

软件工程基础

第3章 需求分析

苏生

343887454@qq.com

为非洲人开发的手机和为亚洲人开发的手机有什么不同？



光线不佳时，人脸就剩下牙齿
通过眼睛和牙齿来定位，加强曝光

双开双待，节省手机

主打音乐功能，随机赠送一个定制的头戴式耳机

360安全软件为什么能成功?

免费?

用户心中的安全是什么?

今日头条为什么会成功?

拼多多为什么会成功?

考勤系统的需求是什么?

本章内容

3.1 需求分析的概念

3.2 需求的获取

3.3 需求分析/管理的过程

3.4 需求分析的任务

3.5 软件需求规格文档编制

3.6 需求分析模型概述

3.7 结构化分析方法

3.8 数据流图

3.9 数据流图的改进

3.10 编写数据字典

3.11 什么是对象

3.12 面向对象的软件开发模型

3.13 用例图

3.14 用例之间的关系

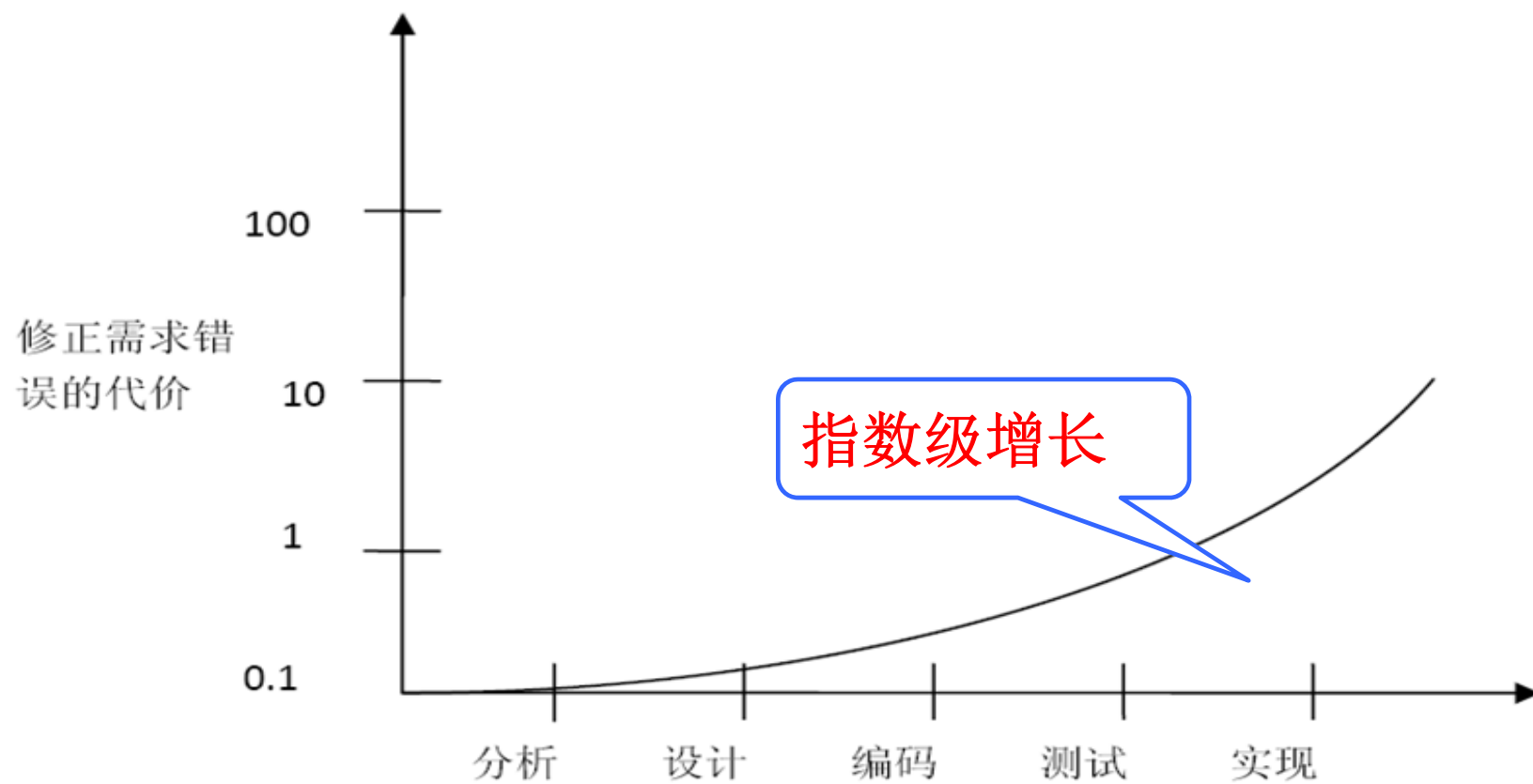
3.1 需求分析的概念

定义

确定系统必须具有的功能和性能，系统要求的运行环境，并且预测系统发展的前景。

换句话说需求就是以一种清晰、简洁、一致且无二义性的方式，对一个待开发系统中各个有意义方面的陈述的一个集合。

修正需求错误的代价



3.2 需求的获取

定义

软件需求获取指的是

- 软件需求的来源
- 软件工程师收集这些软件需求的方法。

它也称为需求抓取、需求发现和需求获得。

需求的类型



- 描述系统应该做什么，即为用户和其它系统完成的功能、提供的服务。

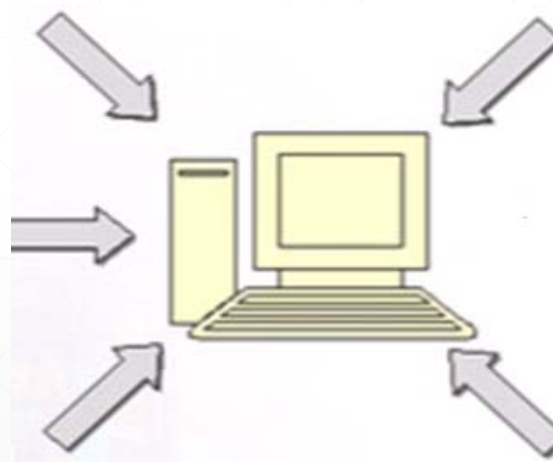
- 必须遵循的标准，外部界面的细节，实现的约束条件，质量属性等等
-

需求的来源

用户目标

领域知识

投资者



运行环境

组织环境

需求获取技术

- - 采访
 - - 设定情景（用例）
 - - 原型
 - - 会议
 - - 观察商业过程和工作流
-

需求获取面临的挑战



经验：

- 1) 尽可能地分析清楚哪些是稳定的需求，哪些是易变的需求。以便在进行系统设计时，将软件的核心建筑在稳定的需求上，否则将会吃尽苦头。
- 2) 在合同中一定要说清楚“做什么”和“不做什么”。

需求诱导十原则

1. 倾听



2. 有准备的沟通



3. 需要有人推动



4. 最好当面沟通



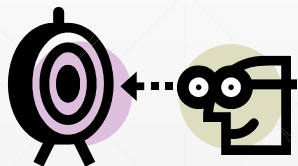
6. 保持通力协作



5. 记录所有决定



7. 聚焦并协调话题



8. 采用图形表示



9. 继续前进原则



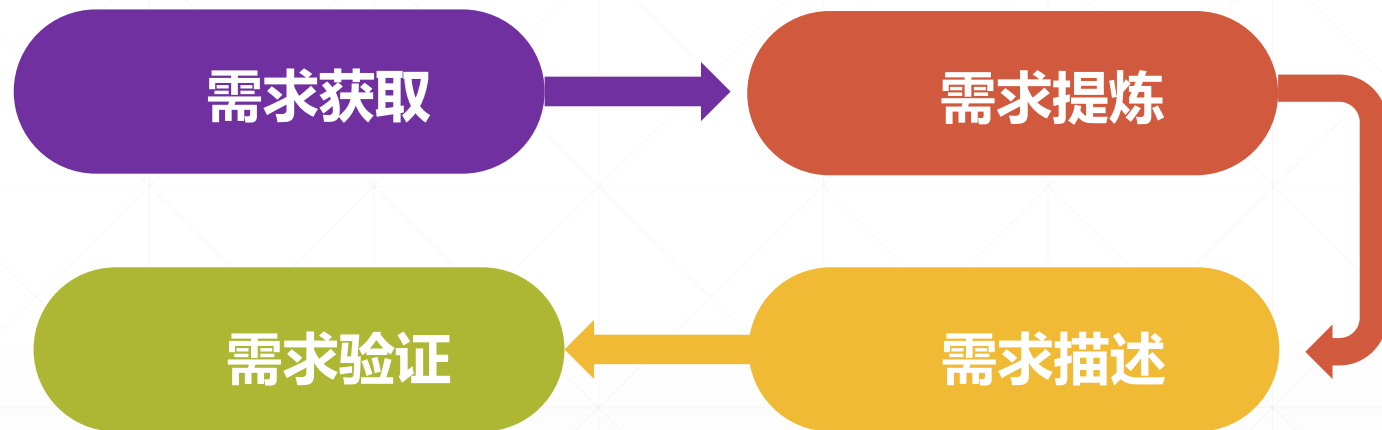
10. 谈判双赢原则



3.3 需求分析/管理的过程

- 请见上一讲

需求确认



需求变更

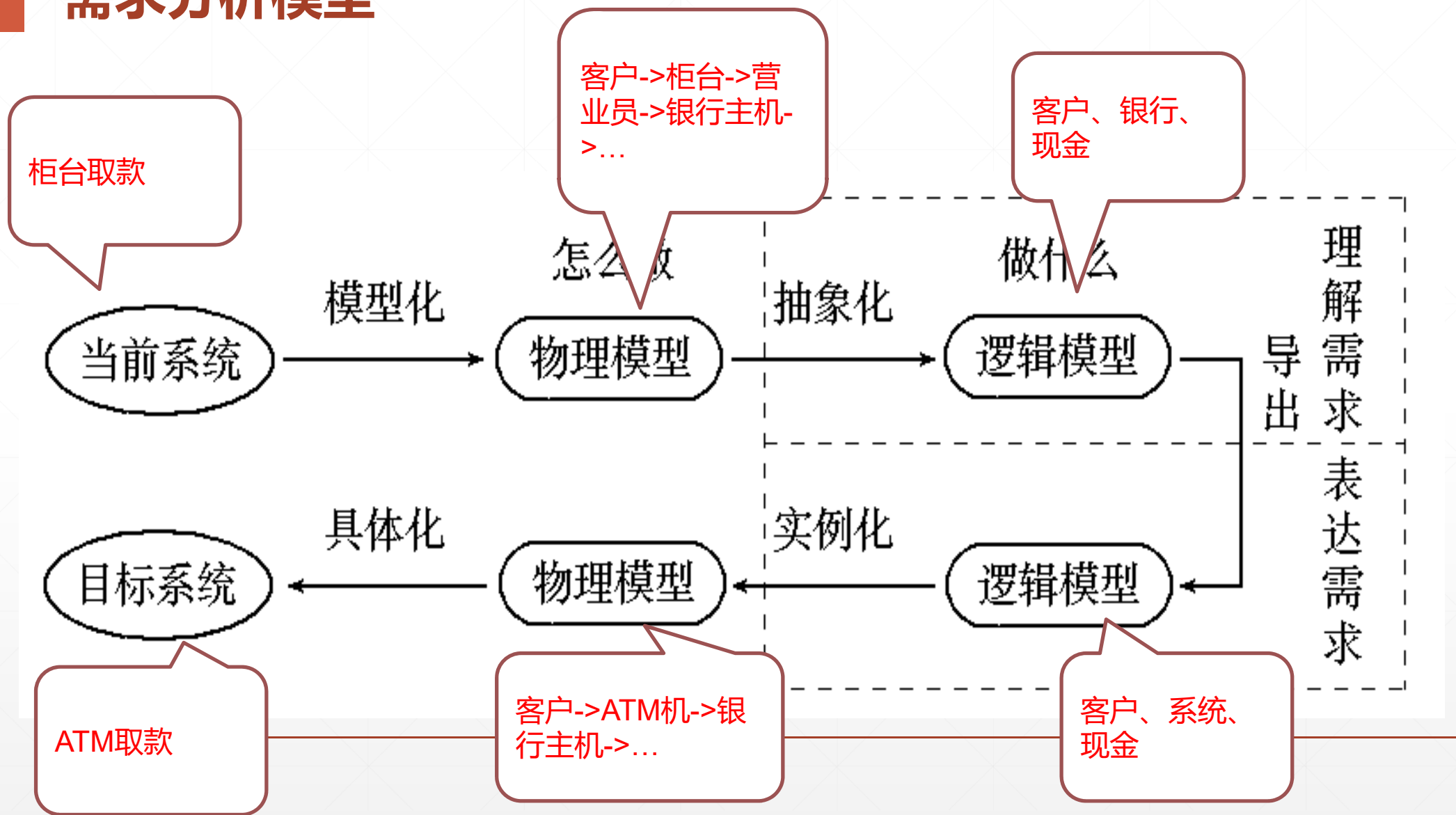
第二步：需求提炼（需求分析）

定义

- 对应用问题及环境的理解和分析，为问题涉及的信息、功能及系统行为建立模型。将用户需求精确化、完全化，最终形成下一步的需求规格说明书。

- 需求提炼（需求分析）的**核心**在于**建立分析模型**。
 - 需求提炼（需求分析）采用多种形式描述需求，通过建立需求的**多种视图**，揭示出一些更深的问题。
 - 需求提炼（需求分析）还包括与客户的交流以澄清某些易混淆的问题，并明确**哪些需求更为重要**，其目的是确保所有风险承担者尽早地对项目**达成共识**并对将来的产品有个相同而清晰的认识。
-

