

问题：软件危机

- 软件危机定义

在计算机软件开发和维护过程中遇到的一系列严重问题。

- 软件危机包含两方面的内容

- (1) 如何开发软件，以满足对软件日益增长的需求；
- (2) 如何维护数量不断膨胀的已有软件。

60年代后期开始认真研究解决软件危机的方法，从而逐步形成计算机科学领域中的一门新兴学科——**计算机软件工程**。

软件危机的典型表现

- (1) 对软件开发成本和进度的**估计**常常很不准确。
- (2) 用户对“已完成的”软件系统**不满意**的现象 经常发生。
- (3) 软件产品的**质量**往往靠不住。
- (4) 软件常常是不可**维护**的。
- (5) 软件通常没有适当的**文档**资料。
- (6) 软件**成本**在计算机系统总成本中所占的比例逐年上升。
- (7) 软件开发生产率提高的**速度**，远远跟不上计算机应用迅速普及深入的趋势。

产生软件危机的原因

与软件本身的特点有关

软件缺乏“可见性”——管理和控制软件开发相当困难
规模庞大，且程序复杂性将随着程序规模的增加呈指数上升。

与软件开发与维护的认识和观念不正确有关

忽视软件需求分析的重要性——许多软件开发工程失败的主要原因之一

认为软件开发就是写程序并设法使之运行，没有足够的文档，
轻视软件维护。——**软件产品=程序+文档+数据**

软件开发与维护的方法不正确

对系统需求没有清楚和准确的认识就进入开发阶段
忽视对软件开发过程的管理

结论

- 软件开发不是某个程序员的个人的抽象思维过程或编程技巧。
- 软件开发应该是一种组织良好，管理严密，各类人员协同配合，共同完成的工程项目。
- 以工程的原理，原则和方法进行软件开发。

问题：软件工程的定义

- 软件工程是指导计算机软件开发和维护的工程学科。它采用工程的概念、原理、技术和方法来开发与维护软件，结合管理技术和技术方法。
- 软件工程是从管理和技术两方面研究如何更好地开发和维护计算机软件的一门新兴学科。
- 软件工程是用科学知识和技术原理来定义、开发、维护软件的一门科学。

1.3 软件生命周期

- 什么叫软件生命周期
一个软件从开始计划起，到废弃不用为止。
- 软件生命周期的三个时期：
软件定义，软件开发，软件维护。

问题：软件生命周期各阶段的目标是什么

软件定义时期的任务

- 确定软件开发工程必须完成的总目标；
- 确定工程的可行性；
- 导出实现工程目标应该采用的策略及系统必须完成的功能；
- 估计完成该项工程需要的资源和成本，并且制定工程进度表。

这个时期的工作通常又称为系统分析，由系统分析员负责完成。

软件定义时期通常进一步划分成3个阶段，即问题定义、可行性研究和需求分析。

开发时期的任务

具体设计和实现在前一个时期定义的软件。

通常由下述4个阶段组成：

总体设计，详细设计，编码和单元测试，综合测试。

系统设计

系统实现

维护时期的任务

主要是使软件持久地满足用户的需要。具体地说，

- 当软件在使用过程中**发现错误**时应该加以改正；
- 当**环境改变**时应该修改软件以适应新的环境；
- 当**用户有新要求**时应该及时改进软件以满足用户的新需要。

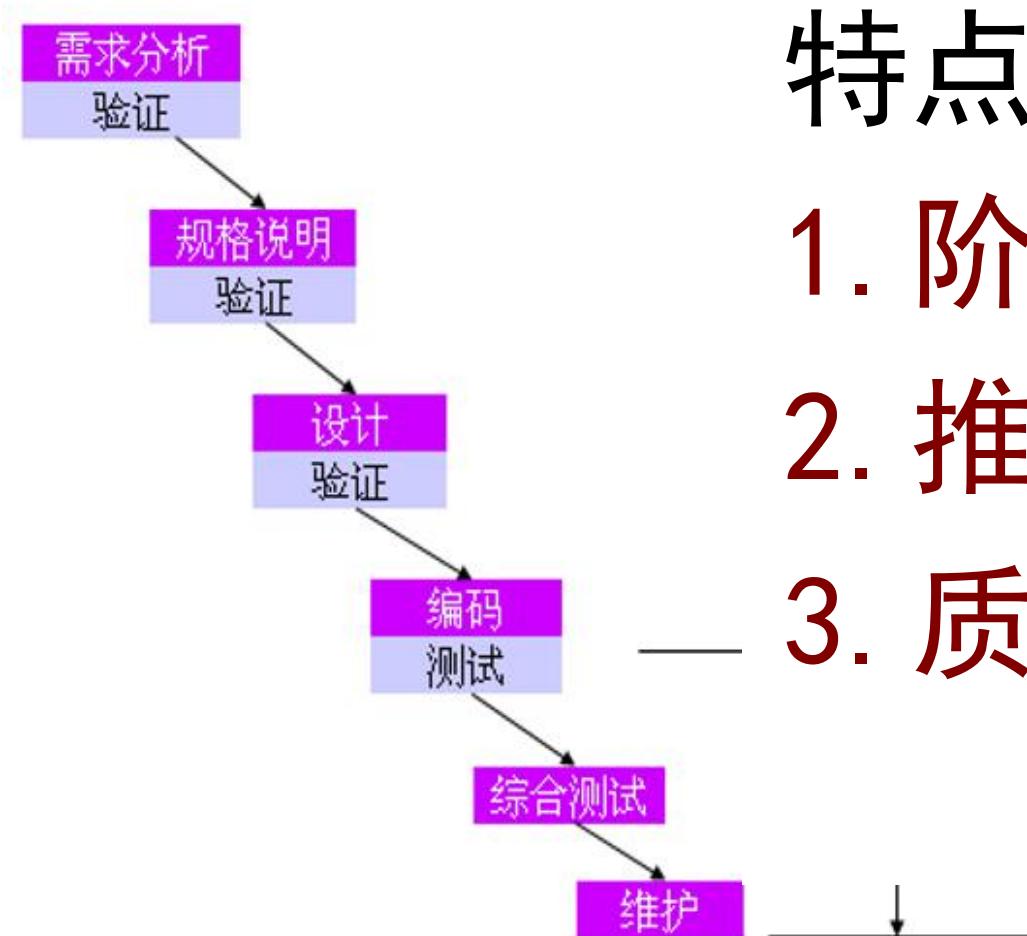
每一次维护活动本质上都是一次压缩和简化了的定义和开发过程。

问题：软件过程模型都有哪些，各自的特点是什么

常用软件过程模型

- 瀑布模型
- 快速原型模型
- 增量模型
- 螺旋模型
- 喷泉模型
- Rational 统一过程
- 敏捷过程与极限编程
- 微软公司软件开发过程

(1) 瀑布模型



特点：

1. 阶段间具有顺序性和依赖性
2. 推迟实现的观点
3. 质量保证的观点

图2 传统的瀑布模型

(2) 快速原型模型

不带反馈环

本质：“快速”

