*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。

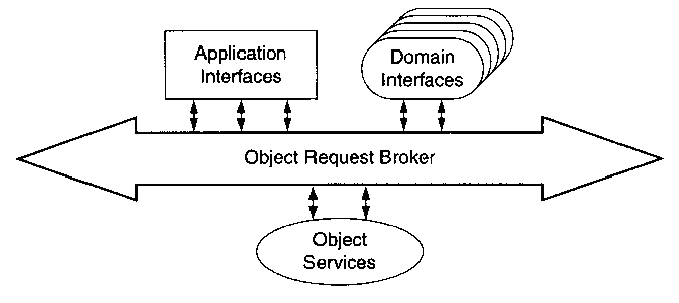
*(2).*引用模型*(Reference Model)*：用来说明对象之间如何交互。引用模型提供的接口种类，通常是按对象接口编组，所有接口种类有一个对象请求代理*(Object Request Broker,ORB)*按概念链接在一起。通常，一个*ORB*可以在客户机和对象之间进行通信，当请求发送给对象时，透明地激活那些没有运行的对象。

*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。

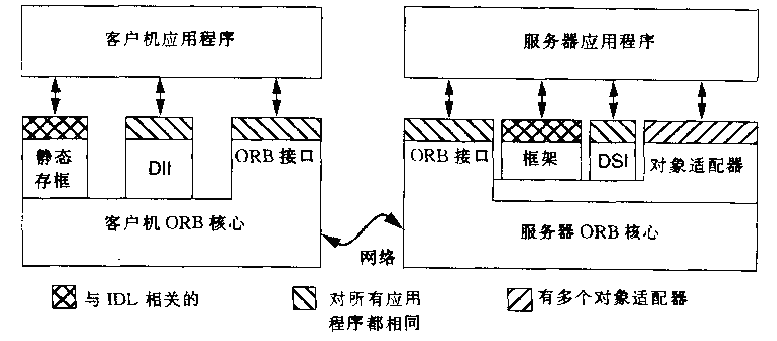
*(7).*伺服程序：*Servant,*是一个编程语言实体，用来实现一个或多个*CORBA*对象。伺服程序也称为具体化的*CORBA*对象，伺服程序存在于服务器应用程序上下文中，是一个特定类的对象实例。*CORBA*只是一个规范，*CORBA*使用对象定义语言*(Interface Definition Language,IDL)*定义分布式程序的对象，以及对象之间的交互操作，具体的实现由不同的编程语言如*C++*或*Java*来提供，伺服程序就是这些具体编程语言中的程序对象。

*(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).*客户端调用静态桩*(static**stubs)*向服务器发出请求，桩*(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。

BOA对象适配器:是服务器上管理对象引用和实现的主要机制,作用:给实现提供ORB服务,提供在对象的接口与该对象的服务器实现之间的绑定,生成并解释对象引用,并把对象引用映射到实现上,激活以及冻结实现,注册实现,隐含的通过框架激活实现方法.

**一、OMG**

  1.成立于1989年的非盈利性联盟；

  2. 以促进在分布式系统开发中面向对象技术的理论与实践的发展为目标；

  3.为使该组织所采纳的技术具有开放性，OMG所采用的方法是，针对某一领域发出RFP（Request For Proposal）,然后以各方提交的建议为基础，经过一系列的讨论和协商，产生最终的规范。

  4.OMG负责制订并维护一套规范（not software）：

        支持分布式、异类（heterogeneous）环境的软件开发项目。

         覆盖了从分析、设计到编码、部署、运行和管理的整个软件开发过程。

          这些规范是一种工业或行业标准

  5.OMG主要规范

    UML：Unified Modeling Language

   CORBA：Common Object Request Broker Architecture。

    MDA:   Model Driven Architecture

**由OMG制定的最关键的规范——对象管理结构（Object Management Architecture, OMA）和它的核心（也就是CORBA规范），提供了一个完整的体系结构。这个结构以足够的灵活性、丰富的形式适用了各类分布式系统。**

**二、OMA**

1.OMA描述了面向对象技术在分布式处理中的运用。它包括两部分：

         对象模型（Object Model）：定义如何描述分布式异质环境中的对象

          参考模型（Reference Model）：描述对象之间的交互(组件，接口，协议)

2.OMA对象模型

            在OMA对象模型中，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。如：

interface printer

{

attribute model;

void print(in string buffer);

};

