第5章 架构模式

设计模式提供了用于解决具体的问题模式，而架构模式是粗线条的模式，它的作用不是用于解决局部问题，而是要影响一个系统的基本结构。

本章介绍的架构模式都是作者在实践中经常使用的。根据[Bus98]可以把架构模式分为下面几种类型：

- **系统的结构**：分层架构模式、管道和过滤器架构模式。
- **自适应系统**：插件架构模式。
- **分布式系统**：中介模式、面向服务的架构模式。
- **交互式系统**：模型-视图-控制器模式。

下面简要介绍这些模式。

（1）**分层架构模式**（见 5.1 节）把一个系统的架构划分为不同的层次，每一层可以调用下一层的服务（一个系统的水平分层）。这个模式要求运行的流程是：客户端把系统的最上层作为服务器进行调用，最上层又作为客户，把下一层视为服务器进行调用，系统以这种方式一直调用到最后一层。

（2）**管道和过滤器架构模式**（见 5.2 节）把一个应用的基本形式构造成按时间顺序加工处理的链状结构，它们之间通过输出和输入相互耦合：一个进程的输出是下一个进程的输入（**面向数据流的系统架构**）。这个架构是把一个系统划分成由进程组成的链状结构，每个进程都接收前一个进程产生的结果，然后再进一步加工。因此，管道用于解决两个相邻进程的异步解耦。

（3）**插件架构模式**（见 5.3 节）对于特定的应用，把可扩展的部分构造成接口形式，这样就可以通过使用插件实现这些接口，由插件管理者（利用运行时的环境）负责实例化和加载。应用在没有扩展的情况下也可以正常运行。通过使用插件架构模式，没有扩展的基本应用可以具有广泛的用户，而扩展后的应用针对某些用户群体。

（4）**中介模式**（见 5.4 节）构造的系统是由多个客户端和服务器组成的一个架构，中介作为中间者实例对客户提出的需求根据其所需的服务找到相应的服务器，并把服务器的应答送至客户端。客户组件和服务器组件通过中介进行交流。在分布式系统中，中介作为中间件分布在系统中的所有节点计算机中。

（5）**面向服务架构**[1]的设计模式（见 5.5 节）把系统的架构划分为组件或子组件，它们与系统设计阶段的应用实例或部分应用实例相对应，通过提供服务反映自身的功能，因此业务流程的改变只有以下影响：

- 作为应用实例所代表的相应组件必须被修改。
- 在使用现有基本服务的基础上生成新的组件。

---

1 面向服务架构缩写为 SOA。

模型-视图-控制器模式（MVC）[1]（见 5.6 节）把应用分解成模型组件（加工处理/数据管理）、视图（输入）和控制器（输出）。因此模型不依赖于输入和输出。使用这个策略易于更换系统中的视图和控制器（图形界面组件与加工处理相分离），因此应用的核心部分比输入和输出程序有更长的使用期。

一个架构模式可以含有不同的设计模式。在一个架构模式中，设计模式间是否需要通信，或者多少模式间需要通信，根据情况有所不同。模型-视图-控制器模式通常含有观察者模式、策略模式和组合模式。分层架构模式则相反，不使用任何设计模式。

第 4 章介绍的设计模式都是面向对象设计模式，模式也可以具有非面向对象的特征。例如，分层架构模式和插件模式都没有使用面向对象的技术。

5.1　分层架构模式

分层架构模式在建模过程中把一个**系统的结构**进行分层，各层之间的关系是客户/服务器。

5.1.1　名称/其他可用的名称

层次模型、分层。

5.1.2　问题

一个软件系统可以水平分割成合理的多个子系统层。系统中相关联的部分被集中放在一个独立的层内。每一层都要对其所包含的功能进行抽象定义。上层功能的实现需要调用下层的功能并得到回复。相邻的任何两层都符合客户与服务器的关系。当服务器层调用更深一层时，它就被视为客户层。

