远洋地产系统规划

开发规范

远洋地产有限公司

版本.1

胡箐

顾问

版本历史

变更记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 作者 | 版本 | 变更内容 |
| 2010-8-3 | 贾彦军 | .1 | 建立文档 |
|  |  |  |  |
|  |  |  |  |

审阅记录

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 审阅版本 | 职位 | 日期 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

内容大纲

[1. 总体介绍 1](#_Toc268621306)

[1.1. 目标 1](#_Toc268621307)

[1.2. 范围 1](#_Toc268621308)

[1.3. 内容 1](#_Toc268621309)

[1.4. 术语 1](#_Toc268621310)

[2. 开发过程管理 2](#_Toc268621311)

[2.1. 任务管理 2](#_Toc268621312)

[2.1.1. 创建任务 2](#_Toc268621313)

[2.1.2. 定义任务 3](#_Toc268621314)

[2.1.3. 将任务与其他工作项关联 5](#_Toc268621315)

[2.1.4. 向任务添加详细信息、附件或超链接 5](#_Toc268621316)

[2.1.5. 更改任务的状态 5](#_Toc268621317)

[2.1.6. 查看任务 6](#_Toc268621318)

[2.2. 源代码管理 7](#_Toc268621319)

[2.2.1. 本地工作区映射 7](#_Toc268621320)

[2.2.2. 新增文件 8](#_Toc268621321)

[2.2.3. 文件签入 9](#_Toc268621322)

[2.2.1. 文件签出 9](#_Toc268621323)

[2.2.2. 查看签出的文件列表 9](#_Toc268621324)

[3. 开发标准 10](#_Toc268621325)

[3.1. 解决方案结构 10](#_Toc268621326)

[3.1.1. Presentation 11](#_Toc268621327)

[3.1.2. Service 11](#_Toc268621328)

[3.1.3. Framework 11](#_Toc268621329)

[3.1.4. Persistence 12](#_Toc268621330)

[3.2. 代码规范 12](#_Toc268621331)

[3.2.1. 编码原则、习惯 12](#_Toc268621332)

[3.2.2. 大小写约定 13](#_Toc268621333)

[3.2.3. 命名规则 15](#_Toc268621334)

[3.2.4. 代码格式 19](#_Toc268621335)

[3.2.5. 代码注释 24](#_Toc268621336)

[3.2.6. 异常处理 26](#_Toc268621337)

[3.2.7. 示例 29](#_Toc268621338)

[3.2.8. Visual Studio格式化设置 34](#_Toc268621339)

[3.2.9. 使用CodeSnippet 39](#_Toc268621340)

[4. 附录 44](#_Toc268621341)

[4.1. 参考资料 44](#_Toc268621342)

[4.2. C#关键字 45](#_Toc268621343)

[4.3. C#文档注释标记 46](#_Toc268621344)

1. 总体介绍
   1. 目标

指导远洋地产信息系统项目建设开发过程中的工作。保证代码规范性和一致性，提高代码质量，降低代码维护难度。

## 范围

本文档主要从项目组织和开发细节的角度进行规范，将通过解决方案组织结构和代码规范两部分规范开发人员代码书写。

本文档不涉及具体的系统实现逻辑，也不涉及具体技术方面的指导。

## 内容

第２节介绍开发过程中涉及到的管理工作的操作，包括管理“任务”工作项，源代码管理。

第3节介绍开发标准。包括系统解决方案结构，和代码开发规范。

第4节列出规范所使用的参考资料。解释开发语言C#中的关键字和注释标记。

## 术语

* SOA：Service Oriented Architecture（面向服务架构）的缩写。通过以服务的视角对分布式系统进行分析和建设，构建灵活，高集成的信息系统，从而提高信息系统和业务的匹配程度，并灵活应对未来业务的变化。
* 系统架构：架构是指对一个事物的宏观描述，而系统架构指的是信息系统整体的组成和各组成的关系描述，通常系统架构设计会是系统整个蓝图的规划，会作为后续系统设计和开发的基础。
* 服务：SOA架构中系统地基本组成，通过定义服务功能和控制要求，而屏蔽服务的内部逻辑，将系统之间的依赖关系建立在服务契约之上，从而解除了系统之间的依赖，提供了灵活性和扩展能力。
* 领域：一定业务上下文的约束，用于对复杂业务场景进行按照一定的方式进行划分，从而专注于某一特定业务场景。领域驱动的开发模式将整个信息系统划分多个领域，并通过对不同领域分别建模的方式对系统进行分解和开发。本系统将采用SOA和领域建模结合的方式进行进行系统建设。
* 模块：系统架构的组成部分，通过将系统划分成多个相关关联的模块，可以将系统功能分解给各个模块完成，从而降低系统的整体复杂性。
* 层：用于软件设计过程中对具有相似依赖关系的部分进行归类的方式，和模块之间没有绝对的包含关系。可以是几个模块构成一层，如数据服务总线和数据库构成总体架构数据层；也可以是模块内部分几层，如数据服务总线内部也含有界面层，逻辑层和数据层。具体关系决定于上下文环境。
* 组件：功能实现的物理部分，通常对应于具体的exe或dll。模块对应可能由多个组件构成。
* ESB：Enterprise Service Bus（企业服务总线）的缩写。系统和服务之间通讯的桥梁，通过在企业架构构建专门的通讯总线，可以有效管理服务管理，进行通讯路由，消息转换和消息监控。

1. 硬件环境
   1. 服务器规划

远洋地产信息系统开发服务器根据不同用途分为四类服务器环境，分别是：

* ALM环境：（application lifecycle management）应用程序生命周期管理环境。
* Dev环境：开发环境。
* Test环境：测试环境。
* Lab环境：试验环境。

服务器机器名命名规则：

* 环境名+“-”+“用途缩写”+“二位数字”。例如：alm-vsts01

服务器规划方案图如所示：



图 17

* 1. ALM环境详细规划

IP段：10.0.9.200 – 10.0.9.219

域：sinooceanland.com

命名规则：alm-\*\*0\*

表 3

|  |  |  |  |
| --- | --- | --- | --- |
| IP | 命名 | 用途 | 安装软件 |
| 200 | alm-vsts01 | TFS前端 | SharePoint Server 2007 |
| 201 | alm-vsts02 | TFS应用服务器 | TFS2010 |
| 202 | alm-db01 | TFS数据库服务器 | SQL Server 2008 |
|  |  |  |  |

表 4

|  |  |  |  |
| --- | --- | --- | --- |
| IP | 硬件(cpu/内存/硬盘) | 帐号 | 帐号用途 |
| 200 | xeon 3.2/4G/68G | ALM\_WSSSERVICE | SharePoint 服务帐号 |
| 201 | xeon 3.2/4G/68G | ALM\_TFSSERVICE | TFS服务帐号 |
| ALM\_TFSBUILD | TFS生成服务帐号 |
| ALM\_TFSPROXY | TFS代理服务帐号 |
| ALM\_TFSREPORTS | TFS报表服务帐号 |
| 202 | xeon 2.8/2G/273G | ALM\_SQLSERVICE | SQL服务帐号 |

* 1. Dev环境详细规划

IP段：10.0.9.220 – 10.0.9.239

域：sinooceanland.com

命名规则：dev-\*\*0\*

表 5

|  |  |  |  |
| --- | --- | --- | --- |
| IP | 命名 | 用途 | 安装软件 |
| 220 | dev-db01 | 开发数据库服务器（备份） | SQL Server 2008 |
| 221 | dev-moss01 | MOSS开发 | SharePoint Server 2007 |
| 222 | dev-app01 | 开发应用 | Visual Studio 2010 |
|  |  |  |  |

表 6

|  |  |  |  |
| --- | --- | --- | --- |
| IP | 硬件(cpu/内存/硬盘) | 帐号 | 帐号用途 |
| 220 | xeon 2.3/16G/136G | DEV\_SQLSERVICE | SQL服务帐号 |
| 221 | xeon 3.2/4G/68G | DEV\_WSSSERVICE | SharePoint 服务帐号 |
| 222 | xeon 3.2/4G/68G |  |  |
|  |  |  |  |

* 1. Test环境详细规划

IP段：10.0.9.240 – 10.0.9.255

域：sinooceanland.com

命名规则：test-\*\*0\*

表 7

|  |  |  |  |
| --- | --- | --- | --- |
| IP | 命名 | 用途 | 安装软件 |
| 240 | test-db01 | 测试数据库服务器 | SQL Server 2008 |
| 241 | test-moss01 | MOSS测试+测试应用 | SharePoint Server 2007 |
|  |  |  |  |

表 8

|  |  |  |  |
| --- | --- | --- | --- |
| IP | 硬件(cpu/内存/硬盘) | 帐号 | 帐号用途 |
| 240 | xeon 3.2/4G/68G\*2 | TEST\_SQLSERVICE | SQL服务帐号 |
| 241 | xeon 3.0/2G/68G | TEST\_WSSSERVICE | SharePoint 服务帐号 |
|  |  |  |  |

* 1. Lab环境详细规划

IP段：10.0.9.180 – 10.0.9.199

宿主机域：sinooceanland.com

虚拟机域：lab.sinooceanland.com

命名规则：lab-\*\*0\*

表 9

|  |  |  |  |
| --- | --- | --- | --- |
| IP | 命名 | 用途 | 安装软件 |
| 180 | lab-host01 | 试验环境宿主机 | Visual Studio 2010 |
| 181-\* | lab-\*\*0\* | 试验环境虚拟机 | — |
|  |  |  |  |

表 10

|  |  |  |  |
| --- | --- | --- | --- |
| IP | 硬件(cpu/内存/硬盘) | 帐号 | 帐号用途 |
| 180 | xeon 3.2/32G/150G | Administrator | 服务器管理员 |
|  |  |  |  |

