1.CORBA：Common Object Request Broker Architecture,通用对象请求代理体系。是由对象管理组(Object Management Group, OMG)制定的一种标准的面向对象分布式应用程序体系规范，旨在为异构分布式环境中，硬件和软件系统的互联而提出的一种解决方案。

2.解决异构分布式系统两条主要原则：

(1).寻求独立于平台的模型和抽象，这样有助于解决大部分问题。

(2).在不牺牲太多性能的前提下，尽可能隐藏底层的复杂细节。

3.对象管理组(OMG)定义的两类与平台无关的分布式对象模型：

(1).对象模型(Object Model)：用来定义在一个异构环境中，如何描述分布式对象接口。对象模型将对象定义为永恒不变、始终唯一的，对象模型封装实体，这些实体只能被严格定义的接口访问，客户机通过向对象发请求，才能使用对象提供的服务，对象的实现细节和它的位置对于客户机上隐藏的。

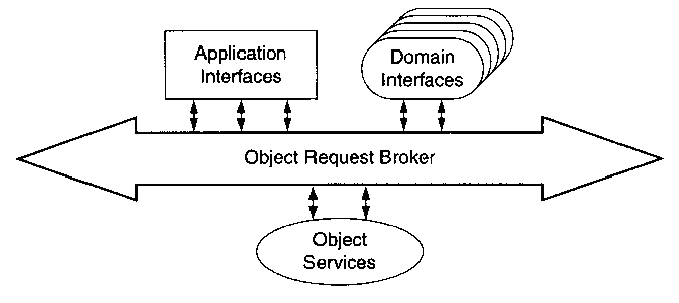
(2).引用模型(Reference Model)：用来说明对象之间如何交互。引用模型提供的接口种类，通常是按对象接口编组，所有接口种类有一个对象请求代理(Object Request Broker,ORB)按概念链接在一起。通常，一个ORB可以在客户机和对象之间进行通信，当请求发送给对象时，透明地激活那些没有运行的对象。

4.对象管理组(OMG)的接口种类：

(1).对象服务接口：Object Services,OS.是与领域无关(水平定向)的接口，对象管理组定义的命名服务(Naming Service)和交易服务(Trading Service)都是对象服务。对象服务通常被认为是分布式计算[**架构**](http://lib.csdn.net/base/architecture)的核心部分。

(2).领域接口：Domain Interface,DI.其作用与对象服务接口类似，但是领域接口针对领域而已，它与特定领域相关(垂直定向)。

(3).应用程序接口：Application Interface,AI.是专门为特定应用程序而开发的，并不是对象管理组所制定的标准。但是如果某些应用程序的接口出现在许多不同的应用程序中，那么这些应用程序接口应作为其他接口种类中的一类成为接口标准化的备选项。



5.CORBA常用术语：

(1).ORB：Object Request Broker,对象请求代理，在一个面向对象的分布式环境中，ORB可以为应用程序、服务器、网络设施之间分发消息提供关键通信设施。可以将ORB想象成一组软件总线，它提供了一个公用消息传递接口，通过这个接口，不同类型的对象可以以对等层策略进行通信。ORB是CORBA的核心组件，提供了识别和定位对象、处理连接管理、传送数据和请求通信的框架结构。

(2).CORBA对象：CORBA Object,是一个“虚拟”的实体，可以有对象请求代理(ORB)定位，并且可以被客户程序请求调用。

(3).目标对象：Target Object,在一个CORBA请求调用的上下文中，目标对象是指这个请求目标的CORBA对象。CORBA对象模型是一个单调度模型(single-dispatching model)，即一个请求的目标对象只能由这个请求的对象引用来确定。

(4).客户程序：Client,是一个实体，由它来向CORBA对象发出调用请求。

(5).服务程序：Server,是一个拥有一个或多个CORBA对象的应用程序，用于处理客户程序请求。

(6).请求：Request,是一个由客户程序所提出的CORBA对象的调用操作。请求从一个客户机传给服务器中的目标对象，如果这个请求要求一个CORBA对象作为响应，目标对象负责返回结果。

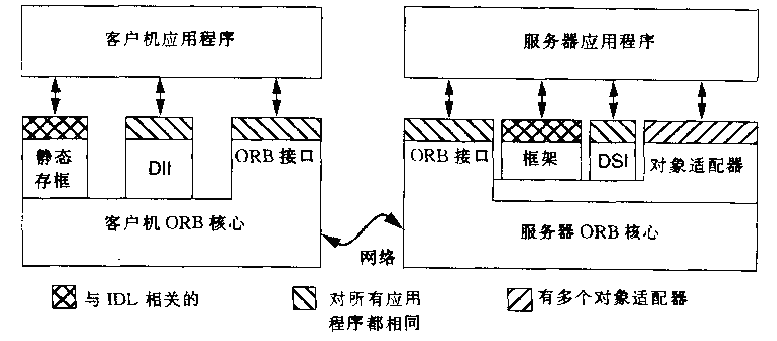
(7).伺服程序：Servant,是一个编程语言实体，用来实现一个或多个CORBA对象。伺服程序也称为具体化的CORBA对象，伺服程序存在于服务器应用程序上下文中，是一个特定类的对象实例。CORBA只是一个规范，CORBA使用对象定义语言(Interface Definition Language,IDL)定义分布式程序的对象，以及对象之间的交互操作，具体的实现由不同的编程语言如C++或[**Java**](http://lib.csdn.net/base/java)来提供，伺服程序就是这些具体编程语言中的程序对象。

(8).可互操作的对象引用：Interoperable Object Reference,IOR.存储几乎所有ORB间协议信息，用于建立客户机和目标对象之间的通信，为ORB的互操作提供标准化的对象引用格式。每个IOR指定一个或多个所支持的协议，对于每个协议，IOR包括那个协议所专有的信息。对于IIOP，每个IOR包括一个主机名，TCP/IP端口号和一个对象密钥，密钥根据所给出的主机名和端口组合来识别目标对象。

一个IOR主要有三个部分组成：仓库ID，终点信息和对象密钥。

个人理解IOR类似于[**数据库**](http://lib.csdn.net/base/mysql)连接信息或者JNDI连接信息对象等，用于传输对象之间的操作信息。

6.CORBA体系：



(1).客户端调用静态存根(static stubs)向服务器发出请求，存根(stubs)是代理对象支持的客户端程序。

(2).服务器端调用静态框架(static skeleton)处理客户端请求，框架(skeleton)是服务器端程序。

7.CORBA的接口定义语言：

Interface Definition Lanuage,IDL.IDL是CORBA的基本抽象，它从实现中分离出对象接口，用于描述客户机和服务器程序之间交互操作用到的数据类型和对象接口。因为IDL只描述接口，不描述实现，它是一个纯说明性语言，因此IDL无法编写可执行的语句，也无法解说对象的状态，IDL定义把焦点集中在对象接口、其他接口所支持的操作和操作时可能引发的异常上。IDL与编程语言无关，这是CORBA支持异构系统和独立开发应用程序集成的关键。

IDL定义由一个IDL编译器编译成一个具体的实现语言如C++等，IDL编译器将IDL中这些与编程语言无关的对象和接口定义翻译成特定编程语言的类型定义和API，开发者使用这些编译处理的类型和API来提供应用程序功能和与ORB交互。

IDL编译成特定编程语言的转换[**算法**](http://lib.csdn.net/base/datastructure)由CORBA来确定，并成为语言映射(Language Mapping)。

8.对象管理组的生命周期服务(Life Cycle Service)：包括对象的创建、拷贝、移动和撤销，以及使用回收(Evictor)模式实现对大对象内存消耗限制和无用存储单元回收策略。

