COM：moniker、UDT、control

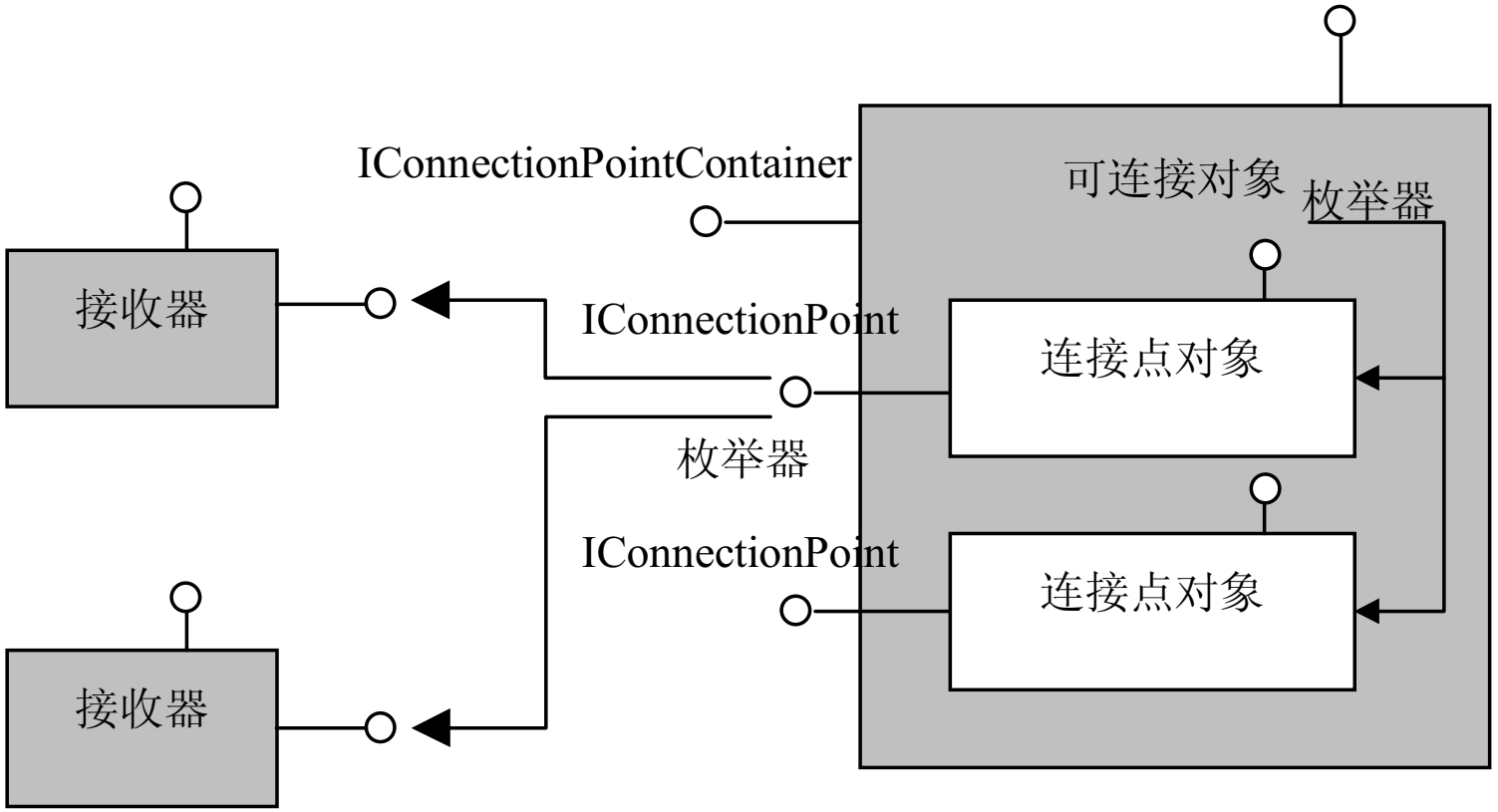
潘爱民

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

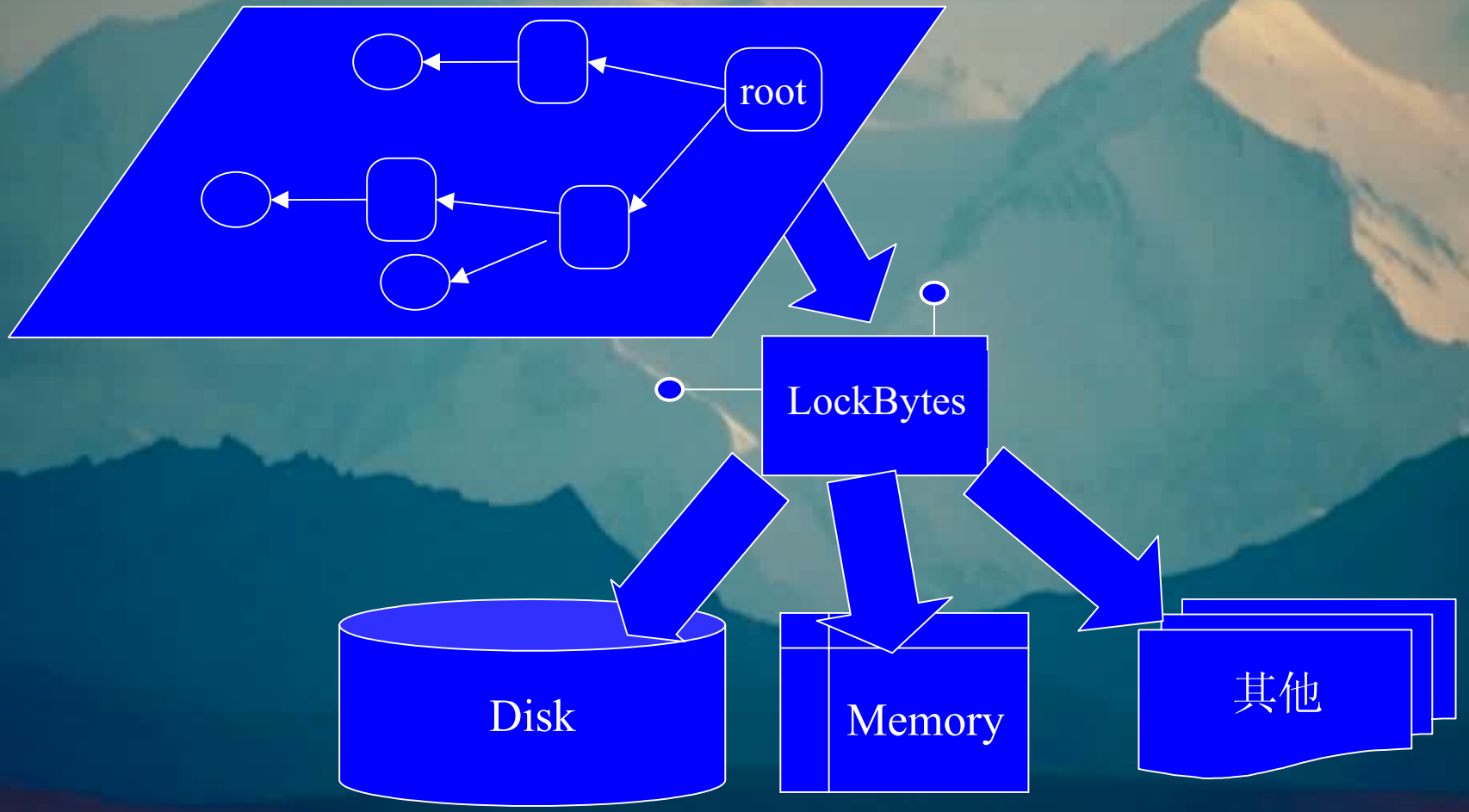
内容

- 复习：COM可连接对象、结构化存储
- COM命名服务：moniker
- UDT：统一数据传输
- Active Control

复习：可连接对象的基本结构



复合文档模型



命名和绑定技术(moniker)

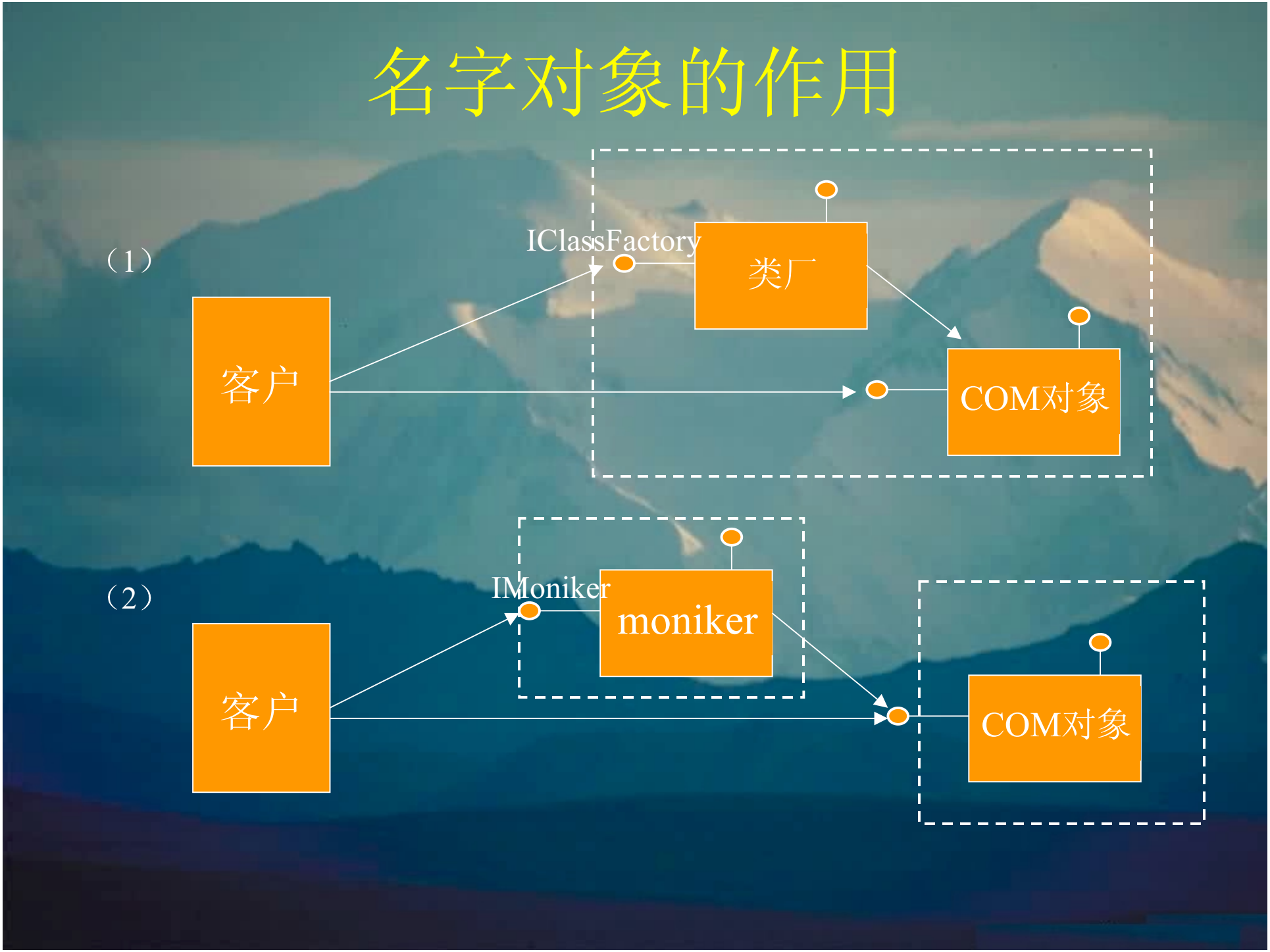
内容：

- 名字技术基础
- IMoniker接口
- 复合名字对象
- COM名字对象分类和应用

名字技术基础

- Moniker：名字对象(也是COM对象)为组件对象提供了符号化的表示方法
- 命名
  - 名字空间
- 绑定：
  - 对象的状态：激活状态或者运行状态、被动状态
  - 绑定：使对象从被动态自动进入运行态 —— 激活、连接
  - 所以也被称为“永久智能对象”