这种方式可以限制相互依赖关系的数量。更易于开发系统的每一个功能，更改代码也只会产生有限的影响。

5.1.3　解决方法

如果一个系统高度复杂，可以在软件架构设计时，在水平方向把系统分割成相互连接的多个层次，即层次模型。因此每一层（tier 或者 layer）都包含系统的一个抽象形式的子任务（例如通信和数据管理）。分层通常应用在功能方面上。图 5-1 是一个层次模型的示意图。

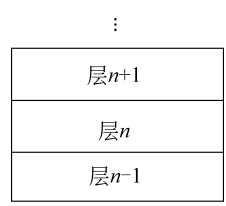

图 5-1　层次模型

下层为上层提供服务（service）并为上层的调用返回结果，即下层的功能是服务于上层的。下层要回答上层的服务请求。因此层

1　模型-视图-控制器模式缩写为 MVC。

次关系是符合客户/服务器原则的。

分层架构通常有下列规则：

（1）第 *n* 层只依赖于它下方的第 *n*–1 层。

（2）一个层不依赖于它的上一层。

（3）每一层只为上一层提供服务。

（4）上一层通过下一层的接口使用下一层提供的服务。

下一层不能使用上一层的服务。但是上一层可以使用下一层的服务。

每一层都需要通过接口使用下一层的服务。层的数量不能过多。上一层只可以使用下一层的服务。模式的基本含义是一层只能调用下面相邻的一层。这种模式被称为**严格分层**（strict layering）。**层桥接**（layer bridging）的定义则是一层可以调用它下面的所有层。层桥接不需遵守上面的第（1）条规则。图 5-2 的示例说明了层的桥接，图中第 4 层直接调用了第 2 层。

在**严格分层**中，一个层只能调用它下面相邻的层。**层桥接**则可以直接调用这个层下面的所有层。

图 5-2 层的桥接示例

每一层在系统中表达的是一个抽象层。在以前的文献中（例如[Dij68]），层也被称为抽象机器[1]，层通过接口向上层提供的功能被看作是一个抽象机器的操作。因此从整体上看，就形成了一个层次结构的抽象机器。

5.1.3.1 类图

在本章中，层和层的功能用面向对象方法中的类表示。但是层的概念并不属于面向对象。图 5-3 中没有使用非面向对象的方法，这里讨论层次模型的基本形式，第 *n* 层的类使用了第 *n*–1 层的类，一个层代表一个组件（子系统），通过类实现一个组件。

只能对实现组件的类进行实例化，而不能对组件 Schicht_n 进行实例化。组件在这里是抽象的，它可以由类实现，这涉及所谓的**间接实现**[Hit05，第 145 页]。