对象管理器的生命周期服务类似于EJB中EJB对象的创建、钝化、激活以及被Java虚拟机垃圾回收等。

9.CORBA对象请求代理(ORB)间协议：

(1).GIOP：General Inter ORB Protocol,通用ORB间协议，是一类抽象的协议，指定了转换语法和消息格式的标准集，运行独立开发的ORB可以在任何一个面向连接的传递中进行通信。

(2).IIOP：Internet Inter-ORB Protocol,因特网ORB间协议，指定GIOP如何在TCP/IP上实现。

10.GIOP消息格式：

(1).Request：

从客户机发送到服务器，请求调用服务器一个操作或读写一个属性。

(2).Reply：

从服务器发送到客户机，只是为了响应客户机的请求，包含一个操作调用的结果。

(3).CancelRequest；

客户机通知服务器它已对一个操作的结果失去了兴趣。

注意：当操作执行时，一个CancelRequest将不会中止一个操作，相反，它只是通知服务器当操作完成时，不再需要向客户机传递应答。

(4).LocateRequest：

客户机使用该请求从一个对象中获得当前的地址信息。

(5).LocateReply：

服务器使用该响应消息对客户机的LocateRequest进行应答，LocateRequest和LocateReply可以降低定位对象的开销。

(6).CloseConnection：

服务器使用该消息通知客户机该服务器程序准备关闭连接。

(7).MessageError：

客户机/服务器都可以发送该消息，用于响应任何错误的GIOP消息。

(8).Fragment：

客户机/服务器都可以发送的消息，用于决定是否已存储片方式发送消息。

11.IOR结构：

CORBA使用可互用的对象引用(IOR)作为识别一个对象的通用手段，IOR包含一个对象的接口类型和一个/多个的协议配置文件。每个配置文件包含客户机使用一个特定协议发送一个请求所需的信息。单个IOR可能同时包含几个协议的寻址信息，使得单个CORBA对象可以通过不同的传输进行访问。

12．CORBA请求调用步骤:

客户机通过发送消息来调用CORBA操作，当客户机调用CORBA操作时，ORB完成以下操作流程：

(1).定位目标对象。

(2).调用服务器应用程序。

(3).传递调用所需的参数。

(4).必要时，激活调用目标对象的伺服程序。

(5).等待请求结束。

(6).如果调用成功，返回out/inout参数和将返回值传给客户机。

(7).如果调用失败，返回一个异常给客户机。

**CORBA的事件机制以及对象适配器价绍**

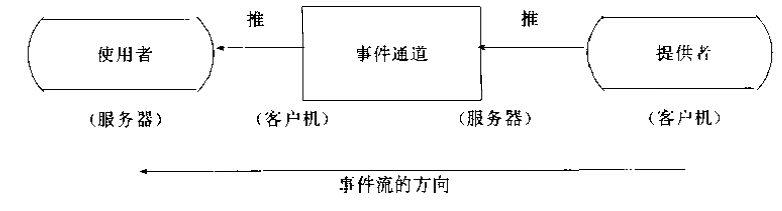
1.对象管理组事件服务(Event Service)：

CORBA通过使用事件服务实现异步调用。事件模型中事件提供者生成事件，事件使用者接收事件，事件提供者和事件使用者都连接在一个事件通道上。事件通道将事件从提供者传送到使用者，且不需要提供者事先了解使用者情况。

对象管理组事件服务提供两种事件发送模型：推模型(push model)和拉模型(pull model)。

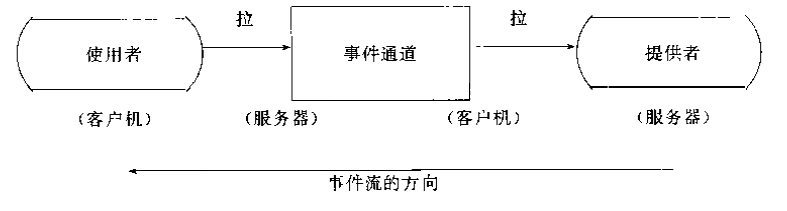
(1).推模型：

事件提供者将事件推送给事件通道，事件通道将事件推送给事件使用者。



(2).拉模型：

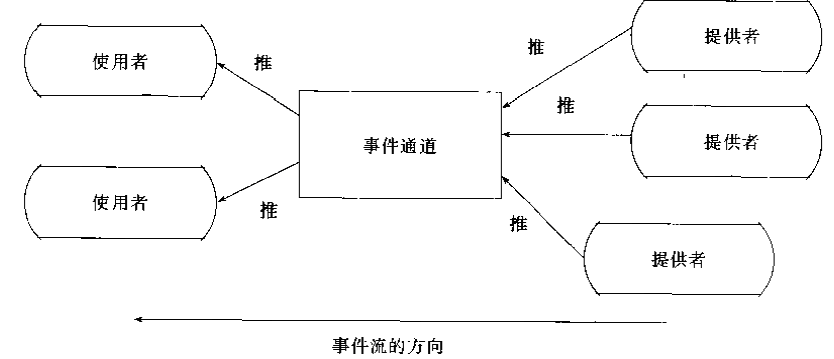
事件使用者从事件通道拉回事件，而事件通道又从事件提供者拉回事件。



2.事件通道支持的事件发送模型：

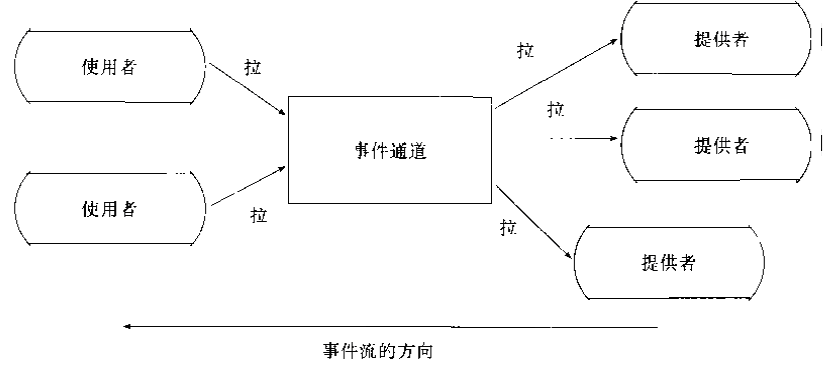
(1).经典的推模型：

事件提供者将事件推向事件通道，事件通道又将这些事件再推向所有注册的事件使用者。事件提供者是事件主动发起者，而事件使用者只是被动的等待接收事件，事件通道扮演通报者角色。经典的推模型是最常用的事件发送模型。



