基于 ActiveX Scripting 组件的调试器实现*)

于国良　韩文报
(信息工程大学信息研究系　郑州 450002)

摘　要　利用可重用性的组件或对象,重新构建新应用程序,能明显提高程序开发效率和改善程序质量。本文首先介绍了基于组件的 ActiveX Scripting 的技术框架,分析了框架中各组件的功能,并在此基础上给出了利用这些可重用的组件构建调试器程序的具体实现。

关键词　ActiveX Scripting,可重用组件,进程调试管理器,接口

Implementation of Debugger Based on Components in ActiveX Scripting

YU Guo-Liang　HAN Wen-Bao
(Department of Information Research, Information Engineering University, Zhengzhou 450002)

Abstract　Using reusable components or objects to reconstruct new application can greatly improve efficiency in programming and quality of codes. This paper firstly describes technique framework of ActiveX Scripting and analyzes function of every component in it. Then an implementation of debugger is provided based on reusable components.
Keywords　ActiveX scripting, Reusable components, Process debug manager, Interface

1　引言

利用可重用性的组件或对象,重新构建新应用程序,能明显提高程序开发效率和改善程序质量。首先,这些组件和对象作为一个完整的功能实体,可以直接拿来插入到应用程序中应用,能明显提高效率;再者,这些组件和对象都经过了严格的测试和用户的实际应用,已经证明了是高质量的部件,所以不需要再进行测试,即可保证程序的质量。基于组件技术的开发方法作为第三代面向对象的开发方法[1],正在体现出越来越明显的效益和优势。

Microsoft 的 COM 作为可重用性的组件开放标准,有很强的扩充和扩展能力。COM 规定了对象模型和编程要求,定义了二进制接口标准,使对象可以和其它对象相互操作。这些对象可以用不同的语言、不同的结构实现。ActiveX Scripting 技术作为 COM 的一个组成部分,通过提供一个完整的接口框架,使应用程序能够在运行过程中嵌入脚本语言支持,使用户能通过对脚本语言的编程、调试来对应用程序进行更有力、更方便的控制[2]。

这种技术来源于一个应用程序对另一个应用程序的引用,即在不了解另一个应用程序的实现细节的情况下控制它的运行,它采用解释性的宏语言来对另一个应用程序进行调用(也就是 Automation 技术)。随着 COM 发展,Microsoft 提供了更为灵活的方式,即 ActiveX Scripting 技术。像 Microsoft 的 Office 软件中,就广泛应用了这种技术,它把方便、简洁的 Visual Basic for Application(VBA)语言作为一种嵌入语言,使用户不仅可以对 Office 界面进行任意定制,还可以通过编写 VBA 语言实现较为复杂的功能扩充。为了帮助开发人员迅速有效地进行这类应用程序的开发,Microsoft 提供了稳定方便的 ActiveX Scripting 框架。

ActiveX Scripting 技术是 Microsoft 对 Script 语言从开发、调试、运行的基于组件的一个完整解决方案,它提供了一套标准的接口和组件,使开发人员能利用这些接口和组件迅速地开发出基于 Script 语言的应用。但是,由于它的领域性限制以及接口的复杂多样性,对它的应用和研究却并不多。

本文通过对一个基于 ActiveX Scripting 组件的 Script 语言调试器的具体实现,展示了 ActiveX Scripting 技术和应用,以便软件开发者熟练使用这种技术。

2　ActiveX Scripting 的技术架构

Microsoft 的 ActiveX 技术架构[2]包括 5 个部分:Script Host(脚本宿主程序)、Language Engine(脚本语言引擎程序)、Process Debug Manager(进程调试管理器程序)、Machine Debug Manager(本机调试管理器程序)、Debugger IDE(调试界面程序)。它们的关系如图 1 所示。

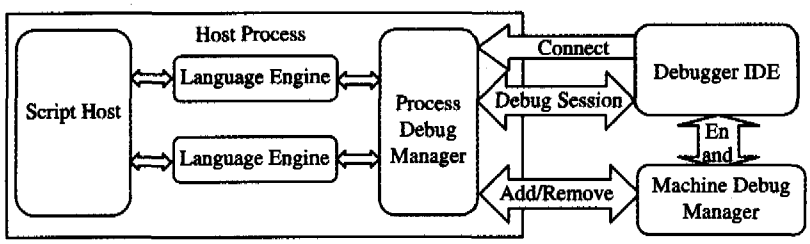

图 1　ActiveX 技术架构各部分的关系

*)基金项目:国家自然科学基金资助项目(No. 19971096, No. 90104035)。于国良　系统分析员、博士研究生,主要研究方向:软件工程、中间件技术。

