中间件技术
Middleware technology

第五章 组件技术概述

赖永炫 博士/教授
厦门大学 软件工程系

COM组件技术介绍

一、组件的概念

二、COM组件的定义和特点

三、COM组件技术简介

四、.Net组件技术

组件的概念

组件是近代工业发展的产物，通过接口的标准化，使得功能模块化。

- 同样的组件可应用于多类产品和多个领域，扩展了市场范围；同时，各功能组件一般由更专业的厂商生产，提高了质量，降低了成本。

软件行业借鉴了工业界的这个想法，把数据和方法的封装对象称为为组件。

卡耐基梅隆大学把组件定义为"一个不透明的功能实体，能够被第三方组装，且符合一个构件模型"。

组件的定义

对象管理小组（Object Management Group, OMG）的"建模语言规范"中将组件定义为：

"系统中一种物理的、可代替的部件、它封装了实现并提供了一系列可用的接口。一个组件代表一个系统中实现的物理部分，包括软件代码（源代码，二进制代码，可执行代码）或者一些类似内容，如脚本或者命令文件。"

组件的叫法

C++ Builder中叫组件，Delphi中叫部件，在Visual BASIC中叫控件，而在JAVA EE中则成为Java Bean。

- 比如，在C++ Builder中一个组件就是一个从TComponent派生出来的特定对象。组件可以有自己的属性和方法。属性是组件数据的简单访问者，方法则是组件的一些可见的功能。

组件技术

组件技术是一种优秀的软件重用技术。采用组件开发软件就像搭积木一样容易，组件是具有某种特定功能的软件模型，它几乎可以完成任何任务。

组件技术的基本思想是将复杂的大型系统中的基础服务功能分解为若干个独立的单元，即软件组件。

利用组件之间建立的统一的严格的连接标准，实现组件间和组件与用户之间的服务连接。

组件技术

连接是建立在目标代码级上的，与平台无关。

- 只要遵循组件技术的规范，任何人可以用自己方便的语言去实现可复用的软件组件，
- 应用程序或其它组件的开发人员也可以按照其标准使用组件提供的服务，而且客户和服务组件任何一方版本的独立更新都不会导致兼容性的问题。
- 这 犹如在独立的应用程序间建立了相互操作的协议，从而在更大程度上实现了代码重用和系统集成，降低了系统的复杂程度。

组件技术

组件技术将面向对象特性（例如封装和继承）与（逻辑或物理的）分布结合起来。

事实上，组件技术不是一个明确的范畴，在一定程度 上，它是进行操作的一个场所。

组件技术使面向对象技术进入到成熟的实用化阶段。

组件技术

在组件技术的概念模式下，软件系统可以被视为相互协同工作的对象集合，其中每个对象都会提供特定的服务，发出特定的消息，并且以标准形式公布出来，以便其他对象了解和调用。

组件间的接口通过一种与平台无关的语言IDL(Interface Define Language)来定义， 而且是二进制兼容的，使用者可以直接调用执行模块来获得对象提供的服务。

组件技术的好处

大大改变了软件开发的方式

可把软件开发的内容分成若干个层次，将每个层次封装成一个个的组件

在构建应用系统时，将这些个组件有机地组装起来就成为一个系统，就象是用零件组装出一台机器一样：

- 组件可替换，以便随时进行系统升级和定制；
- 可以在多个应用系统中重复利用同一个组件；
- 可以方便的将应用系统扩展到网络环境下；
- 部分组件与平台和语言无关，所有的程序员均可参与编写。

主要组件技术

1. Microsoft COM/DCOM
2. OMG CORBA
3. SUN JavaBeans/ Java EE
4. Microsoft .NET
5. Service Orient Architecture / SOA

静态链接库vs. 动态链接库

静态链接库：能做到代码共享，但是只能是在编译链接阶段

在运行时，程序调用的是已经链接到自己程序内的库函数

如果每个程序都包含所有用到的公共库函数，则这是很大的浪费，即增加了链接器的负担，也增大了可执行程序的大小，还加大了内存的消耗。

因此，出现了 DLL(Dynamic Link Libraries)

DLL还不能算组件技术，但它是软件重用的鼻祖

静态链接库vs. 动态链接库

采用动态链接，对公用的库函数，系统只有一个拷贝（位于系统目录的*.DLL文件），而且只有在应用程序真正调用时，才加载到内存。在内存中的库函数，也只有一个拷贝，可供所有运行的程序调用。当再也没有程序需要调用它时，系统会自动将其卸载，并释放其所占用的内存空间。

应用程序是通过系统来调用动态链接库的，因此每个DLL都有一个类似于main的入口函数。

COM组件对象模型

Component Object Model

Component Object Model (*COM*) is a binary-interface standard for software componentry introduced by Microsoft in 1993. It is used to enable interprocess communication and dynamic object creation in a large range of programming languages. The term *COM* is often used in the software development industry as an umbrella term that encompasses the OLE, OLE Automation, ActiveX, COM+ and DCOM technologies.

http://en.wikipedia.org/wiki/Component_Object_Model

COM组件对象模型

微软推出的COM组件对象模型（The Component Object Model）是一种基于COM的、已经编译好的软件组件，可为应用程序、操作系统以及其他组件提供服务。

COM组件对象模型开发自定义的COM组件就如同开发动态的，面向对象的API。

多个COM对象可以连接起来形成应用程序或组件系统，并且组件可以在运行时刻，在不被重新链接或编译应用程序的情况下被卸下或替换掉。

COM组件定义

1、COM/DCOM组件技术是由Microsoft公司与DEC公司于1995年提出的，COM代表Component Object Model（组件对象模型），DCOM代表Distributed Component Object Model。

2、COM不是一种语言，而是一种标准、规范，包括一套标准API、一个标准的接口集以及COM用于支持分布式计算的网络协议。

COM组件定义

3、组件对象模型COM是Microsoft公司的一种技术基石，在Microsoft公司的MSDN中是这样定义的：“COM是软件组件互相通信的一种方式，它是一种二进制和网络标准，允许任意两个组件互相通信，而不管它们在什么计算机上运行（只要计算机是相连的），不管计算机运行的什么操作系统（只要该操作系统支持COM），也不管该组件机是用什么语言编写的。”

COM组件定义

4、COM组件是遵循COM规范编写、以Win32动态链接库(DLL)或可执行文件(EXE)形式发布的可执行二进制代码，能够满足对组件架构的所有需求。遵循COM的规范标准，组件与应用、组件与组件之间可以互操作，极其方便地建立可伸缩的应用系统。

