biololo

新手上路

发表于 2009-10-13 21:25 | 只看该作者 打印 字体大小: 1 #

Runtime Type Information 运行时类型信息（一）

一：获得对象的RTTI（以下代码基于Delphi 6/7）

RTTI(**Runtime Type Information** 运行时类型信息)指的是对象在编译的时候，将其属性、方法的类型信息编译进特殊的区域，使得应用程序可以运行时，动态的查找对象的成员（fields）和属性（properties），以及定位函数（methods）。能够编译进RTTI的成员主要是被声明在对象的published部分，对象published部分的成员与public部分的成员具有相同的可见性，但区别就在于此。当在持久化和反持久化的时候，RTTI被用来访问属性的值、在对象浏览器（Object Inspector）中显示属性，以及关联事件（Event）和事件句柄函数（Event Handler）。Published部分的属性类型被限定在几种特殊的类型中，只能是Ordinal（有序类型）、string、class、interface、variant和函数指针类型，数组属性是不能被声明为published。当然也不是每一个被声明在published部分的成员都可以被持久化下来，比如record、array，同时声明在published部分的函数不支持overload。任何一个类希望拥有RTTI，需要在其类声明的时候加上编译开关{$M+}，或者其父类在声明的时候有{$M+}，所以最简单的方式获得RTTI就是从TPersistent继承。

对象属性的RTTI

特别注意，并不是所有类型的属性都可以被编译到RTTI中。
以下是获得属性、类型的方法

```
function GetObjProps(AObj: TPersistent): String;
var
StrList: TStringList;
```

```
PropCount, I: Integer;
PropList: PPropList;
begin
StrList:= TStringList.Create;
try
  PropCount:= GetPropList(AObj, PropList);
  try
    if PropCount>0 then
      for I := 0 to PropCount-1 do
        StrList.Append(Format('Property %s : %s ;',[PropList[I]^.Name,
PropList[I]^.PropType^^.Name]));
  finally
    if PropCount>0 then FreeMem(PropList,PropCount*SizeOf(Pointer));
  end;
  Result:= StrList.Text;
finally
  StrList.Free;
end;
end;
```

当自己制作一个属性浏览器的时候，就可以通过TypInfo单元中的各种方法，获得属性名称、类型、值的读写。

对象函数的RTTI

之所在很早的时候函数就被编译进RTTI中，并不是为了在Delphi中实现反射，因为反射的概念只是Java、.NET这种基于VM的语言的一个小特性，而基于VM产生出的各种特性，包括垃圾回收、平台无关这些都是编译型语言的硬伤，所以Delphi不会特地为了反射而反射，只是用在一些特殊的领域。Delphi中支持函数的RTTI最早是为了实现事件句柄（EventHandler）的持久化。
在Delphi语言创建之初就确定了对Property-Event的支持，这个特性也是Delphi最富魅力的特性之一。所谓的Event（事件），是消息或内部逻辑中发出的特定的请求，Event的定义需要明确特定的事件意义以及特定的编程接口，它是基于消息机制的一种逻辑扩展接口，Event本身并不直接包含逻辑，它只是一个锚点，真正的执行逻辑在EventHandler中，EventHandler依赖于外部的注入。
例如：
对象声明了一个处理消息WM_LBUTTONDBLCLK的函数，在接收到该消息后执行

```
if Assigned(FOnDblClick) then FOnDblClick(Self);
```

则外部就可以通过在EventOnDBClick属性上挂接处理函数，来实现对事件的响应，注入逻辑。

我们知道，在IDE和持久化机制中，针对事件属性关联的EventHandler是声明在窗体对象published部分的函数（准确的说，由于EventHandler关联关系是需要持久化的，所以在Delphi的持久化机制中，EventHandler一定要是TReader/TWriter的Root下published中声明的函数），这是一种比较古老也过时的设计，现在由于AOP（Aspect-Oriented program面向方面编程）概念的发展，受其影响已将EventHandler发展为委托对象，事件的发起者只需要将自己注册在委托对象上，事件的处理逻辑也不直接响应事件，而也是注册在委托对象上，这样设计的好处是由于中间存在了一个delegation，也就提供了更方便更灵活的注入逻辑的机制，在后期加入和改变事件处理逻辑的时候也最大限度的保障了原有逻辑的稳定。这也是.Net中发展出委托的原因。当然，在很多国内的书籍中介绍到.Net的委托的时候，都会提到『不必关心具体的执行者，只要知道你的消息交给哪一个委托就好了』，这样的解释并没有切题，因为无论是Delphi中比较古老的设计，还是现今的委托，消息的发起者都不需要关心接收者的处理。

