自动化(Automation)

潘爱民

http://www.icst.pku.edu.cn/compcourse

内容

- 自动化基础
- 自动化对象实现
- 自动化对象应用
- 自动化编程

自动化产生与发展

- 弱类型的高级语言(比如Visual Basic)如何使用COM?
- VBA(或VBScript)
- 自动化与COM的关系
- 自动化的广泛应用

几个概念

- 自动化对象
  - 实现了IDispatch接口的COM对象
- 属性和方法
- ODL(对象描述语言)
- 类型库
- 自动化兼容的数据类型

属性(property)和方法(method)

- 自动化对象的两个基本特性，都具有符号化的名字，用DISPID来标识
- 属性是指自动化对象的数据特征
  - 属性可以由索引，索引可以是整数，也可以是其他类型
- 方法是指自动化对象所提供的功能服务
  - 方法比属性要灵活得多，可以包含参数

类型库(typelib)

- 类型信息是客户程序与组件对象之间通讯的基础
- IDL和ODL
  - 接口类型信息使用interface或dispinterface关键字描述
  - 对象类型信息使用coclass关键字描述
- library关键字描述库信息
  - 一个组件程序中的所有对象放在一个ODL文件中，并用library关键字描述库信息
- 工具MIDL：从ODL(IDL)编译成TLB文件

ODL描述举例

```
[
    uuid(3C591B20-1F13-101B-B826-00DD01103DE1),       // LIBID_Point.
    helpstring("Point 1.0 Type Library"),
    lcid(0x0000),
    version(1.0)
]
library Point
{
    importlib("stdole.tlb");
    [
        uuid(3C591B25-1F13-101B-B826-00DD01103DE1),  // IID_IPoint.
        helpstring("IPoint interface."),
    ]
    interface IPoint : IUnknown
    {
        [propget, helpstring("Returns and sets x coordinate.")]
        HRESULT x([out, retval] int* retval);
```

未完

ODL描述举例(续一)

```
    [propput, helpstring("Returns and sets x coordinate.")]
    HRESULT x([in] int Value);
    [propget, helpstring("Returns and sets y coordinate.")]
    HRESULT y([out, retval] int* retval);
    [propput, helpstring("Returns and sets y coordinate.")]
    HRESULT y([in] int Value);
    [helpstring("Move Point To.")]
    HRESULT MoveTo([in] int newX, [in] int newy);
}

[
    uuid(3C591B26-1F13-101B-B826-00DD01103DE1),  // IID_DIPoint.
    helpstring("DIPoint interface."),
]
dispinterface DIPoint
{
    interface IPoint;
}
```

未完

ODL描述举例(续二)

```
    [
        uuid(3C591B21-1F13-101B-B826-00DD01103DE1),    // CLSID_Point
        helpstring("Point Class"),
        appobject
    ]
    coclass Point
    {
        [default] dispinterface IPoint;
    }
} // for library
```

另一个ODL接口例子

```
[          uuid(BFB73347-822A-1068-8849-00DD011087E8),
           version(1.0),  helpstring("Useful help string."),
           helpcontext(2480)
]
dispinterface MyDispatchObject {
  properties:
    [id(1)] int x;   // An integer property named x.
    [id(2)] BSTR y;   // A string property named y.
  methods:
    [id(3)] void show();      // No arguments, no result.
    [id(11)] int computeit(int inarg, double *outarg);
};
```

IDispatch接口

```
class IDispatch : public IUnknown
{
   public:
      virtual HRESULT GetTypeInfoCount( UINT *pctinfo) = 0;
      virtual HRESULT GetTypeInfo( UINT iTInfo, LCID lcid,
                                   ITypeInfo **ppTInfo) = 0;
      virtual HRESULT GetIDsOfNames(REFIID riid, LPOLESTR *rgszNames,
                                   UINT cNames, LCID lcid, DISPID *rgDispId) = 0;
      virtual HRESULT Invoke( DISPID dispIdMember, REFIID riid, LCID lcid,
                                   WORD wFlags, DISPPARAMS *pDispParams,
                                   VARIANT *pVarResult,
                                   EXCEPINFO *pExcepInfo, UINT *puArgErr) = 0;
};
```

分发ID(DISPID)

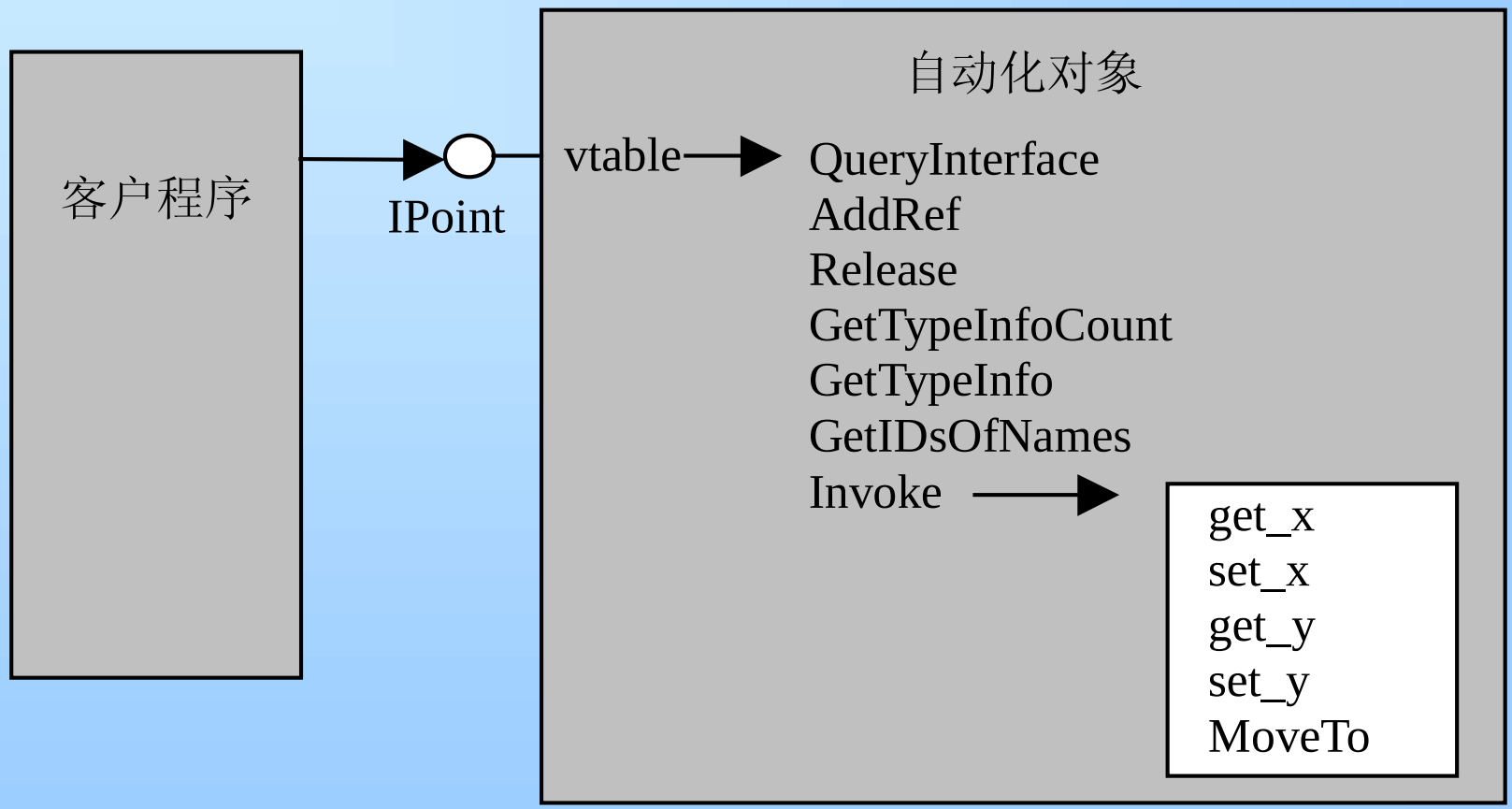
- 整数，0和负数有特殊含义

保留的分发 ID 及其含义

分发 ID	值	说明
DISPID_VALUE	0	接口的缺省成员，如果在客户程序中不指定属性或方法，则用此缺省成员。
DISPID_UNKNOWN	-1	在 GetIDsOfNames 函数中返回此值表明相应的成员或参数名没有找到分发 ID。
DISPID_PROPERTYPUT	-3	在属性设置函数中指示一个接收值的参数。
DISPID_NEWENUM	-4	集合对象的_NewEnum 方法。
DISPID_EVALUATE	-5	对象的 Evaluate 方法，在控制器的脚本语言中可以用方括号[]来表示。
DISPID_CONSTRUCTOR	-6	表示具有与构造函数相同功能的方法。
DISPID_DESTRUCTOR	-7	表示具有与析构函数相同功能的方法。

