

第5章（结构化）总体设计

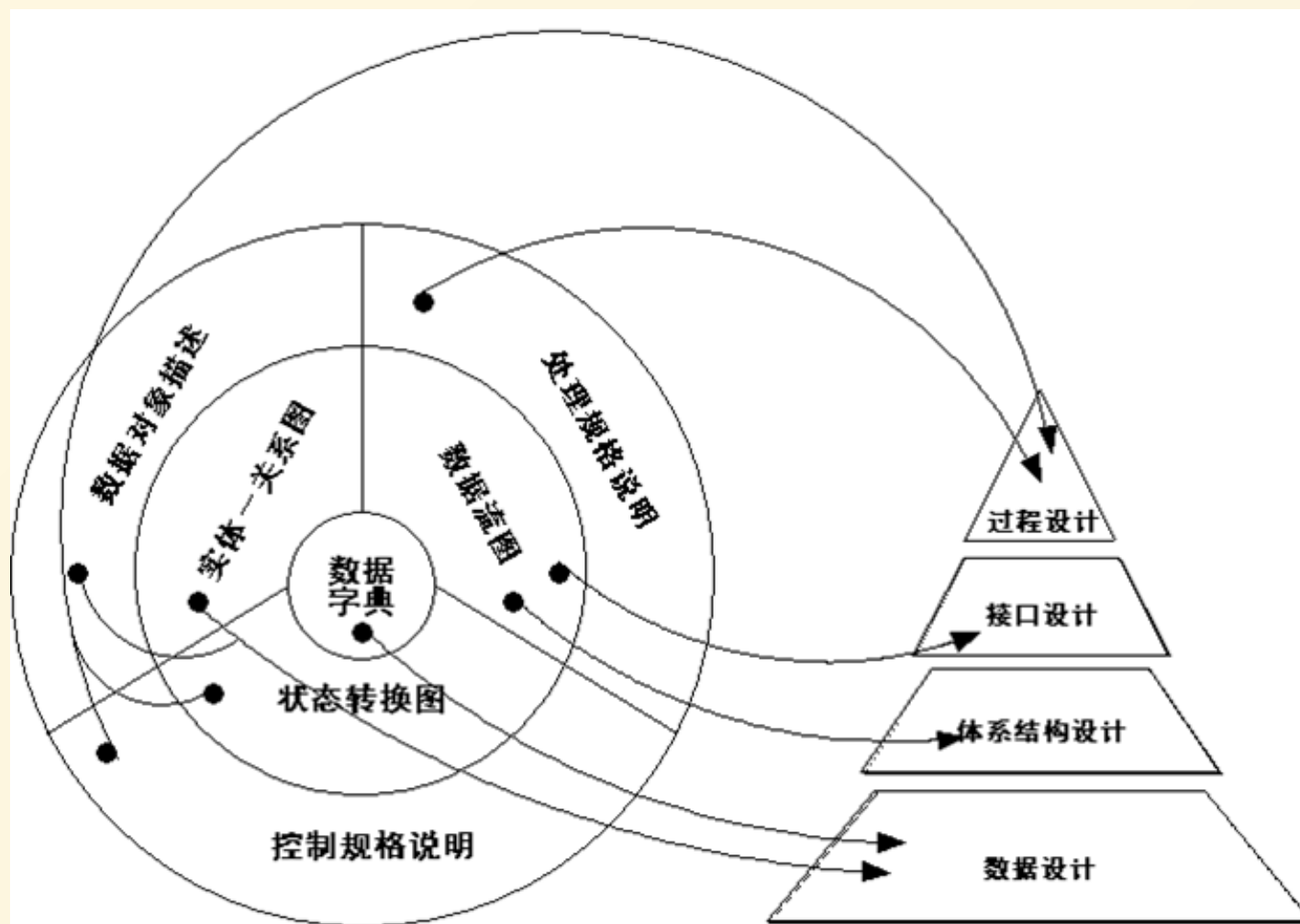
目录

- 5.1 设计过程
- 5.2 设计原理
- 5.3 启发规则
- 5.4 描绘软件结构的图形工具
- 5.5 面向数据流的设计方法

结构化设计

- 传统软件工程方法学采用结构化设计技术（SD）。
- 从工程管理角度结构化设计分两步：
 - 总体设计（概要设计）：将软件需求转化为数据结构和软件系统结构。
 - 详细设计（过程设计）：通过对结构细化，得到软件详细数据结构和算法。

结构化设计与分析的关系



5.1（总体）设计过程

1. 设想供选择的方案
2. 选择合理的方案
3. 推荐最佳方案
4. 功能分解
5. 设计软件结构
6. 设计数据库
7. 制定测试计划
8. 书写文档
9. 审查和复审

5.2 设计原理

- 模块化
- 抽象
- 逐步求精
- 信息隐藏和局部化
- 模块独立

模块化

如果一个大型程序仅由一个模块组成，很难被人理解。

设函数 $C(x)$ 定义问题 x 的复杂程度，函数 $E(x)$ 定义解决问题 x 需要的工作量（时间）。对于两个问题 P_1 和 P_2 ，

如果 $C(P_1) > C(P_2)$,

那么 $E(P_1) > E(P_2)$

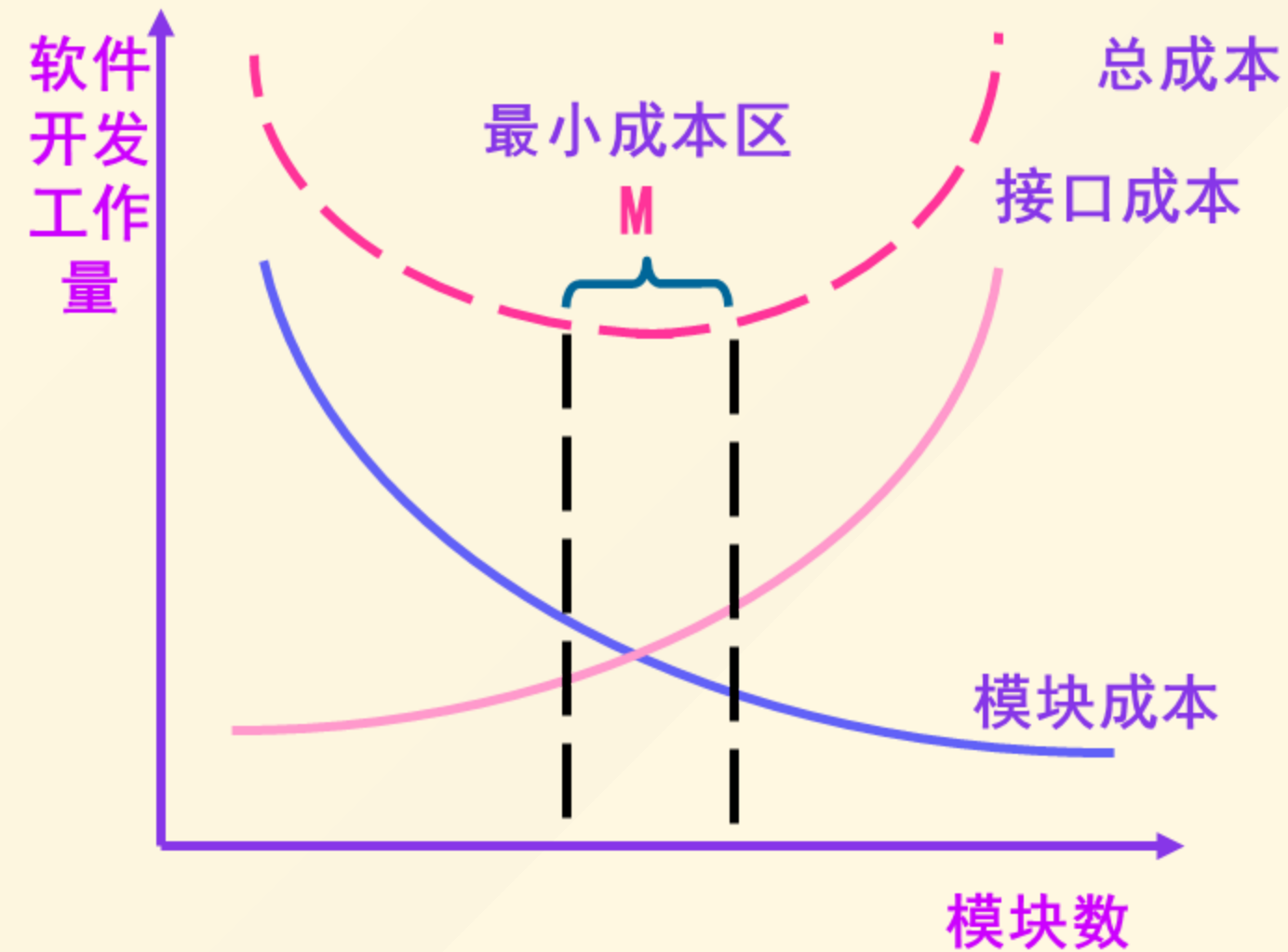
根据解决问题的经验，有一个规律是：

$$C(P_1 + P_2) > C(P_1) + C(P_2)$$

于是有

$$E(P_1 + P_2) > E(P_1) + E(P_2)$$

模块化与软件成本



抽象

抽出事物的本质特性，暂不考虑细节。

In a famous paper, ***The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information*** (1956), **George Armitage Miller** proposed as a law of human cognition and information processing that **humans can effectively process no more than seven units, or chunks, of information, plus or minus two pieces of information, at any given time.**

逐步求精

- 求精是指为了能集中精力解决主要问题，尽量推迟对细节问题的考虑，实际上是一个细化过程，与抽象是互补的概念。
- 抽象使得设计者能够说明过程和数据，同时却忽略底层细节；
- 求精帮助设计者在设计过程中揭示底层细节。

信息隐藏和局部化

- 每个模块的实现细节对于其他模块来说是隐藏的。也就是说，模块中所包含的信息是不允许其他不需要这些信息的模块访问的。
- 每个客户只能通过接口来了解该模块，而所有的实现都隐蔽起来。

模块独立

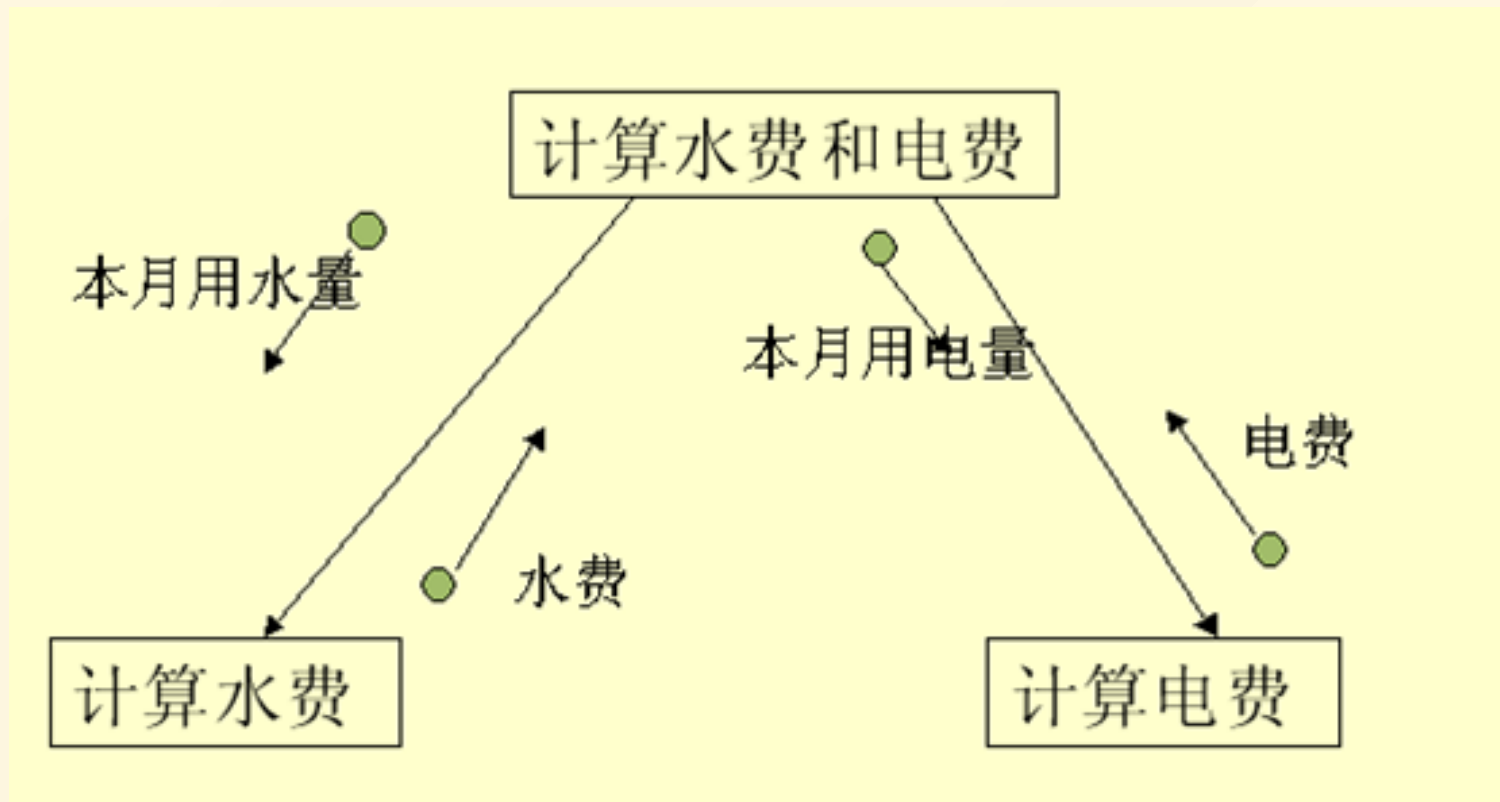
- 模块独立：模块具有独立功能且和其他模块没有过多的相互作用。
- 模块独立的好处
 - 容易开发：分工合作方便。
 - 容易测试和维护：修改工作量较小，错误传播范围小，扩充功能容易。
- 两个定性度量标准
 - 耦合（coupling）
 - 内聚（cohesion）

