发信人: eGust (就像矗立在彩虹之巅), 信区: Delphi
标　题: 自Delphi 7以来的Delphi 2009测试版新语法特性
发信站: 水木社区 (Mon Aug 25 22:26:48 2008), 站内

自Delphi 7以来的Delphi 2009测试版新语法特性
by eGust
==========

New Delphi language features since Delphi 7

这部分对从Delphi 7到Delphi 2007的新语法特性语法进行一个简介，主要内容来自于CodeGear官方网站D2007的 "What's New" 中对新语法特性的介绍的部分:
http://www.codegear.com/products/delphi/win32/whats-new/

1. 内联函数 (Inlining)
D7中的inline关键字作为保留字并不会对编译器产生实际作用，在2009中此关键字起到内嵌到代码中起到实际作用。语法如下:

```
function foo: Integer; inline;
```

内部函数/过程也可以使用，但在D2009测试版中，类方法的内部函数使用inline后不认Self指针；类的子过程/子函数，也可以使用inline关键字，但没有实际效果，且虚方法/继承方法 (virtual/override) 不能使用。

2. 重载运算符 (Operator Overloading)
可以重载部分运算符，如+、-、类型转换等，在D2006只支持到record，但从2007开始支持到Class，以下示例修改自官网:

```
  TMyClass = class
    // Addition of two operands of type TMyClass
    class operator Add(a, b: TMyClass): TMyClass;
    // Subtraction of type TMyClass
    class operator Subtract(a, b: TMyClass): TMyclass;
    // Implicit conversion of an Integer to type TMyClass
    class operator Implicit(a: Integer): TMyClass;
    // Implicit conversion of TMyClass to Integer
    class operator Implicit(a: TMyClass): Integer;
    // Explicit conversion of a Double to TMyClass
    class operator Explicit(a: Double): TMyClass;
  end;

class operator TMyClass.Add(a, b: TMyClass): TMyClass;
begin
  //...
end;

var
  x, y: TMyClass
begin
  x := 12; // Implicit conversion from an Integer
  y := x + x; // Calls TMyClass.Add(a, b: TMyClass): TMyClass
end;
```

3. 类助手 (Class Helpers)
Helper是对原Class的扩展，是我们在不修改原类的基础上增加类方法，并加入原类的空间中。在Delphi中，对对象的调用实际上采用了两个步骤，首先是把对象地址放入eax寄存器中，然后call类方法，所以如果不使用继承类增加数据的话，用父类调用继承类的方法是没问题的，所以其实这样的方法在D7中也可以使用，但却很麻烦。所以Class Helper起到的就是这个作用，在Class Helper中可以增加的就是与实例无关的内容，所以任何需要增加实例Size的活VMT的功能不能声明，例如变量、

虚方法等，但只占用类空间的没关系，如class var。在应用上我们可以通过这种方法方便的给VCL一类控件加上某个属性。

```
  TFoo = class helper for TControl
    private
      function GetA: Integer;
    public
      class var X: Integer;
      procedure MSG(var Message: TMessage); message WM_MYMESSAGE;
      procedure FooProc;
      property A: Integer read GetA;
    end;
  // ...
  procedure TForm1.Foofoo;
  begin
    FooProc; // TControl -> TWinControl -> TScrollingWinControl-> TCustomForm -> TForm -> TFrom1: Call
TFoo.FooProc
  end;
```

4. strict关键字（Keyword “strict”）

众所周知，在Delphi中，类的private和protected域中的变量可以被同一单元中可以自由的被访问（Delphi的类没有“友元”的概念，但同一个unit中可以说自动友元化了），而并非是真正的私有或只能被继承类访问。而strict关键字的作用就是使该内容变成严格OO意义上的private/protected作用域，这点没有什么多说的。语法：

```
  strict private
    // Blah...
  strict protected
    // Blah...
```

5. 结构方法（Records with Methods）

也没什么特别的，就是和class差不多，就一个不用创建和销毁、不能继承、没有作用域之类的类，很容易掌握，所以这里就不多介绍了。但是很有意思的是带参数的constructor可以通过编译，可能是为了初始化的方便吧。

