CORBA 技术系列丛书

CORBA 系统结构、原理与规范

The Common Object Request Broker: Architecture and Specification

[美] OMG 编著

信息产业部电信研究院 韦乐平

朗讯科技贝尔实验室 薛君敖 主持编译

北京邮电大学 孟洛明

電子工業出版社

Publishing House of Electronics Industry

北京·BEIJING

内 容 简 介

本书是 OMG 的 CORBA 2.3.1 版规范的中文译本，由信息产业部电信研究院、北京邮电大学、清华大学和朗讯科技贝尔实验室的多位专家、教授和工程师共同完成。中国科学院及中国工程院院士、北京邮电大学教授陈俊亮为本书作序。

本书可作为高等院校通信与计算机相关专业的教学参考书，以及相关专业软件工程师的实用参考书。

图书在版编目（CIP）数据

CORBA 系统结构、原理与规范/美国对象管理组织编：韦乐平等译.-北京：电子工业出版社，2000.6

书名原文：The Common Object Request Broker: Architecture and Specification

ISBN 7-5053-5944-4

Ⅰ.C… Ⅱ.①美… ②韦… Ⅲ.分布式处理系统-应用软件，CORBA Ⅳ.TP316.4

中国版本图书馆 CIP 数据核字（2000）第 08696 号

丛 书 名：CORBA 技术系列丛书

书　　名：**CORBA 系统结构、原理与规范**

原 书 名：The Common Object Request Broker: Architecture and Specification

编 著 者：[美] OMG

主持编译：韦乐平　薛君敖　孟洛明

责任编辑：张　毅　　zhangyi@phei.com.cn

特约编辑：秋　阳

印 刷 者：北京民族印刷厂

装 订 者：河北省涿州桃园装订厂

出版发行：电子工业出版社　URL:http://www.phei.com.cn

北京市海淀区万寿路 173 信箱　邮编：100036

经　　销：各地新华书店

开　　本：787×1092　1/16　印张：　38　字数：　972.8 千字

版　　次：2000 年 6 月第 1 版　2000 年 6 月第 1 次印刷

书　　号：ISBN 7-5053-5944-4
TP・3109

印　　数：4000 册　　定　　价：98.00 元

版权贸易合同登记号　图字：01-2000-0106 号

第1章 对象模型

本章介绍了支撑 CORBA 体系结构的具体对象模型，该对象模型源自 OMG 在 OMA 指南中定义的抽象核心对象模型（在本书前言中给出了一系列关于 OMA 指南和其他 CORBA 参考书目的有关信息）。

1.1 概述

对象模型提供了一组经过合理组织的对象概念和术语的表达。它从计算的观点定义了一个不完全的模型，其中，计算的观点体现了采用现有技术实现对象这一关键特征。由于 OMG 的对象模型并不限定实现时所采用的特定技术，所以它是一种抽象的对象模型。本章描述的对象模型则是一种具体的对象模型，具体的对象模型在以下几个方面区别于抽象的对象模型：

- 通过定义请求参数的格式或定义用来描述类型的语言等方法，可以将抽象的模型细化，使其更具体。
- 通过引入模型所定义的实体的特定实例，如特定的对象、特定的操作和具体的类型，它可以使抽象模型得到扩展更为充实。
- 通过在模型中减少实体或增加实体使用的附加限制条件，可以对模型进行限制。

对象系统是一系列对象的集合，它通过精心定义的封装接口将服务的请求者（客户）和服务的提供者相分离，尤其是让客户与诸如数据的表示和可执行代码等服务的具体实现相分离。

对象模型首先描述对客户有意义的概念，包括对象的产生和标识、请求和操作、类型和特征标记（signatures）等。其次，对象模型还要描述与对象实现有关的概念，包括诸如方法，执行引擎和激活等。

对象模型在对客户有意义的概念上，做了很明确而详细的规定，但对对象的实现，只是建议性的，其目的在于最大程度地允许采用不同的对象技术、不同的方法去实现对象。

除了前面讨论的对象模型之外，对象系统还具有一些其他的特征，其中一些概念属于应用体系结构方面的内容，一些与应用对象技术的具体应用领域相关，这些概念更适合在体系结构参考模型中论述，例如复合对象（compound object）、链接（link）、对象的复制、变化管理（change management）和事务处理（transaction）等概念。控制结构的细节也在对象模型的定义范围之外：对象模型并不指明客户或服务器采用单线程或多线程来实现，也不指明如何编写事件循环程序，以及线程如何创建、删除或同步等。

本章描述的是一种客户向对象发送消息这样的经典对象模型。从概念上说，对象负责解释消息以决定完成什么样的服务。在经典对象模型中，消息指明提供服务的对象和零个或多个实际的参数。在大多数经典的对象模型中，一个可区分的关键参数是必需的，

