Delphi

部件开发指南

瞿继双 伯晓晨 编著

西南交通大学出版社

Delphi 部件开发指南

瞿继双　伯晓晨　编著

西南交通大学出版社

·成　都·

图书在版编目（CIP）数据

Delphi 部件开发指南/瞿继双，伯晓晨编著．—成都：西南交通大学出社，2000．2
ISBN 7-81057-415-9

Ⅰ.D… Ⅱ.①瞿…②伯… Ⅲ.Delphi 语言－程序设计 Ⅳ.TP312

中国版本图书馆 CIP 数据核字（1999）第 57983 号

Delphi 部 件 开 发 指 南
瞿继双 伯晓晨 编著
*
出 版 人 宋绍南
责任编辑 成 鹏
封面设计 刘 捷
西南交通大学出版社出版发行
（成都二环路北一段 111 号 邮政编码：610031 发行科电话：7600564）
http://press.swjtu.edu.cn
E-mail: cbs@center2.swjtu.edu.cn
成都市报华印装厂印刷
*
开本：787 mm × 1092 mm 1/16 印张：25
字数： 609 千字 印数：1～2000 册
2000 年 1 月第 1 版 2000 年 1 月第 1 次印刷
ISBN 7-81057-415-9 / TP · 186
定价： 55.00 元

前　言

可视化编程诸如 VC、VB、Delphi、C++ Builder 等工具的实现，为用户编写应用程序带来了一次重大革命，其中尤以 Delphi 以其快速的编译速度、丰富的控件箱、强大的数据库处理功能倍受广大用户喜爱。可视化编程的核心就是部件对象。尽管 Delphi 已经提供了极其丰富的控件箱，但相对于现实中各种纷繁复杂的应用，它仍显得渺小。而真正能以不变应万变的就是掌握编写部件的基本方法，从而根据 Delphi 的基本 VCL 类库开发出符合实际要求的部件，这正是可视化编程的灵魂所在。

迄今为止，关于 Delphi 编程的书籍可以说浩如烟海，但其中针对如何开发自定义部件方面的编程专著还没有先例。本书致力于 Delphi 部件开发方法的探讨，希望能对广大 Delphi 用户产生强大吸引力。

本书由九章组成。第一章简单地回顾了 Delphi Object Pascal 编程语言，并分析了 Delphi 对象的基本特征。然后在第二章深入剖析了 Delphi VCL 类的构成，并对 TObject、TWinControl 等基本类库及其属性、方法、事件作了详细剖析。接下来第三章详细阐述了开发自定义部件的基本方法和过程，讲述了如何创建部件及其属性、方法和事件，以及如何注册部件。紧接着第四章对 Delphi 标准界面控件进行了深入分析，并举例分析了如何从 TWinContrl 和 TGraphicControl 及其后代对象继承开发出功能更强大的控件。其后第五章讲述了如何开发属性编辑器，并举例分析了如何创建属性编辑器。对于 Delphi 功能强大的数据库应用程序开发部件，本书的讲述也非常详细和深入，第六章在对数据库主要部件进行详细剖析以后，通过一个部件开发实例阐述了如何开发自定义数据库部件。之后第七章通过一个实例讲述了专业图像处理部件的开发方法，扩充了的 Delphi 中 TImage 部件的功能。对于数值计算方面的编程问题，本书独辟蹊径，在第八章讲述了如何把数值计算问题以部件方式可视地进行。最后第九章讲述了用于工控程序界面中的常用部件开发，从而把 Delphi 的部件开发映射到了各种工业应用中。

当然，本书所涉及的内容是有限的，但本书所讲述的 Delphi 部件开发方法的应用却是极其广泛的，读者可以根据这些方法开发出各种各样的自定义部件。

本书有五个主要特色。其一是对 Delphi VCL 类的基本抽象类 TObject 直至 TWinControl 和 TGraphicControl 作出了详细剖析，因为这些抽象类是创建各种对象的基础，也是开发自定义部件的基础。其二是对书中开发部件时所涉及到对象尤其是抽象基类的属性、方法和事件进行了全面的阐述，并对其中的主要属性、方法和事件进行了详细分析，充分描述了它们的各种功能和用法。属性、方法和事件是部件的本质特征，因此创建它们是开发自定义部件的主要任务。而部件必须从某一个对象（通常是抽象基类）继承而来，继承这些对象的同时也继承了它们的部分或全部属性、方法和事件，因此对这些对象的属性、方法和事件进行分析对于开发自定义部件是非常有用的。其三是在分析 TWinControl 和 TGraphicControl 类以前的 VCL 类时，分析了它们所有的属性、方法和事件，包括从 TObject 等祖先对象继承而

