声　　明

一、本文可以自由转载于各种网络媒体。

二、转载时请保证本文(含本声明)的完整性。

三、未经本人许可，本文的部分或全部，不得在非网络媒体(包括书籍、报刊、杂志等)上引用或转载。

四、如果本文存在错误，请及时与我联系(见文末)。请不要修改本文的内容。必要时，我会发布修正版本。

五、本声明具有法律效力。

作者：Aimingoo

邮件：aim@263.net

网站：http://www.doany.net/

时间：2003.05.29

CpuWHelper 的开发技术概要

CpuWHelper 是一个用于在 Delphi 的 CPU 窗口中获取汇编源码的工具。它可以作为一个 Expert 被集成到 Delphi IDE 内部。下面讲述它的设计技术要点，并对部分源码进行分析。

一、 Delphi CPU 窗体中为什么不能获取汇编源码

Delphi CPU 窗口设计得非常不错,有着绝大多数的汇编级调试器应该有的功能。——奇怪的是，从它出现的第一天起，它的右键菜单上就没有“复制代码”项。

在 Google Group 中搜索有关 CPU 窗口的信息时，你能够得到的只一个建议：使用 Borland Turbo Debugger。

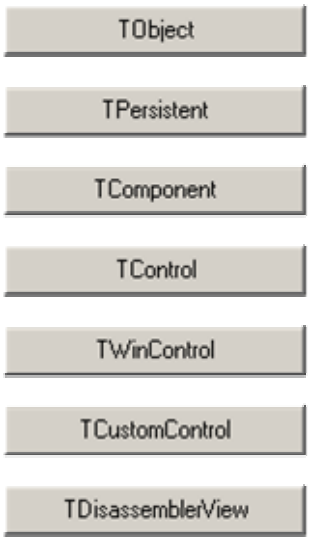
此外，你能够在 vladimir 的网站上(http://www.donpac.ru/~vladimir/)得到一份俄文的 vtExperts 1.11，这个不错的工具能够在 D4-D6 版本中获取 CPU 窗口中的汇编码。但是它在一年前便停止更新了。

为什么没有更多的第三方产品来解决这个问题呢？

“这的确是一个非常棘手的问题”——下面的分析将告诉你这一切。

二、 CPUPane 的结构分析

通常会认为这个功能会很容易实现，因为从表面看起来，Delphi CPU 窗口中的汇编代码是写在一个 Memo 中的，可以简单地通过取 Memo.text 得到内容。

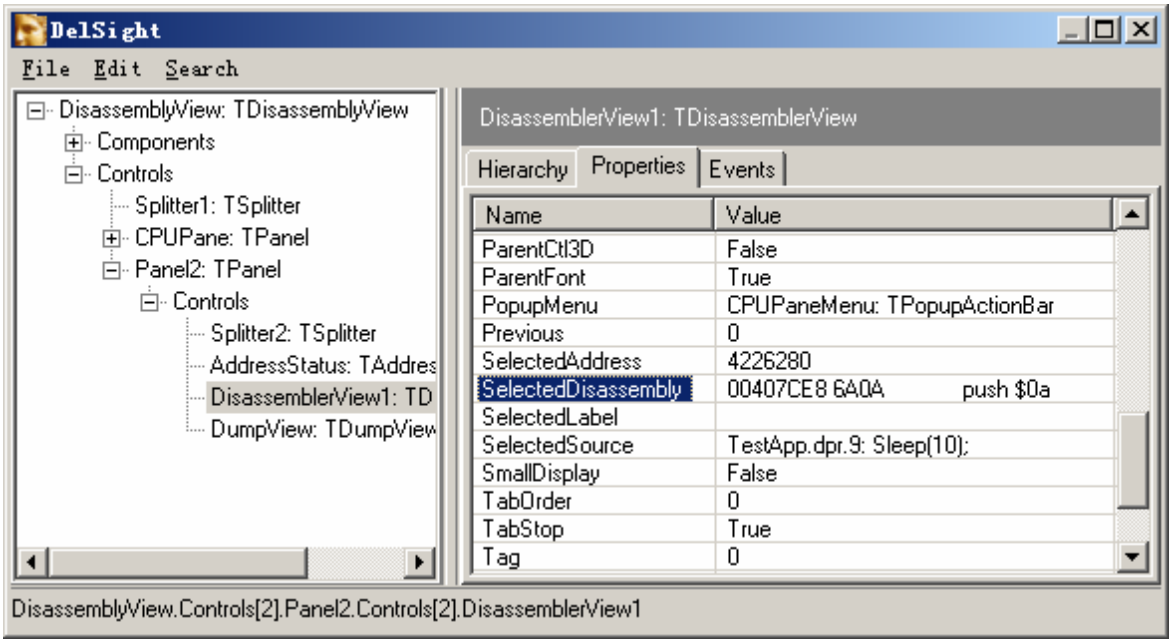


但事实上不是。在这里，Delphi 用了一个定制控件(TCustomControl)来模拟 Memo 的效果。所有文字都是写上去的：它没有真正的“页”的概念，而是以行为单位“PaintLine”。右图表明了 CPU 窗体汇编代码区的对象继承关系。