- 自动化接口通过分发ID管理方法和属性

客户与自动化对象交互示意图



IDispatch::Invoke

- 参数dispIdMember　　指定DISPID
- 参数lcid　　指定本地化标识
- 参数wFlags　　指示调用类型
  - DISPATCH_METHOD、DISPATCH_PROPERTYGET、DISPATCH_PROPERTYPUT　、 DISPATCH_PROPERTYPUTREF
- 参数pDispParams
  - 包括调用的参数数组、参数的DISPID数组、数组中参数个数等信息
- 参数pVarResult　　保存返回值信息
- 参数pExcepInfo　　保存异常信息
- 参数puArgErr　　错误参数的索引值

自动化兼容的数据类型(一)

```
typedef struct tagVARIANT VARIANT;
typedef struct tagVARIANT VARIANTARG;

typedef struct tagVARIANT
{
    VARTYPE         vt;
    unsigned short  wReserved1;
    unsigned short  wReserved2;
    unsigned short  wReserved3;
    union
    {
        // 传值类型
        short           iVal;
        long            lVal;
        byte            bVal;
        float           fltVal;
        double          dblVal;
        VARIANT_BOOL    bool;
        SCODE           scode;
        CY              cyVal;
        DATE            date;
        BSTR            bstrVal;
        IUnknown        *punkVal;
        IDispatch       *pdispVal;
        SAFEARRAY       *parray;

        // 待续
```

自动化兼容的数据类型(二)

```
// 引用类型
  short          *piVal;
  long           *plVal;
  byte       *pbVal;
  float          *pfltVal;
  double          *pdblVal;
  VARIANT_BOOL    *pbool;
  SCODE           *pscode;
  CY             *pcyVal;
  DATE            *pdate;
  BSTR           *pbstrVal;
  IUnknown       **ppunkVal;
  IDispatch      **ppdispVal;
  VARIANT        *pvarVal;
  void           *byref;
    };
};
```

自动化兼容的数据类型(三)

- 布尔型VARIANT_BOOL

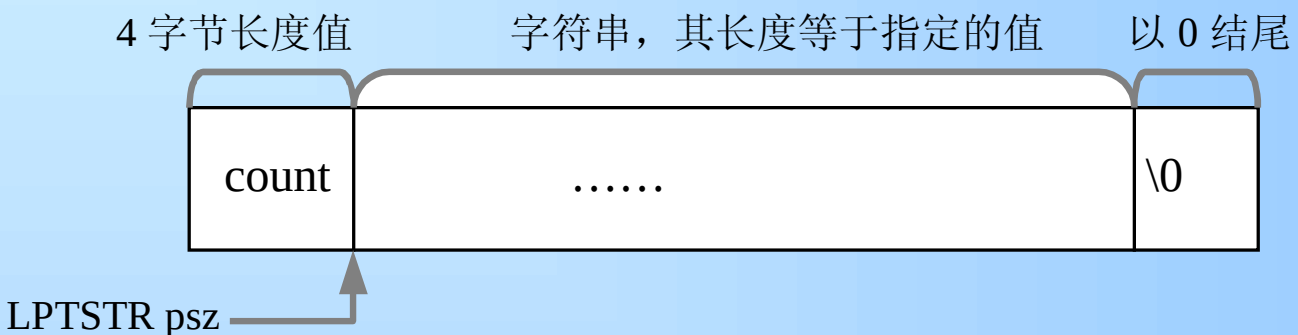
```
typedef short VARIANT_BOOL;
/* 0 == FALSE, -1 == TRUE */
```

- 货币类型CY

```
typedef struct tagCY
    {
      unsigned long Lo;
      long       Hi;
    } CY;
```

- 日期类型DATE
  - 浮点数，整数部分表示自1899年12月30日以来的天数，小数部分为时间值

Basic字符串类型BSTR



- OLE提供了一组API函数处理BSTR：

  SysAllocString、SysAllocStringLen、SysFreeString、SysReAllocString、SysReAllocStringLen以及SysStringLen等

SAFEARRAY类型

```
typedef struct  tagSAFEARRAY
  {
      USHORT cDims;                          // 数组维数
      USHORT fFeatures;                      // 标志信息
      ULONG cbElements;                      // 数组中每个元素的大小
      ULONG cLocks;                          // 数组锁的次数
      PVOID pvData;                          // 指向数组的数据
      SAFEARRAYBOUND rgsabound[ ];           // 数组每一维的边界信息
  }SAFEARRAY;
typedef struct  tagSAFEARRAYBOUND
  {
      ULONG cElements;                       // 该维的元素个数
      LONG lLbound;                          // 该维的下界
  } SAFEARRAYBOUND;
```

- OLE也提供了一套API函数用来处理SAFEARRAY结构

自动化数据类型的转换

- Invoke函数的数据类型转换能力为弱数据类型开发环境提供了极大的便利
- OLE提供了两个类型转换函数：VariantChangeType和VariantChangeTypeEx
- OLE也提供了一组专门的类型转换函数Var<type>From<type>，比如VarR4FromI2、VarUI2FromDisp等

属性和方法调用的参数传递

- Invoke函数的参数pDispParams

```
typedef struct  tagDISPPARAMS
{
    VARIANTARG *rgvarg;                  //  参数数组
    DISPID *rgdispidNamedArgs;           //  参数的分发ID数组
    UINT cArgs;                          //  数组中参数个数
    UINT cNamedArgs;                     //  命名参数个数
} DISPPARAMS;
```

参数顺序

- 在rgvarg数组中，参数的顺序与客户程序中调用的参数左右顺序刚好相反

  比如：

  Object.Method(arg1, arg2, arg3)

  对应Invoke函数的pDispParams参数的DISPPARAMS结构中，cArgs为3，表明方法调用有3个参数，rgarg数组的成员分别为：arg3对应rgvarg[0]、arg2对应rgvarg[1]、arg1对应rgvarg[2]。

可选参数

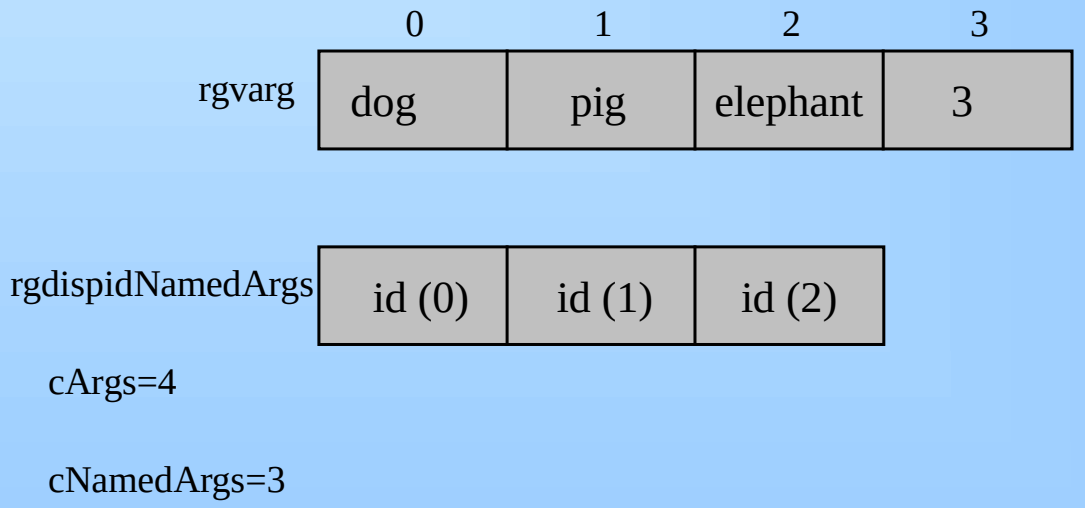
- ODL文件中，我们可以把方法的参数标记为可选的(optional)
- 可选参数也会出现在DISPPARAMS结构中
- 如果vt域为VT_ERROR并且scode域为DISP_E_PARAMNOTFOUND，则此参数为可选参数