用途：获取用户的真正需求

必须迅速地构建原型后根据用户意见
迅速地修改原型

快速原型的部分可以用到最终软件中

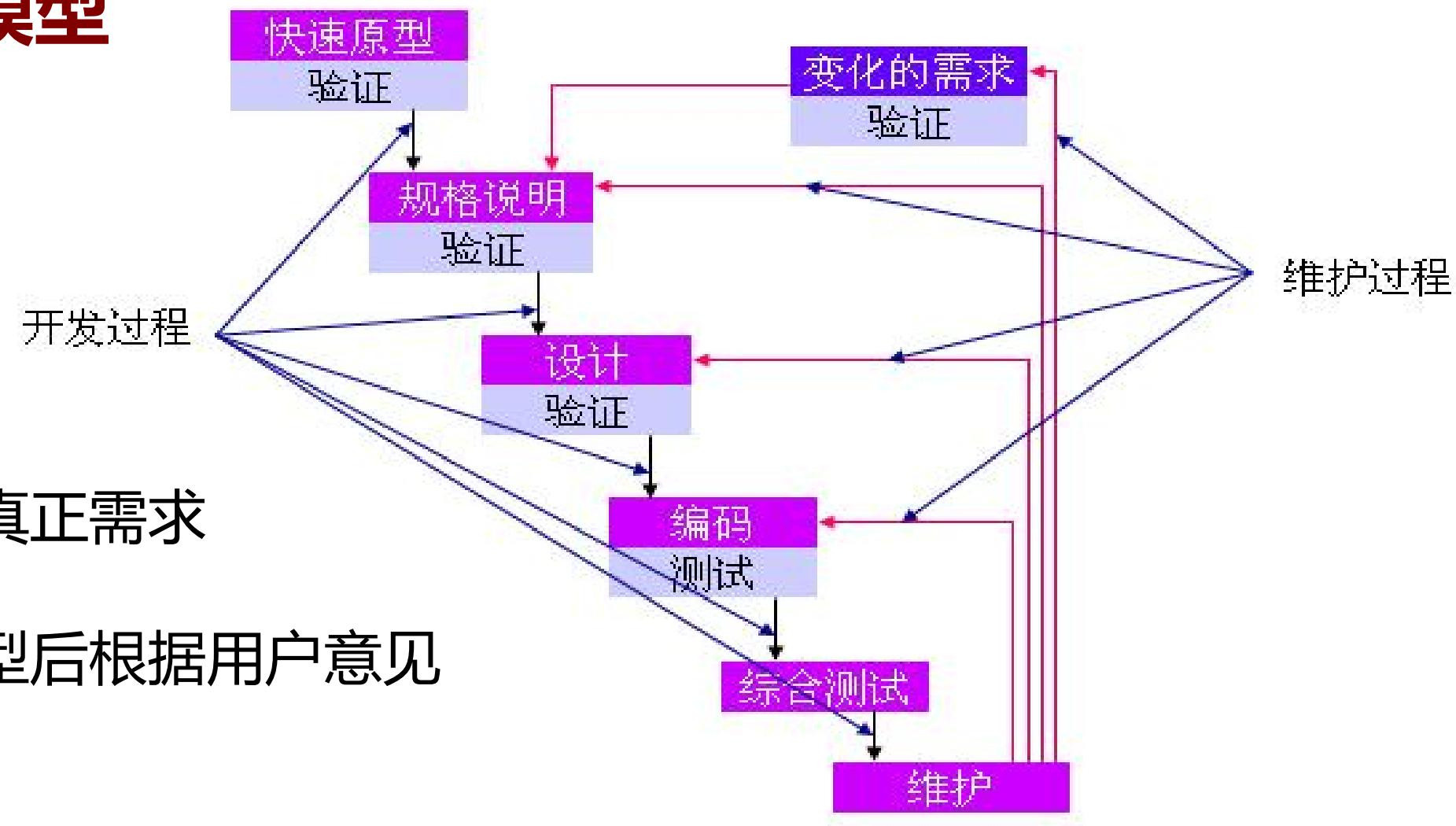


图4 快速原型模型

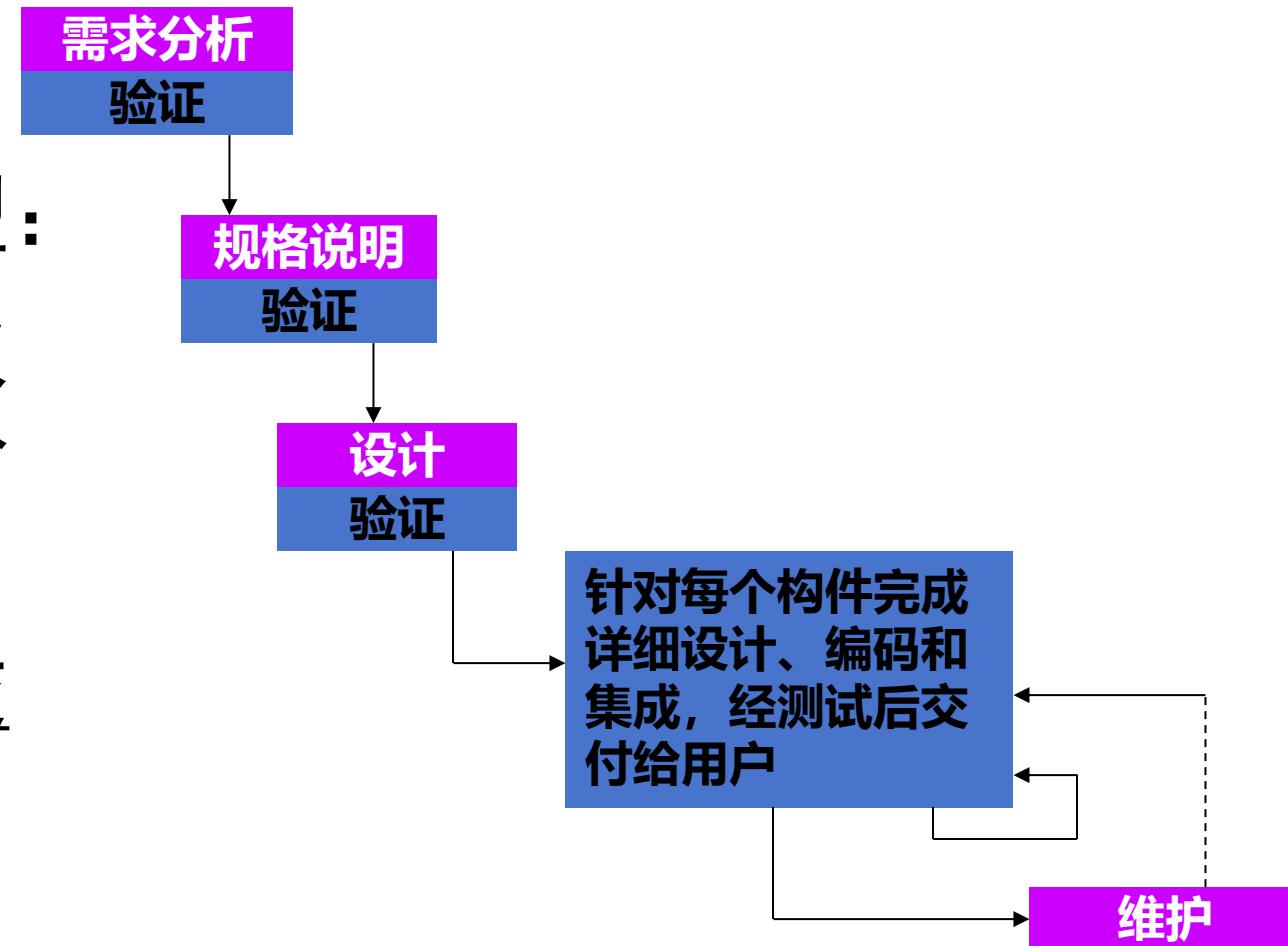
(3) 增量模型 (渐增模型)

是一种渐进地开发逐步完善的软件版本的模型。

区别于瀑布和快速原型模型：

瀑布和快速原型模型是一次把满足所有需求产品提交给用户。

增量模型是分批向用户提交产品。



(4) 螺旋模型

螺旋模型将瀑布模型和增量模型结合起来，加入了风险分析。

螺旋模型的基本思想是降低风险。

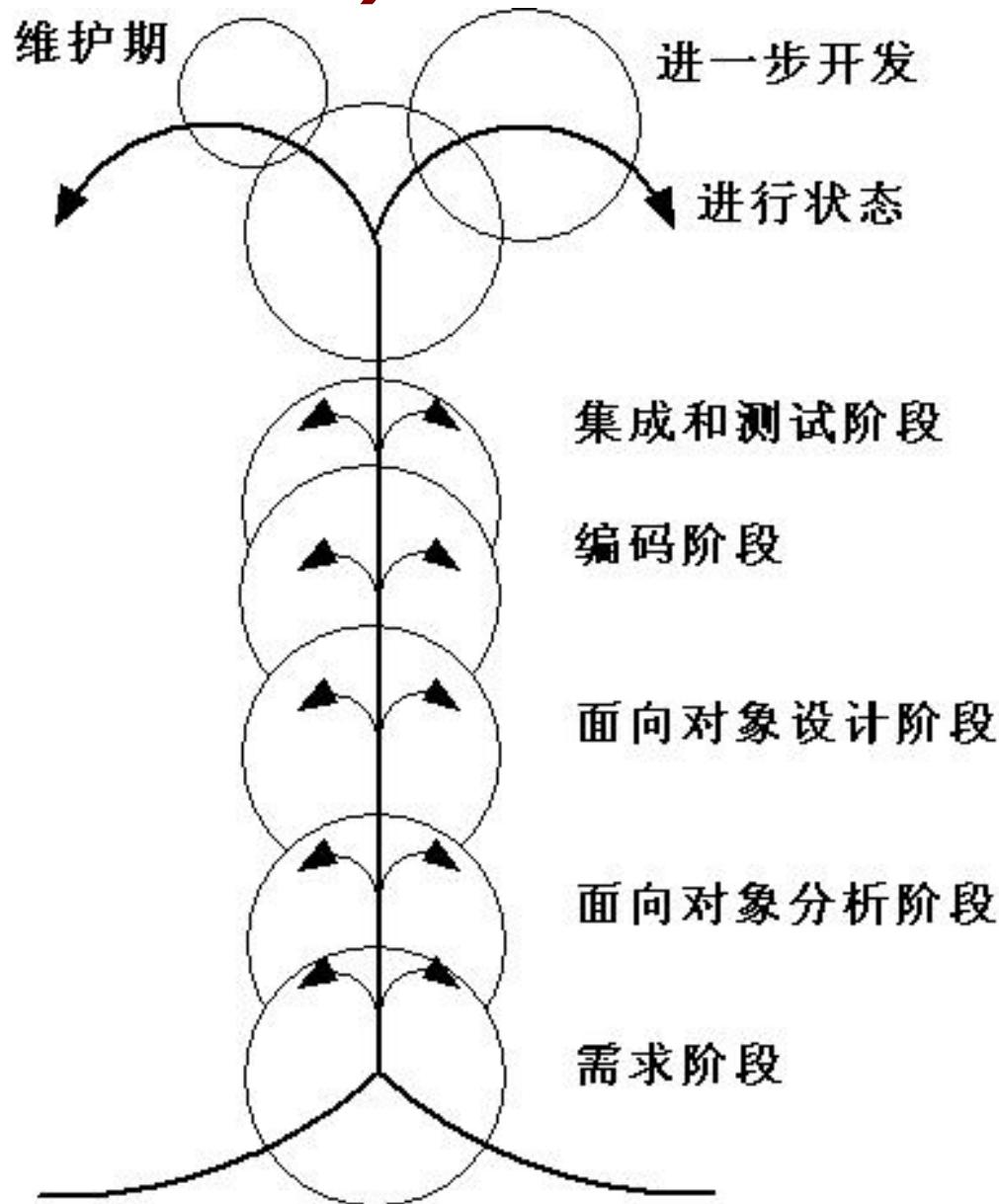
简化的螺旋模型可看作在每个阶段之前都增加了风险分析过程的快速原型模型。

螺旋模型的特点

- 风险驱动，需要相当丰富的风险评估经验和专门知识，否则风险更大
- 主要适用于内部开发的大规模软件项目，随着过程的进展演化，开发者和用户能够更好的识别和对待每一个演化级别的风险
- 随着迭代次数的增加，工作量加大，软件开发成本增加

