**C# 命名規範**

 C#語言開發規範

 作者ching

1.  命名規範

a) 類

【規則1-1】使用Pascal規則命名類名，即首字母要大寫。

eg:

Class Test

{

    ...

}

【規則1-2】使用能夠反映類功能的名詞或名詞短語命名類。

【規則1-3】不要使用“I”、“C”、“\_”等特定含義首碼。

【規則1-4】自訂異常類應以Exception結尾。

eg:

Class TestException

{

    ...

}

【規則1-5】檔案名要能反映類的內容，最好是和類同名。

b) 類欄位(類成員)

【規則2-1】用camel規則來命名類成員變數名稱，即首單詞（或單詞縮寫）小寫。

【規則2-2】類欄位變數名前可加“\_”首碼。

【規則2-3】堅決禁止在普通變數前加“m\_”（這是VC老命名規則）。

eg:

Class Test

{

    privatestring myName;

    privatestring \_myCoCo;

    ...

}

c）方法

   【規則3-1】方法名採用Pascal規則，第一個字元要大寫。

【規則3-2】方法名應使用動詞或動詞短語。

【規則3-3】類中訪問修飾符或功能相同的方法應該放在一起， 且公共或實現介面的方法在前。

eg:

Class Test

{

      ...

      publicvoid GetData(...)

      {

         ...

      }

      privatevoid GetName(...)

      {

         ...

      }

      privatestatic void GetCount(...)

      {

         ...

      }

}

   d）屬性

      【規則4-1】使用名詞定義屬性，屬性使用Pascal規則，首字元大寫

      【規則4-2】屬性和相應欄位名稱要關聯， 可以使用“重構”功能表來生成屬性。

eg:

Class Test

{

   private string myName;

   public string MyName

   {

      set

      {

         myName = Value;

      }

      get

      {

         return myName;

      }

   }

}

   e）參數

      【規則5-1】參數採用camel規則命名，且首字元小寫。

      【規則5-2】使用描述性參數名稱，參數名稱應當具有最夠的說明性。

      【規則5-3】不要給參數加匈牙利語類型標記法的首碼。

      【規則5-4】檢查方法所有輸入參數的有效性。

      eg:

      ClassTest

      {

         public void GetData(string name,string strFlag)

         {

            ...

         }

      }

   f）常量

      【規則6-1】唯讀常量使用Pascal命名規則，即首字母大寫。

   【規則6-2】枚舉名使用Pascal規則命名，枚舉成員本質屬於常量，命名規則同上。

      【規則6-3】枚舉值從小到大順序定義。

      【規則6-4】靜態欄位或屬性採用Pascal規則，即首字元大寫。

      eg:

      ClassTest

      {

         public const double Pi = 3.14159365753;

         private readonly double Pai = 3.14159365753;

      }

   g）介面

      【規則7-1】介面定義使用Pascal規則，且必須以大寫“I”開頭。

      【規則7-2】介面名稱要有意義，中間不要有底線“\_”等字元。

   【規則7-3】如果類實現了介面，名稱儘量和介面相同， 只是省掉“I”字元。

   eg:注意這裡的介面修飾符只能用public和internal

   interfaceITest

   {

      ...

   }

   h）事件

      【規則8-1】委託名稱採用Pascal規則，即首字元大寫。

   【規則8-2】定義事件的委託要使用EventHandler尾碼，且包括sender和e兩個參數。

   【規則8-3】事件用到的參數，名稱要帶EventArgs尾碼。

   eg:

   ClassTest

   {

      private delegate void DoTask();

      private event DoTask DoTaskEventHandler;

      private event DoTask DoTaskEventHandler(Object sender,EventArgse);

      private event DoTaskDoTaskEventHandler2(string   strEventArgs)

      {

         ...

      }

      ...

   }

i) 命名空間

【規則9-1】命名空間名稱採用Pascal規則，且首字元大寫。

【規則9-2】命名空間名稱儘量反映其內容所提供的整體功能。

eg:

namespace MyTest

{

      ClassTest

    {

       ...

    }

    ...

}

2.  注釋規範

a）文件頭部注釋

【規則1-1】檔都包含檔頭， 要說明檔案名、作者、創建時間、變更記錄。

      【規則1-2】推薦採用.NET形式書寫頭部注釋。(待考查)

   b）類及其成員注釋

      【規則2-1】對方法和類使用“///”三斜線注釋。

   【規則2-2】代碼行文注釋採用“//”和“/\*\*/”進行，應該儘量說明問題。

3.行文規範

   a）縮寫規範

      【規則1-1】識別字應當直觀可望文知意，不提倡使用任何縮寫。

   【規則1-2】字串變數推薦是用“str”或“s”開頭，採用string.Empty來初始化。

   【規則1-3】普通物件可以以“obj”開頭。

   【規則1-4】縮寫可自行定義，一般取單詞的前/後字元組成，以含義直觀為準則。

   【規則1-5】一般情況下不要讓縮寫破壞識別字的含義。

b）排版

   【規則2-1】每行語句至少占一行，如果語句過長（超過一屏），則該語句斷為兩行顯示。

   【規則2-2】把相似的內容放在一起，比如欄位、屬性、方法、事件等，使用“#region--#endregion”命令分組。

   【規則2-3】多個程式元素進行對等操作時， 操作符之前、之後或者前後都要加空格。

   【規則2-4】每個方法的來源程式行數原則上應該少於200行。(如果超過過多，則需要另寫一方法)

   【規則2-5】語句嵌套層次不得超過3層。

   【規則2-6】避免相同的程式碼片段在多個地方出現。(儘量避免代碼重複，能複用則複用)

c）語句結構

   【規則3-1】如果使用了異常結構，一定要處理異常， 一般是要寫日誌檔。

   【規則3-2】分支語句不應該使用複雜長條件， 應該將長條件封裝成方法。

   【規則3-3】switch語句，case後面必須接break。

   【規則3-4】禁止使用goto語句進行跳轉。

   【規則3-5】行文中嚴禁出現“魔數”，特定含義的常數必須定義成枚舉或常量。

   【規則3-6】不同類型的操作符混合使用時，使用括弧給出優先順序。

   【規則3-7】不允許使用複雜的操作符組合等。

   【規則3-8】迴圈、判斷語句的區塊部分用花括弧括起來， 即使只有一條語句。(return;待議)

   【規則3-9】在switch語句中總是要有default字句，建議使用斷言。

   【規則3-10】每個類和方法完成單一的功能，不設計多用途面面俱到的類或方法。

   【規則3-11】嚴禁使用未經初始化的變數，變數通常使用構造方法來初始。

d）代碼縮進

   【規則4-1】碰到大括弧要換行。

   【規則4-2】不允許使用Java中的括弧換行規範。

e）大小寫

   【規則5-1】不要創建名稱相同，但大小寫區別的任何元素。

   【規則5-2】應當大寫僅有兩個字元的縮寫。(如果只有兩個字元，則都大寫)

   【規則5-3】不要把易混淆的數位和字元放在一起。

   【規則5-4】使用英文命名識別字。

f）重名規範

   【規則6-1】不允許變數名、類名、屬性名、 方法名等與系統識別字重名。（系統識別字見附表）

g）SQL編碼規範

   【規則7-1】SQL語句全部大寫。(本人的習慣是小寫，為了性能，最好是大寫，我自己也得把習慣改改咯，哈哈哈)

   【規則7-2】對較為複雜的SQL語句加上注釋，說明其功能。

   【規則7-3】連接子OR、IN、AND、以及=、<=、>=等前後加空格。

   【規則7-4】使用明確的列代替 SELECT \*。

h）軟體架構

   【規則8-1】資料庫中每一張表對應一個實體類/資料傳輸物件（DTO）。

   【規則8-2】實體類名稱使用表名，也可帶有Dto尾碼。

   【規則8-3】三層架構應當合理使用，不應生搬硬套。

   【規則8-4】三層架構元素推薦使用尾碼：

      資料傳輸對象         XxxxDto

      DAO工廠            XxxDAOFactory

      DAO介面            IXxxxDAO

      服務介面           IxxxxService

      DAO的資料庫實現     XxxxDAOOracle/XxxxDAOInfomix

      業務邏輯           XxxxManager

i）系統

   【規則9-1】在我國內不建議隨便使用設計模式等代碼模式，因為並不流行。

   【規則9-2】系統輸入、資源操作（如記憶體分配、檔及目錄操作）、網路操作（如通信、調用等）、任務間的操作（如通信、調用等）時必須進行錯誤、超時、或則異常處理。

   【規則9-3】模組的編寫應有完善的測試方面的考慮。

附表

表1 各種類型命名規範總結

|  |  |  |  |
| --- | --- | --- | --- |
| 類型 | 命名規則 | 注意事項 | 實例 |
| 類或結構 | Pascal | 首字元大寫 | HttpContext |
| 介面 | Pascal | 加首碼I | IDataAdaper |
| 枚舉名 | Pascal | 首字元大寫 | CommandType |
| 枚舉值 | Pascal | 首字元大寫 | CommandType.Text |
| 事件 | Pascal | 首字元大寫 | SelectedIndexChanged |
| 自訂異常 | Pascal | 加尾碼Exception | ArgumentException |
| 類公共欄位 | Pascal | 首字元大寫 | MaxValue(或\_MaxValue) |
| 方法 | Pascal | 首字元大寫 | ToString() |
| 命名空間 | Pascal | 首字元大寫 | System.Xml |
| 屬性 | Pascal | 首字元大寫 | BackColor |
| 保護或私有欄位 | Camel | 首字元小寫 | myVariable |
| 參數 | Camel | 首字元小寫 | cmdText |

表2 資料類型縮寫規則

|  |  |  |
| --- | --- | --- |
| 資料類型 | 資料類型縮寫 | 標準命名實例 |
| **Bool** | b/is | IsVisable |
| **Float** | F | FPrice |
| **Double** | D | DPrice |
| **Unit** | U | UAge |
| **Int** | I | INumber |
| **Char** | Ch | ChCode |
| **Byte** | Bt | BtImages |
| **String** | Str | StrName |
| **Struct** | St | StStudent |
| **Window** | Wnd | WndMain |
| **ArrayList** | Lst | LstStudents |
| **Array/strong>** | Arr | ArrStudents |
| **Hashtable** | Ht | Htstudents |

