biololo

新手上路

发表于 2009-10-13 21:32 | 只看该作者 打印 字体大小: 1 #

Runtime Type Information 运行时类型信息（三）

三：通过运行时信息动态驱动函数

有了前面的一些知识，我们就可以获得对象RTTI中函数的信息了。至于如何调用执行这些函数，这里有两种方式。

第一种：

调用TObject.MethodAddress根据函数名称获得函数地址，这时由于只是得到函数地址，需要在调用端明确知道函数类型才能调用。形如以下代码：

```
var
P: procedure(Sender: TObject) of object;
begin
with TMethod(P) do
begin
  Data:= Sender;
  Code:= Sender.MethodAddress('OnbtnClick');
end;
P(Sender);
end;
```

这种调用，形式简单，关联速度快，调用速度几乎不受损失。持久化机制中，对于Event-EventHandler关联和事件调用，就是采用该种方式。缺点是，该种方式获取函数地址的时候，并不能得到函数的类型和参数列表，除非在调用端明确知道函数类型，否则无法执行函数。基于此方法而设计的系统，例如Delphi持久化Event，都是依靠Event才知道Handler的类型，他们之间的关联有效性往往是在先期某个阶段就有检查。

第二种：

这是一种更加灵活一点的函数动态执行方式，使用ObjAuto单元中ObjectInvoke方法（参数：Instance，声明函数的对象；PMethodInfoHeader，函数的头信息；Params，传入参数的开放数组，传入参数的顺序和声明顺序相反，例如，声明成procedure pro(P1: Integer; P2: String)，传参的时候，该数组要写成['str',2]）。能够被ObjectInvoke驱动的函数应是声明在对象的published部分，对象声明的时候要加上编译开关{$M+}{$METHODINFO ON}。

假定有以下的类声明：

```
type TMyType = type String;
type
ITest = interface(IInvokable)
['{E6344DBD-8663-40F2-8C7A-C6DFC4FCCA51}']
  procedure ShowMsg(); stdcall;
  function AddStr(A1: String; A2: Integer): String; stdcall;
  function GetName(AComponent: TComponent): String; stdcall;
  procedure IncNum(var AInt: Integer); stdcall;
  function BuildObjByClass(AClassName: String): TComponent; stdcall;
end;
{$METHODINFO ON}
TTest = class(TInterfacedPersistent, ITest)
private
  FF2: Integer;
  FF1: String;
  FF3: TObject;
  FF4: TMyType;
public
  procedure AfterConstruction; override;
  procedure BeforeDestruction; override;
published
  property F1: String read FF1 write FF1;
  property F2: Integer read FF2 write FF2;
  property F3: TObject read FF3 write FF3;
  property F4: TMyType read FF4 write FF4;
  procedure ShowMsg(); stdcall;
  function AddStr(A1: String; A2: Integer): String; stdcall;
```

```
  function GetName(AComponent: TComponent): String; stdcall;
  procedure IncNum(var AInt: Integer); stdcall;
  function BuildObjByClass(AClassName: String): TComponent; stdcall;
end;
{$METHODINFO OFF}
```

驱动函数ShowMsg，这个比较简单，没有参数和返回值。

```
ObjAuto.ObjectInvoke(FTest, GetMethodInfo(FTest, 'ShowMsg'), [] ,[]);
```

驱动函数AddStr，两个不同类型的参数，传参的时候是倒序，有一个返回值，但都是简单类型。

```
var
ResultValue: Variant;
begin
ResultValue:= ObjAuto.ObjectInvoke(FTest, GetMethodInfo(FTest, 'AddStr'), [] , [1,
'BBB']);
ShowMessage(VarToStr(ResultValue));
end;
```

驱动函数IncNum，注意参数是var，传入的是引用，给Variant变量赋值的时候有些讲究。

```
var
ResultValue: Variant;
Param: Variant;
I: Integer;
begin
I:= 1;
with TVarData(Param) do
begin
  VType:= varInteger or varByRef;
  VPointer:= @I;
end;
ObjAuto.ObjectInvoke(FTest, GetMethodInfo(FTest, 'IncNum'), [] ,[Param]);
ShowMessage(IntToStr(I));
```

```
end;
```

驱动函数BuildObjByClass，返回的是对象类型，对象地址在TVarData.VPointer上，但Variant不支持对象类型，故TVarData.V**Type**为unknown。

```
var
ResultValue: Variant;
begin
ResultValue:= ObjAuto.ObjectInvoke(FTest,
GetMethodInfo(FTest,'BuildObjByClass'), [] , ['TForm1']);
ShowMessage(TComponent(TVarData(ResultValue).VPointer).ClassName);
...
end;
```

在这里，并没有给出函数GetName是如何驱动，这是因为在使用ObjectInvoke的时候，ObjectInvoke参数为variant类型，由于Variant不支持对象以及指针类型，所以TVarData.V**Type**为unknown，尽管TVarData.VPointer上可能有值，但在ObjectInvoke内部会对传入参数类型做检查，抛出类型异常。有人提出过修改ObjectInvoke的实现，去除传入参数的类型检查，但我觉得此举不甚严谨，因为类型检查在保证程序的鲁棒性是非常必要的，由于ObjectInvoke参数类型为Variant，其本身就不能携带完整实参类型信息（比之更进一步的方法是在驱动接口方法中的实现），另外还有一些类型的完整检查只能依赖于声明信息，例如结构体、枚举、集合、没有RTTI的对象，这些内容在Java、.Net中都比较容易获得，但在Delphi中却很难。最后，看一下VCL中使用ObjectInvoke的地方，主要是针对WebSnap，也即是说ObjectInvoke出现的并不算晚，但是针对Web调用，不支持传递指针也是可以接受的。

bhylolo@gmail.com

高级回复 | 发新话题

replysubmit