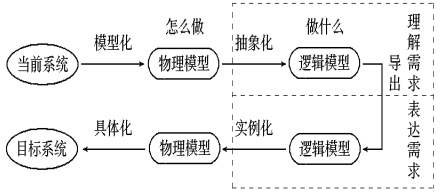
软件需求分析

软件需求分析的任务和过程
结构化分析方法
原型化方法
动态分析方法

软件需求分析的任务

- 深入描述软件的功能和性能
- 确定软件设计的约束和软件同其它系统元素的接口细节
- 定义软件的其它有效性需求

- 需求分析研究的对象是软件项目的用户要求
- 准确地表达被接受的用户要求
- 确定被开发软件系统的系统元素
- 将功能和信息结构分配到这些系统元素中



- 需求分析的任务就是借助于当前系统的逻辑模型导出目标系统的逻辑模型，解决目标系统的“做什么”的问题。

- 通常软件开发项目是要实现目标系统的物理模型
- 目标系统的具体物理模型是由它的逻辑模型经实例化，即具体到某个业务领域而得到的

需求分析的过程

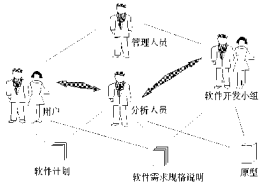
(1) 问题识别

- 从系统的角度来理解软件并评审 软件范围是否恰当
- 确定对目标系统的综合要求，即软件的需求
- 提出这些需求实现条件，以及需求应达到的标准

软件的需求包括：

- 功能需求
- 性能需求
- 环境需求
- 可靠性需求
- 安全保密要求
- 用户界面需求
- 资源使用需求
- 成本消耗需求
- 开发进度需求
- 预先估计以后系统可能达到的目标

问题识别的另一项工作是建立分析所需要的通信途径，以保证能顺利地对问题进行分析。



(2) 分析与综合

- 从信息流和信息结构出发，逐步细化所有的软件功能，找出系统各元素之间的联系、接口特性和设计上的约束，分析它们是否满足功能要求，是否合理。剔除其不合理的部分，增加其需要部分。最终综合成系统的解决方案，给出目标系统的详细逻辑模型。

常用的分析方法

- 面向数据流的结构化分析方法（SA）
- 面向数据结构的Jackson方法（JSD）
- 结构化数据系统开发方法（DSSD）
- 面向对象的分析方法（OOA）等

(3) 编制需求分析阶段的文档

- 软件需求说明书
- 数据要求说明书
- 初步的用户手册
- 修改、完善与确定软件开发实施计划

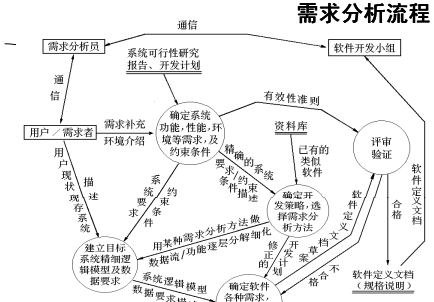
(4) 需求分析评审

- 系统定义的目标是否与用户的要求一致；
- 系统需求分析阶段提供的文档资料是否齐全；
- 文档中的所有描述是否完整、清晰、准确反映用户要求；
- 与所有其它系统成分的重要接口是否都已经描述；

- 被开发项目的数据流与数据结构是否足够，确定；
- 所有图表是否清楚，在不补充说明时能否理解；
- 主要功能是否已包括在规定的软件范围之内，是否都已充分说明；
- 设计的约束条件或限制条件是否符合实际；
- 开发的技术风险是什么；

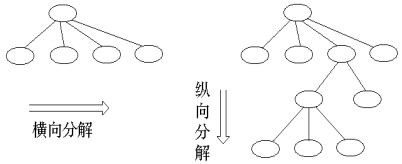
- 是否考虑过软件需求的其它方案；
- 是否考虑过将来可能会提出的软件需求；
- 是否详细制定了检验标准，它们能否对系统定义是否成功进行确认；

需求分析流程



软件需求分析的原则

- 需要能够表达和理解问题的信息域和功能域
- 要能以层次化的方式对问题进行分解和不断细化
- 要给出系统的逻辑视图和物理视图



软件需求规格说明的原则

- 从现实中分离功能，即描述要“做什么”而不是“怎样实现”
- 要求使用面向处理的规格说明语言（或称系统定义语言）
- 如果被开发软件只是一个大系统中的一个元素，那么整个大系统也包括在规格说明的描述之中