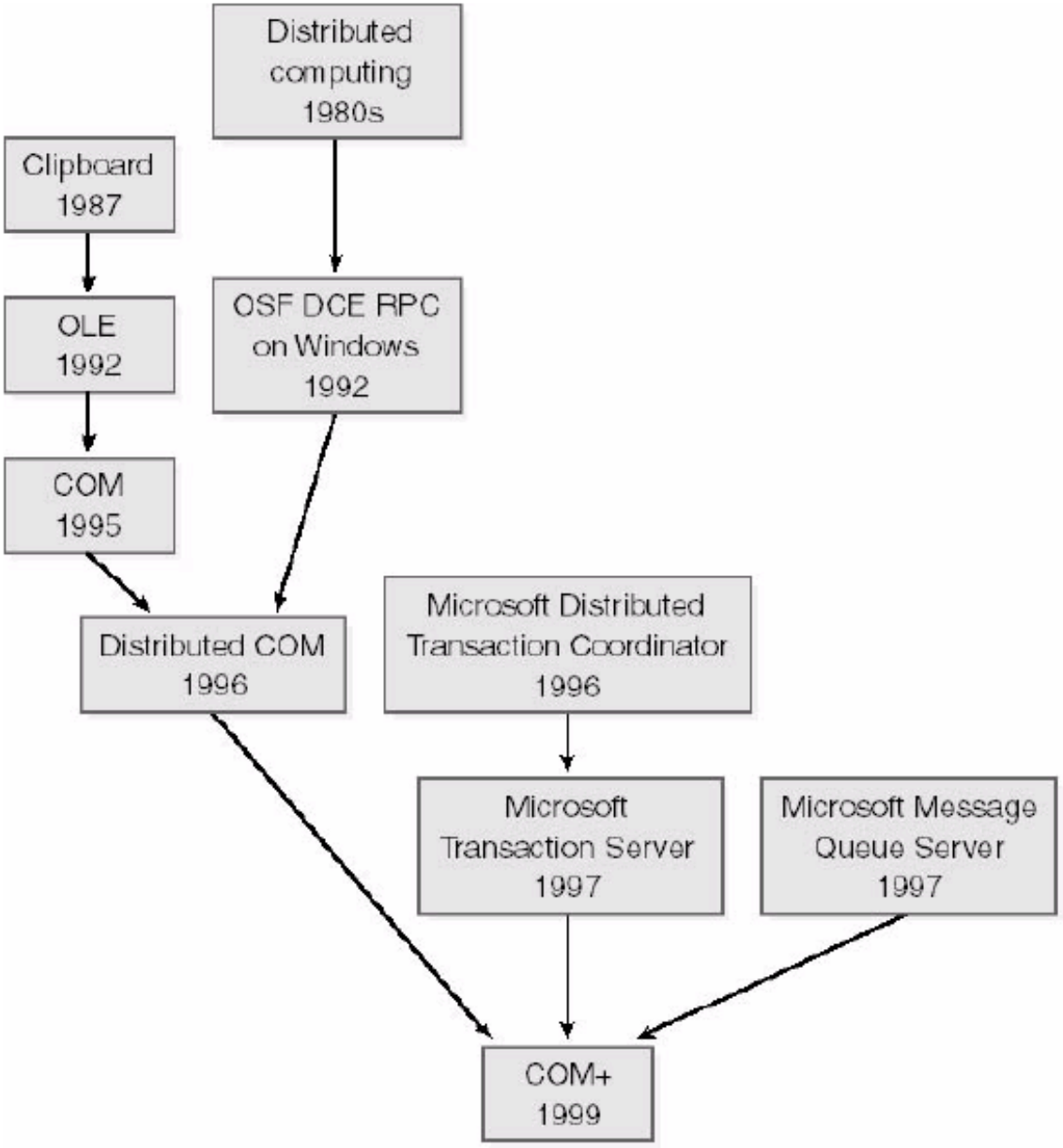
5、COM是一种技术标准,其商业品牌则称为ActiveX。ActiveX是Microsoft遵循COM/DCOM规范而开发的用于Internet的一种对象连接与嵌入技术（OLE）。ActiveX是从复合文档技术OLE成长以来的。OLE最初发布的版本只是瞄准复合文档，但在OLE2中引入了COM。一般常用的COM组件有两类:ActiveX DLL和ActiveX 控件 。

COM /DCOM

COM/DCOM（Component Object Model，构件对象模型/Distrubuted Component Object Model） 是Microsoft提出的一个(分布的)二进制兼容构件的规范。

只要遵守这种规范，不管用什么编程语言和工具开发的COM构件，也不管是否运行在同一台机器上，还是运行在不同的机器上，都可以被使用。

COM的演化


Distributed computing 1980s
Clipboard 1987
OLE 1992
OSF DCE RPC on Windows 1992
COM 1995
Distributed COM 1996
Microsoft Distributed Transaction Coordinator 1996
Microsoft Transaction Server 1997
Microsoft Message Queue Server 1997
COM+ 1999

RPC and COM+

The OSF members decided to address the issue of distributed computing. Out of this effort grew the specifications for the Distributed Computing Environment (DCE), an environment for creating distributed systems

One outcome of the DCE effort was a specification for communicating between computers. This specification, known as Remote Procedure Calls (RPCs), allows applications on different computers to communicate

