

2024年西电计科软件工程概论课程期末复习笔记

2024年西电计科软件工程概论课程期末复习笔记

考纲

第一章 概论

- 1.1 相关概念
- 1.2 软件生命周期
- 1.3 软件生存周期模型

第二章 可行性研究

- 2.1 可行性研究的任务与分类
- 2.2 ★ 数据流图 (DFD: Data Flow Diagram)
 - 基本概念
 - DFD 特性
 - ★ 教材购销系统的顶层 DFD
- 2.3 数据字典 (DD, Data Dictionary)

第三章 需求分析

- 3.1 ★ 需求分析的任务
- 3.2 ★ 需求分析的可使用工具
- 3.3 软件需求规格说明包含内容
- 3.4 ★ 结构化分析方法 (SA法)
- 3.5 ER图
- 3.6 状态转换图
- 3.7 项目管理定义

第四章 形式化说明技术

- 4.1 形式化技术的分类
- 4.2 有穷状态机
- 4.3 Petri 网
 - 基本特性
 - 电梯例子
 - 自动售货机
 - 生产者/消费者系统

第五章 总体设计

- 5.1 软件设计阶段
- 5.2 总体设计的任务及工具
- 5.3 模块独立性
 - 内聚
 - 耦合
 - 降低模块间耦合度
- 5.4 启发式规则
 - 相关概念
- 5.5 面向数据流的设计方法
 - 相关概念
 - 变换分析
 - 事务分析
 - 例题

第六章 详细设计

- 6.1 结构程序设计
- 6.2 程序流程图

6.3 盒图 (N-S图)

程序流程图到盒图的转换

6.4 PAD 图 (Problem Analysis Diagram)

程序流程图到 PAD 图的转换

6.5 实例

6.6 判定表与判定树 (了解)

6.7 面向数据结构的设计方法

Jackson 图——分析和描述数据结构

Jackson 伪代码

Jackson 实例

6.8 程序复杂度度量

1. 代码行度量法

2. McCabe法 (环路复杂性度量)

3. halstead法

第七章 实现

7.1 白盒测试

逻辑覆盖

基本路径测试

基本路径测试举例

7.2 黑盒测试

7.3 白盒测试和黑盒测试关系、区别

7.4 习题

第八章 软件维护、软件质量软件文档与产权保护

8.1 软件维护

8.2 软件文档

第九章 面向对象方法学

9.1 面向对象方法的优点:

9.2 ★ 面向对象设计六大原则

9.3 ★ 顺序图和协作图的区别

9.4 ★ UML设计三种模型: 对象模型 动态模型 功能模型

类图

用例图

对象图

顺序图

协作图

状态图

活动图

考纲

软工期末

四道大题（25），三道大题里融入一点点概念

85分分析题，14-15分概念题

范围：

可行性分析：数据流图

注意符号规范！！！

1. 源点与汇点是矩形框
2. 加工是圆形，椭圆，圆角矩形
3. 数据存储是两条横线

注意命名，必须有命名（加工，数据存储，数据流）！！！起名要准确，

细化数据流图带上符号

一致性检查

读题！！！有没有分层要求

需求分析

结构化：数据流图细化

面向对象：

必须会：用例图，类图，对象图

用例图：参与者，用例，边界

包含（《include》实心箭头，1指向多），泛化（空心箭头），扩展（《extend》实心箭头，多指向1）

参与者与用例之间的线没有箭头

能看懂：顺序图

了解：状态图

总体设计

数据流图转化为软件结构（模块）

变换流和事务流

详细设计

程序流程图，盒图，pad图

测试：

单元，集成，系统，交付（用户）

前三不需要用户参与

白盒测试

要有程序流程图，或者给代码画出程序流程图

8种覆盖（实际6种）

黑盒测试

等价类划分，边界设计测试用例，课件上的例子

例子看懂就会了

第一章 概论

1.1 相关概念

- **什么是软件工程、软件工程研究什么、什么是软件**

- 软件工程

指导计算机软件开发和维护的一门工程学科。采用工程的概念、原理、技术和方法来开发与维护软件，把经过时间考验而证明正确的管理技术和当前能够得到的最好的技术方法结合起来，以经济地开发出高质量的软件并有效地维护它，这就是软件工程。

- **★ 什么是软件**

- 运行时能提供所要求功能和性能的指令或计算机程序集合
 - 程序能满意地处理信息的数据结构
 - 描述程序功能需求及程序如何操作和使用所要求的文档

- 软件工程研究什么

工序、规范、质量、工具、人

- 软件的分类

- 系统软件、应用软件、嵌入式软件、可重用软件

1.2 软件生命周期

- **★ 软件生命周期有哪些阶段，每个阶段做什么**

- **问题定义：**确定要解决的问题，给出 **问题性质报告、工程目标和规模报告、访问调查**
 - **可行性研究：**用最小代价在尽可能短的时间内确定问题能否解决，即确定工程规模和目标，估计系统成本和效益。
 - **需求分析：**确定系统必须完成的工作，提出完整、准确、清晰、具体的要求
 - **总体/概要设计：**如何解决这个问题
 - **详细设计：**解法具体化，
 - **编码和单元测试：**书写程序，测试编写的每个模块
 - **综合测试：**集成测试、验收测试
 - **软件维护：**改正性维护、适应性维护、完善性维护、预防性维护

- **★ 软件工程诞生的原因：软件危机**

- 什么是软件危机，为什么会爆发软件危机

- 软件危机是在计算机软件开发和维护过程中遇到的一系列严重问题
 - 怎样满足对软件日益增长的需求

- 怎样维护数量不断膨胀的已有软件
- **☆软件危机爆发的原因**
- 软件本身的特点：逻辑部件规模庞大
- 不正确的开发和维护方法：忽视需求分析、软件开发定义为程序编写，轻视软件维护

1.3 软件生存周期模型

- **☆软件生存周期模型**
 - **瀑布模型**：合格进入下一阶段，否则进入前一阶段
 - **快速原型模型**：快速构建可以运行的原型系统
 - **螺旋模型**：将开发过程分为几个螺旋周期，沿螺旋线旋转

第二章 可行性研究

2.1 可行性研究的任务与分类

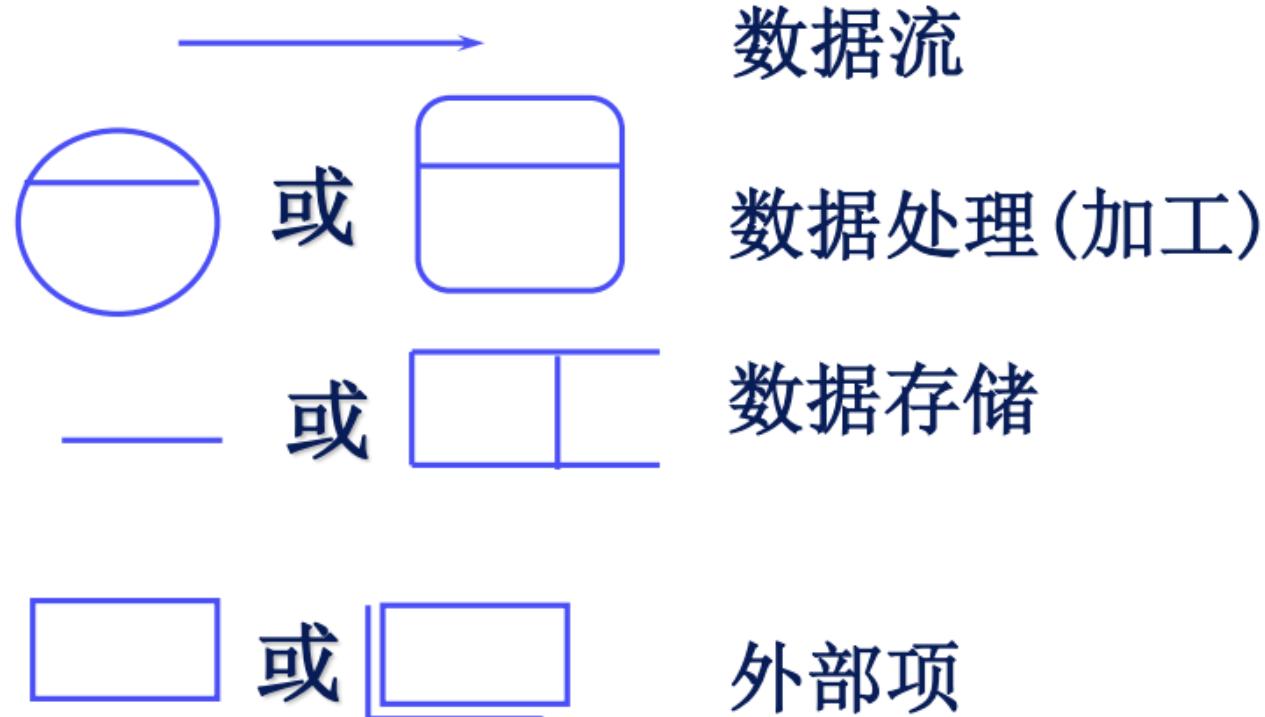
- 可行性研究的目的是用最小的代价，在尽可能短的时间内确定问题是否能够解决；
- 可行性方法是从哪四个方面进行分析的（**可行性研究分类**）：
 - 技术可行性
 - 经济可行性
 - 操作可行性
 - 法律可行性

2.2 ☆数据流图 (DFD: Data Flow Diagram)

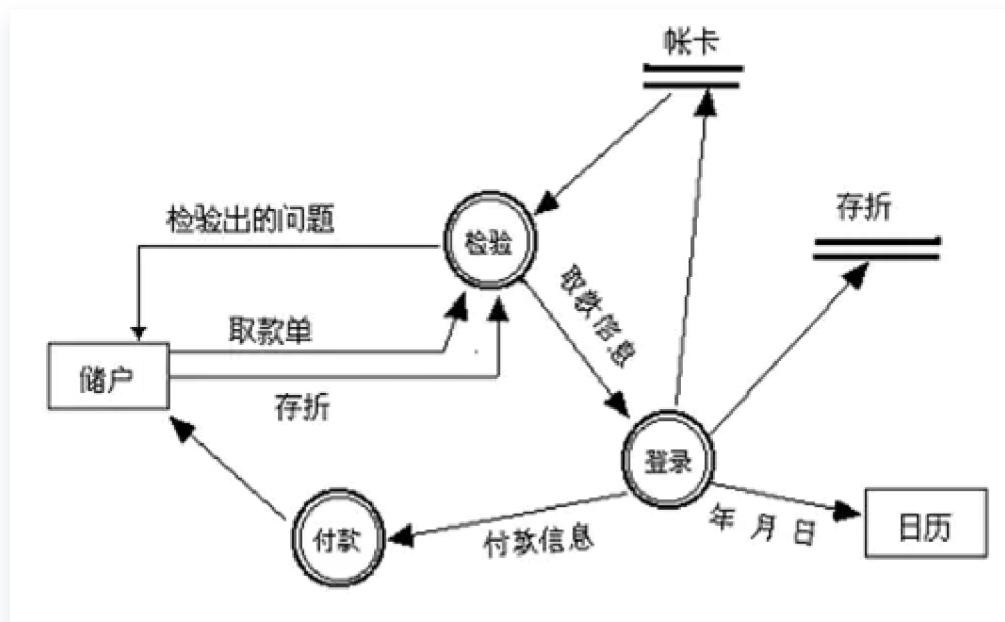
基本概念

- 数据流图描绘系统的逻辑模型；
- 只描绘信息在系统中流动和处理情况；

- 基本成分



- 描述底层软件结构的 **数据流图**



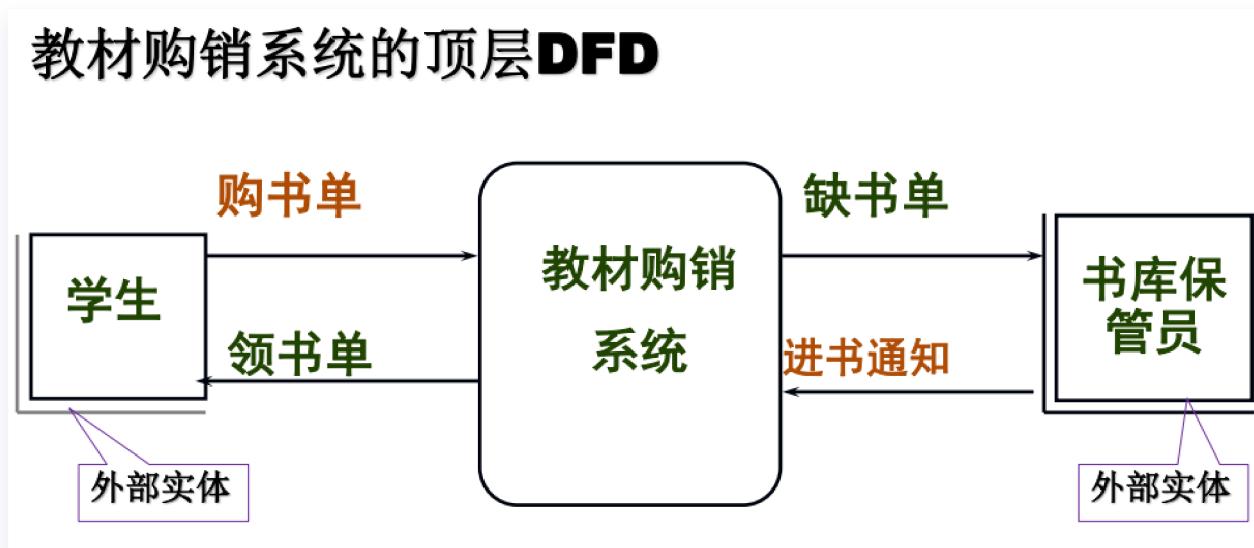
- 圆圈：加工
- 方框：数据输入或输出
- 箭头：数据流
- 双横线：数据存储文件

DFD 特性

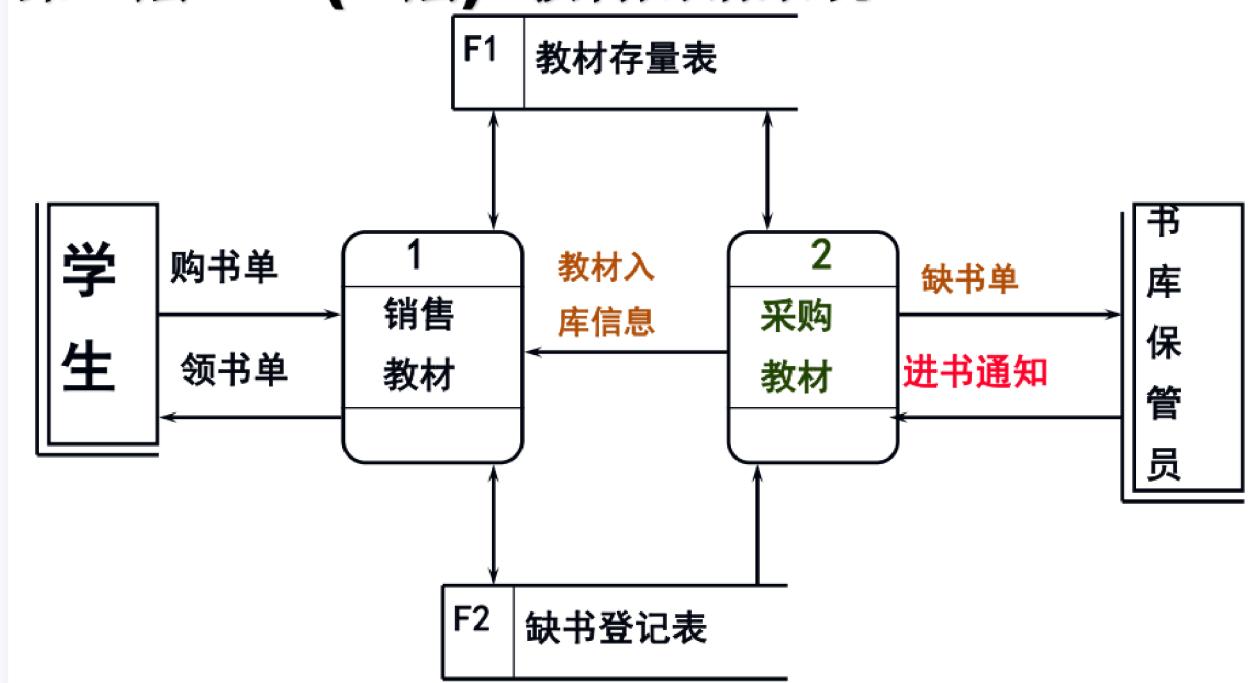
- DFD 不表示程序的控制结构，只描述数据的流动；
- DFD 图绘制
 - 先画出顶层 DFD;

- 自顶向下画出各层 DFD;
- 由外向里画 DFD;
- 优点:
 - 便于实现：采用逐步细化的扩展方法，可避免一次引入太多细节；
 - 便于使用：用一组图代替一张总图，方便阅读；

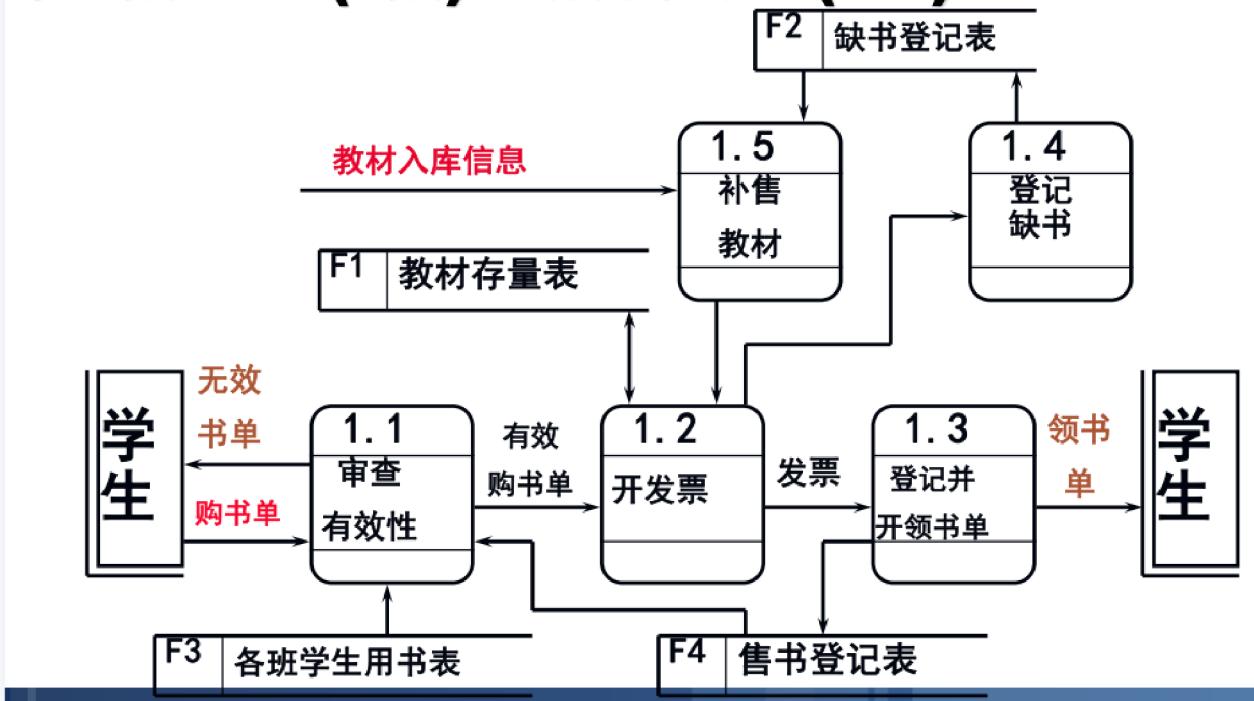
★ 教材购销系统的顶层 DFD



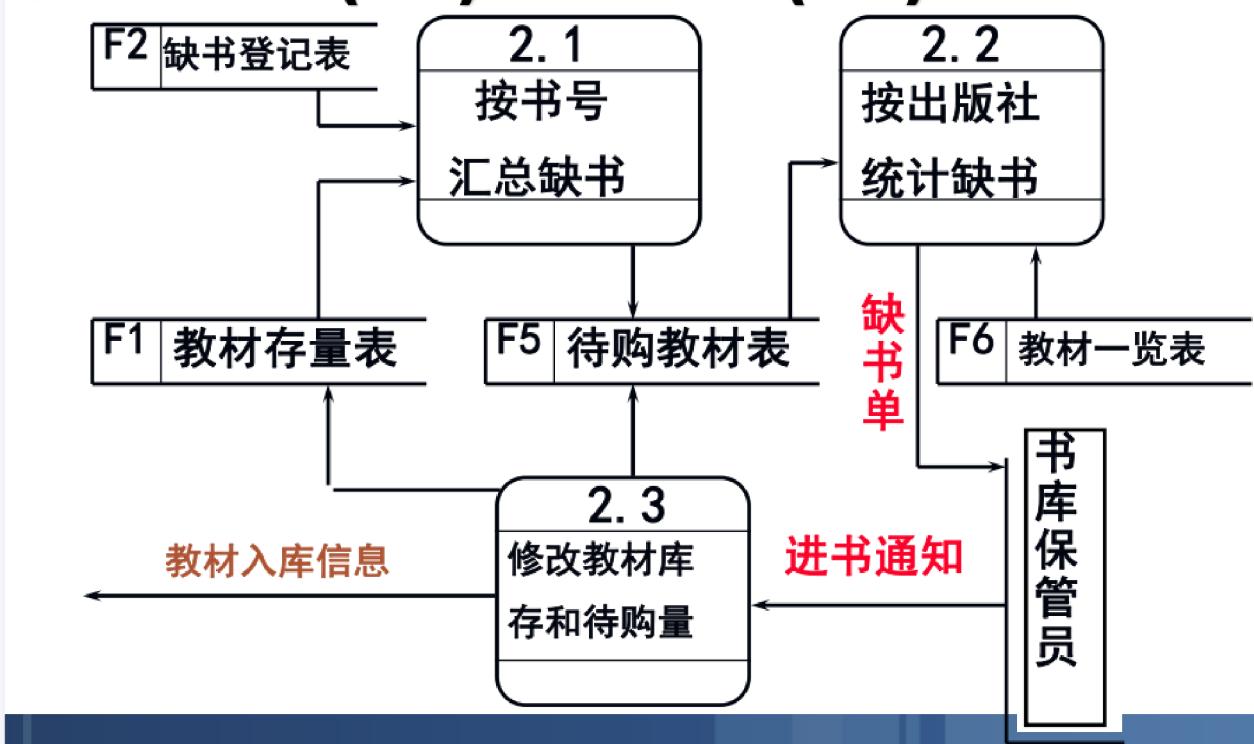
第二层DFD(0层) 教材购销系统



第三层DFD (1层) 销售子系统(1.0)



第三层DFD (1层) 采购子系统(2.0)



2.3 数据字典 (DD, Data Dictionary)

- ★ 定义：数据字典是关于数据的信息的集合，也就是对数据流图中包含的所有元素的定义的集合；
- ★ 任务：对于数据流图中出现的所有被命名的图形元素在字典中作为一个词典加以定义，使得每一个图形元素的名字都有一个确切的解释；

- 关系：数据流图和数据字典共同构成系统的逻辑模型；
- 组成：数据流，数据流分量，数据存储，处理；
- 数据结构描述

符号	含义	解释
=	被定义为	例如， $x=a+b$ ，表示x由a和b组成
+	与	例如， $x=[a,b]$, $x=[a b]$,表示x由 a或由b组成
[...,...]	或	
[... ...]	或	
{...}	重复	例如， $x=\{a\}$ ，表示x由0个或多个a组成
m{...}n	重复	例如， $x=3\{a\}8$ 表示x中至少出现3次a，至多出现8次a
(...)	可选	例如， $x=(a)$ ，表示a可在x中出现，也可不出现
“...”	基本数据元素	例如， $x="a"$ ，表示 x为取值为a的数据元素
..	联结符	例如， $x=1..9$ ，表示x可取1到9之中的任意值