需求分析模型



第三步：需求规格说明书

定义

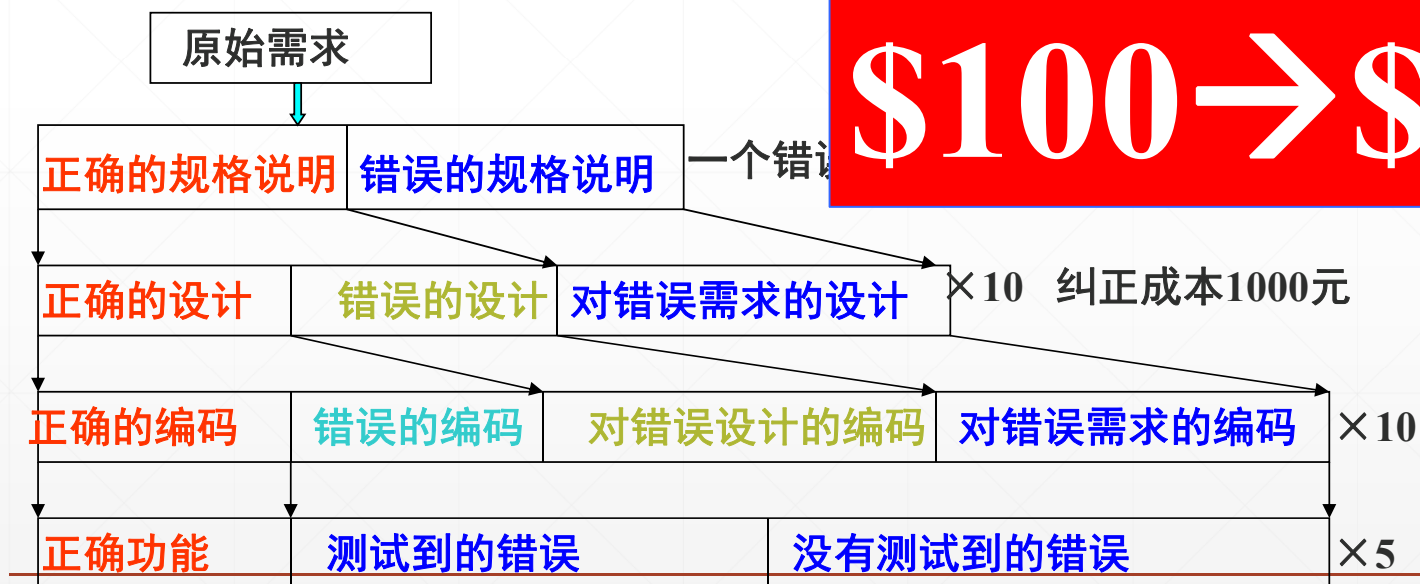
软件需求规格说明书（SRS）-----软件系统的需求规格说明，是待开发系统的行为的完整描述。它包含了功能性需求和非功能性需求。

- 需求分析工作完成的一个基本标志是形成了一份完整的、规范的需求规格说明书。
 - 需求规格说明书的编制是为了使用户和软件开发者双方对该软件的初始规定有一个共同的理解，使之成为整个开发工作的基础。
-

第四步：需求验证

需求验证的**重要性**：如果在后续的开发或当系统投入使用时才发现需求文档中的错误，就会导致更大代价的返工。由需求问题而对系统做变更的成本比修改设计或代码错误的成本要大的多。假设需求阶段引入1个错误的需求，设计时对这个需求需要5~10条设计实现，1条设计需要5~10条程序，1条程序需要3~5种测试组合测试。

\$100 → \$50000!!



需求验证的工作

对需求文档需执行以下类型的检查：

(1) 有效性检查

检查不同用户使用不同功能的有效性。

(2) 一致性检查

在文档中，需求之间不应该冲突。

(3) 完备性检查

需求文档应该包括所有用户想要的功能和约束。

(4) 现实性检查

检查保证能利用现有技术实现需求。

需求验证技术

- (1) 需求评审
 - (2) 利用原型检验系统是否符合用户的真正需要
 - (3) 对每个需求编写概念性的测试用例。
 - (4) 编写用户手册。用浅显易懂的语言描述用户可见的功能。
 - (5) 自动的一致性分析。可用CASE工具检验需求模型的一致性。
-

需求分析/管理的过程

- 请见上一讲

需求确认

需求获取

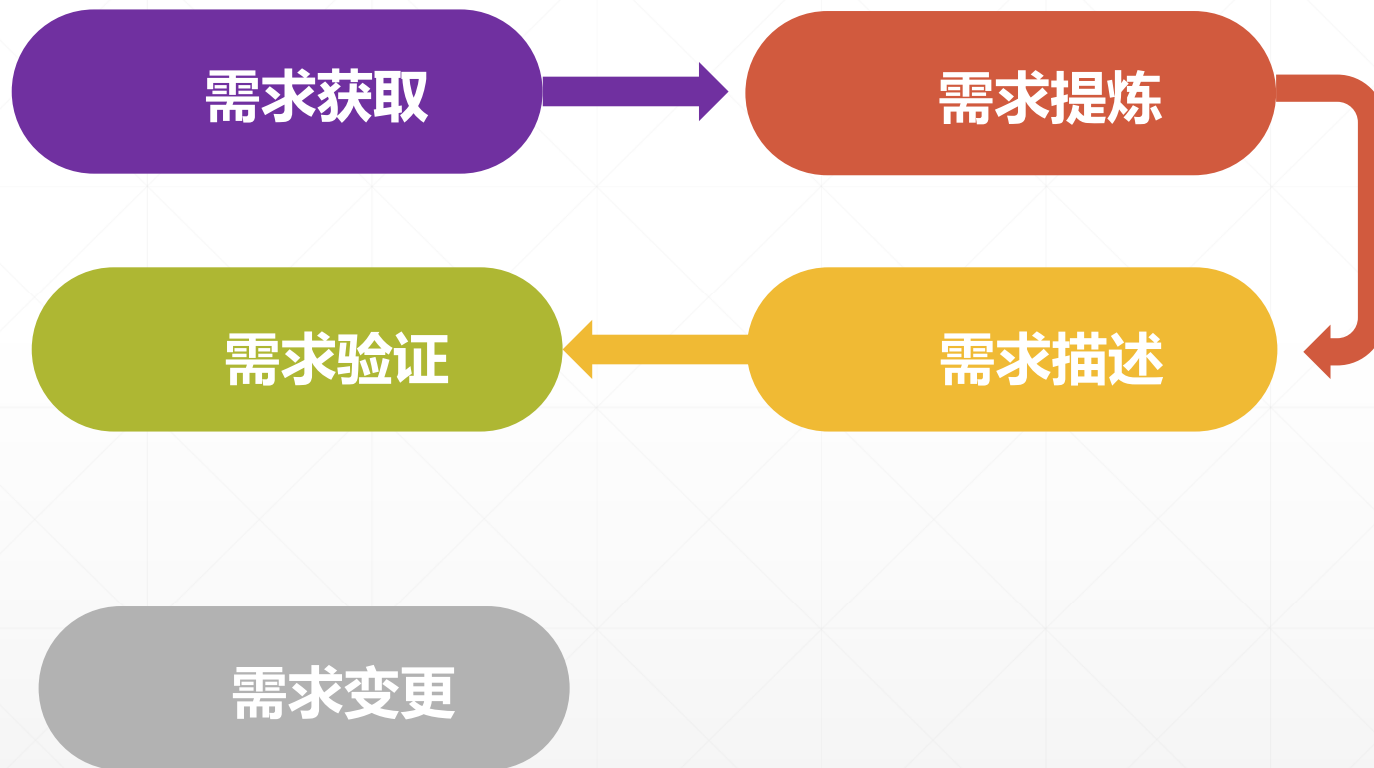
需求提炼

需求验证

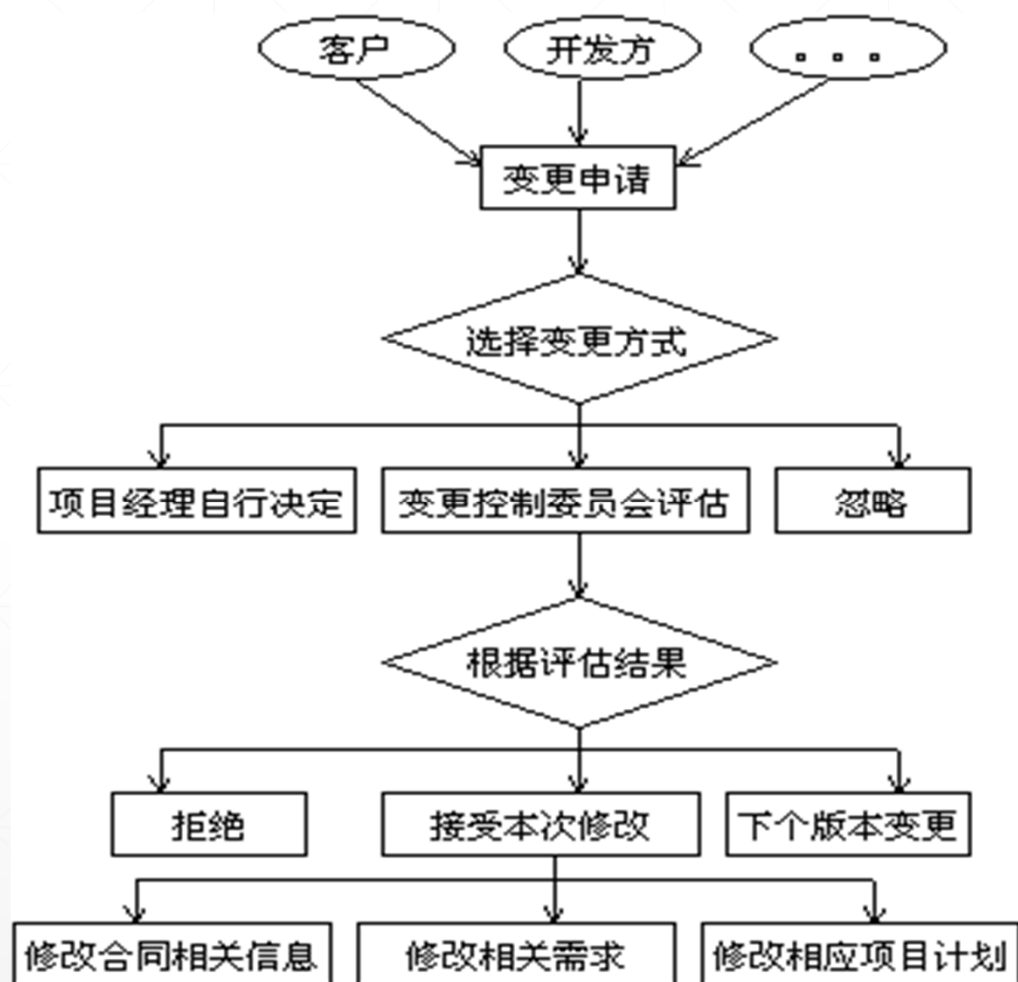
需求描述

需求变更

需求变更



需求变更处理流程



3.4 需求分析的任务

- 建立分析模型
- 编写需求说明

准确地定义未来系统的目标，确定为了满足用户的需求系统必须做什么。

用《需求规格说明书》规范的形式准确地表达用户的需求。

3.5 软件需求规格文档编制

沟通活动通用任务集

- 1、识别主要客户和相关利益者
 - 2、与主要客户会谈“上下文无关的问题”，以确定
 - ① 业务需要和商业价值
 - ② 最终用户的特性\需要
 - ③ 需要的用户可见输出
 - ④ 业务约束
 - 3、写一页项目范围的说明
 - 4、评审范围说明，并应客户要求做出相应修改
-