5.1.3.2 参与者

这个架构模式的参与者是一个层里的对象和它下面的一个层中的对象：

Klasse_aX_der_Schicht_n/第 *n* 层的类 aX

这个类的对象要求第 *n*–1 层里类 Klasse_bY_der_Schicht_n–1 的对象提供服务。

Klasse_bY_der_Schicht_n–1/第 *n*–1 层的类 bY

这个类的对象可以提供服务。如果在它的下面还有层，它也可以调用下一层中的对象。

5.1.3.3 动态行为

图 5-4 说明了层之间的客户/服务器的关系：

1 抽象机器的概念在这里是与实际机器相对的，实际机器是具体实现的硬件。现在人们更多地将抽象机器称为虚拟机器。

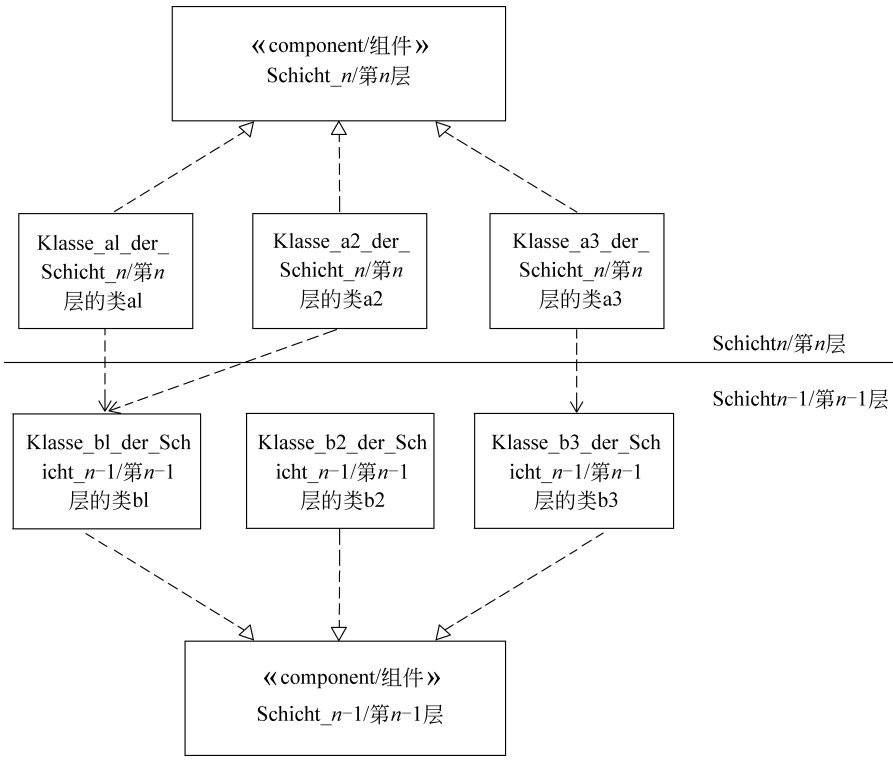

图 5-3　相邻组件的对象间相互依赖关系

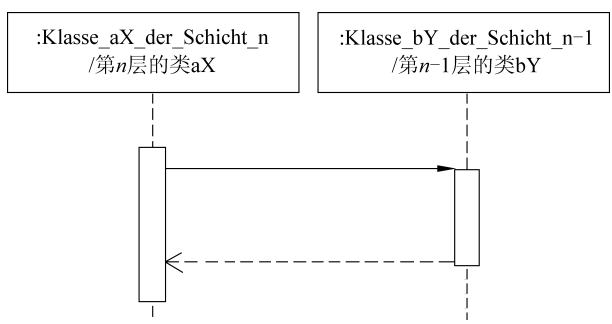

图 5-4　相邻两层的对象间的调用和应答

第 *n*–1 层为第 *n* 层提供服务，同时它也使用第 *n*–2 层提供的服务。

5.1.4　评价

根据“分而治之”的原则，一个复杂的、难以掌握的问题要被分割成尽量没有依赖关系的多个小问题，这些小问题则易于理解和解决。分层模式遵循这个原则，把复杂的问题分解到各个层中。层次结构也遵守客户/服务器原则。

5.1.4.1　优点

分层架构模式具有下列优点：

- 每一层都把一个具体功能进行了抽象化，层相对于整体更易于理解。
- 层也可以被复用。
- 层是稳定的，可以被标准化。
- 可以降低代码的相互依赖程度，即受到更改代码影响的层很少。
- 确定接口后，各层可以并行开发。
- 这些层可以很好地从下至上进行集合并测试。

5.1.4.2 缺点

分层架构模式的缺点如下：

- 严格分层的规则往往过于严格，它规定一个层只能调用下面相连的一个层。但是如果系统性能提高了，可以规避这个规则（层桥接）。
- 很明显的一个问题就是逐层传递比直接传递耗时。
- 改动会影响到层。一些工作，比如修改错误可能会影响到所有的层。这就意味着更多的消耗。
- 很难确定“正确的”层数。