第三章 需求分析

3.1 ★需求分析的任务

需求分析的任务就是借助于当前系统的逻辑模型到处目标系统的逻辑模型，解决目标系统“做什么”的问题；

3.2 ★ 需求分析的可使用工具

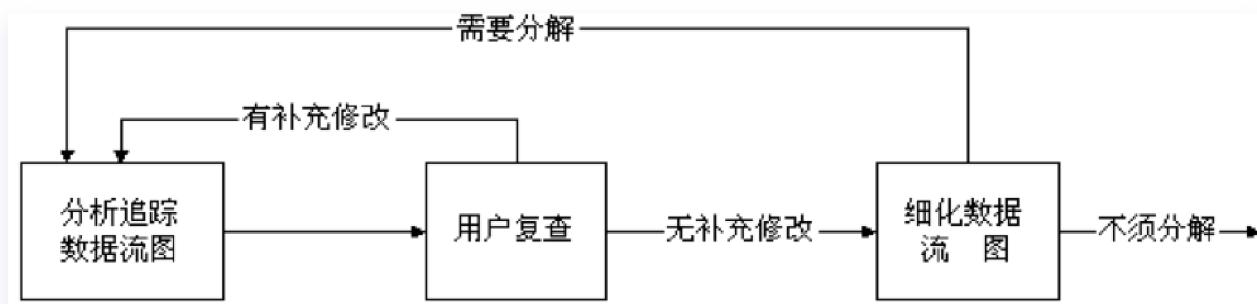
- 数据流图
- 实体-联系图
- 状态转换图
- 数据字典和处理算法

3.3 软件需求规格说明包含内容

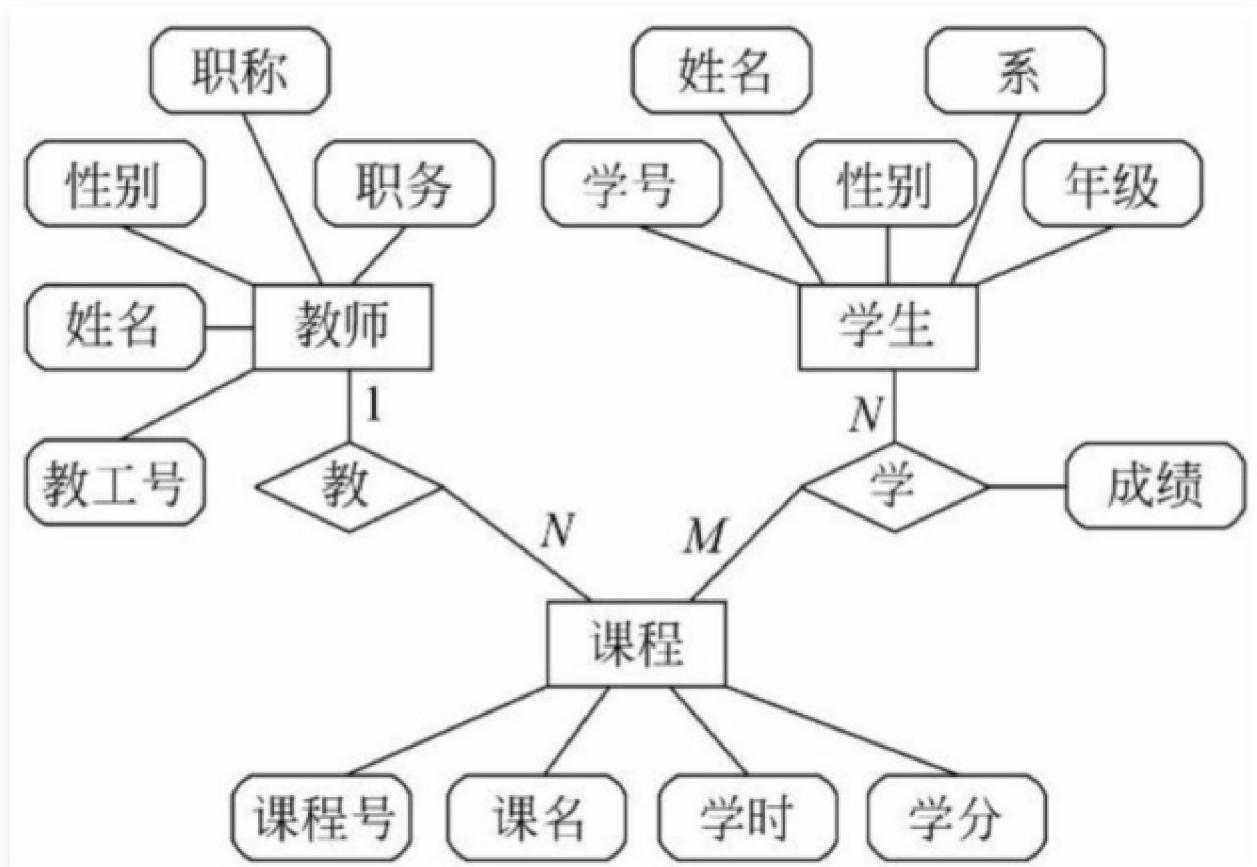
- 任务概述：对目标、用户特点的说明
- 需求规定：对功能、性能的规定
- 运行环境规定：对整个系统的操作环境的说明

3.4 ★结构化分析方法 (SA法)

SA 法是面向数据流自上而下逐步求精进行需求分析的方法，由可行性研究得出目标系统的高层数据流图，需求分析的目的之一是把数据流和存储定义在元素级。



3.5 ER图



第一、第二、第三范式的定义是：

第一范式（1NF） 每个属性值都必须是原子值，即仅仅是一个简单值而不含内部结构。

第二范式（2NF） 满足第一范式条件，而且每个非关键字属性都由整个关键字决定（而不是由关键字的一部分来决定）

第三范式（3NF） 符合第二范式条件，每个非关键字属性都仅由关键字决定，而且一个非关键字属性不能仅仅是另一个非关键字属性的进一步描述（即一个非关键字属性值不依赖于另一个非关键字属性值）

用教学管理为例说明如何规范化：

有三个实体，即课程、学生和教师，用三个关系保存它们的信息：

学生（学号，姓名，性别，年龄，专业，籍贯）

教师（职工号，姓名，年龄，职称，工资级别，工资）

课程（课程号，课程名，学分，学时，课程类型）

为表示实体型之间的关联，又建立两个关系：

选课（学号，课程号，听课出勤率，作业完成率，分数）

教课（职工号，课程号）

这五个关系，组成了数据库的模型。在每个关系中，属性名下加下划线指明关键字。并规定关键字能唯一地标识一个元组。

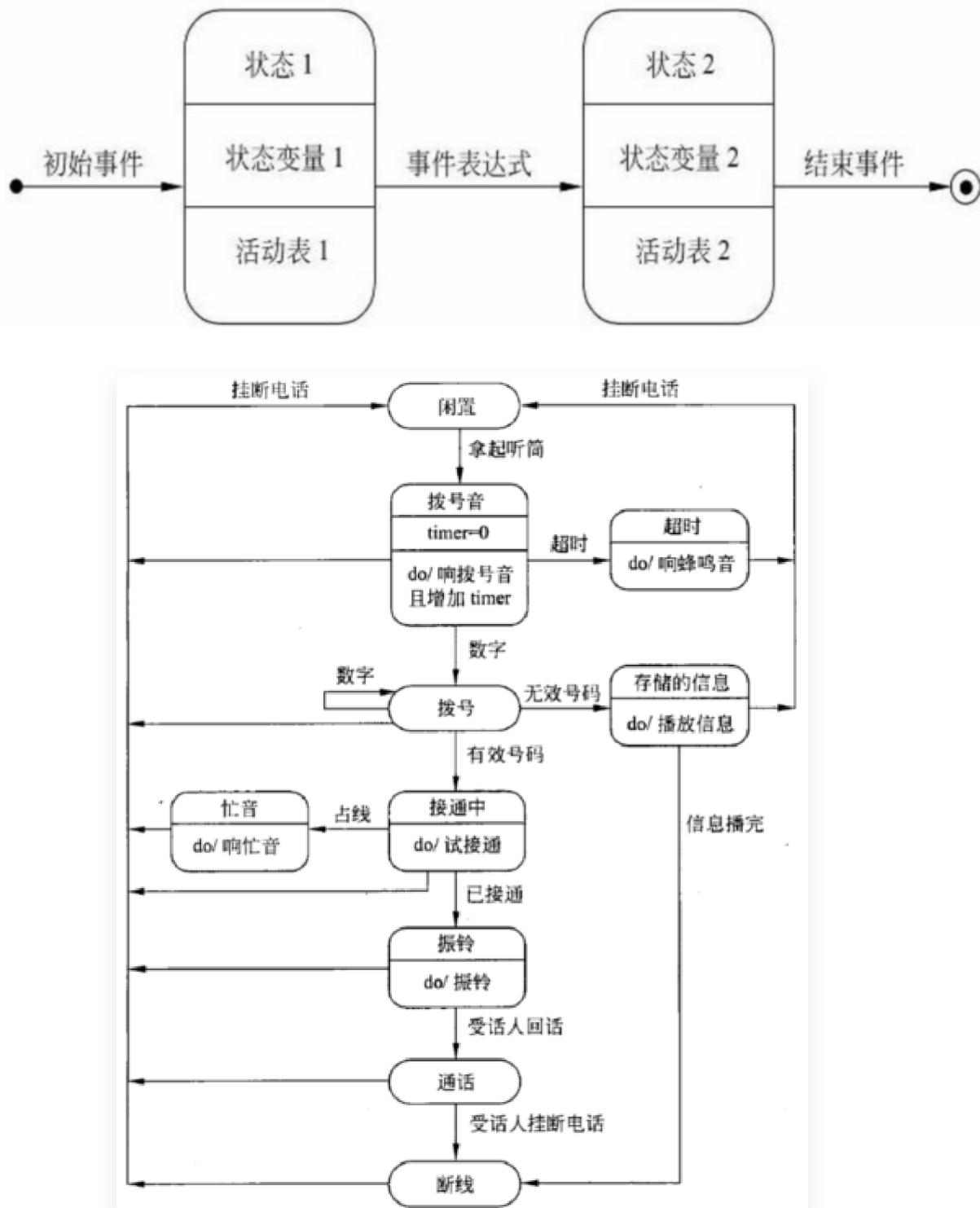
3.6 状态转换图

- 状态转换图通过描绘系统的状态及引起系统状态转换的事件，来表示系统的行为；
- 状态是任何可以被观察到的系统行为模式，一个状态代表系统的一个行为模式；
- 事件是在某个特定时刻发生的事情，它是对引起系统做动作或从一个状态转换到另一个状态的外界事件的抽象；
- 符号
 - 初态：实心圆表示
 - 终态：一对同心圆
 - 中间状态：圆角矩形，平衡线将其分割为上中下三部分，上部分表示状态的名称【必需】，中部分表示状态变量的名字和值【可选】，下部分表示活动表【可选】；

- 状态之间：带箭头的连线表示状态转换，可用时间表达式表明触发的事件；

■ 状态图中使用的符号

活动表中出现的标准事件：**do**, **exit**, **entry**



3.7 项目管理定义

一个确定的时间范围内，为了完成一个既定的目标，并通过特殊形式的临时性组织运行机制，通过有效的计划、组织、领导与控制，充分利用既定有限资源的一种系统管理方法。

4.1 形式化技术的分类

- 根据说明目标软件系统的方式
 - 面向模型的形式化方法
 - 面向属性的形式化方法
- 根据表达能力
 - 基于模型的方法
 - 基于逻辑的方法
 - 代数方法
 - 过程代数方法
 - 基于网络的方法

4.2 有穷状态机

状态集J: {保险箱锁定, A, B, 保险箱解锁, 报警}。

输入集K: {1L, 1R, 2L, 2R, 3L, 3R }。

转换函数T: 由当前状态和当前输入确定下一个状态（次态），如表4. 1所示。

初始态S: 保险箱锁定。

终态集F: {保险箱解锁, 报警}。

- 当前状态+事件+谓词 \implies 下一个状态

4.3 Petri 网

基本特性

目的一确定系统中隐含的定时问题的一种有效技术。

Petri网包含4种元素：一组位置P、一组转换T、输入函数I和输出函数O。Petri网的组成。其中，

- 一组位置P，在图中用圆圈代表位置。
- 一组转换T，在图中用短直线表示转换。
- 两个用于转换的输入函数

$$I(t_1) = \{P_2, P_4\}$$

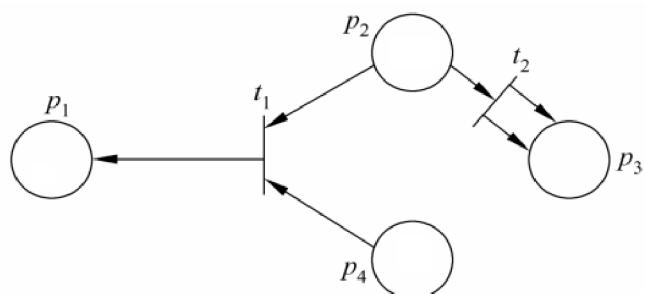
$$I(t_2) = \{P_2\}$$

- 两个用于转换的输出函数。

$$O(t_1) = \{P_1\}$$

$$O(t_2) = \{P_3, P_4\}$$

注意输出函数O(t₂)中有两个P₃，是因为有两个箭头有t₂指向P₃。



- 更形式化的Petri网结构，是一个四元组C=(P, T, I, O)。其中，
P={P₁, ..., P_n}是一个有穷位置集，n>=0。
T={t₁, ..., t_m}是一个有穷转换集，m>=0，且T和P不相交。
I: T→P[∞]为输入函数，是由位置到转换无序单位组的映射。
O: T→P[∞]为输出函数，是由转换到位置无序单位组的映射。
- Petri网的标记是在Petri网中权标(token)的分配。
- 通常，当每个输入位置所拥有的权标数大于等于从该位置到转换的线数时，就允许转换。

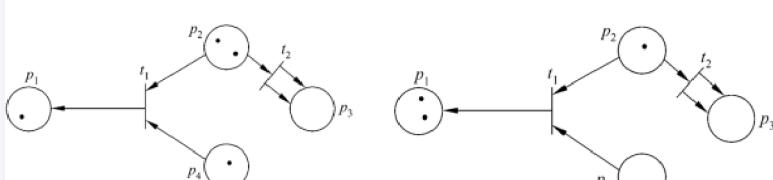


图4.6 带标记的Petri网

图4.6所示Petri网的标记为

(1, 2, 0, 1)，t₁和t₂都可以被激发。

- 假设t₁被激发了，则结果如图4.7所示，标记为(2, 1, 0, 0)。此时，只有t₂可以被激发。
- 如果t₂也被激发了，则权标从P₂中移出，两个新权标被放在P₃上，结果如图4.8所示，标记为(2, 0, 2, 0)。

图4.7 图4.6的Petri网在转换t1被激发后的情况

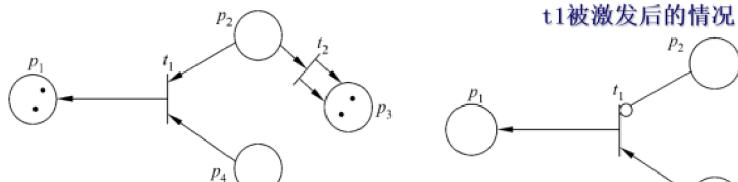


图4.8 图4.7的Petri网在转换t2被激发后的情况

图4.9 含禁止线的Petri网

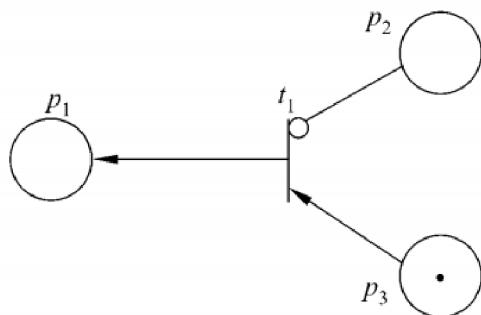
- 更形式化的说，Petri网 $C = (P, T, I, O)$ 中的标记 M ，是有一组位置 P 到一组非负数的映射：

$$M: \rightarrow P \{0, 1, 2, \dots\}$$

这样，带有标记的Petri网成为一个五元组 (P, T, I, O, M) 。

- 对Petri网的一个重要扩充是加入禁止线。

当每个输入线上至少有一个标权，而禁止线上没有标权，因此转换 t_1 可以被激发。



电梯例子

现在把Petri网应用于上一节讨论过的电梯问题。当用Petri网表示电梯系统的规格说明时，每个楼层用一个位置 F_f 代表 ($1 \leq f \leq m$)，在Petri网中电梯是用一个权标代表的。在位置 F_f 上有权标，表示在楼层 f 上有电梯。

1. 电梯按钮

电梯问题的第一个约束条件描述了电梯按钮的行为，现在复述一下这个约束条件。

第一条约束 C1：每部电梯有 m 个按钮，每层对应一个按钮。当按下一个按钮时该按钮指示灯亮，指示电梯移往相应的楼层。当电梯到达指定的楼层时，按钮将熄灭。

为了用Petri网表达电梯按钮的规格说明，在Petri网中还必须设置其他的位置。电梯中楼层 f 的按钮，在Petri网中用位置 EB_f 表示 ($1 \leq f \leq m$)。在 EB_f 上有一个权标，就表示电梯内楼层 f 的按钮被按下了。

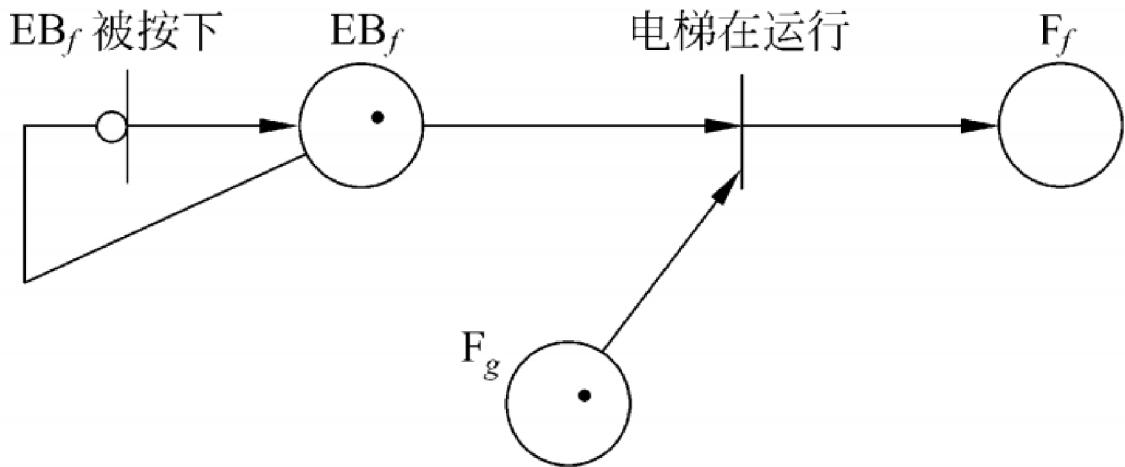


图4.10 Petri网表示的电梯按钮

电梯按钮只有在第一次被按下时才会由暗变亮，以后再按它则只会被忽略。图4.10所示的Petri网准确地描述了电梯按钮的行为规律。首先，假设按钮没有发亮，显然在位置EBf上没有权标，从而在存在禁止线的情况下，转换“EBf被按下”是允许发生的。假设现在按下按钮，则转换被激发并在EBf上放置了一个权标，如图4.10所示。以后不论再按下多少次按钮，禁止线与现有权标的组合都决定了转换“EBf被按下”不能再被激发了，因此，位置EBf上的权标数不会多于1。

假设电梯由g层驶向f层，因为电梯在g层，如图4.10所示，位置Fg上有一个权标。由于每条输入线上各有一个权标，转换“电梯在运行”被激发，从而EBf和Fg上的权标被移走，按钮EBf被关闭，在位置Ff上出现一个新权标，即转换的激发使电梯由g层驶到f层。

2. 楼层按钮

在第二个约束条件中描述了楼层按钮的行为。

第二条约束C2：除了第一层与顶层之外，每个楼层都有两个按钮，一个要求电梯上行，另一个要求电梯下行。这些按钮在按下时发亮，当电梯到达该层并将向指定方向移动时，相应的按钮才会熄灭。

在Petri网中楼层按钮用位置 FB_f^u 和 FB_f^d 表示，分别代表f楼层请求电梯上行和下行的按钮。底层的按钮为 FB_1^u ，最高层的按钮为 FB_m^d ，中间每一层有两个按钮 FB_f^u 和 FB_f^d ($1 < f < m$)。

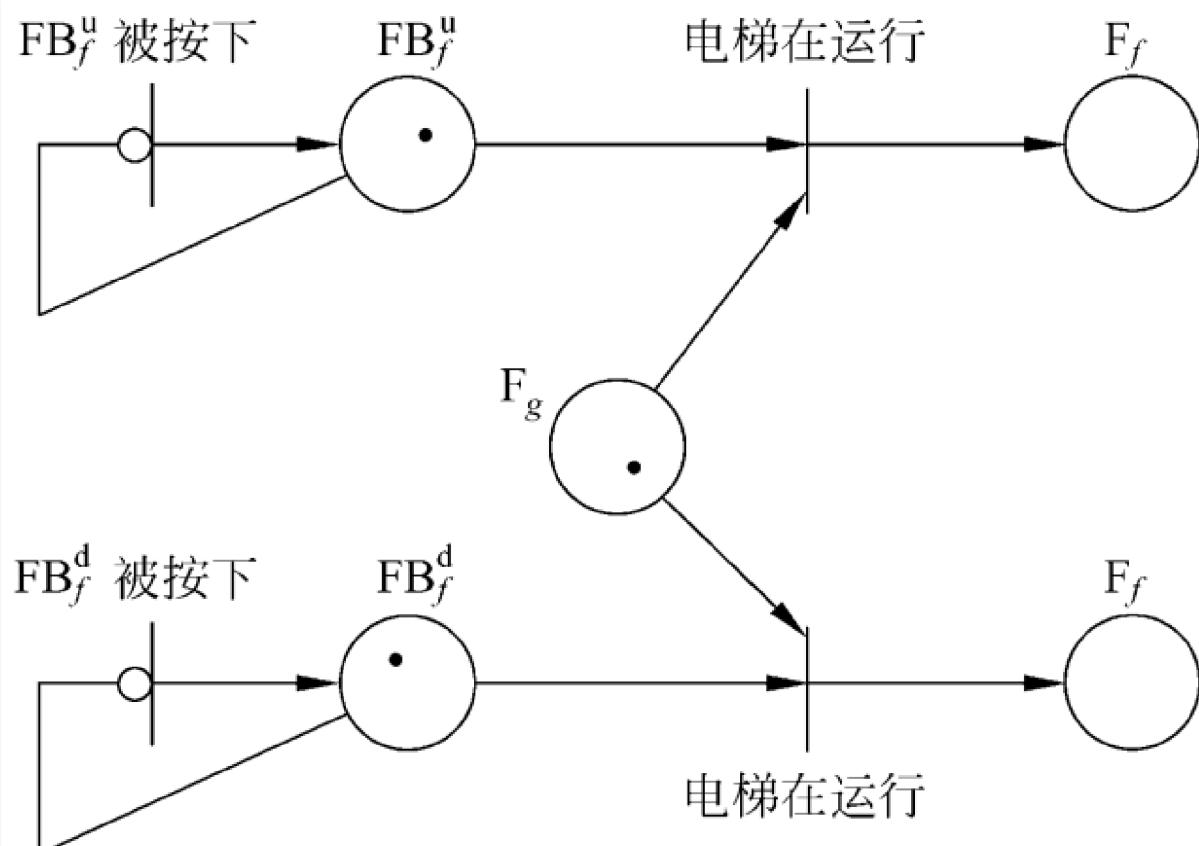


图4.11 Petri网表示楼层按钮

图4.11所示的情况为电梯由g层驶向f层。根据电梯乘客的要求，某一个楼层按钮亮或两个楼层按钮都亮。如果两个按钮都亮了，则只有一个按钮熄灭。图4.11所示的Petri网可以保证，当两个按钮都亮了的时候，只有一个按钮熄灭。但是要保证按钮熄灭正确，则需要更复杂的Petri网模型。

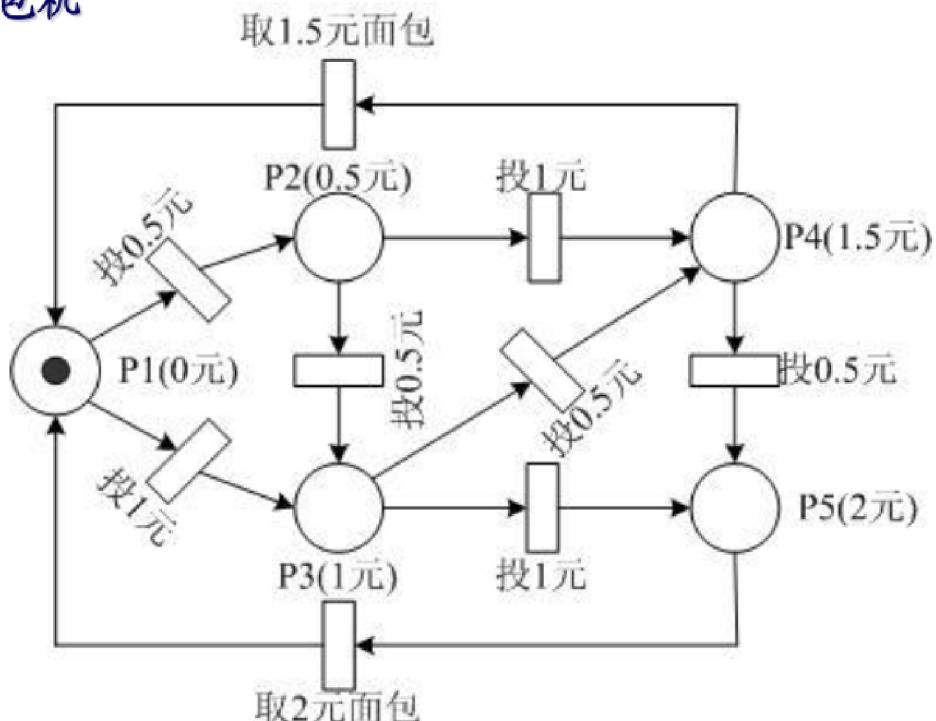
最后，考虑第三条约束。

第三条约束C3：当电梯没有收到请求时，它将停留在当前楼层并关门。

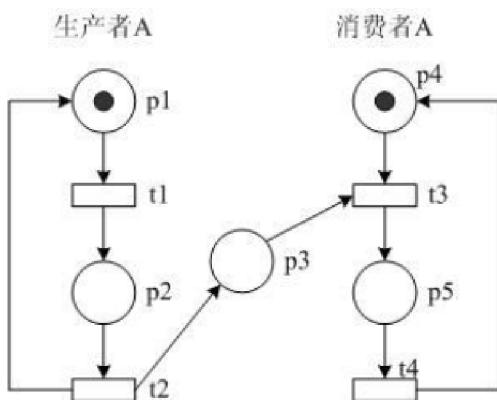
这条约束很容易实现，如图4.11所示，当没有请求(FBf^u 和 FBf^d 上无权标)时，任何一个转换“电梯在运行”都不能被激发。

