第3章 面向对象中间件框架

计算机之间的通信如果缺乏一个统一的标准，那么它们之间的通信就像不同语言的人互相交流，根本无法有效沟通。20世纪分布式系统就曾面临这样的窘境，之后国际标准化组织提出了OSI-RM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。本章将介绍一些经典的面向对象的中间件框架，从宏观上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。

3.1 开放分布式处理参考模型

3.1.1 面向对象技术

面向对象是相对于面向过程来讲的。面向对象方法把相关的数据和方法组织为一个整体来看待，从更高的层次来进行系统建模，更贴近事物的自然运行模式。其中，对象(Object)是面向对象技术中的核心概念，经常用于描述真实世界中具有一定状态和行为的实体。类(Class)是对对象实体的抽象，是对这些实体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。

1. 继承

继承(Inheritance)体现了一种层次结构和重用机制，它从已有类中派生出一个新的类，这个新类共享该已有类的结构和行为且自身也拥有额外自定义的内容。这很好地体现出了真实世界中的一般与特殊之间的关系。例如，动物与狗之间的关系，狗可以继承来自动物的一些基础属性与行为(例如五脏器官、会呼吸)，也可以拥有属于狗的独特行为(汪汪叫)。面向对象的这一特性使得我们在设计软件系统时更容易将真实世界代入其中，更好地分析与设计整个软件架构。

2. 封装

封装(Encapsulation)是指将对象和类的具体实现细节包装起来，并有选择地决定可供使用者读取的状态和调用的行为。通过这一屏蔽层，使得那些内部的实现对使用者是透明的。使用者只需要关心所要使用的功能和数据，更好地将注意力集中在所要解决的问题上。

3. 抽象

抽象(Abstraction)即抽取一组实体的与应用相关的共同特性并忽略那些不相关的特性。而共同特征是相对的，需要根据应用场景进行判断。例如苹果与计算机，可以抽取“价格”这一共同特征并属于“商品”这个类。而若对于“水果”与“电子设备”来说，苹果与计算机分别归属于这不同的两个类。所以在抽象时，共同与不同，取决于从什么角度上来抽象。

4. 多态

多态(Polymorphism)是指不同的对象可以共享同一行为,并且可以有不同的实现能力。例如,“动物”类拥有“奔跑”的行为,“人”和“狗”类都继承于“动物”,且其都拥有“奔跑”的行为。但是两者的对象各自拥有不一样的“奔跑”行为(人通过双腿着地奔跑,而狗通过四肢着地奔跑)。

3.1.2 RM-ODP 标准组成

为了实现计算机异构系统之间的通信,在 20 世纪国际标准化组织(International Organization for Standardization,ISO)提出了 OSI-RM(Open System Interconnection Reference Model)标准,为网络的发展打下了基础。然而 OSI-RM 还是有其局限性,缺乏一个面向应用的模型标准。于是之后又有了开放分布式处理参考模型 RM-ODP(Reference Model of Open Distributed Processing)。RM-ODP 不仅是一个标准,它还提出了一个分布式处理领域内的标准应该严格遵循的模型规范,可以说是对标准提出的标准。

1. 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。观点是一个完整系统规范的细分,在系统的分析或设计过程中,将与某个特定关注领域相关的特定信息集中在一起。虽然单独指定,但观点并不完全独立;每个观点中的关键条目被标识为与其他观点中的条目相关。此外,每个观点实质上都使用相同的基本概念。然而,这些观点是足够独立的,可以简化关于完整规范的推理。由 RM-ODP 定义的体系结构确保了观点之间的相互一致性,使用公共对象模型提供了将它们绑定在一起的黏合剂。

RM-ODP 框架提供了以下五种关于系统及其环境的通用、互补的观点。

企业观点(Enterprise Viewpoint):集中于系统的目的、范围和策略。它描述了业务需求以及如何满足这些需求。

信息观点(Information Viewpoint):关注信息的语义和所进行的信息处理。它描述了系统管理的信息以及支持数据的结构和内容类型。

计算观点(Computational Viewpoint):通过将系统的功能分解为在接口处交互的对象来实现分布。它描述了系统提供的功能及其功能分解。

工程观点(Engineering Viewpoint):集中于支持系统中对象之间的分布式交互所需的机制和功能。它描述了系统为管理信息和提供功能而执行的处理的分布。