耦合

- 耦合：指软件结构内不同模块彼此之间相互依赖（连接）的紧密程度。
- 耦合的种类
 - 数据耦合
 - 控制耦合
 - 公共环境耦合
 - 内容耦合

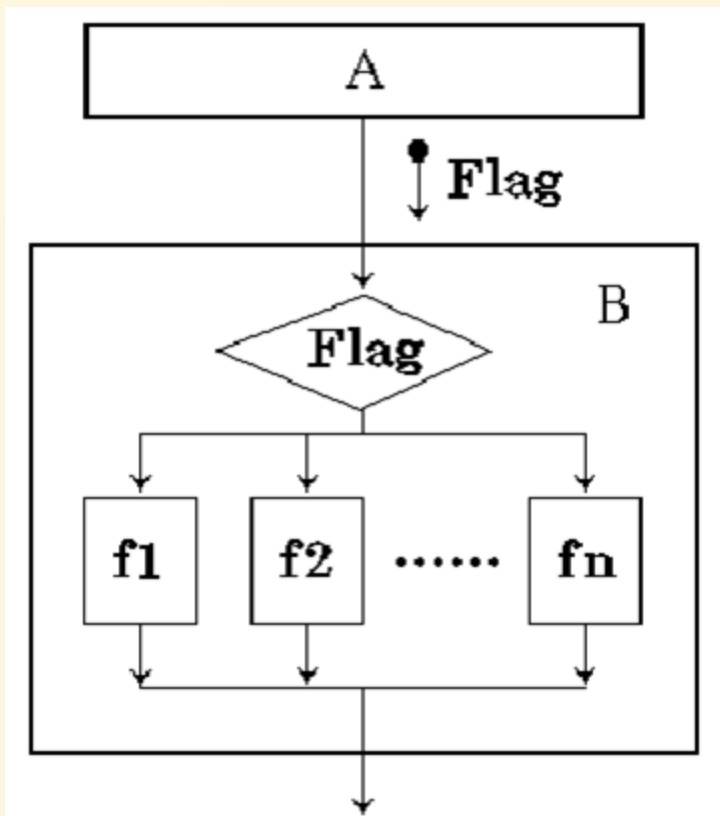
数据耦合

- 两个模块之间只是通过参数交换信息，而且交换的信息仅仅是数据。
- 数据耦合是最低程度的耦合。



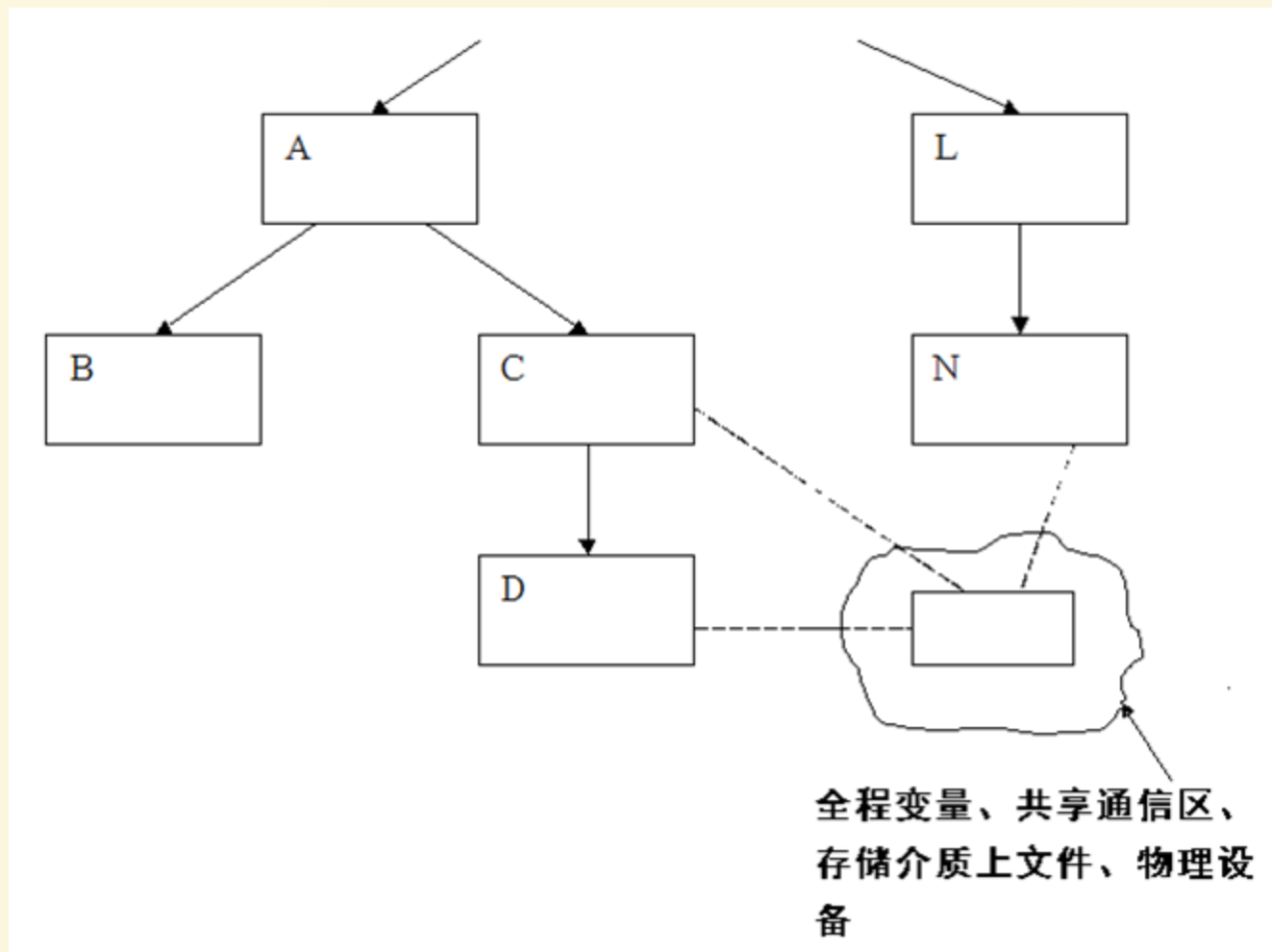
控制耦合

- 两个模块之间所交换的信息包含控制信息。
- 控制耦合是中等程度的耦合。



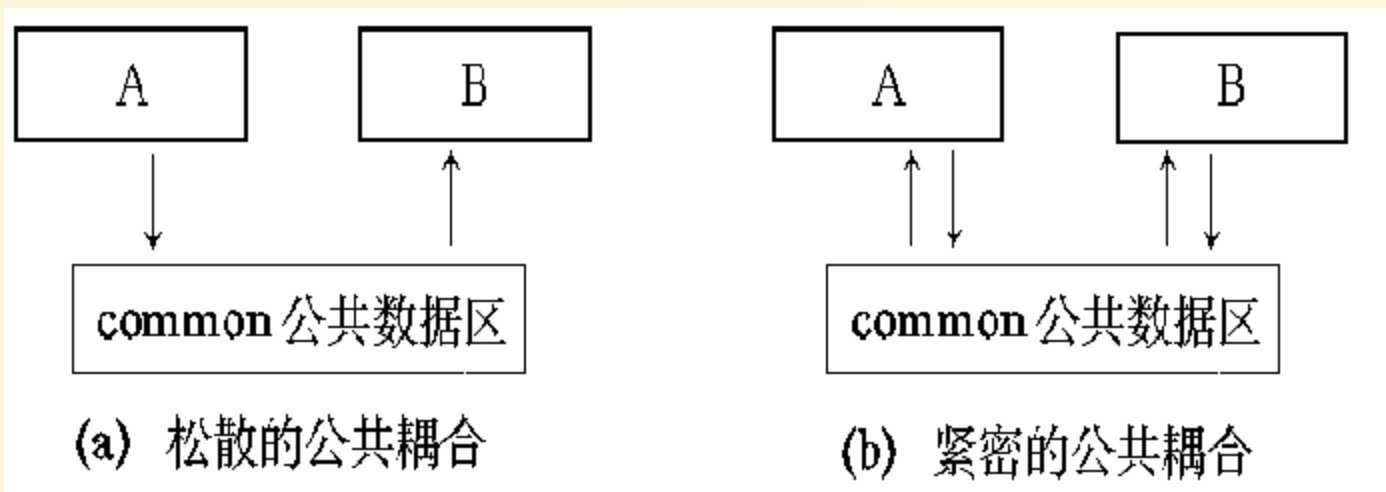
公共环境耦合

- 两个或多个模块通过一公共数据环境相互作用。



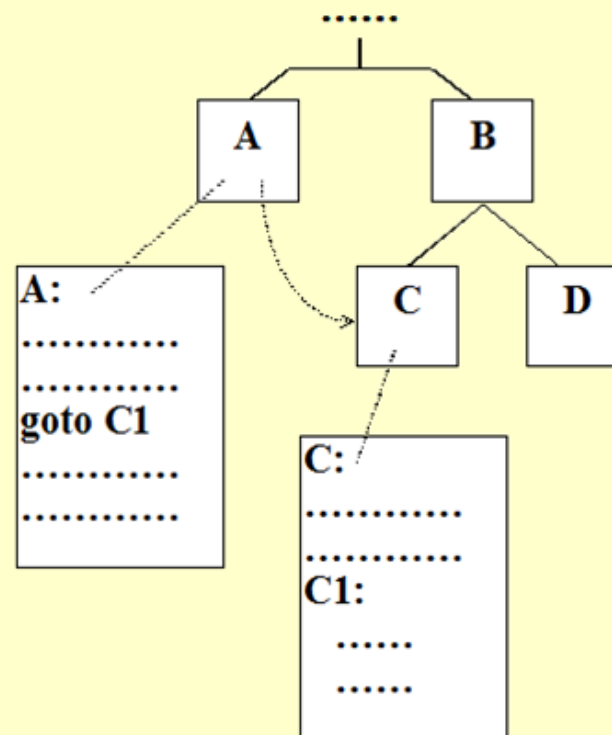
公共环境耦合的复杂程度

- 公共环境耦合的复杂程度随耦合的模块个数而变化。如果只有两个模块有公共环境，那么存在两种可能：
 - 一模块送数据，另一模块取，等价数据耦合。
 - 两模块既往公共环境送又从里面取，介于数据耦合和控制耦合之间。

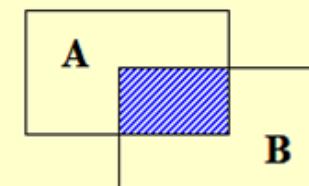


内容耦合

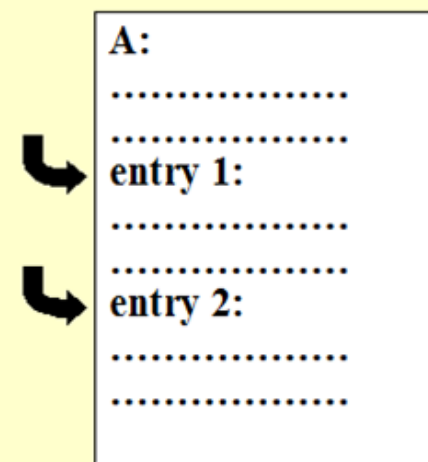
- 内容耦合是最高程度的耦合。
 - 一模块访问另一模块内部数据
 - 一模块不通过正常入口转到另一模块内部
 - 两模块有部分程序代码重叠（汇编程序）
 - 一模块有多个入口



不通过正常入口
进入程序内部



部分代码重叠



一个模块有多个入口

耦合性原则

软件设计应追求尽可能松散耦合，避免强耦合，这样模块间的联系就越小，模块的独立性就越强，对模块的测试、维护就越容易。

因此建议：**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。**

内聚

- 内聚：一个模块内部各个元素彼此结合的紧密程度。它是衡量一个模块内部组成部分间整体统一性的度量。
- 内聚的种类
 - 功能内聚
 - 顺序内聚
 - 通信内聚
 - 过程内聚
 - 时间内聚
 - 逻辑内聚
 - 偶然内聚

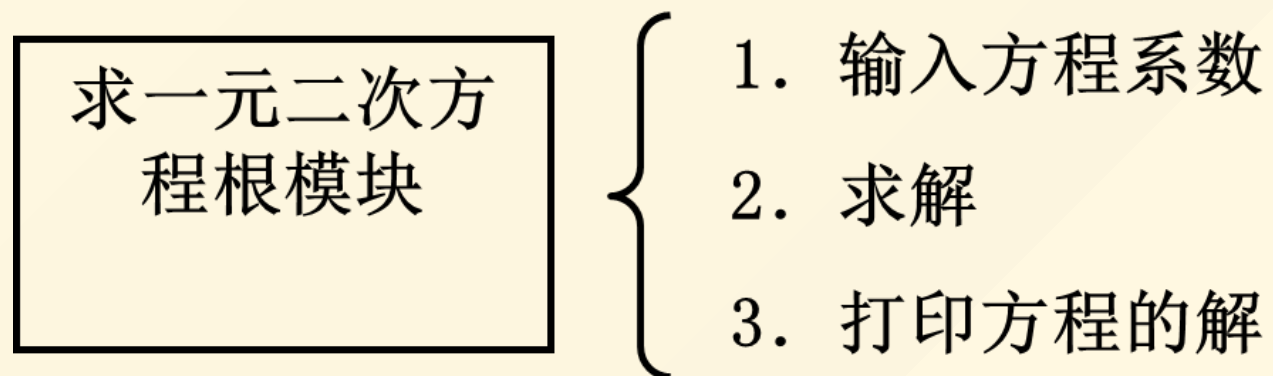
功能内聚

如果一个模块内所有处理元素完成一个，而且仅完成一个功能，则称为功能内聚。

功能内聚是最高程度的内聚。但在软件结构中，并不是每个模块都能设计成一个功能内聚模块。

顺序内聚

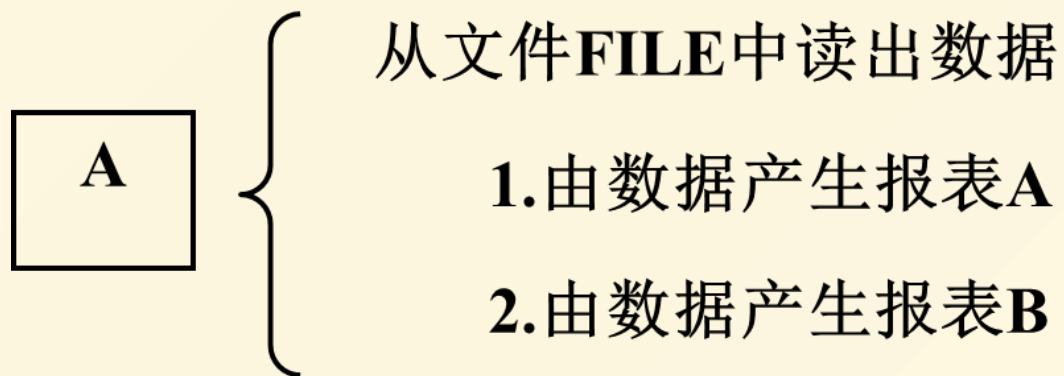
如果一个模块内处理元素和同一个功能密切相关，而且这些处理元素必须顺序执行，则称为顺序内聚。



顺序内聚示例

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。



通信内聚示例

过程内聚

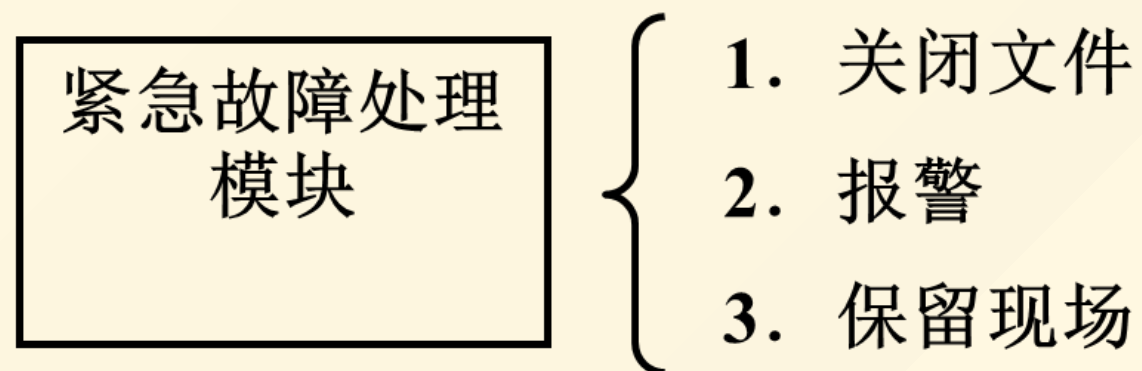
如果一个模块内的处理元素是相关的，而且必须以特定的次序执行，称为过程内聚。

过程内聚与顺序内聚的区别是：

顺序内聚是数据流从一个处理单元流到另一个处理单元，而过程内聚是控制流从一个动作流向另一个动作。

时间内聚

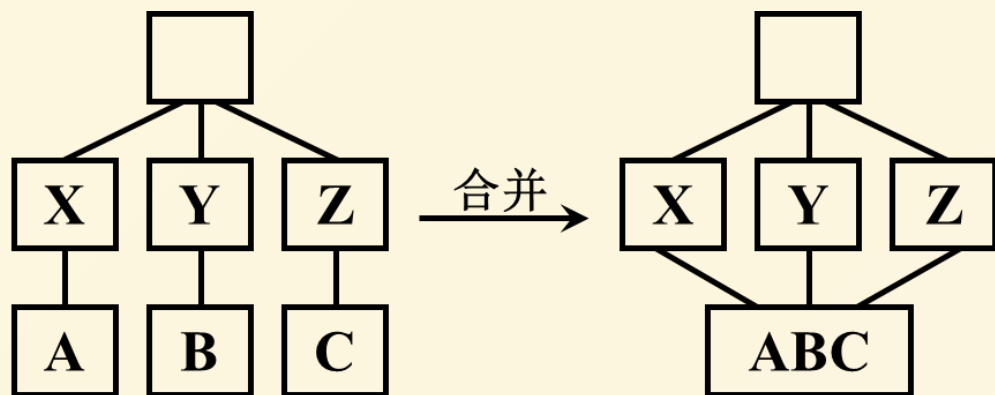
如果一个模块包含的任务必须在同一段时间内执行，称为时间内聚。也称为瞬时内聚。



时间内聚示例

逻辑内聚

如果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。



逻辑内聚示例

对逻辑内聚模块的调用，常常需要有一个功能开关，由上层调用模块向它发出一个控制信号，在多个关联性功能中选择执行某一个功能。这种内聚较差，增加了模块之间的联系，不易修改。

