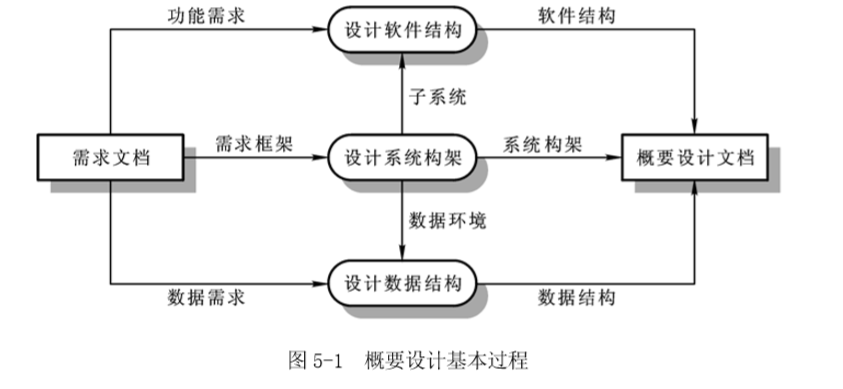
# [软件工程之软件概要设计](https://www.cnblogs.com/youcong/p/9500921.html)

在完成对软件系统的需求分析之后，接下来需要进行的是软件系统的概要设计。一般说来， 对于较大规模的软件项目，软件设计往往被分成两个阶段进行。首先是前期概要设计，用于确 定软件系统的基本框架；然后是在概要设计基础上的后期详细设计，用于确定软件系统的内部 实现细节。 概要设计也称总体设计，其基本目标是能够针对软件需求分析中提出的一系列软件问题， 概要地回答问题如何解决。例如，软件系统将采用什么样的体系构架、需要创建哪些功能模块、 模块之间的关系如何、数据结构如何？软件系统需要什么样的网络环境提供支持、需要采用什 么类型的后台数据库等。 应该说，软件概要设计是软件开发过程中一个非常重要的阶段。可以肯定，如果软件系统 没有经过认真细致的概要设计，就直接考虑它的算法或直接编写源程序，这个系统的质量就很 难保证。许多软件就是因为结构上的问题，使得它经常发生故障，而且很难维护。

一、概要设计过程和任务

1.设计过程

概要设计基本过程如图 5-1 所示，它主要包括三个方面的设计。首先是系统构架设计，用 于定义组成系统的子系统，以及对子系统的控制、子系统之间的通信和数据环境等；然后是软 件结构和数据结构的设计，用于定义构造子系统的功能模块、模块接口、模块之间的调用与返 回关系，以及数据结构、数据库结构等。



 概要设计要求建立在需求分析基础之上，软件需求文档是软件概要设计的前提条件。只有这样，才能使得开发出来的软件系统最大限度地满足用户的应用需要。 实际上，概要设计的过程也就是将需求分析之中产生的功能模型、数据模型和行为模型等 分析结论进行转换，由此产生设计结论的过程。在从分析向设计的转换过程中，概要设计能够 产生出有关软件的系统构架、软件结构和数据结构等设计模型来。这些结论将被写进概要设计 文档中，作为后期详细设计的基本依据，能够为后面的详细设计、程序编码提供技术定位。 需要注意的是，概要设计所能够获得的还只是有关软件系统的抽象表达式，需要专心考虑 的是软件系统的基本结构，至于软件系统的内部实现细节如何，则被放到以后详细设计中去解 决。例如模块，概要设计中的模块只是一个外壳，虽然它有确定的功能边界，并提供了通信的 接口定义，但模块内部还基本上是空的，诸多具体的功能加工细节则必须等到详细设计完成以 后才能确定下来。因此，在有关软件设计的全部工作中，概要设计所提供的并不是最终设计蓝 图，而只是一份具有设计价值的具体实施方案与策略，用于把握系统的整体布局。尽管概要设 计并不涉及系统内部实现细节，但它所产生的实施方案与策略将会最终影响软件实现的成功与 否，并影响到今后软件系统维护的难易程度。

2.设计任务

概要设计阶段的任务既包括技术方面的，也包括管理方面的，具体说来，主要有以下几个方面：

（1）制定规范

具有一定规模的软件项目总是需要通过团队形式实施开发，例如，组成一个或几个开发小组 来承担对软件系统的开发任务。为了适应团队式开发的需要，在进入软件开发阶段之后，首先应 该为软件开发团队制定在设计时应该共同遵守的规范，以便协调与规范团队内各成员的工作。

概要设计时需要制定的规范或标准包括：

　　　　　　a.需要采用的管理规则，例如操作流程、交流方式、工作纪律等。

　　　　　　b.设计文档的编制标准，包括文档体系、文档格式、图表样式等。

　　　　　　c.信息编码形式，硬件、操作系统的接口规约，命名规则等。

　　　　　　d.设计目标、设计原则。

（2）系统构架设计

系统构架设计就是根据系统的需求框架，确定系统的基本结构，以获得有关系统创建的总 体方案。其主要设计内容包括：

　　　　 a.根据系统业务需求，将系统分解成诸多具有独立任务的子系统。

　　　     b.分析子系统之间的通信，确定子系统的外部接口。

　　　　 c.分析系统的应用特点、技术特点以及项目资金情况，确定系统的硬件环境、软件环境、 网络环境和数据环境等。

　　　　 d.根据系统整体逻辑构造与应用需要，对系统进行整体物理部署与优化。 很显然，当系统构架被设计完成之后，软件项目就可按每个具有独立工作特征的子系统为 单位进行任务分解了，由此可以将一个大的软件项目分解成许多小的软件子项目。

（3）软件结构设计

软件结构设计是在系统构架确定以后，对组成系统的各个子系统的结构设计。例如，将系统进一步分解为诸多功能模块，并考虑如何通过这些模块来构造软件。

软件结构设计主要内容包括：

　　　　a.确定构造子系统的模块元素。

　　　　b.根据软件需求定义每个模块的功能。

　　　　c.定义模块接口与设计模块接口数据结构。

　　　　d.确定模块之间的调用与返回关系。

　　　　e.评估软件结构质量，进行结构优化

（4）公共数据结构设计

概要设计中还需要确定那些将被许多模块共同使用的公共数据的构造。例如，公共变量、 数据文件以及数据库中数据等，可以将这些数据看作为系统的公共数据环境。 对公共数据的设计包括：

　　　　a.公共数据变量的数据结构与作用范围。

　　　　b.输入、输出文件的结构。

　　　　c.数据库中的表结构、视图结构以及数据完整性等。

（5）安全性设计

系统安全性设计包括：操作权限管理设计、操作日志管理设计、文件与数据加密设计以及 特定功能的操作校验设计等。概要设计需要对以上方面的问题作出专门的说明，并制定出相应 的处理规则。 例如操作权限，假如应用系统需要具有权限分级管理的功能，则概要设计就必须对权限分级管理中所涉及的分级层数、权限范围、授权步骤以及用户账号存储方式等，从技术角度作出专门的安排。

（6）故障处理设计

软件系统工作过程中难免出现故障，概要设计时需要对各种可能出现的来自于软件、硬件 以及网络通信方面的故障作出专门考虑。例如，提供备用设备、设置出错处理模块、设置数据备份模块等。

（7）可维护性设计

软件系统在投入使用以后必将面临维护，如改正软件错误、扩充软件功能等。对此，概要设计需要作出专门安排，以方便日后的维护。例如，在软件中设置用于系统检测维护的专用模 块；预计今后需要进行功能扩充的模块，并对这些接口进行专门定义。

（8）编写文档

概要设计阶段需要编写的文档包括：概要设计说明书、数据库设计说明书、用户操作手册。 此外，还应该制定出有关测试的初步计划。 上述文档中，概要设计说明书是概要设计阶段必须产生的基本文档，涉及系统目标、系统 构架、软件结构、数据结构、运行控制、出错处理、安全机制等诸多方面的设计说明。

（9）概要设计评审

在诸多概要设计任务完成之后，应当组织对概要设计的评审。 概要设计评审内容主要包括:

　　　　a.需求确认：确认所设计的软件是否已覆盖了所有已确定的软件需求。

　　　　b.接口确认：确认该软件的内部接口与外部接口是否已经明确定义。

　　　　c.模块确认：确认所设计的模块是否满足高内聚、低耦合的要求，模块的作用范围是否在其控制范围之内。

　　　　d.风险性：该设计在现有技术条件下和预算范围内是否能按时实现。

　　　　e.实用性：该设计对于需求的解决是否实用。

　　　　f.可维护性：该设计是否考虑了今后的维护。

　　　　g.质量：该设计是否表现出了良好的质量特征

二、系统构架设计

大型的综合应用系统大都是由许多子系统组成的。一般说来，这些子系统能够独立运行， 有自己专门的服务任务，并可能需要部署在不同的计算机上工作。 应该说，组成系统的子系统具有一定的独立性，但子系统之间又有着联系。例如，有共同的数据源，相互之间需要通信，并可能需要协同工作。系统构架设计的任务就是根据需求规格 中的需求基本框架，把组成系统的这些子系统、子系统之间的关系、它们之间需要的数据通信 等确定下来，并把它们工作时所需要的设备环境、网络环境和数据环境等也一同确定下来，由 此对系统作出一个合理的、符合应用需要的整体部署。 需求分析中的需求框架是基于用户应用域建立的，概要设计时可以通过需求框架来映射系 统构架。例如，可以利用需求分析中的高层数据流图对系统基本工作流程的描述，来映射系统的基本结构，使得需求分析中对系统的逻辑描述，转换为概要设计中对系统的物理描述。 一般情况下，系统构架设计可以按照以下步骤进行。

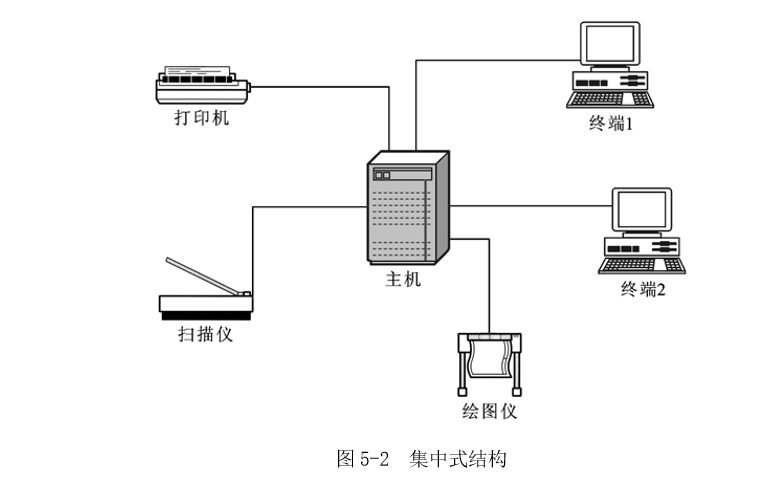
（1）定义子系统。根据需求分析中有关系统的业务划分情况，将系统分解成诸多具有独立任务的子系统。

（2）定义子系统外部接口。分析子系统之间的通信与协作，以获得对子系统外部接口的 定义。

（3）定义系统物理构架。根据系统的整体逻辑结构、技术特点、应用特点以及系统开发的 资金投入情况等，选择合适的系统物理构架，包括：硬件设备、软件环境、网络结构和数据库结构，并将子系统按照所选的物理构架进行合理部署与优化。 下面将介绍几种典型的系统构架。需要注意的是，任何一种结构都会有优点与缺点，尽管 是一些现在看来已经过时的结构也有它存在的现实价值。

1.集中式结构

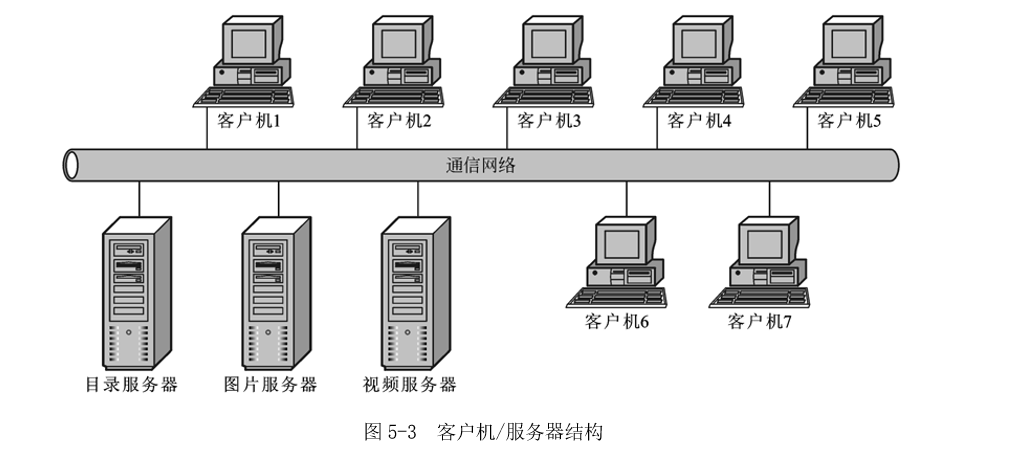
集中式结构是最传统的系统构架，系统由一台计算机主机和多个终端设备组成，其结构如 图 5-2 所示。 集中式结构的特点是系统中的全部软件资源都被集中安装在这一台主机上，包括：操作系统、数据库系统、应用系统和资源文件等。系统的智能处理器也被集中在主机上。用户则是通 过和主机连接的基本无智能的终端设备与系统进行通信。由于所有计算任务都通过主机完成，因此集中式结构对计算机性能有比较高的要求。早期应用中一般使用小型以上计算机提供比较 强大的主机计算支持，操作系统则一般是 Unix 系统，能够向外提供多用户服务。