3.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。

4.在OMA参考模型中，OMG定义了一条为对象所公用的通信总线，即ORB(Object Request Broker)。同时， OMG又定义了对象进出这一总线的接口。这包括：对象服务（Object Services） 、公共设施（common facilities）、应用接口（Application Interface）和域接口（Domain Interface）。

**三、ORB：Object request broker 对象请求代理**

1.概念：是OMA参考模型的核心，是基于分布式对象构建应用程序的基础设施，保证了在异构平台上对象的互操作性与可移植性。

2.作用：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。由ORB提供的通信机制负责完成查找请求的对象实现，让对象实现准备好接收请求，传递构成请求的数据。

3.ORB(Object Request Broker)是对象总线。

1）通过ORB，一个client可以透明的调用同一台机器上或网络上的一个server对象的方法。

2 )ORB解释该调用并负责查找一个实现该请求的对象，

3 )找到后，把参数传给该对象，调用它的方法，最后返回结果。

4) 客户方不需要了解服务对象的位置、通信方式、实现、激活或存储机制。

**四、对象服务：**

1.作用：提供所有应用程序都可能用到的通用服务的接口（基本服务，与具体的应用领域无关的接口）。

2.支持的服务有：名字服务、持久性服务、生命周期服务、事务处理服务、对象事件服务和安全服务等。

3.实质：通用的服务集合，将覆盖对象整个生存期的对象管理任务标准化，可保证各种应用程序均采用一致的风格管理对象

**五、CORBA体系结构**

1.C-ORB-A： “Common”+“ORB”+“Architecture”  通用“orb”体系结构

2.建立在OMG的对象模型基础之上，主要由三个关键部分组成

                作为分布式对象通信基础设施的对象请求代理ORB的体系结构；

                接口定义语言IDL的语法和语义以及到各种程序设计语言的映射；

               保证可互操作性的标准ORB间的通信协议GIOP/IIOP。

**六、IDL语言**

说明性语言，定义组件接口，不定义实现。

语法上可以看成C++的一个子集，规定组件的属性、所继承的父类、产生的异常、事件和各类数据类型。

编译器作用：将IDL映射到具体程序设计语言，产生桩代码和框架代码

调用请求经客户端桩传递给ORB，然后转发到服务端框架，最后到达真正要执行请求的对象实现实例

Run---->External Tools----->IDL Compiler,

f5刷新，**生成Stub与Skeleton**

IDL编译器作用是将IDL映射到具体程序设计语言，产生桩代码和框架代码

Visibroker for java提供的编译器idl2java将IDL映射到java语言，生成java语言的桩和框架语言

每个对象实例都有一个用于唯一标识自身的对象引用。客户程序利用对象引用指明调用的方向

表面上看，客户程序调用的是远程对象实现上的操作，实际被调用的代码是作为其代理的IDL桩

**七、RMI存根和框架的作用？**

◎**Stub(存根类)：**存在于客户端，为客户端编码远程命令并把他们发送到服务器，等待服务器

返回结果，stub再解码返回调用结果给客户端。

◎**Skeleton（框架）：**存在于服务器，是把远程命令解码，调用服务端的远程对象的方法，把

结果再编码发给stub。

**VisiBroker for Java的IDL编译器idl2java为每个接口自动生成7个文件：**

**（1）\_IntefaceNameStub.java**

Hello对象在客户端的桩代码，它实现了Hello接口。

程序员编写的客户程序代码通常不直接调用这个类中的方法。

VisiBroker for Java生成了另外的辅助类HelloHelper。

**（2）IntefaceName.java**

客户程序代码中，程序员通常使用的是操作接口的派生接口Hello(位于Hello.java)