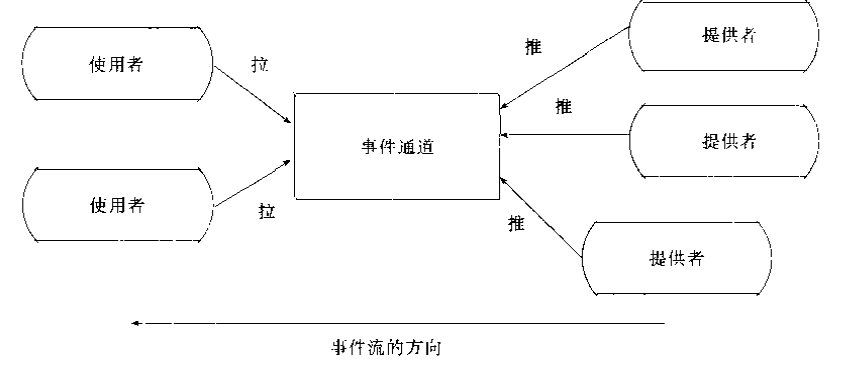
(2).经典的拉模型：

事件使用者将事件从事件通道中拉出来，然后事件通道又将事件从事件提供者中拉出来。事件使用者是事件的主动发起者，事件提供者被动等待事件被拉走，事件通道起着中介作用。



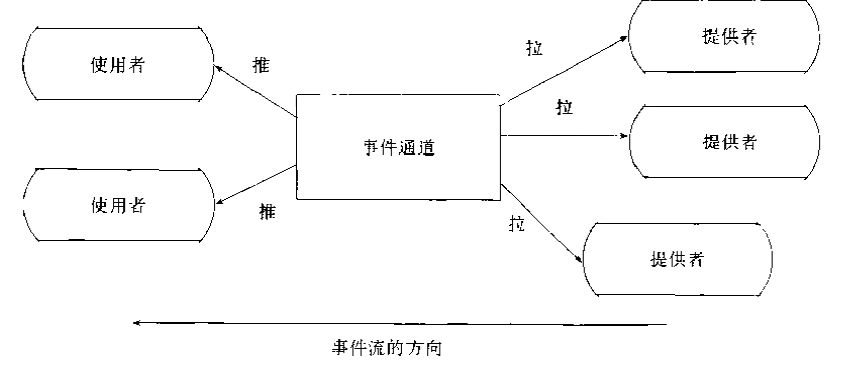
(3).混合推/拉模型：

事件提供者将事件推向事件通道，事件使用者又把事件从事件通道中拉出来，在此模型中事件提供者和事件使用者都是主动方，事件通道起着队列的作用，事件通道只是将事件提供者推入的事件数据存储起来，直至事件使用者将事件数据从事件通道中拉走。



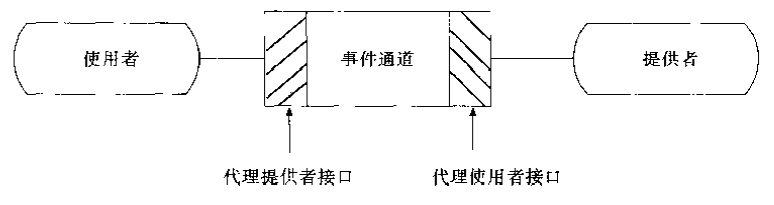
(4).混合拉/推模型：

事件通道将事件从事件提供者拉出，然后将事件推送给事件使用者，在此模型中事件提供者和事件使用者都是被动的，事件通道起着一个只能代理的作用。

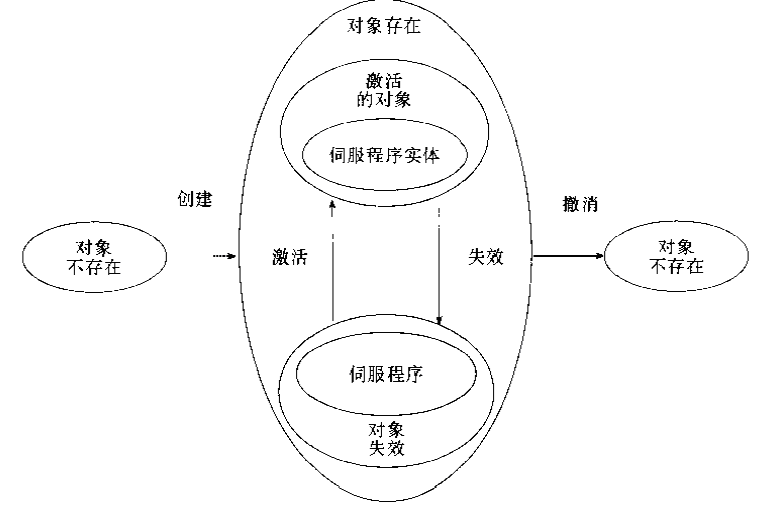


3.事件通道：

事件通道既是事件提供者，又是事件使用者。事件通道通过代理接口代表实际的事件提供者和事件使用者。



4.CORBA对象状态和伺服程序声明周期：



5.对象适配器POA：

POA：Portable Object Adapter，可移植对象适配器。在CORBA中，对象适配器作为伺服程序和对象请求代理(ORB)之间的纽带，适配器将一个对象接口配置给调用程序所需要的不同接口，即一个对象适配器是一个插入式对象，它用来作为代理，允许调用程序在不知道对象实际接口情况下调用一个对象的请求。

POA是一个对象管理器，类似于[**Java**](http://lib.csdn.net/base/java)中的EJB容器，或者[**spring**](http://lib.csdn.net/base/javaee)的IoC容器，主要用于负责创建、激活、定位以及回收对象等操作。

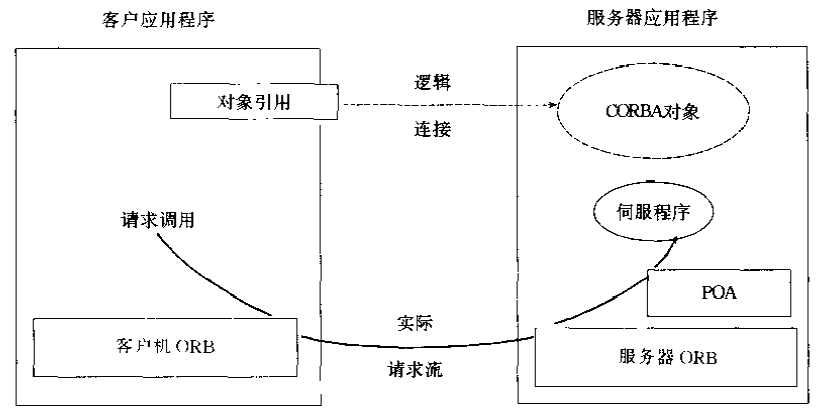
POA主要作用为：

(1).创建对象引用。

(2).创建对象标识符。

(3).创建并注册伺服程序，并伺服程序通过POA来实例化CORBA对象。

6.CORBA通过POA请求调用流程：



7.POA策略：

POA规范的关键特性是一个应用程序可以包含多个POA实例，每一个POA实例代表具有相似性能的一组对象。这些性能通过POA创建时所指定的POA策略来控制。所有的服务器应用程序至少有一个POA，也就是Root POA，它具有标准的策略集。