自动售货机

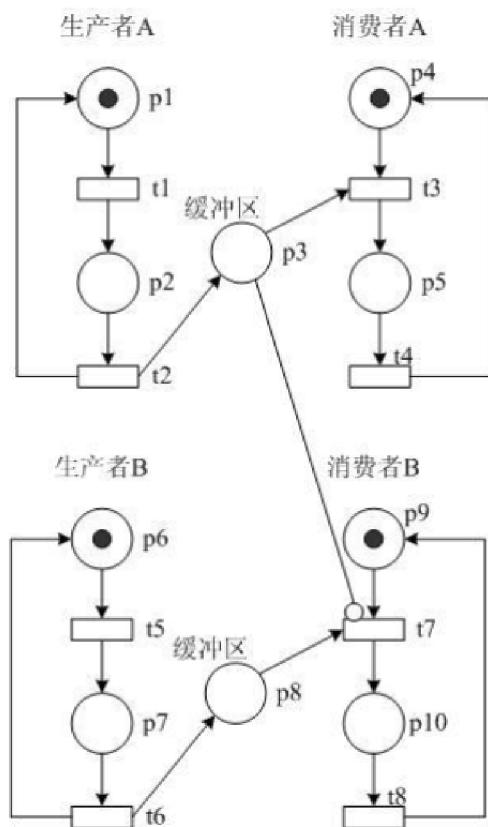
1. 自动售面包机



2. 生产者/消费者系统



其中 p₁ 表示生产者处于就绪；
p₂ 表示生产者正在生产产品；
p₃ 表示仓库； p₄ 表示消费者处于就绪； p₅ 表示消费者正在消费产品。



第五章 总体设计

5.1 软件设计阶段

- 概要/结构设计：将软件需求转化为数据结构和软件的系统结构；
- 详细/过程设计：即过程设计，通过对结构表示进行细化，得到软件的详细的数据结构和算法；

5.2 总体设计的任务及工具

从全局的高度，花较少的成本，从抽象的层次上分析对比多种可能的系统实现方案和软件结构，从中选出最佳方案和最合理的软件结构，从而用较低的成本开发出较高质量的软件系统。

- 结构设计：确定程序由哪些模块组成及几个模块之间的关系；
- 过程设计：确定每个模块的处理过程；
- **结构设计是总体设计阶段的任务，过程设计是详细设计阶段的任务；**

描绘软件结构的图形工具：层次图、HIPO图、结构图

5.3 模块独立性

模块独立性可由耦合、内聚两个定性标准度量；

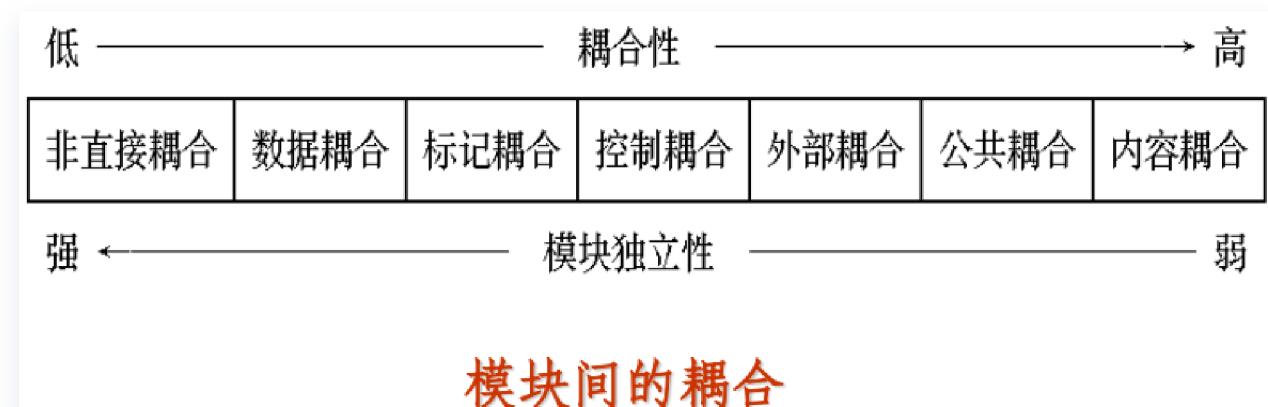
内聚

内聚是模块功能强度的度量；它是衡量一个模块内部各个元素彼此结合的紧密程度；

耦合

耦合是模块之间的互相连接的紧密程度的度量；它是衡量不同模块彼此间互相依赖的紧密程度；

模块独立性比较强的模块应是高内聚低耦合的模块；



降低模块间耦合度

1. 尽量使用数据耦合，少用控制耦合，限制公共耦合的范围，避免使用内容耦合；
2. 降低接口的复杂度；

5.4 启发式规则

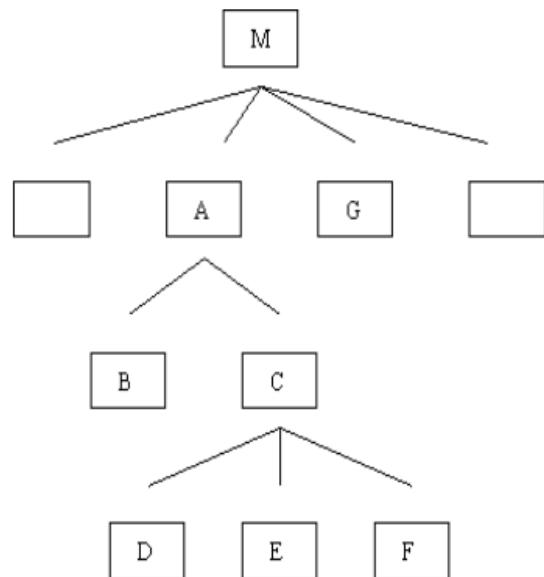
相关概念

- 深度：软件结构中的控制层数；
- 宽度：同一层上模块总数的最大值；
- 扇出：一个模块直接控制（调用）的模块数目；
- 扇入：一个模块有多少个上级模块调用它；

- 模块控制域

4. 模块的作用域应该在控制域之内

模块的作用域：该模块本身以及所有直接或间接从属于它的模块集合。模块A的控制域是A、B、C、D、E、F。



5. 力争降低模块接口的复杂程度

5.5 面向数据流的设计方法

相关概念

设计是一个将信息需要转换成数据结构、程序结构和过程性表示的多步骤过程；



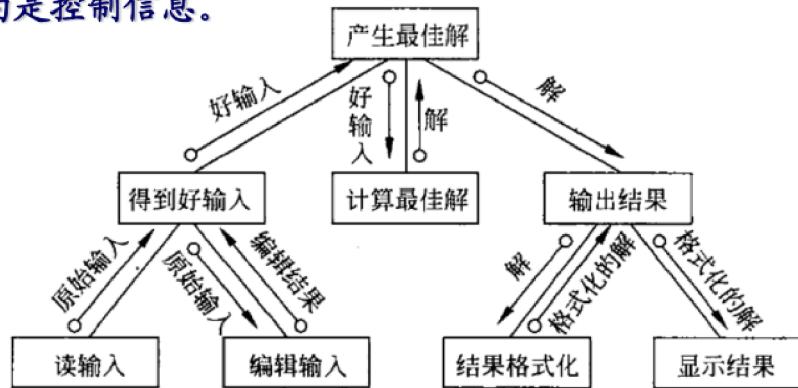
变换分析

将具有变换特点的数据流图按预先确定的模式映射成软件结构；

5.4.2 结构图

结构图是进行软件结构设计的另一个有力工具。

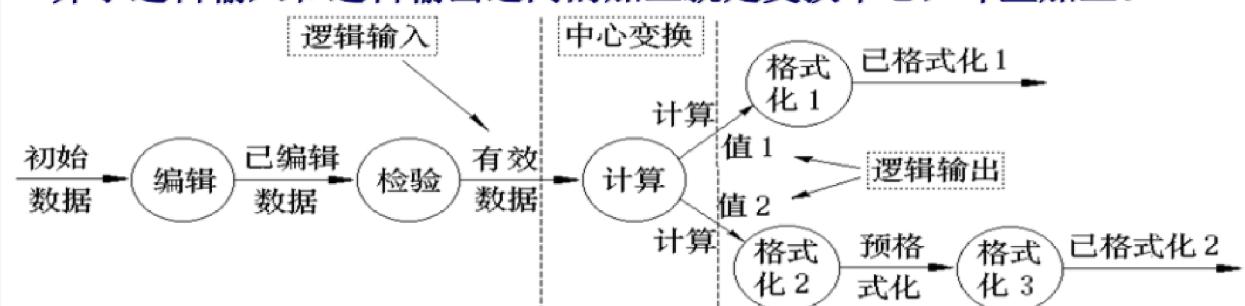
按惯例总是图中位于上方的方框代表的模块调用下方的模块，用带注释的箭头表示模块调用过程中来回传递的信息。如果希望进一步表明传递信息是数据还是控制信息，用尾部形状区分：尾部是空心圆表示传递的是数据，实心圆表示传递的是控制信息。



变换分析设计步骤：

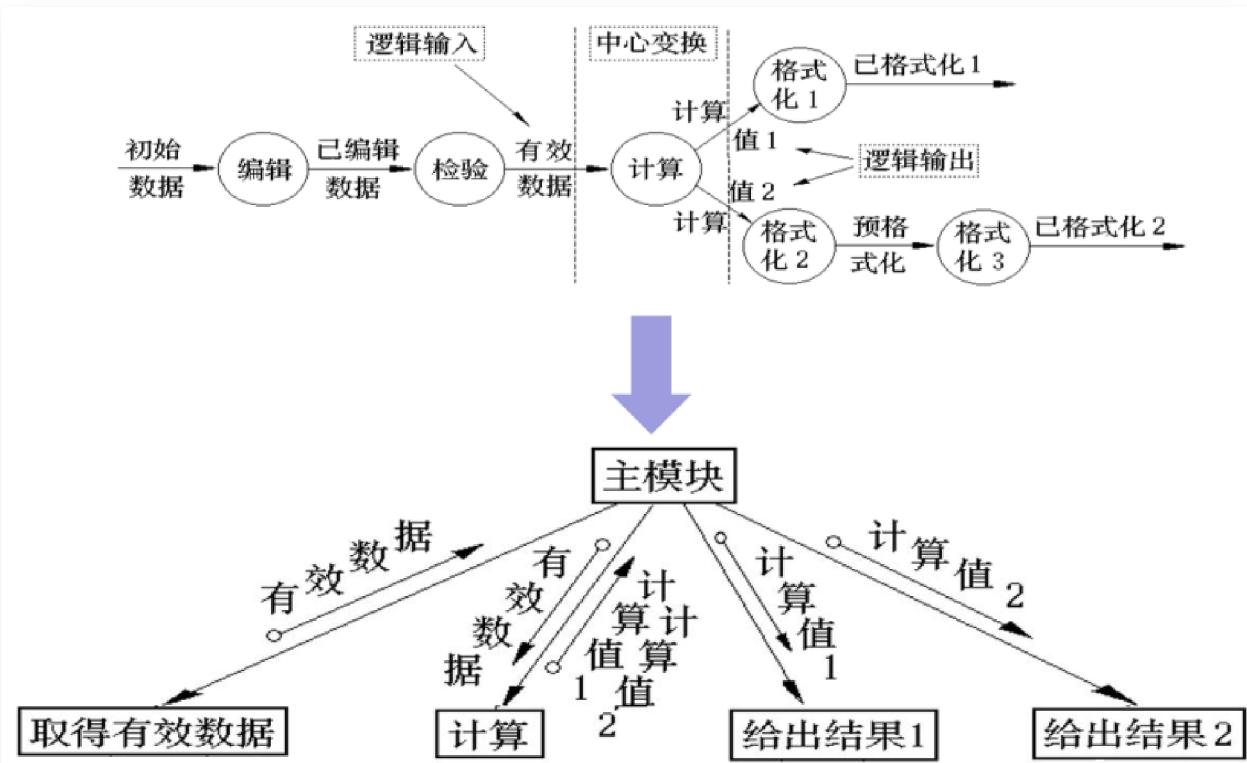
①确定DFD中的变换中心、逻辑输入和逻辑输出：

- 从物理输入端开始，沿着数据流方向向系统中心寻找，直到有这样 的数据流，它不能再被看做是系统的输入，则它的前一个数据流 是系统的逻辑输入。
- 从物理输出端开始，逆数据流方向向中间移动，可以确定系统的逻辑输出。
- 介于逻辑输入和逻辑输出之间的加工就是变换中心，即主加工。



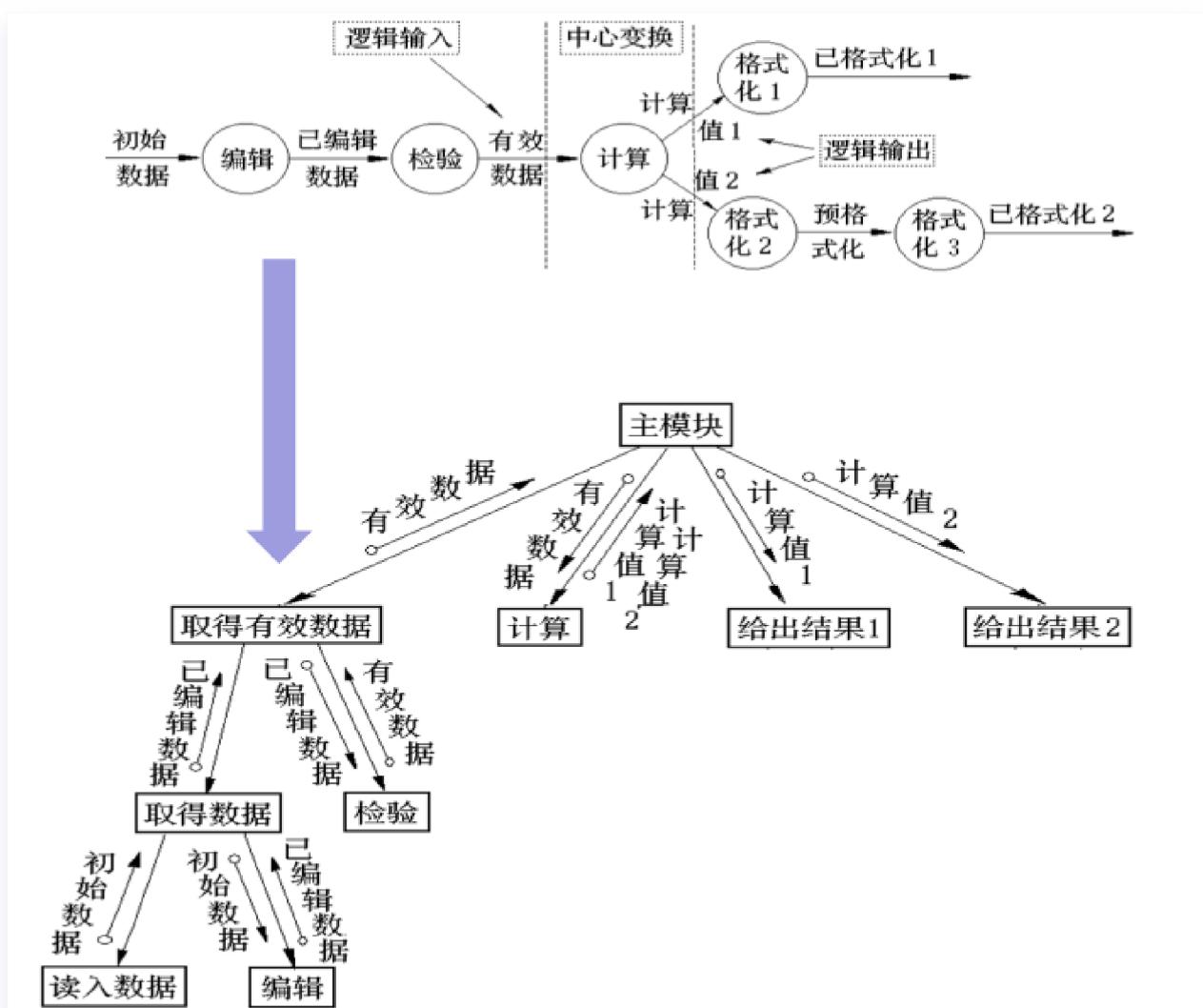
② 设计软件结构的顶层和第一层——变换结构：

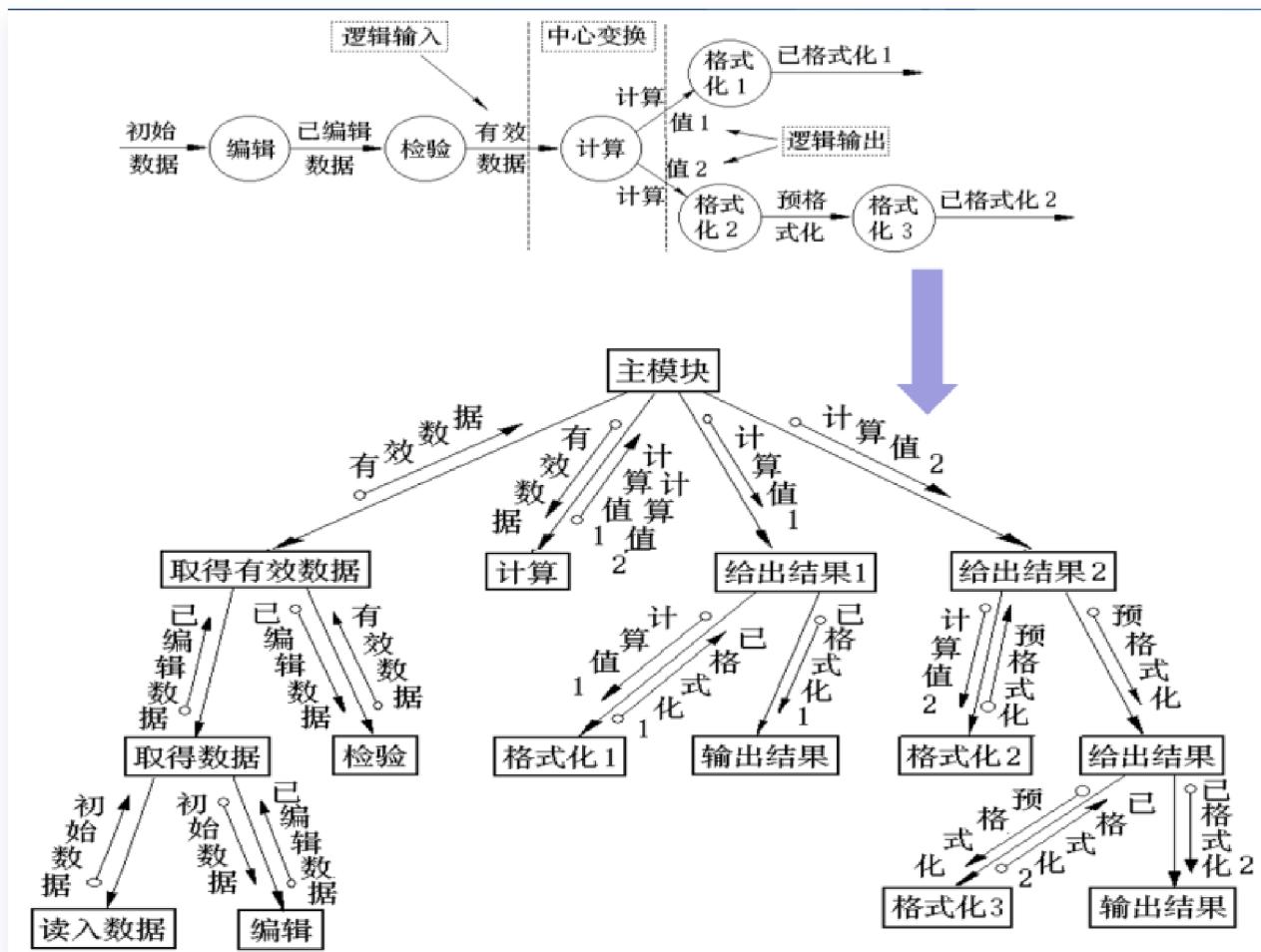
- 顶层即主模块，主要完成所有模块的控制。
- 第一层至少有**3**个功能模块：输入、输出和变换模块，即为逻辑输入设计一个**输入模块**，其功能是为顶层模块提供相应数据，为逻辑输出设计一个**输出模块**，其功能是输出顶层模块的信息，为变换中心设计一个**变换模块**，其功能是将逻辑输入进行变换加工，然后逻辑输出。



③ 设计中、下层模块。对第一层的输入、变换、输出模块自顶向下逐层分解。

- 输入模块下属模块的设计：可设计两个下属模块，一个接收，一个转换。
- 输出模块下属模块的设计：可设计两个下属模块，一个转换，一个发送。
- 变换模块下属模块的设计：按照模块独立性原则来组织其结构，一般对每个基本加工建立一个功能模块。





④ 设计的优化:

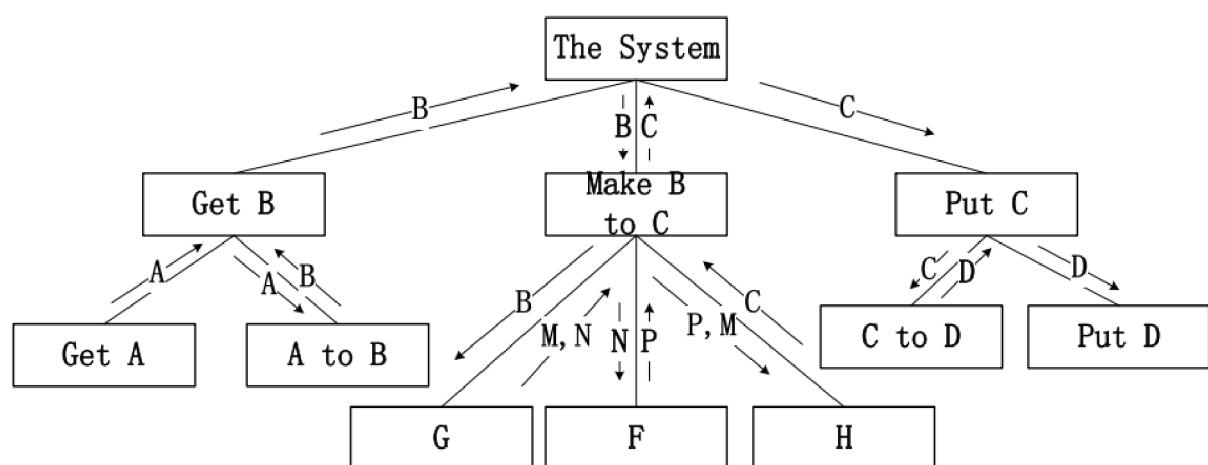
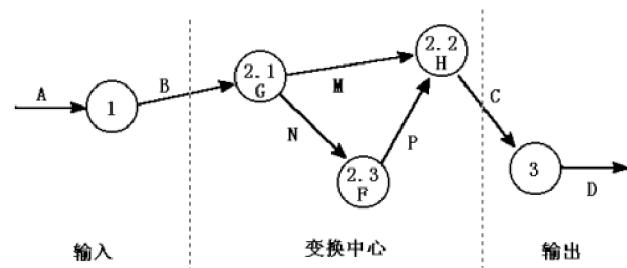
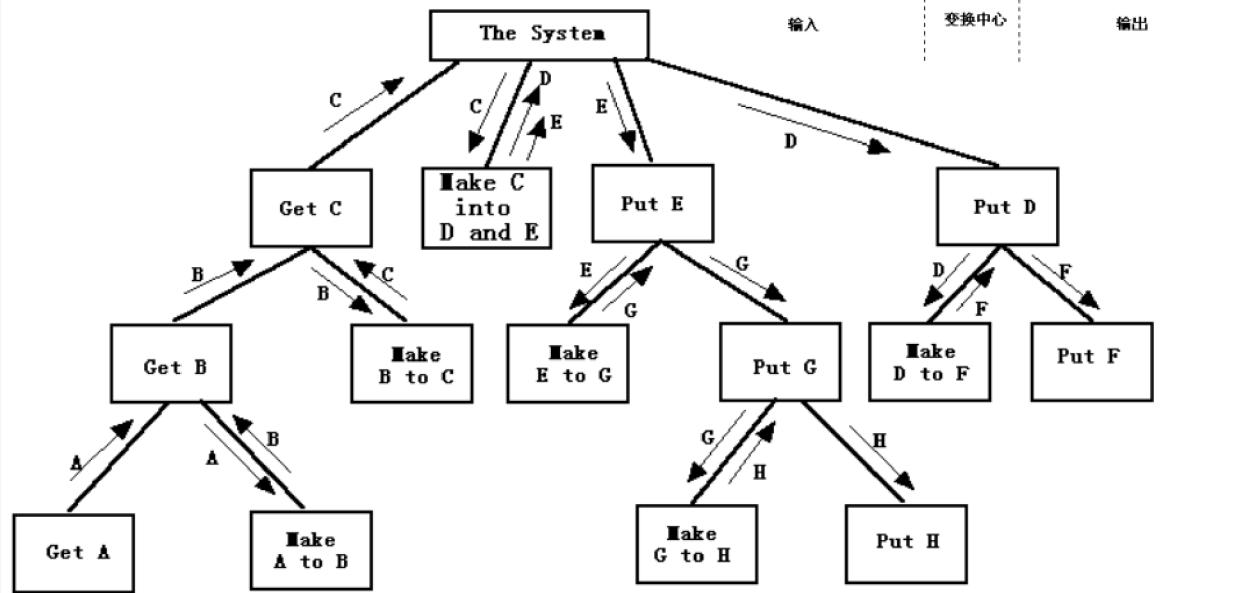
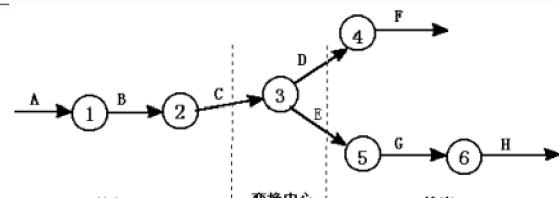
- **输入部分求精:** 为每个物理输入设置专门模块，其他输入模块与转换数据模块可适当合并。
- **输出部分求精:** 为每个物理输出设置专门模块，其他输出模块与转换数据模块可适当合并。
- **变换部分求精:** 根据设计准则，对模块进行合并或调整。

