第三章　面向对象中间件框架

李 会 格

前言

- 计算机之间的通信如果缺乏一个统一的标准，那么就像不同语言的人互相交流，根本无法有效沟通。
- 为了解决分布式系统的通信问题，国际标准化组织提出了OSI-RM（Open System Interconnection Reference Model）标准，为网络的发展打下了基础。从那以后标准不断更新换代，也产生了许多基于统一标准的中间件框架，成为分布式系统中必不可少的通信桥梁。

- 本章首先介绍开放分布式处理参考模型RM-ODP，然后介绍一些经典的面向对象的中间件框架，帮助读者了解中间件框架的历史和中间件平台的内部工作原理和机制。

大纲

- 开放分布式处理参考模型
  - ✓ 面向对象技术
  - ✓ ODP标准组成
  - ✓ ODP功能组成
- CORBA框架
  - ✓ OMA介绍
  - ✓ CORBA介绍
  - ✓ CORBA的优势与发展
- COM组件模型
  - ✓ 组件的概念
  - ✓ COM的发展历程
  - ✓ COM组件
  - ✓ DCOM组件
  - ✓ COM+组件
  - ✓ .NET组件

面向对象技术

- 面向对象是相对于面向过程来讲的，该方法把事物的状态和行为视为一个对象(object)来看待，更贴近事物的自然运行模式。而类（Class）是对象整体的抽象，是对这些对象的属性和行为的统一规范性定义。

- 面向对象有以下四个特征：
  - 继承（Inheritance）
  - 封装（Encapsulation）
  - 抽象（Abstraction）
  - 多态（Polymorphism）

大纲

- 开放分布式处理参考模型
  - ✓ 面向对象技术
  - ✓ ODP标准组成
  - ✓ ODP功能组成
- CORBA框架
  - ✓ OMA介绍
  - ✓ CORBA介绍
  - ✓ CORBA的优势与发展
- COM组件模型
  - ✓ 组件的概念
  - ✓ COM的发展历程
  - ✓ COM组件
  - ✓ DCOM组件
  - ✓ COM+组件
  - ✓ .NET组件

ODP标准组成

- 开放分布式处理参考模型(Referenced Model of Open Distributed Processing, RM-ODP)是一种标准，它创建一个面向应用的参考模型来对付分布的应用，它能使应用之间实现互通和互操作。
- ODP标准还规定了使用于开放式分布处理领域内的其他标准确必须遵循的参考模型，可以说是对标准提出的标准。

- ODP标准的组成
  - ✓ 1、观点 Viwepoint
  - ✓ 2、透明性 Transparencies

ODP观点

- 观点把对于一个系统的说明分成若干个不同的侧面。每个观点对同一个分布式系统的某个不同侧面进行描述。

- RM-ODP框架提供的5种观点：
  - ✓ 企业观点 Enterprise viewpoint：描述了业务需求以及如何满足这些需求。
  - ✓ 信息观点 Information viewpoint：描述了信息的语义以及数据的结构和内容类型。
  - ✓ 计算观点 Computational viewpoint：描述了系统提供的功能及其功能分解。
  - ✓ 工程观点 Engineering viewpoint：描述为支持分布式交互所需的机制和功能。
  - ✓ 技术观点 Technology viewpoint：描述为了提供信息处理而选择的技术。

ODP透明性

- 透明性屏蔽了由系统的分布所带来的复杂性，使得用户在使用系统时不用去关心系统是分布的还是集中的，从而可以更加集中注意力放在需要做的工作上，提高工作效率。十大透明性：
  - ✓ 访问透明性：本地和远程访问方法之间应该没有明显的区别。
  - ✓ 位置透明性：在计算中，使用名称来标识资源，用户不需要知道资源的物理位置
  - ✓ 迁移透明性：如果对象（进程或数据）迁移，应该对用户隐藏这一点。
  - ✓ 失败透明性：如果发生软件或硬件故障，应该对用户隐藏这些故障
  - ✓ 重定位透明性：在交互中，如果某些对象被替换或移动而导致了某种程度上的不一致，系统仍然可以保持运行。

ODP透明性

- 十大透明性(续):
  - ✓ 复制透明性

    用户不必关心数据库在网络中各个结点的复制情况，更新引起的问题将由系统去处理
  - ✓ 持久透明性

    开发人员很少或根本不做任何工作的情况下存储和检索持久性数据，比如Java序列化对象到文件，不需要做很多的努力。
  - ✓ 事务处理透明性

    在分布式环境中往往需要维护一组相关对象之间的协调。对用户屏蔽协调相关的信息
  - ✓ 缩放透明性

    系统应该能够在不影响应用程序算法的情况下增长或缩小。
  - ✓ 并发透明性

    用户和应用程序应该能够访问共享数据或对象，而不会相互干扰

ODP功能组成

- RM-ODP平台的通用功能具有如下四种：
  - ✓ 管理功能（Management Functions）：是对分布式系统中的基本组成要素的管理，比如节点管理、对象管理、对象串管理等等。
  - ✓ 协作功能（Coordination Functions）：提供多个对象之间的交互协调从而达到某种功能需求，比如事件通知、取消激活与再激活、复制迁移、事务处理等等。
  - ✓ 仓库功能（Repository Functions）：提供分布式系统的信息存储与检索功能，比如信息组织、重定位、类型仓库等等。
  - ✓ 安全功能（Security Functions）：提供分布式系统的安全管理机制，比如访问控制、安全审核、用户认证、密钥管理等等。

大纲

- 开放分布式处理参考模型
  - ✓ 面向对象技术
  - ✓ ODP标准组成
  - ✓ ODP功能组成
- CORBA框架
  - ✓ OMA介绍
  - ✓ CORBA介绍
  - ✓ CORBA的优势与发展
- COM组件模型
  - ✓ 组件的概念
  - ✓ COM的发展历程
  - ✓ COM组件
  - ✓ DCOM组件
  - ✓ COM+组件
  - ✓ .NET组件