表3 Windows控制項縮寫規則

|  |  |  |  |
| --- | --- | --- | --- |
| 控制項類型 | 控制項名稱 | 控制項類型縮寫 | 實例 |
| **Label** | 標籤框 | Lbl | LblMessage |
| **LinkLabel** | 超連結標籤框 | Llbl | LlblToday |
| **Button** | 按鈕 | Btn | BtnSave |
| **TextBox** | 文字方塊 | Txt | TxtName |
| **MainMenu** | 功能表列 | Mmnu | MmnuFile |
| **CheckBox** | 多選框 | Chk | ChkStock |
| **RadioButton** | 單選框 | Rbtn | RbtnSelected |
| **GroupBox** | 下拉式列示方塊 | Gbx | GbxMain |
| **PictureBox** | 圖片框 | Pic | PicImage |
| **Panel** |  | Pnl | PnlBody |
| **DataGrid** |  | Dgrd | DgrdView |
| **ListBox** |  | Lst | LstProducts |
| **CheckedListBox** |  | Clst | ClstChecked |
| **ComboBox** | 下拉式列示方塊 | Cbo | CboMenu |
| **ListView** | 列表視圖 | Lvw | LvwBrowser |
| **TreeView** | 樹狀檢視 | Tvw | TvwType |
| **TabControl** |  | Tctl | TctlSelected |
| **DateTimePicker** |  | Dtp | DtpStartDate |
| **HscrollBar** |  | Hsb | HsbImage |
| **VscrollBar** |  | Vsb | VsbImage |
| **Timer** |  | Tmr | TmrCount |
| **ImageList** |  | Ilst | IlstImage |
| **ToolBar** | 工具列 | Tlb | TlbManage |
| **StatusBar** | 狀態列 | Stb | StbFootPrint |
| **OpenFileDialog** |  | Odlg | OdlgFile |
| **SaveFileDialog** |  | Sdlg | SdlgSave |
| **FoldBrowserDialog** |  | Fbdlg | FbdlgBrowser |
| **FontDialog** |  | Fdlg | FdlgFoot |
| **ColorDialog** |  | Cdlg | CdlgColor |
| **PrintDialog** |  | Pdlg | PdlgPrint |

表 4 資料庫物件縮寫規範

|  |  |  |  |
| --- | --- | --- | --- |
| 資料庫物件 | 名稱 | 簡寫 | 實例 |
| **Connection** |  | Con | ConNorthwind |
| **Command** |  | Cmd | CmdReturnProducts |
| **Parameter** |  | Parm | ParmProductID |
| **DataAdapter** |  | Dap | DapProducts |
| **DataReader** |  | Dtr | DtrProducts |
| **DataSet** |  | Ds | DsNorthwind |
| **DataTable** |  | Dt | DtProduct |
| **DataRow** |  | Drow | DrowRow |
| **DataColumn** |  | Dcol | DcolProductID |
| **DataRelation** |  | Drl | DrlMasterDetail |
| **DataView** |  | Dvw | DvwFilteredProducts |

大家都知道寫程式應該有個好的命名規範，為了工作方便，貼出來。  
**1 ADO.NET 命名規範**

|  |  |  |
| --- | --- | --- |
| 資料類型 | 資料類型簡寫 | 標準命名舉例 |
| 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 |

**2 WinForm Control 命名規範**

|  |  |  |
| --- | --- | --- |
| 資料類型 | 資料類型簡寫 | **標準命名舉例** |
| Label | lbl | lblMessage |
| LinkLabel | llbl | llblToday |
| Button | btn | btnSave |
| TextBox | txt | txtName |
| MainMenu | mmnu | mmnuFile |
| CheckBox | chk | chkStock |
| RadioButton | rbtn | rbtnSelected |
| GroupBox | gbx | gbxMain |
| PictureBox | pic | picImage |
| Panel | pnl | pnlBody |
| DataGrid | dgrd | dgrdView |
| ListBox | lst | lstProducts |
| CheckedListBox | clst | clstChecked |
| ComboBox | cbo | cboMenu |
| ListView | lvw | lvwBrowser |
| TreeView | tvw | tvwType |
| TabControl | tctl | tctlSelected |
| DateTimePicker | dtp | dtpStartDate |
| HscrollBar | hsb | hsbImage |
| VscrollBar | vsb | vsbImage |
| Timer | tmr | tmrCount |
| ImageList | ilst | ilstImage |
| ToolBar | tlb | tlbManage |
| StatusBar | stb | stbFootPrint |
| OpenFileDialog | odlg | odlgFile |
| SaveFileDialog | sdlg | sdlgSave |
| FoldBrowserDialog | fbdlg | fgdlgBrowser |
| FontDialog | fdlg | fdlgFoot |
| ColorDialog | cdlg | cdlgColor |
| PrintDialog | pdlg | pdlgPrint |

**3 WebControl 命名規範**

|  |  |  |
| --- | --- | --- |
| 資料類型 | 資料類型簡寫 | 標準命名舉例 |
| AdRotator | adrt | Example |
| Button | btn | btnSubmit |
| Calendar | cal | calMettingDates |
| CheckBox | chk | chkBlue |
| CheckBoxList | chkl | chklFavColors |
| CompareValidator | valc | valcValidAge |
| CustomValidator | valx | valxDBCheck |
| DataGrid | dgrd | dgrdTitles |
| DataList | dlst | dlstTitles |
| DropDownList | drop | dropCountries |
| HyperLink | lnk | lnkDetails |
| Image | img | imgAuntBetty |
| ImageButton | ibtn | ibtnSubmit |
| Label | lbl | lblResults |
| LinkButton | lbtn | lbtnSubmit |
| ListBox | lst | lstCountries |
| Panel | pnl | pnlForm2 |
| PlaceHolder | plh | plhFormContents |
| RadioButton | rad | radFemale |
| RadioButtonList | radl | radlGender |
| RangeValidator | valg | valgAge |
| RegularExpression | vale | valeEmail\_Validator |
| Repeater | rpt | rptQueryResults |
| RequiredFieldValidator | valr | valrFirstName |
| Table | tbl | tblCountryCodes |
| TableCell | tblc | tblcGermany |
| TableRow | tblr | tblrCountry |
| TextBox | txt | txtFirstName |
| ValidationSummary | vals | valsFormErrors |
| XML | xmlc | xmlcTransformResults |

Array             arr           arrShoppingList   
Boolean         bln           blnIsPostBack   
Byte               byt           bytPixelValue   
Char                     chr           chrDelimiter   
DateTime       dtm         dtmStartDate   
Decimal         dec         decAverageHeight   
Double           dbl           dblSizeofUniverse   
Integer           int           intRowCounter   
Long               lng           lngBillGatesIncome   
Object           obj           objReturnValue   
Short           shr     shrAverage   
Single           sng         sngMaximum   
String                 str         strFirstName

### The End

# [C#命名規則](http://www.cnblogs.com/Yellow0-0River/p/4240654.html)

# 1.     命名規則和風格 ****Naming Conventions and Style****

1.        類和方法名採用Pascal風格  
    Use Pascal casing for typeand method names   
    public class SomeClass   
    {   
       public SomeMethod(){}   
    }

2.        區域變數和方法參數採用camel風格  
    Use camel casing for local variablenames and method arguments   
    int number;   
    void MyMethod(int someNumber)   
    {}

3.        介面名採用I作為首碼  
    Prefix interface name with I    
    interface IMyInterface   
    {..}

4.        私有成員變數採用m\_作為首碼  
    Prefix private member variables withm\_   
    public class SomeClass   
    {   
       private int m\_Number;   
    }

5.        自訂屬性類名採用Attribute作為尾碼  
    Suffix custom attribute classes withAttribute.

6.        自訂異常類名採用Exception作為尾碼  
    Suffix custom exception classes withException.

7.        採用動詞-物件對命名方法，例如ShowDialog()  
    Name methods using verb-object pair, such as ShowDialog()

8.        有返回值的方法應該取名表示其返回值，例如GetObjectState()  
    Methods with return values should have a name describing thevalue returned, such as GetObjectState().

9.        採用描述性的變數名。  
    Use descriptive variable names.   
    a) 避免採用單字母的變數名，如i或t；而是採用index或temp。  
       Avoid  single character variablenames, such as i or  t. Use  index or temp instead.   
    b) 對public和protected成員避免採用用匈牙利命名法。  
       Avoid using Hungarian notation for publicor protected members.    
    c) 不要採用縮寫（例如將number縮寫為num）。  
       Do not abbreviate words (such as numinstead of number).

10.    總是使用C#預定義的類型，而不是使用System命名空間中的別名。例如：採用object不用Object，採用string不用String,採用int不用Int32。  
     Always use C# predefined types rather than the aliasesin the  System namespace.  
     For example:    
     object NOT Object   
     string NOT String   
     int    NOT Int32

11.    對於泛型，類型採用大寫字母。當處理.NET類型Type時保留尾碼Type。  
     With generics,  use capital letters for types.Reserve suffixing Type when dealing with the .NET type Type.  
     // 正確：  
     //Correct:   
     public class LinkedList   
     // 避免使用：  
     //Avoid:    
     public class LinkedList

12.    採用有意義的命名空間名，例如產品名稱或公司名稱。  
     Use meaningful namespaces suchas the product name or the company name

13.    避免使用類的全稱，而是採用using語句。  
     Avoid fully qualified typenames. Use the using statement instead.

14.    避免在命名空間內使用using語句。  
     Avoid putting a usingstatement inside a namespace.

15.    將所有framework命名空間名放在一起，後面放自訂或協力廠商的命名空間名。  
     Group all framework namespacestogether and put custom or third party namespaces underneath.  
     using System;   
     using System.Collections;   
     using System.ComponentModel;   
     using System.Data;   
     using MyCompany;   
     using MyControls;

16.    注釋縮進和其注釋的代碼在同一層次。  
       Indent comment at the same level ofindentation as the code you are documenting.

17.    所有注釋要經過拼寫檢查。拼寫錯誤的注釋表明開發的草率。  
      All comments should pass spell checking.Misspelled comments indicate sloppy development.

18.    所有成員變數應該定義在前面，和屬性或方法間空開一行。  
      All member variablesshould be declared at the top, with one line separating them from theproperties or methods.   
     public class MyClass   
     {   
        int m\_Number;   
        string m\_Name;

       public void SomeMethod1()   
        {}   
        public void SomeMethod2()   
        {}   
     }

19.    區域變數的定義盡可能靠近它的初次使用。  
       Declare a localvariable as close as possible to its first use.

20.    檔案名應該體現其包含的類。  
       A file name shouldreflect the class it contains.

21.    當使用partial類型且每部分分配一個檔時，以類型名加P和序數命名每個檔。  
       When using partial types and allocating apart per file, name each file after the type suffixed with a P and anordinal number:  
     //In MyClassP1.cs   
     public partial class MyClass   
     {}  
     //In MyClassP2.cs   
     public partial class MyClass   
     {}

22.    左大括弧總是放在新行中。  
       Always place anopen curly brace ({) in a new line.