5.1.5 使用范围

所有使用客户/服务器原则建模的系统都可以使用分层模型。相似的功能在一个层中被抽象化。一个高层也常常体现为下一个层的抽象概括。例如，操作系统层是计算机硬件层抽象化的结果。

层次模型应用于计算机硬件、操作系统和应用软件的合作（见 5.1.5.1 节）。层次模型应用于计算机应用软件（见 5.1.5.2 节）。5.1.5.4 节介绍了 ISO/OSI 模型中用于计算机通信系统的层次模型。

实际上每一个应用软件的接口（API）都代表了一个层，它是下一个层次的抽象化结果。例如，程序语言 C 的标准化库是下一层具体操作系统的抽象化结果。特别是输入和输出分别由两个层完成：低层的输入和高层的输出。

虚拟机也体现了分层架构模式的应用。例如，Java 虚拟机就是在实际硬件的基础上抽象的结果。Java 虚拟机可以处理以字节为单位的代码，比正常的机器代码更紧凑，更概括。

外观模式引入了一个附加的层，用于封装一个子系统的类。

5.1.5.1 层次模型应用于计算机硬件、操作系统和应用软件

一个操作系统就是一个系统程序，它有两个基本的任务：

- 对维持计算机正常运转的部件进行管理和抽象化。
- 提供用户支持。

图 5-5 可以帮助读者更好地理解层次模型。

“其他系统程序”可以是 UNIX 的命令接口或者是一

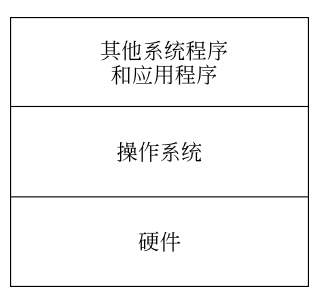

图 5-5 计算机的层次结构

个 DBMS[1]等。

操作系统既有对硬件的接口，也包含对用户程序的接口，这两类接口都必须“服务于”操作系统。

操作系统在系统程序和应用程序面前隐藏了硬件。它对硬件进行了抽象化处理，展示给用户的是虚拟机，虚拟机与实际的硬件相比更易于编程。操作系统的目标如下：

- 合理分配计算机的系统资源，并可以无差错地提供给用户。这意味着对于通过竞争获取进程、存储和输入输出设备的程序，要合理有序地分配资源。
- 软件开发人员不必了解硬件的具体细节，可以节省开发成本。