OMA介绍

- 对象管理组织(OMG, Object Management Group)是由几百家信息系统供应商组成的联合体。由OMG制定的规范——对象管理结构(OMA, Object Management Architecture)和它的核心——公共对象请求代理体系结构(CORBA, Common Object Request Broker Architecture)，构成了一个完整的体系结构，以其足够的灵活性、丰富的形式适用于各类分布式系统。

- OMA包括两部分：
  - ✓ 对象模型（Object Model）：定义了如何描述分布式环境中的对象
  - ✓ 参考模型（Reference Model）：描述对象之间的交互。

OMA对象模型

- 对象是一个被封装的实体，它具有一个不可改变的标识，并能给客户用户提供一个或多个服务。

```
interface printer
{
    attribute model;
    void print(in string buffer);
};
```

- 对象的访问方式是通过向对象发出请求来完成的。请求信息包括目标对象、所请求的操作、0个或多个实际参数和可选的请求上下文（描述环境信息）。每个对象的实现和位置，对客户都是透明的。

OMA参考模型

- OMA参考模型定义了一条为对象所公用的通信总线，即ORB(Object Request Broker)。
- OMA定义了对象进出这一总线的界面。包括：

✓ 公共设施 (Common Facilities)

通用领域内定义的对象，是面向最终用户的应用

✓ 域界面 (Domain Interface)

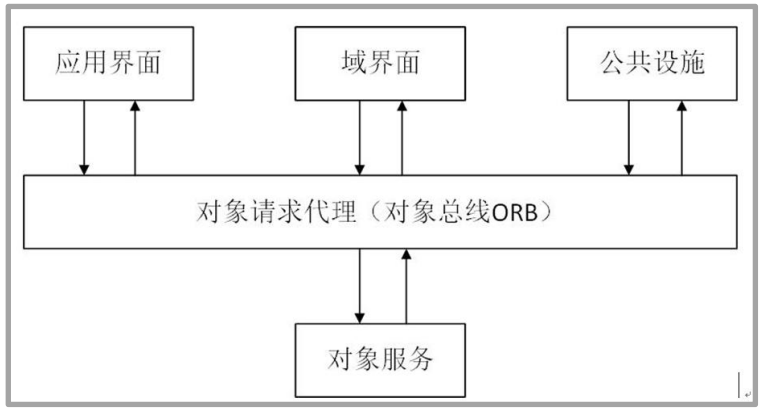
专用领域内定义的对象，是针对某一特殊应用领域提供的接口

✓ 对象服务 (Object Services)

为公共设施和各种应用对象提供的基本服务的集合对象服务，是与具体的应用领域无关的接口

✓ 应用界面(Application Interface)

由厂商针对某一具体应用所定义的界面。



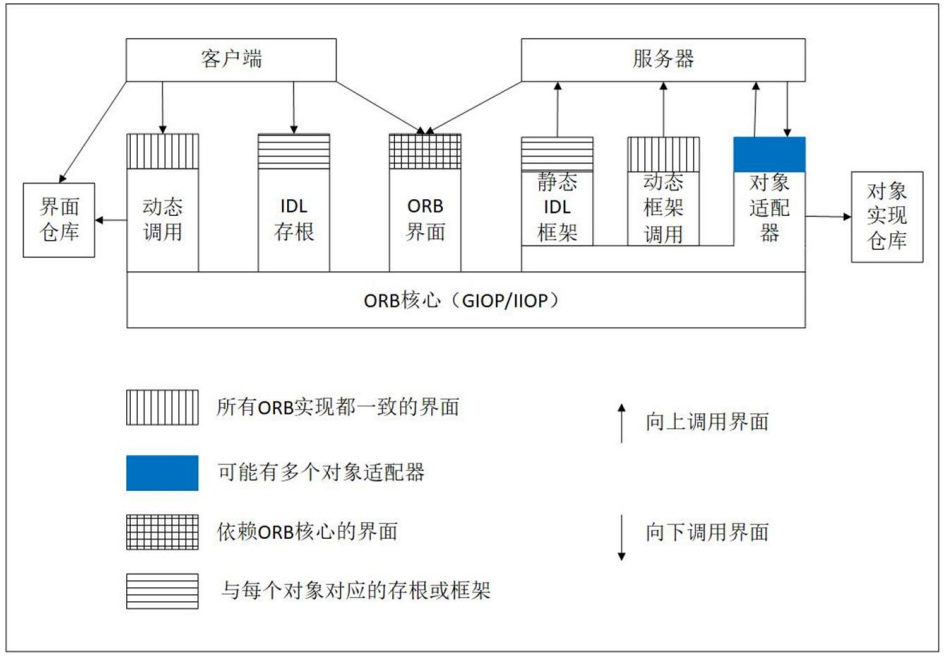
OMA参考模型

大纲

- 开放分布式处理参考模型
  - ✓ 面向对象技术
  - ✓ ODP标准组成
  - ✓ ODP功能组成
- CORBA框架
  - ✓ OMA介绍
  - ✓ CORBA介绍
  - ✓ CORBA的优势与发展
- COM组件模型
  - ✓ 组件的概念
  - ✓ COM的发展历程
  - ✓ COM组件
  - ✓ DCOM组件
  - ✓ COM+组件
  - ✓ .NET组件

CORBA介绍

- 公共对象请求代理体系结构（CORBA）规范的目标是推广一种面向对象的方法来构建和集成分布式软件应用程序。
- CORBA规范包括如下：
  1. ORB核心（Object Request Broker Core）
  2. 接口定义语言和语言映射
  3. 存根和框架
  4. 动态调用
  5. 对象适配器
  6. 接口仓库和实现仓库
  7. ORB之间的互操作