命名参数(named argument)

- DISPPARAMS结构的cNamedArgs成员指定了在rgarg数组中命名参数的个数
- 命名参数可以不受次序约束

比如： BSTR CompareWord (int cNumbers, BSTR Word1, BSTR Word2, BSTR Word3)

retVal = Obj.CompareWord (3, Word1="dog", Word2="pig", Word3="elephant")



IDispatchEx接口

- 派生于IDispatch
- IDispatchEx接口最主要的特性是增加了对成员的管理，尤其是动态增加和删除成员的特性

IDispatchEx接口使用例子

```
cmd1.CommandText = "AuthorsByYearBorn"
cmd1.CommandType = adCmdStoredProc
cmd1.Name = "AuthorsSP"
cmd1.ActiveConnection = Cn


Private Sub RunSPButton_Click()
Cn.authorsSP 1947, 1948, rs
End Sub
```

自动化对象实现：GetTypeInfoCount

- 类型库支持
  - 首先把本接口所在的idl文件编译成tlb文件，以便利用COM提供的类型库功能实现本接口的类型支持

```
STDMETHODIMP CImpIDispatch::GetTypeInfoCount(UINT *pctInfo)
{
   *pctInfo=1;       //Because we have implemented GetTypeInfo member.
   return NOERROR;
}
```

自动化对象实现： GetTypeInfo

```
STDMETHODIMP CImpIDispatch::GetTypeInfo(UINT itInfo,
                                        LCID lcid , ITypeInfo **ppITypeInfo)
{
    ……
    if (*ppITI == NULL) {
        hr=LoadRegTypeLib(LIBID_Point, 1, 0, lcid, &pITypeLib);
        if (FAILED(hr))       {
            hr=LoadTypeLib(OLETEXT("Point.TLB"), &pITypeLib);
        }
        hr=pITypeLib->GetTypeInfoOfGuid(IID_DIPoint,  ppITI);
        pITypeLib->Release();
        if (FAILED(hr))   return hr;
    }
     (*ppITI)->AddRef();
    *ppITypeInfo=*ppITI;
    return NOERROR;
}
```

GetIDsOfNames实现

```
HRESULT GetIDsOfNames(REFIID riid, LPOLESTR *rgszNames,
                      UINT cNames, LCID lcid, DISPID *rgDispId)
```

- 第一种方法，由于自动化对象知道自己所有的方法和属性以及它们的分发ID，所以它可以利用这些知识提供GetIDsOfNames服务

- 第二种方法，利用自动化接口的类型信息对象实现GetIDsOfNames函数。
  - 调用ITypeInfo::GetIDsOfNames函数

    ```
    HRESULT GetIDsOfNames(LPOLESTR *rgszNames,
                          UINT cNames,  MEMBERID *pMemId);
    ```
  - 或者OLE的封装函数DispGetIDsOfNames

ITypeLib和ITypeInfo接口

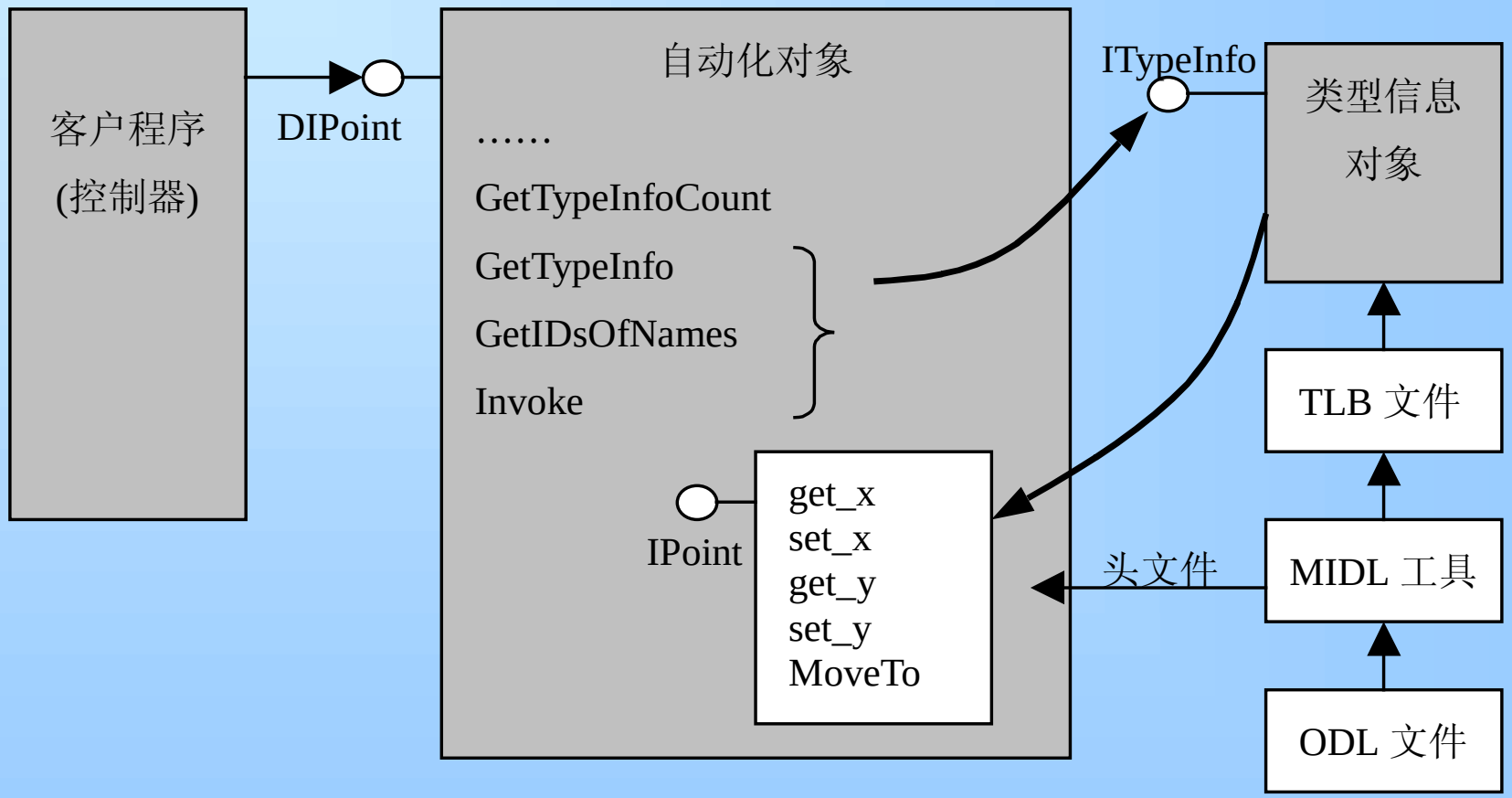
- COM实现的typelib对象
  - LoadRegTypeLib，通过注册表
  - LoadTypeLib
  - 通过ITypeLib接口操纵typelib对象
  - 第三方语言通过typelib支持COM
- typeinfo对象
  - 类型库中的类型对象，例如接口、结构、对象
  - 通过ITypeInfo接口操纵typeinfo对象

Invoke函数实现

- 第一种方法适合于比较简单的情形，Invoke函数负责所有的成员处理，包括参数提取、转换等等，然后根据成员分发ID的不同，执行不同的分支

- 第二种方法也要用到类型库信息对象ITypeInfo接口指针。

类型库与自动化对象之间的关系



MIDL生成的头文件中接口定义

```
MIDL_INTERFACE("3C591B25-1F13-101B-B826-00DD01103DE1")
IPoint : public IUnknown
{
public:
    virtual /* [helpstring][propget] */ HRESULT STDMETHODCALLTYPE get_x(
        /* [retval][out] */ int __RPC_FAR *retval) = 0;
    virtual /* [helpstring][propput] */ HRESULT STDMETHODCALLTYPE put_x(
        /* [in] */ int Value) = 0;
  ……
  virtual /* [helpstring] */ HRESULT STDMETHODCALLTYPE MoveTo(
        /* [in] */ int newX,
        /* [in] */ int newy) = 0;
};
MIDL_INTERFACE("3C591B26-1F13-101B-B826-00DD01103DE1")
DIPoint : public IDispatch
{
};
```

