分布计算环境

北京邮电大学计算机学院

Chapter 3

面向对象的分布计算环境

主要内容

- 分布式系统中的面向对象技术
- CORBA技术:Common Object Request Broker Architecture

绪论

- OMG组织制定的一个工业规范，是一个体系结构和一组规范
  - 1991年 1.1，2012年 3.3，2020年 3.4 beta
- 目的：在分布式环境下实现应用的集成，使基于对象的软件成员，在分布的、异构的环境下可重用、可移植、**可互操作**
  - OMG的理念：不存在统一的硬件平台、不存在统一的操作系统、编程语言、网络协议、应用模式，必须在互操作上达成一致
- 方法：提供一个框架，如果符合这一框架，就可以在主要的硬件平台和操作系统上建立一个异质的分布式应用
- CORBA结合了当时计算机工业中的两个重要趋势：面向对象软件开发和客户机/服务器计算

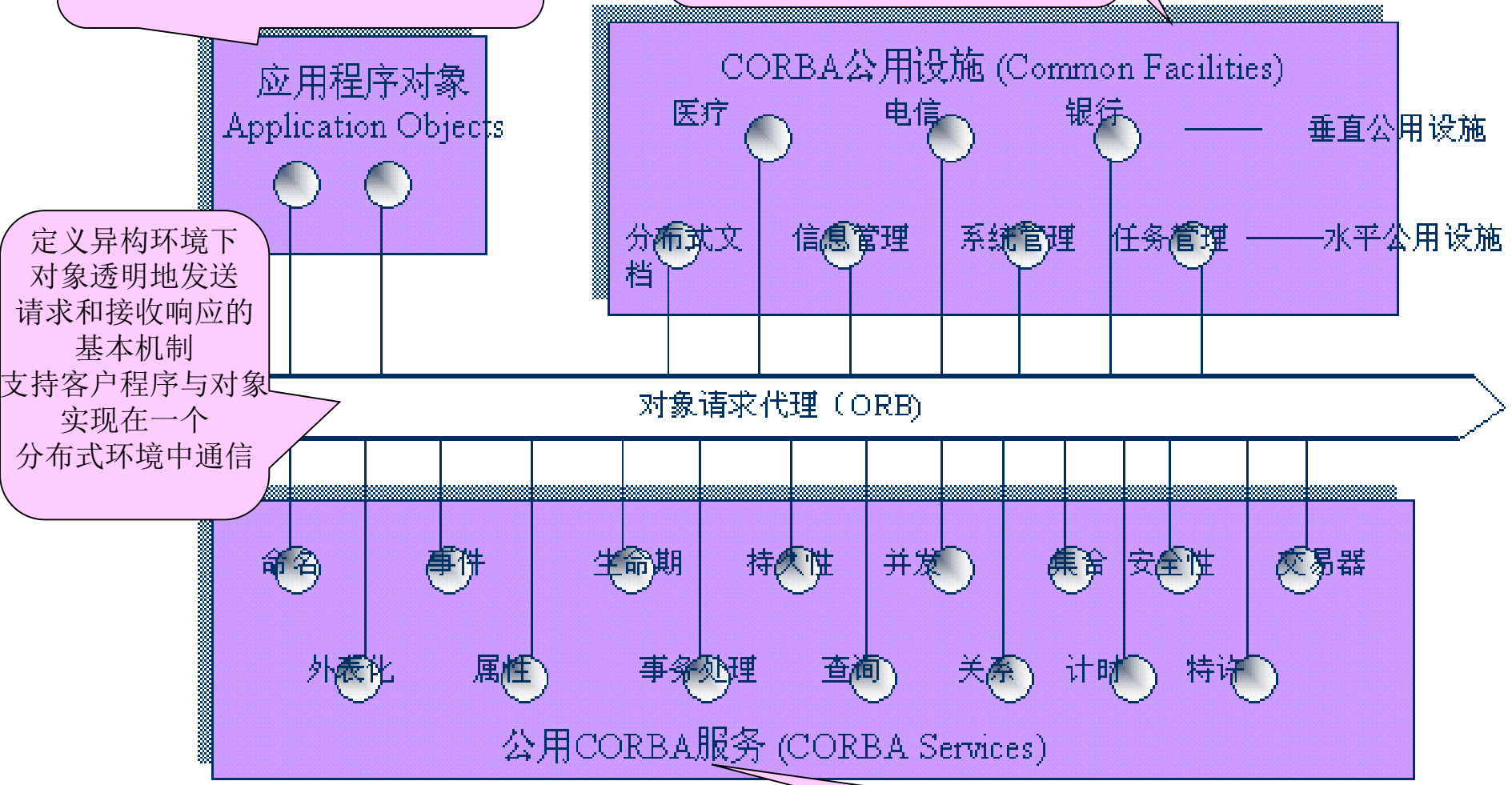
主要内容

- 对象管理体系结构OMA
- **OMG的接口定义语言IDL**
- **对象请求代理ORB**
- CORBA服务
- ORB之间的互操作
- CORBA中的基本原理

对…结构OMA

由CORBA支持的顶层业务对象和应用系统,是针对特定应用开发的接口

可用于大多数应用领域的面向终端用户的工具接口
水平公用设施：领域间可共享
垂直公用设施：面向某个领域

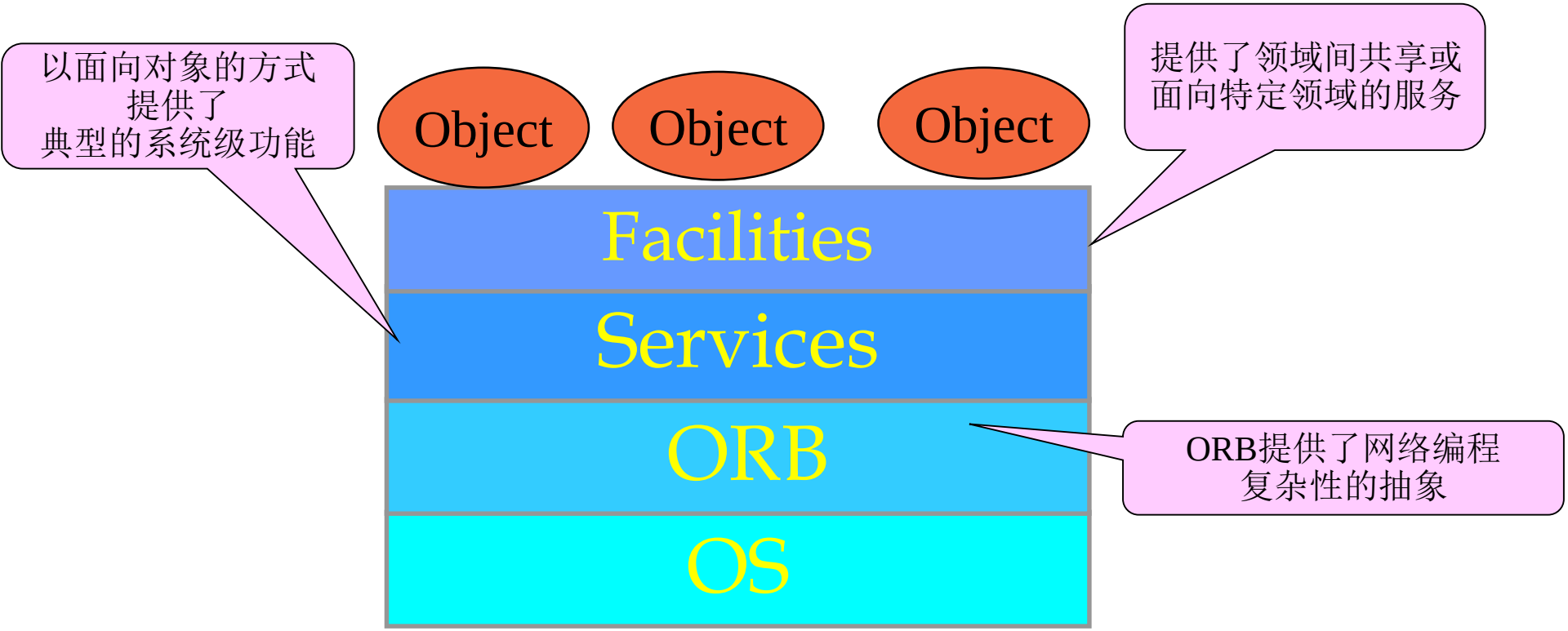


定义异构环境下对象透明地发送请求和接收响应的基本机制
支持客户程序与对象实现在一个分布式环境中通信

基于分布式对象的所有应用程序都可能用到的通用服务的接口

OMA

- CORBA不只定义了面向对象的远程调用机制，它还定义了不同抽象层次的框架



主要内容

- 对象管理体系结构OMA
- **OMG的接口定义语言IDL**
- **对象请求代理ORB**
- CORBA服务
- ORB之间的互操作
- CORBA中的基本原理

OMG接口定义语言IDL

- 仅定义接口，不定义实现
- 分隔“对象作什麽 (WHAT)”与“如何做 (HOW)”
- 强类型、面向对象、语言中立的说明(描述)型语言
- ANSI C++ 的子集
- 支持多继承
- 支持到多种语言的映射