CORBA体系结构

ORB核心

- ORB(Object Request Broker)是客户机和服务器应用程序之间的中介。

  ORB核心的功能是把客户发出的请求传递给目标对象，并把目标对象的执行结果返回给发出请求的客户。

  其重要特征是：提供了客户和目标对象之间的交互透明性。

- 屏蔽的内容包括：
  - ✓ 对象位置：客户不必知道目标对象的物理位置。它可能与客户一起驻留在同一个进程中或同一机器的不同进程中，也有可能驻留在网络上的远程机器中。
  - ✓ 对象实现：客户不必知道有关对象实现的具体细节。例如，设计对象所用的编程语言、对象所在节点的操作系统和硬件平台等。

ORB核心

- (续)屏蔽的内容包括：
  - ✓ 对象的执行状态：当客户向目标对象发送请求时，它不必知道当时目标对象是否处于活动状态（即是否处于正在运行的进程中）。此时，如果目标对象不是活动的，在把请求传给它之际，ORB会透明地将它激活。
  - ✓ 对象通信机制：客户不必知道ORB所用的下层通信机制，如，TCP/IP、管道、共享内存、本地方法调用等。
  - ✓ 数据表示：客户不必知道本地主机和远程主机对数据表示方式，如高位字节在前还是在后等，是否有所不同。

✓ 通过ORB，客户机应用程序可以在不知道服务器应用程序的位置或服务器应用程序将如何完成请求的情况下请求服务。这使得程序员不需要关心底层编程的问题，将更多时间与精力集中在应用层面的设计。

接口定义语言和语言映射

- 在客户向目标对象发送请求之前，它必须知道目标对象所能支持的服务。对象是通过接口定义来说明它所能提供的服务。
- 在CORBA中，对象的接口是利用OMG IDL（OMG Interface Definition Language）来定义的。OMG IDL是一个纯说明性语言，与具体主机上的编程语言无关。这就很自然将接口与对象实现分离，使得不同的语言来实现的对象之间可以进行互操作。
- 语言映射是指将IDL的特性映射为具体语言的实现。在OMG制定的语言映射标准中已经涵盖多种语言，其中包括C、C++、SmallTalk、Ada95、Cobol、Java等等。

存根和框架

- 除了将IDL特性映射到特定的语言之外，OMG IDL编译器还根据接口定义生成客户端存根和服务端框架。
  - 存根的作用是代表客户端创建并发出请求。客户端只要把请求交给存根，不用去关心ORB是否存在，存根则负责将请求的参数进行封装和发送，以及接收返回数据和解包。
  - 框架的作用则是把请求交给CORBA对象实现。它解包请求参数，标识客户端请求的服务，调用对象实现，封装执行结果，并将其返回给客户机。

- 因为存根和框架都是根据用户接口定义编译的，所以它们都与特定的接口相关。而且在请求实际发生之前，存根和框架分别直接连接到客户端程序和对象实现。因此，通过存根和框架进行的调用通常称为静态调用。

动态调用

- CORBA还支持两种用于动态调用的接口：
  - ✓ 动态调用接口（Dynamic Invocation Interface）：支持客户端的动态请求调用。
  - ✓ 动态框架接口（Dynamic Skeleton Interface）：支持服务端的动态对象调用。

- 动态调用的作用：
  - ✓ 通过动态调用接口，客户端可以在程序执行过程中动态地向任何对象发送请求，而不必像静态调用那样在编译时就需要提供目标对象的接口信息。
  - ✓ 使用动态调用接口时需要人为定义所需操作、请求参数等等。
  - ✓ 动态调用接口允许用户在没有静态存根的情况下发送请求。类似地，动态框架接口允许用户在没有静态框架信息的条件下来获取对象实现。

对象适配器

- 对象适配器是对象实现和ORB之间的连接桥梁。而且，对象适配器极大减轻了ORB的任务，从而简化了ORB的设计。

- 具体来说，对象适配器执行以下操作：
  1. 对象注册：使用对象适配器提供的操作，CORBA实现库中具有编程语言形式的实体可以注册为CORBA的对象实现。
  2. 对象引用生成：对象适配器为CORBA对象生成对象引用。客户端应用程序通过对象引用访问对象实例。
  3. 服务器进程激活：如果目标对象所在的服务器在客户端发出请求时没有运行，那么对象适配器会自动激活服务器。
  4. 对象激活：根据需要自动激活目标对象。
  5. 对象撤销：如果在预定的时间片内没有向目标对象发送请求，那么对象适配器将撤消该对象以节省系统资源。
  6. 对象调用：对象适配器将请求分配给已注册的对象。

接口仓库和实现仓库

- ORB提供了两个用于存储有关对象信息的服务：接口仓库和实现仓库。
  - ✓ 接口仓库存储每个接口的模块信息，包括IDL编写的接口定义、常量、类型等。
    - 本质上，它也是一个对象。应用程序可以调用接口仓库的方法，就像其他CORBA对象提供的方法一样。
    - 接口仓库允许应用程序在运行过程中访问OMG IDL类型的系统。例如，当应用程序在运行过程中遇到未知类型的对象时，就能利用接口仓库来访问系统中的所有接口信息。
    - 由于这些优良特性，接口仓库的引入很好地支持了CORBA的动态调用。
  - ✓ 实现仓库所完成的功能类似于接口仓库，不同的是它存储对象实现的信息。当需要激活某一对象类型的实例时，ORB便会访问实现仓库。