Invoke函数实现举例

```
STDMETHODIMP_(long) CPoint::get_x (void)
{
   return m_x;
}
……
STDMETHODIMP CImpIDispatch::Invoke(
    DISPID dispID, REFIID riid,
    LCID lcid, unsigned short wFlags,
    DISPPARAMS *pDispParams,
    VARIANT *pVarResult,
    EXCEPINFO *pExcepInfo,
    UINT *puArgErr)
{
    ……
    hr=GetTypeInfo(0, lcid, &pTI);
    if (FAILED(hr))   return hr;
    hr=pTI->Invoke((IPoint *)m_pObj,
                   dispID, wFlags,
                   pDispParams,
                   pVarResult,
                   pExcepInfo,
                   puArgErr);
    if (产生内部异常)
    {
        …… // 异常处理
        pTI->Release();
        return DISP_E_EXCEPTION;
    }
    pTI->Release();
    return hr;
}
```

Invoke函数调用过程中的异常处理(一)

- 异常信息由EXCEPINFO结构定义

```
typedef struct tagEXCEPINFO {
    WORD  wCode;                    // 错误(异常)代码
    WORD  wReserved;                // 保留字
    BSTR  bstrSource;               // 发生异常的源对象，一般为对象的ProgID
    BSTR  bstrDescription;          // 错误说明
    BSTR  bstrHelpFile;             // 相关的帮助文件路径
    DWORD dwHelpContext;            // 相关的帮助环境ID
    PVOID pvReserved;               // 保留
    HRESULT (__stdcall *pfnDeferredFillIn)(struct tagEXCEPINFO *);
                                    // 用于填充EXCEPINFO结构、
                                    // 并由对象提供的回调函数
    SCODE scode;                    // 错误的状态码，它与wCode不能同时使用
} EXCEPINFO, * LPEXCEPINFO;
```

Invoke函数调用过程中的异常处理(二)

- 异常信息
  - Invoke函数直接填充
  - 或者通过回调函数由客户控制填充信息
- 如果用类型库实现Invoke函数，那么有两种支持异常的办法：
  - 通过中间变量传递异常信息
  - 使用OLE提供的错误对象

多语种——本地化

- LCID(LocaleID或者locale identifier)函数参数。LCID被定义为一个32位的整数。

- 参看Winnt.h

```
//  +-------------+---------+-------------------------+
//  |   Reserved  | Sort ID |      Language ID        |
//  +-------------+---------+-------------------------+
//   31          20 19     16 15                     0   bit

#define LANG_CHINESE         0x04
#define LANG_ENGLISH         0x09
#define SORT_DEFAULT         0x0     // sorting default
```

用聚合方式实现自动化对象

- CreateStdDispatch函数创建内部聚合对象
- 需要类型库
- 不支持多语种

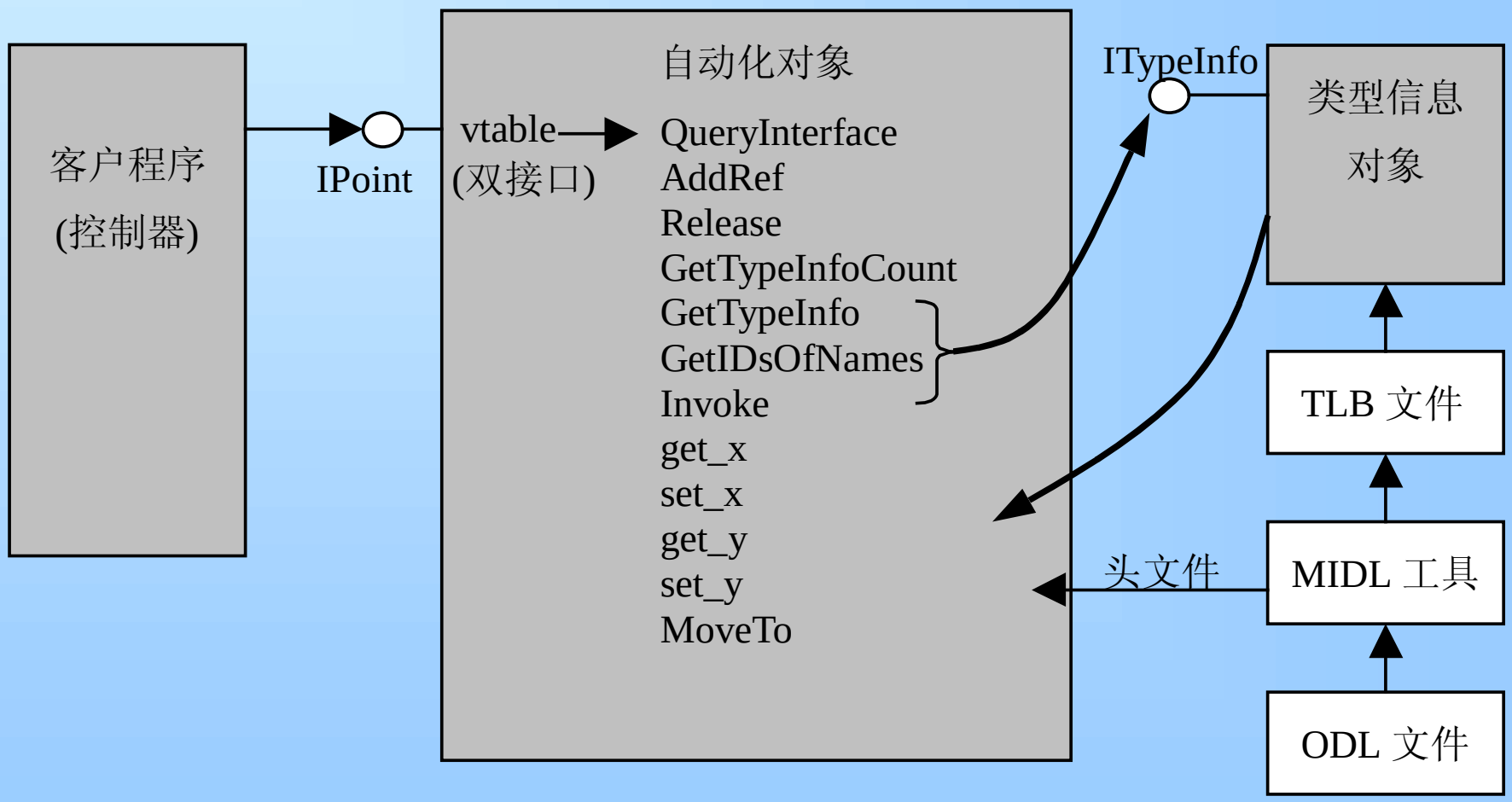
自动化对象应用

- 双接口(dual interface)
- 迟绑定和早绑定
- 自动化集合对象
- 以IDispatch作为出接口 *
- 自动化控制器

dual interface

- 一个接口，既有IDispatch接口的灵活性，又有vtable接口的效率
- 从IDispatch接口派生
  - 使用自动化兼容的数据类型
- 通过vtable可以访问属性和方法

双接口结构图


客户程序
(控制器)
IPoint
自动化对象
vtable
(双接口)
QueryInterface
AddRef
Release
GetTypeInfoCount
GetTypeInfo
GetIDsOfNames
Invoke
get_x
set_x
get_y
set_y
MoveTo
ITypeInfo
类型信息
对象
TLB 文件
MIDL 工具
头文件
ODL 文件