应该说，集中式结构具有非常好的工作稳定性。另外，由于集中式结构是直接通过终端接 口实现多用户通信，因此系统还具有较高的安全保密特性。集中式结构优点是高稳定性和高安 全性。但集中式结构有较苛刻的设备要求，系统建设费用、运行费用都比较高，而且系统灵活 性不够好，系统结构不便于扩充。

2. 客户机/服务器结构

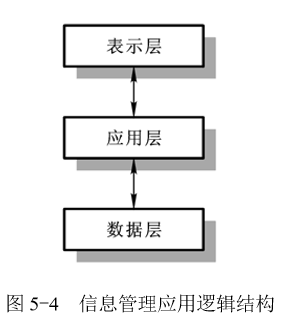
客户机/服务器结构是一种分布与集中相互结合的结构，系统依靠网络被分布在许多台不同的计算机上，但通过其中的服务器计算机提供集中式服务。 图 5-3 所示的是客户机/服务器结构的典型应用。这是一个提供视频信息和图片信息的多用户超文本应用系统，系统中有多个服务器，能够分别提供不同的信息服务。其中，视频服务器提供视频数据服务，传送的数据需要快速同步，但分辨率相对较低；图片服务器提供图片数据服务，数据需要以高分辨率发送；目录服务器则提供目录查询服务，能够支持各种查询，并 能够与超文本信息进行链接。 与集中式结构中的无智能终端不同，客户机/服务器结构中的客户机是具有智能的，需要 安装客户程序，并需要通过客户程序访问服务器。例如图 5-3 中的诸多客户机，它们可以通过一个用户界面客户程序对服务器进行访问，并可以通过这个用户界面程序显示从服务器返回的 图片或视频信息。 在客户机/服务器结构中，客户机是主动的，它主动地向服务器提出服务请求；而服务器 则是被动的，它被动地接受来自客户机的请求。一般说来，客户机在向服务器提出服务请求之 前，需要事先知道服务器的地址与服务；但服务器却不需要事先知道客户机的地址，而是根据 客户机主动提供的地址向客户机提供相应的服务。



客户机/服务器结构的优越性是结构灵活、便于系统逐步扩充。例如上面的多用户超文本 应用系统，也许用户最初只需要提供图片信息服务，因此系统初期开发时可以只创建图片服务 器，只是当用户需要提供视频信息服务时，才创建视频服务器。显然，这有利于应用系统的分 阶段实现与逐步扩充。另外，由于客户机/服务器中服务器可以采用高性能微机配置，因此其建 设成本、运行费用等也明显低于基于小型机的集中式结构。 需要注意的是，客户机/服务器结构是一种逻辑结构，客户机或服务器都只是角色概念。 因此可以将客户软件和服务软件安装在一台计算机上，而使这台计算机既担任客户机角色，又担任服务器角色。当然，也可以在一台计算机上安装多个服务软件，而使这台计算机承担多个 服务器角色，例如，它可以既是图片服务器，又是视频服务器。

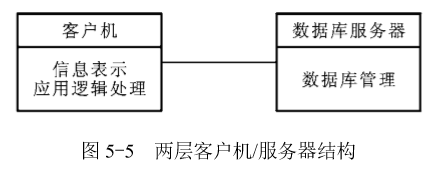
 3.多层客户机/服务器结构

客户机/服务器结构已被广泛应用基于数据库的信息服务领域，并演变出了多层客户机/服 务器结构。 图 5-4 所示是一个有关信息管理应用的逻辑结构。这是一个三层逻辑结构，其中的表示层 是指对用户的信息服务；应用层是指对数据的应用逻辑加工； 数据层是指系统中的数据库管理。在集中式系统中，以上各个不同层面的元素都被部署在一台主机上，这时或许没有必要在它们之间清楚地划分边界。但值得注意的是，当采用客户机/ 服务器结构时，这些元素却有可能要被分布到不同的计算机上 去，因此必须在它们之间给出一个清楚的边界。



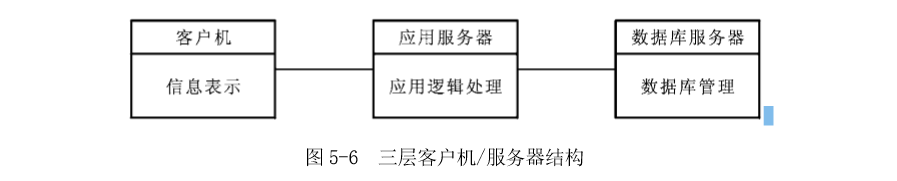
（1）两层结构

两层客户机/服务器结构是将信息表示与应用逻辑处理都 放在了客户机上，而服务器只需要管理数据库事务，其结构如 图 5-5 所示。这时的客户机一般被称为“胖客户机”。  
  
两层结构的优点是结构简单，技术上比较容易实现。而且应用软件基本上是在客户端工作， 因此非常方便客户端数据的计算与表现，能够有 效保证系统的工作性能。 两层结构的缺陷是在管理与维护上存在困 难。这时的客户端承担了信息表示与应用计算双 重任务，客户端程序复杂，并且被分散在许多不 同的客户计算机上。除非应用软件的功能非常稳 定或客户端口很少，否则当应用软件由于用户应用规则的变化，而不得不对其功能进行改造时， 系统中数目庞大的客户机群，将会使系统变更显得非常麻烦，需要很大的系统变更开销。



（2）三层结构

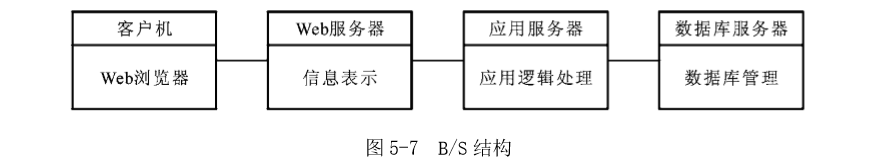
三层结构是鉴于两层客户机/服务器结构在管理与维护上存在的困难而专门提出的，这就 是使“胖客户机”减肥，使它尽量简单，变成“瘦客户机”。更具体地说，就是将“胖客户机” 中比较复杂，并且容易发生变化的应用逻辑部分提取出来，将它放到一个专门的“应用服务器” 上，由此产生的结构如图 5-6 所示。



显然，三层结构使信息表示、应用逻辑处理和数据库管理被自然地分成了三个独立的部分。 其中，应用逻辑处理是应用系统中的最易发生需求改变的部分，被放到了应用服务器上。与两层结构比较，三层结构给系统维护带来了很大的便利，当用户应用规则发生变化时，需要改变的不是数目庞大客户端，而是一个或少数几个应用服务器。但是，三层结构使实现软件的技术难度加大、开发周期延长，并且对服务器设备也有更高的要求。 需要指出的是，三层客户机/服务器结构也是逻辑结构，因此一个单一的服务器计算机可以既是应用服务器，又是数据库服务器。然而许多情况下，为了使系统具有更高的性能指标， 或使系统具有更好的稳定性，系统中大都分别设置有专门的应用服务器和数据库服务器。而在一些实际开发中，为了使应用服务器能够承担来自于许多客户机的计算请求，还往往需要根据软件系统的计算负荷状态，对应用服务器进行专门的配置设计，例如，按事务进行任务分工， 并在系统中设置多台专门应用服务器，它们分别完成一些特定的服务请求任务。

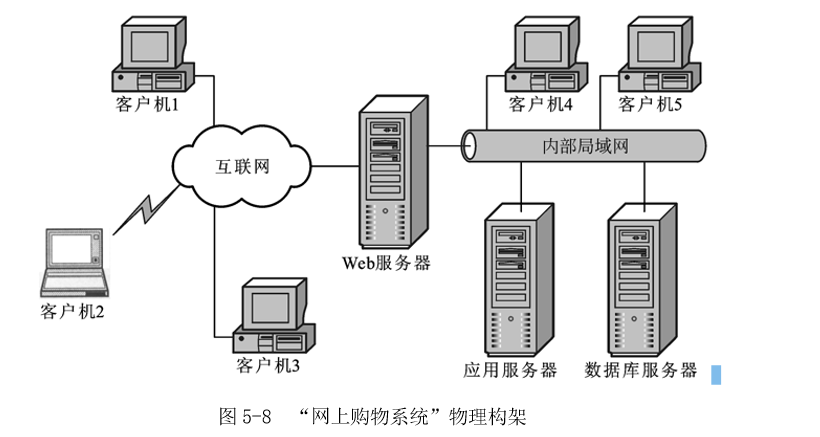
（3）B/S结构

B/S 结构是基于 Web 技术与客户机/服务器结构的结合而提出来的一种多层结构，其中的 B 是指 Web 浏览器（Browse），S 是指应用服务器与数据服务器（Server）。目前，这种结构已被 广泛应用于网络商务系统之中，例如网上银行、网上商店等。其结构图如图 5-7 所示。 B/S 结构将信息表示集中到了专门的“Web 服务器”上。因此，与三层客户机/服务器结构 比较，B/S 结构多了一层服务器。应该说，B/S 结构使客户端程序更加简化。这时的客户机上已 经不需要专门的应用程序了，只要有一个通用的 Web 浏览器（例如，Microsoft Internet Explorer），就可以实现客户端数据的应用。



 B/S 结构的优点是不需要对客户机进行专门的维护（客户机上没有专门的应用程序），特别 适合于客户机位置不固定或需要依靠互联网进行数据交换的应用系统。缺点是最终用户信息需 要通过 Web 服务器获取，并通过网络传送到客户机上，因此系统的数据传输速度以及系统的稳定性，都将明显低于三层客户机/服务器结构。 如同其他客户机/服务器结构一样，B/S 结构也是逻辑结构，因此一个单一的服务器计算机 可以既是 Web 服务器，又是应用服务器和数据库服务器。但如果需要使系统具有更高的性能或 更加稳定的运行状态，那么则有必要将 Web 服务、应用处理和数据管理从物理上分离开来，设 置专门的 Web 服务器计算机、应用服务器计算机和数据库服务器计算机。 在许多应用中，B/S 结构和三层客户机/服务器往往被结合起来使用。例如“网上购物系统”， 其面向消费者的购物操作一般采用的是 B/S 结构，但面向购物中心工作人员的相关操作，为了 保证系统运行稳定快捷，则可能会采用三层客户机/服务器结构。其对应的物理构架如图 5-8      所示。

B/S 结构的优点是不需要对客户机进行专门的维护（客户机上没有专门的应用程序），特别 适合于客户机位置不固定或需要依靠互联网进行数据交换的应用系统。缺点是最终用户信息需 要通过 Web 服务器获取，并通过网络传送到客户机上，因此系统的数据传输速度以及系统的稳 定性，都将明显低于三层客户机/服务器结构。 如同其他客户机/服务器结构一样，B/S 结构也是逻辑结构，因此一个单一的服务器计算机 可以既是 Web 服务器，又是应用服务器和数据库服务器。但如果需要使系统具有更高的性能或 更加稳定的运行状态，那么则有必要将 Web 服务、应用处理和数据管理从物理上分离开来，设 置专门的 Web 服务器计算机、应用服务器计算机和数据库服务器计算机。 在许多应用中，B/S 结构和三层客户机/服务器往往被结合起来使用。例如“网上购物系统”， 其面向消费者的购物操作一般采用的是 B/S 结构，但面向购物中心工作人员的相关操作，为了 保证系统运行稳定快捷，则可能会采用三层客户机/服务器结构。其对应的物理构架如图 5-8 所示。



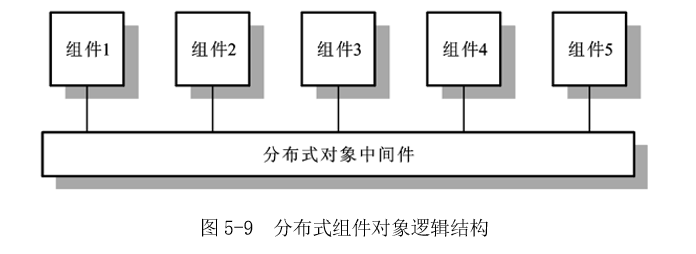
（4）组件对象分布式结构

组件是一个对象包。组件对象分布式结构就是通过组件而使软件系统中的组件对象被分布到网络上的多台计算机上。组件对象具有一些公共接口，能够向外提供服务，不同组件的对象 之间可以通过公共接口相互通信和协同工作。 应该说，传统的客户机/服务器结构是一种不对称的分布式结构，其客户机和服务器具有不同的地位。客户机能够接受来自服务器的服务，但不能接受其他客户机的服务；服务器能够 扮演客户机的角色，由此接受来自其他服务器的服务，但它不能请求来自客户机的服务。 然而，组件对象分布式结构则是一种比较对称的结构，在这种结构中，已经没有了客户机与服务器之间的界限，这些对象既可充当服务器，也可充当客户机，其角色只是决定于它是在 提供服务还是在请求服务。

