物联网中间件

第4章 分布式面向对象组件 CORBA

参考材料

- CORBA
  - 课本 8.1， 8.2.1

早期的中间件

- 早期的中间件比如sockets 以及 ONC‘s Remote Procedure Calls等等，都是面向函数（functionally oriented）的。它们都是在面向对象范式（object-oriented paradigm）在工业界大规模应用之前出现的。

- Sun ONC RPCs 与C语言以及UNIX系统高度捆绑，不支持C++、Java等面向对象语言

面向对象编程兴起

- 上世纪八十年代末，面向对象编程兴起，因此也产生了调用远程对象（Remote Object）的需求。

使用分布式面向对象组件的中间件(Middleware using Distributed Object-Oriented Components)

- Common Object Request Broker Architecture(CORBA)
- .Net Remoting
- Java RMI
- Enterprise Jave Bean

CORBA

- Common Object Requst Broker Architecture(CORBA) 公共对象代理体系结构
  - Object Management Group（OMG）协会与1989年提出
    - 1991发布CORBA 1.0
    - 1997年发布CORBA 2.0
    - 2002年发布CORBA 3.0
  - OMG是一个有300家公司参与的行业协会，几乎包含了当时所以的相关企业，UML也是协会的成果之一
  - 特点：Do everything for everybody
  - 上世纪90年代末最主流的中间件

CORBA 的兴衰

- 前沿风险技术->最受欢迎的中间件->默默无闻的小众技术（1991-2006）

- 目前只在大公司内网，以及嵌入式领域还有部分应用

CORBA 的衰落

- CORBA失败的原因：
  - 防火墙的兴起，CORBA无法穿越防火墙
  - 太复杂，以至于企业无法招聘到足够的训练有素的CORBA程序员
  - 参见《The Rise and Fall of CORBA - MICHI HENNING》， June 30, 2006, ACM Queue.

CORBA的设计理念

- 为所有的编程语言/操作系统/计算机硬件组合提供连接
  - 举例：运行在Linux上用JAVA写的程序可以使用CORBA连接运行在Windows上用C++写的程序

- Location transparency（位置透明）：不需要关心被调用的对象的真实位置

CORBA的基本流程

- Client/Server模式（与peer-to-peer，publish-and-subscribe模式相对来说）
- CORBA在客户端使用一个代理对象（proxy object)，这个代理对象就代表远程对象。
- 当客户端调用代理对象中的一个方法时，代理对象将参数序列化并使用General Inter-Orb Protocol（GIOP）协议传输到服务器，GIOP协议在TCP/IP之上
- 服务器接受到消息后，调用合适的对象，并将参数传递过去，计算完成 后，将结果使用GIOP返回给客户端。