(5) 喷泉模型（面向对象模型）

特点：主要用于支持面向对象开发过程体现了软件创建所固有的**迭代**和**无缝**的特征。



问题：数据流图的基本组成和用途

2.4 数据流图（DFD -- Data Flow Diagram）

- 描绘信息流和数据从输入移动到输出的过程中所经受的变换。
- 基本组成：源点或终点、处理、数据存储、数据流
- 没有任何具体的物理部件，它只是描绘数据在软件中流动和被处理的逻辑过程，是系统逻辑功能的图形表示。

用途：

- 一. 作为交流信息的工具
- 二. 作为分析和设计的工具

问题：掌握分层数据流图的优点和遵循的原则，如课本2.4.2的例题

分层数据流图

为表达数据加工情况，需采用层次结构数据流图。

顶层数据流图包含一个加工项；

底层流图指加工项不再分解的数据流图；

中间层流图只在顶层和底层之间，对其上层父图的细化。

分层 DFD 图的优点

- . 便于实现 —— 采用逐步细化的扩展方法，可避免一次引入过多的细节，有利于控制问题的复杂度；
- . 便于使用 —— 用一组图代替一张总图，方便用户及软件开发人员阅读。

画分层 DFD 的指导原则

1. 注意数据流图中成分的命名
2. 注意父图和子图的平衡
3. 掌握分解的速度

一般来说，每一个加工每次可分为 2-4个子加工，最多不得超过7个。

4. 遵守加工编号规则

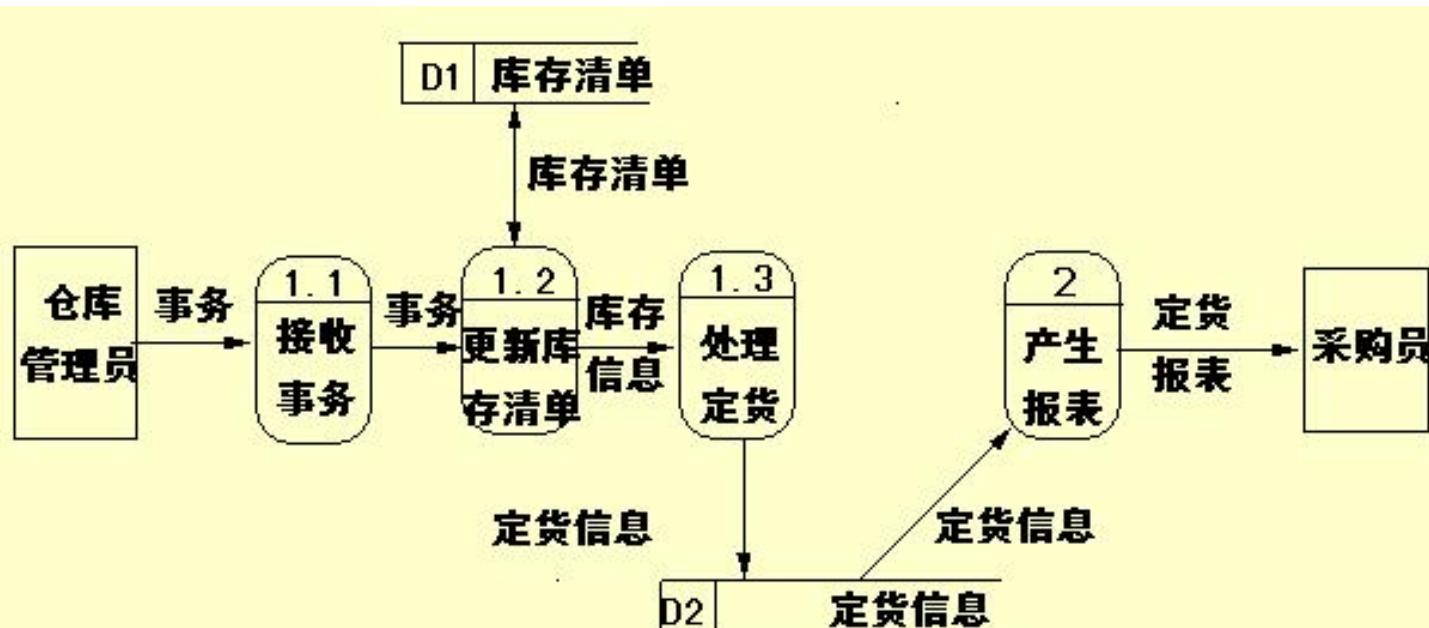
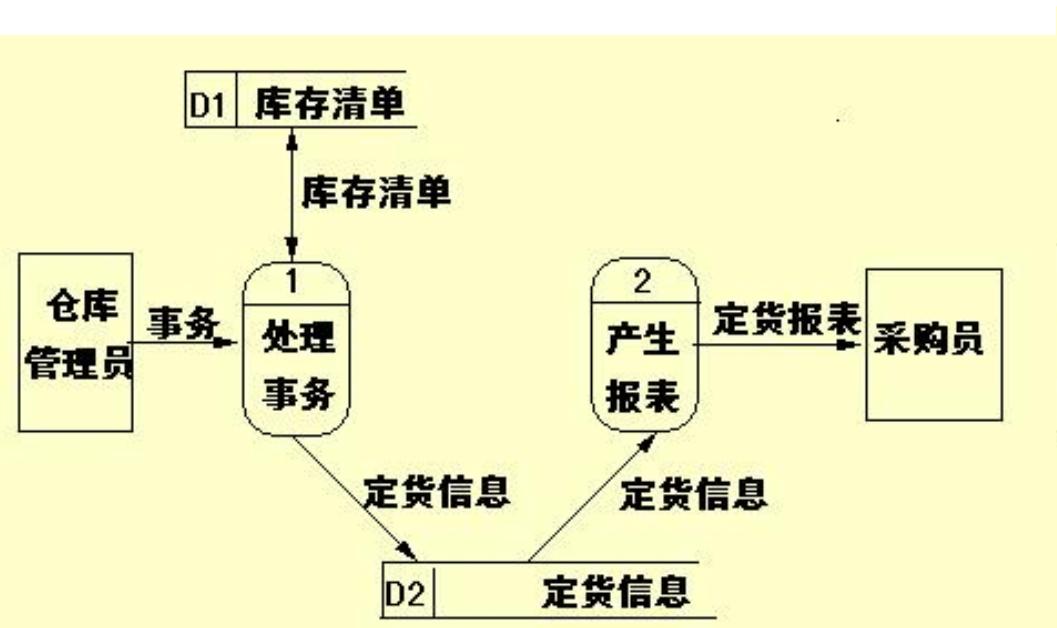
顶层加工不编号。第二层的加工编号为1, 2, 3, …, n号。第三层编号为1. 1, 1. 2, 1. 3…n. 1, n. 2…等号，依此类推。

2.4.2例子—订货系统

某工厂有一个库房，存放该厂生产需要的物品，库房中的各种零件的数量及各种物品库存量临界值等数据记录在库存文件中。



当库房中零件数量有变化时，应更新库存文件，若某种零件的库存量少于库存临界值，则报告采购部门以便订货，每天向采购部门送一份采购报告。



问题：数据字典的基本组成和用途

2.5 数据字典 & 用途-- DD (Data Dictionary)

数据流图和数据字典共同构成系统的逻辑模型。

数据流图和数据字典共同构成全面描述软件需求说明书的核心。

- 基本组成：数据流、数据元素、数据存储、处理
- 用途：
- 数据字典 (data dictionary) 记录有关数据方面的信息，作为对数据流图的补充和解释。
- 数据字典的任务是对于数据流图中出现的所有被命名的图形元素，包括数据流，数据元素，数据存储和处理，在数据字典中作为一个词条加以定义。