技术观点(Technology Viewpoint):集中于系统的技术选择。它描述了为提供信息的处理、功能和表示而选择的技术。

2. 透明性

透明性是对用户(程序员、系统开发人员、用户或应用程序)隐藏的分布式系统的某些方面。透明性是通过在需要透明性的接口下面的一层包含一些机制来提供的。透明性屏蔽了由系统的分布所带来的复杂性。在分布式系统中,透明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。目前已经有许多为分布式系统定义的透明性。

重要的是要认识到并非所有这些都适用于每个系统,或者在相同的接口级别上都可用。

以下所罗列的也并非囊括所有的透明性。事实上，所有的透明性都有一个相关的成本，对于分布式系统实现者来说，意识到这一点非常重要。

(1) 访问透明性(Access Transparency)：本地和远程访问方法之间应该没有明显的区别，换言之，显性交流可能是隐藏的。例如，从用户的角度来看，对远程服务(如打印机)的访问应该与对本地打印机的访问相同。从程序员的角度来看，远程对象的访问方法可能与访问同一类的本地对象相同。

(2) 位置透明性(Location Transparency)：在计算中，使用名称来标识资源，而用户不需要知道这些资源的物理位置，也不必关心系统拓扑结构的细节。由于使用了索引，名称可以用于访问资源，这些资源可以位于本地或远程，并且允许任何有权访问这些资源的人使用这些资源。

(3) 迁移透明性(Migration Transparency)：如果对象(进程或数据)迁移(为了提供更好的性能或可靠性，或者为了隐藏主机之间的差异)，应该对用户隐藏这一点。

(4) 失败透明性(Failure Transparency)：如果发生软件或硬件故障，应该对用户隐藏这些故障。这在分布式系统中很难提供，因为通信子系统可能会出现部分故障，并且可能不会报告。故障透明度将尽可能由与访问透明度相关的机制提供。

(5) 重定位透明性(Relocation Transparency)：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。

(6) 复制透明性(Replication Transparency)：如果系统提供了复制(出于可用性或性能原因)，则不应涉及用户。对于数据库而言，复制透明性是指用户不必关心在网络中各个结点的数据库复制情况，更新操作引起的问题将由系统去处理。

(7) 持久透明性(Persistence Transparency)：指在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。例如，Java 序列化是透明持久化的一种形式，因为它可以用来直接将 Java 对象持久化到文件，而无须付出太多的努力。

(8) 事务处理透明性(Transaction Transparency)：在分布式环境中，往往需要维护一组相关对象之间的协调，而事务处理透明性需要对用户屏蔽这些协调相关的信息。

(9) 缩放透明性(Scaling Transparency)：系统应该能够在不影响应用程序算法的情况下增长。优雅的成长和演进是大多数企业的重要要求。系统还应能够在需要时缩小到小环境，并根据需要节省空间和时间。

(10) 并发透明性(Concurrency Transparency)：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。这在分布式系统中需要非常复杂的机制，因为存在真正的并发而不是中央系统的模拟并发。例如，分布式打印服务必须为每个文件提供与中央系统相同的原子访问，以便打印输出不会随机交错。

3.1.3 RM-ODP 功能组成

RM-ODP 平台的通用功能具有如下四种。

(1) 管理功能(Management Functions)：对分布式系统中的基本组成要素的管理，例如，结点管理、对象管理、对象串管理等。

(2) 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，例如，事件通知、取消激活与再激活、复制迁移、事务处理等。

(3) 仓库功能(Repository Functions)：提供分布式系统的信息存储与检索功能，例如，

信息组织、重定位、类型仓库等。

(4) 安全功能(Security Functions)：提供分布式系统的安全管理机制，例如，访问控制、安全审核、用户认证、密钥管理等。

3.2 CORBA 框架

3.2.1 OMA 介绍

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

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)和参考模型(Reference Model)。对象模型定义如何描述分布式异质环境中的对象，参考模型描述对象之间的交互。