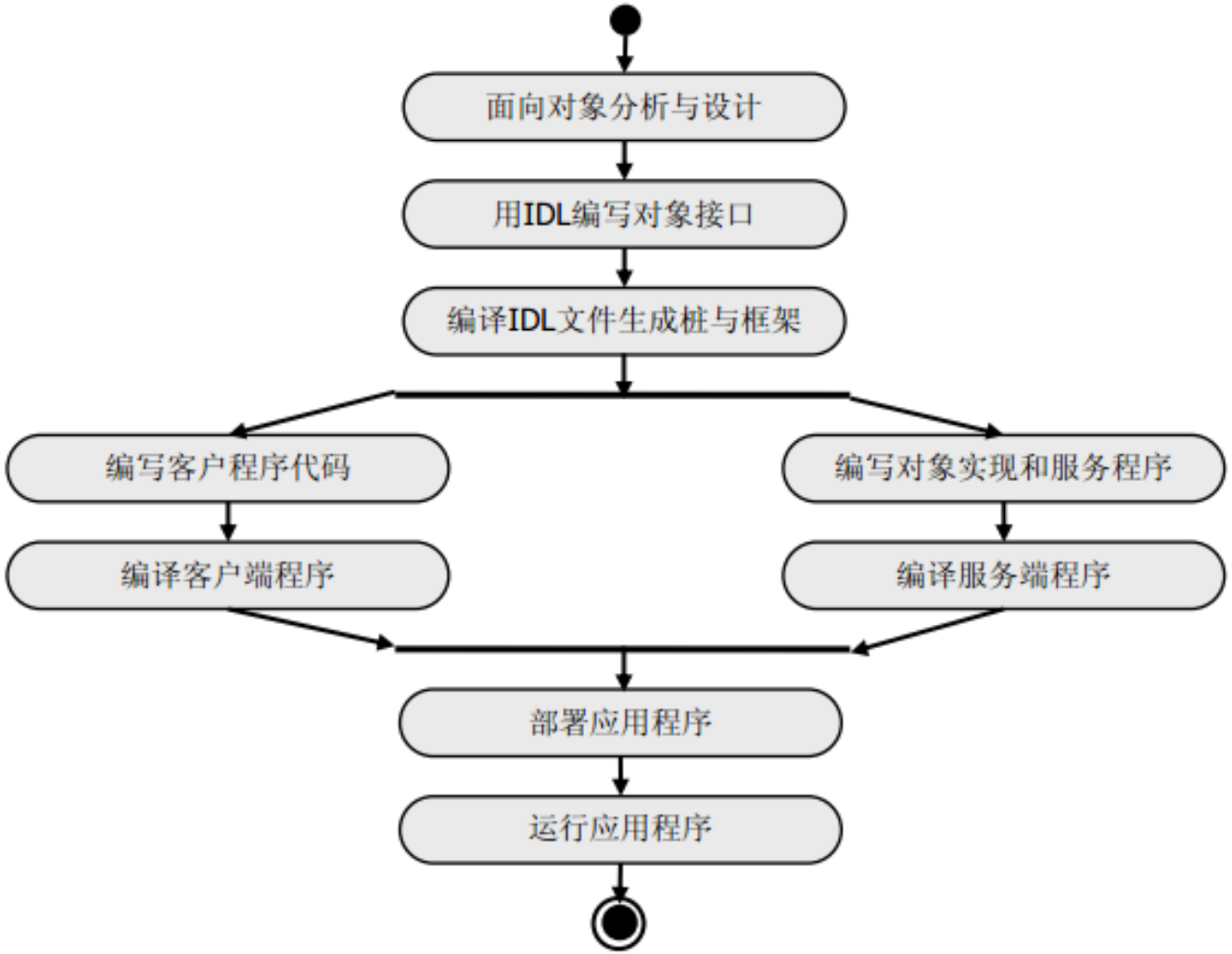
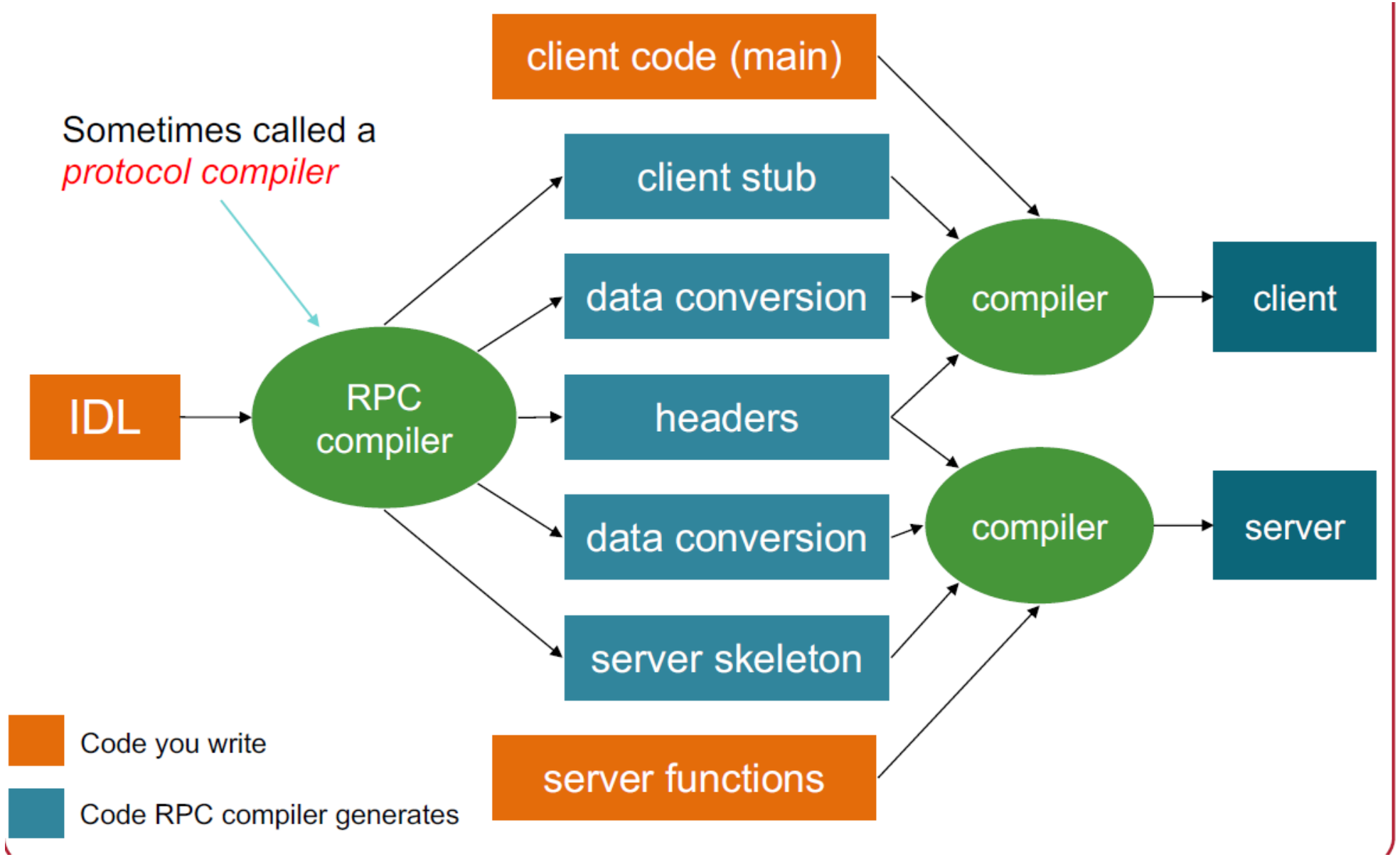