23.    匿名方法模仿普通方法的佈局，和匿名委託定義放在一行。  
       With anonymous methods mimic the codelayout of a regular method, aligned with the anonymous delegatedeclaration.   
     a) 遵守將左大括弧放在新行的規則。  
         Comply with placing an opencurly brace in a new line   
     delegate void SomeDelegate(string someString);   
     //正確  
     //Correct:    
     public void InvokeMethod()   
     {   
        SomeDelegate someDelegate =delegate(string name)   
                                   {   
                                      MessageBox.Show(name);   
                                   };   
        someDelegate("Juval");   
     }   
     //避免採用：  
     //Avoid   
     public void InvokeMethod()   
     {   
        SomeDelegate someDelegate =delegate(string name){MessageBox.Show(name);};  
        someDelegate("Juval");   
     }

24.    沒有參數的匿名方法使用空括弧。  
       Use empty parenthesis on parameter-lessanonymous methods  
     a) 僅當匿名方法可能被用於任何委託時省略括弧。  
         Omit the parenthesis only ifthe anonymous method could have been used on any delegate.  
     delegate void SomeDelegate();   
     //Correct    
     SomeDelegate someDelegate1 = delegate()   
                                 {   
                                    MessageBox.Show("Hello");   
                                 };   
     //Avoid    
     SomeDelegate someDelegate1 = delegate   
                                 {   
                                    MessageBox.Show("Hello");   
                                 };

# 2.     編碼慣例   Coding Practices

1.        避免在一個檔中放多個類。  
    Avoid putting multiple classes in asingle file.

2.        一個檔應該只對一個命名空間提供類型。避免在同一檔中有多個命名空間。  
    A single file should only contribute types to a singlenamespace. Avoid having multiple namespaces in the same file.

3.        避免檔長度超過500行(除了機器自動產生的代碼)。  
    Avoid files with more than 500 lines(excluding machine-generated code).

4.        避免方法定義超過25行。  
    Avoid methods with more than 25lines.

5.        避免超過5個參數的方法。使用結構傳遞多個參數。  
    Avoid methods with more than 5arguments. Use structures for passing multiple arguments.

6.        每行應該不超過80個字元。  
    Lines should not exceed 80characters.

7.        不要手工編輯任何機器生成的代碼。  
    Do not manually edit any machinegenerated code.    
    a) 如果修改機器生成的代碼，修改代碼格式和風格以符合本編碼標準。  
      If modifying machine generated code, modify theformat and style to match this coding standard.  
    b) 盡可能採用partial類以分解出需要維護的部分。  
      Use partial classes whenever possible to factorout the maintained portions.

8.        避免對顯而易見的內容作注釋。  
   Avoid comments that explain theobvious.     
   a) 代碼應該是自解釋的。 由可讀性強的變數和方法組成的好的代碼應該不需要注釋。  
      Code should be self explanatory.  Good codewith readable variable and method names should not require comments.

9.        僅對操作的前提、內在演算法等寫文檔。  
    Document only operationalassumptions, algorithm insights and so on.

10.    避免方法級的文檔。  
     Avoid method-level documentation.  
     a) 對API文檔採用大量的外部文檔。  
         Use extensive externaldocumentation for API documentation.   
     b) 方法級注釋僅作為對其他開發人員的提示。   
         Use method-level comments onlyas tool tips for other developers.

11.    決不要硬編碼數值，而總是聲明一個常量。  
     Never hard-code a numericvalue, always declare a constant instead.

12.    僅對本來就是常量的值使用const修飾符，例如一周的天數。  
     Use the  const directiveonly on natural constants such as the number of days of week.

13.    避免對唯讀變數使用const修飾符。在此情況下，採用readonly修飾符。   
     Avoid using const on read-only variables. For that,use the readonly directive.  
     public class MyClass   
     {   
        public readonly int Number;   
        public MyClass(int someValue)   
        {   
           Number =someValue;   
        }   
        public const int DaysInWeek = 7;   
     }

14.    對任何假設採用assert。  
      Assert every assumption.   
      a) 平均地，每5行中就有一行是斷言。   
        On average, every fifth line is anassertion.   
      using System.Diagnostics;   
      object GetObject()   
      {  
            object obj =GetObject();   
            Debug.Assert(obj!= null)

15.    每行代碼應該經過白盒測試。  
      Every line of code should be walked through in a搘hitebox?testing manner.

16.    僅捕獲已經顯式處理了的異常。  
      Only catch exceptions for which you haveexplicit handling.

17.    在拋出異常的catch語句中，總是拋出最初異常以保持最初錯誤的堆疊位置。  
      In a catch statement that throws an exception,always throw the original exception to maintain stack location of originalerror.  
     catch(Exception exception)   
     {      
        MessageBox.Show(exception.Message);   
        throw;  //Same as throwexception;   
     }

18.    避免將錯誤代碼作為方法的返回值。  
      Avoid error code as methods return values.

19.    避免定義自訂的異常類。  
      Avoid defining customexception classes.

20.    定義自訂異常時：  
      When defining custom exceptions:   
      a) 從ApplicationException繼承   
        Derive the custom exception fromApplicationException.   
      b) 提供自訂的序列化。   
          Provide customserialization.

21.    避免在一個程式集中有多個Main()方法。  
      Avoid multiple Main()methods in a single assembly.

22.    僅對最需要的類型標記為public，其他的標記為internal。  
      Make only the most necessary types  public,mark others as internal.

23.    避免採用friend程式集，因為這樣增加了程式集間的耦合度。  
      Avoid friend assemblies, as it increasesinter-assembly coupling.

24.    避免使用依賴於從特定位置運行的程式集的代碼。  
      Avoid code that relies on an assembly runningfrom a particular location.

25.    儘量減少應用程式集(用戶端EXE程式集)的代碼。採用類庫而不要包含業務邏輯層代碼。  
      Minimize code inapplication assemblies (EXE client assemblies). Use class libraries instead tocontain business logic.

26.    避免對枚舉提供明確的值。  
      Avoid providing explicitvalues for enums .   
     //Correct    
     public enum Color   
     {      
        Red,Green,Blue   
     }   
     //Avoid    
     public enum Color   
     {      
        Red = 1,Green = 2,Blue = 3   
     }

27.    避免對枚舉指定類型。  
      Avoid specifying a type for an enum.   
     //Avoid    
     public enum Color : long   
     {      
        Red,Green,Blue   
     }

28.    if語句總是使用括弧，即使它包含一句語句。  
      Always use a curly bracescope in an if statement, even if  it conditions a single statement.

29.    避免使用?:條件算符。  
      Avoid using the trinary conditionaloperator.

30.    避免在布林條件陳述式中調用函數。賦值到區域變數並檢查它們的值。  
     Avoid function calls inBoolean conditional statements. Assign into local variables and check onthem:   
     bool IsEverythingOK()   
     {...}  
     //避免：  
     //Avoid:    
     if(IsEverythingOK())   
     {...}  
     //採用：  
     //Instead:    
     bool ok = IsEverythingOK();   
     if(ok)   
     {...}

31.    總是使用從0開始的陣列。  
     Always use zero-basedarrays.

32.    總是使用一個for迴圈顯式地初始化一個參考類型的陣列。  
     Always explicitly initialize an array of referencetypes using a for loop.   
     public class MyClass   
     {}   
     MyClass[] array = new MyClass[100];   
     for(int index = 0; index < array.Length; index++)   
     {   
        array[index] = new MyClass();   
     }

33.    不用提供public或protected成員變數，而是使用屬性。  
     Do not provide public orprotected member variables. Use properties instead.

34.    避免使用new繼承修飾符，而是使用override。  
       Avoid using the new inheritance qualifier.Use override instead.

35.    對非密封類總是將public和protected方法標記為virtual。  
       Always mark  public and protected methods as  virtual in a non sealed class.

36.    除非涉及到交互操作，永遠不要用不安全的代碼。  
       Never use unsafe code unless when using interop.

37.    避免顯式類型轉換。使用as演算法防護性地轉換類型。  
       Avoid explicitcasting. Use the as operator to defensively cast to a type.  
       Dog dog = new GermanShepherd();   
       GermanShepherd shepherd = dog asGermanShepherd;   
       if(shepherd != null)   
       {...}

38.    類成員有委託時：  
       With delegates as class members:   
       a) 使用前將委託複製到區域變數，以避免併發衝突。  
          Copy a delegate to alocal variable before publishing to avoid concurrency race condition.   
       b) 調用前始終檢查委託是否為空。  
          Always check a delegatefor null before invoking it.   
       public class MySource   
       {   
          public eventEventHandler  MyEvent;   
          public void FireEvent()   
         {   
            EventHandlertemp = MyEvent;   
            if(temp !=null)   
            {   
              temp(this,EventArgs.Empty);   
            }   
         }   
      }

39.    不要提供public的事件成員變數，而是使用事件訪問器。  
       Do not providepublic event member variables. Use event accessors instead.   
       public class MySource   
       {   
          MyDelegate m\_SomeEvent;   
          public event MyDelegateSomeEvent   
          {   
             add   
            {   
               m\_SomeEvent += value;   
             }   
             remove  
            {   
               m\_SomeEvent -= value;   
             }   
          }   
       }

40.    使用Programming .NET Components中定義的EventsHelper類安全地發佈事件。  
       Use the  EventsHelper class definedin Programming .NET Components to publish events defensively.

41.    總是使用介面。  
       Always use interfaces.   
       a) 參見Programming .NET Components第一和第三章。   
          See Chapters 1 and 3 in Programming .NET Components.

42.    類和介面中方法和屬性的比例至少是2：1。  
       Classes and interfaces should have atleast 2:1 ratio of methods to properties.

43.    避免使用一個成員的介面。  
       Avoid interfaces with one member.

44.    努力使每個介面擁有3-5個成員。  
       Strive to have 3-5 members per interface.

45.    每個介面不用超過20個成員。  
       No more than 20 members per interface.   
       a) 12可能是實際應用的極限了。   
           12 is probably apractical limit.

46.    避免將事件作為介面成員。  
       Avoid events as interface members.

47.    避免使用抽象方法，而是使用介面代替。  
       Avoid abstract methods, use interfacesinstead.

48.    在類層次中暴露介面。  
       Expose interfaces on class hierarchies.   
       a) 參見Programming .NET Components第三章。   
          See Chapter 3 in Programming .NET Components.

49.    優先使用明確的介面實現。  
       Prefer using explicit interfaceimplementation.   
       a) 參見Programming .NET Components第三章。   
          See Chapter 3 in Programming .NET Components.