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，并能给客户提供一个或多个服务。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。

如图 3-1 所示，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界面。包括：

(1) 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。

(2) 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。

(3) 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。

(4) 应用界面(Application Interface)：由厂商针对某一具体应用所定义的界面。

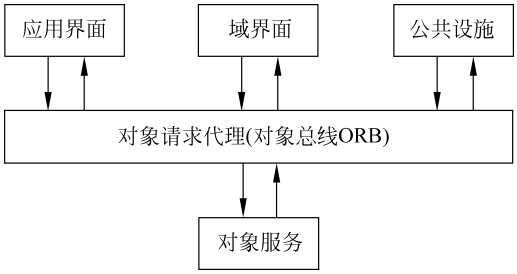

图 3-1 OMA 参考模型

3.2.2 CORBA 介绍

公共对象请求代理体系结构(CORBA)是一种用于创建基于对象的分布式应用程序的

规范。CORBA 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。如图 3-2 所示，CORBA 规范包括如下几部分。

- ORB 核心(Object Request Broker Core)。
- 接口定义语言和语言映射(Interface Definition Language and Language Mapping)。
- 存根和框架(Stub and Skeleton)。
- 动态调用(Dynamic Invocation)。
- 对象适配器(Object Adapter)。
- 接口仓库和实现仓库(Interface Repository and Implementation Repository)。
- ORB 之间的互操作(Interoperability between ORB)。

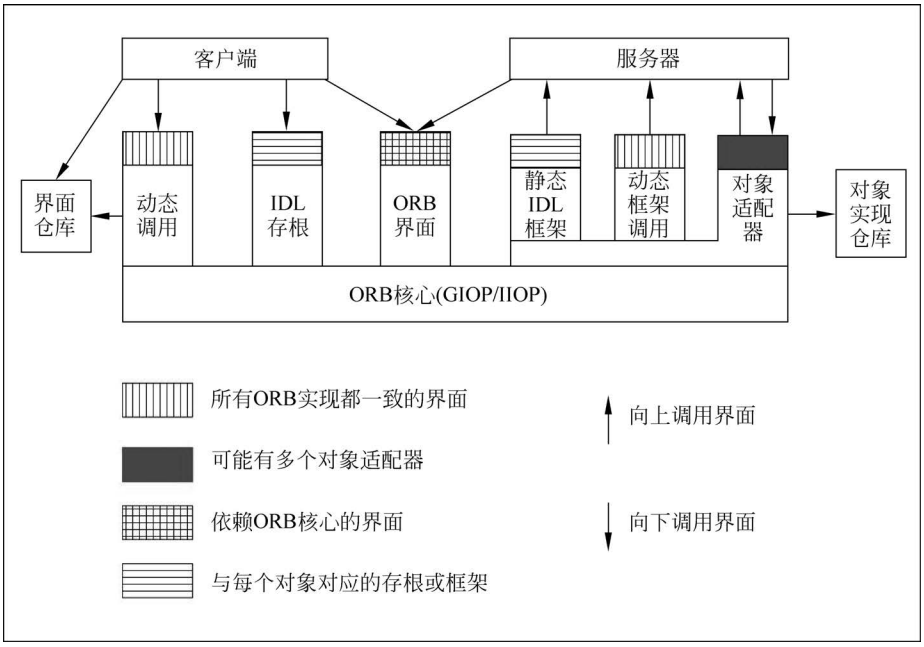

图 3-2　CORBA 体系结构

下面将简要介绍每个部分的内容。

1. ORB 核心

CORBA 将 ORB 定义为客户机和服务器应用程序之间的中介。ORB 将客户端请求传递到适当的服务器应用程序，并将服务器响应返回给请求的客户端应用程序。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，其中屏蔽的内容主要包括以下内容。

(1) 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。

(2) 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、对象所在结点的操作系统和硬件平台等。

(3) 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。

(4) 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。

(5) 数据表示：客户不必知道本地主机和远程主机的数据表示方式(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，能将更多时间与精力集中在应用层面的设计。

