1 目的

一.为了统一公司软件开发设计过程的编程规范

二.使网站开发人员能很方便的理解每个目录,变量，控件，类，方法的意义

三.为了保证编写出的程序都符合相同的规范，保证一致性、统一性而建立的程序编码规范。

四.编码规范和约定必须能明显改善代码可读性，并有助于代码管理、分类范围适用于企业所有基于.NET平台的软件开发工作

2 范围

本规范适用于开发组全体人员，作用于软件项目开发的代码编写阶段和后期维护阶段。

3 注释规范

3.1 概述

a) 注释要求英文及英文的标点符号。

b) 注释中，应标明对象的完整的名称及其用途，但应避免对代码过于详细的描述。

c) 每行注释的最大长度为100个字符。

d) 将注释与注释分隔符用一个空格分开。

e) 不允许给注释加外框。

f) 编码的同时书写注释。

g) 重要变量必须有注释。

h) 变量注释和变量在同一行，所有注释必须对齐，与变量分开至少四个“空格”键。

如：int m\_iLevel,m\_iCount; // m\_iLevel ....tree level

// m\_iCount ....count of tree items

string m\_strSql; //SQL

i) 典型算法必须有注释。

j) 在循环和逻辑分支地方的上行必须就近书写注释。

k) 程序段或语句的注释在程序段或语句的上一行

l) 在代码交付之前，必须删掉临时的或无关的注释。

m) 为便于阅读代码，每行代码的长度应少于100个字符。

3.2 自建代码文件注释

对于自己创建的代码文件（如函数、脚本），在文件开头，一般编写如下注释：

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

FileName:

Copyright (c) 2004-xxxx \*\*\*\*\*\*\*\*\*公司技术开发部

Writer:

create Date:

Rewriter:

Rewrite Date:

Impact:

Main Content（Function Name、parameters、returns）

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

3.3 模块（类）注释

模块开始必须以以下形式书写模块注释：

///<summary>

///Module ID：<模块编号，可以引用系统设计中的模块编号>

///Depiction：<对此类的描述，可以引用系统设计中的描述>

///Author：作者中文名

///Create Date：<模块创建日期，格式：YYYY-MM-DD>

///</summary>

如果模块只进行部分少量代码的修改时，则每次修改须添加以下注释：

///Rewriter Rewrite Date：<修改日期:格式YYYY-MM-DD> Start1：

/\* 原代码内容\*/

///End1：

将原代码内容注释掉，然后添加新代码使用以下注释：

///Added by Add date：<添加日期，格式：YYYY-MM-DD> Start2：

///End2：

如果模块输入输出参数或功能结构有较大修改，则每次修改必须添加以下注释：

///<summary>

///Log ID：<Log编号,从1开始一次增加>

///depiction：<对此修改的描述>

///Writer：修改者中文名

///Rewrite Date：<模块修改日期，格式：YYYY-MM-DD>

///</summary>

3.4 类属性注释

在类的属性必须以以下格式编写属性注释：

/// <summary>

/// <Properties depiction>

/// </summary>

3.5 方法注释

在类的方法声明前必须以以下格式编写注释

/// <summary>

/// depiction：<对该方法的说明>

/// </summary>

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

/// <returns>

///<对方法返回值的说明，该说明必须明确说明返回的值代表什么含义>

/// </returns>

///Writer：作者中文名

///Create Date：<方法创建日期，格式：YYYY-MM-DD>

3.6 代码间注释

代码间注释分为单行注释和多行注释：

//<单行注释>

/\*多行注释1

多行注释2

多行注释3\*/

代码中遇到语句块时必须添加注释（if,for,foreach,……）,添加的注释必须能够说明此语句块的作用和实现手段（所用算法等等）。

4 命名总体规则

 名字应该能够标识事物的特性。

 名字一律使用英文单词，而不能为拼音。

 名字尽量不使用缩写，除非它是众所周知的。

 名字可以有两个或三个单词组成，但不应多于三个，控制在3至30个字母以内。

 在名字中，多个单词用大写第一个字母（其它字母小写）来分隔。例如：IsSuperUser。

 名字尽量使用前缀而不是后缀。

 名字中的单词尽量使用名词，如有动词，也尽量放在后面。例如：FunctionUserDelete（而不是FunctionDeleteUser）。

5 命名规范

5.1 变量（Variable）命名

a) 程序文件(\*.cs)中的变量命名

程序中变量名称 = 变量的前缀 +代表变量含意的英文单词或单词缩写。

 类模块级的变量请用“m\_” +数据类型缩写作为前缀（其中，m 为“memory”缩写，数据类型缩写见附件中的《数据类型缩写表》）。

public class hello

{

private string m\_strName;

private DateTime m\_dtDate;

}

 类的属性所对应的变量，采用属性名前加“m\_”+ 类型缩写 前缀的形式

public class hello

{

private string m\_strName;

public string Name

{

get

{

return m\_strName;

}

}

}

过程级的变量使用类型缩写前缀

public class hello

{

void say()

{

string strSayWord;

}

}

 过程的参数使用“p\_”+ 类型缩写作为前缀（其中，p 为“parameter”缩写）

public class hello

{

void say(string p\_strSayWord)

{

}

}

补充说明：

针对异常捕获过程中的Exception变量命名，在没有冲突的情况下，统一命名为exp；

如果有冲突的情况下，可以用“exp”+ 标志名称，如：expSql。

Try

{

//your code

try

{

//code

}

catch(Exception exp)

{

//your code

}

}

catch(Exception expSql)

{

//your code

}

补充：如果捕获异常不需要作任何处理，则不需要定义Exception实例。

例：

try

{

//your code

}

catch( Exception exp)

{

}

 鉴于大多数名称都是通过连接若干单词构造的，请使用大小写混合的格式以简化它们的阅读。每个单词的第一个字母都是大写.

 即使对于可能仅出现在几个代码行中的生存期很短的变量，仍然使用有意义的名称。仅对于短循环索引使用单字母变量名，如 i 或 j。

 在变量名中使用互补对，如 min/max、begin/end 和 open/close。

 不要使用原义数字或原义字符串，如 For (i = 1;i <= 7;i++)。而是使用命名常数，如 For (i = 1;i <= NUM\_DAYS\_IN\_WEEK;i++) 以便于维护和理解。

b) 控件命名

控件命名 = 控件名称前二到三个字母 + 名称，如Labl控件（labUserName）

5.2 常量命名

常量名也应当有一定的意义，格式为 NOUN 或 NOUN\_VERB。常量名均为大写，字之间用下划线分隔。

例：

private const bool WEB\_ENABLEPAGECACHE\_DEFAULT = true;

private const int WEB\_PAGECACHEEXPIRESINSECONDS\_DEFAULT = 3600;

private const bool WEB\_ENABLESSL\_DEFAULT = false;

注：

变量名和常量名最多可以包含 255 个字符，但是，超过 25 到 30 个字符的名称比较笨拙。此外，要想取一个有实际意义的名称，清楚地表达变量或常量的用途，25 或 30 个字符应当足够了。

5.3 类（Class）命名

a) 名字应该能够标识事物的特性。

b) 名字尽量不使用缩写，除非它是众所周知的。

c) 名字可以有两个或三个单词组成，但通常不应多于三个。

d) 在名字中，所有单词第一个字母大写。例如 IsSuperUser，包含ID的，ID全部大写，如CustomerID。

e) 使用名词或名词短语命名类。

f) 少用缩写。

g) 不要使用下划线字符 (\_)。

例： public class FileStream

public class Button

public class String

5.4 接口（Interface）命名

和类命名规范相同，唯一区别是 接口在名字前加上“I”前缀

例：

interface IDBCommand;

interface IButton;