- 规格说明必须包括系统运行环境
- 规格说明必须是一个认识模型
- 规格说明必须是可操作的
- 规格说明必须容许不完备性并允许扩充
- 规格说明必须局部化和松散耦合

软件需求方法

- 需求分析方法由对软件问题的信息域和功能域的系统分析过程及其表示方法组成
- 大多数的需求分析方法是由信息驱动的
- 信息域具有三种属性：信息流、信息内容和信息结构。

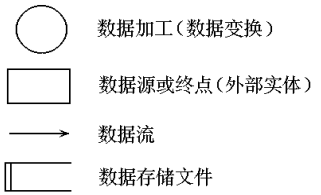
结构化分析方法

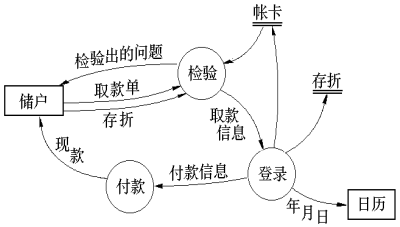
- 面向数据流进行需求分析的方法
- 结构化分析方法适合于数据处理类型软件的需求分析

- 具体来说，结构化分析方法就是用抽象模型的概念，按照软件内部数据传递、变换的关系，自顶向下逐层分解，直到找到满足功能要求的所有可实现的软件为止
- 结构化分析方法使用工具：数据流图，数据词典，结构化英语，判定表与判定树

数据流图

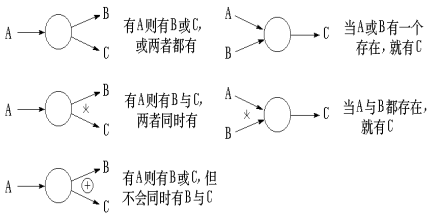
- 数据流图中的主要图形元素





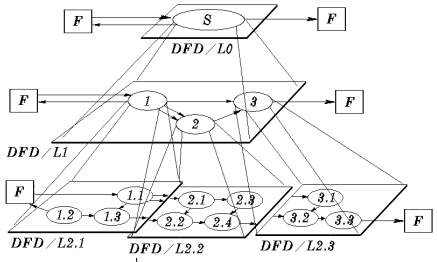
描述银行取款过程的数据流图

数据流与数据加工之间的关系



数据流图的层次结构

- 为了表达数据处理过程的数据加工情况，需要采用层次结构的数据流图。按照系统的层次结构进行逐步分解，并以分层的数据流图反映这种结构关系，能清楚地表达和容易理解整个系统

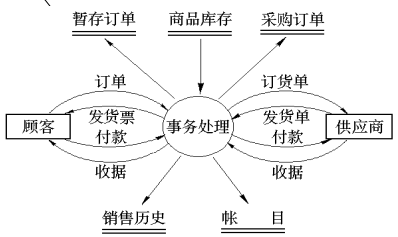


分层数据流图

- 在多层数据流图中，顶层流图仅包含一个加工，它代表被开发系统。它的输入流是该系统的输入数据，输出流是系统所输出数据
- 底层流图是指其加工不需再做分解的数据流图，它处在最底层
- 中间层流图则表示对其上层父图的细化。它的每一加工可能继续细化，形成子图。

结构化分析方法步骤示例

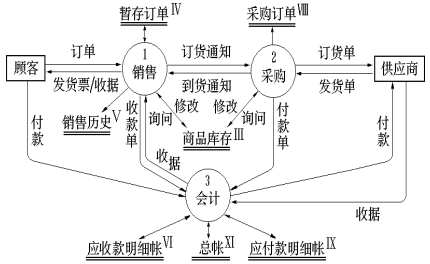
商店业务处理系统



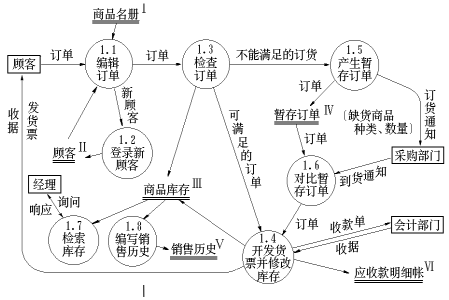
- 这个数据流图只是一个高层的系统逻辑模型，它反映了目标系统要实现的功能
- 数据流图绘制步骤
  - 首先确定系统的输入和输出
  - 根据商店业务，画出顶层数据流图，以反映最主要业务处理流程

