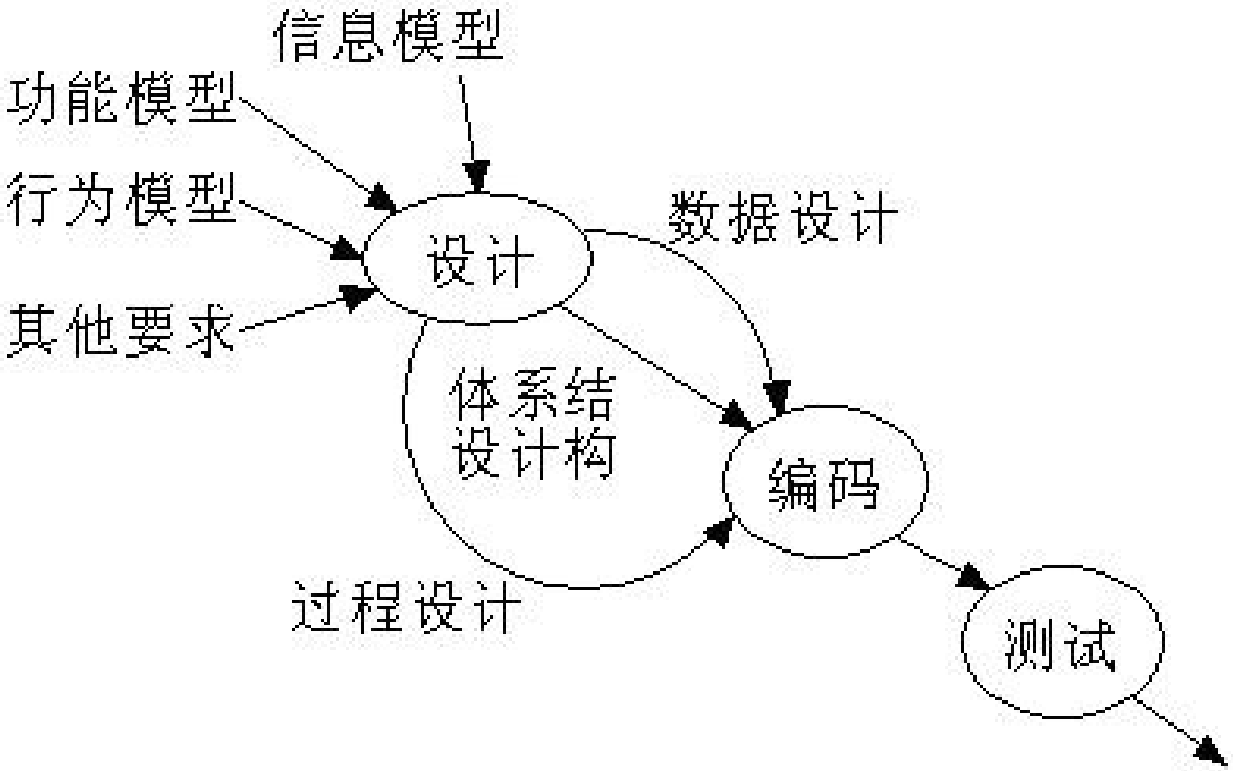
第5章 总体设计

本章内容：

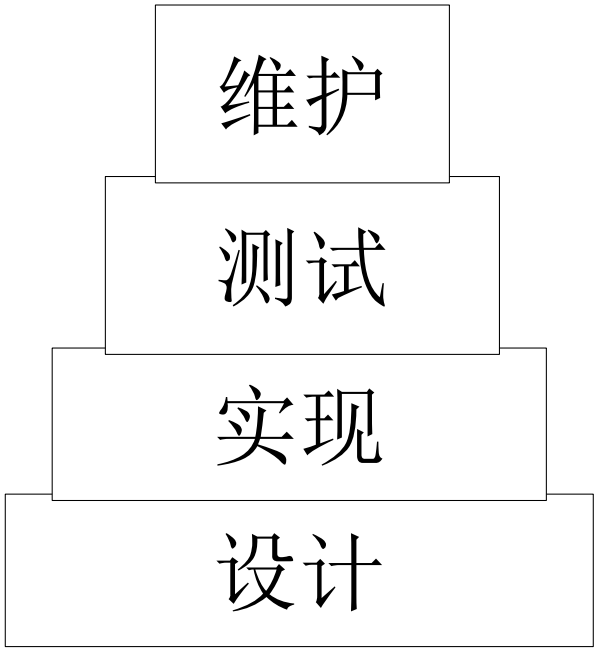
5.1 软件设计的重要性

软件设计处于软件工程过程的技术核心地位。



在设计中所作的决策将最终影响软件实现的成功与否、也影响软件维护的难易程度。所以，在软件设计过程中的这些决策是开发阶段非常关键的一步。

软件设计的重要性还反映在质量（quality）上。



有设计

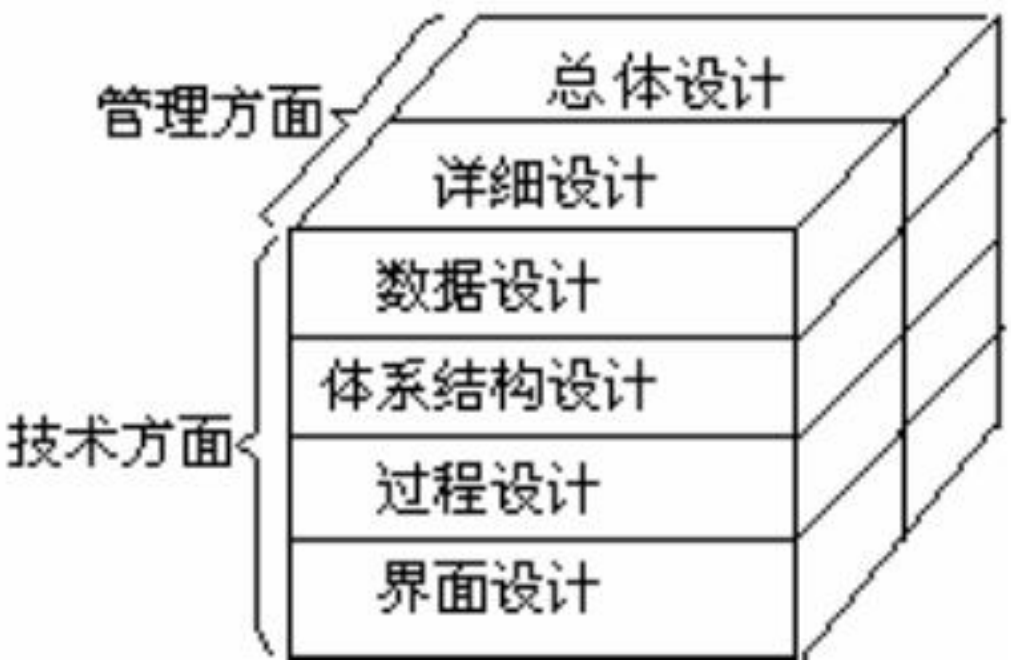
没有设计

5.2 设计过程

软件设计是一个把需求转换为软件表达式的过程。

从软件工程的角度讲是分为总体设计和详细设计。总体设计主要是把需求转换为数据结构和软件体系结构，而详细设计主要集中在体系结构表达式的细化，从而产生详细的数据结构和软件的算法表达式。

总体设计和详细设计除了必须有先进的设计技术外，还要有同步的管理技术支持。



软件设计原则：

（1）设计应当模块化（modular），也就是说，软件应被逻辑地划分为能完成特定功能和子功能的构件。

（2）设计应形成具有独立功能特征的模块（如子程序或过程）。

（3）设计应使模块之间和与外部环境之间接口的复杂性尽量地减少。

（4）设计应该有一个分层的组织结构，这样人们可对软件各个构件进行理性的控制。

（5）设计应有性质不同的可区分的数据和过程表达式。

（6）设计应利用软件需求分析中得到的信息和可重复的方法。

5.3 软件总体设计

1. 软件系统结构设计

（1）采用某种设计方法，将一个复杂的系统按功能划分成模块。

（2）确定每个模块的功能。

（3）确定模块之间的调用关系。

（4）确定模块之间的接口，即模块之间传递的信息。

（5）评价模块结构的质量。

2. 数据结构及数据库设计

1）数据结构的设计

根据需求分析阶段对系统数据的组成、操作约束和数据之间的关系的描述，确定数据结构特性。

2）数据库的设计

一般的软件系统都有数据的存储，存储要借助数据库技术。数据库的设计是指数据存储文件的设计。设计包括以下三个方面：

（1）概念设计。

（2）逻辑设计。

（3）物理设计。

3. 网络系统设计

如果采用的是网络环境，则要进行网络系统的设计。要分析网络负荷与容量，遵照网络系统设计原则，确定网络系统的需求。要进行网络结构设计，选择好网络操作系统，确定网络系统配置，制定网络拓扑结构。

4. 软件总体设计文档

总体设计说明书是总体设计阶段结束时提交的技术文档。按国标GB8576–88的《计算机软件产品开发文件编制指南》规定，软件设计文档可分为“总体设计说明书”、“详细设计说明书”和“数据库设计说明书”。这些文档的内容与格式请参考有关资料。

5. 评审

在该阶段，对设计部分是否完整地实现了需求中规定的功能、性能等要求，设计方案的可行性、关键的处理及内外部接口定义正确性、有效性以及各部分之间的一致性等，都一一进行评审。

5.4 设计基本原理

软件设计要回答下列问题：

（1）使用什么样的准则才能把软件划分成为各个单独的构件？

（2）怎样把功能或数据结构的细节从软件概念表达式中分离出来？

（3）定义软件设计的技术质量有统一的准则吗？

5.4.1 抽象

抽象是认识复杂现象过程中使用的思维工具，即抽出事物本质的共同特性而暂不考虑它的细节，不考虑其他因素。

随着对抽象不同层次的展开，过程抽象（Procedural Abstraction）和数据抽象（Data Abstraction）就建立了。

5.4.2 细化

逐步细化是一种自顶向下的设计策略。程序的体系结构开发是由过程细节层次不断地细化而成的。分层的开发则是以逐步的方式由分解一个宏功能直到获得编程语言语句。

细化实际上是一个详细描述的过程。在高层抽象定义时，从功能说明或信息描述开始，就是说明功能或信息的概念，而不给出功能内部的工作细节或信息的内部结构。细化则是设计者在原始说明的基础上进行详细说明，随着不断的细化（详细说明）给出更多的细节。

5.4.3 模块化

模块具有以下几种基本属性：

（1）接口：指模块的输入与输出。

（2）功能：指模块实现什么功能。

（3）逻辑：描述内部如何实现要求的功能及所需的数据。

（4）状态：指该模块的运行环境，即模块的调用与被调用关系。

模块化是指解决一个复杂问题时自顶向下逐层把软件系统划分成若干模块的过程。每个模块完成一个特定的子功能，所有的模块按某种方法组装起来，成为一个整体，完成整个系统所要求的功能。

问题复杂性与工作量关系

设问题x，表示它的复杂性函数为C(x)，解决它所需的工作量函数为E(x)。对于问题P1和P2；如果：

C(P1)＞C(P2)

即P1比P2复杂，那么：

E(P1)＞E(P2)

即问题越复杂，所需要的工作量越大。

根据解决一般问题的经验，规律为：

C(P1+P2)＞C(P1)+C(P2)

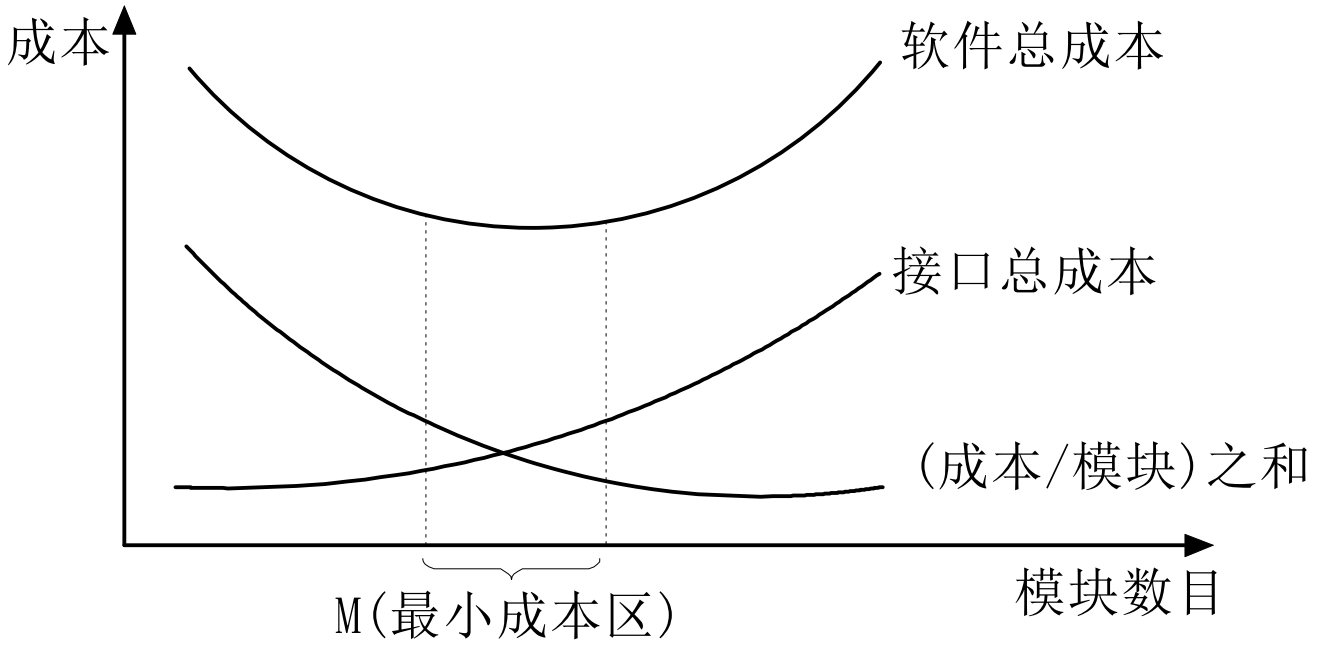
即一个问题同问题组合而成的复杂度大于分别考虑每个问题的复杂度之和。这样，可以推出：

E(P1+P2)＞E(P1)+E（P2)