双接口定义

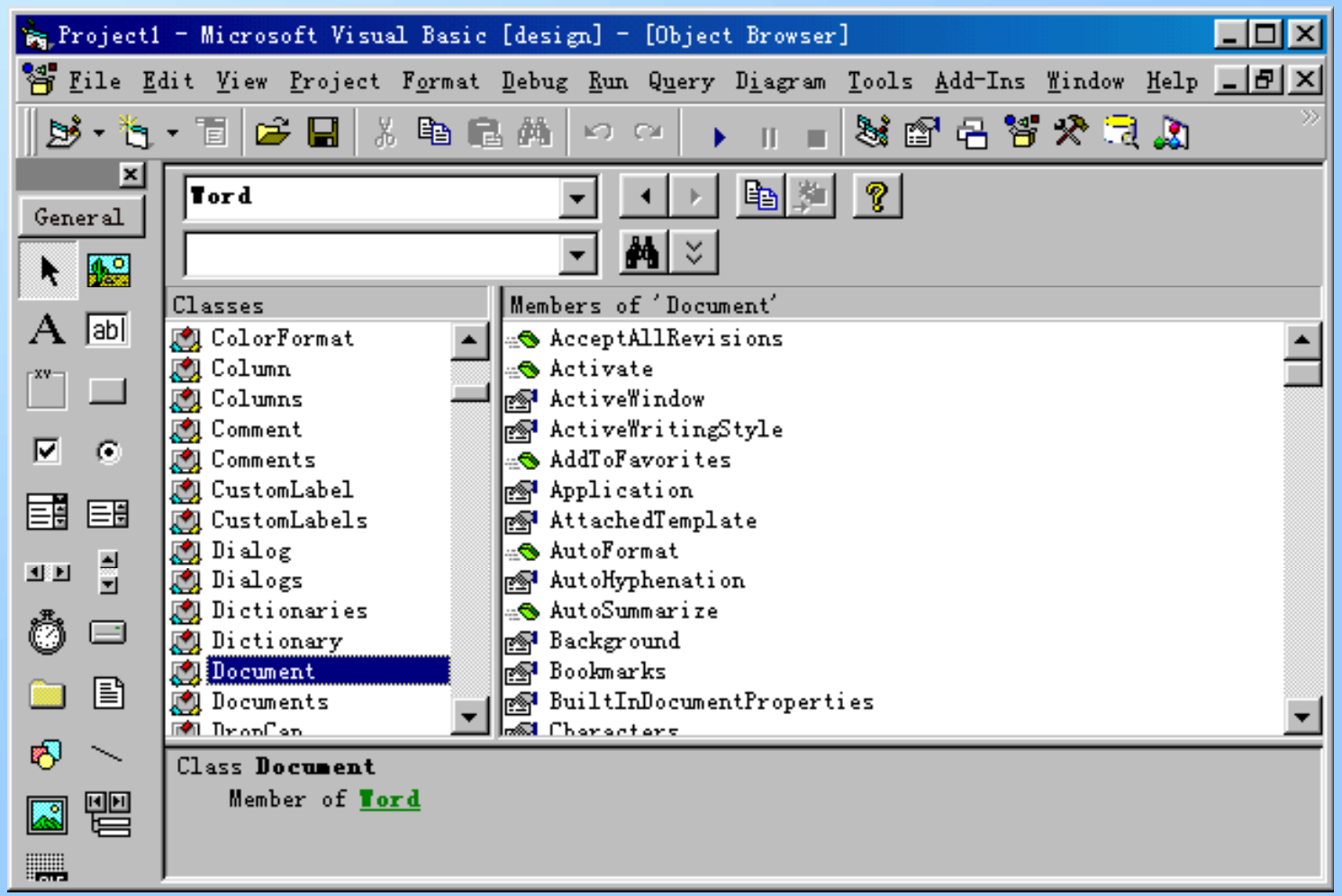
```
[
    uuid(3C591B25-1F13-101B-B826-00DD01103DE1),  // IID_IPoint.
    helpstring("Point object."),
    dual
]
interface IPoint : IDispatch
{
    [propget, helpstring("Returns and sets x coordinate.")]
    HRESULT x([out, retval] int* retval);
    [propput, helpstring("Returns and sets x coordinate.")]
    HRESULT x([in] int Value);
    [propget, helpstring("Returns and sets y coordinate.")]
    HRESULT y([out, retval] int* retval);
    [propput, helpstring("Returns and sets y coordinate.")]
    HRESULT y([in] int Value);
    [helpstring("Move Point To.")]
    HRESULT MoveTo([in] int newX, [in] int newy);
}
```

迟绑定和早绑定

- 早绑定(early binding)
  - compile time，利用typelib
  - vtable binding，dispid binding
  - 一般用于编译环境，例如VB、VC等，以及VBA，效率高
- 迟绑定(late binding)
  - runtime
  - dispid binding
  - 可以用于脚本环境，例如ASP等，灵活性

早绑定

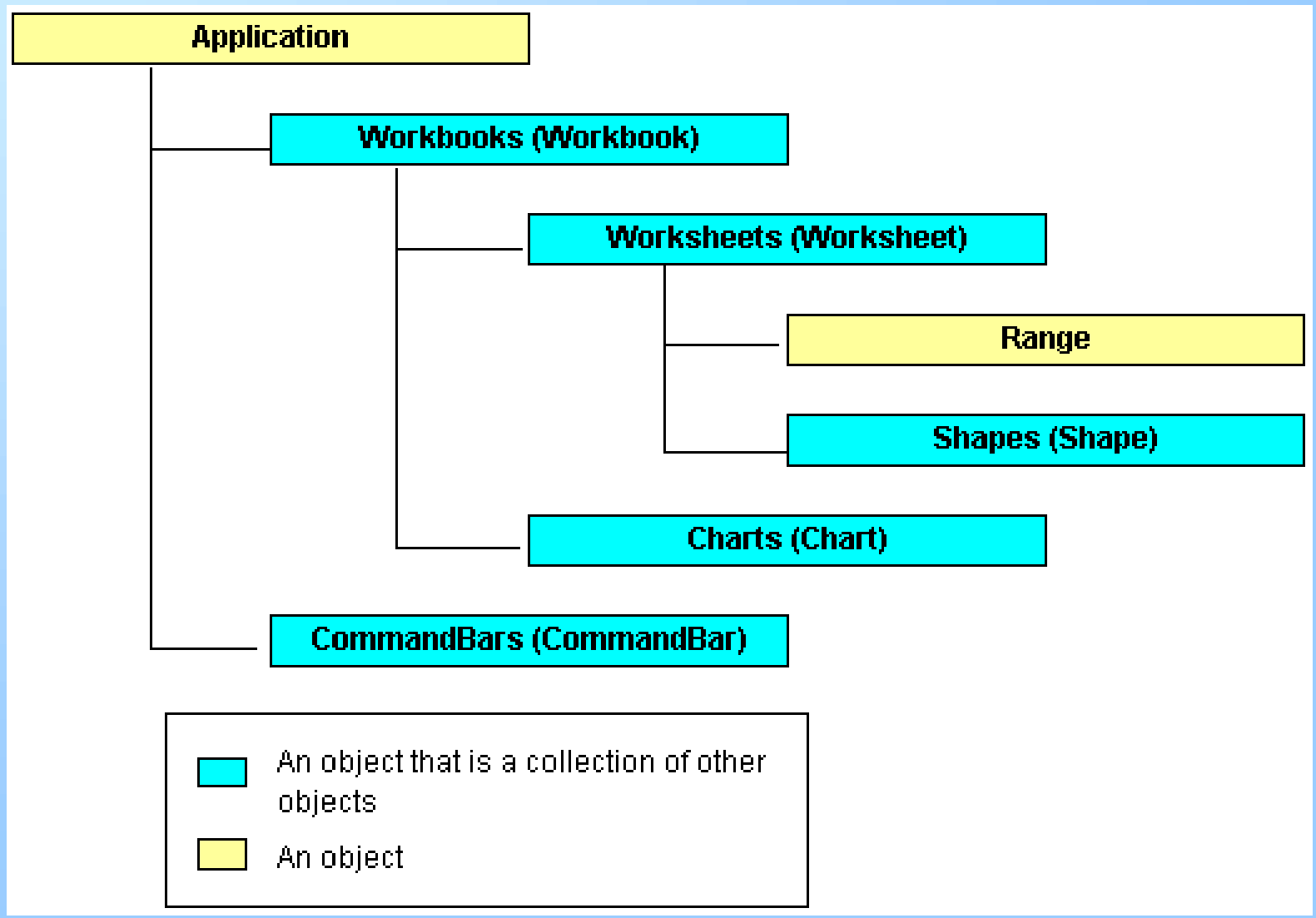
- 例如，在VB中，开发时刻检查类型信息



自动化集合对象

- 集合对象也是自动化对象，但要求：
  - 1 作为一组同类对象(或数值)的容器对象，它必须提供枚举这些成员的方法
  - 2 它必须支持Add、Remove和Item方法以及Count属性
- 索引值可以是整数，也可以是其他类型
- 利用标准属性或方法_NewEnum提供“for…each”语法结构的枚举特性
- 集合对象的命名