图 3-1 CORBA 应用的典型开发过程


client code (main)
Sometimes called a
protocol compiler
client stub
data conversion
compiler
client
IDL
RPC
compiler
headers
compiler
server
data conversion
server skeleton
server functions
Code you write
Code RPC compiler generates

CORBA中的接口

- Static interface （静态接口）：
  - 在编译时定义
- Dynamic interface （动态接口）：
  - 运行时使用

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，并称为语言映射(Language Mapping)。

例子

```
interface echo {
    string echostring (in string the_echostring);
};
```

CORBA的基本概念

- ORB：Object Request Broker, 对象请求代理。
- 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。
- 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

- CORBA对象： CORBA Object, 是一个“虚拟”的实体，可以由对象请求代理(ORB)定位，并且可以被客户程序请求调用。(客户机 stub)

- 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。 （服务器 skeleton）

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

- 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。

- 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。

伺服程序（Servant）

- 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。

可互操作的对象引用（Interoperable Object Reference IOR）

- 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。

IOR结构

- 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，根据所给出的主机名、端口和密钥组合来识别目标对象。

CORBA对象请求代理(ORB)间协议：

- **GIOP**（General Inter-ORB Protocol）
  - GIOP（通用对象请求代理间通信协议）提供了一个标准传输语法（低层数据表示方法）和ORB之间通信的信息格式集。
  - GIOP只能用在ORB与ORB之间，而且，只能在符合理想条件的面向连接传输协议中使用。它不需要使用更高一层的RPC机制。
  - 这个协议是简单的（尽可能简单，但不是简单化），可升级的，使用方便。
  - 它被设计为可移动的、高效能的表现、较少依靠其它的低层传输协议。当然，由于不同传输使用不同版本的GIOP，它们可能不能直接协作工作，但它能很容易的连接网络域。

CORBA对象请求代理(ORB)间协议：

- **IIOP** （Internet Inter-ORB Protocol）
  - IIOP（Internet对象代理间通信协议) 元件指出如何通过TCP/IP连接交换GIOP信息。
  - IIOP为Internet提供了一个标准的协作工作协议，它使兼容的ORB能基于现在流行的协议和产品进行“out of the box”方式的协作工作。
  - 该协议能用于任何ORB与IP（Internet Protocol）域之间的协作工作，除非ORB选择了特殊的协议。这时，它是TCP/IP环境下基本的inter-ORB 协议，最普遍的传输层。

- GIOP 不基于任何特别的网络协议，OMG 在最广泛使用的通信传输平台 -- TCP/IP 上标准化 GIOP，GIOP 加 TCP/IP 等于 IIOP

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（servant）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主要作用

