ASP.NET 编码规范

1 编码规范概述

ASP.NET 编码分为两大部分，一部分为客户端的静态文件编码，另一部分为包含服务器端脚本的动态文件编码。静态文件编码分 Script 编码和 HTML 编码两部分。服务器端编码则分为服务器脚本、客户端脚本、HTML 脚本三部分。

编码规范采用如下约定：

1. 所有客户端脚本一律使用 JavaScript
2. 所有服务器端脚本一律使用 C#
3. 静态页面输出一律使用 HTML 脚本
4. 本规范不适用于由服务器端脚本所产生的客户端脚本代码。

两种常用的命名法：

Pascal 命名法格式 - 所有单词第一个字母大写，其他字母小写。

Camel 命名法格式 - 除了第一个单词，所有单词第一个字母大写，其他字母小写。

2 静态文件编码规范

输出部分采用 HTML 标记语言。静态文件脚本部分采用 JavaScript 编写。

2.1 HTML 标记语言编码规范

2.1.1 标记的换行规范：

* 一个标记必须占用一行。不得出现两个标记在同一行的情况(同一标记的关闭标记除外)，如：

<tr><td>text</td></tr>

而必须写成：

<tr>

<td>text</td>

</tr>

2.1.2 标记的关闭规范

* 静态文件内容必须包含在<body></body>标记中间

* <body>标记必须包含在<html></html>标记中间

* 对于需要关闭的标记，如：

<html><title><body><table><tr><td> <textarea><select><opti

on>

必须同其关闭标记同时出现。如

<body>… …….…. …..</body>

* 不得出现交叉包含的语句，如：

…..

2.1.3 标记的属性赋值规范

对于接受属性的标记，属性值必须使用双引号或者单引号包围。如:

<body bgcolor=”red”>

<font size=’7’>

注意：必须确保属性的赋值无警告或错误。

2.1.4 标记的缩进规范

* 最高一级的父标记采用左对齐顶格方式书写。

* 下一级标记采用左对齐向右缩进一个 Tab 的方式书写

在下一级依此类推，分别左对齐相对于父标记向右缩进一个 Tab 的方式书写

* 同一级标记的首字符上下必须对齐。

注意：在进行代码缩进时，可使用 VS2005 及其以上版本的开发环境中的自动缩进功能。

2.2 客户端 JavaScript 规范

2.2.1 变量命名规范

* 常量以及全局变量名必须全部使用大写字母

* 变量名首字母必须小写

* 变量名必须使用其类型的所写字符串开始。各种类型的所写字符串如下：

    整型变量：int

    长整型变量：lng

    浮点型变量：flt

    双精度变量：dbl

    对象引用变量：obj

    字符串变量:str

    Date 类型变量：dtm

* 变量名必须采用有意义的单词命名，如：

strUserName、lngArrayIndex

* 变量名除首字母小写外，其他单词首字符必须大写

* 如果变量名过长可以使用单词缩写，除了被广泛了解的单词缩写以外，所有使用单词所写的变量名必须在定义时给出注释,如：

var strAdName //用于表示 Administrator 帐户的名称

var strAdminName //不用给出注释，Admin 被广泛了解

2.2.2 变量使用规范

* 变量使用前必须定义。没有定义的变量禁止使用

* 变量的使用尽量缩小到小的作用域。如循环使用

```
for(var I=0;I<12;I++){



}
```

而不是：

```
var I;

for(I=0;I<12,I++){
```

}

2.2.3 对象命名规范

各种页面对象如 text 输入框、按钮、下拉选择框在命名时必须使用以下对应前缀：

* text 输入框:txt

* button 按钮：btn

* select 下拉选择框：sel

* option 项:opt

* form 表单：frm

* frame 框架：fra

* hidden 表单项：hdn

* div 标记：div

* span 标记:spn

* 对话框对象:dlg

* 窗口对象：wnd

2.2.4 函数以及子过程命名规范

* 函数的命名采用 Pascal 命名格式，命名内容格式使用**动词 + 名词对**的方式，且命名能够体现函数的功能

* 函数命名的动词前缀必须是同函数功能相关的完整动词

如：GetUserName，CreateNewUser，UpdateUserInfos

3 动态文件编码规范

3.1 命名规范

3.1.1 类的命名规范

* 类名使用 Pascal 命名法。如：public class HelloWorld{ ...}

* 以 Class 声明的类，都必须以名词或名词短语命名，体现类的作用。如：

Class Indicator

* 当类只需有一个对象实例（全局对象，比如 Application 等），必须以 Class 结尾，如：

Class ScreenClass

Class SystemClass

* 当类只用于作为其他类的基类，根据情况，以 Base 结尾。如：

Class IndicatorBase

3.1.2 变量命名规范

变量的基本命名规范：

- 常量必须全部使用 CONST_前缀。
- 变量名首字母必须小写。

- 变量名采用 Camel 命名法（除首字母小写外，其他单词首字符必须大写），命名格式：数据类型缩写+名词对。各种常见数据类型的缩写如下：

  * 整型变量：int

  * 长整型变量：lng

  * 浮点型变量：flt

  * 双精度变量：dbl

  * 对象引用变量：obj

  * 字符串变量:str

  * Date 类型变量：dtm

- 变量名必须采用有意义的单词命名。如： strUserName、lngArrayIndex

- 当变量名过长时，可使用单词缩写。但除了被广泛了解的单词缩写之外，所有缩写的变量名必须在定义的后面给出注释。如：

```
dim strAdName  '用于表示 Administrator 帐户的名称

dim strAdminName  '不用给出注释，Admin 被广泛了解
```

类的成员变量的特有命名规范：

类的所有成员变量前加下划线“_”，以区别于函数参数和函数中的局部变量。

3.1.3 函数命名及使用规范

3.1.3.1 命名规范

* 函数命名使用 Pascal 命名法。第一个单词的首字母大写，后面每一个单词的首字母大写。

* 函数命名格式：动词 + 一个或多个名词。注意：函数命名的动词前缀必须是同函数功能相关的完整动词。并且，函数的命名必须能够体现函数的功能。

如：GetUserName(), CreateNewUser, UpdateUserInfosFromDatabase()

3.1.3.2 使用规则

* 如果函数有传入参数，则必须在函数的头部判断参数的合法性，不合法则不继续操作，并给予响应的提示。

* 尽量使用函数封装代码块

* 连续代码块尽量不要超过 50 行。最多不得超过 70 行

* 尽量使用局部变量。

* 如有涉及到全局的资源（如 Connection），尽量作为函数的参数传入。

* 所有在函数内部创建打开的资源，在退出函数前必须关闭释放。如：sqldatareader

3.1.4 控件命名规范

建议是使用控件名简写作为前缀，并且简写的首字母小写，并且整个名字符合 Camel 规范。

控件命名格式：控件名简写前缀+英文描述

注意：英文描述中的单词首字母大写

主要控件名简写对照表

控件名	简写
Label	lbl
TextBox	txt
Button	btn
CheckBox	chk
RadioButton	rdo
CheckBoxList	chklst

RadioButtonList	rdolst
ListBox	lst
DropDownList	ddl
DataGrid	dg
DataList	dl
Image	img
Table	tbl
Panel	pnl
LinkButton	lnkbtn
ImageButton	imgbtn
Calender	cld
AdRotator	ar
RequiredFieldValidator	rfv
CompareValidator	cv
RangeValidator	rv

RegularExpressionValidator	rev
ValidatorSummary	vs
CrystalReportViewer	rptvew

3.2 注释规范

3.2.1 文件头部注释

在代码文件的头部进行注释，标注出创建人、创建时间、修改人、修改时间、修改内容、代码的功能，这在团队开发中必不可少，它们可以使后来维护/修改的同伴在遇到问题时，在第一时间知道他应该向谁去寻求帮助，并且知道这个文件经历了多少次迭代、经历了多少个程序员的手。

样本：

```
/**********************************************************************************

** 创建人： Eunge

** 创建时间：2004-6-8

** 修改人：Koffer

** 修改时间：2004-12-9

** 修改内容：添加/修改/删除函数 X（ ）
```