对象层次模型示意图


Application
Workbooks (Workbook)
Worksheets (Worksheet)
Range
Shapes (Shape)
Charts (Chart)
CommandBars (CommandBar)
An object that is a collection of other objects
An object

对象层次模型

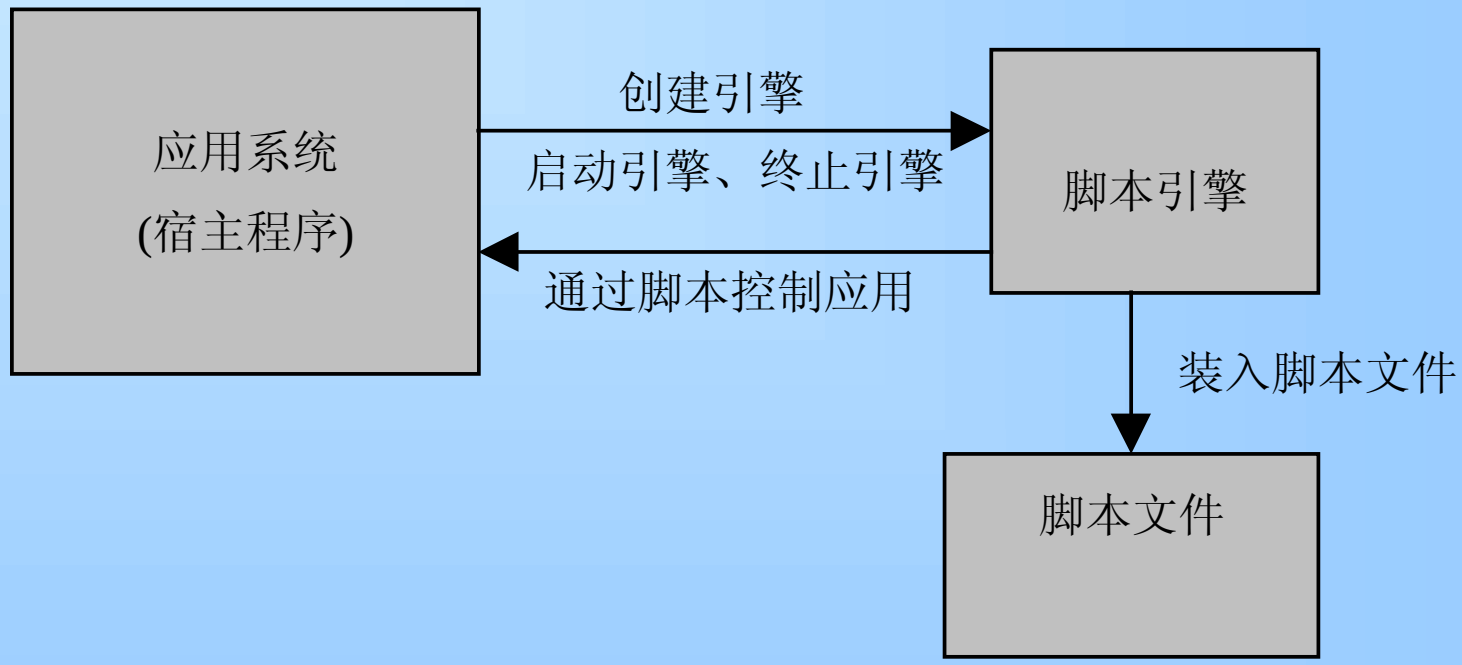
- object hierarchy或object model
- Application对象，全局对象，可创建
- 大多数对象都是不可创建的
- 层次之间的关联
  - 集合与元素的关系，通过枚举器访问
  - 属性关系，下级对象是上级对象的一个属性

自动化控制器(automation controller)

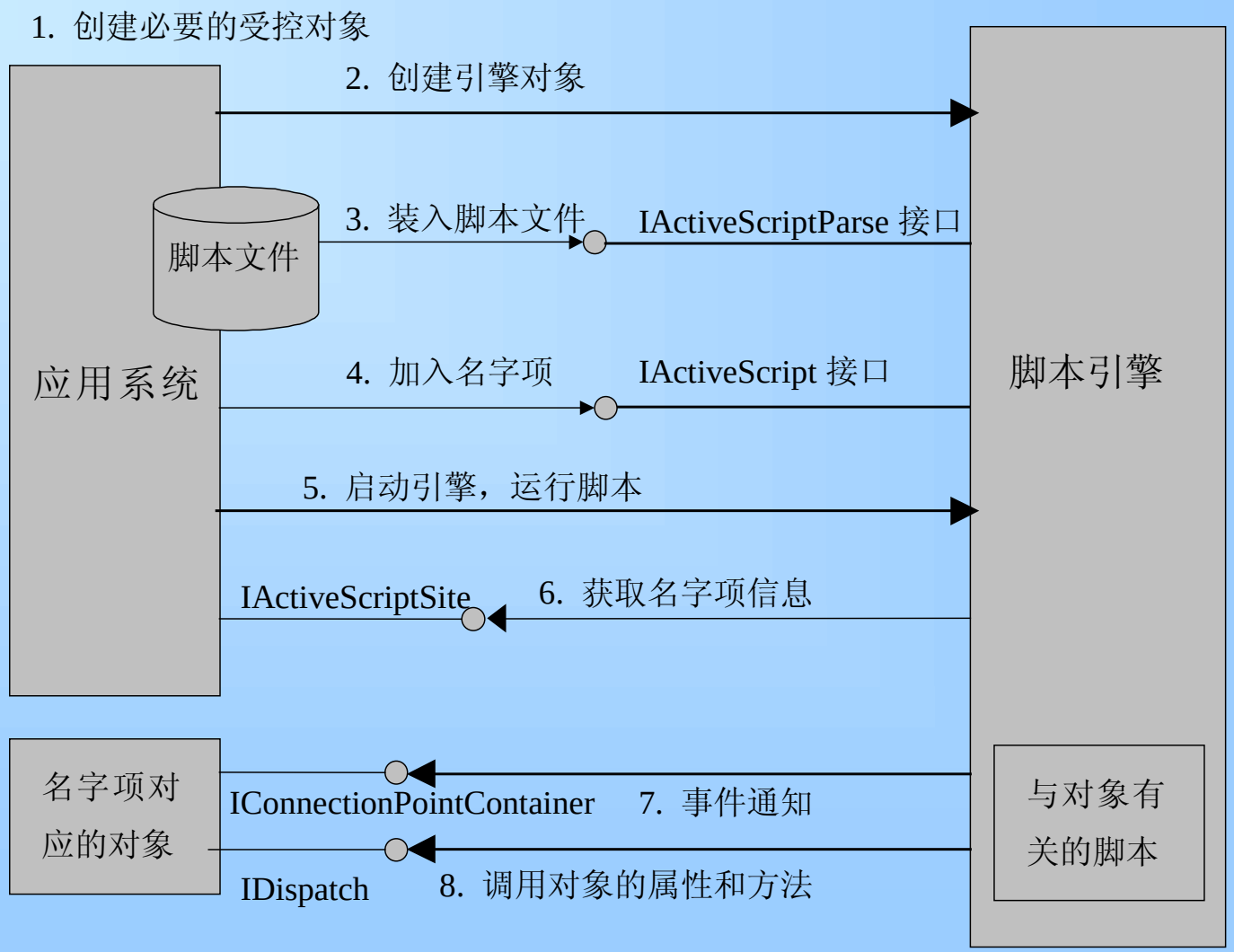
- 自动化控制器也是一个COM客户，功能强大的自动化控制器往往考虑以下特性：
  - 对象创建机制。
  - 对象析构机制。
  - 如何连接到一个已经在运行的自动化对象。
  - 如何访问自动化对象的属性。
  - 如何调用自动化对象的方法。
  - 提供事件处理机制。
  - 从一个接口转移到另一个接口上。
  - 向用户提供UI形式的对象类型信息。

