

[论坛](#)[搜索](#)[帮助](#)[导航](#)[首页](#)[CodeGear 中文空间](#)[公司官方网址](#)[Embarcadero 相关技术论坛](#) » [Delphi 2010新技术](#) » [Runtime Type Information 运行时类型信息 \(四\)](#)[回复](#)[发帖](#)[返回列表](#)

biololo

 发表于 2009-10-13 21:34 | 只看该作者[打印](#) 字体大小:

1 #



新手上路



## Runtime Type Information 运行时类型信息 (四)

### 四：接口的RTTI

接口也可以应用RTTI，其原理和内容都是和对象的相差无几，在runtime type information中就提到了在接口声明的时候加上编译开关{\$M+}就可以为接口产生RTTI，由于接口中所有的方法和函数都是published的，所以接口RTTI中包含了接口的所有方法和函数，同时只要是父接口声明的时候有{\$M+}，则从其派生的接口中也都有RTTI，这点和对象是一致的，与对象TPersistent相对应的是IInvokable。而且，对于接口，并不需要{\$METHODINFO ON}，仅仅是{\$M+}就可以将所有的函数信息，包括返回值、参数等都编译进RTTI，这些信息的结构和声明在单元IntfInfo中。

下面这个方法是获得接口的RTTI，呵呵，打印出来像不像代码？😄

```
function TForm1.GetIntfRTTI(APInfo: Pointer): String;
var
  IntfMetaData: TIntfMetaData;
  I, J: Integer;
  RoutinePrefix, Params: String;
  StrList: TStringList;
begin
  StrList := TStringList.Create;
  try
```

```

IntfInfo.GetIntfMetaData(APIInfo, IntfMetaData);
with StrList do
begin
  Clear;
  Append(Format('Unit %s', [IntfMetaData.UnitName]));
  Append(Format(' %s = interface(%s)',
[IntfMetaData.Name,IntfMetaData.AncInfo^.Name]));
  Append(Format(' [%s]', [GUIDToString(IntfMetaData.IID)]));
  for I := 0 to Length(IntfMetaData.MDA)-1 do
  begin
    Params: = '';
    for J := 0 to IntfMetaData.MDA[I].ParamCount-1 do
    begin
      Params: = Params+
IntfMetaData.MDA[I].Params[J].Name+':'+IntfMetaData.MDA[I].Params[J].Info^.Name;
      if (J+1)<IntfMetaData.MDA[I].ParamCount then
        Params: = Params+' ';
    end;
    if IntfMetaData.MDA[I].ResultInfo<> nil then
      RoutinePrefix: = 'function ' +IntfMetaData.MDA[I].Name+'(%s):'+
      IntfMetaData.MDA[I].ResultInfo^.Name+';
'+CallingConventionName[IntfMetaData.MDA[I].CC]
    else
      RoutinePrefix: = 'procedure ' +

      IntfMetaData.MDA[I].Name+'(%s);'+CallingConventionName[IntfMetaData.MDA[I].CC];
    Append(' ' +Format(RoutinePrefix, [Params]));
  end;
  Append(' end;');
end;
Result: = StrList.Text;
finally
  StrList.Free;
end;

```

```
end;
```

该方法调用的形式如下：

```
MemoIRTTI.Lines.Text:= GetIntfRTTI(TypeInfo(ITest));
```

通过函数**Type**Info获得接口的类型信息。

对于接口函数，也有类似的方法来驱动，在单元Invoker中TInterfaceInvoker.Invoke方法（参数：Obj，接口的实现对象；IntfMD，接口的运行时信息，通过单元IntfInfo中函数GetIntfMetaData获得；MethNum，所要驱动函数在TIntfMetaData.MDA数组中的index，通过单元IntfInfo中函数GetMethNum获得；Context，用以传递参数和返回值，其本身是一个列表，包含了实参地址和返回值地址）。

下面就是用Invoke来驱动接口方法的代码，接口ITest和实现类TTest如前所声明：

驱动函数ShowMsg，这个依然是最简单，没有参数，没有返回值，但仍然需要创建对象TInvContext。注意，GetMethNum最后一个参数有默认值，该参数的表示的意义是，当函数有overload的时候，需要指明函数参数的个数用以确定需要获得是哪一个重载函数，若没有overload，则为-1。

```
var
```

```
IntfMetaData: TIntfMetaData;
```

```
InvC: TInvContext;
```

```
MIndex: Integer;
```

```
begin
```

```
IntfInfo.GetIntfMetaData(TypeInfo(ITest), IntfMetaData, True);
```

```
InvC:= TInvContext.Create;
```

```
try
```

```
    MIndex:= GetMethNum(IntfMetaData, 'ShowMsg');
```

```
    InvC.SetMethodInfo(IntfMetaData.MDA[MIndex]);
```

```
    FIntfInvoke.Invoke(FTest, IntfMetaData, MIndex, InvC);
```

```
finally
```

```
    InvC.Free;
```

```
end;
```

```
end;
```

驱动函数**AddStr**，这里有两个参数和一个返回值，传实参的时候，不需要填入隐式参数**Self**，**TInvContext**中填充的是包含实参以及返回值的变量的地址，填充序号从**0**开始。

```
var
IntfMetaData: TIntfMetaData;
InvC: TInvContext;
MIndex: Integer;
P1: String;
P2: Integer;
MResult: String;
begin
IntfInfo.GetIntfMetaData(TypeInfo(ITest), IntfMetaData, True);
InvC := TInvContext.Create;
try
    MIndex := GetMethNum(IntfMetaData, 'AddStr');
    InvC.SetMethodInfo(IntfMetaData.MDA[MIndex]);
    P1 := 'BBB';
    P2 := 3;
    InvC.SetParamPointer(0, @P1);
    InvC.SetParamPointer(1, @P2);
    InvC.SetResultPointer(@MResult);
    FIntfInvoke.Invoke(FTest, IntfMetaData, MIndex, InvC);
    ShowMessage(MResult);
finally
    InvC.Free;
end;
end;
```