ORB之间的互操作

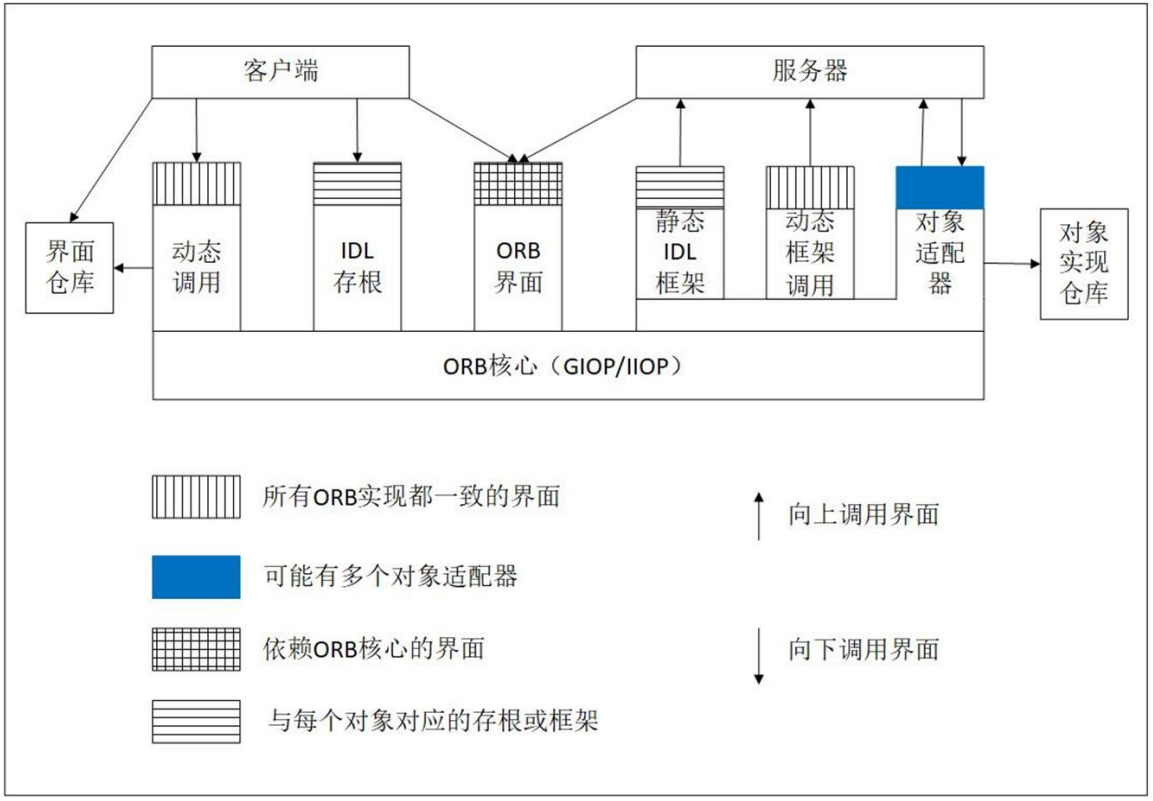
- 在发布CORBA 2.0之前，ORB产品的最大缺点是：不同厂商所提供的ORB产品之间并不能互操作。为了达到异构ORB系统之间互操作的目的，CORBA 2.0规范中定义了标准通信协议GIOP（General Inter-ORB Protocol）。GIOP协议由3个部分组成：

✓ 公共数据表示（Common Data Representation）：在对CORBA分布式对象进行远程调用时，用于表示作为参数或结果传递的结构化或原始数据类型。

✓ GIOP消息格式（GIOP Message Formats）：它定义了用于ORB间对象请求、对象定位和信道管理的7种消息的格式和语义。

✓ GIOP传输层假设（GIOP Transport Assumptions）：GIOP协议可运行于多种传输层协议之上，只要传输层协议是面向连接的、可靠的，所传递的数据可以为任意长度的字节流，提供错序通知功能，连接的发起方式可以映射到TCP/IP这样的一般连接模型。

CORBA体系结构



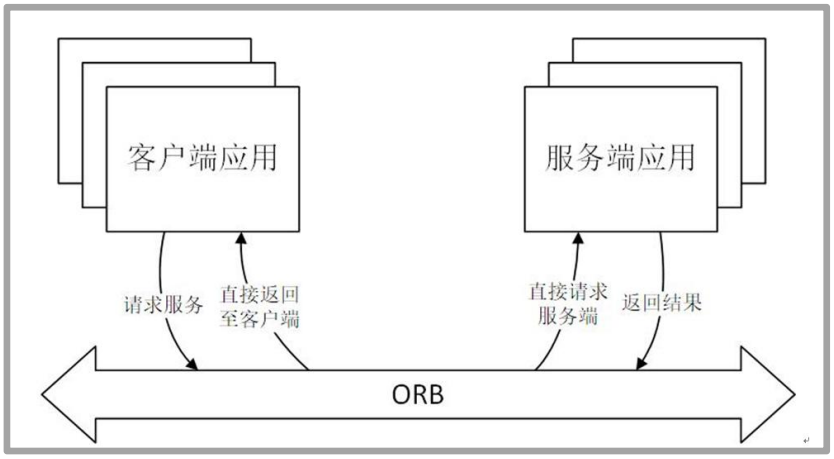
- CORBA包括如下：
  1. ORB核心
  2. 接口定义语言和语言映射
  3. 存根和框架
  4. 动态调用
  5. 对象适配器
  6. 接口仓库和实现仓库
  7. ORB之间的互操作

大纲

- 开放分布式处理参考模型
  - ✓ 面向对象技术
  - ✓ ODP标准组成
  - ✓ ODP功能组成
- CORBA框架
  - ✓ OMA介绍
  - ✓ CORBA介绍
  - ✓ CORBA的优势与发展
- COM组件模型
  - ✓ 组件的概念
  - ✓ COM的发展历程
  - ✓ COM组件
  - ✓ DCOM组件
  - ✓ COM+组件
  - ✓ .NET组件