- 对象引用的生成与解释;
- 对象实现的注册;
- 根据对象引用找到它对应的对象实现（定位）;
- 服务器进程的激活;
- 对象的激活与撤销;

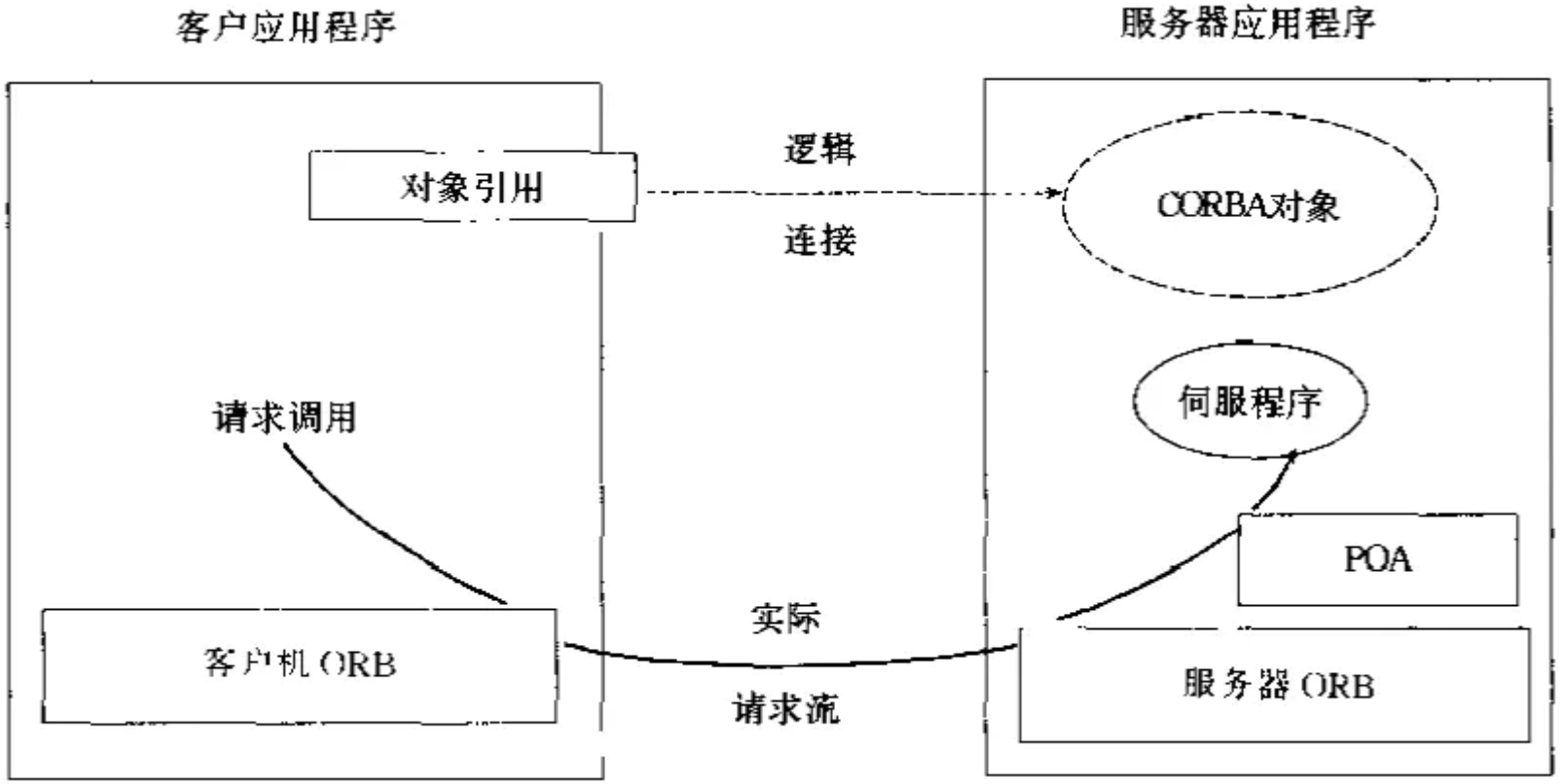
服务器端伪代码

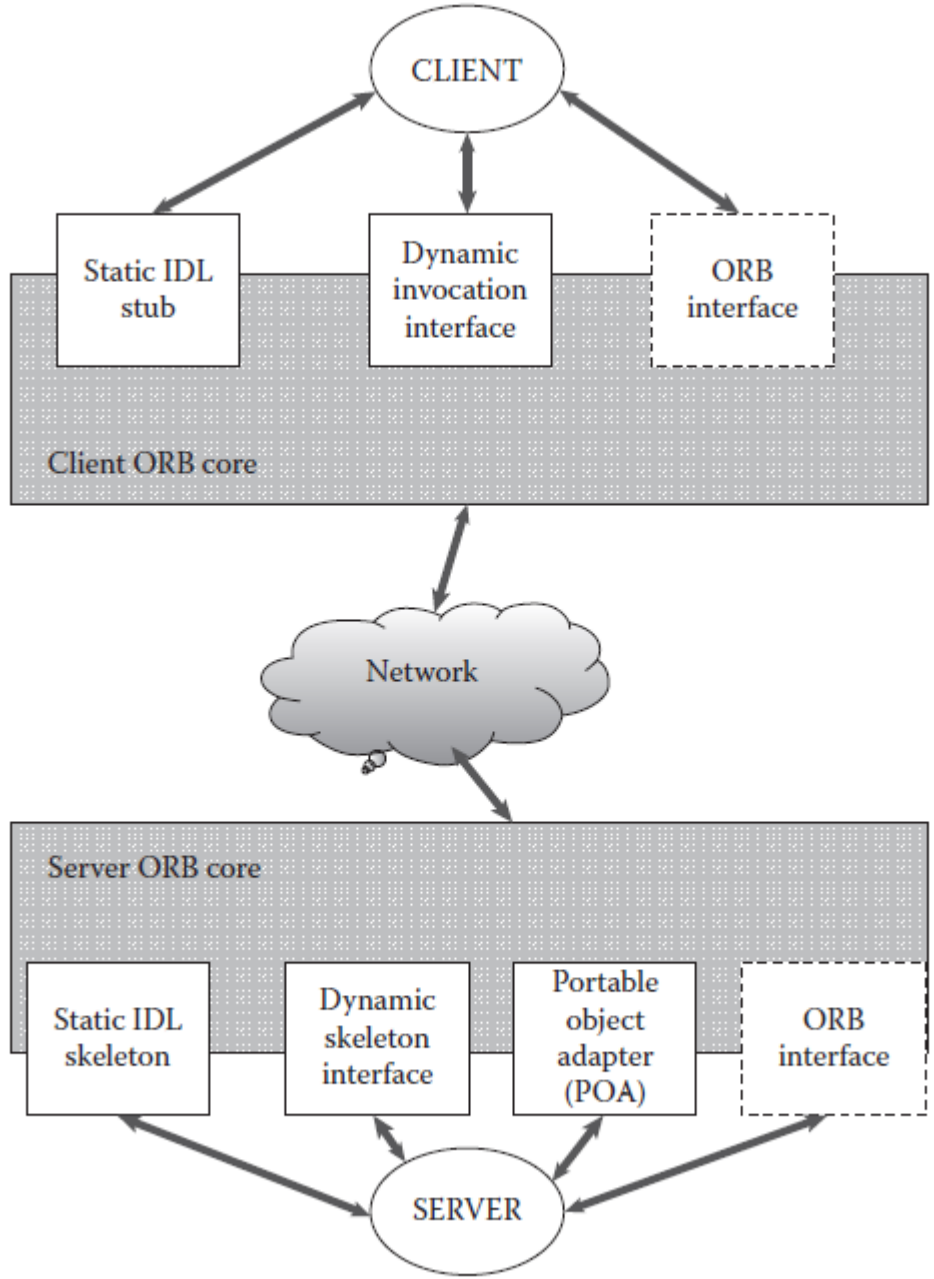
- 1.初始化ORB
- 2. 得到一个指向POA的引用
- 3. 创建一个实现CORBA接口的servant对象
- 4.创建一个CORBA对象并将其与之前创建的servant对象连接在一起
- 5.得到CORBA对象的引用（地址）
- 6.把CORBA对象的引用（地址）存到一个.ior文件中
- 7. 服务器开始监听