1. 开发过程管理
   1. 项目访问

项目参与人员可以使用Web方式或通过Visual Studio工具（安装过程请见章节）访问TFS服务器。

* + 1. 通过Web访问

1. 在Web浏览器输入地址<http://alm-vsts02:8080/tfs/web/> ，输入账户信息。如9所示。

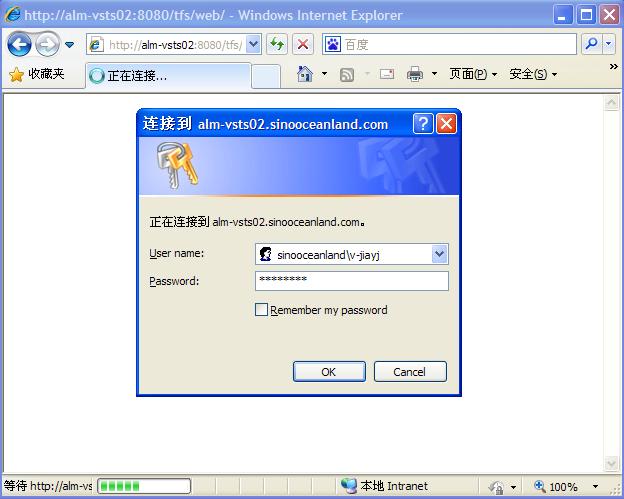


图 9

1. 进入站点后，可以查看当前项目内容，如站点导航栏所示，包括项目工作项、项目源代码。左侧下拉列表框显示了权限范围内项目列表，可点击切换。如10所示。

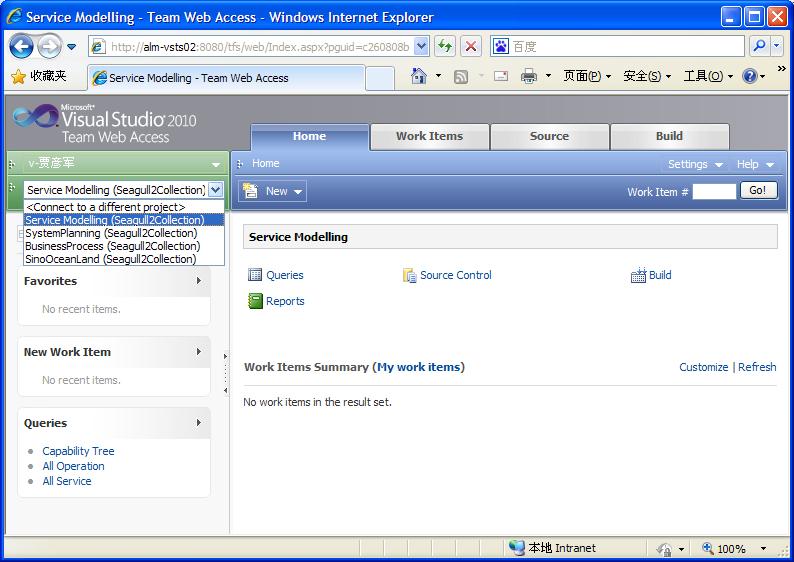


图 10

* + 1. 通过项目门户访问

1. 在Web浏览器输入地址http://alm-vsts01/tfs/Seagull2Collection/项目名称/ ，输入账户信息。
2. 进入项目门户后通过左侧Team　Web　Access链接访问，访问过程同。
   * 1. 通过Visual　Studio工具访问
3. 打开VS，点击Team菜单栏，选择“Connection to Team　Foundation　Server”项，如11所示。

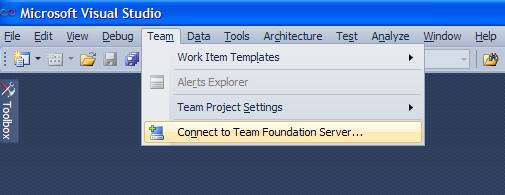


图 11

1. 出现“Connect to Team　Project”对话框，如12所示。如果是第一次连接TFS服务器，需要添加服务器，点击“Servers”进入添加服务器主页面，如12所示。

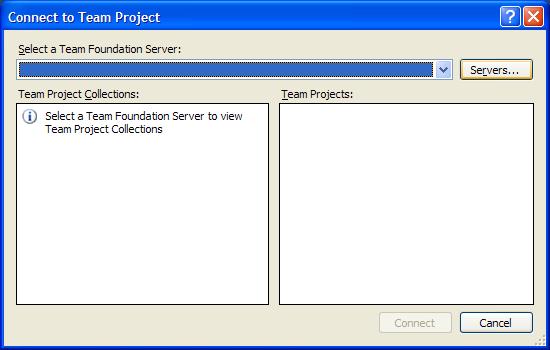


图 12

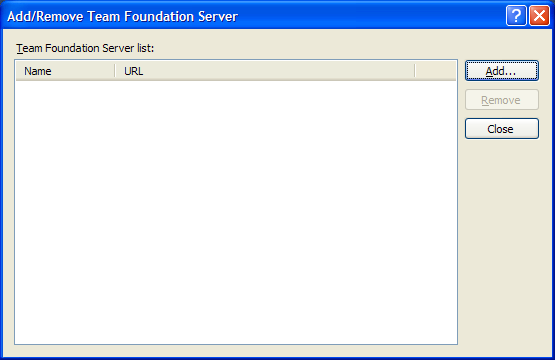


图 13

1. 点击“Add”新增服务器，输入服务器地址：<http://alm-vsts02:8080/tfs/seagull2collection>后点击确定，如14所示。

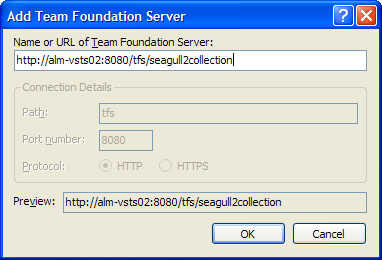


图14

1. 返回“Connect　ｔｏ　Team　Project”对话框，此时列表中出现刚才增加的服务器名称，输入身份信息后即可选择相应的项目进行连接。如5所示。

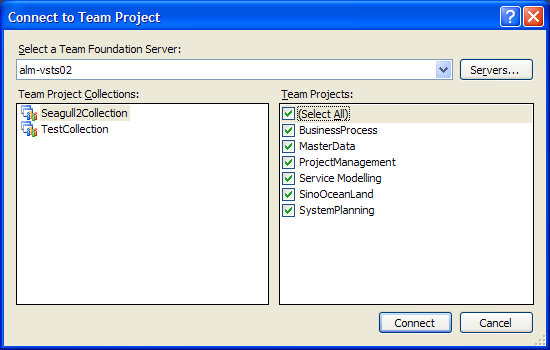


图 105

1. 成功连接后，点击“View”菜单，选择“Team　Explorer”，通过“Team　Explorer”查看项目内容，如所示。

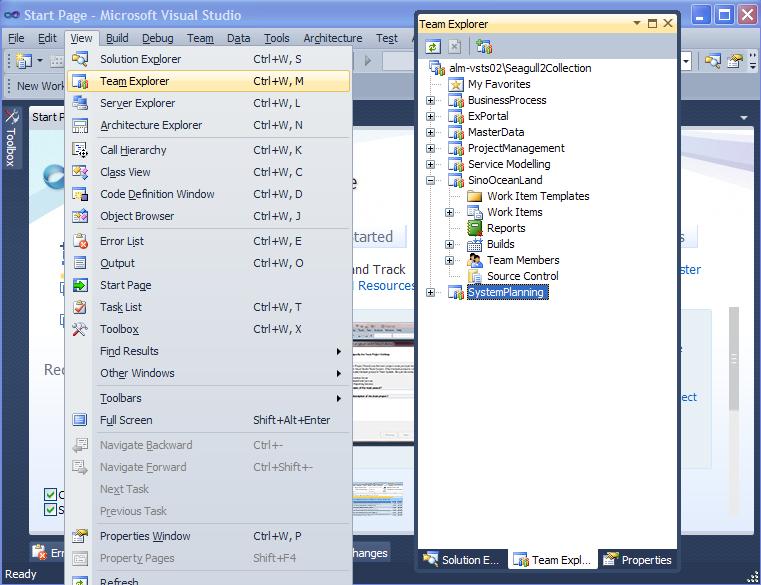


图 11

注：当使用VS2005、VS2008工具时需要安装兼容包，兼容包地址：\\10.0.9.201\Visual Studio Team System升级包

* 1. 源代码管理

源代码管理是开发过程中重要的环节，源代码管理也是开发人员必须掌握的内容。远洋地产信息系统采用TFS进行源代码管理。TFS源代码管理中有众多操作，这些内容是开发人员经常使用并且应该非常熟悉的内容，下面介绍一些基本操作，完整、详细的源代码管理操作请见[附录](#VisualStudio的源代码管理)。

* + 1. 本地工作区映射

选择需要映射的源代码分支，如果本地路径显示“未映射”，则需要将服务器文件映射到本地工作区，点击“未映射”链接出现映射对话框，选择本地工作区路径即可，如图 5所示。