(1).生存期范围策略：

a.TRANSIENT：暂态策略， 默认标准的生存期策略。

b.PERSISTENT：持久态策略。

(2).对象标识符策略：

a.SYSTEM\_ID：POA在创建对象标识符时，自动产生唯一的ID，POA默认策略。

b.USER\_ID：POA在创建对象标识符时，通过产生异常来防止两个相同的ID。

(3).对象与伺服程序之间映射策略：

a.UNIQUE\_ID：POA对每个对象标识符必须映射到一个不同的伺服程序，POA默认策略。

b.MULTIPLE\_ID：POA对多个对象标识符可以映射到同一个伺服程序。

(4).隐式激活策略：

a.IMPLICIT\_ACTIVATION：允许POA隐式创建和激活CORBA对象。

b.NO­\_IMPLICIT\_ACTIVATION：只允许POA显示创建和激活CORBA对象，POA默认策略。

(5).请求与伺服程序之间匹配策略：

a.USE\_ACTIVE\_OBJECT\_MAP\_ONLY：POA为请求使用已有的伺服程序，POA默认策略。

b.USE\_DEFAULT\_SERVANT：POA为请求使用默认的伺服程序。

c.USE\_SERVANT\_MANAGER：POA通过调用伺服程序管理器确定创建一个新的伺服程序或复用一个已存在的伺服程序。

(6).对象标识符到伺服程序关联策略：

a.RETAIN：每次接到一个请求时，POA期望应用程序提供目标对象标识符作为查找伺服程序的索引，POA默认策略。

b.NON\_RETAIN：POA不需要应用提供目标对象标识符，而是通过调用应用程序提供的伺服程序管理器或应用程序提供的默认伺服程序管理器处理请求。

(7).请求线程策略：

a.ORB\_CTRL\_MODEL：ORB受控模型，允许多个并且的请求由多线程来处理，POA默认策略。

b.SINGLE THREAD\_MODEL：单线程模型，POA使用单线程顺序处理请求。

**CORBA**

Corba 概述

1、简介

计算机网络是典型的异构体系。

造成异构的原因：

>网络技术随时间不断在改进。

>任何一种计算机、操作系统、网络平台的组合都是为了能在一个网络中使某部分的性能达到最好。

>网络内的多样化使得它具有更大的回转余地。

解决异构的分布式系统的应用程序的两条原则：

>寻找独立于平台的模型和抽象。

>在不牺牲太多性能的情况下，经可能隐藏底层的复杂细节。

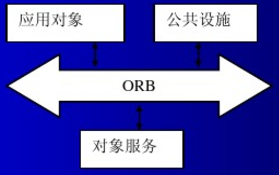
2、OMG对象管理组织

OMG用两个相关的模型来描述如何以与平台无关的方式来指定分布式对象及它们之间的交互。

对象模型(Object Model)用来定义在一个异构环境中，如何描述分布式对象接口。

引用模型(Reference Model)用来说明对象间如何交互。

OMG对象管理体系结构如图1：



<1>公共设施

横向设施：是指在通用领域内定义的对象。

纵向设施：是指在专用领域内定义的对象。

<2>对象服务

命名服务、事件服务、事件处理服务、交易服务、生命周期服务、安全服务、通知服务。

<3>对象请求代理(ORB)

它是CORBA的基础，是在分布式环境下，CORBA应用所使用的基于对象模型的软件总线。

笔记2

TAO CORBA 编程基本概念

一 、CORBA 及TAO简介

CORBA是一个为简化跨平台应用而提出的规范。

独立于网络协议、编程语言、硬件平台。

支持异构的分布式计算环境和不同编程语言间的对象重用。

CORBA可以作为不同平台应用间信息传递的中间件。

CORBA通过引入经过充分验证的有效的框架结构和通信手段，最大限度简化了网络通信相关应用的设计和开发，使得我们专注于业务逻辑的实现，无需关心通信细节。

CORBA也叫作软总线，表名它作为数据通信的基本特性。

二、CORBA实现。

有商用的ORBacus、VisiBroker，也有一些优秀的开源实现，如：TAO、omniORB、MICO等。

这些实现都遵守相同的规范，所以接口基本一致，熟练应用一种CORBA实现后，使用其他基本没什么问题。

**TAO**（The ACE ORB）是美国华盛顿大学的Douglas C. Schmidt教授领导开发的一个实时CORBA平台，它是一个免费的开放源码项目，用C++语言开发，符合CORBA2.6规范。

支持语言： C++

支持平台：Win32、 常见的Uinx/Linux,实时操作系统VxWorks等。这里TAO支持的平台最多。

支持服务：Naming、Event、Notification、Security、Time、Scheduling、Logging、Lifecycle、Trading、Concurrency、Lifecycle、A/V Streaming、Load balancing等。

本系列文章将以当前最新的**ACE-6.3+TAO-2.3+CIAO 1.3**为例，简要介绍如何应用TAO进行CORBA C++编程，其中部分内容（尤其是编译器配置相关的内容）是Windows平台特有的，但其它大多数信息在各平台上都是相同或者类似的。

三、基本概念

本文不打算深入介绍CORBA相关的理论基础（已有很多书籍、文章讨论了这些内容），但在进入下一主题前，为了便于后续问题的讨论，这里简要介绍一下CORBA的基本原理，并对几个重要的基本概念进行解释，以便为没有相关知识的朋友扫清障碍。下图3-1是CORBA的基本模型：

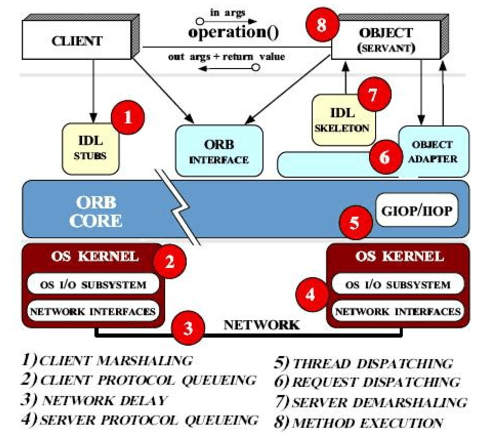


图 3-1

图中除上面那根灰线以上的两个部分需要我们直接介入外，其他部分基本可以由IDL编译器自动生成或由CORBA实现为我们完成。借助CORBA提供的中间件机制，我们在进行方法调用时，无需考虑被调用对象所在的具体位置，而是像调用一个本地模块的方法一样”直接”与其通信(因此，有些应用中，CORBA也是被用于本地进程间通信, 以降低各模块间的耦合度，同时也便于系统结构的调整)。

上图包含的部分：

* Client ：即客户程序，Client通过调用服务程序提供的接口实现特定功能。
* Object ：CORBA Object是一个语言无关的虚拟的实体，它可以由ORB定位，并可被客户程序请求调用，它最终由某种语言来具体实现，从而成为应用程序的一部分。
* Servant ：即实例化的Object，Servant是组成服务程序，具体为Client提供服务的某个运行中（且被注册到OA）的C++/Java（或其他语言）对象。Object与Servant是一对多的关系，一个Object可以被实例化多次，注册到一个或多个OA中为一个或多个Client提供服务，各Servant对应不同的IOR（多个IOR具有相同的Type信息，但唯一标识不同），并且，每个Servant都可以由IOR唯一确定，对于持久Object，该IOR在两次启动同一系统时不会发生变化，除非服务程序所在主机IP或配置信息等发生改变，而对于非持久Object，该IOR在两次启动同一服务程序时，总是变化的（因为其中往往包含一个时间戳）。
* Stub 即桩，或称存根，Stub负责“假扮”Servant为Client提供一个本地化的接口，并负责与ORB进行交互，完成调用请求的列集（marshaling，即打包）与散（unmarshaling，即解包）。列集的主要作用是按双方ORB可以识别的方式对需要传递数据进行格式化，而散集则将收到的数据中还原成上层应用可以理解的内存数据形式，对于远端指针，散集时ORB还负责分配相应的内存空间，并将指针的内容复制到本地内存空间以为本地应用提供一个远端指针的镜像，从而保证本地应用对远端指针的“透明”访问。
* Skeleton 与Stub对应，负责调用参数在Server端的解包和调用返回的打包。
* ORB Core 屏蔽底层通信细节的中间件，所有远程访问请求在Client和Server端的ORB间被传递，使得上层应用无需关注平台、语言、网络结构等方面的差异。
* Object Adapter（OA） OA是ORB与Servant之间的桥梁，主要负责：