客户端伪代码

- 1. 初始化ORB
- 2. 取出存在.ior文件中的CORBA对象引用（这个CORBA对象代表服务器端的接口）
- 3.使用CORBA对象的引用，创建一个代理对象，用这个代理对象来代表客户端的接口
- 4. 使用代理对象调用远程过程

CORBA调用流程


客户应用程序
服务器应用程序
对象引用
逻辑
连接
CORBA对象
伺服程序
请求调用
POA
实际
客户机 ORB
服务器 ORB
请求流


CLIENT
Static IDL stub
Dynamic invocation interface
ORB interface
Client ORB core
Network
Server ORB core
Static IDL skeleton
Dynamic skeleton interface
Portable object adapter (POA)
ORB interface
SERVER

- 客户端调用静态存根*(static stubs)*向服务器发出请求
- 服务器端调用静态框架*(static skeleton)*处理客户端请求

存根（stubs）

- 存根本质上是一段程序代码

- 存根的实际工作包括接收客户程序的请求，对请求的参数进行封装和发送，以及对返回结果的接收和解封装。

存根特点

- 存根是自动生成的，不需要程序员参与。由ORB厂商提供的IDL编译器根据IDL定义生成相应的客户端存根。

- 存根是静态的，一经生成便不再改变，除非修改相应的IDL并重新编译生成。

- 存根与ORB的具体实现相关。不同的ORB厂商会有不同的ORB实现方式，即使相同的IDL定义也可能生成不同的存根。

框架(skeleton)

- 框架实际上是一段程序代码
- 框架负责对客户的请求进行解码，定位所请求的对象的方法，执行该方法，并把执行结果编码后发送回客户。
- 从作用上讲，框架就像一个位于服务器端的存根

框架特点

- 框架是自动生成的，不需要程序员手工书写。由ORB厂商提供的IDL编译器根据IDL定义，生成相应的服务器端框架。每个IDL定义都会有自己的框架。

- 框架是静态的，一经生成便不再改变，除非修改相应的IDL并重新生成。

- 框架与ORB的具体实现相关。不同的ORB厂商会有不同的ORB实现方式，即使相同的IDL定义也可能生成不同的框架。

静态调用

- 通过存根和框架的调用被通称为静态调用。
- 存根和框架都是从用户的IDL定义编译而来
- 在请求真实发生之前，存根和框架早已分别被直接连接到客户程序和对象实现中去。
- 存根和框架分别充当客户应用程序和服务器应用程序与ORB之间的粘合剂

假设有一个国王，身边有一群所谓的学者，如哲学家、数学家、神学家等。实际上他们并没有什么知识，唯一的法宝就是每人都有一张神奇的“名片”，能帮助他们找到真正的答案。

有一天国王突然对哲学发生了兴趣，他找来最赏识的哲学家，向他提了一个问题（①）。

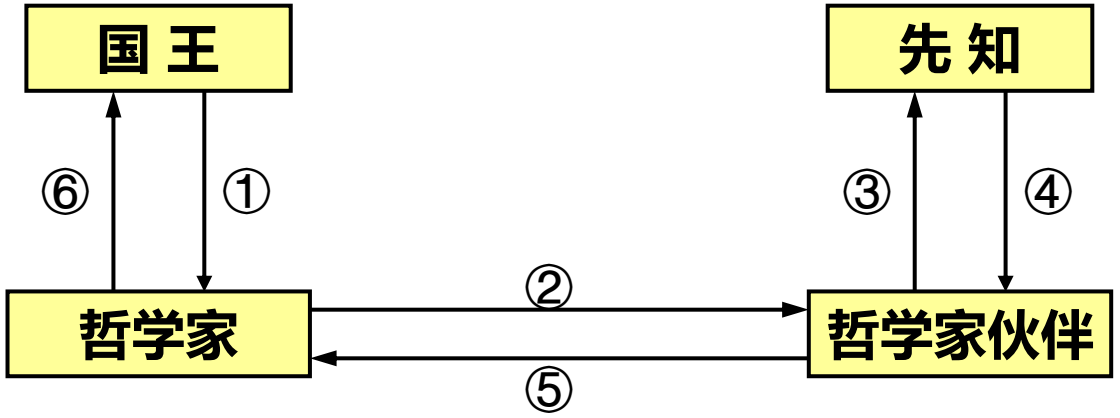
哲学家对这个问题一窍不通，于是他按照名片上的信息打了一个电话，电话通往一个遥远的国度（②）。