名字对象的作用

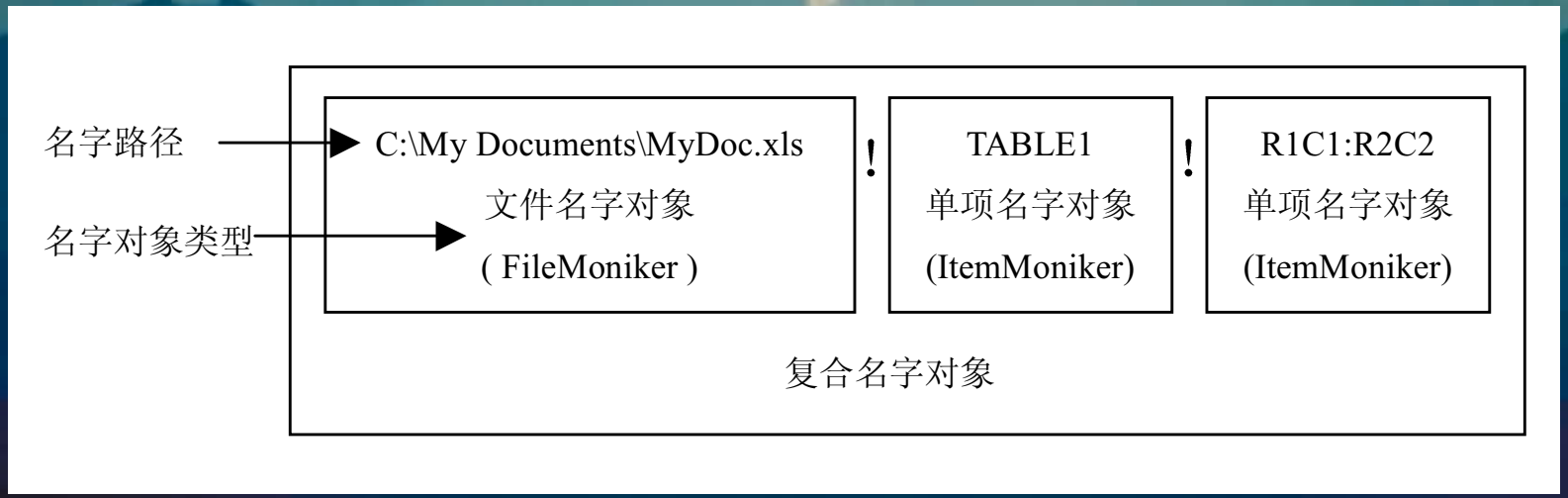


概念：COM名字对象

- 名字对象与文件名的比较
  - 名字对象表达的是com对象——智能启动
  - 文件名表达的是文件
- 名字对象封装了组件对象的状态处理
  - 封装性带来了一致性和多态性
  - 标准接口IMoniker
- 客户通过名字对象建立与com对象的连接
  - 名字对象是客户与对象之间的桥梁

使用名字对象：绑定过程

- 客户创建名字对象
  - API函数，如CreateFileMoniker
- 绑定到名字对象所指的对象
  - 调用IMoniker::BindToObject
- 举例：



IMoniker接口

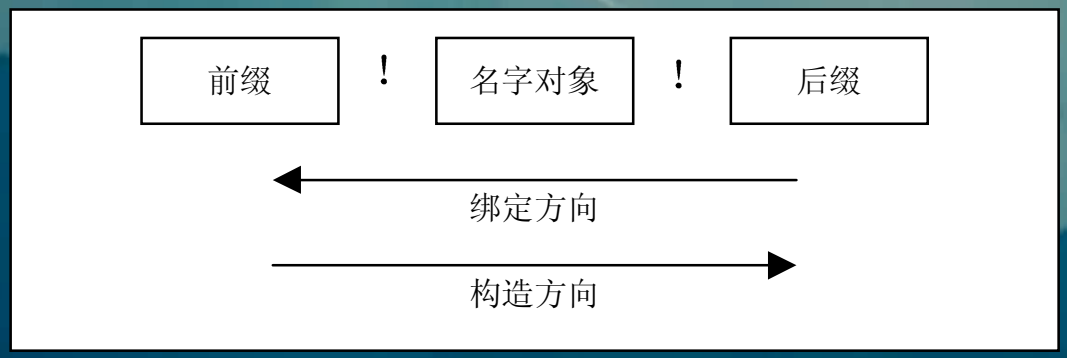
```
class IMoniker : public IPersistStream
{
public :
virtual HRESULT BindToObject(IBindContext *pbc, IMoniker *pmkToLeft, REFIID riid, void **ppvObj) = 0;
virtual HRESULT BindToStorage(IBindContext *pbc, IMoniker *pmkToLeft, REFIID riid, void **ppvObj) = 0;
virtual HRESULT Reduce (IBindContext *pbc, DWORD dwReduceHowFar,
                        IMoniker **ppmkToLeft, IMoniker **ppmkReduced) = 0;
virtual HRESULT ComposeWith(IMoniker *pmkRight, BOOL fOnlyIfNotGeneric, IMoniker **ppmkComposite) = 0;
virtual HRESULT Enum(BOOL fForward, IEnumMoniker **ppEnum) = 0;
virtual HRESULT IsEqual(IMoniker *pmkOtherMoniker) = 0;
virtual HRESULT Hash(DWORD *pdwHash) = 0;
virtual HRESULT IsRunning(IBindContext *pbc, IMoniker *pmkToLeft, IMoniker *pmkNewlyRunning) = 0;
virtual HRESULT GetTimeOfLastChange(IBindContext *pbc, IMoniker *pmkToLeft,FILETIME *pFileTime) = 0;
virtual HRESULT Inverse(IMoniker **ppmk) = 0;
virtual HRESULT CommonPrefixWith(IMoniker *pmkOther, IMoniker **ppmkPrefix) = 0;
virtual HRESULT RelativePathTo(IMoniker *pmkOther, IMoniker **ppmkRelPath) = 0;
virtual HRESULT GetDisplayName(IBindContext *pbc, IMoniker *pmkToLeft, LPOLESTR *ppszDisplayName) = 0;
virtual HRESULT ParseDisplayName(IBindContext *pbc, IMoniker *pmkToLeft,
                        LPOLESTR pszDisplayName, ULONG *pchEaten, IMoniker **ppmkOut) = 0;
virtual HRESULT IsSystemMoniker(DWORD *pdwMksys) = 0;
};
```

名字管理

```
HRESULT IsEqual(IMoniker *pmkOtherMoniker);
HRESULT Hash(DWORD *pdwHash);
HRESULT IsRunning(IBindContext *pbc, IMoniker *pmkToLeft,
                  IMoniker *pmkNewlyRunning);
HRESULT GetTimeOfLastChange(IBindContext *pbc, IMoniker *pmkToLeft,
                  FILETIME *pFileTime);
```

IMoniker绑定

HRESULT BindToObject(IBindContext *pbc, IMoniker *pmkToLeft,
REFIID riid, void **ppvObj);
HRESULT BindToStorage(IBindContext *pbc, IMoniker *pmkToLeft,
REFIID riid, void **ppvObj);



复合名字对象的管理

```
HRESULT Enum(BOOL fForward, IEnumMoniker **ppEnum);
HRESULT Inverse(IMoniker **ppmk);
HRESULT IsSystemMoniker(DWORD *pdwMksys);
HRESULT CommonPrefixWith(IMoniker *pmkOther, IMoniker **ppmkPrefix);
HRESULT RelativePathTo(IMoniker *pmkOther, IMoniker **ppmkRelPath);
HRESULT ComposeWith(IMoniker *pmkRight, BOOL fOnlyIfNotGeneric,
                    IMoniker **ppmkComposite);
HRESULT Reduce (IBindContext *pbc, DWORD dwReduceHowFar,
                    IMoniker **ppmkToLeft, IMoniker **ppmkReduced);
```

系统名字对象

```
typedef enum tagMKSYS
{
        MKSYS_NONE = 0,
        MKSYS_GENERICCOMPOSITE = 1,
        MKSYS_FILEMONIKER = 2,
        MKSYS_ANTIMONIKER = 3,
        MKSYS_ITEMMONIKER = 4,
        MKSYS_POINTERMONIKER = 5,
        //  MKSYS_URLMONIKER = 6,
        MKSYS_CLASSMONIKER = 7
} MKSYS;
```

名字解析

```
HRESULT GetDisplayName(IBindContext *pbc, IMoniker *pmkToLeft,
                       LPOLESTR *ppszDisplayName);
HRESULT ParseDisplayName(IBindContext *pbc,
                       IMoniker *pmkToLeft, LPOLESTR pszDisplayName,
                        ULONG *pchEaten, IMoniker **ppmkOut);
```

- 显示名分隔符：“！”、“\”、“/”、“:”或“[”

复合名字对象

- 通用复合名字对象

```
HRESULT CreateGenericComposite(IMoniker *pmkFirst,
                               IMoniker *pmkRest, IMoniker **ppmkComposite);
```

- 按从左到右的顺序组合，满足结合律
- 举例：文档内部的电子表格

ROT表

- COM使用ROT表管理当前系统中正在运行的、已经被注册的名字对象

```
class IRunningObjectTable : public IUnknown
{
    virtual HRESULT Register(DWORD grfFlags, IUnknown *pUnkObject,
                             IMoniker *pmkObject, DWORD *pdwRegister) = 0;
    virtual HRESULT Revoke(DWORD dwRegister) = 0;
    virtual HRESULT IsRunning(IMoniker *pmkObject) = 0;
    virtual HRESULT GetObject(IMoniker *pmkObject, IUnknown **ppUnkObject) = 0;
    virtual HRESULT NoteChangeTime(DWORD dwRegister, FILETIME *pft) = 0;
    virtual HRESULT GetTimeOfLastChange(IMoniker *pmkObject, FILETIME *pft) = 0;
    virtual HRESULT EnumRunning(IEnumMoniker **ppEnum) = 0;
};
```

- 客户调用GetRunningObjectTable函数访问ROT表

绑定环境对象

- 也是COM实现的系统对象

```
WINOLEAPI CreateBindCtx(DWORD reserved, IBindCtx **ppbc);
class IBindCtx : public IUnknown
{
virtual  HRESULT RegisterObjectBound(IUnknown *punk) = 0;
virtual  HRESULT RevokeObjectBound(IUnknown *punk) = 0;
virtual  HRESULT ReleaseBoundObjects()= 0;
virtual  HRESULT SetBindOptions(BIND_OPTS *pbindopts) = 0;
virtual  HRESULT GetBindOptions(BIND_OPTS *pbindopts) = 0;
virtual  HRESULT GetRunningObjectTable(IRunningObjectTable **pprot) = 0;
virtual  HRESULT RegisterObjectParam(LPOLESTR pszKey, IUnknown *punk) = 0;
virtual  HRESULT GetObjectParam(LPOLESTR pszKey, IUnknown **ppunk) = 0;
virtual  HRESULT EnumObjectParam(IEnumString **ppenum) = 0;
virtual  HRESULT RevokeObjectParam(LPOLESTR pszKey) = 0;
};
```

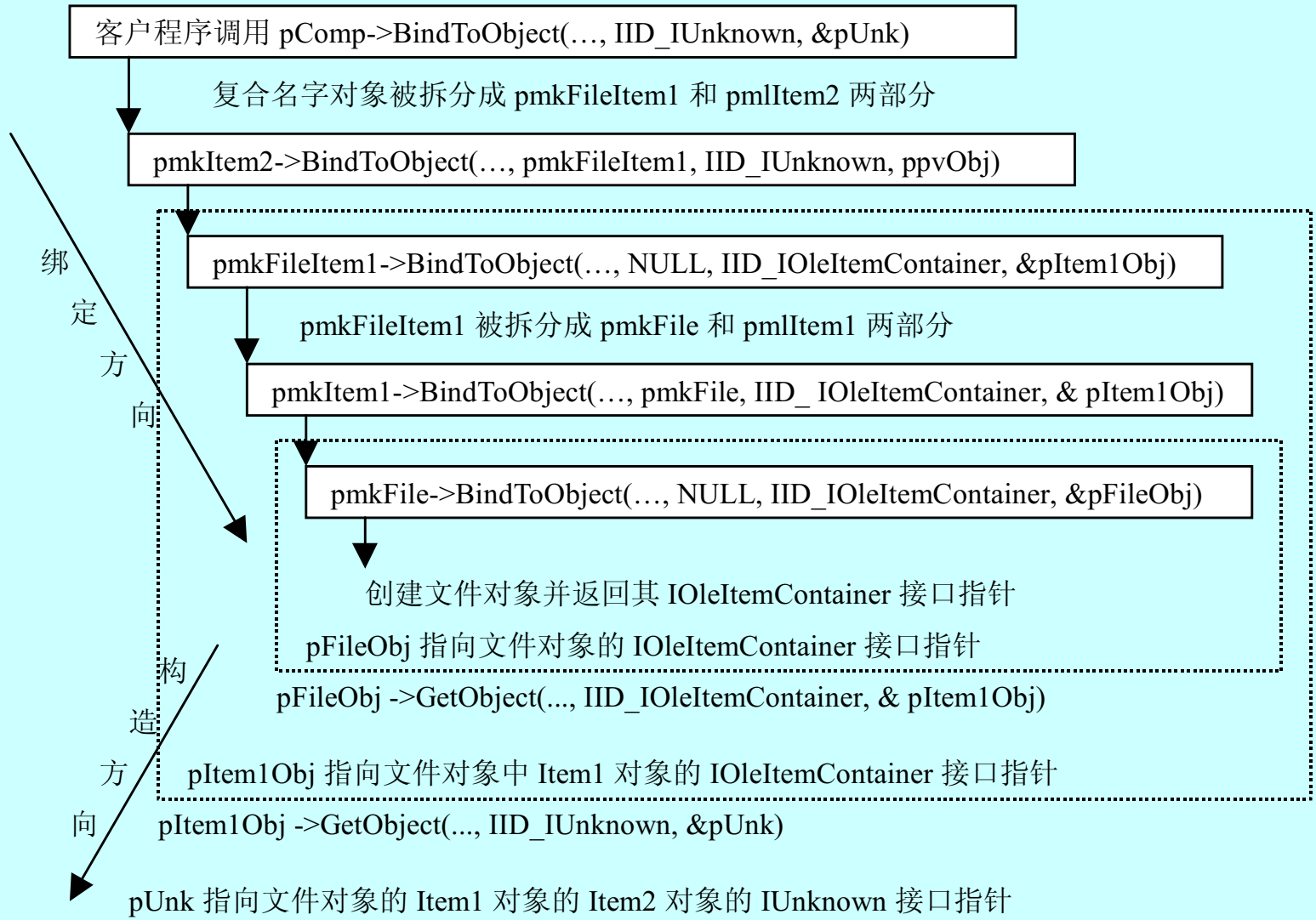
绑定环境对象(续)