50.    永遠不要假設一種類型支援某個介面。防護性地檢查是否支援該介面。  
       Never assume a type supports an interface.Defensively query for that interface.  
       SomeType obj1;   
       IMyInterface obj2;   
         
       /\* Some code to initialize obj1, then: \*/   
       obj2 = obj1 as IMyInterface;   
       if(obj2 != null)   
      {   
        obj2.Method1();   
       }   
       else   
      {   
         //Handle error in expectedinterface    
      }

51.    將呈現給使用者的字串永遠不用硬編碼，而是使用資源。  
       Never hardcode strings that will bepresented to end users. Use resources instead.

52.    發佈時可能修改的字串永遠不用硬編碼，例如連接字串。  
       Never hardcode strings that might changebased on deployment such as connection strings.

53.    構建一個長字串時，使用StringBuilder，不要用string。  
       When building along string, use StringBuilder, not string.

54.    避免提供帶結構的方法。  
       Avoid providing methods onstructures.    
       a)  參數化的構造函數是鼓勵使用的。  
           Parameterized constructors are encouraged.    
       b)  可以重載算符。  
            Can overloadoperators.

55.    當提供靜態成員變數時，總是提供一個靜態構造函數。  
       Always provide a static constructor whenproviding static member variables.

56.    只要可以用前期綁定就不要用後期綁定。  
       Do not uselate-binding invocation when early-binding is possible.

57.    對應用程式進行日誌和跟蹤。  
       Use application logging and tracing.

58.    除非在switch語句中跳轉，永遠不要用goto語句。  
       Never use gotounless in a switch statement fall-through.

59.    switch語句中總是使用default用於加斷言。  
       Always have adefault case in a switch statement that asserts .   
       int number = SomeMethod();   
       switch(number)   
       {   
          case 1:   
            Trace.WriteLine("Case 1:");   
             break;  
          case 2:   
            Trace.WriteLine("Case 2:");   
             break;  
          default:   
            Debug.Assert(false);   
             break;  
       }

60.    除非在構造函數中調用另一個構造函數，否則不用使用this。  
       Do not use the  this reference unlessinvoking another constructor from within a constructor.  
       //Example of proper use of this  
       public class MyClass   
       {   
          public MyClass(stringmessage)   
          {}   
          public MyClass() :this("hello")   
          {}   
       }

61.    除非為了解決調用基類構造函數時成員名的衝突，否則不要使用base訪問基類的成員。  
       Do not use the base word to access baseclass members unless you wish to resolve a conflict with a subclassesmember of the same name or when invoking a base class constructor.   
     //Example of proper use of 抌ase?  
     public class Dog   
     {   
        public Dog(string name)   
        {}   
        virtual public void Bark(inthowLong)   
        {}   
     }   
     public class GermanShepherd : Dog   
     {   
        public GermanShepherd(string name):base(name)   
        {}   
        override public void Bark(inthowLong)    
        {   
          base.Bark(howLong);     
        }   
     }

62.    根據Programming .NET Components第四章中的範本實現Dispose()和Finalize()方法。  
       Implement  Dispose() and Finalize() methods based on the template  in Chapter 4 of Programming.NET Components.

63.    使用泛型的代碼中避免與System.Object進行類型轉換，而是使用限制或as算符。  
       Avoid casting to and from System.Object in code that uses generics. Use constraints or the as operatorinstead:  
       class SomeClass   
       {}   
        //避免 Avoid:   
       class MyClass    
      {      
          void SomeMethod(Tt)      
         {   
           object temp =t;         
           SomeClass obj =(SomeClass)temp;      
         }   
       }   
       //正確 Correct:   
       class MyClass where T : SomeClass   
       {      
          void SomeMethod(Tt)      
          {   
            SomeClass obj = t;      
          }   
       }

64.    泛型介面不要定義限制。介面層的限制通常能用強類型代替。  
       Do not define constraints in genericinterfaces. Interface level-constraint can often be replaced bystrong-typing.  
       public class Customer   
       {...}  
       //避免 Avoid:   
       public interface IList where T :Customer    
       {...}  
       //正確  Correct:   
       public interface ICustomerList :IList    
       {...}

65.    不要在介面中定義與方法相關的限制。  
       Do not define method-specific constraintsin interfaces.

66.    在資料結構中總是優先使用[C#](http://dev.21tx.com/dotnet/csharp/)泛型。  
       Always prefer using C# generics in datastructures.

# 3.     專案設置和專案結構 Project Settings and Project Structure

1.        總是以4級警告建立專案（圖略）。

Always buildyour project with warning level 4

2.        在發佈版中將警告作為錯誤（注意這不是VS.NET的缺省設置）（圖略）。  
    Treat warning as errors in Releasebuild (note that this is not the default of VS.NET).     a) 雖然是可選的，本標準也推薦在調試版中將警告作為錯誤。

Although it is optional, this standard recommend treating warningsas errors in debug builds as well.

3.        永遠不要抑制特定的編譯警告（圖略）。

Never suppressspecific compiler warnings.

4.        總是在應用程式的設定檔中顯式地說明支援的執行階段版本。參見Programming .NET Components第五章。

Always explicitly state your supported runtime versions in theapplication configuration file. See Chapter 5 in Programming .NET Components.

5.        避免顯式地自訂版本改向和綁定到CLR程式集。

   Avoid explicit custom version redirection and binding to CLR assemblies.

6.        避免顯式的預編譯定義(#define)。使用專案設置定義條件編譯常量。

   Avoid explicit preprocessor definitions (#define). Use the projectsettings for defining conditional compilation constants.

7.        不要在AssemblyInfo.cs中放任何邏輯。

   Do not put any logic inside AssemblyInfo.cs.

8.        除了在AssemblyInfo.cs，不要在任何檔中放程式集屬性。

   Do not put any assembly attributes in any file besides AssemblyInfo.cs.

9.        在AssemblyInfo.cs中提供所有欄位，例如公司名稱、描述、版權等。

Populate all fields in AssemblyInfo.cs such as company name, description, copyright notice.

10.    所有程式集應該使用相對路徑引用。

   All assembly references should use relative path.

11.     不允許在程式集中迴圈引用。

   Disallow cyclic references between assemblies.

12.    避免多模組的程式集。

   Avoid multi-module assemblies.

13.    缺省總是以非檢查的方式運行（為了性能考慮），但是對易於溢出或下溢的操作顯式使用檢查模式（圖略）。

Always run code unchecked by default (for performance sake), butexplicitly in checked mode for overflow or underflow prone operations.

    int CalcPower(int number,int power)

    {

       int result = 1;

       for(int count = 1;count <= power;count++)

       {

          checked

          {

              result \*= number;

          }

       }

       return result;

    }

14.  避免使用Exception視窗（Debug|Exceptions）篡改異常處理。

    Avoid tampering with exception handling using the Exception window(Debug|Exceptions).

15.  努力對同一邏輯應用程式中（通常是一個解決方案）的所有程式集和用戶端使用統一的版本號。

    Strive to use uniform version numbers on all assemblies and clients inthe same logical application (typically a solution).

16. Visual Studio.NET應用的設定檔命名為App.config，並將其包括在項目中。

    Name your Visual Studio.NET application configuration file asApp.config, and include it in the project.

17.  避免使用顯式代碼來排除方法(#if#endif)，而是使用條件方法。

    Avoid explicit code exclusion of method calls  (#if...#endif). Use conditional methodsinstead.

    public class MyClass

    {

       [Conditional("MySpecialCondition")]

       public void MyMethod()

       {}

    }

18.  將VS.NET缺省的專案結構改為標準的佈局，對專案檔案夾和檔應用統一的結構。

    Modify VS.NET default project structure to your project standard layout,and apply uniform structure for project folders and files.

19.  連結一個包含所有解決方案級資訊的全域共用檔（圖略）。

    Link all solution-wide information to a global shared file:

20.  定位字元選用"插入空格"，使用3個空格代替定位字元。

    Insert spaces for tabs. Use 3 spaces instead of tabs。

    a) 在工具|選項|文字編輯器|C#|定位字元中設置

       Tools|Options|Text Editor|C#|Tabs

21.  發佈版中應該包含調試符號。

    Release build should contain debug symbols.

22.  總是對程式集簽名，包括用戶端應用程式。

    Always sign your assemblies, including the client applications.

23.  總是使用專案的SNK檔對交互操作程式集簽名（圖略）。

    Always sign interop assemblies with the project's SNK file

4  Framework特別指導  
   Framework Specific Guidelines

**4.1 資料訪問**

    Data [Access](http://dev.21tx.com/database/access/)   
1.  總是使用類型安全的資料集或者資料表。避免使用原始的[ADO.NET](http://dev.21tx.com/dotnet/adonet/)。  
    Always use type-safe data sets  or data tables . Avoidraw ADO.NET.   
2.  訪問資料庫時總是使用事務。  
    Always use transactions when accessing a database.    
     a) 總是使用服務元件事務。   
        Always use Enterprise Servicestransactions.   
     b) 不要使用ADO.NET事務。  
        Do not use ADO.NETtransactions.    
3.  總是將事務隔離級別設置為序列的。  
    Always use transaction isolation level set to Serializable.   
     a) 使用其它隔離級別需要管理層決定。   
        Requires management decision to useanything else.   
4.  不要使用[伺服器](http://www.21tx.com/server/)流覽器將資料庫連接拖到Windows表單、[ASP.NET](http://dev.21tx.com/dotnet/aspnet/)表單或Web服務中。這樣做耦合了介面層和資料層。  
    Do not use the Server Explorer to drop connections onwindows forms, ASP.NET forms or web services. Doing so couples the presentationtier to the data tier.   
5.  避免使用[SQL Server](http://dev.21tx.com/database/mssql/)驗證。  
    Avoid SQL Server authentication.    
     a) 而是使用Windows驗證。   
        Use Windows authentication instead.   
6.  將訪問SQL Server的元件以調用該元件用戶端不同的身份運行。  
    Run components accessing SQL Server under separate identityfrom that of the calling client.   
7.  總是在高層用類型安全的類包裝存儲過程。僅在那個類中調用存儲過程。  
    Always wrap your stored procedures in a high level, typesafe class. Only that class invokes the stored procedures.   
8.  避免在存儲過程中放任何邏輯。  
    Avoid putting any logic inside a stored procedure.   
     a) 如果存儲過程中有IF，你可能犯了錯誤。   
     If there is an IF inside a storedprocedure, you are doing something wrong.