5.5 方法（Method）命名

和类命名规范相同。

5.6 命名空间（NameSpace）命名

和类命名规范相同。

6 编码规则

6.1 错误检查规则

a) 编程中要考虑函数的各种执行情况，尽可能处理所有流程情况。

b) 检查所有的系统调用的错误信息，除非要忽略错误。

c) 将函数分两类：一类为与屏幕的显示无关， 另一类与屏幕的显示有关。对于与屏幕显示无关的函数，函数通过返回值来报告错误。对于与屏幕显示有关的函数，函数要负责向用户发出警告，并进行错误处理。

d) 错误处理代码一般放在函数末尾。

e) 对于通用的错误处理，可建立通用的错误处理函数，处理常见的通用的错误。

6.2 大括号规则

将大括号放置在关键词下方的同列处，例如：

if ($condition) while ($condition)

{ {

... ...

} }

6.3 缩进规则

使用一个“Tab”为每层次缩进。例如：

function func()

{

if (something bad)

{

if (another thing bad)

{

while (more input)

{

}

}

}

}

6.4 小括号规则

a) 不要把小括号和关键词（if 、while等）紧贴在一起，要用空格隔开它们。

b) 不要把小括号和函数名紧贴在一起。

c) 除非必要，不要在Return返回语句中使用小括号。因为关键字不是函数，如果小括号紧贴着函数名和关键字，二者很容易被看成是一体的。

6.5 If Then Else规则

如果你有用到else if 语句的话，通常最好有一个else块以用于处理未处理到的其他情况。可以的话放一个记录信息注释在else处，即使在else没有任何的动作。其格式为：

if (条件1) // 注释

{

}

else if (条件2) // 注释

{

}

else // 注释

{

}

注：if 和循环的嵌套最多允许4层

6.6 比较规则

总是将恒量放在等号/不等号的左边。一个原因是假如你在等式中漏了一个等号，语法检查器会为你报错。第二个原因是你能立刻找到数值而不是在你的表达式的末端找到它。例如：

if ( 6 == $errorNum ) ...

6.7 Case规则

default case总应该存在，如果不允许到达，则应该保证：若到达了就会触发一个错误。Case的选择条件最好使用int或string类型。

6.8 对齐规则

变量的申明和初始化都应对齐。例如：

int m\_iCount;

int i,j;

float m\_fIncome,m\_fPay;

m\_iCount = 0;

i = 1;

m\_fIncome = 0.3;

6.9 单语句规则

除非这些语句有很密切的联系，否则每行只写一个语句。

6.10 单一功能规则

原则上，一个程序单元（函数、例程、方法）只完成一项功能。

6.11 简单功能规则

原则上，一个程序单元的代码应该限制在一页内（25~30行）。

6.12 明确条件规则

不要采用缺省值测试非零值。例如：使用“if ( 0 != f( ) )”而不用“if ( f( ) )”。

6.13 选用FALSE规则

大部分函数在错误时返回FALSE、0或NO之类的值，但在正确时返回值就不定了（不能用一个固定的TRUE、1或YES来代表），因此检测一个布尔值时应该用 FALSE、0、NO之类的不等式来代替。例如：使用“if ( FALSE != f( ) )” 而不用“if (TRUE == f( ) )”。

6.14 独立赋值规则

嵌入式赋值不利于理解程序，同时可能回造成意想不到的副作用，应尽量编写独立的赋值语句。例如：使用“a = b + c ; e = a + d;”而不用“e = ( a = b + c ) + d ”。