它标明将要完成的操作。对象对消息的解释包括选择一种基于指定操作的方法，当然，在实际运行中，方法的选择可以由对象或 ORB 去完成。

1.2 对象语义

一个对象系统为客户提供服务，服务的客户是任何能请求服务的应用实体。

本节将定义与对象语义相关概念，即与客户有关的概念的定义。

1.2.1 对象

对象系统包含一系列叫做对象的实体。对象是一个可识别的、被封装的应用实体，它可以提供一个或多个能够被客户请求的服务。

1.2.2 请求

客户通过发送请求的方式请求服务。

请求（request）被广泛地用于指从一个客户发出的一个事件到与此事件相关的最后一个事件为止的一系列相关的事件。例如：

- 客户接收到从服务器发出的与请求相关的最后响应。
- 服务器在接收到单向请求后执行相关的操作。
- 与请求相关的一系列事件由于某种故障而终结。请求的发起就是一种事件。

与请求相关联的信息包括操作、目标对象、零个或多个（实际的）参数和可选的请求上下文。

请求格式是一种描述或模式（pattern），为了能够发出请求，这种模式可以被评估或执行多次。正如在 OMG 的 IDL 语法和语义一章中所描述，请求的格式由特定的语言联编（binding）所定义。另一种请求格式包括对动态请求接口的调用，以此来创建一个请求调用的结构，并向该请求调用结构中增加参数，最后发出这个请求调用（可参考“动态调用接口”一章中对请求格式的介绍）。

值（value）可以描述任何东西，可以是请求中一个合理的实际参数。具体地说，值是一个 OMG IDL 数据类型的实例。另外，存在一种非对象值，即对对象的引用值。

对象引用（object reference）被描述为能够确定地标识一特定对象的一个值。特别是在某种时间和空间的实际限制下，对象引用在每次请求中都表示同一个对象。一个对象可以被多个不同的对象引用所表示。

请求可以包含用来向目标对象传递数据的参数；它也可以包含一个请求上下文，用于提供关于请求的附加信息。请求上下文是一种字符串到字符串的映射。

一个请求可导致一个为了满足客户需求而必须执行的服务。服务完成后，一种可能的输出是如果对此请求定义了结果，那么执行结果将被返回给客户。

如果在请求的执行过程中，发生非正常情况，系统将返回一个异常。异常信息中将会附加与该异常相关的返回参数。

请求参数由其所在的位置来标识，一个参数可以是输入参数、输出参数或输入-输出参数。请求可以返回单一的返回结果值，也可以将结果存储在输出参数和输入-输出参数中将结果返回。

以下的语义适用于所有的请求：

- 对于任何参数值的别名，无法确保其已被清除还是仍被保留。
- 不能确保对别名输出（aliased output）参数的书写顺序。
- 有异常返回的情况下，返回的结果和存储在输出、输入/输出参数中的值没有意义。

对于 CORBA 规范所允许的值和例外的描述，请参见本章 1.2.4 节“类型”和 1.2.8.3 节“异常”。

1.2.3 对象创建和删除

对象可以被创建和删除，但从客户的角度来看，并不存在一个特殊的机制来创建对象和删除对象，对象的创建和删除仅仅是客户发送的请求所产生的结果。对客户来说，对象创建的结果是一个表示新对象的对象引用。

1.2.4 类型

类型（type）是一种可标识的实体，该实体具有一个与其值相关联的谓词（具有布尔返回值的单个参数数学函数）。如果该谓词作用于某个实体得到的结果为真，那么该实体满足这种类型。满足一种类型的实体称为此类型的一个成员。

在参数说明中使用的类型，用于限制可能存在的参数或表示可能返回的结果的特征。

在任何时候都能够满足某指定类型的值的集合称为该类型的扩展（extension）。

对象类型（object type）是一种其成员为对象引用的类型。换句话说，对象类型只被对象引用所满足。

在本模型中对数据类型的限制如下：

1.2.4.1 基本类型

- 16 位、32 位和 64 位有符号和无符号的二进制补码整数。
- 单精度（32 位）、双精度（64 位）和扩展双精度（由至少 64 位的尾数、一个符号位和至少 15 位的指数组成）的 IEEE 浮点数。
- 最多到 31 位有效数字的定点十进制数。
- 由 ISO Latin-1（8859.1）和其他单字节或多字节字符集合所定义的字符。
- 取值为真和假的布尔类型。
- 保证在不同系统间传递时不需任何转换的不透明的 8 位组。
- 由标识符按照顺序组成的枚举类型。