偶然内聚

如果一个模块由完成若干毫无关系的功能处理元素偶然组合在一起的，就叫偶然内聚。

偶然内聚是最差的一种内聚。

常犯这种错误的一种情况是：有时在写完程序后，发现一组语句在多处出现，于是为了节省空间而将这些语句作为一个模块设计，就出现偶然内聚。

内聚性原则

软件设计中应该：**力求做到高内聚，尽量少用中内聚，不用低内聚。**

5.3 启发规则

1. 改进软件结构提高模块独立性

通过模块分解或合并，降低耦合、提高内聚。

2. 模块规模应该适中

规模过大：可能分解不充分；规模过小，开销大，接口成本高。

3. 深度、宽度、扇出和扇入都应适当

4. 模块的作用域应该在控制域之内

5. 力争降低模块接口的复杂度

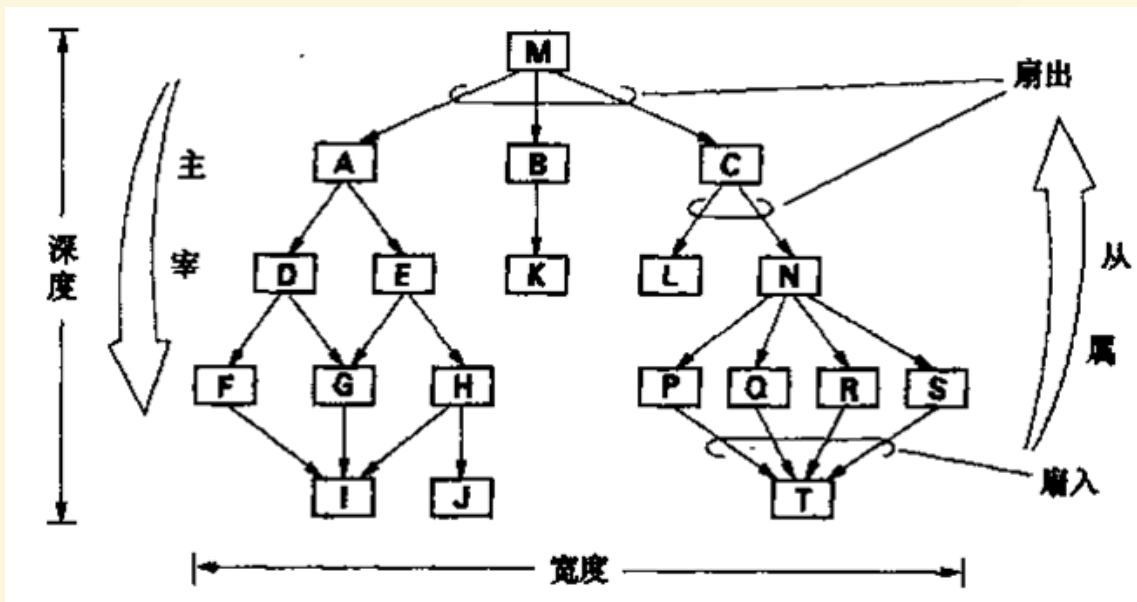
6. 设计单入口、单出口的模块

7. 模块功能应该可以预测

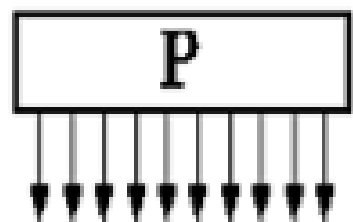
输入数据相同，产生同样输出。

深度、宽度、扇出和扇入都应适当

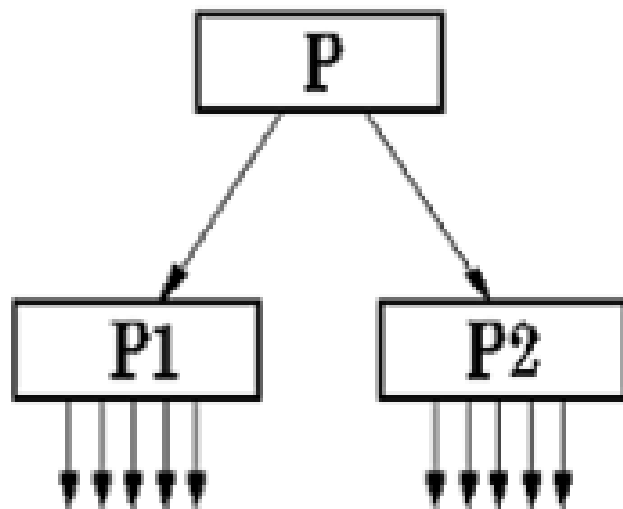
- 深度：软件结构中控制的层数；
- 宽度：软件结构内同一个层次上的模块总数的最大值；
- 扇出：一个模块直接控制（调用）其它模块的数目；
- 扇入：一个模块被其它模块调用的数目



对扇出过大的改进

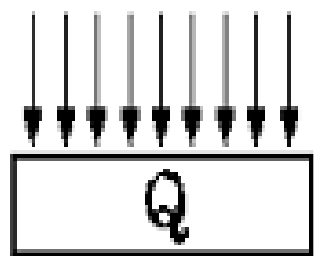


(a)

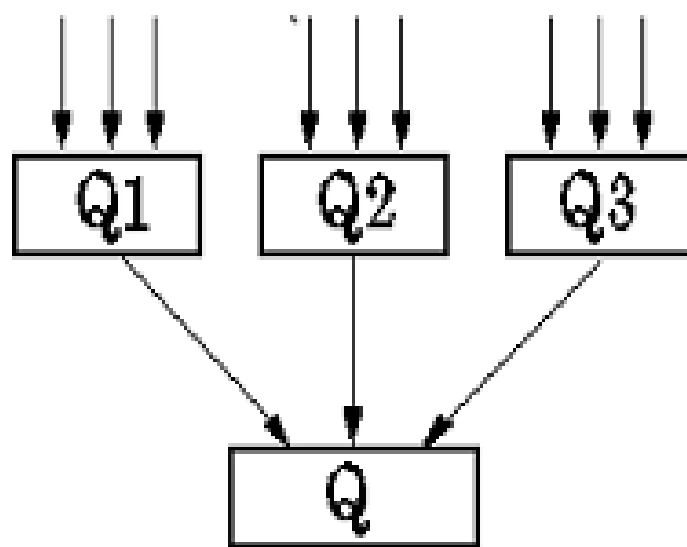


(b)

对扇入过大的改进



(c)



(d)

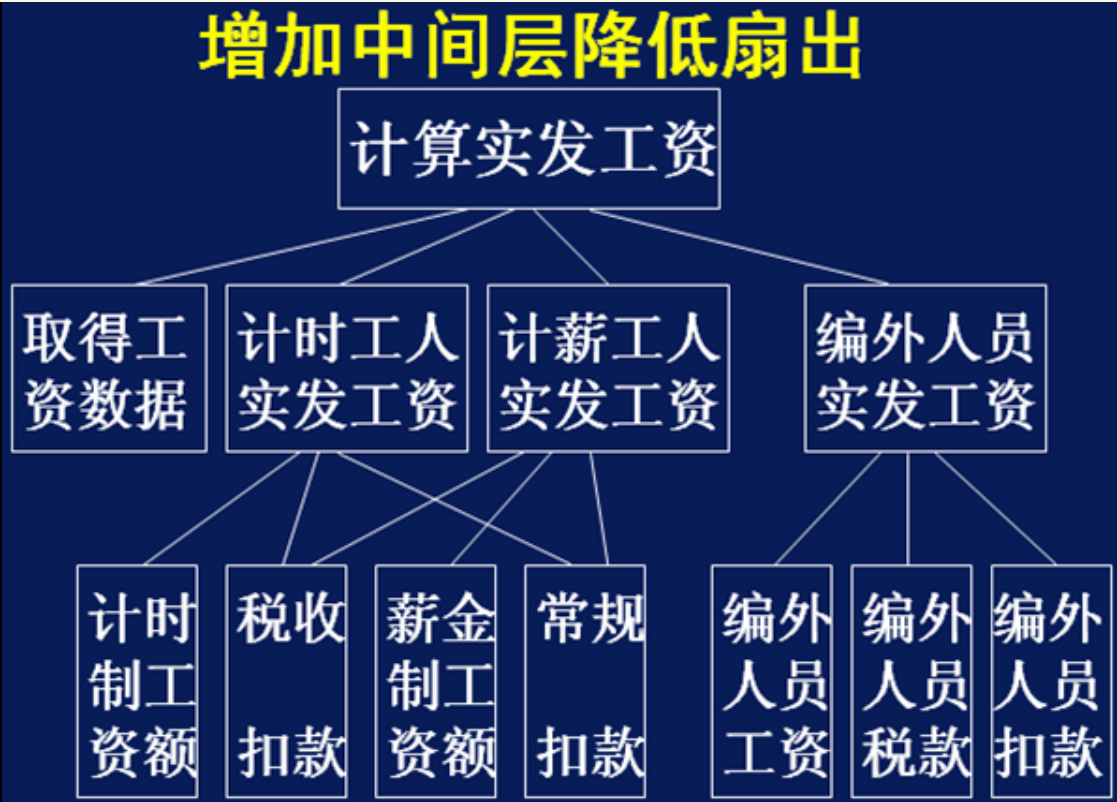
扇出改进实例：改进前

高扇出的模块结构举例：



避免平铺结构

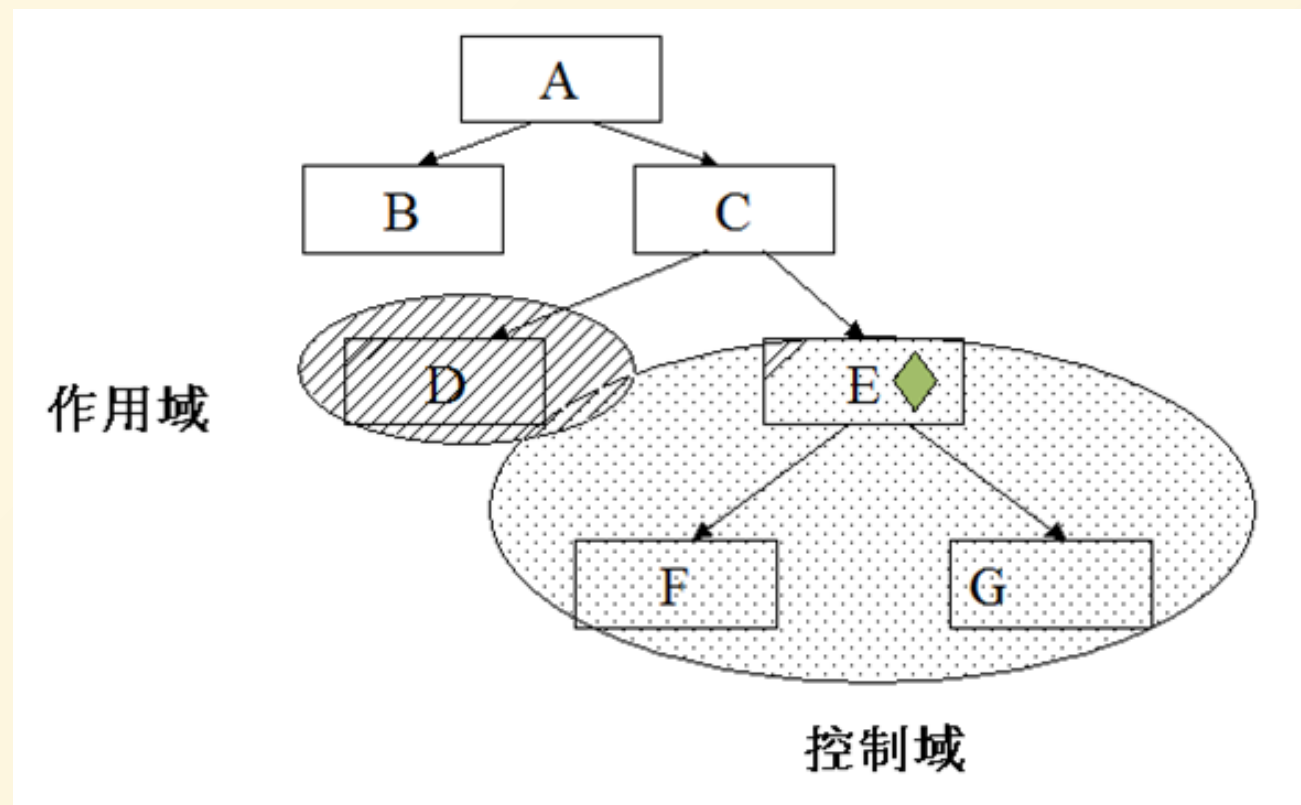
扇出改进实例：改进后



模块的作用域应该在控制域之内

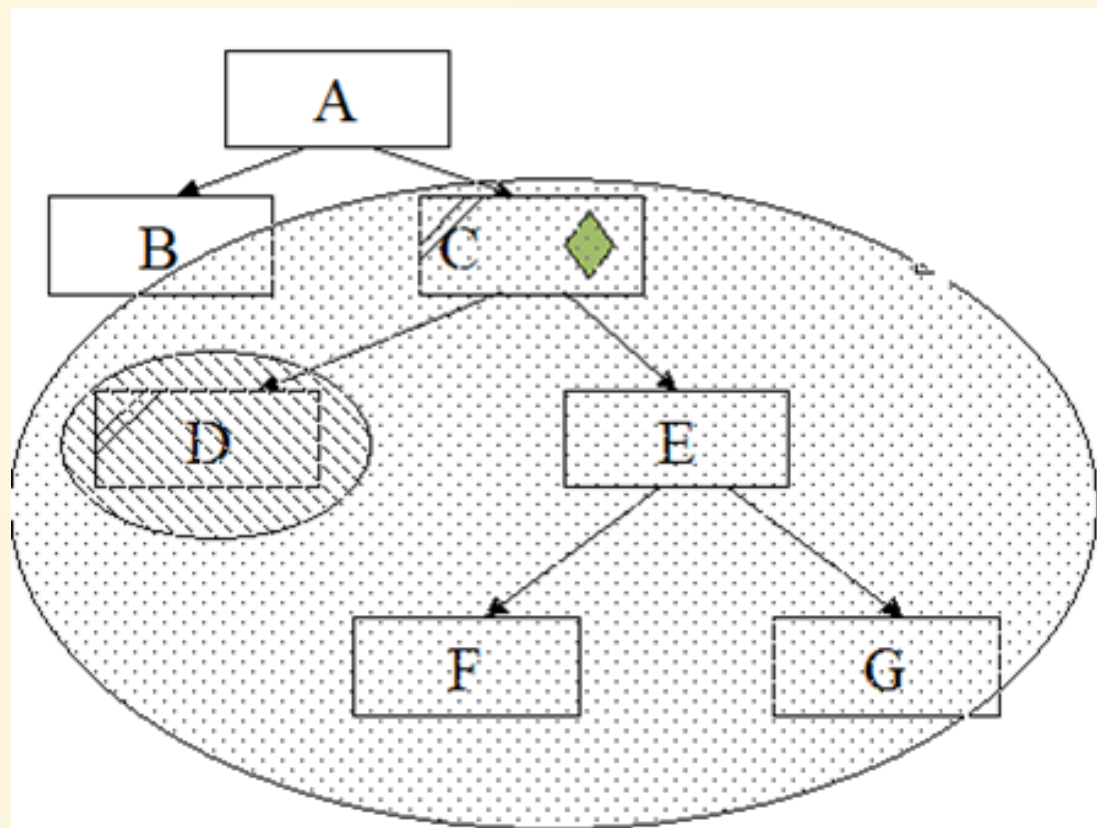
- 作用域：受该模块内判定影响的所有模块集合。
- 控制域：模块本身及所有直接或间接从属它的模块集合。
- 若模块作用域不在控制域内，会增大模块间控制耦合。