驱动函数**IncNum**，这里函数原型在声明的时候，参数为**var**，但是在调用的时候，却与其他的形式并没有什么区别，其中原因在于由于实参填充的是变量的地址，至于实参变量是否需要被改写，由函数**invoke**内部判断。

```
var
IntfMetaData: TIntfMetaData;
InvC: TInvContext;
MIndex: Integer;
```

```

P1: Integer;
begin
IntfInfo.GetIntfMetaData(TypeInfo(ITest), IntfMetaData, True);
InvC: = TInvContext.Create;
try
    MIndex: = GetMethNum(IntfMetaData, 'IncNum');
    InvC.SetMethodInfo(IntfMetaData.MDA[MIndex]);
    P1: = 3;
    InvC.SetParamPointer(0, @P1);
    FIntfInvoke.Invoke(FTest, IntfMetaData, MIndex, InvC);
    ShowMessage(IntToStr(P1));
finally
    InvC.Free;
end;
end;

```

驱动函数GetName，有意思的地方来了。还记得在ObjectInvoke中，不能驱动参数为对象的这种函数吗？但是在这里却可以了，因为这里传递实参不再使用Variant开放数组，就不受传参类型的限制了。只是，这里参数类型检查仍然不是很严谨，尽管这个函数是VCL实现，例如，对于函数GetName参数要求传入的是TComponent类型，但是如果实参填入的是一个TPersistent的对象，仍然是可以通过类型检查而调用到这个函数的，只是该函数内部在访问Component.Name属性的时候，才会抛错。

```

var
IntfMetaData: TIntfMetaData;
InvC: TInvContext;
MIndex: Integer;
MResult: String;
begin
IntfInfo.GetIntfMetaData(TypeInfo(ITest), IntfMetaData, True);
InvC: = TInvContext.Create;
try
    MIndex: = GetMethNum(IntfMetaData, 'GetName');
    InvC.SetMethodInfo(IntfMetaData.MDA[MIndex]);
    InvC.SetParamPointer(0, @Self);

```

```

    InvC.SetResultPointer(@MResult);
    FIntfInvoke.Invoke(FTest, IntfMetaData, MIndex, InvC);
    ShowMessage(MResult);
finally
    InvC.Free;
end;
end;

```

有一个小技巧，枚举出程序中所有RegisterClass的类。

```

var
    FFinder: TClassFinder;
begin
    FFinder:= TClassFinder.Create();
    try
        Result:= FFinder.GetClasses(Proc);
    finally
        FFinder.Free;
    end;
end;

```

其中TClassFinder声明在Classes单元中，Proc是回调函数，类型为TGetClass = procedure (AClass:TPersistentClass) of object，在遍历内部的注册列表的时候循环调用回调函数，在外部实现回调函数的时候，判断每次传入进来的类是否是所需要，如果需要则记录保存下来。

### 关于RTTI在动态连接库中的使用

DLL不能传递RTTI，所以在DLL之间、或DLL与exe之间传递对象的时候，不能显示和使用对象的RTTI。如果需要在传递对象的时候同时拥有其RTTI，则需要使用package来替代DLL，因为package之间、package与exe之间是共享RTTI。

最后是一个关于代码格式的问题，很多年前我和别人打过一个赌，下面两种书写格式：

```
if then begin
```

```
end
```

和

```
if then  
begin
```

```
end
```

哪种更加规范。于是我在VCL中搜索，发现其实两种书写方式都有，但是明显后者多得多，究其原因是在Menu->Tools->Editor

Options...->Source Options->Edit Code

Templates中声明的代码模板格式是后一种，在输入的时候只需要键入ifb

组合键Ctrl+j 就可以生成了，习惯后，使用起来是不是很快捷呢？

以上说明皆是基于Delphi 6/7版本，后继版本内容略有差异，但原理大体相当。

收藏 分享 评分

bhylolo@gmail.com

回复 引用

订阅 报告 道具 TOP

biololo



发表于 2009-10-13 22:31 | 只看该作者

2 #



新手上路



似乎发错地方了，各位看官将就将就吧，权当是抛砖引玉，而且写的也很长，看完也是很痛苦的。

bhylolo@gmail.com

回复 引用

报告 道具 TOP

mustapha.wang



发表于 2009-10-13 23:02 | 只看该作者

3 #

好文章。



注册会员




年年难过年年过，事事无成事事成。

回复

引用

报告 道具 TOP

vclclx

 发表于 2009-10-13 23:36 | 只看该作者

4 #



还没看懂，先赞一个。

中级会员



回复

引用

报告 道具 TOP

biololo

 发表于 7 天前 21:59 | 只看该作者

5 #



我写了这篇文章主要是因为看到D2010的反射时想到，既然是基于RTTI，所以现在看到许多反射的特性在很早以前的版本中都已经存在了，写着写着就长了，这中间一些内容在Delphi帮助中也很少提及，但写完后自己回头看也觉得看的很累，呵呵，看来写文章纯属聊以自慰。

新手上路



bhylolo@gmail.com

回复

引用

报告 道具 TOP

返回列表





[高级回复](#) | [发新话题](#)

发表回复

Powered by **Discuz!** 7.0.0

© 2001-2009 Comsenz Inc.

**Embarcadero Tech** | [联系我们](#) | [论坛统计](#) | [Archiver](#) | [WAP](#)

GMT+8, 2009-10-21 23:39, Processed in 0.076145 second(s), 8 queries, Gzip enabled.