## [C#命名規則、開發習慣和風格](http://www.cnblogs.com/netshuai/archive/2008/06/29/1231934.html)

## 1.     檔命名組織

### 1-1檔命名

1.        檔案名遵從Pascal命名法，無特殊情況，副檔名小寫。

2.        使用統一而又通用的檔副檔名： C# 類 .cs

### 1-2文件注釋

1.         在每個檔頭必須包含以下注釋說明

1 在每個檔頭必須包含以下注釋說明

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

            // Copyright (C) 2004 軟體有限公司

            // 版權所有。

            //

            // 檔案名：

            // 檔功能描述：

            //

            //

            // 創建標識：

            //

            // 修改標識：

            // 修改描述：

            //

            // 修改標識：

            // 修改描述：

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

檔功能描述只需簡述，具體詳情在類的注釋中描述。

創建標識和修改標識由創建或修改人員的拼音或英文名加日期組成。如：

   Jiekengxu 20040408

一天內有多個修改的只需做一個在注釋說明中做一個修改標識就夠了。

在所有的代碼修改處加上修改標識的注釋。

## 2.     代碼外觀

### 2-1列寬

    代碼列寬控制在110字元左右。

### 2-2換行

      當運算式超出或即將超出規定的列寬，遵循以下規則進行換行

          1、在逗號後換行。

          2、 在操作符前換行。

          3、規則1優先於規則2。

     當以上規則會導致代碼混亂的時候自己採取更靈活的換行規則。

### 2-3縮進

     縮進應該是每行一個Tab(4個空格)，不要在代碼中使用Tab字元。

 Visual Studio.Net設置：工具->選項->文字編輯器->C#->定位字元->插入空格

### 2-4空行

空行是為了將邏輯上相關聯的代碼分塊，以便提高代碼的可閱讀性。

    在以下情況下使用兩個空行

    1、介面和類的定義之間。

    2、枚舉和類的定義之間。

    3、類與類的定義之間。

    在以下情況下使用一個空行

         1、方法與方法、屬性與屬性之間。

         2、方法中變數聲明與語句之間。

         3、方法與方法之間。

         4、方法中不同的邏輯塊之間。

      　 5、方法中的返回語句與其他的語句之間。

         6、屬性與方法、屬性與欄位、方法與欄位之間。

         7、注釋與它注釋的語句間不空行，但與其他的語句間空一行。

### 2-5空格

在以下情況中要使用到空格

        1、 關鍵字和左括符 “(” 應該用空格隔開。如

           while (true)

           注意在方法名和左括符 “(” 之間不要使用空格，這樣有助於辨認代碼中的方法調用與關鍵字。

2、  多個參數用逗號隔開，每個逗號後都應加一個空格。

3、  除了 . 之外，所有的二元操作符都應用空格與它們的運算元隔開。一元操作符、++及--與操作    數間不需要空格。如

                     a += c + d;

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

                while  (d++ = s++)

                {

                    n++;

                }

                PrintSize(“size is “ + size + “\n”);

4、 語句中的運算式之間用空格隔開。如

       for (expr1; expr2; expr3)

### 2-6括弧 - ()

      1、 左括弧“(” 不要緊靠關鍵字，中間用一個空格隔開。

      2、 左括弧“(”  與方法名之間不要添加任何空格。

      3、 沒有必要的話不要在返回語句中使用()。如

        if (condition)

            Array.Remove(1)

            return 1

### 2-7花括弧 - {}

       1、 左花括弧 “{” 放於關鍵字或方法名的下一行並與之對齊。如

             if (condition)

             {

             }

             public int Add(int x, int y)

             {

             }

       2、 左花括弧 “{” 要與相應的右花括弧 “}”對齊。

       3、 通常情況下左花括弧 “{”單獨成行，不與任何語句並列一行。

       4、 if、while、do語句後一定要使用{}，即使{}號中為空或只有一條語句。如

             if (somevalue == 1)

             {

                 somevalue = 2;

             }

5、 右花括弧 “}” 後建議加一個注釋以便於方便的找到與之相應的 {。如

               while (1)

               {

                   if (valid)

                   {

                   } // if valid

                   else

                   {

                   } // not valid

                } // end forever

## 3.     程式注釋

### 3-1注釋概述

1、修改代碼時，總是使代碼周圍的注釋保持最新。

2、在每個常式的開始，提供標準的注釋樣本以指示常式的用途、假設和限制很有幫助。注釋樣本應該是解釋它為什麼存在和可以做什麼的簡短介紹.

3、避免在代碼行的末尾添加注釋；行尾注釋使代碼更難閱讀。不過在批註變數聲明時，行尾注釋是合適的；在這種情況下，將所有行尾注釋在公共定位停駐點處對齊。

4 、避免雜亂的注釋，如一整行星號。而是應該使用空白將注釋同代碼分開。

5 、避免在塊注釋的周圍加上印刷框。這樣看起來可能很漂亮，但是難於維護。

6 、在部署發佈之前，移除所有臨時或無關的注釋，以避免在日後的維護工作中產生混亂。

7 、如果需要用注釋來解釋複雜的代碼節，請檢查此代碼以確定是否應該重寫它。盡一切可能不注釋難以理解的代碼，而應該重寫它。儘管一般不應該為了使代碼更簡單以便於人們使用而犧牲性能，但必須保持性能和可維護性之間的平衡。

8 、在編寫注釋時使用完整的句子。注釋應該闡明代碼，而不應該增加多義性。

9 、在編寫代碼時就注釋，因為以後很可能沒有時間這樣做。另外，如果有機會複查已編寫的代碼，在今天看來很明顯的東西六周以後或許就不明顯了。

10 、避免多餘的或不適當的注釋，如幽默的不主要的備註。

11、 使用注釋來解釋代碼的意圖。它們不應作為代碼的連線翻譯。

12、 注釋代碼中不十分明顯的任何內容。

13 、為了防止問題反復出現，對錯誤修復和解決方法代碼總是使用注釋，尤其是在團隊環境中。

14 、對由迴圈和邏輯分支組成的代碼使用注釋。這些是幫助原始程式碼讀者的主要方面。

15 、在整個應用程式中，使用具有一致的標點和結構的統一樣式來構造注釋。

16 、用空白將注釋同注釋分隔符號分開。在沒有顏色提示的情況下查看注釋時，這樣做會使注釋很明顯且容易被找到。

17 、在所有的代碼修改處加上[修改標識](http://writeblog.csdn.net/fckeditor/editor/fckeditor.html?InstanceName=ctl00_ContentPlaceHolder1_EntryEditor1_FCKEditor&Toolbar=Default#%E4%BF%AE%E6%94%B9%E6%A0%87%E8%AF%86)的注釋。

18 、為了是層次清晰，在閉合的右花括弧後注釋該閉合所對應的起點。

    namespace Langchao.Procument.Web

{

} // namespace Langchao.Procument.Web

### 3-2文檔型注釋

 該類注釋採用.Net已定義好的Xml標籤來標記，在聲明介面、類、方法、屬性、欄位都應該使用該類注釋，以便代碼完成後直接生成代碼文檔，讓別人更好的瞭解代碼的實現和介面。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)

   {

           }

### 3-3單行注釋

      該類注釋用於

1 方法內的代碼注釋。如變數的聲明、代碼或程式碼片段的解釋。注釋示例：

          //

// 注釋語句

          //

         private int number;

或

         // 注釋語句

         private int number;

          2 方法內變數的聲明或花括弧後的注釋, 注釋示例：

               if ( 1 == 1)    // always true

               {

                  statement;

               }           // always true

### 3-4注釋標籤

|  |  |  |
| --- | --- | --- |
| 標籤 | 用法 | 作用 |
| <c> | c>text</c>    text 希望將其指示為代碼的文本。 | 為您提供了一種將說明中的文本標記為代碼的方法。使用 [<code>](file:///C:\Documents%20and%20Settings\zjy.CAIAO.000\Local%20Settings\Temp\vclrfcode.htm) 將多行指示為代碼 |
| <para> | <para>content</para>  content段落文本。 | 用於諸如 [<remarks>](file:///C:\Documents%20and%20Settings\zjy.CAIAO.000\Local%20Settings\Temp\vclrfremarks.htm) 或 [<returns>](file:///C:\Documents%20and%20Settings\zjy.CAIAO.000\Local%20Settings\Temp\vclrfreturns.htm) 等標記內，使您得以將結構添加到文本中。 |
| <param> | <param name='name'>description</param>  name 為方法參數名。將此名稱用單引號括起來 (' ')。 | 應當用於方法聲明的注釋中，以描述方法的一個參數。 |
| <paramref> | <paramref name="name"/>    name  要引用的參數名。將此名稱用雙引號括起來 (" ")。 | <paramref> 標記為您提供了一種指示詞為參數的方法。可以處理 XML 檔，從而用某種獨特的方法格式化該參數。 |
| <see> | <see cref="member"/>    cref = "member" 對可以通過當前編譯環境進行調用的成員或欄位的引用。編譯器檢查到給定代碼元素存在後，將 member 傳遞給輸出 XML 中的元素名。必須將 member 括在雙引號 (" ") 中。 | 使您得以從文本內指定連結。使用 [<seealso>](file:///C:\Documents%20and%20Settings\zjy.CAIAO.000\Local%20Settings\Temp\vclrfseealso.htm) 指示希望在“請參閱”一節中出現的文本。 |
| <seealso> | <seealso cref="member"/>    cref = "member" 對可以通過當前編譯環境進行調用的成員或欄位的引用。編譯器檢查到給定代碼元素存在後，將 member 傳遞給輸出 XML 中的元素名。必須將 member 括在雙引號 (" ") 中 | 使您得以指定希望在“請參閱”一節中出現的文本。使用 [<see>](file:///C:\DOCUME~1\ZJYCAI~1.000\LOCALS~1\Temp\vclrfsee.htm) 從文本 |
| <example> | <example>description</example>    description  代碼示例的說明。 | 使用 <example> 標記可以指定使用方法或其他庫成員的示例。一般情況下，這將涉及到 [<code>](file:///C:\Documents%20and%20Settings\zjy.CAIAO.000\Local%20Settings\Temp\vclrfcode.htm) 標記的使用。 |
| <code> | <code>content</code>    content 為希望將其標記為代碼的文本。 | 記為您提供了一種將多行指示為代碼的方法。使用 [<c>](file:///C:\DOCUME~1\ZJYCAI~1.000\LOCALS~1\Temp\vclrfc.htm) 指示應將說明中的文本標記為代碼 |
| <summary> | <summary>description</summary>    此處description 為對象的摘要。 | 應當用於描述類型成員。使用 [<remarks>](file:///C:\Documents%20and%20Settings\zjy.CAIAO.000\Local%20Settings\Temp\vclrfremarks.htm) 以提供有關類型本身的資訊。 |
| <exception> | <exception cref="member">description</exception>  cref = "member" 對可從當前編譯環境中獲取的異常的引用。編譯器檢查到給定異常存在後，將 member 轉換為輸出 XML 中的規範化元素名。必須將 member 括在雙引號 (" ") 中。  description 說明。 | <exception> 標記使您可以指定類能夠引發的異常。 |
| <include> | <include file='filename' path='tagpath[@name="id"]' />  filename 包含文檔的檔案名。該檔案名可用路徑加以限定。將 filename 括在單引號中 (' ')。  Tagpath：filename 中指向標記名的標記路徑。將此路徑括在單引號中 (' ')。  name 注釋前邊的標記中的名稱說明符；名稱具有一個 id。  id  位於注釋之前的標記的 id。將此 id 括在雙引號中 (" ")。 | <include> 標記使您得以引用描述原始程式碼中類型和成員的另一文件中的注釋。這是除了將文檔注釋直接置於原始程式碼檔中之外的另一種可選方法。  <include> 標記使用 XML XPath 語法。有關自訂 <include> 使用的方法，請參閱 XPath 文檔。 |
| <list> | <list type="bullet" | "number" | "table">     <listheader>        <term>term</term>        <description>description</description>     </listheader>     <item>        <term>term</term>        <description>description</description>     </item>  </list>    term  定義的項，該項將在 text 中定義。    description  目符號清單或編號清單中的項或者 term 的定義。 | <listheader> 塊用於定義表或定義列表中的標題行。定義表時，只需為標題中的項提供一個項。  列表中的每一項用 <item> 塊指定。創建定義清單時，既需要指定 term 也需要指定 text。但是，對於表、專案符號清單或編號清單，只需為 text 提供一個項。  清單或表所擁有的 <item> 塊數可以根據需要而定。 |
| <permission> | <permission cref="member">description</permission>    cref = "member" 對可以通過當前編譯環境進行調用的成員或欄位的引用。編譯器檢查到給定代碼元素存在後，將 member 轉換為輸出 XML 中的規範化元素名。必須將 member 括在雙引號 (" ") 中。    description  成員的訪問的說明。 | <permission> 標記使您得以將成員的訪問記入文檔。**System.Security.PermissionSet** 使您得以指定對成員的訪問。 |
| <remarks> | <remarks>description</remarks>    description 成員的說明。 | <remarks> 標記是可以指定有關類或其他類型的概述資訊的位置。[<summary>](file:///C:\Documents%20and%20Settings\zjy.CAIAO.000\Local%20Settings\Temp\vclrfsummary.htm) 是可以描述該類型的成員的位置。 |
| <returns> | <returns>description</returns>    description 返回值的說明。 | <returns> 標記應當用於方法聲明的注釋，以描述返回值。 |
| <value> | <value>property-description</value>    property-description 屬性的說明。 | <value> 標記使您得以描述屬性。請注意，當在 Visual Studio .NET 開發環境中通過代碼嚮導添加屬性時，它將會為新屬性添加 [<summary>](file:///C:\Documents%20and%20Settings\zjy.CAIAO.000\Local%20Settings\Temp\vclrfsummary.htm) 標記。然後，應該手動添加 <value> 標記以描述該屬性所表示的值。 |
|  |  |  |