- 经过分析，商店业务处理的主要功能应当有销售、采购、会计三大项。主要数据流输入的源点和输出终点是顾客和供应商。
- 然后从输入端开始，根据商店业务工作流程，画出数据流流经的各加工框，逐步画到输出端，得到第一层数据流图

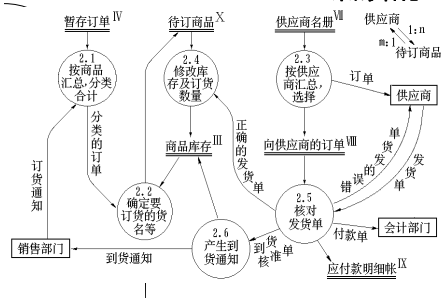
第一层数据流图



加细每一个加工框　　销售细化



采购细化



检查和修改数据流图的原则

- 数据流图上所有图形符号只限于前述四种基本图形元素
- 数据流图的主图必须包括前述四种基本元素，缺一不可
- 数据流图的主图上的数据流必须封闭在外部实体之间
- 每个加工至少有一个输入数据流和一个输出数据流

- 在数据流图中，需按层给加工框编号。编号表明该加工所处层次及上下层的亲子关系
- 规定任何一个数据流子图必须与它上一层的一个加工对应，两者的输入数据流和输出数据流必须一致。此即父图与子图的平衡
- 可以在数据流图中加入物质流，帮助用户理解数据流图

- 图上每个元素都必须有名字
- 数据流图中不可夹带控制流
- 初画时可以忽略琐碎的细节，以集中精力于主要数据流

数据词典

- 数据词典与数据流图配合，能清楚地表达数据处理的要求
- 词条描述 —— 对于在数据流图中每一个被命名的图形元素，均加以定义，其内容有:名字，别名或编号，分类，描述，定义，位置，其它，等

（1）数据流词条描述

- 数据流名：
- 说明：简要介绍作用即它产生的原因和结果
- 数据流来源：来自何方
- 数据流去向：去向何处
- 数据流组成：数据结构
- 数据量流通量：数据量，流通量

（2）数据元素词条描述

- 数据元素名：
- 类型：数字（离散值，连续值），文字（编码类型）
- 长度：
- 取值范围：
- 相关的数据元素及数据结构：

（3）数据文件词条描述

- 数据文件名：
- 简述：存放的是什么数据
- 输入数据：
- 输出数据：
- 数据文件组成：数据结构
- 存储方式：顺序，直接，关键码
- 存取频率：

（4）加工逻辑词条描述

- 加工名：
- 加工编号：反映该加工的层次
- 简要描述：加工逻辑及功能简述
- 输入数据流：
- 输出数据流：
- 加工逻辑：简述加工程序，加工顺序

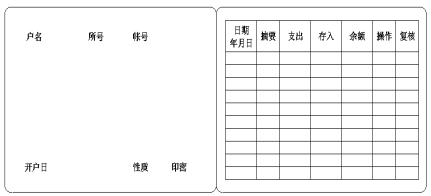
（5）源点及汇(终)点词条描述

- 名称：外部实体名
- 简要描述：什么外部实体
- 有关数据流：
- 数目：

数据结构的描述

符 号	含 义	举　例
=	被定义为	
+	与	$x = a+b$
[...,...] 或 [...|...]	或	$x = [a,b]$，$x = [a\|b]$
{ ... }或 m{...}n	重复	$x = \{a\}$，　$x = 3\{a\}8$
(...)	可选	$x = (a)$
"..."	基本数据元素	$x =$ "a"
..	连结符	$x = 1..9$

存折格式



- 存折＝户名＋所号＋帐号＋开户日＋性质＋（印密）＋1{存取行}50
- 户名＝2{字母}24
- 所号＝"001".."999"
- 帐号＝"00000001".."99999999"
- 开户日＝年＋月＋日
- 性质＝"1".."6"　注："1"表示普通户，"5"表示工资户等
- 印密＝"0"　注：印密在存折上不显示
- 存取行＝日期＋（摘要）＋支出＋存入＋余额＋操作＋复核