1、产生对象引用；

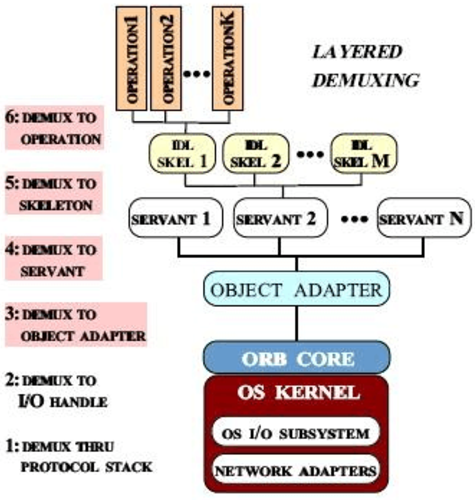
2、激活Servant或反之；

3、将请求分解后传递给对应的Servant；

4、与Skeleton配合完成对Servant提供方法的调用。

OA与Servant也是一对多的关系。

下面的图形进一步说明了ORB、OA、Skeleton、Servant之间的关系：



CORBA Programming with TAO - 2.Compile Source Code（编译ACE/TAO/CIAO）

摘要：

简要介绍如何在Windows平台上编译ACE/TAO/CIAO，及编译过程中需注意的问题。

正文：

随着ACE/TAO/CIAO研究者研究的不断深入，整个平台变得越来越庞大，涉及的工程及目录数目也越来越多，下面简要介绍一下如何在Windows平台上编译ACE/TAO/CIAO，其过程在其它平台上是类似的。

一、目录结构

在开始之前，先简要介绍一下整个产品的目录结构。ACE（Adaptive Communication Environment）是整个ACE/TAO/CIAO体系的基础，由操作系统适配层、对并发/进程间通信/虚拟内存管理等的OO封装、ACE框架、网络服务组件等几个部分组成；而TAO（The ACE ORB）则是在ACE基础上建立起来的一个CORBA ORB实现；CIAO（Component Integrated ACE ORB）则是在TAO基础上的一个CORBA Component Model实现，同时CIAO也是目前相对比较成熟的一个CCM实现。 正是由于三者之间的上述关系，在目录结构上整个源码包也采用了类似下面的结构：

 ACE\_ROOT

├─ace

├─docs

├─...

├─TAO

│  ├─tao

│  ├─docs

│  ├─orbsvcs（TAO实现的众多CORBA服务被放置在这个目录下）

│  ├─...

│  ├─CIAO

│  │  ├─ciao

│  │  ├─docs

│  │  ├─DAnCE

│  │  ├─RACE

│  │  ├─...

三个产品的帮助文档，可以在相应节点下的docs目录中找到，其中除了对相应产品特定模块或服务的说明外，还包括一些不错的Tutorial，对于我们理解相应产品很有帮助（ACE/TAO/CIAO的较完整的文档是收费的，这些随源码发布的文档虽然少，但仍颇为有用）。

二、关于编译器

可以使用VC6/7.1/8来编译ACE/TAO/CIAO，不推荐使用VC7，但由于VC6的Project Settings不支持环境变量，会对我们使用CIAO带来一些麻烦（如需手动修改自动生成的工程文件），因此，笔者推荐读者使用VC8，即Visual C++ 2005来进行编译（注：现在VC++ 2005 Express可以从MS的网站上免费获得，但若安装该版本的话，需按照http://lab.msdn.microsoft.com/express/visualc/usingpsdk/default.aspx的指示安装SDK并进行相关设置）。VC8在调试器的设计上有了很大改进，同时对C++标准的支持也有了很大进步（但也变得非常耗资源），因此，下面的讨论将基于ACE-5.5+TAO-1.5+CIAO-0.5 + VC2005 Express，不保证在其它ACE或编译器版本下完全一致。

三、基本设置

首先，设置好如下环境变量：

ACE\_ROOT

TAO\_ROOT

CIAO\_ROOT

分别指向ACE根目录，TAO根目录（即%ACE\_ROOT%/TAO），CIAO根目录（即%TAO\_ROOT%/CIAO）。 在%ACE\_ROOT%/ace目录下添加一个文件config.h，并在其中添加如下内容：

/\* -\*- C++ -\*- \*/

//============================================================================= /\*\*

\* @file config.h \* \*/

//============================================================================= #define ACE\_HAS\_STANDARD\_CPP\_LIBRARY 1

#include "ace\config-win32.h"

若是在其它平台上进行编译，还需要添加文件：

$ACE\_ROOT/include/makeinclude/platform\_macros.GNU 具体方法见ACE Installation Guide。

此外，为了便于今后使用ACE/TAO/CIAO，应当将 %ACE\_ROOT%/bin（部分通用工具的输出目录）

%ACE\_ROOT%/lib（所有.lib、.dll文件的输出目录） 添加到PATH环境变量中。

并且，需打开VC8，选择Tools -> Options -> Projects and Solutions -> VC++ Directories，添加如下编译器环境变量：

Executable files: $(ACE\_ROOT)\bin $(CIAO\_ROOT)\bin

Include files: $(ACE\_ROOT) $(TAO\_ROOT)

$(TAO\_ROOT)\orbsvcs $(CIAO\_ROOT) $(CIAO\_ROOT)\ciao

Library files: $(ACE\_ROOT)\lib

四、开始编译

现在可以开始编译了，但是，在笔者的Win2000中编译ACE5.5/TAO1.5/CIAO0.5时遇到的第一个问题就是：源码包中的VC Workspace/Solution打开之后一片空白，一个工程也没有。（上一版本没有这个问题。最终通过比较源码包中的.sln文件和通过下面的方法得到的.sln文件，发现其实是由于文件是Unix格式造成的） 几经周折，终于想到了一个办法：自己运行mwc.pl重新生成工程文件。 要运行mwc.pl自然需要安装Perl，安装过程略。

下面简单说说mwc的工作原理：

mwc.pl根据参数中指定的.mwc文件的内容到相应目录下查找.mpc文件，在没有指定.mwc文件或.mwc文件中没有指定查找目录的情况下会尝试进入各子目录，查找.mpc文件（相当于一个makefile文件），按照.mpc文件生成工程信息，若连一个.mpc文件也找不到，就将当前目录下所有可识别的文件当作源文件，产生一个默认的工程文件。最终，mwc.pl将所有工程文件合并到一个独立的Solution/Workspace。 .mwc文件的结构其实很简单，下面是%TAO\_ROOT%/TAO\_ACE.mwc的全部内容：

// -\*- MPC -\*-

// TAO\_ACE.mwc,v 1.1 2006/02/16 09:20:48 jwillemsen Exp

workspace {

../ace

../apps/gperf/src

../ACEXML/common

../ACEXML/parser/parser

../ACEXML/apps/svcconf

../Kokyu/Kokyu.mpc

../protocols

tao

TAO\_IDL

utils

orbsvcs

exclude {

    orbsvcs/tests

orbsvcs/performance-tests

orbsvcs/examples

}

}

其作用就是告诉mwc.pl到哪些目录下去查找.mpc文件（若没有指定，则表示应检查所有子目录），而其中的exclude则表示应略过那些目录，如果你愿意，完全可以自己编写.mwc文件（不过，工程具体的描述文件.mpc则比较复杂，一言难尽，想一探究竟的话，最好还是看看MPC的说明文档

http://www.cs.wustl.edu/~schmidt/ACE\_wrappers/MPC/USAGE或http://downloads.ociweb.com/MPC/MakeProjectCreator.pdf）。