组件对象分布式结构是基于对象中间件建立的。主要用于对象分布式计算的中间件有：

•  CORBA（通用对象请求代理模型）：由对象管理组织（OMG）定义，提供了一组通用的与 机器无关的分布式对象计算方法。可用在 Unix、Linux 或 Windows 上。

•    DCOM（分布式组件对象模型）：由微软公司定义，主要用在微软公司的操作系统上。  可以把对象中间件看成是能够在其上插拔组件的软件总线。这就像硬件总线允许不同的接 口卡插于其上以支持硬件设备之间的通信一样，对象中间件作为软件总线，也允许在其上添加 或移走组件对象，或进行组件对象之间的相互通信。其逻辑结构如图 5-9 所示。



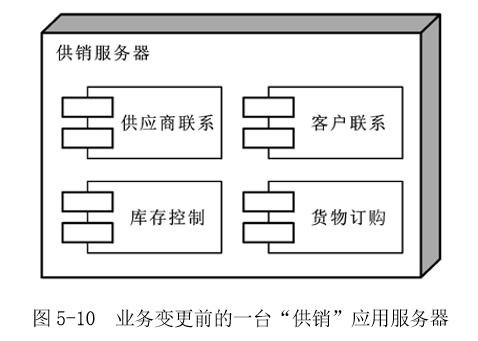
应该说，组件对象分布式结构是一个开放的体系构造，它不仅允许新的组件对象根据需要 增加进来，而且还允许使用不同语言编写的对象。现在，组件对象分布式结构已经成为构造软 件系统的一种基本模式，诸多运行在互联网上的远程服务系统，它们即是按照这种结构构造的。  
 在应用系统中采用组件对象分布式结构时，可依照以下步骤来进行系统结构设计：

（1）按照系统功能服务对系统进行逻辑分块，确定系统的逻辑构造。

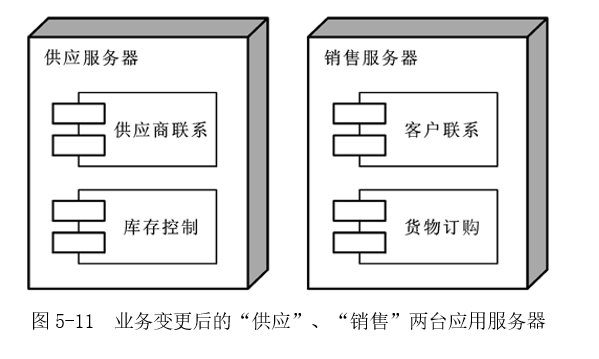
（2）以系统逻辑构造为依据，设计系统的分布式组件对象。

（3）以组件之间的通信与协作为依据，对组件进行物理部署。

组件对象分布式结构具有灵活的构架，系统伸缩性好，能够给系统的功能调整与扩充带来 便利。但是，采用完全分布的组件对象结构，其技术难度太大，内部结构也较复杂。相比之下， 客户机/服务器结构是一种更加容易理解的结构，并与现实的组织机构、业务过程具有一致性。 因此，在许多实际应用中，往往将组件对象分布式结构与客户机/服务器结构结合起来，其中组 件对象分布式结构被作为一种实现技术嵌入到客户机/服务器系统之中。这种构架的特点是：系 统先按照客户机/服务器结构进行逻辑构造；然后无论是客户机还是服务器，都按照组件对象分 布式结构进行内部构造设计，它们都由组件组成，并都需要通过对象中间件实现通信。显然， 这样的系统将更加具有柔性，系统的变更也将会变得更加容易。例如，一个二层客户机/服务器 结构，只需要对其组件对象进行重新部署，就可以将其改造成为三层甚至多层客户机/服务器   结构。 实际上，分布式组件对象技术能够给应用服务器的创建带来更好的可伸缩性，由诸多组件构造的应用服务器可以更加自由地进行物理部署。例如一个供销管理系统，假设其包括供应商 联系、库存控制、客户联系、货物订购等业务内容，并且这些业务都由一个供销部门负责，则 系统构造可按以上业务创建组件，并将这些组件部署在一台应用服务器上，如图 5-10 所示。



但假如业务流程发生了变化，例如，由于业务量扩大，该供销部门需要拆分为供应部、销售部两个部门时，考虑到业务归口与服务器负荷的影响，上面的一台应用服务器也需要改变为两台应用服务器，分别承担供应与销售的事务处理。显然，系统所采用的组件对象分布式结构 能够很方便地适应这种业务需求的变更，需要进行的系统改造只不过是将用于业务处理的组件 重新进行部署或安装，如图 5-11 所示。



三、软件结构设计

软件结构设计是对组成系统的各个子系统的进一步分解与规划。例如，将子系统按照其功 能要素分解成具有一定的功能边界的模块，然后以模块为单位来构造软件。显然，需求分析阶 段已经建立起的有关系统的功能模型、数据模型或状态机模型，可以作为软件结构设计的前提 依据。 具体说来，软件结构设计包括以下几方面的内容：

（1）确定构造子系统的模块元素。

（2）定义每个模块的功能。

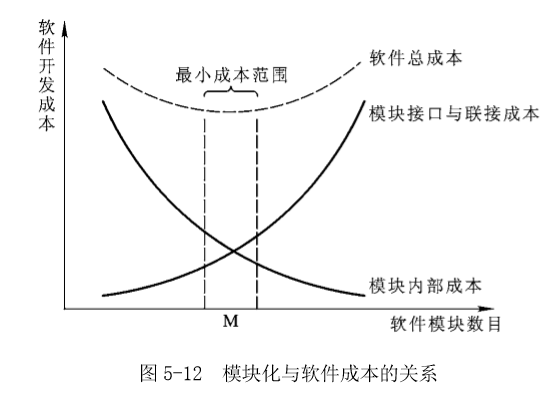
（3）定义模块接口，设计接口的数据结构。

（4）确定模块之间的调用与返回关系。

（5）评估软件结构质量，进行结构优化。

1.模块概念

（1）模块化

模块概念产生于结构化程序设计思想，这时的模块被作为构造程序的基本单元，例如函数、 过程。 在结构化方法中，模块是一个功能单位，因此模块可大可小。它可以被理解为所建软件系 统中的一个子程序系统，也可以是子程序系统内一个涉及多项任务的功能程序块，并可以是功 能程序块内的一个程序单元，例如函数、过程。也就是说，模块实际上体现出了系统所具有的 功能层次结构。 模块可以使软件系统按照其功能组成进行分解，而通过对软件系统进行分解，则可以使一 些大的复杂的软件问题分解成诸多小的简单的软件问题。从软件开发的角度来看，这必然有利 于软件问题的有效解决。对于这个结论，可以进行以下论证： 现假设函数 C （ P ）用于度量问题 P 的复杂程度，函数 E （ P ）用于度量为解决问题 P 所需 要的工作量（用时间计算）。并假设有两个问题 P 1和 P 2。 如果有： C （ P 1）> C （ P 2） 则必有： E （ P 1）> E （ P 2） 这是一个显而易见的经验性结论。它表明：问题越复杂，解决这个问题所需要的工作量就 越大，需要花费时间就越多。 另一个来源于经验的结论是： C （ P 1 + P 2）> C （ P 1）+ C （ P 2） 它表明：如果能够把两个结合在一起的问题隔离开来单独解决，则可以使这原本结合在一 起的问题的复杂程度下降。 显然，从上面两个经验性结论中可以作出以下推论： E （ P 1 + P 2）> E （ P 1）+ E （ P 2） 它表明：如果能够把两个结合在一起的问题隔离开来单独解决，则可以使这原本难于解决 的问题变得容易解决起来。这个结论就是模块化的基本依据，其作用就是能够使大的复杂的问 题被“各个击破”。 上述结论表明：模块化可以使软件问题简化。但是，得出这个结论需要有下面的前提条件： 大模块被分解成诸多小模块之后，小模块基本上处于相互隔离的独立状态，它们之间没有太多 的通信。否则，小模块之间由于存在太多的通信，将会导致小模块接口复杂起来，其结果是， 虽然每个模块内部简化了，但整个软件问题反而复杂起来。 许多人都有一个良好的愿望，那就是依靠模块化使系统不断分解而使整个系统不断简化， 由此使软件开发成本不断下降。然而这却可能做不到，因为随着系统的分解，系统中模块数目 将会增加，模块接口也会增加，软件构造会由此变得复杂起来，模块连接的难度也会由此加大。实际上还有一个因素也在阻碍着这个愿望的实现，这就是软件系统的模块化分解本身也是一件 并不轻松的工作，假如分解系统所需要的工作量，已经超过因为模块简化而减少的工作量，那 就意味着，进一步分解系统已是一件得不偿失的事情了。 因此，软件系统模块化分解，从软件成本角度来看，有一个最小成本模块数范围。如图 5-12 所示，若分解程度低于这个最小成本模块数范围，则可继续分解，以降低软件开发成本；但若 分解程度已经达到这个最小成本模块数范围，则就没有必要再进行分解了，否则将会反而增加 软件开发的成本。  
 

（2）抽象化

抽象是人所具有的一种高级思维活动，是以概括的方式抽取一系列事物的共同特征，由此 把握事物的本质属性。抽象也是人类解决复杂问题时强有力的手段，能够使人从事物复杂的外 部表象中发现事物的内在本质规律，由此找到解决问题的有效途径。 这种抽象的方法也被运用于软件工程之中。实际上，软件工程的每一个阶段，都能够看到 抽象思维方法的作用，从软件的定义到软件的设计，直到软件编码与最终实现。但是，在软件工程的不同阶段，抽象层次与对象却各不相同，并需要运用不同的语言进行描述。例如，在软件分析阶段，抽象对象是系统的外部特征，需要采用的描述语言是软件问题所处环境中的用户 术语；而在设计阶段，抽象对象则是系统的内部构造，需要采用的描述语言则是更加规范的适 合于系统构造的过程化语言。 概要设计中的功能模块往往被看成是一个抽象 化的功能黑盒子，其特征如图 5-13 所示。虽然它已 是一个与软件实现直接相关的实体单元，可以看到 它清晰的外观，但是却看不到它内部更加具体的实 现细节。 抽象的作用是对事物现象的高度概括，但抽象 的最终目的则是走到它的对立面，产生出具体的结 果。因此，一个完整的抽象化过程是既包含抽象又 包含具体的循环演变过程。应该说，正是由抽象到具体的不断演变，才使得人的抽象认识能够不断地产生出有意义的结果，并不断地推动着人类 思想的进步。 这种由抽象到具体的不断演变也一直贯穿于软件工程过程之中，这就是自顶向下、逐步细 化，其中，顶是抽象的，而细化则是这个抽象的顶具体化的结果。这意味着，软件工程的每一 个阶段的推进，都具有从抽象到具体的转化。 例如需求分析，它建立在可行性研究基础之上。可行性研究所获得的是有关整个计算机系 统的外部模型，这是一种对整个系统的高度抽象。但是，随着需求分析的进行，可行性研究中 高度抽象的系统模型被逐步地转变成需求分析模型中每一个功能局部的规格定义。显然，这是 软件从高层定义到低层定义的具体化演变。 又如概要设计，它是在需求分析基础上进行的，需要根据需求分析中的基本要求设计软件 系统的基本构架，需要按照需求分析所提出的功能规格，确定软件系统中模块的构成，定义模 块的接口，确定模块之间通信，设计对模块的控制。显然，这是软件从系统定义到系统设计的 具体化演变。 应该说，概要设计中的结论，对于软件内部构造而言则仍是抽象的。比如概要设计中的功 能模块，尽管它有惟一的名称标识，有明确的功能定义，并设置有数据的输入输出接口。但是， 模块的内部实现细节则处于待定状态，需要等到详细设计完成以后才能得出更加具体的算法结论。

（3）信息隐藏

设计软件结构时一个不可回避的问题是：为了得到一种高质量的模块组合，应该如何分解 软件。对此，不得不了解什么是“信息隐蔽”。 信息隐蔽是指每个模块的内部实现细节对于其他模块来说是隐蔽的。也就是说，模块中所 包含的信息，例如，模块内部的数据、语句或过程等，不允许其他不需要这些信息的模块使用。 显然，信息隐蔽有利于模块相互之间的隔离，可以使每个模块更加具有独立性，并可以使模块 之间的通信受到限制。例如，模块之间只能传递那些对于其功能实现而言是必须的信息。 通过限制模块需要使用的数据的作用范围，例如，定义模块内部局部变量，可以产生出模块内部信息隐蔽的效果。 模块内部信息隐蔽的好处是可以使软件系统更加健壮，更加方便维护。 以模块中的错误为例，假如模块内部信息是隐蔽的，则模块中存在的这个错误将比较难于扩散到其他模块。否则，系统可能因为一个小错误的扩散，而使整个系统崩溃。实际上，信息 隐蔽还使软件错误定位更加方便。由于软件出错位置容易发现，因此，整个软件纠错工作的效 率、质量都会随之提高。 一些模块的功能有时需要根据用户需求的变更而适时地进行一些功能改造。当需要对模块 进行功能改造时，模块内部的信息隐蔽会使对模块改造所带来的影响限制在需要改造的模块之 内，而与其他模块无关。显然，这将使软件系统的局部修改变得更加便利。