作用域不在控制域的软件结构图



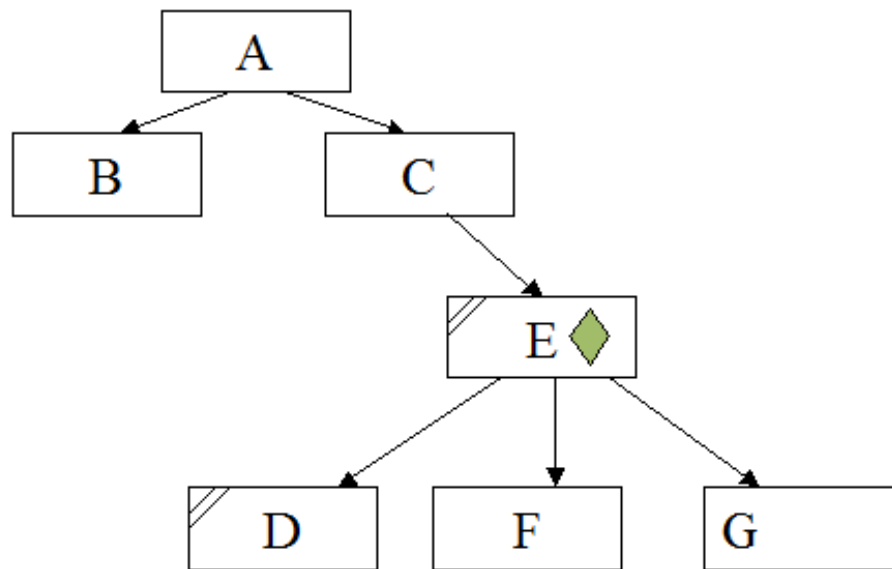
模块E做出的判定影响模块D，而D不在E的控制域内。

改善方案一



判定点上移

改善方案二



将在作用域
不在控制域
内的模块下
移

5.4 描绘软件结构的图形工具

层次图

矩形框表示模块，连线表示调用关系。

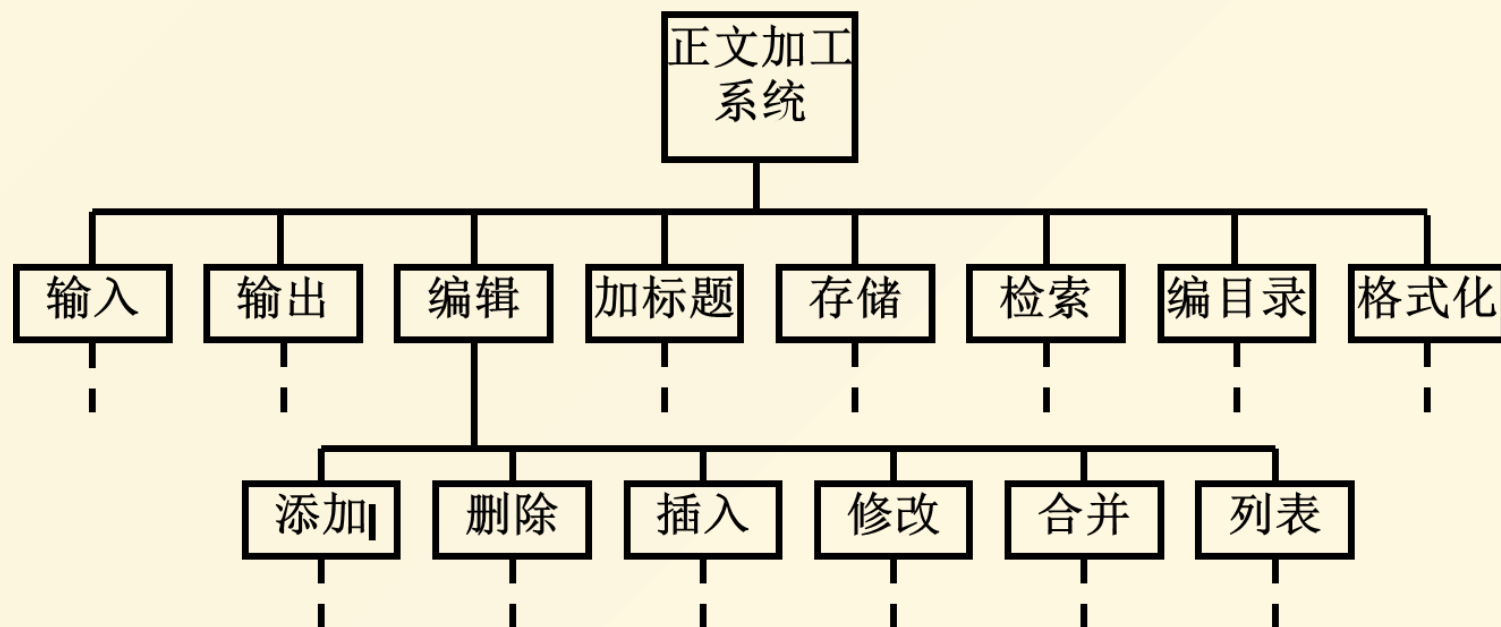
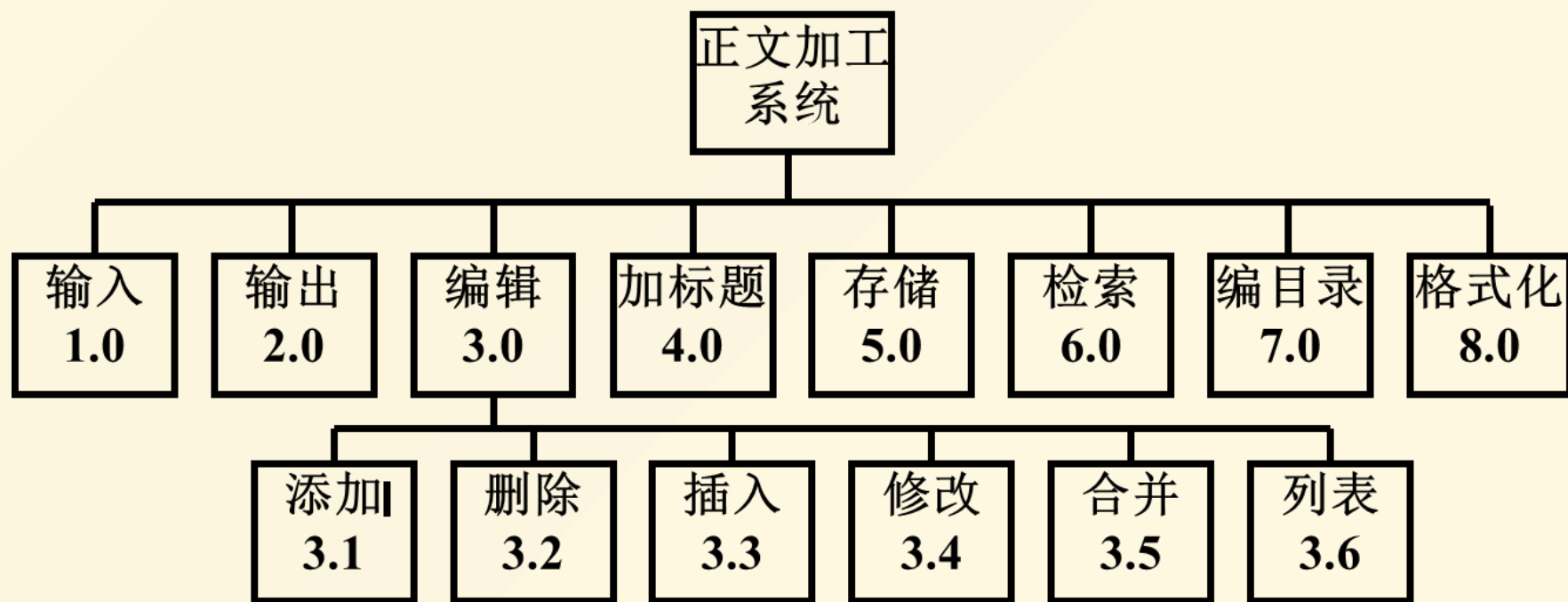


图5.3 正文加工系统的层次图

HIPO图（带编号的层次图 + 输入/处理/输出图）

带编号的层次图（H图）



输入/处理/输出图（IPO图）

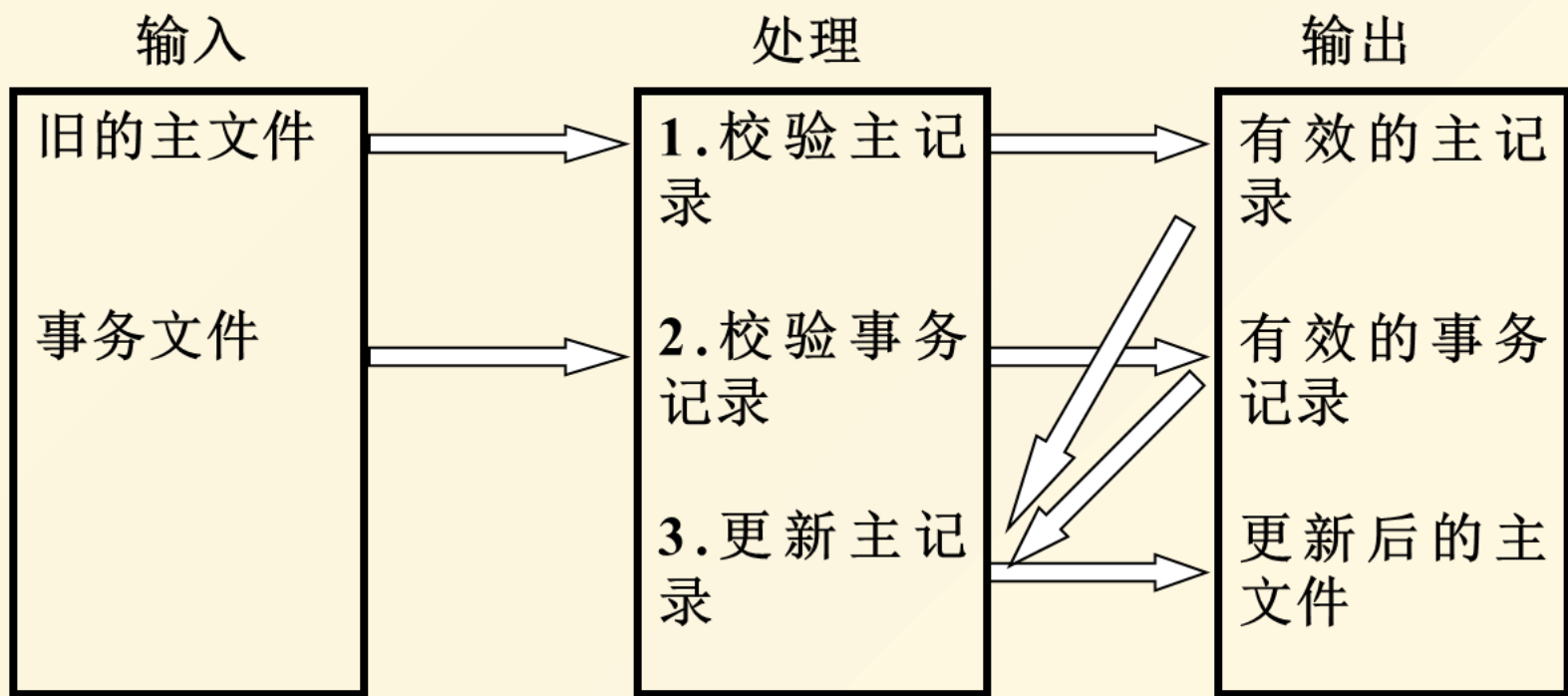


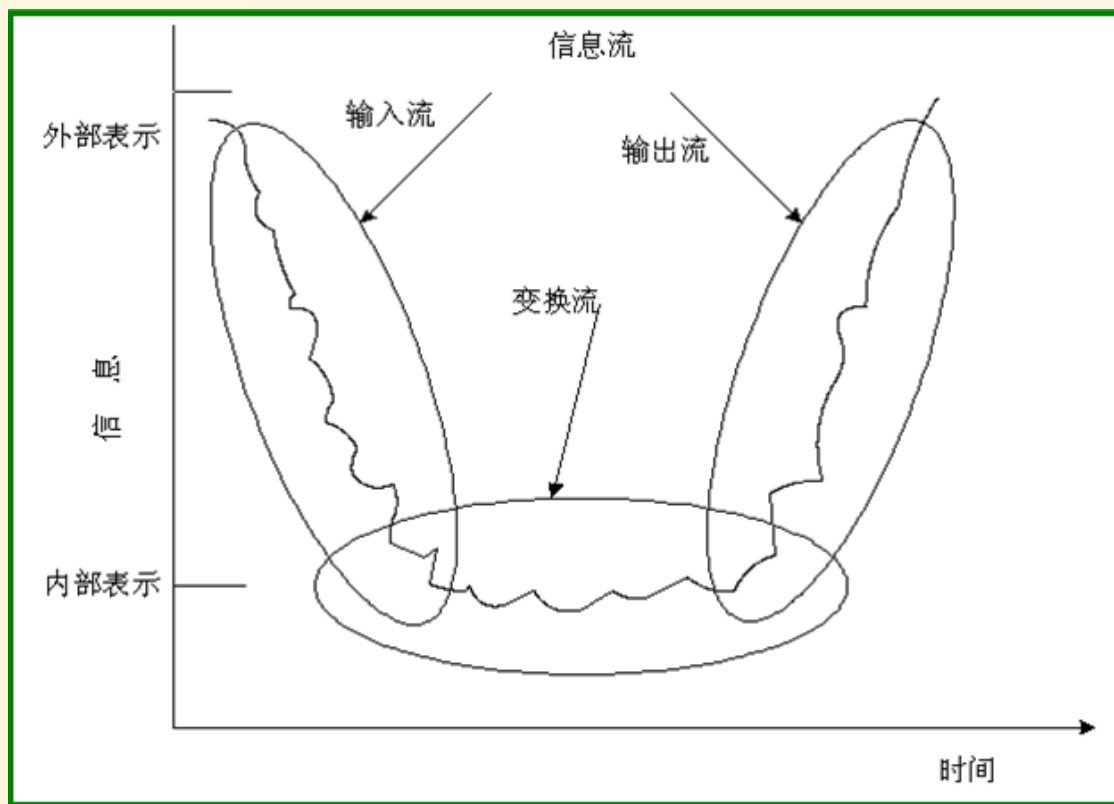
图3.7 IPO图的一个例子

5.5 面向数据流的设计方法

- 面向数据流的设计要解决的任务，就是将软件需求分析阶段生成的逻辑模型**数据流图**映射到表达软件系统结构的**软件层次图**或者**软件结构图**。
- 数据流的类型
 - 变换型数据流（变换流）
 - 事务型数据流（事务流）