下面看一段持久化机制中的代码：

```
procedure WriteMethodProp;
var
  Value: TMethod;
begin
  Value := GetMethodProp(Instance, PropInfo);
  WritePropPath;
  if Value.Code = nil then
    WriteValue(vaNil)
  else
    WriteIdent(FLookupRoot.MethodName(Value.Code));
end;
```

这段是TWriter.WriteProperty中持久化Event的子函数，基本逻辑就是：根据函数地址，在LookupRoot中找寻到函数名称，将其持久化。这里的LookupRoot就等于Root。而在TReader中反持久化的时候，代码如下：

```
  tkMethod:
    if NextValue = vaNil then
    begin
      ReadValue;
      SetMethodProp(Instance, PropInfo, NilMethod);
    end
    else
```

```
    begin
      Method.Code :=  FindMethod(Root, ReadIdent);
      Method.Data := Root;
      if Method.Code <> nil then SetMethodProp(Instance, PropInfo, Method);
    end;
```

这下就一目了然了，Data是Root，Code是根据函数名称在Root下找寻到函数地址。

如果我们想在Delphi中实现委托对象的话，可以在委托对象持久化的时候记录下Event的关联关系，例如，可以是以下的dfm文件：

```
...
  delegation: TNotifydelegation
     Events=<
         item
            host = Button1
            Event = 'OnClick'
         end>
  end
```

而不必拘泥于一定要生成如下形式，

```
...
  Button1: TButton
     OnClick = delegation.OnNotify
  end
```

因为如果生成这种形式的话，需要改写VCL中的一些代码。

最简单的情况下，函数的RTTI是通过如下形式获得到的：

```
var
VMT: Pointer;
MethodInfo: PMethodInfoHeader;
begin
  VMT := PPointer(AObj)^;
  MethodInfo := PPointer(Integer(VMT) + vmtMethodTable)^;
  ...
end;
```

这是一段摘自VCL中的代码，其意义是对象首地址负偏移vmtMethodTable（vmtMethodTable=-56在单元System中有相关常量值的定义）

是RTTI方法表的入口地址，注意，方法表入口首先存储的当前对象的方法数量，然后首地址偏移2 Byte后才是所有函数的名称。在单元ObjAuto中相关结构体定义了方法表的内存结构。
当只有{$M+}的时候，方法表的内存布局是以下结构：

```
TMethodInfoHeader = packed record
  Len: Word;
  Addr: Pointer;
  Name: ShortString;
end;
```

其中Len是该函数信息结构的大小（当只有{$M+}时，Len=TMethodInfoHeader结构体的大小，注意Name是变体；当有{$M+}{$METHODINFO ON}时，Len=TMethodInfoHeader+TReturnInfo+TParamInfo+...+TParamInfo），Addr指向代码段函数地址，Name为函数名。

当有{$M+}{$$METHODINFO ON}时，内存布局如下：

```
TMethodInfoHeader = packed record
  Len: Word;
  Addr: Pointer;
  Name: ShortString;
end;
+
TReturnInfo = packed record
  Version: Byte; // Must be 1
  CallingConvention: TCallingConvention;
  ReturnType: ^PTypeInfo;
  ParamSize: Word;
end;
+
TParamInfo = packed record
  Flags: TParamFlags;
  ParamType: ^PTypeInfo;
  Access: Word;
  Name: ShortString;
end;
+
```

```
...
+
TParamInfo = packed record
  Flags: TParamFlags;
  ParamType: ^PTypeInfo;
  Access: Word;
  Name: ShortString;
end;
```

其中，函数有多少参数就有多少TParamInfo结构体。任何对象函数，都包含第一个隐式参数Self，所以任何函数都至少包含一个TParamInfo结构体。

在最新版的Delphi中，为了更好的支持反射，于是默认情况扩展了RTTI信息，所以函数表内容变成了不但含有函数头信息，还包含了返回值和参数信息，故而编译后可执行程序的体积也变得庞大。

bhylolo@gmail.com