2. 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的。在 CORBA 中，对象的接口是利用 OMG IDL(OMG Interface Definition Language)来定义的，它类似于 C++语法。OMG IDL 不是编程语言，而是一个纯说明性语言，并且与具体主机上的编程语言无关。这就很自然地将接口与对象实现分离，使得虽然是用不同的语言来实现的对象，但对象之间又可以进行互操作。

由于不能用 OMG IDL 直接去实现分布式应用，所以需要进行语言映射，即把 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。

3. 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；框架的作用则是把请求交给 CORBA 对象实现。换句话说，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。框架在请求的接收端提供类似存根的服务。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。

4. 动态调用

除了前面所提到的通过存根和框架进行的静态调用，CORBA 还支持两种用于动态调用的接口：动态调用接口(Dynamic Invocation Interface)和动态框架接口(Dynamic Skeleton Interface)，它们分别是支持客户端的动态请求调用和支持服务端的动态对象调用。

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。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。

5. 对象适配器

在 CORBA 中，提供了基本对象适配器和移动对象适配器。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销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。

6. 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。

7. 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 由以下 3 部分组成。

(1) 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。

(2) 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。

(3) GIOP 传输层假设(GIOP Transport Assumptions):GIOP 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,提供乱序通知功能,连接的发起方式可以映射到 TCP/IP 这样的一般连接模型。

GIOP 只是一种抽象协议,独立于任何特定的网络协议,在实现时必须映射到具体的传输层协议或者特定的传输机制之上。GIOP 到 TCP/IP 的映射又称为 IIOP(Internet Inter-ORB Protocol)。

3.2.3 CORBA 的优势与发展

CORBA 规范通过定义以下内容,为构建分布式应用程序提供了一个广泛而一致的模型。

(1) 用于构建分布式应用程序的对象模型。

(2) 客户端和服务器应用程序使用的一组通用的应用程序编程对象。

(3) 描述在分布式应用程序开发中使用的对象接口的语法。

(4) 支持使用多种编程语言编写的应用程序。

CORBA 规范描述了如何实现 CORBA 模式开发，也描述了开发人员用来开发应用程序的编程语言绑定。为了说明使用 CORBA 体系结构的优势，以下将传统的客户机/服务器应用程序开发技术与 CORBA 开发技术进行比较。

1. 传统客户机/服务器模式

客户机/服务器计算是一种应用程序开发方法，它允许程序员在联网的机器系统之间分配处理，从而能够更有效地利用机器资源。在客户机/服务器计算中，应用程序由两部分组成：客户机应用程序和服务器应用程序。这两个应用程序通常运行在不同的机器上，通过网络连接，如图 3-3 所示。

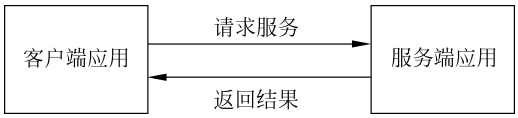

图 3-3　传统客户机/服务器模式

客户机应用程序请求信息或服务，通常为用户提供显示结果的方法。服务器应用程序满足一个或多个客户机应用程序的请求，通常执行计算密集型功能。客户机/服务器模式的主要优点如下。

- 计算功能运行在最合适的机器系统上。
- 开发人员可以在多个服务器之间平衡应用程序处理的负载。
- 服务器应用程序可以在许多客户机应用程序之间共享。

例如，桌面系统为许多业务用户提供了一个易于使用的图形环境来显示信息。但是，桌面系统的磁盘空间和内存可能受到限制，并且通常是单用户系统。更大、更强大的服务器系统更适合执行计算密集型功能，并提供多用户访问和共享数据库访问。因此，较为强大的系统通常运行应用程序的服务器部分。这样，分布式桌面系统和网络服务器为部署分布式客户机/服务器应用程序提供了一个完美的计算环境。