3.5 软件需求规格文档编制

沟通活动通用任务集

- 5、与客户/最终用户进行协作，确定：
 - ① 采用标准格式记录客户可见的使用场景
 - ② 输入和输出
 - ③ 重要的软件特性、功能和行为
 - ④ 客户定义的商业风险
 - 6、描述场景、输入/输出、特性/功能以及风险
 - 7、与客户细化场景、输入/输出、特性/功能以及风险
 - 8、为每个用户场景、特性、功能和行为分配客户定义的优先级
 - 9、回顾搜集的所有信息并修订
 - 10、为计划活动做准备
-

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
-

软件需求规格说明的结构

IEEE标准为需求文档提出了以下结构，组织机构内部可以基于此标准扩展：

(1) 引言

(2) 综合描述

(3) 需求描述

(4) 附录（词汇表、分析模型、待定问题列表）

(5) 索引

- a. 功能需求
- b. 数据需求与功能有关的数据定义和数据关系
- c. 性能需求响应时间、容量要求、用户数等
- d. 物理需求物理接口、通信接口
- e. 设计需求软件支持环境、报表、数据命名等
- f. 软件质量属性（可维护性、可靠性、可移植性、可用性、安全性等）
- g. 其他需求
- h. 假设和依赖性

这一节是文档中最实质性的部分，由于在机构组织的实践中存在极大的变数，对这一节定义的标准结构可以进行增删。

3.6 需求分析模型概述

其基本思想是用系统工程的思想 and 工程化的方法，根据用户至上的原则，自始至终按照结构化、模块化，自顶向下地对系统进行分析与设计。

- 面向过程分析模型
- 面向对象分析模型

由**5个层次**（主题层、对象类层、结构层、属性层和服务层）和**5个活动**（标识对象类、标识结构、定义主题、定义属性和定义服务）组成。

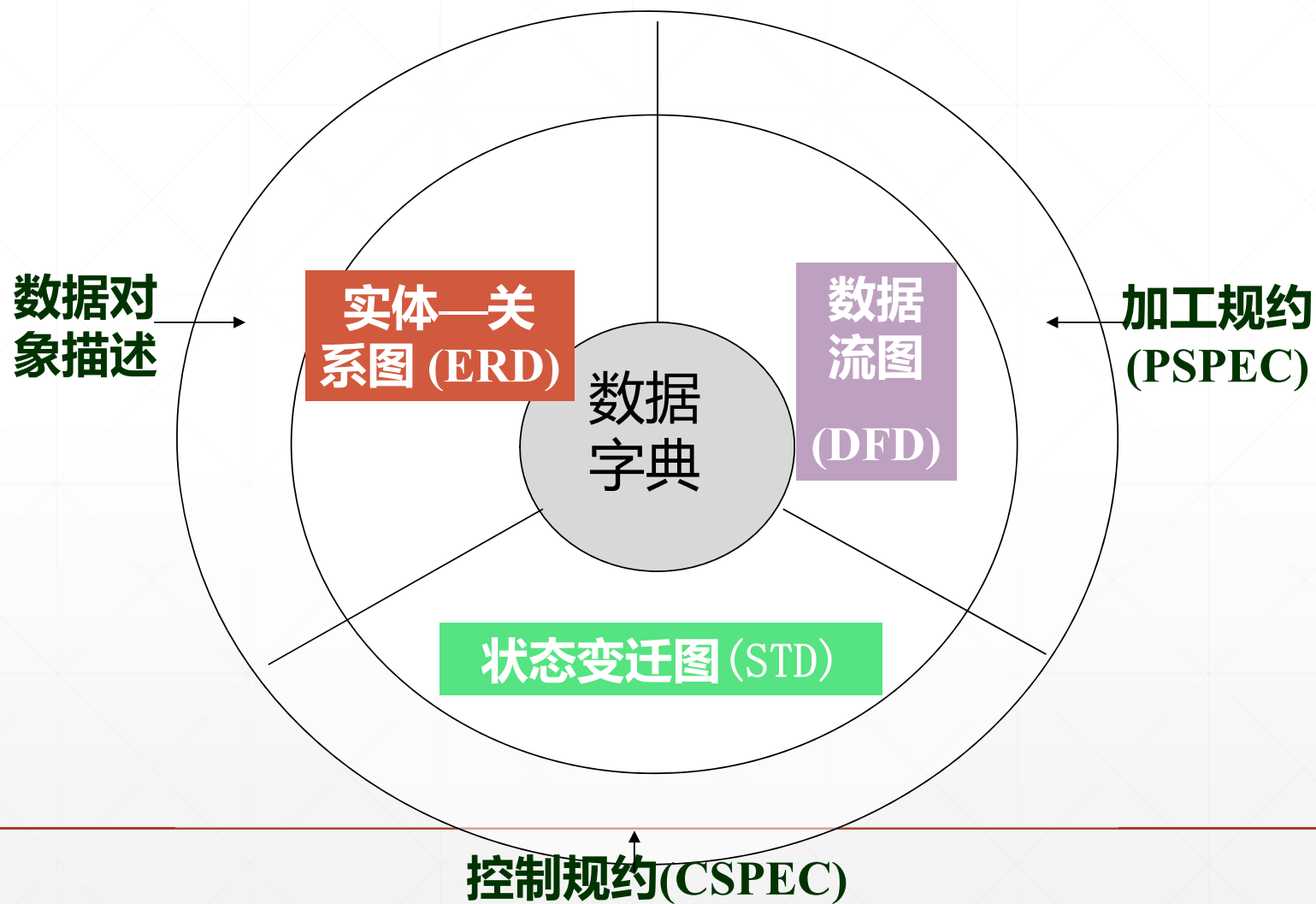
分析模型描述工具

	面向过程的需求分析	面向对象的需求分析
数据模型	实体-联系图 (ERD) 数据字典 (DD)	类图、类关系图
功能模型	数据流图 (DFD)	用例图
行为模型	状态变迁图 (STD)	活动图、时序图、状态图

3.7 面向过程分析模型—结构化分析方法

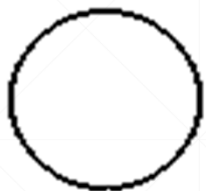
- 面向数据流进行需求分析的方法
 - 结构化分析方法适合于数据处理类型软件的需求分析
 - 具体来说，结构化分析方法就是用抽象模型的概念，按照软件内部数据传递、变换的关系，自顶向下逐层分解，直到找到满足功能要求的所有可实现的软件为止
-