如需修改本地工作区，可以进行“移除映射”操作，移除后可以重新进行映射。

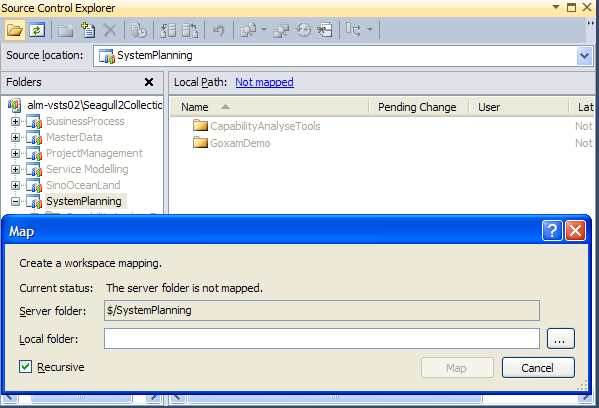


图 5

* + 1. 新增文件

进入源代码控制浏览器，选择要新增文件的节点，然后点击左上“+”新增项按钮，在出现的对话框中选择需要新增的文件，点击完成即可。如图 6所示：

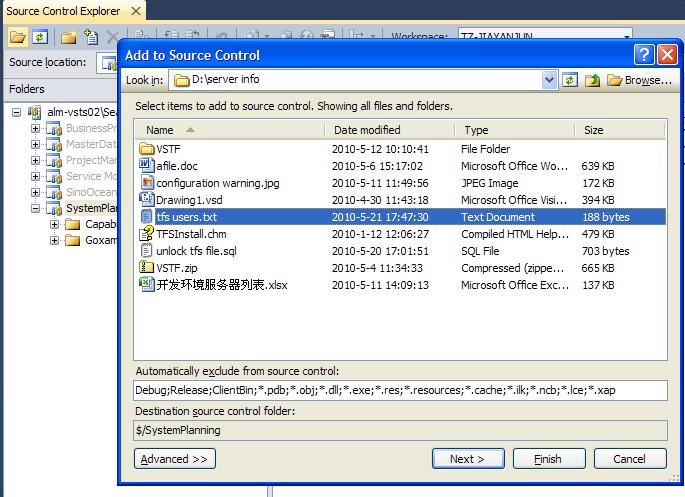


图 6

* + 1. 文件签入

文件编辑完成后，点击右键选择“签入”，“签入”后的文件将发布到服务器上。签入时应在注释框内添加此次编辑过的内容，指定文件编辑所关联的工作项内容。填写这些内容也是项目的强制要求。

当不需要提交修改时选择“撤销挂起的更改”选项。注意此选项会将本地工作区内的文件一并撤销，如果内容有用，应做好备份工作。

* + 1. 文件签出

选择需要签出的文件，获取“最新版本”后，点击签出即可。

* + 1. 查看签出的文件列表

在 Visual Studio 的“视图”菜单上单击“挂起签入”。“挂起签入”窗口中显示所有签出的文件。

# 开发标准

* 1. 代码规范
     1. 编码原则、习惯

1. 尽量保持代码的简洁、简单，代码能够清晰的表达设计思路。
2. 代码应具备自解释能力，通过良好的命名表达代码逻辑，提高代码可阅读性。
3. 避免魔数，将其替代为可以表示含义的枚举、常量、变量。
4. 减少单个函数的长度，尽量控制在30行内。函数体内容顺序为：
5. 参数检查 — 参数转换 — 核心逻辑 — 返回值转换 — 返回
   * 1. 大小写约定
        1. 大小写样式
6. Pascal 大小写

将标识符的首字母和后面连接的每个单词的首字母都大写。可以对三字符或更多字符的标识符使用 Pascal 大小写。例如：BackColor

1. Camel大小写

标识符的首字母小写，而每个后面连接的单词的首字母都大写。例如：backColor

* + - 1. 标识符大小写规则

下表中列出常见的代码元素的样式规范和示例

|  |  |  |
| --- | --- | --- |
| 标识符 | 样式 | 示例 |
| 类 | Pascal | AppDomain |
| 枚举类型 | Pascal | ErrorLevel |
| 枚举值 | Pascal | FatalError |
| 事件 | Pascal | ValueChanged |
| 异常类 | Pascal | WebException |
| 只读的静态字段 | Pascal | RedValue |
| 接口 | Pascal | IDisposable |
| 方法 | Pascal | ToString |
| 命名空间 | Pascal | System.Drawing |
| 参数 | Camel | typeName |
| 属性 | Pascal | BackColor |
| 常量 | 大写 | REGEX\_PARSE\_URL |

除了遵循以上大小写约定外还需注意以下约定（除常量为特例）：

1. 如果标识符由多个单词组成，请不要在各单词之间使用分隔符，如下划线（“\_”）或连字符（“-”）等。而应使用大小写来指示每个单词的开头。
2. 所有公共的成员如：方法、属性，都应使用Pascal大小写样式
   * + 1. 首缩写词的大小写规则

首字母缩写词是由术语或短语中各单词的首字母构成的单词。例如，HTML 是 Hypertext Markup Language 的首字母缩写。只有在公众广为认知和理解的情况下，才应在标识符中使用首字母缩写词。首字母缩写词不同于缩写词，因为缩写词是一个单词的缩写。例如，ID 是 identifier 的缩写。通常情况下，库名不应使用缩写词。

首字母缩写词的大小写取决于首字母缩写词的长度。所有首字母缩写词都至少包含两个字符。为方便说明准则，如果一个首字母缩写词只包含两个字符，则将其视为短型首字母缩写词。包含三个或三个以上字符的首字母缩写词为长型首字母缩写词。

下列准则为短型和长型首字母缩写词指定了正确的大小写规则。标识符大小写规则优先于首字母缩写词大小写规则。

1. 两字符首字母缩写词的两个字符都要大写，但当首字母缩写词作为大小写混合格式的标识符的首个单词时例外。例如，名为 DBRate 的属性是一个采用 Pascal 大小写格式的标识符，它使用短型首字母缩写词 (DB) 作为首个单词。又如，名为 ioChannel 的参数是一个采用大小写混合格式的标识符，它使用短型首字母缩写词 (IO) 作为首个单词。
2. 包含三个或三个以上字符的首字母缩写词只有第一个字符大写，但当首字母缩写词作为大小写混合格式的标识符的首个单词时例外。例如，名为 XmlWriter 的类是一个采用 Pascal 大小写格式的标识符，它使用长型首字母缩写词作为首个单词。又如，名为 htmlReader 的参数是一个采用大小写混合格式的标识符，它使用长型首字母缩写词作为首个单词。
3. 如果任何首字母缩写词位于采用大小写混合格式的标识符开头，则无论该首字母缩写词的长度如何，都不大写其中的任何字符。例如，名为 xmlStream 的参数是一个采用大小写混合格式的标识符，它使用长型首字母缩写词 (xml) 作为首个单词。名为 dbServerName 的参数是一个采用大小写混合格式的标识符，它使用短型首字母缩写词 (db) 作为首个单词。
   * + 1. 组合词和常用术语的大小写规则

不要将所谓的紧凑格式组合词中的每个单词都大写。这种组合词是指写作一个单词的组合词，如“endpoint”。例如，hashtable 是一个紧凑格式的组合词，应将其视为一个单词并相应地确定大小写。如果采用 Pascal 大小写格式，则该组合词为 Hashtable；如果采用大小写混合格式，则该组合词为 hashtable。若要确定某个单词是否是紧凑格式的组合词，请查阅最新的词典。

下表列出了不是紧凑格式组合词的一些常用术语。术语先以 Pascal 大小写格式显示，后面的括号中的是其大小写混合格式。

BitFlag (bitFlag)

FileName (fileName)

LogOff (logOff)

LogOn (logOn)

SignIn (signIn)

SignOut (signOut)

UserName (userName)

WhiteSpace (whiteSpace)

* + 1. XML命名规则

节点名称使用Pascal样式，属性名称使用Camel样式。示例如下：

<Instances refID="1">

<Customer id="1" type="Customer">

<Property name="Name" value="Jack" isRef="false" />

<Property name="Age" value="1" isRef="false" />

<Property name="Order" value="2" isRef="true" />

<Customer>

<Order id="2" type="Order">

<Property name="OrderID" value="111111" isRef="false" />

<Property name="Address" value="东城区" isRef="false" />

<Property name="Money" value="1000" isRef="false" />

<Property name="Product" value="4" isRef="true" />

<Property name="ProductList" value="6" isRef="true" />

<Order>

<ProductList id="6" type="Object">

<Items>

<Item id="7" type="int" />

</Items>

</ProductList>

<Product id="4" type="Product">

<Property name="ProductName" value="茶杯" isRef="false" />

<Property name="Price" value="1000" isRef="false" />

<Property name="ProductImage" value="5" isRef="true" />

<Product>

<Image id="5" type="Image">

<value>

base64string

</value>

</Image>

</Instances>

* + 1. 数据库命名规则
       1. 通用命名规则

1. 各种命名全部采用大写命名，表名采用以系统模块作为前缀的方式，单词之间使用“\_”分隔符
2. 所有的SQL关键字全部采用大写
3. 所有的表及字段尽可能的使用完整的单词货词组来表示，如果超过允许的长度例外。
   * + 1. 表命名
4. 每个表必须有一个表所属的模块前缀，该前缀在模块设计是定义好，模块前缀与表名之间使用下划线连接（之所以使用表前缀及使用下划线，主要是考虑编码时IDE的智能引用比较方便方便），比如SYSTEM\_ROLE，表示系统模块的角色表。
5. 表名以单数形式表示。
6. 关联表的命名使用被关联的表名组合命名，如:SYSTEM\_ROLEMODULEACTION表示系统模块下的角色模块操作表
   * + 1. 字段命名
7. 如果字段为bool型，则使用"ISSHOW","ISVALID","HASPASSED","HASEXAMINED","IGNORECASE"这种形式表示。常见的bool类型关键字有：Is,Has等
8. 日期类型的字段必须包含Date关键字,时间类型的字段必须包含Time
   * + 1. 存储过程命名

SP\_表名\_方法,如：SP\_NEWS\_ADD,SP\_NEWS\_UPDATE;

* + - 1. 视图命名

VI\_表名，如：VI\_NEWS;

* + 1. C#命名规则
       1. 通用命名规则

通用命名规则讨论的是如何为库元素选择最适当的名称。这些准则适用于所有标识符。后面各节讨论特定元素（如命名空间或属性）的命名。

**选择名称**

1. 请选择易读的标识符名称。例如，英文属性名称 HorizontalAlignment 比 AlignmentHorizontal 更具可读性。
2. 可读性比简洁性更重要。属性名称 CanScrollHorizontally 比 ScrollableX（指 X 轴，但意义不明确）更好。
3. 不要使用下划线、连字符或任何其他非字母数字字符。
4. 不要使用匈牙利表示法。匈牙利表示法是在标识符中使用一个前缀对参数的某些元数据进行编码，如标识符的数据类型。
5. 避免使用与常用编程语言的关键字（C#关键字见附件1）冲突的标识符。
   * + 1. 程序集和 DLL 的命名规则