2.模块的独立性

模块的独立性是指软件系统中每个模块都只涉及自己特定的子功能，并且模块接口简单， 与软件中其他模块没有过多的联系。

模块独立性是衡量软件中模块质量最重要的指标，是设计与优化软件结构时必须考虑的重要因素。当软件中的每个模块都具有很好的独立性时，软件系统不仅更加容易实现，并且会使今后的维护更加方便。 模块的独立性一般采用耦合和内聚这两个定性的技术指标进行度量。其中，耦合用来反映 模块之间互相连接的紧密程度，模块之间的连接越紧密，联系越多，耦合性就越高。内聚用来反映模块内部各个元素彼此结合的紧密程度，一个模块内部各个元素之间结合越紧密，则它的 内聚性就越高。显然，为了使模块具有较强的独立性，要求模块是高内聚、低耦合。

（1）耦合

耦合是软件结构中各个模块之间相互关联程度的度量。耦合的强弱取决于各个模块之间接 口的复杂程度、接口数据对模块内部计算的影响程度和调用模块的方式。 模块之间的耦合形式主要有：非直接耦合、数据耦合、控制耦合、公共耦合和内容耦合。 其中，非直接耦合和数据耦合是较弱的耦合，控制耦合和公共耦合是中等程度的耦合，内容耦 合则是强耦合。 模块化设计的目标是尽量建立模块间耦合松散的系统。因此，在设计软件结构时一般也就 要求尽量采用非直接耦合和数据耦合，少用或限制使用控制耦合和公共耦合，绝对不能使用内 容耦合。 为了更好地认识模块之间的耦合，下面将对上述各种耦合形式给出必要说明。

a.非直接耦合

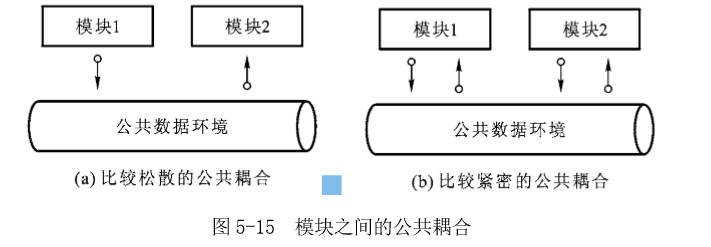
如果两个模块之间没有直接关系，它们之间的联系仅限于受到共同主模块的控制与调用， 则称这种耦合为非直接耦合。 由于非直接耦合的模块之间没有数据通信，因此模块的独立性最强。

b.数据耦合

当一个模块访问另一个模块时，如果彼此之间是通过模块接口处的参数实现通信，并且参 数所传递的数据仅用于计算，而不会影响传入参数模块的内部程序执行路径，则称这种耦合为 数据耦合。 数据耦合是一种松散的耦合。依靠这种类 型的耦合，模块之间既能实现通信，又有比较 强的独立性。因此，软件结构设计中提倡使用 这类耦合

c.控制耦合

当一个模块访问另一个模块时，如果彼此 之间是通过模块接口处的参数实现通信，并且 参数传递了开关、标志、名字等控制信息，由 此影响了传入参数模块的内部程序执行路径， 则称这种耦合为控制耦合如图 5-14 所示。 由于接口参数影响着内部程序执行路径， 因此控制耦合比数据耦合要强一些，会使模块 的独立性有所下降。当需要通过一个单一的接口传递模块内多项功能的选择信息时，往往需要用到控制耦合。显然，在控制耦合形式下，对 所控制模块的修改，需要受到控制参数的限制。

d.公共耦合 公共耦合是一种通过访问公共数据环境而实现通信的模块耦合形式，例如，独立于模块而 存在的数据文件、数据表集、公共变量等，都可以看作为公共数据环境。 公共耦合中的公共数据环境是提供给任何模块的，当模块之间的耦合是公共耦合时，那些 原本可以依靠接口提供的对数据的限制也就没有了。因此，相比依靠接口的耦合形式，公共耦 合必然会使模块的独立性下降。也正因为如此，实际应用中，只有在模块之间需要共享数据， 并且通过接口参数传递不方便时，才使用公共耦合。 需要注意的是：模块之间公共耦合的复杂程度，将会随着耦合模块个数的增加而显著增加。 为了降低公共耦合带来的复杂性和提高模块的独立性，实际应用中还会针对公共耦合专门设置 一些限制，例如不使用公共耦合数据传递控制信息。 图 5-15 中的模块采用了访问公共数据环境的公共耦合形式。其中图（a）是一个模块只往 公共数据环境里送进数据，另一个模块只从公共数据环境中取出数据，这是一种比较松散的公 共耦合；而图（b）则是两个模块都可以向公共数据环境里送进数据，又都可以从公共数据环境 中取出数据，这就是一种比较紧密的公共耦合。  
  

另外，公共耦合还会带来以下一些方面的影响： •  由于所有公共耦合模块都会与某一个公共数据环境有关，因此修改公共数据环境中某个 数据的结构，将会影响到所有被耦合的模块。

•  由于没有办法控制各个模块对公共数据的存取，因此公共耦合会影响软件模块的可靠性 和适应性。

•  由于公共数据环境需要被许多模块使用，因此不得不使用具有公共意义的数据名称。显 然，这会使得程序的可读性有所下降。

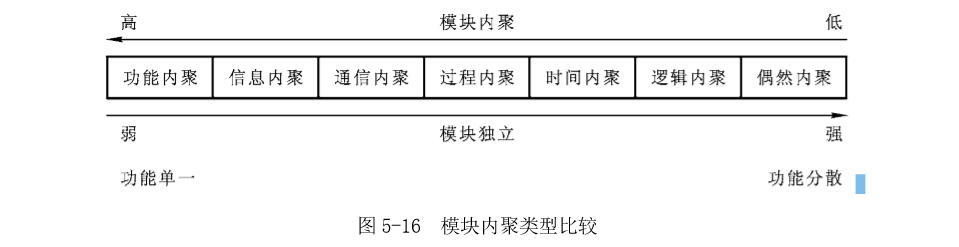
e.内容耦合 如果发生下列情形，两个模块之间就发生了内容耦合。

•  一个模块直接访问另一个模块的内部数据。

•  一个模块不通过正常入口转到另一模块内部。 •  两个模块有一部分程序代码重叠。

•  一个模块有多个入口。 内容耦合是一种非常强的耦合形式，严重影响了模块独立性。当模块之间存在内容耦合时，模块的任何改动都将变得非常困难，一旦程序有错则很难修正。因此，设计软件结构时，也就 要求绝对不要出现内容耦合。所幸的是，大多数高级程序设计语言已经设计成不允许出现内容 耦合，它一般只会出现在汇编语言程序中。

（2）内聚

内聚是对模块内部各个元素彼此结合的紧密程度的度量。模块内部各个元素之间的联系越紧密，则它的内聚程度就越高。模块的设计目标是尽量使模块的内聚程度高，以达到模块独立、 功能集中的目的。 模块内聚的主要类型有：功能内聚、信息内聚、通信内聚、过程内聚、时间内聚、逻辑内 聚和偶然内聚，如图 5-16 所示。其中，功能内聚和信息内聚属于高内聚，通信内聚和过程内聚 属于中等程度的内聚，时间内聚、逻辑内聚和偶然内聚则属于低内聚；而且，模块内聚程度越 高，其功能越集中、独立性越强。  
 

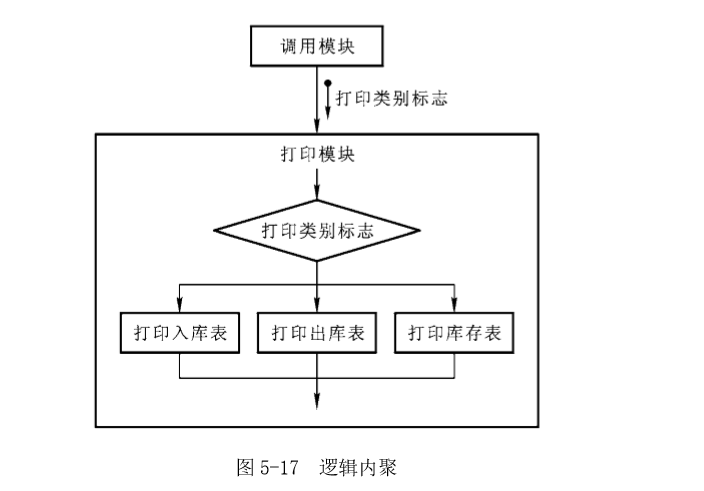
内聚所体现的是模块的内部功能构造，耦合所体现的是模块之间的联系。一般情况下，内 聚和耦合是相互关联的，模块的内聚程度越高，则模块间的耦合程度就会越低，但这也不是绝 对的。值得指出的是，比起模块之间的耦合来，模块的内聚更显重要。因此，实际设计中应当 把更多的注意力放在如何提高模块的内聚程度上。 模块内聚的提高依赖于对模块功能的正确认识，应该通过定义使每一个模块都具有明确的 功能。为了更好地认识模块内聚，下面将对上述几种内聚类型分别加以说明。

a.偶然内聚

当模块内各部分之间没有联系，或即使有联系，这种联系也很松散时，将会出现偶然内聚。偶然内聚往往产生于对程序的错误认识或没有进行软件结构设计就直接编程。例如，一些 编程人员可能会将一些没有实质联系，但在程序中重复多次出现的语句抽出来，组成一个新的 模块，这样的模块就是偶然内聚模块。 偶然内聚模块由于是随意拼凑而成，模块内聚程度最低、功能模糊，很难进行维护。

b.逻辑内聚

逻辑内聚是把几种相关的功能组合在一起形成为一个模块。在调用逻辑内聚模块时，可以 由传送给模块的判定参数来确定该模块应执行哪一种功能。例如图 5-17 中的打印模块。 逻辑内聚模块比偶然内聚模块的内聚程度要高，因为它表明了各部分之间在功能上的相关 关系。但是它每次执行的不是一种功能，而是若干功能中的一种，因此它不易修改。另外，在 调用逻辑内聚模块时，需要进行控制参数的传递，由此增加了模块间的耦合。



c.时间内聚

时间内聚模块一般是多功能模块，其特点是模块中的各项功能的执行与时间有关，通常要 求所有功能必须在同一时间段内执行。例如初始化模块，其功能可能包括给变量赋初值、连接 数据源、打开数据表、打开文件等，这些操作要求在程序开始执行的最初一段时间内全部完成。  
 时间内聚模块比逻辑内聚模块的内聚程度又稍高一些，其内部逻辑比较简单，一般不需要 进行判定转移。

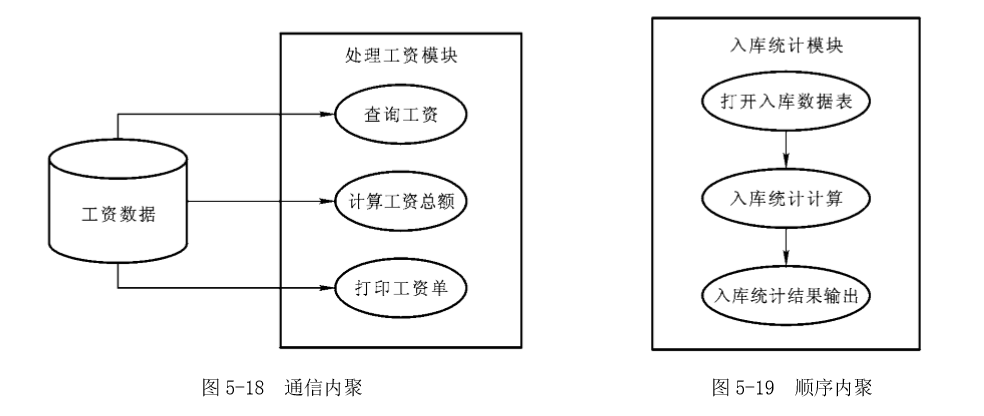
d.过程内聚

如果一个模块内的处理是相关的，而且必须以特定次序执行，则称之为过程内聚模块。在 使用流程图设计程序的时侯，常常通过流程图来确定模块划分，由此得到的就往往是过程内聚 模块。例如，可以根据流程图中的循环部分、判定部分和计算部分将程序分成三个模块，这三 个模块就是过程内聚模块。 过程内聚模块的内聚程度比时间内聚模块的内聚程度更强一些，但过程内聚模块仅包括完 整功能的一部分，因此模块之间的耦合程度比较高。

e.通信内聚

如果一个模块内各功能部分都使用了相同的输入数据或产生了相同的输出数据，则称之为 通信内聚模块。例如图 5-18 中的处理工资模块。 通信内聚模块由一些独立的功能组成，因此其内聚程度比过程内聚程度要高。

f.顺序内聚

如果一个模块内的诸多功能元素都和某一个功能元素密切相关，而且这些功能元素必须顺 序安排（前一项功能的数据输出作为后一项功能的数据输入），则称之为顺序内聚模块。当根据 数据流图划分模块时，由此得到的通常就是顺序内聚模块。例如图 5-19 中的入库统计模块。 顺序内聚模块也包含着多项功能，但它有一个核心功能，其他功能则围绕着这个核心而安 排。因此，顺序内聚程度比通信内聚程度更高。  
 

g.功能内聚

如果一个模块中各个部分都是完成某一具体功能必不可少的组成部分，各个部分协同工 作、紧密联系、不可分割，则称该模块为功能内聚模块。 功能内聚模块的特征是功能单一、接口简单，因此其容易实现、便于维护。与其他内聚类 型相比，功能内聚具有最高的内聚程度，软件结构设计时应以其作为追求目标。