CORBA优势

1. 正式分离应用程序的客户端和服务器部分。
   - 在ORB的协助下，客户机应用程序只需要知道它们可以发出什么请求，以及如何发出请求；它们不需要使用服务器或数据格式的任何实现细节进行编码。
   - 服务器应用程序只需要知道如何满足请求，而不需要知道如何将数据返回给客户端应用程序。



CORBA开发模式

CORBA优势

2. 从逻辑上将应用程序分离为可以执行特定任务的对象，称之为操作。
   - 在面向对象计算中，对象是组成应用程序的实体，而操作是服务器可以对这些对象执行的任务。例如，银行应用程序可以有客户帐户的对象，以及存款、取款和查看帐户余额的操作。
3. 提供数据封送处理以使用远程或本地计算机应用程序发送和接收数据。
   - 例如，CORBA模型根据需要自动格式化Big-Endian或Little-Endian。
4. 对应用程序隐藏网络协议接口。
   - CORBA模型处理所有的网络接口过程中，应用程序只看到对象。这些应用程序可以在不同的机器上运行。
   - 由于所有的网络接口代码都由ORB处理，因此如果以后将应用程序部署到支持不同网络协议的计算机上，则不需要任何与网络相关的更改。

CORBA发展历程和产品

- CORBA主要版本的发展历程如下：
  - 1990年11月，OMG发表《对象管理体系指南》，初步阐明了CORBA的思想；
  - 1991年10月，OMG推出1.0版，其中定义了接口定义语言（IDL）、对象管理模型以及基于动态请求的API和接口仓库等内容；
  - 1991年12月，OMG推出了CORBA 1.1版，在澄清了1.0版中存在的二义性的基础上，引入了对象适配器的概念；
  - 1996年8月，OMG基于以前的升级版本，完成了2.0版的开发，该版本中重要的内容是对象请求代理间协议（IIOP，Internet Inter-ORB Protocol）的引入，用以实现不同厂商的ORB真正意义上的互通;
  - 1998年9月，OMG发表了CORBA 2.3版，增加了支持CORBA对象的异步实时传输、服务质量规范等内容。宣布支持CORBA 2.3规范的中间件厂商包括Inprise(Borland)、Iona、BEA System等著名的CORBA产品生产商。

  - 比较著名的CORBA产品有IONA的Orbix、ExpertSoft的Power CORBA以及Inprise的Visibroker。还有一些优秀的成果可供研究，如Mico, Orbacus, TAO等。

CORBA发展局限

- 尽管有多家供应商提供CORBA产品，但是仍找不到能够单独为异种网络中的所有环境提供实现的供应商。不同的CORBA实现之间会出现缺乏互操作性的现象，从而造成一些问题。
- 而且由于供应商常常会自行定义扩展，而CORBA又缺乏针对多线程环境的规范，对于像C或C++这样的语言，源码兼容性并未完全实现。
- 另外CORBA过于复杂。要熟悉CORBA并进行相应的设计和编程，需要许多个月来掌握，而要达到专家水平则需要好几年。

大纲

- 开放分布式处理参考模型
  - ✓ 面向对象技术
  - ✓ ODP标准组成
  - ✓ ODP功能组成
- CORBA框架
  - ✓ OMA介绍
  - ✓ CORBA介绍
  - ✓ CORBA的优势与发展
- COM组件模型
  - ✓ 组件的概念
  - ✓ COM的发展历程
  - ✓ COM组件
  - ✓ DCOM组件
  - ✓ COM+组件
  - ✓ .NET组件

组件的概念

- 组件（Component）技术是一种优秀的软件重用技术。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。

- 采用组件开发软件就像搭积木一样容易。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。

大纲

- 开放分布式处理参考模型
  - ✓ 面向对象技术
  - ✓ ODP标准组成
  - ✓ ODP功能组成
- CORBA框架
  - ✓ OMA介绍
  - ✓ CORBA介绍
  - ✓ CORBA的优势与发展
- COM组件模型
  - ✓ 组件的概念
  - ✓ COM的发展历程
  - ✓ COM组件
  - ✓ DCOM组件
  - ✓ COM+组件
  - ✓ .NET组件

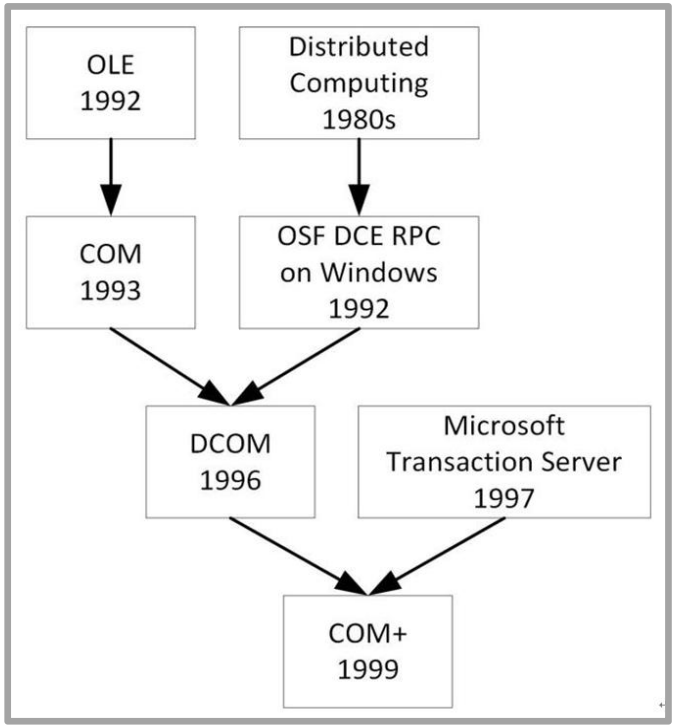
COM概念

- 以组件对象模型（COM, Component Object Model）为代表的组件技术大大改变了软件开发的方式。用户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组件。在构建应用时，像装零件一样将这些组件有机地组装起来就成为一个系统。组件技术的特性如下：
  1. 组件可替换，以便随时进行系统升级和定制；
  2. 可以在多个应用系统中重复利用同一个组件；
  3. 可以方便的将应用系统扩展到网络环境下；
  4. 部分组件与平台和语言无关，所有的程序员均可参与编写。