6.15 定义常量规则

对于代码中引用的常量（尤其是数字），应该define成一个大写的名字，在代码中引用名字而不直接引用值。

6.16 模块化规则

某一功能，如果重复实现一遍以上，即应考虑模块化，将它写成通用函数。并向小组成员发布。同时要尽可能利用其它人的现成模块。

6.17 交流规则

共享别人的工作成果，向别人提供自己的工作成果。

在具体任务开发中，如果有其它的编码规则，则在相应的软件开发计划中予 以明确定义。

7 编程准则

7.1 变量使用

a) 不允许随意定义全局变量。

b) 一个变量只能有一个用途；变量的用途必须和变量的名称保持一致。

c) 所有变量都必须在类和函数最前面定义，并分类排列。

7.2 数据库操作

a) 查找数据库表或视图时，只能取出确实需要的那些字段。

b) 使用无关联子查询，而不要使用关联子查询。

c) 清楚明白地使用列名，而不能使用列的序号。

d) 用事务保证数据的完整性。

7.3 对象使用

a) 尽可能晚地创建对象，并且尽可能早地释放它。

7.4 模块设计原则

a) 不允许随意定义公用的函数和类。

b) 函数功能单一，不允许一个函数实现两个及两个以上的功能。

c) 不能在函数内部使用全局变量，如要使用全局变量，应转化为局部变量。

d) 函数与函数之间只允许存在包含关系，而不允许存在交叉关系。即两者之间只存在单方向的调用与被调用，不存在双向的调用与被调用。

7.5 结构化要求

a) 禁止出现两条等价的支路。

例如：if (a == 2)

else if (a== 3)

//

else if (a == 2)