操作系统可以生成与其他系统程序、应用程序和计算机硬件的连接。在硬件的基础上构建软件的层次，从而发挥软件程序的功效。

5.1.5.2　层次模型应用于计算机应用软件

层次模型不仅可以应用于信息系统，还可以应用于嵌入式系统[2]（embedded systems）。下面只介绍层次模型在信息系统中的应用。在这里不再考虑操作系统。

一个计算机的应用软件总体包含下列功能：

- 输入输出（用户接口）。
- 数据处理。
- 数据的持久性存储。

在使用应用于商业目的的数据库时，数据的持久性存储包括：

- 数据的调用（DBMS 的接口和与所使用的程序语言相对应的 API）。
- 数据库管理系统的核心。
- 操作系统的功能。
- 数据在硬盘上的持久性存储。

这里的数据库管理系统掩盖了操作系统。

没有使用应用于商业目的的数据库，而是直接调用硬盘上的数据时，数据的持久性存储包括：

- 数据的调用（文件系统的接口和与所使用的程序语言相对应的 API）。
- 操作系统的功能。
- 在硬盘上数据的持久性存储。

在余下的各节中，只关注使用了商业数据库的这种情况。不再讲解关于程序直接从硬盘中读取和写入，或者从硬盘读取数据。

通过使用数据库管理系统，形成了 4 个层次：

- 输入输出层；
- 加工处理层；
- 数据访问层；
- 数据库管理系统核心。

1　DBMS 就是数据库管理系统。

2　例如可以参考应用于汽车工业的 AUTOSAR 架构[Kin09]。

下面具体讲解这 4 个层：

- **输入输出接口**（Man-Machine Interface，MMI）处在输入输出层，它可以启动事件驱动的应用程序。
- **加工处理层**的对象是控制对象和主存（缓存数据）里的实体对象。
- **数据访问层**把数据的持久性存储进行抽象化。因为在专有的数据库管理系统中包含特定的数据访问功能，所以数据访问层不是总需要由自己开发。数据访问层中的类负责持久存储和按需加载数据。也可以在数据访问层上面的一层中增加自己编写的不依赖于数据库的接口（见图 5-6）。因此，可以更换不依赖于数据库的数据访问层下的数据库管理系统部分，而不涉及加工处理层的对象。

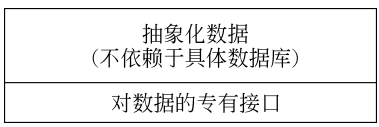

图 5-6　数据访问层结构

- 如果不需要对数据库管理系统进行更换，就可以在**抽象数据层**中放弃专有接口的封装。这种情况下，在数据访问层中存在的只有底层的**依赖生产厂家的部分**。
- **数据库管理系统**核心处在数据库管理系统的中心部位。它把实例对象的数据存储到硬盘上或者用于下载数据。

5.1.5.3　单机系统和多机系统的比较

无论是对单机系统（stand-alone system）还是分布式系统，划分成层都是非常重要的。对于分布式系统还要增加用于通信的功能。客户/服务器系统常用的有二层架构（two-tier architectures）和三层架构（three-tier architectures）。二层架构模式重要应用于客户/服务器系统，对服务器和客户的功能要进行分配。三层架构模式分为一个客户计算机、一个应用服务器计算机和一个数据服务器计算机。下面对这些模式进行详细介绍。

1．单机系统的层次模型

如图 5-7 所示，在层次模型的架构中，一台计算机中的功能类被分配在各个层中。最上层是**输入输出层**，即**用户接口**。用户接口连接着**加工处理层**。加工处理层含有控制对象和实体对象。