所得结果对于模块化和软件具有重要的意义。那么，从上面所得的不等式是否可以得出这样的结论：

如果把软件无限细分，那么最后开发软件所需要的工作量就小得可以忽略了。但是，事实上，影响软件开发工作量的因素还有许多

模块数目与软件成本



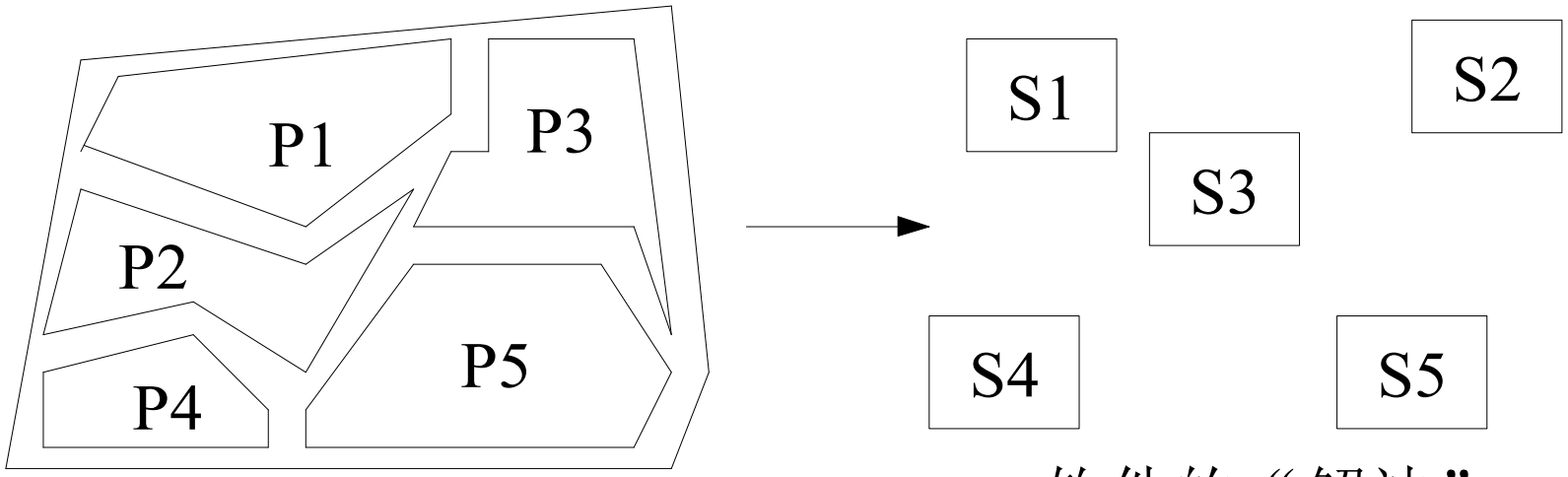
5.4.4 软件体系结构

软件总体设计的主要任务就是软件结构的设计。软件体系结构（software architecture）包含了计算机程序的两个重要特性：

（1）过程构件（模块）的层次结构。

（2）数据结构。

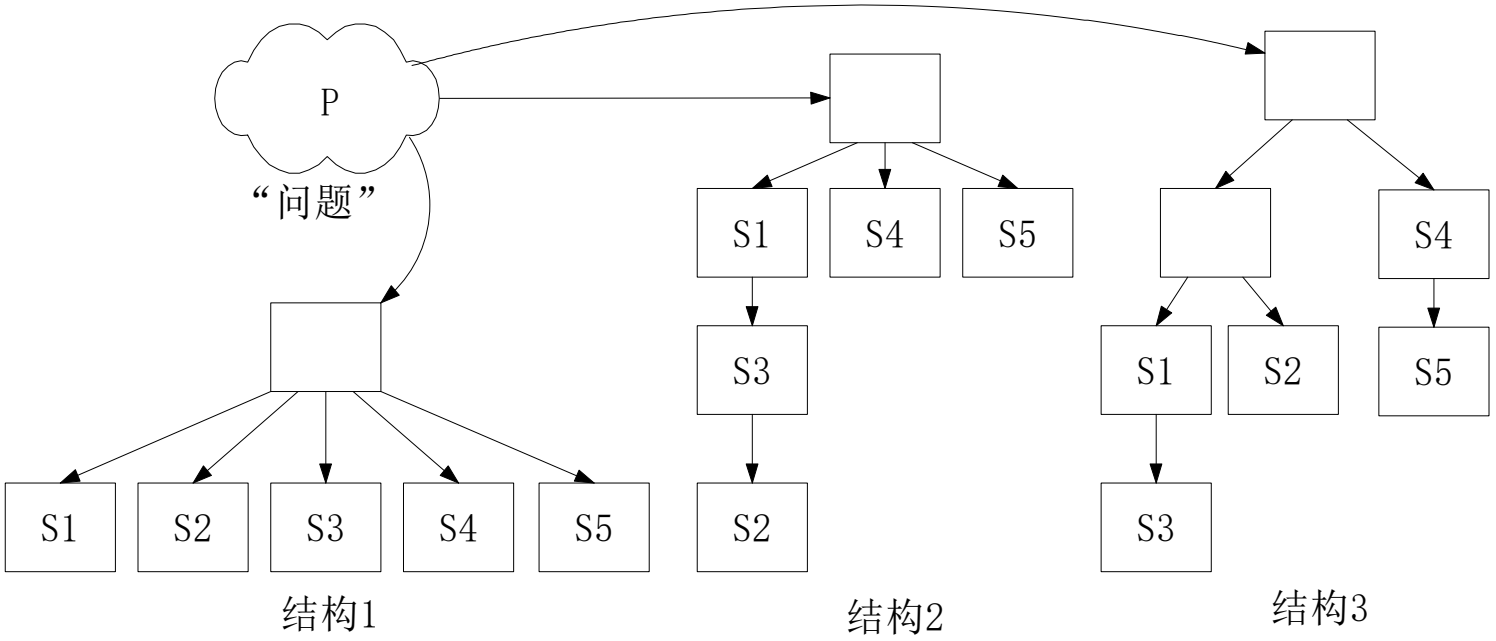
结构化演化



软件求解的“问题”

软件的“解法”

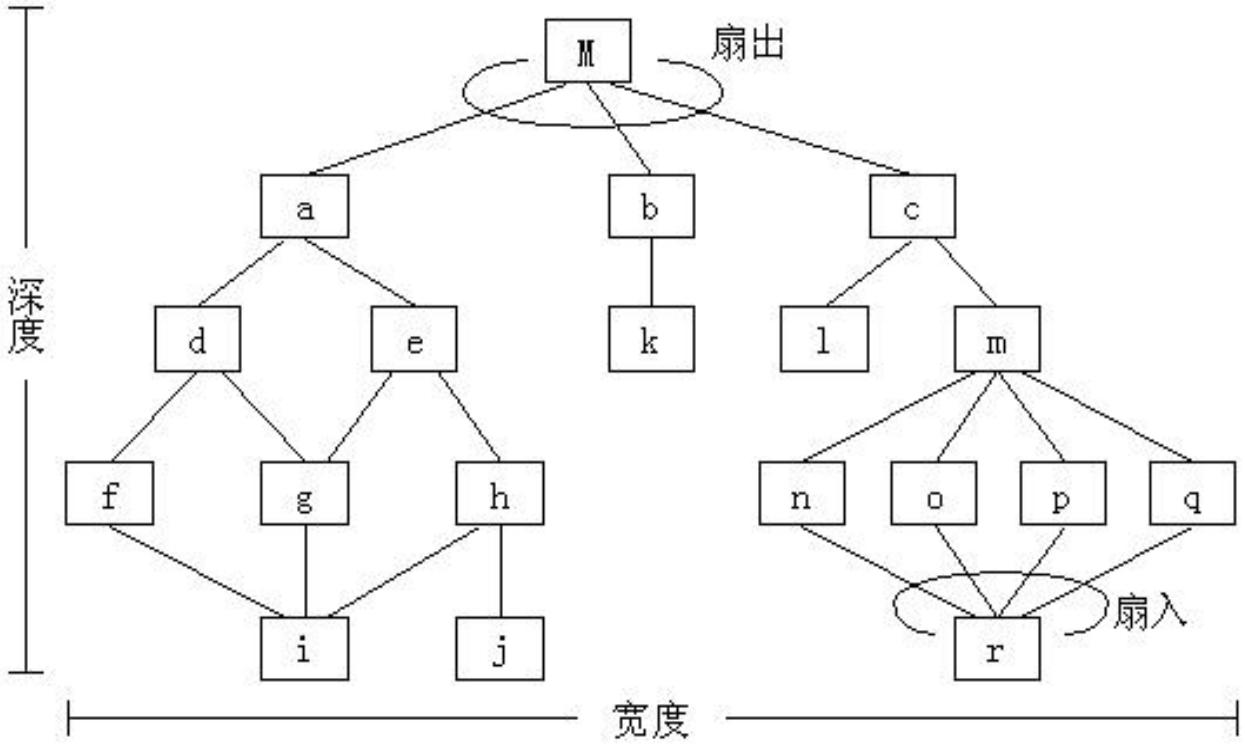
不同结构



5.4.5 程序结构

程序结构（program structure）给出了程序构件（模块）的组织（通常叫分层），这种组织包含了控制的层次。它们不给出软件的过程方面，如过程的序列、决策的出现或次序，或操作的重复等。

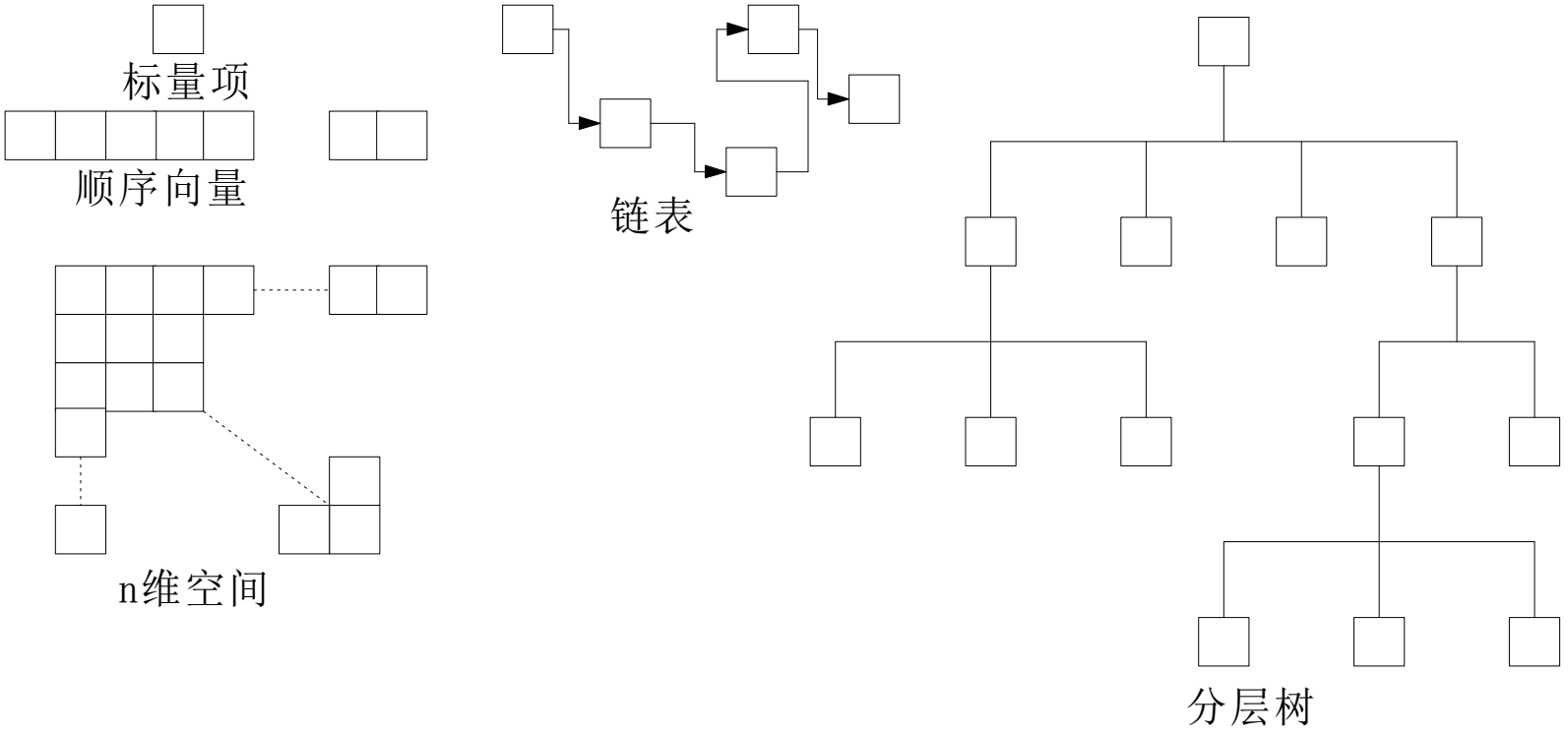
结构专用名词



5.4.6 数据结构

在软件体系结构的表达式中，数据结构与程序结构同样重要。数据结构决定信息的组织、存取方法、结合的程度，以及可选的处理方法。

典型的数据结构



当组织标量项作为一列或连接的组时，就形成一个顺序向量（sequential vector）。向量是数据结构中最常用的。可看下面这个C语言程序段的例子：

```
int aa[100];
…
procedure ps （int aa； int n； int sum）
{ int i;
{
```

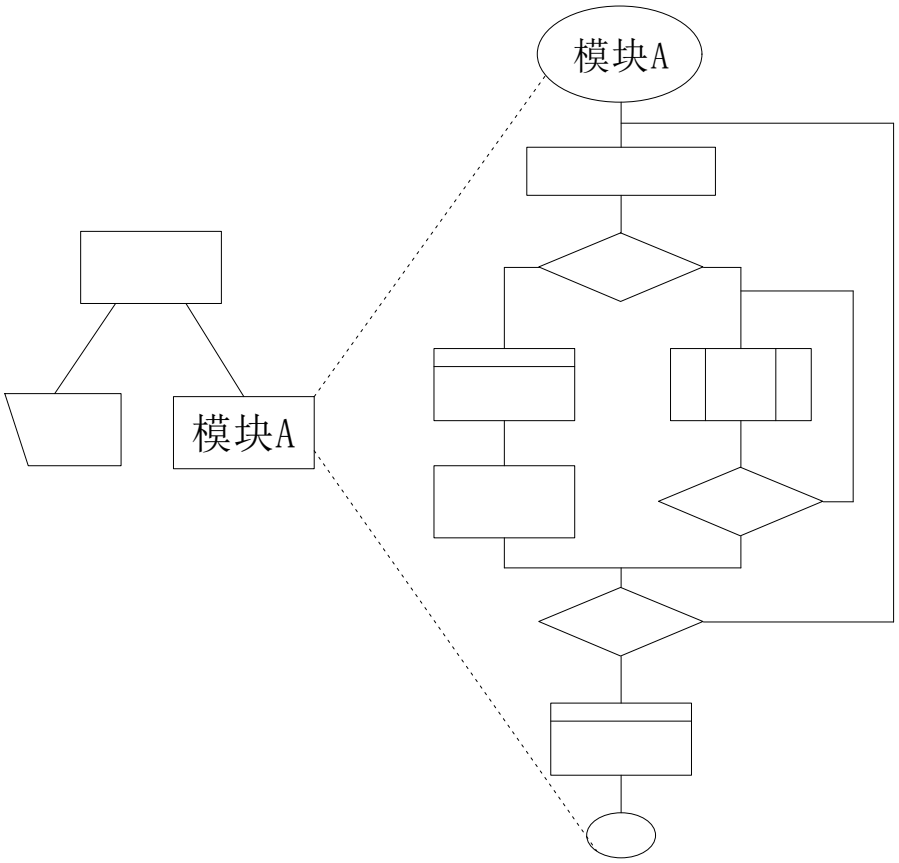
```
sum=0;
   for （i=1;i<= n ;i++）
    sum=sum +aa[i];
};
}
…
```

例子中定义aa为100个标量整数项的顺序向量。在过程ps中标引aa的每个元素的存储，这样数据结构的每个元素都可按定义的顺序引用。

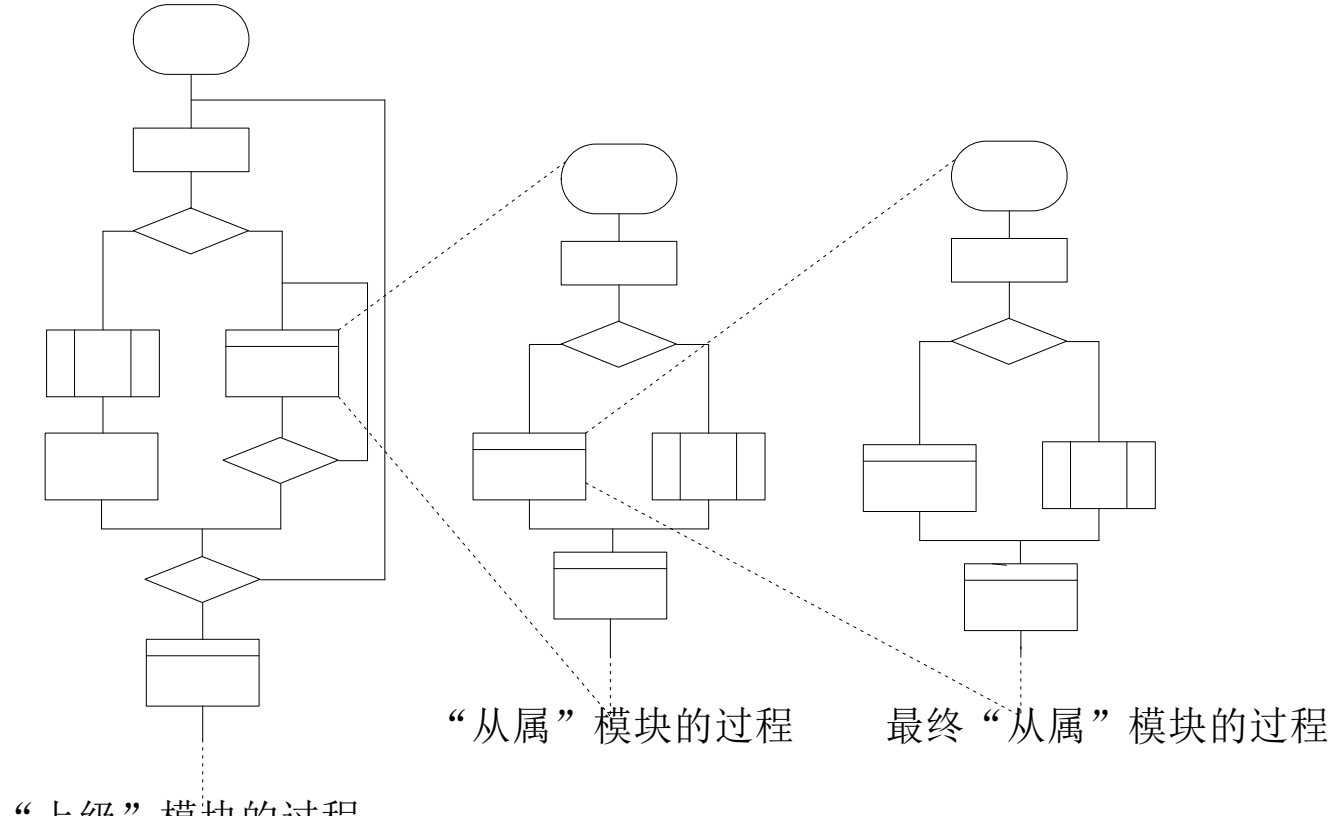
5.4.7 软件过程

软件过程（software procedure），则侧重于每一个单独模块的处理细节研究。过程必须提供精确的事件的顺序、确切的抉择点、重复的操作，以及数据的组织与结构处理规格说明。

