# 初心CMS项目规范

## 文件组织

### 1.1文件命名

1.1.1  文件名遵从Pascal（帕斯卡）命名法，无特殊情况，扩展名小写。

1.1.2  使用统一而又通用的文件扩展名：C#类.cs。

### 1.2文件注释

1.2.1 在每个文件头必须包含以下注释说明

/\*----------------------------------------------------------------

    // Copyright (C) 2004 软件有限公司

    // 版权所有。

    //

    // 文件名：

    // 文件功能描述：

    //

// 创建标识：（例如：Brian 20160918）

// 修改标识：（写人名和时间）

    // 修改描述：（写修改的方法名和修改摘要）

    //

    // 修改标识：

    // 修改描述：

--------------------------------------------------------------------\*/

一天内有多个修改的只需做一个在注释说明中做一个修改标识就够了，在所有的代码修改处加上修改标识的注释。

## 代码组织

### 2.1.1声明规范

#### 2.1.1.1变量声明

一行只建议作一个声明，并按字母顺序排列；建议在变量声明时就对其做初始化。

尽量在同一个地方对变量进行声明，避免不同层次的重复命名。

#### 2.1.1.2类和接口的声明

 1、在方法名与其后的左括号间没有任何空格。

 2、左花括号 “{” 出现在声明的下行并与之对齐，单独成行。

 3、方法间用一个空行隔开。

#### 2.1.1.3字段的声明

不要使用是**public**或**protected**的实例字段。如果避免将字段直接公开给开发人员，可以更轻松地对类进行版本控制，原因是在维护二进制兼容性时字段不能被更改为属性。考虑为字段提供**get**和**set**属性访问器，而不是使它们成为公共的。**get**和**set**属性访问器中可执行代码的存在使得可以进行后续改进，如在使用属性或者得到属性更改通知时根据需要创建对象。例如：

 public class Control: Component

 {

    private int handle;

    public int Handle

    {

       get

       {

          return handle;

        }

     }

}

### 2.2.1命名规范

#### 2.2.1.1命名概述

名称应该说明“什么”而不是“如何”。通过避免使用公开基础实现（它们会发生改变）的名称，可以保留简化复杂性的抽象层。

**命名原则是**：

选择正确名称时的困难可能表明需要进一步分析或定义项的目的。使名称足够长以便有一定的意义，并且足够短以避免冗长。唯一名称在编程上仅用于将各项区分开。表现力强的名称是为了帮助人们阅读；因此，提供人们可以理解的名称是有意义的。不过，请确保选择的名称符合适用语言的规则和标准。

以下几点是推荐的命名方法。

1、避免容易被主观解释的难懂的名称。

2、在类属性的名称中包含类名是多余的。

3、只要合适，在变量名的末尾或开头加计算限定符（Avg、Sum、Min、Max、Index）。

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

5、布尔变量名应该包含 **Is**，这意味着**Yes/No**或**True/False**值。

6、在命名状态变量时，避免使用诸如**Flag**的术语。状态变量不同于布尔变量的地方是它可以具有两个以上的可能值。不是使用 documentFlag，而是使用更具描述性的名称，如 documentFormatType。

7、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。 可能的情况下，尽量不要使用原义数字或原义字符串。

#### 2.2.1.2大小写规则

|  |  |  |
| --- | --- | --- |
| 标识符 | 大小写 | 示例 |
| 类 | Pascal | AppDomain |
| 枚举类型 | Pascal | ErrorLevel |
| 枚举值 | Pascal | FatalError |
| 事件 | Pascal | ValueChange |
| 异常类 | Pascal | WebException  注意 总是以 Exception 后缀结尾。 |
| 只读的静态字段 | Pascal | RedValue |
| 接口 | Pascal | IDisposable  注意 总是以 I 前缀开始。 |
| 方法 | Pascal | ToString |
| 命名空间 | Pascal | System.Drawing |
| 属性 | Pascal | BackColor |
| 公共实例字段 | Pascal | RedValue  注意 很少使用。属性优于使用公共实例字段。 |
| 受保护的实例字段 | Camel | redValue  注意 很少使用。属性优于使用受保护的实例字段。 |
| 私有的实例字段 | Camel | redValue |
| 参数 | Camel | typeName |
| 方法内的变量 | Camel | backColor |

### 2.3.1类

1、使用 **Pascal** 大小写。

2、用名词或名词短语命名类。

3、使用全称避免缩写，除非缩写已是一种公认的约定，如URL、HTML

4、不要使用类型前缀，如在类名称上对类使用 C 前缀。例如，使用类名称 FileStream，而不是CFileStream。

5、不要使用下划线字符 (\_)。

6、有时候需要提供以字母I开始的类名称，虽然该类不是接口。只要I是作为类名称组成部分的整个单词的第一个字母，这便是适当的。例如，类名称 IdentityStore 是适当的。在适当的地方，使用复合单词命名派生的类。派生类名称的第二个部分应当是基类的名称。例如，ApplicationException 对于从名为 Exception 的类派生的类是适当的名称，原因ApplicationException 是一种Exception。请在应用该规则时进行合理的判断。例如，Button 对于从 Control 派生的类是适当的名称。尽管按钮是一种控件，但是将 Control 作为类名称的一部分将使名称不必要地加长。