事务分析

事务流：存在某种作业数据流，可以引发一个或多个处理，这些处理能够完成该作业要求的功能，这种数据流即是事务流；

例题

请检查自己的结果！



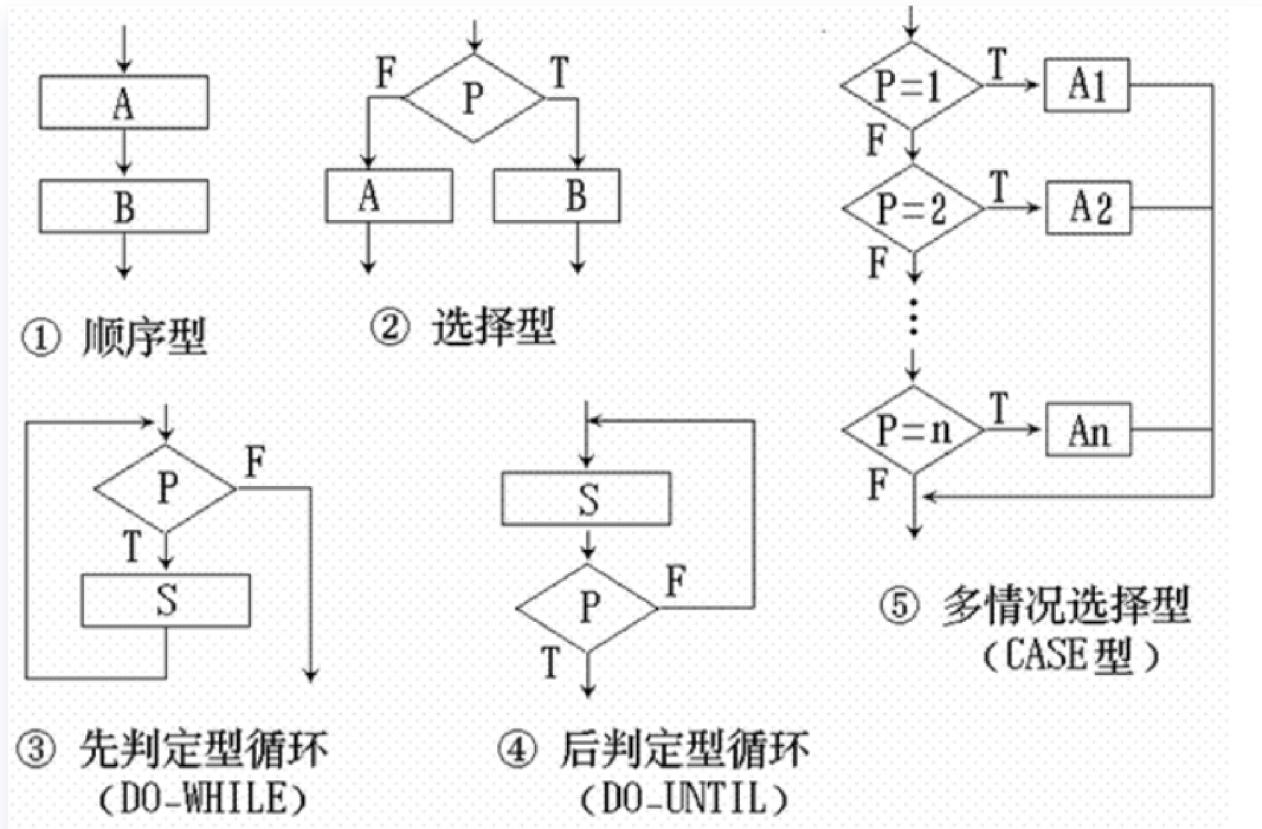
6.1 结构程序设计

程序的三种基本控制结构：顺序控制结构、分支控制结构、循环控制结构；

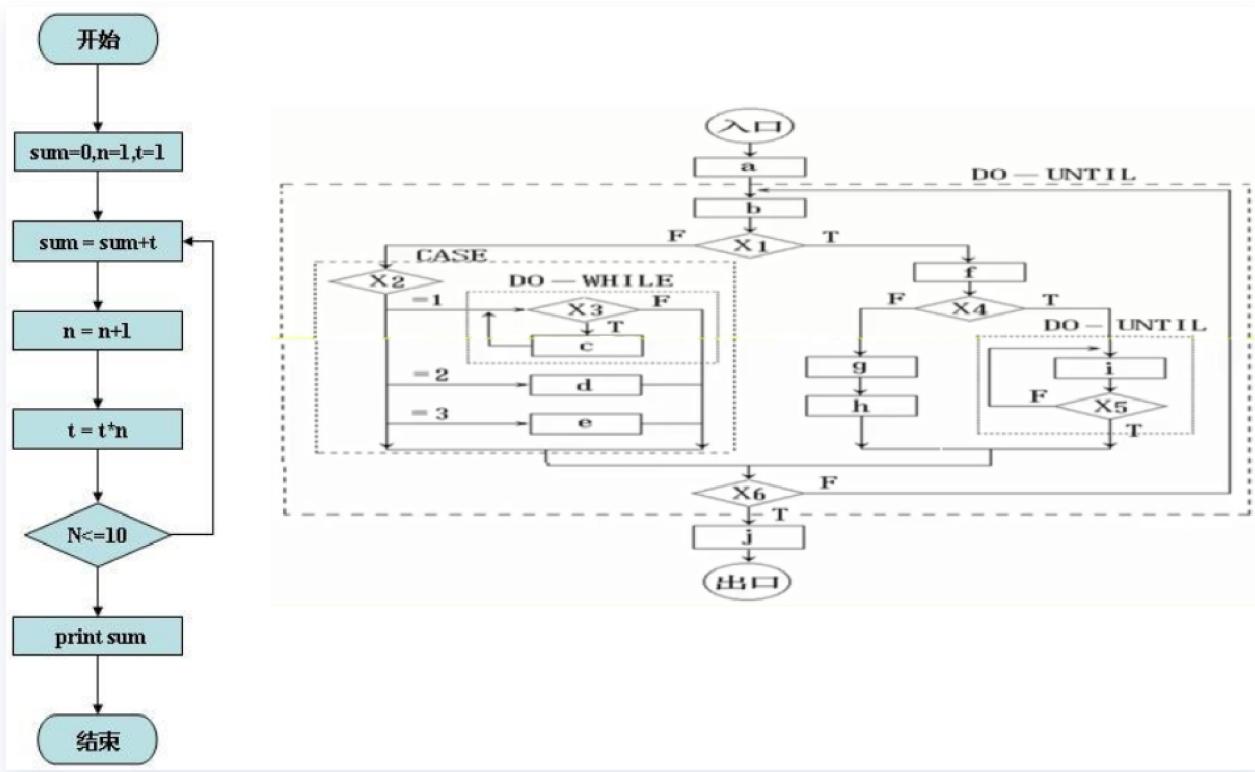
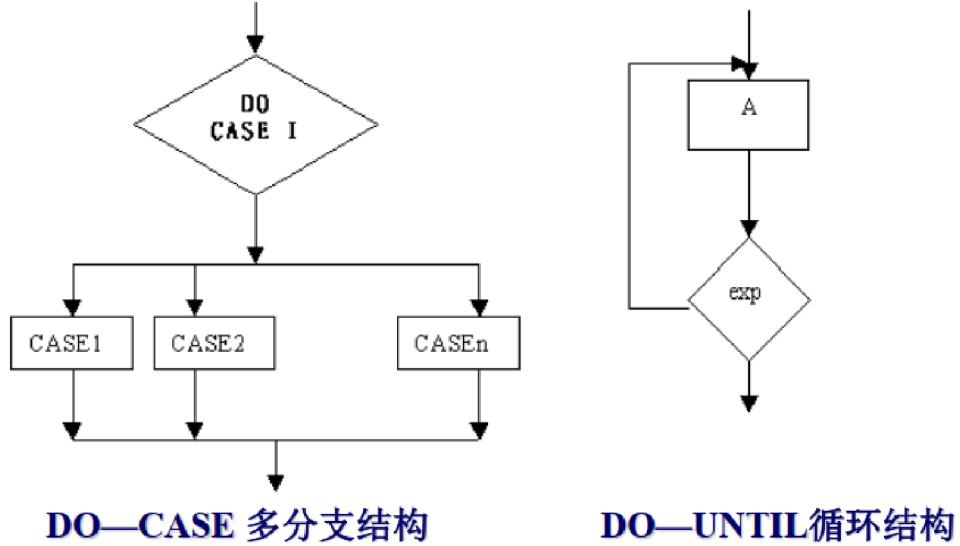
★ 软件详细设计过程使用的工具：程序流程图、盒图、PAD图、判定表、判定树。

6.2 程序流程图

又称程序框图，程序流程图中的箭头代表控制流而不是数据流；

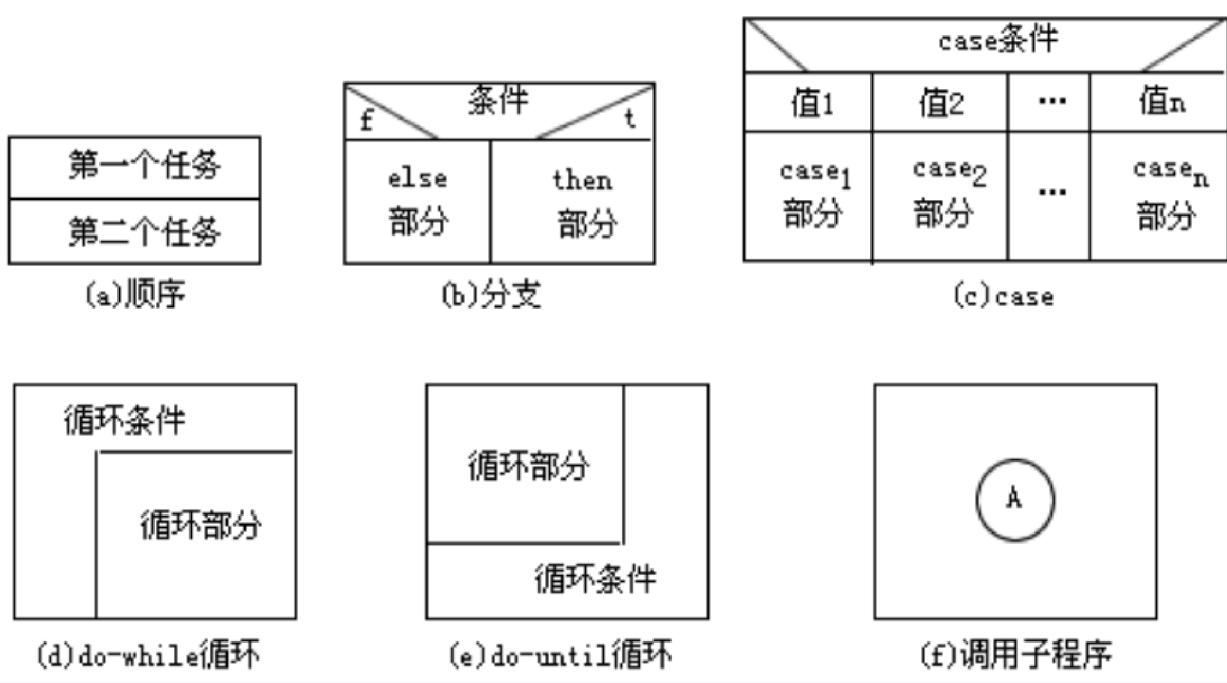


在三种基本控制结构中，可以实现任何单入口单出口的程序，但从实际使用方便起见，常常允许使用下面二种控制。

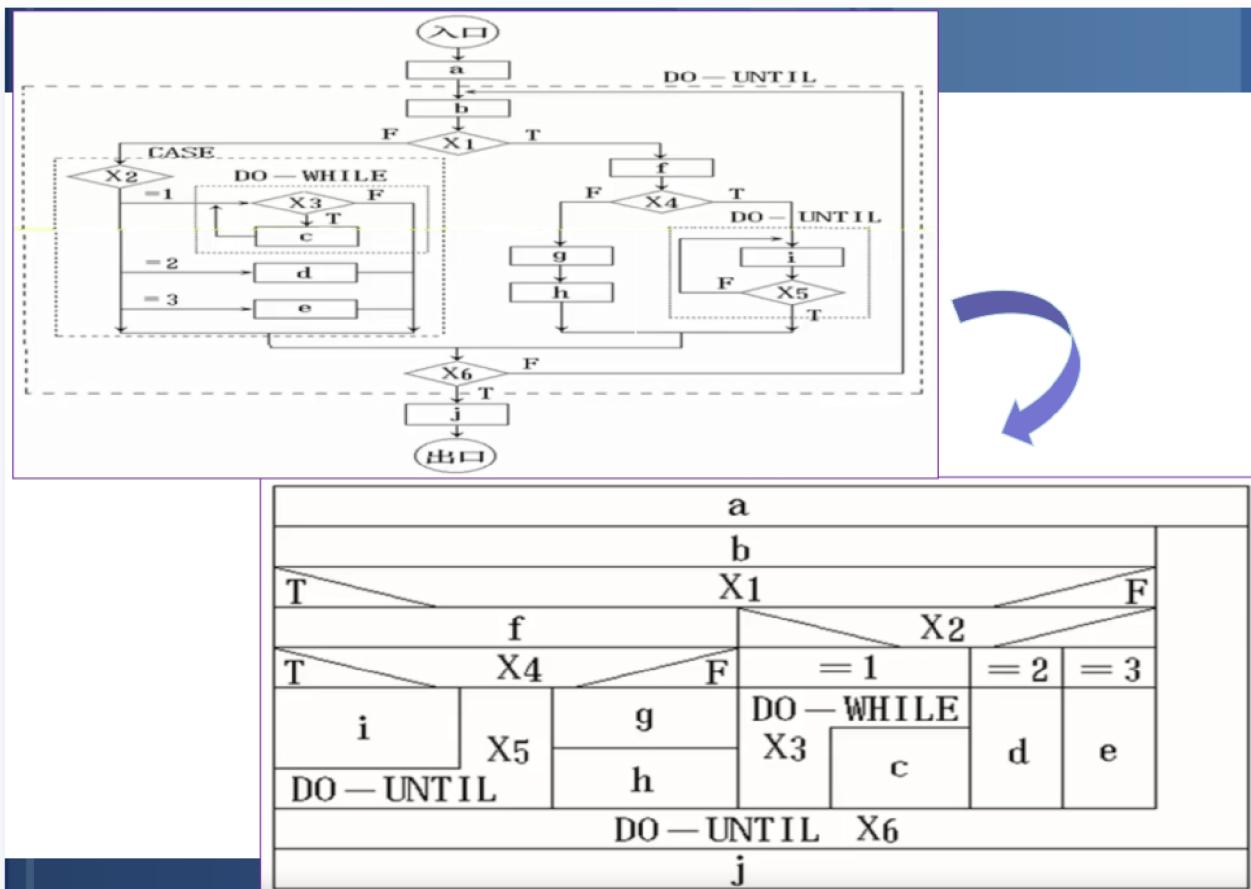


6.3 盒图 (N-S图)

- 功能域明确，易确定局部和全程数据的作用域；
- 易表现嵌套关系，也可表示模块的层次结构；



程序流程图到盒图的转换

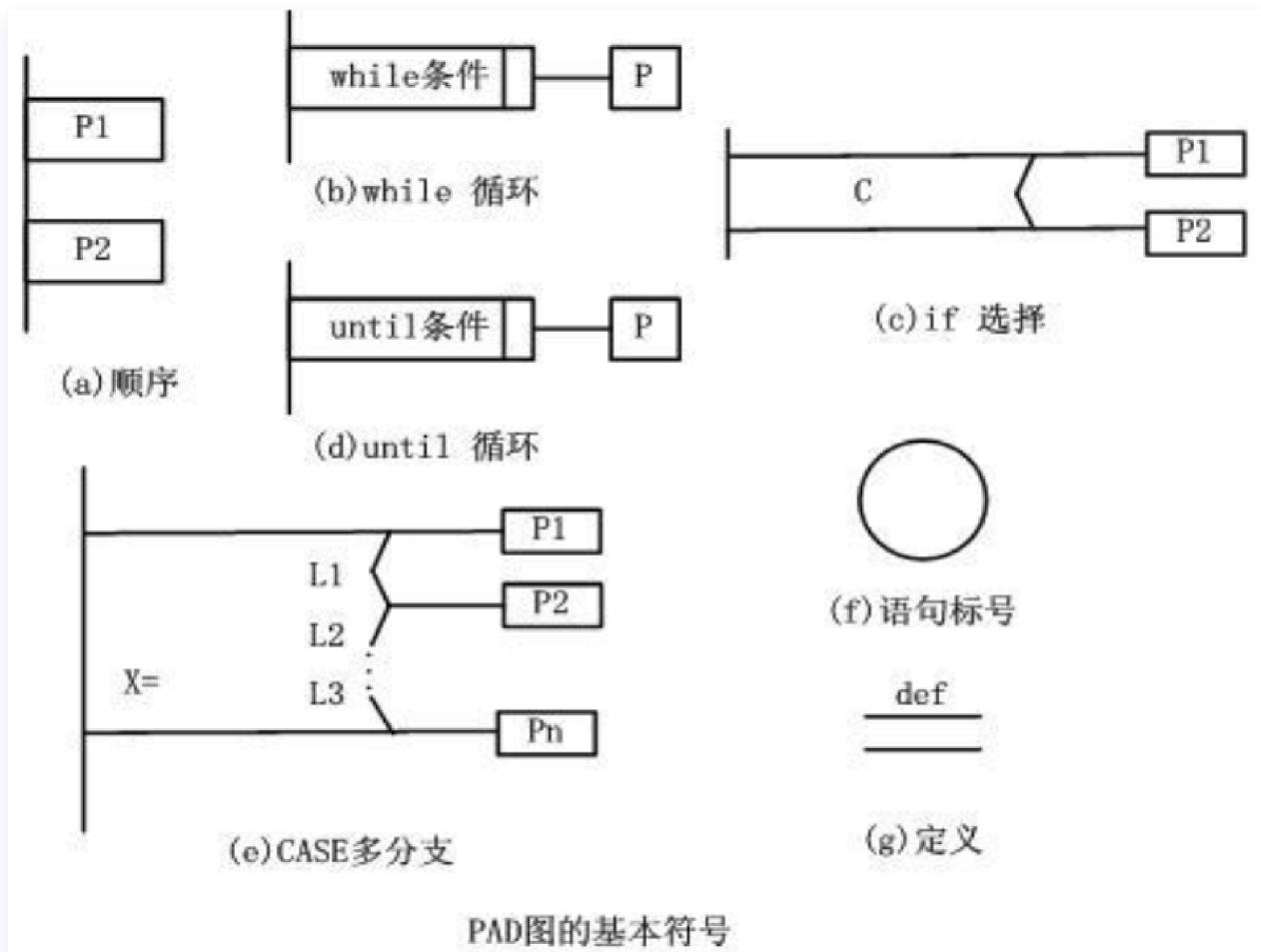


6.4 PAD 图 (Problem Analysis Diagram)

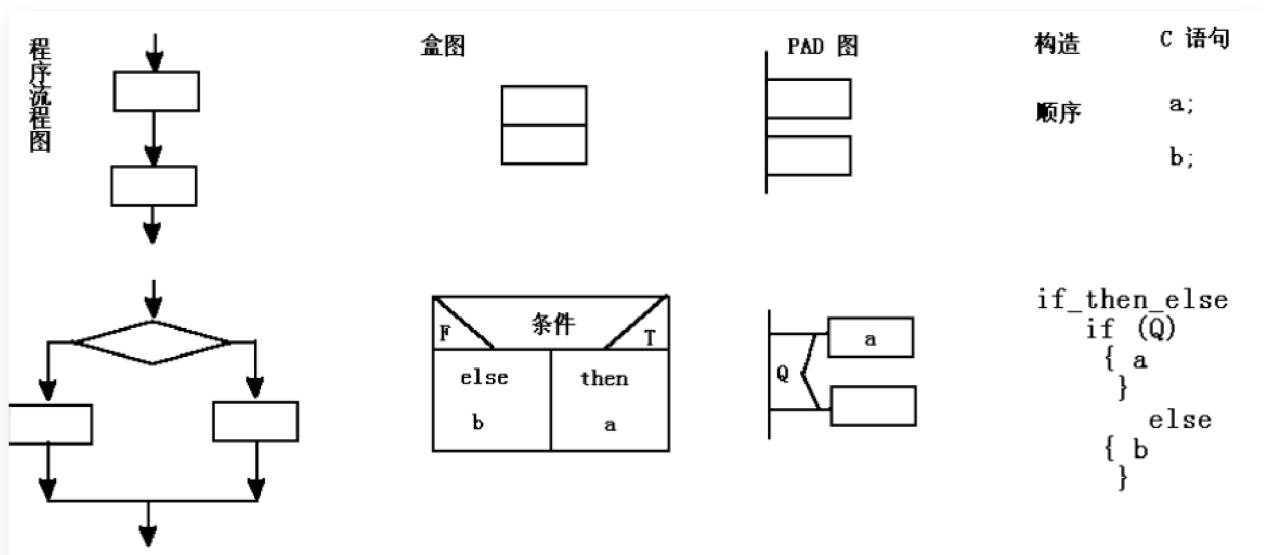
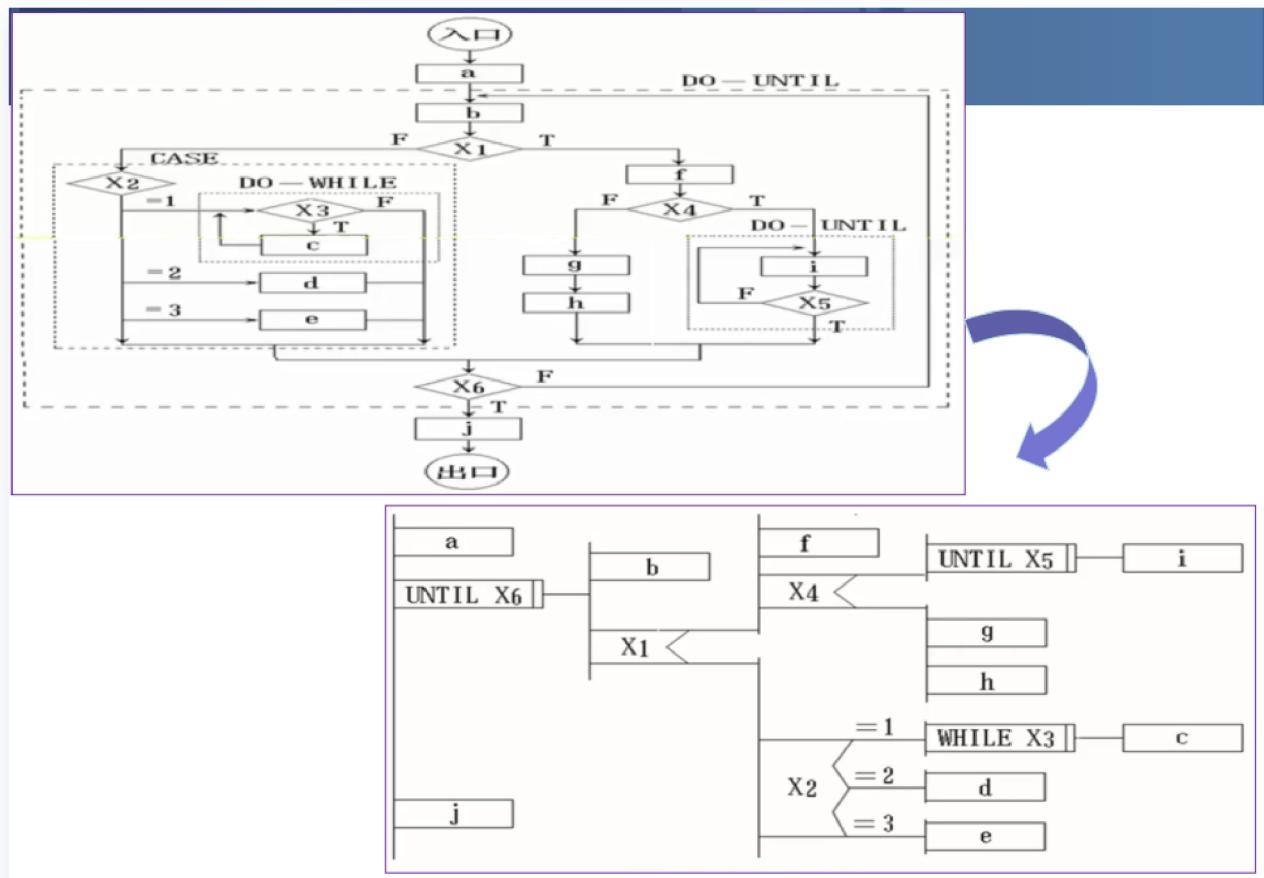
- 二维树形结构的图表示程序的控制流；
- 易翻译成程序代码；

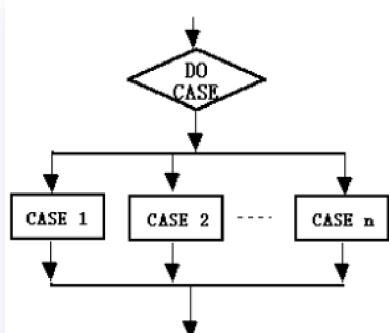
- 优点:

- PAD符号设计出的程序是程序化程序，程序结构清晰；
- PAD 图表现的程序逻辑易懂易记，程序自上而下，从左到右顺序执行；
- 易将PAD图转换成高级程序语言源程序；

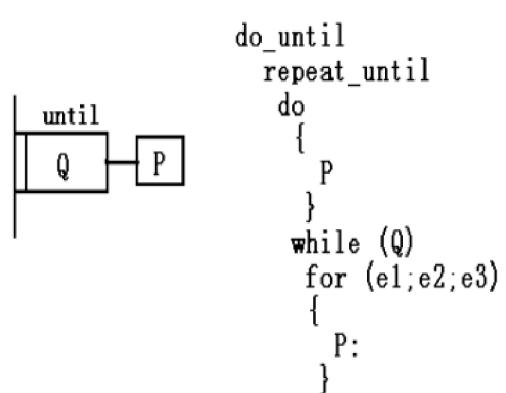
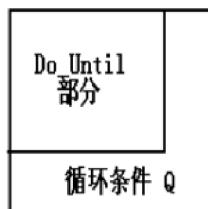
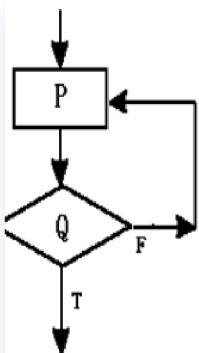
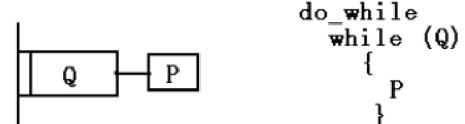
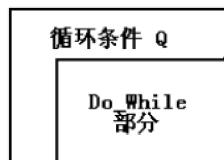
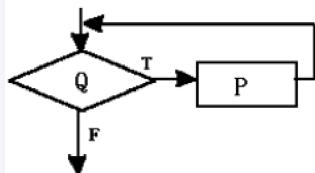
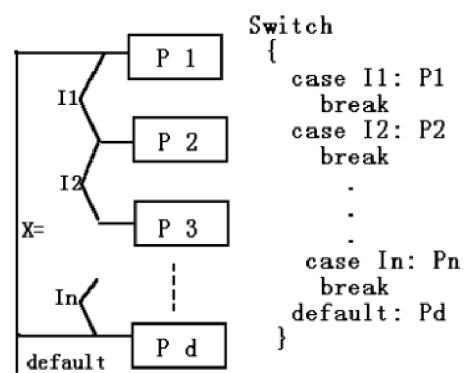


程序流程图到 PAD 图的转换





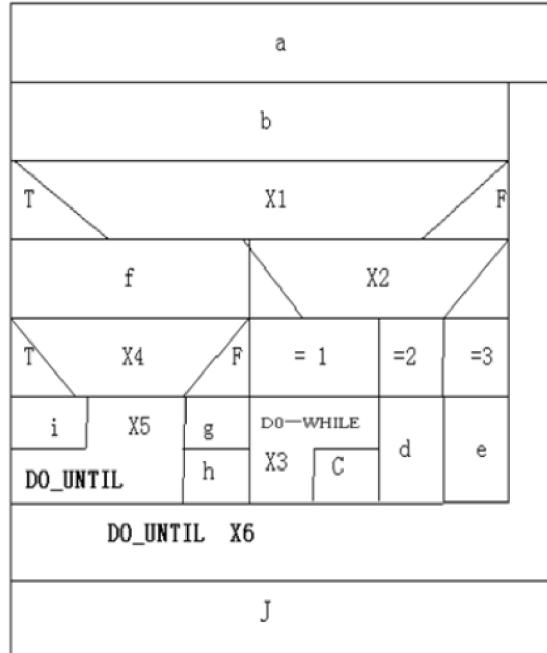
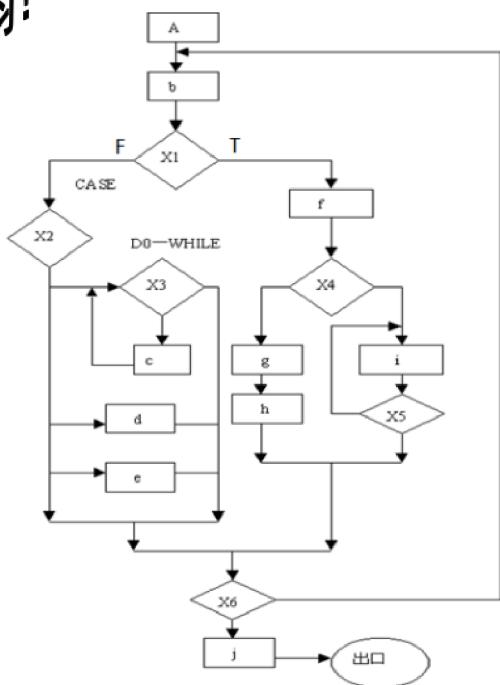
CASE 条件			
值 1	值 2	...	值 n
case1 部分	case2 部分	case n 部分



6.5 实例

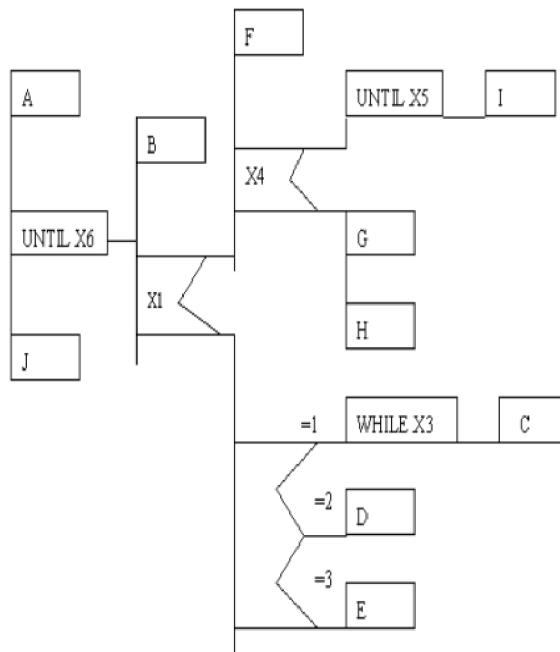
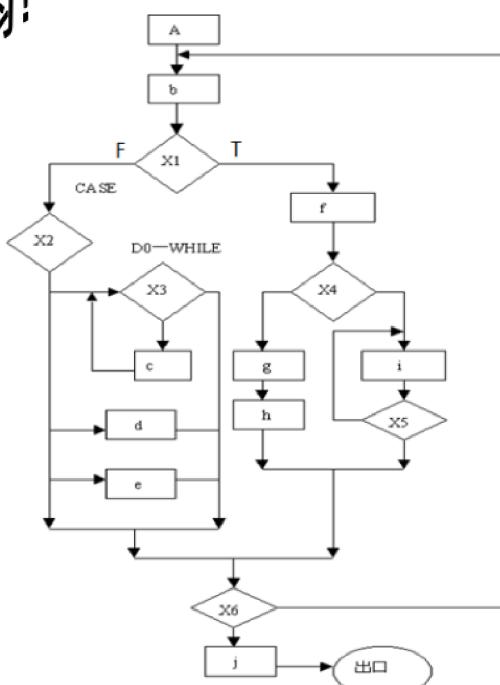
程序流程图 转换成 盒图

课堂练习!



程序流程图 转换成 PAD图

课堂练习!



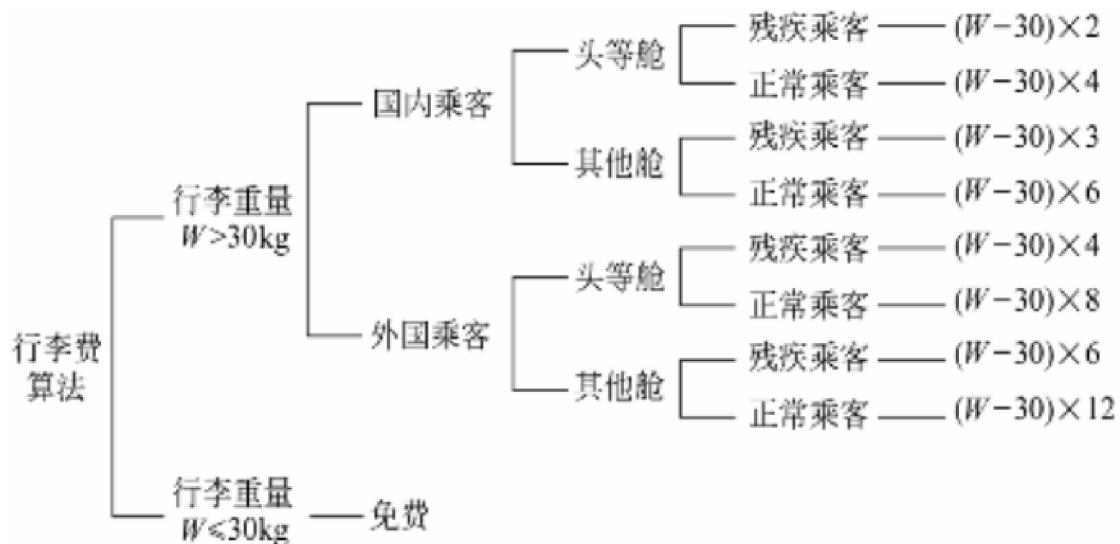
6.6 判定表与判定树（了解）

- 判定表更加简洁，判定树的数据元素重复较多，尤其是接近树的叶端；

判定表表示计算行李费的算法

乘客条件 收费标准(元) 标准	W<=30	W>30							
		国内乘客				国外乘客			
		一般舱		头等舱		一般舱		头等舱	
		残疾	普通	残疾	普通	残疾	普通	残疾	普通
免费	✓					✓			
$2 \times (w-30)$		✓							
$3 \times (w-30)$				✓					
$4 \times (w-30)$			✓			✓			
$6 \times (w-30)$					✓			✓	
$8 \times (w-30)$							✓		
$12 \times (w-30)$									✓

用判定树表示计算行李费的算法



6.7 面向数据结构的设计方法

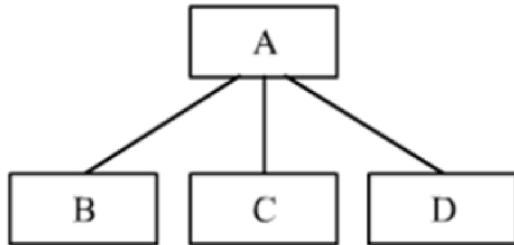
Jackson 图——分析和描述数据结构

数据结构种类（逻辑关系）：顺序，选择和重复；

- 便于表示层次结构，实行自顶向下分解；
- 既能表示数据结构也能表示程序结构；

1. 顺序结构

顺序结构的数据由一个或多个数据元素组成，每个元素按确定次序出现一次。

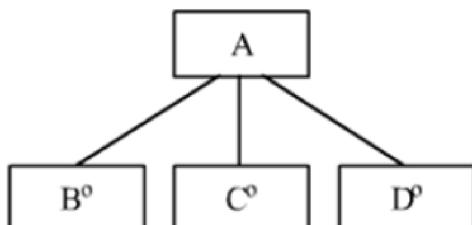


顺序结构

A由B、C、D三个元素顺序组成（每个元素只出现一次，出现的次序依次是B、C和D）

2. 选择结构

选择结构的数据包含两个或多个数据元素，每次使用这个数据时按一定条件从这些数据元素中选一个。

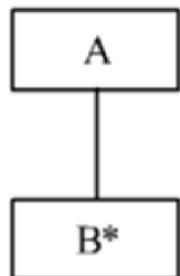


选择结构

根据条件A是B或C或D中的某一个（注意：在B、C和D的右上角有小圆圈标记）

3. 重复结构

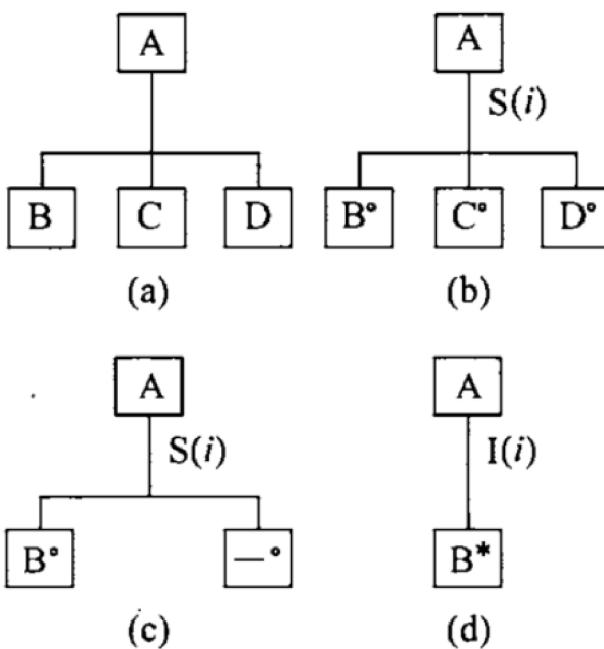
根据使用时的条件由一个数据元素出现零次或多次构成。



重复结构

A由B出现N次 ($N \geq 0$) 组成 (注意: 在B的右上角有星号标记)

改进的Jackson图



(a) **顺序结构**, B、C、D中任一个都不能是选择出现或重复出现的数据元素(即, 不能是右上角有小圆圈或星号标记的元素);

(b) **选择结构**, S右面括号中的数字*i*是分支条件的编号;

(c) **可选结构**, A或者是元素B或者不出现(可选结构是选择结构的一种常见的特殊形式);

(d) **重复结构**, 循环结束条件的编号为*i*。

Jackson 结构程序设计方法基本上由5个步骤组成：

- 1、分析并确定输入数据和输出数据的逻辑结构，并用**Jackson**图描绘这些数据结构。
- 2、找出输入数据结构和输出数据结构中有对应关系的数据单元。
- 3、用下述三条规则从描绘数据结构的**Jackson**图导出描绘程序结构的**Jackson**图。

第一，为每对有对应关系的数据单元，按照它们在数据结构图中的层次在程序结构图的相应层次画一个处理框（注意，如果这对数据单元在输入数据结构和输出数据结构中所处的层次不同，则和它们对应的处理框在程序结构图中所处的层次与它们之中在数据结构图中层次低的那个对应）。

第二，根据输入数据结构中剩余的每个数据单元所处的层次，在程序结构图的相应层次分别为它们画上对应的处理框。

第三，根据输出数据结构中剩余的每个数据单元所处的层次，在程序结构图的相应层次分别为它们画上对应的处理框。

总之，描绘程序结构的**Jackson**图应该综合输入数据结构和输出数据结构的层次关系而导出来。在导出程序结构图的过程中，由于改进的**Jackson**图规定在构成顺序结构的元素中不能有重复出现或选择出现的元素，因此可能需要增加中间层次的处理框。

- 4、列出所有操作和条件（包括分支条件和循环结束条件），并且把它们分配到程序结构图的适当位置。
- 5、用伪码表示程序。

Jackson方法中使用的伪代码和**Jackson**图是完全对应的，下面是和三种基本结构对应的伪代码。

(I)顺序结构

与上面顺序结构对应的伪码如下，其中“**seq**”和“**end**”是关键字

```
A seq  
  B  
  C  
  D  
A end
```

■ (II)选择结构

“**select**”、“**or**”和“**end**”是关键字，**cond1**、**cond2**和**cond3**分别是执行**B**、**C**或**D**的条件；

```
A select cond1  
  B  
A or cond2  
  C  
A or cond3  
  D  
A end
```

■ (III)重复结构

“**iter**”、“**until**”、“**while**”和“**end**”是关键字（重复结构有**until**和**while**两种形式），**cond**是条件；

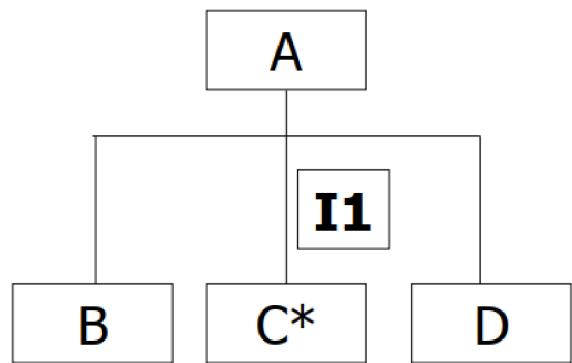
```
A iter until(或while) cond  
  B  
A end
```

下面看一个例子：在实际工作中，会遇到不同结构组合的情况。如图：

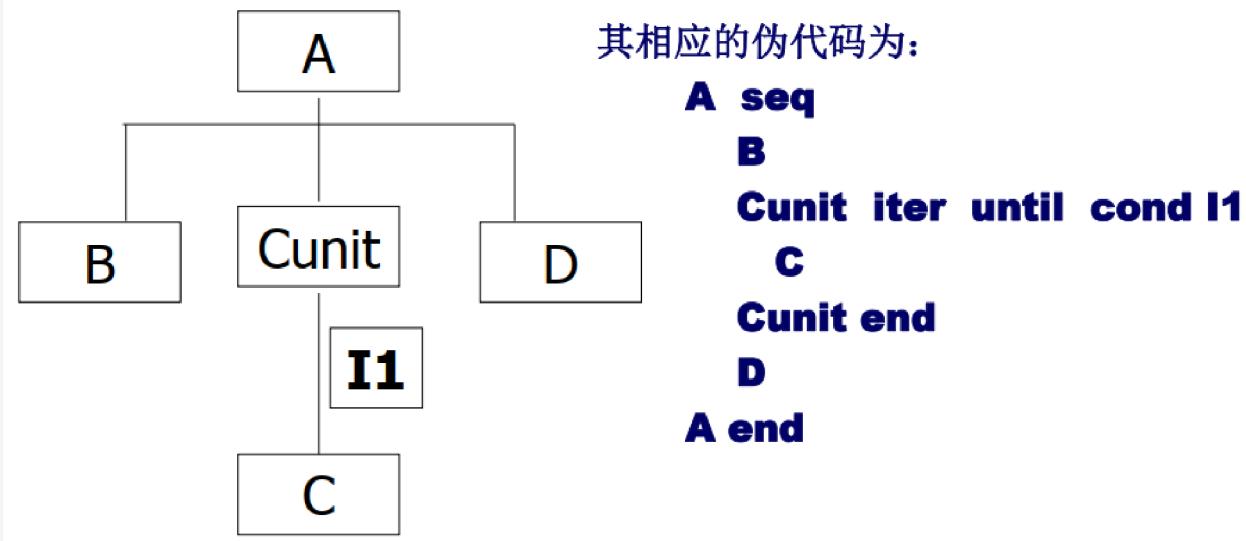
■ 下面两种伪代码：

(a) **A seq**
 B
 C iter until cond I1
 D
A end

(b) **A iter until condI1**
 B
 C
 D
A end



从上图可见，执行是顺序地执行和的处理框，而框要根据条件执行**0次或多次**。上面两种分析（a）是正确的。但是，这种画法容易引起混淆，故可以改为：



Jackson 方法举例

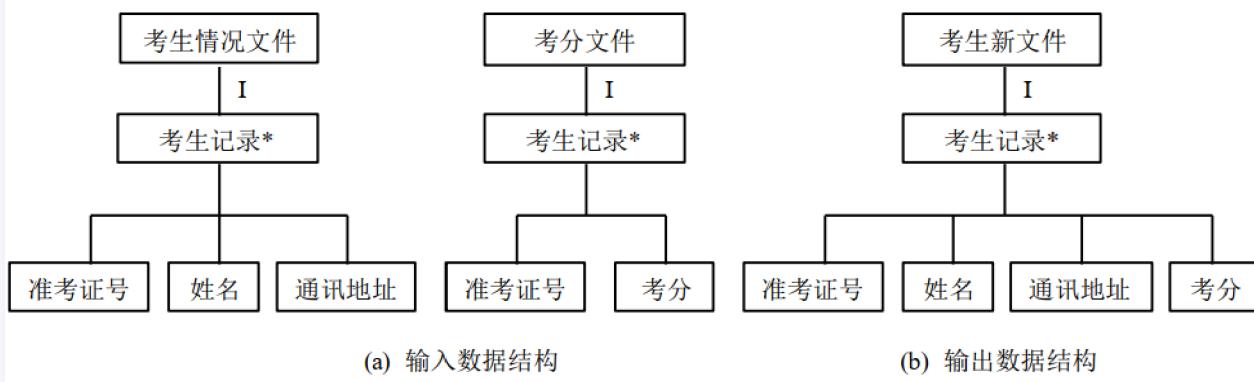
【例】高考后将考生的基本情况文件（简称**考生基本情况文件**）和考生高考成绩文件（简称**考分文件**）合并成一个新文件（简称**考生新文件**）。考生基本情况文件和考分文件都是由**考生记录**组成的。

为简便起见，考生基本情况文件中的考生记录的内容包括：**准考证号、姓名、通讯地址**。考分文件中的考生记录的内容包括：**准考证号和各门考分**。

合并后的考生新文件自然也是由考生记录组成，内容包括：**准考证号、姓名、通讯地址和各门考分**。

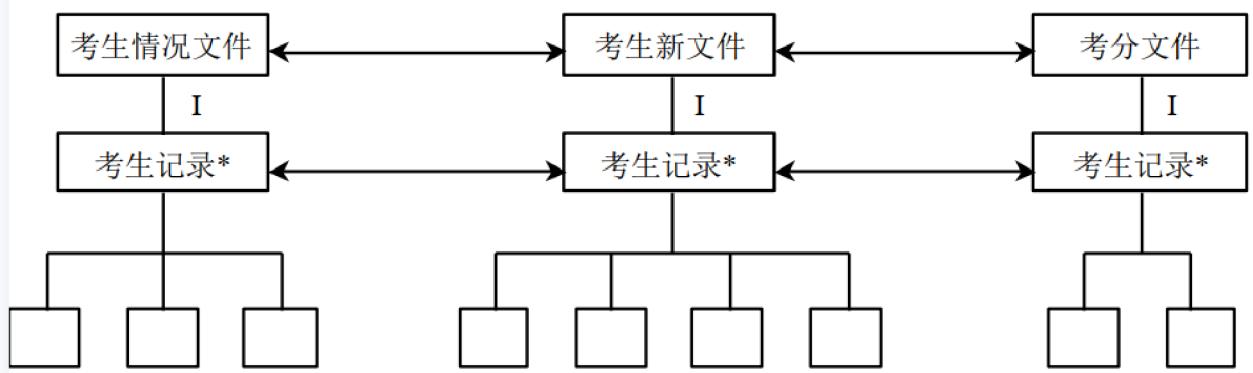
第一步 数据结构表示

对要求解的问题进行分析，确定**输入数据**和**输出数据**的逻辑结构，并用**Jackson**图描述这些数据结构。



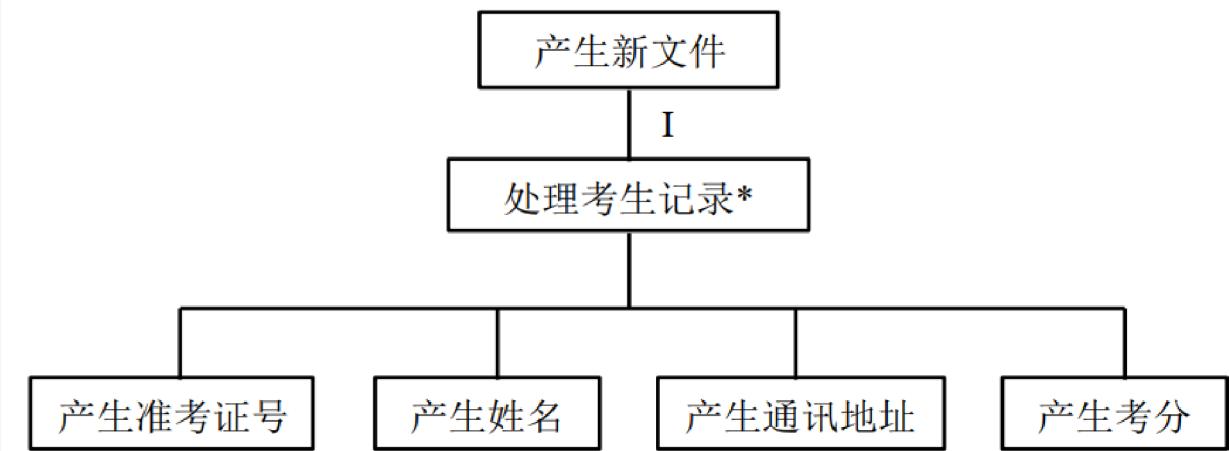
第二步 找出输入数据结构和输出数据结构的对应关系

找出输入数据结构和输出数据结构中有对应关系的数据单元，即有直接因果关系、在程序中可以同时处理的数据单元。需要注意的是，对于重复的数据单元，必须是重复的次序、次数都相同才有可能有对应关系。



第三步 确定程序结构图

实际上，这一步是一个综合的过程：每对有对应关系的数据单元合画一个处理框，没有对应关系的数据单元则各画一个处理框。



- 第三步中，每一个模块均是处理单元；
- 建立的是程序结构图；

第四步 列出并分配所有操作和条件

列出所有**操作和条件**（包括分支条件和循环结束条件），并把它们分配到程序结构图的适当位置。

操作:

- (1) 停止;
- (2) 打开两个输入文件;
- (3) 建立输出文件;
- (4) 从输入文件中各读一条记录;
- (5) 生成一条新记录;
- (6) 将新记录写入输出文件;
- (7) 关闭全部文件;

条件:

I (1) 文件结束。

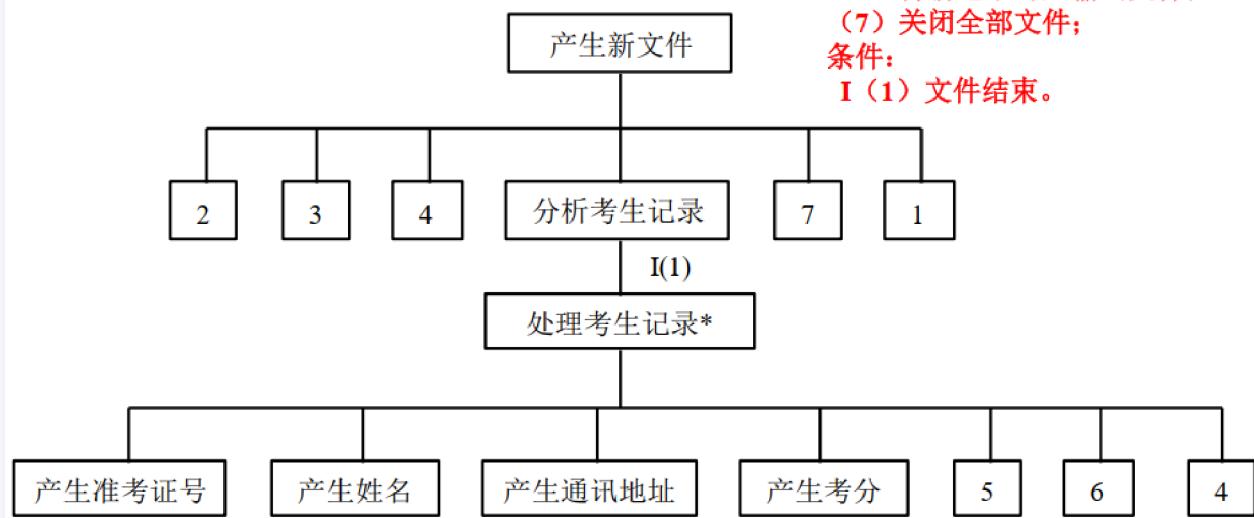
把操作和条件分配到程序结构图的适当位置

操作:

- (1) 停止;
- (2) 打开两个输入文件;
- (3) 建立输出文件;
- (4) 从输入文件中各读一条记录;
- (5) 生成一条新记录;
- (6) 将新记录写入输出文件;
- (7) 关闭全部文件;

条件:

I (1) 文件结束。



- 每一行的单元均是顺序执行；

第五步 用伪码表示程序

```
产生新文件 seq
    打开两个输入文件
    从输入文件中各读一条记录
    分析考生记录iter until文件结束
        处理考生记录 seq
            产生准考证号
            产生姓名
            产生通讯地址
            产生考分
            生成一条新记录
            将新记录写入输出文件
            从输入文件中各读一条记录
        处理考生记录 end
    关闭全部文件
    停止
产生新文件 end
```

Jackson方法举例

【例】一个正文文件由若干记录组成，每个记录是一个字符串，要求统计每个记录中空格字符的个数及文件中空格字符的总个数。要求输出数据格式是每复制一行字符串后，另起一行打印出这个字符串中的空格数，最后打印出文件空格的总个数，使用**Jackson**方法设计该程序结构。

第一步 数据结构表示

第二步 找出输入数据结构和输出数据结构的对应关系

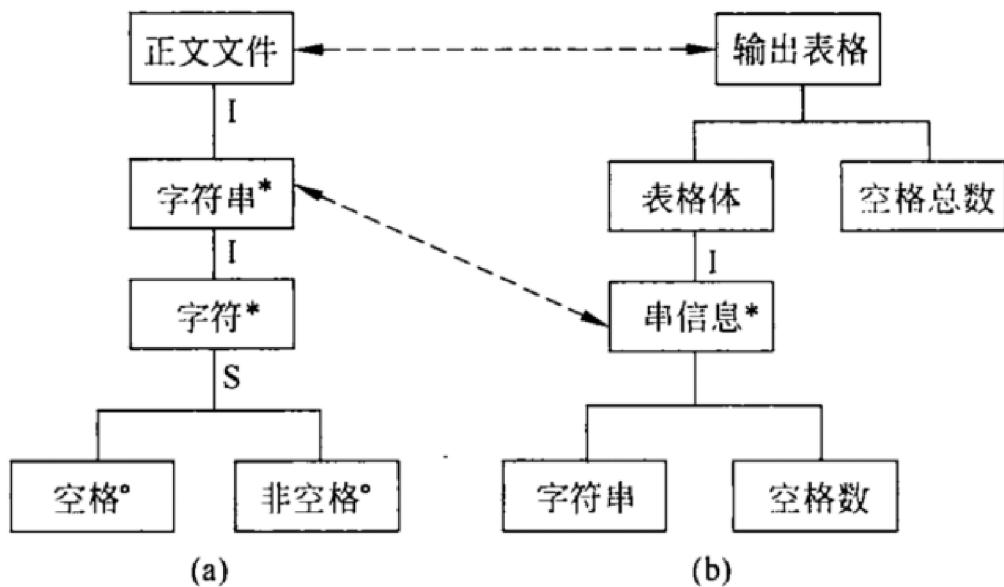


图 6.12 表示输入输出数据结构的 Jackson 图

第三步 确定程序结构图

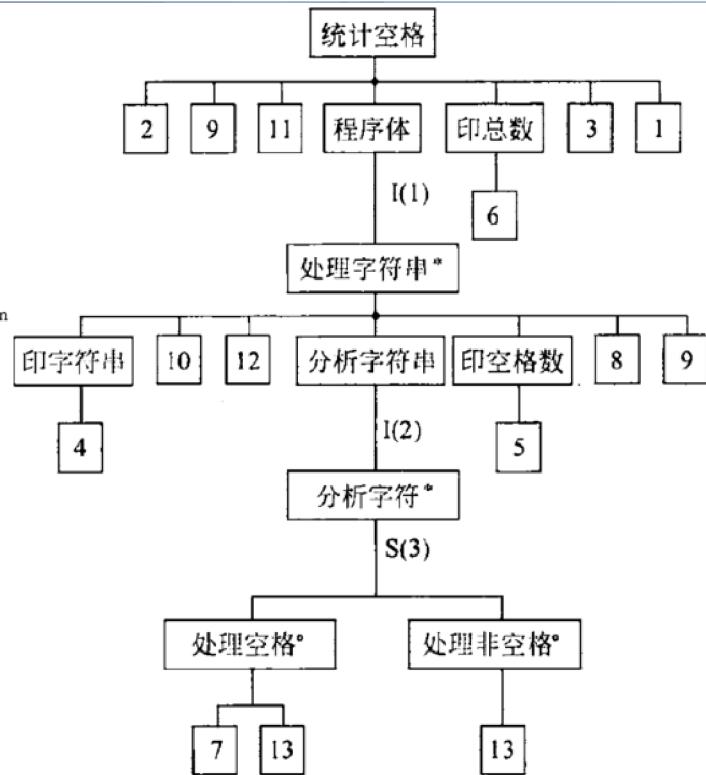


第四步 列出并分配所有操作和条件

- | | |
|-----------------------------|--------------------------------|
| (1) 停止 | (2) 打开文件 |
| (3) 关闭文件 | (4) 印出字符串 |
| (5) 印出空格数目 | (6) 印出空格总数 |
| (7) sum := sum + 1 | (8) totalsum := totalsum + sum |
| (9) 读入字符串 | (10) sum := 0 |
| (11) totalsum := 0 | (12) pointer := 1 |
| (13) pointer := pointer + 1 | I(1) 文件结束 |
| I(2) 字符串结束 | S(3) 字符是空格 |

分配到程序结构图的适当位置。

- | | |
|-----------------------------|--------------------------------|
| (1) 停止 | (2) 打开文件 |
| (3) 关闭文件 | (4) 印出字符串 |
| (5) 印出空格数目 | (6) 印出空格总数 |
| (7) sum := sum + 1 | (8) totalsum := totalsum + sum |
| (9) 读入字符串 | (10) sum := 0 |
| (11) totalsum := 0 | (12) pointer := 1 |
| (13) pointer := pointer + 1 | I(1) 文件结束 |
| I(2) 字符串结束 | S(3) 字符是空格 |



第五步 用伪码表示程序

```
统计空格 seq  
    打开文件  
    读入字符串  
    totalsum := 0  
    程序体 iter until 文件结束  
        处理字符串 seq  
            印字符串 seq  
            印出字符串  
            印字符串 end  
            sum := 0  
            pointer := 1  
            分析字符串 iter until 字符串结束  
                分析字符 select 字符是空格  
                    处理空格 seq  
                        sum := sum + 1  
                        pointer := pointer + 1  
                    处理空格 end  
                分析字符 or 字符不是空格  
        处理非空格 seq  
            pointer := pointer + 1  
        处理非空格 end  
    分析字符串 end  
    印空格数 seq  
        印出空格数目  
    印空格数 end  
    totalsum := totalsum + sum  
    读入字符串  
    处理字符串 end  
    程序体 end  
    印总数 seq  
        印出空格总数  
    印总数 end  
    关闭文件  
    停止  
统计空格 end
```

6.8 程序复杂度度量

1. 代码行度量法

- 前提
 - 程序复杂性随程序规模的增加不均衡地增长；
 - 分而治之的方法可以控制程序规模；
- 做法：统计一个程序模块的源代码行数，并以源代码行数作为程序复杂性的度量；

2. McCabe法（环路复杂性度量）

McCabe 复杂性度量又称环路复杂性度量，基于程序控制流的度量方法；

它认为程序的复杂性很大程度上取决于程序图的复杂性。

单一的顺序结构最为简单，**循环和选择**所构成的环路越多，程序就越复杂。

McCabe基于程序模块的程序图中**环路的个数**，因此要先画出程序图。**程序图是退化的程序流程图**，将流程图中每个处理退化成一个**结点**，流线变成连接不同结点的**有向弧**。

■ 第一步：画出控制流图

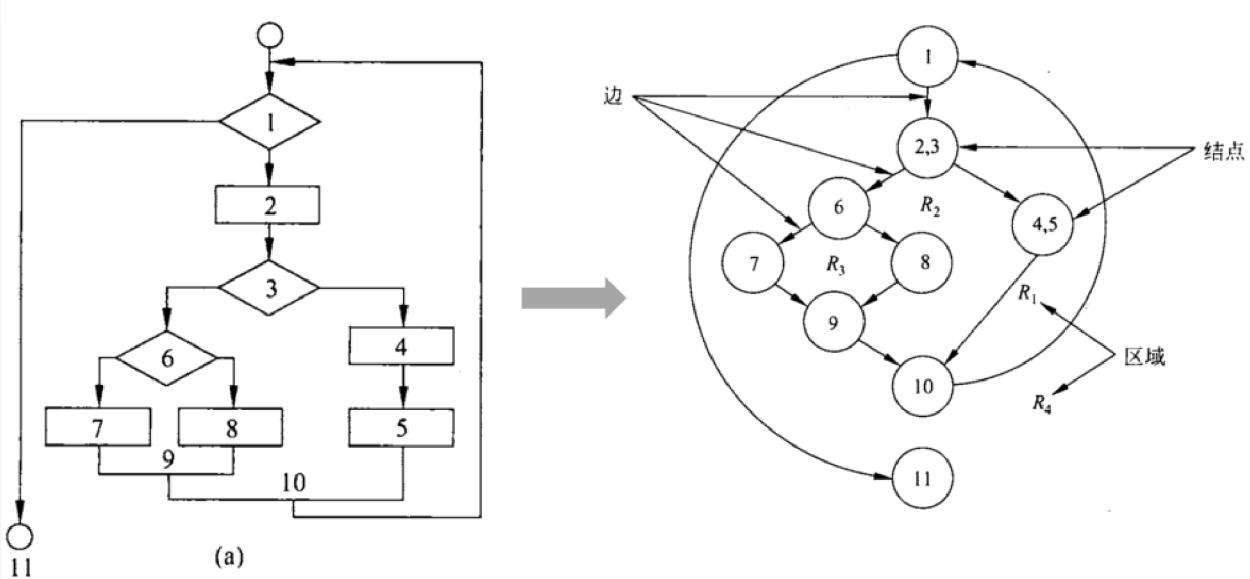
流图用来描述程序控制结构。可将流程图映射到一个相应的流图(**假设流程图的菱形决定框中不包含复合条件**)。

在流图中，每一个圆称为流图的**结点**，代表一个或多个语句。一个处理方框序列和一个菱形决测框可被映射为一个结点。

流图中的箭头，称为**边或连接**，代表控制流，类似于流程图中的箭头。

一条边必须终止于一个结点，即使该结点并不代表任何语句(**例如：if-else-then结构**)。

由边和结点限定的范围称为**区域**。计算区域时应包括图外部的范围。

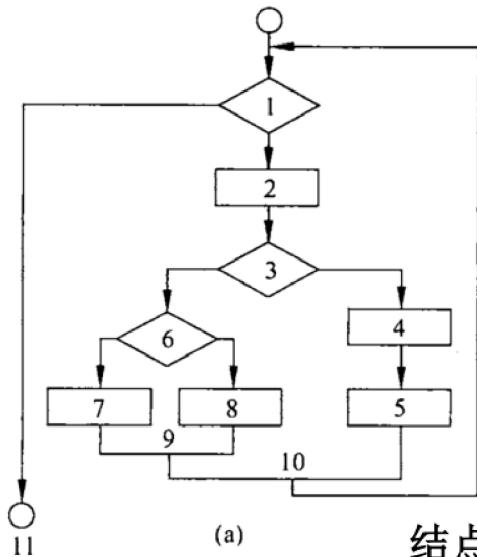


■ 第二步：计算圈复杂度

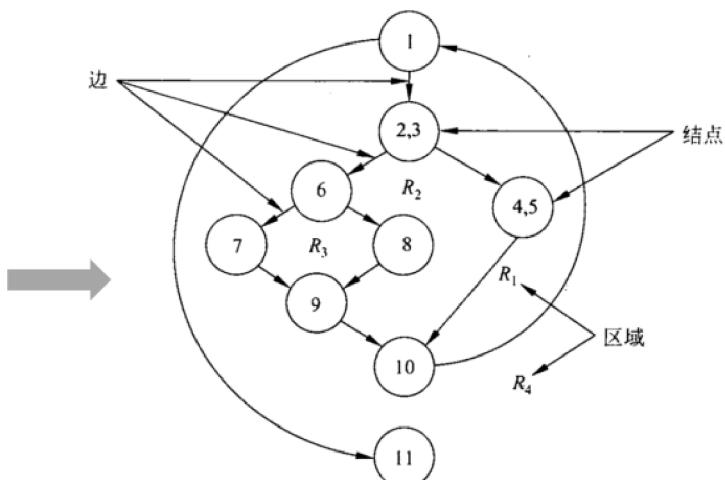
圈复杂度是一种为程序逻辑复杂性提供定量测度的软件度量，将该度量用于计算程序的基本的独立路径数目，为确保所有语句至少执行一次的测试数量的上界。独立路径必须包含一条在定义之前不曾用到的边。有以下三种方法计算圈复杂度：

- 流图中区域的数量对应于环型的复杂性；
- 给定流图 G 的圈复杂度 $V(G)=E-N+2$ ， E 是流图中边的数量， N 是流图中结点的数量；
- 给定流图 G 的圈复杂度 $V(G)$ ，定义为 $V(G)=P+1$ ， P 是流图 G 中判定结点的数量。

本例中环形复杂度为4



(a)



结点数 $n=9$, 弧数 $m=11$, 则

$$V(G) = m - n + 2 = 11 - 9 + 2 = 4。$$

即：程序图中区域数。

几点说明

环路复杂度取决于程序控制结构的复杂度。当程序的分支数目或循环数目增加时，其复杂度也增加。

环路复杂度是可加的。例如，模块A的复杂度为3，模块B复杂度为4，则模块A与模块B的复杂度为7。

3. halstead法

以程序中出现的操作符和操作数为计数对象，以它们的出现次数作为计数目标来测算程序容量和工作量；

任意程序 P，总是由操作符和操作数通过有限次的组合连缀而成。

P 的符号表词汇量 $n = n_1 + n_2$ (n_1 : 惟一操作数数量, n_2 : 惟一操作符数量)。设 N_1 是 P 中出现的所有操作数, N_2 是程序中出现的所有操作符。

度量指标如下:

- 程序长度: $N=N_1+N_2$
- halstead 预测长度: $H=n_1*\log_2n_1 + n_2*\log_2n_2$
- 预测程序包含错误个数: $E = N*\log_2(n_1+n_2) / 3000$

第七章 实现

★ 软件测试目的: 尽可能多的发现并排除软件中的错误

- 单元测试: 单元测试是针对模块, 它不是一个独立的程序。因此, 模块自己不能运行, 要靠其它部分来调动和驱动, 这样就要为每个单元测试开发两个软件: 驱动程序和连接程序。
- 集成测试: 集成测试是组装软件的系统技术。
- 系统测试
- 确认(交付)测试: 向用户表明系统能满足要求, 测试范围与系统测试范围类似, 确认测试是黑盒测试。
- 前三种测试不需要用户参与

7.1 白盒测试

逻辑覆盖

- 概念: 根据程序内部的逻辑结构测试程序内部的主要执行通路是否能正确工作

白盒测试: 由称为结构测试, 典型的α测试

- ① 进行逻辑测试, 它完全了解结构及处理过程。
- ② 按照程序内部的逻辑测试程序, 检查程序中每条通路是否都能按预定要求正确工作。

- 覆盖技术分类:

- 语句覆盖：每条语句至少执行一次
- 判定覆盖：每个判定的每个分支至少执行一次
- 条件覆盖：判定表达式的每个条件的所有可能都要取一次
- 判定和条件覆盖同时满足
- 条件组合覆盖：每个判定表达式的条件的各种组合都出现一次（条件覆盖只关注单个条件的覆盖）
- 路径覆盖

六种覆盖标准的对比

发 现 错 误 能 力 ↓ 弱 → 强	语句覆盖	每条语句至少执行一次
	判定覆盖	每个判定的每个分支至少执行一次
	条件覆盖	每各判定的每个条件应取到各种可能的值
	判定/条件覆盖	同时满足判定覆盖和条件覆盖
	条件组合覆盖	每个判定中各条件的每一种组合至少出现一次
	路径覆盖	使程序中每一条可能的路径至少执行一次

★ 根据流程图给出各种覆盖测试用例和覆盖路径

eg：根据流程图给出路径覆盖测试用例：

路径覆盖用例：

$a = 2, b = 0, x = 2$

(覆盖路径：1、2、3、4、5)

$a = 2, b = 1, x = 2$

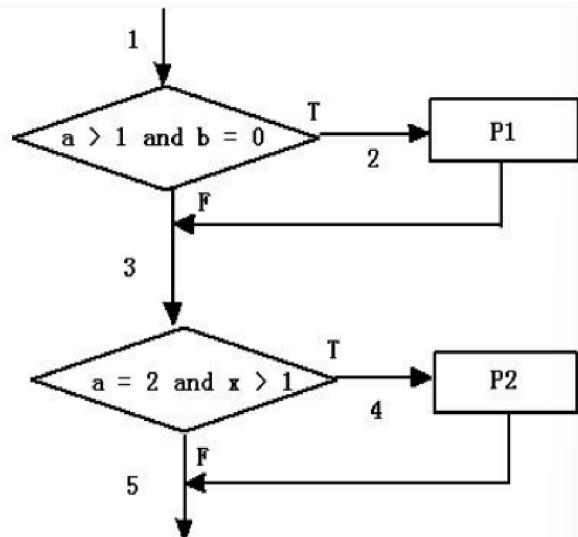
(覆盖路径：1、3、4、5)

$a = 1, b = 1, x = 1$

(覆盖路径：1、3、5)

$a = 3, b = 0, x = 1$

(覆盖路径：1、2、3、5)



执行每个测试用例并与期望值比较，一旦完成所有测试用例测试者就可以确定程序中的所有语句至少被执行了一次

基本路径测试

- **★ 基本路径测试法：**

1. 根据过程设计绘制相应流图（控制流图）

2. 确定流图的环形复杂度 $V(G)$

- $V(G) = \text{所有条件语句数量} + 1$

3. 确定**线性独立的路径**的一个基本集（判断有多少通路）：

- 独立路径：独立路径至少包含一条在**定义该路径之前不曾用过的边**；

$V(G)$ 的值提供了程序控制结构中线性独立的路径的数量，在过程求平均值中，我们指定六条路径：

路径1: 1—2—10—11—13

路径2: 1---2---10---12---13

路径3: 1---2---3---10---11---13

路径4: 1---2---3---4---5---8---9---2--- ...

路径5: 1---2---3---4---5---6---8---9---2---

...

路径6: 1---2---3---4---5---6---7---8---9---

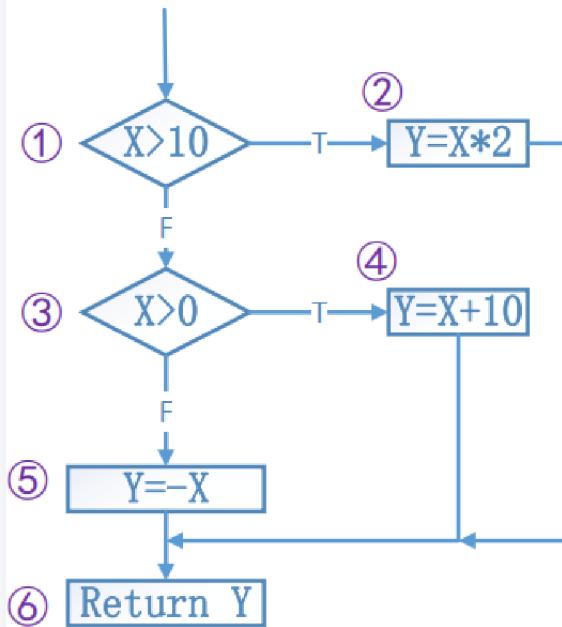
...

4. 准备测试用例，强制执行基本集中的每条路径

基本路径测试举例

基本路径测试方法练习

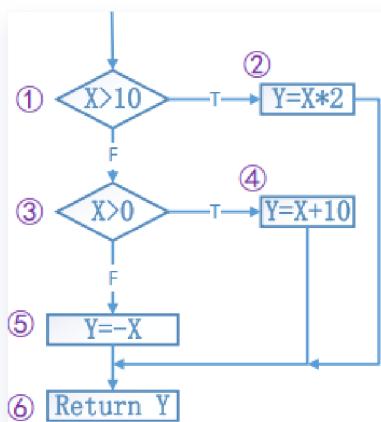
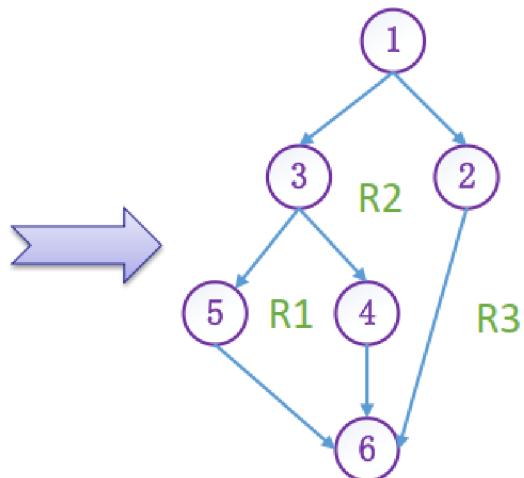
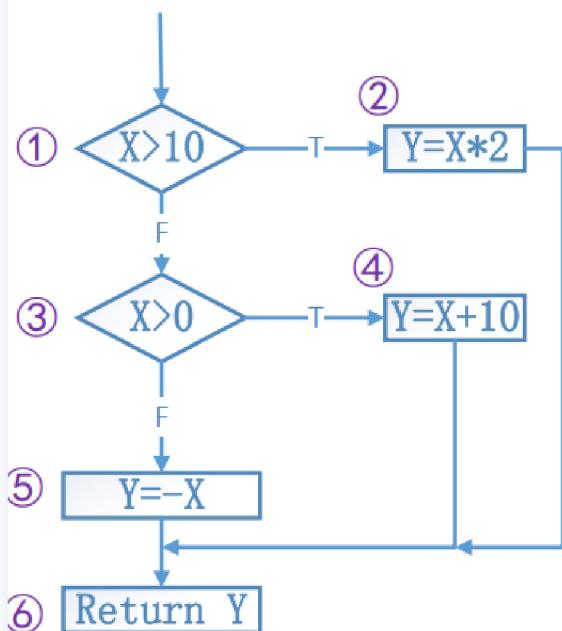
根据程序流程图画出其控制流图，计算其环路复杂度，并根据基本路径设计测试用例



基本路径测试方法练习

①根据程序流程图画出其控制流图

②计算其环路复杂度为**3**



基本路径测试方法练习

③确定独立路径基本集合

路径**1**: 1—2—6

路径**2**: 1—3—4—6

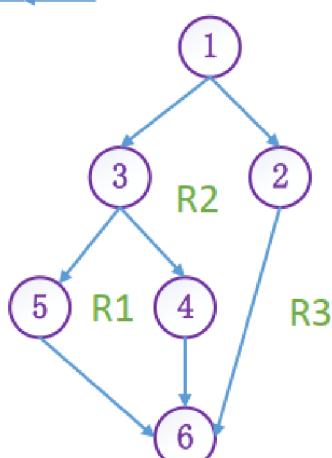
路径**3**: 1—3—5—6

④设计测试用例

X=15 覆盖1—2—6

X=5 覆盖1—3—4—6

X=0 覆盖1—3—5—6



7.2 黑盒测试

黑盒测试：又称为功能测试，典型的 β 测试

- ① 完全不考虑程序内部结构和处理过程。
- ② 具体是在程序接口进行测试，输入数据，输出信息完整性。

1. 等价划分：只需要测试所有输入数据中划分出的每个等价类中的一个典型值即可

- 划分等价类：有效等价类 和 无效等价类
 - 根据等价类设计用例

应用题：判定三角形类别的测试范例

(一) 采用等价划分

划分等价类并编号



输入条件	合理等价类	不合理等价类
是否为三角形	a. 任意两边之和 大于第三边	b. 任意两边之和 小于第三边 c. 任意两边之和 等于第三边
	d. 三个正整数	e. 多于三个正整数 f. 少于三个正整数 g. 含有负数 h. 含有非数字字符
判断三角形	i. 等边三角形 j. 等腰三角形 k. 一般三角形	