- 管理已被绑定的对象：
  - RegisterObjectBound、RevokeObjectBound、ReleaseBoundObjects
- 管理绑定参数：(文件访问模式、超时设置等)
  - SetBindOptions、GetBindOptions
- 管理绑定过程中的对象参数：
  - RegisterObjectParam、GetObjectParam、
  - EnumObjectParam、RevokeObjectParam

复合名字对象绑定过程的剖析

- IMoniker::BindToObject绑定过程：
  - （1）检查ROT表。
  - （2）分解。pmkRight : pmkLeft，最右边部分分离。
  - （3）调用pmkRight->BindToObject(...,pmkLeft, ...)。
  - （4）执行pmkRight->BindToObject。
    - 如果pmkLeft为简单名字对象，则可终止循环；
    - 否则， pmkRight往往要调用pmkLeft->BindToObject，从而形成自右向左的循环绑定过程。
- 举例：File!Item1!Item2

File!Item1!Item2的绑定和构造过程



IOleItemContainer接口

```
class IParseDisplayName : public Iunknown
{
virtual HRESULT ParseDisplayName(IBindContext *pbc, LPOLESTR pszDisplayName,
                        ULONG *pchEaten, IMoniker **ppmkOut) = 0;
};
class IOleContainer : public IParseDisplayName
{
virtual HRESULT EnumObjects(DWORD grfFlags, LPENUMUNKNOWN *ppEnum) = 0;
virtual HRESULT LockContainer(BOOL fLock) = 0;
};
class IOleItemContainer : public IOleContainer
{
virtual HRESULT GetObject(LPOLESTR pszItem, DWORD dwSpeedNeeded,
                        IBindCtx *pbc, REFIID riid, void **ppv) = 0;
virtual HRESULT GetObjectStorage(LPOLESTR pszItem, IBindCtx *pbc,
                        REFIID riid, void **ppvStorage) = 0;
virtual HRESULT IsRunning(LPOLESTR pszItem) = 0;
};
```

COM名字对象分类

- COM提供的系统名字对象
- URL名字对象
- 自定义名字对象

系统名字对象

- 文件名字对象(File Moniker)

  WINOLEAPI CreateFileMoniker(LPCOLESTR lpszPathName, IMoniker **ppmk);

- 复合名字对象(Composite Moniker)

  WINOLEAPI CreateGenericComposite(IMoniker *pmkFirst, IMoniker *pmkRest,
  IMoniker **ppmkComposite);

- 单项名字对象(Item Moniker)

  WINOLEAPI CreateItemMoniker(LPCOLESTR lpszDelim, LPCOLESTR lpszItem,
  IMoniker **ppmk);

- 举例：

  CreateFileMoniker(“File”, &pmkFile);
  CreateItemMoniker(“!”, “Item1”, &pmkItem1);
  pmkFile->ComposeWith(pmkItem1, FALSE, &pmkComp1);
  CreateItemMoniker(“!”, “Item2”, &pmkItem2);
  pmkComp1->ComposeWith(pmkItem2, FALSE, &pmkComp2);

系统名字对象(续)

- 反-名字对象(Anti-moniker)

  WINOLEAPI CreateAntiMoniker(IMoniker **ppmk);

- 指针名字对象(Pointer Moniker)

  WINOLEAPI CreatePointerMoniker(IUnknown *punk, IMoniker **ppmk);

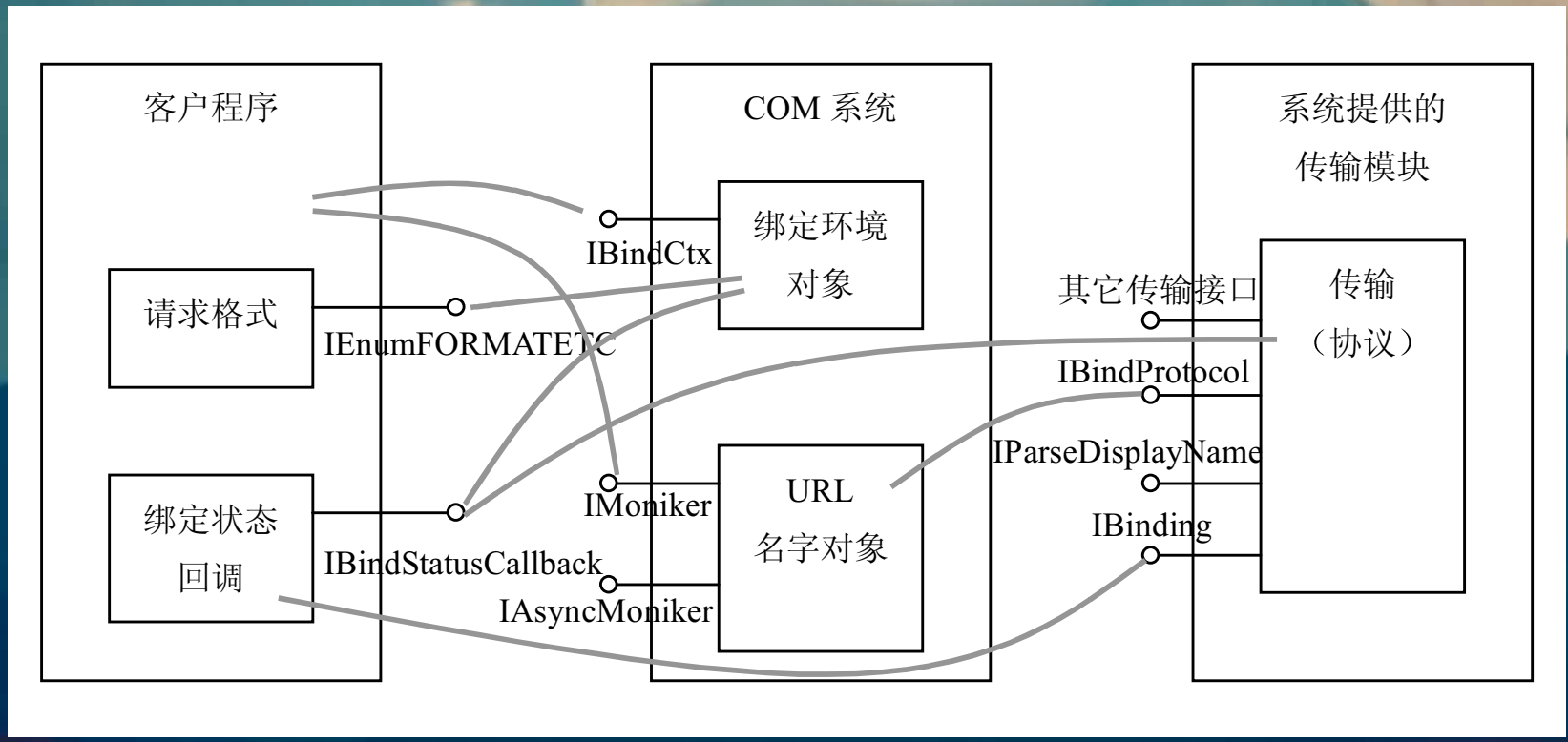
- 类名字对象(Class Moniker)

  WINOLEAPI CreateClassMoniker (REFCLSID rclsid, IMoniker **ppmk);

URL名字对象

- 异步名字对象。

  标志是实现了IAsyncMoniker接口



自定义名字对象

- 由于文件名字对象、单项名字对象、复合名字对象和类名字对象所实现的组合功能非常强大，所以自定义名字对象很少使用
- 由于IMoniker接口成员众多，我们根据需要实现其中的成员
- 同时提供一条创建自定义名字对象的途径

名字对象的应用与发展

- 是OLE链接对象的重要技术保障
- COM+又扩充了新的名字对象
  - 比如queue:、new:
- 作为客户与COM对象之间连接的一种强有力的手段
- VBScript中访问对象的主要机制

MFC对名字对象的支持

- COleLinkingDoc
- 在OLE服务程序中，三个操作涉及到名字对象：新创建文档然后执行保存操作、打开复合文件操作、执行剪贴板拷贝操作

  * COleLinkingDoc也实现了IOleItemContainer接口
- 在OLE客户程序中，四个操作涉及到名字对象：客户链接到一个对象、保存文档的时候、客户程序装入文档的时候、激活链接对象的时候

例子

- VC带的例子：OClient和Scribble



统一数据传输（UDT）

内容：

- 数据传输机制
- 通过剪贴板传输数据
- 拖-放数据传输协议

数据交换与传输协议的分离

- 数据格式的统一
  - 数据对象：信息实体，通过IDataObject接口暴露内部信息
- 传输协议
  - 一个应用程序如何得到另一个应用程序所提供的数据对象即IDataObject接口指针
  - 在Windows平台上，最基本的传输协议为剪贴板、拖-放，应用程序通常利用这两种协议获得数据对象

统一数据传输：应用

- 剪贴板技术
  - 三个标准操作：剪切、复制、粘帖
- 拖-放技术
  - 一种简便的对象移动或拷贝操作，比剪贴板操作更为方便，而且只涉及到源和目标两方

数据传输机制

- 数据结构FORMATETC和STGMEDIUM
- 数据对象和IDataObject接口
- 通报连接机制

数据结构FORMATETC

```
typedef WORD CLIPFORMAT;
typedef struct  tagFORMATETC
{
        CLIPFORMAT cfFormat;      // 剪贴板数据格式
        DVTARGETDEVICE *ptd;      // 设备有关的信息
        DWORD dwAspect;           // 图形数据的表现方式
        LONG lindex;              // dwAspect成员的补充
        DWORD tymed;              // 数据的存储介质
} FORMATETC;
```

标准数据格式

标准剪贴板格式表

预定义值	格式说明	预定义值	格式说明
CF_TEXT	文本格式	CF_DIB	设备无关的位图格式
CF_BITMAP	位图格式	CF_PALETTE	Windows 调色板格式
CF_METAFILEPICT	图元文件格式	CF_PENDATA	Windows 笔划数据格式
CF_SYLK	Windows 符号链接格式	CF_RIFF	资源交换文件格式
CF_DIF	Windows 数据交换格式	CF_WAVE	波形语音文件格式
CF_TIFF	TIF 图象格式	CF_UNICODETEXT	Unicode 文本格式
CF_OEMTEXT	OEM 文本格式	CF_ENHMETAFILE	增强的图元文件格式

填充FORMATETC结构的宏

```
//Macro to set all FormatEtc fields
#define SETFORMATETC(fe, cf, asp, td, med, li)   \
    ((fe).cfFormat=cf, \
    (fe).dwAspect=asp, \
    (fe).ptd=td, \
    (fe).tymed=med, \
    (fe).lindex=li)

//Macro to set interesting FormatEtc fields defaulting the others.
#define SETDEFAULTFORMATETC(fe, cf, med)  \
    ((fe).cfFormat=cf, \
    (fe).dwAspect=DVASPECT_CONTENT, \
    (fe).ptd=NULL, \
    (fe).tymed=med, \
    (fe).lindex=-1)
```

数据结构STGMEDIUM

```
typedef struct tagSTGMEDIUM
{
    DWORD tymed;                          // 存储介质的类型
    union
    {
        HBITMAP hBitmap;
        HMETAFILEPICT hMetaFilePict;
        HENHMETAFILE hEnhMetaFile;
        HGLOBAL hGlobal;
        LPOLESTR lpszFileName;
        IStream *pstm;
        IStorage *pstg;
    } u;                                  // 数据真正的位置
    IUnknown *pUnkForRelease;             // 控制介质资源的释放
} STGMEDIUM;
```

资源释放—ReleaseStgMedium 函数

- 如果pmedium-> pUnkForRelease为非NULL，那么分两步：
  - 对于TYMED_FILE介质类型，它用标准内存管理器释放文件名字符串，对于TYMED_ISTREAM和TYMED_ISTORAGE介质类型，调用IStream::Release或IStorage::Release，其它类型跳过这一步；
  - 调用pmedium-> pUnkForRelease->Release。
- 如果pmedium-> pUnkForRelease为NULL，那么按不同的介质类型执行不同的释放处理：
  - 对于TYMED_HGLOBAL类型调用GlobalFree函数释放；
  - 对于TYMED_GDI类型调用DeleteObject函数释放；
  - 对于TYMED_ENHMF类型删除增强的图元文件；
  - 对于TYMED_MFPICT类型删除图元文件；
  - 对于TYMED_FILE类型先删除文件，再用标准内存管理器释放文件名字符串；
  - 对于TYMED_ISTREAM调用IStream::Release成员函数释放；
  - 对于TYMED_ISTORAGE调用IStorage::Release成员函数释放。

数据对象

- IDataObject接口

```
class IDataObject : public IUnknown
{
public:
    virtual HRESULT GetData( FORMATETC *pformatetcIn, STGMEDIUM *pmedium) = 0;
    virtual HRESULT GetDataHere( FORMATETC *pformatetc,
                        STGMEDIUM *pmedium) = 0;
    virtual HRESULT QueryGetData( FORMATETC *pformatetc) = 0;
    virtual HRESULT GetCanonicalFormatEtc(FORMATETC *pformatectIn,
                        FORMATETC *pformatetcOut) = 0;
    virtual HRESULT SetData( FORMATETC *pformatetc, STGMEDIUM  *pmedium,
                        BOOL fRelease) = 0;
    virtual HRESULT EnumFormatEtc( DWORD dwDirection,
                        IEnumFORMATETC **ppenumFormatEtc) = 0;
    virtual HRESULT DAdvise(FORMATETC *pformatetc,  DWORD advf,
                        IAdviseSink *pAdvSink, DWORD *pdwConnection) = 0;
    virtual HRESULT DUnadvise( DWORD dwConnection) = 0;
    virtual HRESULT EnumDAdvise( IEnumSTATDATA **ppenumAdvise) = 0;
};
```