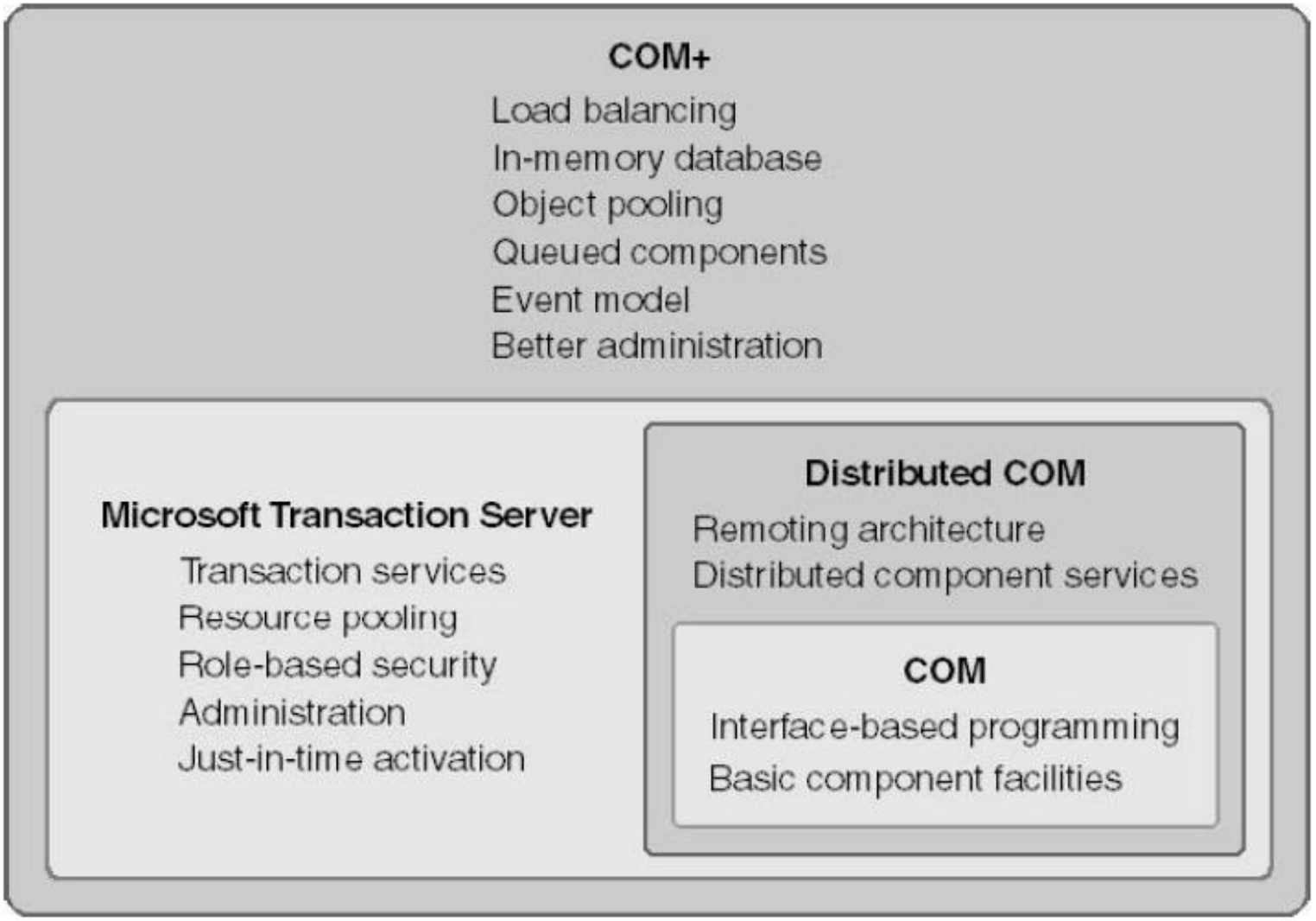
COM+ uses RPCs for its intercomputer communication.

From COM to COM+

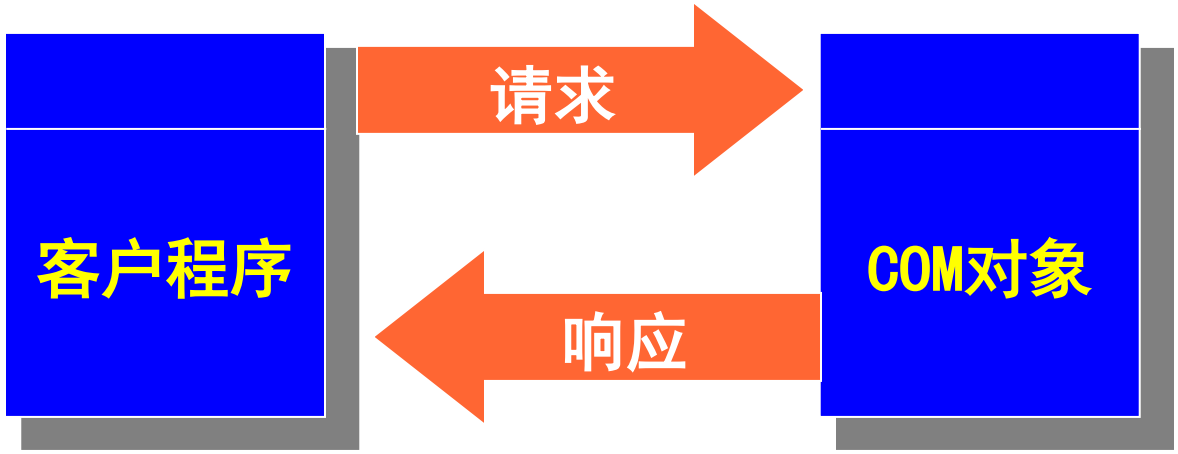
- COM+ currently encompasses two areas of functionality: a fundamental programming architecture for building software components (which was first defined by the original COM specification), plus an integrated suite of component services with an associated run-time environment for building sophisticated software components.

- Components that execute within this environment are called configured components. Components that do not take advantage of this environment are called unconfigured components; they play by the standard rules of COM