COM概念

- 组件对象模型COM是微软公司于1993年推出的基于二进制接口标准的软件组件。它用于在大量编程语言中实现进程间通信和动态对象创建。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的历史发展



COM组件的发展

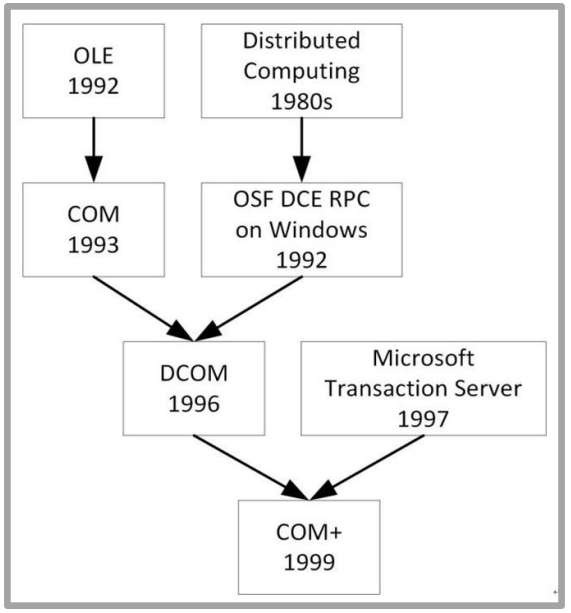
- COM一词在软件开发行业中经常作为一个总括术语使用，它经历了DLL、OLE、COM、DCOM、COM+的演变过程。

COM的历史发展

- 静态链接库只能在编译阶段使用，因此每个程序都包含所有用到的公共库函数，这样会造成很大的浪费，既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。
- 动态链接库（DLL, Dynamic Link Library）因运而生。
  - ✓ 数据的库文件采用动态链接，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（每个DLL都有一个类似于main的入口函数），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。
- DLL还不能算组件技术，且看似与COM组件没有关系，但它是软件重用的鼻祖，其中的动态链接特性在COM中不可或缺。可以说COM技术继承了DLL的优点并进行了扩展。

COM的历史发展

- 对象连接与嵌入技术（OLE, Object Linking and Embedding）
  - OLE是微软推出的一种允许将应用程序作为对象进行互相连接的机制。OLE有两个版本，分别为1.0和2.0。其中1.0版本是以windows的DDE技术来搭建的，性能上差强人意。而OLE 2.0引入了COM技术，并充分发挥了COM的优势，很大程度上提升了运行效率。
- COM
  - 在1993年，COM规范才正式诞生。COM是一种技术标准，其商业品牌则称为ActiveX。ActiveX是Microsoft遵循COM规范而开发的用于Internet的一种开放集成平台。一般常用的COM组件有两类：ActiveX DLL和ActiveX控件。



COM组件的发展

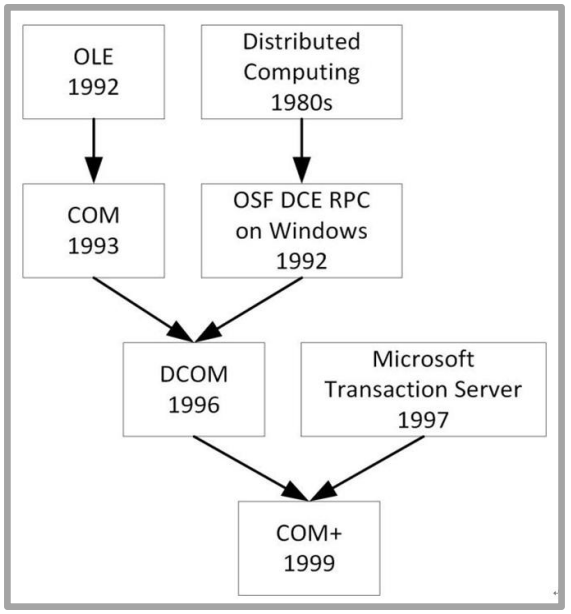
COM的历史发展

➢DCOM

- 1996年，微软又推出DCOM（Distributed COM，它屏蔽了分布式系统中COM对象的位置差异性，使得程序员不需要关心COM对象的实际位置就可以直接调用COM对象。

➢COM+

- 1999年微软又在Windows中引入COM+技术，它将许多编码性的工作转换成了管理性的操作。这是目前COM技术的最新应用。COM+目前包含两个功能领域：用于构建软件组件的基本编程架构（最初由COM规范定义），以及一个集成的组件服务套件和一个用于构建复杂软件组件的相关运行环境。



