第 3 章　软件设计基础

软件设计是软件工程的重要阶段，它将软件的编码往后延长了一个阶段，体现了软件工程“推迟实现”的原则。当需求工程完成以后，系统已明确描述“软件必须做什么”这一问题。因此在设计阶段需要回答“软件怎么实现”的问题。

软件设计也曾被认为是“编程”，而缺乏对软件设计工程化的指导，缺乏对软件设计过程的管理，缺乏对软件设计质量的评估。随着软件工程不断发展和认识的加深，逐步出现了一些实用的软件设计方法，这些方法包括软件设计的原理、过程和工具，提高了软件质量。

3.1　软件设计概述

软件设计的基本概念于 20 世纪 60 年代末被陆续提出。作为软件开发阶段的开始，它为软件“大厦”规划施工“蓝图”，提供“如何实现软件需求”的决策。这些决策对如何划分系统、如何组织功能，如何体现性能，如何存储数据，如何分离功能与数据，如何有效统一完成任务，如何定义设计质量标准等一系列问题做出回答。

软件设计的目标就是要构造一个高内聚、高可靠性、高可维护性和高效的软件模型，为提高软件质量提供坚实的基础。

3.1.1　软件设计与软件需求

软件设计是需求工程的后续阶段，它根据描述的信息域需求，包括功能需求、性能需求、领域需求、数据需求等的定义，进行数据设计、体系结构设计、界面设计和过程设计，并通过这 4 个层面的设计，将现实世界的具体问题（需求）转换为信息设计的逻辑问题（设计方案）。

软件设计的依据是需求规格说明和数据规格说明，并将它们映射为软件设计的各部分内容。图 3-1 描述了两者间的映射关系。

体系结构设计是把握软件系统的整体架构，从宏观的角度设计软件各主要组成部分间的关系，不拘泥于具体实现细节，它是设计的核心部分。数据设计侧重于对数据文件、数据结构、数据对象等实体的设计。界面设计是对系统外部交互接口的定义，包括人机交互、软件与外部系统的数据交换，也是控制系统运行的工具。过程设计是将软件需求的描述性信息转换为信息领域中结构化、半结构化的过程性描述。

图 3-1 所示的映射关系说明了实现软件设计各部分的内容与需求分析各部分间相关联的要点：

（1）围绕数据字典。数据字典是需求分析的核心，数据设计是软件设计的基础。依

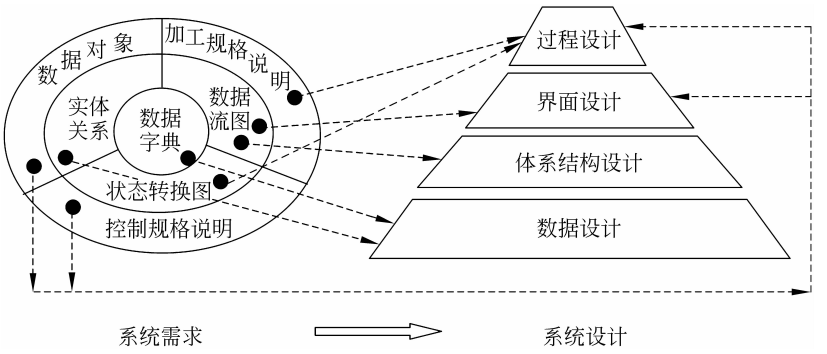

图 3-1　软件需求与软件设计的映射关系

据数据字典结构化定义，完成数据库、数据文件及数据结构的设计。

(2) 研究、分析数据流图。面向数据流的设计将数据流图映射为软件结构图。软件结构的映射有两种方式：变换型和事务型。最终通过启发式规则改进软件结构图，直至得到优化的软件逻辑结构。

