第6章　代码标准文档

本章内容：

- 一般的源代码格式规则
- Object Pascal
- 文件
- 窗体与数据模块
- 包
- 组件
- 代码标准文档升级

本章描述书中使用的Delphi编码标准。一般来说，本章主要是为开发组提供一个方法，使他们在编程时有一致格式可遵循。这样，开发组中每个编程人员编写的代码能够被其他人理解。这要求大家使用一致的代码样式。

本章不可能包括代码标准的每个细节。但是，足以让您能够开始工作。你可以自由地使用和修改这些标准，以满足不同的需要。不过，我们并不建议你在这些标准上花太多的时间。我们之所以介绍这些标准，是因为当新的开发人员加入开发组时，他们可能已经对 Borland的标准很熟悉。与大多数代码标准文档一样，你可以在下面网址找到最新版本的帮助：

www.xapware.com/ddg

本章不包含用户界面标准。用户界面标准是独立于其他标准的，并且同样是重要的。大量的书籍和Microsoft文档包含有关这个标准的说明，因此这里不再赘述。建议你到Microsoft Developers Network和其他可能的地方去查看一下这些信息。

6.1　一般的源代码格式规则

6.1.1　缩进

缩进就是每级间有两个空格。不要在源代码中放置制表符。这是因为，制表符的宽度随着不同的设置和代码管理实用程序(打印、文档及版本控制等)而不同。

通过使用Tools | Emvironment菜单，在Environment Options对话框的General页上，不要选中Use Tab Character和Optimal Fill复选框，这样，制表符就不会被保存。

6.1.2　边距

边距设置为80个字符。源代码一般不会因写一个单词而超过边距，但本规则比较灵活。只要可能，长度超过一行的语句应当用逗号或运算符换行。换行后，应缩进两个字符。

6.1.3　begin...end

begin语句必须单独占一行。例如，下面第一行是错误的，而第二行正确：

```
for i:=0 to 10 do begin  // 错, begin与for在同一行
for i:=0 to 10 do        // 对, begin在另外一行中
begin
```

本规则的一个特殊情况是，当begin为else语句的一部分时，例如：

```
if some statement=then
begin
...
end
else begin
    SomeOtherStatement;
end;
```

end语句总单独一行。

当begin不为else语句的一部分时，相应的end语句与begin语句的缩进量相同。

6.2 Object Pascal

6.2.1 括号

在左括号与下一字符之间没有空格。同样，右括号与前一字符也没有空格。下面的例子演示了正确与不正确的空格。

```
CallProc( Aparameter );  // 错!
CallProc(Aparameter);    // 正确!
```

不要在语句中包含多余的括号。在源代码中，括号只有在确实需要时才使用。下面的例子演示了正确与不正确用法：

```
if (I=42) then              // 错，括号是多余的
if (I=42) or (J=42) then    // 正确，必须使用括号
```

6.2.2 保留字和关键字

Object Pascal语言的保留字和关键字总是完全的小写。

6.2.3 过程和函数

1. 命名与格式

例程名应当以大写字母开始，且大小写交错以增加可读性。下面是一个不正确的写法：

```
procedure thisisapoorlyformattedroutinename;
```

改成这样写就对了：

```
procedure ThisIsMuchMoreReadableRoutineName;
```

例程名应当有意义。进行一个动作的例程最好在名称前加上表示动作的动词为前缀。例如：

```
procedure FormatHardDrive;
```

设置输入参数值的例程名应当以Set为其前缀，例如：

```
procedure SetUserName;
```

获取数值的例程名应当以Get为其前缀，例如：

```
function GetUserName:string;
```

2. 形参

(1) 格式

只要可能，同一类型的形参应当归并在一起：

```
procedure Foo(Param1,Param2,Param3:Imteger;Param4:string);
```

(2) 命名

所有形参的名称都应当表达出它的用途。如果合适的话，形参的名称最好以字母 A为前缀，例如：

```
procedure SomeProc(AUserName:string; AUserAge:integer);
```