IDataObject接口的成员(一)

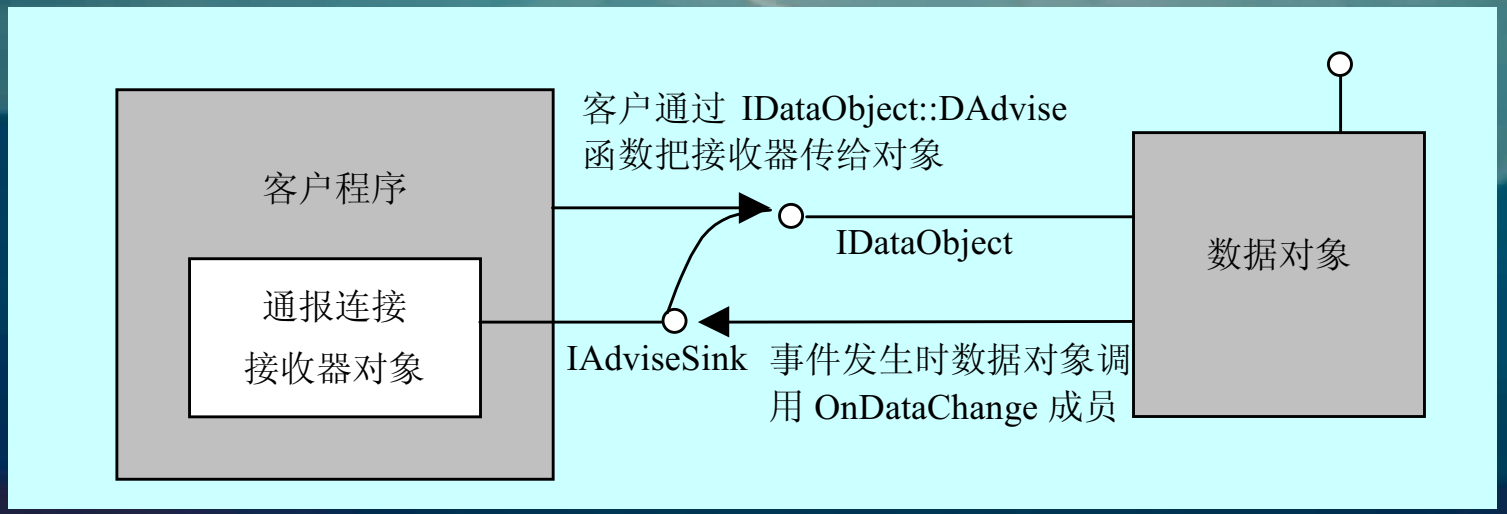
- GetData(FORMATETC *, STGMEDIUM *)
- GetDataHere(FORMATETC *, STGMEDIUM *)
- QueryGetData(FORMATETC *)
- GetCanonicalFormatEtc(FORMATETC *pformatectIn, FORMATETC *pformatetcOut)

IDataObject接口的成员(二)

- SetData(FORMATETC *pformatetc, STGMEDIUM *pmedium, BOOL fRelease)
- EnumFormatEtc(DWORD dwDirection, IEnumFORMATETC **)
- DAdvise(FORMATETC *, DWORD advf, IAdviseSink *, DWORD *)
- DUnadvise(DWORD)
- EnumDAdvise(IEnumSTATDATA **)

通报连接(advisory connection)机制

```
class IAdviseSink : public IUnknown
{
    virtual void OnDataChange(FORMATETC * pformatetc, STGMEDIUM *pmedium) = 0;
    virtual void OnViewChange(DWORD dwAspect, LONG lindex) = 0;
    virtual void OnRename(IMoniker *pmk) = 0;
    virtual void OnSave(void) = 0;
    virtual void OnClose(void) = 0;
};
```



剪贴板传输数据机理(一)

- 常用的7个Win32 API函数

```
BOOL WINAPI OpenClipboard( HWND hWndNewOwner);
BOOL WINAPI CloseClipboard( VOID);
BOOL WINAPI EmptyClipboard(VOID);
HANDLE WINAPI SetClipboardData( UINT uFormat, HANDLE hMem);
HANDLE WINAPI GetClipboardData( UINT uFormat);
BOOL WINAPI IsClipboardFormatAvailable( UINT format);
UINT WINAPI EnumClipboardFormats( UINT format);
```

- 剪贴板是系统全局共享，进程独占方式
- 存储介质仅限于全局内存

剪贴板传输数据机理(二)

- 剪贴板所有者为一个Windows窗口
  - 调用EmptyClipboard时被OpenClipboard指定的窗口

- 延迟供应(delayed rendering)：

  (1)SetClipboardData的参数hMem可以是NULL

  (2) 客户需要数据时，所有者窗口处理
  WM_RENDERFORMAT消息

  (3)所有者窗口被删除之前处理
  WM_RENDERALLFORMATS消息

OLE剪贴板(一)

- OLE API函数

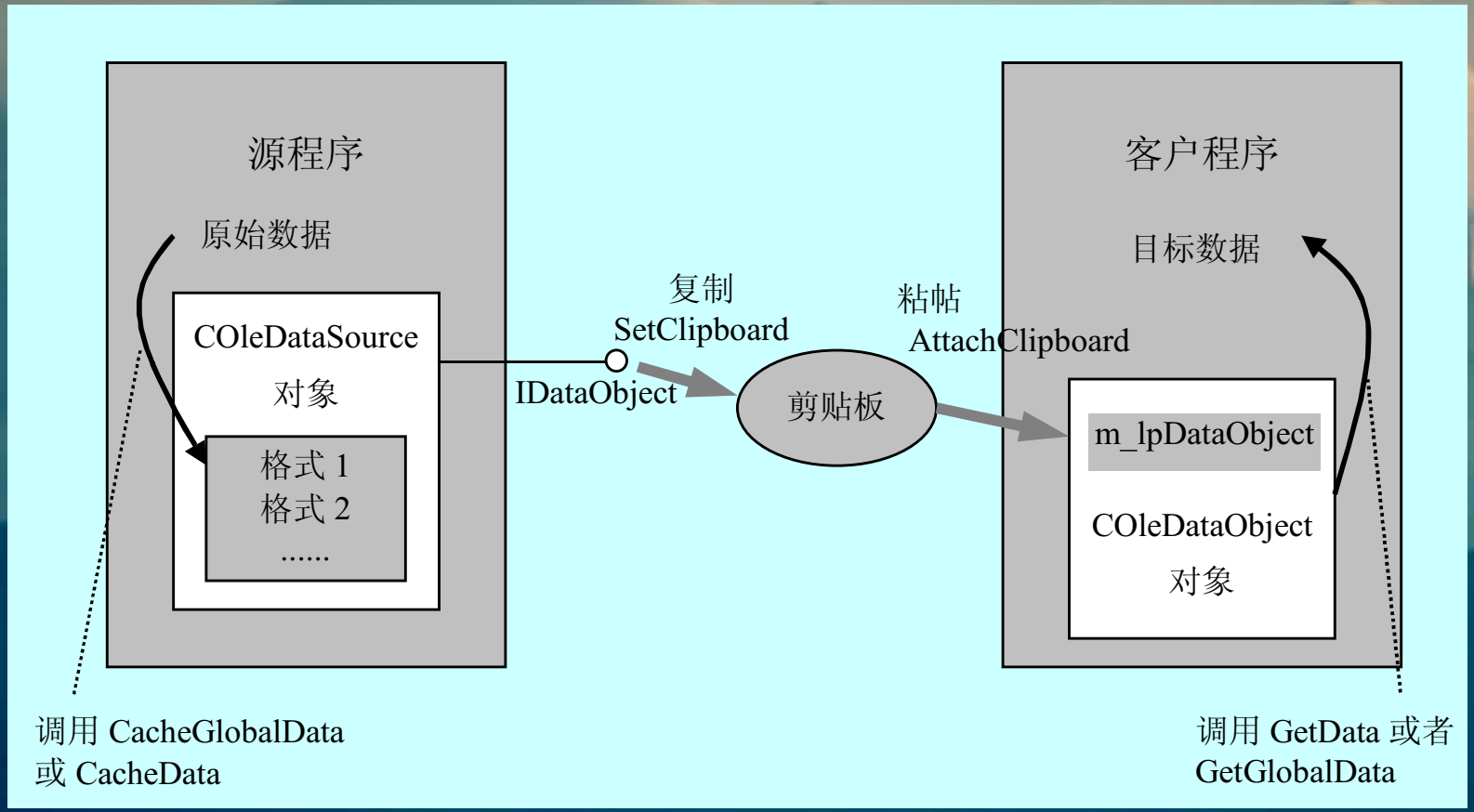
```
WINOLEAPI  OleSetClipboard(IDataObject *pDataObj);
WINOLEAPI  OleGetClipboard(IDataObject ** ppDataObj);
WINOLEAPI  OleFlushClipboard(void);
WINOLEAPI  OleIsCurrentClipboard(IDataObject *pDataObj);
```

- OleSetClipboard内部把所有权交给OLE内部隐藏窗口
- 针对以全局内存作为存储介质的数据格式，OleSetClipboard使用“延迟供应”方式调用SetClipboardData放到剪贴板上
- 清空剪贴板，可调用OleSetClipboard(NULL)

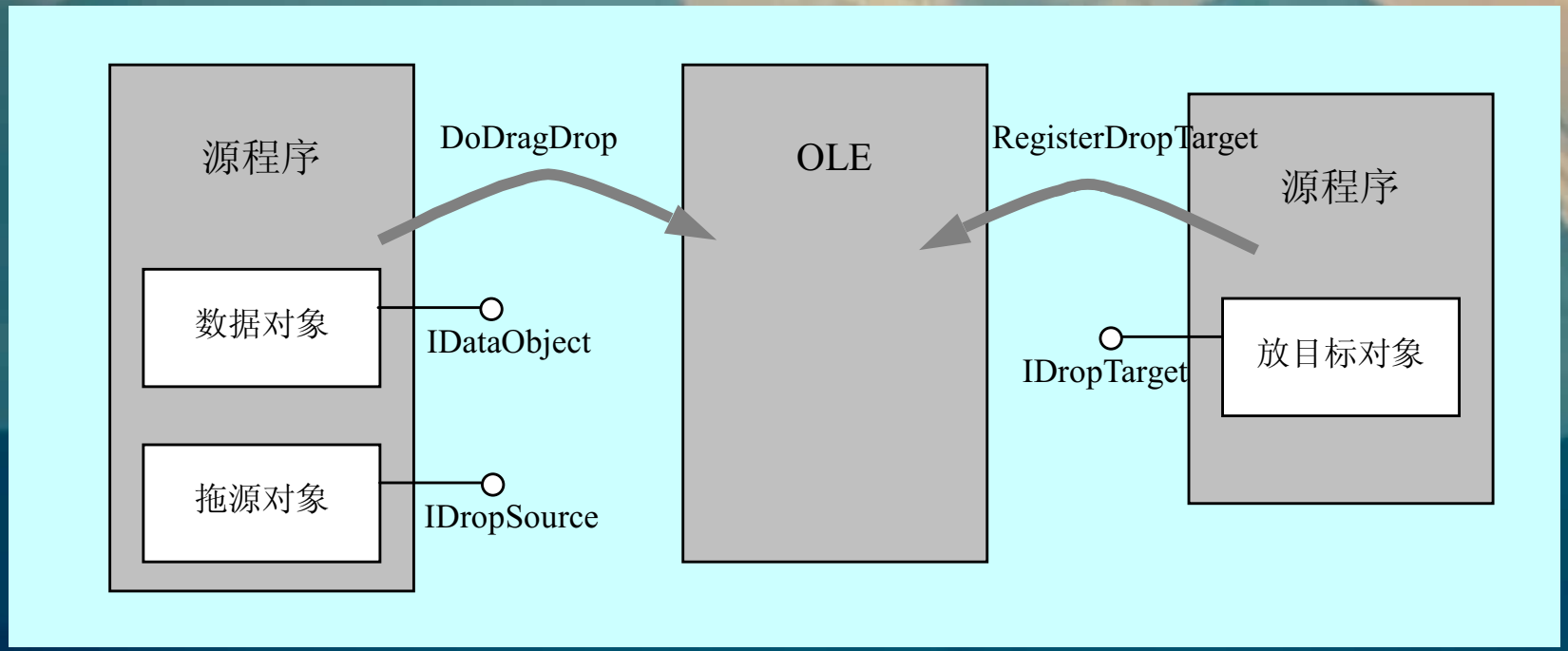
OLE剪贴板(二)

- 客户方调用GetClipboardData只能访问到以全局内存作为存储介质的数据格式
- 使用OleGetClipboard函数可以访问到源数据对象的所有格式
- OleSetClipboard函数返回不同的数据对象
  (1)源数据对象仍在运行，则直接返回(有可能是代理对象)
  (2)源程序调用了OleFlushClipboard函数，OLE创建一个缺省的数据对象，供客户使用
  (3)剪贴板上的数据非数据对象，返回一个缺省数据对象，但数据格式受限制