 3.结构化设计建模

软件结构设计涉及模块功能、模块接口与模块调用关系等问题，为了使这些问题能够集中 清晰地表达出来，软件结构设计需要借助于一定的图形工具来建立设计模型，例如软件结构图、 HIPO 图。

（1）软件结构图

软件结构图由 Yourdon 于 20 世纪 70 年代提出，并被广泛应用于软件结构设计中，能够有 效说明软件中模块之间的调用与通信。 软件结构图使用矩形框表示模块（框内注明模块的名字或主要功能），使用带箭头的直线 段连接上下级模块，以表示上级模块对下级模块的调用。此外，软件结构图还可以在调用箭头 旁使用带注释的箭头，以表示上级模块在调用下级模块时参数的传递与结果的返回，其基本图 形符号如表 5-1 所列。  
表

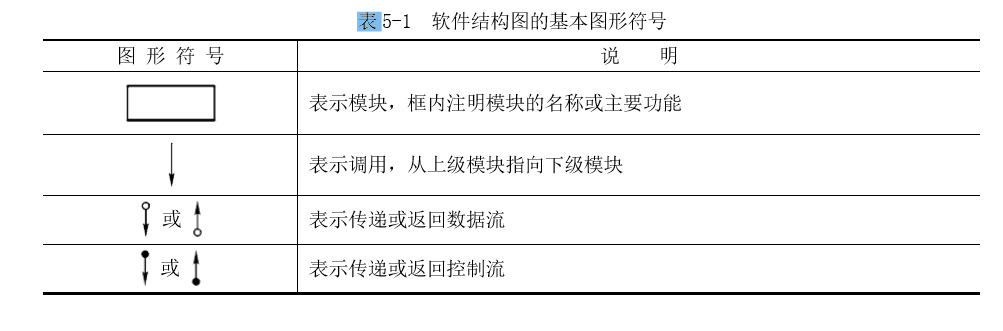
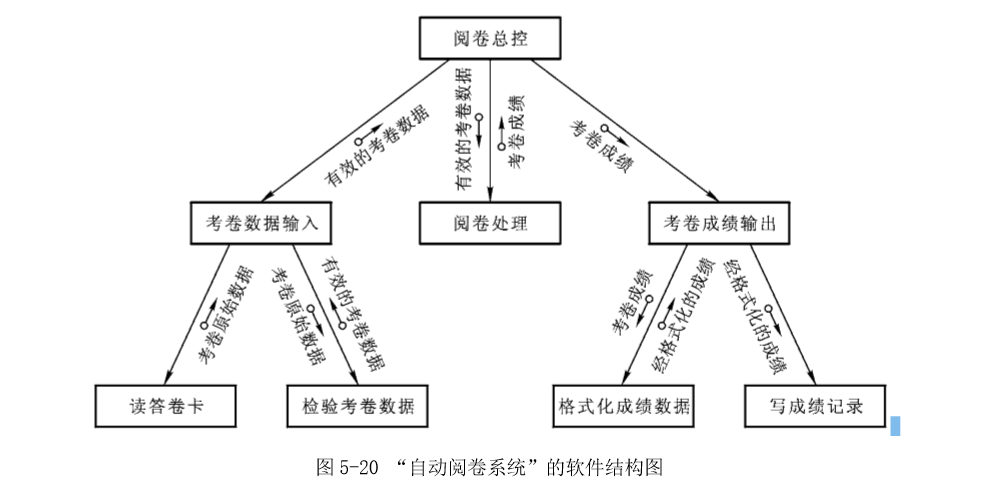


图 5-20 是一个自动阅卷系统的软件结构图。阅卷总控模块为顶层模块，其调用考卷数据 输入、阅卷处理和考卷成绩输出这三个模块，以进行考卷输入、成绩计算和成绩输出的控制。 在更下一级模块调用中，考卷数据输入模块通过调用读答卷卡模块将考卷原始数据输入，然后 调用检验考卷数据模块产生出适合评分计算的有效数据；考卷成绩输出模块则通过调用格式化 成绩数据模块对计算出的成绩数据进行输出前的格式化处理，然后调用写成绩记录模块实现成 绩的存档操作。  
 

（2）HIPO 图

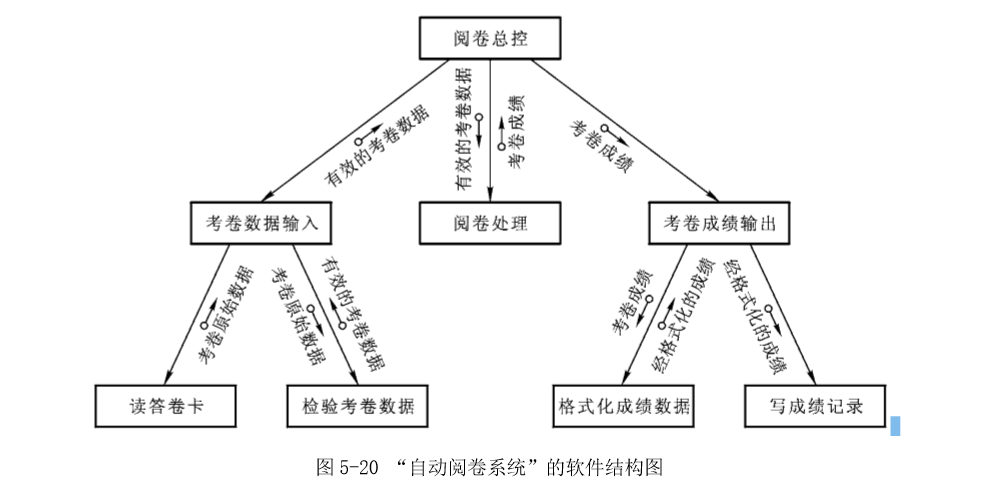
HIPO 图是美国 IBM 公司推出的“H 图”与“IPO 图”的组合。

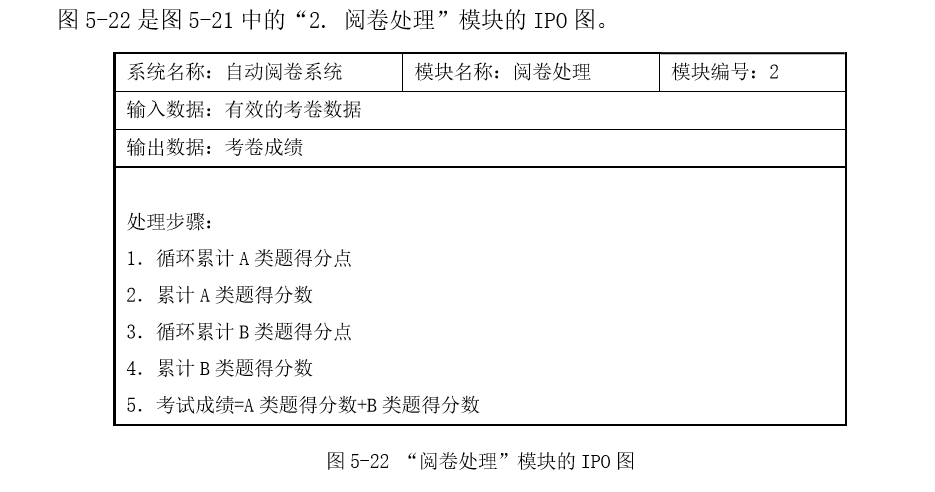
　　a.H 图

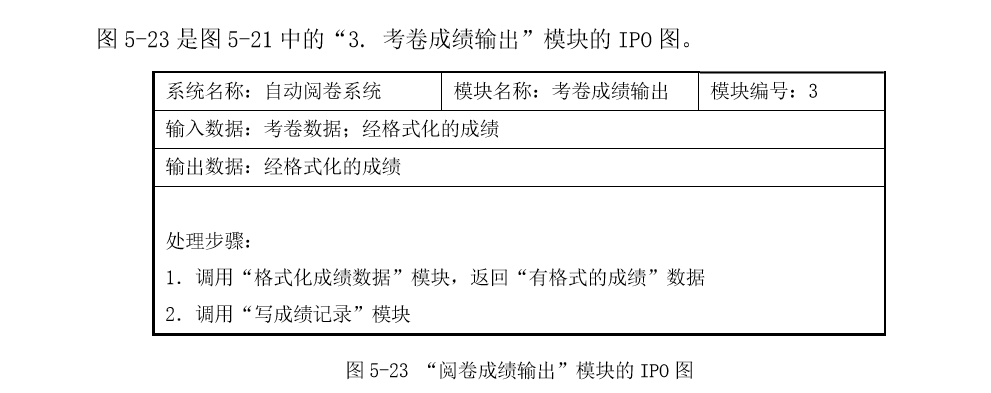
H 图是软件层次图的简称，用于描述软件结构上的分层调用关系，作用类似于软件结构图， 但不涉及调用时的数据流、控制流等附加信息。 H 图的优点是清晰度高，能够用于正式文档中对软件结构的描述。图 5-21 是自动阅卷系统 的 H 图，可以看出，它比上面的软件结构图要清晰得多。 为了能使 H 图中的模块具有可追踪性，由此可以与它所对应的 IPO 图联系起来。图中模块 除了最顶层的之外，其他的模块都需要按照一定的规则编号，以方便检索。

　　b.IPO图

IPO 图是“输入—处理—输出图”的简称，其中的“I”是指输入，“O”是指输出，“P” 是指处理。可以使用 IPO 图对模块进行外部特征描述，涉及输入、输出接口和基本加工步骤等。 在 HIPO 图中，需要针对 H 图中的每个模块给出 IPO 图描述。







从上面的 IPO 图举例可以看到，IPO 图提供了有关模块的更加完整的定义和说明。显然， 这将有利于由概要设计到详细设计的过渡。

4.软件结构优化

软件结构设计往往需要经历多次反复，其作用是可以不断地改进软件结构，提高软件质量。 为了改进软件结构，软件设计人员进行了长期的实践与总结，由此得到了一些优化设计的原则。 本节将介绍这些原则。

（1）使模块功能完整

一个完整的功能模块，不仅能够完成指定的功能，而且还应当能够将完成任务的状态通知 使用者。 具体说来，一个完整的功能模块应当包含以下几个部分的内容：

　　a.执行规定功能的部分。

　　b.出错处理的部分。当模块不能完成规定的功能时，必须回送出错标志，并向它的调用 者报告出现这种例外情况的原因。

　　c.如果需要返回一系列数据给它的调用者，在完成数据加工时，应当给它的调用者返回 一个该模块执行是否正确结束的“标志”。

（2）使模块大小适中

许多情况下，可以用模块中所含语句的数量的多少来衡量模块的大小。有人认为限制模块 的大小也是减少模块复杂性的有效手段之一，因而要求把模块的大小限制在一定的范围之内， 例如将模块内的语句限制在 50～100 行左右。当然，这只能作为参考。 一般说来，模块过大的原因往往是模块的功能太多，因此有必要对大模块做进一步的分解。 大多数情况下，可以从大模块中分解出一些下级模块或同层模块。 软件结构设计时，也有可能出现模块过小的问题。如果模块太小，则要注意这个模块的功 能是否完整，是否将一个完整的功能分解到多个模块中去了。许多情况下，可以考虑将较小的 模块与调用它的上级模块合并。如果模块虽小但功能内聚性好，或者它为多个模块所共享，或 者调用它的上级模块很复杂，则不要贸然将小模块与其他模块合并，而是需要做更加细致的分析。

（3）使模块功能可预测

一个功能可预测的模块可以被看成是一个“黑箱”，无论内部处理细节如何，只要输入数 据是相同的，就总能产生相同的输出结果。 例如，在模块中使用了静态变量。因为静态变量会在模块内部产生较难预见的存储，而如 果这个静态变量又被用为多项功能的选择标记，则可能会使得模块的功能难以被调用者预知。 由于这个原因，设计者要特别慎重地使用静态变量。 模块功能可预测还意味着，模块的功能必须明确清晰地定义，应该使模块具有很好的内   聚性。