一个模块内的过程



过程的分层



5.5 体系结构设计

软件体系结构设计（architectural design）的主要目标是设计一个模块化的程序结构。体系结构设计融合了程序结构和数据结构，接口定义能使数据流经程序。要给出各个模块之间的控制关系。

5.5.1 软件结构图

软件结构图是软件系统的模块层次结构，反映了整个系统的功能实现，即将来程序的控制层次体系。

软件结构往往用树状或网状结构的图形来表示。

1. 模块

用方框表示，并用名字标识该模块，名字应体现该模块的功能。

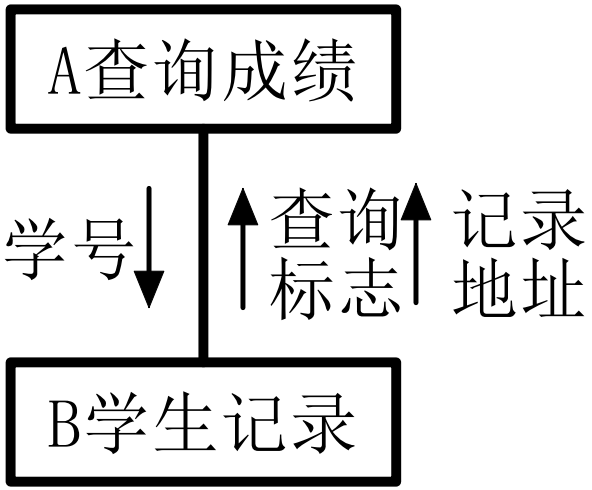
2. 模块的控制关系

两个模块间用单向箭头或直线连接起来表示它们的控制关系。

3. 模块间的信息传递

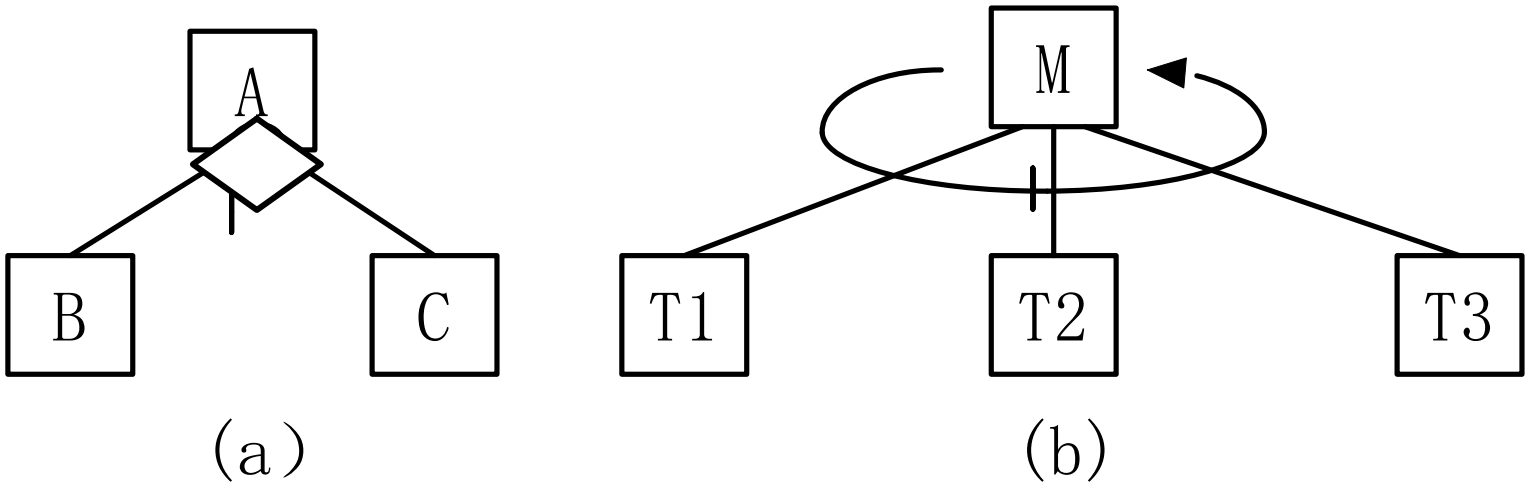
模块间还经常用带注释的短箭头表示模块调用过程中来回传递的信息。

模块间的控制关系及信息传递

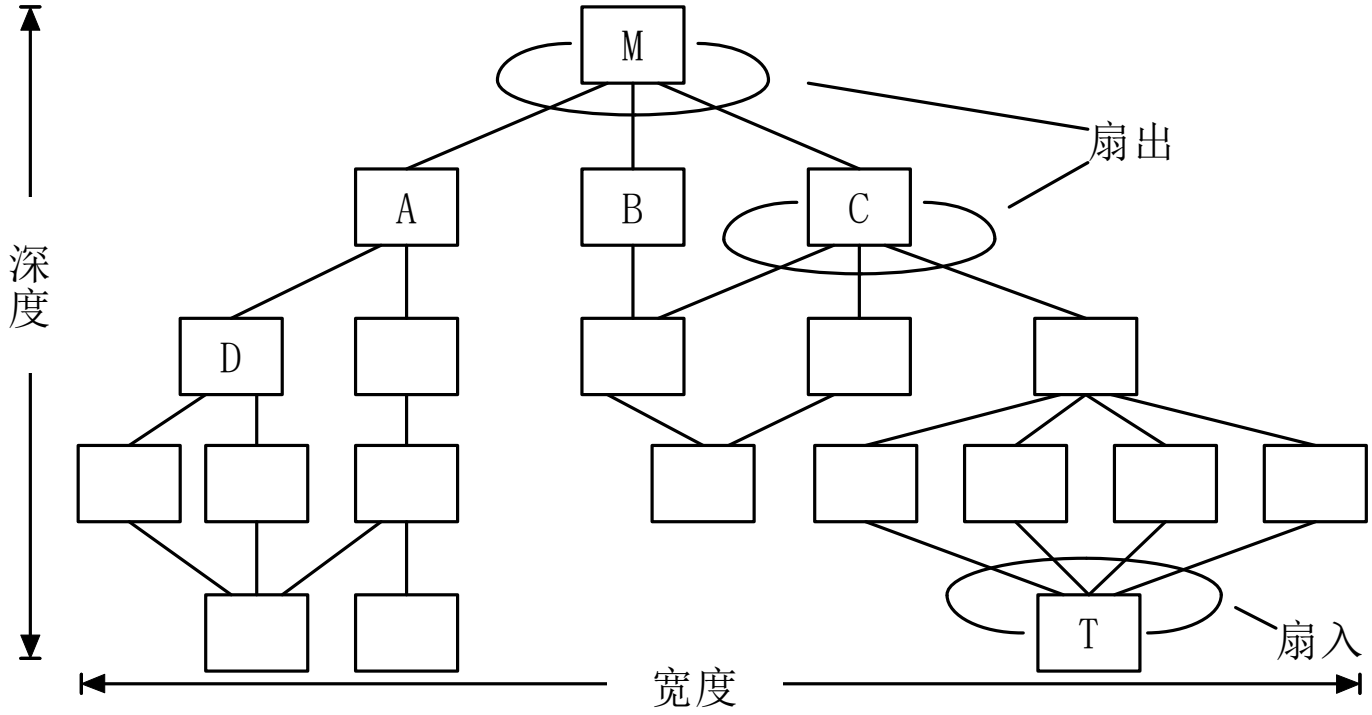


4. 两个附加符号

表示模块有选择调用或循环调用



5. 结构图的形态特征



结构图的形态特征包括：

（1）深度：指结构图控制的层次，即模块的层数。

（2）宽度：指一层中最大的模块个数。

（3）扇出：指一个模块直接下属模块的个数。

（4）扇入：指一个模块直接上属模块的个数。

6. 画结构图应注意的事项

（1）同一名字的模块在结构图中仅出现一次。

（2）调用关系只能从上到下。

（3）不严格表示模块的调用次序，习惯上从左到右。

5.5.2 模块的大小

前面在讨论模块设计的原理时，已经知道一个系统应当由若干个模块构成。其目的是为了降低系统的复杂度。

模块设计的准则不应该是语句的多少，而应当是模块是否是一个独立的功能。

5.5.3 扇出和扇入与深度和宽度

由结构图的形态特征可以知道，一个系统的大小和系统的复杂程度在一定程度上可以用深度和宽度表示。

大量的系统研究表明，认为高层模块应有较高的扇出，低层模块特别是底层模块应有较高的扇入。扇入越大，表示该模块被更多的上级模块共享。多个扇入入口相同，这就避免了程序的重复，因此希望扇入高一点。但过多又可能是把许多不相关的功能硬凑在一起，形成通用模块，这样的模块必然是低聚合的。

扇出实际上是对问题解的分解

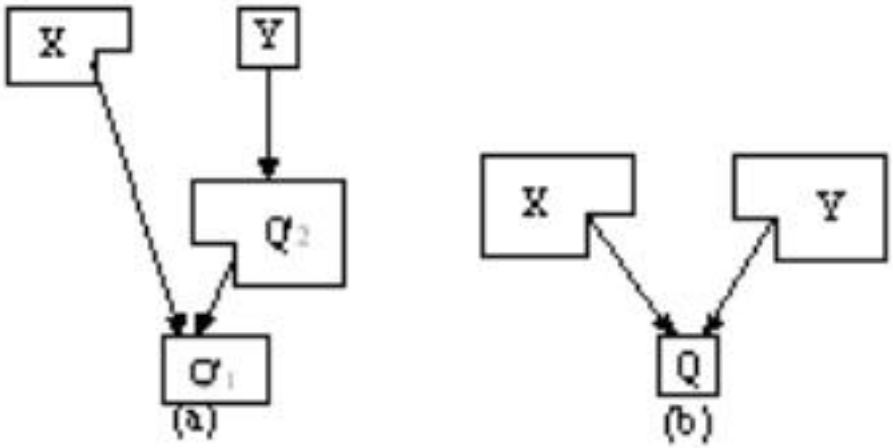


图 5-14　分解模块

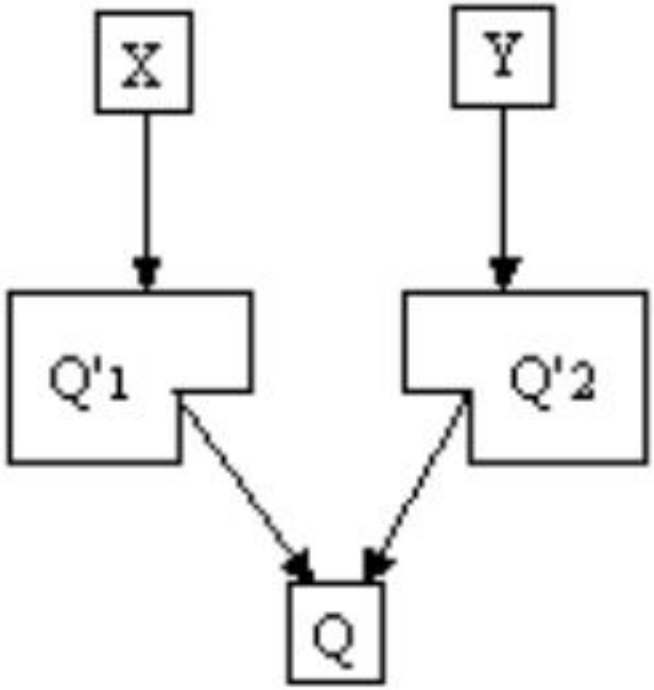


图 5-15　分解模块

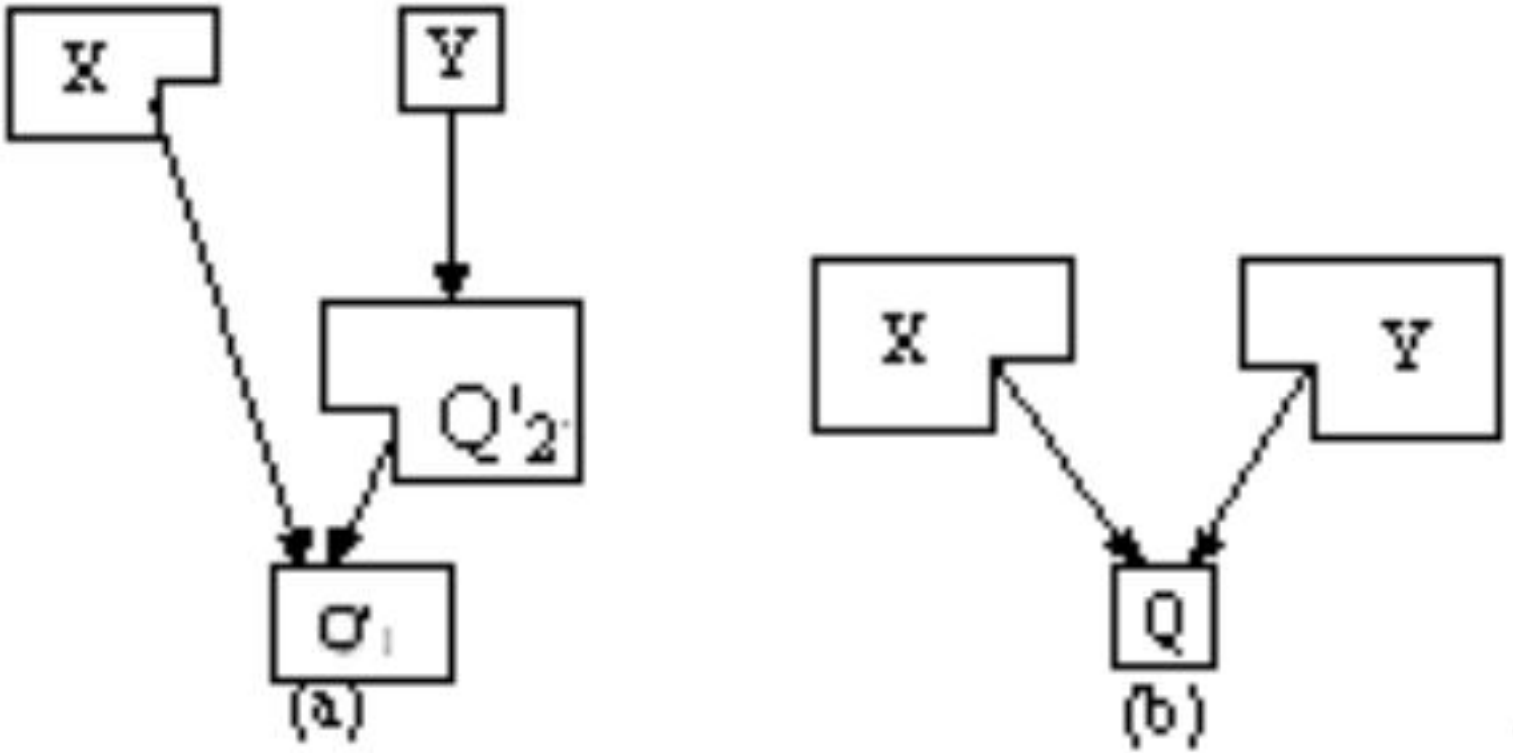


图 5-16　分解模块

5.5.4 模块的耦合

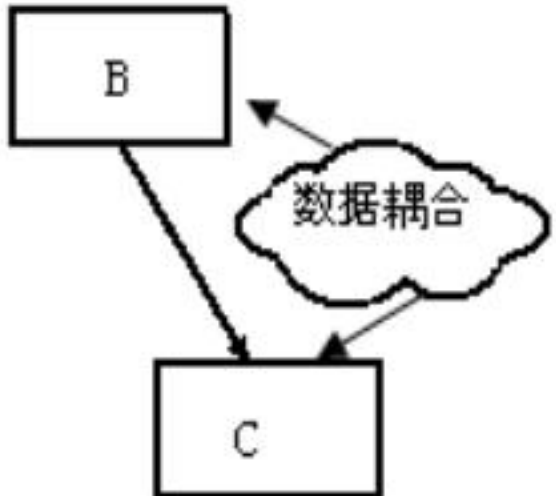
耦合（Coupling）表示软件结构内不同模块彼此之间相互依赖（连接）的紧密程度，是衡量软件模块结构质量好坏的度量，是对模块独立性的直接衡量指标。