(3) 数据流在不同变换模块中转换，提供了模块接口设计、人机交互界面设计的数据结构或控制信息，进一步提高了模块的独立性。

(4) 依据控制规格说明和状态转换图、加工规格说明，进行过程设计，描述模块内部具体算法流程。

软件设计是软件开发阶段的起始过程，关系到整个软件开发阶段的质量。图 3-2 所示的开发阶段信息流描述了软件设计从软件需求到软件编码的过程，起到承上启下的作用。

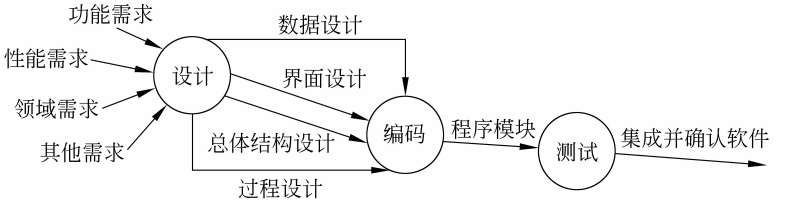

图 3-2　开发阶段信息流

图 3-2 回答了“软件设计不是编码”这一困扰人们的问题。编码只是设计的具体实现过程，代码把设计人员的思路物理地表现出来。只有良好的软件设计，才是代码实现并优化的基础和根本，否则编码就是“浮沙地上盖高楼”，不仅稳定性差，难以维护，严重时还会导致系统的失败。

3.1.2　软件设计的任务

软件设计主要回答软件“如何做”的问题。通过软件需求规格说明，建立软件设计模型，并通过设计模型来确定是否满足需求，是否达到设计质量标准。通过对软件需求的设

计，往往得到多个设计方案。这些设计方案的选择会最终影响软件实现的成败，也会影响软件维护。选定的设计方案要将软件的体系结构、数据结构、数据文件、系统内部和外部间的接口、算法的过程描述等相关部分详细定义，同时还要考虑实现时在技术上、空间上、时间上的可行性。

从软件工程的角度，一般把软件设计分为概要设计和详细设计两个子阶段，如图 3-3 所示。

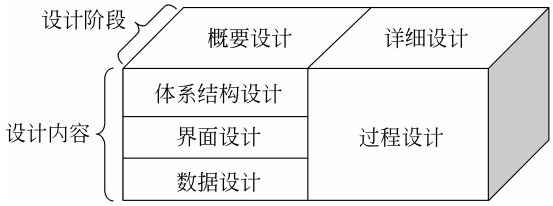

图 3-3　软件设计阶段的划分及任务

由此可以看到，软件设计的任务是完成概要设计和详细设计子阶段的任务，包括了体系结构设计、界面设计、数据设计和过程设计。

1. 概要设计

概要设计也称总体设计，主要任务是基于数据流图和数据字典，确定系统整体软件结构，划分软件体系结构的各子系统或模块，确定它们之间的关系。确切地说，概要设计是要完成体系结构设计、界面设计和数据设计。

- 体系结构设计：确定各子系统模块间的数据传递、调用关系。在结构化设计中，体现为模块划分，并通过数据流图和数据字典进行转换。在面向对象设计中，体现为主题划分，主要确定类及类间关系。
- 界面设计：包括与系统交互的人机界面设计，以及模块间、系统与外部系统的接口关系。在结构化设计中，根据数据流条目，定义模块接口、全局的数据结构。在面向对象设计中，定义关联类、接口类、边界类等，既满足人机交互界面数据的统一，也完成类间数据的传递。
- 数据设计：包括数据库、数据文件和全局数据结构的定义。在结构化设计中，通过需求阶段的实体关系图、数据字典建立数据模型。在面向对象设计中，通过类的抽象与实例化，以及类的永久存储设计，完成数据设计过程。

2. 详细设计

详细设计的任务是在概要设计的基础上，具体实现各部分的细节，直至系统的所有内容都有足够详细的过程描述，使得编码的任务就是将详细设计的内容“翻译”成程序设计语言。确切地说，详细设计的任务是完成过程设计。

过程设计包括确定软件各模块内部的具体实现过程及局部数据结构。在结构化设计中，模块独立性约束了数据结构与算法相分离的情况，使得两者在设计时务必有局部性，减少外部对两者的影响。在面向对象设计中，类的封装性较好地体现了算法和数据结构