MFC对剪贴板的支持示意图



拖-放数据传输协议


源程序
DoDragDrop
OLE
RegisterDropTarget
源程序
数据对象
IDataObject
IDropTarget
放目标对象
拖源对象
IDropSource

拖-放数据传输协议：源

- 实现数据对象和“拖源”对象，“拖源”对象实现了接口IDropSource

```
class IDropSource : public IUnknown
{
    virtual HRESULT QueryContinueDrag( BOOL fEscapePressed,
                                       DWORD grfKeyState) = 0;
    virtual HRESULT GiveFeedback( DWORD dwEffect) = 0;
};
```

- WM_LBUTTONDOWN消息控制函数中调用OLE函数：DoDragDrop

拖-放数据传输协议：目标

- 实现“放目标”对象，“放目标”对象实现了接口IDropTarget：

```
class IDropTarget : public IUnknown
{
    virtual HRESULT DragEnter(IDataObject *pDataObj, DWORD grfKeyState,
                              POINTL pt, DWORD *pdwEffect) = 0;
    virtual HRESULT DragOver( DWORD grfKeyState, POINTL pt, DWORD *pdwEffect);
    virtual HRESULT DragLeave( void) = 0;
    virtual HRESULT Drop( IDataObject *pDataObj, DWORD grfKeyState,
                              POINTL pt, DWORD *pdwEffect) = 0;
};
```

- 为了支持“放”特性，调用OLE API函数RegisterDragDrop，把对象与窗口联系起来

```
WINOLEAPI  RegisterDragDrop(HWND hwnd, IDropTarget
    *pDropTarget);
```

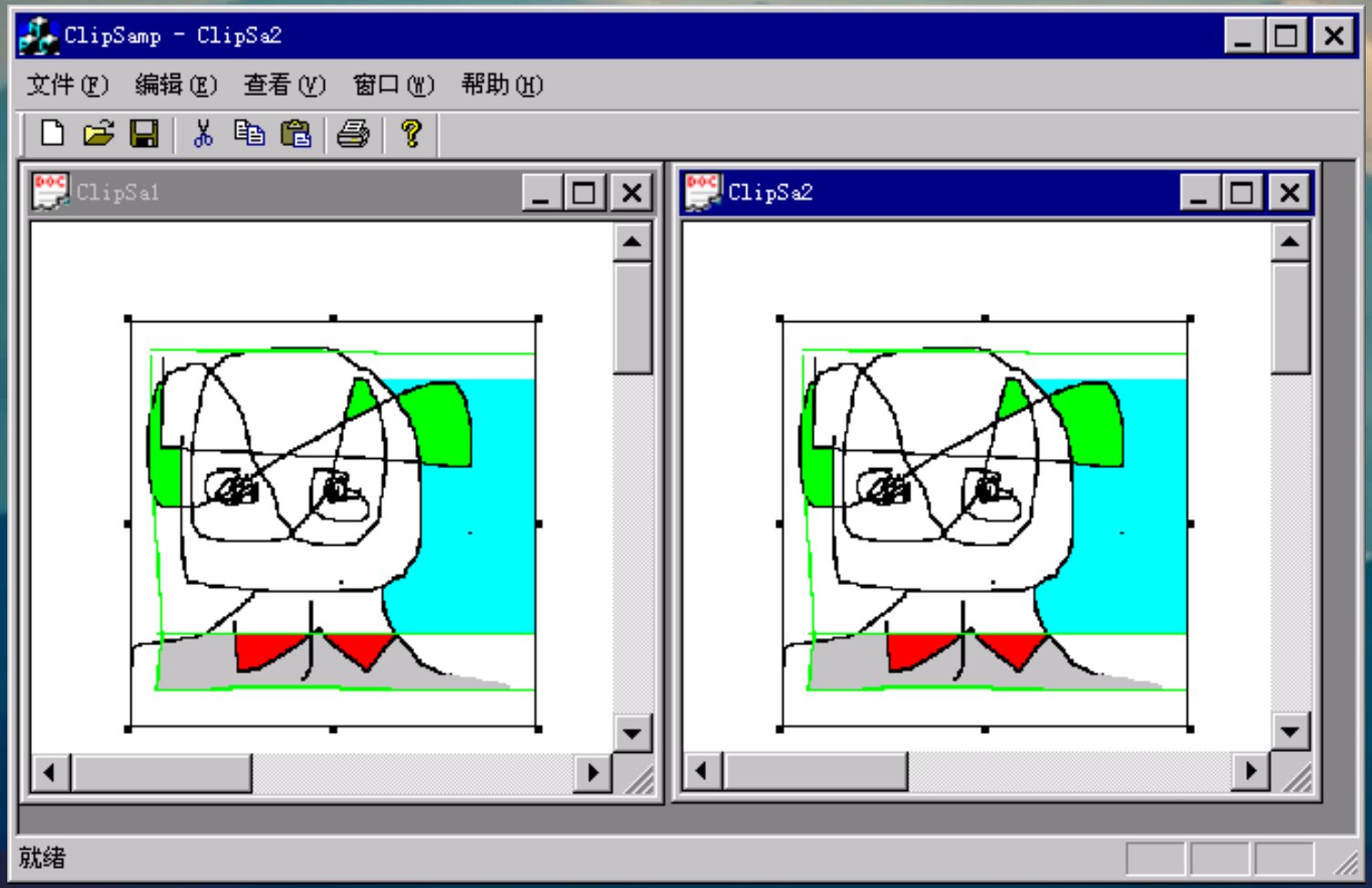
DoDragDrop函数

- 原型

```
WINOLEAPI  DoDragDrop(IDataObject *pDataObj,
                      IDropSource *pDropSource, DWORD dwOKEffects,
                      DWORD *pdwEffect);
```

- DoDragDrop函数内部是一个循环，它不断检查鼠标和键盘的状态，以及询问“放”目标窗口和“源”窗口，并作出反应。

支持剪贴板操作、拖-放操作的例子程序



ActiveX控制(ActiveX Control)

内容：

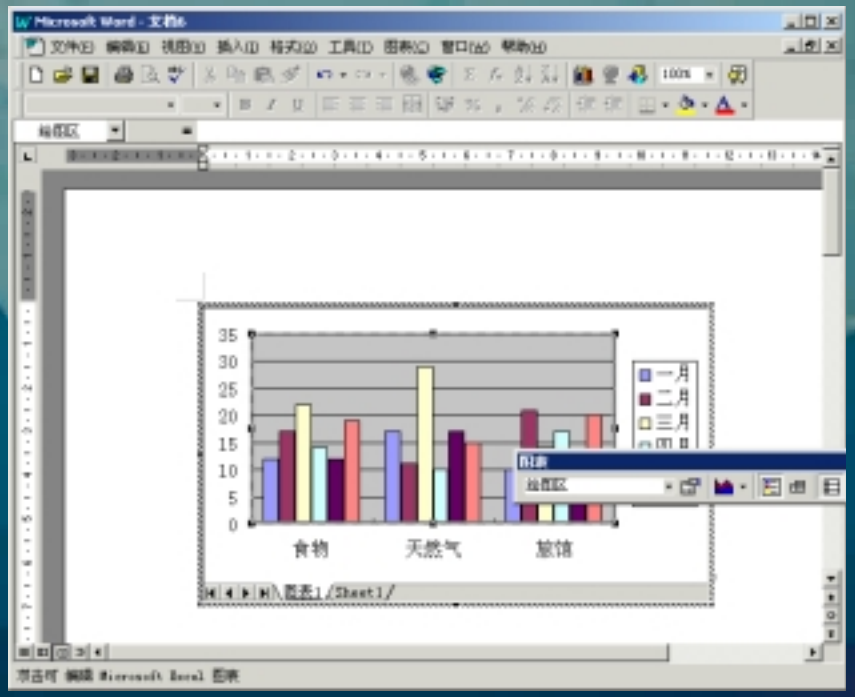
- ActiveX控制基础
- ActiveX控制结构
- ActiveX控制与Internet
- ActiveX控制开发

ActiveX控制基础

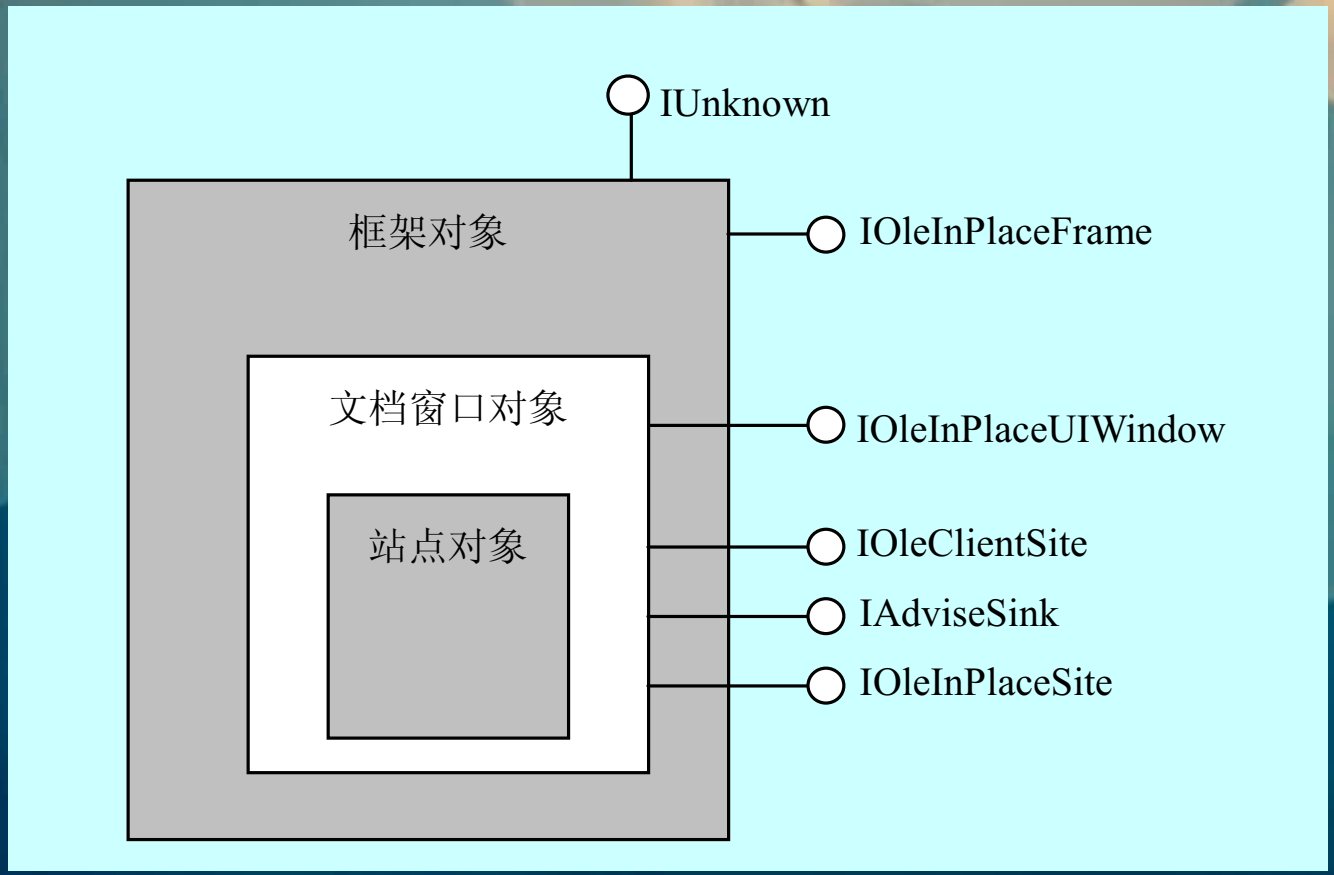
- OLE嵌入对象技术
- 实地激活(in-place activation)
- 属性页(Property Page)
- 属性变化通知

实地激活(in-place activation)