耦合强弱取决于模块间接口的复杂程度，进入或访问一个模块的点，以及通过接口的数据。

1. 数据耦合

如果两个模块彼此间通过参数交换信息，而且交换的信息仅仅是数据，那么这种耦合称为数据耦合。

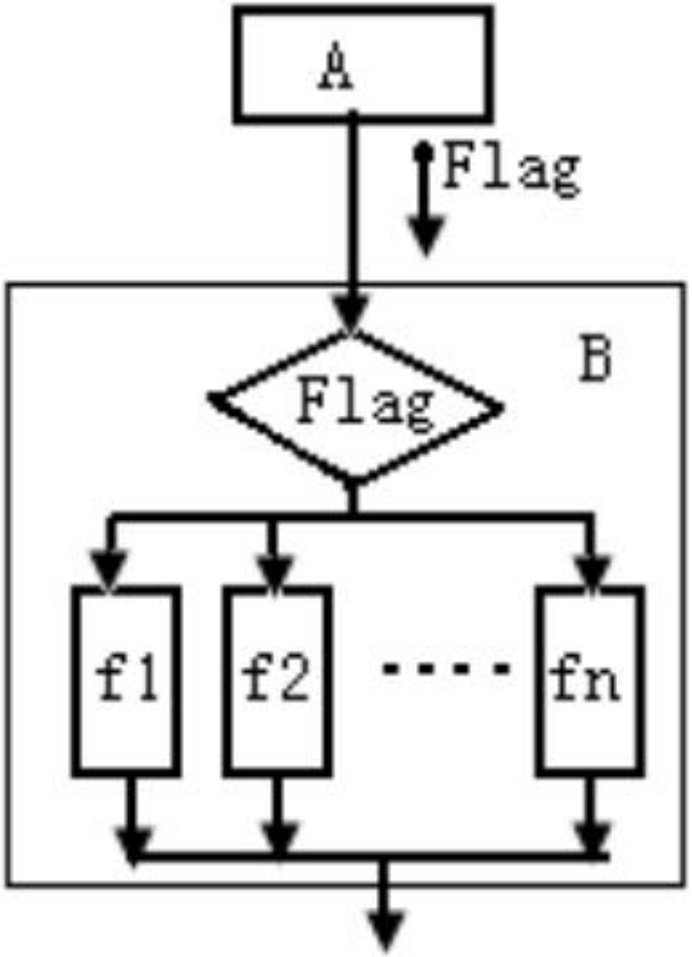
数据耦合是低耦合。系统中必须存在这种耦合，因为只有当某些模块的输出数据作为另一些模块的输入数据时，系统才能完成有价值的功能。



2. 控制耦合

如果传递的信息中有控制信息，则这种耦合称为控制耦合。控制耦合是中等程度的耦合，它增加了系统的复杂程度。

控制耦合往往是多余的，在把模块适当分解之后通常可以用数据耦合代替它。

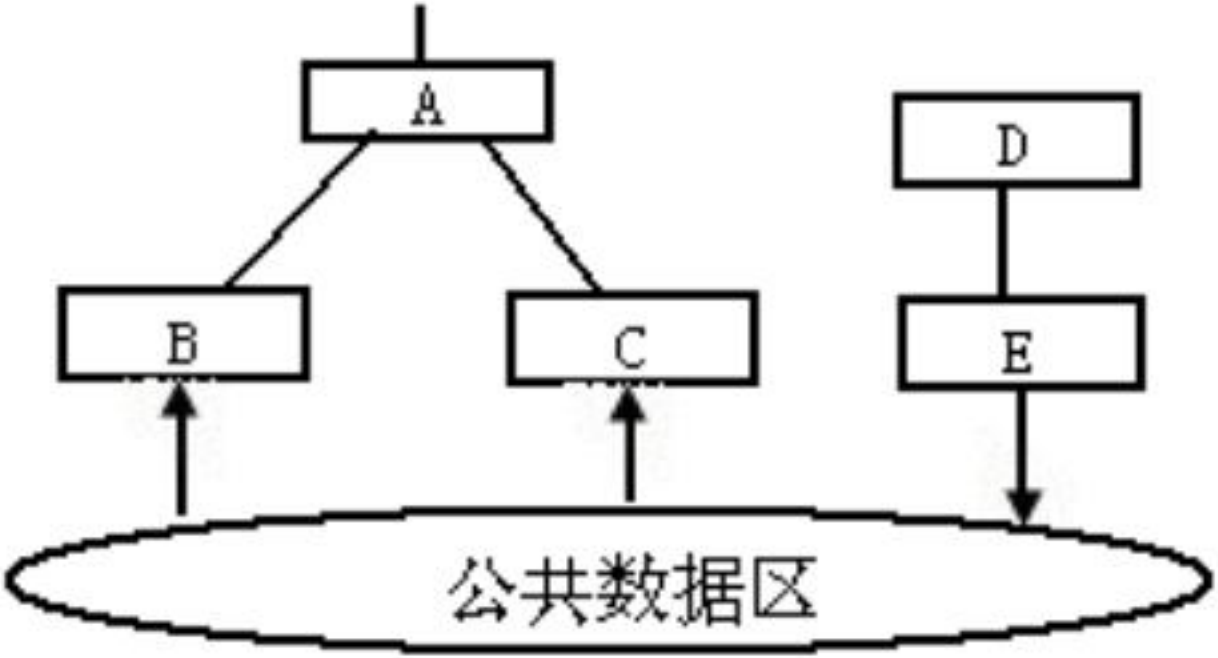

A
Flag
B
Flag
f1
f2
fn

3. 公共环境耦合

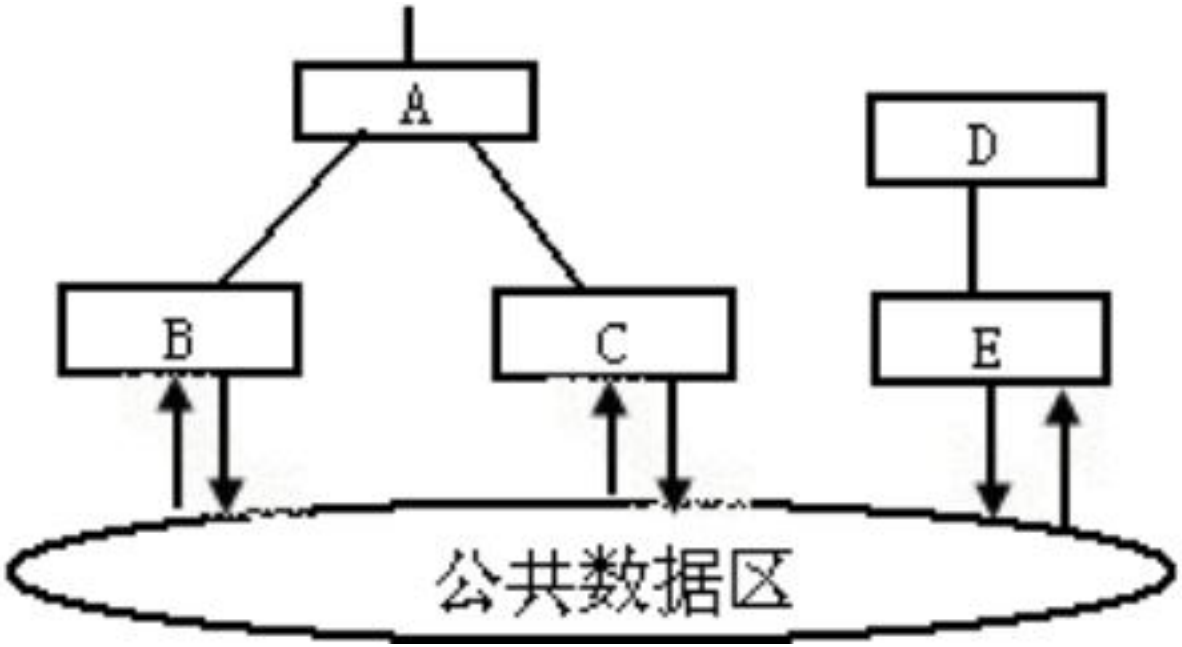
当两个或多个模块通过一个公共数据环境相互作用时，它们之间的耦合称为公共环境耦合（即公用耦合）。

公共环境耦合的复杂程度随耦合的模块个数而变化，当耦合的模块个数增加时复杂程度显著增加。

松散的公用耦合



紧密的公用耦合



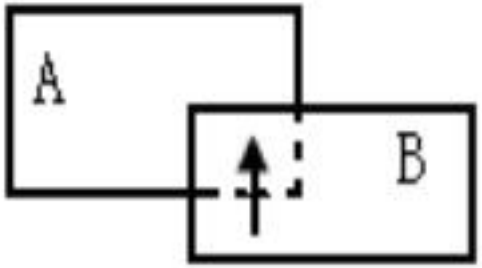
（1）一个模块往公共环境送数据，另一个模块从公共环境取数据。这是数据耦合的一种形式，是比较松散的耦合。

（2）两个模块都既往公共环境送数据又从里面取数据，这种耦合比较紧密，介于数据耦合和控制耦合之间。

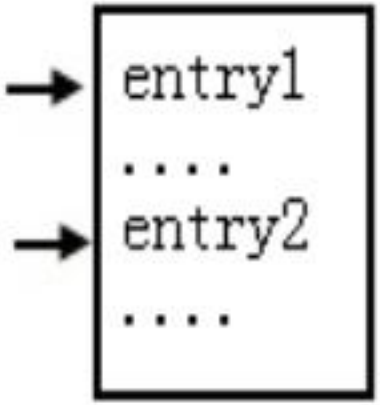
4. 内容耦合

最高程度的耦合是内容耦合。

(a)进入另一块内部



(B) 代码重复



(C)多入口

（1）一个模块访问另一个模块的内部数据；

（2）一个模块不通过正常入口而转到另一个模块的内部；

（3）两个模块有一部分程度代码重叠（只可能出现在汇编程序中）；

（4）一个模块有多个入口（这表明一个模块有几种功能）。

应该坚决避免使用内容耦合。事实上许多高级程序设计语言已经设计成不允许在程序中出现任何形式的内容耦合。

总之，耦合是影响模块结构和软件复杂程度的一个重要因素，应该采用如下设计原则：尽量使用数据耦合，少用控制耦合，限制公共环境耦合，完全不用内容耦合。

5.5.5 模块的内聚

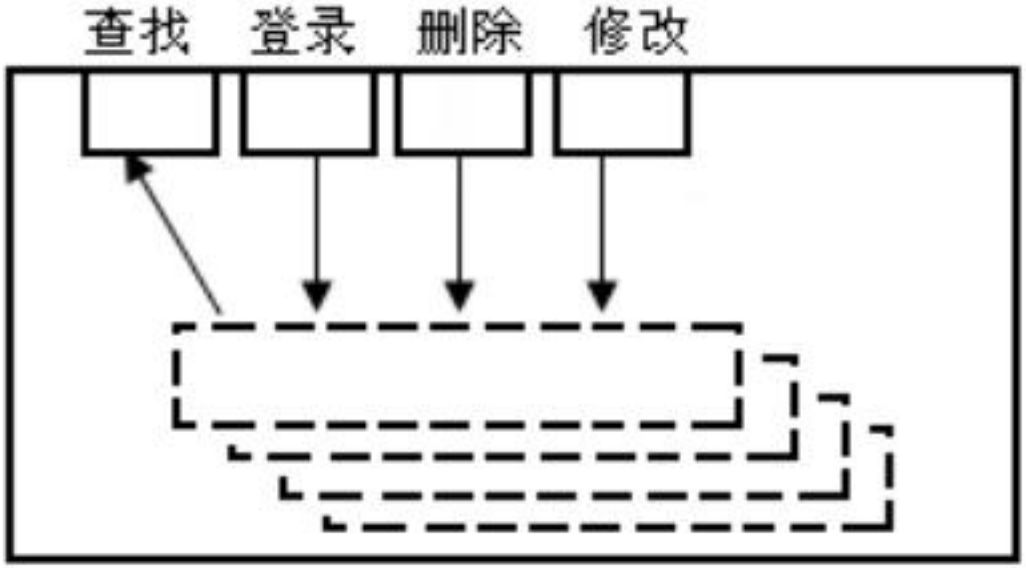
内聚标志一个模块内各个元素彼此结合的紧密程度，它是信息隐蔽和局部化概念的自然扩展。简单地说，理想内聚的模块只做一件事情。