来的，而对于 TWinControl 和 TGraphicControl 类以后的 VCL 类，只分析了它们新创建的以及从 TWinControl 和 TGraphicControl 类以后的对象继承的属性、方法和事件。其四是本书在举例分析创建自定义部件时，一般都在完成部件的创建后利用它们创建应用程序，其目的一方面是向导读者演示这些部件的应用，另一方面用于测试部件。其五是本书列出了举例所开发出部件和部件应用程序的源代码清单，以供广大读者分析。而且代码清单的格式保持了 Delphi IDE 中的代码风格，使得读者阅读代码非常习惯和轻松。

本书的一至六、八章由瞿继双编写，七、九章由伯晓晨编写。在编写过程中得到了西南交通大学出版社及唐晴编辑的鼎力支持和协作，在此表示诚挚的感谢。另外，向给予我们支持和帮助的老师和朋友表示深深的谢意。

本书的读者为具有一定可视化编程知识和 Delphi Object Pascal 编程语言知识的 Delphi 用户，对于利用 Delphi 开发环境开发专业软件的用户尤其有价值。另外，由于编者水平有限，书中错误在所难免，恳请广大读者批评指正。

编者

1999.10

目　录

第一章 Delphi 面向对象编程技术

Delphi 是一种全新的可视化编程环境，它为用户提供了一种方便、快捷地开发 Windows 应用程序的强有力工具。Delphi 使用了大量 Windows 图形用户界面的先进特性和设计思想，采取了完整的面向对象编程技术(OOP：Object-Oriented Programming)，它的编译速度相当快，而且拥有强大的的数据库应用程序开发功能。因此，对于广大的应用程序开发人员来讲，Delphi 能够大大提高编程效率，因而是一种梦寐以求的开发工具。

Delphi 实际上是 Pascal 语言的一种版本，即 Object Pascal 语言，但它与传统的 Pascal、C++ 语言等具有 OOP 性能的程序语言有着天壤之别。传统 OOP 编程语言往往需要程序员编写大量的代码创建对象，使得与用户交互性能较差，而 Object Pascal 语言与 Delphi 内核相结合，使得用户不必自己建立对象，而只需可视地对对象进行操作，Delphi 就自动地为用户创建好对象，同时生成对象的源代码。

1.1 Delphi 的 Object Pascal 参考

Object Pascal 是一种高层编译、强类型的编程语言，它支持结构化和面向对象设计，而且代码易于阅读，编译速度快，能够使用多个有标准部件编程的库单元文件。Object Pascal 还具有支持 Delphi 部件框架和快速应用程序开发的特点。

1.1.1 Delphi 的数据类型、变量和常量

1.1.1.1 数据类型

Delphi 的 Object Pascal 预定义了多种数据类型，包括整型、实型、布尔型、字符型、字符串型、指针型等。

整数数据类型（Integer）的范围是－32 768 到 32 767，每个数据占 2 字节的内存；短整型数据类型（Shortint）的范围是从－128 到 127，每个数据占 1 字节内存；长整型数据类型（Longint）的范围为－2 147 483 648 到 2 147 483 647，每个数据占 4 字节内存；字节（Byte）的范围从 0 到 255，每个数据占 1 字节；字（Word）的范围是从 0 到 65 535，每个数据占 2 字节内存。

单精度数据类型（Single）可以包含 7 到 8 位有效小数部分，每个数据占用 4 字节的内存；双精度数据类型（Double）可以包含 15 到 16 位有效小数部分，每个数据占用 8 字节的内存；扩展实数类型（Extended）包含 19 到 20 位有效小数部分，每个数据占用 10 字节内存；计算类型（Comp）可以包含 19 到 20 位有效小数部分，每个数据占用 8 字节内存。实