问题：模块独立程度的度量标准

5.2.5 模块独立性

两个定性度量标准：耦合和内聚。（低耦合，高内聚）

耦合：定义：软件结构中不同模块间互连程度度量。

取决于模块间接口复杂程度，通过接口数据追求尽可能松散耦合系统。

- 非直接耦合、数据耦合、控制耦合、特征耦合、公共环境耦合

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。

模块独立--内聚

定义：模块内各元素彼此结合紧密程度。

- 功能内聚：10分
- 顺序内聚：9分
- 通信内聚：7分
- 过程内聚：5分
- 时间内聚：3分
- 逻辑内聚：1分
- 偶然内聚：0分

设计原则：力求做到高内聚，如：功能内聚、顺序内聚、通信内聚
不要使用低内聚，如：时间内聚、逻辑内聚和偶然内聚。

问题：总体设计的图形工具

5.4 描绘软件结构的图形工具

- 层次图

- 层次图用来描绘软件的层次结构。
- 层次图中的一个矩形框代表一个模块，方框间的连线表示调用关系而不像层次方框图那样表示组成关系。

- HIPO图

在H图(层次图)里除了最顶层的方框之外，每个方框都加了编号。

- 结构图

一个方框代表一个模块；方框之间的箭头(或直线)表示模块的调用关系。

空心圆表示传递数据；实心圆表示传递控制信息

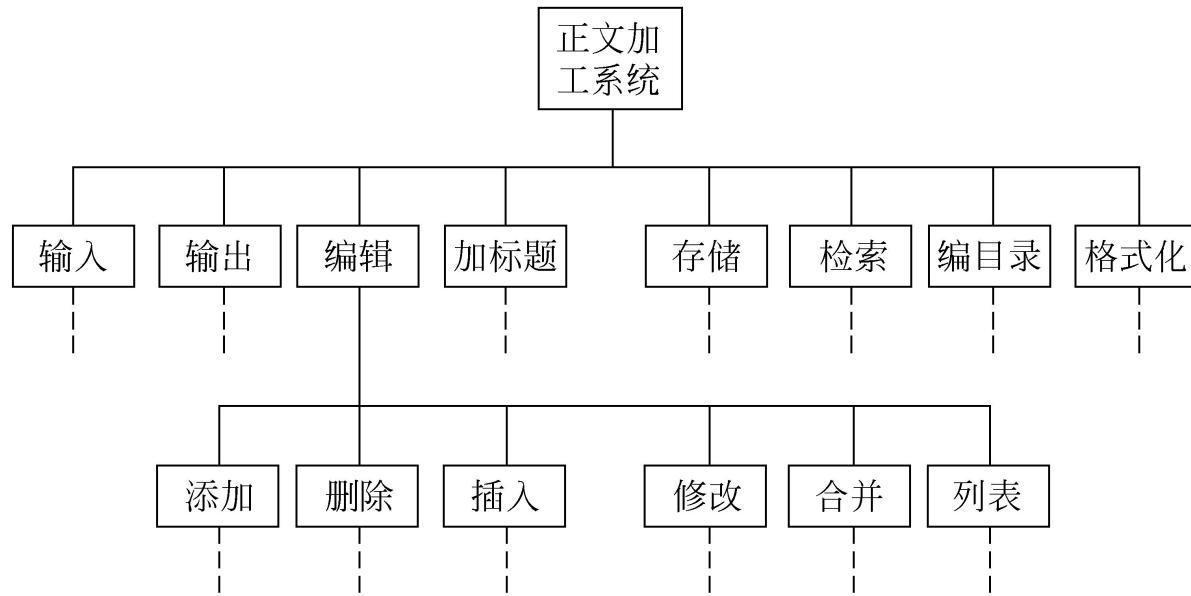


图5.3 正文加工系统的层次图

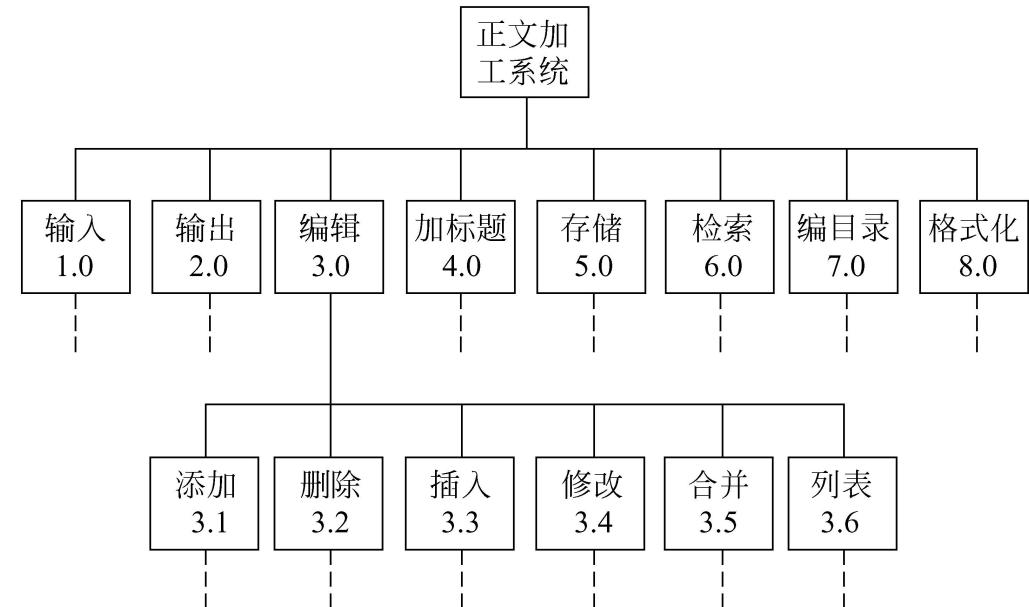
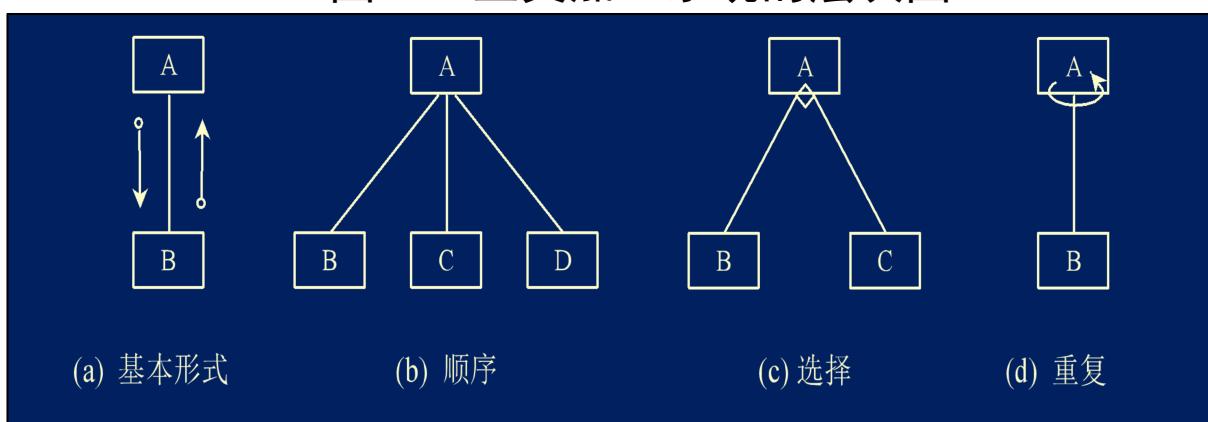
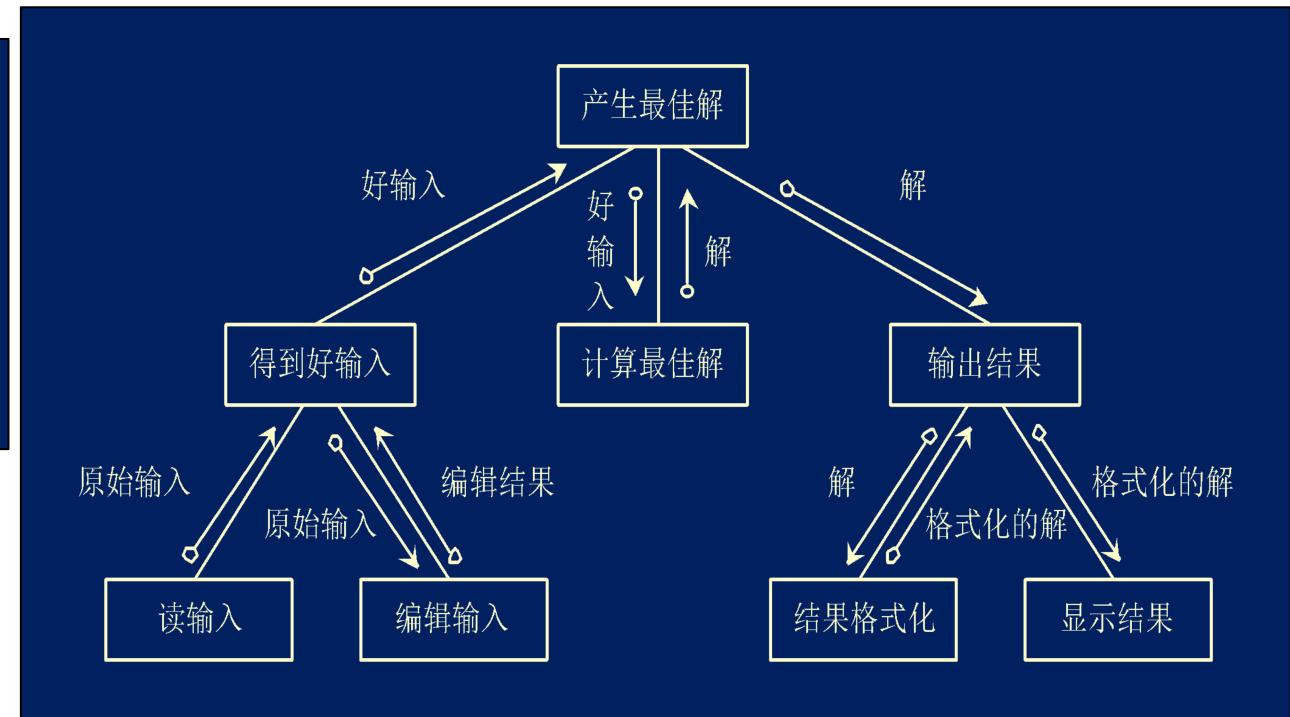


图5.4 带编号的层次图 (H图)



软件结构图的基本符号

图5.5 结构图的例子—产生最佳解的一般结构



问题：面向数据流的设计方法

5.5 面向数据流的设计方法

- 信息流类型：变换流，数据流
- 面向数据流设计过程
- 变换分析
- 事务分析
- 设计优化

变换分析：

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. 完成一级分解
6. 完成二级分解
7. 对初步软件结构精化