的内部性。类的继承性提供了多个类(类家族)共同实现过程设计的机制。

根据软件项目的规模和复杂度,概要设计和详细设计既可以合并为软件设计阶段,也可以反复迭代,直至完全实现软件需求内容。图 3-4 给出了软件设计的工作流。

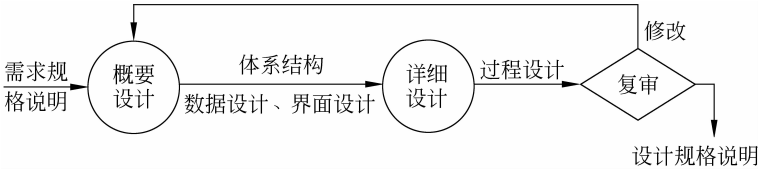

图 3-4　软件设计的工作流

3.1.3　软件设计的原则

随着软件技术的不断进步,一些良好的设计原则不断地被提出,并指导着软件设计过程,确保软件质量。

1. 分而治之

分而治之是解决大型、复杂程度高的问题时所采用的策略。把大问题划分成若干个小问题,把对一个大问题的求解转换为对若干个小问题的解答,这样就极大地降低了问题的复杂度。模块化是在软件设计上实现分而治之思想的技术手段。在结构化设计中,模块可以是函数、过程,甚至是代码片段。在面向对象设计中,类是模块的主要形式。

2. 重用设计模式

重用是指同一事物不做修改或稍做改动就能多次使用的机制。由于概要设计完成的是系统软件结构,因而重用的内容是软件设计模式。软件设计模式针对一类软件设计的过程和模型,而不是面对一次具体的软件设计。通过重用设计模式,不仅使得软件设计质量得到保证;而且把资源集中于设计中的新流程、新方法中,并在设计时更进一步考虑新流程、新方法在将来的重用。

3. 可跟踪性

软件设计的任务之一就是确定软件各部分间的关系。因为设计系统结构就是要确定系统各部分、各模块间的相互调用或控制关系,以便在需要修改模块时能掌握与修改模块有关的其他部分,并正确地追溯问题根源。

4. 灵活性

设计的灵活性是指设计具有易修改性。修改包括对已有设计的增加、删除、改动等活动。发生修改的原因一是用户需求发生变更;二是设计存在缺陷;三是设计需要进行优化;四是设计利用重用。软件设计灵活性主要通过系统描述问题的抽象来体现。抽象是

对事物相同属性或操作的统一描述，具有广泛性。因此，系统设计和设计模式的抽象程度越高，覆盖的范围就越大。如“鸟”对“麻雀”的抽象，既能体现麻雀能飞的特性，也覆盖了其他鸟类的说明。但抽象是一把双刃剑，过度地抽象反而会引起理解和设计上的困难。例如用“生物”去抽象“麻雀”实体，则作为鸟的很多特征将难以在“生物”中定义。

5. 一致性

一致性在软件设计方法和过程中都得到体现。在软件设计中，界面视图的一致性保证了用户体验和对系统的忠诚度，如 Windows 操作系统的界面，虽历经多个版本的变更，但用户操作方式基本没有改变。用统一的规则和约束规范模块接口定义，确保编码阶段对接口和数据结构的统一操作，减少数据理解上的歧义，使得软件质量得到保证。在软件设计过程中，团队已成为软件开发的基本组织形式。不同人员集体完成同一软件项目，保持开发进度的一致性是项目成功的关键之一。

3.2 软件体系结构设计

当提及体系结构时，容易引发与建筑物的物理结构做比较。在修建建筑物时，要兼顾它的外观与内部的统一。建筑物外观除了自身设计外，还会考虑与功能相结合，与周围环境相融合，并能充分利用建筑物内部空间。因此，这样的设计理念和过程不仅指导当前建筑项目的设计，而且能为将来的建筑设计所共享。那么，软件体系结构设计是否也具有这样的特征呢？