这就意味着，没有办法直接取出这些汇编代码。

利用象 DelSight (http://www.friedrich.st/)这样的工具，可以进一步地查看 CPU 窗体的信息。下面是利用 DelSight

分析 CPU 窗体的情况：



通过分析和筛选，列出对象 DisassemblerView1 的一些重要属性和事件如下：

```
重要属性：
=========
SelectedAddress      ==> 当前选择行的内存地址
SelectedDisassembly  ==> 当前选择行的汇编代码(只读)
SelectedLabel        ==> 选择行是否是 Label(只读)
SelectedSource       ==> 当前选择行的源代码(只读)
TopAddress           ==> 在当前 CPU 窗口顶部，开始反汇编的内存地址

重要事件：
=========
OnGetLabel(Sender: TObject; Address: Integer; var RetLabel: String; var Result: Boolean);
OnGetNextAddr(Sender: TObject; Address: Integer; var NewAddr: Integer);
OnGetPrevAddr(Sender: TObject; Address: Integer; var NewAddr: Integer);
OnIsBreakpoint(Sender: TObject; Address: Integer; var Result: Boolean);
OnAddressChanged(Sender: TObject; Address: Integer);
OnDisassemble(Sender: TObject; var Address: Integer; var Result: String; var InstSize: Integer);
OnPaintEndOfLine(Sender: TObject; Address: Integer; Canvas: TCanvas; Const Rect: TRect;
                 IsSourceOrLabel: Boolean);
```

这里只列出了部分的属性和事件。上面对于事件的接口描述是不符合 Delphi 语法的，但相信稍为熟悉 Delphi 的人就可以看得明白。

上述事件的接口描述，是通过 RTTI 来分析得到的。目前并没有第三方软件可以帮助我来做这些事。此外，还有一部分事件接口涉及到一个名为” TCPUSourcePos”的数据结构，由于没有更多的资料，我无法确定这个结构的定义，所以，相关的事件，也就没列在上面，包括：OnFindLineCode(), OnGetFileName(), OnFindSourceLine(), OnGetSourceText()等。

三、 获取汇编源码的核心函数

分析到这里，已经无法再找到更多的关于 CPU 窗体的信息了。我做过更近一步的分析，主要包括：对 CPU 反汇编功能所在包 COREIDE70.BPL 进行反汇编分析；利用 DEDE 对 COREIDE70.BPL 进行分析等。但是，我只能找到一些零星的、与目标功能无关的一些资料。

面对上面的这些信息，我的第一方案其实是：HOOK OnDisassemble()或者 OnPaintEndOfLine()，对它们操作的文本内容进行分析！

在这条思路上我走了很远。而且也利用 HOOK 到 OnDisassemble()的技术完成了 CpuWHelper 的第一个版本。但是，其中有两个问题是致命的：

1. 通过 HOOK OnDisassemble()，的确能得到汇编源码，但是，得不到调试信息中的 Delphi 源码！
2. 考虑到速度，IDE 在实现反汇编时，建立了一个 Cache。因此，部分反汇编码(在再次反汇编时)，将不会通过 OnDisassemble()!

我居然在这条死胡同的那一端，面墙而立了近三周的时间！直到有一天从 Diabllo II 中退出时，忽然想到：TDisassemblerView 的属性里不是有“ SelectedSource ”吗？ !!!

是呀！属性 SelectedAddress 是可写的，意味着可以指定任意内存地址进行，而属性 SelectedSource 则表明了在该地址上的源程序码。

当然，如果要这样做，则需要从起始地址开始，计算出每一个汇编指令的长度，然后将递增的内存地址赋给 SelectedAddress。

计算出每一个汇编指令的长度是困难的。但是，很庆幸的，你可以看到：

```
OnDisassemble(Sender: TObject; var Address: Integer; var Result: String; var InstSize: Integer);
```