变换流

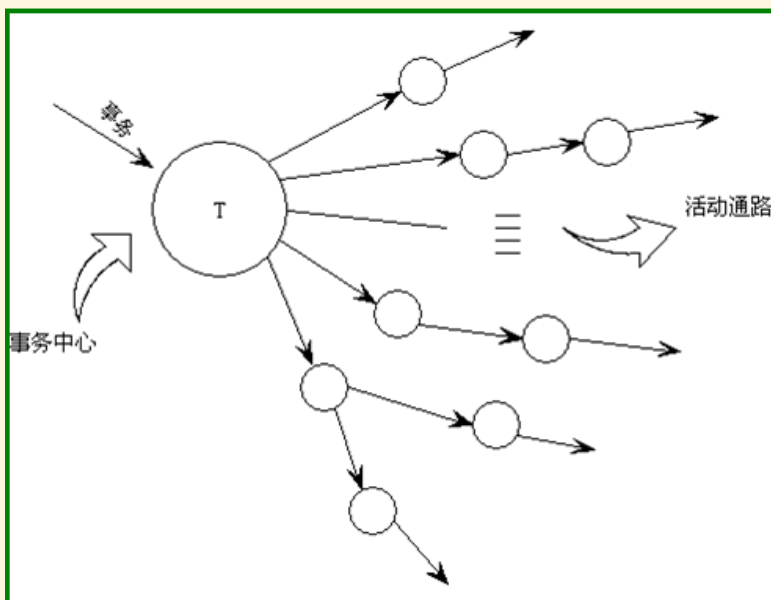
信息沿输入通路进入系统，由外部形式变换成内部形式，通过变换中心加工处理后再沿输出通路变换成外部形式离开软件系统。



事务流

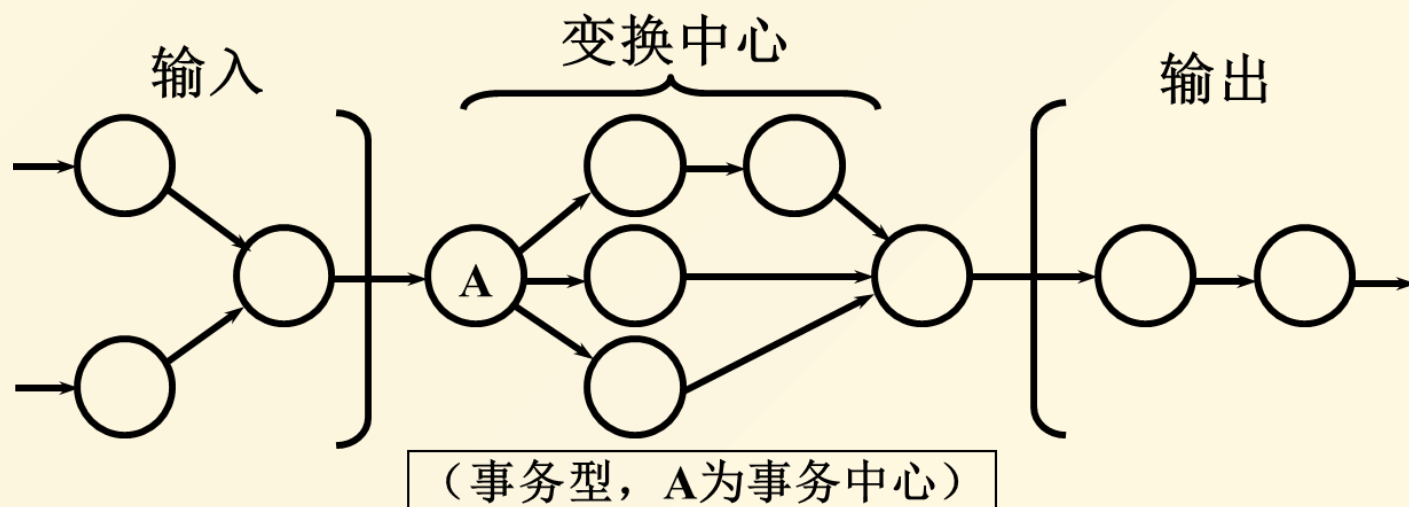
信息沿输入通路到一处理，由处理根据输入信息类型在若干动作序列中选一个执行。处理称事务中心，完成任务：

- (1) 接收输入信息（ 又称事务 ）；
- (2) 分析每个事务确定类型；
- (3) 根据事务类型选取一活动通路。



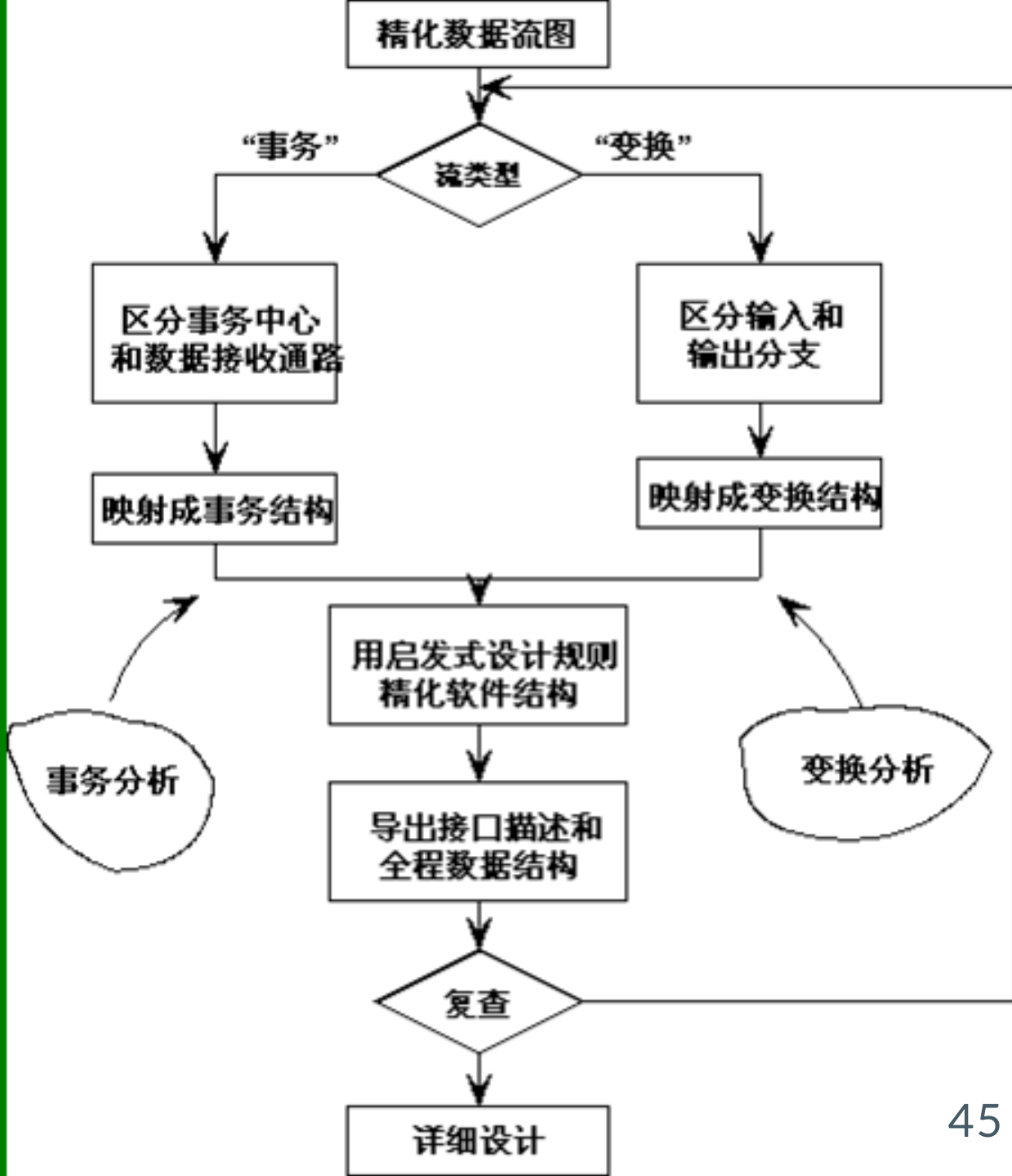
混合型数据流图

通常，一个实际系统的数据流图是变换型和事务型两种类型的混合体。如图所示，中间子块属事务型数据流，如果把中间子块视为一个处理整体的话，整个程序属变换型程序。



混合型数据流图

面向数据流设计过程



面向变换流的设计（变换设计）

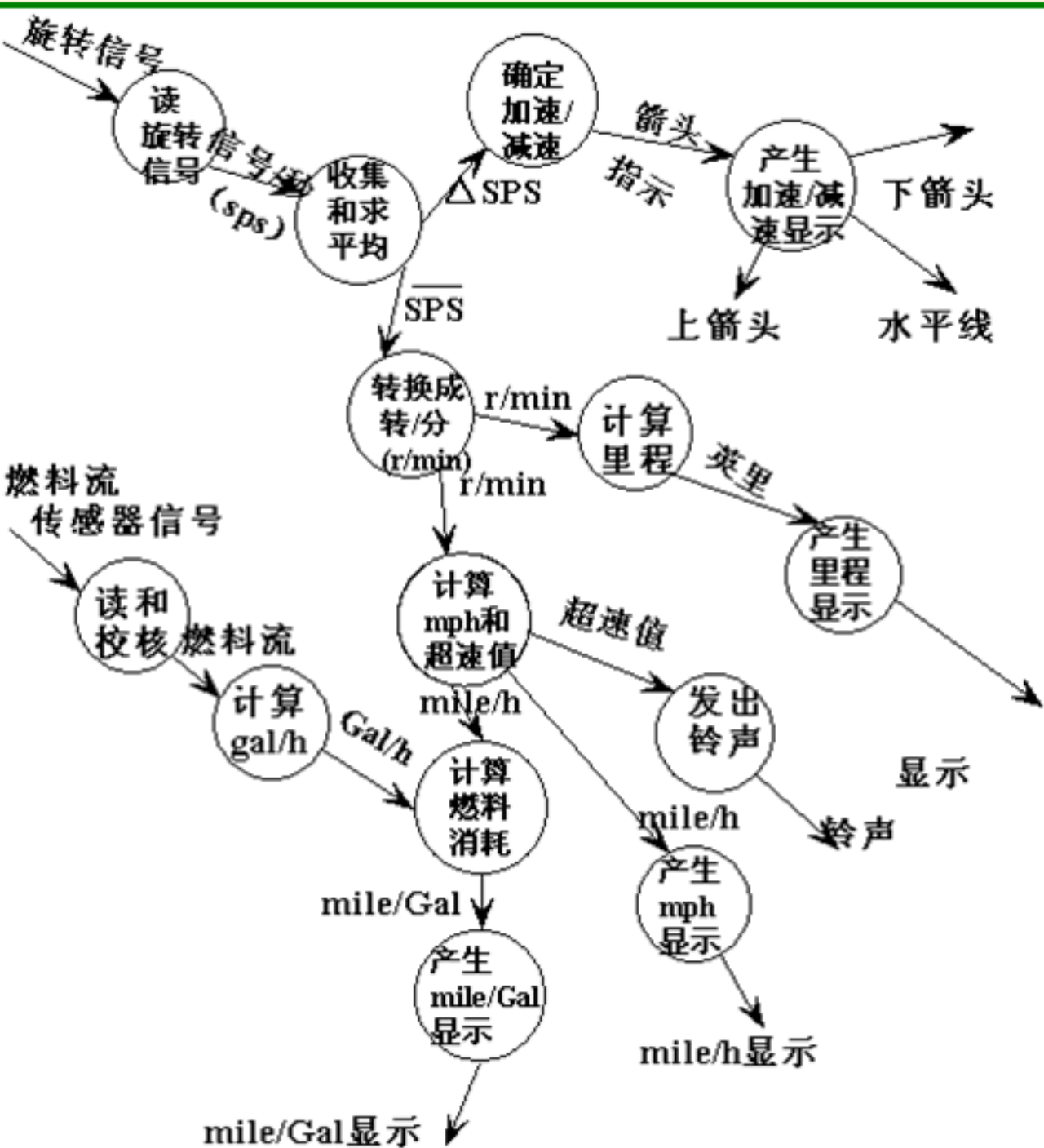
- 将具有变换流特点的数据流图映射成软件结构。
- 设计步骤：
 1. 复查基本系统模型
 2. 复查并精化数据流图
 3. 确定数据流图具有变换特性还是事务特性
 4. 找出变换中心
 5. 完成第一级分解
 6. 完成第二级分解
 7. 对软件结构进行精化