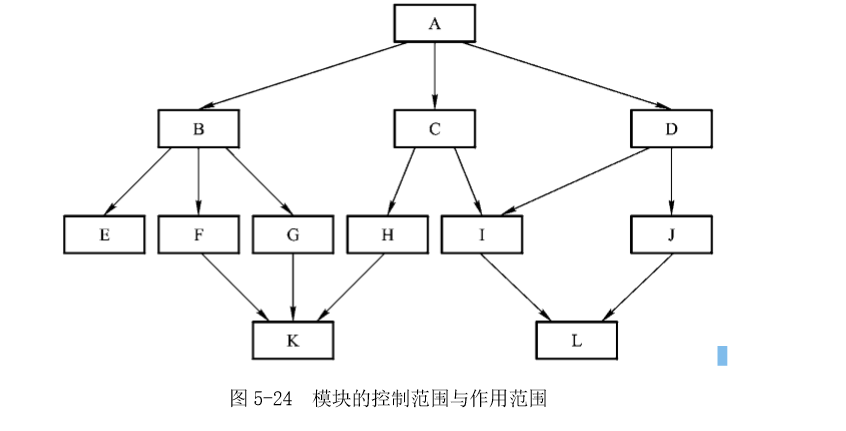
（4）尽量降低模块接口的复杂程度

模块接口是模块与外界进行通信的通道，较复杂的接口往往会带来较高的耦合。因此，应 努力降低模块接口的复杂程度。对此，可以从以下两个方面作出考虑。

a.接口参数尽量采用简单数据类型，以使接口容易理解。例如，尽量使用基本字符、整 数类型，而不是数组、指针、结合体类型。

b.限制接口参数的个数。接口参数个数太多，往往是由于模块功能混杂，不得不依靠参 数进行调控。因此，过多的参数往往意味着模块还有进一步分解的必要。

（5）使模块作用范围限制在其控制范围之内

模块的控制范围包括它本身及其所有从属于它的直接或间接下级模块。如图 5-24 所示， 模块 B 的控制范围是模块 E、F、G、K，模块 C 的控制范围是模块 H、I、L、K，模块 D 的控制范 围是模块 I、J、L。 模块的作用范围则是指模块内一个判定的影响范围，凡是受这个判定影响的所有模块都属 于这个判定的作用范围。其中，如果一个判定的作用范围包含在这个判定所在模块的控制范围 之内，则这种结构是简单的；否则，其结构就是不简单的，需要进行结构调整。 例如，图 5-24 中的模块 F。如果它做出一个判定之后，接着需要模块 G 工作，由于模块 G 不在模块 F 控制范围之内，因此模块 F 必须返回一个信号给模块 B，再由 B 把信号传送给模块 G。 显然这不是一个好的设计，它增加了模块之间数据的传送量，并使模块之间出现了控制耦合， 因此需要对其结构进行调整。  
 

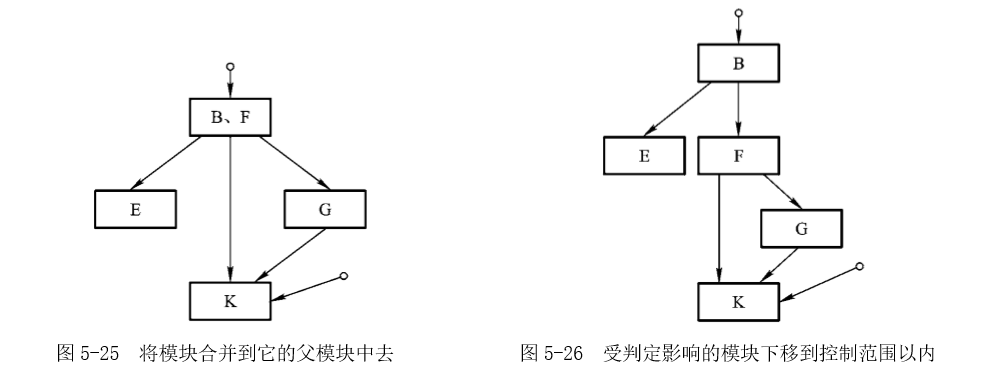
在设计过程中，如果发现模块的作用范围不在控制范围之内时，可以采用如下办法进行结 构调整，把模块的作用范围移到其控制范围之内。

（1）将判定所在的模块合并到它的父模块中去，或上移到层次较高的位置上去，由此可使 判定处于一个较高的位置，以达到有效的控制。例如，将模块 F 与模块 B 合并，如图 5-25 所示， 由此可使模块 G 受到控制。

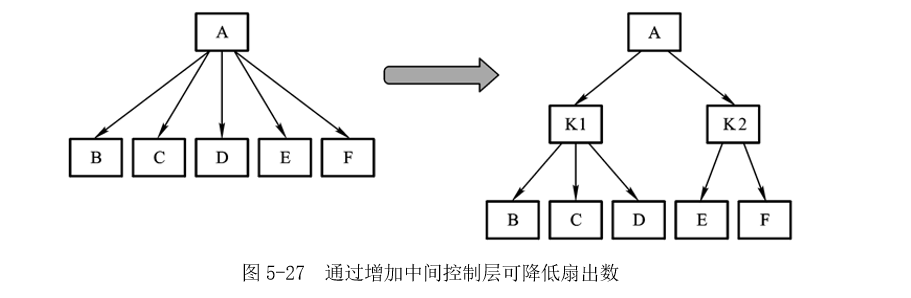
（2）将受判定影响的模块下移到控制范围以内。例如，将模块 G 下移到模块 F 之下，如图 5-26 所示，由此可使模块 G 受到模块 F 控制。 需要注意的是：在改进模块的结构时，应当根据具体情况通盘考虑，既要尽量小范围地调 整结构，使改进后的软件结构能够最好地体现问题的原来结构，又要考虑改进后的结构在实现 上的可行性。

（6）深度、宽度、扇出和扇入应当适当

软件的深度是指软件结构的层数。例如，图 5-24 中的软件结构，其深度为 4 层。



软件的宽度是指软件结构同一个层次上模块的总个数的最大值。例如，图 5-24 中的软件 结构，其第二层宽度是 3，第三层宽度是 6，第四层宽度是 2。其整个软件的宽度是 6。 软件结构上的深度、宽度，在一定程度上反映出了软件系统的规模和复杂程度。也就是说， 软件规模越大越复杂，其深度、宽度也会越大。一般说来，深度、宽度会受模块的扇出、扇入 影响，并应该有一个合适的大小。深度、宽度太小，往往表示模块分解得还不够细，需要进一 步分解。而深度、宽度太大，则可能表示模块分解得太细小了，或功能冗余模块太多了，以致 软件结构变得复杂起来。一般说来，软件结构越复杂，模块之间的耦合就会越大。因此，假如 软件结构过于复杂，就有必要将一些模块进行适当的合并，将那些功能相同或相似的模块合并 起来作为公共模块使用。 模块的扇出是指模块直接调用的下级模块的个数。比较适当的扇出数为 2～5，一般不要超 过 9。如果一个模块的扇出过大，就表明该模块具有过分复杂的控制功能，需要协调和控制过 多的下属模块。对此，应当适当增加中间层次的控制模块，将比较集中的控制分解开来，如图 5-27 所示。但扇出过小也不好，这样将使得软件结构的深度大大增加，会带来过多的调用和返 回的时间开销，由此降低软件的工作效率。



模块的扇入是指模块受到了多少个直接上级模块的调用。一个模块的扇入越大，则共享该 模块的上级模块的数目就越多。如果一个模块的扇入数太大，而它又不是公用模块，则说明该 模块可能具有多项功能。在这种情况下，应当对它做进一步的分析，并将其功能做进一步的分解。

一般说来，各个不同层次的模块具有以下特点：顶层模块起全局控制作用；中间层次的模 块起局部控制作用，并适当承担一些简单的操作提示功能；底层模块担任具体加工任务。因此， 一个设计得比较好的软件结构通常应具有这样的特征：顶层模块高扇出，中层模块低扇出，底 层模块高扇入。

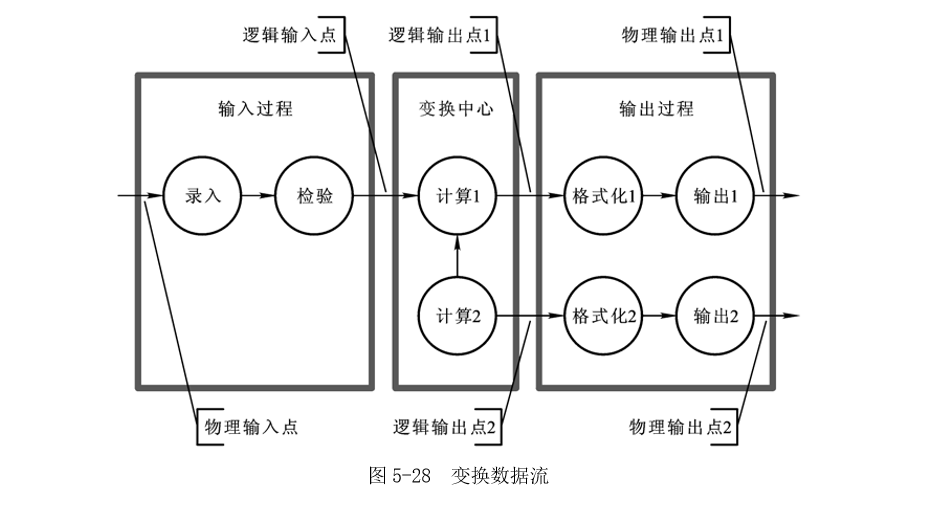
四、面向数据流的结构设计

作为构造软件的基本框架，软件结构应该与需要分析时建立的分析模型保持一致。一种非 常有效的设计思路是，基于需求分析中的数据流模型进行软件结构映射，由此产生出软件系统 的基本设计模型。 为了方便从数据流模型中映射出软件结构来，需要对数据流进行合理的分类。例如，将数 据流分为变换流或事务流，然后按照它们各自不同的特点分别采取不同的映射方法。

1.变换流分析与设计

（1）变换流

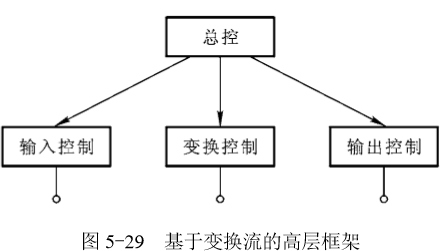
变换数据流所体现出的是数据从输入到加工再到输出的一般步骤。变换数据流对数据的加 工流程如图 5-28 所示，这就是数据首先需要经过输入过程，将外部数据形态转变为适合进行加 工处理的内部形态；然后经过变换中心，将已转成内部形态的输入数据加工成一种新的数据形 态；接着再经过输出过程，将经过加工产生的新的数据结果转换成适合向外导出的数据形态。



（2）高层框架

由于变换数据流将整个过程分割成了输入、变换和输出三个部分，因此，对软件结构的映 射也就可以是在总控模块之下，将软件分为输入、变换、输出三个部分。为了减轻总控模块的 控制负担，可以针对这三个部分，分别设置控制模块，由此获得对各个区段的有效控制。

图 5-29 是基于变换流的高层框架图。



（3）下层模块

当软件结构的高层框架被确定下来以后，接着需要考虑的是那些涉及具体操作的下级模 块。为了确定下级模块，并能够按照一定规则 将这些下级模块挂接到上面的框架上去，有必要对变换数据流进行分段研究。

（1）输入部分

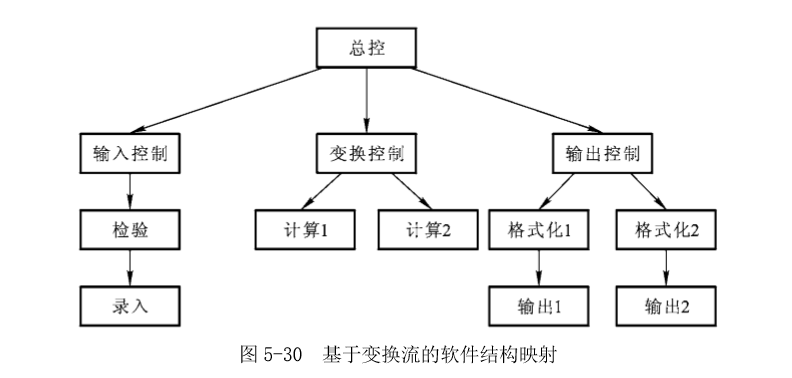
一般情况下，可以把来源于外部端口的数 据流称为物理输入点，而把由输入过程进入到变 换中心的数据流称为逻辑输入点。图 5-28 中的 输入部分从“物理输入点”到“逻辑输入点”， 其模块挂接规则是从“逻辑输入点”往“物理输入点”进行搜索，所遇到的每一个处理框可以作 为一个功能模块依次挂接到“输入控制”模块之下。

（2）变换部分

图 5-28 中的变换中心涉及“计算 1”与“计算 2”两个处理框。可以将这两个处理框对应 为功能模块直接挂接到“变换控制”模块之下。

（3）输出部分

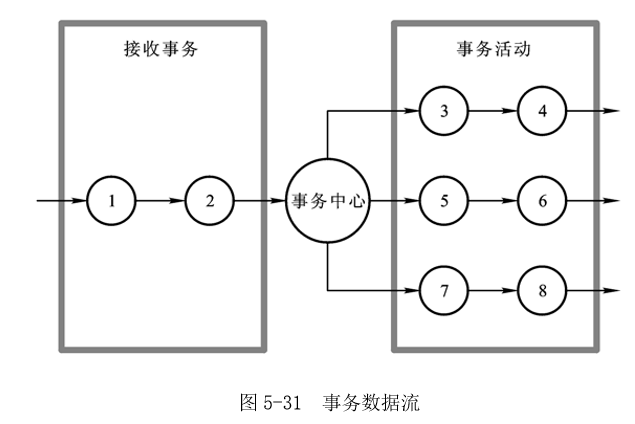
一般情况下，可以把由变换中心流向输出过程的数据流称为逻辑输出点，而把由输出过程 流向外部端口的数据流称为物理输出点。图 5-28 中的输出过程从“逻辑输出点”到“物理输出 点”，其模块挂接规则是从“逻辑输出点”往“物理输出点”搜索，所遇到的每一个处理框可以 当做一个模块依次分层挂接。 依照上述方法，可以获得对图 5-28 中的变换数据流的软件结构映射，产生的软件结构如 图 5-30 所示。