### 2.3.2接口

1、用名词或名词短语，或者描述行为的形容词命名接口。例如，接口名称使用描述性名词、名词短语、形容词。

2、使用 Pascal 大小写。

3、少用缩写。

4、给接口名称加上字母**I**前缀，以指示该类型为接口。在定义类/接口对（其中类是接口的标准实现）时使用相似的名称。两个名称的区别应该只是接口名称上有字母**I**前缀。

5、不要使用下划线字符 (\_)。

6、当类是接口的标准执行时，定义这一对类/接口组合就要使用相似的名称。两个名称的不同之处只是接口名前有一个I前缀。

### 2.3.3特性

应该总是将后缀 Attribute 添加到自定义属（特）性类。例如：

public class ObsoleteAttribute

  {

}

### 2.3.4枚举

  枚举(**Enum**)值类型从**Enum**类继承。

1、对于 **Enum** 类型和值名称使用 Pascal 大小写。

 2、少用缩写。

3、不要在 **Enum** 类型名称上使用 **Enum** 后缀。

4、对大多数 **Enum** 类型使用单数名称，但是对作为位域的 **Enum** 类型使用复数名称。

5、总是将 FlagsAttribute 添加到位域 **Enum** 类型。

### 2.3.5参数

 1、使用描述性参数名称。参数名称应当具有足够的描述性，以便参数的名称及其类型可用于在大多数情况下确定它的含义。

2、对参数名称使用 **Camel** 大小写。

3、使用描述参数的含义的名称，而不要使用描述参数的类型的名称。开发工具将提供有关参数的类型的有意义的信息。因此， 通过描述意义，可以更好地使用参数的名称。少用基于类型的参数名称，仅在适合使用它们的地方使用它们。

4、不要使用保留的参数。保留的参数是专用参数，如果需要，可以在未来的版本中公开它们。相反，如果在类库的未来版本中需要更多的数据，请为方法添加新的重载。

### 2.3.6方法

1、 使用动词或动词短语命名方法。

2 、使用 **Pascal** 大小写。

### 2.3.6属性

1、使用名词或名词短语命名属性。

2、使用 **Pascal** 大小写。

3、不要使用匈牙利语表示法。

4、考虑用与属性的基础类型相同的名称创建属性。

### 2.3.7事件

1、对事件处理程序名称使用 EventHandler 后缀。

2、指定两个名为 sender 和 e 的参数。sender 参数表示引发事件的对象。sender 参数始终是object 类型的，即使在可以使用更为特定的类型时也如此。与事件相关联的状态封装

在名为 e 的事件类的实例中。对 e 参数类型使用适当而特定的事件类。

3、用 EventArgs 后缀命名事件参数类。

4、考虑用动词命名事件。

5、使用动名词（动词的“ing”形式）创建表示事件前的概念的事件名称，用过去式表示事

件后。例如，可以取消的 Close 事件应当具有 Closing 事件和 Closed 事件。不要使用BeforeXxx/AfterXxx 命名模式。

6、不要在类型的事件声明上使用前缀或者后缀。例如，使用 Close，而不要使用 OnClose。

7、通常情况下，对于可以在派生类中重写的事件，应在类型上提供一个受保护的方法（称为OnXxx）。此方法只应具有事件参数 e，因为发送方总是类型的实例

### 2.3.8常量

所有单词大写，多个单词之间用 "\_" 隔开。

例如public const string PAGE\_TITLE = "Welcome";

### 2.3.9字段

1、private、protected 使用 Camel 大小写。

2、public 使用 Pascal 大小写。

3、拼写出字段名称中使用的所有单词。仅在开发人员一般都能理解时使用缩写。字段名称不要使用大写字母。

4、不要对字段名使用匈牙利语表示法。好的名称描述语义，而非类型。

5、不要对字段名或静态字段名应用前缀。具体说来，不要对字段名称应用前缀来区分静态和非静态字段。例如，应用 g\_ 或 s\_ 前缀是不正确的。

6、对预定义对象实例使用公共静态只读字段。如果存在对象的预定义实例，则将它们声明为对象本身的公共静态只读字段。使用 Pascal 大小写，原因是字段是公共的。

### 2.3.10静态字段

1、使用名词、名词短语或者名词的缩写命名静态字段。

2、使用 Pascal 大小写。

3、对静态字段名称使用匈牙利语表示法前缀。

4、建议尽可能使用静态属性而不是公共静态字段。

## 程序注释

### 3.1.1注释概述

1、修改代码时，总是使代码周围的注释保持最新。

2、在每个例程的开始，提供标准的注释样本以指示例程的用途、假设和限制很有帮助。注释样本应该是解释它为什么存在和可以做什么的简短介绍.