Component Services



COM是基于客户/服务器模型的，每个COM对象的工作方式类似于服务器，可接收和处理来自客户程序的请求，并产生自动回复信息。



COM对象和客户程序

COM构件模型

建立在二进制层次上的标准

---编程语言和开发工具无关

COM规范

---平台无关

---定义了大量的标准接口（如IUnknown、IClassFactory、IDispatch等等）用于各种用途

COM实现

---平台相关

---Windows实现了规范及许多辅助功能

COM功能

*基本功能

-- IUnknown

-- IDispatch

*扩展功能

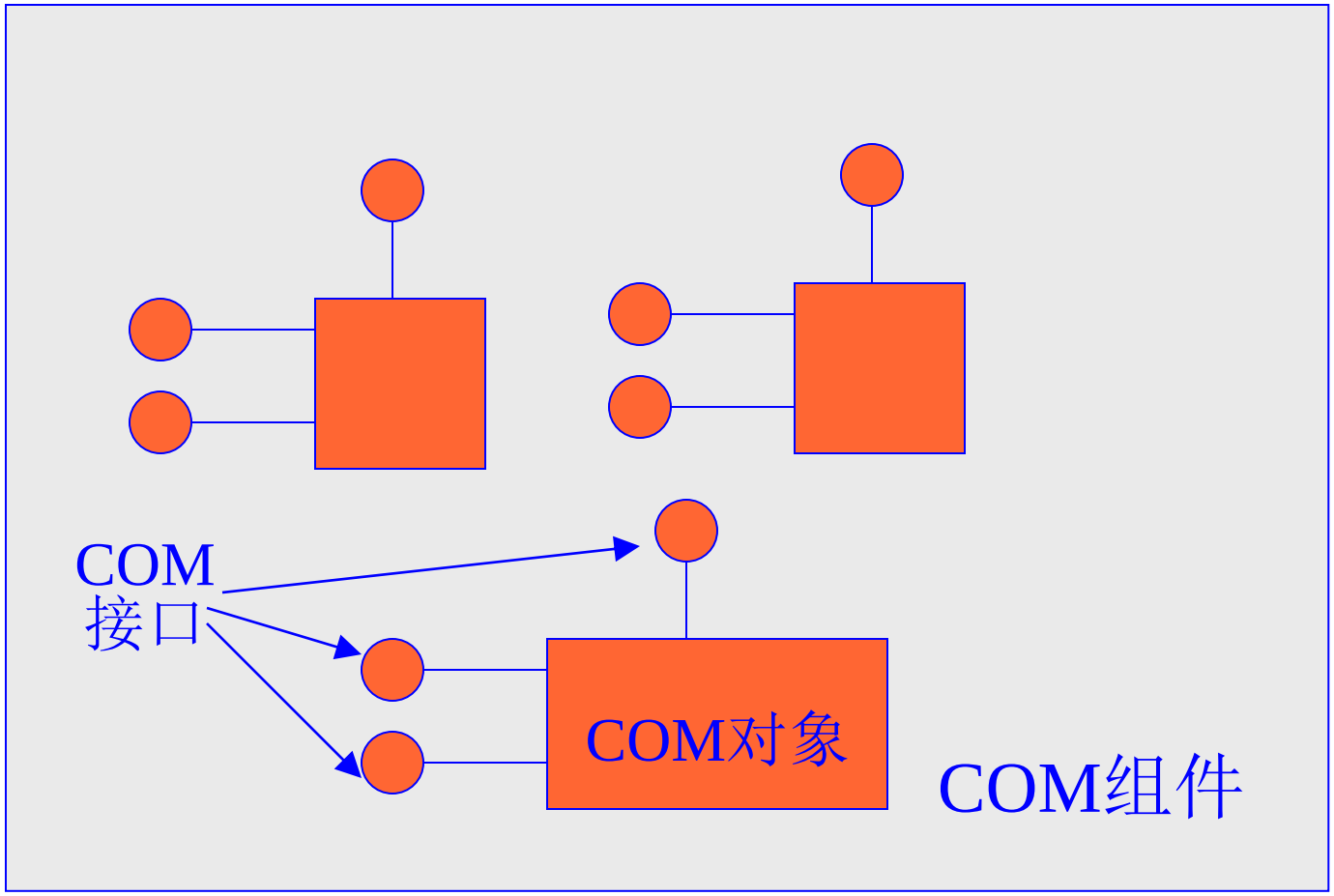
--自动化

--连接点

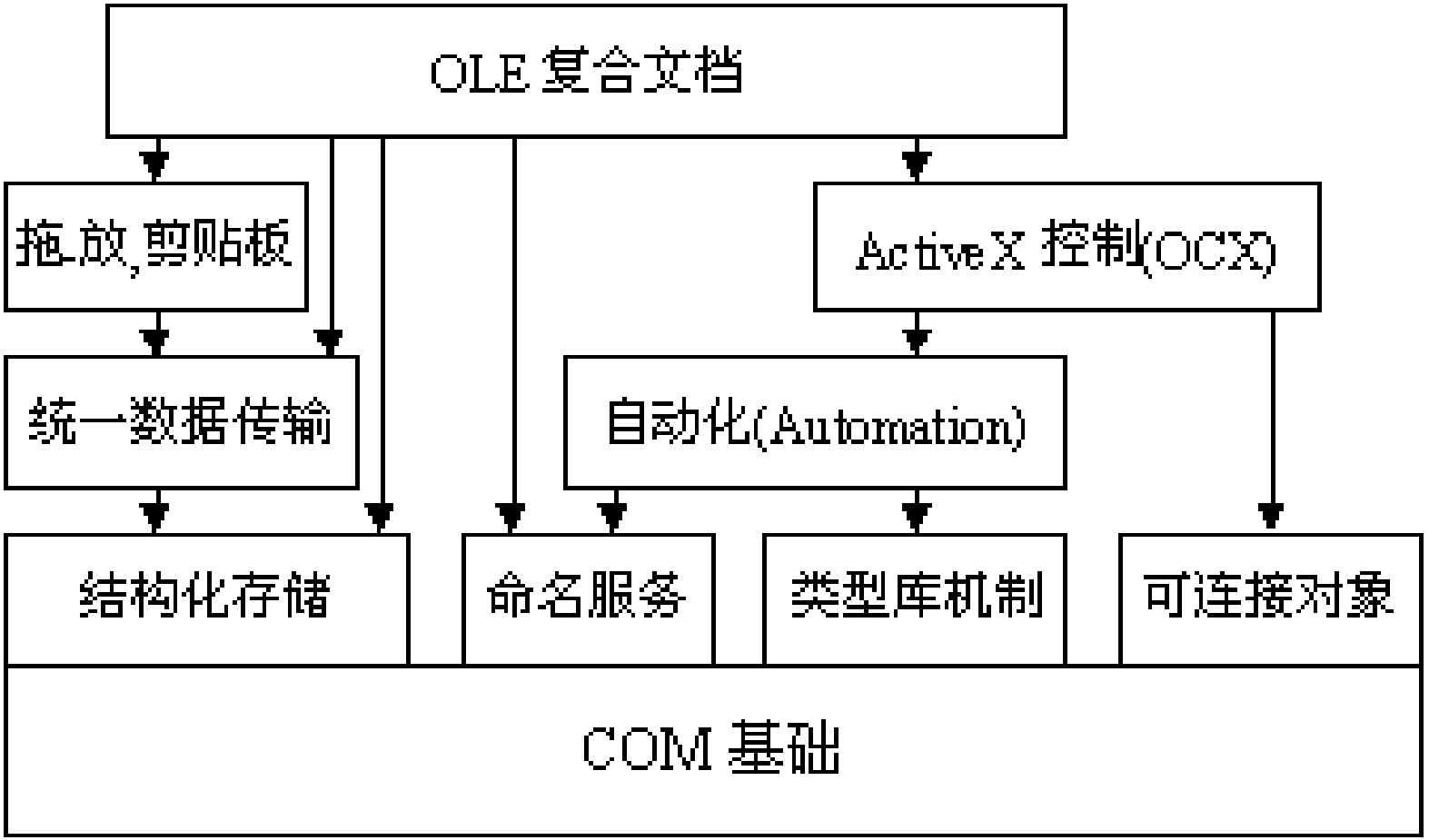
--结构化存储

--名字服务

COM基础结构



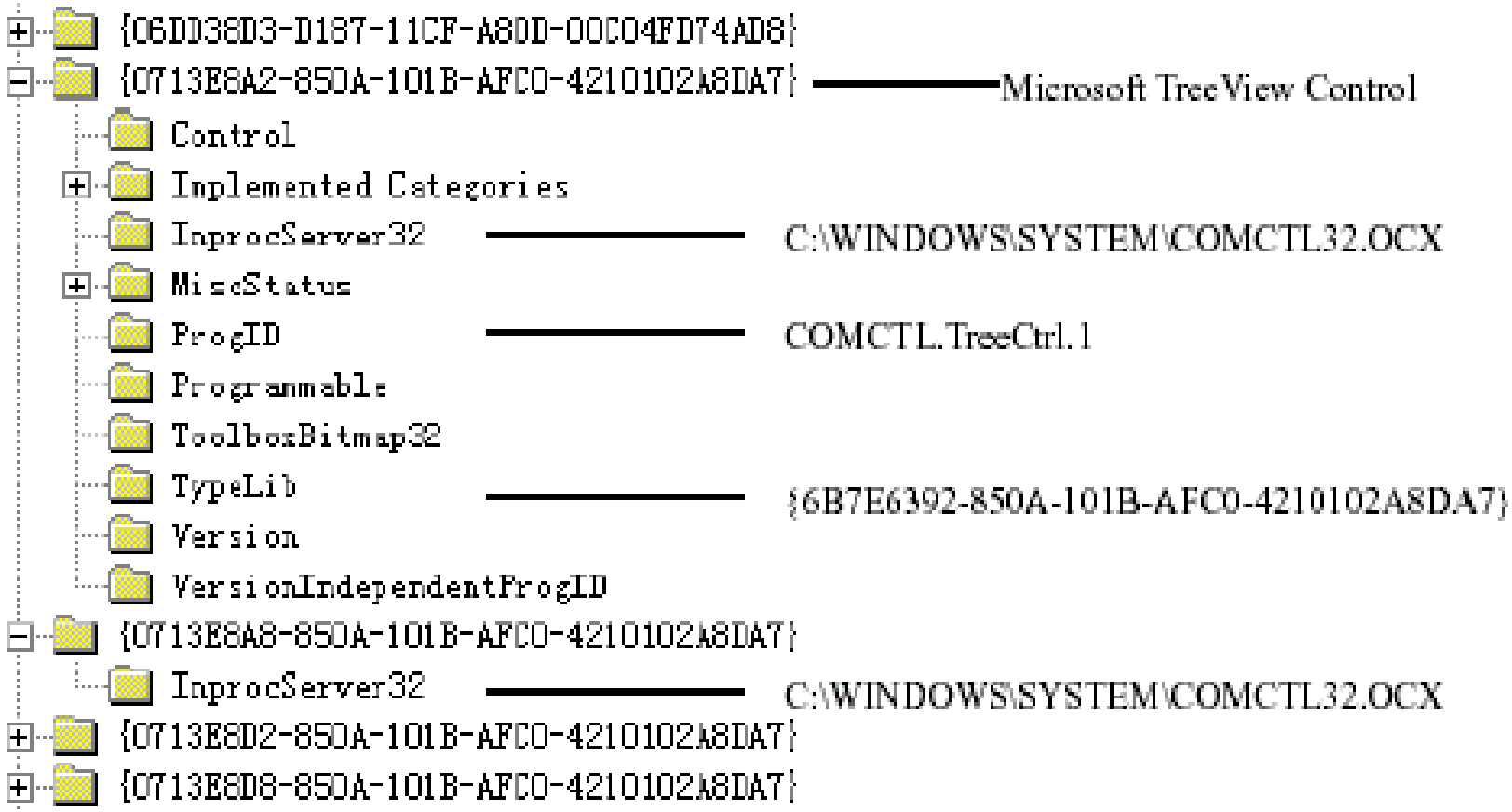
COM体系结构


OLE 复合文档
拖放,剪贴板
ActiveX 控制(OCX)
统一数据传输
自动化(Automation)
结构化存储
命名服务
类型库机制
可连接对象
COM 基础

COM组件发布

注册表

--发布组件信息、对象信息、接口信息等



COM应用

*OLE　Object Linking and Embedding，

对象连接与嵌入

*ActiveX Control

使用 ActiveX, 可轻松方便的在 Web页中插入 多媒体效果、交互式对象、以及复杂程序

*OLE DB

微软的战略性的通向不同的数据源的低级应用程序接口

*ASP

……

在COM技术中，组件和接口是其核心概念

几个基本概念

COM接口

- 客户与对象之间的协议，客户使用COM接口调用COM对象的服务

COM对象

- 实现COM接口
- 通过COM接口提供服务
- 可包含多个接口

COM构件（组件）

- COM对象的载体，可包含多个COM对象
- 可独立发布的二进制组件
- 在Windows平台上为DLL或者EXE

COM接口

- 接口标识——IID
- IUnknown
- COM接口二进制结构

COM接口的标识——IID

- 是GUID(globally unique identifier) 的一种用法，GUID是一个128位的长整数，具有全球唯一性

  例子：{54BF6567-1007-11D1-B0AA-444553540000}

- C语言结构和定义：

```
typedef struct _GUID { DWORD Data1; WORD Data2;