面向过程的分析建模工具总览



3.8 功能模型——数据流图

- 数据流图中的主要图形元素：



数据加工（数据变换）



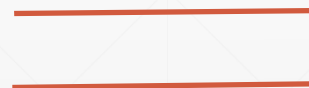
数据源或终点（外部实体）



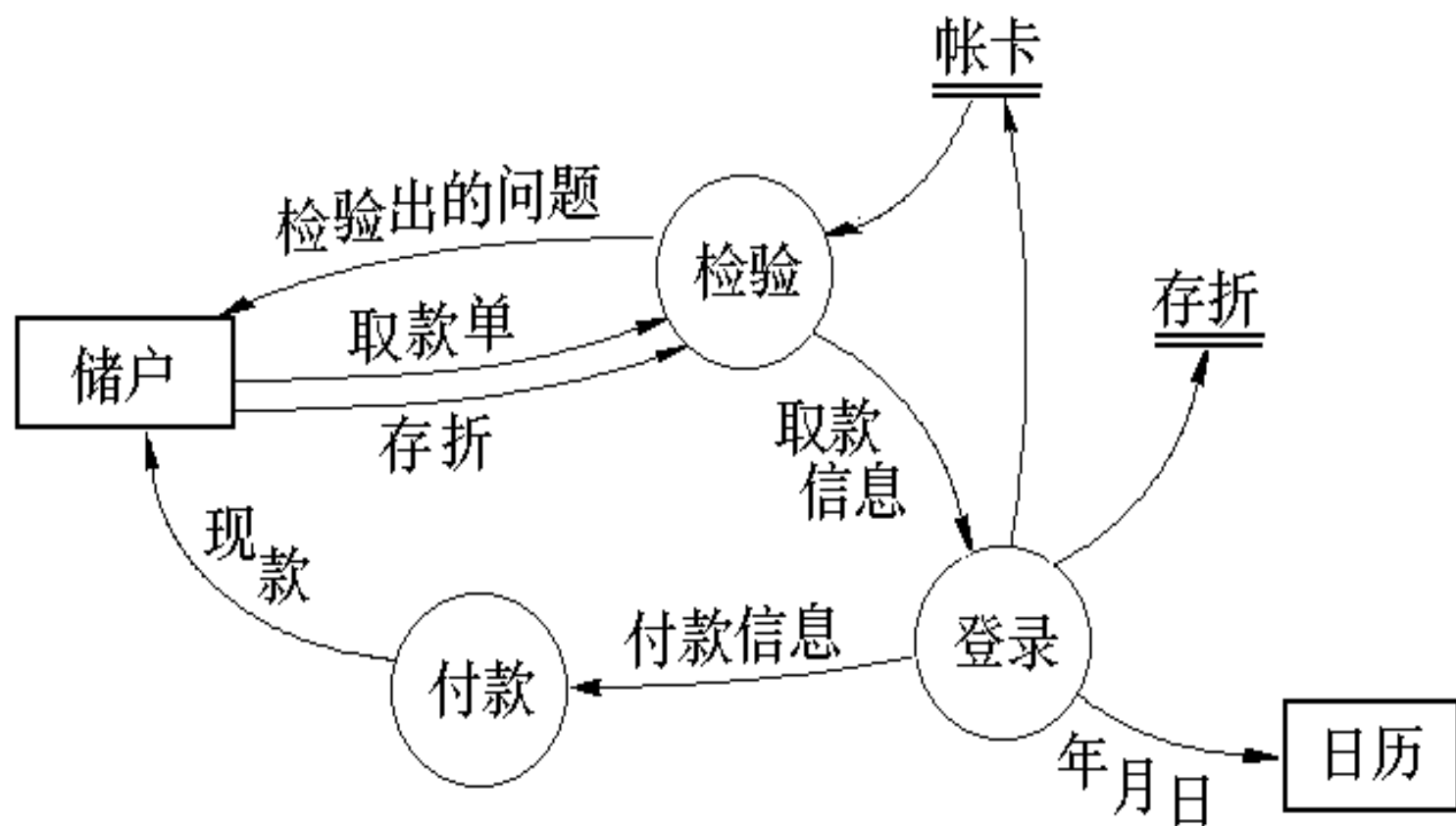
数据流



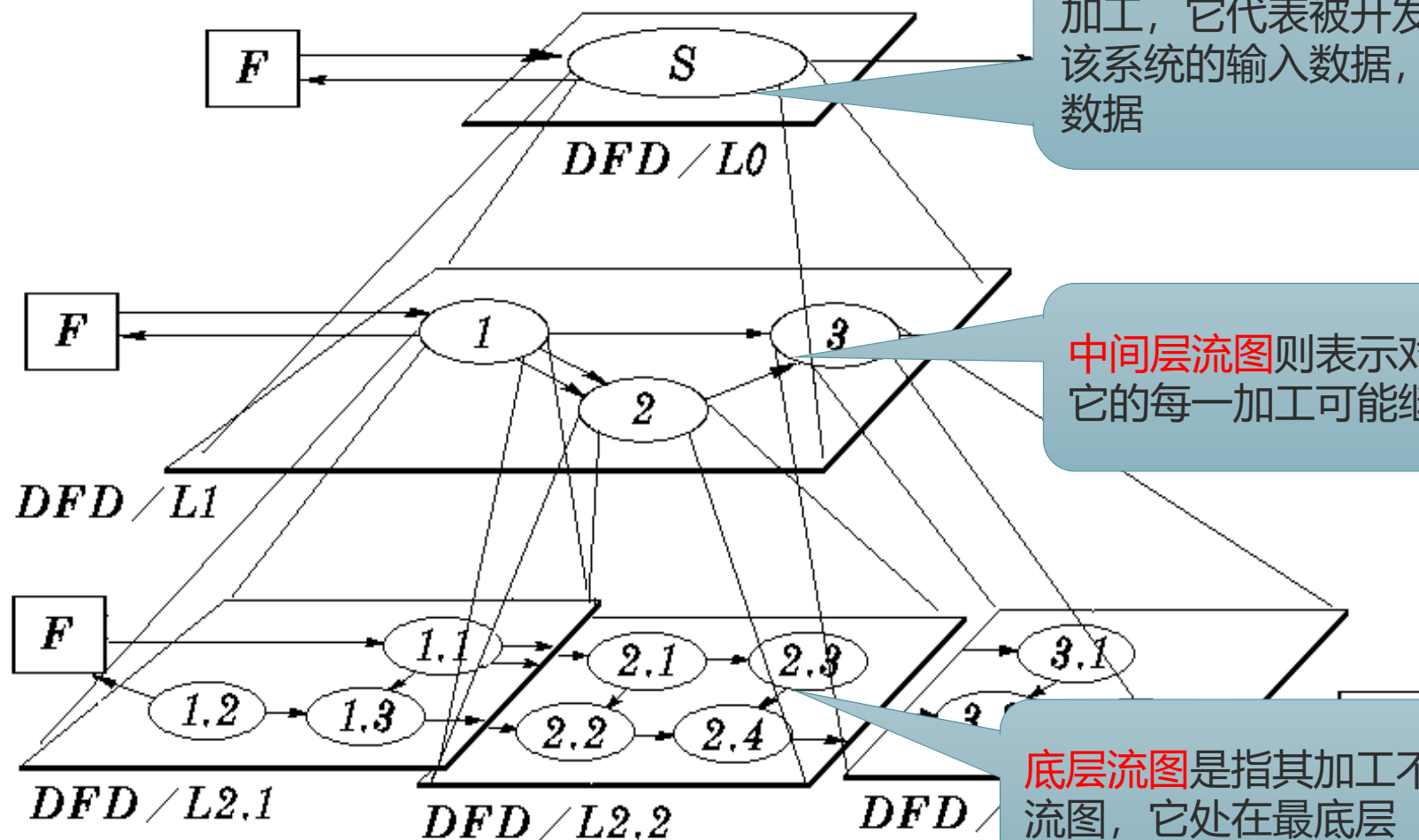
数据存储文件



数据流图图示例



数据流图的层次结构



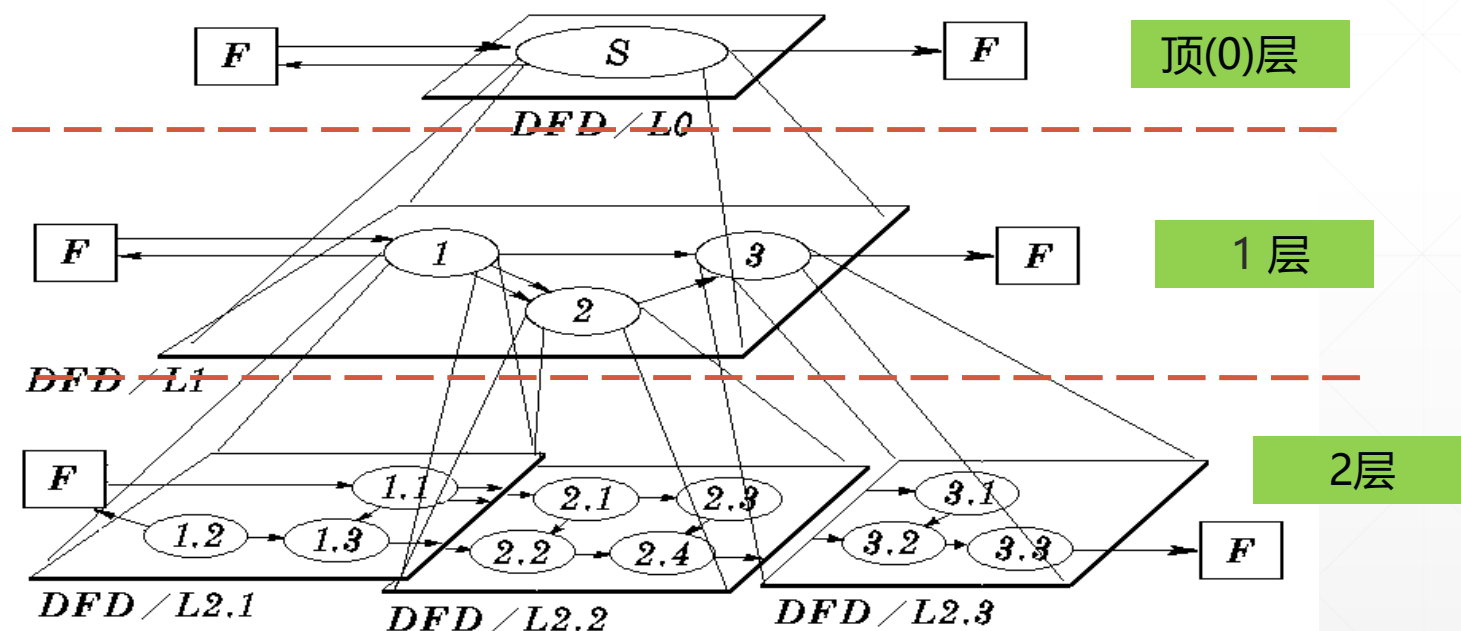
在多层数据流图中，**顶层流图**仅包含一个加工，它代表被开发系统。它的输入流是该系统的输入数据，输出流是系统所输出数据

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

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

(1) 加工

- 表示对数据进行的操作, 如“处理选课单”、“产生发票”等
- 加工的编号, 说明这个加工在层次分解中的位置 (分层DFD)

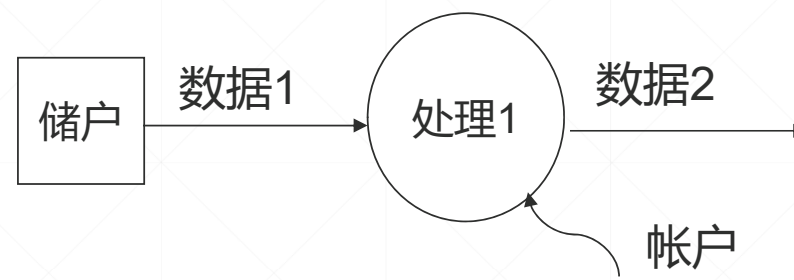


加工的命名

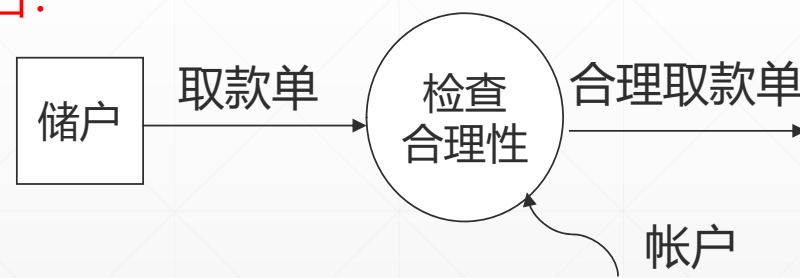
- 加工的命名注意事项

- 1) 顶层的加工名就是整个系统项目的名字
- 2) 尽量最好使用动宾词组，也可用主谓词组
- 3) 不要使用空洞的动词

无意义的名字：



较好的命名：

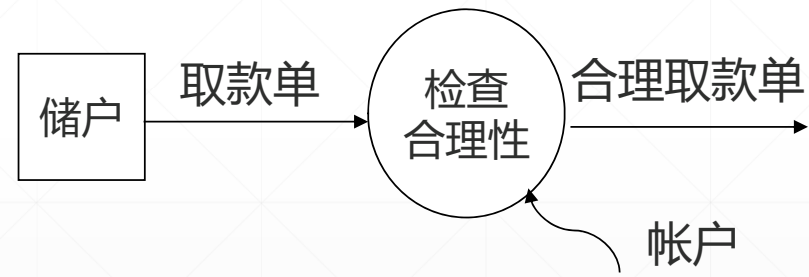


(2) 外部实体 (数据源点/终点)

- 位于系统之外的信息提供者或使用者,称为外部实体。即存在于系统之外的人员或组织。如“学务科”等
 - 说明数据输入的源点(数据源)或数据输出的终点(数据终点)
 - 起到更好的理解作用,但不是系统中的事物
-

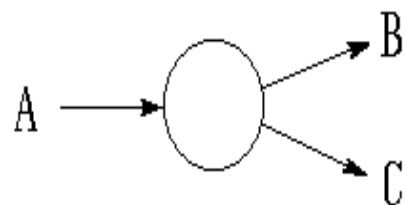
(3) 数据流

- 表示数据和数据流向, 由一组固定成分的数据组成 如“选课单”由“学号、姓名、课程编号、课程名”等成分组成
- 数据流可从加工流向加工, 也可在加工与数据存储或外部项之间流动; 两个加工之间可有多股数据流