对mwc.pl的工作原理有了基本的了解后，就可以运行mwc.pl生成工程文件了。执行mwc.pl的办法很简单，如要生成使用CIAO的基本工程，可以在%CIAO\_ROOT%下运行：

mwc.pl -type vc8 CIAO\_TAO.mwc

上述命令将产生一个名为CIAO\_TAO.sln的Solution文件，有了.sln文件，就可以正式进行编译了。

应用TAO进行CORBA应用开发，下面的几个工程是必须的：

* TAO\_IDL\_EXE（即tao\_idl，也就是我们的idl编译器，自然少不了，并且后续工程也需要用到该编译器。该工程依赖于ACE、TAO\_IDL\_BE、TAO\_IDL\_FE、gperf等多个工程，编译TAO\_IDL\_EXE是会自动先编译这几个工程）
* PortableServer（即POA库，编写CORBA应用程序哪能离得开POA，该工程依赖于ACE、TAO、CodeSet、AnyTypeCode等工程）
* Naming\_Service（即命名服务，这是对于我们开发CORBA应用相当有用的一个基本服务，该工程依赖的其它工程就更多了，就不再一一列举了）
* 而要使用CIAO，则还必须编译%CIAO\_ROOT%/DAnCE下的多个工程，这些工程包括：

ExecutionManager

NodeManager

Plan\_Launcher

NodeApplication

这几个工程是部署与配置引擎（Deployment And Configuration Engine，DAnCE）的一部分，负责处理面向组件的应用的部署与配置。

由于工程之间存在的复杂依赖关系，你只需编译最后这三个工程，就可以自动完成约30个工程的编译工作。编译完以上工程后，你的%ACE\_ROOT%/lib和%ACE\_ROOT%/bin下会多出来一些.lib、.dll和.exe文件。作为一个ORB服务，Name\_Service.exe被放在其对应的子目录%TAO\_ROOT%/orbsvcs/Name\_Service下。

  注：

1. 为了编译CIAO\_XML\_Utils，需要安装Xerces-C++ 2.7.0，但http://xml.apache.org/xerces-c/提供的下载采用VC6编译，该版本不能用于VC7及以上版本，如果你用的不是VC6，则需要下载源码重新编译。如果你用的是VC++ 2005 Express，由于没有ATL及MFC，在编译时会遇到一些麻烦，可以在SDK Dir\include下自行添加一个WINRES.h，在其中添加如下代码（两行，空行是必须的）：

 #include <winresrc.h>

2、如果要试用CIAO，还需用到cidl编译器cidlc，该工具可以从http://www.dre.vanderbilt.edu/cidlc/binary/下载。

参考：

1.      Building and Installing ACE and Its Auxiliary Libraries and Services.

http://www.cs.wustl.edu/~schmidt/ACE\_wrappers/ACE-INSTALL.html

2.      Building and Installing CIAO.

http://www.cs.wustl.edu/~schmidt/ACE\_wrappers/TAO/CIAO/CIAO-INSTALL.html 3.      The Makefile, Project, and Workspace Creator (MPC).

http://downloads.ociweb.com/MPC/MakeProjectCreator.pdf

CORBA Programming with TAO - 3.IDL Data Type（数据类型与Mapping）

**摘要：**

简要介绍TAO支持的IDL数据类型及其C++ Mapping关系。

一、简单基本数据类型（Simple Basic Types）

TAO支持以下简单基本数据类型（%TAO\_ROOT%/tao/Basic\_Types.h）：

 IDL  C++

boolean  CORBA::Boolean

char  CORBA::Char

octet  CORBA::Octet

short  CORBA::Short

unsigned short  CORBA::UShort

long  CORBA::Long

unsigned long  CORBA::ULong

long long  CORBA::LongLong

unsigned long long  CORBA::ULongLong

wchar  CORBA::WChar

float  CORBA::Float

double  CORBA::Double

long double  CORBA::LongDouble

以上各简单基本类型对应的C++类型只是对应平台上基本类型的typedef（Java虚拟机之所以能够在各平台上保证统一的数据长度，其实现原理也是如此），但在编写应用程序时，为了保证程序的可移植性，应该总是使用CORBA命名空间中的类型标识。此外，需要注意：在上面的所有类型中，没有我们熟悉的C++基本类型byte（被Octet取代）、int（被Long取代）。

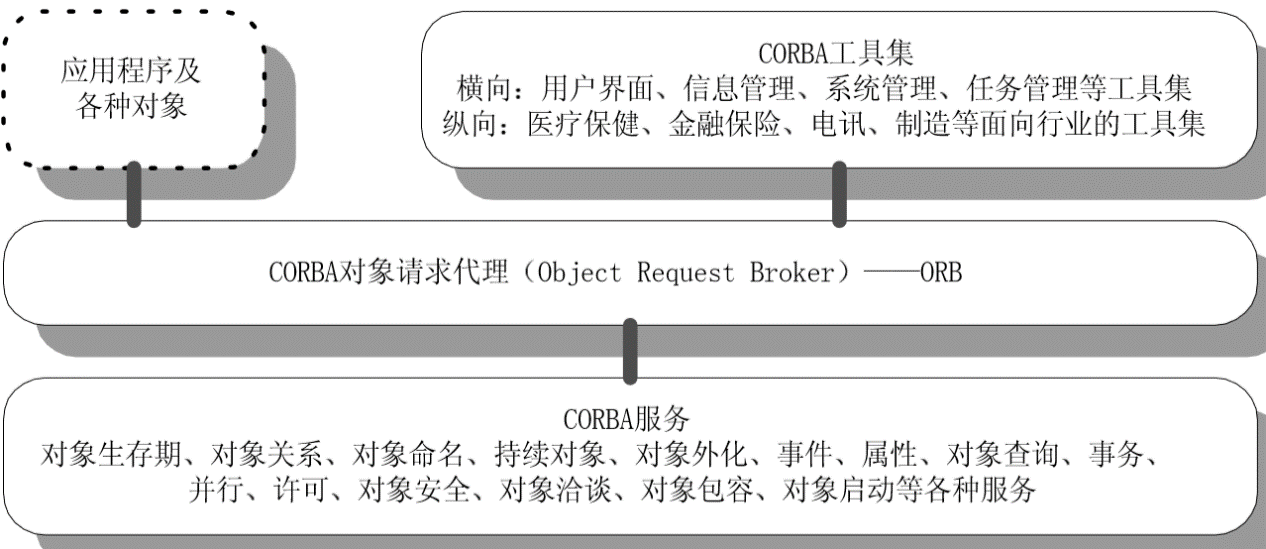
摘要

适配器对象是公共对象请求代理体系结构中一个主要组成本分。适配器对象帮助对象请求代理向服务器实现传递客户端请求。

适配器对象提供如下服务：

一、Corba对象请求代理——ORB

下图是corba的整体框架，开发只需要关注虚线部分，其余由corba提供。



Orb 简称为对象请求代理。Orb是联结应用程序、各种对象、corba服务、corba工具集的核心。同样也造就了orb对这些的分割，实现了corba分布式软件集成和即插即用的全部秘诀。

ORB主要实现一下分割：

* 对象方法、服务的“定义”与它们的“实现”之间的分割。通过接口定义语言OMG IDL，我们可以获得规范、通用的对象方法、服务定义；借助ORB，这些定义可以在任何编程语言、代码模块中真正实现。这种分割有助于进行具体软件编码互换、编程语言互换以及版本互换。
* 请求“客户”与响应“服务器”之间的分割。客户对其对象方法、服务的请求并不直接传递给被请求服务器，而是转交给ORB，由ORB监察服务器的位置状态，决定服务绑定的方式。这种分割有助于分布式对象进行跨平台、跨协议的“逻辑集成”，是一种“伪客户/服务器”方式。
* CORBA “基础设施开发者” 与CORBA “收益者” 的分割。真正完全实现CORBA的所有标准难度太大。作为一般CORTBA用户，我们都可以借助ORB产品来简化开发任务。ORB就是某些软件公司建造好的，进行CORBA软件开发的基础设施。