3、避免在代码行的末尾添加注释；行尾注释使代码更难阅读。不过在批注变量声明时，行尾注释是合适的；在这种情况下，将所有行尾注释在公共制表位处对齐。

4、避免杂乱的注释，如一整行星号。而是应该使用空白将注释同代码分开。

5、避免在块注释的周围加上印刷框。这样看起来可能很漂亮，但是难于维护。

6、在部署发布之前，移除所有临时或无关的注释，以避免在日后的维护工作中产生混乱。

7、如果需要用注释来解释复杂的代码节，请检查此代码以确定是否应该重写它。尽一切可能不注释难以理解的代码，而应该重写它。尽管一般不应该为了使代码更简单以便于人们使用而牺牲性能，但必须保持性能和可维护性之间的平衡。

8、在编写注释时使用完整的句子。注释应该阐明代码，而不应该增加多义性。

9、在编写代码时就注释，因为以后很可能没有时间这样做。另外，如果有机会复查已编写的代码，在今天看来很明显的东西六周以后或许就不明显了。

10、避免多余的或不适当的注释，如幽默的不主要的备注。

11、 使用注释来解释代码的意图。它们不应作为代码的联机翻译。

12、 注释代码中不十分明显的任何内容。

13、为了防止问题反复出现，对错误修复和解决方法代码总是使用注释，尤其是在团队环境中。

14、对由循环和逻辑分支组成的代码使用注释。这些是帮助源代码读者的主要方面。

15、在整个应用程序中，使用具有一致的标点和结构的统一样式来构造注释。

16、用空白将注释同注释分隔符分开。在没有颜色提示的情况下查看注释时，这样做会使注释很明显且容易被找到。

17、在所有的代码修改处加上[修改标识](http://writeblog.csdn.net/fckeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_FCKEditor&Toolbar=Default" \l "修改标识)的注释。

18、为了是层次清晰，在闭合的右花括号后注释该闭合所对应的起点。

### 3.1.2方法注释

该类注释采用.Net已定义好的Xml标签来标记，在声明接口、类、方法、属性、字段都应该使用该类注释，以便代码完成后直接生成代码文档，让别人更好的了解代码的实现和接口。如

/// <summary>

/// 根据用户名密码登录后台系统

/// </summary>

///修改记录： 修改人 时间 修改内容说明

/// <param name="userName">用户名</param>

/// <param name="passWord">密码</param>

/// <returns></returns>

[HttpPost("Login")]

public async Task<IActionResult> Login(string username, string password)

{

var data = await userService.Login(username, password);

return Json(data);

}

### 3.1.3单行注释

1、 方法内的代码注释。如变量的声明、代码或代码段的解释。

2、 方法内变量的声明或花括号后的注释。

3、 方法内部注释尽量做到50%的覆盖率。

### 3.1.4类注释

每个类必须包含类注释。例如：

/// <summary>

/// 登陆/注册/忘记密码实现功能页面

/// <auther>

/// <name>Kencery</name>

/// <date>2016-9-11</date>

/// </auther>

/// 修改记录：2016-9-11 类创建

/// 修改记录：2016-9-11 修改某方法，修改内容简介。

/// </summary>

注：

1． 匈牙利命名法是一种编程时的命名规范。基本原则是：变量名=属性+类型+对象描述，其中每一对象的名称都要求有明确含义，可以取对象名字全称或名字的一部分。要基于容易记忆容易理解的原则。保证名字的连贯性是非常重要的。

2. 帕斯卡命名法指当[变量](http://baike.baidu.com/view/296689.htm)名和函式名称是由二个或二个以上[单字](http://baike.baidu.com/view/1185088.htm)连结在一起，而构成的唯一识别字时，用以增加变量和函式的可读性。单字之间不以空格断开或[连接号](http://baike.baidu.com/view/55019.htm)（-）、底线（\_）连结，第一个单字首字母采用[大写字母](http://baike.baidu.com/view/257886.htm)；后续单字的首字母亦用大写字母

3. 骆驼式命名法（Camel-Case）又称驼峰命名法，是指混合使用大小写字母来构成变量和函数的名字。第一个函数名使用了骆驼式命名法——函数名中的每一个逻辑断点都有一个大写字母来标记；第二个函数名使用了下划线法----函数名中的每一个逻辑断点都有一个下划线来标记。

例如，下面是分别用骆驼式命名法和下划线法命名的同一个函数：

printEmployeePaychecks()；

print\_employee\_paychecks()；