- 由可变长度的字符数组组成的字符串类型；字符串的长度是一个非负整数，字符串长度在运行时获得。字符串长度有一个被定义的最大值上限。
- 由可变长度的超大字符数组组成的超大字符串类型；字符串的长度是一个非负整数，字符串长度在运行时获得。字符串长度有一个被定义的最大值上限。
- 可以表示任何可能的基本类型和构造类型的“ANY”类型。
- 可以表示来自任何超大字符集的字符的超大字符类型。
- 超大字符串，包含在运行时可获得的长度，和一个可变长度的超大字符数组。

1.2.4.2 构造类型

- 记录类型（也称结构）：由（名字，值）这样的二目组组成的有序集合。
- 可区分的联合类型：包括一个鉴别器（此鉴别器总应该有值），和紧接着该鉴别器的一个符合鉴别器取值类型的实例。
- 序列类型：由单一类型组成的长度可变的数组。序列的长度在运行时获得。
- 数组类型：由单一类型组成的具有固定形式的多维数组。
- 接口类型：指明该类型的实例所必须支持的操作的集合。
- 值类型：描述该类型的实例必须支持的状态及操作集合。

请求中实体的取值被限制为以上类型。合法的实体如图 1-1 所示。对所定义的实体不存在其他特别的表示。

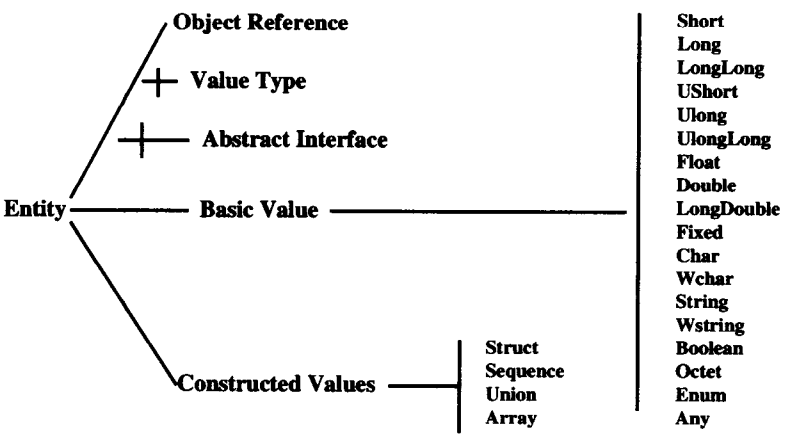

图 1-1 合法的值

1.2.5 接口

接口（Interface）是客户可能通过其发送请求的对象中所定义的一系列操作的描述。接口阐明了由支持接口的对象所提供的服务是如何通过这组操作被访问的语法描述。如果一个对象按照接口中所定义的操作提供服务的话，我们说此对象满足该接口（参见 1.2.8 节“操作”）。

一个给定接口的接口类型是一个对象类型，这样，当且仅当所引用的对象满足此接口，一个对象引用才满足此类型。

接口由 OMG 的 IDL 描述，接口继承提供了允许对象支持多接口的复合机制。主接口（principal interface）是对象所支持的最特殊的接口，它由接口继承图的传递闭包中的所有操作组成。

1.2.6 值类型

值类型（Value Types）是一种具有接口和结构的多种特性的实体。值类型对可被客户请求的操作和可被客户访问的状态进行了描述。值类型的实例通常是编程语言中本地具体的实现（implementation）。

一个值类型，除了其本身定义的操作和状态外，还可从其他值类型继承，并通过多重继承来支持其他接口。

值类型由 OMG IDL 来描述。

抽象值类型是一种不包括状态、仅包括操作的值类型。

1.2.7 抽象接口

抽象接口是一种在运行时可表示常规的接口或值类型的实体。和抽象值类型一样，抽象接口不包括状态，仅包括操作。和抽象值类型不同的是，抽象接口并不具有传值（pass-by-value）语义，并且和常规的接口不同的是，抽象接口并不具有引用传递（pass-by-reference）语义。相反，实体运行时的类型决定了所使用的语义。

1.2.8 操作

操作（operation）是一种可标识的实体，它指明了可被请求的不可再分的服务提供原语。请求一个操作的动作被称为操作调用。一个操作由操作标识符来标识。

操作包括一个参数特征标识（signature），用于描述请求参数和返回结果的合法取值。具体地说，参数用法说明包括：

- 请求操作所需参数的说明
- 操作结果的说明
- 由于对操作的调用而引起的用户异常的标识
- 有可能影响调用的附加上下文信息的说明
- 客户在操作调用时期望的执行语义的指示

操作具有通用性，即对于采用不同实现的对象，可对其施加相同的操作请求，但此时对象可能表现出明显不同的行为。在本模型中，操作的通用性是通过 IDL 接口的继承机制，以及将接口规范和接口的实现相分离来实现的。