变换设计实例

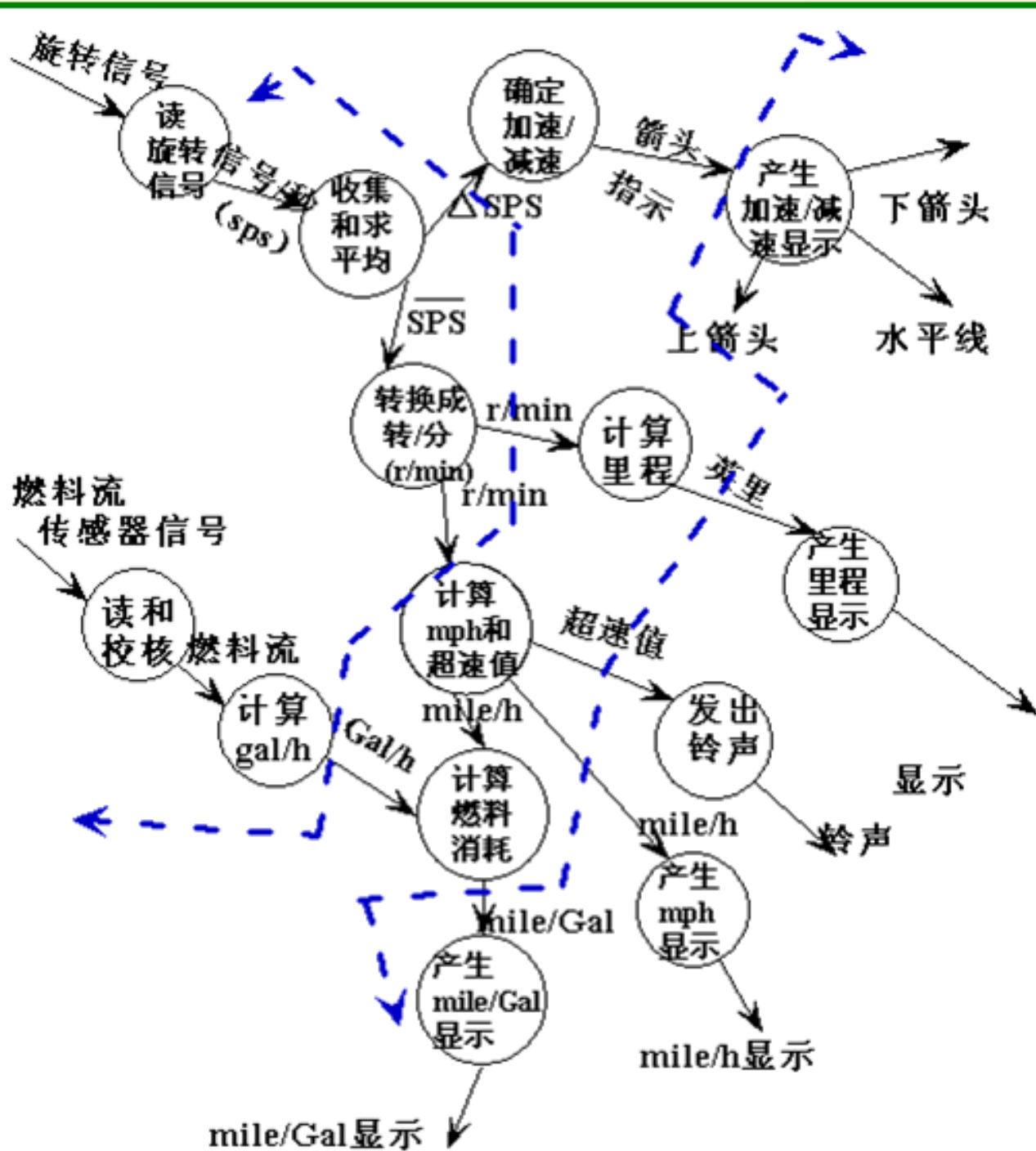
例：软件做在只读存储器中，使设备具有某些“智能”。假设的仪表板将完成下述功能：

- (1) 通过模/数转换实现传感器和微处理机接口；
- (2) 在发光二极管面板上显示数据；
- (3) 指示每小时英里数 (mile/h)、行驶里程和每加仑油行驶的英里数 (mile/Gal) 等；
- (4) 指示加速或减速；
- (5) 超速警告：发出超速警告铃声。

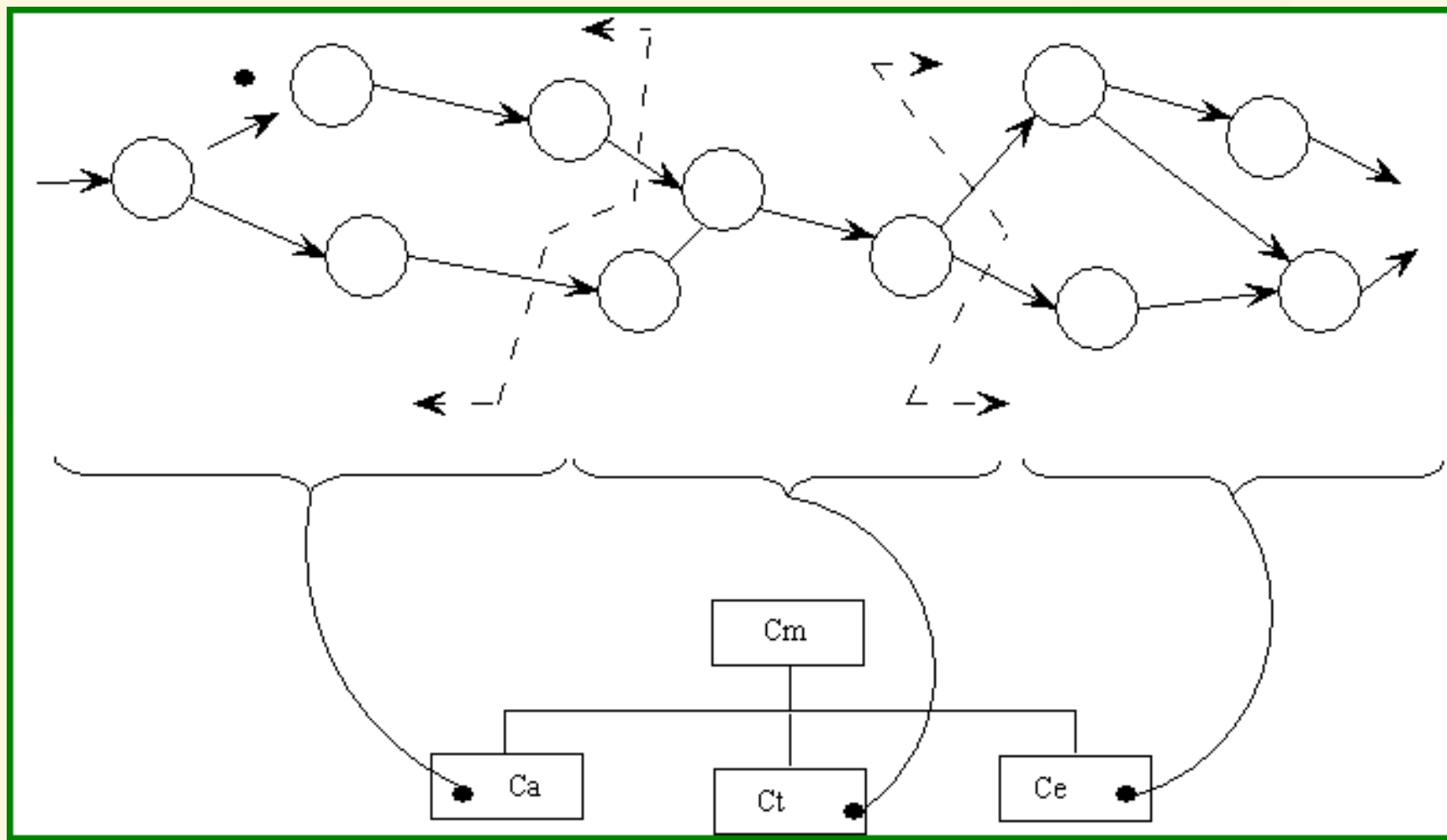
数字仪表板系统数据流图



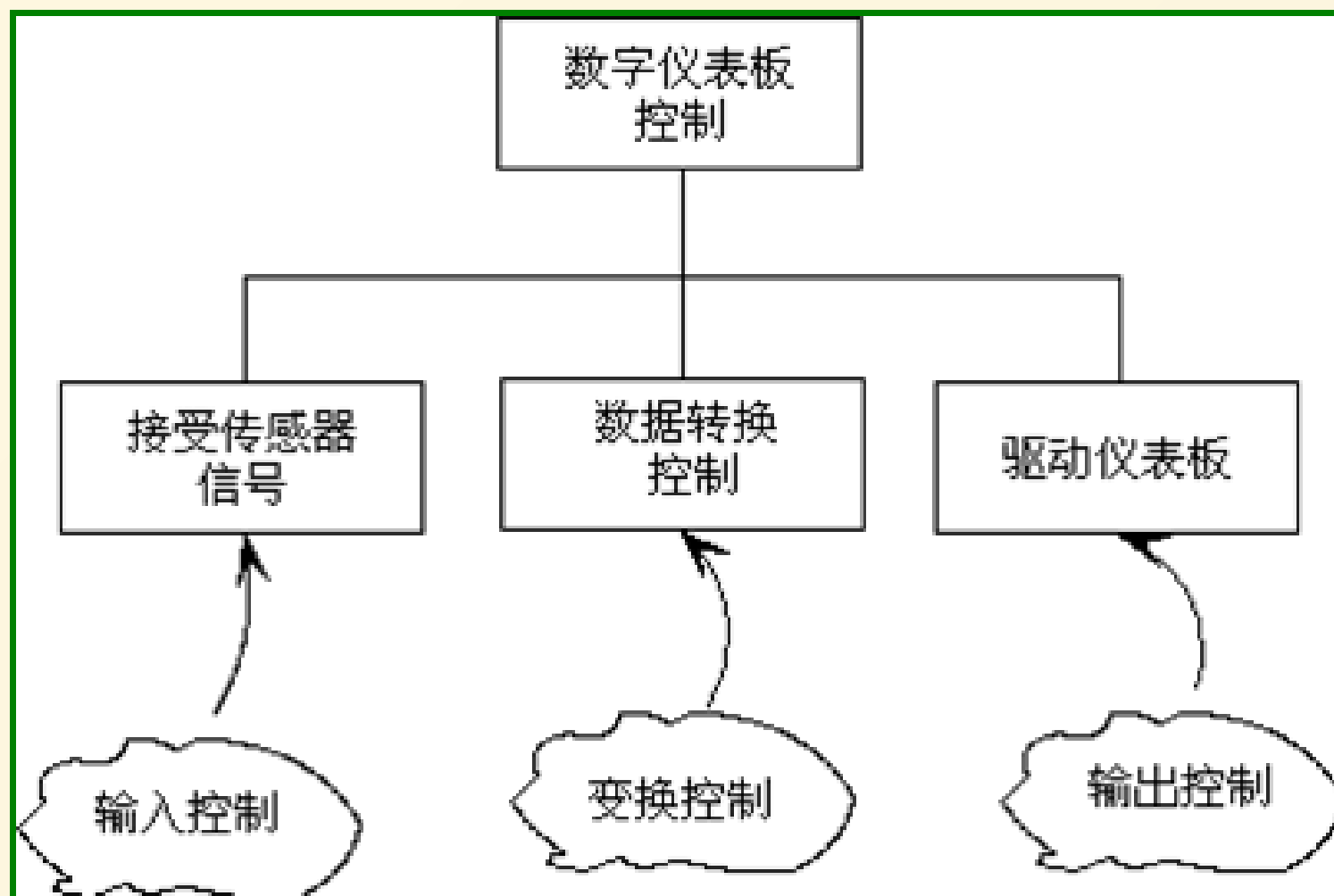
具有边界的数据流图



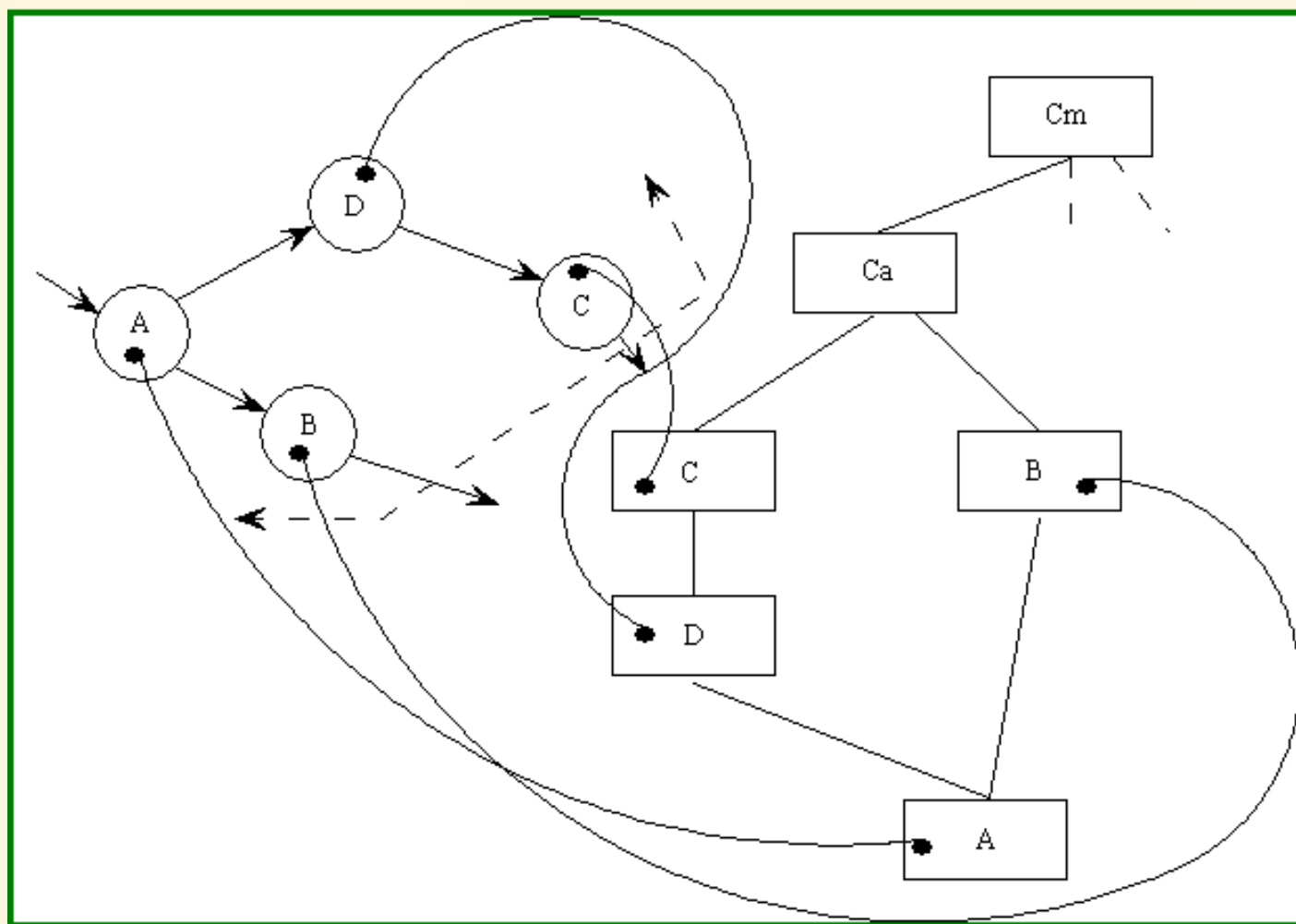
一级分解



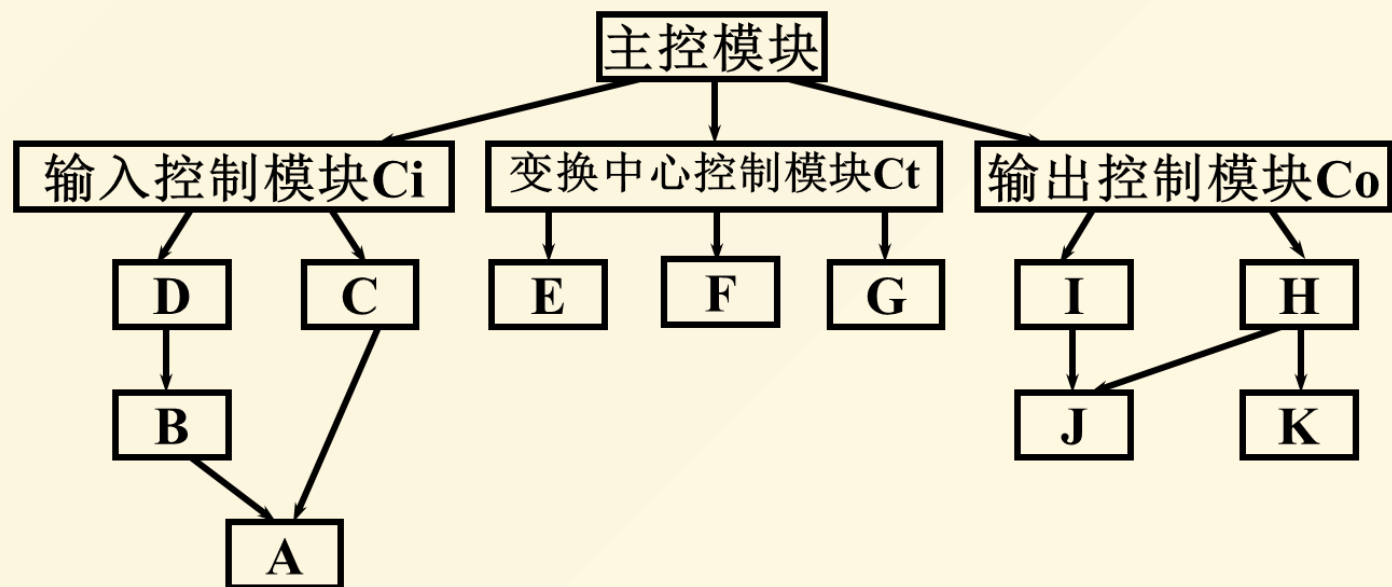
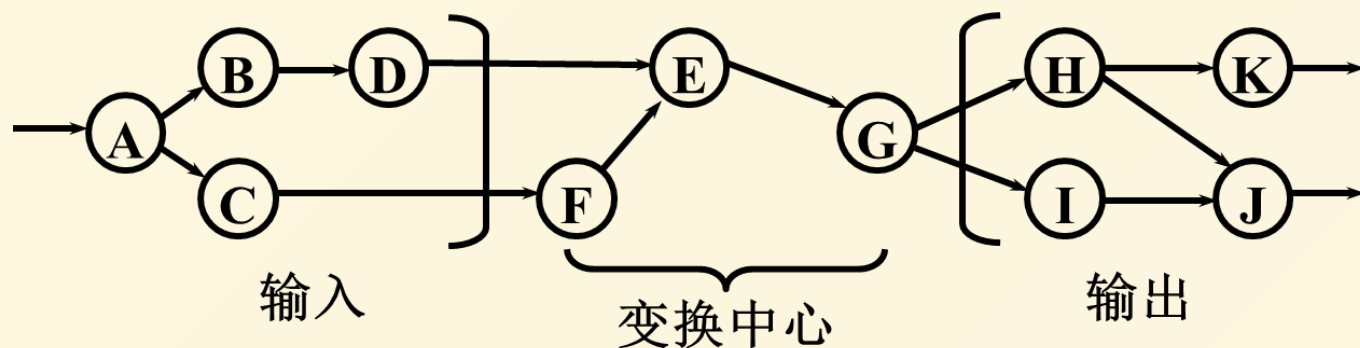
数字仪表板的第一级分解