2.事务流分析与设计

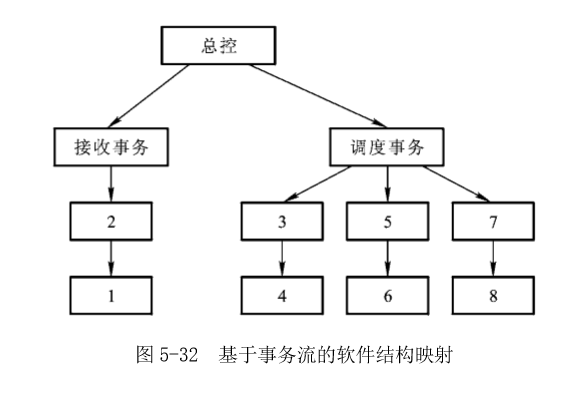
（1）事务流

当输入的数据流可以引发多个不同的事务活动流程，并且数据流图中有一个明显的事务调 度中心时，这种数据流被称为事务数据流。其特征如图 5-31 所示。需要注意事务流中的事务调 度中心与变换流中的变换中心的区别，事务调度中心并不对输入数据进行加工，而只是根据不 同的输入数据作出不同的事务流程选择。

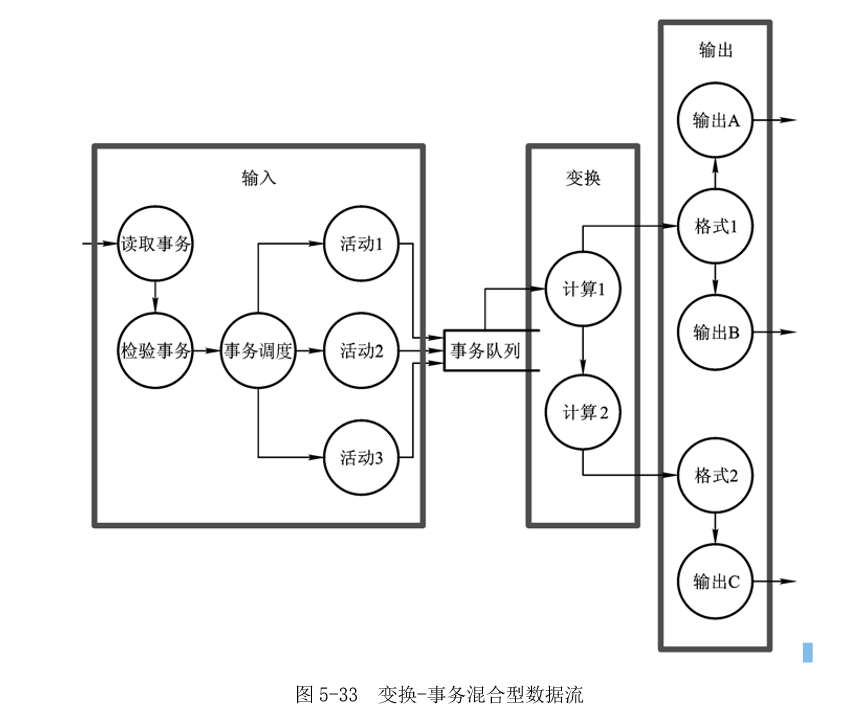


（2）软件结构

事务数据流以事务调度中心为核心，在此之前为接收事务，在此之后为事务分流处理。因 此，基于事务流的软件结构映射也就可以是在总控模块之下将软件分为接收事务与事务活动两 个部分。为了减轻总控模块的控制负担，可以针对这两个部分分别设置控制模块，例如“接收 事务控制”模块、“调度事务控制”模块，以获得对各个区段的有效控制。 在事务流软件框架确定以后，接着需要考虑其下层模块的挂接。对此，以“事务调度中心” 为起点，分别搜索事务输入流与事务活动流，将所遇到的每一个处理框当做一个功能模块依次 挂接到“接收事务”模块或 “调度事务”模块之下。 依照上述方法，可以获得对图 5-31 中的事务数据流的软件结构映射，产生的软件结构如 图 5-32 所示。

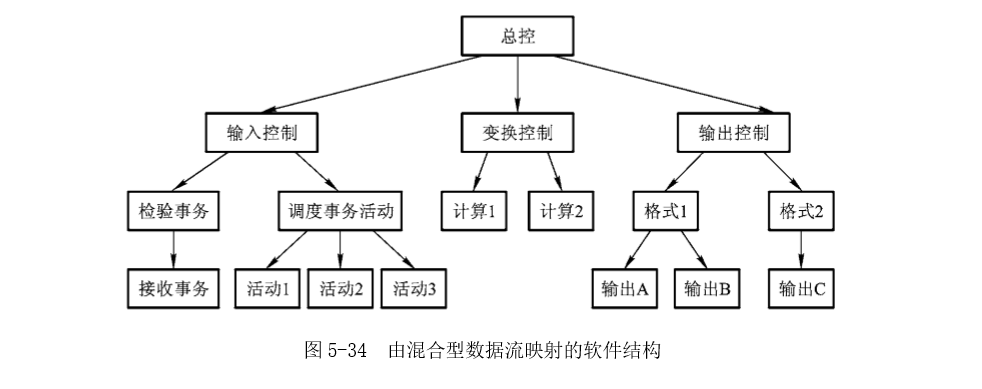


3. 混合流分析与设计

前面分别讨论了变换分析与事务分析。其中，变换分析是软件结构设计的主要方法，大部 分软件系统都可以按照变换分析方法进行设计。但是，在很多情况下仅使用变换分析是不够的， 还需要采用其他方法，事务分析就是一种非常有效的方法。例如商业数据处理系统，其主要组 成部分就往往使用事务处理方法进行设计。 软件系统也可以是变换流与事务流的混合，如图 5-33 所示为典型的变换流与事务流的混 合。对于这样的系统，通常采用变换分析为主、事务分析为辅的方式进行软件结构设计。其一 般设计思路如下：  
 

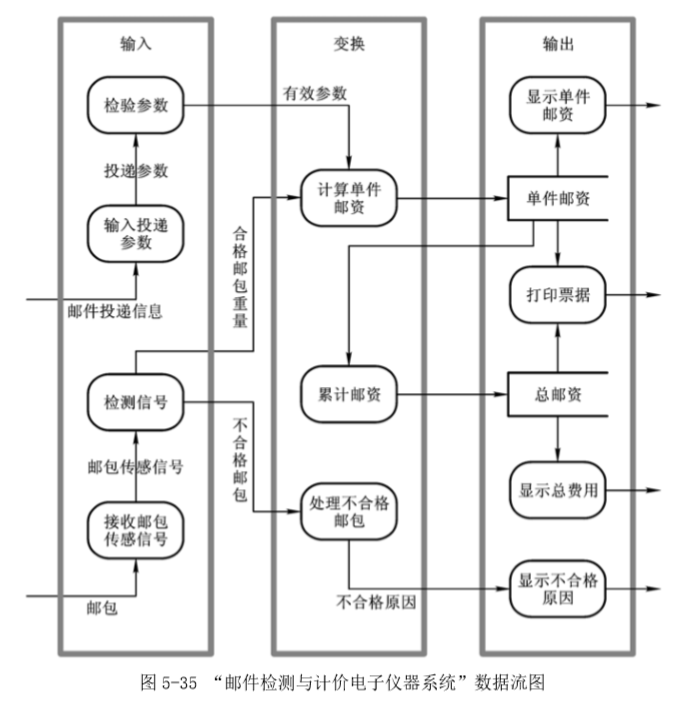
（1）首先利用变换分析方法把软件系统分为输入、变换和输出三个部分，由此设计出软件 系统的上层构架，例如顶层和第一层模块。

（2）然后根据数据流图各部分的结构特点，适当地选择变换分析或事务分析，由此设计出 软件系统的下层构架。 如图 5-33 中的混合数据流，即可以根据上述设计思路进行软件结构映射，由此可以产生 出如图 5-34 所示的软件结构初始方案。



4.设计举例

问题：“邮件检测与计价电子仪器系统”软件结构设计。 该问题数据流如图 5-35 所示。这是一个典型的变换流问题，图中数据流已被划分为输入、 变换和输出三个部分，并具有以下特征：



（1）输入部分

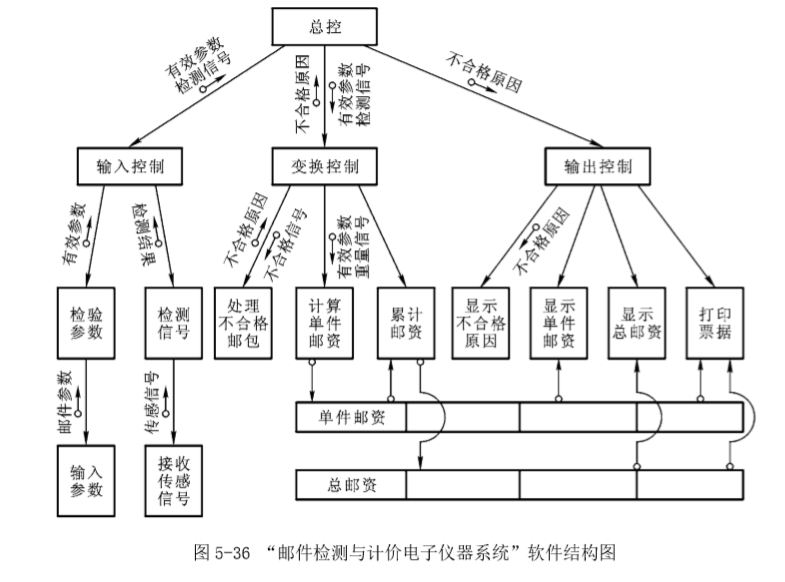
有两个输入支路。支路一：用于输入与检验邮件投递参数，包括邮件种类、投递地区、邮 资计价标准等参数；支路二：用于接收邮包传感信号并检测邮包，包括邮包重量、邮包封装状 态、邮包违禁状态等信号。

（2）变换部分

包括计算单件邮资、累计邮资、处理不合格邮包等变换处理，分别用于合格邮包的邮资计 价和不合格邮包的处理提示。

（3）输出部分

有四个输出支路，包括：显示单件邮资、显示邮资总费用、打印票据、显示不合格邮包处 理提示。 基于上述对数据流的分析，可以方便地根据变换流的结构映射规则产生出它所对应的软件 基本结构。 该问题软件结构如图 5-36 所示。



 5. 数据库结构设计

数据库技术产生于 20 世纪 70 年代初期，从这个时期起，数据库技术经历了层次型、网状 型、关系型三种模型。在数据库应用早期，由于层次型、网状型具有的性能优势，它们占据了 初期阶段的主流位置。但从 20 世纪 80 年代起，随着计算机性能的迅速提高，关系型数据库所具有的数学计算方面的优势受到了重视，它逐渐成为主流数据库，并得到了层次型、网状型不 曾有过的更加广泛的应用。例如 Oracle、SQL Server 这些耳熟能详的通用大型数据库都是关系 型数据库。 数据库就是与特定主题或目标相联系的信息的集合，例如人事数据库、工资数据库等。数 据库的作用是能够为软件系统提供后台数据存储与运算。许多应用系统需要依赖数据库提供数 据服务，尤其是一些信息管理系统，则更是以数据库为中心进行部署。 与一般数据文件比较，数据库具有以下方面的优越性：

（1）数据库具有特殊的数据存储结构，例如，关系型数据库中的表结构，更加适宜对数据 进行有效的组织、存储和检索。

（2）数据库具有更加完善的数据完整性约束机制，例如，可以通过设置和字段相关的主键、 外键，从而建立起数据表集之间的关联；可以设置对数据字段、记录的检验规则，以限制数据 存储范围。

（3）数据库能够实现数据存储结构与数据表现形式的有效隔离，例如，可以通过数据视图 而获得对数据表集更加有效的应用组合。 本节将简要介绍数据库的结构设计，并将从逻辑结构设计和物理结构设计这两个方面分别 给予说明。

1.逻辑结构设计

需求分析中已建立了有关数据库的数据关系模型。但是，数据关系模型是基于对用户应用 域的分析而构造的，是一个有关现实数据环境的数据库概念模型。显然，这种接近于现实世界 的概念模型与计算机世界之间的距离太大了，不能够直接向数据库实现过渡。因此，为了方便 数据库的创建，需要将数据库概念模型进行转换，建立一种更加接近计算机世界的数据库模型。  
 概要设计中需要建立的有关数据库的逻辑结构，就是一种与计算机世界更加接近的数据模 型，它提供了有关数据库内部构造的更加接近于实际存储的逻辑描述，因此能够为在某种特定 的数据库管理系统上进行数据库物理创建提供便利。