当参数名与类的特性或字段同名时，前缀A就有必要了。

(3) 参数顺序

形参的顺序主要要考虑寄存器调用规则。

最常用的参数应当作为第一个参数，按使用频率依次从左到右排。

输入参数位于输出参数之前。

范围大的参数应当放在范围小的参数之前。例如：

```
SomeProc(APlanet,AContinent,ACountry,AState,ACity).
```

有些则例外。例如，在事件处理过程中， TObject类型的Sender参数往往是第一个要传递的参数。

(4) 常量参数

要使记录、数组、短字符串或接口类型的参数不能被例程修改，就应当把形参标以 Const。这样，编译器将以最有效的方式生成代码，保证传递的参数不可变。

如果其他类型的参数希望不被例程所修改，也可以标上 Const。尽管这对效率没有影响，但这给例程的调用者带来了更多的信息。

(5) 命名冲突

当两个单元中含有相同名称的例程时，如果调用该例程，实际被调用的是 Uses 子句中较后出现的那个单元中的例程。为避免这种情况，可在方法名前加想要的单元名，例如：

```
SysUtils.FindClose(SR);
```

或Windows.FindClosr(Handle);

6.2.4 变量

1. 变量的命名与格式

变量的名称应当能够表达出它的用途。

循环控制变量常常为单个字母，诸如 I、J或K。也可以使用更有意义的名称，例如 UserIndex。布尔变量名必须能清楚表示出True和False值的意义。

2. 局部变量

局部变量用于例程内部，遵循其他变量的命名规则。

如果需要的话，应当在例程的入口处立即初始化变量。局部的 AnsiString类型的变量自动被初始化为空字符串，局部的接口和 dispinterface类型的变量自动被初始化为 nil，局部的Variant和OleVariant类型的变量自动被初始化为Unassigned。

3. 全局变量

一般不鼓励使用全局变量。不过，有时候需要用到。即使如此，也应当把全局变量限制在需要的环境中。例如，一个全局变量可能只在单元的实现部分是全局的。

全局数据如果将由许多单元使用，就应移动到一个公用单元里被所有对象使用。

全局数据可在声明时直接初始化为一个值。注意，所有全局变量自动进行零初始化，因此，不要将全局变量初始化为诸如0、nil、或Unassigned等空值。零初始化的全局变量在 .EXE文件中不占空间。

零初始化的数据保存在虚拟的数据段中，而虚拟数据段只在应用程序启动时才分配内存。非零初始化的全局数据则在.EXE文件中占空间。

6.2.5 类型

1. 大小写规则

类型标识符是保留字，应当全部小写。Win32 API类型常常全部大写，并且遵循诸如Windows.pas或其他API单元中关于特定类型名的规则。对于其他变量名，第一个字母应大写，其他字母则大小写交错。下面是一些例子：

```
var
  MyString:string;          // 保留字
  WindowsHandle:HWND;       //Win32 API 类型
  I:Integer;                //在System单元中引入的类型标识
```

2. 浮点型

不鼓励使用Real类型，因为它只是为了与老的 Pascal代码兼容而保留的。通常情况下，对于浮点数应当使用Double。Double可被处理器优化，是IEEE定义的标准的数据格式。当需要比 Double提供的范围更大时，可以使用Extend。Extend是intel专用的类型，Java不支持。当浮点变量的物理字节数很重要时(可能使用其他语言编写DLL)，则应当使用Single。

3. 枚举型

枚举类型名必须代表枚举的用途。名称前要加 T字符作为前缀，表示这是个数据类型。枚举类型的标识符列表的前缀应包含2~3个小写字符，来彼此关联。例如：

```
TSongType=(stRock,stClassical,stCountry,stAlternative,stHeavyMetal,stRB);
```