## 4.     申明

### 4-1每行聲明數

一行只建議作一個聲明，並按字母順序排列。如

          int level;   //推薦

          int size;    //推薦

          int x, y;    //不推薦

### 4-2初始化

      建議在變數聲明時就對其做初始化。

### 4-3位置

      變數建議置於塊的開始處，不要總是在第一次使用它們的地方做聲明。如

         void MyMethod()

          {

              int int1 = 0;         // beginning of method block

if (condition)

{

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

                  ...

              }

          }

        不過也有一個例外

                for (int i = 0; i < maxLoops; i++)

                {

                    ...

                }

        應避免不同層次間的變數重名，如

            int count;

            ...

void MyMethod()

{

                if (condition)

                {

                    int count = 0;     // 避免

                     ...

                 }

                 ...

}

### 4-4類和介面的聲明

      1 在方法名與其後的左括弧間沒有任何空格。

      2 左花括弧 “{” 出現在聲明的下行並與之對齊，單獨成行。

      3 方法間用一個空行隔開。

### 4-5欄位的聲明

不要使用是 public 或 protected 的實例欄位。如果避免將欄位直接公開給開發人員，可以更輕鬆地對類進行版本控制，原因是在維護二進位相容性時欄位不能被更改為屬性。考慮為欄位提供 get 和set 屬性訪問器，而不是使它們成為公共的。 get 和 set 屬性訪問器中可執行代碼的存在使得可以進行後續改進，如在使用屬性或者得到屬性更改通知時根據需要創建物件。下面的代碼示例闡釋帶有get 和 set 屬性訪問器的私有實例欄位的正確使用。 示例：

            public class Control: Component

            {

               private int handle;

               public int Handle

               {

                  get

                  {

                     return handle;

                  }

               }

}

## 5.     命名規範

### 5-1命名概述

名稱應該說明“什麼”而不是“如何”。通過避免使用公開基礎實現（它們會發生改變）的名稱，可以保留簡化複雜性的抽象層。例如，可以使用 GetNextStudent()，而不是 GetNextArrayElement()。

命名原則是：

選擇正確名稱時的困難可能表明需要進一步分析或定義項的目的。使名稱足夠長以便有一定的意義，並且足夠短以避免冗長。唯一名稱在程式設計上僅用於將各項區分開。表現力強的名稱是為了幫助人們閱讀；因此，提供人們可以理解的名稱是有意義的。不過，請確保選擇的名稱符合適用語言的規則和標準。

以下幾點是推薦的命名方法。

1、避免容易被主觀解釋的難懂的名稱，如方面名 AnalyzeThis()，或者屬性名 xxK8。這樣的名稱會導致多義性。

2、在類屬性的名稱中包含類名是多餘的，如 Book.BookTitle。而是應該使用 Book.Title。

3、只要合適，在變數名的末尾或開頭加計算限定詞（Avg、Sum、Min、Max、Index）。

4、在變數名中使用互補對，如 min/max、begin/end 和 open/close。

5、布林變數名應該包含 Is，這意味著 Yes/No 或 True/False 值，如 fileIsFound。

6、在命名狀態變數時，避免使用諸如 Flag 的術語。狀態變數不同於布林變數的地方是它可以具有兩個以上的可能值。不是使用 documentFlag，而是使用更具描述性的名稱，如 documentFormatType。 （此項只供參考）

7、即使對於可能僅出現在幾個代碼行中的生存期很短的變數，仍然使用有意義的名稱。僅對於短迴圈索引使用單字母變數名，如 i 或 j。 可能的情況下，儘量不要使用原義數位或原義字串，如

For i = 1 To 7。而是使用命名常數，如 For i = 1 To NUM\_DAYS\_IN\_WEEK 以便於維護和理解。

### 5-2大小寫規則

大寫

識別字中的所有字母都大寫。僅對於由兩個或者更少字母組成的識別字使用該約定。例如：

System.IO

System.Web.UI

下表匯總了大寫規則，並提供了不同類型的識別字的示例。

|  |  |  |
| --- | --- | --- |
| 識別字 | 大小寫 | 示例 |
| 類 | Pascal | AppDomain |
| 枚舉類型 | Pascal | ErrorLevel |
| 枚舉值 | Pascal | FatalError |
| 事件 | Pascal | ValueChange |
| 異常類 | Pascal | WebException  注意 總是以 Exception 尾碼結尾。 |
| 唯讀的靜態欄位 | Pascal | RedValue |
| 介面 | Pascal | IDisposable  注意 總是以 I 首碼開始。 |
| 方法 | Pascal | ToString |
| 命名空間 | Pascal | System.Drawing |
| 屬性 | Pascal | BackColor |
| 公共實例欄位 | Pascal | RedValue  注意 很少使用。屬性優於使用公共實例欄位。 |
| 受保護的實例欄位 | Camel | redValue  注意 很少使用。屬性優於使用受保護的實例欄位。 |
| 私有的實例欄位 | Camel | redValue |
| 參數 | Camel | typeName |
| 方法內的變數 | Camel | backColor |

### 5-3縮寫

為了避免混淆和保證跨語言交交互操作，請遵循有關區縮寫的使用的下列規則：

      1 不要將縮寫或縮略形式用作識別字名稱的組成部分。例如，使用 GetWindow，而不要使用 GetWin。

      2 不要使用電腦領域中未被普遍接受的縮寫。

3 在適當的時候，使用眾所周知的縮寫替換冗長的片語名稱。例如，用 UI 作為 User Interface 縮

寫，用 OLAP 作為 On-line Analytical Processing 的縮寫。

4在使用縮寫時，對於超過兩個字元長度的縮寫請使用 Pascal 大小寫或 Camel 大小寫。例如，使用 HtmlButton 或 HTMLButton。但是，應當大寫僅有兩個字元的縮寫，如，System.IO，而不是 System.Io。

5 不要在識別字或參數名稱中使用縮寫。如果必須使用縮寫，對於由多於兩個字元所組成的縮寫請

使用Camel 大小寫，雖然這和單詞的標準縮寫相衝突。

### 5-4命名空間

      1、命名命名空間時的一般性規則是使用公司名稱，後跟技術名稱和可選的功能與設計，如下所示。

CompanyName.TechnologyName[.Feature][.Design]

        例如：

namespace Langchao.Procurement             //浪潮公司的採購單管理系統

         namespace Langchao.Procurement.DataRules   //浪潮公司的採購單管理系統的業務規則模組

      2、命名空間使用Pascal大小寫，用逗號分隔開。

3、TechnologyName 指的是該項目的英文縮寫，或軟體名。  
4、命名空間和類不能使用同樣的名字。例如，有一個類被命名為Debug後，就不要再使用Debug作為一個名稱空間名。

### 5-5類

1、使用 Pascal 大小寫。

2、用名詞或名詞短語命名類。

3、使用全稱避免縮寫，除非縮寫已是一種公認的約定，如URL、HTML

4 、不要使用類型首碼，如在類名稱上對類使用 C 首碼。例如，使用類名稱FileStream，而不是

CFileStream。

5 、不要使用底線 (\_)。

6 、有時候需要提供以字母 I 開始的類名稱，雖然該類不是介面。只要 I 是作為類名稱組成部分的整個單詞的第一個字母，這便是適當的。例如，類名稱 IdentityStore 是適當的。在適當的地方，使用複合單詞命名派生的類。派生類名稱的第二個部分應當是基類的名稱。例如，ApplicationException 對於從名為 Exception 的類派生的類是適當的名稱，原因ApplicationException 是一種Exception。請在應用該規則時進行合理的判斷。例如，Button 對於從 Control 派生的類是適當的名稱。儘管按鈕是一種控制項，但是將 Control 作為類名稱的一部分將使名稱不必要地加長。

            public class FileStream

            public class Button

public class String

### 5-6介面

      以下規則概述介面的命名指南：

      1、用名詞或名詞短語，或者描述行為的形容詞命名介面。例如，介面名稱IComponent 使用描述性

名詞。介面名稱 ICustomAttributeProvider 使用名詞短語。名稱 IPersistable 使用形容詞。

2、使用 Pascal 大小寫。

      3、少用縮寫。

4、給介面名稱加上字母 I 首碼，以指示該類型為介面。在定義類/介面對（其中類是介面的標準

實現）時使用相似的名稱。兩個名稱的區別應該只是介面名稱上有字母 I 首碼。

5、不要使用底線 (\_)。

6、當類是介面的標準執行時，定義這一對類/介面組合就要使用相似的名稱。兩個名稱的不同之處

只是介面名前有一個I首碼。

          以下是正確命名的介面的示例。

                public interface IServiceProvider

                public interface IFormatable

          以下代碼示例闡釋如何定義 IComponent 介面及其標準實現 Component 類。

               public interface IComponent

               {

                   // Implementation code goes here.