一定要为程序集 DLL 选择指示大的功能块（如 System.Data）的名称。程序集和 DLL 的名称不必对应于命名空间名称，但是在命名程序集时沿用命名空间名称是合理的。

考虑按下面的模式命名 DLL：<Company>.<Component>.dll

其中 <Component> 包含一个或多个以圆点分隔的子句。例如，SinoOcean.Seagull2.Common.dll。

* + - 1. 命名空间的命名规则

为命名空间选择的名称应指示命名空间中的类型所提供的功能。例如，System.Net.Sockets 命名空间包含的类型允许开发人员使用套接字通过网络进行通信。

命名空间名称的一般格式如下：<Company>.(<Product>|<Technology>)[.<Feature>][.<Subnamespace>]

例如，SinoOcean.Seagull2.Framework.ProcessEngine.RuleEngine 。

使用 Pascal 大小写格式，并用句点分隔命名空间各部分（如 Microsoft.Office.PowerPoint）。

适当的时候可考虑使用复数命名空间名称。例如，使用 System.Collections，而不要使用 System.Collection。

命名空间和其中的类型不要使用相同的名称。例如，不要在将“Debug”用作命名空间名称的同时，又在该命名空间中提供一个名为“Debug”的类。

命名空间一般准则

不要引入宽泛的类型名称，如 Element、Node、Log 和 Message。在通常情况下，这样极可能导致类型名称冲突。应对宽泛的类型名称进行限定（例如 FormElement、XmlNode EventLog、SoapMessage）。

* + - 1. 类、结构和接口的命名规则

1. 按照 Pascal 大小写格式，使用名词或名词短语（或偶尔使用形容词短语）为类、接口和值类型命名。
2. 接口应以字母 I 开头，以指示该类型为接口。
3. 考虑在派生类的末尾使用基类名称。例如，从 Stream 继承的 Framework 类型以 Stream 结尾，从 Exception 继承的类型以 Exception 结尾。
4. 泛型类型参数的名称

用描述性名称为泛型类型参数命名，除非单个字母的名称已完全可以自我说明而无需描述性名称。如：IDictionary<TKey, TValue>

对具有一个单字母类型参数的类型，考虑将字母 T 用作这些类型的类型参数名称。

考虑在参数名称中指示置于类型参数上的约束。例如，可以将带有 ISession 约束的参数命名为 TSession。

1. 为继承自 Exception 的类型添加 Exception 后缀。
2. 向自定义特性类添加 Attribute 后缀。
3. 向在事件中使用的类型（如 C# 事件的返回类型）的名称添加 EventHandler 后缀。
4. 向扩展 System.EventArgs 的类添加 EventArgs 后缀。
5. 向实现 System.Collections.IDictionary 或 System.Collections.Generic.IDictionary<TKey, TValue> 的类型添加 Dictionary 后缀。注意，System.Collections.IDictionary 是特定类型的集合，但此准则的优先级高于以下更为一般的集合准则。
6. 向实现 System.Collections.IEnumerable、System.Collections.ICollection、System.Collections.IList、System.Collections.Generic.IEnumerable<T>、System.Collections.Generic.ICollection<T> 或 System.Collections.Generic.IList<T> 的类型添加 Collection 后缀。
7. 向从 System.IO.Stream 继承的类型添加 Stream 后缀。
8. 向从 System.Security.CodeAccessPermission 继承的类型或实现 System.Security.IPermission 的类型添加 Permission 后缀。
9. 枚举的名称

不要将 Enum 用作枚举类型的后缀。

不要在标志枚举的名称中添加 Flags 作为后缀。

对枚举使用单数名称，除非枚举值是位域。

对使用位域值的枚举（也称为标志枚举）使用复数名称。

不要在枚举值名称中使用前缀。例如，不要对 ADO 枚举使用 ad 之类的前缀，也不要对多格式文本枚举使用 rtf 之类的前缀，依此类推。

* + - 1. 方法的命名规则

使用动词或动词短语作为方法的名称。通常，方法对数据进行操作，因此，使用动词描述方法的操作可使开发人员更易于了解方法所执行的操作。定义由方法执行的操作时，应从开发人员的角度仔细选择明确的名称。不要选择描述方法如何执行其操作的动词，也就是说，不要使用实现细节作为方法名称。

* + - 1. 属性的命名规则

1. 使用名词、名词短语或形容词作为属性的名称。名词短语或形容词适合于属性，因为属性保存数据。
2. 不要使用与 Get 方法同名的属性。例如，不要将一个属性命名为 EmployeeRecord，又将一个方法命名为 GetEmployeeRecord。开发人员会不知道使用哪个成员来完成其编程任务。
3. 使用肯定性短语作为布尔值属性的名称（如使用 CanSeek 而不使用 CantSeek）。此外，还可以为布尔值属性添加前缀（如 Is、Can 或 Has），但要注意使用得当。考虑为属性提供与其类型相同的名称。如果某个属性已强类型化为某个枚举，则该属性可与该枚举同名。例如，如果有一个名为 CacheLevel 的枚举，则返回其中一个枚举值的属性也可以命名为 CacheLevel。
   * + 1. 事件的命名规则
4. 使用动词或动词短语作为事件的名称。
5. 在为事件命名时，使用现在时或过去时表示时间上的前后概念。例如，在窗口关闭之前引发的关闭事件可命名为“Closing”，在窗口关闭之后引发的关闭事件可命名为“Closed”。
6. 不要使用“Before”或“After”作为前缀或后缀来指示之前和之后发生的事件。
7. 在事件处理程序签名中使用命名为“sender”和“e”的两个参数。sender 参数的类型应为 Object，e 参数应是 EventArgs 的实例或继承自它。
   * + 1. 字段的命名规则
8. 字段的命名准则适用于静态公共字段和静态受保护字段。不要定义公共实例字段或受保护实例字段。
9. 在字段名称中使用 Pascal 大小写格式。
10. 使用名词或名词短语作为字段的名称。
11. 不要在字段名称中使用前缀。例如，不要使用 g\_ 或 s\_ 来区分静态字段和非静态字段。
    * + 1. 参数的命名规则
12. 对参数名称使用 Camel 大小写。
13. 使用描述性参数名称。在大多数情况下，参数名称及其类型应足以确定参数的用法。
14. 考虑使用反映参数含义的名称而不是反映参数类型的名称。
    * 1. 代码格式
         1. 列宽

代码列宽控制在110字符左右。

* + - 1. 换行

当表达式超出或即将超出规定的列宽，遵循以下规则进行换行

1. 在逗号后换行。
2. 在操作符前换行。
3. 规则1优先于规则2。

当以上规则会导致代码混乱的时候自己采取更灵活的换行规则。

* + - 1. 缩进

缩进应该是每行一个Tab，保持在代码中使用Tab字符。

Visual Studio.Net设置：工具->选项->文本编辑器->C#->制表符->插入空格 取消选项

* + - 1. 空行

空行是为了将逻辑上相关联的代码分块，以便提高代码的可阅读性。

在以下情况下使用两个空行

1. 接口和类的定义之间。
2. 枚举和类的定义之间。
3. 类与类的定义之间。

在以下情况下使用一个空行

1. 方法与方法、属性与属性之间。
2. 方法中变量声明与语句之间。
3. 方法与方法之间。
4. 方法中不同的逻辑块之间。
5. 方法中的返回语句与其他的语句之间。
6. 属性与方法、属性与字段、方法与字段之间。
7. 注释与它注释的语句间不空行，但与其他的语句间空一行。
   * + 1. 空格

在以下情况中要使用到空格

1. 关键字和左括符 “(” 应该用空格隔开。如while (true) 注意在方法名和左括符 “(” 之间不要使用空格，这样有助于辨认代码中的方法调用与关键字。
2. 多个参数用逗号隔开，每个逗号后都应加一个空格。
3. 除了 . 之外，所有的二元操作符都应用空格与它们的操作数隔开。一元操作符、++及--与操作数间不需要空格。如

a += c + d;

a = (a + b) / (c \* d);

while (d++ = s++)

{

n++;

}

PrintSize(“size is “ + size + “\n”);

1. 语句中的表达式之间用空格隔开。如 for (expr1; expr2; expr3)
   * + 1. 括号
2. 左括号“(” 不要紧靠关键字，中间用一个空格隔开。
3. 左括号“(” 与方法名之间不要添加任何空格。
4. 没有必要的话不要在返回语句中使用()。
   * + 1. 花括号
5. 左花括号 “{” 放于关键字或方法名的下一行并与之对齐。如

if (condition)

{

}

public int Add(int x, int y)

{

}

1. 左花括号 “{” 要与相应的右花括号 “}”对齐。
2. 通常情况下左花括号 “{”单独成行，不与任何语句并列一行。
3. if、while、do语句后一定要使用{}，即使{}号中为空或只有一条语句。如

if (somevalue == 1)

{

somevalue = 2;

}

* + - 1. 声明

1. 每行声明数

一行只建议作一个声明，并按字母顺序排列。如

int level; //推荐

int size; //推荐

int x, y; //不推荐

1. 初始化

建议在变量声明时就对其做初始化。

1. 位置

变量建议置于块的开始处，不要总是在第一次使用它们的地方做声明。如

void MyMethod()

{

int int1 = 0; // beginning of method block

if (condition)

{

int int2 = 0; // beginning of "if" block

...