枚举类型的变量实例的名称与类型相同，但没有前缀 T，也可以给变量一个更加特殊名称，诸如FavoriteSongTypel、FavoriteSongType2等等。

4. Variant和OleVariant

一般不建议使用Variant和OleVariant。但是，当数据类型只有在运行期才知道时 (常常是在COM和数据库应用的程序中)，这两个类型对编程就有必要。当进行诸如自动化 ActiveX控件的COM编程时，应当使用OleVariant；而对于非COM编程，则应当使用 Variant。这是因为，Variant能够有效地保存Delphi的原生字符串，而OleVariant则将所有字符串转换为OLE字符串(即WideChar字符串)，且没有引用计数功能。

6.2.6 构造类型

1. 数组类型

数组类型名应表达出该数组的用途。类型名必须加字母 T为前缀。如果要声明一个指向数组类型的指针，则必须加字母P为前缀，且声明在类型声明之前。例如：

```
type
  PCycleArray=^TCycleArray;
  TCycleArray=array[1..100] of integer;
```

实际上，数组类型的变量实例与类型名称相同，但没有 T前缀。

2. 记录类型

记录类型名应表达出记录的用途。类型名必须加字母 T为前缀。如果要声明一个指向记录类型的指计，则必须加字母P为前缀，且其声明在类型声明之前。例如：

```
type
    PEmployee=^TEmployee;
    TEmployee=record
          EmployeeName:string;
          EmployeeRate:Double;
```

6.2.7 语句

1. If语句

在if/then/else语句中，最有可能执行的情况应放在then子句中，不太可能的情况放在else子句中。

为了避免出现许多if语句，可以使用case语句代替。

如果多于5级，不要使用if语句。请改用更清楚的方法。

不要在if语句中使用多余的括号。

如果在if语句中有多个条件要测试，应按照计算的复杂程度从右向左排。这样，可以使代码充分利用编译器的短路估算逻辑。例如，如果Condition1比 Condition2快，Condition2比Condition3快，则if语句应这样构造：

```
if Conditior1 and Condition2 and Condition3 then
```

2. case语句

(1) 概述

case语句中每种情况的常量应当按数字或字母的顺序排列。

每种情况的动作语句应当简短且通常不超过 4~5行代码。如果动作太复杂，应将代码单独放在一个过程或函数中。

Case语句的else子句只用于默认情况或错误检测。

(2) 格式

case语句遵循一般的缩进和命名规则。

3. while语句

建议不要使用Exit过程来退出while循环。如果需要的话，应当使用循环条件退出循环。

所有对while循环进行初始化的代码应当位于while入口前，且不要被无关的语句隔开。

任何业务的辅助工作都应在循环后立即进行。

4. for语句

如果循环次数是确定的，应当用for语句代替while语句。

5. repeat语句

repeat语句类似于while循环，且遵循同样的规则。

6. with语句

(1) 概述

with语句应小心使用。要避免过度使用with语句，尤其是在with语句中使用多个对象或记录。例如：

```
with Record1,Record2 do
```

这些情况很容易迷惑编程人员，且导致调试困难。

(2) 格式

with语句也遵循本章关于命名和缩进的规则。

6.2.8 结构化异常处理

1. 概述

异常处理主要用于纠正错误和保护资源。这意味着，凡是分配资源的地方，都必须使用 try...finally 来保证资源得到释放。不过，如果是在单元的初始 /结束部分或者对象的构造器 /析构器中来分配 /释放资源则例外。

2. try...finally的用法

在可能的情况下，每个资源分配应当与try...finally结构匹配，例如，下面代码可能导致错误：

```
SomeClass1:=TSomeClass.Create;
SomeClass2:=TSomeClass.Create;
try
    { do some code }
finally
    SomeClass1.Free;
    SomeClass2.Free;
end;
```

上述资源分配的一个安全方案是：

```
SomeClass1:=TSomeClass.Create;
try
    SomeClass2:=TSomeClass.Create;
    try
        { do some code }
    finally
        SomeClass2.Free;
    end;
finally
    SomeClass1.Free;
end;
```