2．客户/服务器系统的层次模型

在一个两层客户/服务器架构的框架中，下面的这些层要分配到两台分离的计算机中：

- 输入输出层；
- 加工处理层；
- 数据持久存储层。

因为要分配到两台计算机上，所以每台计算机都额外需要一个用于数据传输的通信层，如图 5-8 所示。

根据加工处理层处在服务器端计算机还是客户端计算机，可以区分为瘦客户端架构（图 5-9）或胖客户端架构（图 5-10）。

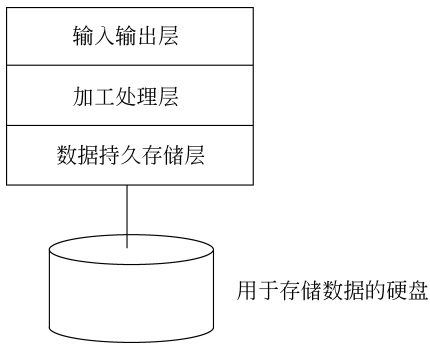

图 5-7　单机系统的层次模型

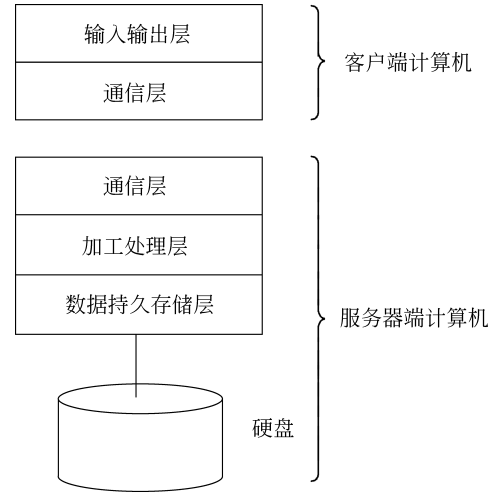

图 5-8　客户/服务器系统中计算机的层次模型

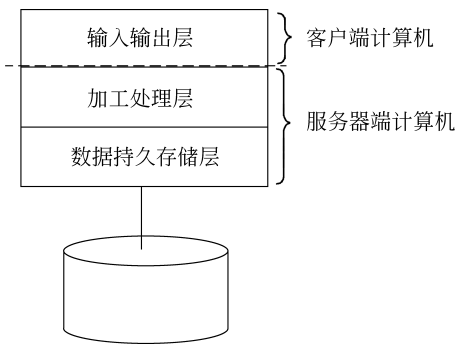

图 5-9　瘦客户端[1]

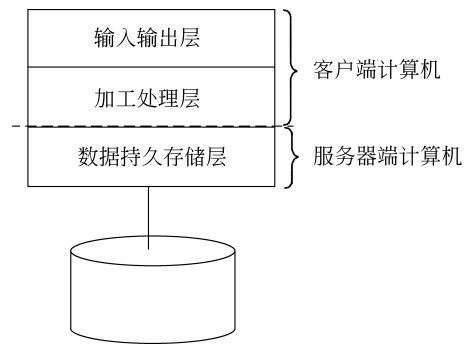

图 5-10　胖客户端[2]

每台计算机都要额外增加**通信层**，用于计算机之间的通信。通信层要分别加到图 5-9 和图 5-10 中虚线的上方和下方。

3．三层架构

在一个**三层客户/服务器架构**的框架中，每台独立的计算机要包含下面这些层：

- 输入输出层；
- 加工处理层；
- 数据持久存储层。

系统分层后被分布在不同的计算机上，所以双方都需要通信层，如图 5-11 所示。

4．二层架构和三层架构的优点

二层架构中，使用瘦客户端的优点是，所有功能都在服务器端，服务器处在中心位置，可以面对不同的用户。

1　在瘦客户端架构中，客户端计算机只能使用输入输出层中的对象。

2　在胖客户端架构中，服务器端计算机只包含数据持久存储层。

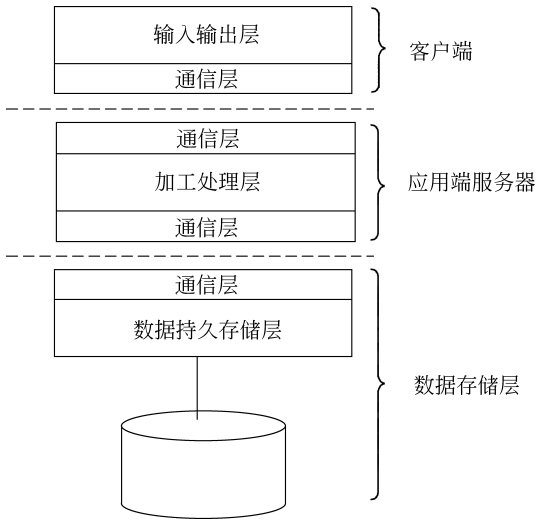

图 5-11 带有通信的三层架构模式

三层架构的优点是它的可扩展性。如果必要，应用端服务器的计算机可以被复制。数据以单一的形式集中在一个位置或者分布在多个计算机的分区中。

5.1.5.4 ISO/OSI 层次模型

层次模型的一个典型实例是 ISO/OSI 层次模型[Ros90]。它用于计算机通信系统，即网络协议。通信系统就是借助于网络系统实现相互间的通信的应用。ISO/OSI **参考模型**构造了一个 7 层的通信系统，每一层负责解决特定的问题，每一层问题的解决都要借助于其下面的层。