数数据类型（Real）可以包含 11 到 12 位有效小数部分，每个数据占用 8 字节内存，它相当于双精度数据类型。

布尔型数据类型（Boolean）只包含 True 或 False 两个值，每个数据占用 1 字节内存。

字符型数据类型（Char）是一个 ASCII 字符，最长可达 255 个 ASCII 字符。

指针型数据类型（Pointer）可以指向任何特定类型，如函数入口地址、数组首元素等。

字符串型数据类型（PChar）是一个指向以零结尾的字符串的指针，它实际上是 String 类型的源类型，其主要用途是提供与 C 和 Windows 的 API 的接口类型。

从上述内容中可以看出，整数类型和实数类型都各有多种类型，同一类型中，任意一种数据类型与其他同类型的数据都兼容。因此用户可以将一种类型的值赋给相同类型中不同的变量或属性，其基本要求就是这个值的范围在被赋值的变量或属性的可能值范围内。例如，对于一个 Smallint 型的变量，可以接受在 -32 768 到 32 767 范围内的任意整数，但用户不能将 40 000 赋给它，因为 40 000 已经超出了 Smallint 数据类型的范围了。

用户可以通过选用 Delphi 的 IDE 中的 Options|Project，在 Compiler Options Page 中选择 Range Checking 以打开范围检查功能，Delphi 将会自动检查数据变量的范围匹配情况。如果 Range Checking 没有被打开，那么程序代码将可以不受影响地执行，不过变量或属性被赋的值将不是用户所期望的值。

1.1.1.2 变量

变量是程序代码中代表一个内存地址的标识符，而此地址的内存内容在程序代码执行时可以被改变。用户可以使用变量以读取或写入内存中的变量值。

在使用变量前必须对它进行声明，同时说明它的类型。变量声明的保留字是 var，声明变量时只需在变量标识符前加上它即可。变量说明左边是变量的名称，右边则是该变量的类型，中间用(:)隔开，如下面所示：

```
var identifier: type = constantExpression;
```

例如：

```
var
  X, Y, Z: Double;
  I, J, K: Integer;
  Digit: 0..9;
  Okay: Boolean;
```

1.1.1.3 常量

常量在声明时就被赋予了一个值，它在代码被编译时就已经给出了值，并且在程序执行过程中是不可改变的。常量表达式可以是整数类型、实数类型、字符类型、布尔类型、枚举值类型；True、False 和 Nil 等特殊的常量值；由操作符连接的以上数据类型组合；集合等等。

常量表达式不能包含变量、指针和一般的函数调用，但可以调用 Abs、Exp、Length 等 Delphi 的预定义函数。

声明常量的语法如下：

```
const identifier: type = value
```

例如：

```
const
  Pi = 3.14159;
  Max: Integer = 100;
```

1.1.2　Delphi 的自定义数据类型

Object Pascal 除了预定义数据类型外，用户还可以自定义各种数据类型，如枚举类型、子界类型、数组类型、集合类型、记录类型、对象类型等等。

1.1.2.1　枚举类型

一个枚举类型的说明列出了所有这种类型可以包括的值。下面给出了 TDays 枚举数据类型的定义：

```
type
  TDays=( Sunday ,Monday,Tuesday,Wednesday,Thursday,Friday,Saturday);
```

在定义了数据类型后，用户就可以通过如下代码定义该类型的变量：

```
var
  WeekDay:TDays;
```

在枚举类型中，括号中的每一个值都有一个由说明它的位置决定的整形值，这相当于一个索引值。例如 Sunday 的整形值为 0，Monday 的整形值为 1，…。用户还可以把 DayOfWeek 声明为一个整形变量，并将一星期的每一天赋一个整形值以达到相同的效果，但毫无疑问，使用枚举类型会使程序具有更好的可读性，因而更容易编写代码。当用户在枚举类型中列出某个值时，同时也说明了这个值是一个标识符。例如用户的程序中如果已经定义了 TDays 类型并且说明了 WeekDay 变量，则程序中便不能使用 Monday 等枚举值为变量，因为它们已经被说明成为标识符了。

1.1.2.2　子界类型

子界类型是在整型、布尔型、字符型或枚举型等数据类型中定义一定范围的值。如果用户想限制一个变量的取值范围时，子界类型则提供了很好的手段。下面给出了一组子界类型声明：

```
type
  TMonth = 1..12;
  TMonths = ( Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec);      {枚举型}
  TSpring = Jan..Mar;      {一个 TMonths 类型的子界类型}
```