操作参数使用说明的通用格式如下：

[oneway] <op_type_spec> <identifier> （param1, ..., paramL）
[raises（except1,...,exceptN）] [context（name1, ..., nameM）]

其中：

- 对于按照以上格式定义的操作的请求，其中的可选关键字“**oneway**”表明一种最大努力的语义，它缺省的语义包括以下两种含义：如果操作能够成功地返回结果，该操作只能被执行一次；如果有异常返回，该操作最多一次。
- **<op_type_spec>**是返回结果的类型。
- **<identifier>**为接口中的操作提供了一个名字。
- 操作中所需的参数，这些参数被标以修饰符 **in**、**out** 或 **inout**，以指示信息流的方向（关于执行请求的对象）。
- 可选表达式 **raises** 标识用户所定义的异常，异常的产生可以中断对操作的调用。如果该表达式不存在，就不会有用户定义的异常产生。
- 可选表达式 **context** 表示可以被对象的实现所利用的请求上下文信息，除此之外，不存在其他伴随请求传送的上下文信息。

1.2.8.1 参数

参数通过其本身所具有的模式和类型来描述。其中，模式表示值的传递是从客户到服务器（**in**），还是从服务器到客户（**out**），或者双向（**inout**）进行。参数的类型规定了按照前面定义的模式进行的传输中参数的可能取值。

1.2.8.2 返回结果

是一个可被识别的输出（**out**）参数。

1.2.8.3 异常

异常是用来标识操作请求没有被成功地完成，异常的产生通常伴随着附加的具体异常信息。

附加的异常信息是一个特定格式的记录。作为一个记录，它可以由前面 1.2.4 节中描述的任意类型组成。

所有的参数用法说明都隐含着系统异常；标准的系统异常在 3.17 节“标准异常”中介绍。

1.2.8.4 上下文

请求上下文提供了能够影响请求完成的附加的特定操作信息。

1.2.8.5 执行语义

本对象模型中，定义了两种执行语义：

- 最多一次（at-most-once）：如果操作请求能够成功地返回，那么该操作是只被执行一次（exactly once）；如果返回异常，该操作是最多被执行一次（at-most-once）。
- 最大努力（best-effort）：最大努力的操作是纯请求（request-only）的操作，即，它不能返回任何结果，而且请求者从不与操作是否结束相同步。

所期望的执行语义总是与具体的操作相关联，从而防止客户和对象的实现采用不同的执行语义。

特别要注意，客户在同步或延迟同步（deferred-synchronous）的方式下都能发起语义为最多一次的操作。

1.2.9 属性

接口中可以定义属性。属性在逻辑上等同于声明一对存取函数：一个用来获取属性的值，另一用来设置属性的值。

如果属性被定义为只读属性，在这种情况下，只存在一个获取属性值的取值函数的定义。

1.3 对象实现

本节定义与对象实现（Object Impletementation）相关的概念（例如在计算系统中与对象行为的实现有关的概念）。

对象系统的实现能够完成这样的计算活动，该计算活动用来影响被请求的服务的行为。这些活动可包括通过计算得到请求的结果和对系统的状态的更新，在此过程中，可产生一些附加的请求。

本实现模型包括两部分：执行模型和构造模型。执行模型描述服务如何执行；构造模型描述服务如何定义。

1.3.1 执行模型：执行服务

在计算系统中，被请求的服务通过执行一些对数据进行操作的代码来完成。被操作的数据用来表示计算系统状态的组成元素，代码完成被请求的服务，从而改变系统的状态。

为了完成所定义的服务而执行的代码称作方法（method）。方法是对计算的不可改变的描述，它可以被执行引擎（execution engine）所解释。方法有一个不可变的属性，称作方法格式（method format）。方法格式定义了可解释此方法的执行引擎的集合。执行引擎可以被认为是一种抽象的机器（不是程序），它能够对以某种格式描述的方法进行解释，从而完成被描述的计算。执行引擎为方法的执行定义了一个动态上下文。方法的执行称为方法的激活（method activation）。

客户发出的请求将调用目标对象定义的某个方法。请求者提供的输入参数将传递给调用的方法，输出、输入-输出参数和返回结果值（或异常及其参数）将被返回给请求者。

调用被请求的服务会导致相应方法的执行，该方法可对一个对象的持续状态进行操作。如果方法或状态的持续形式不能被执行引擎访问，需要首先将方法或状态复制到执行上下文中，整个过程称为激活（activation），其逆过程称为终止（deactivation）。

1.3.2 构造模型

计算对象系统必须提供一系列机制来实现请求的行为。这些机制包括对象状态的定义、方法的定义,以及定义对象的基础体系结构如何选择需要执行的方法、如何选择与对象状态相关的部分，以便该方法能够对对象的状态进行访问。在对象系统中还需定义一些机制来描述与对象创建相关的具体动作，例如怎样选择合适的方法与新创建的对象相关联。