一个实例

```
interface Hello
{
        void say_hello();
};
```

IDL到编程语言的映射

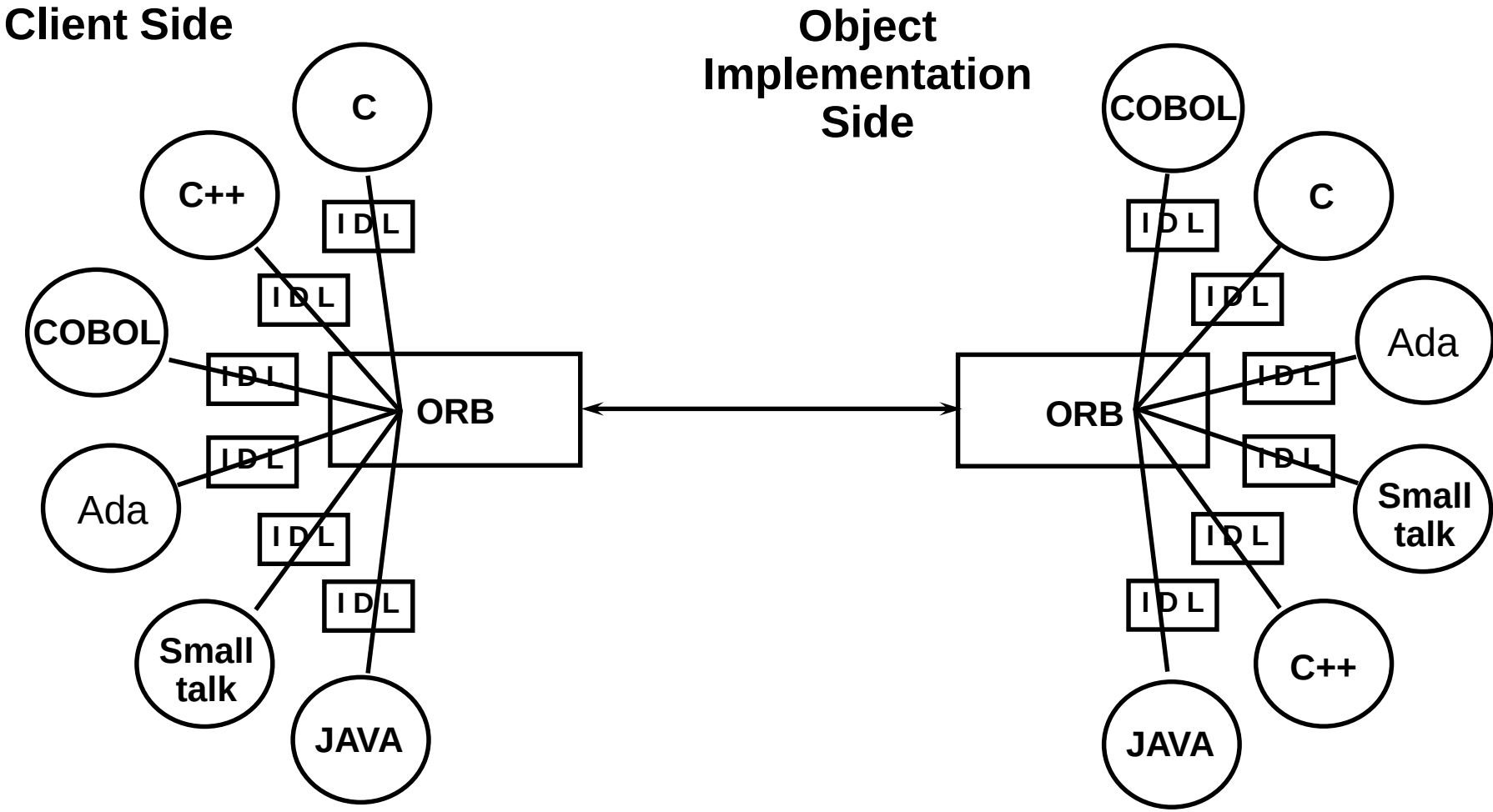
- 定义相应编程语言所用到的数据类型的定义，如：

OMG IDL	Java
short	short
long long	long long
octet	byte

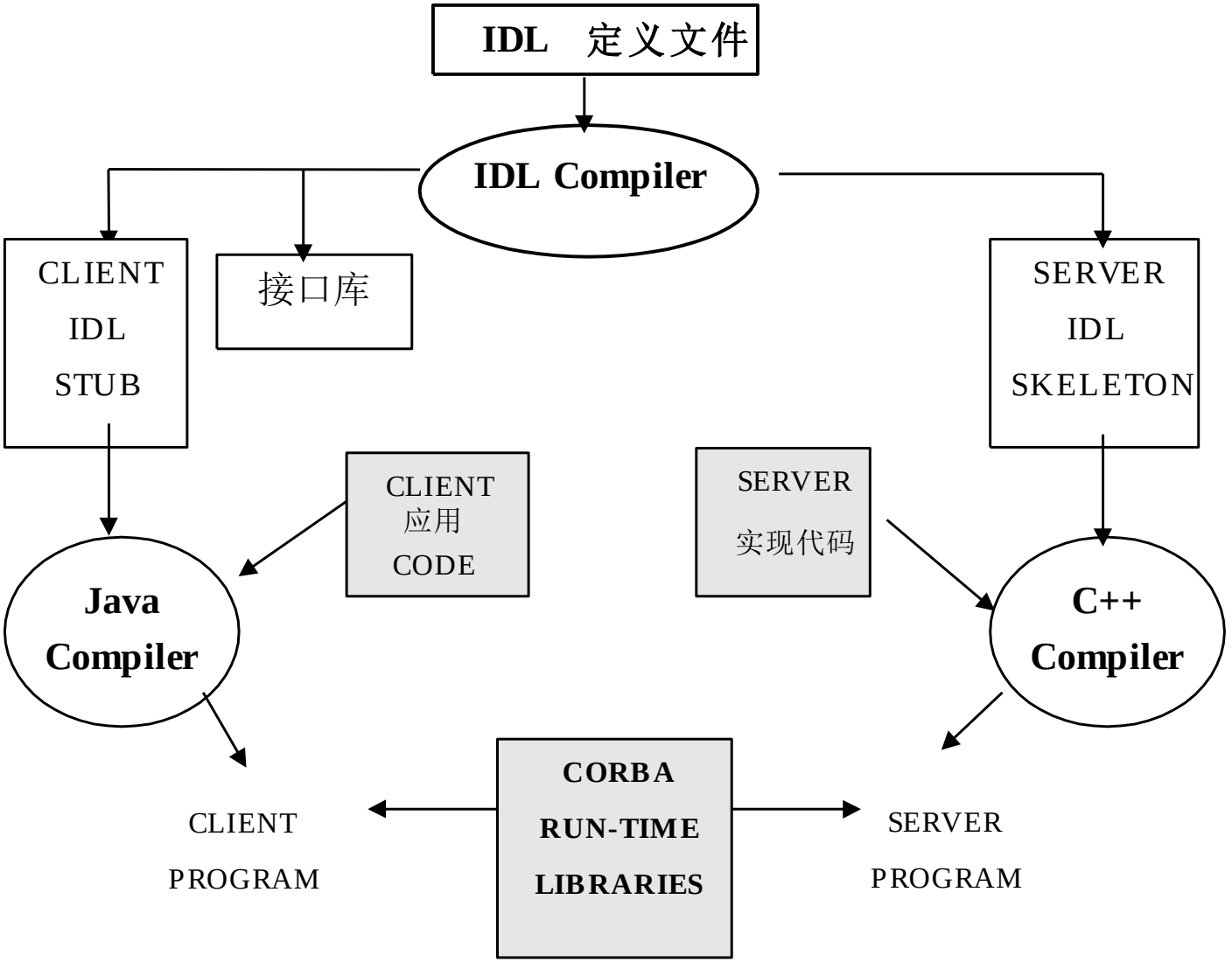
- ……

- 目前有：Ada、C、C++、Java、Lisp、COBOL、Python、Smalltalk

Role of CORBA IDL



IDL的使用


IDL 定义文件
IDL Compiler
CLIENT
IDL
STUB
接口库
SERVER
IDL
SKELETON
CLIENT
应用
CODE
SERVER
实现代码
Java
Compiler
C++
Compiler
CORBA
RUN-TIME
LIBRARIES
CLIENT
PROGRAM
SERVER
PROGRAM

IDL和Stub例

- Hello.idl定义：

```
interface Hello{
    void say_hello();
}
```

IDL到Java编译例：
jidl --package hello Hello.idl

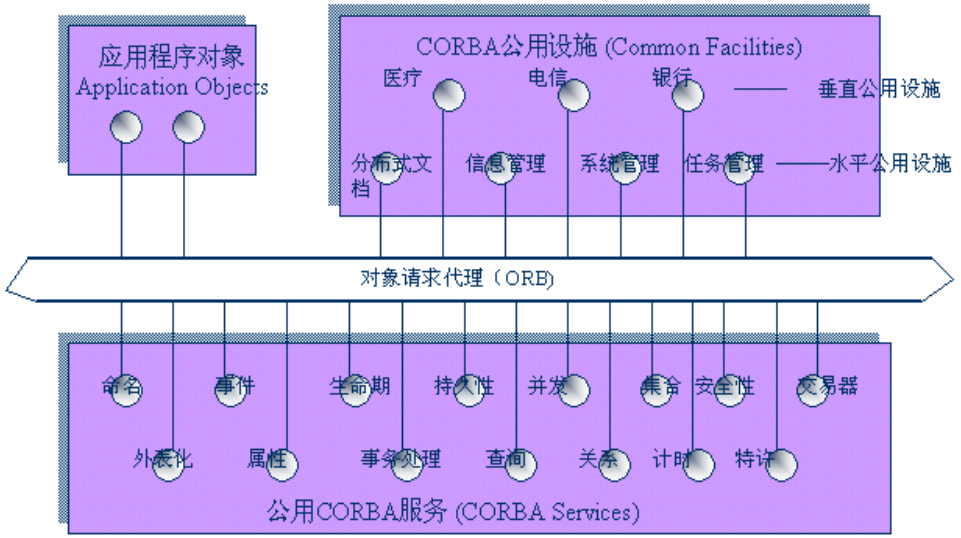
- 客户端Stub：
  - HelloOperations.java：定义public interface HelloOperations
  - Hello.java：定义接口 Interface Hello
  - _HelloStub.java：桩代码，定义了class _HelloStub
  - HelloHelper.java：定义 public class HelloHelper对象
- 服务器端Skeleton：
  - HelloOperations.java：定义public interface HelloOperations
  - HelloHolder.java： 定义public final class HelloHolder
  - HelloPOA.java：定义类abstract public class HelloPOA

例:

```
public interface HelloOperations
{
    //
    // IDL:Hello/say_hello:1.0
    //
   void   say_hello();
}

public interface Hello extends HelloOperations,
                                org.omg.CORBA.Object,
                                org.omg.CORBA.portable.IDLEntity
{
}
```

主要内容

- 对象管理体系结构OMA
- **OMG的接口定义语言IDL**
- **对象请求代理ORB**
- CORBA服务
- ORB之间的互操
- CORBA中的基本



对象请求代理ORB

- 对象请求代理ORB（Object Request Broker）：定义异构环境下对象透明地发送请求和接收响应的基本机制。ORB 为客户隐藏：
  - 对象位置
  - 对象实现方式
  - 对象执行状态
  - 对象通信机制
- ORB并不需要作为一个单独的组件来实现。它定义了一系列的接口，任何一种支持了该接口的实现方式都是可行的