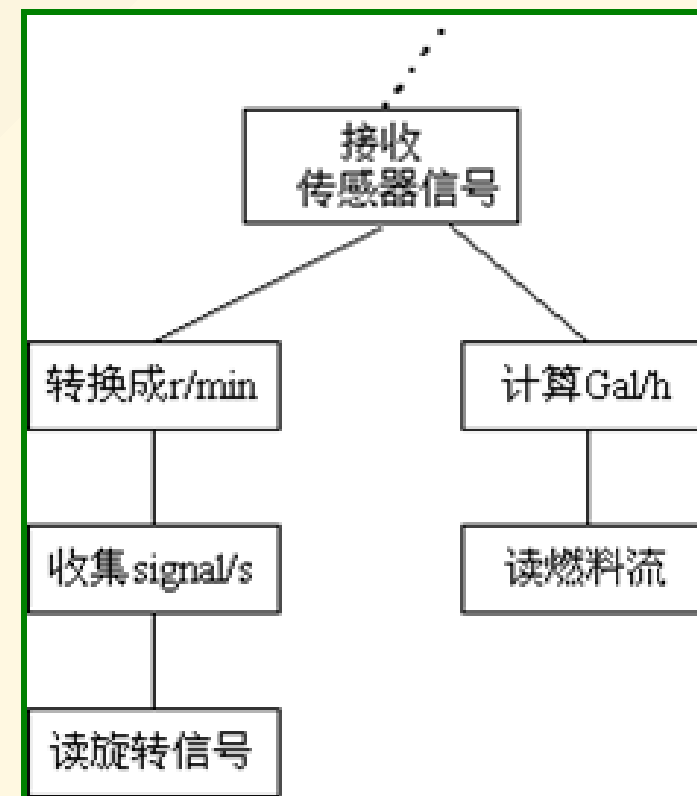
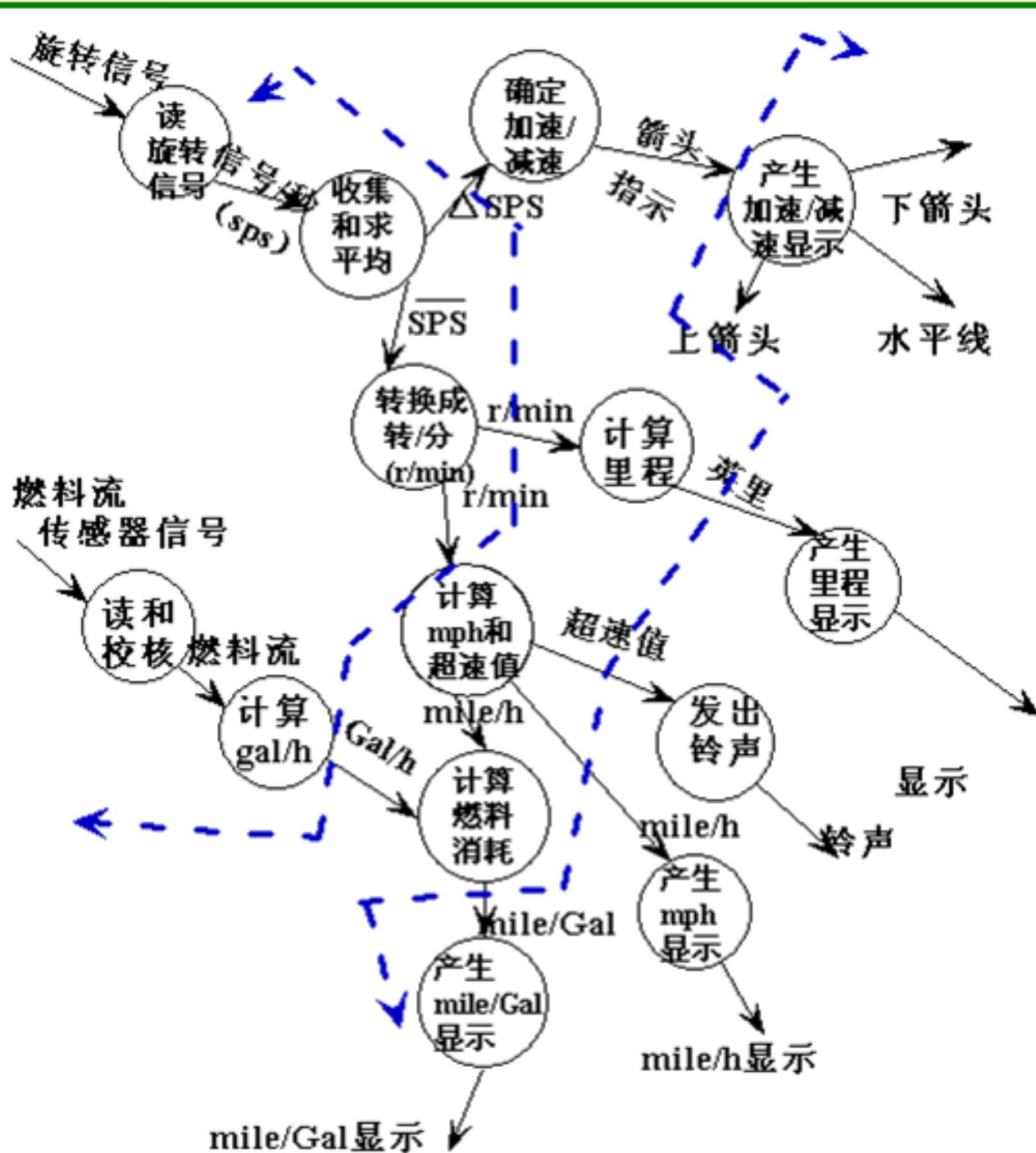
二级分解



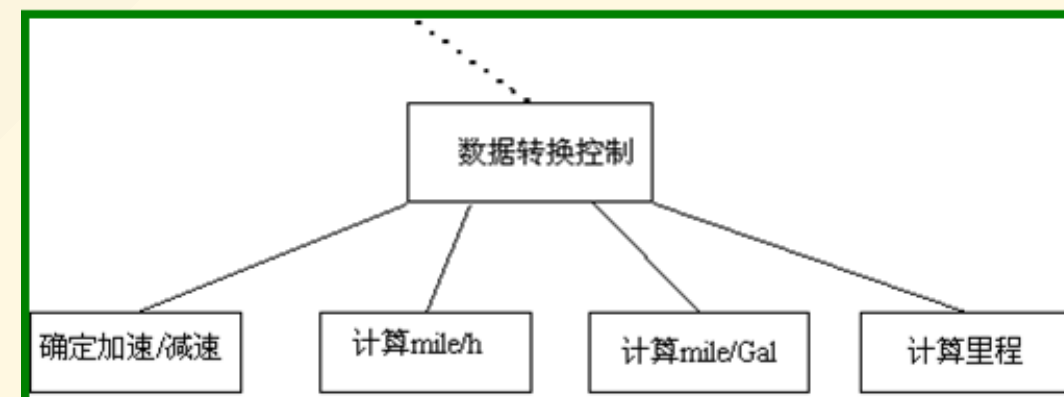
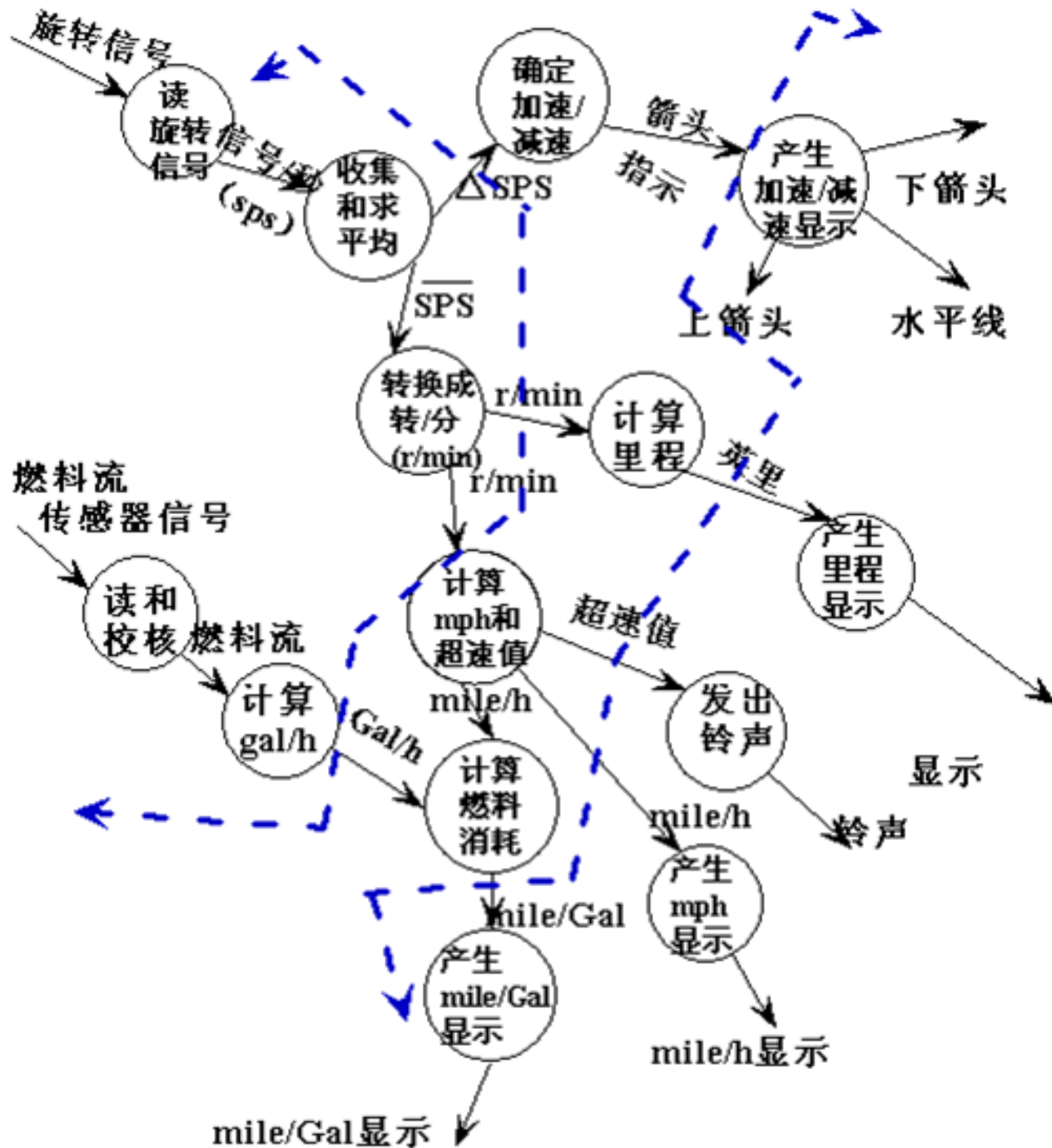
二级分解后得到的软件结构