WORD Data3;  BYTE Data4[8]; } GUID;

extern "C" const GUID IID_IString =
{ 0x54bf6567, 0x1007, 0x11d1,
{ 0xb0, 0xaa, 0x44, 0x45, 0x53,0x54, 0x00, 0x00}
 } ;
```

IUnknown接口

- 所有的COM接口都从IUnknown派生
- C++定义：

```
class IUnknown
{
  public:
   virtual HRESULT__stdcall QueryInterface(
  const IID& iid, void **ppv) = 0 ;
  virtual ULONG __stdcall AddRef() = 0;
  virtual ULONG __stdcall Release() = 0;
};
```

IUnknown接口

C定义：

```
typedef struct IUnknownVtbl{
    HRESULT ( STDMETHODCALLTYPE __RPC_FAR *QueryInterface )(
                    IUnknown __RPC_FAR * This, /* [in] */ REFIID riid,
                   /* [iid_is][out] */ void __RPC_FAR *__RPC_FAR *ppvObject);
   ULONG ( STDMETHODCALLTYPE __RPC_FAR *AddRef )(
                    IUnknown    __RPC_FAR * This);
   ULONG ( STDMETHODCALLTYPE __RPC_FAR *Release )(
                   IUnknown __RPC_FAR * This);
} IUnknownVtbl;

interface IUnknown{
     CONST_VTBL struct IUnknownVtbl __RPC_FAR *lpVtbl;
};
```

基本接口——IUnknown

每一个COM接口都派生于IUnknown。

- 这个名字有点误导人，其中没有未知（Unknown）接口的意思。
- 它的原意是如果有一个指向某COM对象的IUnknown指针，就不用知道潜在的对象是什么，因为每个COM对象都实现IUnknown。

IUnknown 有三个方法：

AddRef()