ORB体系结构

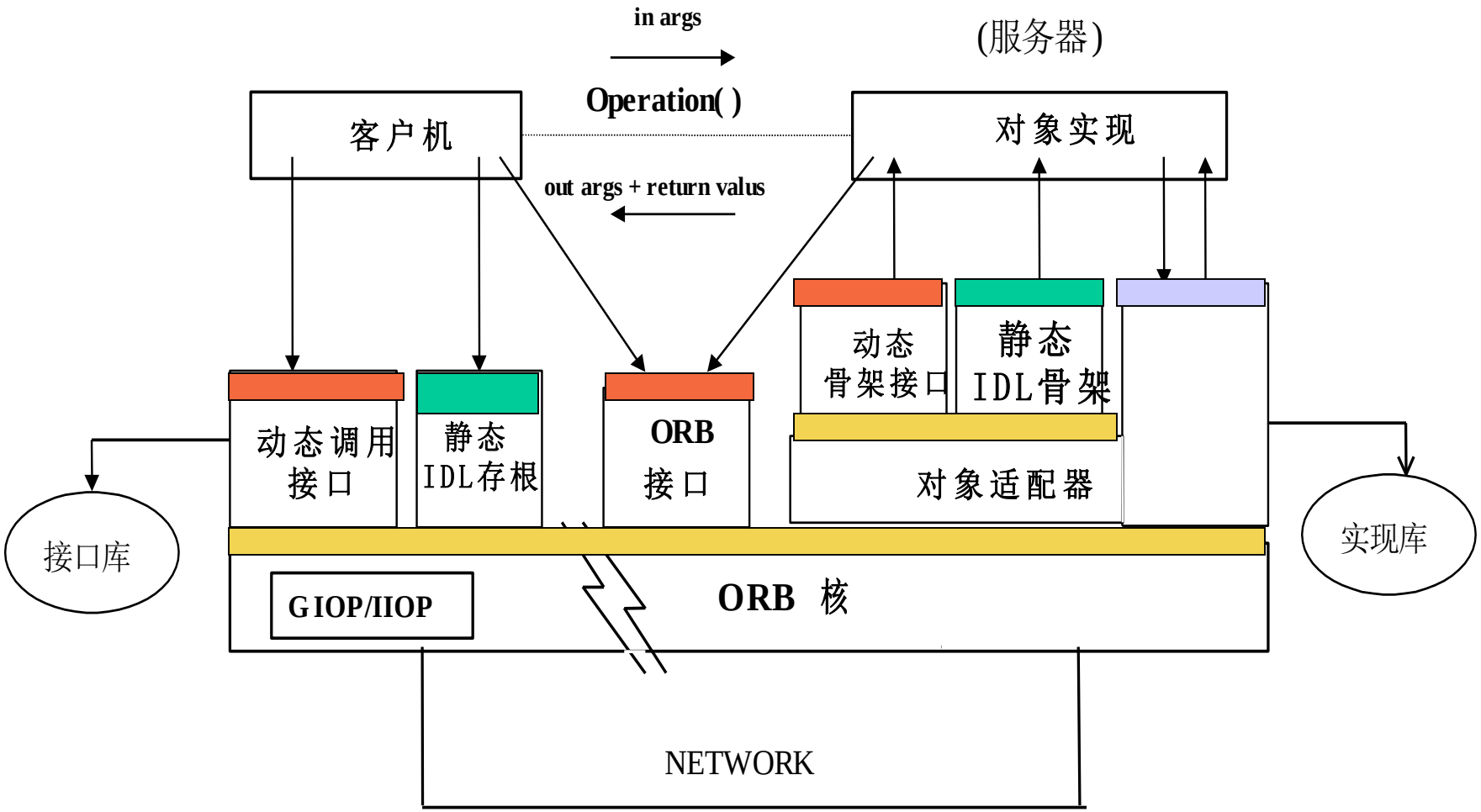


图3-3 CORBA ORB的体系结构

ORB主要内容

- 接口
- 静态存根和骨架
- 对象实现和客户
- 对象引用
- ORB核心
- 对象适配器

接口

- 客户程序与对象实现之间的界面，描述了客户可访问的对象操作的一个集合
  - 完全独立于对象所处的位置、实现对象的程序设计语言以及对象接口中未反映的其他特性
- 客户程序只能通过对象的接口定义掌握对象的逻辑结构，并通过发送请求来影响对象的行为与状态
- 对象接口采用接口定义语言IDL定义

```
interface Hello
{
        void say_hello();
};
```