COM组件的发展

大纲

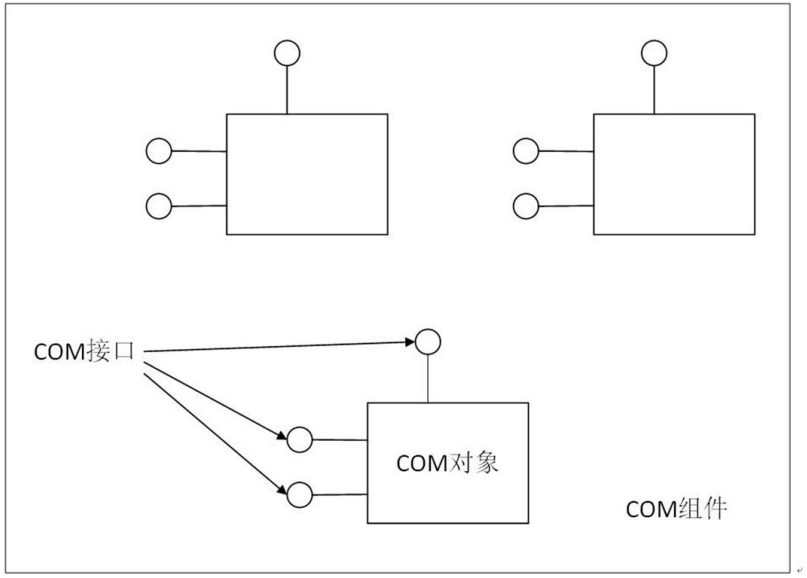
- 开放分布式处理参考模型
  - ✓ 面向对象技术
  - ✓ ODP标准组成
  - ✓ ODP功能组成
- CORBA框架
  - ✓ OMA介绍
  - ✓ CORBA介绍
  - ✓ CORBA的优势与发展
- COM组件模型
  - ✓ 组件的概念
  - ✓ COM的发展历程
  - ✓ COM组件
  - ✓ DCOM组件
  - ✓ COM+组件
  - ✓ .NET组件

COM定义

- COM的定义
  - 在Microsoft公司的开发者网络（MSDN, Microsoft Developer Network）中是这样定义COM的：“COM是软件组件互相通信的一种方式，它是一种二进制的网络标准，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。”
- 按照COM的标准规范，用户可以开发自定义的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技术概念

- 在COM技术中，有以下比较重要的概念：
  1. 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。
  2. COM对象：通过COM接口提供服务。COM对象与其调用方之间的交互被建模为客户机/服务器关系，客户机是从系统请求COM对象的调用者，而服务器是向客户机提供服务的COM对象。
  3. COM组件：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，是COM对象的载体，能够满足对组件架构的所有需求。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。



COM接口、对象和组件模型示意

COM接口

- 在COM技术中，组件和接口是其核心概念。

- 在COM对象通过接口（成员函数的集合）公开其功能。COM接口定义组件的预期行为和责任，并指定一个强类型协定，该协定提供一小组相关操作。COM组件之间的所有通信都是通过接口进行的，组件提供的所有服务都通过其接口公开。调用者只能访问接口成员函数。内部状态对调用者不可用，除非它在接口中公开

- 每个接口都有自己唯一的接口标识符，名为IID，它消除了与人类可读名称可能发生的冲突。IID是一个全局唯一标识符（GUID, Globally Unique Identifier。创建新接口时，必须为该接口创建新标识符。当调用方使用接口时，它必须使用唯一标识符。

COM接口

```
1   class IUnknown
2   {
3       public:
4           virtual HRESULT _stdcall QueryInterface(const IID& iid,void** ppv) = 0;
5           virtual ULONG _stdcall AddRef() = 0;
6           vittual ULONG _stdcall Release() = 0;
7   };
```

IUnknown接口示意图

- 所有接口都继承自IUnknown接口，所有COM对象都需要实现IUnknown接口。
- IUnknown接口包含多态性和实例生存期管理的基本COM操作。
  - Query Interface成员函数为COM提供多态性。调用QueryInterface在运行时确定COM对象是否支持特定接口。如果COM对象实现了请求的接口，那么它将在ppvObject out参数中返回接口指针，否则返回NULL。
  - COM对象实例的生存期由其引用计数控制。IUnknown成员函数AddRef和Release控制计数。AddRef增加计数，Release减少计数。当引用计数为零时，释放成员函数可能会释放实例。

大纲

- 开放分布式处理参考模型
  - ✓ 面向对象技术
  - ✓ ODP标准组成
  - ✓ ODP功能组成
- CORBA框架
  - ✓ OMA介绍
  - ✓ CORBA介绍
  - ✓ CORBA的优势与发展
- COM组件模型
  - ✓ 组件的概念
  - ✓ COM的发展历程
  - ✓ COM组件
  - ✓ DCOM组件
  - ✓ COM+组件
  - ✓ .NET组件

DCOM组件

- 微软在1996年提出的DCOM（ Distributed Component Object Model ）是分布式环境中的COM技术，更确切地说，DCOM是对COM的增强与扩展。就其添加到COM的扩展而言，DCOM必须解决以下问题：
  1. 编组–序列化和反序列化参数和返回值。
  2. 分布式垃圾回收–确保在例如客户端进程崩溃或网络连接丢失时，释放接口客户端保留的引用。
  3. 它必须将客户端浏览器中保存的数十万个对象与一次传输结合在一起，以最大程度地减少带宽利用率。
- 解决这些问题的关键因素之一是使用DCE/RPC（分布式计算环境/远程过程调用）作为DCOM背后的底层RPC机制。DCE/RPC严格定义了关于编组和谁负责释放内存的规则。

DCOM组件

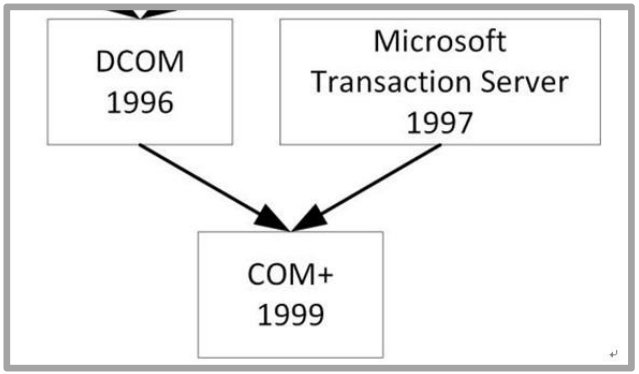
- 在当时DCOM是CORBA的主要竞争对手。这两种技术的支持者认为它们有一天会成为互联网上代码和服务重用的模型。
- 这两种技术中在因特网防火墙上、在未知和不安全的机器上工作都会遇到困难。
- 这意味着普通的HTTP请求与Web浏览器结合在一起会战胜这两种技术。微软曾经试图通过向DCE/RPC添加一个称为ncacn_http（网络计算体系结构面向连接的协议）的额外http传输来阻止这种情况，但失败了。后来恢复了这个功能，以支持通过HTTP的Microsoft Exchange 2003连接。