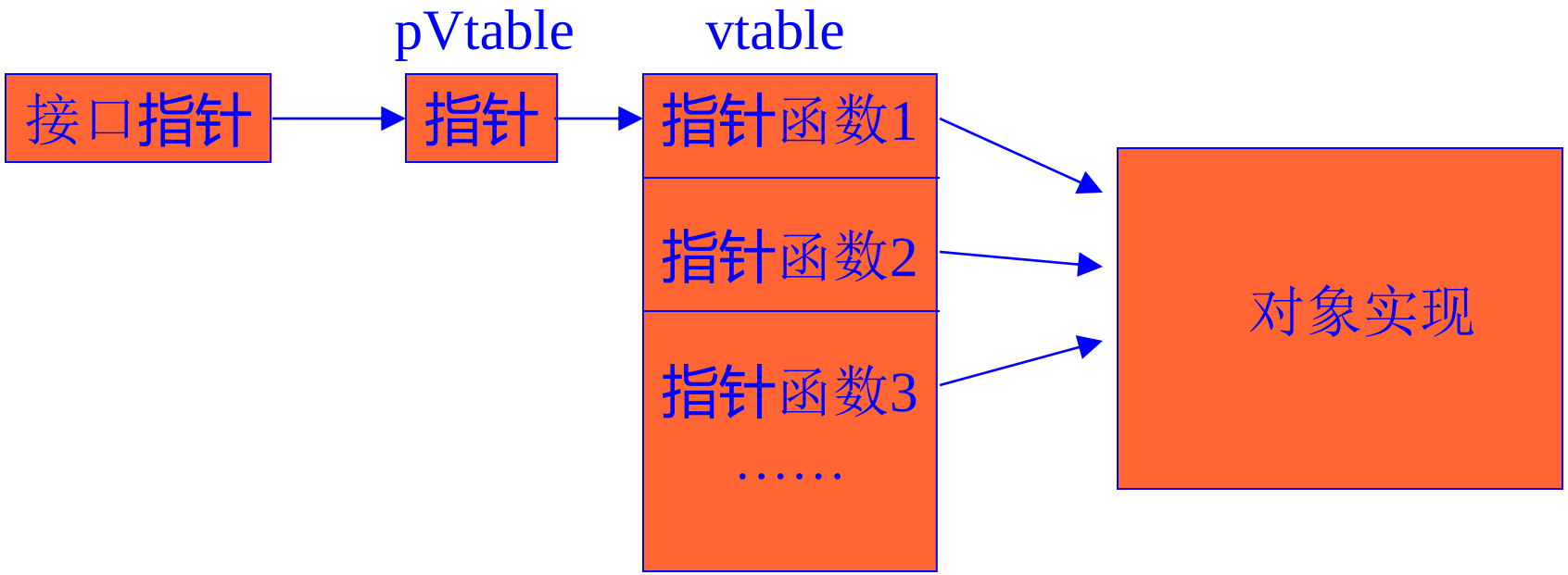
- – 通知COM对象增加它的引用计数。如果你进行了一次接口指针的拷贝，就必须调用一次这个方法，并且原始的值和拷贝的值两者都要用到。
-

QueryInterface() – 从COM对象请求一个接口指针。当coclass实现一个以上的接口时，就要用到这个方法。

- 当你用CoCreateInstance()创建对象的时候，你得到一个返回的接口指针。如果这个COM对象实现一个以上的接口（不包括IUnknown），你就必须用QueryInterface()方法来获得任何你需要的附加的接口指针