总结：ORB可以帮助corba对象相互“理解”对方，可以在corba对象间传递信息、请求，可以“管理”corba对象之间的分布与集成。

二、CORBA基本服务

CORBA基本服务为基于CORBA的分布式软件开发提供了对象级别的服务。在CORBA中，有16个基本服务。

* Lifecycle Service：对象生存期服务。提供与对象创建、移动、拷贝、释放等操作有关的服务。
* Relationship Service: 对象关系服务。提供表达对象之间相互关系的服务。相信软件开发人员都明白，在 “对象” 世界中 “乱搞关系” 绝对会引起轩然大波，导致系统混乱。
* Naming Service: 对象命名服务。为对象提供姓名、别名、引用的服务。
* Object Collection Service: 对象包容服务。为对象提供数组、队列、栈、集合等容器。
* Object Startup Service：对象启动服务。决定对象如何启动以及启动时的状态。
* Security Service: 对象安全服务。提供基于时间的各种业务服务。
* Secure Time Service：对象时间服务。时间服务允许对象获取当前时间、确定事件发生顺序、计算时间间隔、实现基于时间的各种业务。
* Trader Service：对象洽谈服务。如果你知道你需要的服务，但不知道那个是最合适的服务、那个是最经济的对象，就可以使用对象洽谈服务。
* Licensing Service：许可服务。许可服务允许用户只有在开发者认可的情况下才能使用对象，获取服务。
* Concurrency Service：并行服务。解决对象并行操作中资源冲突、资源共享、同步、死锁等问题的服务。
* Object Transaction Service：事务服务。提供事物处理服务：“要么统统提交，要么统统滚回”。这是电子商务对于“一手交钱，一手交货”的理解。
* Event Service：事件服务。事件服务允许对象注册、注销自己所关心的事件，并提供传递事件消息的信道。
* Object Properites Service：对象属性服务。为对象提供动态属性的服务。
* Object Query Service：对象查询服务。提供一种操作语言和服务，能够随意查询对象数据、属性、方法、分布等情况。
* Externalization Service：对象外化服务。提供一种类似 “流” 的机制，提取、恢复对象的状态。
* Persistent Object Service：持续对象服务。提供保持对象状态(数据)的服务。

以上服务是软件集成时，各种对象可能需要的公共服务。

三、CORBA工具集

CORBA工具集就是CORBA Facilities。它们为基于CORBA的分布式软件开发提供了 “应用” 级别的服务。

CORBA接口及接口定义语言OMG IDL

1. CORBA灵活的伪客户/服务器方式源自于IDL

对象请求代理ORB是联结CORBA对象、CORBA应用程序、CORBA基本服务以及

CORBA工具集的核心。

从面向对象的角度考虑问题，CORBA可以简化为如下表述：可集成的分布式对象(程序)和一个能够从逻辑上集成这些对象，被称作ORB的对象群，通过请求、响应的方式实现功能互联。如图2.1所示。

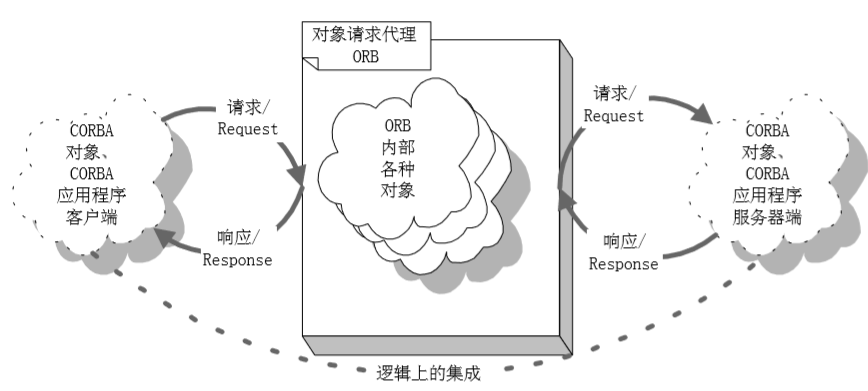


图2.1

分布式对象、程序之间的请求/响应方式就是传统的客户/服务器方式。但是，CORBA中的请求 “客户” 和响应 “服务器” 之间并没有直接进行 “通信” 。 CORBA客户向ORB发送请求，并从ORB接收响应；CORBA服务器接收来自ORB的请求、调度，并向ORB发送响应。实际上，这是一种ORB为中间件的伪客户/服务器方式。

CORBA的这种伪客户/服务器方式具有更大的灵活性，可以满足分布式软件开发下列现实需求：

1. CORBA中的接口

按照面向对象分析的方法，CORBA中至少应该存在三组对象(或者说 “类” )：CORBA

客户对象、ORB对象、**CORBA对象实现(或者说 “CORBA服务器对象”** )。它们的关系如图2.1所示。

根据第一章提到的CORBA的五个 “跨越” 目标，我们很容易对这三组对象提出一下要

求：

* 希望选择不同厂家提供的、或者不同语言编写的CORBA客户对象。
* 希望选择不同厂家提供的对象请求代理ORB
* 希望选择不同厂家提供的、或者不同语言编写的CORBA对象实现。

为了满足这三个要求，实现CORBA所期望的软件即插即用，我们发现，需要在以上

三组对象之间引入新的对象来处理用于用户选择不同产品而导致的集成障碍，如图2.2所示。

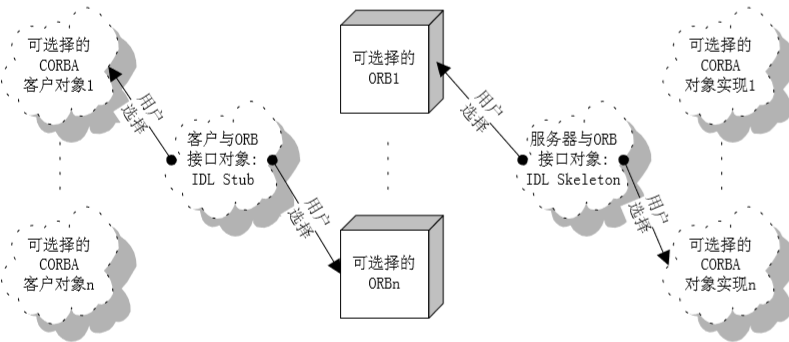


图2.2 新引入的对象——客户与ORB接口对象、服务器与ORB接口对象

**在CORBA标准中，客户与ORB接口对象被称为 “接口存根” —— IDL Stub； 服务器与ORB接口对象被称为 “接口框架” —— IDL Skeleton。**

**这样，CORBA客户对象与CORBA对象实现之间至少被三组对象隔离：接口存根、ORB、接口框架。**这种隔离为软件集成带来了巨大的灵活性，但同时需要CORBA客户对象、接口存根、ORB、接口框架、CORBA对象实现 各个部分能够相互理解各自的数据、结构和行为。这一要求可以通过接口技术以及CORBA接口定义语言—— OMG IDL来实现。

OMG IDL 不是编程语言，不能用来编写具体的操作算法，但是它可以用来定义对象能够进行的所有操作，包括全部输入，输出参数以及返回结果；甚至连可能产生的操作错误也不例外。