事务分析：

有明显事务特点（事务中心），采用事务分析方法。
软件结构：一接收分支和一发送分支。

**问题：详细设计的图形工具；
会画程序流程图、盒图、PAD图、判定表、判定树**

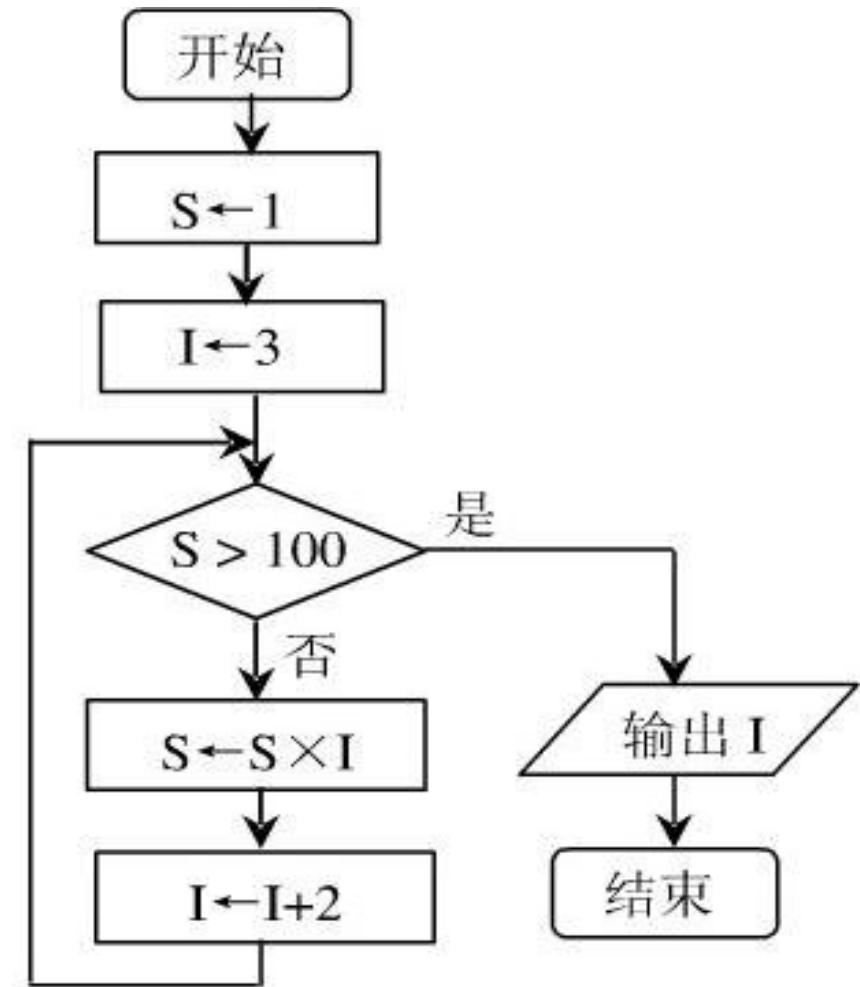
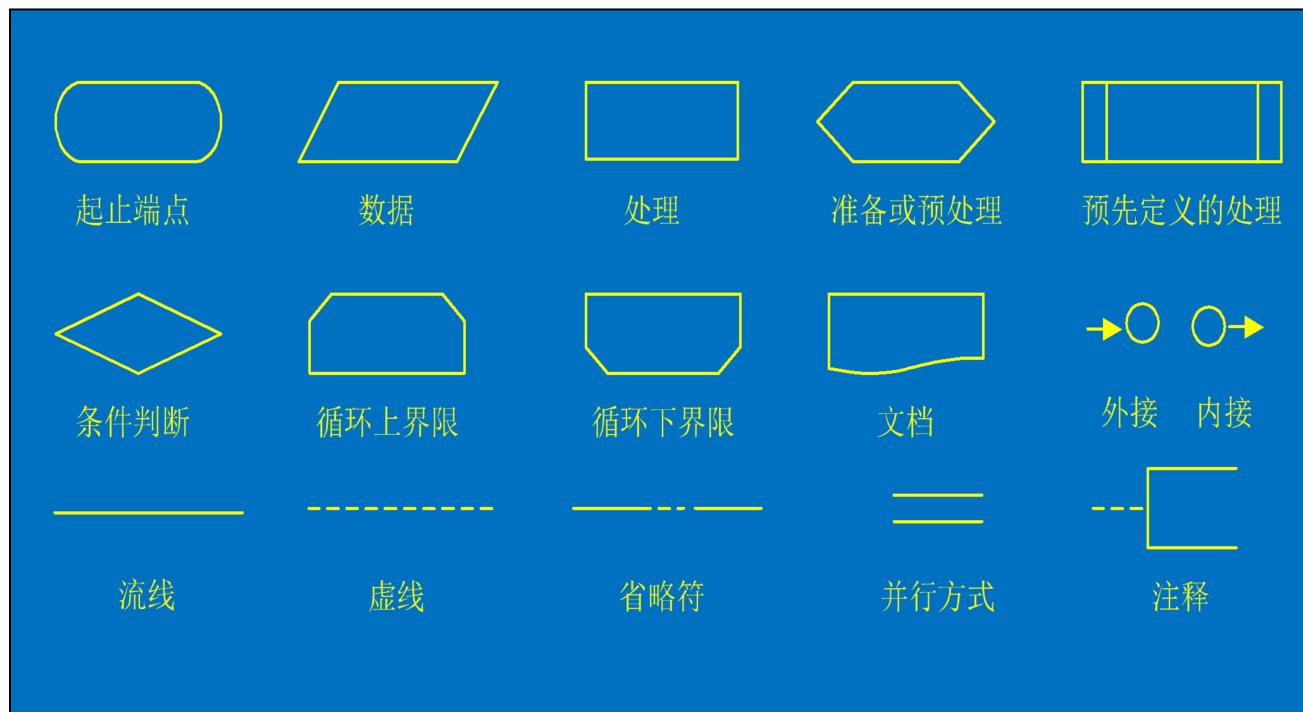
6.3 过程设计的工具

- 程序流程图
- 盒图
- PAD图
- 判定表
- 判定树
- 过程设计语言

6.4 面向数据结构的程序设计方法

Jackson图

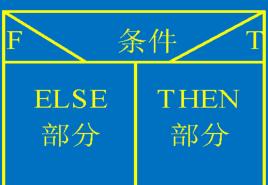
6.3.1 程序流程图



盒图练习 (求素数、求方程的解)

第一个任务
第二个任务
第三个任务

(a) 顺序结构



(b) 选择结构



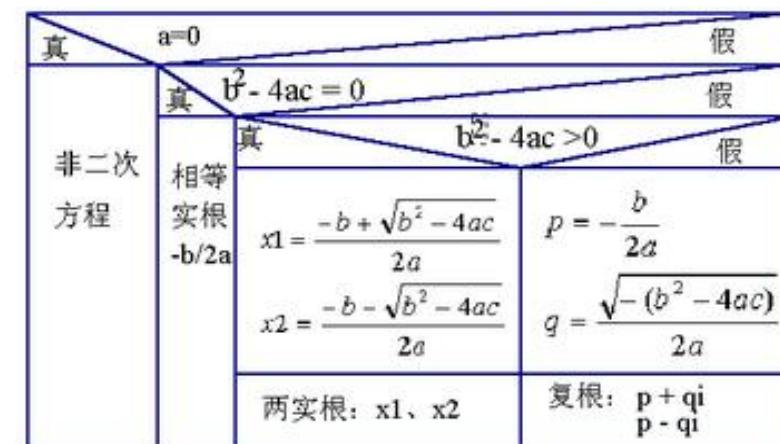
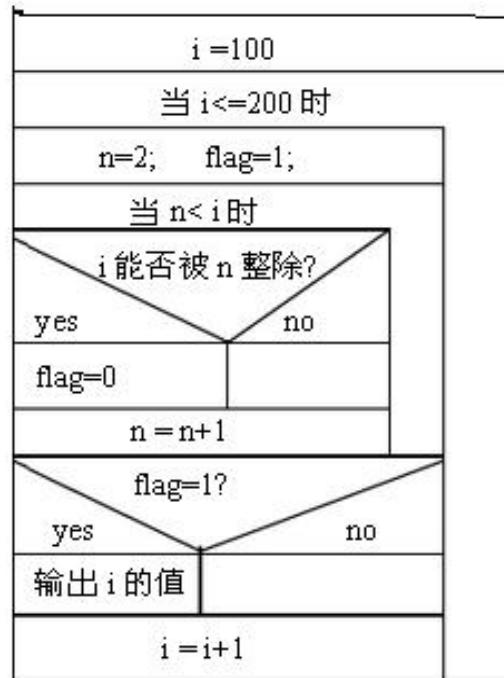
(c) 多分支结构