3.2.1 体系结构设计概述

软件体系结构为软件系统设计提供了一套关于数据、行为、结构的指导性框架，该框架提供了描述系统数据、数据间关系的静态特征，也对数据的操作、系统控制和通信等活动提供了具有动态特征的描述过程。系统静态特征体现了系统的组织结构，系统动态特征则体现系统操作流程的拓扑过程，共同构成设计决策的基本指导方针。具有良好设计的体系结构具有普适性，能满足不同的软件需求。

体系结构设计是软件设计的早期活动，它的作用集中在两点：

- 提供软件设计师能预期的体系结构描述。例如提起浏览器/服务器(B/S)模式，多层架构、数据库存储、客户端、逻辑服务器等一系列描述就浮现在设计师的脑海里。
- 数据结构、文件组织、文件结构体现了软件设计的早期抉择，这些抉择将极大地影响着后续的软件开发人员，影响着软件产品的最后成功。

下面介绍几种常见的、被广泛使用的软件体系结构模型。

3.2.2 以数据为中心的数据仓库模型

数据仓库模型是一种集中式模型。早期的数据是应用级的数据库或数据文件。随着应用的不断扩展，数据规模越来越大，数据形式也越来越多样和复杂。如何有效组织这些数据，这些数据如何高效提供信息成为软件设计时首先关注的问题。数据仓库模型就是在这些应用发展背景下提出的。图 3-5 所示为集中式数据仓库模型的一个抽象。

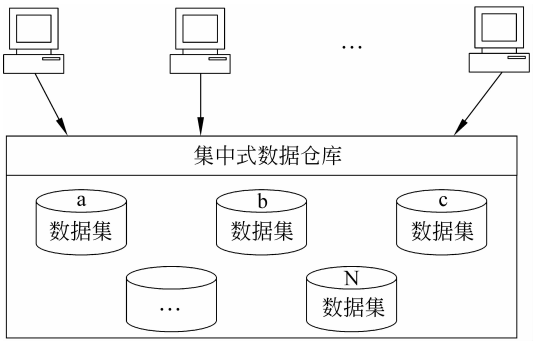

图 3-5 集中式数据仓库模型的一个抽象

数据仓库模型是能独立提供数据服务的封闭式数据环境。它不单独集成到某一应用系统中，而是为具体的应用系统提供服务。这些服务既有通用的公共服务，也有专门设计的领域服务。例如公共搜索引擎提供的检索服务，既有基于关键词的通用检索，也有为企业服务的垂直检索。而无论何种服务，数据仓库模型后端都是统一的集中式数据管理。

数据仓库模型有着明显的优点：

(1) 数据统一存储和管理，确保了数据的实时性。

(2) 数据仓库对数据复杂性的统一封装有利于数据共享。

(3) 采用黑板模型，与某类数据有关的应用系统能及时获取数据。

(4) 采用数据订阅推送模型，应用系统在有数据更新时，能自动获得数据，而不用采取询问方式，这就提高了数据管理效率。

(5) 各应用系统间仅通过数据仓库完成数据交换，在功能上没有关联，增加、删除应用系统及其部分功能，将不会影响其他应用系统的正确运行。

但集中式数据仓库也存在着不足：

(1) 为了对数据仓库数据进行操作，不同应用系统的数据视图必须统一，否则难以达到数据共享的目的，但这不可避免地会降低各应用系统的效率。因为统一的数据视图需要通过各应用系统进行转换。同样，面对不同的数据提供统一的访问接口，也增加了数据仓库设计的复杂性，降低了数据传递的效率。

(2) 如果应用系统的数据结构发生改变，就需要单独设计数据适配器，以实现新的结构与数据仓库在数据上的匹配。这不仅增加了应用系统设计的复杂度，而且有时甚至难以完成这样的数据匹配。