6. 抽象类和固实类（Abstract and Sealed Classes）

这两个概念在OO中也并不陌生，抽象类是不应该创建实例的（但D2006起的编译器就不给检查，连个Warning都没有，这还有啥用啊 -.- ），而固实类是不能被继承的。语法：

```
  TAnAbstractClass = class abstract  // or (TParentClass)
    // Blah...
  end;
  TASealedClass = class sealed(TAnAbstractClass) // or empty
    // Blah...
  end;
```

7. 类常量、类变量、类属性与静态类方法（Class const/var/property and Static Class Methods）

老的Delphi中只提供了类方法，而没有提供类变量、类常量和类属性，这真的是很不方便。这里先区分一下我所使用的类（Class）和对象（Object）即类的实例（Instance of Class）。当在Delphi中声明一个类的时候，这个类是有实际地址的，该地址记录了许多类的相关信息，比如实例的Size啊、虚方法信息啊一堆东西，而创建一个对象的时候则把类实例化，在堆（Heap）中分配一块地址，包括内部数据和VMT之类的东西。在调用实例的时候，首先要知道对象地址，然后才能访问内部变量和调用方法时使用Self指针即实例地址；而在调用类方法的时候，eax中的并不是实例的地址而是类的地址，然后再call方法，这时的Self指针并非实例地址而是类地址。所以对于每一个类和继承类来说，包括它和它的继承类的所有实例，类变量、常量都是同一个，这样就存在了一个唯一的可供使用的变量或常量，方便同步并且不需要使用较多的内存（可以参考C#中的类，不过C#中不允许从实例直接访问类变量、常量、方法）。而静态类方法则是在使用这个类方法的时候不传入class地址，也就是说没有Self指针，这样的类方法的访问开销要小于普通的类方法；这自然也就意味着，该类方法不能被继承（不能virtual/override）。另外，类属性的get/set方法必须使用静态类方法。

```
  TFooClass = class
  private
    class procedure SetFoo(const Value: Integer); static; // A Static Class Method
  protected
    class var FX : Integer; // class var
  public
```

```
    const FC: Integer = 10; // class const
    class procedure VirtualProc; virtual;
    class property X: Integer read FX write FX; // class property
    class property Foo: Integer read FC write SetFoo;
  end;
```

8. 类内部类型与嵌套类 (Class Types and Nested Classes)

可以说现在的Class的域几乎相当于原来的整个unit，以前不能放里面的元素现在都可以放里面了，这个也没什么好多说的，试验一下就很容易明白了。

9. 终方法 (Final Methods)

这个也是建立在虚方法的基础上的，在override后使用final关键字，则表示该虚方法不能再被子类继承下去了。

```
  TAClass  = class
  public
    procedure Foo; virtual;
  end;
  TFinalMethodClass  = class(TAClass)
  public
    procedure Test; override; final; // A Final Method
  end;
```

10. For-in循环 (For-in Loop)

这个应该是受.Net影响吧，支持遍历一个数组或提供了GetEnumerator函数的类。GetEnumerator要求返回一个类的实例，该类包含有Current属性和MoveNext方法。

```
procedure Foo(List: TStrings);
  i : Integer;
  lst : array[0..100]of Integer;
  s : string;
begin
  for i in lst do ;
  for s in List do ; // Support of TStrings.GetEnumerator
end;
```

==========

Delphi 2009测试版的新语法特性

结束了对到Delphi 2007的语法回顾，终于正式进入到最激动人心的2009新语法部分了。

1. String的变化与增强了的Exit

为全面支持Unicode，Delphi 2009中的所有跟String相关的部分都由原来的AnsiString变成了UnicodeString。这自然也就意味着，原来一些用String声明的函数现在可能会有一些问题（好在我都不厌其烦的用AnsiString和WideString）。这同时意味着Char已经是WideChar而不再是AnsiChar、PChar也是PWideChar而不再是PAnsiChar了。在使用D2009编程时一定要时刻小心和注意这个地方。

而Exit则变成了类似C的return的作用，不过退出参数是可选的，这样才能兼容以前的代码和Result关键字。虽然说这是一个小改进，但是减少了每次不厌其烦的

```
  if(not True)then
  begin
    Result  := nil;
    Exit;
  end;
```