** 修改人：Ken

** 修改时间：2005-01-29

** 修改内容：添加/修改/删除函数 Y（ ）

** 描述：

** 主要用于产品信息的资料录入，...

**********/

我们甚至可以在这段文件头注释中加入版权信息、文件名、版本信息等。

3.2.2 函数、属性、类等注释

请使用///三斜线注释，这种注释是基于 XML 的，不仅能导出 XML 制作帮助文档，而且在各个函数、属性、类等的使用中，编辑环境会自动带出注释，方便你的开发。以 protected，protected Internal，public 声明的定义注释请都以这样命名方法。

例如：

/// <summary>

/// 功能：用于从 ERP 系统中捞出产品信息的类

/// 创建人：**

/// 创建时间：2009-2-14

```
/// 修改人：**

/// 修改时间：2009-2-15

/// 修改内容：添加/修改/删除函数（变量）***：

/// </summary>

class ProductTypeCollector

{

    /// <summary>

    /// 产品编号

    /// </summary>

    private string _productId;

    /// <summary>

    /// 产品名称

    /// </summary>

    private string _productName;

    …
```

```
/// <summary>
/// 用于从 ERP 系统中捞出产品信息的类
/// </summary>
public void SaveProductInfos()
{
    …
}

/// <summary>
/// 获取产品信息
/// </summary>
/// <param name=" productId">产品编号</param>
/// <returns>产品信息</returns>
public string GetProductInfos(string productId)
```

```
    {
        //1  参数合法性检查
        …
        //2  操作步骤 2
        //2.1  操作步骤 2.1
        …
        //2.m  操作步骤 2.m
        …
        //3  操作步骤 3
        …
        //n  操作步骤 n
        …
    }
    …
}
```

3.2.3 程序流程及逻辑点注释

在我们认为逻辑性较强的地方加入注释，说明这段程序的逻辑是怎样的，以方便我们自己后来的理解以及其他人的理解，并且这样还可以在一定程度上排除 BUG。在注释中写明我们的逻辑思想，对照程序，判断程序是否符合我们的初衷，如果不是，则我们应该仔细思考修改的是注释还是程序了。

3.3 缩进规范

* 代码排版采用左对齐的方式。

* 相同级别的两行左侧对齐。

* 不同级别的两行相差一个‘Tab’。

如：

```
class User

{

    Public string GetUserName()

    {

        …

        return _userName;
```

```
        }

    }
```

注意：在进行代码缩进时，可使用 VS2005 及其以上版本的开发环境中的自动缩进功能。

3.4 异常处理规范

1、什么时候用 Try catch？什么时候用 Finally?

1) 数据库操作

2) 文件操作

2、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。

3、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。

4、只捕捉特定的异常，而不是一般的异常。

好：

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

不好：

```
void ReadFromFile ( string fileName )
{
    try {     // read from file. }
```

```
                catch (Exception ex)     {

                        // Catching general exception is bad.

                         // 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.

                        return "";

                }

        }
```

5、不必在所有方法中捕捉一般异常。不管它，让程序崩溃。这将帮助你在开发周期发现大多数的错误。

6、你可以用应用程序级（线程级）错误处理器处理所有一般的异常。遇到”以外的一般性错误“时，此错误处理器应该捕捉异常，给用户提示消息，在应用程序关闭或用户选择”忽略并继续“之前记录错误信息。

7、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.

8、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。

9、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。

3.5 Request、Session、Application 使用规范

* 所有需要放入 Session、Application 中的对象，必须采用有意义的英文名字。除了被广泛了解的单词缩写以外，不得采用单词缩写。如：

```
Session(“cp”) = strCurrentUserIP  ‘不允许

Session(“CurrentUserIP”) = strCurrentUserIP

Session(“Pwd”) = strPwd  ‘允许，Pwd 被广泛了解为密码
```

* 所有需要用到的 Request、Session、Application 中的元素的代码中，必须在代码头部赋值给代码内声明的变量。

* 如果获得 Form 中提交的内容，必须使用 Request.Form(“itemName”).