(3) 随着网络技术的发展,数据共享带来的访问控制的复杂性、安全性、效率、备份、存储、恢复策略等一系列问题,影响了仓库模型的有效利用。

3.2.3 客户端/服务器模式的分布式结构

集中式数据仓库模型在带来数据一致性访问优势的同时,也造成在网络环境下难以分布应用的缺陷。

“网络就是计算机”,“网络就是共享”。但随着中心服务器运算压力不断加大,对维护、更新升级都带来了极大的困难和成本的增加,而发出请求的客户端(现在还包括智能终端)运算能力和资源却闲置,造成极大浪费。因此,挖掘网络计算能力,共享计算的云计算等新的分布式计算模型被提出。

分布式结构模型是充分利用、整合网络中计算机各自的计算能力,从而提高整个网络系统运行的能力和效率。早期分布式模型采用两层结构,如图 3-6 所示。

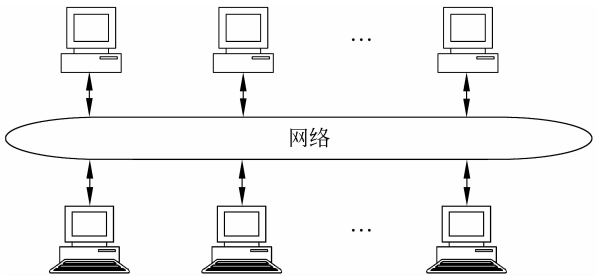

图 3-6 两层的分布式结构

两层结构由两个典型的应用组成:客户端实现用户界面视图,服务器端完成系统逻辑功能和数据访问。系统的运行过程通过“请求——响应——结果”模式来实现,这实质上是一种远程调用方式。这种设计模式通过网络的屏蔽,允许客户端与服务器端的软硬件配置不同,体现了分布式模型的灵活性。然而对应用系统来说,却隐含多层的逻辑划分:应用系统交互、系统逻辑功能和数据访问等。

以目前应用较多的三层网络设计模式为例,用两层分布式设计模式映射三层逻辑,产生如图 3-7 所示的不同方法。

图 3-7(a)被称为“胖客户端”模型,即应用系统逻辑全部在客户端,服务器只提供数据访问。这种设计模型降低了服务器的计算压力,减轻了网络带宽的拥塞。但系统更新烦琐,需通知各客户端自行更新。

图 3-7(c)被称为“瘦客户端”模型,即应用系统逻辑全部在服务器,客户端只提供用户界面进行访问。这种设计模型可实现更新的一致性而不影响客户端访问。但这会造成服务器的计算压力,同时也增加了网络带宽的负载。

图 3-7(b)是图 3-7(a)和图 3-7(c)的折中,与客户端计算有关的逻辑放在客户端完成,需要频繁访问数据的逻辑部署在服务器端,这样不仅减轻了服务器运算压力,也有效节约了网络带宽资源。但对于客户端和服务器共同完成任务的操作,增加了系统部署和控制