ORB的各种接口

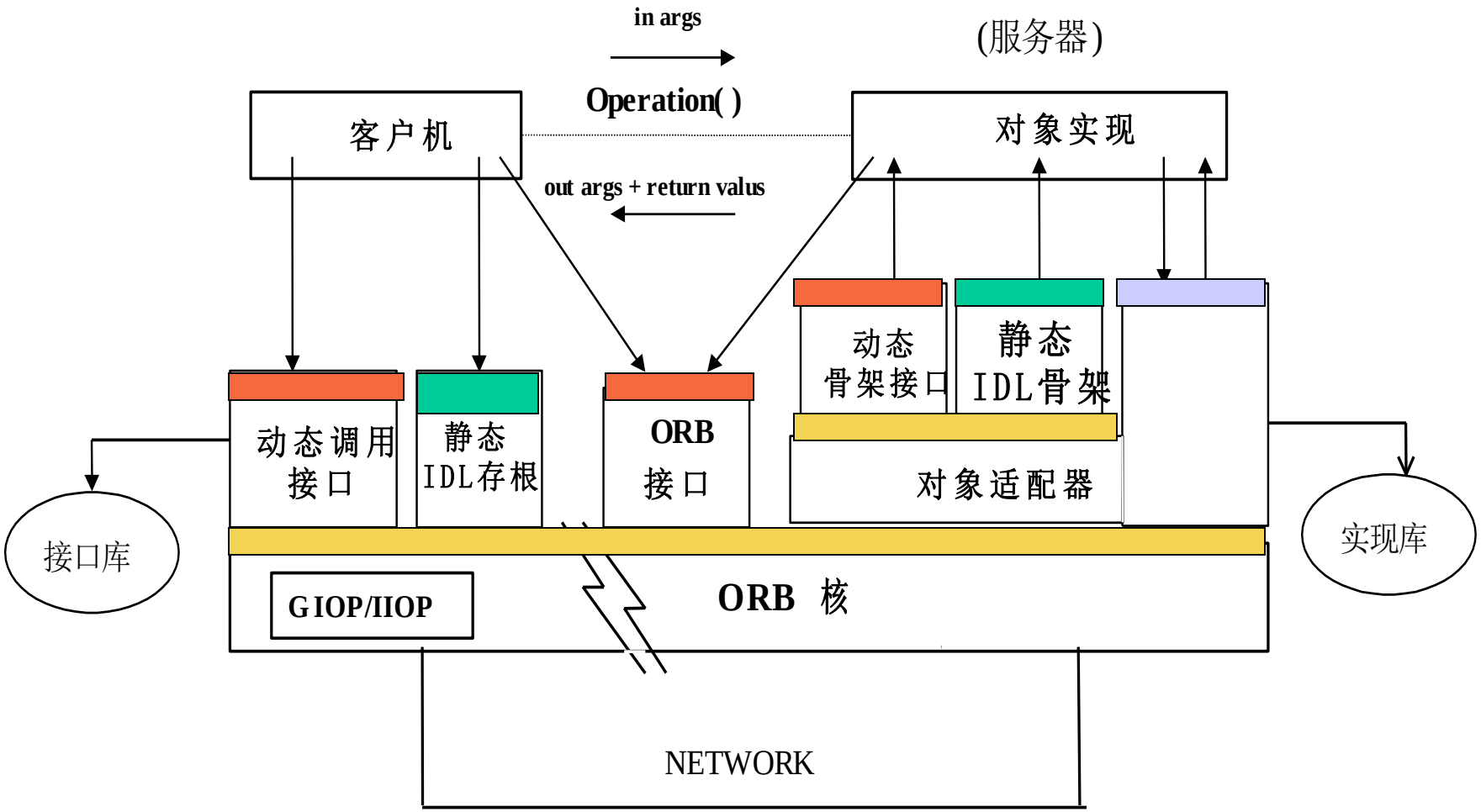


图3-3 CORBA ORB的体系结构

ORB主要内容

- 接口
- 静态存根和骨架
- 对象实现和客户
- 对象引用
- ORB核心
- 对象适配器

静态存根和骨架

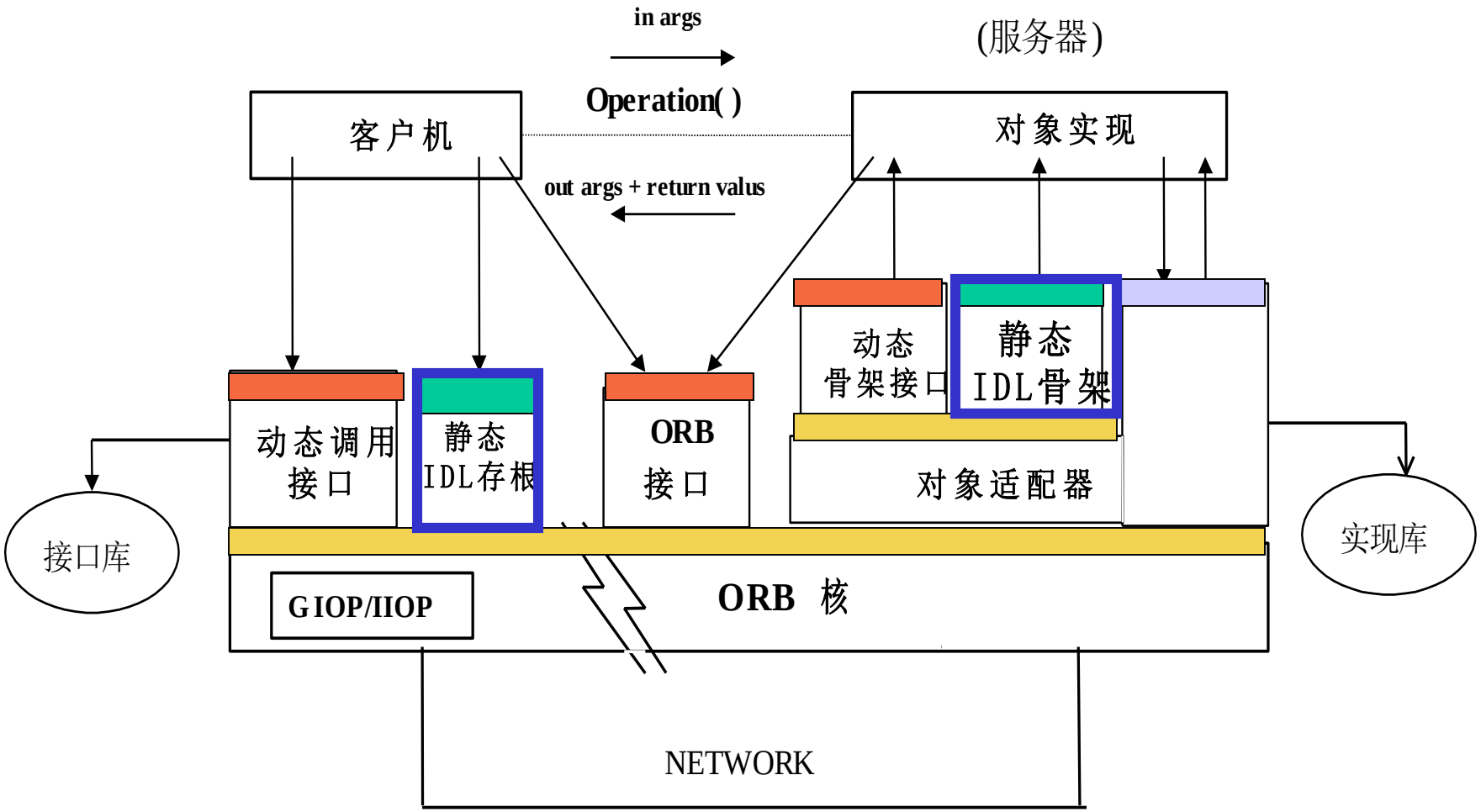
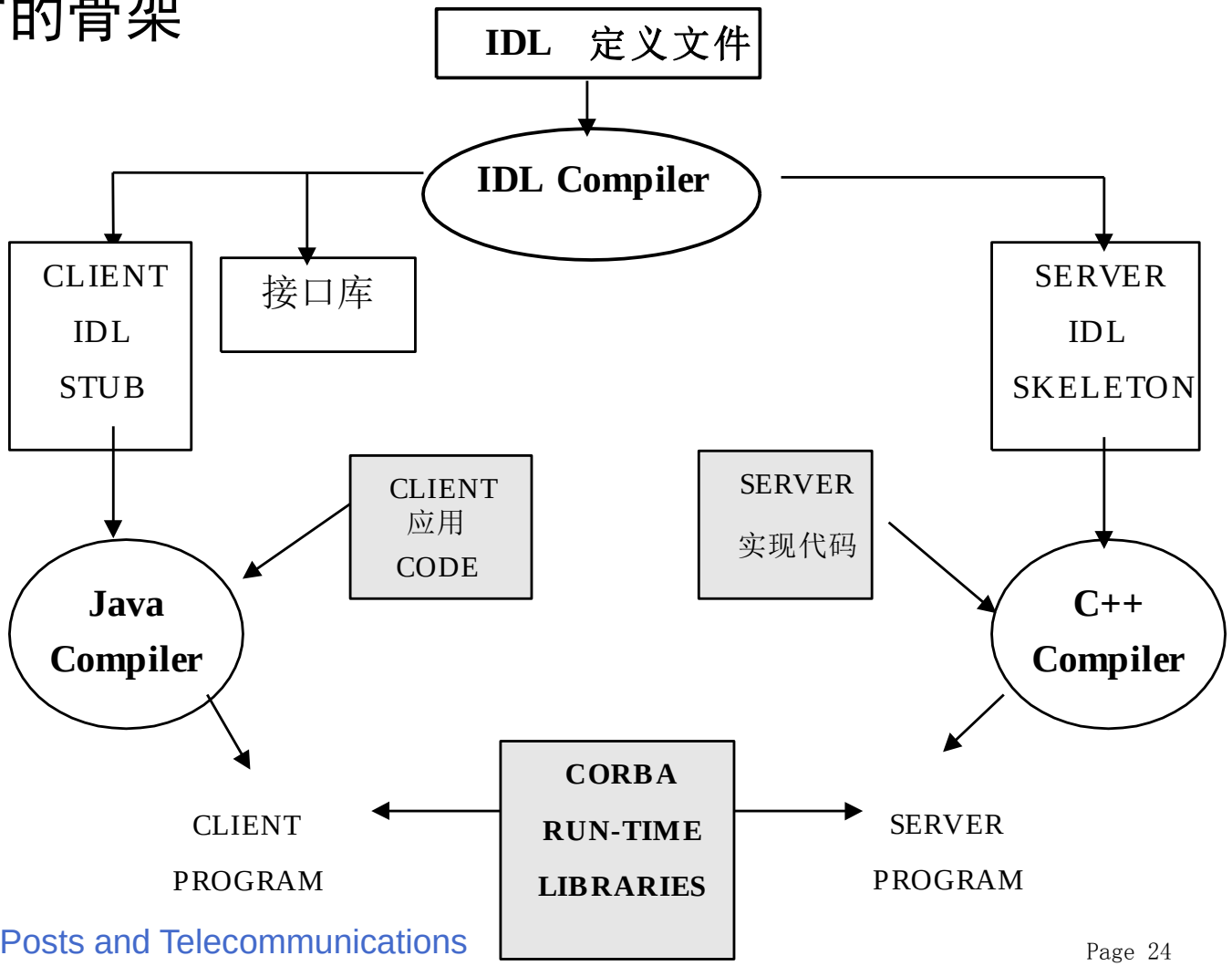


图3-3 CORBA ORB的体系结构

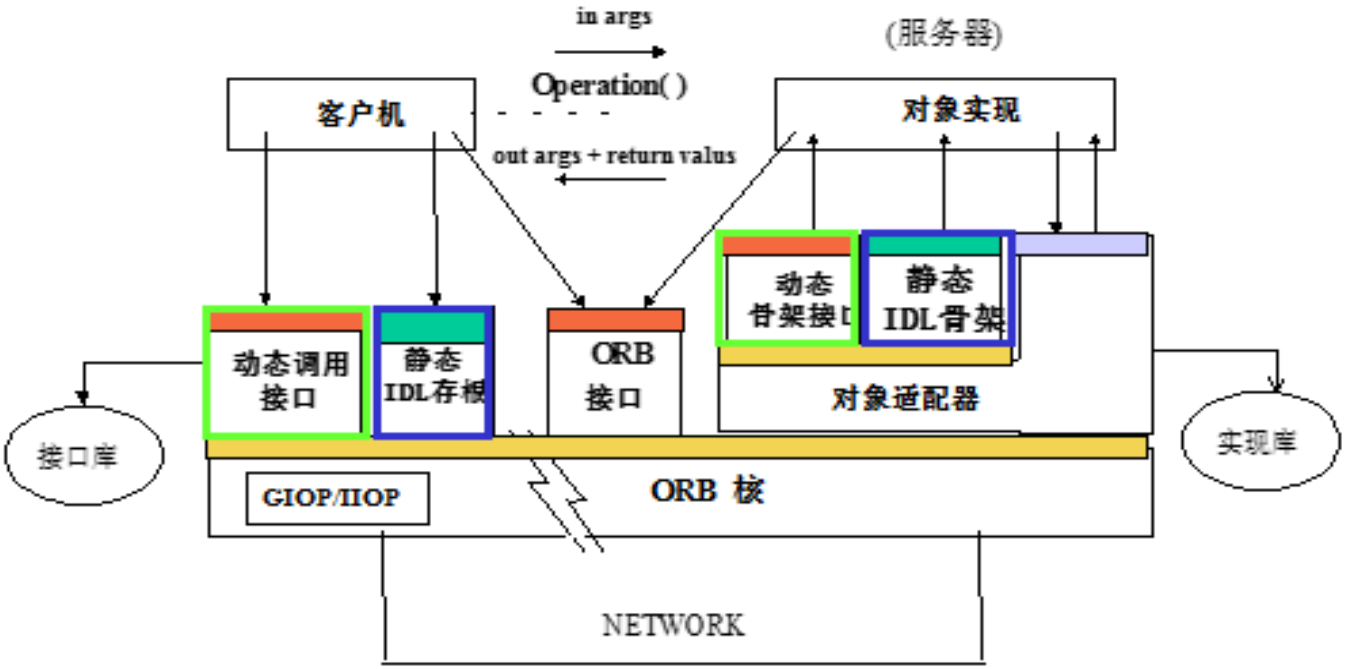
从IDL到存根和骨架

- OMG IDL编译器根据接口定义来产生客户方的存根和服务方的骨架



静态存根

- 静态IDL存根 (IDL stubs):
  - 编译时确定的的静态接口，位于客户对象本地, 对客户来说相当于远程的执行对象。由存根向ORB提交请求
  - 负责对请求参数的封装和发送，以及对返回结果的接收和解封装，并以适当的格式进行通信传输



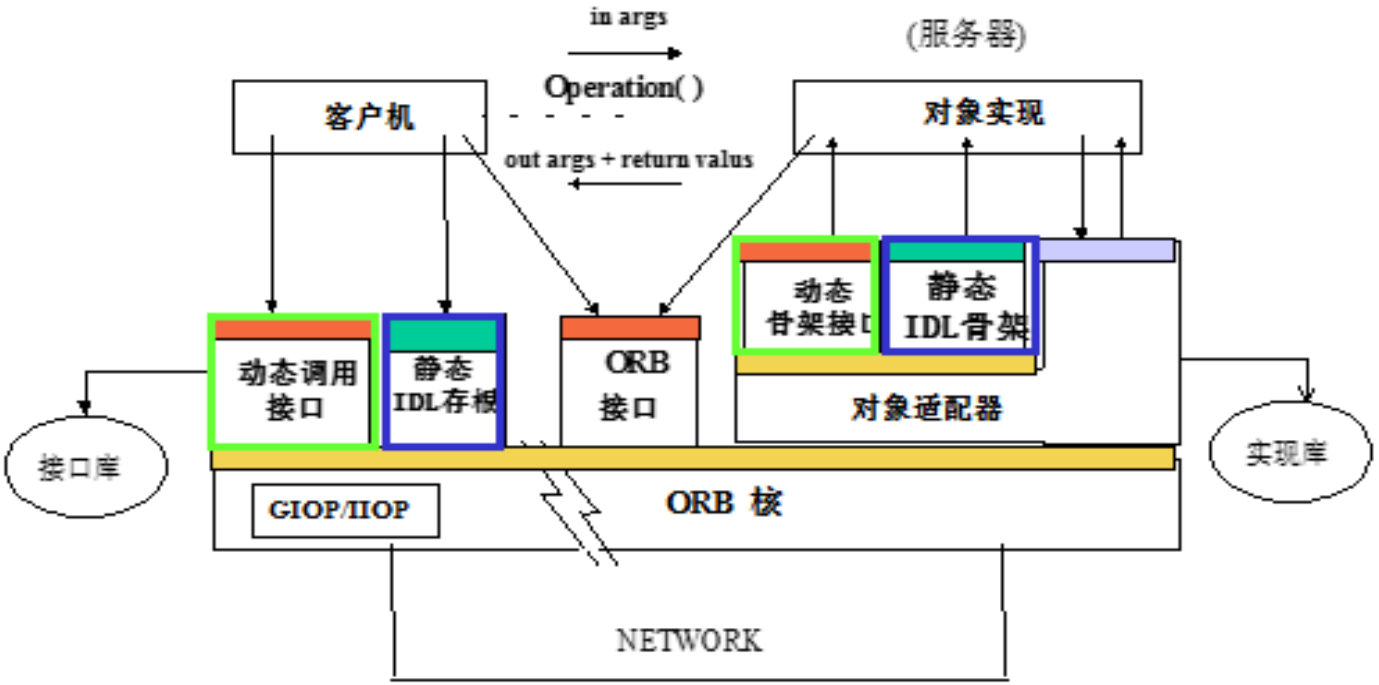
例： _HelloStub

```
public class _HelloStub
                extends org.omg.CORBA.portable.ObjectImpl
                 implements Hello {
    public void say_hello() {
          ……
         org.omg.CORBA.portable.OutputStream out = null;
         org.omg.CORBA.portable.InputStream in = null;
         ……
          out = _request("say_hello", true);
          in = _invoke(out);
          ……
     }
}
```

```
interface Hello
{
        void say_hello();
};
```