对象实现——简称实现，是定义了创建一个对象和允许一个对象参与提供一套合适的服务所需的信息。实现通常包括对对象状态进行操作的方法的定义，同时也包括与对象的预期类型的信息。

第 2 章　CORBA 概述

公共对象请求代理结构（CORBA）的提出，使得各种各样的对象系统能够进行集成。之所以在本章中定义 CORBA 所具有的这些特征，其动机在本章的开始尚不明显，但随着对于CORBA的实现、策略、优化，以及对CORBA的应用等问题讨论的不断深入，CORBA本身所具有的灵活性的特点会越来越明显。

2.1　对象请求代理结构

图 2-1 给出了由客户向对象实现（object implementation）发送请求的过程，客户是指使用某个对象所实现的某种操作的实体，对象实现是指具体实现这一对象的代码与数据。

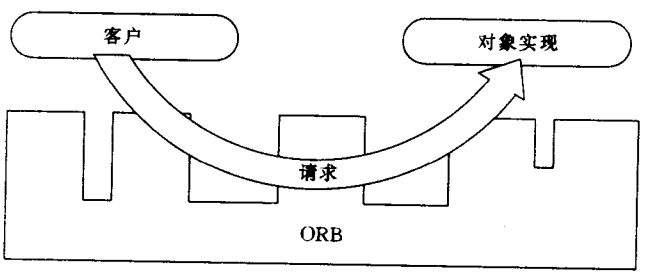

图 2-1　通过对象请求代理传送的请求

ORB 提供了如下支持分布式处理的机制：为请求查找具体的对象实现、让对象实现作好接收请求的准备、传送构成请求的数据等。客户所看到的接口完全独立于对象所在的物理位置、实现对象的编程语言、以及在对象的接口中没有反映出来的其他特性。

图 2-2 给出了一个单独的对象请求代理（ORB）的结构，其中条形框表示 ORB 所提供的接口，箭头指示这个 ORB 是否被调用或者正在通过接口执行一个调用。

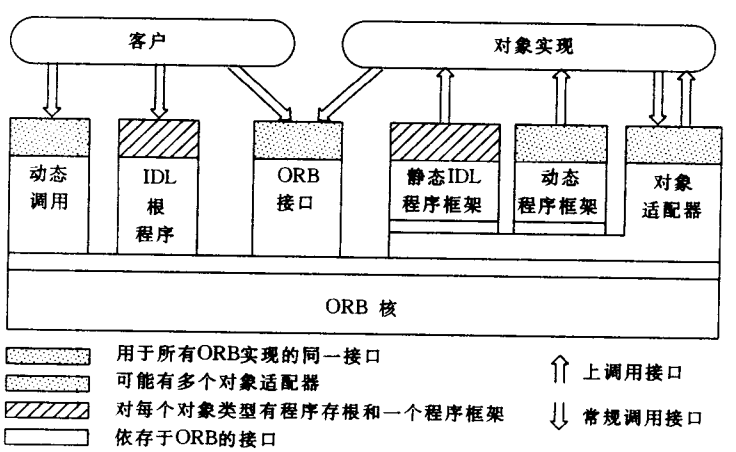

图 2-2　对象请求接口的结构

客户通过发送请求使用对象实现所提供的服务，具体实现过程中，客户可以使用动态调用接口（独立于目标对象接口的另一种对象接口）或静态 OMG IDL 根程序（stub）（具体的根程序取决于目标对象所支持的接口）。有时，为了调用某些特殊功能，用户可能需要与 ORB 进行直接交互。

对象实现要么通过 OMG IDL 产生的程序框架（skeleton），要么通过动态程序框架（dynamic skeleton）接收作为上行调用（up-call）的请求。在处理请求或其他任何时候，对象实现均可以调用对象适配器和 ORB。

对象接口的定义可以采用两种方法，其一是用一种接口定义语言静态地对接口进行定义，我们称之为 OMG 接口定义语言（OMG IDL），这种语言根据对象可能实现的某种操作和操作中需要的参数来定义对象的类型；另一种方法是将接口添加到接口库服务中，该服务将构成接口的组件表达为对象，并允许在运行时刻对其进行访问。在以上任何一种 ORB 实现中，接口定义语言（在本书中，我们可以对其定义进行扩展）与接口库具有相同的表达能力。

客户通过访问对象的对象引用、了解对象的类型以及所需执行的操作的基础上执行请求，对于一个请求的初始化，客户可以通过调用针对某个特定对象的根程序例程或动态地构造请求来实现，如图 2-3 所示。