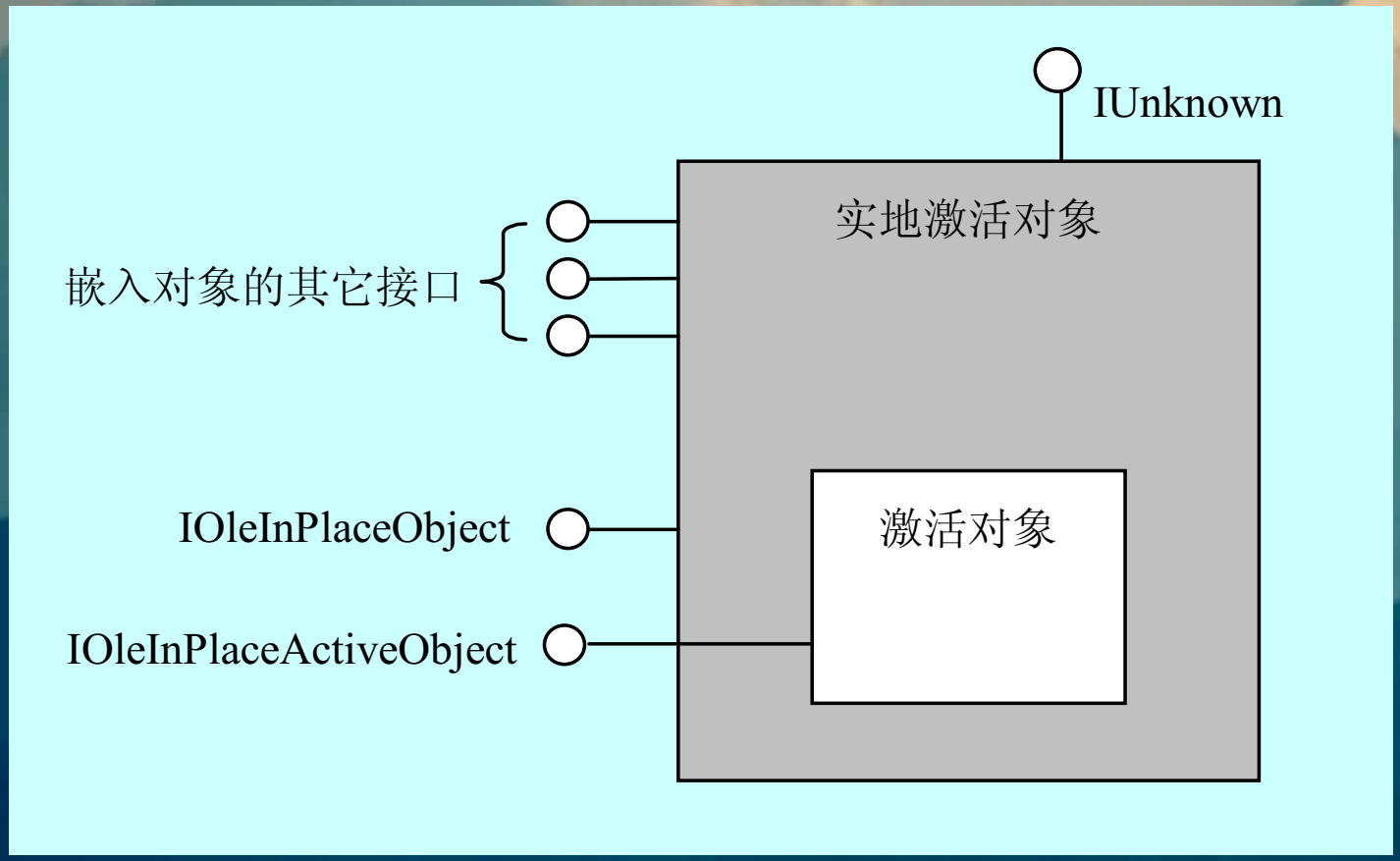
- 实地激活是指OLE对象的一种界面特性，具有实地激活特性的对象可以直接在包容器窗口内部进行编辑，这种特性有时也被称作实地编辑(in-place editing)或可视编辑(visual editing)



包容器程序结构


IUnknown
框架对象
IOleInPlaceFrame
文档窗口对象
IOleInPlaceUIWindow
站点对象
IOleClientSite
IAdviseSink
IOleInPlaceSite

支持实地激活特性的对象结构

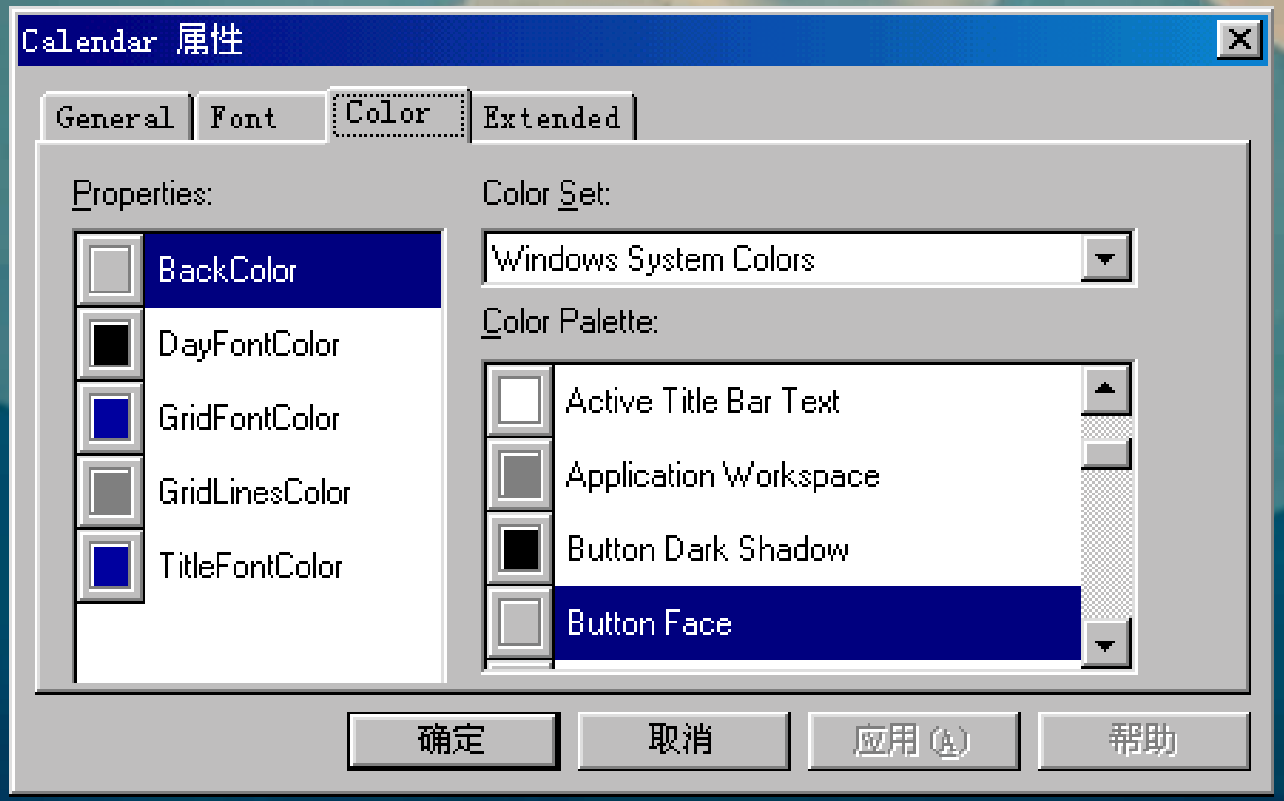


实地激活要点

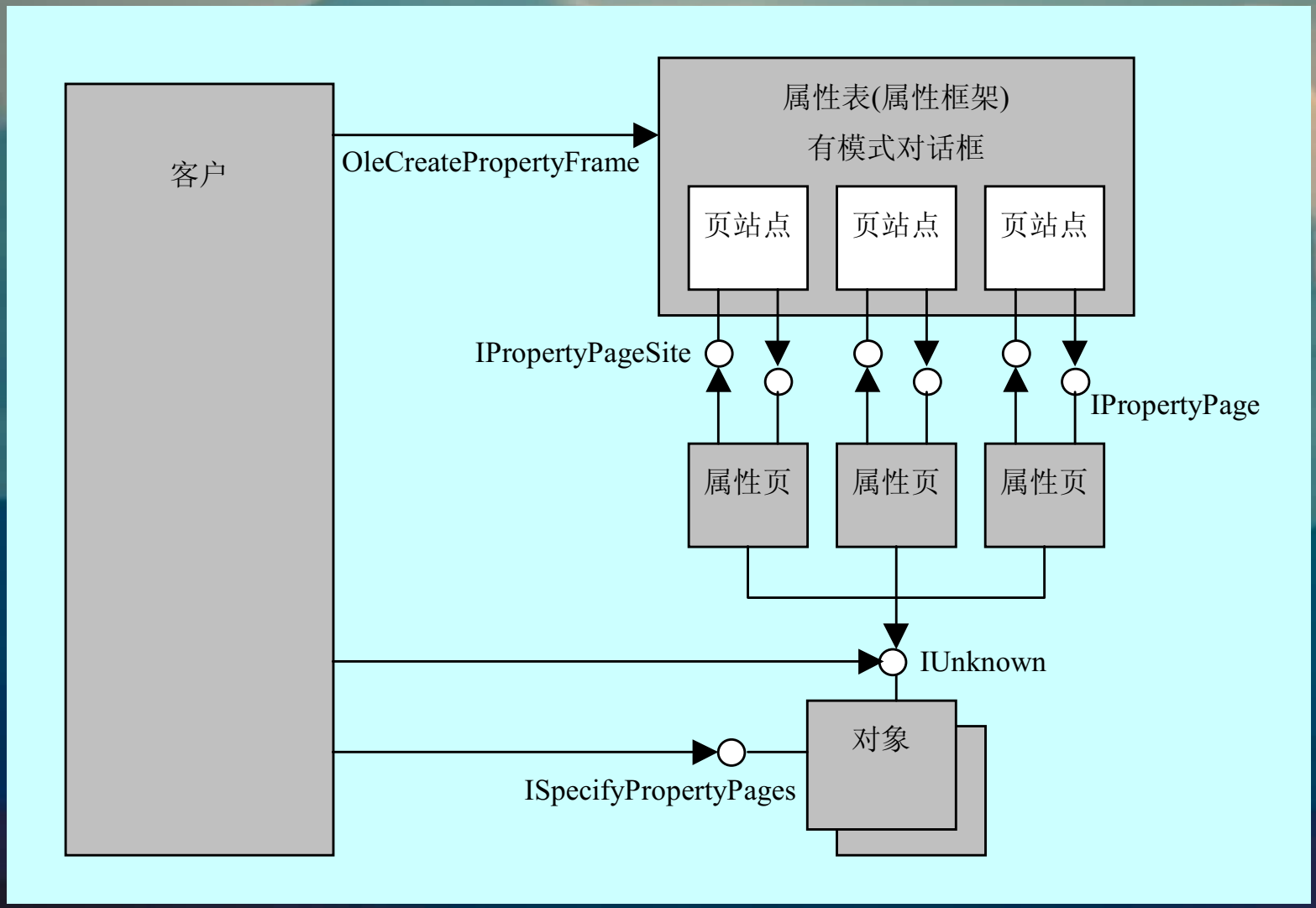
- 界面处理
  - 菜单合并、工具条、快捷键
- 消息发送、焦点控制
  - 鼠标消息、键盘消息
- 窗口大小和位置调整
- Undo支持
- Active Control可以有所简化

属性页(Property Page)

- 例子



客户、属性表、属性页和对象之间的结构关系



属性页技术
——COM对象

- COM对象如果要支持属性页特性，那么它必须实现ISpecifyPropertyPages接口

```
class ISpecifyPropertyPages : IUnknown
{
   HRESULT GetPages(CAUUID *pPages) = 0;
};
```

属性页技术
——客户程序

- OleCreatePropertyFrame

```
STDAPI OleCreatePropertyFrame(HWND hWndOwner, UINT x, UINT y,
        LPCOLESTR lpszCaption, ULONG cObjects, IUnknown **lplpUnk,
        ULONG cPages, CLSID *lpPages, LCID lcid,
        DWORD dwReserved, LPVOID pvReserved);
```

属性页技术
——属性页对象

```
class IPropertyPage : public IUnknown
{
    HRESULT SetPageSite(IPropertyPageSite *pPageSite) = 0;
    HRESULT Activate(HWND hWndParent, LPCRECT prc, BOOL fModal) = 0;
    HRESULT Deactivate(VOID) = 0;
    HRESULT GetPageInfo(PROPPAGEINFO *pPageInfo) = 0;
    HRESULT SetObjects(ULONG cObjects, IUnknown **ppUnk) = 0;
    HRESULT Show(UINT nCmdShow) = 0;
    HRESULT Move(LPCRECT prc) = 0;
    HRESULT IsPageDirty(VOID) = 0;
    HRESULT Apply(VOID) = 0;
    HRESULT Help(LPCOLESTR pszHelpDir) = 0;
    HRESULT TranslateAccelerator(LPMSG pMsg) = 0;
};
```

属性页技术
——属性页站点对象

```
class IPropertyPageSite : public IUnknown
{
    HRESULT OnStatusChange(DWORD flags) = 0;
    HRESULT GetLocaleID(LCID *pLocaleID) = 0;
    HRESULT GetPageContainer(IUnknown **ppUnk) = 0;
    HRESULT TranslateAccelerator(LPMSG pMsg) = 0;
};
```