* 如果获得 QueryString 中提交的内容，必须使用 Request.QueryString(“itemName”)，不得在代码中出现 Request(“。。。”)这样的引用方式

4 文件命名规范

4.1 数据库命名规范

4.1.1 数据文件命名规范

命名格式：**系统所属单位+_+系统名称+_+文件类型。**

如：系统所属单位为 lztc，系统名称为 netshop,则数据库文件命名为 lztc_netshop_database.mdf，lztc_netshop_log.log。

注意：文件名全部采用小写。

4.1.2 数据库表命名规范

若该数据库库中有多个系统，表命名格式：**Tab_+系统名称_+单词或多个单词。**系统名是开发系统的缩写，系统名称全部采用小写英文字符,如 Tab_bbs_Title, Tab_bbs_ForumType。

若该数据库库中只含有一个系统，那么表命名格式：**Tab_+单词或多个单词。**单词选择能够概括表内容的一个或多个英文单词，如 Tab_UserInfo, Tab_UserType。

关连表命名规则为 **Re_表 A_表 B。**Re 是 Relative 的缩写，如：Re_User_ArticleType, Re_User_FormType。

注意：表名长度不能超过 30 个字符，表名中含有的单词全部采用单数形式，单词首写字母要大写，多个单词间不用任何连接符号。

4.1.3 数据表字段命名规范

数据表字段命名格式：**表别名+_+单词**

格式说明：字段名全部采用小写英文单词，单词之间用”_”隔开。如：

user_name,user_pwd。

表别名规则：如果表名是一个单词，别名就取单词的前 4 个字母；如果表名是两个单词，就各取两个单词的前两个字母组成 4 个字母长的别名；如果表的名字由 3 个单词组成，你不妨从头两个单词中各取一个然后从最后一个单词中再取出两个字母，结果还是组成 4 字母长的别名。

4.1.4 数据库视图命名规范

视图命名格式：**View_表 A_表 B_表 C。**

格式说明：View 表示视图。这个视图由几个表产生就用”_”连接几个表的名，如果表过多可以将表名适当简化，但一定要列出所有表名。

4.1.5 存储过程命名规范

存储过程命名格式：**Proc_存取过程名_表名**（缩写)。

比如：Proc_User_Del，Proc_ArticleType_AddData。

4.1.6 SQL 语句编写规范

SQL 语句编写规则：关键字必须大写，其他字段书写按上述命名规则。

比如：SELECT user_id, user_name FROM Tab_User WHERE user_id = 'tom'

4.2 文件夹及文件命名规范

文件夹命名一般采用英文，长度一般不超过 20 个字符，命名采用小写字母。除特殊情况才使用中文拼音，一些常见的文件夹命名如：images(存放图形文件)，flash（存放 Flash 文件），style(存放 CSS 文件)，scripts（存放 Javascript 脚本），inc(存放 include 文件)，link（存放友情链接）,media(存放多媒体文件)等。

文件命名统一用小写的英文字母、数字和下划线的组合。命名原则的指导思想一是使得你自己和工作组的每一个成员能够方便的理解每一个文件的意义，二是当我们在文件夹中使用“按名称排例”的命令时，同一种大类的文件能够排列在一起，以便我们查找、修改、替换、计算负载量等等操作。

4.2.1 图片的命名原则

格式：前缀_单词对

前缀表示此图片的大类性质，例如广告、标志、菜单、按钮等。 单词对说明此图片的使用位置。

说明：

- ✧ 放置在页面顶部的广告、装饰图案等长方形的图片取名： banner
- ✧ 标志性的图片取名为： logo
- ✧ 在页面上位置不固定并且带有链接的小图片我们取名为 button
- ✧ 在页面上位置固定并且不带有链接的背景图片我们取名为 backimg
- ✧ 在页面上某一个位置连续出现，性质相同的链接栏目的图片我们取名： menu
- ✧ 装饰用的照片我们取名： pic
- ✧ 不带链接表示标题的图片我们取名： title

范例：

banner_sohu.gif 、banner_sina.gif、 menu_aboutus.gif 、menu_job.gif、 title_news.gif、 logo_police.gif、 logo_national.gif 、pic_people.jpg 、 backimg_notes。