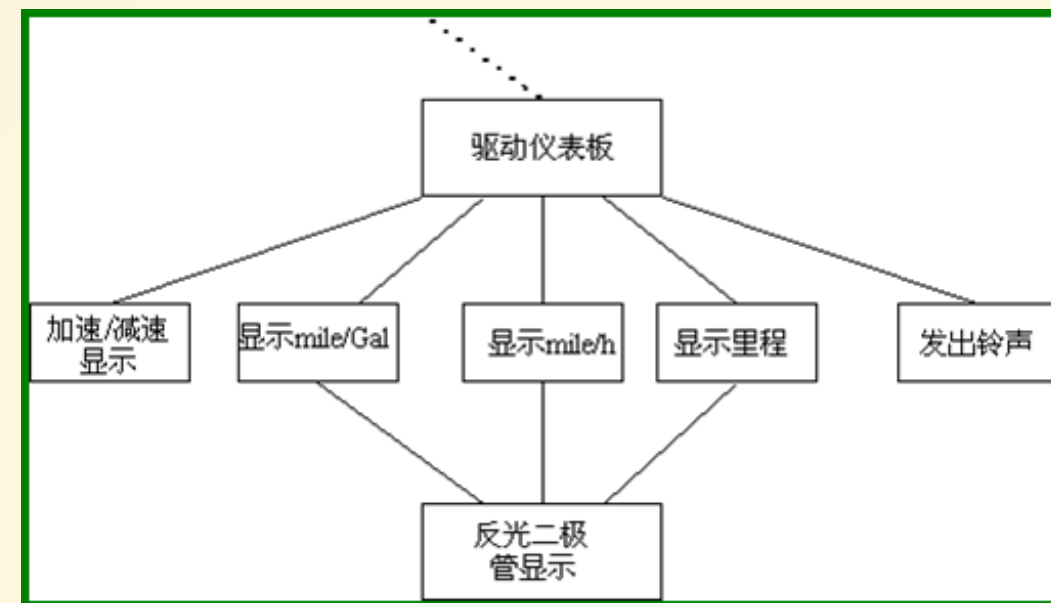
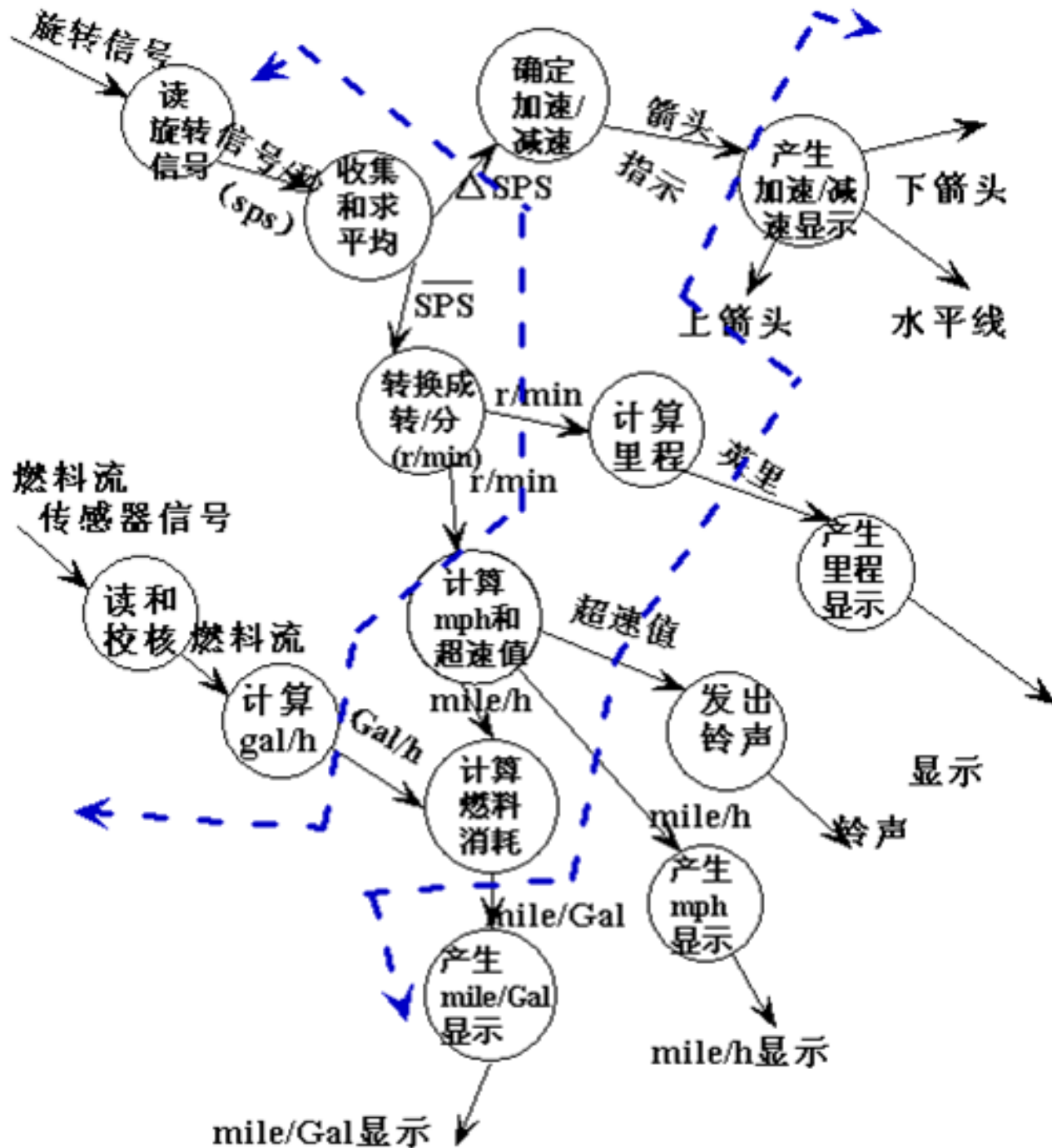
数字仪表板的第二级分解-输入结构



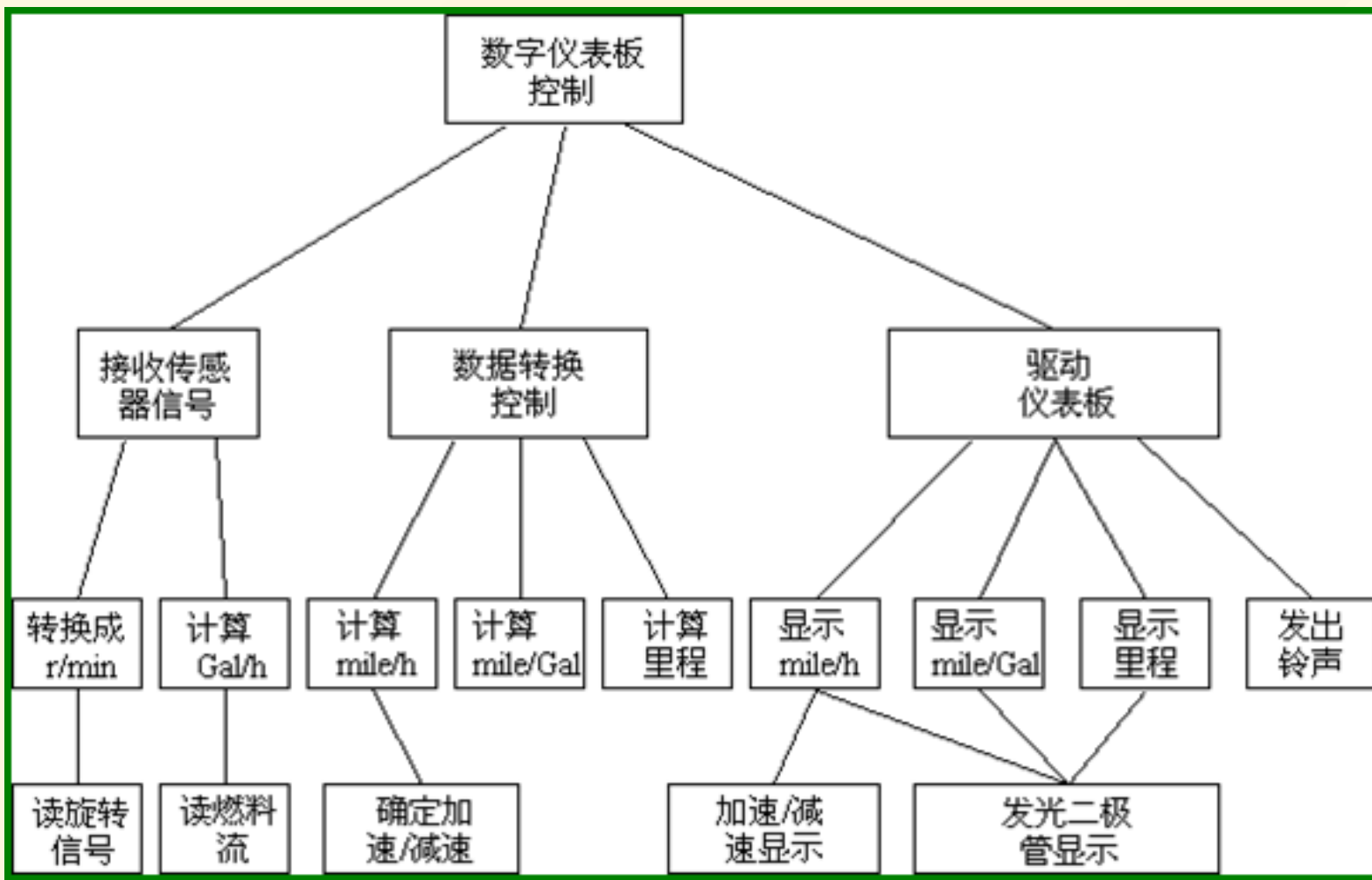
数字仪表板的第二级分解-变换结构



数字仪表板的第二级分解-输出结构

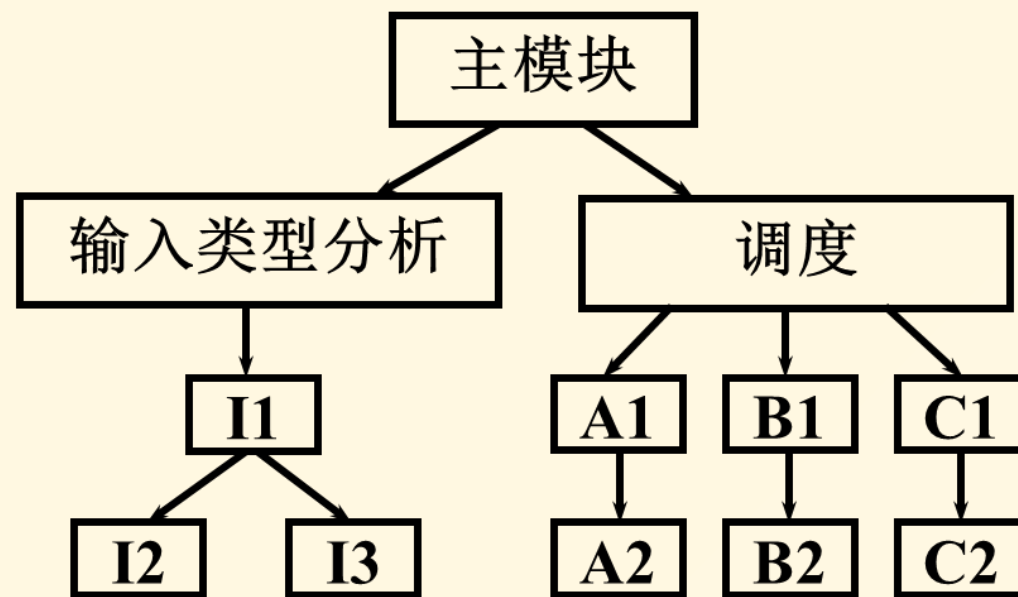
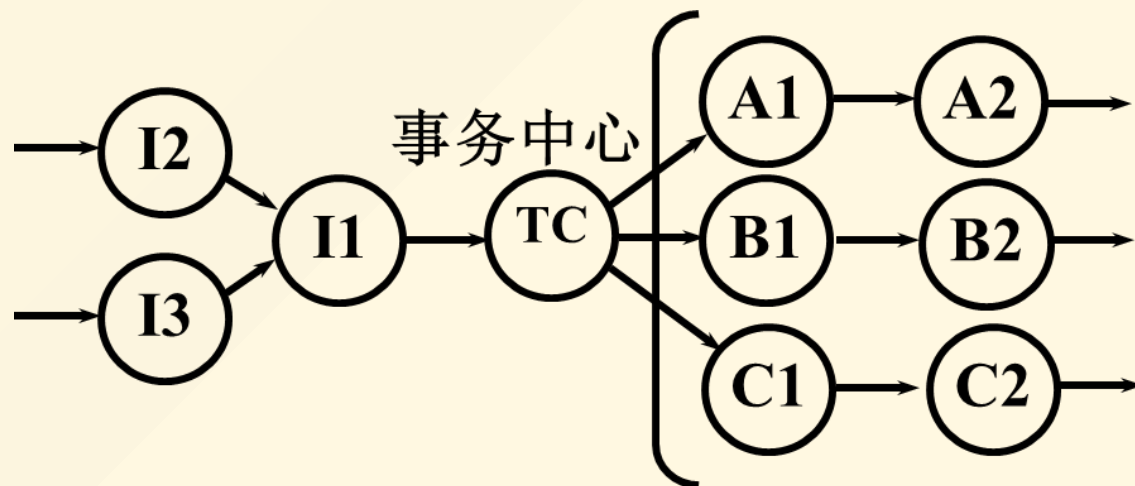


对初步软件结构精化



面向事务流的设计 (事务设计)

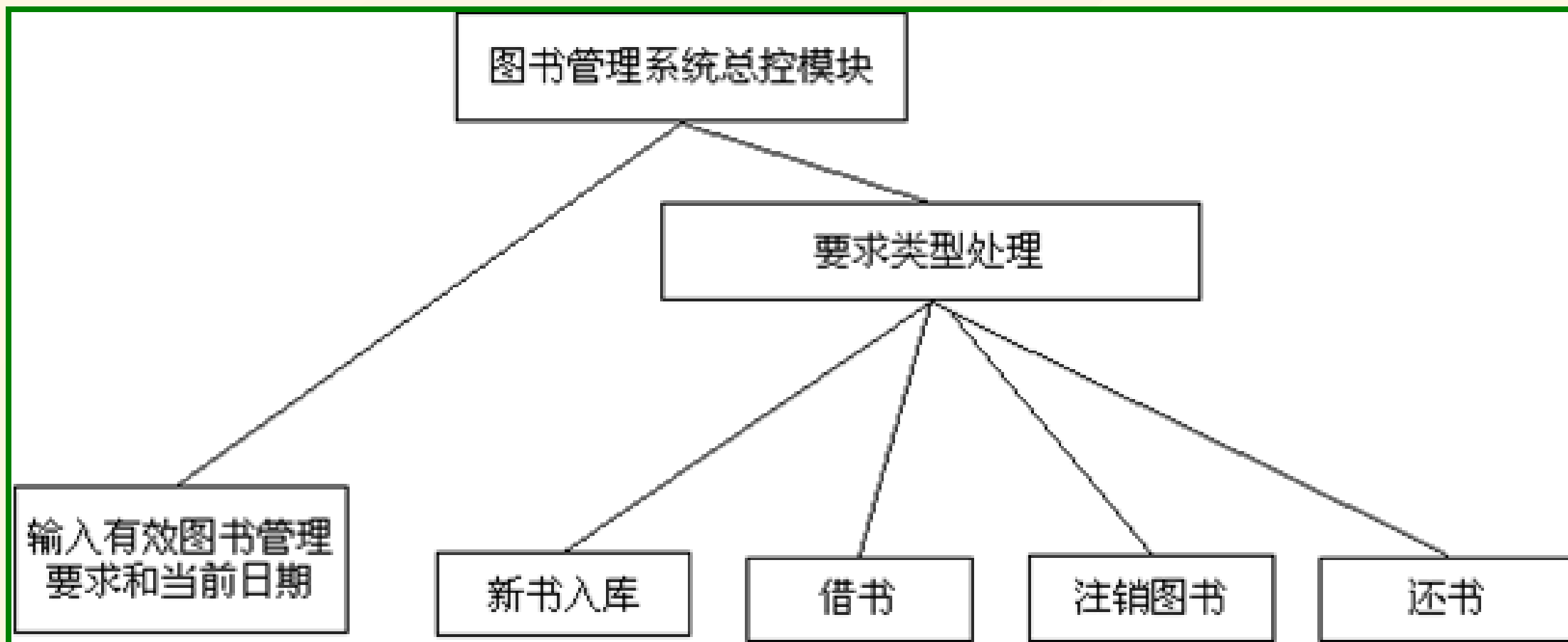
- 将事务型数据流图映射成软件结构。
- 设计步骤与变换设计的步骤大体相同。
 - 建立主控模块、输入类型分析模块和事务调度模块
 - 分别设计输入类型分析模块和调度模块的下层模块结构



事务设计实例：数据流图



事务设计实例：得到的软件结构



面向数据流的软件设计 练习（教材习题2.4、3.5、5.3）

某医院打算开发一个以计算机为中心的患者监护系统：

医院对患者监护系统的基本要求是随时接收每个病人的生理信号(脉搏、体温、血压、心电图等)，定时记录病人情况以形成患者日志，当某个病人的生理信号超出医生规定的安全范围时向值班护士发出警告信息，此外，护士在需要时还可以要求系统打印某个指定病人的病情报告。

1. 分层次地画出描述本系统功能的数据流图。
2. 根据数据流图导出软件的总体结构，用软件层次图表示。

概要设计说明书

该说明书是概要设计阶段的工作成果，它应说明功能分配、模块划分、程序的总体结构、输入输出以及接口设计、运行设计、数据结构设计和出错处理设计等，为详细设计提供基础。

本章重点

- 设计原理
- 启发规则
- 用层次图表示软件结构
- 面向数据流的设计方法