这5个部分与外部的交互或者相互的交互都通过接口来进行，每一个部分都注册为一个单独的COM组件。

在这5部分中，Script Host控制Language Engine的运行，为Language Engine提供运行空间，同时为Language Engine提供其名字空间中对象的引用。在Script Host程序中，必须实现的接口为IActiveScriptSite，脚本引擎通过它来获取其宿主程序的信息，以及在脚本语言中名字的引用。可选的IActiveScriptSiteWindow接口是提供脚本语言中的窗口等UI功能。要实现脚本的调试功能，便需实现IActiveScriptSiteDebug接口。Script Host和Language Engine的关系如图2所示。吕思伟，潘爱民在文[3]中介绍了调用VBScript的Script Host的实现。

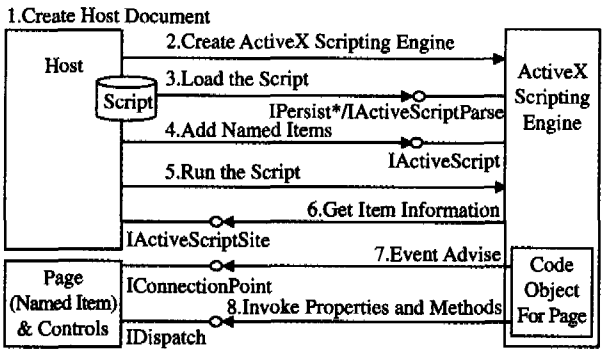

图2 Script与Language Engine的关系

Language Engine的主要作用就是对脚本进行词法和语法分析，然后执行脚本。另外Language Engine程序也提供对脚本调试功能的实现，比如断点的设置、对断点进行不同的语法颜色表示、脚本中对象的监视、运行栈空间内容的枚举等。这里所说的调试，与平常软件开发中所说的调试是一致的。Debugger IDE显示的是用户界面，但调试的实际内部功能的实现却是由Language Engine来完成的，调试程序中用户界面显示的内容是引擎程序的执行情况。

Debugger IDE就是提供一个调试的用户界面，所以有时也称为调试程序IDE环境。它的意义与我们通常所说的调试程序没有什么区别，比如进行断点的设置、对断点进行不同的语法颜色表示、脚本中对象的监视、运行栈空间内容的枚举等。要注意的是，这里的调试功能并不是由Debugger IDE提供。它提供的只是一个调试的用户界面，其实质只是建立一个到脚本的执行时内存空间的一个引用，同时用不同的动作来响应不同的中断事件。由于它只是建立内存空间的一个引用，所以需要实现的接口非常简单：IDebugApplication和IDebugSessionProvider，用IDebugApplication把调试程序注册到Machine Debug Manager中去，用IDebugSessionProvider来和Process Debug Manager建立连接。由于脚本的执行是由Language Engine在Script Host的地址空间上运行的，调试程序只是对Script Host地址空间上的一个引用，所以调试程序与具体要调试的脚本语言无关。

Process Debug Manager主要是对调试程序进行管理，它可以同时支持几个调试程序。它监视正在运行的所有调试程序的进程以及所有线程，同时协调Language Engine、Debugger IDE、Machine Debug Manager之间的关系。

Machine Debug Manager是一个任务管理器，它把本机中所有的调试程序管理起来。

ActiveX Scripting的调试过程：Language Engine在执行脚本的过程中，当它碰到所设置的断点时，把本身挂起，同时调用Process Debug Manager中IRemoteDebugAppliction接口，由IRemoteDebugAppliction接口连接Debugger IDE中IApplicationDebugger接口，触发Debugger IDE中IApplicationDebugger接口中的OnHandleBreakPoint事件。在OnHandleBreakPoint中获取断点上下文的状态、内存空间的引用。当然也可以通过别的窗口把内存执行空间的情况显示出来，比如断点的位置、对断点进行不同的语法颜色表示、脚本中对象的监视、运行栈空间内容的枚举等。

3 调试器程序的实现

在Microsoft的整个ActiveX Script技术架构中，由于每一部分都是一个独立的COM组件，所以每一部分都可以根据用户的需要进行重建。例如，可以建立自己的Language Engine，对自己所设计的脚本语言进行词法和语法分析。在实际工作中，为了实现中文语言脚本来进行测试工作的批处理，我们实现了中文脚本语言引擎，来对中文脚本语言进行词法和语法分析，同时也实现了中文脚本语言调试器。