子界型限定了变量的可能取值范围。当范围检查打开时（在库单元的 Implementation 后面有{$R*.DFM}字样表示范围检查打开，否则用户可以在 Options|Project|Complier Options 中选择 Range Cheking 来打开范围检查），如果变量取到子界以外的值，Delphi 将会给出一个范围检查错误。

1.1.2.3 数组类型

数组是某种数据类型的有序组合，它其实是一个指针，用以指向储存第一个元素的地址。下面的代码说明了一个 Integer 类型的数组变量：

```
var
  Score: array [0..9] of Integer;
```

它表示 Score 指向一个含有 10 个 Integer 类型元素的数据列的第一个元素 Score[0]的地址。数组中的任一元素可以通过数组下标来访问。

用户还可以先把数组定义成为一个类型：

```
type
  TScore = array [0..9] of Integer;
```

则变量说明变为：

```
var
  Score: TScore;
```

1.1.2.4 字符串类型

字符串类型事实上是一个一维字符数组，其声明过程如下所示：

```
type
  TName: string[20];
var
  MyName: TName;
```

在上面的声明中，TName 数据类型的变量最多可以包含 20 个字符。如果用户没有指定字符串的大小，Delphi 会缺省认为字符串包含的最大值为 255 个字符。

给字符串赋值的格式如下：

```
MyName := 'Robert';      或
MyName := '陈鹏';
```

当用户给字符串类型变量赋的值多于所定义的最大字符数值时，则 Delphi 将截去多余的字符。在内存中，字符串通常比所声明的大小多占用一个字节的空间，因为第一个位置是一个包含这个数组大小的字节。用户可以使用索引值来访问字符串的字符，例如访问 MyName[1]可以得到 MyName 的第一个字符'R'。

1.1.2.5 集合类型

集合类型是一群相同类型元素的组合，这些类型必须是有限类型如整形、布尔型、字符型、枚举型和子界型。当需要检验一个值是否属于一个特定集合时，使用集合类型将非常有用。

集合类型的声明如下：

```
type
  TSomeInts = 1..250;
  TIntSet = set of TSomeInts;
```

该声明过程还可以直接写成：

```
type TIntSet = set of 1..250;
```

表 1.1 列出了集合的操作运算。

表 1.1 集合的操作运算

操作符	运算操作	操作数类型	运算结果类型	示 例
+	加	set	set	Set1 + Set2
-	减	set	set	S - T
*	乘	set	set	S * T
<=	子集	set	Boolean	Q <= MySet
>=	扩展集	set	Boolean	S1 >= S2
=	等于	set	Boolean	S2 = MySet
<>	不等于	set	Boolean	MySet <> S1
in	为…成员	ordinal, set	Boolean	A in Set1

1.1.2.6 记录类型

记录类型可以使用户的程序访问的一组复杂的数据集合，它可以把整数类型、实数类型、字符类型、布尔类型、枚举类型等组织在同一个结构中，从而便于对记录对象的使用和管理。记录中包含可以保存数据的域，每一个域有一个数据类型。

首先声明两个枚举类型：

```
Type
  TSex: ( Male, Female );
  TLevel: ( Undergraduate,Graduate );
```

下面的代码声明了一个记录类型：

```
type
  TStudent = record
    Name : string[20];
    Age: 15..36;
    Sex: TSex;
```

```
    Level: TLevel;
end;
```

在声明了一个记录类型后，就可以定义记录变量了，如下所示：

```
var
  Student: TStudent;
```

用户还可以使用如下方式访问记录的域：

```
Student.Age := 25;
```

1.1.3 Object Pascal 的语句结构

Object Pascal 与其他大多数计算机语言一样，具有顺序结构、分支结构和循环结构。其中分支结构的句法有：

```
if expression then statement1 else statement2
```

以及

```
goto label
```

而循环结构的句法包括 for 循环和 while...do 循环：

```
for counter := initialValue to finalValue do statement 或
for counter := initialValue downto finalValue do statement
while expression do statement
```