3. try...except的用法

如果你希望在发生异常时执行一些任务，可以使用 try...except。通常，没有必要为了简单地显示一个错误信息而使用try...except，因为Application对象能够自动根据上下文做到这一点。如果要在子句中激活默认的异常处理，可以再次触发异常。

4. try...except...else的用法

不鼓励使用带else子句的try...except，因为这将阻塞所有的异常，包括你没有准备处理的异常。

6.2.9 类

1. 命名与格式

类的名称应当表达出类的用途。类名前要加字母T，表示它是一个类型。例如：

```
type
    TCustomer=class(TObject);
```

类的实例名称与类名相同，只不过没有前缀T。

```
var
    Customer:TCustomer;
```

注意 关于组件的命名，请参阅6.6节“组件”。

2. 字段

(1) 命名与格式

字段的命名遵循与变量相同的规则，只不过要加前缀F，表示这是字段。

(2) 可见性

所有字段必须为私有。如果要在类的作用域之外访问字段，可借助于类的属性来实现。

3. 方法

(1) 命名与格式

方法的命名遵循与过程和函数相同的规则。

(2) 静态方法

当你不希望一个方法被派生类覆盖时，应当使用静态方法。

(3) 虚拟方法与动态方法

当你希望一个方法能被派生类覆盖，应当使用虚拟方法。如果类的方法要被多个派生类直接或间接地使用，则应当用动态方法。例如，某一个类含有一个被频繁覆盖的方法，并有 100个派生类，则应将方法定义为动态的，这样可以减少内存的开销。

(4) 抽象方法

如果一个类要创建实例，则不要使用抽象方法。抽象方法只能在那些从不创建实例的基类中使用。

(5) 属性访问方法

所有属性访问方法应当定义在类的私有或保护部分。

属性访问方法遵循与过程和函数相同的规则。用于读的方法应当加 Get前缀，用于写的方法应当加Set前缀，并且有一个叫Value的参数，其类型与属性的类型相同。例如：

```
TSomeClass=class(TObject)
private
    FSomeField:Integer;
protected
    function GetSomeField:Integer;
    procedure SetSomeField(Value:Integer);
public
    property SomeField:Integer GetSomeField write SetSomeField;
end;
```

4. 属性

属性作为私有字段的访问器，遵循与字段相同的命名规则，只不过没有 F前缀。

属性名应为名词，而不是动词。属性是数据，而方法是动作。

数组属性名应当是复数，而一般的属性应当是单数。

5. 访问方法的使用

尽管不是必须，但还是建议你使用写访问方法来访问代表私有字段属性。

6.3 文件

6.3.1 项目文件

项目文件的名称应当具有描述意义。例如，“The Delphi 5 Developer’s Guide Bug Manager”的项目名称为DDGBugs.dpr，一个系统信息程序的名称为SysInfo.dpr。

