名与存放位置

应该分功能和使用类型将对应文件分类存放，方便管理。例如：

```
WERROOT
Js
Css
images
```

```
jsp
+---modeA
    +---addUser.aspx
    +---addUser.css
    +---addUser.js
```

c)、开头注释

写出 aspx 页面所展示的大概功能

```
<%--
用户信息的显示页面
auther wang xxx
--%>
```

d)、ASPX 声明语句

如果非特殊要求程序的 ASPX 页面 ，html 统一用成 UTF-8 编码

```
<%@ page language="C#" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
```

<2>、JAVASCRIPT 编码规范

a)、单个文件长度不能超过 800 行（C#也是这样） // 目的：支持尽可能的模块化封装，强制精炼简化代码。利于维护

b)、不要调用平台封装以外的方法，经由平台统一提供方法调用.// 利于统一管理，功能修改，迁移。

c)、注释完备（功能，参数，返回值的说明）

EX:

```
/**
  *@description:  去除字符串前后的空格
  *@param  des    {string} 要清理的字符串
  *@return result  {string} 清理结果
  */
 function trim(des){
    var result=null;
    ... some code ...
    return result;
 }
```

d)、JS 的返回值类型约束较弱，要求在方法设计之初提供严格的对外调用返回.{Number ,boolean, String/null, Object/null 禁止 undefine 返回} //严格的类型可在后续的编程活动中精炼，清晰的定义流程。

e)、禁止全局变量，方法随意往 WINDOW 空间添加，按照业务，功能模块，维护自己的数据对象。//避免变量冲突，覆盖。利于快速释放空间。

EX:

```
var somevirable;  // 随意的添加到 WINDOW 空间，容易造成数据的相互覆盖，
```

紊乱。

```
var moduleDAO = new Object();  //构建模块数据承载对象
moduleDAO.somevirable=someValue;  // 在本模块数据对象上进行数据操作。
var moduleDAO = null;  // 在所有数据无用之后方便，迅速的释放空间。
```

f)、采用见名知意的命名方式，业务方法采用 “业务模块+功能命名” 的层次方式来命名，风格统一，可采用 JAVA 对变量，常量，方法的命名约定。

EX:

```
nbrHallSelectFreeNumber();  //选号厅空闲号码选择
```

g)、逻辑块 之间使用空行隔开。

EX:

```
{
    (SUBMODULE1)
    LINE1;
    LINE2;
    (SUBMODULE2)
    LINE3;
    LINE4;
    LINE5;
    (OTHRE SUBMODULES)
    LINE6;
    ...
}
```