你应该感到奇怪，它的入口 Address 是变量参数。那么，过程返回的时候，它会是什么值呢？

只要简单的测试，就可以发现，它正好返回下一个指令地址！

到这里，问题就全部解决了。在 CpuWHelper 中，关于反汇编的核心代码如下：

```
function TWizardMod.Call_Disassemble(Line：Integer; StartType：DWORD)：String;
begin
  //...

  while Line > 0 do
  begin
    RTTI_SetIntegerProp(DisComp, 'SelectedAddress', P);
    S_Source := RTTI_GetStringProp(DisComp, 'SelectedSource');
```

```
    //Get Next Address To P, and Get ASM Code to S_ASM
    FDisassemble(DisComp, P, S_ASM, L);

    if S_Source <> '' then
      Result := Result + S_Source + #$0D#$0A;
    Result := Result + S_ASM + #$0D#$0A;
    dec(Line);
  end;
//..
end;
```

四、 集成到 IDE 所遇见的问题

现在的问题，只剩下写一个 Expert 并将它装到 Delphi IDE 里去了。

由于是要向 CPU 窗体的鼠标右键菜单上添加菜单项，因此，有两件事是必须的要做的：

1. 确定 CPU 窗体是打开的，或者至少它被创建过
2. 查找到右键菜单，并添加菜单项

首先要知道的是，Delphi IDE 并不总是创建 CPU 窗体，而只是在需要的时候创建并打开它。这意味着，不能够在加载 Expert 的时候，操作(一个不存在的)CPU 窗体。Delphi 只有在用户处于调试状态并选中“ \Debug Windows\CPU ”菜单项时，才会创建并打开 CPU 窗体，一量该窗体被创建，直到 Delphi 关闭时，才会被释放。

根据这个流程，最好的方法便是在创建这个窗体之后，立即安装菜单项。因此，CpuWHelper 中采用修改 DebugCPUCommand 动作的 OnExecute 事件的方法，使点中”CPU”菜单项(或按下 Ctrl+Alt+C 热键)的行为发生改变，使之完成安装 Expert 的过程。

关于如何修改 OnExecute 事件，请参见另外的一篇文章《如何跨单元、跨类地访问 Delphi 类的私有域方法》。这里就不再详述。

五、 关于 ObjectFromHWnd()

在 Delphi 中有一个专门的函数，用以通过窗体句柄获得 Delphi 对象。这个函数是 FindControl()。

但是，在 Expert 中，你却不能用它来做任何事。——这也不全对，实际上，如果你以包方式来加载 Wizard，这个函数是可用的。

那么，为什么用 DLL 形式的 Expert 就不呢？

分析 RTL 源码发现，在 Delphi 初始化每一个 TWinControl 对象时，将会在窗体的属性(PropData)中加入一些标志，其中的一个标志是用于存放该对象的内存地址的。而 FindControl 就是通过查看该属性来获取对象在内存中的起始地址。——即内存实例地址。

这个“ 标志 ”，其固定形式为：

```
ControlAtomString := Format('ControlOfs%.8X%.8X',
                            [HInstance, GetCurrentThreadID]);
```

由于在 DLL 中，HInstance 的值与 HOST 进程的 HInstance 并不一致，所以，在 DLL 中的 `ControlAtomString` 也就与 HOST 进程不一致。那么，通过 B 标志去查 A 标志的属性，自然什么也得不到了。

通过 Win32API GetWindowLong()，可以获得一个窗体所在的(真实的)实例句柄。这样，可以在 DLL 中重新构造针对于任何一个窗体句柄的 `ControlAtomString`。

不过还是要注意的是，在 Delphi4-7 中建立的窗体都会在属性中加标志。但是，D4-D6 是通过 GetProp()来取属性，而 D7 中却是通过注册消息的方法，让窗体返回自己的句柄地址。——尽管如此，GetProp()也仍然是可以使用的。

基于这样的技术原理，我在 CpuWHelper 中，完成了函数 ObjectFromHwnd()。

六、 版本兼容

版本问题是很让人头痛的。通过分析发现，D4-D6 中，存在一个 IDE 实现上的 BUG：如果你用 AddMenu()向 CPU 右键菜单中加入超过 2 个的项，则，你会发现，第三个菜单会出现在 IDE 的 File 菜单下。而完全相同的代码，在 D7 中不会有问题。

此二，D7 和 D6 的.DFM 文件是可通用的，在 D5 以下，却不行。因此，我留下了一个 CopySetting.txt，方便转换到 D4 下编译。

在 D2-D3 中的插件和 RTTI 实现机制与 D4 以上完全不样。因此，D4 以下的版本就没有再做了。D4 版本的 CpuWHelper 也存在一些 BUG，但目前我不打算修正它。

七、 其它

本文所述基于 CpuWHelper 1.2 版本的代码。以后版本的技术上的细节将会出现在 README.TXT 文档中。计划中将出现的一些功能包括：更全面的配置功能和配置信息保存，多语言版本，以及添加模拟命令行的功能等。

如果有其它问题或者需求，可以跟我联系：aim@263.net。