               }

               public class Component: IComponent

               {

                   // Implementation code goes here.

}

### 5-7屬性 (Attribute)

應該總是將尾碼 Attribute 添加到自訂屬性類。以下是正確命名的屬性類的示例。

          public class ObsoleteAttribute

          {

}

### 5-8枚舉 (Enum)

       枚舉 (Enum) 數值型別從 Enum 類繼承。以下規則概述枚舉的命名指南：

           1 對於 Enum 類型和值名稱使用 Pascal 大小寫。

           2 少用縮寫。

           3 不要在 Enum 類型名稱上使用 Enum 尾碼。

           4 對大多數 Enum 類型使用單數名稱，但是對作為位域的 Enum 類型使用複數名稱。

           5 總是將 FlagsAttribute 添加到位域 Enum 類型。

### 5-9參數

       以下規則概述參數的命名指南：

           1、使用描述性參數名稱。參數名稱應當具有足夠的描述性，以便參數的名稱及其類型可用於在大多數情況下確定它的含義。

       2、對參數名稱使用 Camel 大小寫。

           3、 使用描述參數的含義的名稱，而不要使用描述參數的類型的名稱。開發工具將提供有關參數的類型的有意義的資訊。因此， 通過描述意義，可以更好地使用參數的名稱。少用基於類型的參數名稱，僅在適合使用它們的地方使用它們。

           4、不要使用保留的參數。保留的參數是專用參數，如果需要，可以在未來的版本中公開它們。相反，如果在類庫的未來版本中需要更多的資料，請為方法添加新的重載。

           5、不要給參數名稱加匈牙利語類型標記法的首碼。

      以下是正確命名的參數的示例。

               Type GetType(string typeName)

string Format(string format, args() As object)

### 5-10方法

       以下規則概述方法的命名指南：

           1 使用動詞或動詞短語命名方法。

           2 使用 Pascal 大小寫。

           3 以下是正確命名的方法的實例。

              RemoveAll()

              GetCharArray()

Invoke()

### 5-11屬性 (property)

        以下規則概述屬性的命名指南：

            1 使用名詞或名詞短語命名屬性。

            2 使用 Pascal 大小寫。

            3 不要使用匈牙利語標記法。

4 考慮用與屬性的基礎類型相同的名稱創建屬性。例如，如果聲明名為 Color 的屬性，則屬

性的類型同樣應該是 Color。請參閱本主題中後面的示例。

          以下代碼示例闡釋正確的屬性命名。

                public class SampleClass

                 {

                      public Color BackColor

                      {

                           // Code for Get and Set accessors goes here.

                       }

                  }

            以下代碼示例闡釋提供其名稱與類型相同的屬性。

                 public enum Color

                 {

                     // Insert code for Enum here.

                  }

                  public class Control

{

                       public Color Color

                       {

get

{

// Insert code here.

}

set

{

// Insert code here.

}

                        }

                   }

           以下代碼示例不正確，原因是 Color 屬性是 Integer 類型的。

                public enum Color

{

// Insert code for Enum here.

}

                public class Control

                {

                    public int Color

                    {

                        // Insert code here

                     }

                }

           在不正確的示例中，不可能引用 Color 枚舉的成員。Color.Xxx 將被解釋為訪問一個成員，

該成員首先獲取 Color 屬性（ C# 中為 int 類型）的值，然後再訪問該值的某個成員（該成

員必須是 System.Int32 的實例成員）。

### 5-12事件

        以下規則概述事件的命名指南：

            1、對事件處理常式名稱使用 EventHandler 尾碼。

2、指定兩個名為 sender 和 e 的參數。sender 參數表示引發事件的物件。sender 參數始

終是object 類型的，即使在可以使用更為特定的類型時也如此。與事件相關聯的狀態封裝

在名為 e 的事件類的實例中。對 e 參數類型使用適當而特定的事件類。

3、用 EventArgs 尾碼命名事件參數類。

    4、考慮用動詞命名事件。

5、使用動名詞（動詞的“ing”形式）創建表示事件前的概念的事件名稱，用過去式表示事

件後。例如，可以取消的 Close 事件應當具有 Closing 事件和 Closed 事件。不要使用

BeforeXxx/AfterXxx 命名模式。

            6、不要在類型的事件聲明上使用首碼或者尾碼。例如，使用 Close，而不要使用 OnClose。

7、通常情況下，對於可以在派生類中重寫的事件，應在類型上提供一個受保護的方法（稱為

OnXxx）。此方法只應具有事件參數 e，因為發送方總是類型的實例。

    以下示例闡釋具有適當名稱和參數的事件處理常式。

            public delegate void MouseEventHandler(object sender, MouseEventArgs e);

      以下示例闡釋正確命名的事件參數類。

                public class MouseEventArgs : EventArgs

                {

                   int x;

               int y;

                  public MouseEventArgs(int x, int y)

                  {

this.x = x;

this.y = y;

}

                  public int X

                  {

                      get

                      {

                          return x;

                      }

                  }

                  public int Y

                  {

                 get

                       {

                           return y;

                       }

                  }

                }

### 5-13常量 (const)

        以下規則概述常量的命名指南：

        所有單詞大寫，多個單詞之間用 "\_" 隔開。 如

                 public const string PAGE\_TITLE = "Welcome";

### 5-14欄位

        以下規則概述欄位的命名指南：

            1、private、protected 使用 Camel 大小寫。

            2、public 使用 Pascal 大小寫。

3、拼寫出欄位名稱中使用的所有單詞。僅在開發人員一般都能理解時使用縮寫。欄位名稱不

要使用大寫字母。下麵是正確命名的欄位的示例。

class SampleClass

{

        string url;

  string destinationUrl;

}

            4、不要對欄位名使用匈牙利語標記法。好的名稱描述語義，而非類型。

5、不要對欄位名或靜態欄位名應用首碼。具體說來，不要對欄位名稱應用首碼來區分靜態和非靜態欄位。例如，應用 g\_ 或 s\_ 首碼是不正確的。

6、對預定義物件實例使用公共靜態唯讀欄位。如果存在物件的預定義實例，則將它們聲明為

物件本身的公共靜態唯讀欄位。使用 Pascal 大小寫，原因是欄位是公共的。下面的代碼

示例闡釋公共靜態唯讀欄位的正確使用。

                  public struct Color

                  {

                      public static readonly Color Red = new Color(0x0000FF);

                      public Color(int rgb)

{

 // Insert code here.}

                          public Color(byte r, byte g, byte b)

                          {

// Insert code here.

                           }

                      public byte RedValue

                      {

                           get

                           {

                                return Color;

                           }

                       }

                   }

### 5-15靜態欄位

       以下規則概述靜態欄位的命名指南：

           1、使用名詞、名詞短語或者名詞的縮寫命名靜態欄位。

           2、使用 Pascal 大小寫。

           3、對靜態欄位名稱使用匈牙利語標記法首碼。

           4、建議盡可能使用靜態屬性而不是公共靜態欄位。

### 5-16集合

集合是一組組合在一起的類似的類型化對象，如雜湊表、查詢、堆疊、字典和清單，集合的命名

建議用複數。

### 5-17措詞

避免使用與常用的 .NET 框架命名空間重複的類名稱。例如，不要將以下任何名稱用作類名稱：

System、Collections、Forms 或 UI。有關 .NET 框架命名空間的列表，請參閱類庫。

另外，避免使用和以下關鍵字衝突的識別字。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| AddHandler | AddressOf | Alias | And | Ansi |
| As | Assembly | Auto | Base | Boolean |
| ByRef | Byte | ByVal | Call | Case |
| Catch | CBool | CByte | Cchar | CDate |
| CDec | CDbl | Char | Cint | Class |
| CLng | CObj | Const | Cshort | CSng |
| CStr | CType | Date | Decimal | Declare |
| Default | Delegate | Dim | Do | Double |
| Each | Else | ElseIf | End | Enum |
| Erase | Error | Event | Exit | ExternalSource |
| False | Finalize | Finally | Float | For |
| Friend | Function | Get | GetType | Goto |
| Handles | If | Implements | Imports | In |
| Inherits | Integer | Interface | Is | Let |
| Lib | Like | Long | Loop | Me |
| Mod | Module | MustInherit | MustOverride | MyBase |
| MyClass | Namespace | New | Next | Not |
| Nothing | NotInheritable | NotOverridable | Object | On |
| Option | Optional | Or | Overloads | Overridable |
| Overrides | ParamArray | Preserve | Private | Property |
| Protected | Public | RaiseEvent | ReadOnly | ReDim |
| Region | REM | RemoveHandler | Resume | Return |
| Select | Set | Shadows | Shared | Short |
| Single | Static | Step | Stop | String |
| Structure | Sub | SyncLock | Then | Throw |
| To | True | Try | TypeOf | Unicode |
| Until | volatile | When | While | With |
| WithEvents | WriteOnly | Xor | Eval | extends |
| instanceof | package | var |  |  |

## 6.     語句

### 6-1行一個語句

         每行最多包含一個語句。如

             a++;       //推薦

             b--;       //推薦

a++; b--; //不推薦

### 6-2複合陳述式

複合陳述式是指包含"父語句{子語句;子語句;}"的語句，使用複合陳述式應遵循以下幾點

            1 子語句要縮進。

2 左花括弧“{” 在複合陳述式父語句的下一行並與之對齊，單獨成行。

3 即使只有一條子語句要不要省略花括弧“ {}”。 如

                 while (d + =  s++)