4.2.2 动态语言文件命名规则

格式：性质_描述

说明：描述可以有多个单词，用”_”隔开。性质一般是该页面的概要。

范例：register_form.asp，register_post.asp，topic_lock.asp

5 良好的编程习惯

好的行为形成好的习惯，好的习惯养成好的性格，好的性格编织好的命运。

遵守良好的编程习惯，写出成功的好程序。

5.1 避免使用大文件

如果一个文件里的代码超过 300～400 行，必须考虑将代码分开到不同类中。

5.2 避免写太长的方法

一个典型的方法代码在 1～25 行之间。如果一个方法发代码超过 25 行，应该考虑将其分解为不同的方法。

5.3 方法名需能看出它作什么

别使用会引起误解的名字。如果名字一目了然，就无需用文档来解释方法的功能了。

好：

void SavePhoneNumber (string phoneNumber) { // Save the phone number. }

不好：

```
1. // This method will save the phone number.

2. void SaveData ( string phoneNumber ) { // Save the phone
   number. }
```

5.4 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。

5.5 使用 C# 或 VB.NET 的特有类型

而不是 System 命名空间中定义的别名类型。（为什么）

好： int age; string name; object contactInfo;

不好： Int16 age; String name; Object contactInfo;

5.6 别在程序中使用固定数值

用常量代替。

5.7 别用字符串常数

用资源文件。（为什么）

5.8 必要时使用 enum

别用数字或字符串来指示离散值。

好：

```
enum MailType { Html, PlainText, Attachment }

void SendMail (string message, MailType mailType)

{

        switch ( mailType )

        {

                case MailType.Html:

                // Do something

                break;

                case MailType.PlainText:

                // Do something

                break;
```

```
                case MailType.Attachment:
                // Do something
                break;
                default:
                // Do something
                break;
            }
        }
```

不好：

```
        void SendMail (string message, string mailType)
        {
            switch ( mailType )
            {
                case "Html":
```

```
                    // Do something
                    break;
                    case "PlainText":
                    // Do something
                    break;
                    case "Attachment":
                    // Do something
                    break;
                    default:
                    // Do something
                    break;
                }
            }
```

5.9 别把成员变量声明为 public 或 protected

都声明为 private 而使用 public/protected 的 Properties. 而且，成员变量前缀为“_”。

5.10 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。

永远别设想你的代码是在“C:”盘运行。你不会知道，一些用户在网络或“Z:”盘运行程序。

5.11 人性化消息提示

 错误消息需能帮助用户解决问题。永远别用象"应用程序出错", "发现一个错误" 等错误消息。而应给出象 "更新数据库失败。请确保登陆 id 和密码正确。" 的具体消息。

 显示错误消息时，除了说哪里错了，还应提示用户如何解决问题。不要用 象 "更新数据库失败。"这样的，要提示用户怎么做："更新数据库失败。请确保登陆 id 和密码正确。"

 显示给用户的消息要简短而友好。但要把所有可能的信息都记录下来，以助诊断问题。

5.12 多使用 StringBuilder 替代 String

String 对象是不可改变的。每次使用 System.String 类中的方法之一时，都要在内存中创建一个新的字符串对象，这就需要为该新对象分配新的空间。在需要对字符串执行重复修改的情况下，与创建新的 String 对象相关的系统开销可能会非常昂贵。如果要修改字符串而不创建新的对象，则可以使用 System.Text.StringBuilder 类。例如，当在一个循环中将许多字符串连接在一起时，使用 StringBuilder 类可以提升性能。

以下方法常用于修改 StringBuilder 的内容。

StringBuilder.Append　将信息追加到当前 StringBuilder 的结尾。

StringBuilder.AppendFormat　用带格式文本替换字符串中传递的格式说明符。

StringBuilder.Insert　将字符串或对象插入到当前 StringBuilder 对象的指定索引处。

StringBuilder.Remove　从当前 StringBuilder 对象中移除指定数量的字符。

StringBuilder.Replace　替换指定索引处的指定字符。