八、 编码风格

<1>、变量声明

为了保持更好的阅读习惯,请不要把多个变量声明写在一行中,即一行只声明一个变量。

例如：

```
String strTest1, strTest2;
```

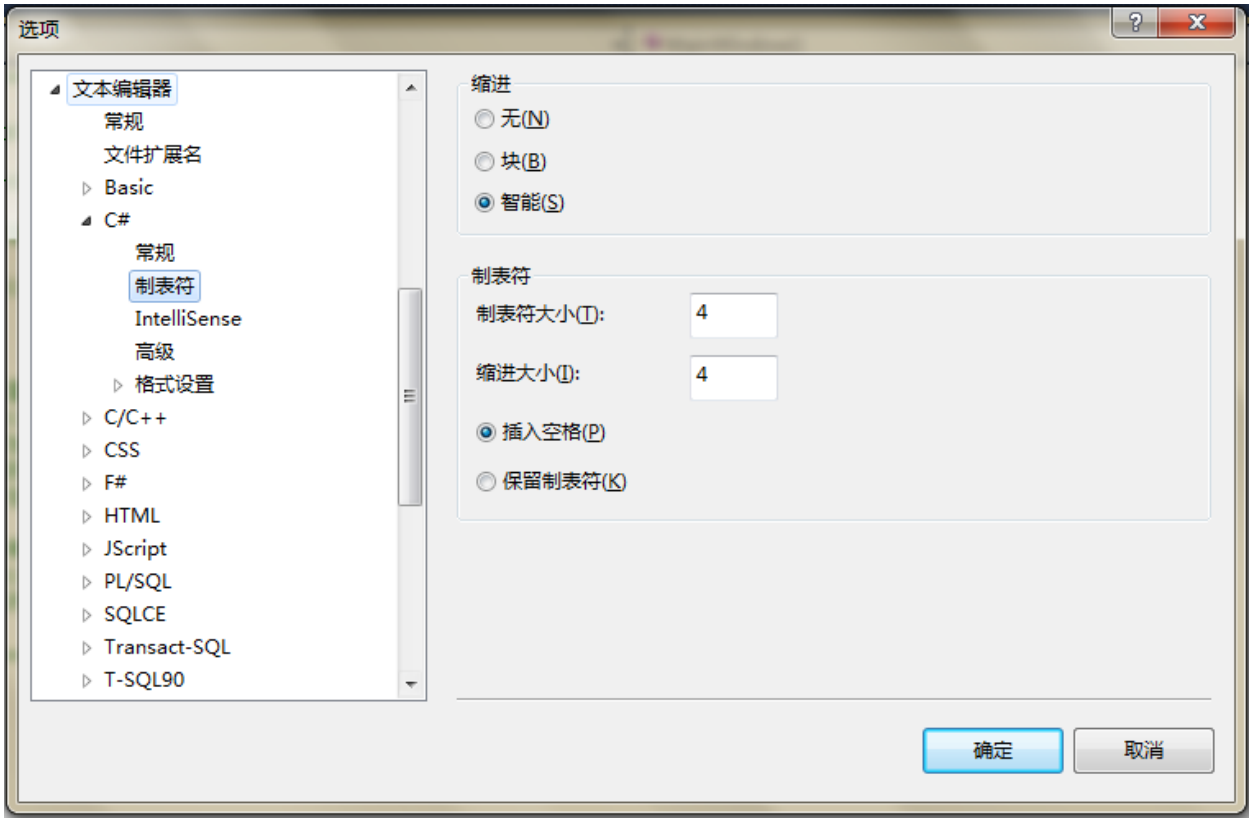
应写成：

```
String strTest1;
String strTest2;
```

<2>、代码缩进

一致的代码缩进风格，有利于代码的结构层次的表达，使代码更容易阅读和传阅；

代码缩进使用 Tab 键实现，最好不要使用空格，为保证在不同机器上使代码缩进保持一致，特此规定 C#的 Tab 键宽度为 4 个字符，设定界面如下(工具 - 选项)：



避免方法中有超过 5 个参数的情况，一般以 2,3 个为宜。如果超过了，则应使用 struct 来传递多个参数。

为了更容易阅读，代码行请不要太长，最好的宽度是屏幕宽度（根据不同的显示分辨率其可见宽度也不同）。请不要超过您正在使用的屏幕宽度。（每行代码不要超过 80 个字符。）程序中不应使用 goto 语句。在 switch 语句中总是要 default 子句来显示信息。方法参数多于 8 个时采用结构体或类方式传递，操作符/运算符左右空一个半角空格所有块的{}号分别放置一行，并嵌套对齐，不要放在同一行上

<3>、空白

空行将逻辑相关的代码段分隔开，以提高可读性。

下列情况应该总是使用两个空行：

a)、一个源文件的两个片段(section)之间。

b)、类声明和接口声明之间。

下列情况应该总是使用一个空行：

a)、两个方法之间。

b)、方法内的局部变量和方法的第一条语句之间。

c)、块注释（参见"5.1.1"）或单行注释（参见"5.1.2"）之前。

d)、一个方法内的两个逻辑段之间，用以提高可读性。

下列情况应该总是使用空格：

a)、空白应该位于参数列表中逗号的后面，如：

```
void UpdateData(int a, int b)
```

b)、所有的二元运算符，除了"."，应该使用空格将之与操作数分开。一元操

作符和操作数之间不因该加空格，比如：负号("-")、自增("++")和自减("--")。例　如：

```
a += c + d;
d++;
```

c)、for 语句中的表达式应该被空格分开，例如：

```
for (expr1; expr2; expr3)
```

d)、强制转型后应该跟一个空格，例如：

```
char c;
int a = 1;
c = (char) a;
```

```
public void Dispose()
{
    // 如果事务开启过，则释放事务对象
    if (this.IDbTrans != null)
        this.IDbTrans.Dispose();

    // 如果连接已经打开，则关闭连接并释放资源
    if (this.IDbConn.State == ConnectionState.Open)
    {
        this.IDbConn.Close();
        this.IDbConn.Dispose();
    }
}
```