1. 功能内聚

如果模块内所有处理元素属于一个整体，完成一个单一的功能，则称为功能内聚。功能内聚是最高程度的内聚。

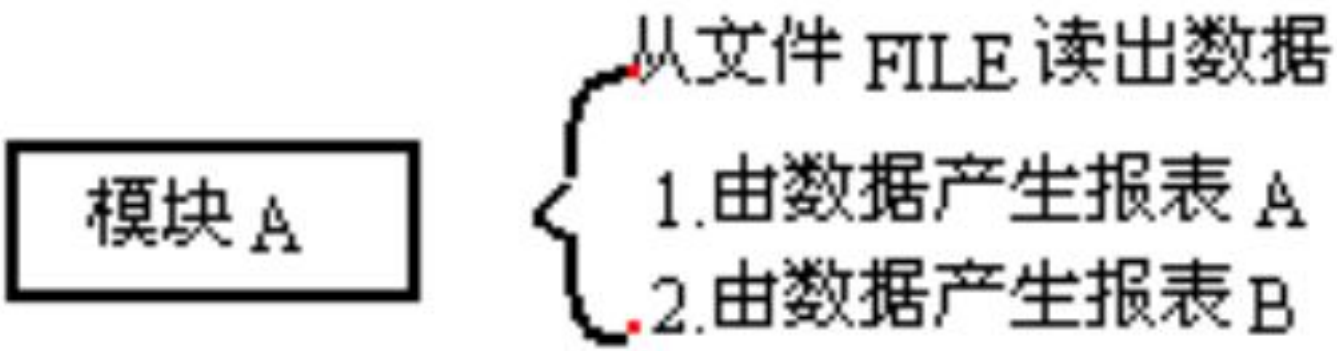
2. 信息内聚

信息内聚模块能完成多种功能，各个功能都在同一数据结构上操作，每一项功能有一个惟一的入口点。



3. 通信内聚

如果一个模块中所有处理元素都使用同一个输入数据和（或）产生同一个输出数据，称为通信内聚（Communicational Cohesion）。



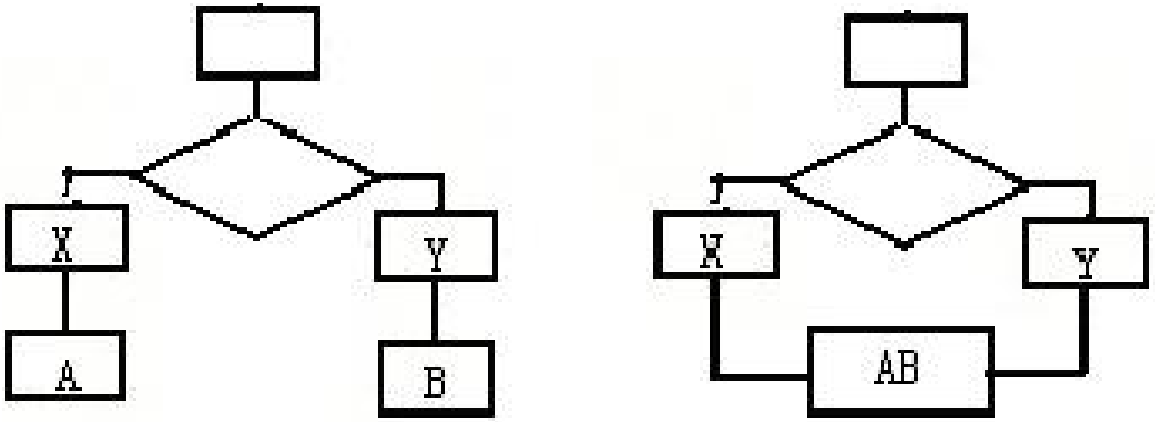
4. 过程内聚

如果一个模块内部的处理元素是相关的，而且必须以特定次序执行，则称为过程内聚。

过程内聚与顺序内聚的区别主要在于：顺序内聚中是数据流从一个处理元流到另一个处理元，而过程内聚中是控制流从一个动作流向另一个动作。

5. 逻辑内聚

如果一个模块完成的任务在逻辑上属于相同或相似的一类，称为逻辑内聚（Logical Cohesion）。

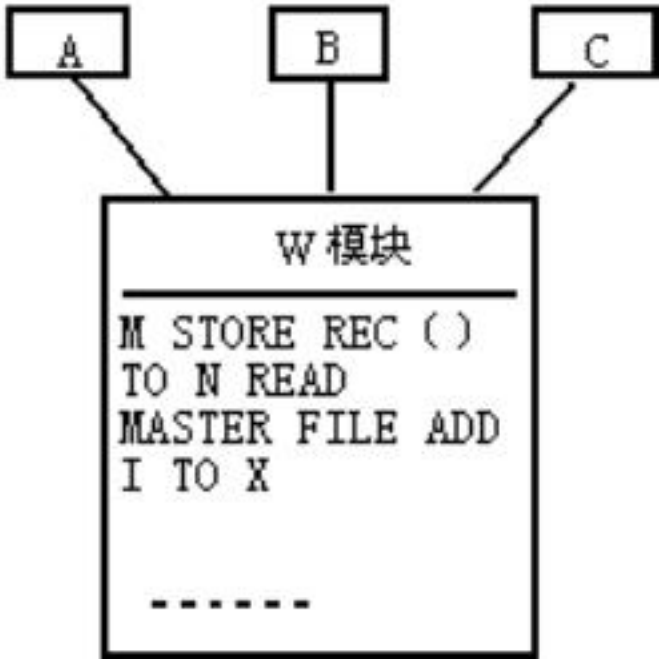


6. 时间内聚

如果一个模块包含的任务必须在同一段时间内执行，就叫时间内聚。

7. 偶然内聚

如果一个模块完成一组任务，这些任务彼此间即使有关系，关系也是很松散的，就叫做偶然内聚。



5.5.6 结构设计的一般准则

1. 模块独立性准则

（1）如果若干模块之间耦合强度过高，每个模块内功能不复杂，可将它们合并，以减少信息的传递和公共区的引用。

（2）若有多个相关模块，应对它们的功能进行分析，消去重复功能。

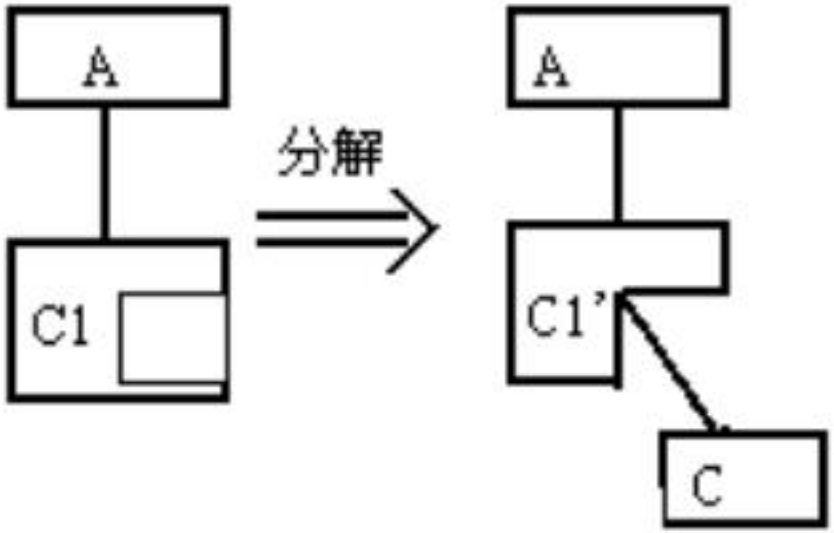


图 5-26　模块的分解

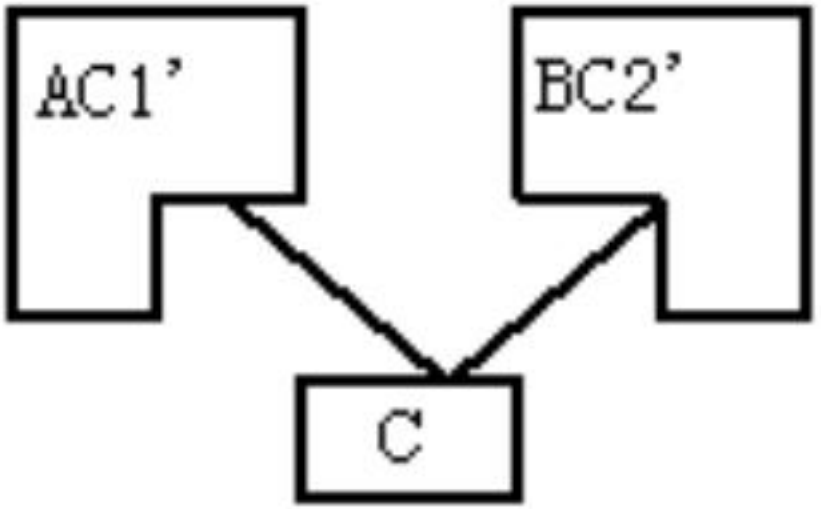


图 5-27　模块的合并

2. 软件结构的形态特征准则

软件结构的深度、宽度、扇入及扇出应适当。

3. 模块的大小准则

在考虑模块独立性的同时，为了增加可理解性，模块的大小最好在50～150条语句左右，可以用1～2页打印纸打印，便于人们阅读与研究。

4. 模块的接口准则

模块的接口要简单、清晰及含义明确，便于理解，易于实现、测试与维护。

模块接口的复杂性是软件发生错误的一个重要原因。因此，设计模块接口时，应尽量使传递的信息简单并与模块的功能一致。

5.5.7 模块的作用域与控制域

一个模块的作用范围应在其控制范畴之内，且条件判定所在的模块应与受影响的模块在层次上尽量靠近。

如果在设计过程中，发现模块作用范围不在其控制范围之内，可用以下方法加以改进：

（1）上移判断点。使该判断的层次升高，以扩大它的控制范围。

（2）下移受判断影响的模块。将受判断影响的模块下移到判断所在模块的控制范围内。

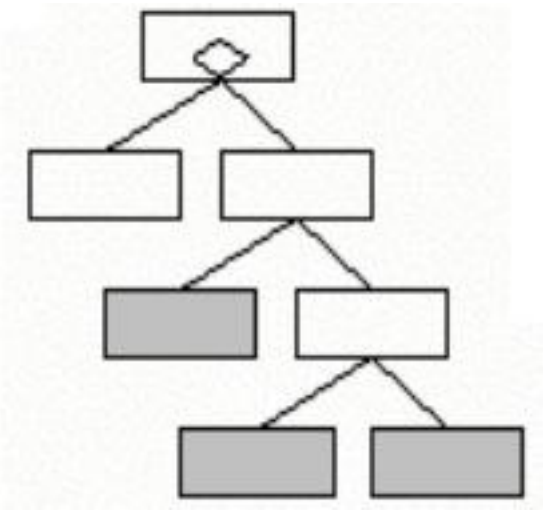


图5-28 模块示意图1

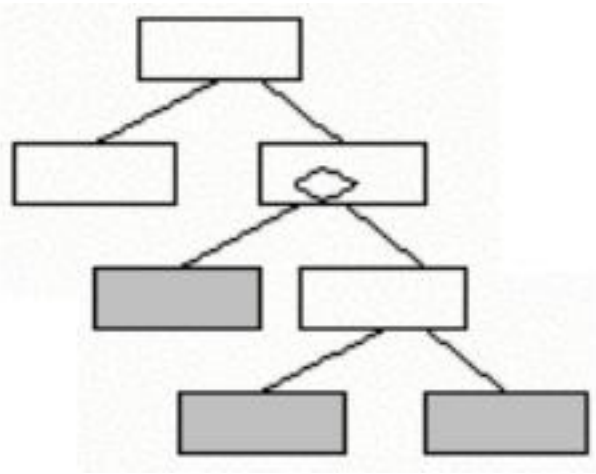


图5-29 模块示意图2

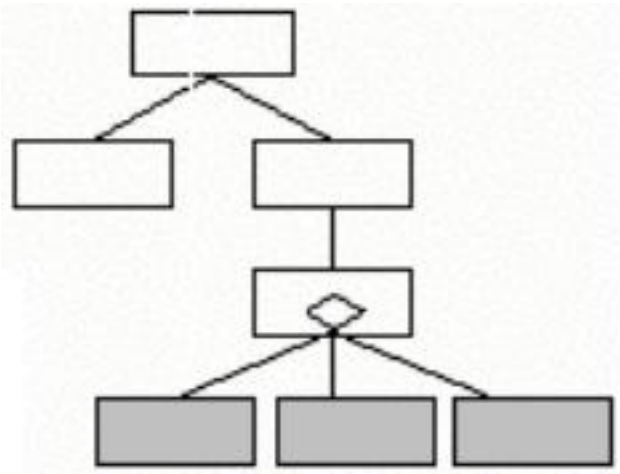


图5-30 模块示意图3

5.6 结构化设计

1. 变换型数据流图

根据信息系统的模型，信息一般是以外部形式进入系统，通过系统处理后，然后离开系统。从其过程可以得出，变换流的数据流图是一个线性结构。

变换型的数据流是由输入、变换（或称处理）和输出三部分组成 。

5.6.1 数据流的类型

变换型数据流图

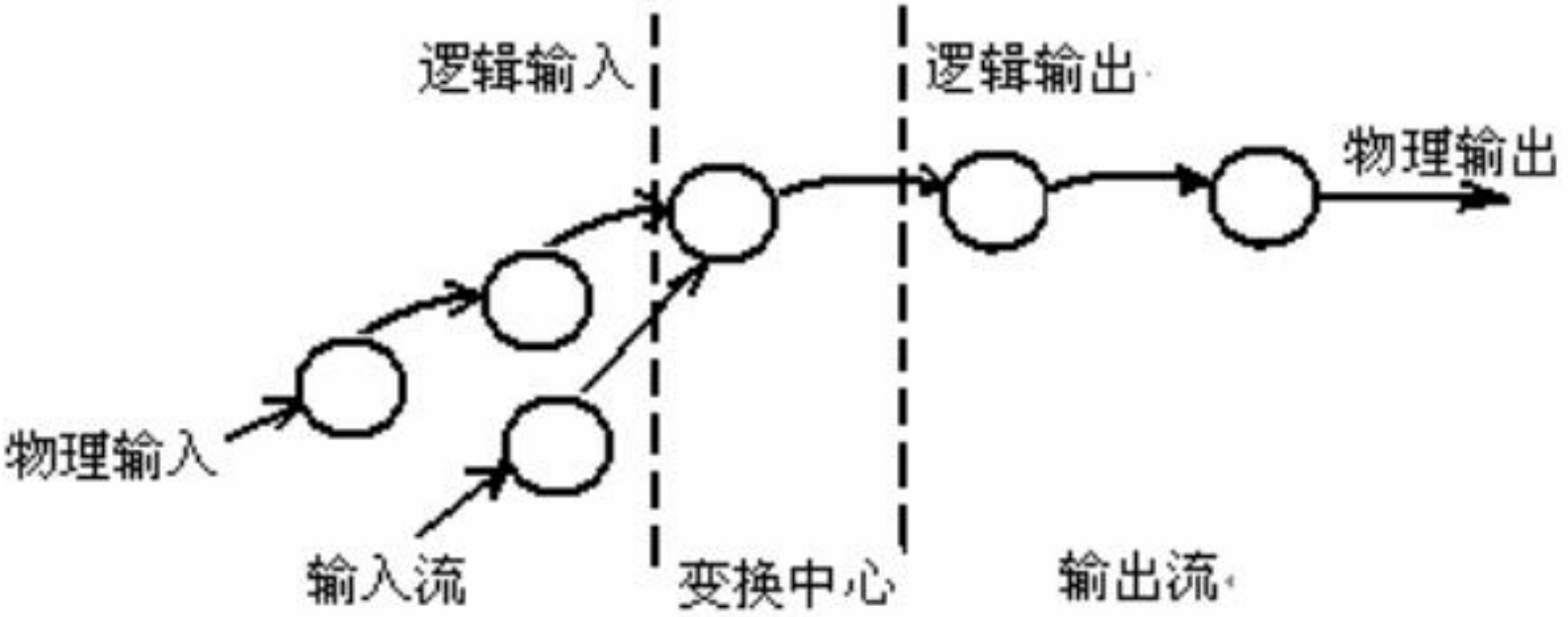


图5-31　变换型数据流图

2. 事务型数据流图

基本系统模型意味着变换流。因此，原则上可以讲所有的信息流都可以归结为这一类。

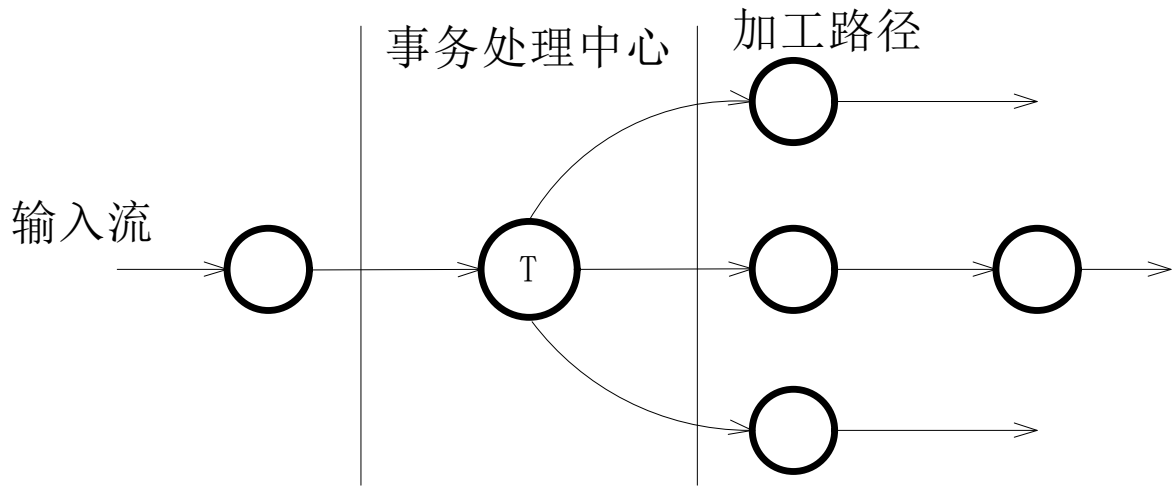


图5-32　事务型数据流图

图中的处理T称为事务中心，它完成下述任务：

（1）接收输入数据。

（2）分析每个事务，确定其类型。

（3）根据事务选择一条活动通路。

5.6.2 过程步骤

结构化设计方法转换的步骤：

（1）数据流图：把数据流图转换成软件结构图前，设计人员要参照规范说明书，仔细地研究分析数据流图并参照数据字典，认真理解其中的有关元素，检查有无遗漏或不合理之处，进行必要的修改。