尽管传统客户机/服务器模式提供了在异构网络中分布处理的方法，但它有以下缺点。

- 客户机必须知道以何种网络协议访问服务器。客户机/服务器应用程序可能使用相同的单一网络协议或不同的协议。如果它们使用多个协议，应用程序必须在逻辑上为每个网络重复特定于协议的代码。
- 当计算机之间使用不同数据格式通信时，必须显式处理数据格式转换。例如，有些机器将整数值从最低字节地址读取到最高字节地址(Little-Endian)，而其他机器则从最高字节地址读取到最低字节地址(Big-Endian)。一些机器系统也可能对浮点数或文本字符串使用不同的格式。如果应用程序向使用不同数据格式的计算机发送数据，但应用程序不转换数据，则数据会被误解。通过网络传输数据并将其转换为目标系统上的正确表示形式称为数据封送处理。在许多传统客户机/服务器模式中，应用程序必须执行所有数据封送。数据封送处理要求应用程序使用网络和操作系统的功能将数据从一台计算机移动到另一台计算机。它还要求应用程序执行所有数据格式转换，以确保以发送数据的方式读取数据。

- 扩展应用程序的灵活性较差。传统客户机/服务器模式将客户机和服务器应用程序连接在一起。因此，如果客户机或服务器应用程序发生更改，程序员必须更改接口、网络地址和网络传输。此外，如果程序员将客户机和服务器应用程序移植到支持不同网络接口的计算机上，则必须为这些应用程序创建新的网络接口。

2. CORBA 开发模式

CORBA 模型为开发分布式应用程序提供了一种更加灵活的方法。CORBA 模型改进了以下几点。

- 正式分离应用程序的客户端和服务器部分。CORBA 客户机应用程序只知道如何请求完成某项任务，而 CORBA 服务器应用程序只知道如何完成客户机应用程序请求它完成的任务。由于这种分离，开发人员可以更改服务器完成任务的方式，而不影响客户机应用程序请求服务器应用程序完成任务的方式。
- 从逻辑上将应用程序分离为可以执行特定任务的对象，称为操作。CORBA 是基于分布式对象计算模型，它结合了分布式计算（客户机和服务器）和面向对象计算（基于对象和操作）的概念。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账户的对象，以及存款、取款和查看账户余额的操作。
- 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。
- 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。而且，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 模型允许客户机应用程序向服务器应用程序发出请求，并从服务器应用程序接收响应，而无须直接知道信息源或其位置。在 CORBA 环境中，客户机和服务器应用程序与 ORB 通信。图 3-4 显示了客户机/服务器环境中的 ORB。

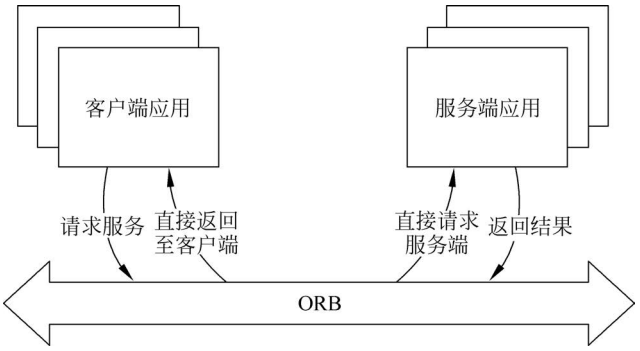

图 3-4　CORBA 开发模式

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。这意味着程序员可以改变服务器的应用程序及代码，而不必担心因此会影响客户端的程序及代码。例如，只

要客户机和服务器应用程序之间的接口不变，程序员就可以在不改变客户机应用程序的情况下发展和创建服务器应用程序的新实现。此外，他们也可以在不改变服务器应用程序的情况下创建新的客户端应用程序。

3. CORBA 产品及其复杂性

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(Internet Inter-ORB Protocol，IIOP)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。

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。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。这些都制约了 CORBA 技术的发展和普及。

3.3 COM 组件模型

3.3.1 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组件。

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(Component)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。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，应用程序或其他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。

组件技术将面向对象特性（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成熟的实用化阶段。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。

3.3.2 COM 的发展历程

以组件对象模型（Component Object Model，COM）为代表的组件技术大幅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一个组件。在构建应用系统时，将这些组件有机地组装起来就成为一个系统，就像是用零件组装出一台机器一样。组件技术的特性如下。

- 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。
- 可以方便地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。