}

}

应避免不同层次间的变量重名。如

int count;

...

void MyMethod()

{

int count = 0; // 避免

}

1. 类和接口的声明

在方法名与其后的左括号间没有任何空格。

左花括号 “{” 出现在声明的下行并与之对齐，单独成行。

方法间用一个空行隔开。

1. 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。下面的代码示例阐释带有get 和 set 属性访问器的私有实例字段的正确使用。 示例：

public class Control: Component

{

private int handle;

public int Handle

{

Get

{

return handle;

}

}

}

* + 1. 代码注释
       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的注释。
    * + 1. 文件注释

每个.cs文件的开始位置使用该注释

// <copyright file="XX.cs" company="×××有限公司">您的文件信息</copyright>

* + - 1. 文档型注释

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>MyMethod is a method in the MyClass class.

///<para>Here's how you could make a second paragraph in a description.

///<see cref="System.Console.WriteLine"/>

///for information about output statements.

///</para>

///<seealso cref="MyClass.Main"/>

///</summary>

public static void MyMethod(int Int1)

{

}

详细文档注释标签请见附件2

* + - 1. 多行注释

该类注释用于

1. 不再使用的代码。
2. 临时测试屏蔽某些代码。

用法

/\*

[修改标识]

[修改原因]

. . . (the source code )

\*/

* + - 1. 单行注释

该类注释用于：方法内的代码注释。如变量的声明、代码或代码段的解释。

注意：//与注释语句之间有一个空格

注释示例：

//// 注释语句

private int number;

或者：空行（推荐方式）

// 注释语句

private int number;

或者：

private int number; // 注释语句（使用于批量的变量声明）

* + 1. 异常处理
       1. 使用异常返回错误信息

程序内部应使用抛出异常来处理错误信息，并在最上层应用程序捕获并处理异常。异常分为：

1. 公共语言运行时异常，常用如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 异常类型 | 基类型 | 说明 | 示例 |
| [Exception](http://msdn.microsoft.com/zh-cn/library/5whzhsd2.aspx) | Object | 所有异常的基类。 | 无（使用此异常的派生类）。 |
| [SystemException](http://msdn.microsoft.com/zh-cn/library/system.systemexception.aspx) | Exception | 所有运行时生成的错误的基类。 | 无（使用此异常的派生类）。 |
| [IndexOutOfRangeException](http://msdn.microsoft.com/zh-cn/library/system.indexoutofrangeexception.aspx) | SystemException | 仅当错误地对数组进行索引时，才由运行时引发。 | 在数组的有效范围外对数组进行索引：  arr[arr.Length+1] |
| [NullReferenceException](http://msdn.microsoft.com/zh-cn/library/system.nullreferenceexception.aspx) | SystemException | 仅当引用 null 对象时，才由运行时引发。 | object o = null;  o.ToString(); |
| [AccessViolationException](http://msdn.microsoft.com/zh-cn/library/system.accessviolationexception.aspx) | SystemException | 仅在访问无效内存时由运行时引发。 | 当与非托管代码或不安全的托管代码互操作时或者使用无效指针时发生。 |
| [InvalidOperationException](http://msdn.microsoft.com/zh-cn/library/system.invalidoperationexception.aspx) | SystemException | 当处于无效状态时，由方法引发。 | 从基础集合移除 Item 后调用 Enumerator.GetNext()。 |
| [ArgumentException](http://msdn.microsoft.com/zh-cn/library/system.argumentexception.aspx) | SystemException | 所有参数异常的基类。 | 无（使用此异常的派生类）。 |
| [ArgumentNullException](http://msdn.microsoft.com/zh-cn/library/system.argumentnullexception.aspx) | ArgumentException | 由不允许参数为 null 的方法引发。 | String s = null;  "Calculate".IndexOf (s); |
| [ArgumentOutOfRangeException](http://msdn.microsoft.com/zh-cn/library/system.argumentoutofrangeexception.aspx) | ArgumentException | 由验证参数是否位于给定范围内的方法引发。 | String s = "string";  s.Chars[9]; |
| [ExternalException](http://msdn.microsoft.com/zh-cn/library/system.runtime.interopservices.externalexception.aspx) | SystemException | 在运行时的外部环境中发生或针对这类环境的异常的基类。 | 无（使用此异常的派生类）。 |
| [ComException](http://msdn.microsoft.com/zh-cn/library/system.runtime.interopservices.comexception.aspx) | ExternalException | 封装 COM HRESULT 信息的异常。 | 在 COM 互操作中使用。 |
| [SEHException](http://msdn.microsoft.com/zh-cn/library/system.runtime.interopservices.sehexception.aspx) | ExternalException | 封装 Win32 结构化异常处理信息的异常。 | 在非托管代码 interop 中使用。 |

1. 程序生成的异常（自定义异常）
   * + 1. 异常信息的描述
2. 存在的问题
3. 违反的规律
4. 可能的解决方案

如：Nod无法打开文件nod.config：请确认该文件存在并且当前用户拥有足够的权限

如：OPath解析查询语句错误：[和]必须匹配，请检查[]的完整性

* + - 1. 异常处理原则

1. 不要通过在框架代码中捕捉非特定异常（如 System.Exception、System.SystemException 等）来处理错误。如果捕捉异常是为了再次引发或传输给其他线程，则可以捕捉这些异常。
2. 避免通过在应用程序代码中捕捉非特定异常（如 System.Exception、System.SystemException 等）来处理错误。某些情况下，可以在应用程序中处理错误，但这种情况极少。应用程序不应该处理异常，否则可能导致意外状态或可利用状态。如果不能预知所有可能的异常原因，也不能确保恶意代码不能利用产生的应用程序状态，则应该允许应用程序终止，而不是处理异常。
3. 如果捕捉异常是为了传输异常，则不要排除任何特殊异常。只捕捉能够合法处理的异常，而不要在 catch 子句中创建特殊异常的列表。在非特定异常处理程序中，不能处理的异常不应视为特殊处理的特殊情况。
4. 如果了解特定异常在给定上下文中引发的条件，请考虑捕捉这些异常。应该只捕捉可以从中恢复的异常。例如，试图打开不存在的文件而导致的 FileNotFoundException 可以由应用程序处理，因为应用程序可以将问题传达给用户，并允许用户指定其他文件名或创建该文件。如果打开文件的请求会生成 ExecutionEngineException，则不应该处理该请求，因为没有任何把握可以了解该异常的基础原因，应用程序也无法确保继续执行是安全的。
5. 不要过多使用 catch。通常应允许异常在调用堆栈中往上传播。捕捉无法合法处理的异常会隐藏关键的调试信息。
6. 使用 try-finally 并避免将 try-catch 用于清理代码。在书写规范的异常代码中，try-finally 远比 try-catch 更为常用。使用 catch 子句是为了允许处理异常（例如，通过纪录非致命错误）。无论是否引发了异常，使用 finally 子句即可执行清理代码。如果分配了昂贵或有限的资源（如数据库连接或流），则应将释放这些资源的代码放置在 finally 块中。
7. 捕捉并再次引发异常时，首选使用空引发。这是保留异常调用堆栈的最佳方式。
   * 1. 示例

下面是摘自Microsoft Patterns&Practice Enterprise Library 源代码片段，用于代码规范的总体示例。

//===============================================================================

// Microsoft patterns & practices Enterprise Library

// Data Access Application Block

//===============================================================================

// Copyright ? Microsoft Corporation. All rights reserved.

// THIS CODE AND INFORMATION IS PROVIDED "AS IS" WITHOUT WARRANTY

// OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT

// LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND

// FITNESS FOR A PARTICULAR PURPOSE.

//===============================================================================

using System;

using System.Data;

using System.Data.Common;

using System.Globalization;

using System.Transactions;

using Microsoft.Practices.EnterpriseLibrary.Data.Instrumentation;

using Microsoft.Practices.EnterpriseLibrary.Data.Properties;

namespace Microsoft.Practices.EnterpriseLibrary.Data

{

/// <summary>

/// Represents an abstract database that commands can be run against.

/// </summary>

/// <remarks>

/// The <see cref="Database"/> class leverages the provider factory model from ADO.NET. A database instance holds

/// a reference to a concrete <see cref="DbProviderFactory"/> object to which it forwards the creation of ADO.NET objects.

/// </remarks>

public abstract class Database

{

static readonly ParameterCache parameterCache = new ParameterCache();

static readonly string VALID\_PASSWORD\_TOKENS = Resources.Password;

static readonly string VALID\_USER\_ID\_TOKENS = Resources.UserName;

private readonly ConnectionString connectionString;

private readonly DbProviderFactory dbProviderFactory;

/// <summary>

/// The <see cref="IDataInstrumentationProvider"/> instance that defines the logical events used to instrument this <see cref="Database"/> instance.

/// </summary>

protected IDataInstrumentationProvider instrumentationProvider;

/// <summary>

/// Initializes a new instance of the <see cref="Database"/> class with a connection string and a <see cref="DbProviderFactory"/>.

/// </summary>

/// <param name="connectionString">The connection string for the database.</param>

/// <param name="dbProviderFactory">A <see cref="DbProviderFactory"/> object.</param>

protected Database(string connectionString,

DbProviderFactory dbProviderFactory)

: this(connectionString, dbProviderFactory, new NullDataInstrumentationProvider())

{

}

/// <summary>

/// Initializes a new instance of the <see cref="Database"/> class with a connection string,

/// a <see cref="DbProviderFactory"/> and an <see cref="IDataInstrumentationProvider"/>.