属性变化通知

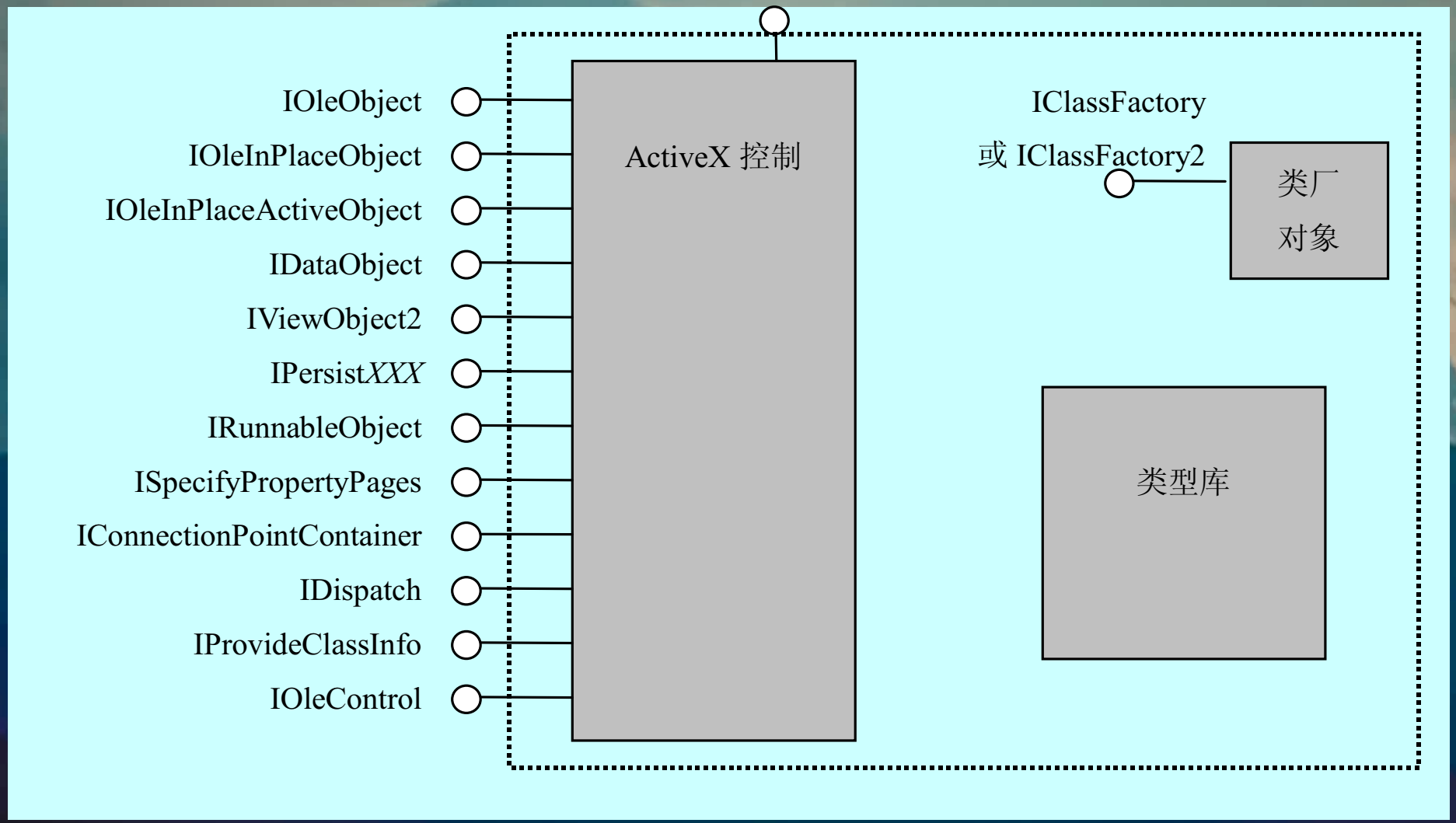
- 使用了可连接对象机制，出接口为

```
class IPropertyNotifySink : public IUnknown
{
   HRESULT OnChanged(DISPID dispid) = 0;
   HRESULT OnRequestEdit(DISPID dispid) = 0;
};
```

ActiveX控制相关技术列表

功能要求	使用的技术
属性和方法管理	自动化 属性变化通知(包括可连接对象机制)
事件管理	自动化 可连接对象(以 IDispatch 作为出接口)
用户界面特性(可视性)	实地激活、OLE 嵌入对象 可视对象(实现了接口 IViewObject2) 统一数据传输
状态永久性机制	结构化存储 永久对象

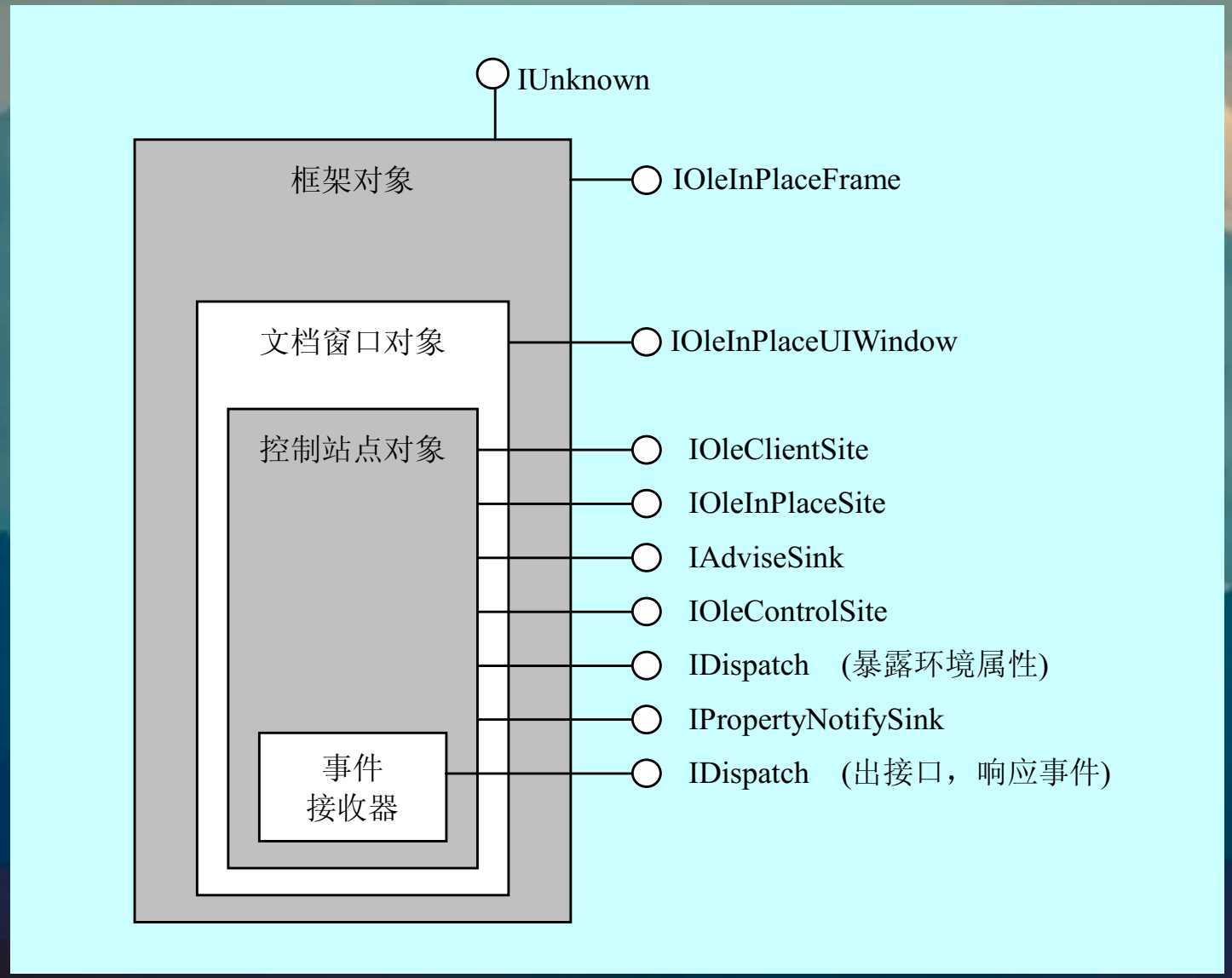
ActiveX控制结构



ActiveX控制包容器相关技术列表

功能要求	使用的技术
布局特性	OLE 复合文档 OLE 拖-放机制
永久特性	结构化存储 永久对象
包容器环境属性	自动化 控制站点对象
事件机制	自动化 可连接对象
包容器扩展控制	包容和聚合两种重用模型
键盘功能	IOleControl 和 IOleControlSite 接口

包容器基本结构


IUnknown
框架对象
IOleInPlaceFrame
文档窗口对象
IOleInPlaceUIWindow
控制站点对象
IOleClientSite
IOleInPlaceSite
IAdviseSink
IOleControlSite
IDispatch (暴露环境属性)
IPropertyNotifySink
事件
接收器
IDispatch (出接口，响应事件)

IOleObject接口

```
class IOleObject : public IUnknown
{
public:
    virtual HRESULT SetClientSite( IOleClientSite *pClientSite) = 0;
    virtual HRESULT GetClientSite( IOleClientSite **ppClientSite) = 0;
    virtual HRESULT SetHostNames( LPCOLESTR szContainerApp, LPCOLESTR szContainerObj) = 0;
    virtual HRESULT Close(DWORD dwSaveOption) = 0;
    virtual HRESULT SetMoniker( DWORD dwWhichMoniker,  IMoniker *pmk) = 0;
    virtual HRESULT GetMoniker( DWORD dwAssign, DWORD dwWhichMoniker, IMoniker **ppmk) = 0;
    virtual HRESULT InitFromData( IDataObject *pDataObject,
                                  BOOL fCreation, DWORD dwReserved) = 0;
    virtual HRESULT GetClipboardData( DWORD dwReserved, IDataObject **ppDataObject) = 0;
    virtual HRESULT DoVerb( LONG iVerb, LPMSG lpmsg, IOleClientSite *pActiveSite, LONG lindex,
                            HWND hwndParent, LPCRECT lprcPosRect) = 0;
    virtual HRESULT EnumVerbs(IEnumOLEVERB **ppEnumOleVerb) = 0;
    virtual HRESULT Update( void) = 0;
    virtual HRESULT IsUpToDate( void) = 0;
    virtual HRESULT GetUserClassID( CLSID *pClsid) = 0;
    virtual HRESULT GetUserType( DWORD dwFormOfType, LPOLESTR *pszUserType) = 0;
    virtual HRESULT SetExtent( DWORD dwDrawAspect, SIZEL *psizel) = 0;
    virtual HRESULT GetExtent( DWORD dwDrawAspect, SIZEL *psizel) = 0;
    virtual HRESULT Advise( IAdviseSink *pAdvSink, DWORD *pdwConnection) = 0;
    virtual HRESULT Unadvise( DWORD dwConnection) = 0;
    virtual HRESULT EnumAdvise( IEnumSTATDATA **ppenumAdvise) = 0;
    virtual HRESULT GetMiscStatus( DWORD dwAspect, DWORD *pdwStatus) = 0;
    virtual HRESULT SetColorScheme(LOGPALETTE *pLogpal) = 0;
};
```

IOleClientSite接口

```
class IOleClientSite : public IUnknown
{
public:
        virtual HRESULT SaveObject( void) = 0;
        virtual HRESULT GetMoniker( DWORD dwAssign,
                        DWORD dwWhichMoniker, IMoniker **ppmk) = 0;
        virtual HRESULT GetContainer(IOleContainer **ppContainer) = 0;
        virtual HRESULT ShowObject( void) = 0;
        virtual HRESULT OnShowWindow( BOOL fShow) = 0;
        virtual HRESULT RequestNewObjectLayout( void) = 0;
};
```

IOleControl接口

```
class IOleControl : public IUnknown
{
public:
        virtual HRESULT GetControlInfo( CONTROLINFO *pCI) = 0;
        virtual HRESULT OnMnemonic( MSG *pMsg) = 0;
        virtual HRESULT OnAmbientPropertyChange( DISPID dispID) = 0;
        virtual HRESULT FreezeEvents( BOOL bFreeze) = 0;
};
```

IOleControlSite接口

```
class IOleControlSite : public IUnknown
{
   public:
      virtual HRESULT OnControlInfoChanged( void) = 0;
      virtual HRESULT LockInPlaceActive( BOOL fLock) = 0;
      virtual HRESULT GetExtendedControl( IDispatch **ppDisp) = 0;
      virtual HRESULT TransformCoords( POINTL *pPtlHimetric,
                        POINTF *pPtfContainer, DWORD dwFlags) = 0;
      virtual HRESULT TranslateAccelerator( MSG *pMsg,
                        DWORD grfModifiers) = 0;
      virtual HRESULT OnFocus( BOOL fGotFocus) = 0;
      virtual HRESULT ShowPropertyFrame( void) = 0;
};
```