（2）确定数据流图类型：通常将系统的数据流图视为变换流。

（3）找出变换中心：输入流是一条路径，经过这条路径数据从外部形式转换为内部形式。输出流则相反，从内部形式转换为外部形式。

（4）第一层分解：如果是变换流，要把数据流图映射为一种输入、变换、输出的特殊结构。

（5）第二层分解：这一步的任务是把数据流图中的各个变换映射成为相应的模块。

（6）根据优化准则对软件结构求精。

（7）描述模块功能、接口及全局数据结构。

（8）复查，如果有错，转步骤（2）修改完善，否则进入详细设计。

5.6.3 变换分析设计

变换流的设计是将数据流图到程序结构图的转换。当数据流图具有较明显的变换特征时，则按照下列步骤设计：

1. 确定数据流图中的变换中心、逻辑输入和逻辑输出。

2. 设计软件结构的顶层和第一层。

3. 设计中、下层模块。

（1）输入模块的下属模块的设计

（2）输出模块的下属模块的设计

（3）变换模块的下属模块的设计

4. 设计的优化

（1）输入部分的求精

（2）输出部分的求精

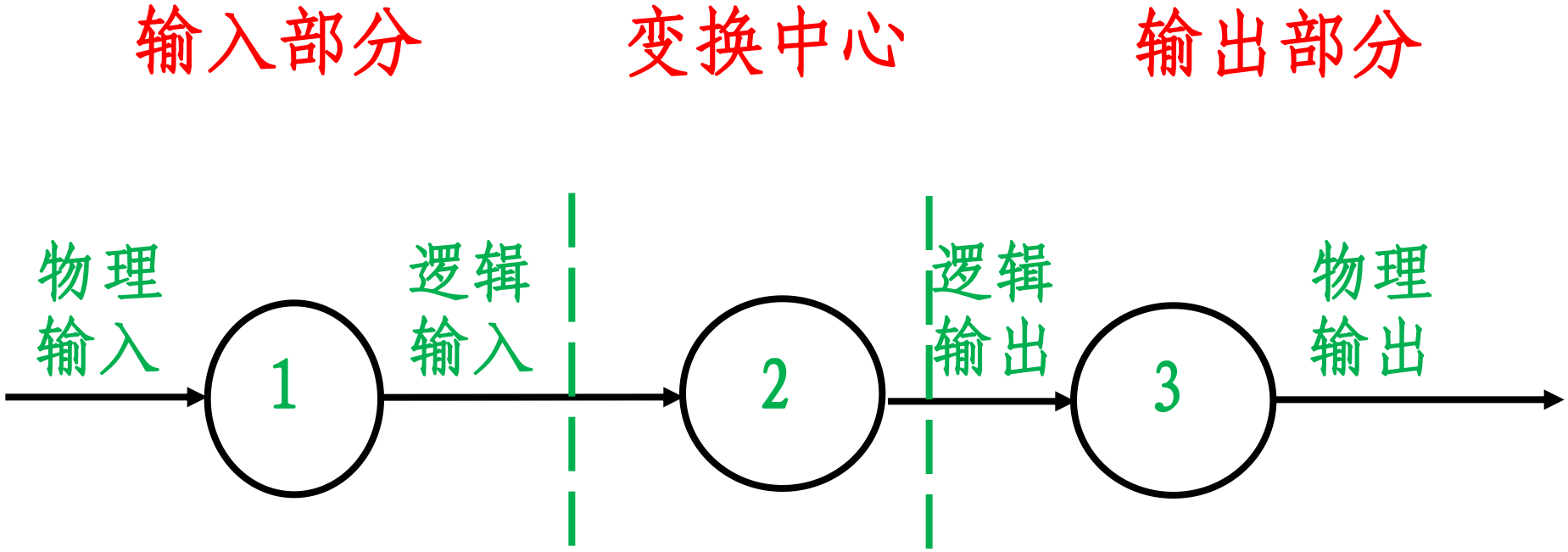
（3）变换部分的求精

变换分析方法：

第一步：

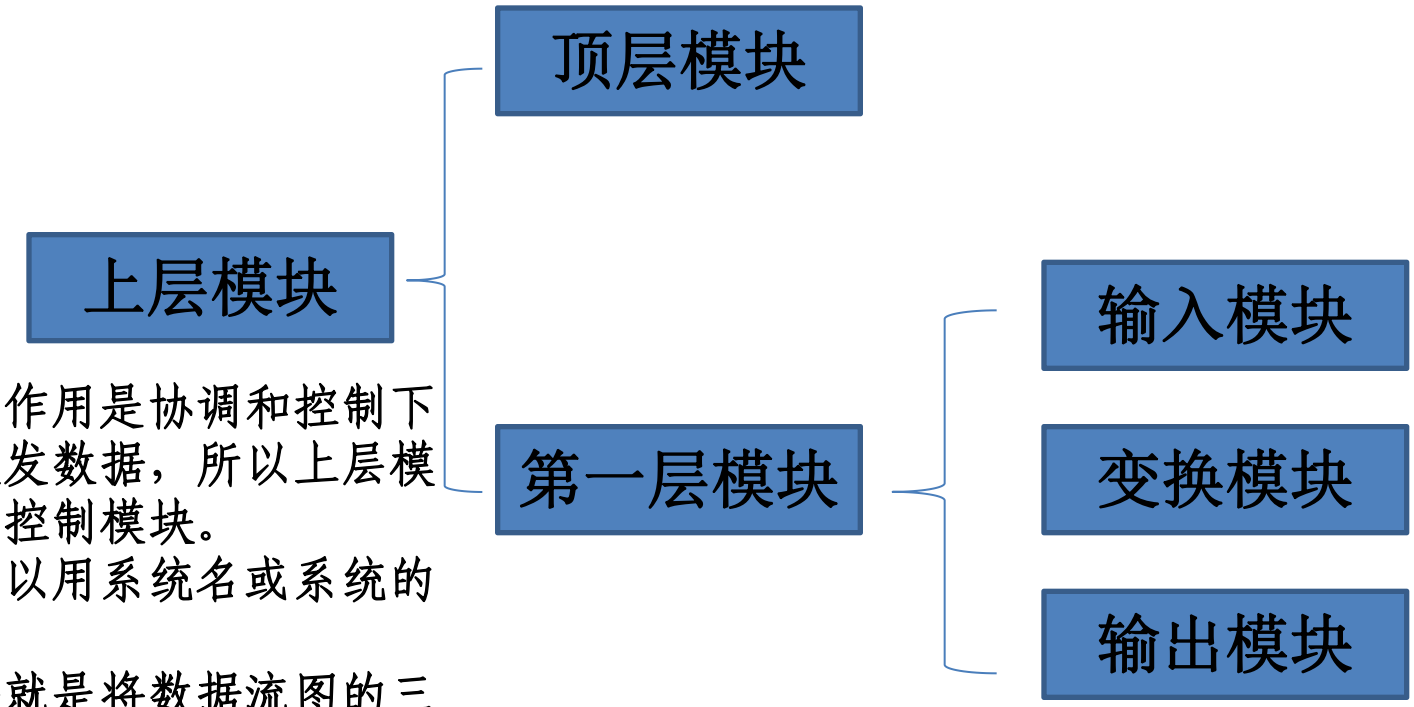
- 确定输入流和输出流的边界，孤立出变换中心。

- 确定边界通常凭经验，例如：
  - 数据流的汇合处往往是系统的主加工；
  - 找系统的逻辑输入和逻辑输出；
  - 看输入、输出量纲是否发生变化。


输入部分
变换中心
输出部分
物理输入
逻辑输入
逻辑输出
物理输出
1
2
3

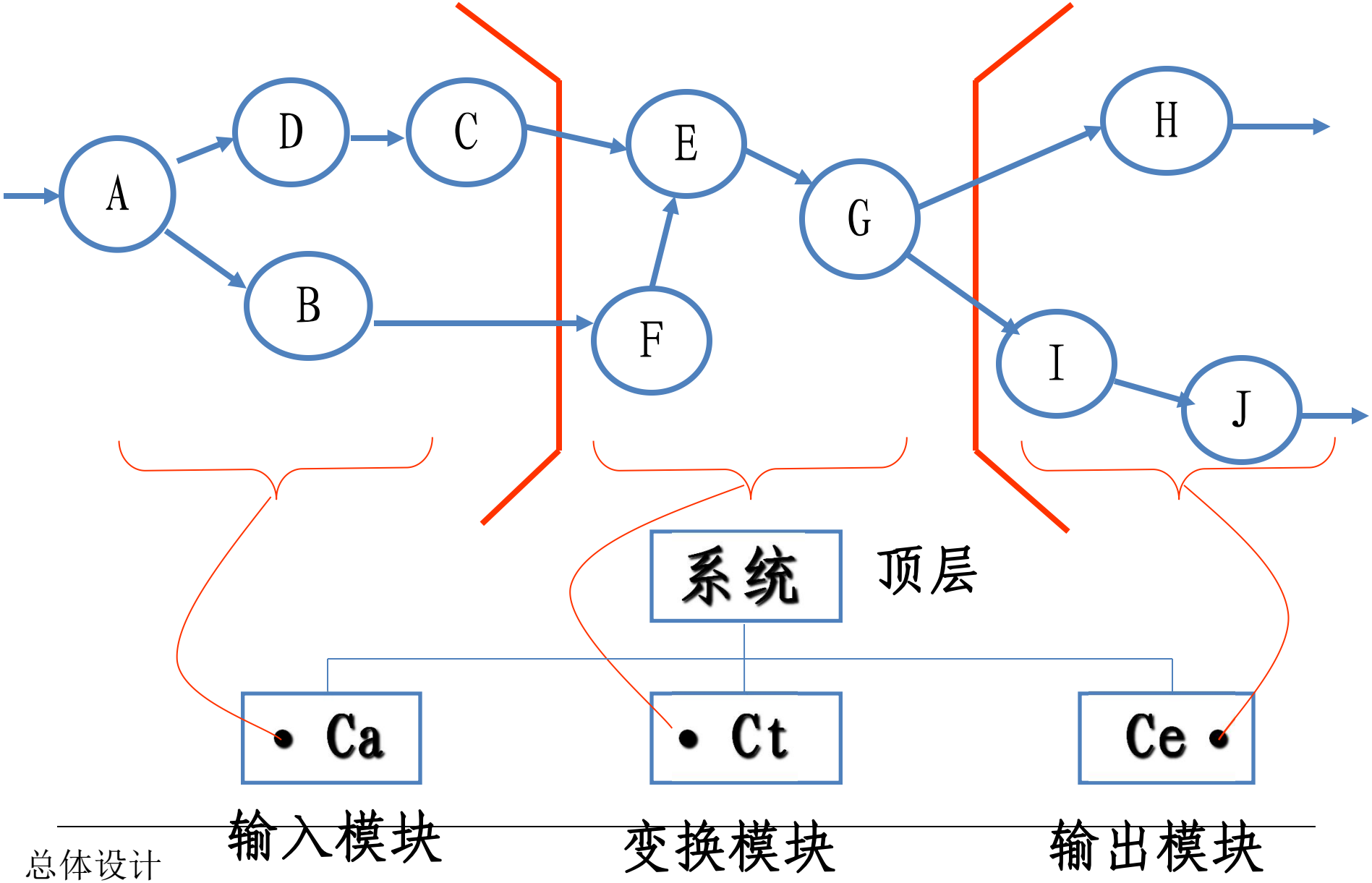
第二步：

- 设计软件结构的上层模块



上层模块的作用是协调和控制下层模块并收发数据，所以上层模块通常称为控制模块。
顶层模块可以用系统名或系统的功能来命名
第一层模块就是将数据流图的三个部分分别转换为三个模块：输入、变换、输出模块

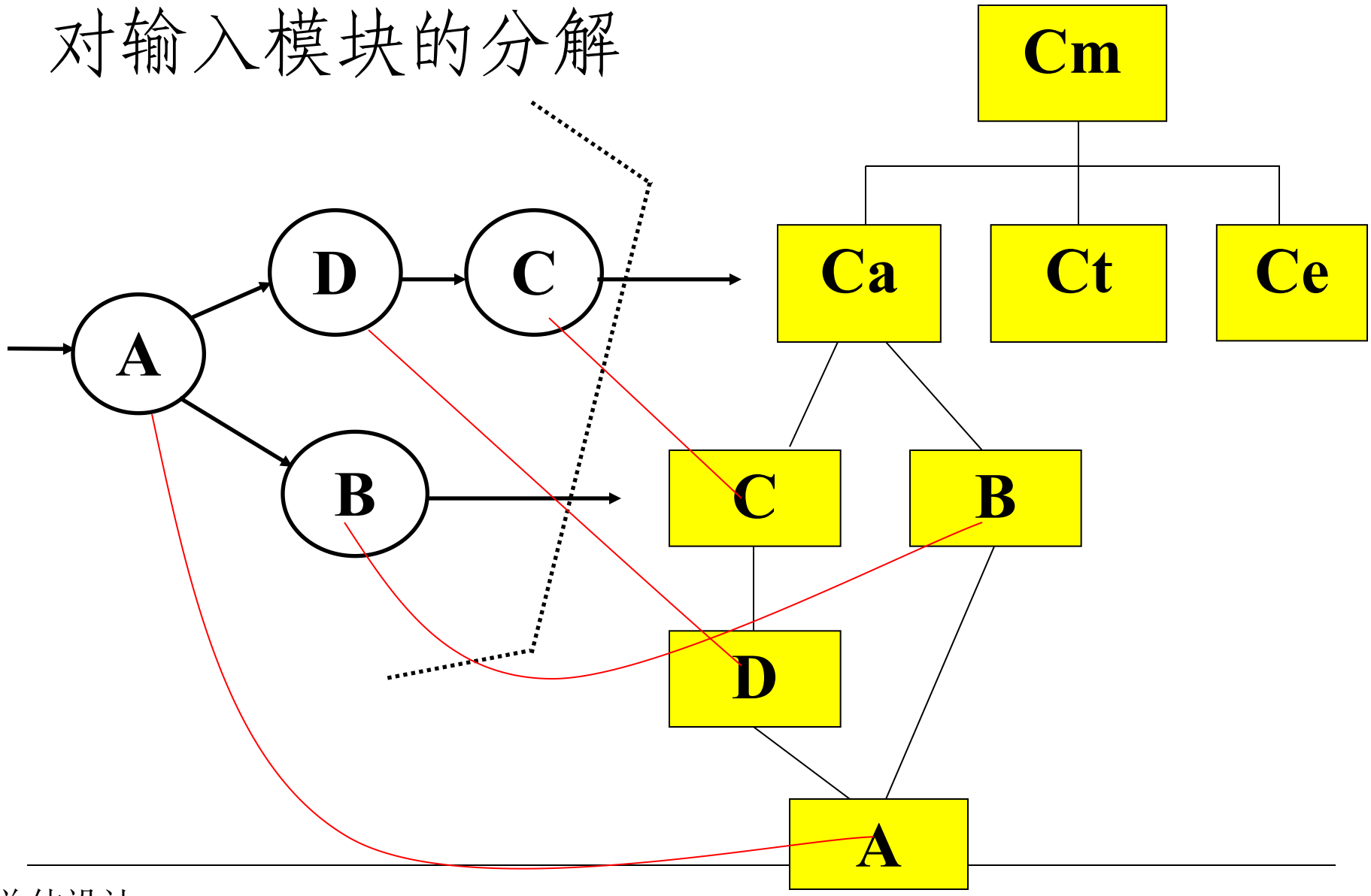
第一级分解法



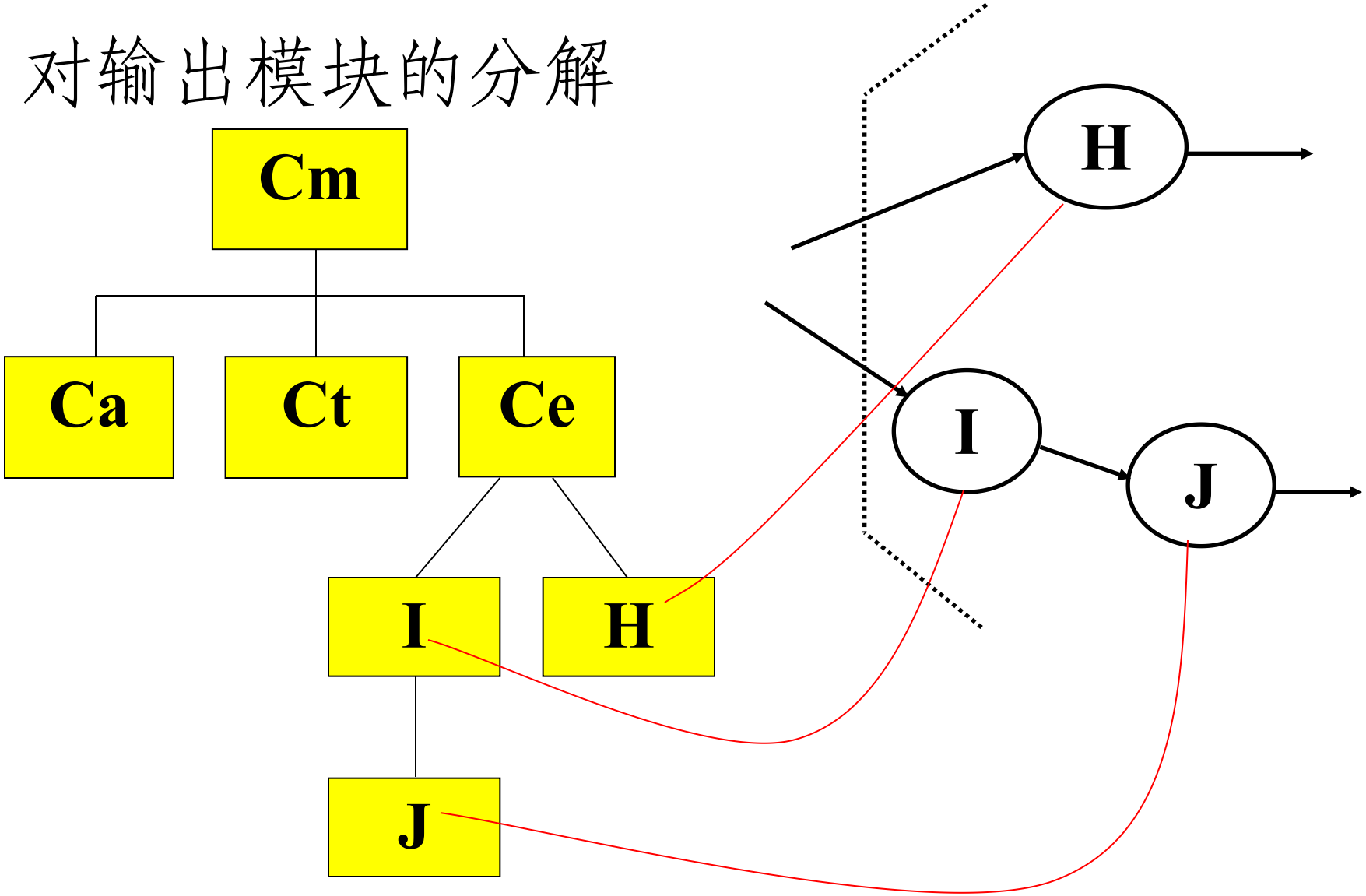
第三步：设计软件结构的下层模块

- 从输入边界开始，沿输入通路向外移动，把输入通路中每个处理映射成Ca下的一个低层模块。
- 从输出边界开始，沿着输出通路向外移动，把输出通路中的每一个模块映射成Ce下的一个低层模块。
- 把变换部分中每个处理，映射成Ct控制下的若干模块。