基本加工逻辑说明

- 对数据流图的每一个基本加工，必须有一个基本加工逻辑说明
- 基本加工逻辑说明必须描述基本加工如何把输入数据流变换为输出数据流的加工规则

- 加工逻辑说明必须描述实现加工的策略而不是实现加工的细节
- 加工逻辑说明中包含的信息应是充足的，完备的，有用的，没有重复的多余信息

用于写加工逻辑说明的工具

- 结构化英语
- 判定表
- 判定树

（1）结构化英语

- 结构化英语的词汇表由
  - 英语命令动词
  - 数据词典中定义的名字
  - 有限的自定义词
  - 逻辑关系词 IF_THEN_ELSE、CASE_OF 、 WHILE_DO、REPEAT_UNTIL等组成。

- 是一种介于自然语言和形式化语言之间的语言
- 语言的正文用基本控制结构进行分割，加工中的操作用自然语言短语来表示
- 其基本控制结构有三种：
  - ◆简单陈述句结构：避免复合语句；
  - ◆重复结构：WHILE_DO 或 REPEAT_UNTIL结构。
  - ◆判定结构：IF_THEN_ELSE 或 CASE_OF结构；

商店业务处理系统中“检查发货单”

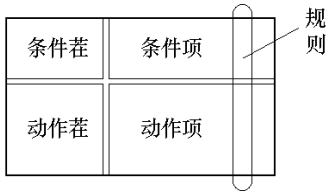
```
IF 发货单金额超过$500 THEN
   IF 欠款超过了60天 THEN
        在偿还欠款前不予批准
   ELSE （欠款未超期）
        发批准书，发货单
   ENDIF
ELSE （发货单金额未超过$500）
   IF 欠款超过60天 THEN
        发批准书，发货单及赊欠报告
   ELSE （欠款未超期）
        发批准书，发货单
   ENDIF
ENDIF
```

（2）判定表

- 如果数据流图的加工需要依赖于多个逻辑条件的取值，使用判定表来描述比较合适

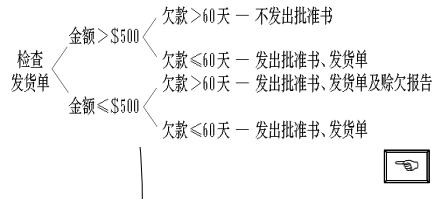
以“检查发货单”为例

		1	2	3	4
条件	发货单金额	＞$500	＞$500	≤$500	≤$500
	赊欠情况	＞60天	≤60天	＞60天	≤60天
操作	不发出批准书	√			
	发出批准书		√	√	√
	发出发货单		√	√	√
	发出赊欠报告			√	



（3）判定树

- 判定树也是用来表达加工逻辑的一种工具。有时侯它比判定表更直观。



原型化方法

- 在开发初期，要想得到一个完整准确的规格说明不是一件容易的事。特别是对一些大型的软件项目。
- 用户往往对系统只有一个模糊的想法，很难完全准确地表达对系统的全面要求。

- 软件开发者对于所要解决的应用问题认识更是模糊不清
- 随着开发工作向前推进，用户可能会产生新的要求，或因环境变化，要求系统也能随之变化；开发者又可能在设计与实现的过程中遇到些没有预料到的实际困难，需要以改变需求来解脱困境。

- 因此规格说明难以完善、需求的变更、以及通信中的模糊和误解，都会成为软件开发顺利推进的障碍。
- 为了解决这些问题，逐渐形成了软件系统的快速原型的概念。

软件原型的分类

- 在软件开发中，原型是软件的一个早期可运行的版本，它反映最终系统的部分重要特性。
  - ◆ 探索型：目的是要弄清对目标系统的要求，确定所希望的特性，并探讨多种方案的可行性。

◆ 实验型：这种原型用于大规模开发和实现之前，考核方案是否合适，规格说明是否可靠。

◆ 进化型：这种原型的目的不在于改进规格说明，而是将系统建造得易于变化，在改进原型的过程中，逐步将原型进化成最终系统。

殷人昆:

原型使用策略

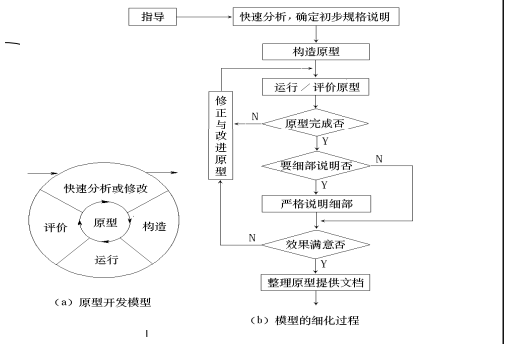
- ◆ 废弃策略
- ◆ 追加策略

建立快速原型，进行系统的分析和构造的好处：

- ◆增进软件者和用户对系统服务需求的理解，使比较含糊的具有不确定性的软件需求（主要是功能）明确化。
- ◆软件原型化方法提供了一种有力的学习手段。

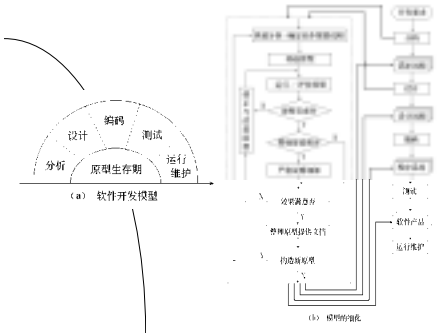
◆ 使用原型化方法，可以容易地确定系统的性能，确认各项主要系统服务的可应用性，确认系统设计的可行性，确认系统作为产品的结果。

◆ 软件原型的最终版本，有的可以原封不动地成为产品，有的略加修改就可以成为最终系统的一个组成部分，这样有利于建成最终系统。



（a）原型开发模型

（b）模型的细化过程



（a） 软件开发模型

（b） 模型的细化

原型开发技术

- 可执行规格说明
- 基于脚本(scenario)的设计
- 自动程序设计
- 专用语言
- 可复用(reusable)的软件
- 简化假设

可执行规格说明

- 可执行规格说明是用于需求规格说明的一种自动化技术。使用这种方法，人们可以直接观察他们用语言规定的任何系统性行为。包括
  - 代数规格说明
  - 有限状态模型
  - 可执行的数据流图

（1）代数规格说明

- 代数规格说明使用集合、定义于这些集合上的函数和定义于这些函数上的方程来描述对象。规格说明的操作语义用这些方程表示。

举例：定义一个无界的栈及其操作

NEW_STACK：→ Stack

PUSH：Stack，Element → Stack

POP： Stack → (Element | Undefined)

POP (NEW_STACK ( )) ＝ Undefined

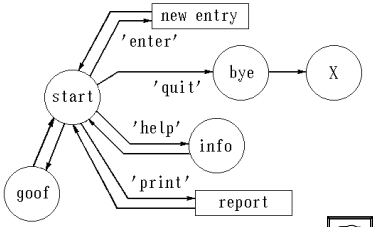
POP (PUSH (stk，elem)) ＝ elem

其中，前三行定义了操作的语法，后两行把它们的语义定义为一些方程。

（2）有限状态模型

- parnas提出的使用最广泛的一种可执行规格说明形式。从一个初始状态开始接收输入，到产生输出，状态在推移变化。施加在状态元素上的约束确定了有效状态的推移。

举例：建立用户／程序对话



（3）可执行的数据流图

- 数据流图是基于结构化开发方法的结构化规格说明
- 用一种可执行的语言程序代替定义处理逻辑的结构化英语，数据流图就成为由可执行语言程序模块组成的网络，在一定环境或工具的支持下就可成为一个可以执行的原型系统。

基于脚本的设计

- 脚本是指用户界面的原型。一个脚本用以模拟在系统运行期间用户经历的事件。它提供了输入—处理—输出的屏幕格式和有关对话的模型。因此，软件开发者能够给用户显示系统的逼真的视图，使用户得以判断是否符合他的意图。

- 可在任一脚本中使用一套可复用的软件模块，以表达某一方面的要求。
- 可使用一种原型语言来描述原型系统。原型开发过程中用这种语言来定义屏幕、数据项、及其相关的操作。从系统的外部描述开始，开发与数据库的接口、错误处理和恢复过程等系统的与外部视图一致的细节。

自动程序设计

- 自动程序设计是指在程序自动生成环境的支持下，利用计算机实现软件的开发。它可以自动地或半自动地把用户的非过程式问题规格说明转换为某种高级语言程序：

- 演绎综合手段：
  - 基于数学推理的构造式证明。
- 程序变换手段：
  - 将一程序转换成另一功能等价的程序，并保持其正确性不变。

- 实例推广手段：
  - 从实例特征出发，将它推广为待编程序的特征，最后得到程序。
- 过程化手段：
  - 研究甚高级语言的编译和知识的过程化。

专用语言

- 专用语言是应用领域的模型化语言。在原型开发中使用专用语言，可方便用户和软件开发者在计划中的系统特性方面的交流。

软件复用技术

- 利用可复用的模块，做出适当的组合，就可得到快速构造的原型系统。
- 为了快速地构造原型，这些模块首先必须有简单而清晰的界面；其次它们应当尽量不依赖其它的模块或数据结构；第三，它们应具有一些通用的功能。