静态骨架

- 静态IDL骨架 (IDL skeletons)：
  - 在本地调用执行对象服务，并与ORB通信
  - 针对执行对象来说，代表了远程客户。骨架接收经ORB来的请求，将请求参数解封装，识别客户所请求的服务，（向上）调用服务器中的对象实现，把执行结果封装，并将结果返回给客户程序



例：HelloPOA类

- HelloPOA.java是Hello对象的服务端框架代码
- xxxPOA类的主要功能
  - 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

例： HelloPOA

```
public abstract class HelloPOA  extends org.omg.PortableServer.Servant
      implements org.omg.CORBA.portable.InvokeHandler,  HelloOperations {
      public org.omg.CORBA.portable.OutputStream
                _invoke(String opName,
                              org.omg.CORBA.portable.InputStream in,
                              org.omg.CORBA.portable.ResponseHandler handler) {
            …….
            return _OB_op_say_hello(in, handler);
       }
       private org.omg.CORBA.portable.OutputStream
                _OB_op_say_hello(org.omg.CORBA.portable.InputStream in,
                                    org.omg.CORBA.portable.ResponseHandler handler) {
            ……..
            sayHello();
            ……..
       }
}
```

ORB主要内容

- 接口
- 静态存根和骨架
- 对象实现和客户
- 对象引用
- ORB核心
- 对象适配器

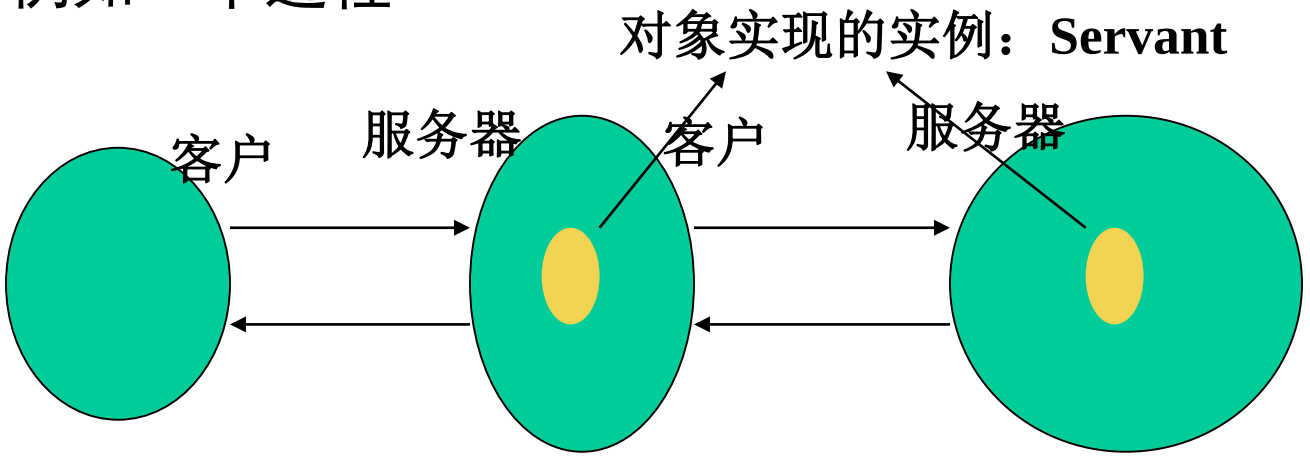
对象实现

- **对象实现（Object Implementation）**
  - 由应用的编程人员实现，通过为对象实例定义数据和为对象方法定义代码提供对象的语义
  - 对象可以具有不同的实现方式
  - 对象实现不依赖于ORB或者客户调用对象的方式
  - 一个对象的实现可以是其它对象的客户
  - 对象实现的实例Servant：译为伺服对象或者仆从

```
public class Hello_impl extends HelloPOA{
    public void    say_hello(){
                System.out.println("Hello World!");
    }
}
```

服务器 Server

- 服务器是一个（或一组）组件，能为其它组件提供某种服务。即，如果某个组件创建了一个对象，并能被其它组件通过对象引用来访问，则拥有对象的组件就是该对象的服务器，其它组件对这个对象的请求操作都将由创建该组件的服务器来执行
  - 例如一个进程



服务器程序例

```
public static void main(String args[]) {
    ……
    //创建对象实现的实例
    Hello_impl helloImpl = new Hello_impl();
    Hello hello = helloImpl._this(orb);
    ……
}
```

客户

- **客户（Client）**
  - 客户程序访问对象的对象引用，并且调用对象上的操作（方法）
  - 客户只知道对象的接口，通过调用感受对象的行为

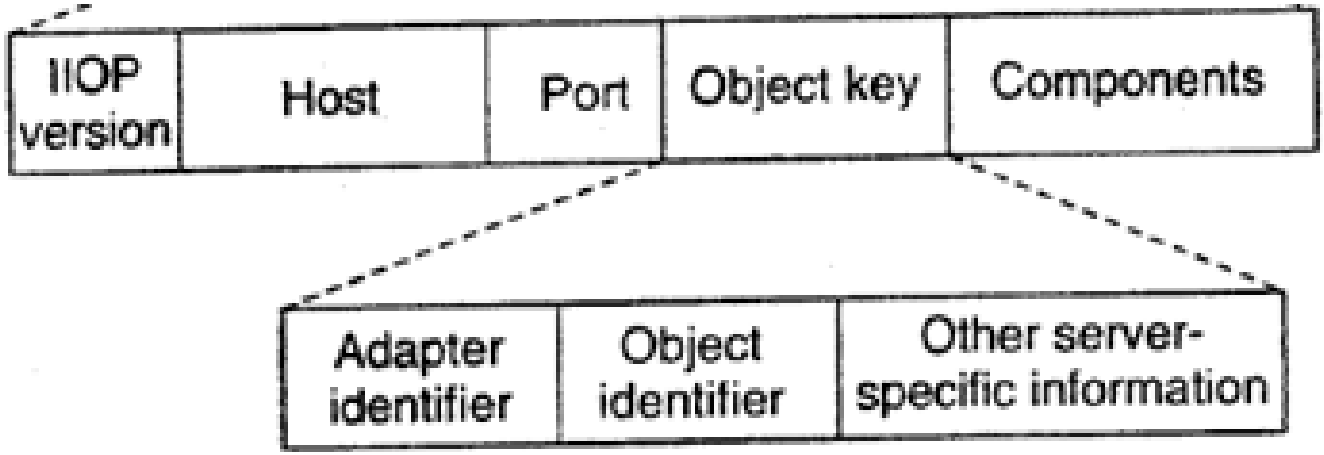
hello.say_hello();

ORB主要内容

- 接口
- 静态存根和骨架
- 对象实现和客户
- 对象引用
- ORB核心
- 对象适配器

对象引用

- 对象引用 — 临时的不透明的句柄，标识ORB中的一个对象实例。它用于定位响应请求的对象实现。
- 可互操作的对象引用IOR：在异构ORB间传递OR。



北京郵電大學

对象引用生成和发布例

```
public static void main(String args[]){
    ……
    Hello_impl helloImpl = new Hello_impl();
    Hello hello = helloImpl._this(orb);
    ……
    String ref = orb.object_to_string(hello); //将IOR转换为String
    //将IOR字符串写入文件
    String refFile = "Hello.ref";
    java.io.FileOutputStream file = new java.io.FileOutputStream(refFile);
    java.io.PrintWriter out =  new java.io.PrintWriter(file);
    out.println(ref);
    out.flush();
    file.close();
    ……
}
```

对象引用的获取

- 从命名服务或交易服务中获取
- 使用工厂对象
- 使用对象引用字符串。转换为字符串，在文件中、email等中传递。Object_to_string()和string_to_object();
- 特定ORB实现的方法
- 获取初始引用，如：CORBA::ORB::resolve_initial_references()
- 作为方法调用的返回值

对象引用获取和使用例

```
public static void main(String args[]){
    ……
   org.omg.CORBA.Object obj =
                    orb.string_to_object("relfile:/Hello.ref");

    Hello hello = HelloHelper.narrow(obj);

    hello.say_hello();
     ……
}
```

例： HelloHelper

```
final public class HelloHelper {
    public static Hello narrow(org.omg.CORBA.Object val)
    {
            ……..
            org.omg.CORBA.portable.ObjectImpl  _ob_impl;
            _HelloStub _ob_stub =  new _HelloStub();
            ……..
            return _ob_stub;
            ……..
     }
     ……
}
```

```
public class _HelloStub
            extends org.omg.CORBA.portable.ObjectImpl
             implements Hello {
  public void say_hello() {
      ……..
```