ActiveX控制功能特性

- 键盘功能、快捷键的处理、焦点控制
- 扩展控制功能
- 环境属性
- 属性页
- 事件控制

用于ActiveX控制的标准分发ID

- ActiveX控制的标准属性
- ActiveX控制的标准方法
- ActiveX控制的标准事件
- 包容器的标准环境属性

ActiveX控制的其它特点

- “缺省”和“取消”按钮
- 事件处理
- 永久特性
- 与ActiveX控制有关的注册表项

ActiveX控制与Internet

- 从桌面环境转向Internet
- 包装ActiveX控制
- 许可证管理
- Web页面中ActiveX控制的初始化
- 脚本支持与初始化安全性

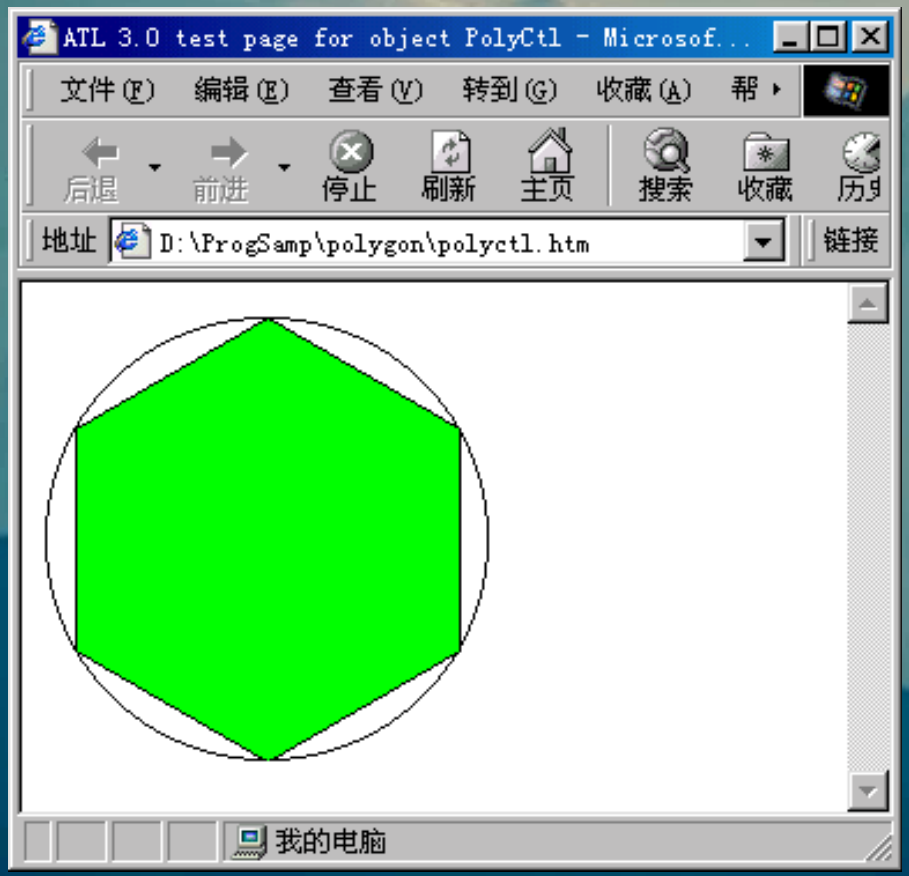
Internet环境带来的问题

- 网络传输
- 安全性
- 包容器如何管理、发布

接口选择

接口	功能与说明
IOleObject	如果控制要与包容器程序的站点对象进行通讯，那么必须实现此接口。
IOleInPlaceObject IOleInPlaceActiveObject	如果控制支持实地激活特性，那么必须实现此接口。
IOleControl	如果控制支持快捷键，或者访问包容器的环境属性，或者控制要求包容器处理事件，那么必须实现此接口。
IDataObject	如果控制提供数据对象的特性，那么必须实现此接口。
IViewObject2	如果控制在非实地激活的状态下也需要显示信息的话，那么必须实现此接口。
IDispatch	如果控制有自定义的属性和方法的话，那么必须实现此接口。
IConnectionPointContainer	如果控制对象支持一个或多个出接口，那么必须实现此接口。
IProvideClassInfo[2]	如果控制对象要直接通过 GetClassInfo 成员函数提供对象类型信息的话，那么必须实现此接口。
ISpecifyPropertyPages	如果控制对象支持属性页特性的话，那么必须实现此接口。
IPersistStream[Init] IPersistStorage 或其它永久接口	如果控制对象支持永久特性的话，那么至少实现一个永久接口。

示例


ATL 3.0 test page for object PolyCtl - Microsof...
文件(F) 编辑(E) 查看(V) 转到(G) 收藏(A) 帮
后退 前进 停止 刷新 主页 搜索 收藏
地址 D:\ProgSamp\polygon\polyctl.htm
链接
我的电脑

```
<HTML>
<HEAD>
<TITLE>ATL 3.0 test page for object PolyCtl</TITLE>
</HEAD>
<BODY>
<OBJECT ID="PolyCtl"
CLASSID="CLSID:2885EE05-A26B-11D1-B49B-00C04F98EFE0">
</OBJECT>
<SCRIPT LANGUAGE="VBScript">
<!--
Sub PolyCtl_ClickIn(x, y)
   PolyCtl.Sides = PolyCtl.Sides + 1
End Sub
Sub PolyCtl_ClickOut(x, y)
   PolyCtl.Sides = PolyCtl.Sides - 1
End Sub
-->
</SCRIPT>
</BODY>
</HTML>
```

HTML代码描述

ActiveX控制的发布

- 客户-服务器协调控制
- 例子一：

```
<OBJECT
    CLASSID="CLSID:2885EE05-A26B-11D1-B49B-00C04F98EFE0"
    codebase="http://webserver/Polygon.dll"
    ALIGN="CENTER" WIDTH=200 HEIGHT=200
      ID="PolyCtl">
</OBJECT>
```

- IE自动下载到“Downloaded Program Files”目录中并注册到客户机上

ActiveX控制的包装

- 例子二：

```
<OBJECT
    CLASSID="CLSID:2885EE05-A26B-11D1-B49B-00C04F98EFE0"
    codebase="http://webserver/Polygon.cab"
    ALIGN="CENTER" WIDTH=200 HEIGHT=200
      ID="PolyCtl">
</OBJECT>
```

- CAB文件，压缩代码提高传输效率
- CAB文件可包含多个代码文件

IE对CAB文件的处理过程

- IE在解析“OBJECT”标记时，它继续查找codebase属性
- 如果codebase指定了ActiveX控制的CAB文件，那么IE定位到CAB文件。
- IE把CAB文件中的有关文件解压出来，并放到“Downloaded Program Files”子目录中。
- IE注册有关的文件。
- IE调用COM API函数创建ActiveX控制对象。

CAB文件

- 包含了ActiveX控制注册和运行所需要的必要信息
- CAB文件包含一个INF文件，
- INF文件是一个文本文件，它描述了CAB文件的所有细节信息
- CAB文件的制作
  - cabarc.exe N polygon.cab atl.dll polygon.dll \ polygon.inf
  - 支持数字签名

Polygon控制的INF文件

```
[version]
    signature="$CHICAGO$"
    AdvancedINF=2.0
[Add.Code]
    polygon.dll=polygon.dll
    atl.dll=atl.dll
[atl.dll]
    file-win32-x86=thiscab
    FileVersion=3,00,0,8166
    DestDir=11
    RegisterServer=yes
[polygon.dll]
    file-win32-x86=thiscab
    clsid={2885EE05-A26B-11D1-B49B-00C04F98EFE0}
    FileVersion=1,0,0,1
    RegisterServer=yes
```

许可证管理

```
typedef struct  tagLICINFO
{
    LONG cbLicInfo;
    BOOL fRuntimeKeyAvail;
    BOOL fLicVerified;
}    LICINFO;

class IClassFactory2 : public IClassFactory
{
public:
      virtual HRESULT GetLicInfo( LICINFO *pLicInfo) = 0;
      virtual HRESULT RequestLicKey( DWORD dwReserved, BSTR *pBstrKey) = 0;
      virtual HRESULT CreateInstanceLic( IUnknown *pUnkOuter,
                                         IUnknown *pUnkReserved, REFIID riid,
                                         BSTR bstrKey, (void **)ppvObj) = 0;
};
```

许可证检查

- 设计时刻的许可证检查由ActiveX控制的包容器程序完成
- 在运行时刻，不同的包容器程序对ActiveX控制的许可证检查方法有所不同。以VB为例：
  - （1）创建应用程序时刻(build)
  - （2）在运行应用程序时刻
- IE的处理有所不同

IE的许可证管理

- IE包含一个许可证管理器组件
- Microsoft引进了许可证包文件(license package file，后缀为LPK)
- IE的许可证管理器组件负责解析LPK文件，并提取出每个CLSID的许可证
- 然后调用IClassFactory2::CreateInstanceLic函数创建ActiveX控制对象

Web页面ActiveX控制的初始化(一)

- 例子

```
<OBJECT
    CLASSID="CLSID:532EB3E0-327A-1203-B7A5-0000C2C55F ED"
    CODEBASE="http://webserver/MyCtrl.cab"
    DATA="http://webserver/MyData.dat"
    ID="MyCtl">
</OBJECT>
```

- IE初始化过程：
  - （1）IE创建URL名字对象
  - （2）然后调用ActiveX控制的IPersistMoniker接口的Load成员函数执行初始化
  - （3）ActiveX控制调用名字对象的IMoniker::BindToStorage函数获取属性数据

Web页面ActiveX控制的初始化(二)

- 例子

```
<OBJECT ID="PolyCtl"
    ALIGN="CENTER" WIDTH=270 HEIGHT=300
    CLASSID="CLSID:2885EE05-A26B-11D1-B49B-00C04F98EFE0"
    codebase="http://webserver/Polygon.dll" >
    <PARAM NAME="Sides" VALUE=5 >
 </OBJECT>
```

- IE初始化过程：
  - （1）IE把“PARAM”属性对生成一个属性包(property bag)对象
  - （2）然后调用ActiveX控制的IPersistPropertyBag接口的Load成员函数执行初始化

脚本支持与初始化安全性

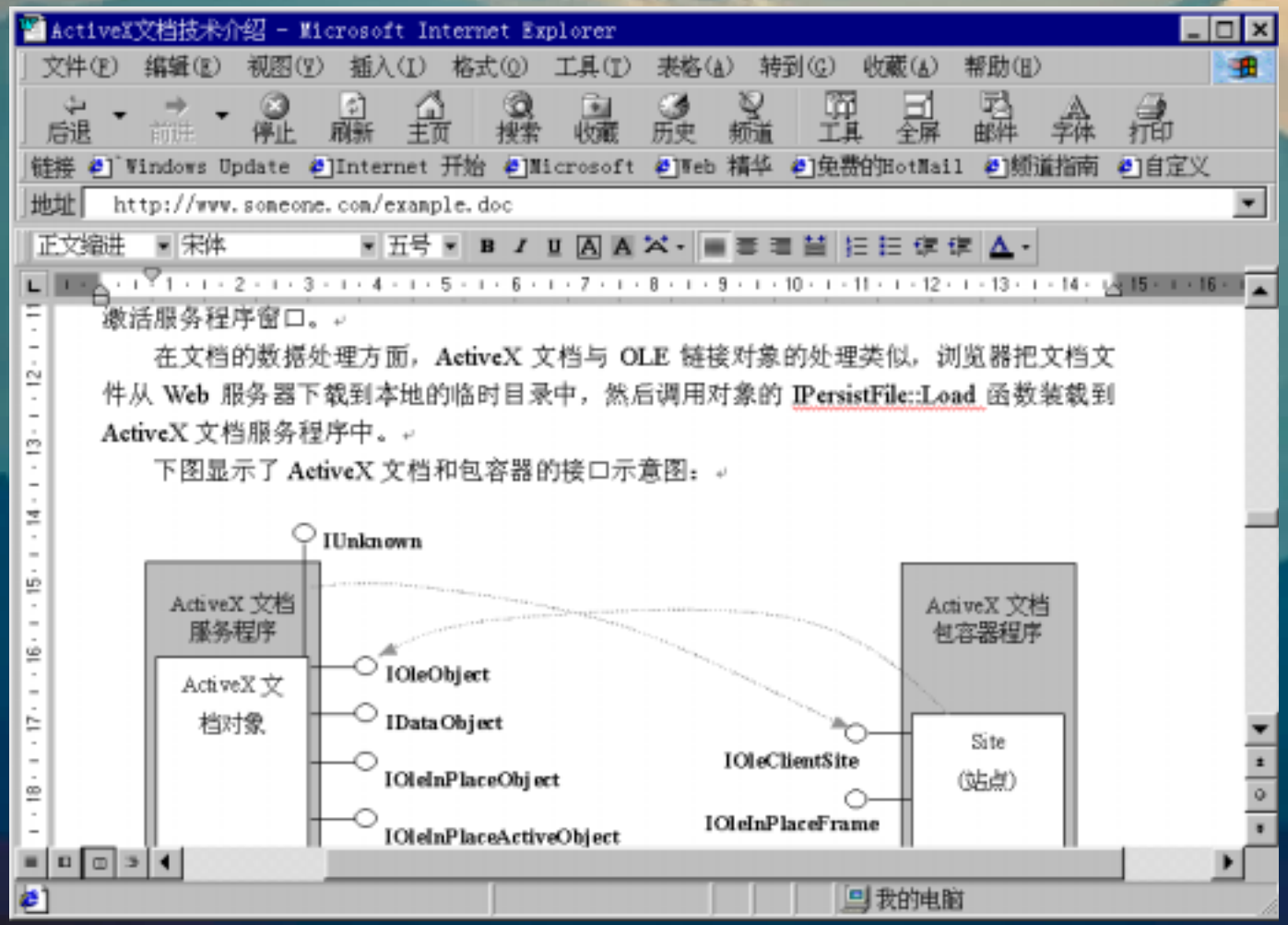
```
class IObjectSafety : public IUnknown
{
public:
        virtual HRESULT GetInterfaceSafetyOptions( REFIID riid,
                                        DWORD *pdwSupportedOptions,
                                        DWORD *pdwEnabledOptions) = 0;
        virtual HRESULT SetInterfaceSafetyOptions( REFIID riid,
                                        DWORD dwOptionSetMask,
                                        DWORD dwEnabledOptions) = 0;
};
```

- 安全性包括初始化安全性和脚本安全性
- 也可以通过注册表项的“组件类别”设置安全性

ActiveX控制开发和应用

- VC集成环境的支持：
  - AppWizard和ClassWizard、
  - ActiveX Control Test Containner
- MFC——COleControl类
  - 用MFC实现ActiveX控制
  - 用MFC实现ActiveX控制包容器
- ATL
- VB

ActiveX Control和Active Document
——Active Document示例



ActiveX Control和Active Document区别

- 程序类型不同
- 界面方式不同
- HTML文件中的使用方式不同
- 数据保存方式不同
- 服务程序转载方式不同