/// </summary>

/// <param name="connectionString">The connection string for the database.</param>

/// <param name="dbProviderFactory">A <see cref="DbProviderFactory"/> object.</param>

/// <param name="instrumentationProvider">Instrumentation provider to use.</param>

protected Database(string connectionString, DbProviderFactory dbProviderFactory, IDataInstrumentationProvider instrumentationProvider)

{

if (string.IsNullOrEmpty(connectionString)) throw new ArgumentException(Resources.ExceptionNullOrEmptyString, "connectionString");

if (dbProviderFactory == null) throw new ArgumentNullException("dbProviderFactory");

if (instrumentationProvider == null) throw new ArgumentNullException("instrumentationProvider");

this.connectionString = new ConnectionString(connectionString, VALID\_USER\_ID\_TOKENS, VALID\_PASSWORD\_TOKENS);

this.dbProviderFactory = dbProviderFactory;

this.instrumentationProvider = instrumentationProvider;

}

/// <summary>

/// <para>Gets the string used to open a database.</para>

/// </summary>

/// <value>

/// <para>The string used to open a database.</para>

/// </value>

/// <seealso cref="DbConnection.ConnectionString"/>

public string ConnectionString

{

get { return connectionString.ToString(); }

}

/// <summary>

/// Adds a new In <see cref="DbParameter"/> object to the given <paramref name="command"/>.

/// </summary>

/// <param name="command">The command to add the parameter.</param>

/// <param name="name"><para>The name of the parameter.</para></param>

/// <param name="dbType"><para>One of the <see cref="DbType"/> values.</para></param>

/// <param name="size"><para>The maximum size of the data within the column.</para></param>

/// <param name="direction"><para>One of the <see cref="ParameterDirection"/> values.</para></param>

/// <param name="nullable"><para>Avalue indicating whether the parameter accepts <see langword="null"/> (<b>Nothing</b> in Visual Basic) values.</para></param>

/// <param name="precision"><para>The maximum number of digits used to represent the <paramref name="value"/>.</para></param>

/// <param name="scale"><para>The number of decimal places to which <paramref name="value"/> is resolved.</para></param>

/// <param name="sourceColumn"><para>The name of the source column mapped to the DataSet and used for loading or returning the <paramref name="value"/>.</para></param>

/// <param name="sourceVersion"><para>One of the <see cref="DataRowVersion"/> values.</para></param>

/// <param name="value"><para>The value of the parameter.</para></param>

public virtual void AddParameter(DbCommand command,

string name,

DbType dbType,

int size,

ParameterDirection direction,

bool nullable,

byte precision,

byte scale,

string sourceColumn,

DataRowVersion sourceVersion,

object value)

{

if (command == null) throw new ArgumentNullException("command");

DbParameter parameter = CreateParameter(name, dbType, size, direction, nullable, precision, scale, sourceColumn, sourceVersion, value);

command.Parameters.Add(parameter);

}

/// <summary>

/// Executes the query for <paramref name="command"/>.

/// </summary>

/// <param name="command">The <see cref="DbCommand"/> representing the query to execute.</param>

/// <returns>The quantity of rows affected.</returns>

protected int DoExecuteNonQuery(DbCommand command)

{

if (command == null) throw new ArgumentNullException("command");

try

{

DateTime startTime = DateTime.Now;

int rowsAffected = command.ExecuteNonQuery();

instrumentationProvider.FireCommandExecutedEvent(startTime);

return rowsAffected;

}

catch (Exception e)

{

instrumentationProvider.FireCommandFailedEvent(command.CommandText, ConnectionStringNoCredentials, e);

throw;

}

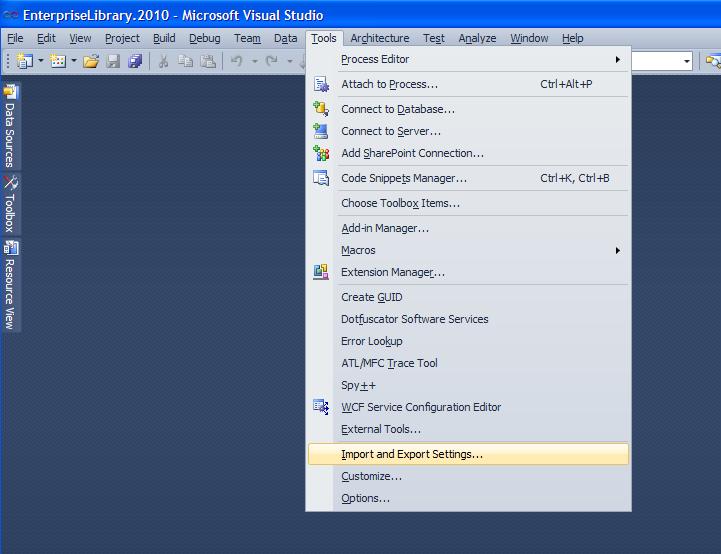
}

}

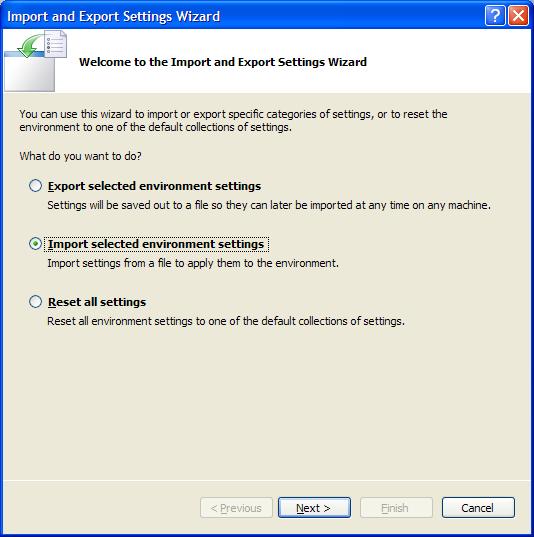
* + 1. Visual Studio格式化设置

导入VS setting

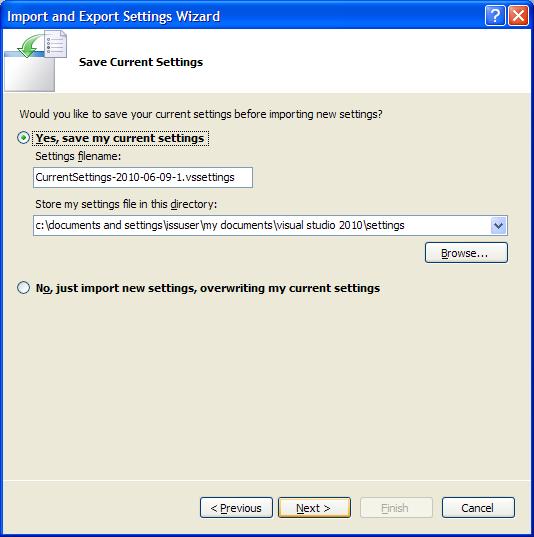
1. Tools->Import and Export Setting



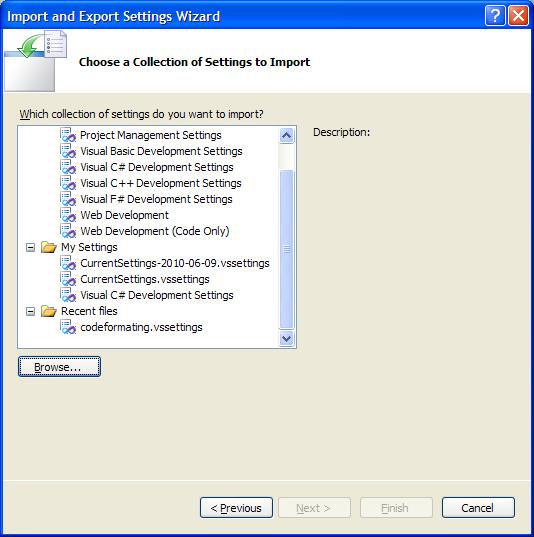
1. Select "Import selected ..."



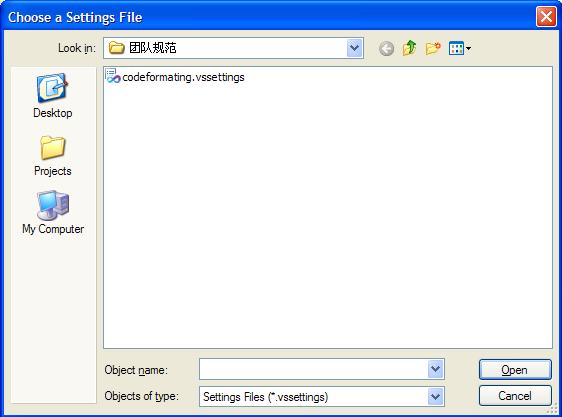
1. Save or not save current setting



1. select browse for "choose a collection..."