- VB、VBScript等

脚本技术：
应用系统、脚本引擎和脚本文件三者关系



脚本技术

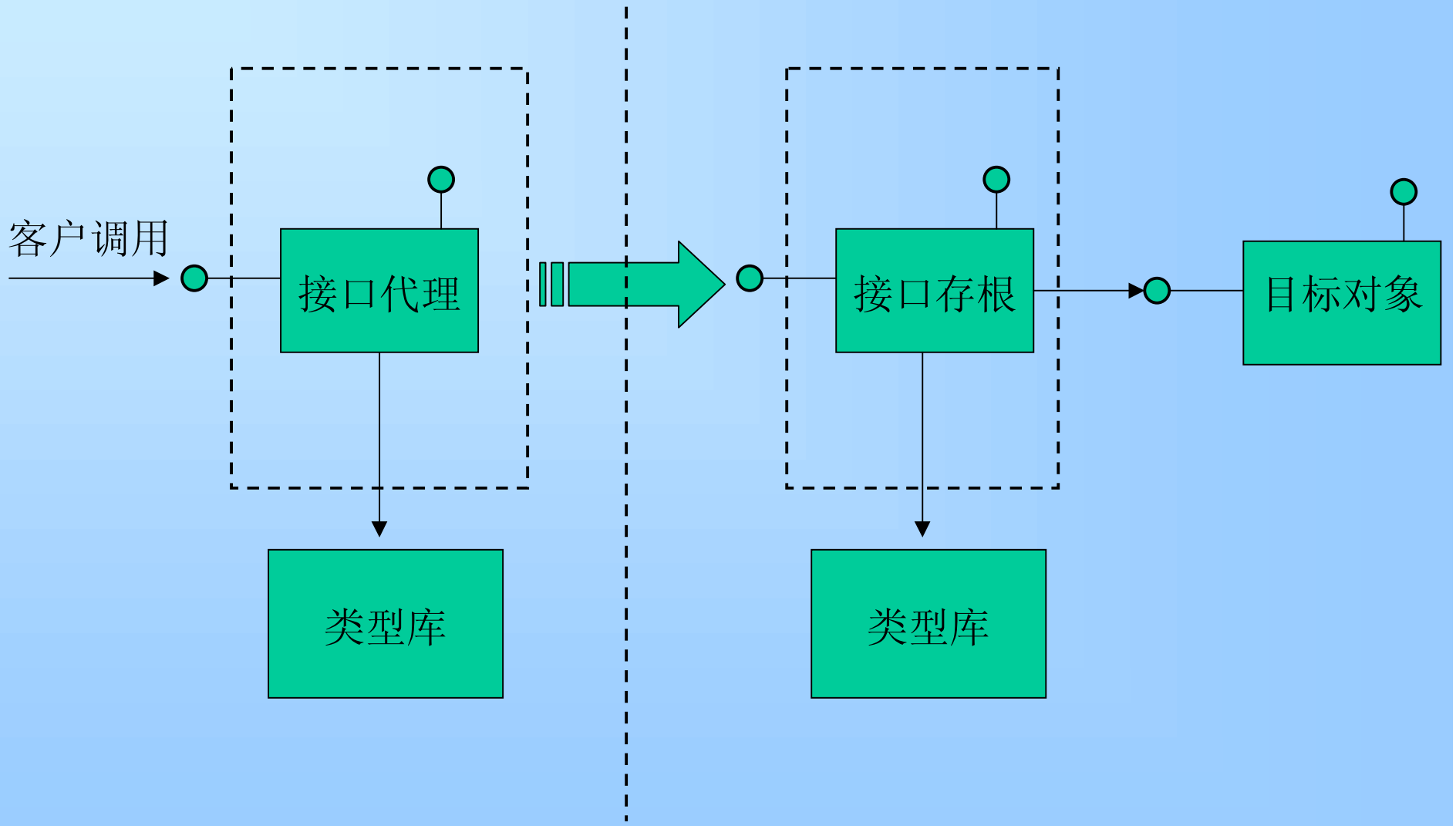

1. 创建必要的受控对象
2. 创建引擎对象
脚本文件
3. 装入脚本文件
IActiveScriptParse 接口
应用系统
4. 加入名字项
IActiveScript 接口
脚本引擎
5. 启动引擎，运行脚本
IActiveScriptSite
6. 获取名字项信息
名字项对应的对象
IConnectionPointContainer
7. 事件通知
与对象有关的脚本
IDispatch
8. 调用对象的属性和方法

自动化对象的marshaling

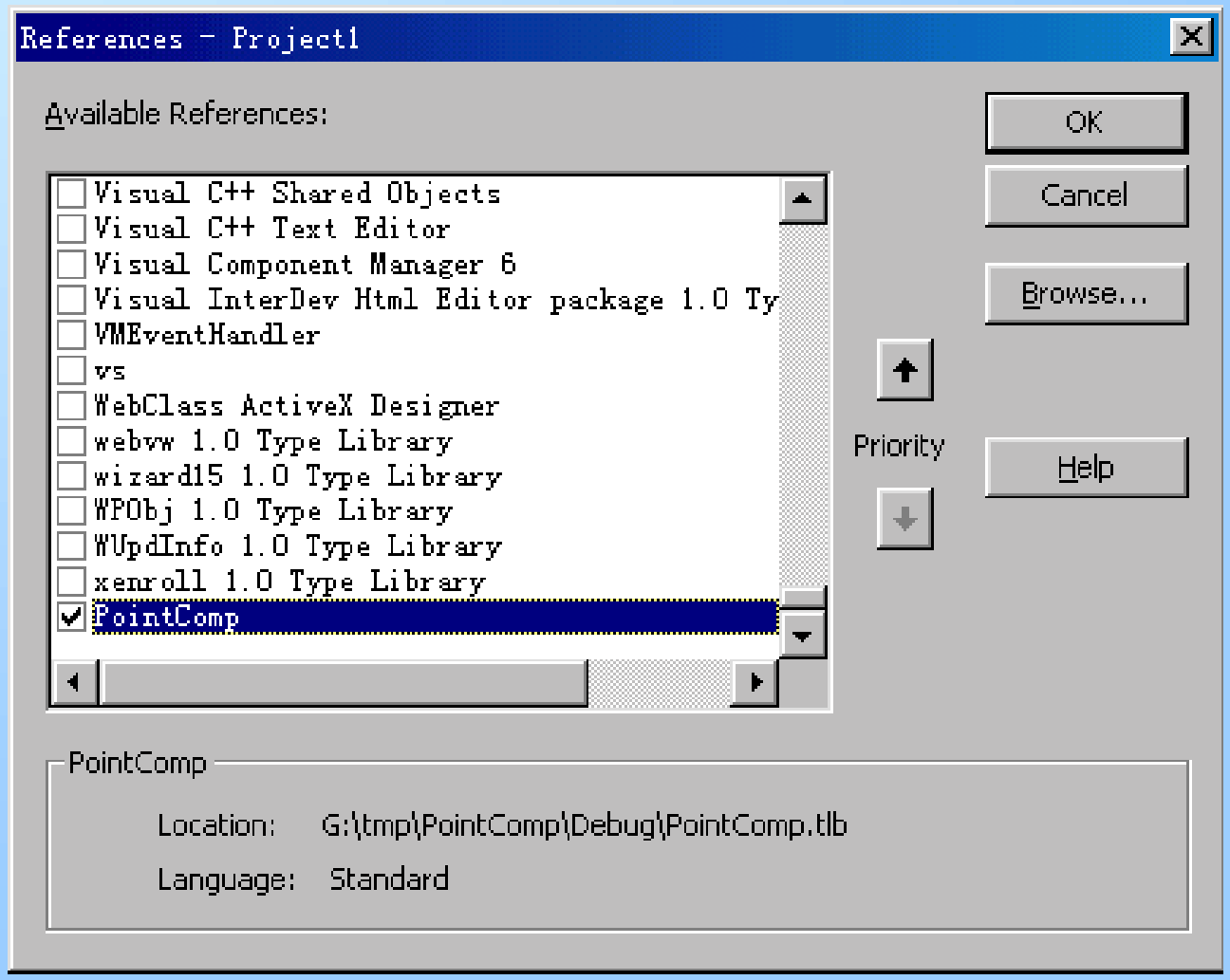
- Universal marshaler
  - CLSID：{00020424-0000-0000-C000-000000000046}
  - 只要接口中的数据类型是自动化兼容的，就可以使用。接口可以不是自动化接口
  - 在描述接口时，加上属性oleautomation或者dual
- 注册：

  [HKCR\Interface\<interface-iid>]
  @="IMyInterface"
  [HKCR\Interface\ <interface-iid>\ProxyStubClsid32]
  @="{00020424-0000-0000-C000-000000000046}"
  [HKCR\Interface\ <interface-iid>\ProxyStubClsid]
  @="{00020424-0000-0000-C000-000000000046}“
  [HKCR\Interface\ <interface-iid>\TypeLib]
  @=" <typelib-id>“
  Version="1.0"