下面的代码示例了分支结构和循环结构的应用。

```
procedure TMatrix.MultMatrix(AMatrix: TMatrix);
var
  A: TMatrix;
  i, j, k: Integer;
begin
  if Columns <> AMatrix.Lines
  then raise EMatrixException.Create('Can not multiply these matrixes');
    A := TMatrix.Create(nil);
    A.Lines := Lines;
    A.Columns := AMatrix.Columns;
    for k := 0 to Lines - 1 do
      for i := 0 to AMatrix.Columns - 1 do
        for j := 0 to Columns - 1 do
          A.Cells[k, i] := A.Cells[k, i] + Cells[k, j] * AMatrix.Cells[j, i];
```

```
    Assign(A);
    A.Free;
    Repaint;
end;
```

1.1.4 过程和函数

过程与函数是程序中执行特定工作的模块化部分，它是 Delphi 实现代码共用的重要途径之一。Delphi 的运行库包含许多过程与函数以供用户在创建应用程序时调用。用户甚至不必知道这些过程和函数的逻辑结构，而只需要了解过程与函数的用途就可以运用它们。当然，对于程序员尤其是高级用户而言，尽可能多对自己所使用的对象进行了解是非常有益的。

在对象中声明的过程和函数称为方法(Method)。过程以保留字 procedure 开头，而函数则以保留字 function 开始，分别如下所示：

```
procedure procedureName(parameterList); directives;
localDeclarations;      {局部变量声明}
begin
  statements
end;

function functionName(parameterList): returnType; directives;
localDeclarations;      {局部变量声明}
begin
  statements
end;
```

过程和函数都可以带参数以进行值传递。

例如前面的代码中，

```
procedure TMatrix.MultMatrix(AMatrix: TMatrix);
```

就是一个过程声明。

所有的事件处理过程都是过程，每一个事件处理过程包含了当这一事件发生时需要执行的程序代码。

1.1.5 Object Pascal 的库单元

Delphi 的库单元（Units）是常量、变量、数据类型、过程和函数的集合，而且能够被多个应用程序所共用。Delphi 已经提供了许多预定义的程序库单元以供用户开发应用程序使用。Delphi 把大量的库单元封装入可视化部件库（VCL: Visual Component Library），它们声明了大量的对象，并且实现了对象的纵多功能以供用户在应用程序中使用。

当用户建立一个窗体时，Delphi 将自动创建一个与该窗体相关联的库单元。用户也可以创建与窗体无关的库单元，还可以使用预定义的只包含数学运算函数的库单元，或是自行编写数学函数库单元。

在 Delphi 应用程序开发中，最重要的一点就是用户应该尽可能地把所创建的对象声明为部件对象，并且添加上所需要的注册过程和资源文件，这样就可以使该对象在 Delphi 的 IDE 中可视地出现，并且在窗体上通过对象观测器进行设计，从而充分地利用 IDE 所提供的强大功能。

1.1.5.1 Object Pascal 库单元的结构

无论一个库单元是否和一个窗体有关，库单元的结构都是相同的，其结构如下：

```
unit <库单元名称>
interface
uses <选择性的库单元列表>
{公有说明}
implementation
uses <选择性的库单元列表>
{私有说明}
{过程和函数的执行部分}
initialization {可选择的}
  {选择性的初始化程序}
end.
```

1.1.5.2 库单元的接口部分

interface 是库单元的接口部分，它决定了本库单元对其他任何库单元或程序的可访问部分。用户可以在接口部分说明变量、常量、数据类型、过程和函数等等。Delphi 在用户可视地设计窗体时，自动地生成窗体的数据类型、变量和事件处理过程等声明。

interface 标志库单元接口部分的开始，其中的声明对要使用这些声明的其他库单元或应用程序是可见的。一个库单元如果想使用其他库单元的声明，只需要在 uses 子句中添加上那些库单元即可。

例如，用户在库单元 unit1 中编写了程序代码，并且想使用 unit2 在其 interface 部分的声明，则可以把库单元 unit2 的名称加入到 unit1 的 interface 部分的 uses 子句中，则任何 unit1 中的程序都可以调用 unit2 中声明的程序。而如果 unit2 中 interface 部分的 uses 子句中出现 unit3 库单元，则 unit1 还可以通过调用 unit2 的同时使用 unit3 库单元在 interface 中声明的程序。如果 unit2 出现在 unit1 的 interface 部分的 uses 子句中，那么 unit1 便不能出现在 unit2