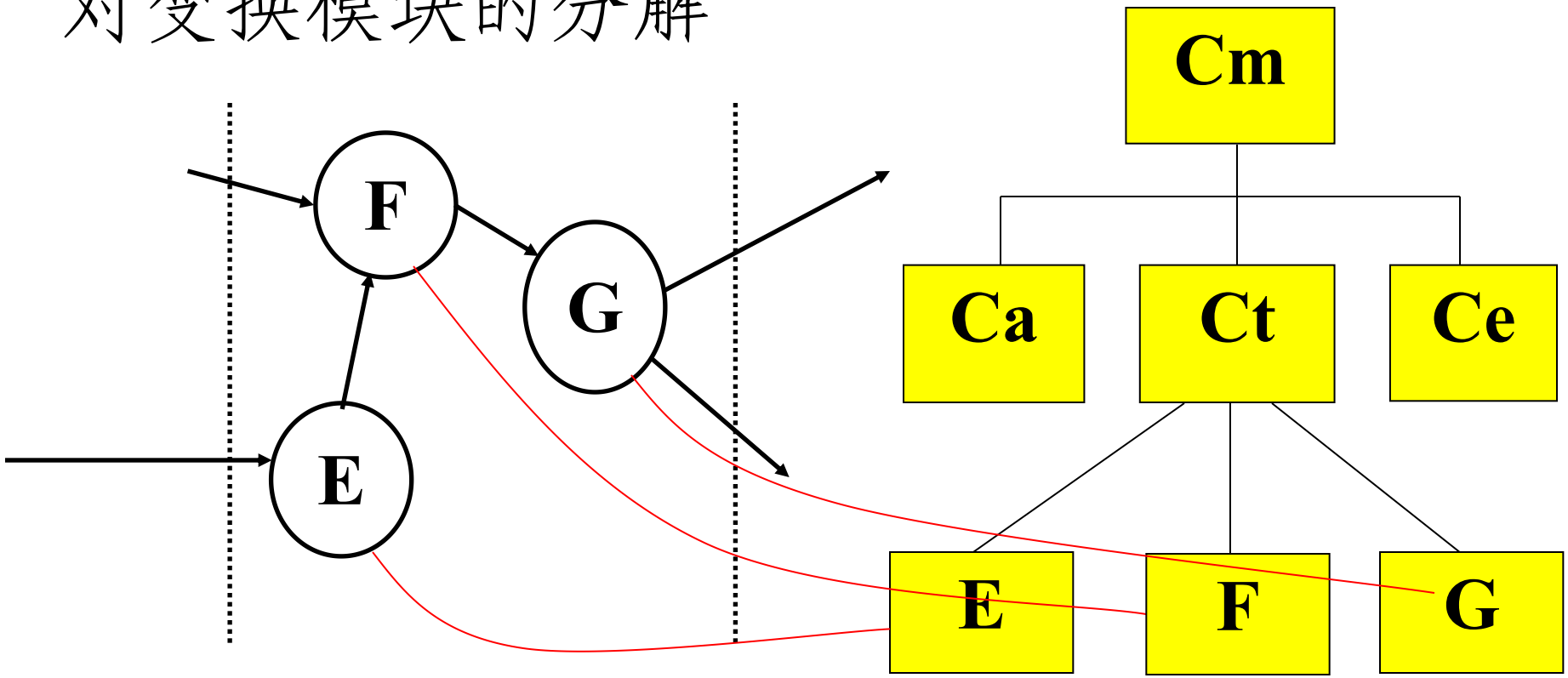
对输入模块的分解



对输出模块的分解



对变换模块的分解



第四步：

- 优化模块结构

5.6.4 事务分析设计

事务流的设计是从事务数据流图到程序结构的变换。对于具有事务型特征的数据流图，则采用事务分析的设计方法。

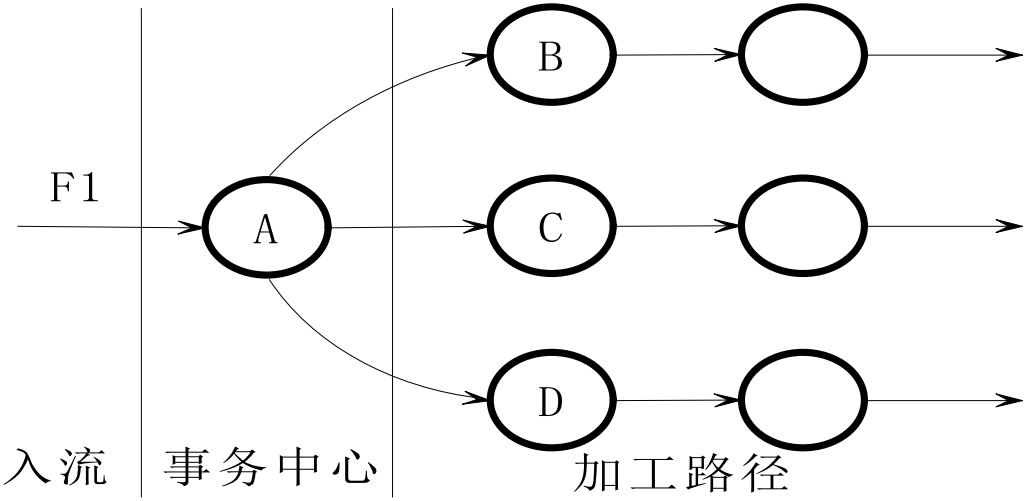
（1）确定数据流图中的事务中心和加工路径。

当数据流图中的某个加工具有明显地将一个输入数据流分解成多个发散的输出数据流时，该加工就是事务中心。从事务中心辐射出去的数据流为各个加工路径。

（2）设计软件结构的顶层和第一层。

事务处理中心和事务处理路径确定后，就可以确定它们的软件结构。

（3）进行事务结构中、下层模块的设计、优化等工作。



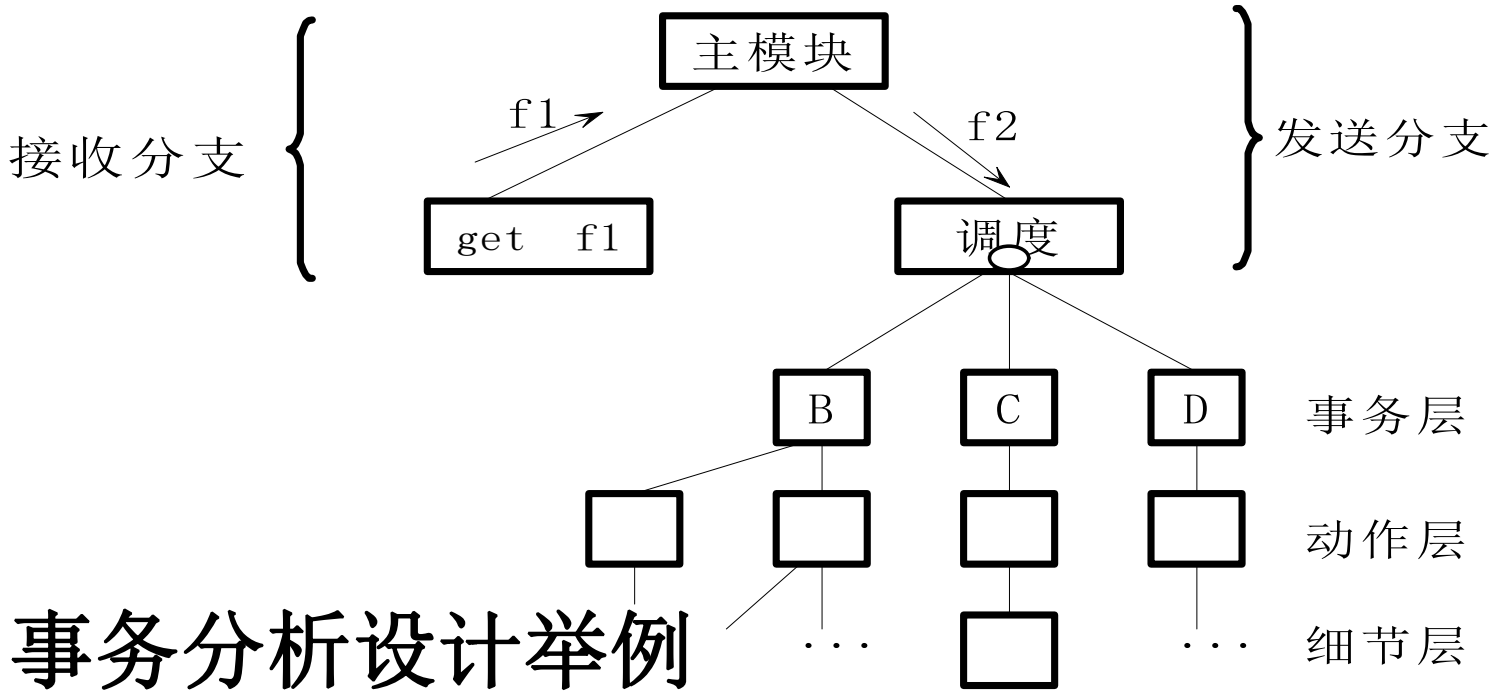


图5-34 事务分析设计举例

5.6.5 混合流设计

1. 混合数据流图的映射

一般中型以上的系统的数据流图中，都会既有变换流，又有事务流。这就是所谓的混合的数据流图，其软件结构设计方法一般采用以变换流为主，事务流为辅的方法。

（1）确定数据流图整体上的类型。

（2）标出局部的数据流图范围，确定其类型。

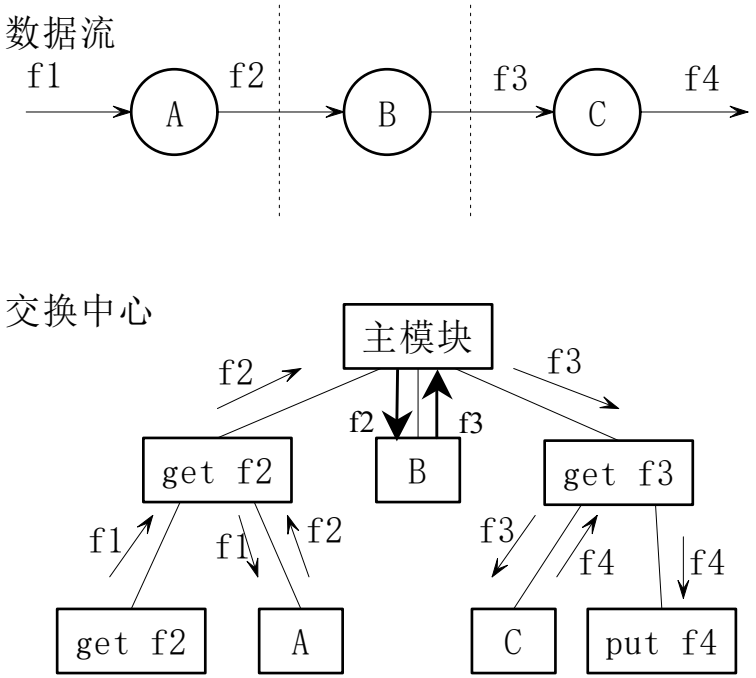
（3）按整体和局部的数据流图特征，设计出软件结构。

2. 分层数据流图的映射

对于一个复杂问题的数据流图结果，往往是分层的。那么对于分层的数据流图映射成软件结构图也应该是分层的。

（1）主图是变换型，子图是事务型

主图：



子图A：

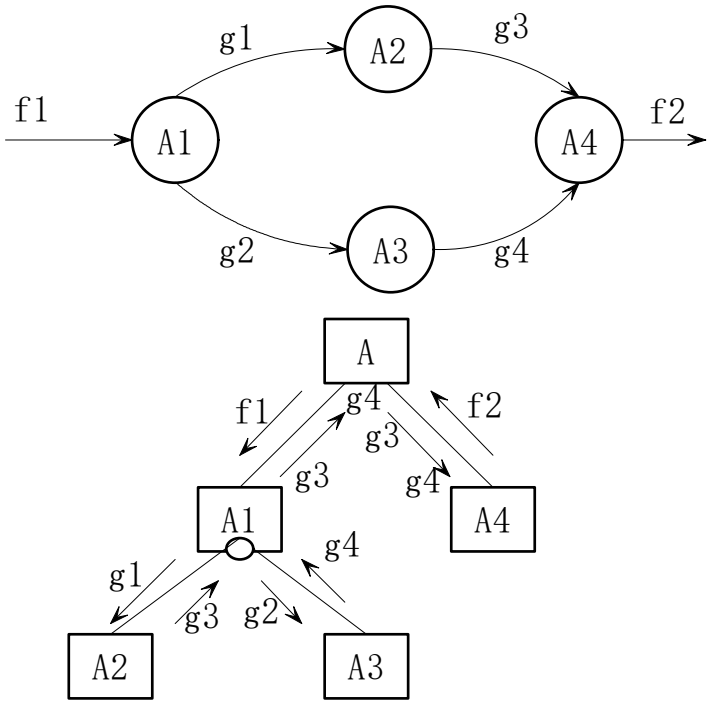


图5-35 主图变换型，子图事务型

（2）主图是事务型，子图是变换型

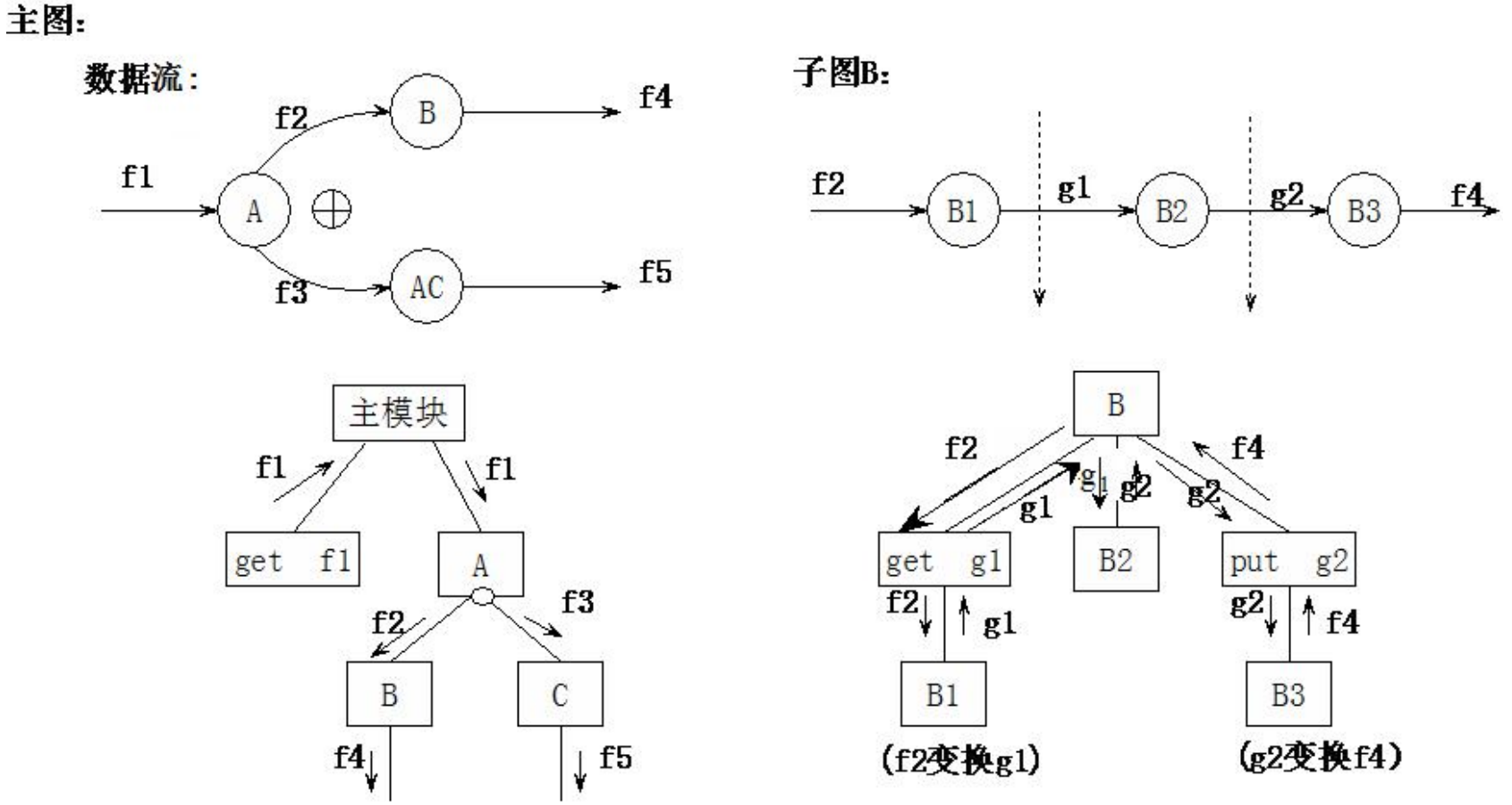


图5-36 主图事务型，子图变换型

5.6.6 结构化设计方法应用示例