2、为合理等价类设计测试用例

测试数据	期望结果	覆盖范围
3, 4, 5	有效数据	a, d, k
3, 3, 3	有效数据	a, d, i
3, 3, 5	有效数据	a, d, j

3、为每个不合理等价类至少设计一个测试用例

测试数据	期望结果	覆盖范围
2, 4, 7	输入无效	b
2, 4, 6	输入无效	c
2, 4, 7, 8	输入无效	e
2, 4	输入无效	f
2, -3, -4	输入无效	g
A, C, 7	输入无效	h

输入等价类	测试用例说明	测试数据	期望结果	选取理由
是否为三角形	任两边之和等于第三边 任两边之和小于第三边	1, 2, 3 1, 2, 4	显示出错 显示出错	两边之和等于第三边 两边之和小于第三边
输入数据	有一个输入数据 有两个输入数据 有四个输入数据 “0”输入数据 负数输入数据 有一个非数字输入数据 全部非数字输入数据	1 1, 2 3, 4, 5, 6 0, 0, 0 -3, -4, -5 3, A, 5 =, +, F	显示出错 显示出错 显示出错 显示出错 显示出错 显示出错 显示出错	最少的数据输入 比有效输入少1 比有效输入多1 输入无效边长 输入错误边长 有一个非法数据 全部是非法数据
判断三角形 (任两边之和大于第三边)	三边相等 任两边相等 三边皆不相等	1, 1, 1 1, 2, 2 2, 3, 4	等边三角形 等腰三角形 一般三角形	三边相等的有效输入 二边相等的有效输入 三边不等的有效输入

2. 边界值分析：不是从等价类中选一数据，而是选特定值使得等价类的每个边界都作为测试目标

(1) 若某输入条件规定了数据大小, 可选正好等于边界值的数据作为合理测试用例, 同时还要选择正好越过边界值的数据作为不合理的测试用例。

如输入值的范围为 $-1.0 \sim +1.0$; 即可选 $-1.0, +1.0, -1.001, +1.001$ 等值作为测试数据

(2) 若某输入规定了数据个数, 则可分别设计边界值和超过边界值的测试用例。

如输入文件有 $1 \sim 256$ 个纪录, 可选0个, 1个, 256个, 257个纪录。

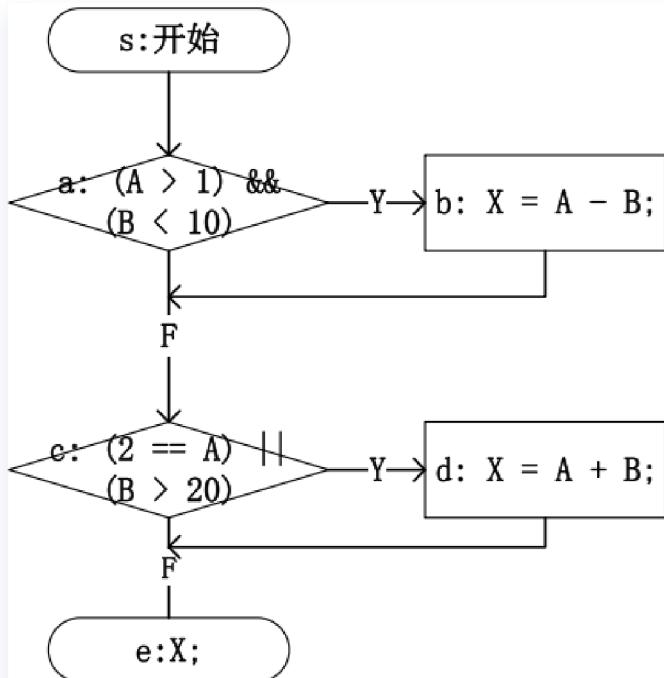
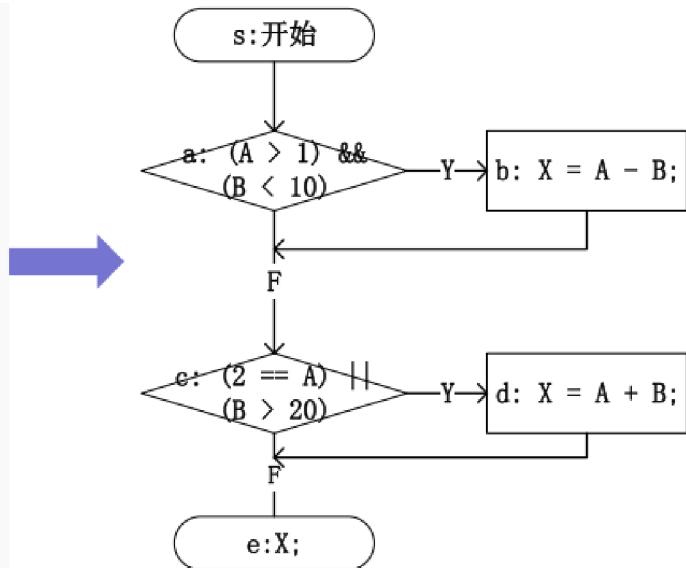
7.3 白盒测试和黑盒测试关系、区别

	黑盒测试	白盒测试
优点	①适用于各阶段测试 ②从产品功能角度测试 ③容易入手生成测试数据	①可构成测试数据使特定程序部分得到测试 ②有一定的充分性度量手段 ③可获较多工具支持
缺点	①某些代码得不到测试 ②如果规格说明有误, 则无法发现 ③不易进行充分性测试	①通常不易生成测试数据 ②无法对未实现规格说明的部分进行测试 ③工作量大, 通常只用于单元测试, 有应用局限
性质	一种确认技术, 回答“我们在构造一个正确的系统吗?”	一种验证技术, 回答“我们在正确地构造一个系统吗?”

7.4 习题

设计测试用例，分别满足语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖。

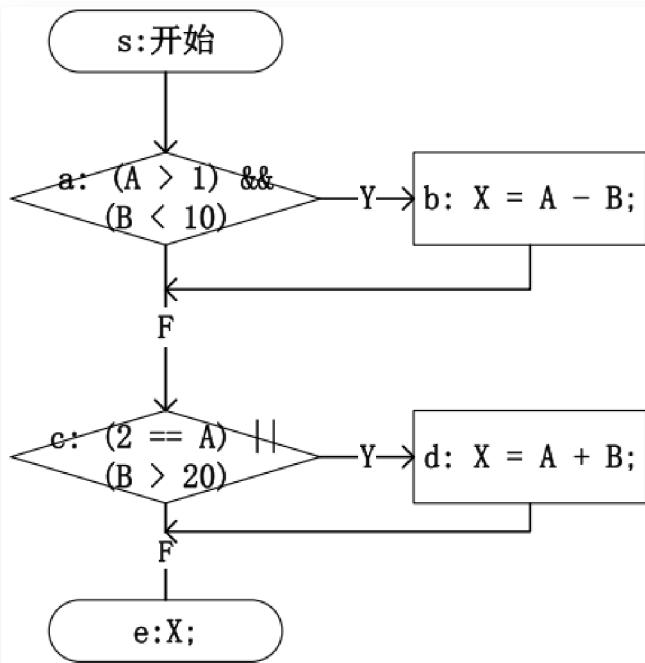
```
int DoTest (int A, int B)
{
    if ((A > 1) && (B < 10))
    {
        X = A - B;
    }
    if((2 == A) || (B > 20))
    {
        X = A + B;
    }
    return X;
}
```



1、语句覆盖：被测程序中每个语句至少执行一次

A=2, B=0, 预计输出结果X=2

路径： s->a->b->c->d->e



2、判定覆盖：不仅每个语句必须至少执行一次，而且每个判定的每种可能的结果都应该至少执行一次

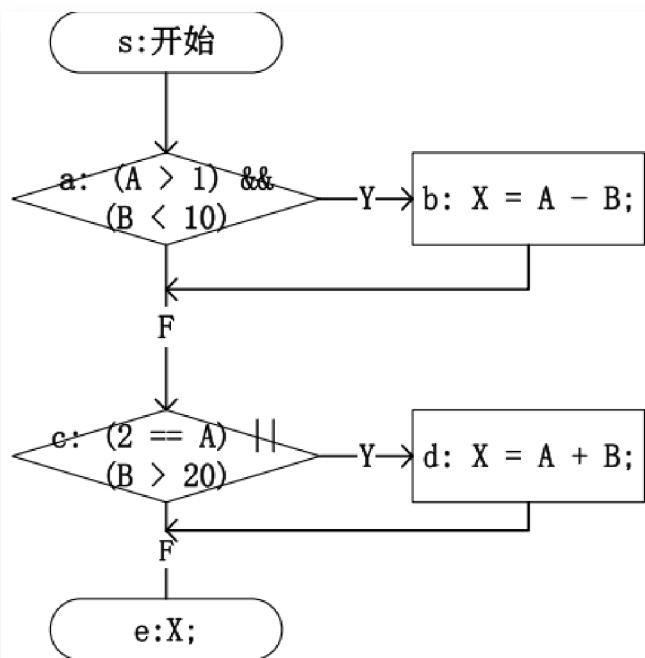
至少两个测试用例，使得ac为TT、FF或TF、FT或FT、TF，

A=3, B=0, 预计输出结果X=3 (ac为TF)

A=2, B=20, 预计输出结果X=22 (ac为FT)

路径: s->a->b->c->e
s->a->c->d->e

这里选择使得ac满足TF、FT



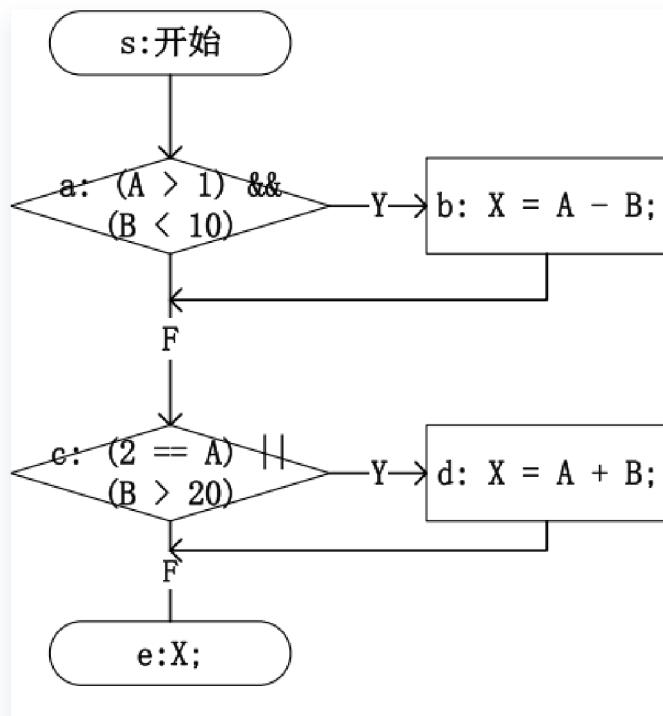
3、条件覆盖：不仅每个语句至少执行一次，而且使判定表达式中的每个条件都取到各种可能的结果

A=2, B=0, 预计输出结果X=2 (TTTF)

A=0, B=21, 预计输出结果X=21 (FFFT)

路径: s->a->b->c->d->e
s->a->c->d->e

四个条件，分别为TTTF、FFFT



4、判定/条件覆盖：不仅每个判定的每种可能的结果都应该至少执行一次，而且使判定表达式中的每个条件都取到各种可能的结果

四个条件，两个判定

A=2, B=0, 预计输出结果X=2 (TTTF, ac为TT)

A=0, B=21, 预计输出结果X=21 (FFFT, ac为FT)

A=1, B=1, 预计输出结果X=? (FTFF, ac为FF)

路径: s->a->b->c->d->e
s->a->c->d->e

s->a->c->e

练习2:

Date 函数包含三个变量：选取 year 和 month，要求输入变量year和month均为整数值，并且满足下列条件：① $1970 \leq year \leq 2022$ ② $1 \leq month \leq 12$ ，列出等价类表并设计测试用例。

步骤1：划分等价类并给予唯一的编号

参数	有效等价类	无效等价类
年	1970≤year≤2012 ①	year<1970 ③ year>2022 ④
月	1≤month≤12 ②	month<1 ⑤ month>12 ⑥

练习2:

Date 函数包含三个变量：选取 year 和 month，要求输入变量year和month均为整数值，并且满足下列条件：① $1970 \leq year \leq 2022$ ② $1 \leq month \leq 12$ ，列出等价类表并设计测试用例。

步骤2：为有效等价类设计测试用例

设计一个新的测试用例，使其尽可能多地覆盖尚未被覆盖的有效等价类，重复这一步，直到所有的有效等价类都被覆盖为止

序号	设计测试用例	覆盖等价类
1	1970年1月	① 1970≤year≤2022 ② 1≤month≤12
2		
3		
.....		

练习2:

Date 函数包含三个变量：选取 year 和 month，要求输入变量year和month均为整数值，并且满足下列条件：① $1970 \leq year \leq 2022$ ② $1 \leq month \leq 12$ ，列出等价类表并设计测试用例。

步骤3：为无效等价类设计测试用例

设计一个新的测试用例，使其只覆盖一个尚未覆盖的无效等价类，重复这一步，直到所有的有效等价类都被覆盖为止

序号	设计测试用例	覆盖等价类
1	1970年1月	① 1970≤year≤2022 ② 1≤month≤12
2	1950年1月	③ Year<1970
3	2023年1月	④ Year>2022
4	1970年-1月	⑤ Month<1
5	1980年13月	⑥ Month>12
.....		

8.1 软件维护

★软件维护的分类、定义以及提高可维护性方法

- 分类：

1. 改正性维护
2. 适应性维护
3. 完善性维护
4. 预测性维护

- 定义：软件已经交付使用后，为改正错误或者满足新的需要而修改软件的过程

- 提高可维护性：

1. 使用软件审查：

- 在检查点进行复查
- 验收检查
- 周期性维护审查
- 对软件包进行检查

2. 改进程序的文档：利用系统开发日志、错误记载、系统维护日志等历史文档

★软件维护副作用：

1. 编码副作用：修改程序源码时可能引入错误
2. 数据副作用：修改数据结构，造成软件设计和数据结构不匹配
3. 文档副作用：对数据流、软件结构、模块逻辑等进行修改后，必须对相关技术文档进行修改，避免文档不能反映当前真实状态

★度量软件可维护性的方法：

1. 可理解性：阅读源代码以及文档了解软件的容易程度
2. 可靠性：程序在给定时间内正确执行的概率
3. 可测试性
4. 可修改性
5. 可移植性
6. 效率
7. 可使用性

8.2 软件文档

★ 文档在软件工程中的作用：

1. 提高软件开发过程的能见度

2. 提高开发效率

- 软件文档的作用：

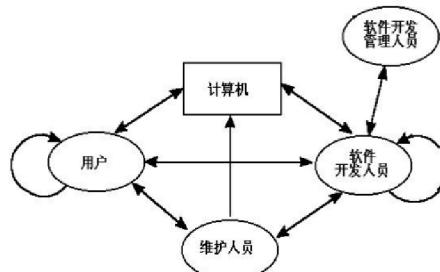
(1) 作为开发人员在一定阶段的工作成果和结束标志。

(2) 记录开发过程中的有关信息，便于协调以后的软件开发、使用和维护。

(3) 提供对软件的运行、维护和培训的有关信息，便于管理人员、开发人员、操作人员、用户之间的协作、交流和了解。使软件开发活动更科学、更有成效。

(4) 便于潜在用户了解软件的功能、性能等各项指标，为他们选购符合自己需要的软件提供依据。

文档在各类人员、计算机之间
的多种桥梁作用可从图中看出。



- 软件文档的分类：【开发文档】 【管理文档】 【用户文档】

软件开发项目生存期各阶段所包含的文档：

阶段 文档 \	可行性研 究与计划	需求 分析	软 件 设 计	编 码 与 单 元 测 试	集 成 与 测 试	运 行 维 护
可行性研究报告	→					
项目开发计划	→	→				
软件需求说明书	→					
数据需求说明书	→					
测试计划		→				
概要设计说明书		→				
详细设计说明书		→				
用户手册		→				
操作手册		→				
测试分析报告			→			
开发进度月报			→			
项目开发总结				→		
程序维护手册 (维护修改建议)						→

9.1 面向对象方法的优点：

1. 与人类习惯的思维方法一致
2. 稳定性好
3. 可重用性好
4. 可维护性好

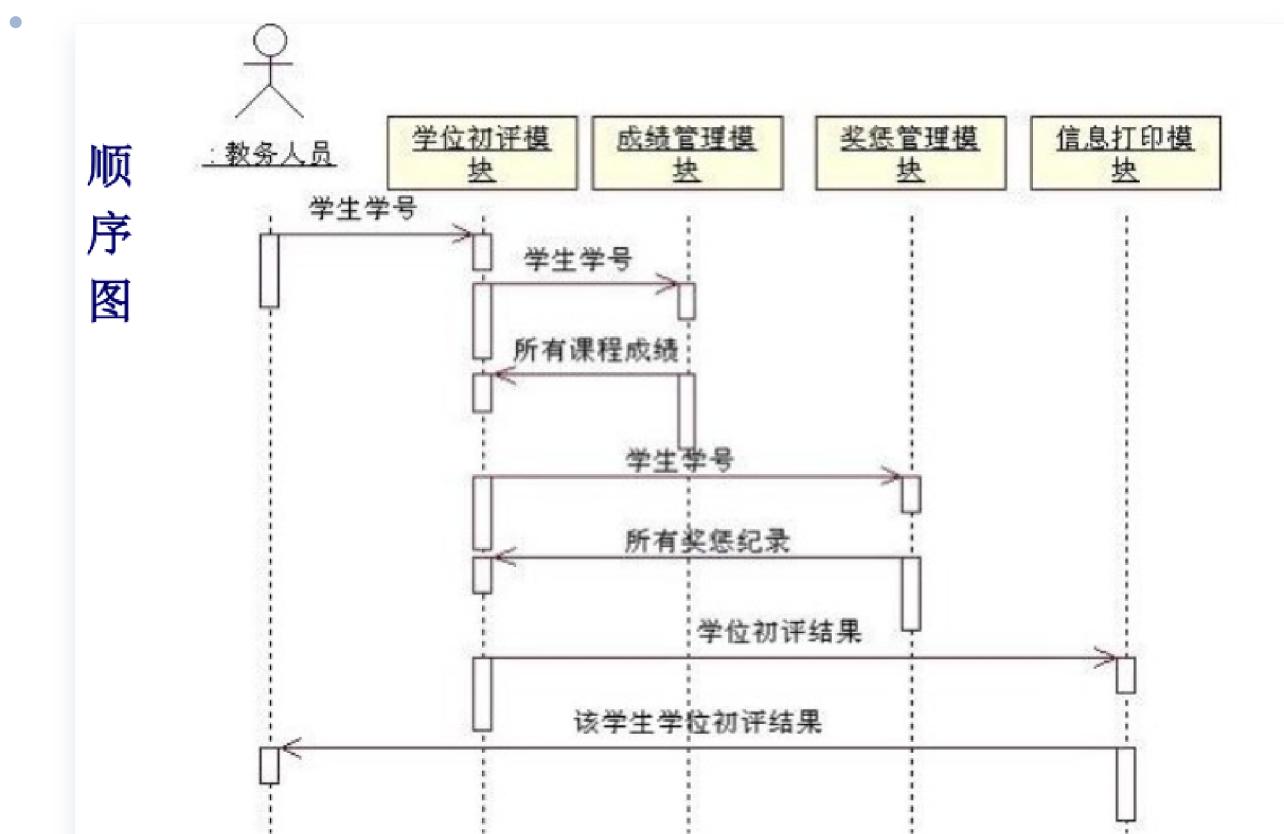
信息隐蔽：模块内包含的信息，对于不需要这些信息的模块来说，是不能访问的

9.2 ★面向对象设计六大原则

- 模块化
- 抽象
- 信息屏蔽
- 弱耦合
- 强内聚
- 可重用

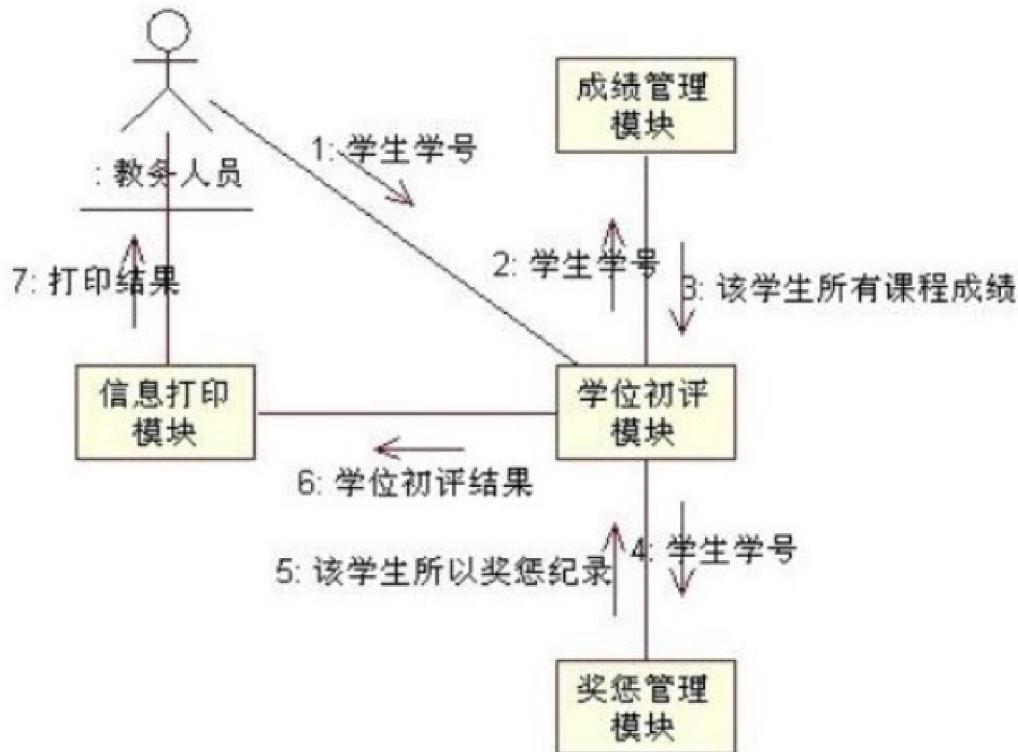
9.3 ★ 顺序图和协作图的区别

- 顺序图：描述了对象之间传递消息的时间顺序，它用来表示用例中的行为顺序，是强调消息时间顺序的交互图；



- 协作图：强调参加交互的各对象的组织，用于描述相互合作的对象间的交互关系和链接关系；

协作图



- 区别：顺序图着重体现交互的时间顺序，协作图则着重体现交互对象间的静态链接关系；

活动图与流程图的区别

- (1) **流程图**着重描述处理过程，它的主要控制结构是顺序、分支和循环，各个处理过程之间有严格的顺序和时间关系。
活动图描述的是对象活动的顺序关系所遵循的规则，它着重表现的是系统的行为，而非系统的处理过程。
- (2) **活动图**能够表示并发活动的情形，而流程图不能。
- (3) **活动图**是面向对象的，而流程图是面向过程的。

9.4 ★UML设计三种模型：对象模型 动态模型 功能模型

9.7.2 三种模型之间的关系

面向对象建模技术所建立的三种模型，分别从三个不同侧面描述了所要开发的系统。这三种模型相互补充、相互配合，使得我们对系统的认识更加全面：

- 功能模型指明了系统应该“做什么”；
- 动态模型明确规定了什么时候(即在何种状态下接受了什么事件的触发)做；
- 对象模型则定义了做事情的实体。

- UML图分类：

- 1、用例图：从外部用户的角度描述系统的功能，并指出功能的执行者。
- 2、静态图：描述系统的静态结构。
 - .类图
 - .对象图
 - .包图
- 3、行为图：刻画系统的动态行为。
 - .交互图(顺序图与合作图)
 - .状态图
 - .活动图
- 4、实现图：描述软件实现系统的组成和分布状况。
 - .构件图
 - .部署图

类图

- 关联关系

- 普通关联是最常见的关联关系，只要类与类之间存在连接关系就可以用普通关联表示。普通关联的图示符号是连接两个类之间的直线。
- 关联一般是双向的，在一个方向上为关联起一个名字，在另一个方向上为关联起另一个名字，如果关联清晰的话，也可不起名字。为避免混淆，在名字前面（或后面）加一个指示关联方向的黑三角。