简化假设

- 简化假设是在开发过程中使设计者迅速得到一个简化的系统所做的假设。尽管这些假设可能实际上并不能成立，但它们在原型开发过程中可以使开发者的注意力集中在一些主要的方面。

- 在修改一个文件时，可以假设这个文件确实存在
- 在存取文件时，待存取的记录总是存在
- 一旦计划中的系统满足用户所有的要求，就可以撤消这些假设，并追加一些细节。

系统动态分析

- 系统的需求规格说明通常是用自然语言来叙述的，但是用自然语言描述往往会出现歧义性。
- 为了直观地分析系统的动作，从特定的视点出发描述系统的行为，需要采用动态分析的方法。

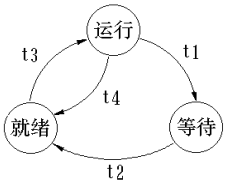
最常用的动态分析方法

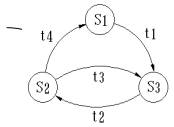
- 状态迁移图
- 时序图
- Petri网

状态迁移图

- 状态迁移图是描述系统的状态如何相应外部的信号进行推移的一种图形表示。
  - ◆ 圆圈“○”表示可得到的系统状态
  - ◆ 箭头“→”表示从一种状态向另一种状态的迁移。

例如，当有多个申请占用CPU运行的进程时，有关CPU分配的进程的状态迁移。





(a) 状态迁移图

事件 \ 状态	S1	S2	S3
t1	S3		
t2			S2
t3		S3	
t4		S1	

(b) 状态迁移表

- 可得到的状态＝就绪，运行，等待
- 生成的事件＝t1，t2， t3， t4
  - t1 — 中断事件 • t2 — 中断已处理
  - t3 — 分配CPU • t4 — 用完CPU时间

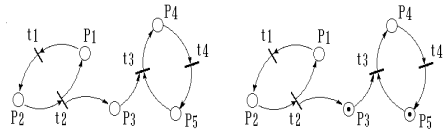
状态迁移图的优点

- 状态之间的关系能够直观地捕捉到
- 由于状态迁移图的单纯性，能够机械地分析许多情况，可很容易地建立分析工具

Petri网

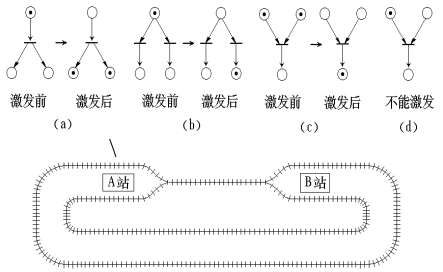
- Petri网已广泛地应用于硬件与软件系统的开发中，它适用于描述与分析相互独立、协同操作的处理系统，也就是并发执行的处理系统。

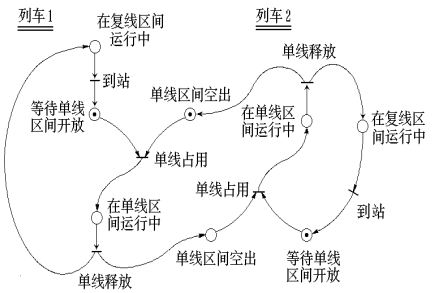
- Petri网简称PNG (Petri Net Graph)，它有两种结点：
  - ◆位置(*place*)：符号为“○”，它用来表示系统的状态。
  - ◆转移(*transition*)：符号为 “?”，它用来表示系统中的事件。
  - ◆图中的有向边表示对转移的输入，或由转移的输出



(a) Petri网 (b) 加了标记的Petri网

- 标记，或称令牌(*token*)，是表明系统当前处于什么状态的标志





进程	得到资源	占用资源运行	释放资源	不使用资源运行
PR1	LOCK R	处理11	UNLOCK R	处理12
PR2	LOCK R	处理21	UNLOCK R	处理22

处理两个进程的同步问题