ORB主要内容

- 接口
- 静态存根和骨架
- 对象实现和客户
- 对象引用
- ORB核心
- 对象适配器

ORB核心

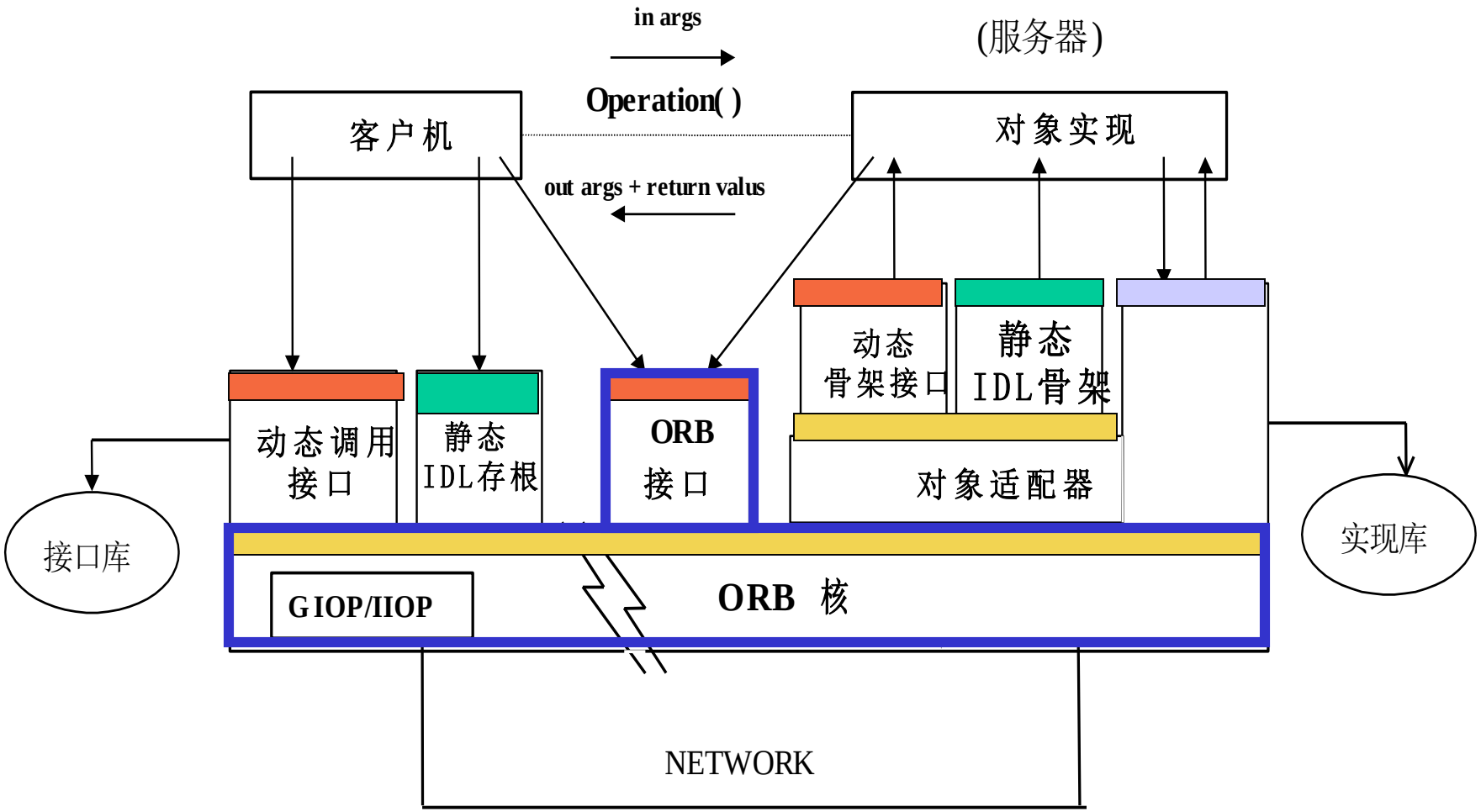


图3-3 CORBA ORB的体系结构

ORB核心及接口

- ORB内核提供了ORB最核心的功能，如ORB的启动、对象引用的解析、服务对象的定位、通信等。其接口同时为客户方和对象实现方所见，可直接为应用程序所使用：
  - 对象引用操作：提供对象引用的串化和反串化及对象引用复制、删除、比较及探测对象引用存在与否等操作。如object_to_string();release()
  - ORB和对象适配器初始化：使应用得到指向ORB的对象引用。如ORB_init()
  - 获取初始对象引用：ORB规定了客户对象在ORB初始时获取初始对象引用的方法， 如resolve_initial_references();

ORB主要内容

- 接口
- 静态存根和骨架
- 对象实现和客户
- 对象引用
- ORB核心
- 对象适配器

对象适配器

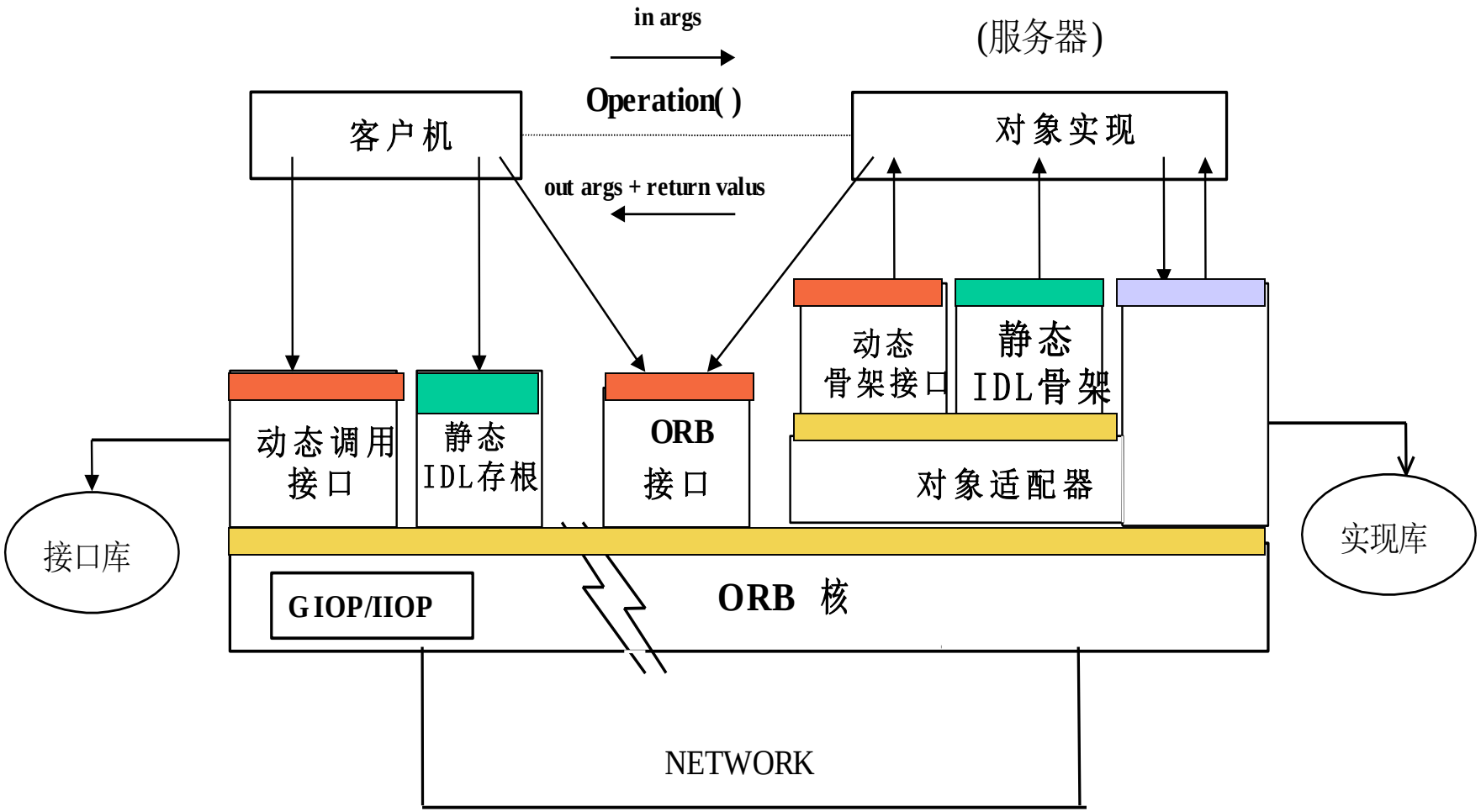


图3-3 CORBA ORB的体系结构

CORBA对象和伺服对象

- CORBA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
  - 从客户程序的角度看，就是它获取到的对象引用IOR。IOR中包含了对象的标识、接口类型及其他信息以查找对象实现

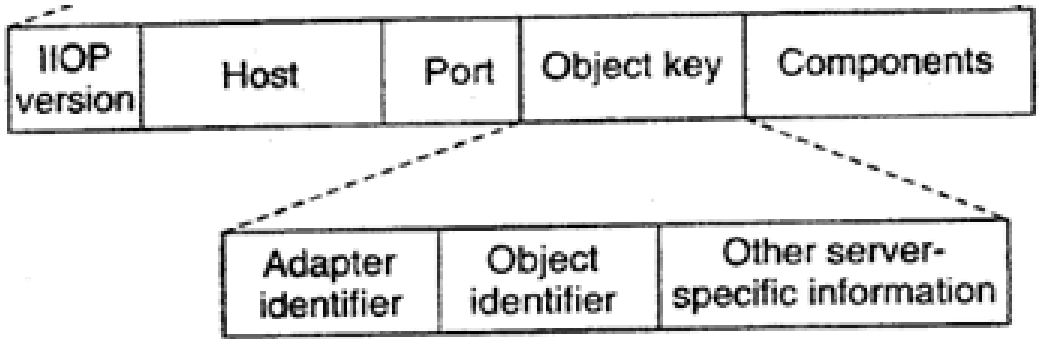
```
org.omg.CORBA.Object obj =
                        orb.string_to_object("relfile:/Hello.ref");
    Hello hello = HelloHelper.narrow(obj);
```

- 伺服对象（Servant）：
  - 指具体程序设计语言的对象实例或实体，通常存在于一个服务程序进程之中
  - Hello_impl helloImpl = new Hello_impl();

CORBA对象、伺服对象间的关联

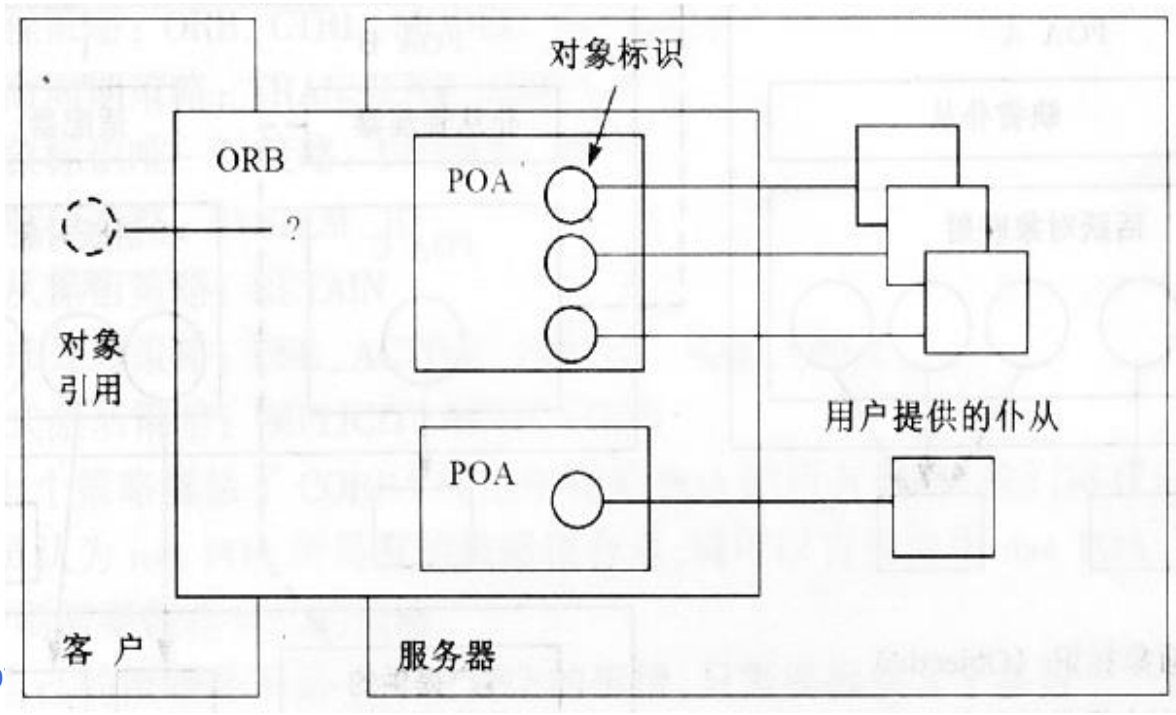
- 服务器利用对象实现创建伺服对象，然后将这些服务端本地的对象实例转换为可供远程使用的CORBA对象：
  - 对象引用＝宿主服务器地址端口+POA标识＋ObjectID
  - 例：CORBA

```
Hello_impl helloImpl = new Hello_impl();
Hello hello = helloImpl._this(orb);
```