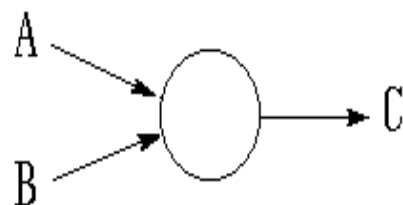


- 数据流的命名原则:
 - 1) 用名词,不要使用意义空洞的名词
 - 2) 尽量使用现实系统已有名字
-

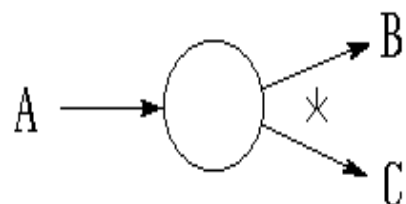
数据流与数据加工之间的关系



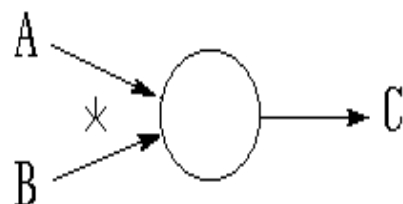
有A则有B或C,
或两者都有



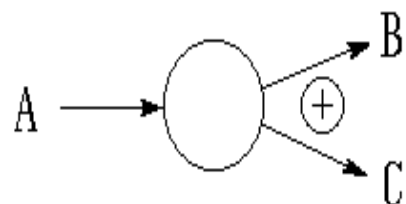
当A或B有一个
存在, 就有C



有A则有B与C,
两者同时有



当A与B都存在,
就有C

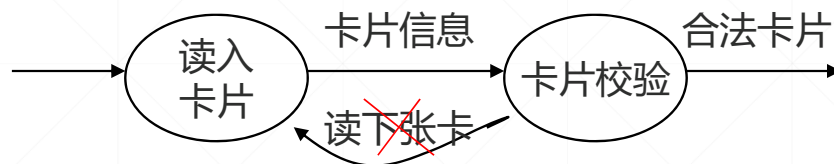


有A则有B或C, 但
不会同时有B与C

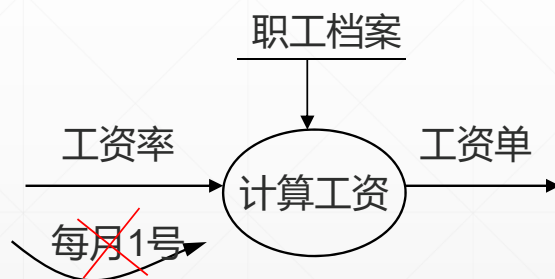
画数据流时需注意的问题

- 1) 不要把控制流作为数据流

如：下图中读下张卡属于控制流，不应画出。

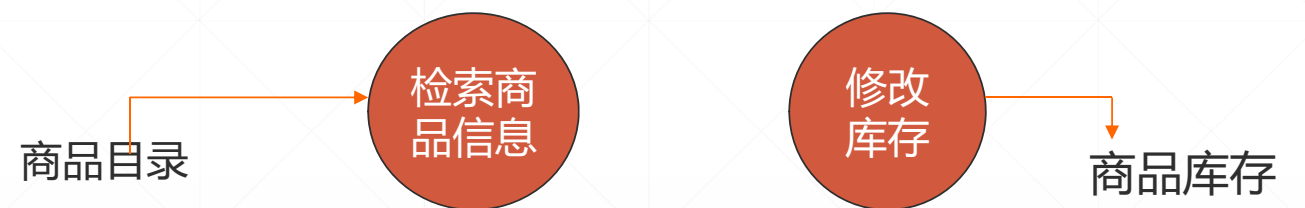


- 2) 不要标出激发条件



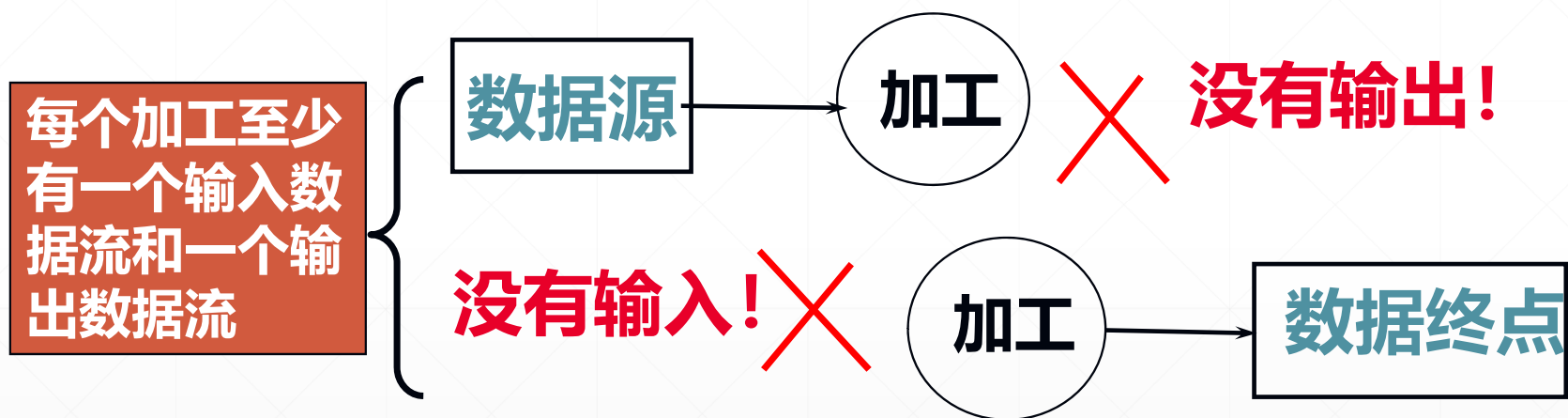
(4) 数据存储

- 表示需要保存的数据流向, 如 “学生档案”、“课程设置” 等
- 数据存储与加工的方向 ↑ “读出”、↓ “写入”

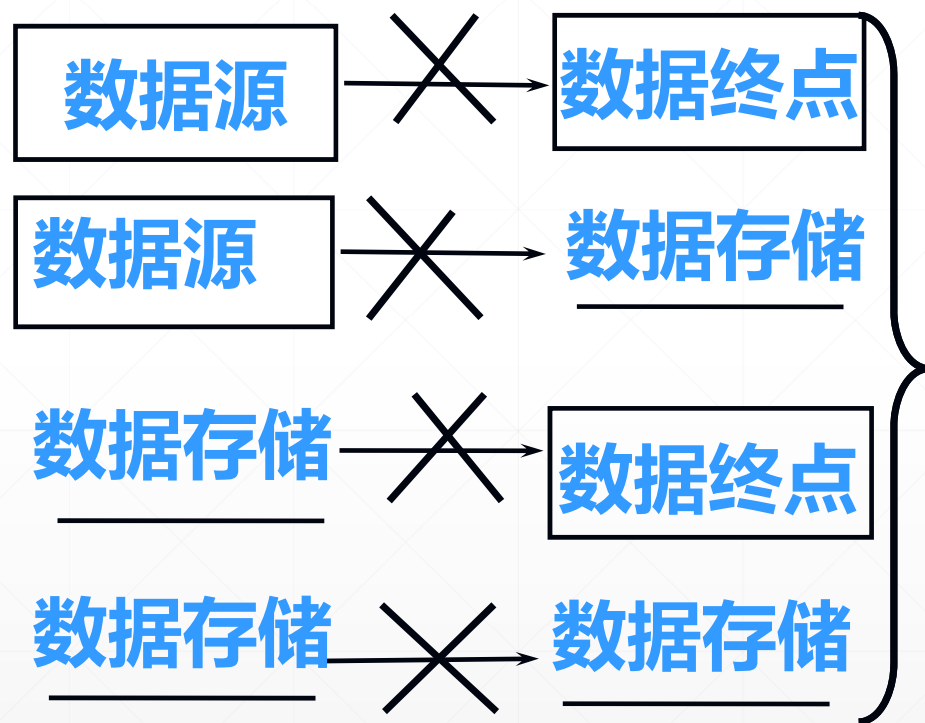


- 分层数据流程图中, 数据存储一般局限在某一层或某几层
 - 命名方法与数据流相似
-

几种错误(1)



几种错误(2)