//

else

//

b) 避免使用GOTO语句

c) 用 IF 语句来强调只执行两组语句中的一组。禁止 ELSE GOTO 和 ELSE RETURN。

d) 用 CASE 实现多路分支

e) 避免从循环引出多个出口。

f) 函数只有一个出口。

g) 不使用条件赋值语句。

h) 避免不必要的分支。

i) 不要轻易用条件分支去替换逻辑表达式

7.6 函数返回值原则

1） 函数返回值

避免使用结构体等复杂类型

使用bool类型：该函数只需要获得成功或者失败的返回信息时候

使用int 类型：错误代码用负数表示，成功返回0

8 代码包规范

本项目中，每个任务在完成一个稳定的版本后，都应打包并且归档。

8.1 代码包的版本号

项目中，代码包的版本号由圆点隔开的两个数字组成，第一个数字表示发行号，第二个数字表示该版的修改号。具体用法如下：

1) 当代码包初版时，版本号为 V1.00；

2) 当代码包被局部修改或bug修正时，发行号不变，修改号第二个数字增1。例如，对初版代码包作了第一次修订，则版本号为 V1.01；

3) 当代码包在原有的基础上增加部分功能，发行号不变，修改号第一个数字增1，例如，对V1.12版的基础上增加部分功能，则新版本号为 V1.20；

4) 当代码包有重要修改或局部修订累积较多导致代码包发生全局变化时，发行号增1。例如，在 V1.15 版的基础上作了一次全面修改，则新版本号为 V2.00。

8.2 代码包的标识

本项目所产生的代码包都有唯一、特定的编码，其构成如下：

S-项目标识-代码包类型-版本号/序号

其中：

1) S：本项目的标识，表明本项目是“XXXX”。

2) 项目标识：简要标识本项目，此标识适用于整个项目的文档。

3) 代码包类型：取自以下表的两位字母编码。

4) 版本号：本代码包的版本号。

5) 序号：四位数字编码，指明该代码包在项目代码库的总序号。

例如：

一个Windows下RAR源码的压缩代码包命名为： S-XXXX-WS-V1.02/0001

项目的代码包分类表

类 型 编 码 注 释

RAR包

（web） 源码文件 WS 源代码文件包

编译文件 WB 编译文件包

安装文件 WI 安装文件包

源码代码 + 安装文件 WA 源代码和安装文件包

项目中所有代码包的标识清单将在《项目开发计划》中予以具体定义。

9 代码的控制

9.1 代码库/目录的建立

项目负责人在VSS中建立项目的文档库目录，即为“Software”目录，以便快速查询。

9.2 代码归档

所有代码在完成一个稳定的版本后，项目负责人都应打包后，存放于VSS中该目的“Software”目录下，并且依据代码包的命名规范为代码包分配一个唯一名称。

10 输入控制校验规则

10.1 登陆控制

用户登陆ID和登陆密码，要限定输入长度范围，必须检查输入合法性。

10.2 数据录入控制

a) TextBox输入

1、 要保持用户输入和数据库接收的长度一致

2、 必须进行输入合法性校验

如：E\_mail格式 XXX@XXX.XXX...

电话格式 020-12345678 (020)12345678

邮政编码是六位

b) 除CheckBox、RadioButton外，禁止在DataGrid内嵌入其他编辑控件，用以添加编辑数据

11. 数据库命名规范:

SqlServer命名规范

使用本系统遵循以下命名规范：

1、表命名：用一个或三个以下英文单词组成，单词首字母大写，如：DepartmentUsers；

2、表主键名称为：表名+ID，如Document表的主键名为：DocumentID

3、存储过程命名：表名+方法,如：p\_my\_NewsAdd,p\_my\_NewsUpdate;

4、视图命名：View\_表名，如：ViewNews;

5、Status为表中状态的列名，默认值为0，在表中删除操作将会改变Status的值而不真实删除该记录；

6、Checkintime为记录添加时间列，默认值为系统时间；

7、表、存储过程、视图等对象的所有都为dbo，不要使用数据库用户名，这样会影响数据库用户的更改。

附件1：数据类型缩写表

数据类型 缩写

string str

int i

char chr

sbyte sb

byte bt

uint ui

long l

ulong ul

float f

double d

bool b

decimal dec

附注：数据库服务器命名规范，控件名缩写+控件作用单词的全称,如：btnNext, txtPassword(密码文本框)，txtRPassword(密码确认文本框)，单词的第一个字母必须大写如果有多个单词，则为控件缩写+控件作用的第一个单词+第二个单词的全称，每个单词名的第一个字母必须大写

附件2：服务器控件名缩写表

a、web控件

web控件名 缩写

AdRotator art

Button btn

Calendar cd

CheckBox chk

CheckBoxList chkl

CompareValidator cpv

CustomValidator ctv

DataGrid dg

DataList dl

DropDownList ddl

HyperLink hl

Image img

ImageButton Ibtn

Label lab

LinkButton lbtn

ListBox lst

Panel pl

PlaceHolder ph

RadioButton rb

RadioButtonList rbl

RangeValidator rv

RegularExpressionValidator rev

Repeater rp

RequiredFieldValidator rfv

Table tb

TableCell tc

TableRow tr

TextBox txt

ValidationSummary vs

XML XML

b、html控件

html控件名 缩写

HtmlAnchor hah

HtmlButton hbtn

HtmlForm hform

HtmlGenericControl hgc

HtmlImage himg

HtmlInputButton（按钮） htxt

HtmlInputButton（重置） hrbtn

HtmlInputButton（提交） hcbtn

HtmlInputCheckBox hick

HtmlInputFile hifile

HtmlInputHidden hihidden

HtmlInputImage hiimg

HtmlInputRadioButton hirb

HtmlInputText（密码） hpwd

HtmlInputText（文本） hitxt

HtmlSelect hslt

HtmlTable htab

HtmlTableCell htc

HtmlTableRow htr

HtmlTextArea htxta

c.ADO.NET控件命名规范

类型 前缀 示例

Connection con conNorthwind

Command cmd cmdReturnProducts

Parameter parm parmProductID

DataAdapter dad dadProducts

DataReader dtr dtrProducts

DataSet dst dstNorthWind

DataTable dtbl dtblProduct

DataRow drow drowRow98

DataColumn dcol dcolProductID

DataRelation drel drelMasterDetail

DataView dvw dvwFilteredProducts