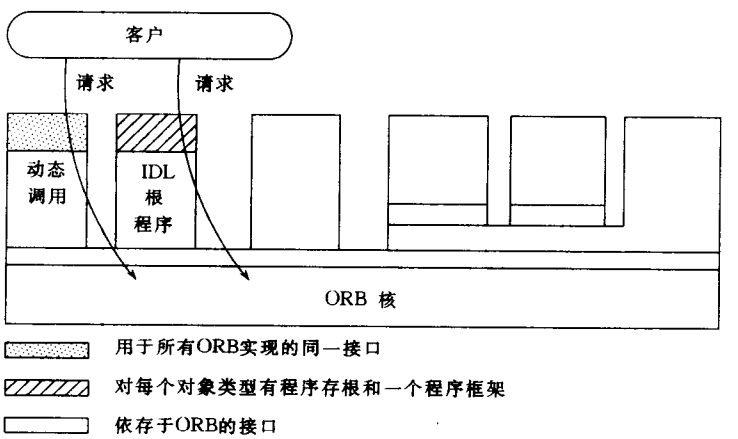

图 2-3　使用根程序或动态调用接口的客户机

不论采用动态方式还是静态根程序接口方式，请求调用都满足相同的语义，消息的接收者并不分辨请求是如何被调用的。

ORB 通过 IDL 程序框架或动态程序框架（参见图 2-4）来定位相应的实现代码、传送参数，以及对对象实现的传送控制。接口和对象适配器都有其特定的程序框架，在执行请求时，对象实现可以通过对象适配器获得 ORB 提供的某些服务，当这一请求完成后，控制权和输出结果返回给客户。

对象实现可以选择合适的对象适配器，对象适配器的选择由对象实现所需要的服务而确定。

图 2-5 显示了客户与对象实现如何有效利用接口及对象实现信息的机制。在这里，接口采用 OMG IDL 方式和/或接口库方式定义，定义的目的是利用接口生成客户根程序以及对象实现的程序框架。

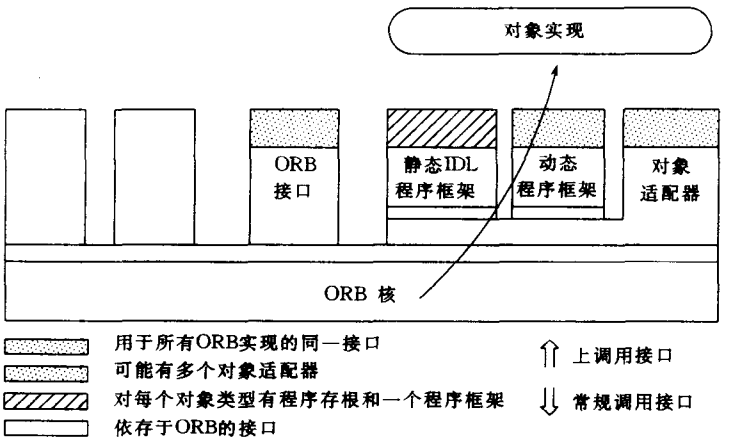

图 2-4 接收请求的对象实现

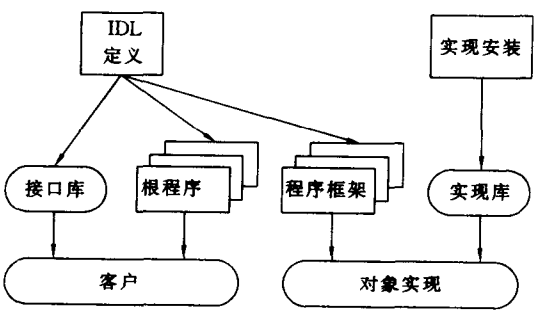

图 2-5 接口与实现资源库

安装时提供的对象实现信息被存储在实现资源库中，以备发送请求时使用。

2.1.1 对象请求代理

在 ORB 结构中，ORB 并不需要作为一个单独的组件来实现，而是通过一系列接口来定义，任何一个提供适当接口的 ORB 实现均是可行的，这些接口分为以下三类：

1．对所有 ORB 实现均相同的操作。

2．对某些特定类型对象的特定操作。

3．对某些特定式样的对象实现的特定操作。

不同的 ORB 所选择的实现方式有很大不同，通过与 IDL 编译器、接口库，以及各种对象适配器的协调工作，ORB 可以向客户机以及那些具备不同特性的对象的实现提供一系列服务。

有可能存在多个 ORB 实现的情况（也可描述为多 ORB），此时，对象引用具有不同的表达方式，对于调用的执行可能采用不同的方法，客户可以同时访问两个由不同的

ORB 实现所管理的对象引用。当需要两个 ORB 同时工作时，ORB 必须能够区分它们自己的对象引用，而客户不必关心对象引用的区分。

ORB 核心是整个 ORB 的一部分，它负责提供基本的对象表示，并负责传送对象请求。CORBA 的设计目标是支持不同的对象机制，这种支持正是通过在 ORB 核心之上构造 ORB 组件来实现的，ORB 组件可以屏蔽不同 ORB 核心之间的差异。