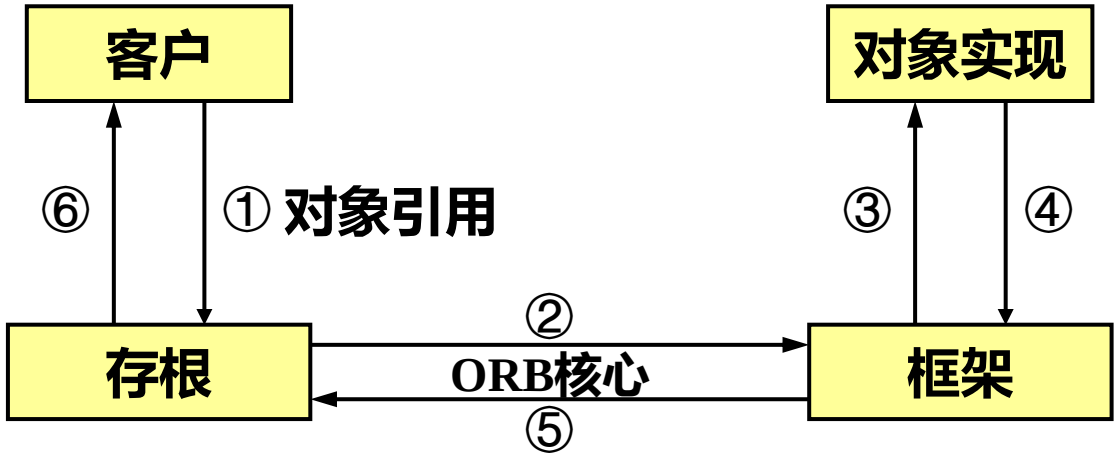
接电话的是一个哲学家的伙伴，其实他什么也不懂，但它是一位真正“先知”的私人代表，他把这个问题转达给了他的主人——那位隐居的“先知”（③）。