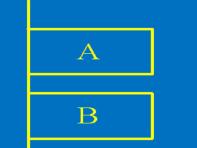
(d) 循环结构



(e) 调用子程序 A



6.3.3 PAD图--基本符号



(a) 顺序结构



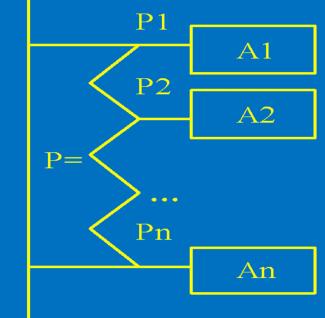
(b) 选择结构



(c) WHILE型循环结构



(d) UNTIL型循环结构



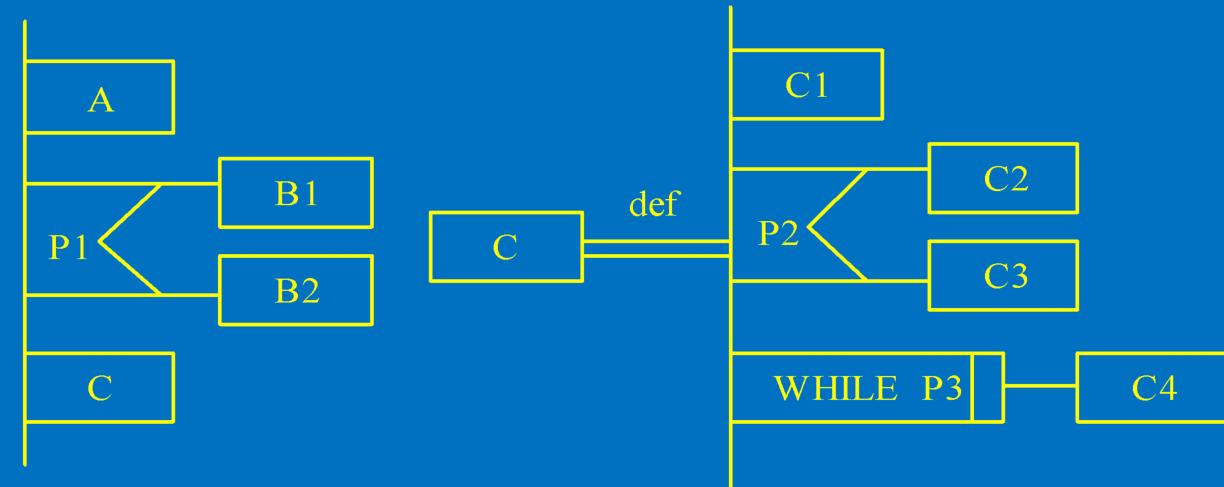
(e) 多分支结构



(f) 语句标号



(g) 定义



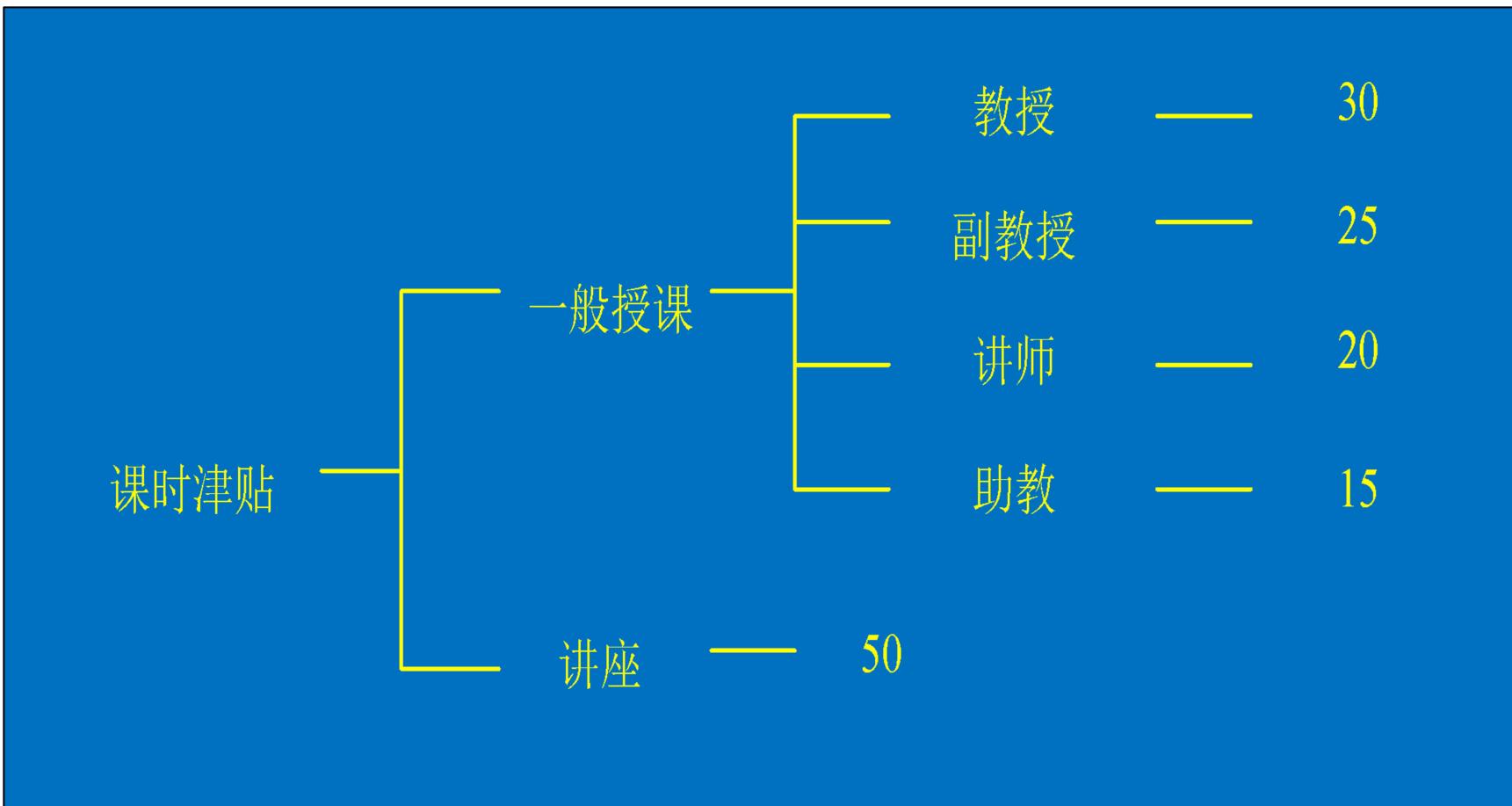
PAD图提供的定义功能

6.3.4 判定表

所有条件	1	2	3	4	5
可能的动作	T	F	F	F	T
	F	T	F	T	F
	F	F	F	F	T
	T	F	F	F	F
	X				
		X			
			X		
				X	
					X

	1	2	3	4	5
教		T	F	F	F
授教		F	T	F	F
譖		F	F	T	F
頑		F	F	F	T
躉	T	F	F	F	F
50	X				
30		X			
25			X		
20				X	
15					X

6.3.4 判定树



6.3.6 过程设计语言（PDL）

- PDL也称为伪码，是用正文形式表示数据和处理过程的设计工具。
 - PDL具有严格关键字外部语法，定义控制结构和数据结构；
 - PDL表示实际操作和条件的内部语法灵活自由，适应各种工程项目需要。
-
- 优点：
可作为注释直接插在源程序中； PDL的编辑方便； 已有自动处理程序存在。
 - 缺点：
不如图形工具形象直观； 不如判定表清晰。

6.4.2 改进的Jackson图

改进的Jackson图（一）



B

C

D



S(i)

B^o

C^o

D^o

(a) 顺序结构



S(i)

B^o

-^o



I(i)

B*

(c) 可选结构

(d) 重复结构

问题：白盒测试方法

7.6 白盒测试技术

7.6.2 控制结构测试

◆ 测试用例: 测试输入数据和预期的输出结果。

点覆盖

◆ 测试方案: 测试目的、测试用例的集合。

边覆盖

◆ 语句覆盖: 被测试程序中的每条语句至少执行一次。

路径覆盖

◆ 判定覆盖: 使得被测程序中每个判定表达式至少获得一次“真”值和“假”值

◆ 条件覆盖: 使得判定表达式中每个条件的各种可能的值至少出现一次。

◆ 判定/条件覆盖: 使得判定表达式中的每个条件的所有可能取值至少出现一次，并使每个判定表达式所有可能的结果也至少出现一次。

◆ 条件组合覆盖: 设计足够多的测试用例，使得每个判定表达式中条件的各种可能的值的组合都至少出现一次。

◆ 路径覆盖: 覆盖被测程序中所有可能的路径

1. 基本路径测试

2. 条件测试

3. 循环测试

问题：黑盒测试方法7.7 黑盒测试技术

- 黑盒测试着重测试软件功能。黑盒测试并不能取代白盒测试，它是与白盒测试互补的测试方法，它很可能发现白盒测试不易发现的其他类型的错误。
- 黑盒测试力图发现下述类型的错误：①功能不正确或遗漏了功能；②界面错误；③数据结构错误或外部数据库访问错误；④性能错误；⑤初始化和终止错误。
- 黑盒测试技术：等价划分法、边界值分析法、错误推测法、因果图法等。
- 着重测试软件的功能。黑盒测试又称功能测试，
- 等价类划分法
 - (1) 把程序的输入数据集合按输入条件划分为若干个等价类，每一个等价类相对于输入条件表示为一组有效或无效的输入。(2) 为每一等价类设计一个测试用例。
- 边界值分析法
 - 输入等价类和输出等价类的边界就是应该着重测试的程序边界情况。选取的测试数据应该刚好等于、刚好小于、刚好大于边界值
- 错误推测法
 - 根据经验、直觉和预感来进行测试

问题：软件维护的定义和分类；

8.1 软件维护的定义

软件维护就是在软件已经交付使用之后，为保证软件在相当长的时期能够正常运作所进行的软件活动。

软件工程的主要目的就是要提高软件的可维护性，减少软件维护所需的工作量，降低软件系统的总成本。

维护的类型有四种：

改正性维护；适应性维护；扩充与完善性维护；预防性维护。

改正性维护

- 为了识别和纠正软件错误、改正软件性能上的缺陷、排除实施中的误使用，所进行的诊断和改正错误的过程就叫做改正性维护。

适应性维护

为使软件适应外部环境（新的硬件、软件配置）、数据环境（数据库、数据格式、数据输入/输出方式、数据存储介质）等可能发生的变化，而去修改软件的过程就叫做适应性维护。

扩充与完善性维护

为了满足新的功能与性能要求，需要修改或再开发软件，以扩充软件功能、增强软件性能、改进加工效率、提高软件的可维护性。的情况下进行的维护活动叫做扩充与完善性维护。

预防性维护

预防性维护是为了提高软件的可维护性、可靠性等，为以后进一步改进软件打下良好基础。

问题：软件测试和调试的目标；

7.2.1 软件测试的目的

- 测试阶段的根本目标是尽可能多地发现并排除软件中潜藏的错误，最终把一个高质量的软件系统交给用户使用。

7.8 调试

- 调试是在测试发现错误之后排除错误的过程。

问题：提高软件质量的启发式规则

5.3 启发规则

1. 尽力提高模块独立性
2. 选择合适的模块规模（50—100行）
3. 模块的深度、宽度、扇出和扇入应适当，平均的扇出为3或4，上限为5--9
4. 模块的作用范围应该在控制范围之内
5. 降低模块接口的复杂程度
6. 设计单入口单出口的模块，避免“病态连接”
7. 模块功能应该可以预测