2.1.2 客户

对象的客户可以访问该对象的对象引用、调用该对象的操作。客户只是根据对象接口了解其逻辑结构，并通过调用掌握对象的行为。尽管我们通常将客户看作是一个能够初始化对象请求的程序或进程，我们也必须认识到，相对于某些特定的对象而言，其部分实现也可以被看作为客户，例如，一个对象的实现可以是其他对象的客户。

客户通常从语言映射的角度来看待对象和 ORB 接口，也就将 ORB 直接带到程序员级别上。客户应该具有最好的可移植性，并且无需修改源码，就可以在任何 ORB 上运行，这些 ORB 拥有实现了需要接口的对象实例而且支持需要的语言映射。客户不必了解对象的具体实现方法，也不必知道这个实现采用哪个对象适配器，或是这个实现必须采用哪个 ORB 进行访问。

2.1.3 对象实现

通常，对象实现通过定义描述对象实例的数据和对象所实现的方法的编码来提供对象的语义。对象实现往往要借助于其他的对象或软件来实现它的具体行为，某些情况下，对象的基本功能对其他的非对象部分有影响。

对象实现的方法可以有多种多样，包括独立的服务器、程序库、方法的程序代码、封装的应用，以及面向对象的数据库等。实际上，通过使用附加的对象适配器，几乎可以支持所有风格的对象实现。

通常，对象实现不依赖于 ORB 或客户调用对象的方式，通过对对象适配器的选择，对象实现可以选择那些依赖于 ORB 的服务的接口。

2.1.4 对象引用

对象引用是用来指明某个 ORB 中的某个对象。按照语言映射的规定，对象引用对于客户和对象实现仅仅是一个抽象的概念，两者可以独立于对象引用的表示。两个 ORB 实现可以选择不同的对象引用表示方式。

传递给客户的对象引用的表示方式只在该客户的生命周期中有效。

对于某种编程语言，所有 ORB 对于一个对象引用（通常称为对象）必须提供同样的语言映射，这种规定使得以某种语言编写的程序对于对象引用的访问可以独立于特定的 ORB。当然， 语言映射也可以提供其他方式访问对象引用，所采用的方式取决于对编程者是否方便。

有一种对象引用比较特殊，它可以确保与所有对象引用之间的差别，而不具体指明任何对象，

2.1.5 OMG 接口定义语言

OMG 接口定义语言（OMG IDL，OMG Interface Definition Language）通过描述对象的接口来定义对象类型，接口包括一系列被命名的操作以及这些操作所需的参数。尽管接口定义语言提供了相关的概念框架来描述那些由 ORB 所控制的对象，但是 ORB 的正常工作不需要接口定义语言的源代码。只要等效的信息可以从根程序例程或运行时从接口库中获得，ORB 就可以正常工作。

利用 IDL，一个特定的对象实现可以向其潜在的客户描述本身可提供的操作，以及如何调用这些操作。根据 IDL 定义，可将 CORBA 对象映射为特定的编程语言或对象系统。

2.1.6 OMG IDL 到编程语言的映射

不同的面向对象或非面向对象编程语言可以采用各自喜欢的不同方式访问 CORBA 对象，面向对象语言可能倾向于将 CORBA 对象看作编程语言对象，即使是非面向对象的编程语言，将对象引用、方法名字的精确 ORB 表示屏蔽起来也是一种很好的想法。对于所有的 ORB 实现，从 OMG IDL 到一种特定编程语言的映射方法应该是相同的。语言映射包括对该编程语言所用到的数据类型的定义，以及一些通过 ORB 能够进行对象访问的函数接口。具体而言，包括客户方根程序接口的结构（采用面向对象的语言时，不必要求本项）、动态调用接口的结构、实现方程序框架、对象适配器以及直接的 ORB 接口结构。

语言映射也定义了对象调用和客户的控制线程之间的交互或实现。通常，映射采用同步调用方式，即当对象操作完成时，例程才返回。为了能够初始化调用过程，并能够实现返回主程序的控制，还需要附加其他映射规则，在这种情况下，需要提供一些专门用于这种语言的例程，以实现程序的控制线程和对象调用之间的同步。