那位先知思考后，对这个问题作了详尽的回答（④）。

哲学家的伙伴对着话筒转达了全部的内容（⑤）。

第二天，哲学家对国王宣布了这个答案（⑥）。





国 王：客户

先 知：对象实现

哲学家：存根

名 片：对象引用

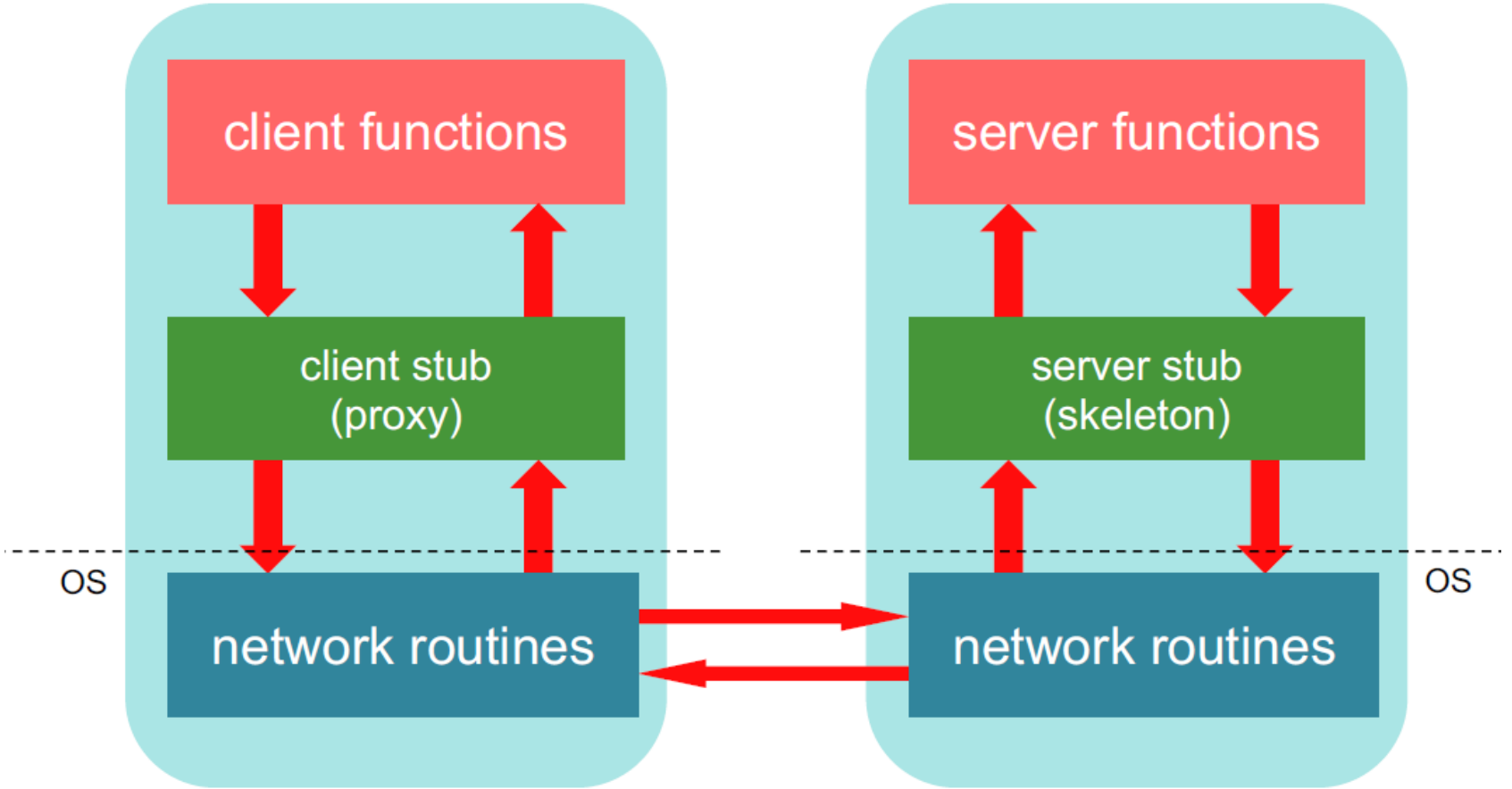
哲学家伙伴：框架

电话机：ORB核心

静态调用过程

- 客户通过代理对象向静态存根发送请求，静态存根负责对请求参数的封装和发送；
- 经封装的参数发送到ORB核心后，ORB核心负责请求的传送，将其送给相应的服务器端对象适配器；
- 服务器端的对象适配器接到请求后，通过静态框架将请求参数解封装，识别客户所请求的服务，并调用执行对象实现中的操作；
- 对象实现的特定操作方法执行完成后，结果将按照对象请求传递和执行路径逆向返回给客户对象

RPC流程


client functions
client stub
(proxy)
server functions
server stub
(skeleton)
OS
OS
network routines
network routines

RPC流程

- 1.客户端调用本地桩函数，桩函数打包参数
- 2.客户端桩函数调用系统底层通信接口发送打包好的网络消息
- 3.消息传送到远程主机
- 4.服务器桩函数得到消息并解包参数
- 5.服务器桩函数**根据调用请求以及参数执行远程过程（服务）**

RPC流程

- 6.执行过程完毕，将结果返回服务器桩函数
- 7.服务器桩函数打包结果，调用远程主机的系统网络服务发送结果
- 8.消息传回客户端上的桩函数
- 9.客户端桩函数调用底层网络服务接收消息
- 10.客户端桩函数返回调用结果

演示

史上最烂的项目

- **苦撑12年，600 万行代码，启动15分钟！史上最烂的项目，把负责人送进监狱！**
  - https://zhuanlan.zhihu.com/p/38973085

- 总共 600 多万行 C++ 代码
- 总共 50000 多个类
- 受编译器版本限制，用的 C++ 语法都是陈旧过时的，只能在某个（早就没有维护）的操作系统上部署
- 基于 CORBA

- 采用的数据库软件来自一家早就破产的公司
- 好几层互相叠加的层共同组成了用户界面，而且这些层没有一个是由原作者维护的
- 运行一个用户界面需要启动 40-50 个子线程
- 在 32 台并行的机器上需要 48 小时进行编译
- 没有采用运行库动态链接技术，一个可执行程序就有好几百兆那么大
- 启动这玩意大约需要 15 分钟
- 然后一般 30 秒到 30 分钟内会崩溃

- 作为整件事情的亲历者，projectfailures 的博主给刚踏入编程世界的年轻人的建议是：
- C++有风险，选择需谨慎
- CORBA就应该被淘汰
- 采用面向对象的数据库是一个非常糟糕的做法
- 最后，远离贪得无厌的项目管理者

答疑