Universal marshaler示意图



在Visual Basic中使用自动化对象(一)

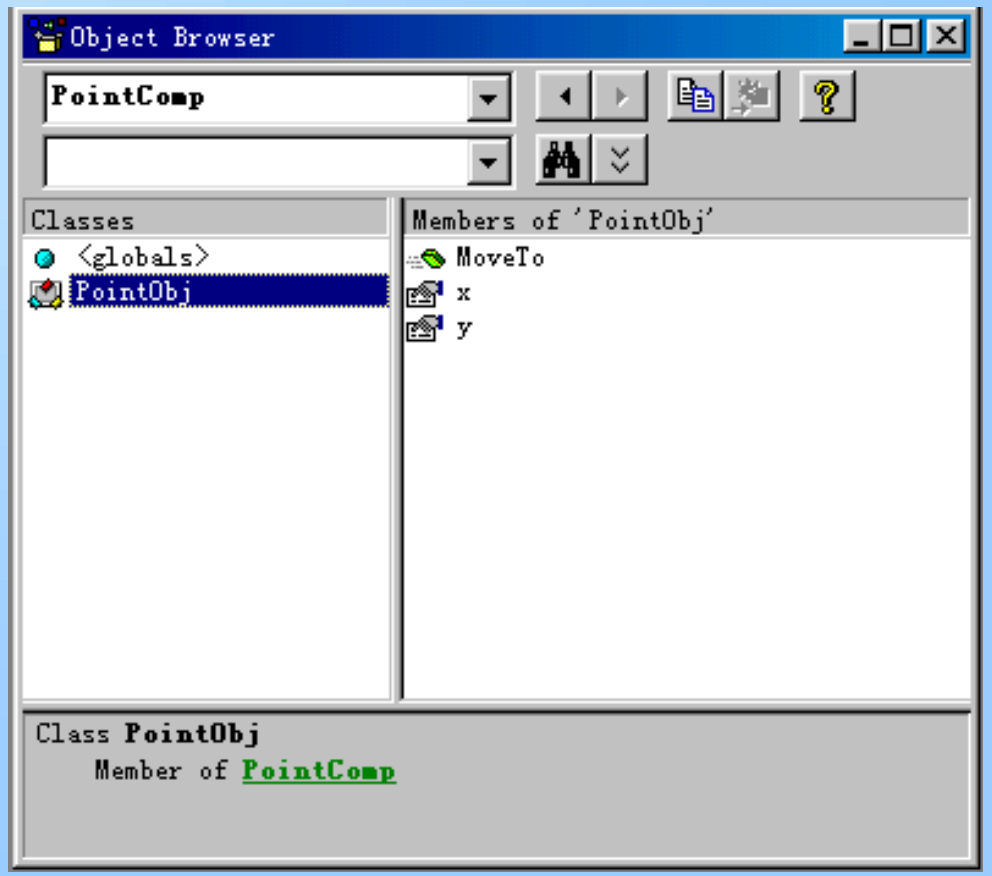


在Visual Basic中使用自动化对象(二)

```
Dim PointObject As New PointComp.PointObj

PointObject.x = 100
PointObject.y = 100

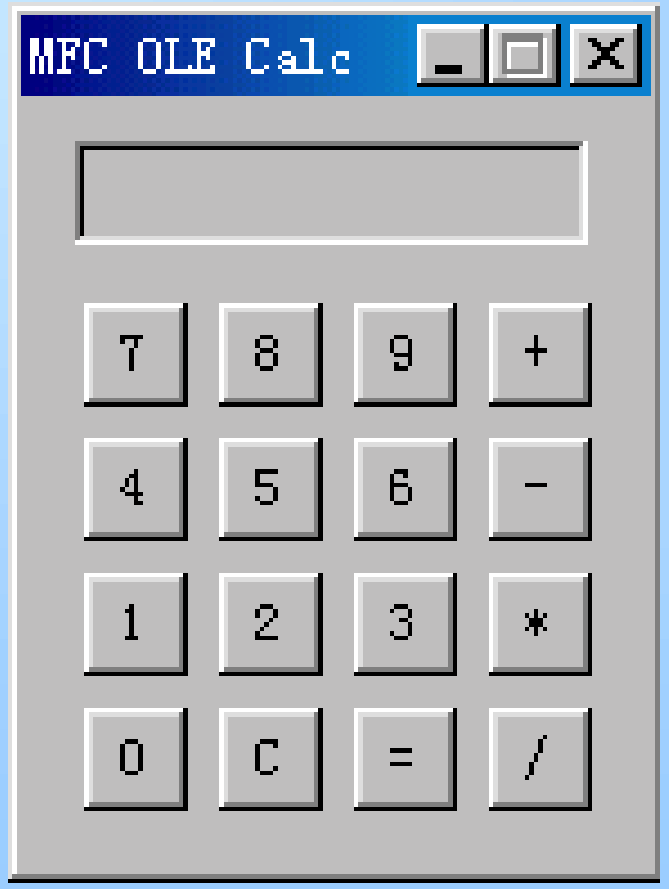
……
PointObject.MoveTo(200, 150)
```



- 对象浏览器

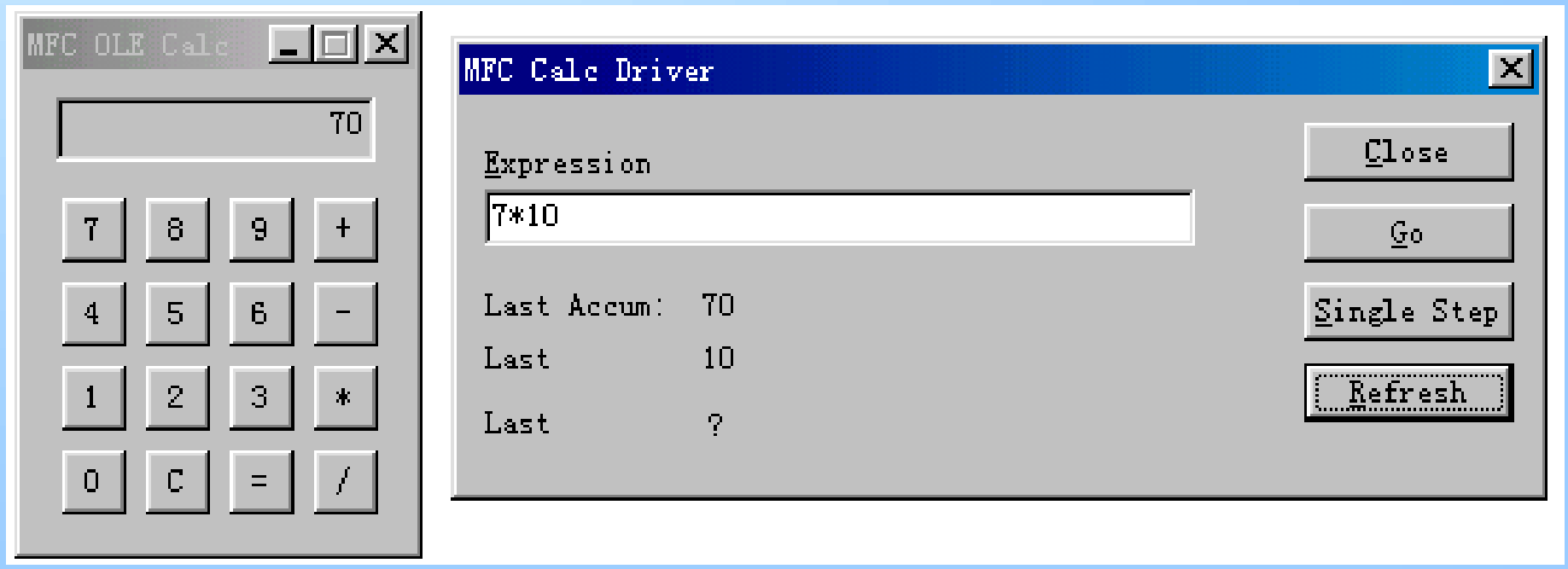
例子程序(一)

- 程序1——具有计算器功能的自动化对象



例子程序(二)

- 程序2——自动化客户程序



例子程序(三)

- 程序3——在Excel中使用计算器对象