**（3）IntefaceNameHelper.java**

IDL编译器为每一个用户自定义类型还生成一个辅助工具类。HelloHelper.java声明了HelloHelper类，

该类为Hello接口定义了许多实用功能和支持功能的静态方法（又称类方法）。

--从Any对象提取或向Any对象插入对象（extract和insert方法）；

--从输入／输出流读写对象（read和write方法）；

--获取对象的库标识和类型码（id和type方法）；

--绑定对象与类型转换操作（bind和narrow方法）等等。

编程时会用到该类中提供的方法。

**（4）IntefaceNameHolder.java**

AccountHolder.java声明的AccountHolder类为传递对象提供支持。

--IDL有三种参数传递方式：in、out和inout。

--in类型的参数以及返回结果与Java的参数传递方式与结果返回方式完全相同。

--out和inout两种类型的参数允许参数具有返回结果的能力，无法直接映射到

Java语言的参数传递机制，这时AccountHolder类为传递out和inout参数提供了一个托架（holder）。

\*In表明实际参数从客户程序传向对象实现

\*Out表明数据从实现对象传递给客户程序，并且对象实现无需从客户程序获取参数的初始值

\*Inout表明数据从客户程序传给对象实现，然后经对象实现加工后再返回给客户程序

**（5）IntefaceNameOperations.java**

(操作基调)定义了Hello接口中定义的所有常量和方法。

**（6）IntefaceNamePOA.java**

Hello对象的服务端框架代码，该类的功能：

-解包in类型的参数并将参数传递给对象实现。

-打包返回值与所有out类型的参数。

--打包（marshal）：指将特定程序设计语言描述的数据类型转换为CORBA的IIOP流格式。

--解包（unmarshal）：从IIOP流格式转换为依赖于具体程序设计语言的数据结构。

\*编写对象实现的最简单途径是继承这些POA类，即把它们作为对象实现的基类。

**（7）IntefaceNmaePOATie.java**

**创建并部署应用程序**

客户端：客户程序代码与IDL编译器自动生成的IDL桩代码一起编译

服务端：对象实现代码与服务程序代码与IDL框架代码一起编译

**八、Server端**

**1.编写对象实现和服务程序**

IDL文件只定义了对象的语法规格说明，必须编写这些对象的具体实现代码

对象实现可用各种语言实现，且与客户程序的语言无关

IDL到编程语言的映射规则

生成IDL框架代码和各种辅助性的java接口或类

编写对象实现代码时必须继承或使用其中的某些接口或类

服务程序

利用POA激活伺服对象供客户对象使用

通常是一个循环执行的进程，不断监听客户程序请求并为之服务

**2.编写服务程序**

通常程序员都会编写一个名为Server.java的服务程序，服务程序创建伺服对象供客户端使用。

**对象适配器：**对象适配器是联系对象实现与ORB本身的纽带。它的引入还大大减轻了ORB

的任务，从而简化了ORB的设计。

作用：管理服务器端伺服对象，对象标识，CORBA对象以及它们之间关联。决定在收到一个

客户请求时应调用哪一个伺服对象，然后调用该伺服对象上的合适操作。对象适配器主要完

成以下工作：对象登记、对象引用(OR, Object Reference)的产生、服务器进程的激活、对

象的激活、对象的撤消、对象向上调用。

**什么是POA？简述创建并激活POA的过程？**

◎**POA：**POA是对象实现与ORB其他组件之间的中介，它将客户请求传送到伺服对象，按需创建子POA，提供管理伺服对象的策略。

◎**创建:**利用create\_POA()方法可以创建一个新的POA作为该POA的子POA, 可以根据实际需要以这种方式创建多个子POA，从而形成一个POA层次

◎**激活:**对象引用通过对象标识与伺服对象建立关联。如果POA采用RETAIN策略，可有三种激活方式：显示激活，隐式激活，按需激活；若采用NON\_RETAIN策略，对象只能按需激活。

**CORBA对象和伺服对象的关系。**

①CORBA对象与具体的伺服对象是两个不同抽象层次的概念：它们之间的彻底分离使得CORBA独立于任何特定程序设计语言，并为服务端程序的可移植性打下基础。对象适配器是一个重要的ORB组件，它负责将抽象的CORBA对象映射到具体的伺服对象。

②CORBA对象可看作是一个具有对象标识、对象接口及对象实现的抽象实体。

③伺服对象（servant）是指具体程序设计语言的对象或实体，通常存在于一个服务程序进程之中。客户程序通过对象引用发出的请求经过ORB担当中介角色，转换为对特定的伺服对象的调用。

④伺服对象通过对象标识关联到CORBA对象。

**三、编写客户程序**

初始化ORB

是应用程序进入CORBA的起点

作用

让ORB了解有新的成员加入

获取ORB伪对象的引用，以备调用ORB内核提供的操作时使用

伪对象(pseudo object)：在CORBA基础设施中的一个对象

获取分布式对象引用，利用其调用对象实现提供的服务

作者：雾郦霁溪  
链接：https://www.jianshu.com/p/2662a2c4034a  
來源：简书  
简书著作权归作者所有，任何形式的转载都请联系作者获得授权并注明出处。