九、 资源释放

所有外部资源都必须显式释放。例如：数据库连接对象、IO 对象等。如下

```
public void Dispose()
{
    // 如果事务开启过，则释放事务对象
    if (this.IDbTrans != null)
        this.IDbTrans.Dispose();

    // 如果连接已经打开，则关闭连接并释放资源
    if (this.IDbConn.State == ConnectionState.Open)
    {
        this.IDbConn.Close();
        this.IDbConn.Dispose();
    }
}
```

十、 错误处理

<1>、不要“捕捉了异常却什么也不做“。如果隐藏了一个异常，你将永远不知道异常到底发生了没有。

<2>、发生异常时，给出友好的消息给用户，但要精确记录错误的所有可能细节，包括发生的时间，和相关方法，类名等。

<3>、只捕捉特定的异常，而不是一般的异常。

正确做法：

```
void ReadFromFile(string fileName)
{
    try
    {
        // read from file.
    }
    catch (FileIOException ex)
    {
        // log error.
        //  re-throw exception depending on your case.
        throw;
    }
}
```

错误做法：

```
void ReadFromFile(string fileName)
{
    try
    {
        // read from file.
    }
    catch (Exception ex)
    {
        // Catching general exception is bad... we will never know whether it
        // was a file error or some other error.

        // Here you are hiding an exception.
        // In this case no one will ever know that an exception happened.
    }
}
```

十一、 其他约定

<1>、一个方法只完成一个任务。不要把多个任务组合到一个方法中，即使那些任务非常小。

<2>、使用 C#的特有类型，而不是 System 命名空间中定义的别名类型。

<3>、别在程序中使用固定数值，用常量代替。

<4>、避免使用很多成员变量。声明局部变量，并传递给方法。不要在方法间共享成员变量。如果在几个方法间共享一个成员变量，那就很难知道是哪个方法在什么 时候修改了它的值。

<5>、别把成员变量声明为 public 或 protected。都声明为 private 而使用 public/protected 的属性

<6> 、不在代码中使用具体的路径和驱动器名。 使用相对路径，并使路径可编程。

<7> 、应用程序启动时作些“自检”并确保所需文件和附件在指定的位置。必要时检查数据库连接。出现任何问题给用户一个友好的提示。

<8>、如果需要的配置文件找不到，应用程序需能自己创建使用默认值的一份。

<9>、如果在配置文件中发现错误值，应用程序要抛出错误，给出提示消息告诉用户正确值。

<10>、DataColumn 取其列时要用字段名，不要用索引号。

例： 正确 DataColumn[“Name”]

不好 DataColumn[0]

<11>、在一个类中，字段定义全部统一放在 class 的头部、所有方法或属性的前面。

<12>、在一个类中，所有的属性全部定义在一个属性块中：

```
#region Properties

// Some Properties...

#endregion
```

十二、 开发技巧

<1>、StringBuilder

由于 String 类型是不可变字符，对于需要字符拼装时建议使用 StringBuilder 类型，使用 append 方法添加字符，可以提高程序执行效率。例如 sql 的拼装。

<2>、Iterator

如果对于List 类型变量需要进行遍历操作,建议先将List 类型转换成 Iterator 类型，再进行遍历可以很大的提高效率。

<3>、**异常处理**

尽量不要在模块层对异常进行捕获，尽量采用 throw 将异常抛出，并且尽量使用 java 本身的异常，在异常抛出的时候，杜绝直接使用 Exception，尽量将所有产生的异常类型都抛出来。在控制层对异常进行捕获和统一的处理调度，可以根据不同的异常进行不同的处理。

在必要的时候可以定义自己使用的异常，方便异常处理和异常调度。

不要将异常抛给界面，在逻辑控制层需要控制住所有的异常，包括系统异常，并且对所有的异常情况都要进行处理。

<4>、**常量**

对于系统中使用的常量，建议统一建立常量类，将所有的常量进行定义，并统一调用，不建议在代码中直接进行定义，避免重复和冲突。

<5>、**编程模式**

尽量使用目前成熟的编程模式，如：工厂模型，单体模型等。

<6>、**复用**

当某功能已经存在比较成熟的方法或第三方工具包，尽量采用，以便提高工作效率和稳定性。