Release() – 通知COM对象减少它的引用计数。

COM接口二进制结构



COM对象

1.客户的交互实体

2.包括属性和方法，或者状态和操作

3.能够提供服务——通过*COM*接口

COM对象的标识——CLSID

CLASS ID 的简称

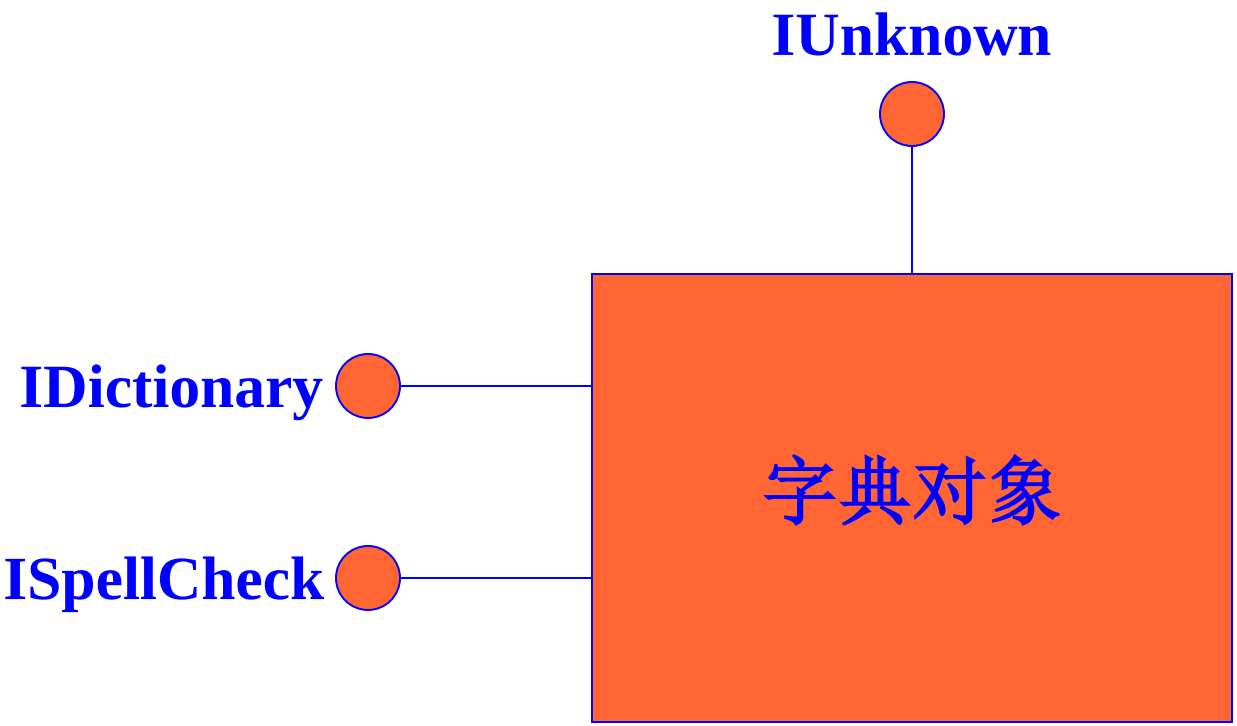
1.是GUID的一种用法

2.创建对象的时候必须要提供CLSID

3.COM对象的身份

- 身份是否一致的可判断性

COM对象和接口图示



COM对象与C++对象的比较

C++对象设计 = 多态性 + (某些)迟绑定 + (某些)封装性 + (实现/接口)继承

COM对象设计 = 多态性 + (完全)迟绑定 + (完全)封装性 + 接口继承 + 二进制重用性

使用和处理COM对象

每一种语言都有其自己处理对象的方式。例如，C++是在栈中创建对象，或者用new动态分配。因为COM必须独立于语言，所以COM库为自己提供对象管理例程

对COM对象管理和C++对象管理所做的一个比较：

- 创建一个新对象

  C++中，用new操作符，或者在栈中创建对象。

  COM中，调用COM库中的API。

  删除对象

  C++中，用delete操作符，或将栈对象踢出。

  COM中，所有的对象保持它们自己的引用计数。调用者必须通知对象什么时候用完这个对象。当引用计数为零时，COM对象将自己从内存中释放。

使用和处理COM对象

由此可见，对象处理的两个阶段：创建和销毁，缺一不可。

当创建*COM*对象时要通知*COM*库使用哪一个接口。

如果这个对象创建成功，*COM*库返回所请求接口的指针。然后通过这个指针调用方法，就像使用常规*C++*对象指针一样。

创建COM对象

为了创建COM对象并从这个对象获得接口，必须调用COM库的API函数，CoCreateInstance()：

HRESULT CoCreateInstance (REFCLSID rclsid, LPUNKNOWN pUnkOuter, DWORD dwClsContext, REFIID riid, LPVOID* ppv);

参数解释：

rclsid：coclass的CLSID，例如，可以传递CLSID_ShellLink创建一个COM对象来建立快捷方式。

pUnkOuter：这个参数只用于COM对象的聚合，利用它向现有的coclass添加新方法。参数值为null表示不使用聚合。

dwClsContext：表示所使用COM服务器的种类。最简单的COM服务器，为一个进程内（in-process）DLL，所以传递的参数值为CLSCTX_INPROC_SERVER。注意这里不要随意使用CLSCTX_ALL（在ATL中，它是个缺省值），因为在没有安装DCOM的Windows95系统上会导致失败。

riid：请求接口的IID。例如，可以传递IID_IShellLink获得IShellLink接口指针。

ppv：接口指针的地址。COM库通过这个参数返回请求的接口。

COM实例的创建

当你调用CoCreateInstance()时，它负责在注册表中查找COM服务器的位置，将服务器加载到内存，并创建你所请求的coclass实例。

以下是一个调用的例子，创建一个CLSID_ShellLink对象的实例并请求指向这个对象IShellLink接口指针。

```
HRESULT hr;
IShellLink* pISL;
 hr = CoCreateInstance ( CLSID_ShellLink, // coclass 的CLSID
                                          NULL, // 不是用聚合
                    CLSCTX_INPROC_SERVER, // 服务器类型
                                IID_IShellLink, // 接口的IID
                               (void**) &pISL ); // 指向接口的指针
 if ( SUCCEEDED ( hr ) )
 {
 // 用pISL调用方法
} else {
 // 不能创建COM对象，hr 为出错代码
 }
```

删除COM对象

不用释放COM对象，只要告诉它们你已经用完对象。

IUnknown是每一个COM对象必须实现的接口，它有一个方法，Release()。

- 调用这个方法通知COM对象你不再需要对象。一旦调用了这个方法之后，就不能再次使用这个接口，因为这个COM对象可能从此就从内存中消失了。

如果不释放接口，这个COM对象（包含代码的DLLs）将保留在内存中，这会增加不必要的开销。

```
// 像上面一样创建COM 对象， 然后，

 if ( SUCCEEDED ( hr ) )
{ // 用pISL调用方法
  // 通知COM 对象不再使用它
    pISL->Release();
}
```