数据流必须要么从某个加工流出、要么流入某个加工，而不能直接从外部项流向数据存储等等。图示的几种流动都是不合理的

示例

仓库管理与订货系统

某仓库业务的工作过程如下：企业职工填写领料单，经主管审查签名批准后，职工到仓库领取零件。

仓库保管员检查领料单是否符合审批手续，填写是否正确等，不正确的领料单退还职工，填写正确的领料单则办理领料手续，进行登记，修改库存量并给予零件。

当某种零件的库存量低于事先规定的临界值时，登记需要采购零件的订货信息，为采购部门提供一张订货单。要求用计算机辅助领料工作和编制订货单。

“仓库业务系统”数据流图

绘制步骤：第一步绘制数据流图顶层

- 首先确定系统的输入和输出，画出顶层数据流图。
- 经过分析，主要数据流输入的源点和输出终点是职工和仓库管理员、采购员。

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

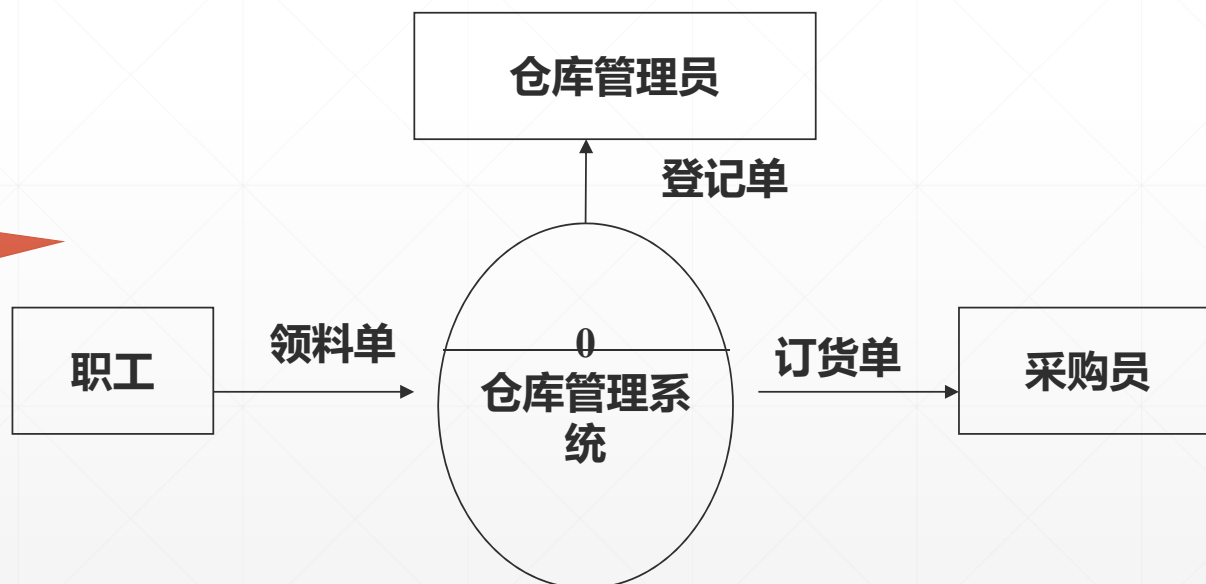


图0 顶层数据流程图

绘制步骤：第二步 绘制数据流图1层

- 从输入端开始，根据仓库业务工作流程，画出数据流流经的各加工框，逐步画到输出端，得到1层数据流图。

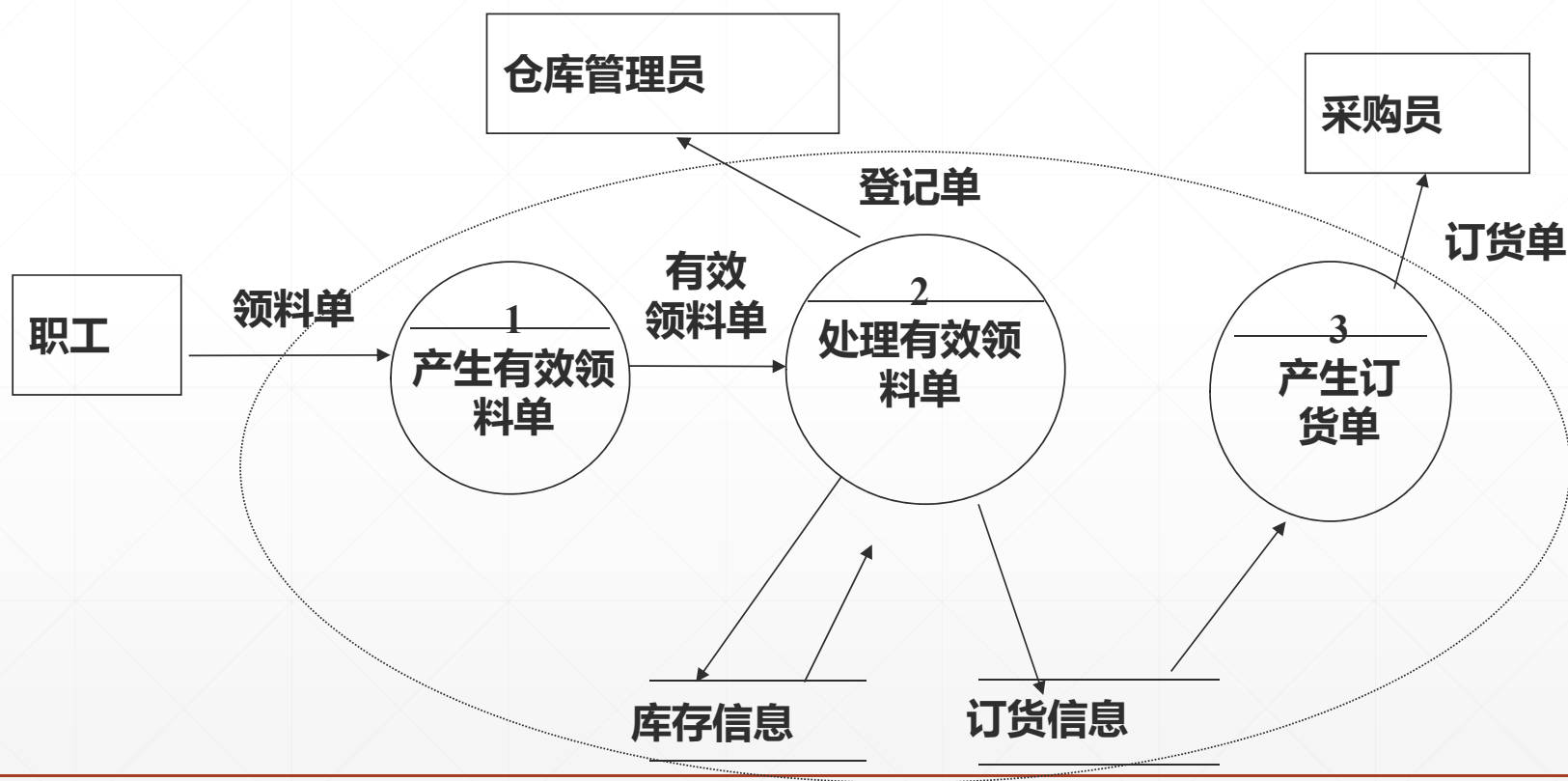


图1 仓库管理系统1层数据流图

绘制步骤：第三步 绘制数据流图2层
加细每一个加工框



图2 加工1产生有效领料单的分解数据流图

图1 仓库管理系统1层数据流图

绘制步骤：第三步 绘制数据流图2层 加细每一个加工框

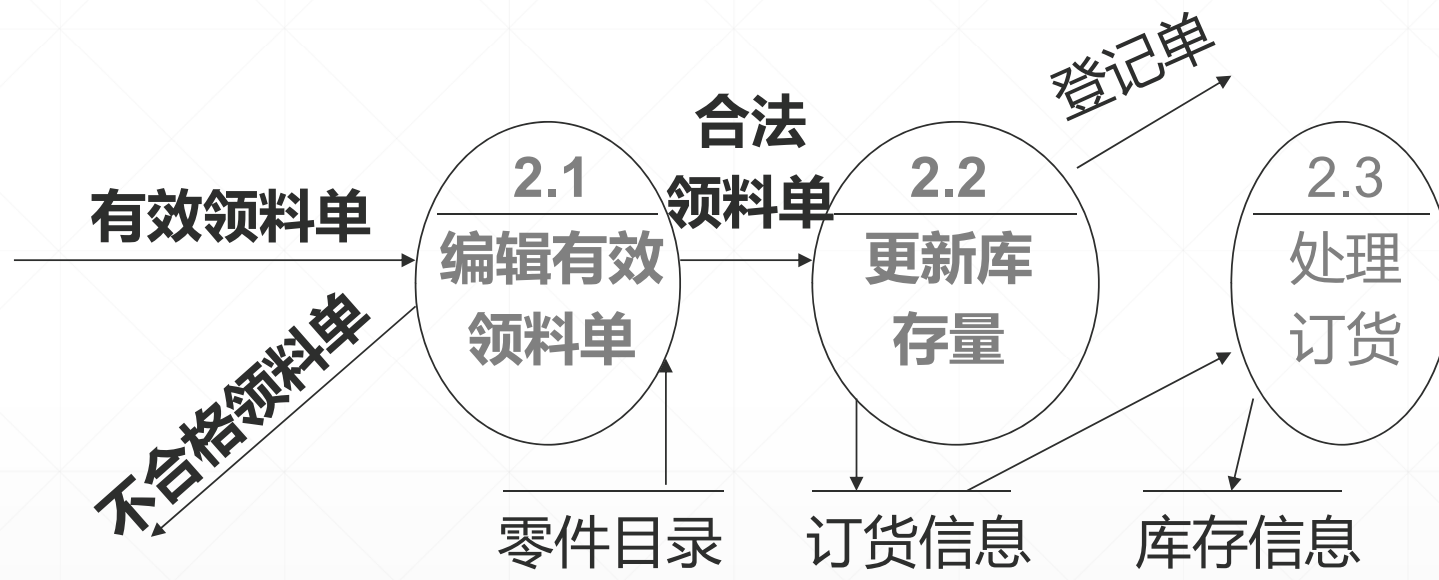


图3 加工2处理有效领料单的分解数据流图

绘制步骤：第四步 合成总体数据流图

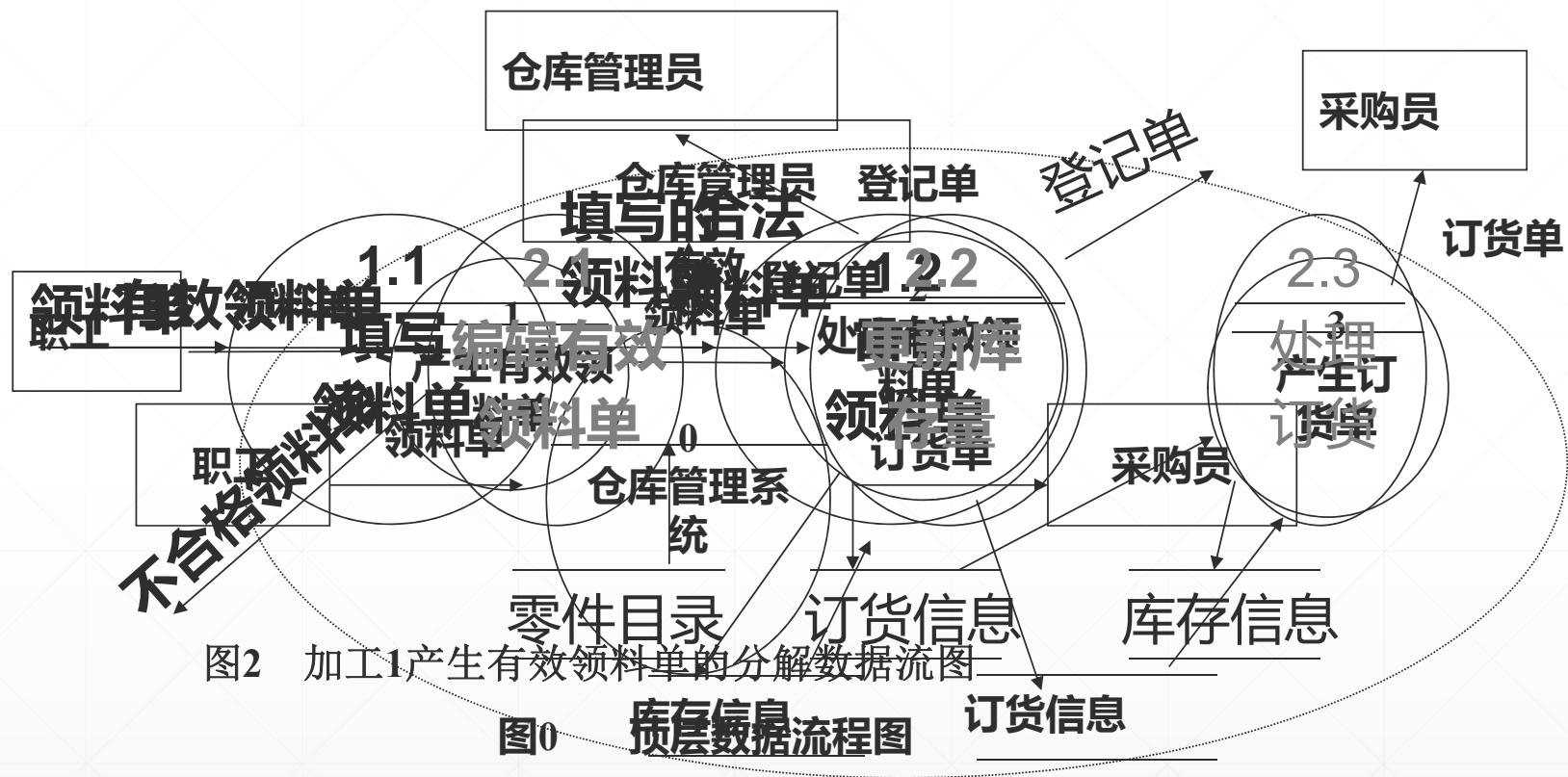


图3 加工处理有效领料单的分解数据流图

绘制步骤：第四步 合成总体数据流图

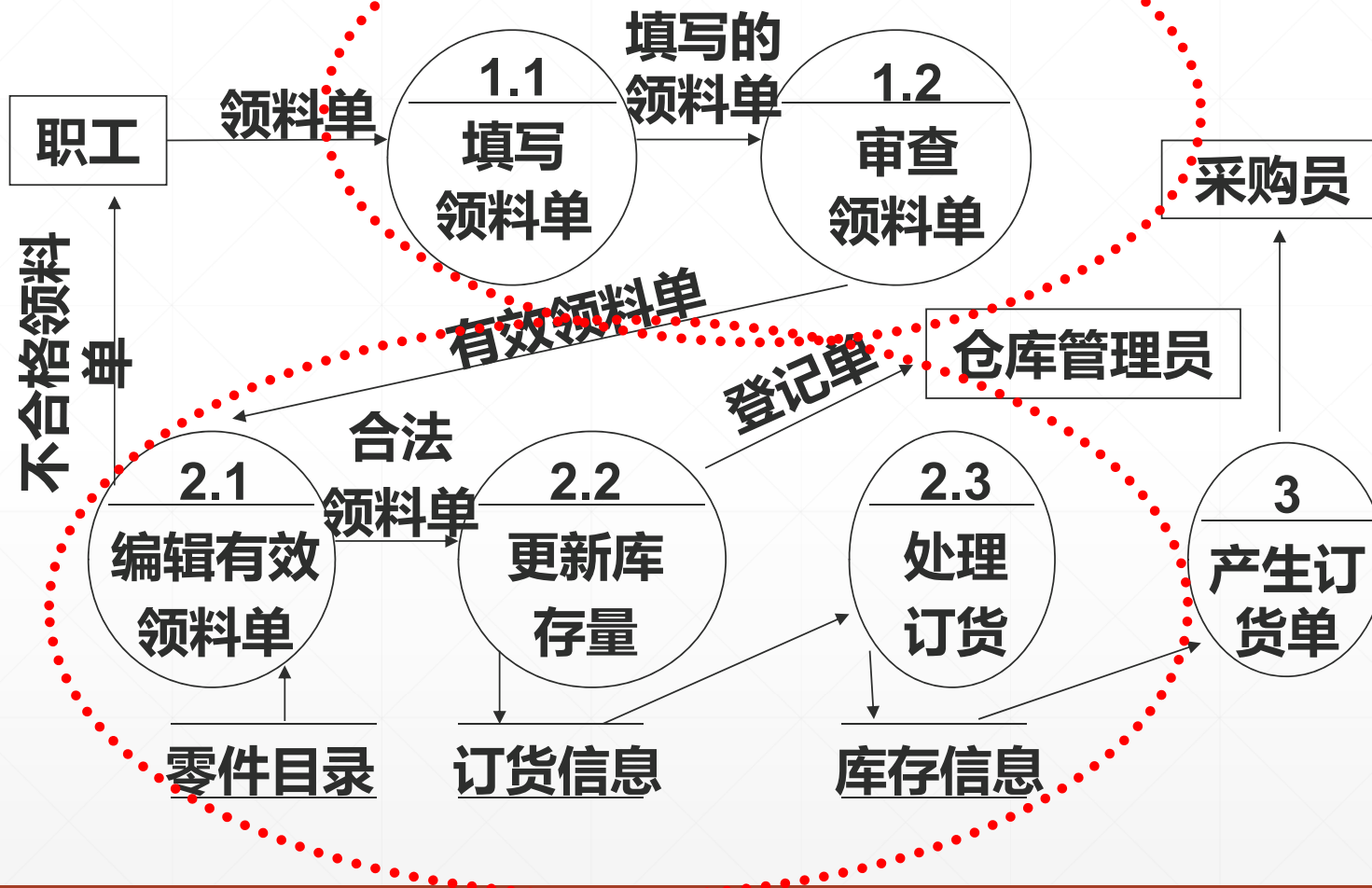


图4 合成的仓库管理系统总数据流程图

绘制步骤：第五步 检查与调整数据流图

在分析过程中，由于每个人的经验和思路不尽相同，对数据流图的分解方案可以有多种形式，不是唯一的。对每一张数据流图**进行检查**，如果太不均衡，就需要进行调整，尽量使分解后的各个软件子系统的复杂性得到均衡，便于今后设计工作的并行开展

3.9 数据流图的改进:



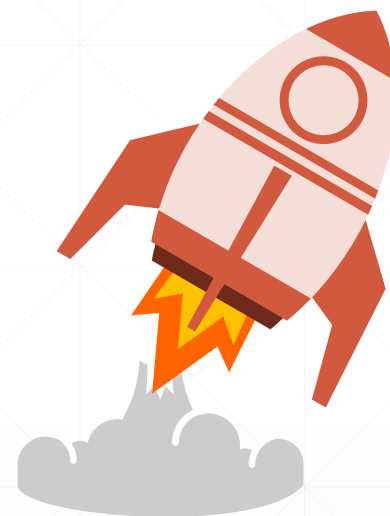
01.

检查正确性



02.

提高易理解性



03.