6.3.2 窗体文件

窗体文件的名称应当表达出窗体的用途，且具有 Frm后缀。例如，About窗体的文件名叫 About-

Frm.dpr，主窗体的文件名叫MainFrm.dpr。

6.3.3 数据模块文件

数据模块文件的名称应当表达出数据模块的作用，且具有 DM后缀。例如，Customers数据模块的文件名叫CustomersDM.dfm。

6.3.4 远程数据模块文件

远程数据模块文件的名称应当表达出远程数据模块的用途。名称后要加 RDM后缀。例如，Customers远程数据模块的文件叫CustomersRDM.dfm。

6.3.5 单元文件

1. 普通单元的结构

(1) 单元名

单元的名称应当有描述性。例如，应用程序的主窗体单元叫 MaimFrm.pas。

(2) Uses句子

Interface部分的Uses子句应当只包含该部分需要的单元。不要包含可能由 Delphi自动添加的单元名。

Implementation部分的Uses子句应当只包含该部分需要的单元，不要有多余的单元。

(3) Interface部分

Interface部分应当只包含需要被外部单元访问的类型、变量、过程与函数的声明。而且，这些声明应当在Implementation部分之前。

(4) Implementation部分

Implementation部分包括本单元私有的类型、变量、过程与函数的实现。

(5) Initialization部分

不要在Initialization部分放置花费时间很多的代码。否则，将导致应用程序启动时显得很慢。

(6) Finalization部分

确保释放所有在Initialization部分中分配的资源。

2. 窗体单元

窗体单元文件的名称与相应的窗体名称相同。例如， About窗体的单元名称叫AboutFrm.pas。主窗体的单元文件名称叫 AboutFrm.pas。

3. 数据模块单元

数据模块单元文件的名称与相应的数据模块名称相同。例如，数据模块单元的名称叫 Customers-DM.pas。

4. 通用的单元

通用单元的名称应当表达出它的用途。例如，一个实用工具单元的名称叫 ugUtilities.pas，包含全局变量的单元名称叫CustomerGlobals.pas。

注意，一个项目中单元名称必须是唯一的。通用单元名不能重名。

5. 组件单元

组件单元应放在单独的路径中，以表明它们是定义组件的单元。它们一般与项目不放在同一路径下。单元文件名称应表达出其内容。

注意，有关组件命名标准的更多信息，请参阅6.6.1节“ 自定义组件 ”。

6.3.6 文件头

所有源文件和项目文件都应具有文件头。一个正确的文件头应包含以下信息：

```
{
    Copyright @ YEAR by AUTHORS
}
```

6.4 窗体与数据模块

6.4.1 窗体

1. 窗体类型的命名标准

窗体类型的名称应当表达出窗体的用途，且要加T前缀,后跟描述性名，最后是Form。例如，About窗体类型名称为：

```
TAboutForm=class(TForm)
```

主窗体的类型名称为:

```
TMainForm=class(TForm)
```

客户登录窗体的类型名称为:

```
TCustomerEntryForm=class(TForm)
```

2. 窗体实例的命名标准

窗体实例的名称与相应的类型名称相同，但没有前缀 T。例如，前面提到的窗体类型与实例的名称为：

类 型 名	实 例 名
TAboutForm	AboutForm
TMainForm	MainForm
TCustomerEntryForm	CustomerEntryForm

3. 自动创建的窗体

除非特别原因,只有主窗体才自动生成。其他所有窗体必须从 Project Options对话框的自动生成列表中删除。更进一步信息,请参阅后面几节。

4. 模式窗体实例化函数

所有窗体单元都应当含有实例化函数，用于创建、设置、模式显示和释放窗体。这个函数将返回由窗体返回的模式结果。传递给这个函数的参数遵循参数传递的规则。之所以要这样封装，是为了便于代码的重用和维护。

窗体的变量应当从单元中移走，改在窗体实例化函数中作为局部变量定义 (注意，要求从Project Options对话框的自动生成列表中移走该窗体。请看前面的内容)。

例如，下面的单元文件演示了GetUserData的实例化函数。

```
Unit UserDataFrm;
Interface
Uses
    Windows,Messages,SysUtils,Classes,Graphics,Controls,Forms,
    Dialogs,StdCtrls;
Type
    TUserDataForm=class(TForm)
        ektUserName:TEdit;
```

```
        edtUserID:TEdit;
    private
        { Private declarations }
    public
        { Public declarations }
    end;
function GetUserData(var aUserName:String;var aUserID:Integer):Word;
implementation
{$R*.DFM}
function GetUserData(var aUserName:String;var aUserID:Integer):Word;
var
    UserDataForm:TuserDataForm;
Begin
    UserDataForm:=TuserDataForm.Create(Application);
    Try
        UserDataForm.Caption:='Getting User Data' ;
        Result:=UserDataForm.ShowModal;
        If(Result=mrOK)then begin
            AUserName:=UserDataForm.edtUserName.Text;
            AUserID:=StrToInt(UserDataForm.edtUserID.Text);
        End;
    Finally
        UserDataForm.Free;
    End;
End;
End.
```