而只需

```
  if(not True)then Exit(nil);
```

即可了。

2.　匿名方法引用（reference to）
以前我们创建一个方法引用的时候会很麻烦，尤其是在类中，需要跳出去在别的地方写一段函数，然后再回来继续写。新的语法reference to避免了这种情况的发生，尤其是许多时候其实我们的方法实际上只有一两句的时候，它可以大大加快开发的速度，就像前面的Exit语法加强一样贴心。不过遗憾的是，这个类lamda方法的语法糖还不够甜。

```
type
  TFoo = reference to function(num: Integer): Integer; // reference to Method
var
  func: TFoo;
  n: Integer;
begin
  func := function(a: Integer): Integer // *NOTE*: Do NOT Write ';'
  begin
    Result := a * a;
  end;
  n := func(10);
```

3.　增强了的反射单元ObjAuto
这个是RTTI里的，按说不算语法上的更新，但涉及到编译器，又在RTTI方面非常有用，所以这里我还是把它拿出来说了。看过李维的《Inside VCL》的应该都知道TypInfo单元在RTTI中的重要性，而ObjAuto可以说是TypInfo的增强版。

先来点儿预备知识：一般情况下，class声明中的默认区域是public，但TPersistent指定了{YM+}编译器参数，使得包括其继承类（所有VCL控件）的默认区域都成为了published。以前我们可以通过TypInfo获取published区域的方法信息，成为了许多控件自动化功能的重要组成部分。而在2009中又增加了{YMETHODINFO ON}/ {YMETHODINFO OFF}编译器选项，使ObjAuto单元能够获取public区域中的方法信息。具体的示例请看这个链接：http://www.cnblogs.com/del/archive/2008/08/16/1269359.html

4.　泛型
终于到了最激动人心、让人欢喜让人忧的泛型了。大家都知道C++难，异常难，用了十几二十年的人严谨点儿的话也不敢说自己非常懂C++。除了各种cast、实例的创建外，操作符的重载、泛型的支持使得C++变成一个自由度极高的语言，而复杂度也因此上了一个量级。有人说C++的精华就在于Templates，也就是泛型的支持，以我对C++的浅见，窃以为此话还是有一定道理的。在Delphi引入泛型的支持后（还好Delphi的操作符是关键字比较多，不适合重载），许多人担心的是，编译速度会不会变慢。在用了D2009beta版之后，至少目前几个测试项目还体会不到速度的明显降低，好在Delphi里没有C/C++那么复杂的预编译宏。不过就我目前的测试来看，Delphi 2009对泛型的支持还不是那么好，比如class<T>中内嵌type定义record中如果使用了类型为T的泛型成员的话，编译是会挂掉的。虽说有还不算太麻烦的用class代替并且不用默认方式的构架/析构过程的trick来避免这一问题（直接用class模拟record会导致如泛型包含string之类时会发生leak），但不够强大的泛型支持还是比较麻烦的事。

泛型的语法并不复杂，类似C++，在元素名称后使用“<>”包围泛型列表，如“Arr<T>=Array of T”“<TA, TB>”等，在其对应的作用域中可见。可以使用泛型的只有两种情况，一种是新定义一种类型的时候，这个类型可以是record、class、interface、array等，另外一种情况是一个class的方法、类方法可以单独定义泛型，但普通函数/过程不能定义泛型。这里就不具体举例了，有兴趣的可以参考Delphi 2009中自带的Generics.Collections单元中的几个基本类型。

5.　其他
由于对Unicode支持是一个大局大变化，所以由此带来的Windows单元、VCL组件单元带来的变化也不小，其他相关的支持单元也加入了许多元素和进行了一些较大的调整。比如新加入的TEncoding和对TStrings、TStream系列的增强，以及TStringBuilder这种.Net中的概念元素等。还有一些变化使得以前版本的控件无法很方便的移植到D2009中，而Delphi如果缺少第三方组件的支持，可以说吸引力会大打折扣。

对于IDE等方面的变化和VCL控件方面的改进等这里就不介绍了。总之，非常期待正式版的出现，这是Delphi近年来最值得期待的一个版本。同时，Delphi 2009正式版中对泛型的支持我仍然要打一个问号，不知能否改进对使用泛型的class内部record的支持，或者出于编译速度的考虑无法达到十全十美，但愿今天的CodeGear会有一个更令人满意的答复。

--
未名湖里游左右，批踢踢上览绿蓝；

※ 来源：·水木社区 newsmth.net·[FROM: 123.113.33.*]