大纲

- 开放分布式处理参考模型
  - ✓ 面向对象技术
  - ✓ ODP标准组成
  - ✓ ODP功能组成
- CORBA框架
  - ✓ OMA介绍
  - ✓ CORBA介绍
  - ✓ CORBA的优势与发展
- COM组件模型
  - ✓ 组件的概念
  - ✓ COM的发展历程
  - ✓ COM组件
  - ✓ DCOM组件
  - ✓ COM+组件
  - ✓ .NET组件

COM+组件

- MTS（Microsoft Transaction Server）是微软为其Windows NT操作系统推出的一个中间件产品，由于它具有强大的分布事务支持、安全管理、资源管理和多线程并发控制等特性，使其成为在Windows平台上开发大型数据库应用系统的首选产品。由于MTS屏蔽了底层实现的复杂性，极大地简化了这类应用的开发，程序员可以将精力集中在业务逻辑上，因而有效地提高了软件的开发效率。



COM+的由来

- COM+是微软组件对象模型（COM）和微软事务服务器（MTS）的进一步结合的成果。COM+构建并扩展了使用COM、MTS和其他基于COM的技术编写的应用程序。
  - COM+处理许多以前必须自己编程的资源管理任务，比如线程分配和安全性。
  - COM+还通过提供线程池、对象池和实时对象激活，使应用程序更具可伸缩性。
  - COM+还通过提供事务支持来帮助保护数据的完整性，即使一个事务跨越网络上的多个数据库也是如此。

COM+组件

- COM+并不是COM的简单升级。COM+的底层结构仍然以COM为基础，它几乎包容了COM的所有内容。COM+综合了COM、DCOM和MTS的技术要素，它把COM组件软件提升到应用层而不再是底层的软件结构。它通过操作系统的各种支持，使组件对象模型建立在应用层上，同时把所有组件的底层细节留给操作系统。因此，COM+与操作系统的结合更加紧密，且不再局限于COM的组件技术，它更加注重于分布式网络应用的设计和实现

- COM+标志着微软的组件技术达到了一个新的高度。它不再局限于一台机器上的桌面系统，它把目标指向了更为广阔的企业内部网，甚至Internet国际互连网络。COM+与多层结构模型以及Windows操作系统为企业应用或Web应用提供了一套完整的解决方案。

大纲

- 开放分布式处理参考模型
  - ✓ 面向对象技术
  - ✓ ODP标准组成
  - ✓ ODP功能组成
- CORBA框架
  - ✓ OMA介绍
  - ✓ CORBA介绍
  - ✓ CORBA的优势与发展
- COM组件模型
  - ✓ 组件的概念
  - ✓ COM的发展历程
  - ✓ COM组件
  - ✓ DCOM组件
  - ✓ COM+组件
  - ✓ .NET组件

.NET组件

- 微软2002年推出的.Net组件与COM有着相似的目标，即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。但是它引入了新概念，实现起来也更容易。

- .NET的核心技术是用来代替COM组件功能的公共语言运行库（CLR, Common Language Runtime）的。.NET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）。

.NET组件

- .Net提供了一种全新的建立和展开组件的方法，也就是程序集Assemblies
  - 使用COM进行编程，开发者必需要在服务器上注册组件，系统注册表中的组件的信息必须被更新。这样做的目的是保证组件的中心位置，以使COM能够找到合适的组件。
  - 使用.Net的Assemblies，Assembly文件把所有需要的Meta Data都压入一个叫Manifests的一个特殊的段中。在.Net中编程，要使Assembly对用户有效，只要简单地把他们放在一个目录中即可。当客户程序请求一个特别的组件的实例的时候，.Net运行期（Runtime）在同一个目录搜寻Assembly，在找到后，分析其中的Manifest，以取得这个组件所提供的类的信息。由于组件的信息是放在Manifest里的，所以开发者就没有必要把组件注册到服务器上。因此，就可以允许几个相同的组件安全地共存在一个相同的机器上了。

COM组件与.NET组件的对比

项目	COM组件（C++）	.Net组件（C＃）
元数据	组件的所有信息存储在类型库中。类型库包含了接口，方法，参数以及UUID等。通过IDL语言来进行描述。	元数据可以通过定制特性来扩展，不用了解IDL。
内存管理	通过引用计数方法来进行组件内存释放管理。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的接口。

COM组件与.NET组件的对比

项目	COM组件（C++）	.Net组件（C＃）
方法绑定	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。	通过反射机制（System.Reflecting）实现后期绑定。
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。

COM组件与.NET组件的对比

项目	COM组件（C++）	.Net组件（C＃）
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的线程管理更为简单。
错误处理	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IConnectionPoint接口和IConnectionPointContainer接口来实现事件处理。	通过event和delegate关键字提供事件处理机制。

本章小结

- 本章首先介绍了国际标准化组织ISO提出的RM-ODP标准，从面向对象的角度阐述其标准组成与功能组成；
- 接着介绍了由OMG提出的OMA体系结构与CORBA规范，同时分析了CORBA相比传统开发模式的优势所在；
- 最后介绍了微软公司提出的COM组件对象模型和.NET组件，并介绍了两者之间的区别。