6.4.2 数据模块

1. 数据模块的命名标准

数据模块类型名称应表达出它的用途，且要加前缀 T，后跟描述性名称，最后是 DataMadule。例如，Customer数据模块的类型名称为：

TCustomerDataModule=class(TDataModule)

Orders数据模块的类型名称为：

TOrderDataModule=class(TDataModule)

2. 数据模块实例的命名标准

数据模块实例的名称应当与相应的类型名称相同，但没有前缀 T。例如，前面的数据模块类型、实例名称如下：

类型名称	实例名
TCustomerDataModule	CustomerDataModule
TOrderDataModule	OrderDataModule

6.5 包

6.5.1 运行期包与设计期包

运行期包中应当只包含所需要的单元。那些属性编辑器和组件编辑器的单元应当放在设计期包中。

注册单元也应当放在设计期包中。

6.5.2 文件命名标准

包的命名遵循下列模式：

iiiLibvv.pkg —— 设计期包

iiiStdvv.pkg —— 运行期包

其中，iii代表一个3字符的前缀，用于标识公司、个人或其他需要标识的事情。vv代表包的版本号，其中也包含了Delphi的版本号。

注意 包名称中的lib或std分别表示这是设计期包还是运行期包。例如，本书中的包是这样命名的：

DdgLib50.pkg —— 设计期包

DdgStd50.pkg —— 运行期包

6.6 组件

6.6.1 自定义组件

1. 组件类型的命名标准

组件的命名与类的命名类似，只不过它有 3个字符的前缀。这些前缀用以标识公司、个人或其他实体。例如，一个时钟组件可以这样声明：

```
TDdgClock=class(TComponent)
```

注意，作为前缀的3个字符要小写。

2. 组件单元

组件单元只能含有一个主要组件，这是指出现在组件选项板上的组件。其他辅助性的组件或对象也可以包含在同一单元中。

3. 注册单元

组件的注册过程应当从组件单元中移走，放在一个单独的单元中。这个注册单元用于注册所有组件、属性编辑器、组件编辑器、向导等。

组件注册应当在设计期包中进行。因此，注册单元应当包含在设计期包而不是运行期包中。

建议注册单元这样命名：

```
XxxReg.pas
```

其中，Xxx为3个字符前缀，以标识公司、个人或其他实体。例如，本书中的注册单元命名为DdgReg.pas。

6.6.2 组件实例的命名规则

组件的名称应当具有描述性。 Delphi没有为组件指定默认的名称。单元命名要使用一个变更了的匈牙利命名规范。在这个标准中，组件名包括两个部分：前缀和性质标识名。

1. 组件的前缀

组件的前缀多是表现组件类型的字母缩写。例如，下面表中的组件前缀。

TButton	btn
TEdit	edt

TSpeedButton	spdbtn
TIistBoxl	lstbx

如上所示，组件类型前缀是组件类型名变化而成的。下面的规则说明如何定义一个组件类型前缀：

1) 从组件类型名中移去T前缀。例如TButton变成Button。

2) 除了第一个元音，删去所有元音字母。例如，Button变成Bttn，Edit变成Edt。

3) 压缩双字母。例如，Bttn变成Btn。

4) 如发生冲突，则在某一组件前缀中加入一个元音。例如在 TBatton组件的前缀中加入元音变为batn，以区别TButton的前缀。

2. 组件性质标识名

组件性质标识名是组件意图的描述。例如，一个用于关闭窗体的 TButton组件可命名为BtnClose。一个编辑人名的组件可命名为EdtFirstName。

6.7 代码标准文档升级

这个文档的升级可以反映 Object Pascal语言和VCL的改变和增强。你可以在 http://www.xapware.com/ddg获得升级。