这里给出基于VBScript语言的调试器的实现。

由于要讨论的只是脚本语言的调试，我们便直接利用现成的Machine Debug Manager、Process Debug Manager和Language Engine组件，来实现调试器。这里实现的是VBScript的调试器（当然也可以换成JScript，只需把VBScript的CLSID改变成JScript的CLSID），实现Language Engine组件功能的是VBScript.dll，实现Machine Debug Manager组件功能的是mdm.exe程序，实现Process Debug Manager组件功能的是pdm.dll。在调试器的实现中，我们把Script host和Debug IDE作为一个程序来实现。调试器功能，比如断点的设置、对断点进行不同的语法颜色表示、脚本中对象的监视、运行栈空间内容的枚举等，其实都是在Language Engine组件中实现的（VBScript.dll）。所以，这里调试器的意思是通过提供一个调试的用户界面，来调用VBScript.dll中实现的调试功能。

首先，我们需要实现IActiveScriptSite、IActiveScriptSiteWindow、IActiveScriptSiteDebug、IApplicationDebugger、IDebugSessionProvider和IDebugExpressionCallBack等6个接口。IActiveScriptSite、IActiveScriptSiteWindow、IActiveScriptSiteDebug用来实现Script host组件的功能，IApplicationDebugger、IDebugSessionProvider用来实现Debug IDE组件的功能，IDebugExpressionCallBack用来收到调试表达式的结果，这里我们不对它进行讨论。IActiveScriptSite和IactiveScriptSiteWindow只是让Script host运行脚本语言，所以它们方法的实现这里都不进行特别的处理，要么简单返回E_NOTIMPL，要么返回S_OK；而IActiveScriptSiteDebug接口，表示Script host支持调试功能，同时必须建立和Process Debug Manager的通信，在发生调试事件时把Language Engine的运行空间以接口形式显示给Process Debug Manager，以便它进行Language Engine的运行空间的引用。所以在它的GetApplication方法中必须获得Process Debug Manager中的IRemoteDebugAppliction接口。IApplicationDebugger是用来截获执行脚本时的调试事件，所以必须对它的OnHandleBreakPoint进行处理。IDebugSessionProvider接口只有一个方法StartDebugSession用来提供Process Debug Manager和Debug IDE的连接。

调试程序的建立，主要包括以下几个步骤：

1）首先创建Process Debug Manager的一个实例，得到调试应用程序对象接口（IDebugApplication）。

通过 CoCreateInstance 方法创建 Process Debug Manager 的实例，从创建的这个实例中得到一个 IProcessDebugManager 接口；调用 IProcessDebugManager 的 CreateApplication 方法，生成一个调试应用程序对象接口（IDebugApplication）；用 IDebugApplication 接口的 SetName 方法设置其名字，用 IProcessDebugManager 接口中的 AddApplication 方法把调试应用程序加进 Machine Debug Manager 中的应用程序运行列表中。

```
hRes = :: CoCreateInstance(CLSID_ProcessDebugManager,
                           NULL, CLSCTX_ALL, IID_
                           IProcessDebugManager,
                           (LPVOID*)&m_pdm);
hr=m_pdm->CreateApplication(&m_pDebugApp);
ASSERT(m_pDebugApp);
hr=m_pDebugApp->SetName(L"Scripting debug Application");
hr=m_pdm->AddApplication(m_pDebugApp, &m_dwAppCookie);
```

2)生成 Debug IDE 框架。

生成实现 IApplicationDebugger、IDebugSessionProvider 接口的类的实例，调用 IDebugSessionProvider 的 StartDebugSession 方法建立和 Process Debug Manager 的连接。

```
class CApplicationDebugger :public IApplicationDebugger,
  public IDebugSessionProvider
  {
    //…
  }
CComObject<CApplicationDebugger> * ApplicationDebugger;
IDebugSessionProvider * DebugSessionProvider;
hRes = CComObject < CApplicationDebugger > :: CreateInstance
(&ApplicationDebugger);
ApplicationDebugger->SetView(this);
hRes = ApplicationDebugger -> QueryInterface ( IID_IDebugSession-
       Provider,(LPVOID* )&debugSessionProvider
       );
hRes = DebugSessionProvider->StartDebugSession(m_debugApplication);
```

3)生成 Script Host 框架。

生成实现 IActiveScriptSite、IActiveScriptSiteWindow、IActiveScriptSiteDebug 接口的类的实例。

```
class CApplicationDebugger :public IActiveScriptSite, public IActive-
    ScriptSiteWindow,
    public IActiveScriptSiteDebug
    {
    //…
    }
CComObject<CActiveScriptSite> * ScriptSite;
hRes = CComObject<CActiveScriptSite> :: CreateInstance (&ScriptSite);
ScriptSite->m_debugApplication = m_debugApplication;
ScriptSite->m_debugApplication->AddRef();
hRes = ScriptSite->QueryInterface(IID_IActiveScriptSite,(LPVOID
* )&m_scriptSite );
```