用OMG IDL 定义对象就得到了CORBA 接口。对于CORBA客户，这个接口意味着一种承若；如果客户向ORB发送符合接口规定的请求，就一定得到响应(出错、返回结果都是一种响应)；对于CORBA对象实现，这个接口意味着一种义务；软件开发人员必须实现接口规定的方法。

在这两种情况下都需要OMG IDL 能够和用户采用的具体编程语言建立映射关系。原因很简单：在客户端，用户需要从具体编程语言中按照接口规定激发相应的对象实现；在对象实现端，ORB应该能够根据接口规定激发软件开发人员采用集体编程语言编写的对象方法实现。综合起来考虑，就是 “每一个OMG IDL句法都应该在具体的corba编程语句中找到以个对应“ 。当然，具体编程语言中可以包含不为 OMG IDL 所支持、理解的句法。

CORBA接口对于接口存根和接口框架也相当重要。

1. CORBA接口框架——IDL Skeleton

如图2.3所示，接口框架对象链接ORB和CORBA对象实现 (服务器实现)。

这里，可以进行两次选择：选择某种编程语言编写的对象实现；选择某个厂商提供的

对象请求代理ORB。两种选择都取决于用户的嗜好、候选产品的性能、候选方案的可行性。但是，被选中的对象实现和ORB只能且必须通过接口框架联系起来。

接口框架可以从CORBA接口中自动获得，如图2.3所示。

对于任何一个CORBA接口，软件开发人员可以借助自动、半自动或者手工的方式，获得OMG IDL映射到自己所使用的具体编程语言中的对象声明，编写代码，完成对象实现。

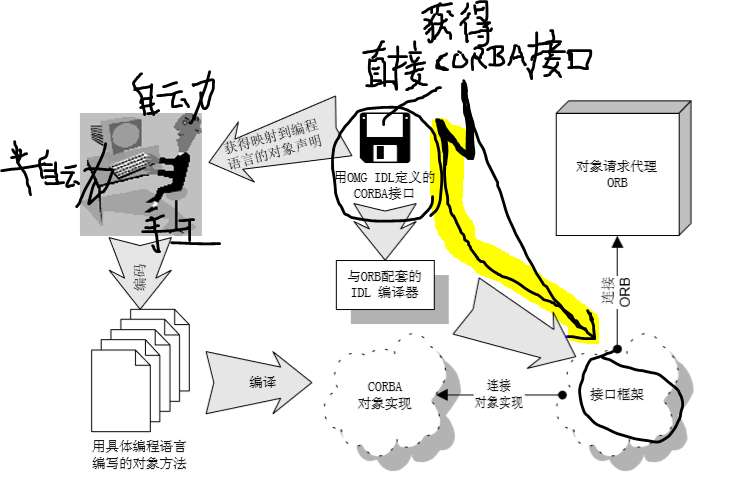


图2.3 CORBA接口经过编译自动生成接口框架

另一方面，我们可以使用IDL编译器进行编译，自动获得与CORBA接口相应的接口框架。接口框架中既包括与指定ORB相连接的函数调用代码，也包含与有关 “对象实现“ 相连接的函数调用代码。这样，接口框架就把 ”对象实现“ 与 ORB连接起来。

OMG IDL 以及 OMG IDL 与CORBA编程语言的映射关系都是标准的，如果指定编程语言，不同程序员对于同一个CORBA接口将会获得相同的函数声明。因此，我们希望任何IDL编译器在编译同一个CORBA接口时，对于同一种语言所产生的与 “对象实现“ 相连接的函数调用代码相同。这样，不论采用那个厂家的ORB，用同一种语言编写同一接口的对象实现时，只要完成具有相同声明的一套函数即可。正是由于这个原因，接口框架如何与 ”对象实现“ 连接是标准化的(主要是通过虚拟函数实现)。

接口框架如何与ORB连接欸没有被标准化，因为这样可以提高产品的灵活性。比如，我们买收音机时，仅仅要求耳机插孔外部符合通用标准，能够和各种立体声耳机连接；至于耳机插孔内部如何走线，大多数用户并不关心。

接口框架的这种特性导致用户在选择ORB产品时略有限制：一旦选定哪个厂家的ORB；就必须选择这个厂家的IDL编译器。ORB与IDL编译器成对出现；配套使用。

1. CORBA接口存根——IDL Stub

**接口存根用来连接CORBA客户和ORB**

**如图2.4 所示，当我们用IDL 编译器编译CORBA 接口获取 “接口框架“ 时，同时也得到了 ”接口存根“ 。不过，”接口框架“ 中主要是一些函数调用代码， ”接口存根“ 中主要是一些函数声明； ”接口存根“ 告诉客户本 ORB 能够提供什么服务，”接口框架“ 告诉ORB应该请何种性质的 ”对象实现“ 来完成指定的服务。**

**与接口框架类似，接口存根与ORB 如何连接也没有被标准化。**

**由于接口存根、接口框架与ORB如何连接取决于ORB生产厂家，我们可以把接口存根以及接口框架当作ORB对象群中两组专门的对象来对待。在以后的内容中我们就采用这种方式讨论问题。**

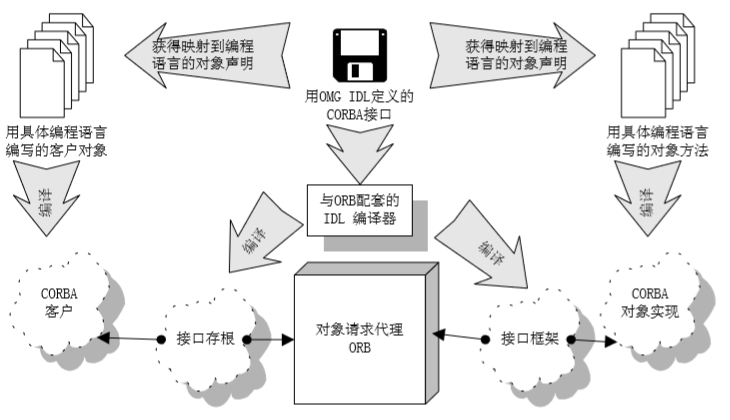


图 2.4 CORBA 接口自动生成 “接口存根“ 、”接口框架“

注意： 在CORBA中，接口是用OMG IDL 定义的。接口不能实现对象的方法，但规定了方法的名称、调用参数、返回值。接口的基本功能有两点：将对象方法标准化，并使得对象声明与实现分离；将真正的对象实现打包，使它能够被不同的语言激发、使用。这两个基本功能是ORB得以按伪客户/服务器方式集成软件的根本保障。在COM/DCOM中，也存在接口。这个接口是采用Microsoft IDL 定义的，它的目的也是将对象方法标准化，并使得对象声明与实现分离；将真正的对象实现打包，使它能够被不同的语言激发、使用。CORBA接口与COM/DCOM接口的不同之处在于两者采用了不同的接口定义语言，前者是OMG IDL ,后者是Mircosoft IDL。

1. OMG IDL 扼要

OMG IDL 是一种非常类似C++的语言。

1. OMG IDL 基本原则

一般银行都会有自己的ATM自动提款机。

ATM通过磁条解读器读取ATM账号，通过功能键选择操作内同，通过键盘输入密码或提款数目，通过专用打印机打印操作记录。

采用OMG IDL 我们可以得到ATM有关的一系列对象的接口定义如下。为了讨论方

便，这里做了一些简化处理。

例子2.1 简化后的ATM接口定义。

/\*这个用OMG IDL 语言编写的接口文件叫做ATM.IDL\*/



从例子2.1中，我们可有得到采用OMG IDL 编写CORBA接口的一些基本原则：

* 注释和C++完全相同