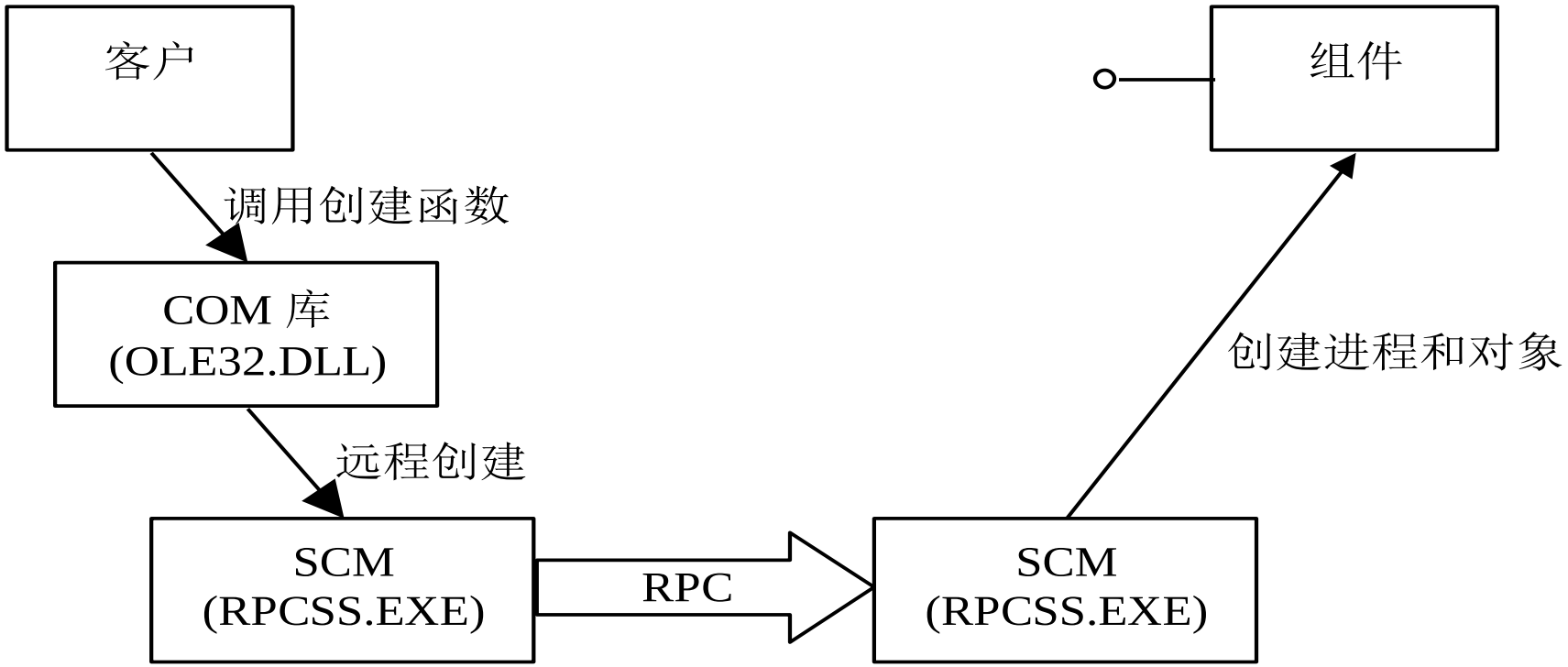
DCOM

DCOM代表Distributed Component Object Model。

DCOM（分布式组件对象模型,分布式组件对象模式）是一系列微软的概念和程序接口，利用这个接口，客户端程序对象能够请求来自网络中另一台计算机上的服务器程序对象。

DCOM基于组件对象模型（COM），COM提供了一套允许同一台计算机上的客户端和服务器之间进行通信的接口。

DCOM组件对象的创建过程



.net组件

COM定义了一个组件模型，组件可以使用不同的编程语言进行编写，其可以在本地进程中使用，也可以跨进程使用或者在网络上使用。

.Net组件的目标也是这样，但这些目标的实现方式是不同于COM的实现方式的。
.Net组件达到了与COM类似的目标，但是它引入了新概念，实现起来也更容易了。

.net组件

. NET组件与COM组件的区别

(1).NET组件具有自描述能力，不依赖于注册表。.NET组件与元数据一起编译，元数据作为编译组件(在.NET中称为Assembly)的一部分进行保存，元数据能够对组件进行详细描述，包括组件的接口、所支持的类型及开发人员添加的自定义信息。由于.NET组件无须在windows注册表中注册，大大简化了.NET应用程序的部署与安装。

.net组件

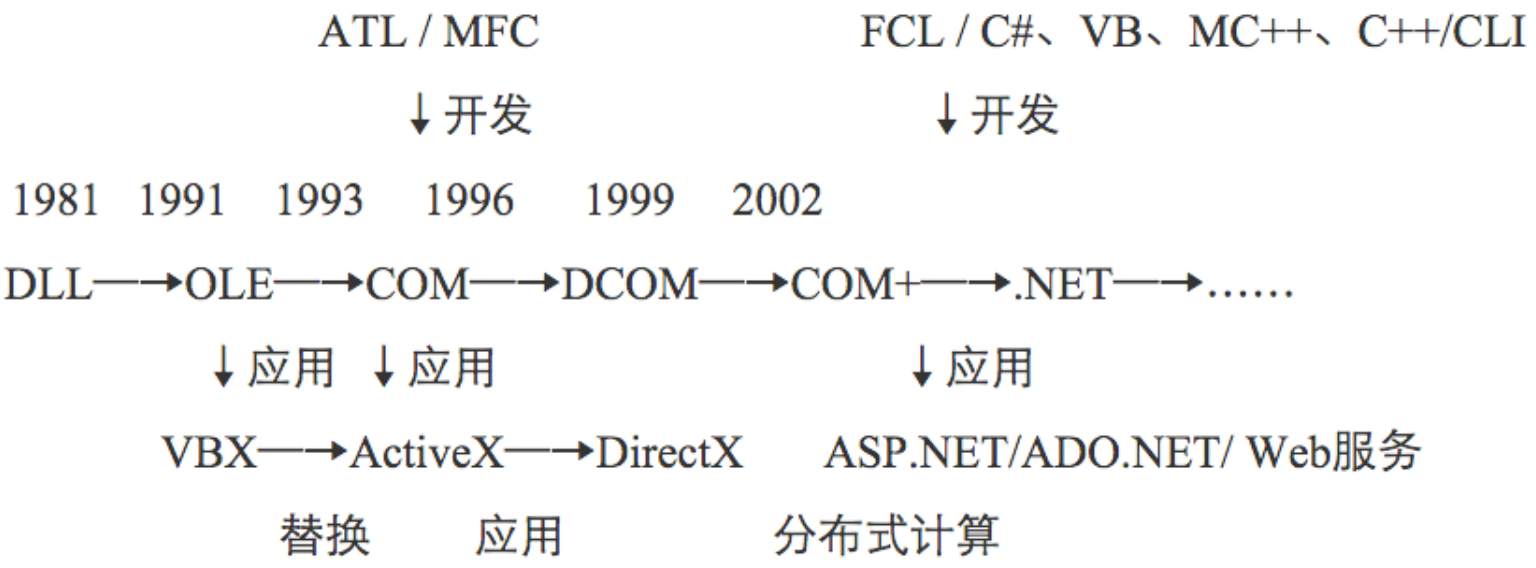
．NET组件与COM组件的区别

(2)．NET组件不再有“DLL Hell”问题。同一组件的多个版本可以同时共存。 .NET组件不像COM组件那样依赖于注册表中的GUID，因此同一组件的多个版本不会产生冲突，.NET应用程序可以通过编辑应用程序配置文件来指定使用哪个版本的DLL。

微软2002年推出了.NET框架

.net的核心技术就是用来代替COM组件功能的CLR（Common Language Runtime公共语言运行库），可采用各种编程语言，利用托管代码来访问（例如C#、VB、MC++），使用的是.NET的框架类库FCL（Framework Class Library）。

微软各组件技术之间的关系



项目	COM组件（C++）	DotNet组件(C＃)
元数据	在COM中，组件的所有信息存储在类型库中（也就是我们前面使用的TLB文件）。类型库包含了接口，方法，参数以及UUID等。这些通过IDL语言来进行描述。	在.Net组件中，它的元数据可以通过定制特性来扩展，所以你可以不用了解IDL。
内存管理	通过引用计数方法来进行组件内存管理。（参见第三章）客户程序必须调用AddRef()和Release()来进行计数管理，但计数为0的时候，销毁组件。	通过垃圾收集器来自动完成。
接口	拥有三种类型的接口，即从IUnknown继承的定制接口，分发接口以及双重接的接口。接口通过QueryInterface函数查询，然后使用。	通过强制类型转换来使用不同口。
方法绑定	COM一般是早期绑定，采用虚拟表来实现；对于分发接口采用了后期绑定。	通过System.Reflecting实现后期绑定。

项目	COM组件（C++）	DotNet组件(C＃)
数据类型	在定制接口中，所有C++的类型可以用于COM；但是对于双重接口和分发接口，只能使用VARIANT,BSTR等自动兼容的数据类型。	采用了Object替代VARIANT，能使用C＃的所有数据类型（用C#实现）。
组件注册	所有的组件必须进行注册。每个接口，组件都具有唯一的ID，包括CLSID和PROGID。	分为私有程序集和共享程序集。私有程序集能在一定程度上解决DLL版本冲突，重写等问题。共享程序集类似于COM。具体见参考文献[4]。
线程模式	使用单元模型，增加了实现难度，必须为不同的操作系统版本增加不同的单元类型。	通过System.Threading来进行处理，相对于COM的线程管理比较简单些。
错误处理	COM中通过实现HRESULT和ISupportErrorInfo接口，该接口提供了错误消息、帮助文件的链接、错误源，以及错误信息对象。	实现ISupportErrorInfo的对象会自动映射到详细的错误信息和一个.Net异常。
事件处理	通过实现连接点的接口IConnectionPoint和接口IConnectionPointContainer来实现事件处理.	通过event和delegate关键字提供事件处理机制。

谢谢