问题：面向对象方法学的基本概念包括：对象，类，实例，消息，方法，继承，多态性，重载；

对象：具有相同状态的一组操作的集合，对状态和操作的封装。

类：具有相同数据和相同操作的一组相似对象；

实例：由某个特定的类所描述的一个具体的对象；

消息：用来请求对象执行某个处理或回答某些信息的要求；是要求某对象执行某个操作的规格说明。

方法：对象所能执行的操作；

属性：类中定义的数据；

封装：信息隐藏，对外界隐藏了对象的实现细节；

继承：能够直接获得已有的性质和特征，而不必重复定义它们；

多态性：子类对象可以象父类对象那样使用，同样的消息既可以发送给父类对象，也可以发送给子类对象；

重载：函数重载和运算符重载。

问题： 面向对象方法需要建立的3个模型， 功能和建立步骤；

用面向对象方法开发软件， 通常需要建立3种形式的模型， 它们分别是

1. 对象模型： 描述系统数据结构；（**最基本、最重要的**）
2. 动态模型： 描述系统控制结构；
3. 功能模型： 描述系统功能的。

面向对象建模

对象模型

- (1) 表示静态的，结构化的系统的“数据”性质；
- (2) 是对对象及对象之间关系的映射；
- (3) 描述系统的静态结构。

9.5 动态模型

动态模型:表示瞬时的、行为化的系统的“控制”性质，
它规定了对象模型中的对象的合法变化序列。

- 1) **事件:** 事件是某个特定时刻所发生的事情。它是引起
对象状态转换的控制信息。
- 2) **状态:** 状态就是对象在其生命周期中的某个特定阶段
所处的某种情形。
- 3) **行为:** 行为是指对象达到某种状态时所做的一系列处
理操作。

9.5 动态模型（续）

- 通常用UML提供的**状态图**来描绘动态模型。

- **注意：**

每个类的动态行为用一张状态图来描绘，各个类的状态图通过共享事件合并起来，从而构成系统的动态模型。

9.6 功能模型

- 通常，功能模型由一组数据流图组成。但在面向对象的方法学中，数据流图远不如结构分析和方法中那么重要了。
- UML提供的用例图也是进行需求分析和建立功能模型的有力工具。
- 以用例图建立起来的系统模型称为用例模型，它描述的是外部行为者所理解的系统功能。
- 用例模型描述的是每一类用户“做什么”

问题：类图的相关概念并会画类图；

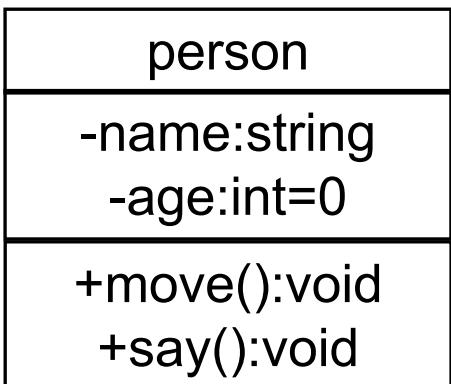
UML类图

类图描述系统中类的静态结构。是面向对象建模最常用的图，描述类与类间的静态关系。

一、类图的基本符号

1. 定义类

类名
属性
服务



2. 定义属性

可见性 属性名：类型名=初值{性质串}

公有（public）/+、私有（private）/-、保护（protected）/#

3. 定义服务

可见性 操作名（参数表）：返回值类型{性质串}

-Show(x:integer=0, y :integer, z :integer): integer

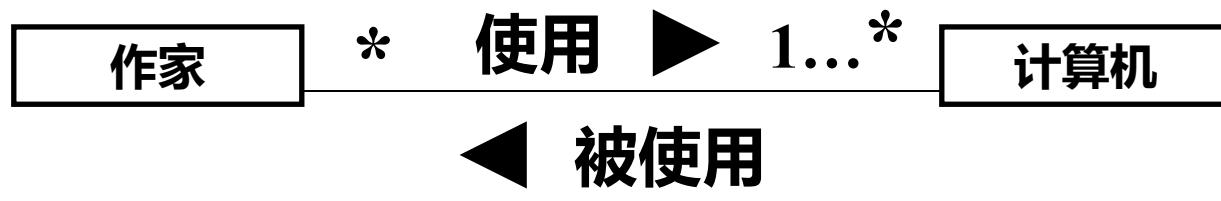
UML的类图

类和类之间的关系

- 关联
- 泛化 (继承)
- 依赖
- 细化 (实现)

1. 关联：两类对象间存在某种语义联系

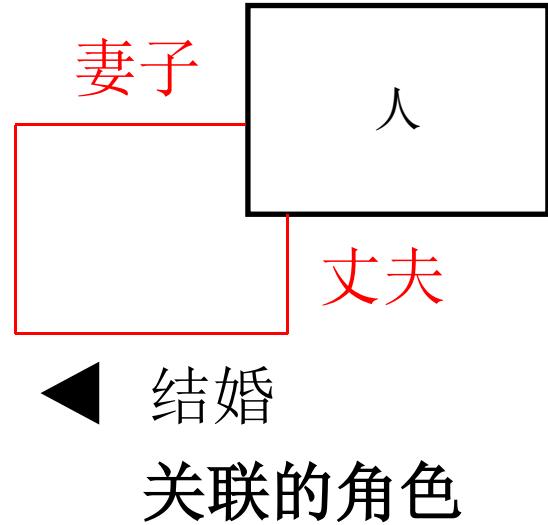
1) 普通关联：双向，用实线连接两个类。



普通关联系例

- 重数 (multiplicity) 的表示方法：
- 0...1 表示 0到1个对象；
- 0...* 或 * 表示 0到多个对象；
- 1+ 或 1...* 表示 1到多个对象；
- 1...15 表示 1到15个对象；
- 3 表示 3个对象。