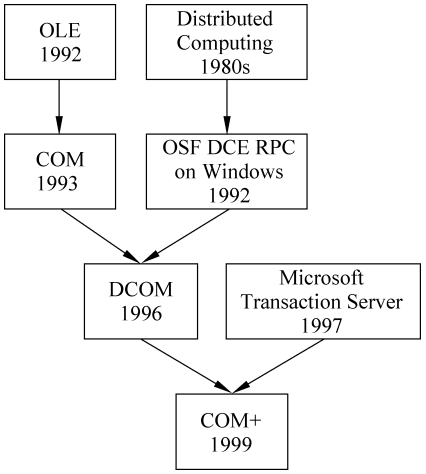

图 3-5　COM 组件的发展

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 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，如图 3-5 所示。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。

动态链接库（Dynamic Link Library，DLL）包含着许多可以被应用程序调用的函数、数据的库文件。与之相对的则是静态链接库，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。因此，才出现了动态链接库。采用动态链接库，对公用的库函数，系统只有一个副本（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,也只有一个副本,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。

对象连接与嵌入技术(Object Linking and Embedding,OLE)是微软推出的应用程序互联技术,它提出了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。

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。

随着分布式计算环境的逐步推广应用,1996 年,微软又推出 DCOM(Distributed COM),它可以实现不同计算机之间的 COM 调用。DCOM,顾名思义也就是分布式的 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。

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。在此环境中执行的组件称为配置组件,不利用此环境的组件称为未配置组件。它们均按照 COM 的标准规则运行。

3.3.3 COM 组件

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。

在 Microsoft 公司的开发者网络(Microsoft Developer Network,MSDN)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),不管计算机运行的是什么操作系统(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。”

在 COM 技术中,有以下比较重要的概念,如图 3-6 所示。

(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。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。

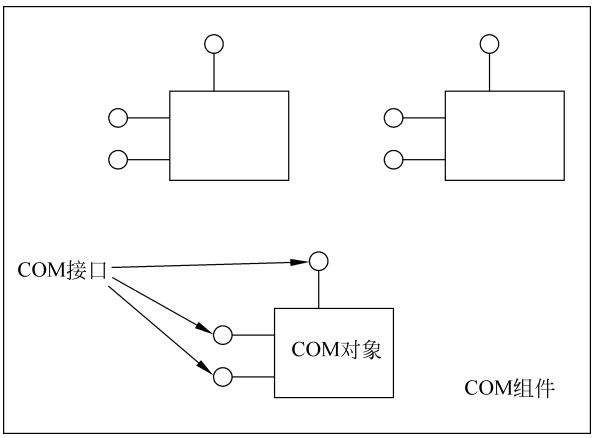

图 3-6　COM 接口、对象和组件模型示意图

在 COM 技术中，组件和接口是其核心概念。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（Globally Unique Identifier，GUID），它与开放软件基金会（Open Software Foundation，OSF）分布式计算环境（Distributed Computing Environment，DCE）定义的通用唯一标识符（Universally Unique Identifier，UUID）相同。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，它必须使用全角唯一标识符。

所有接口都继承自 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。IUnknown 接口有三个成员函数，分别名为 QueryInterface、AddRef 和 Release。所有 COM 对象都需要实现 IUnknown 接口。QueryInterface 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。QueryInterface 成员函数允许在 COM 对象支持的所有接口之间导航。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，因为没有调用方在使用它。

当需要创建 COM 实例时，则需要向服务器传递 CLSID（类的 ID，服务器将 CLSID 与 COM 类相关联）。创建实例的最简单方法是调用 COM 函数 CoCreateInstance。CoCreateInstance 函数创建指定 CLSID 的一个实例，并返回客户端请求的类型的接口指针。客户端负责管理实例的生存期，方法是在客户端使用完实例后调用其释放函数（调用 IUnknown 接口的 Release 函数）。要基于单个 CLSID 创建多个对象，可调用 CoGetClassObject 函数。要连接到已创建并正在运行的对象，可调用 GetActiveObject 函数。

3.3.4 DCOM 组件

微软提出分布式组件对象模型(Distributed Component Object Model,DCOM),用于在联网计算机上的软件组件之间进行通信。DCOM 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。