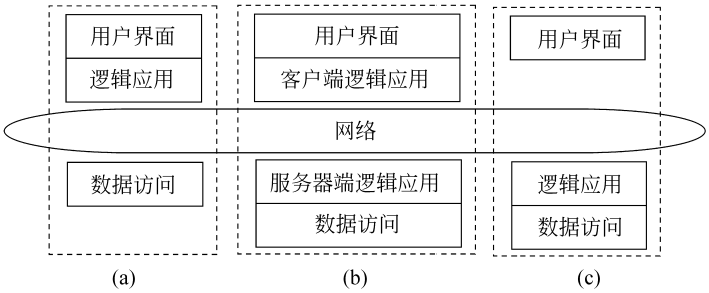

图 3-7　二层分布式模型对应三层应用逻辑的映射

的复杂性。

目前，针对多层逻辑应用提出了多层分布式设计模型，如图 3-8 所示。

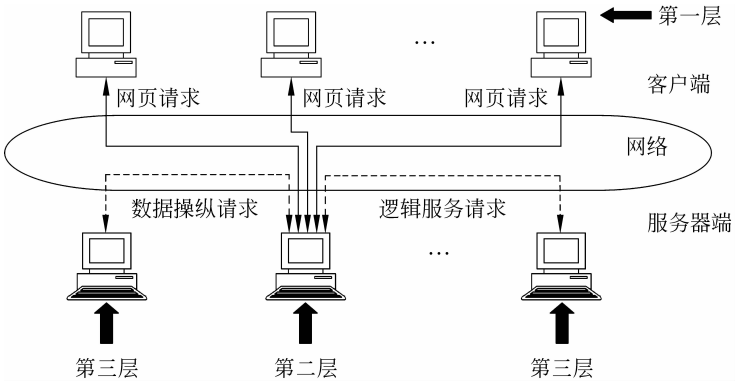

图 3-8　多层客户端/服务器模型

三层客户端/服务器模型体现了系统网络的开放性、灵活性。同时在服务器端，把中间逻辑部分通过网络映射成多层分布式设计模型，以此减轻 Web 服务的访问压力。

总的来说，分布式结构设计具有以下特点：

(1) 共享：实现了数据共享，云计算的提出还能进一步实现计算共享。

(2) 异构性：客户端/服务器允许软硬件配置不同。

(3) 开放性：只要符合互联网协议，任何计算机、局域网、智能设备和物品等都可连入互联网。

(4) 易修改性：由于用户界面、系统逻辑和数据访问分布的不同，各部分具有较强独立性，易于系统的修改和维护。

(5) 透明性：分布式结构中仅需要知道服务器的服务位置，而对后端的逻辑实现、数据存储、数据访问等不必清楚其架构和访问方式。

当然，分布式结构也存在一些不足：

(1) 复杂性：显然，集中式的数据仓库模型共享虽然带来了一些问题，但其控制结构相对简单。而分布式管理要复杂得多，它面对网络通信、服务器端分层等问题的管理，都充满了风险和挑战。

(2) 安全性：身份验证困难。客户端的访问是问答模式，对客户端的响应由服务器提供服务，因而难以验证客户端真实身份。这给病毒、流氓插件等不良软件带来了可乘之机。

(3) 运行状态难以确定。特别是网络通信出现故障时，提交的信息是否有效，是否得到正确响应，都困扰着分布式模型的发展和应用。

3.2.4 层次模型

客户端/服务器模型是一种松散的层次模型，即客户端向服务器发送请求，服务器响应请求。目前，由于网络的异构性，允许客户端和服务器是对等的，即服务器也可以作为客户端发送请求。客户端也能成为服务器，响应其他客户端的请求。

与客户端/服务器不同的是，层次模型将系统划分为若干层次，每个层次提供单向服务，如底层向顶层提供服务。这种设计模式适合增量式开发。系统由底层开始，逐步向上层完成实现，进而完成整个系统。典型的层次模型是国际标准化组织(ISO)的开放式系统接口(OSI)七层网络参考模型，如图 3-9 所示。

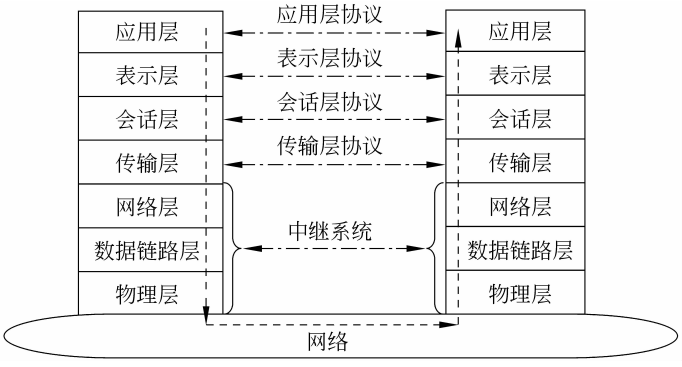

图 3-9 ISO 的 OSI 七层网络参考模型

当网络中的两端完成网络会话时，并非从左端的应用层直接发送到右端的应用层，而是从左端应用层开始，逐层向下传递，利用下层提供的服务对数据进行逐层封装，直到物理层。之后，通过网络传输到右端的物理层。由于网络协议由系统自动完成连接和通信，因而对上层的应用来说感受不到上述层次的传递过程，因此也可以直观地认为七层网络各层间是直接对应关系。