4)生成 Language Engine 实例，装入脚本代码文件。

通过 CoCreateInstance 方法创建 Language Engine 的实例，调用 IActiveScriptParse 接口，由它的 ParseScriptText 方法装入脚本代码文件。

```
CLSID clsid;
hRes = CLSIDFromProgID(L"VBScript", &clsid );
hRes = :: CoCreateInstance(clsid, NULL, CLSCTX_ALL, IID_
       IActiveScript,(LPVOID* )&m_languageEngine);
hRes = m_languageEngine->SetScriptSite(m_scriptSite);
hRes = m_languageEngine->SetScriptState(SCRIPTSTATE_INITIALIZED);
IActiveScriptParse * parser;
hRes = m_languageEngine -> QueryInterface ( IID_IActiveScript-
       Parse,(LPVOID* )&parser);
hRes = parser->InitNew();
CString code;
GetWindowText(code);
VARIANT pvarResult;
EXCEPINFO pexcepinfo;
hRes = parser->ParseScriptText(T2OLE(code), NULL, NULL,
       NULL, 0, 0, SCRIPTTEXT_ISVISIBLE, &pvarResult,
       &pexcepinfo);
hRes = m_languageEngine -> SetScriptState ( SCRIPTSTATE_
       STARTED);
```

5)把断点插入 Language Engine。

把在 Debug IDE 中通过界面设置的断点，在列表中存下它的位置和标记，通过 EnumCodeContextsOfPosition 方法把脚本源文件中的位置信息转为 Language Engine 中的代码上下文位置。

```
for(iter = doc->m_breakpointList. begin(); iter ! = doc->m_
    breakpointList. end();iter++)
    {
      IEnumDebugCodeContexts * edcc;
      IDebugCodeContext * dcc;
      ULONG numFetched = 0;
      hRes = m_debugLanguageEngine->EnumCodeContextsOf-
             Position(0,
             GetEditCtrl(). LineIndex(( * iter). m_line),
             GetEditCtrl( ). LineLength (( * iter). m_
             line), &edcc);
      if(edcc->Next(1,&dcc,&numFetched)==S_OK)
      {
        if(( * iter). m_enabled)
        {
          hRes = dcc -> SetBreakPoint ( BREAKPOINT_EN-
        ABLED);
        } // end if(( * iter). m_enabled)
        else
        {
          hRes = dcc -> SetBreakPoint ( BREAKPOINT_DIS-
        ABLED);
        } //end else
        dcc->Release();
      } // end if (edcc->Next(1, &dcc, &numFetched) == S_
      OK)
      edcc->Release();
} // end for
```

6)执行脚本。

调用 Invoke 方法执行脚本。

```
IDispatch * engineDispatch;
DISPID mainID;
hRes = m_languageEngine -> GetScriptDispatch (NULL, &engine
       Dispatch);
OLECHAR * mainName = L"Main";// Function we want to execute
hRes = engineDispatch -> GetIDsOfNames ( IID_NULL, &main
       Name,1,LOCALE_SYSTEM_DEFAULT, &mainID);
DISPPARAMS dispparamsNoArgs ={NULL, NULL,0,0};
hRes = engineDispatch->Invoke(mainID, IID_NULL, LOCALE_
       SYSTEM_DEFAULT, DISPATCH_METHOD, &dispp-
       aramsNoArgs, NULL, NULL, NULL);
```

事实上，当要调试的脚本在 Language Engine 中执行，碰到断点时，它便产生一个断点事件，通知 Process Debug Manager 的 IRemoteDebugApplication 接口，经由 IRemoteDebugApplication 接口通知 Debug IDE 的 IApplicationDebugger 接口，再由 IApplicationDebugger 接口中的 OnHandleBreakPoint 方法来处理断点事件。

结束语 直接利用 ActiveX Scripting 所提供的组件来进行调试器的开发，使调试器的软件架构非常清晰，同时提高了程序开发效率。实际工作中，我们对利用 ActiveX Scripting 组件开发的调试器进行了长时间的测试。结果表明，这样的调试器的运行非常稳定、可靠。

参 考 文 献

1 Szyperski C. Component Software-Beyond Object-Oriented Programming. Santa Fe, NM: Addison-Wesley, 1998
2 Microsoft. Active Script Debug. 2005-5, http://msdn. microsoft. com/library/en-us/sdbug/Html/sdbug_1. asp
3 吕思伟，潘爱民. Active Script 技术介绍. 2005-5, http://www. csdn. net/develop/author/pam/book12_1. shtm
4 Shepherd G. Visual Programmer. 2005-5, http://www. microsoft. com/msj/1099/visualprog/visualprog1099. aspx
5 Pellegrino M. Active Scripting APIs: Add Powerful Custom Debugging to Your Script-Hosting App. MSDN Magzine, 2000-12