- 编组:序列化和反序列化参数和返回值。
- 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。
- 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,以最大限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。

20 世纪 90 年代 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。然而,这两种技术在因特网防火墙上、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。

3.3.5 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+是微软组件对象模型(COM)和微软事务服务器(MTS)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+处理了许多以前必须自己编程的资源管理任务,例如,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+提供了一个基于微软组件对象模型(COM)的企业开发环境,用于创建基于组件的分布式应用程序。它还提供了创建事务性、多层应用程序的工具。COM+将对传统基于 COM 的开发的增强功能与许多有用的编程和管理服务相结合。增强功能包括线程和安全性的改进,以及同步服务的引入等。对于熟悉 COM 编程的人来说,COM+的改进非常重要,包括以下几点。

- COM+实现了一个称为中性单元线程的线程模型,它允许组件具有序列化访问以及在任何线程上执行的能力。

- COM+支持一个称为 context 的特殊环境的组件，它提供了一组可扩展的属性，这些属性定义了组件的执行环境。
- COM+提供基于角色的安全性、异步对象执行和一个内置的名字对象，它表示对进程外服务器上运行的对象实例的引用。

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 国际互联网络。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。

3.3.6 .NET 组件

COM 定义了一个组件模型，使得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。微软 2002 年推出的.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 组件有着与 COM 类似的目标，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(Common Language Runtime，CLR)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(Framework Class Library，FCL)。

.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，开发者必须要在服务器上注册组件，系统注册表中的组件信息必须被更新。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，只要简单地把它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 组件的对比如表 3-1 所示。

表 3-1 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。类型库包含接口、方法、参数以及 UUID 等。通过 IDL 来进行描述	元数据可以通过定制特性来扩展，不用了解 IDL

续表

项　　目	COM 组件(C++)	.NET 组件(C#)
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
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
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

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。

思　考　题

1. 面向对象有哪些特征?相比于面向过程,有哪些不同?
2. 简述 RM-ODP 框架关于系统及其环境的观点。
3. 简述透明性的概念,对于系统开发者来说有哪些透明性?
4. RM-ODP 平台的通用功能有哪些?
5. 简述传统客户机/服务器模式及其优缺点。

6. CORBA 规范包含哪些模块？简述这些模块的主要内容。

7. CORBA 模式相比于传统客户机/服务器模式有何优势？

8. 什么是接口定义语言？语言映射的作用是什么？

9. 什么是 COM 组件技术？它有哪些特征？

10. DCOM 作为 COM 的增强与扩展，其主要解决哪些问题？

11. 相比于 COM，COM＋改进了哪些方面？

实验 1　跨语言的调用和编程

一、实验目的

理解和掌握跨语言调用的原理，能够基于提供的框架实现简单的跨语言编程和调用。

二、实验平台和准备

操作系统：Windows 或者 Linux 系统。

需要的软件：Java JDK(1.8 或以上)、IDEA 或者 Eclipse 等开发环境。

三、实验内容和要求

搜索跨语言开发调用的常用框架/库，阅读相关文档并实现以下的功能。

某功能 A 用的是 L1 语言编程实现的，请把该功能(非源代码)在 L2 语言的环境下进行调用，并能正确地返回结果。功能 A 可自己实现，或者仅有第三方的可执行文件。跨语言调用一般可基于语言本身的接口、脚本、第三方的库或解决方案。

(1) 功能 A：数据压缩和解压缩功能，语言 L1——C++，语言 L2——Python 和 Java。

(2) 功能 A：图像基本变换功能(2～3 个基本操作)，语言 L1——Python，语言 L2——C++和 Java。

(3) 功能 A：加密和解密功能，语言 L1——Java，语言 L2——C++和 Python。

(4) 功能 A：矩阵变换或者数学运算(选择 2～3 个基本变换)，语言 L1——Python，语言 L2——Node.js 和 C＃。

四、扩展实验内容

请在实验报告中讨论跨语言开发的利弊。除了直接的跨语言调用，还有哪些方式可以进行多语言的协同开发？如果没有第三方的库可以实现直接的跨语言调用，该如何实现多语言的协作呢？请讨论并给出方案。