2) 关联的角色



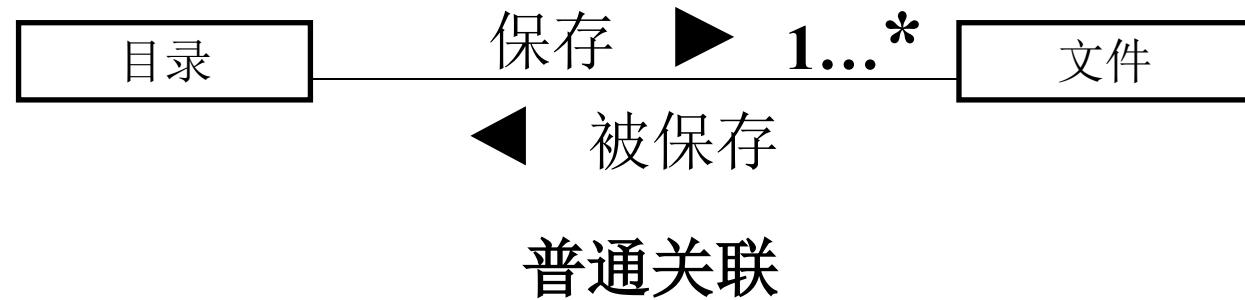
上图是一个递归关联的例子。

这种情况下，标明角色名有助于理解类图。

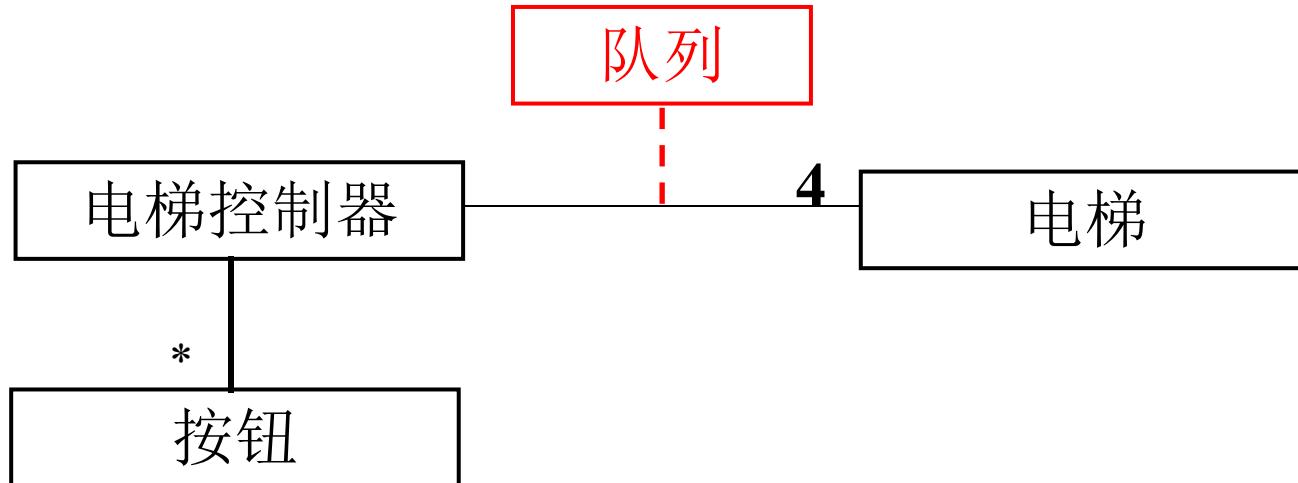
没有标出角色名，则意味着用类名作为角色名

3) 限定关联：限定符放在关联关系末端的矩形内。

利用限定词把一对多，多对多的关系简化成一对一关系



4) 关联类: 为了说明关联的性质, 需要一些附加信息



关联类示例

控制器对象和电梯对象之间的连接，对应着一个队列
(对象)，它存储着控制器和电梯内部按钮的请求信息。

2. 聚集：表示类与类之间是整体与部分的关系。

聚集是关联关系的特例。

1) 共享聚集

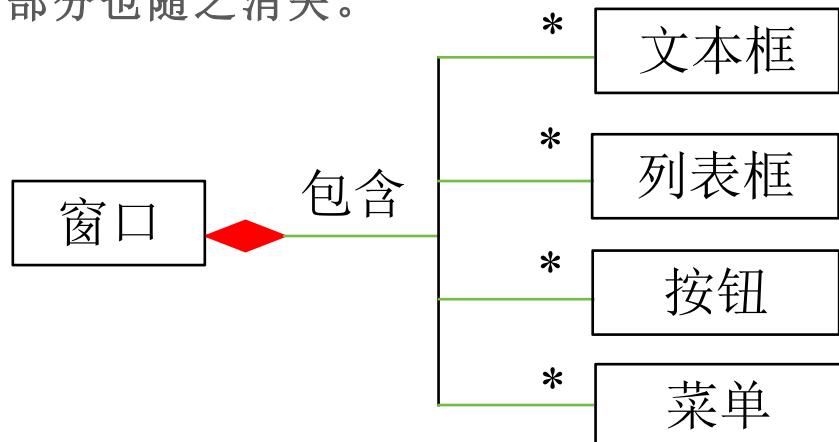
处于部分方的对象可同时参与多个
处于整体方对象的构成。



共享聚集示例 空心菱形

2) 组合聚集

部分类完全隶属于整体类，部分与整体共存，整体不存在了部分也随之消失。



组合聚类实例 实心菱形

3. 泛化（继承）：类与类之间的存在的“一般-特殊”关系。

1) 普通泛化

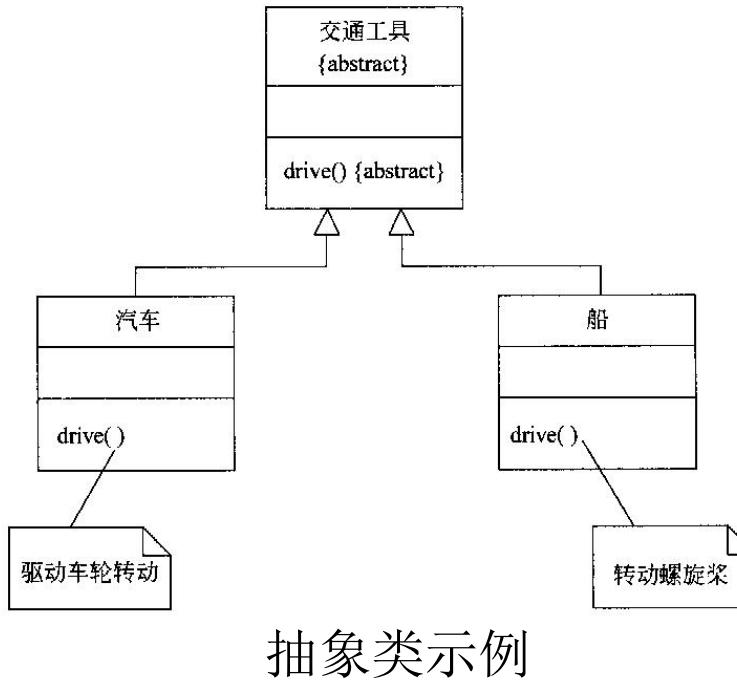
注意：

抽象类通常作为父类，描述子类的公共属性和行为。

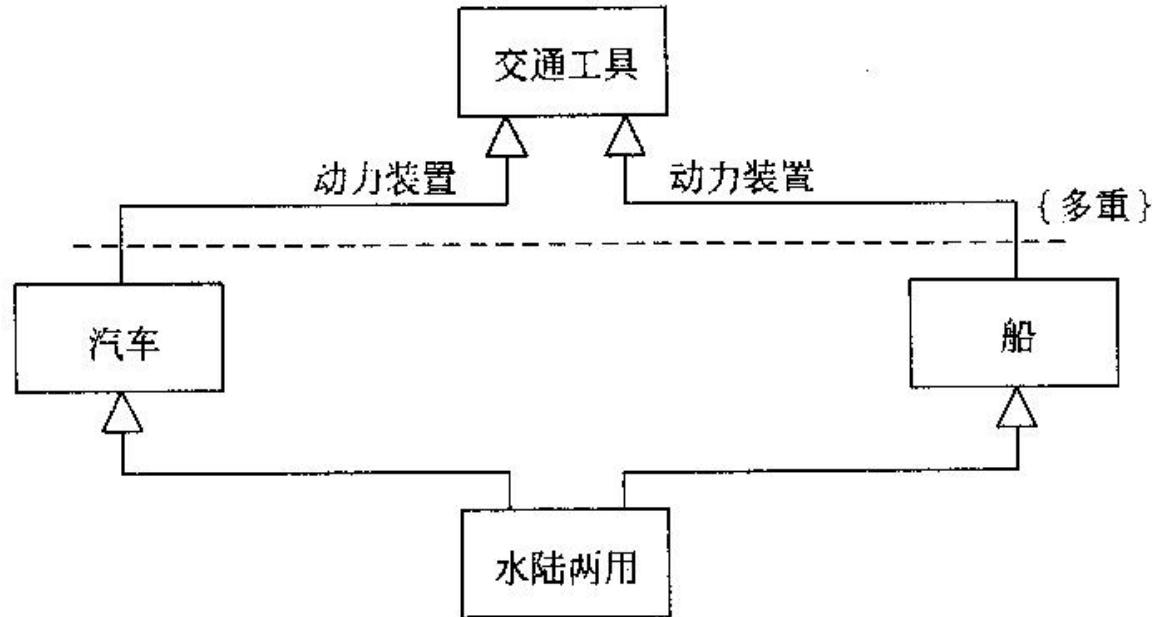
2) 受限泛化

给泛化关系（继承关系）附加约束条件，说明该泛化关系的使用方法和扩充方法。

约束：多重、不相交、完全和不完全。



多重继承：一个子类可以同时多次继承同一个上层基类。



- **完全继承**：指父类的所有子类都已经在类图中穷举出来了。
- **不完全继承**：指父类的所有子类并没有在类图中穷举出来了。随着对问题理解的深入，不完全继承中可以不断扩充子类。

(默认为不完全继承)

多重继承示例：水路两用类两次继承了交通工具类

不相交继承：与多重继承相反，一个子类不能多次继承同一个上层基类。 (默认为不相交继承)

4. 依赖和细化 箭头指向独立的类

1) 依赖关系

描述两个模型元素（类、用例）之间的关系，其中一个模型元素是独立的（被箭头指），另一个依赖于独立的模型元素。

例如：一个友元依赖关系，该关系使得B类的操作可以使用A类中的私有的或保护的成员。



友元依赖关系

虚线箭头

2) 细化关系：当对同一事物在不同抽象层次上描述时，这些描述之间就具有细化关系。



细化关系示例

空心虚线箭头

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模型之间的相关性，通常用于跟踪模型的演变。

●问题：用例图的四个模型要素并会画用例图；

四个模型要素：**系统、行为者、用例、用例间的关系**

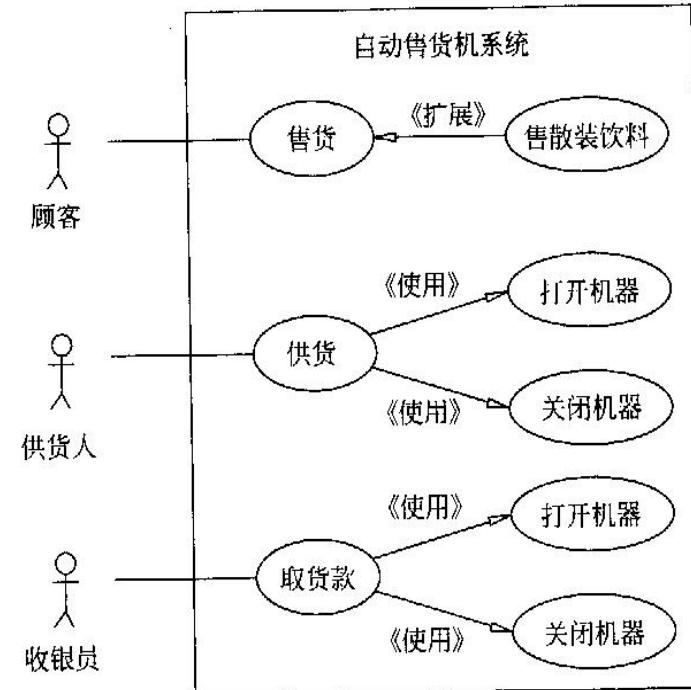
1、**系统**：提供用例的黑盒子

2、**行为者(actor)**:与系统交互的人或其他系统。它代表一种角色。(可以是人、外部硬件或其他系统)

3、**用例(use case)**:行为者感受到的一个完整的**功能**。

用例的特征：

- (1) 代表某些用户可见的功能，实现一个具体的用户目标。
- (2) 总是被行为者启动的，并向行为者提供可识别的值。
- (3) 必须的完整的。



4、用例之间的关系

1) 扩展关系

向一个用例中添加一些动作后构成另一个用例，它们之间构成扩展关系。描述一般行为的变化时采用**扩展关系**；

2) 使用关系

一个用例使用另一个用例，它们之间构成使用关系。两个或多个用例中出现重复描述时可采用**使用关系**。