2.4.3 节中提到的“儿童自然语言对话系统”也是一种层次模型，它通过底层向上层提供统一的数据访问接口，以及建立在数据库信息之上的推理过程，体现了系统自顶向下对数据的分解，以及自底向上对数据的生成过程。图 3-10 描述了儿童自然语言理解系统的层次模型。

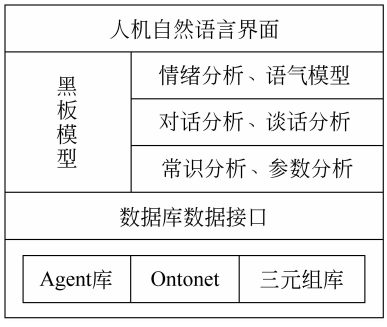

图 3-10 儿童自然语言理解系统的层次分解

3.3 模块化设计

模块是程序语句的集合，它拥有独立的命名，明确的输入、输出和范围。程序设计中的函数、过程、类、库等都可作为模块。模块用矩形框表示，并用模块名称来命名。

3.3.1 软件模块化与分解

对于整个软件系统来说，设计人员不是把它作为一个问题来整体解决，而是把它的全部功能按照一定的原则划分成若干个模块。如果某个模块仍难以理解或实现，则把它再进行划分，得到更小、功能更简单的模块。如此往复，直至所有模块都可解，这个过程就是软件模块化设计。图 3-11 抽象表示了软件模块分解过程。

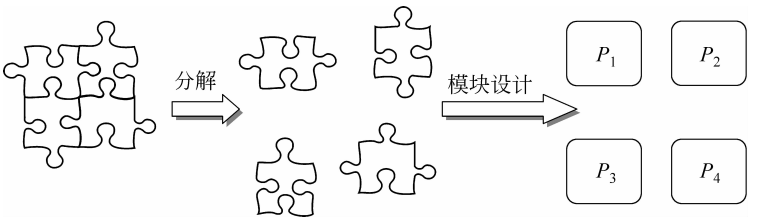

图 3-11 软件模块分解示意

从图 3-11 中可以看到，通过分解，系统被分解成各模块，易于“各个击破”，同时降低了问题的复杂性，使得软件结构清晰，便于阅读和理解，提高软件的可理解性和可维护性。

假设 $C(X)$为问题 X 的复杂度，$E(X)$为解决问题 X 的工作量。若存在问题 P_1 和 P_2，且 $C(P_1)>C(P_2)$，显然 $E(P_1)>E(P_2)$。由经验有 $C(P_1+P_2)\geqslant C(P_1)+C(P_2)$，则 $E(P_1+P_2)\geqslant E(P_1)+E(P_2)$。说明将问题 (P_1+P_2)分解为两个子问题 P_1 和 P_2，问题的工作量和复杂度将会降低。

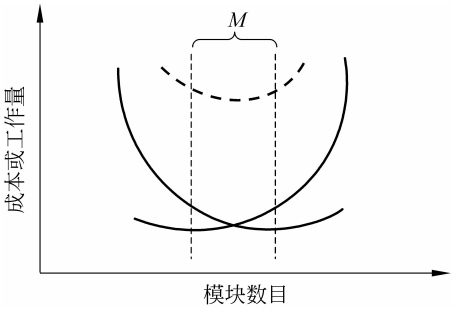

图 3-12 工作量和模块分解的关系

通常情况下，问题总的复杂度会随着模块分解而趋于减小，工作量也同时减少。但如果模块分解过程无限进行下去，不仅问题的复杂度不会减少，相反还会增加。图 3-12 表现了这一关系。

出现这种情形的原因，是随着模块数目的增加，模块间接口的复杂度也会增加。这样就会增加处理接口定义和它们之间调用关系的设计和实现的工作量。图 3-12 所示的 M 区间是一个合适的模块分解范围，它同时兼顾了对问题的分解和模块间设计与实现的工作量。