                  {

                      n++;

                    }

### 6-3 return 語句

        return語句中不使用括弧，除非它能使返回值更加清晰。如

              return;

              return myDisk.size();

              return (size ? size : defaultSize);

## 6-4 if、 if-else、if else-if 語句

        if、 if-else、if else-if 語句使用格式

            if (condition)

            {

                statements;

            }

            if (condition)

            {

                statements;

            }

            else

            {

                statements;

            }

            if (condition)

            {

                statements;

            }

            else if (condition)

            {

                statements;

            }

            else

            {

                statements;

            }

### 6-5 for、foreach 語句

        for 語句使用格式

            for (initialization; condition; update)

            {

                statements;

            }

       空的 for 語句（所有的操作都在initialization、condition 或 update中實現）使用格式

              for (initialization; condition; update);    // update user id

       foreach 語句使用格式

              foreach (object obj in array)

              {

                  statements;

}

**注意**1在迴圈過程中不要修改迴圈計數器。

           2對每個空循環體給出確認性注釋。

### 6-6 while 語句

        while 語句使用格式

            while (condition)

            {

                statements;

            }

         空的 while 語句使用格式

              while (condition);

## 6-7 do - while 語句

         do - while 語句使用格式

              do

              {

                  statements;

              } while (condition);

### 6-8 switch - case 語句

         switch - case 語句使用格式

              switch (condition)

              {

                     case 1:

                         statements;

                         break;

                     case 2:

                         statements;

                         break;

                     default:

                         statements;

                         break;

                 }

           注意：

               1、語句switch中的每個case各占一行。

               2、語句switch中的case按字母順序排列。

               3、為所有switch語句提供default分支。

               4、所有的非空 case 語句必須用 break; 語句結束。

### 6-9 try - catch 語句

          try - catch 語句使用格式

              try

              {

                  statements;

              }

              catch (ExceptionClass e)

              {

                  statements;

              }

              finally

              {

                statements;

              }

### 6-10 using 塊語句

         using 塊語句使用格式

             using (object)

             {

                 statements;

             }

### 6-11 goto 語句

 goto 語句使用格式

             goto Label1:

                 statements;

              Lable1:

                  statements;

1.用Pascal規則來命名方法和類型。  
public class DataGrid  
{  
public void DataBind()  
{

}  
}

2.用Camel規則來命名區域變數和方法的參數.   
public class Product  
{  
private string \_productId;  
private string \_productName;

public void AddProduct(string productId,string productName)  
{

}  
}

3.所有的成員變數前加首碼“\_”。  
public class DataBase  
{  
private string \_connectionString;  
}

4.介面的名稱加首碼 “I”。  
public interface IConvertible  
{  
byte ToByte();  
}

5.自訂的屬性以“Attribute”結尾。  
public class TableAttribute:Attribute  
{

}

6.自訂的異常以Exception結尾。  
public class NullEmptyException:Exception  
{

}

7.方法的命名。一般將其命名為動賓短語。  
public class File  
{  
public void CreateFile(string filePath)  
{

}  
public void GetPath(string path)  
{

}  
}

8．區域變數的名稱要有意義。  
不要用x，y，z等等，用For迴圈變數中可使用i, j, k, l, m, n。  
public class User  
{  
public void GetUser()  
{  
string[] userIds={"ziv","zorywa","zlh"};

for(int i=0,k=userIds.Length;i<k;i++)  
{

}  
}  
}

9．所有的成員變數聲明在類的頂端，用一個換行把它和方法分開。  
public class Product  
{  
private string \_productId;  
private string \_productName;

public void AddProduct(string productId,string productName)  
{

}  
}

10．用有意義的名字命名namespace，如：公司名、產品名。  
namespace Zivsoft//公司命名  
{

}  
namespace ERP//產品命名  
{

}

11．建議區域變數在最接近使用它時再聲明。

12．使用某個控制項的值時，儘量命名區域變數。  
public string GetTitle()  
{  
string title=lbl\_Title.Text;  
return title;  
}

14.把引用的系統的namespace和自訂或協力廠商的用一個換行把它們分開。   
using System;  
using System.Web.UI;  
using System.Windows.Forms;

using CSharpCode;  
using CSharpCode.Style;

15.檔案名要能反應類的內容，最好是和類同名，一個檔中一個類或一組關連類。

16.目錄結構中要反應出namespace的層次。

17.大括弧"{"要新起一行。  
public Sample()  
{  
//  
// TODO: 在此處添加構造函數邏輯  
//  
}

|  |  |  |
| --- | --- | --- |
| 資料類型 | 資料類型簡寫 | 標準命名舉例 |
| Array | arr | arrShoppingList |
| Boolean | bln | blnIsPostBack |
| Byte | byt | bytPixelValue |
| Char | chr | chrDelimiter |
| DateTime | dtm | dtmStartDate |
| Decimal | dec | decAverageHeight |
| Double | dbl | dblSizeofUniverse |
| Integer | int | intRowCounter |
| Long | lng | lngBillGatesIncome |
| Object | obj | objReturnValue |
| Short | shr | shrAverage |
| Single | sng | sngMaximum |
| String | str | strFirstName |

|  |  |  |
| --- | --- | --- |
| **控制項類型** | **縮寫** | **示例** |
| Animated button | ani | aniMailBox |
| Button | btn |  |
| Check box | chk | chkReadOnly |
| ColorDialog | clrdlg |  |
| Combobox,drop-down list box | cbo | cboEnglish |
| Communications | com | comFax |
| ContextMenu | ctxmnu |  |
| Control(used within procedures when the specific type is unknown) | ctr | ctrCurrent |
| CrystalReportViewer | crvw |  |
| Data | dat | datBiblio |
| Data grid | dgd | dgdTitles |
| Data list | dbl | dblPublisher |
| Data repeater | drp | drpLocation |
| Data-bound combo box | dbcbo | dbcboLanguage |
| Data-bound grid | dbgrd | dbgrdQueryResult |
| Data-bound list box | dblst | dblstJobType |
| Datetime picker | dtp | dtpPublished |
| Directory list box | dir | dirSource |
| DomainUpDown | dupd |  |
| Drive list box | drv | drvTarget |
| ErrorProvider | err |  |
| File list box | fil | filSource |
| Flat scroll bar | fsb | fsbMove |
| FontDialog | fntdlg |  |
| Form | frm | frmEntry |
| Frame | fra | fraLanguage |
| Gauge | gau | gauStatus |
| Graph | gra | graRevenue |
| Grid | grd | grdPrices |
| GroupBox | grp |  |
| HelpProvider | hlp |  |
| Hierarchical flexgrid | flex | flexOrders |
| HScroll bar | hsb | hsbVolume |
| Image | img | imgIcon |
| Image combo | imgcbo | imgcboProduct |
| ImageList | ils | ilsAllIcons |
| Label | lbl | lblHelpMessage |
| Line | lin | linVertical |
| LinkLabel | lnk |  |
| List box | lst | lstPolicyCodes |
| ListView | lvw | lvwHeadings |
| MAPI message | mpm | mpmSentMessage |
| MAPI session | mps | mpsSession |
| MCI | mci | mciVideo |
| Menu | mnu | mnuFileOpen |
| Month view | mvw | mvwPeriod |
| MonthCalendar | mcl |  |
| MS Chart | ch | chSalesbyRegion |
| MS Flex grid | msg | msgClients |
| MS Tab | mst | mstFirst |
| NotifyIcon | nti |  |
| NumericUpDown | nupd |  |
| OLE container | ole | oleWorksheet |
| OpenFileDialog | ofdlg |  |
| PageSetUpDialog | psdlg |  |
| Picture box | pic | picVGA |
| Picture clip | clp | clpToolbar |
| PrintDocument | prndoc |  |
| PrintPreviewControl | ppctl |  |
| PrintPreviewDialog | ppdlg |  |
| ProgressBar | prg | prgLoadFile |
| RadioButton | rbtn |  |
| Remote Data | rd | rdTitles |
| RichTextBox | rtf | rtfReport |
| SaveFileDialog | sfdlg |  |
| Shape | shp | shpCircle |
| Slider | sld | sldScale |
| Spin | spn | spnPages |
| Splitter | spt |  |
| StatusBar | sta | staDateTime |
| SysInfo | sys | sysMonitor |
| TabContrl | tab |  |
| TabStrip | tab | tabOptions |
| Text box | txt | txtLastName |
| Timer | tmr | tmrAlarm |
| Toolbar | tlb | tlbActions |
| TrackBar | trb |  |
| TreeView | tre | treOrganization |
| UpDown | upd | updDirection |
| VScroll bar | vsb | vsbRate |

分類: [asp.net](http://www.cnblogs.com/netshuai/category/106079.html)

1.方法、類型用 pascal大寫規則來命名. public class TextBox { public void DataBind() { } }   
2.區域變數、方法的參數 用camel首單詞字母小寫規則來命名. string userName; public AddUser(string userId, byte[] password);   
3.成員變數前加首碼 m\_ public class Database { public string m\_connectionString; }   
4.介面的名稱加首碼 I. interface ICompare { int compare(); }   
5.自訂的屬性以Attribute結尾 public class AuthorAttribute : Attribute { }   
6.自訂的異常以Exception結尾 public class AppException : Exception { }   
7.方法的命名.一般將其命名為[動賓短語](https://www.baidu.com/s?wd=%E5%8A%A8%E5%AE%BE%E7%9F%AD%E8%AF%AD&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLuyPBry7buWu-nvu9n1640ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En10YP1b4rjnYn1f4PHnzrj0Y). ShowDialog() CreateFile() GetPath()   
8.代碼的縮進.要用Tab，而不要用space.   
9.區域變數的名稱要有意義.不要用x，y，z等等. string userName   
10.所有的成員變數聲明在類的頂端，用一個換行把它和方法分開.   
11.用有意義的名字命名namespace，如：產品名、公司名.   
12.建議區域變數在最接近使用它時再聲明.   
13.使用某個控制項的值時，儘量命名區域變數.   
14.把引用的系統的namespace和自訂或協力廠商的分開.   
15.檔案名要能反應類的內容，最好是和類同名，一個檔中一個類.   
16.[目錄結構](https://www.baidu.com/s?wd=%E7%9B%AE%E5%BD%95%E7%BB%93%E6%9E%84&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLuyPBry7buWu-nvu9n1640ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En10YP1b4rjnYn1f4PHnzrj0Y)中要反應出namespace的層次.   
17.大括弧"{"要新起一行. public class AuthorAttribute : Attribute { }