2.1.7 客户方根程序

对于非面向对象语言的映射，对于每一个 IDL 接口类型，都将有一个编程接口支持对生成的根程序的调用。通常，根程序使用一种简单的方式，以便访问 OMG IDL 定义的对象操作，这种方式使得编程者只要熟悉 OMG 接口定义语言和所映射的编程语言即可。该根程序利用那些对 ORB 核心专用的、优化的接口调用 ORB 的其他部分。如果有多于一个的 ORB，则针对于不同的 ORB 就有不同的根程序。在这种情况下，ORB 与语言映射必须协调一致以便将正确的根程序和特定的对象引用相关联。

面向对象的编程语言如 C++和 Smalltalk 等，不需要根程序接口。

2.1.8 动态调用接口

接口允许动态地构造对象调用，也就是说，客户可以不调用一个专门用于特定对象的特定操作的根程序，而是通过一个调用或一连串调用来指定被调用的对象、所要执行的操作、以及这项操作的一组参数。客户代码必须提供关于所要执行的操作的信息以及要传送参数的类型（可能从接口库中或其他运行的资源中得到）。动态调用接口的特征可以因为可编程语言映射的不同而有很大不同。

2.1.9 实现程序框架

对于一种特定的语言映射，都有一个可能依赖于对象适配器的可实现任何类型对象方法的接口。通常，这个接口是一个上行（up-call）接口,在这一接口中，对象实现将编写与这些接口匹配的程序，然后 ORB 通过生成的程序框架调用这些程序。

程序框架的存在并不意味着一定有相应的客户根程序存在（客户也可以通过动态过程调用接口发送请求）。

可以构造一个不使用程序框架来调用实现方法的对象适配器。例如，像 Smalltalk 这样的语言可以动态地创建实现。

2.1.10 动态程序框架接口

动态程序框架接口可以实现对象调用的动态处理，这就是说，在通过接口来达到对象实现时，这一接口采用与客户侧动态过程调用接口的相似的方式访问操作名称及参数，而不必通过专用于某一操作的程序框架访问。其中的参数可以使用静态及动态（或许通过一个接口库决定的）知识来确定。

实现代码必须将所有操作参数描述提供给 ORB，在执行操作的过程中，ORB 将提供所有输入参数值。实现代码在执行操作后，将向 ORB 提供所有输出参数值或异常。随着可编程语言映射之间的不同以及对象适配器之间的不同，动态程序框架接口可能有很大差别，但普遍而言，都是上行接口。

动态程序框架可以由客户根程序或动态调用接口方式实现调用，两者在效果上相同。

2.1.11 对象适配器

对象适配器是对象实现访问 ORB 所提供的服务的主要方式，将来，会出现一些使用范围广泛的对象适配器，它们的接口适合于某些专门类型的对象。ORB 通过对象适配器所提供的服务通常包括：对象引用的生成与解释、方法调用、交互的安全、对象和实现的激活与终止、对象引用到对象实现的映射以及对象实现的登记。

对象粒度（granularity）、生命周期、策略、实现式样及其他特性的多样性使得 ORB 核心很难提供一个方便有效地用于所有对象的单一接口。因此，ORB 可以通过对象适配器来将一组特定的对象实现作为目标，这些对象实现具有类似的功能需求和接口。

2.1.12 ORB 接口

所谓的 ORB 接口是指那些直接面对 ORB 的接口，它们不依赖于对象接口或对象适配器。由于 ORB 的大多数功能是通过对象适配器、根程序、程序框架或动态过程调用的方式提供的，因此只有少数操作对所有的对象来说是通用的。这些操作对客户和对象的实现都是有用的。

2.1.13 接口库

接口库（interface repository）是一种能够提供持久对象的服务，这些对象在运行时以某种方式表示 IDL 信息。ORB 可以使用接口库信息来执行请求，而且，利用接口库中的信息，程序可以处理一些特定的对象，这些对象的接口描述在程序编译时未知，但却可以在运行时通过判断哪些操作对这个对象有效，然后实现该操作的调用。

接口库的功能除了表现在 ORB 上，还表现为可以存储与 ORB 对象接口相关的信息。例如，调试信息、根程序和程序框架库、能够格式化并浏览特定对象的程序等，都可以与接口库相关联。

2.1.14 实现库

实现库（implementation repository）包含允许 ORB 定位和激活对象实现的信息。虽然实现库中的大部分信息与 ORB 或操作环境相关，但是实现库通常也是用于记录这些信息的地方。通常，实现的安装以及激活和执行对象实现的相关的策略的控制是通过对实现库的操作实现的。

实现库的功能除了表现在 ORB 上，还表现为可以存储与 ORB 对象实现相关的附加信息。例如，调试信息、监视控制、资源分配和安全等都可以与实现库相关。

2.2 ORB 举例

遵循公共 ORB 体系结构，可以有很多 ORB 的具体实现。本节我们举例说明其中一些选项。要注意的是，一个特定的 ORB 可能支持多个选项及通信协议。