对象适配器主要功能

- 对象适配器是管理服务端伺服对象（仆从）、对象标识、对象引用及它们之间关联的主要工具
  - 它负责决定在收到客户请求时应调用哪个伺服对象，然后调用该伺服对象上的合适操作
  - 例：POA： Portable Object Adapter
    - ➔CORBA对象（IOR）与伺服对象之间可以不是一一对应的关系



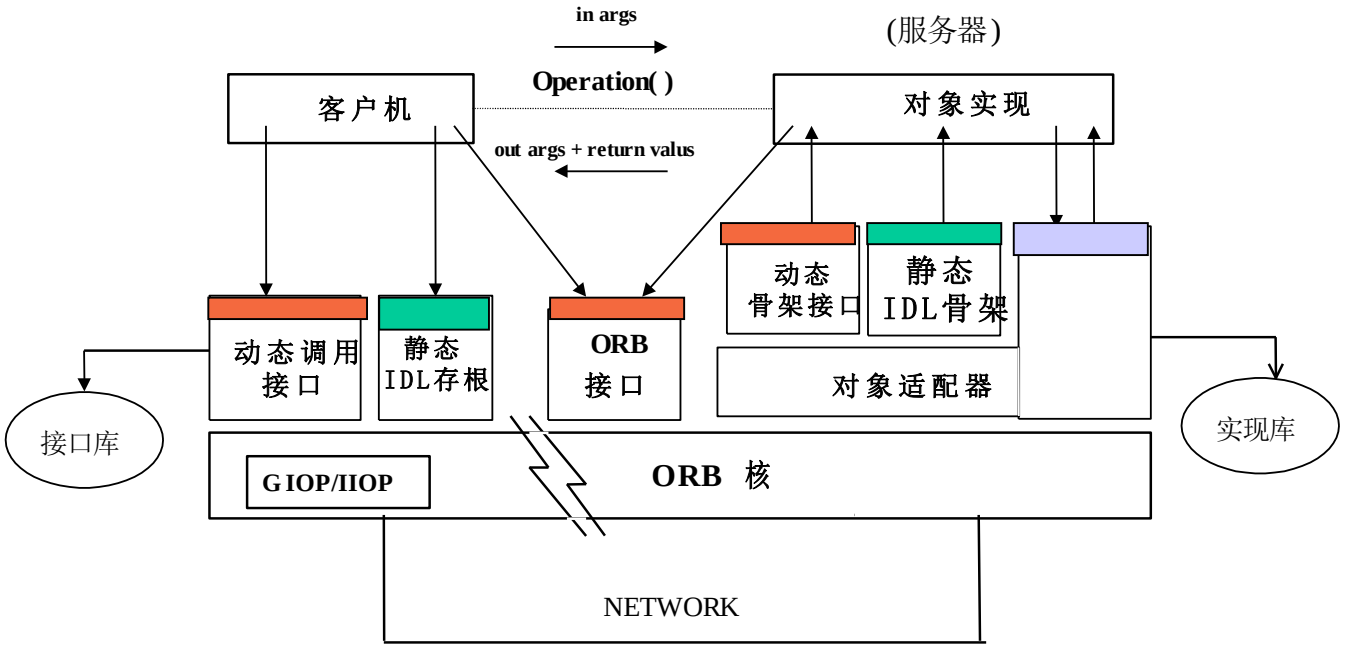
合在一起：Server

```
public static void main(String args[]) {
    ……
    orb = org.omg.CORBA.ORB.init(args,props); //初始化ORB
    //获取根POA
    org.omg.PortableServer.POA rootPOA = org.omg.PortableServer.POAHelper.narrow
                                        (orb.resolve_initial_references("RootPOA"));
    org.omg.PortableServer.POAManager manager =  rootPOA.the_POAManager();

    Hello_impl helloImpl = new Hello_impl(); //创建伺服对象
    Hello hello = helloImpl._this(orb); //生成相应的CORBA对象，IOR
    String ref = orb.object_to_string(hello); //将IOR转换为String
    ……  // 写入文件

    manager.activate();  //激活POA管理器，以允许接受请求
    orb.run();  //将控制权交给ORB，并处于运行状态
     ……
}
```

合在一起：Client

```
public static void main(String args[]) {
    ……
    orb = org.omg.CORBA.ORB.init(args,props); //初始化ORB
    ……
    //获取远端对象的引用
    org.omg.CORBA.Object obj = orb.string_to_object("relfile:/Hello.ref");
    Hello hello = HelloHelper.narrow(obj);
    ……
    hello.say_hello(); //调用远端对象的方法
   ……
   System.exit(status); //退出
}
```

ORB体系结构回顾



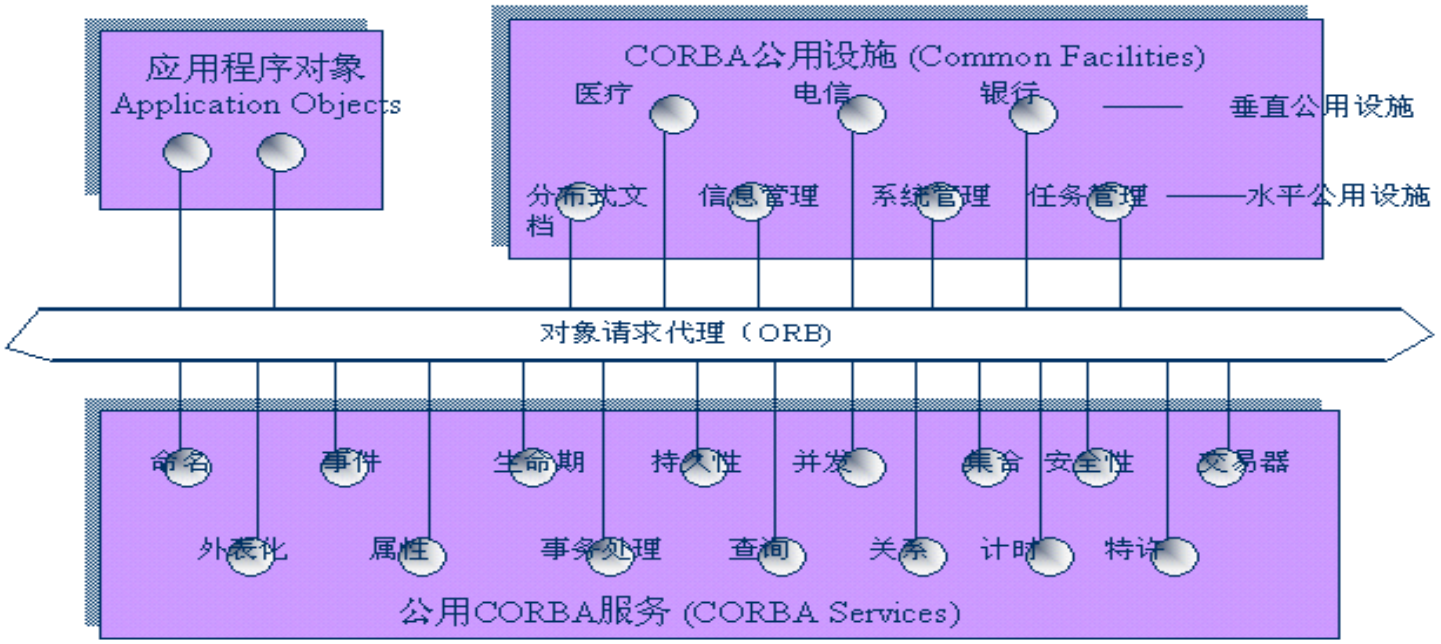
当Client要调用远程对象上的方法时，首先要得到这个远程对象的引用，之后就可以像调用本地对象方法一样调用远程对象的方法。当发出一个调用时，ORB通过客户Stub截取这个调用，对请求进行封装，将调用的名字、参数等编码成标准的方式(称Marshaling，编组)，并将请求发送给ORB核心。ORB核心解析IOR获得服务器IP地址端口号，将请求发送到服务器端。服务器方ORB核心收到请求，基于IOR中的POA标识，将请求送到相应POA，该POA基于IOR中的对象标识，送到相应的骨架对象（可能与伺服对象合为一个对象，因伺服对象类可通过继承骨架类实现）。骨架对象将参数解封装（Unmarshaling，解组），以本地对象方法调用的方式调用伺服对象相应业务方法从而完成调用。服务器对象完成处理后，ORB通过同样的编组/解组方式将结果返回给客户。

主要内容

- 对象管理体系结构OMA
- **OMG的接口定义语言IDL**
- **对象请求代理ORB**
- CORBA服务
- ORB之间的互操作
- CORBA中的基本原理

CORBA服务

- 公共对象服务：系统级的对象框架。扩展了基本的CORBA体系结构
- CORBA 服务代表了一组预先实现的，应用软件开发商通常需要的分布式对象



CORBA服务清单例

- **命名服务（Naming Service）：**为客户程序通过名字查找对象实现提供支持。
- **事件服务（Event Service）：**使对象间的通信能够以松耦合的方式进行。
- **生命周期服务（Life Cycle Service）：**为对象的创建、删除、拷贝和移动提供支持。
- **持久状态服务（Persistent State Service）：**为对象持久状态的维护和管理提供支持。
- **事务服务（Transaction Service）：**为事务处理提供支持。
- **并发服务（Concurrency Service）：**为多个客户协调一致地同时访问共享资源提供支持。