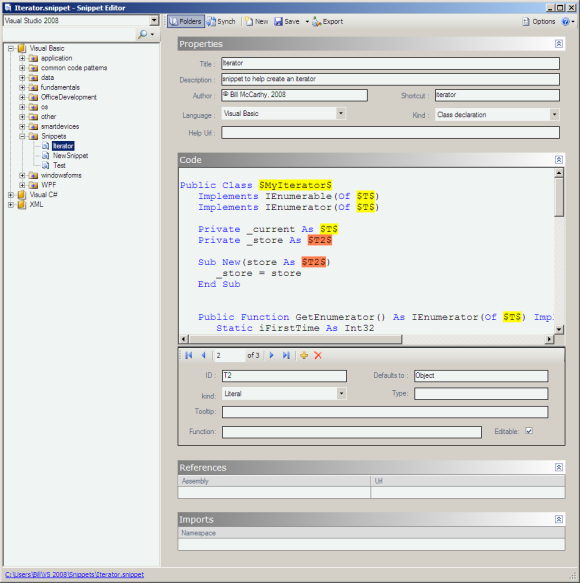


1. Browse…



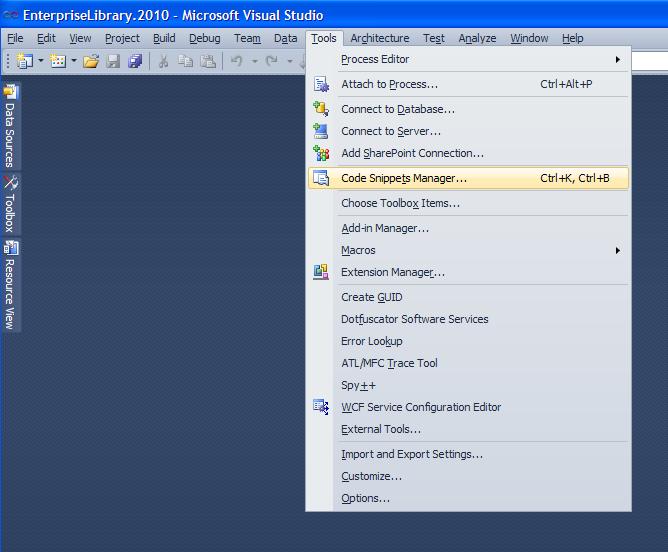
* + 1. 使用CodeSnippet

1. 使用[Snippet Editor](http://snippeteditor.codeplex.com/) 可视化工具编辑Snippet文件

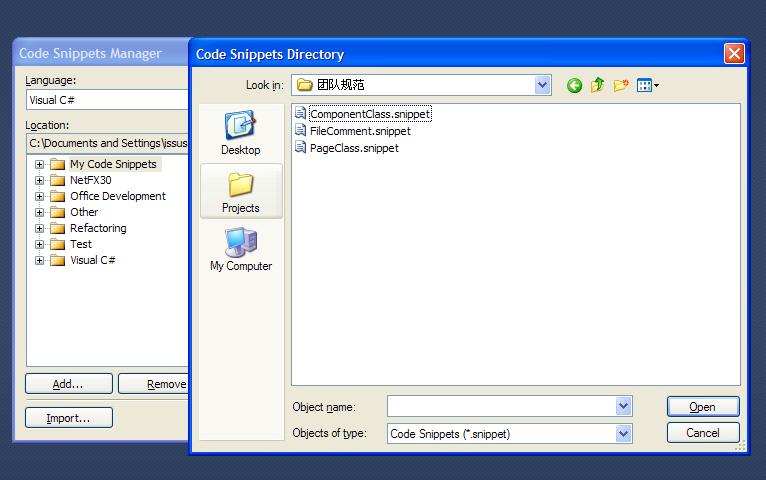


1. 导入Snippet文件

Tools->CodeSnippetManger

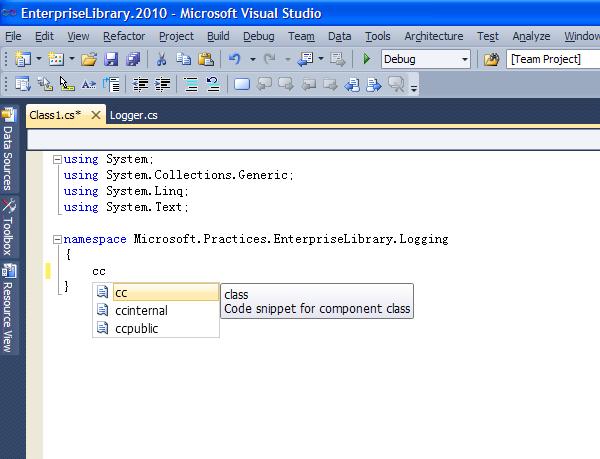


Import...

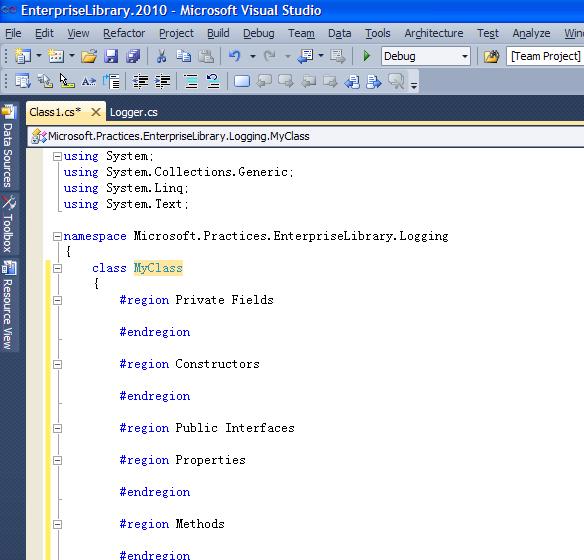


1. 使用CodeSnippet

在代码任何地方键入快捷键。



连续按Tab。



1. 附录
   1. 参考资料

**参考网站**

Team Foundation Source Control Walkthroughs

<http://msdn.microsoft.com/en-us/library/ms181279(VS.80).aspx>

Task (CMMI)

<http://msdn.microsoft.com/en-us/library/ee332492.aspx>

Visual Studio 的源代码管理

<http://msdn.microsoft.com/zh-cn/library/zxd4dfad(VS.80).aspx>

* 1. C#关键字

|  |  |  |  |
| --- | --- | --- | --- |
| [abstract](http://msdn.microsoft.com/zh-cn/library/sf985hc5(v=VS.80).aspx) | [event](http://msdn.microsoft.com/zh-cn/library/8627sbea(v=VS.80).aspx) | [new](http://msdn.microsoft.com/zh-cn/library/51y09td4(v=VS.80).aspx) | [struct](http://msdn.microsoft.com/zh-cn/library/ah19swz4(v=VS.80).aspx) |
| [as](http://msdn.microsoft.com/zh-cn/library/cscsdfbt(v=VS.80).aspx) | [explicit](http://msdn.microsoft.com/zh-cn/library/xhbhezf4(v=VS.80).aspx) | [null](http://msdn.microsoft.com/zh-cn/library/edakx9da(v=VS.80).aspx) | [switch](http://msdn.microsoft.com/zh-cn/library/06tc147t(v=VS.80).aspx) |
| [base](http://msdn.microsoft.com/zh-cn/library/hfw7t1ce(v=VS.80).aspx) | [extern](http://msdn.microsoft.com/zh-cn/library/e59b22c5(v=VS.80).aspx) | [object](http://msdn.microsoft.com/zh-cn/library/9kkx3h3c(v=VS.80).aspx) | [this](http://msdn.microsoft.com/zh-cn/library/dk1507sz(v=VS.80).aspx) |
| [bool](http://msdn.microsoft.com/zh-cn/library/c8f5xwh7(v=VS.80).aspx) | [false](http://msdn.microsoft.com/zh-cn/library/67bxt5ee(v=VS.80).aspx) | [operator](http://msdn.microsoft.com/zh-cn/library/s53ehcz3(v=VS.80).aspx) | [throw](http://msdn.microsoft.com/zh-cn/library/1ah5wsex(v=VS.80).aspx) |
| [break](http://msdn.microsoft.com/zh-cn/library/adbctzc4(v=VS.80).aspx) | [finally](http://msdn.microsoft.com/zh-cn/library/zwc8s4fz(v=VS.80).aspx) | [out](http://msdn.microsoft.com/zh-cn/library/t3c3bfhx(v=VS.80).aspx) | [true](http://msdn.microsoft.com/zh-cn/library/eahhcxk2(v=VS.80).aspx) |
| [byte](http://msdn.microsoft.com/zh-cn/library/5bdb6693(v=VS.80).aspx) | [fixed](http://msdn.microsoft.com/zh-cn/library/f58wzh21(v=VS.80).aspx) | [override](http://msdn.microsoft.com/zh-cn/library/ebca9ah3(v=VS.80).aspx) | [try](http://msdn.microsoft.com/zh-cn/library/0yd65esw(v=VS.80).aspx) |
| [case](http://msdn.microsoft.com/zh-cn/library/06tc147t(v=VS.80).aspx) | [float](http://msdn.microsoft.com/zh-cn/library/b1e65aza(v=VS.80).aspx) | [params](http://msdn.microsoft.com/zh-cn/library/w5zay9db(v=VS.80).aspx) | [typeof](http://msdn.microsoft.com/zh-cn/library/58918ffs(v=VS.80).aspx) |
| [catch](http://msdn.microsoft.com/zh-cn/library/0yd65esw(v=VS.80).aspx) | [for](http://msdn.microsoft.com/zh-cn/library/ch45axte(v=VS.80).aspx) | [private](http://msdn.microsoft.com/zh-cn/library/st6sy9xe(v=VS.80).aspx) | [uint](http://msdn.microsoft.com/zh-cn/library/x0sksh43(v=VS.80).aspx) |
| [char](http://msdn.microsoft.com/zh-cn/library/x9h8tsay(v=VS.80).aspx) | [foreach](http://msdn.microsoft.com/zh-cn/library/ttw7t8t6(v=VS.80).aspx) | [protected](http://msdn.microsoft.com/zh-cn/library/bcd5672a(v=VS.80).aspx) | [ulong](http://msdn.microsoft.com/zh-cn/library/t98873t4(v=VS.80).aspx) |
| [checked](http://msdn.microsoft.com/zh-cn/library/74b4xzyw(v=VS.80).aspx) | [goto](http://msdn.microsoft.com/zh-cn/library/13940fs2(v=VS.80).aspx) | [public](http://msdn.microsoft.com/zh-cn/library/yzh058ae(v=VS.80).aspx) | [unchecked](http://msdn.microsoft.com/zh-cn/library/a569z7k8(v=VS.80).aspx) |
| [class](http://msdn.microsoft.com/zh-cn/library/0b0thckt(v=VS.80).aspx) | [if](http://msdn.microsoft.com/zh-cn/library/5011f09h(v=VS.80).aspx) | [readonly](http://msdn.microsoft.com/zh-cn/library/acdd6hb7(v=VS.80).aspx) | [unsafe](http://msdn.microsoft.com/zh-cn/library/chfa2zb8(v=VS.80).aspx) |
| [const](http://msdn.microsoft.com/zh-cn/library/e6w8fe1b(v=VS.80).aspx) | [implicit](http://msdn.microsoft.com/zh-cn/library/z5z9kes2(v=VS.80).aspx) | [ref](http://msdn.microsoft.com/zh-cn/library/14akc2c7(v=VS.80).aspx) | [ushort](http://msdn.microsoft.com/zh-cn/library/cbf1574z(v=VS.80).aspx) |
| [continue](http://msdn.microsoft.com/zh-cn/library/923ahwt1(v=VS.80).aspx) | [in](http://msdn.microsoft.com/zh-cn/library/ttw7t8t6(v=VS.80).aspx) | [return](http://msdn.microsoft.com/zh-cn/library/1h3swy84(v=VS.80).aspx) | [using](http://msdn.microsoft.com/zh-cn/library/zhdeatwt(v=VS.80).aspx) |
| [decimal](http://msdn.microsoft.com/zh-cn/library/364x0z75(v=VS.80).aspx) | [int](http://msdn.microsoft.com/zh-cn/library/5kzh1b5w(v=VS.80).aspx) | [sbyte](http://msdn.microsoft.com/zh-cn/library/d86he86x(v=VS.80).aspx) | [virtual](http://msdn.microsoft.com/zh-cn/library/9fkccyh4(v=VS.80).aspx) |
| [default](http://msdn.microsoft.com/zh-cn/library/25tdedf5(v=VS.80).aspx) | [interface](http://msdn.microsoft.com/zh-cn/library/87d83y5b(v=VS.80).aspx) | [sealed](http://msdn.microsoft.com/zh-cn/library/88c54tsw(v=VS.80).aspx) | [volatile](http://msdn.microsoft.com/zh-cn/library/x13ttww7(v=VS.80).aspx) |
| [delegate](http://msdn.microsoft.com/zh-cn/library/900fyy8e(v=VS.80).aspx) | [internal](http://msdn.microsoft.com/zh-cn/library/7c5ka91b(v=VS.80).aspx) | [short](http://msdn.microsoft.com/zh-cn/library/ybs77ex4(v=VS.80).aspx) | [void](http://msdn.microsoft.com/zh-cn/library/yah0tteb(v=VS.80).aspx) |
| [do](http://msdn.microsoft.com/zh-cn/library/370s1zax(v=VS.80).aspx) | [is](http://msdn.microsoft.com/zh-cn/library/scekt9xw(v=VS.80).aspx) | [sizeof](http://msdn.microsoft.com/zh-cn/library/eahchzkf(v=VS.80).aspx) | [while](http://msdn.microsoft.com/zh-cn/library/2aeyhxcd(v=VS.80).aspx) |
| [double](http://msdn.microsoft.com/zh-cn/library/678hzkk9(v=VS.80).aspx) | [lock](http://msdn.microsoft.com/zh-cn/library/c5kehkcz(v=VS.80).aspx) | [stackalloc](http://msdn.microsoft.com/zh-cn/library/cx9s2sy4(v=VS.80).aspx) |  |
| [else](http://msdn.microsoft.com/zh-cn/library/5011f09h(v=VS.80).aspx) | [long](http://msdn.microsoft.com/zh-cn/library/ctetwysk(v=VS.80).aspx) | [static](http://msdn.microsoft.com/zh-cn/library/98f28cdx(v=VS.80).aspx) |  |
| [enum](http://msdn.microsoft.com/zh-cn/library/sbbt4032(v=VS.80).aspx) | [namespace](http://msdn.microsoft.com/zh-cn/library/z2kcy19k(v=VS.80).aspx) | [string](http://msdn.microsoft.com/zh-cn/library/362314fe(v=VS.80).aspx) |  |