- **应用层**（**application layer**）。是参考模型中的最高层。它是应用进程和通信系统之间的接口。它提供了用于通信的整体帮助服务或者特殊的通信服务，如文件传输、报文处理系统（MHS）等。
- **表示层**（**pressentation layer**）。其任务是把数据传输的语法让双方都可以理解。数据传输双方的计算机具有相同的本地语法（例如，双方生产厂家相同），这样它们就可以在本地语法的基础上统一传输语法。对于其他情况，就必须采用中性传输语法：抽象语法标记 1/基本编码规则[1]（ASN.1/BER）。该层的其他任务是把本地语法转换成传输语法或者是把传输语法转换成本地语法。
- **会话层**（**session layer**）。用于控制不同系统间的进程同步。
- **传输层**（**transport layer**）。用于终端系统**进程间端到端的链接**，并且把信息分解成包或者把包组装还原成信息。
- **网络层**（**network layer**）。负责搭建网络中**通信终端系统之间端到端的连接**。因此它负责从发送方到接收方之间的**路由选择**。
- **数据链路层**（**data link layer**）。提供了系统间点到点之间的安全连接。它在**比特序**

1 中性语法就是抽象语法标记 1（ASN.1）。它构建了传输语法的第一部分，涉及数据类型。第二部分是 ASN.1 的基本编码规则（BER），它包含了在传输语法中数据编码的一组标准规则。

列上进行**纠错**。

- **物理层（physical layer）**。用于建立系统间**传输“比特”**的不安全连接。当发生错误时，连接就会被中断。

图 5-12 是通信系统中相互作用的两个计算机：

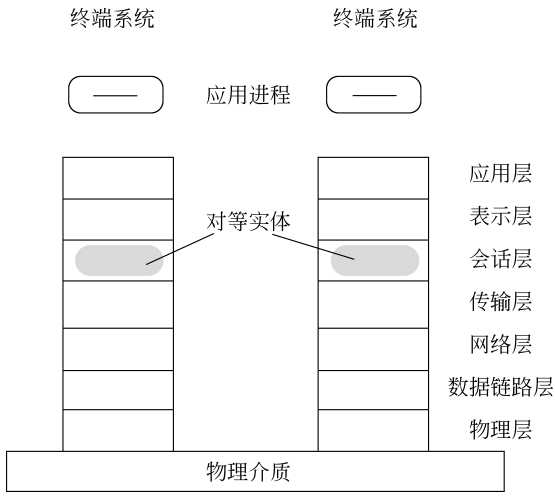

图 5-12　OSI 参考模型的基本结构

在协议栈的高层中，用户功能占主导地位，在低层中则是控制硬件的协议。协议栈中高层的行为建立在低层行为的基础之上。

如果一个层要实现对外应有的反应，在它内部的计算机中应该有一定的工作单位（实例）用于实现对外的行为。OSI 模型描述了一个**计算机中各层里面的工作单位（实例）**的逻辑模型，还确定了每个工作单位对外的功能。每个符合 OSI 模型的计算机在进行通信时应该具有相同的实例。每一层中的对等实例，即发送和接收方相对应的层实例共同完成这个层的任务。

对于 ISO/OSI 系统，下面介绍的场景非常重要：

两个终端系统相互通信，其中第 *n* 层的两个实例通过第 *n* 层的协议进行交流。这一层的消息在它所属计算机的协议栈中传递到传输介质，在接收方继续向上传递到第 *n* 层。在发送方，每经过一层就会增加附加的信息，在接收方相应的层会去除对应的信息，如图 5-13 所示。

这里并不能看出为什么 ISO/OSI 模型划分为 7 个层次。TCP/IP 的架构中定义了 4 个层次，而在 DECnet[1]中是 8 个层次。IBM 公司的 SNA[2]同样也有 8 个层次，但是与 DECnet 划分的层次不同。层次的数量并不重要，重要的是用户（不同的厂商）根据相同的标准实现标准化的层。因此可以确保不同制造商的计算机之间可以相互通信。

1　DECnet 是 Digital Equipment Corporation 公司的通信系统。

2　SNA（Systems Network Architecture）是 IBM 公司 20 世纪 70 年代开发的网络系统架构。