CORBA服务清单例（2）

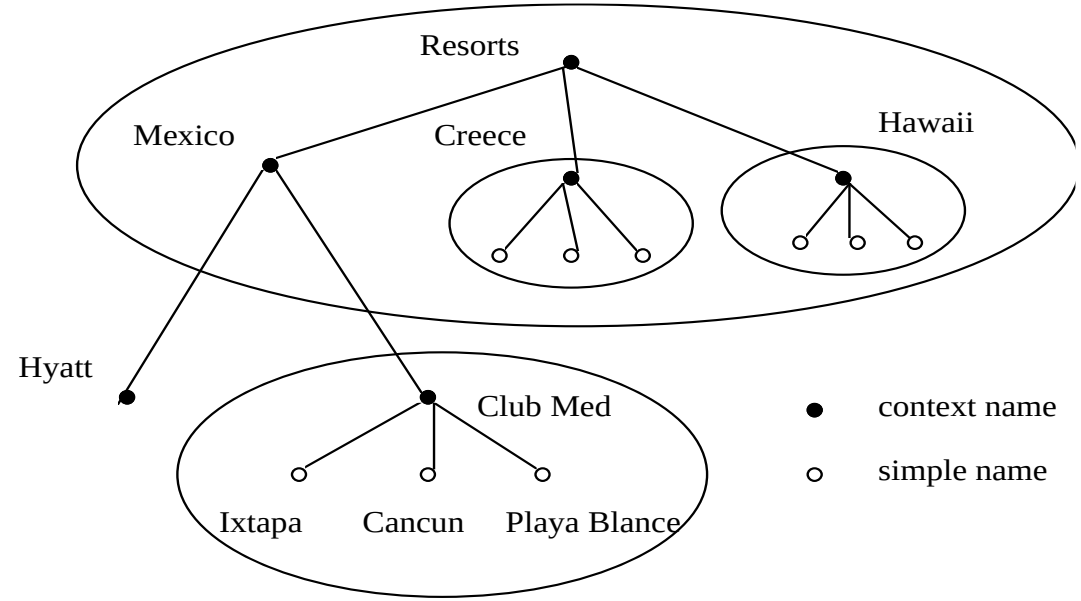
- **关系服务（Relationship Service）：**为实体和它们之间地关系的外在表示提供支持。
- **外部服务（Externalization Service）：**为对象的外部化（externalizing）和 内在化（internalizing）提供支持.
- **查询服务（Query Service）：**:为客户在对象集合中查找满足某种查询标准的对象提供支持。
- **许可证服务（Licensing Service）：**为软件许可证的管理提供支持。
- **属性服务（Property Service）：**为对象属性的定义和操纵提供支持。
- **时间服务（Time Service）：**为客户获取当前时间以及可能的误差提供支持。
- **安全服务（Security Service）：**为基于CORBA的整个系统的安全提供支持。

CORBA服务清单例（3）

- **交易对象服务（Trading Object Service）**：为提供和查找具有特定属性的对象提供支。
- **集合服务（Collections Service）**：为创建以及操纵各种常用集合类型提供支持。
- **通知服务（Notification Service）**：对事件服务的扩展，支持事件的类型定义、过滤条件的设定、服务质量的设定等。
- **增强的时间视图服务（Enhanced View of Time Service）**：为一致地使用具有各种自身特点的时钟提供支持。

命名服务

- 目的：为通过名字查找对象提供支持
- 方法：定义了一系列的接口，使客户可以管理对象的命名空间、在命名空间中查找对象



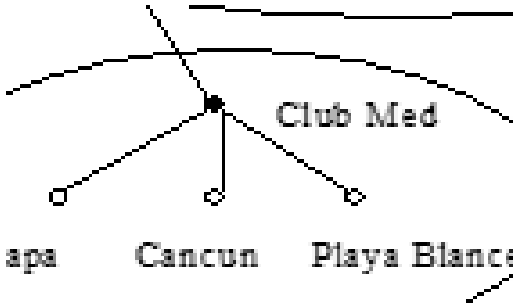
Compound Name:

Resorts	Mexico	Club Med	Playa Blance

CORBA Objects Support Hierachical Naming

命名服务的使用

- 使用**NamingContext**接口提供的API，可以：
  - 将一个对象与一个名字绑定：
    - ➔void bind(in Name n, in Object obj);
  - 在一个命名空间中，删除一个对象绑定：
    - ➔void unbind (in Name n);
  - 通过名字查找一个对象：
    - ➔Object resolve (in Name n);
  - 创建一个命名空间：
    - ➔NamingContext bind_new_context(in Name n);
  - 等等.
- 使用**BindingIterator**接口, 可以在一个给定的命名空间中漫游（Navigate）



CORBA服务小结

- OMG为每个服务定义了标准API
  - 通过IDL定义
- 实现厂商提供支持这些API的具体产品，它们的实现方法可能不同，但接口是一致的
- 用户购买、部署需要的服务产品
  - 买源代码或者可执行代码
  - 安装并在需要的时候实例化
- 在程序中使用时，首先获取服务中相应对象的IOR，然后调用它的操作即可

主要内容

- 对象管理体系结构OMA
- **OMG的接口定义语言IDL**
- **对象请求代理ORB**
- CORBA服务
- ORB之间的互操作
- CORBA中的基本原理

GIOP、IIOP和ESIOP

- 为了不同的ORB间可以互操作，有必要制订传输请求的统一标准，规定传输底层的数据表示方法与消息格式，
  - GIOP： 通用ORB间协议，定义了传送语法和消息格式的标准集，能够在任何面向连接的传输上实现ORB间的互操作性
  - IIOP： Internet ORB间协议，定义了如何在TCP/IP传输上构建GIOP。GIOP和IIOP之间的关系类似于接口定义及其实现
  - ESIOP：能够针对已使用特定分布式计算基础结构的特殊情况（例如DCE-CIOP）构建ORB。

GIOP的基本组成部分

- **The Common Data Representation (CDR) definition：**通用数据表示定义，它实际上是IDL数据类型在网上传输时的编码方案。它对所有IDL数据类型的映射都作了规定
- **GIOP Message Formats：**规定了Client和Server两个角色之间要传输的消息格式。7种消息格式：Request、Reply、CancelRequest、LocateRequest、LocateReply、CloseConnection、MessageError、Fragment
- **GIOP Transport Assumptions:**主要规定在任何面向连接的网络传输层上的一些操作规则。如：传输应该是面向连接的；传输应是可靠的；在非正常情况下断开连接时，传输应能提供某种通知机制等

主要内容

- 对象管理体系结构OMA
- **OMG的接口定义语言IDL**
- **对象请求代理ORB**
- CORBA服务
- ORB之间的互操作
- CORBA中的基本原理

CORBA中的基本原理（1）

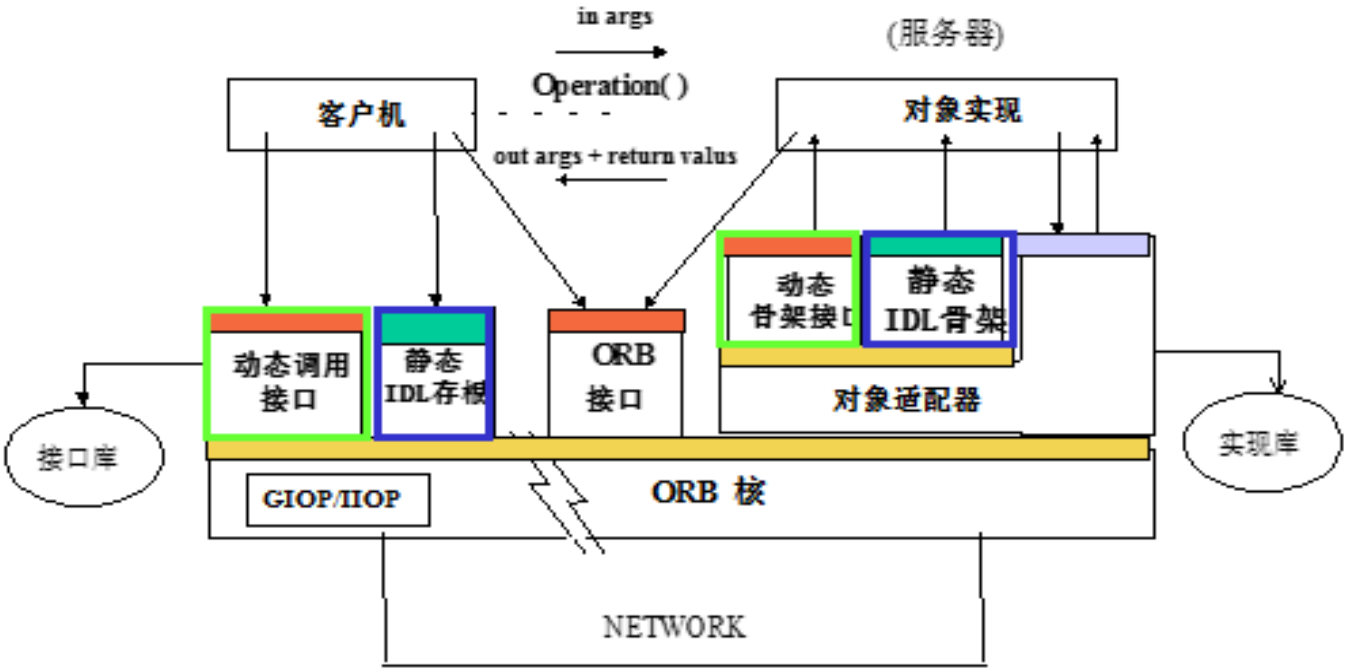
- 体系结构：OMA
  - 面向对象体系结构
  - 是否采用分层、以数据为中心等体系结构，取决于最终基于CORBA实现的应用系统
- 进程和线程
  - 客户端和服务器端均支持多线程
  - 通过客户端和服务器端CORBA环境的协作，支持多种分布透明性
  - 主要使用并发服务器的方式支持并发请求：多线程
  - 客户端如何找到服务器：使用IOR，其中封装了地址，客户端ORB基于该信息找到服务器
  - 服务器端状态问题主要取决于具体实现：常有状态

CORBA中的基本原理（2）

- 通信：
  - 远程方法调用：同步调用
  - GIOP、IIOP （中间件协议）
  - 事件、消息通信：相应的CORBA服务
- 命名：
  - CORBA命名服务：采用结构化命名方法
- 复制、容错、安全：
  - CORBA对对象组提供了一定的支持，复制也可以在应用级进行实现
  - CORBA有相应的容错和安全服务，供应用开发者开发相应的容错和安全功能

CORBA 与分布透明性

- 通过客户方的存根和服务方的骨架提供访问透明性
- 通过ORB核心解析对象引用提供位置透明性



CORBA结束语

- ORB的体系结构：为什么在异构环境下能够透明地发送请求和接收响应
- CORBA服务：初步了解CORBA服务，以及可以怎样使用它们

本章结语

- 先进的分布式软件体系结构与面向对象技术结合在一起，可分享面向对象技术带来的众多好处
- 在分布式对象系统中，对象的一些特性发生了变化
  - 封装性：各种透明性
  - 继承：接口继承
  - 对象引用：增加了远端对象的寻址信息
- 实现分布式对象模型的基本支撑机制
  - 远程调用
  - 通用服务

本章结语（续1）

- CORBA： OMG组织制定的一个工业规范，是一个体系结构和一组规范
- 提供一个框架，如果符合这一框架，就可以在主要的硬件平台和操作系统上建立一个异质的分布式应用
  - OMA
  - ORB
  - CORBA服务：命名服务
  - CORBA设施
  - …….