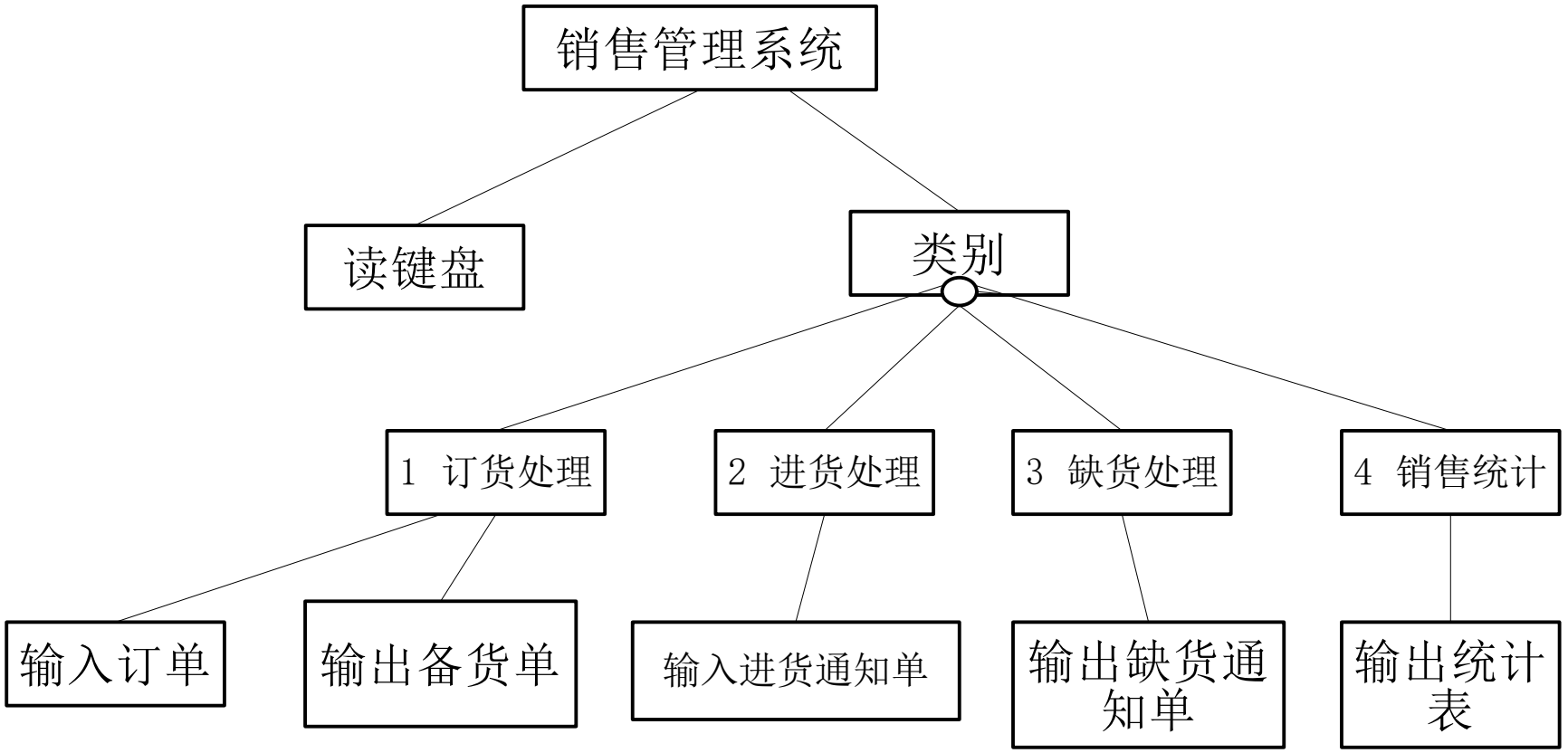


图5-37 销售管理系统软件结构图

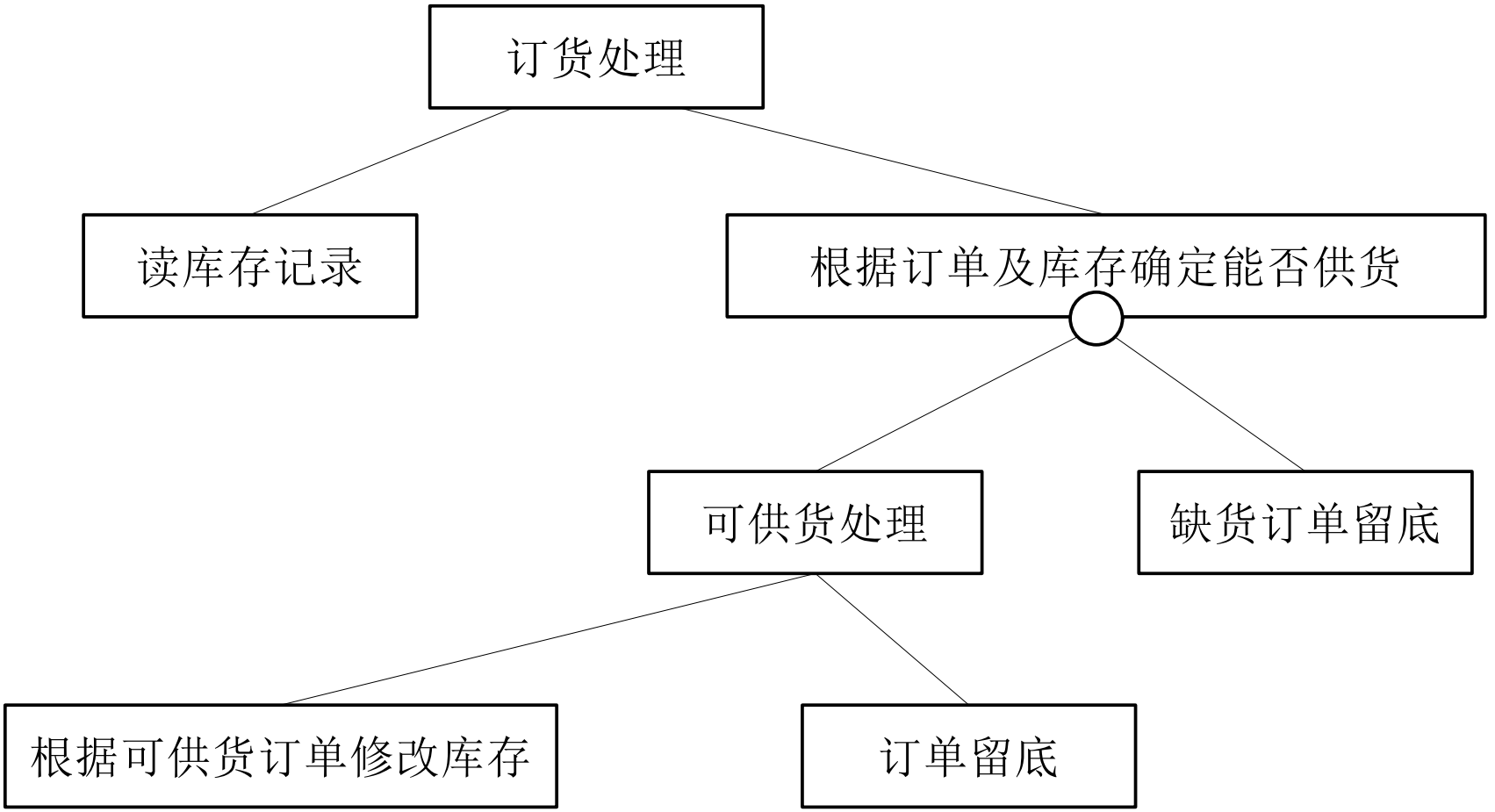
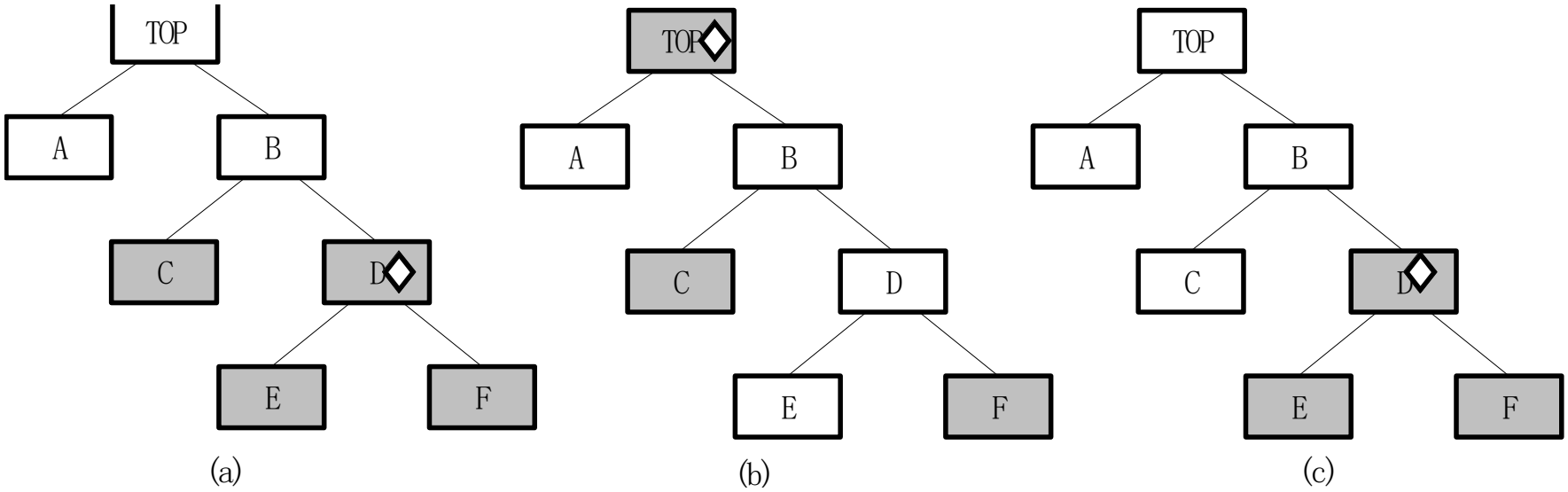


图5-38 子图1

5.6.7 设计的后期处理

由设计的工作流程可知，经过变换分析或事务分析设计，形成软件结构，对于这个结构除了设计文档作为系统设计的补充外，还要对结构进行优化和改进。

（1）为每个模块提供一份接口说明
（2）为每个模块写一份处理说明
（3）给出设计约束或限制
（4）数据结构说明
（5）进行设计评审
（6）设计优化



(a)差的结构图　　(b)不理想的结构图　　(c)理想的结构图

图5-42 模块的判定作用范围

5.7总体设计应用实例

5.7.3总体设计

1、需求规定

1)对功能的规定：

2) 对性能的规定：

3) 其他专门要求

2、运行环境：

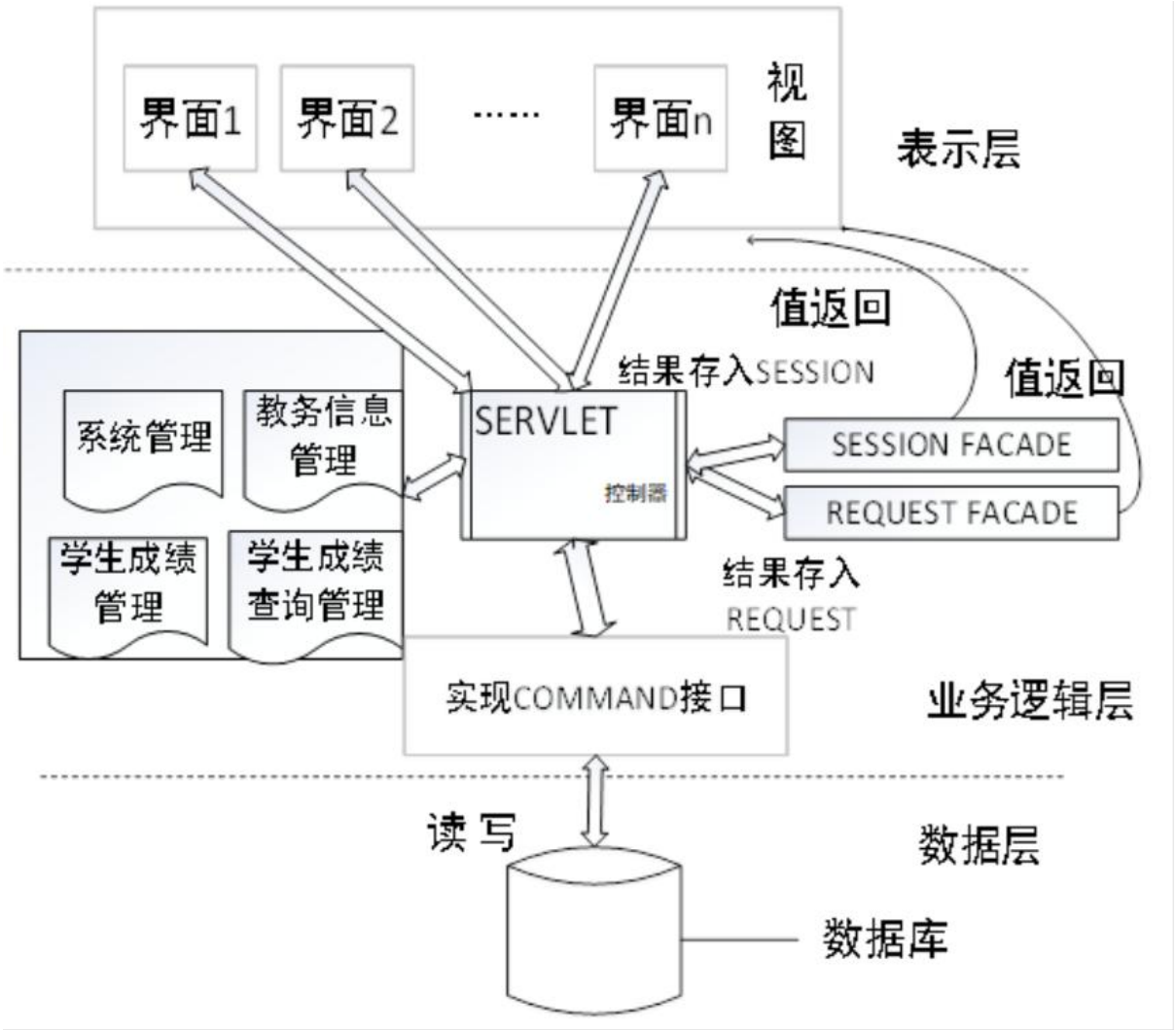
1）硬件环境

2)软件环境。

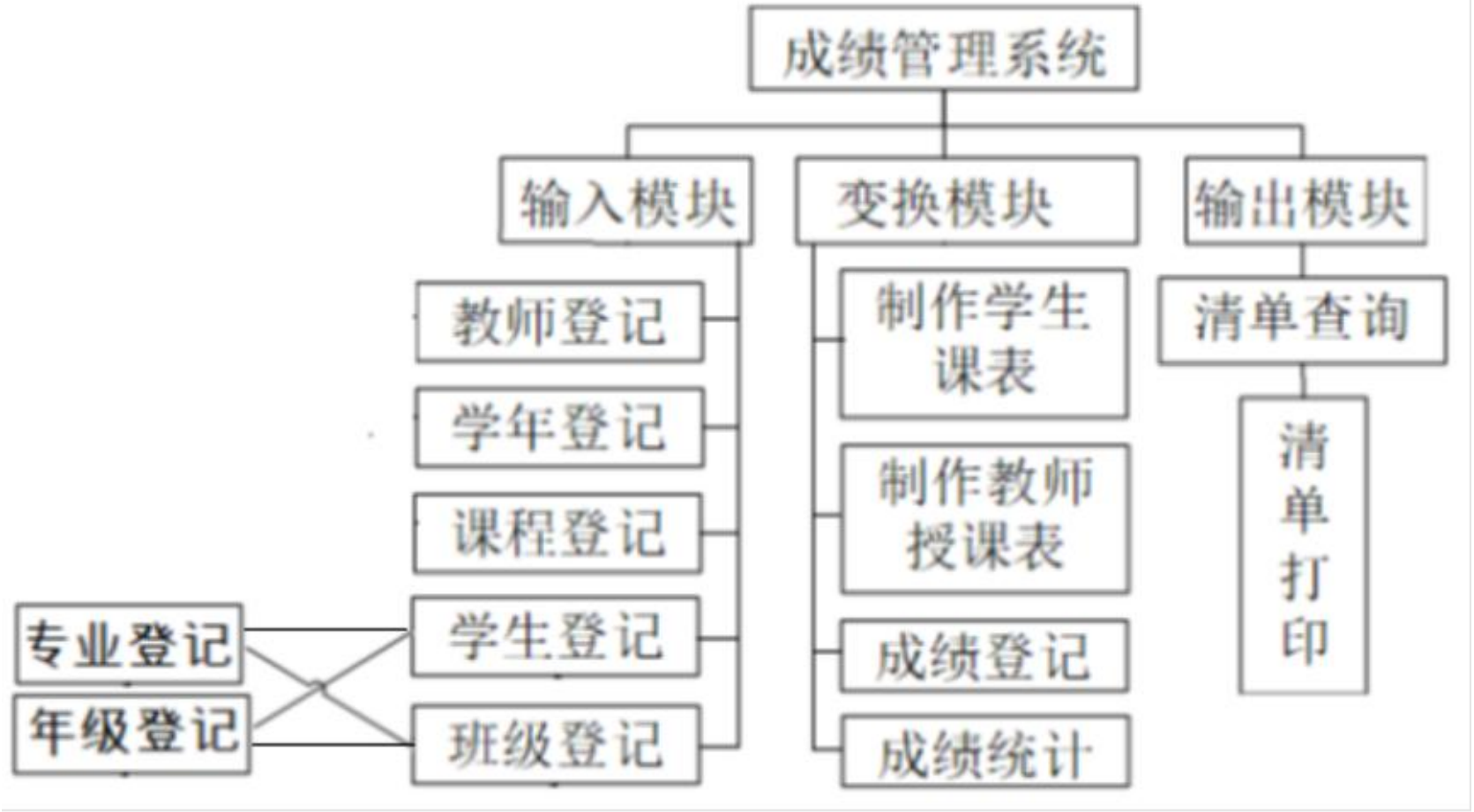
3、基本设计概念和处理流程

4、 结构

1）系统体系结构


界面1
界面2
……
界面n
视图
表示层
值返回
结果存入SESSION
值返回
系统管理
教务信息管理
SERVLET
控制器
SESSION FACADE
REQUEST FACADE
学生成绩管理
学生成绩查询管理
结果存入REQUEST
实现COMMAND接口
业务逻辑层
读 写
数据层
数据库

2系统结构图


成绩管理系统
输入模块
变换模块
输出模块
教师登记
学年登记
课程登记
学生登记
班级登记
专业登记
年级登记
制作学生课表
制作教师授课表
成绩登记
成绩统计
清单查询
清单打印

3功能结构图

(选讲内容)
5.7.4接口设计
5.7.5运行设计
5.7.6系统论结构设计
1、逻辑结构设计
2 物理结构设计要点
5.7.7故障检测与处理机制

谢谢