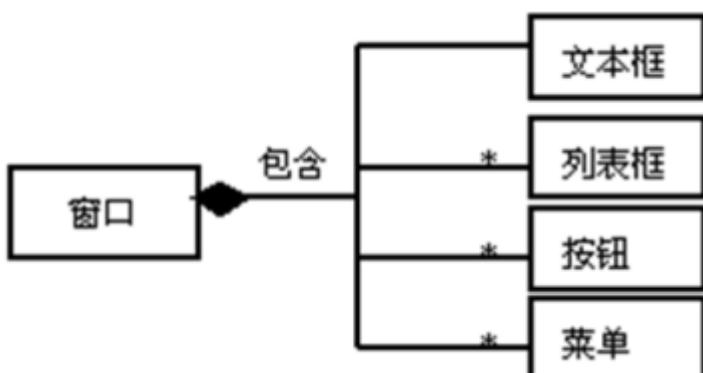


- 聚集关系

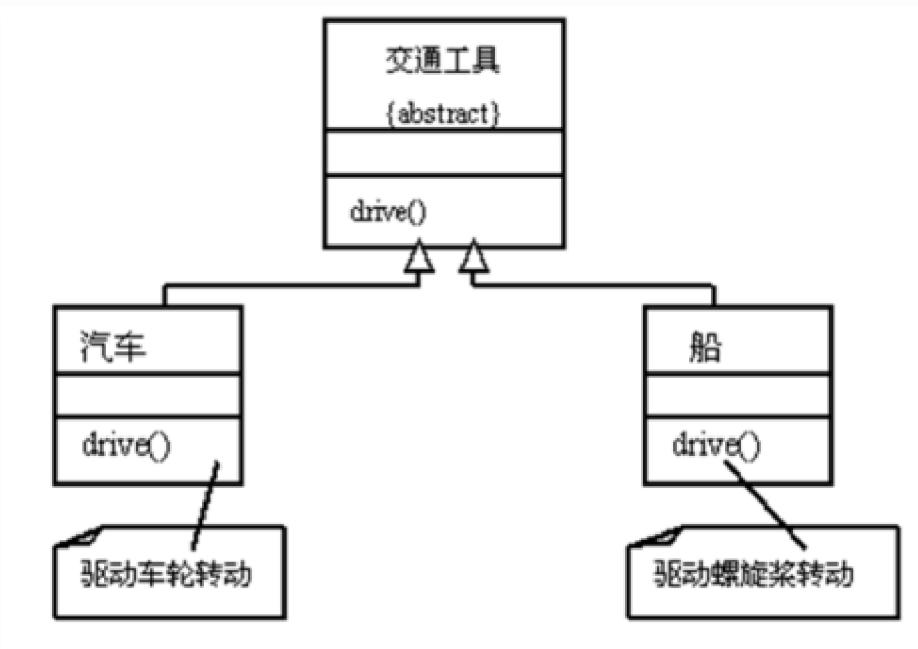
- 聚集和共享聚集的图示符号是在表示关联关系的直线末端紧挨着整体类的地方画一个空心菱形。



- 组成关系

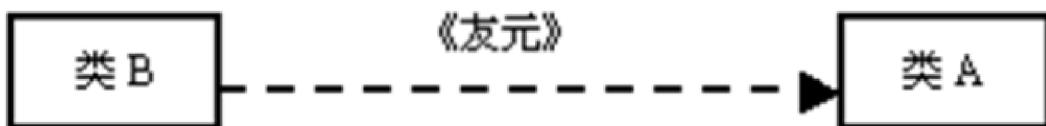


- 泛化关系



- 依赖与细化关系

- 依赖关系描述两个模型元素（类、用例等）之间的语义连接关系：其中一个模型元素是独立的，另一个模型元素不是独立的，它依赖于独立的模型元素，如果独立的模型元素改变了，将影响依赖于它的模型元素。
- 下图表示一个友元依赖关系，该关系使得B类的操作可

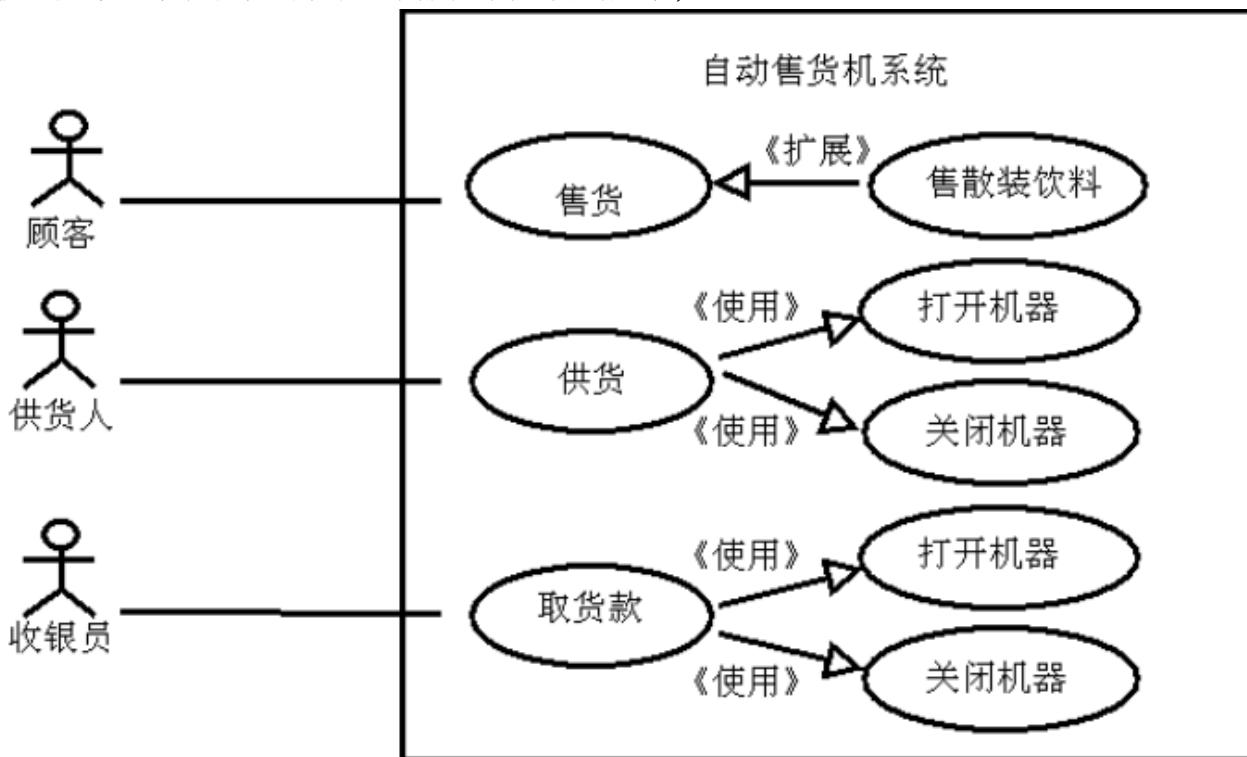


- 当对不同抽象层次上的同一个事物进行描述时，这些层次描述之间具有细化关系。
- 假设两个模型元素A和B描述同一个事物，它们分属不同的抽象层，如果B在A的基础上进行了更详细的描述，则称B细化了A。
- 细化用来协调不同阶段模型之间的关系，表示各个开发阶段不同抽象层次的模型之间的相关性，细化常常用于跟踪模型的演变。



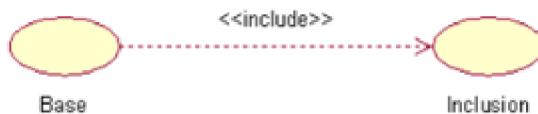
用例图

- 模型元素：系统、行为者、用例及用例之间的关系；



1. 包含

包含关系指用例可以简单地包含其他用例具有的行为，并把它所包含的用例行为作为自身行为的一部分。在UML中，包含关系是通过带箭头的虚线段加<>字样来表示，箭头由基础用例(Base)指向被包含用例(Inclusion)。



在处理包含关系时，具体的做法就是把几个用例的公共部分单独的抽象出来成为一个新的用例。主要有两种情况需要用到包含关系：

第一，多个用例用到同一段的行为，则可以把这段共同的行为单独抽象成为一个用例，然后让其他用例来包含这一用例。

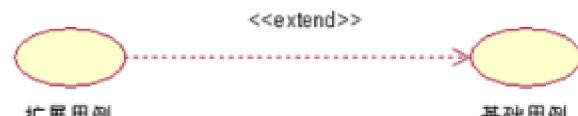
第二，某一个用例的功能过多、事件流过于复杂时，我们也可以把某一段事件流抽象成为一个被包含的用例，以达到简化描述的目的。



2. 扩展

在一定条件下，把新的行为加入到已有的用例中，获得的新用例叫做扩展用例(Extension)，原有的用例叫做基础用例(Base)，从扩展用例到基础用例的关系就是扩展关系。

一个基础用例可以拥有一个或者多个扩展用例，这些扩展用例可以一起使用。

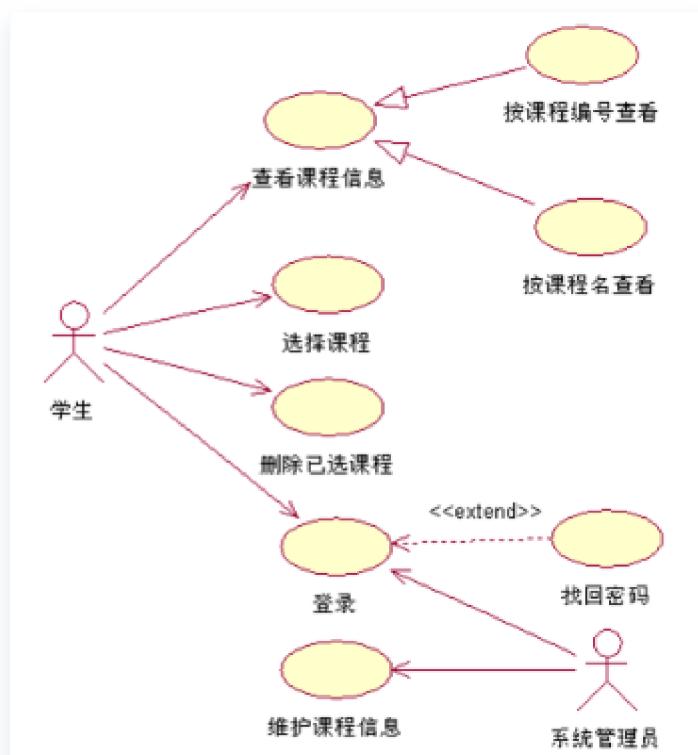
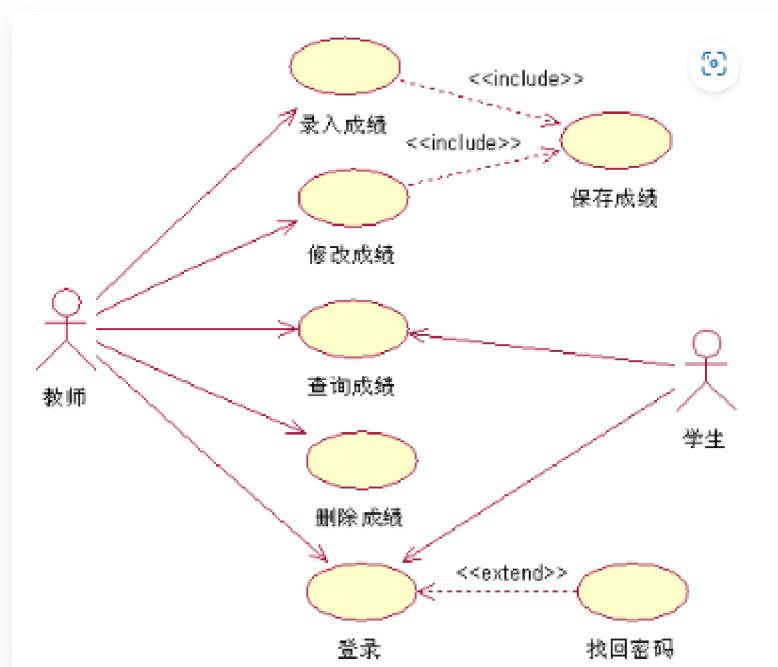


3. 泛化

用例的泛化指的是一个父用例可以被特化形成多个子用例，而父用例和子用例之间的关系就是泛化关系。

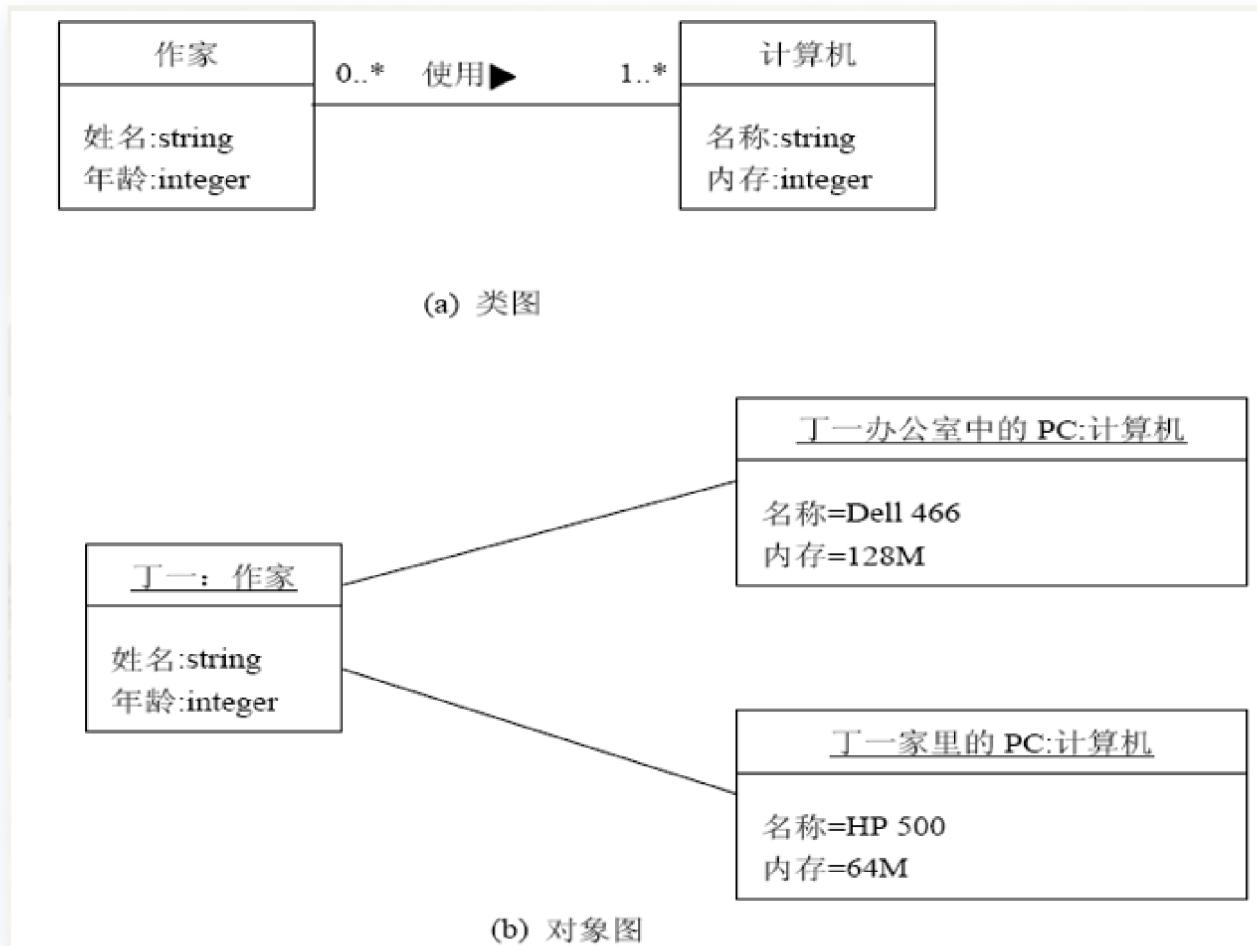
在用例的泛化关系中，子用例继承了父用例所有的结构、行为和关系，子用例是父用例的一种特殊形式。

子用例还可以添加、覆盖、改变继承的行为。在UML中，用例的泛化关系通过一个三角箭头从子用例指向父用例来表示。

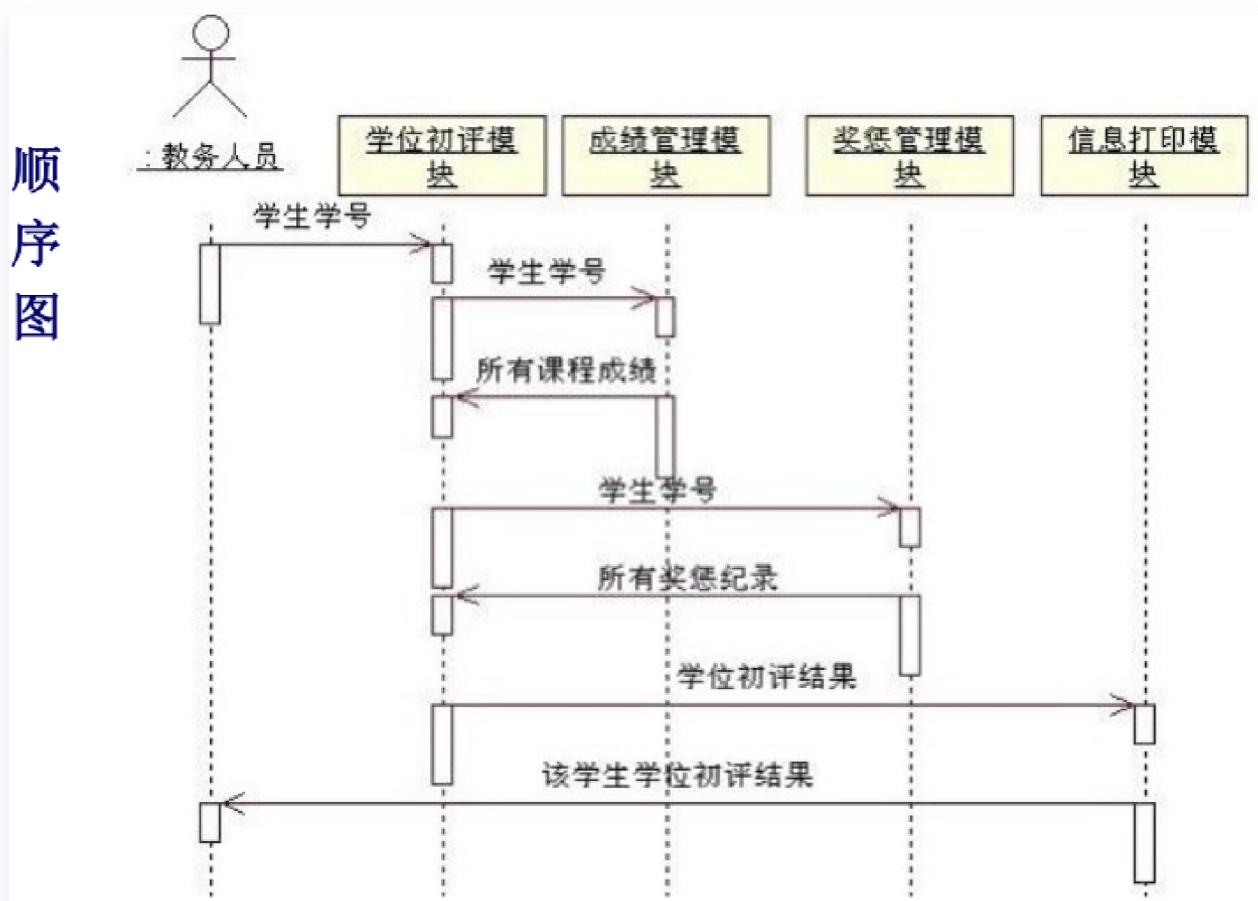


- 扩展、使用关系是泛化关系的两种不同形式，可以理解为类继承；扩展表示生成泛化新的类别；使用表示抽象出新的用例；

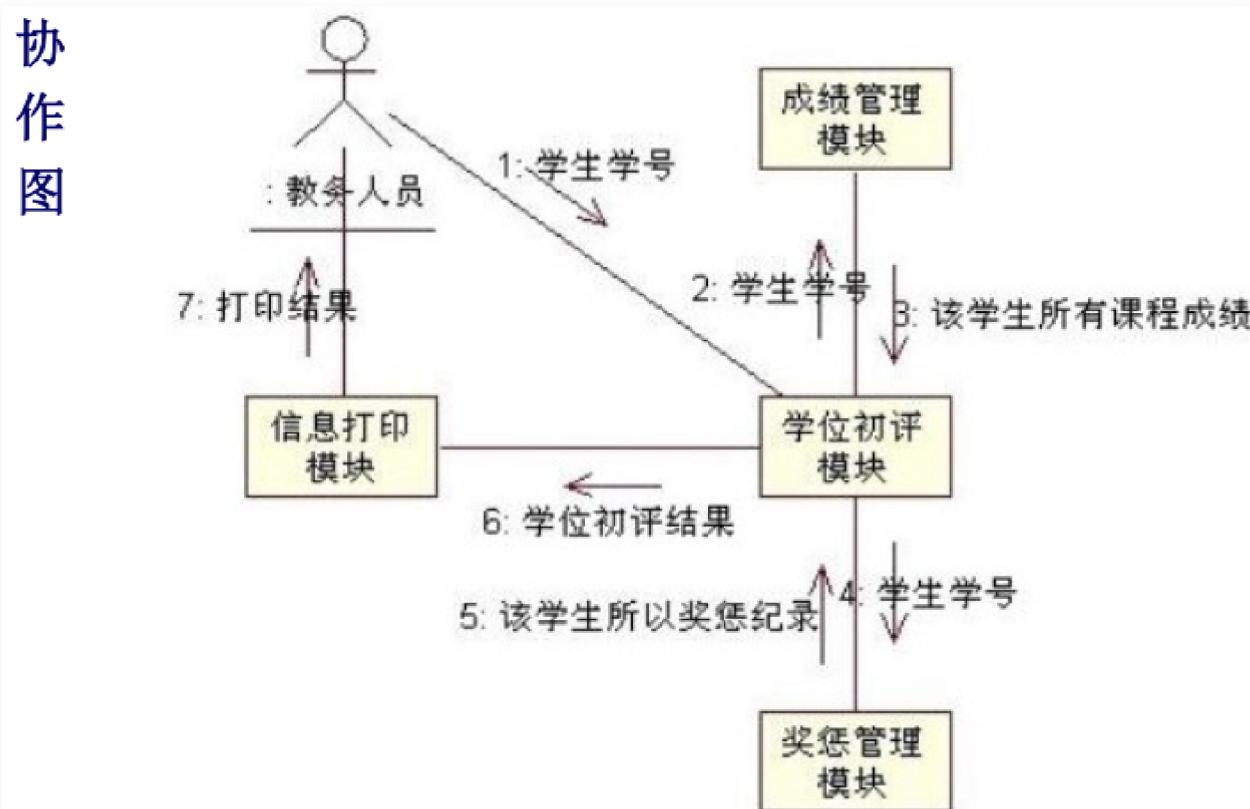
对象图



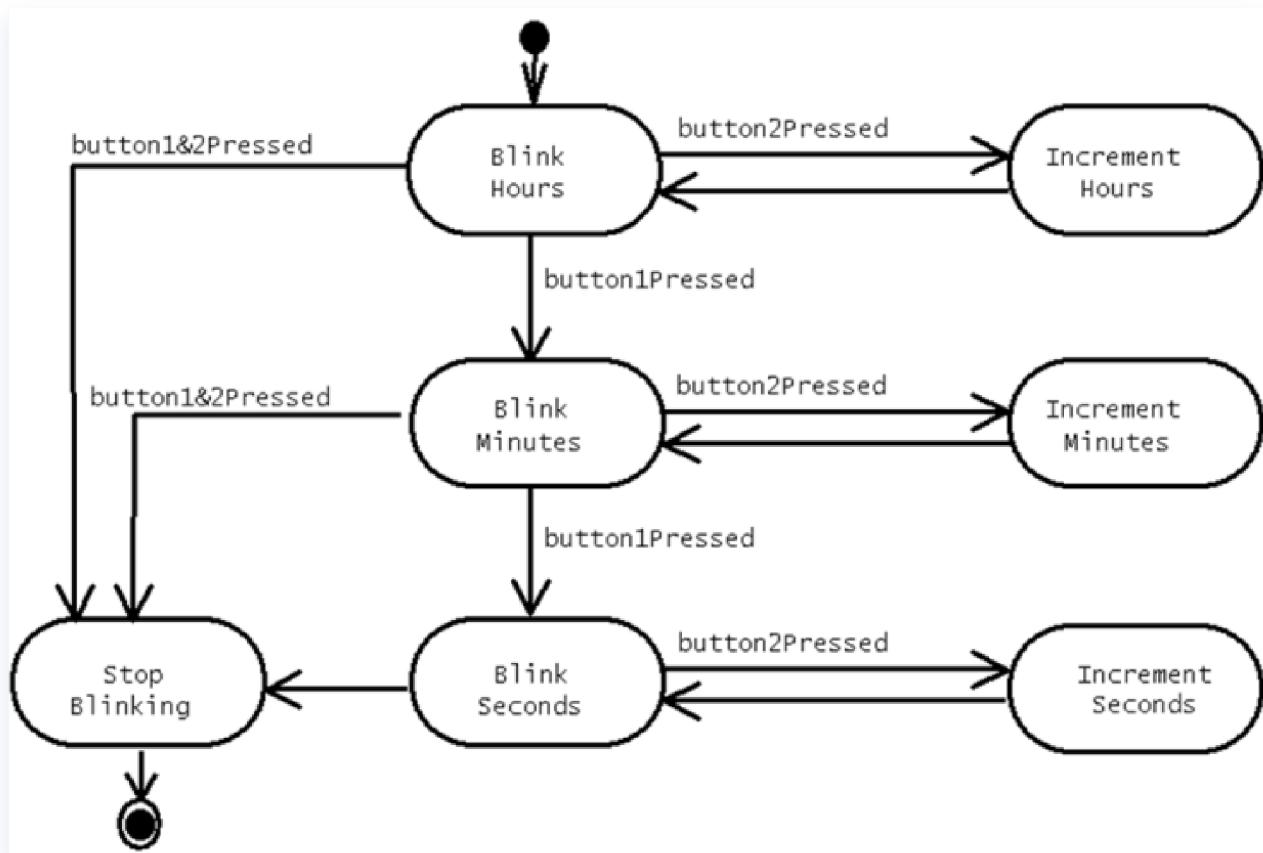
顺序图



协作图



状态图



活动图