（1）设计数据库

在关系型数据库中，数据是以数据表为单位实现存储的。因此，数据库逻辑结构设计首先 需要确定的就是数据库中的诸多数据表。

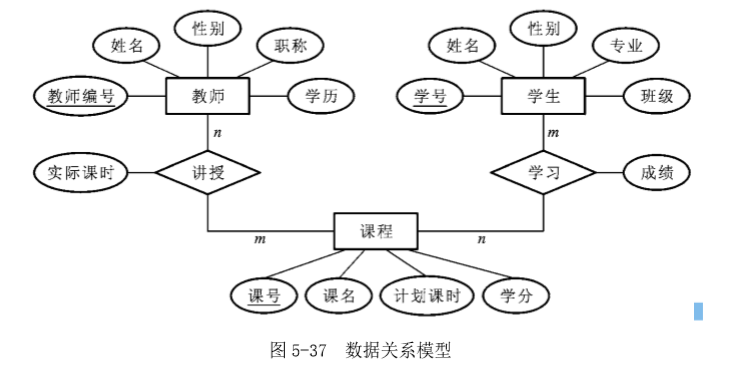
可以按照以下规则从数据关系模型中映射出数据库中的数据表来。

a.数据关系模型中的每一个实体应该映射为数据库逻辑结构中的一个数据表。另外，实 体的属性对应于数据表的字段，实体的码对应于数据表的主键。

b.数据关系模型中的每一个 n : m 关系也应该映射为数据库逻辑结构中的一个数据表。 另外，与该关系相连的各实体的码以及关系本身的属性，应该映射为数据表的字段；而与该关 系相连的各实体的码，则需要组合起来作为关系数据表的主键。

c.数据关系模型中的每一个 1 : n 关系可以映射为一个独立的数据表（映射规则类似 n  
 : m 关系）。但在更多情况下，这个 1 : n 关系则是与它的 n 端对应的实体组合起来映射为一个数 据表。当 1 : n 关系是与 n 端对应实体合并组成数据表时，组合数据表的字段中需要含有 1 端 实体的码属性。

d.数据关系模型中的每一个 1 : 1 关系可以映射为一个独立的数据表，也可以与跟它相 连的任意一端或两端的实体合并组成数据表。实际上，两个依靠 1 : 1 关系联系的数据表可以 设置相同的主键，为了减少数据库中的数据表的个数，往往将它们合并为一个数据表。合并方 法是将其中一个数据表的全部字段加入到另一个数据表中，然后去掉其中意义相同的字段（包 括意义相同但名称不同的字段）。 图 5-37 是一个用于描述教师、课程、学生三者之间关系的数据关系模型图。可以按照上 述规则对这个数据关系模型进行映射，由此可以产生出以下数据表结构： 教师（教师编号，姓名，性别，职称，学历） 课程（课号，课名，计划课时，学分） 学生（学号，姓名，性别，专业，班级） 讲授（教师编号，课号，实际课时） 学习（学号，课号，成绩）



（2）规范数据表

 从数据关系模型映射出来的数据表是直接建立在用户应用域基础上的数据表。实际上，同 一个数据关系模型可以有许多种不同的数据表组合。为了使数据库逻辑结构更加科学合理，设 计过程中，一般还需要按照关系数据库规范原理对数据表进行规范化处理，由此可消除或减少 数据表中存在的不合理现象，例如数据存储冗余、数据更新异常。 关系数据库规范原理是基于数据冗余程度提出的，包含第一范式（1NF）、第二范式（2NF） 、 第三范式（3NF）、BC 范式（BCNF）、第四范式（4NF）和第五范式（5NF）。其中，第一范式规范 化程度最低，数据表内部多余联系最多，数据冗余最大；第五范式规范化程度最高，数据表内 部几乎没有多余的联系，数据冗余最小。 显然，通过提高数据表范式级别可以降低数据表中的数据冗余，并可减少由于数据冗余造 成的数据更新异常。但是，为了提高数据表范式级别，就需要清除数据表内部的多余联系，这 就需要对数据表进行分解。 值得注意的是，对数据表的分解大都会使对数据的查询操作复杂起来（例如多表联接操作），由此会降低数据查询性能。 在数据库实际应用中，为了既能使数据冗余与数据更新异常现象有所减少，又能使数据查 询性能不会出现显著下降，大多选用第三范式作为设计优化依据。

下面是对数据表中第一范式、第二范式和第三范式的描述。

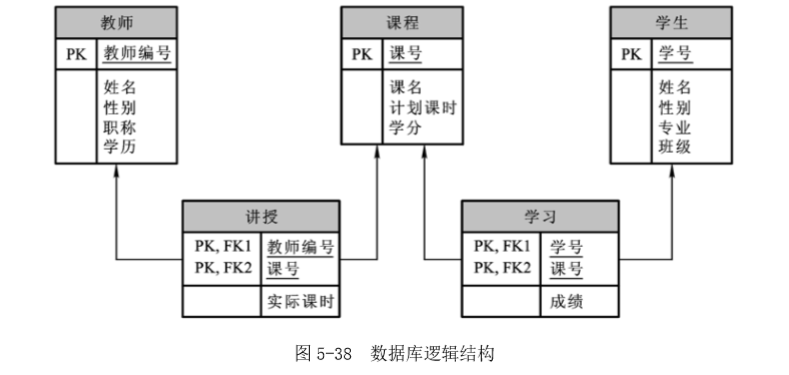
　　a.第一范式 数据表中的每一个字段值都必须是不可再进行分割的原子数据。

　　b.第二范式 满足第一范式条件，而且已经消除数据表中可能存在的非关键字段对关键字段集中个别字 段的部分依赖关系。也就是说，每一个非关键字段都只能由整个关键字段集决定，而不能由关 键字段集中的个别字段决定。

　　c.第三范式 满足第二范式条件，而且已经消除数据表中可能存在的非关键字段之间的传递依赖关系。 也就是说，每一个非关键字段都只能由关键字段集决定，而不能由非关键字段集决定。 一般情况下，如果数据表中的数据需要经常更改，这个数据表可以选用第三范式，甚至 BC 范式。但如果数据表中的数据不会更改或很少更改，却经常需要查询，并且要求有较高的查询 性能，则可以选择第二范式，甚至第一范式。

（3）关联数据表

关联数据表就是将数据关系模型中数据实体之间的关系在数据库逻辑结构中明确体现出 来，它们将作为建立数据表之间参照完整性规则的依据。 图 5-38 是数据库的逻辑模型图，其中的连线表示了数据表之间的关联，连线带箭头一端 为主表，另一端为从表。其中，标记 PK 表示主表中的主键，标记 FK 表示从表中的外键。主表 与从表的关联就建立在主表的主键字段集和从表的外键字段集之间。



（4）设计数据视图

数据视图也称做虚表，原因在于数据视图与数据表一样，都可以将数据以记录集合形式表 现出来。但是，数据视图是面向系统前端应用的不同用户的局部逻辑层。因此，它与面向系统后台全局数据存储的数据表也就有着许多不同之处。 一般说来，数据表具有相对稳定的存储结构，它所存储的数据是具体的，并最终会以物理 形态在数据库中出现。然而，数据视图却是面向用户的外模式，它并不涉及具体的数据存储， 而只是嵌入在数据库中的 SQL 查询操作，因此可以根据用户的应用需要进行比较灵活的数据组 合，满足不同用户的数据应用需要。 数据视图的作用是能够使数据表现与数据存储之间进行有效的逻辑隔离。实际上，数据视 图为数据库前端应用提供了许多便利。例如，可以使用一些更符合前端用户习惯的数据名称， 可以根据用户对数据的特殊需要进行专门的数据视图定义，可以使数据库中的数据具有更高的 安全性，可以简化前端程序员对数据库的复杂编程等。

2.物理结构设计

数据库最终是要存储在物理设备上的。数据库在物理设备上的存储结构与存取方法称为数 据库的物理结构。为一个给定的逻辑数据模型选取一个最适合于应用的物理结构的过程，就是 数据库的物理结构设计。 数据库物理结构设计总是需要依赖于给定的计算机系统，例如，所选用的 DBMS 的特点， 需要开发的应用系统对处理频率和响应时间的要求等。

（1）数据存储结构

在确定数据库中数据的存储结构时，需要综合考虑数据存取时间、存储空间利用率和维护 代价这三方面的因素。并且，这三个方面还常常是相互矛盾的，例如，消除一切冗余数据虽然 能够节约存储空间，但往往会导致检索代价的增加，因此必须进行权衡，选择一个折中方案。 为了提高系统性能，有必要根据数据应用情况将易变部分与稳定部分、经常存取部分和存 取频率较低部分分开存放。例如数据库数据备份、日志文件备份等，由于只在故障恢复时才使 用，而且数据量很大，可以考虑存放在磁带上。

（2）数据索引与聚集

为了提高对数据表中数据的查询速度，可以在数据表的字段或字段集上建立索引。需要注 意的是，索引虽然可以提高查询速度，但索引却需要占用磁盘空间，并且会降低数据更新速度。 因此，对于是否设置索引，往往需要根据实际应用进行权衡。如果数据需要频繁更新或磁盘空 间有限，则不得不限制索引个数。 许多关系型 DBMS 还提供了聚集索引功能，与一般索引比较，它能够带来更高的查询效率。 但必须注意的是，聚集索引只能提高对某些特定字段的查询性能，而且会带来更大的维护开销 和存储空间消耗。因此，只有在聚集字段是最主要的查询字段时，才有建立聚集的必要。

（3）数据完整性

为了使数据库中的数据更加便于维护，还需要在数据库中建立数据完整性规则，包括实体 完整性和参照完整性。 实体完整性是指数据库对数据表中记录的惟一性约束。为了使数据表具有实体完整性，需 要在数据表中设置主键，由此可确保数据表中的每一条记录都是惟一的，也就是说不会出现重 复记录。 数据库参照完整性则是指建有关联的数据表之间存在的“主表”对“从表”的一致性约束。由此可使从表中外键字段的取值能够受到主表中主键字段取值的限制。例如，只有当主表的主 键字段中存在该值后，从表的外键字段才能取用该值；并可使得当主表中主键字段值更改时， 从表中对应的外键字段值可以自动同步更新，或使得当主表中有记录删除时，从表中外键字段 值和主表中主键字段值相同的相关记录也被一同删除。

小结:

1．设计过程与任务 概要设计中首先需要进行的是系统构架设计，然后是软件结构、数据结构等方面的设计。 主要有以下几个方面的设计任务：制定规范、系统构架设计、软件结构设计、公共数据结构设 计、安全性设计、故障处理设计、可维护性设计、编写文档、设计评审。

2．系统构架设计

（1）集中式结构 集中式系统由一台计算机主机和多个终端设备组成。其具有非常好的工作稳定性和安全保 密性。但系统建设费用、运行费用比较高，灵活性不够好，结构不便于扩充。

（2）客户机/服务器结构 客户机/服务器结构依靠网络将计算任务分布到许多台不同的计算机上，但通过其中的服 务器计算机提供集中式服务。其优越性是结构灵活、便于系统逐步扩充。

（3）多层客户机/服务器结构 •  两层结构：将信息表示与应用逻辑处理都放在了客户机上，服务器只需要管理数据库 事务。

•  三层结构：将两层结构的客户机上的容易发生变化的应用逻辑部分提取出来，并放到一 个专门的“应用服务器”上。

•    B/S 结构：是 Web 技术与客户机/服务器结构的结合。其优点是不需要对客户机进行专 门的维护。

（4）组件对象 分布式结构通过组件进行计算分布。它依赖于对象中间件建立，具有灵活的构架，系统伸 缩性好，能够给系统的功能调整与扩充带来便利。

3．软件结构设计 软件结构设计是对组成系统的各个子系统的进一步分解与规划。主要设计内容有：确定模 块元素、定义模块功能、定义模块接口、确定模块调用与返回、进行结构优化。

（1）模块概念

•  模块化：使用构造程序，可使软件问题简化。

•  抽象化：概要设计中的模块被看成是一个抽象化的功能黑盒子。

•  信息隐蔽：每个模块的内部实现细节对于其他模块来说是隐蔽的。

（2）模块的独立性 软件系统中每个模块都只涉及自己特定的子功能，并且接口简单，与软件中其他模块没有 过多的联系。一般采用耦合和内聚这两个定性的技术指标进行度量。耦合用来反映模块相互关联程度，模块间连接越紧密，耦合性就越高。内聚用来反映模块 内元素的结合程度，模块内元素结合越紧密，则内聚性就越高。为提高模块独立性，要求模块 高内聚、低耦合。 耦合形式由低至高是：非直接耦合、数据耦合、控制耦合、公共耦合、内容耦合。 内聚形式由低至高是：偶然内聚、逻辑内聚、时间内聚、过程内聚、通信内聚、顺序内聚、 功能内聚。

（3）设计建模

•  软件结构图：由 Yourdon 于 20 世纪 70 年代提出，被广泛应用于软件结构设计中，能够 有效说明软件中模块之间的调用与通信。

•    HIPO 图：由美国 IBM 公司推出。其中，H 图用于描述软件的分层调用关系，作用类似软 件结构图，IPO 图用于说明描述模块的输入—处理—输出特征。

（4）软件结构优化 主要优化设计原则有：使模块功能完整、 使模块大小适中、使模块功能可预测、尽量降 低模块接口的复杂程度、使模块作用范围限制在其控制范围之内、模块布局合理。

4．面向数据流的结构设计

（1）变换分析 软件结构由输入、变换和输出三个部分组成。

（2）事务分析 软件结构由接收事务与事务活动两个部分组成。

（3）混合流分析与设计 软件系统是变换流与事务流的混合。对于这样的系统，通常采用变换分析为主、事务分析 为辅的方式进行软件结构设计。

5. 数据库结构设计

（1）逻辑结构设计

•  设计数据表

•  规范数据表

•  关联数据表

•  设计数据视图

（2）物理结构设计

•  数据存储结构

•  数据索引与聚集

•  数据完整性