上下文关键字

|  |  |  |
| --- | --- | --- |
| [get](http://msdn.microsoft.com/zh-cn/library/ms228503(v=VS.80).aspx) | [partial](http://msdn.microsoft.com/zh-cn/library/wbx7zzdd(v=VS.80).aspx) | [set](http://msdn.microsoft.com/zh-cn/library/ms228368(v=VS.80).aspx) |
| [value](http://msdn.microsoft.com/zh-cn/library/a1khb4f8(v=VS.80).aspx) | [where](http://msdn.microsoft.com/zh-cn/library/6b0scde8(v=VS.80).aspx) | [yield](http://msdn.microsoft.com/zh-cn/library/9k7k7cf0(v=VS.80).aspx) |

* 1. C#文档注释标记

|  |  |  |
| --- | --- | --- |
| 标签 | 用法 | 作用 |
| <c> | <c>text</c>  text希望将其指示为代码的文本。 | 为您提供了一种将说明中的文本标记为代码的方法。使用 <code>将多行指示为代码 |
| <para> | <para>content</para>  content段落文本。 | 用于诸如 <remarks> 或 <returns> 等标记内，使您得以将结构添加到文本中。 |
| <param> | <param name='name'>description</param>  name 为方法参数名。将此名称用单引号括起来 (' ')。 | 应当用于方法声明的注释中，以描述方法的一个参数。 |
| <paramref> | <paramref name="name"/>  name要引用的参数名。将此名称用双引号括起来 (" ")。 | <paramref> 标记为您提供了一种指示词为参数的方法。可以处理 XML 文件，从而用某种独特的方法格式化该参数。 |
| <see> | <see cref="member"/>  cref = "member" 对可以通过当前编译环境进行调用的成员或字段的引用。编译器检查到给定代码元素存在后，将 member传递给输出 XML 中的元素名。必须将 member括在双引号 (" ") 中。 | 使您得以从文本内指定链接。使用 <seealso>指示希望在“请参阅”一节中出现的文本。 |
| <seealso> | <seealso cref="member"/>  cref = "member" 对可以通过当前编译环境进行调用的成员或字段的引用。编译器检查到给定代码元素存在后，将 member传递给输出 XML 中的元素名。必须将 member括在双引号 (" ") 中 | <seealso> 标记使您得以指定希望在“请参见”一节中出现的文本。使用 <see> 从文本内指定链接。 |
| <example> | <example>description</example>  description  代码示例的说明。 | 使用 <example> 标记可以指定使用方法或其他库成员的示例。一般情况下，这将涉及到 <code>标记的使用。 |
| <code> | <code>content</code>  content 为希望将其标记为代码的文本。 | <code> 标记为您提供了一种将多行指示为代码的方法。使用 <c> 指示应将说明中的文本标记为代码。 |
| <summary> | <summary>description</summary>  此处description 为对象的摘要。 | <summary> 标记应当用于描述类型或类型成员。使用 <remarks> 添加针对某个类型说明的补充信息。  <summary> 标记的文本是唯一有关 IntelliSense 中的类型的信息源，它也显示在对象浏览器中。 |
| <exception> | cref = "member"  对可从当前编译环境中获取的异常的引用。编译器检查到给定异常存在后，将 member 转换为输出 XML 中的规范化元素名。必须将 member 括在双引号 (" ") 中。  有关如何创建对泛型类型的 cref 引用的更多信息，请参见 <see>（C# 编程指南）。  description  异常的说明。 | <exception> 标记使您可以指定哪些异常可被引发。此标记可用在方法、属性、事件和索引器的定义中。 |
| <include> | filename  包含文档的文件名。该文件名可用路径加以限定。将 filename 括在单引号 (' ') 中。  tagpath  filename 中指向标记 name 的标记路径。将此路径括在单引号中 (' ')。  name  注释前边的标记中的名称说明符；name 具有一个 id。  id  位于注释之前的标记的 ID。将此 ID 括在双引号中 (" ")。 | <include> 标记使您得以引用描述源代码中类型和成员的另一文件中的注释。这是除了将文档注释直接置于源代码文件中之外的另一种可选方法。  <include> 标记使用 XML XPath 语法。有关自定义 <include> 使用的方法，请参见 XPath 文档。 |
| <list> | term  要定义的项，该项将在 description 中定义。  description  项目符号列表或编号列表中的项或者 term 的定义。 | <listheader> 块用于定义表或定义列表中的标题行。定义表时，只需为标题中的项提供一个项。  列表中的每一项都用一个 <item> 块来描述。创建定义列表时，既需要指定 term 也需要指定 description。但是，对于表、项目符号列表或编号列表，只需为 description 提供一个项。  列表或表所拥有的 <item> 块数可以根据需要而定。 |
| <permission> | cref = "member"  对可以通过当前编译环境进行调用的成员或字段的引用。编译器检查到给定代码元素存在后，将 member 转换为输出 XML 中的规范化元素名。必须将 member 括在双引号 (" ") 中。  有关如何创建对泛型类型的 cref 引用的信息，请参见 <see>（C# 编程指南）。  description  对成员的访问的说明。 | <permission> 标记使您得以将成员的访问记入文档。使用 PermissionSet 类可以指定对成员的访问。 |
| <remarks> | Description  成员的说明。 | <remarks> 标记用于添加有关某个类型的信息，从而补充由 <summary> 所指定的信息。此信息显示在对象浏览器中。 |
| <returns> | description  返回值的说明。 | <returns> 标记应当用于方法声明的注释，以描述返回值。 |
| <value> | property-description  属性的说明。 | <value> 标记使您得以描述属性所代表的值。请注意，当在 Visual Studio .NET 开发环境中通过代码向导添加属性时，它将会为新属性添加 <summary> 标记。然后，应该手动添加 <value> 标记以描述该属性所表示的值。 |
| <typeparamref> | name  类型参数的名称。将此名称用双引号括起来 (" ")。 | 使用此标记，文档文件的使用者能够以某种独特的方式设置单词的格式，例如以斜体显示。 |
| <typeparam> | name  类型参数的名称。将此名称用双引号括起来 (" ")。 | 在泛型类型或方法声明的注释中应该使用 <typeparam> 标记描述类型参数。为泛型类型或方法的每个类型参数添加标记。  有关更多信息，请参见泛型（C# 编程指南）。  <typeparam> 标记的文本将显示在对象浏览器代码注释 Web 报表 IntelliSense 中。 |
|  | | |