重新分解

(1) 检查正确性

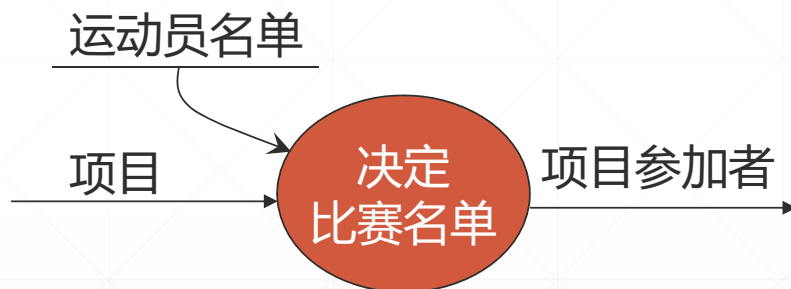
分析员可以从以下几个角度来检查DFD的正确性：

- 数据守恒
 - 数据存储的使用
 - 父图和子图的平衡
-

数据守恒

数据不守恒的情况有**两种**：

1) 某个加工输出的数据并无相应的数据来源，可能是某些数据流被**遗漏**了。



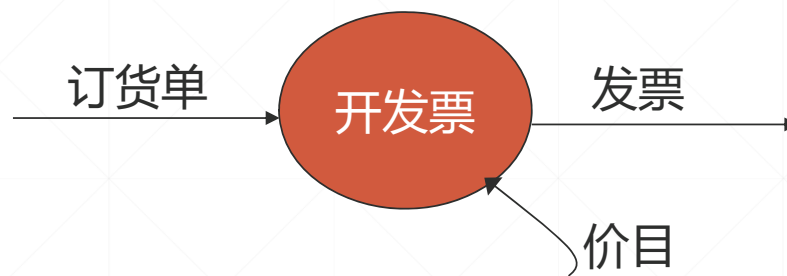
运动员名单=队名+ 姓名+ 项目

项目参加者=项目+ 姓名+ 运动员号

运动员号 并无数据来源

数据守恒

2) 一个加工的输入并没有用到，这不一定是错误。可与用户进一步讨论，是否属于**多余**的数据流。



订货单=单位名+ 货名+ 货号+ 数量

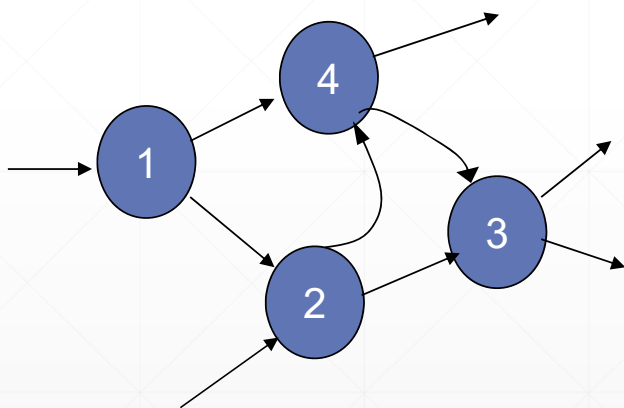
价目=货名+ 单价

发票=单位名+ 货名+ 数量+ 单价+ 总计
经商量， “货号” 确属多余， 故删去。

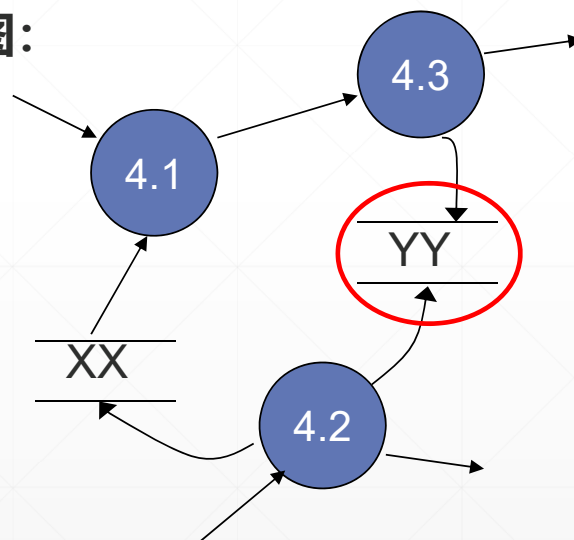
数据存储的使用

判断：是否存在“只读不写”或“只写不读”的数据存储（注意在所有的DFD中检查）

父图：



子图：



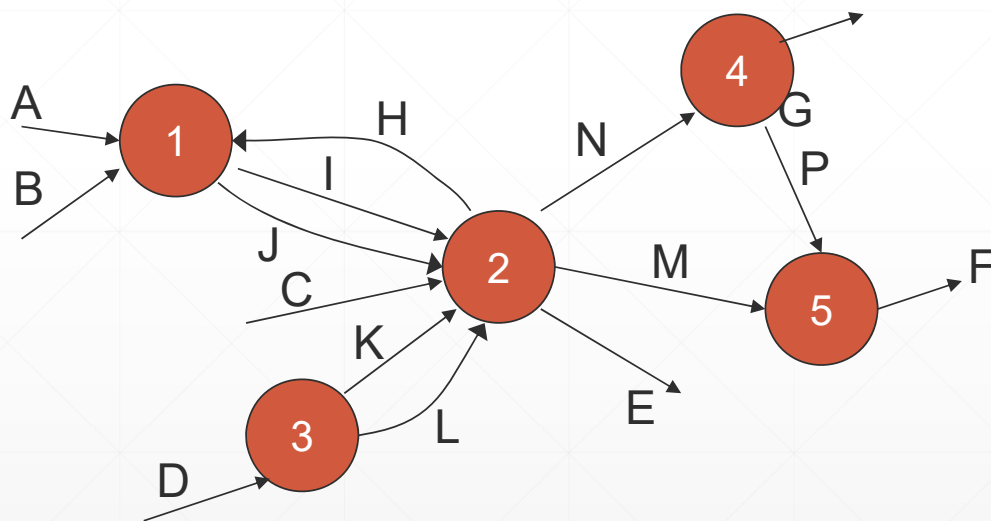
(2) 提高易理解性

- 简化加工之间的联系
 - 注意分解的均匀
 - 适当地命名
-

简化加工之间的联系

应**尽量减少**加工之间输入输出数据流的**数目**。因为加工之间的数据流越少,各个加工的功能就越相对独立。

例:



分解的均匀

即图中各个部分不均匀。

一张图中，如果某些加工已是基本加工（细节），而另一些加工还可进一步分解成三、四层，则应考虑重新分解。

适当地命名

- 名字的意义要明确，容易理解
 - 如果难以为DFD图中的成分（数据流、加工等）命名，往往说明分解不当，可考虑重新分解。
-

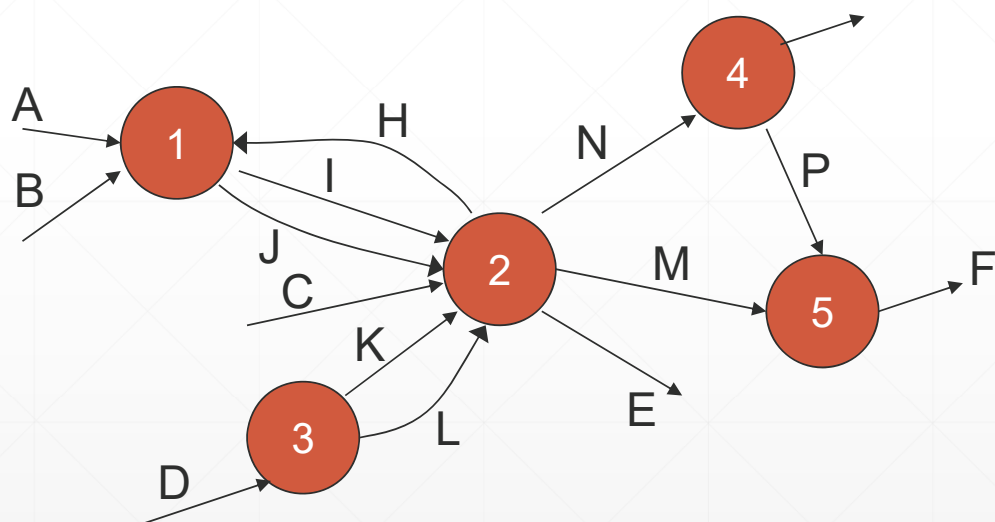
(3) 重新分解

在画第N层时意识到在第N-1层或第N-2层所犯的
错误，此时就需要对第N-1层、第N-2层作重新分
解。

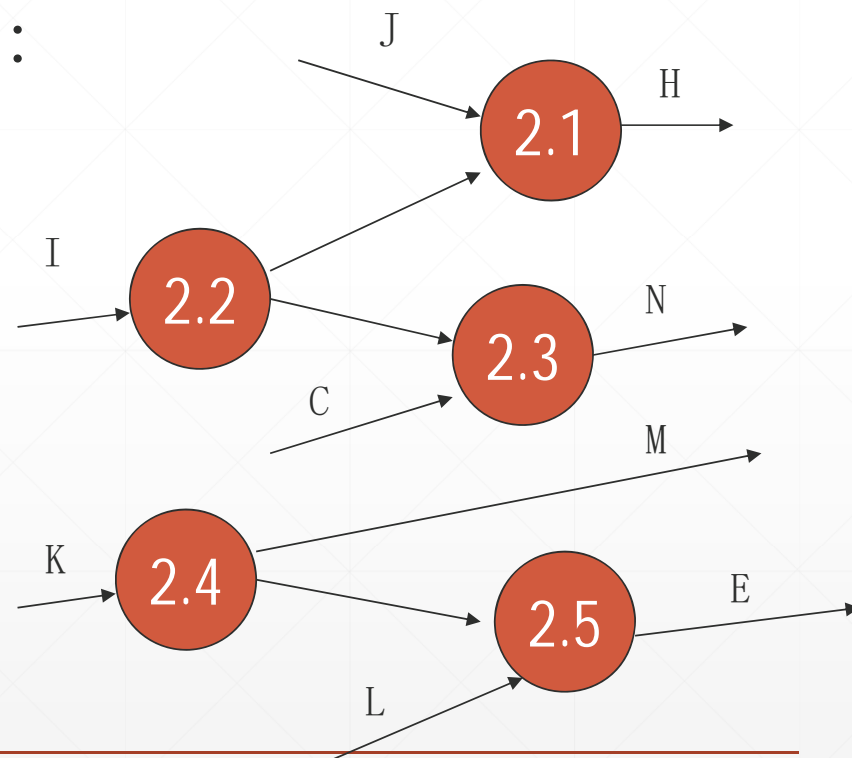
重新分解的做法

1) 把需要重新分解的某张图的所有子图连接成一张。

父图：

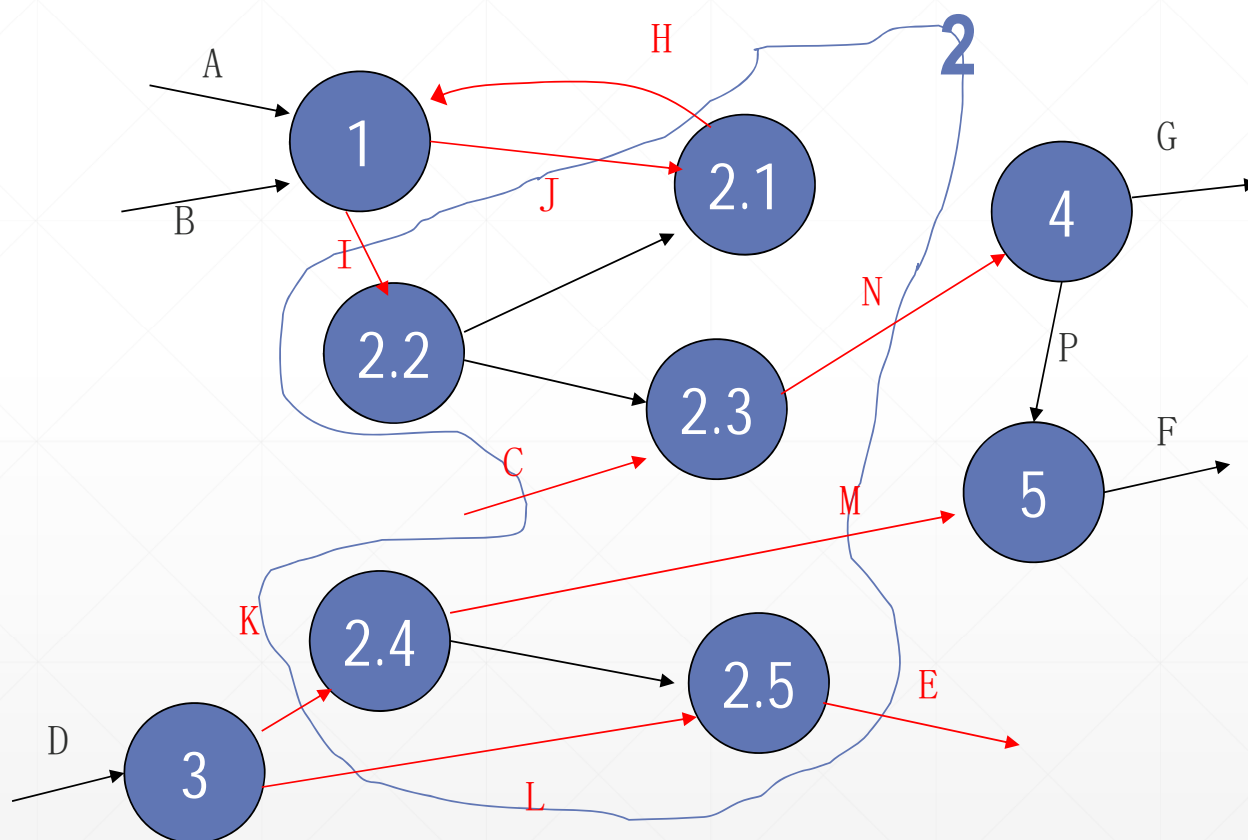


子图：



重新分解的做法

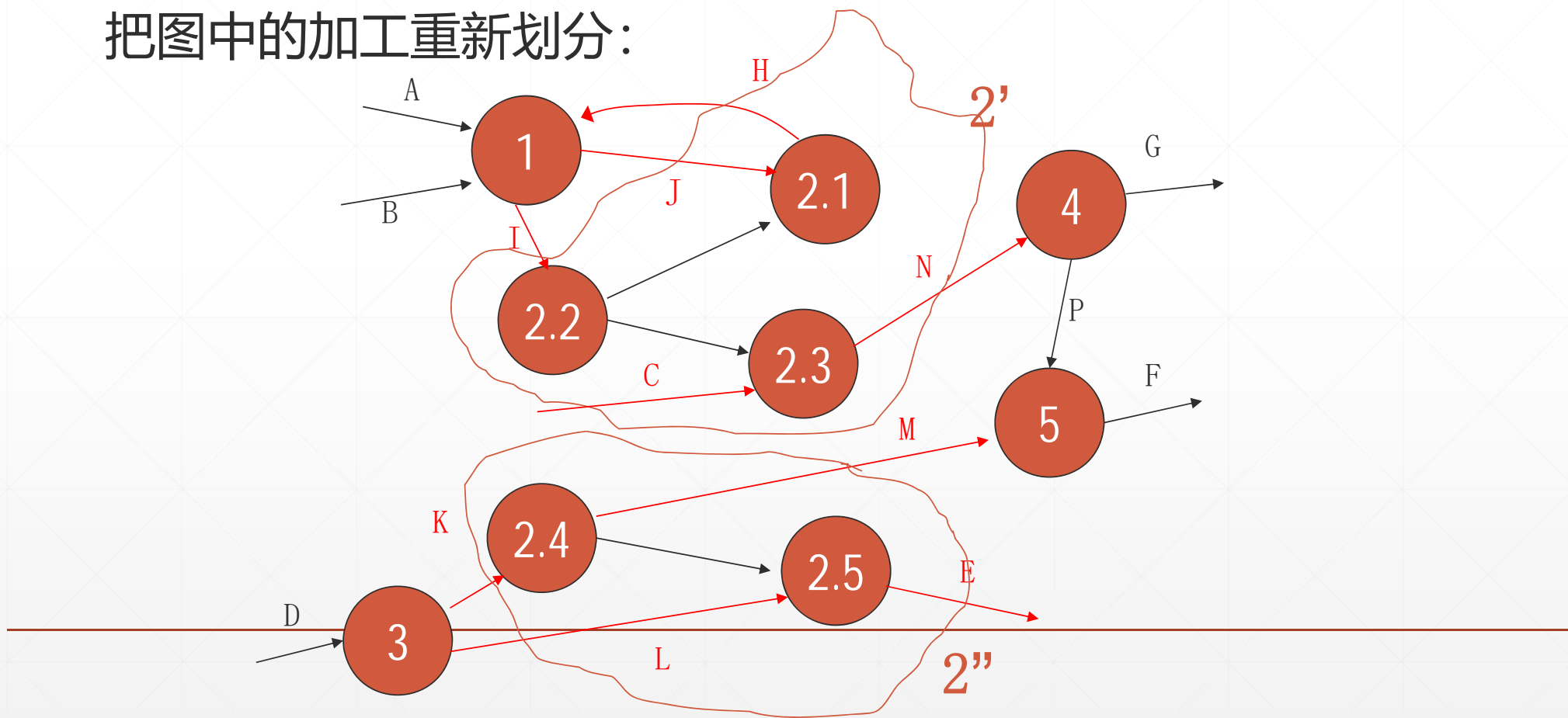
重新连成一张图：



重新分解的做法

2) 把图分成几部分，使各部分之间的联系最少。

把图中的加工重新划分：

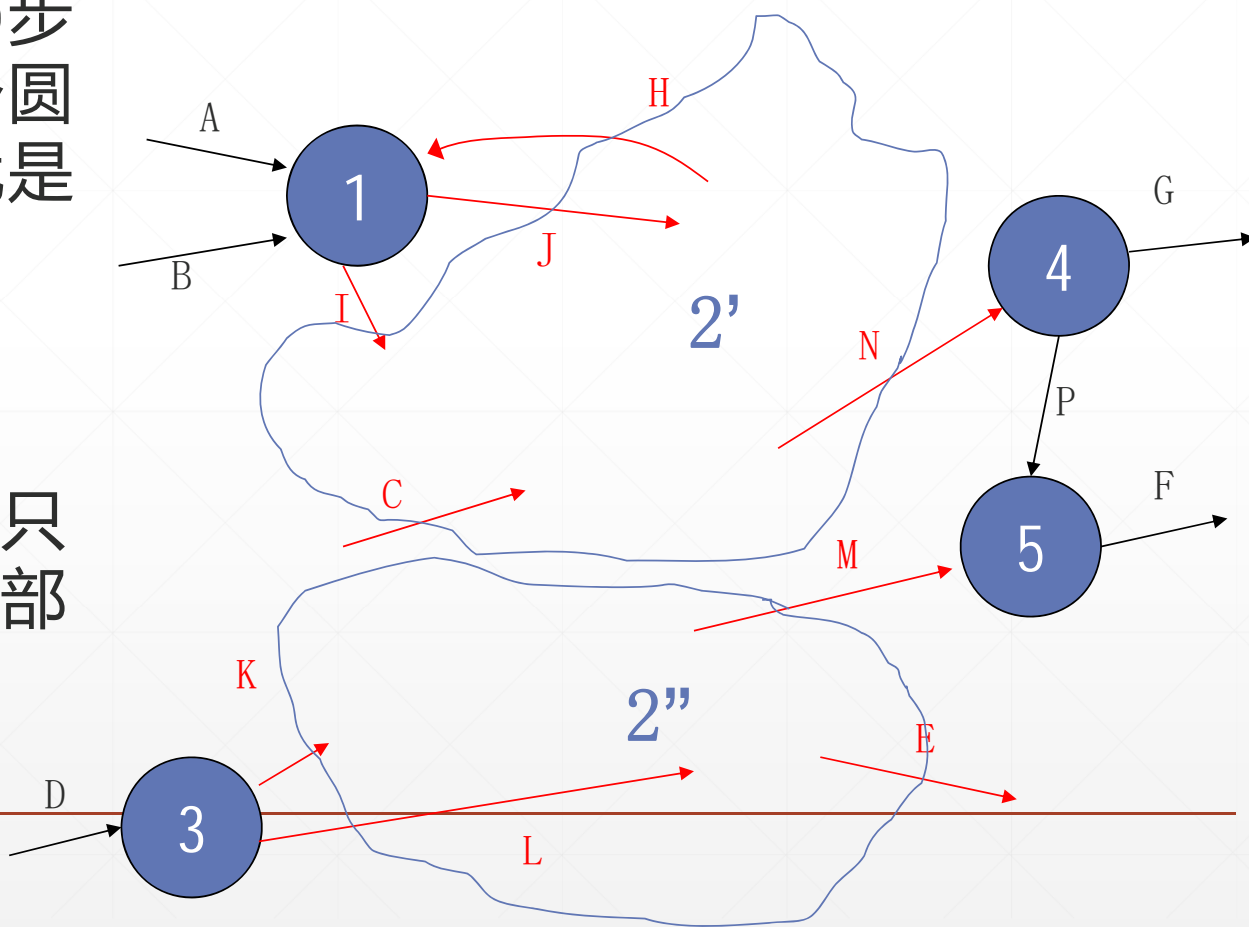


重新分解的做法

3) 重新建立父图，即把第2)步所得的每一部分画成一个圆，而各部分之间的联系就是加工之间的界面。

4) 重新建立各张子图，这只需把第2)步所得的图按各部分的边界剪开即可。

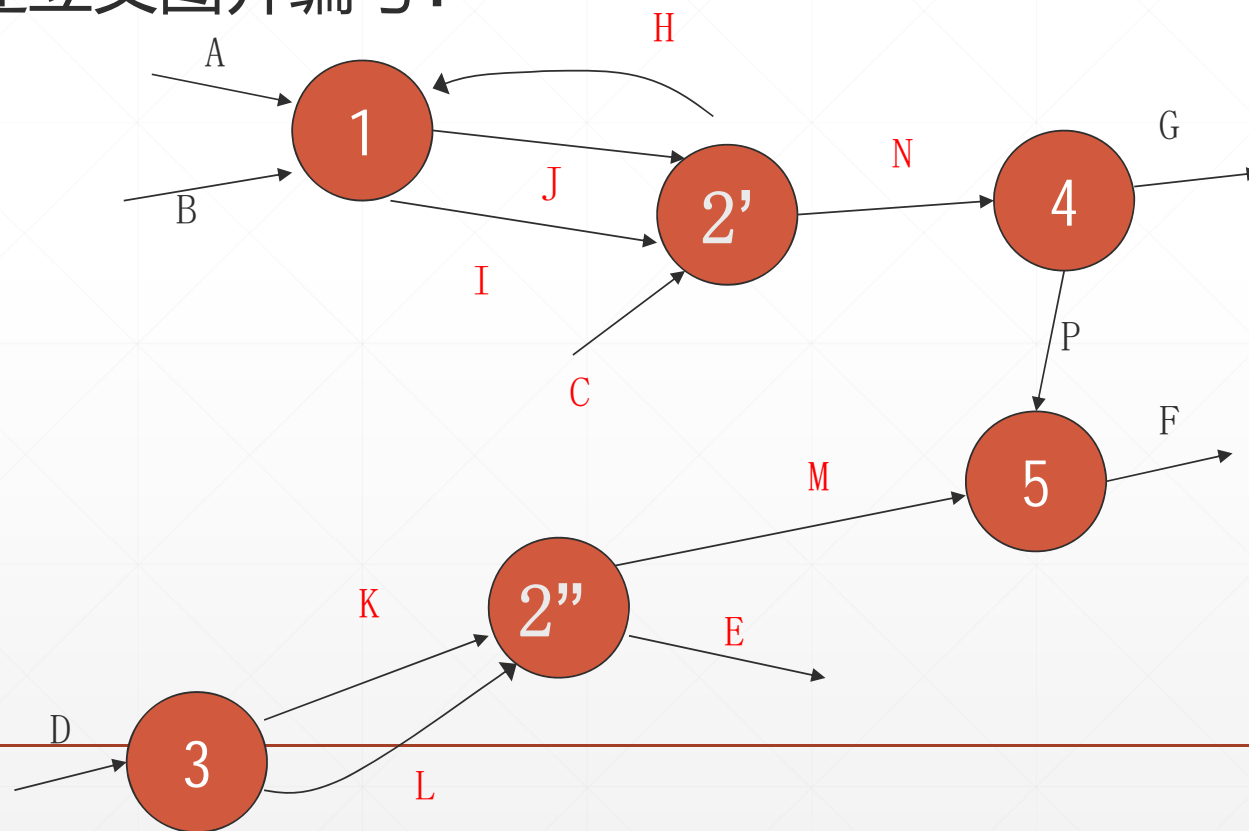
重新建立父图：



重新分解的做法

5) 为所有的加工重新命名和编号。

重新建立父图并编号：



检查数据流图的原则

- 1) 数据流图上所有图形符号只限于前述四种基本图形元素
 - 2) 数据流图的主图必须包括前述四种基本元素，缺一不可
 - 3) 数据流图的主图上的数据流必须封闭在外部实体之间
 - 4) 每个加工至少有一个输入数据流和一个输出数据流
 - 5) 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
 - 6) 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
 - 7) 图上每个元素都必须有名字
 - 8) 数据流图中不可夹带控制流
 - 9) 初画时可以忽略琐碎的细节，以集中精力于主要数据流
-

3.10 功能建模——编写数据字典

编写数据字典，写出系统需求规格说明书，提交审查，并编写测试验收计划、编写初步的用户手册概要。



3.11 什么是对象？

在现实世界中有意义的、与所要解决的问题有关系的任何事物都可以作为**对象**，包括具体的物理实体的抽象、人为的概念、任何有明确边界和意义的东西。

如：一名职工、一本图书、贷款、生产计划、一场演出等。



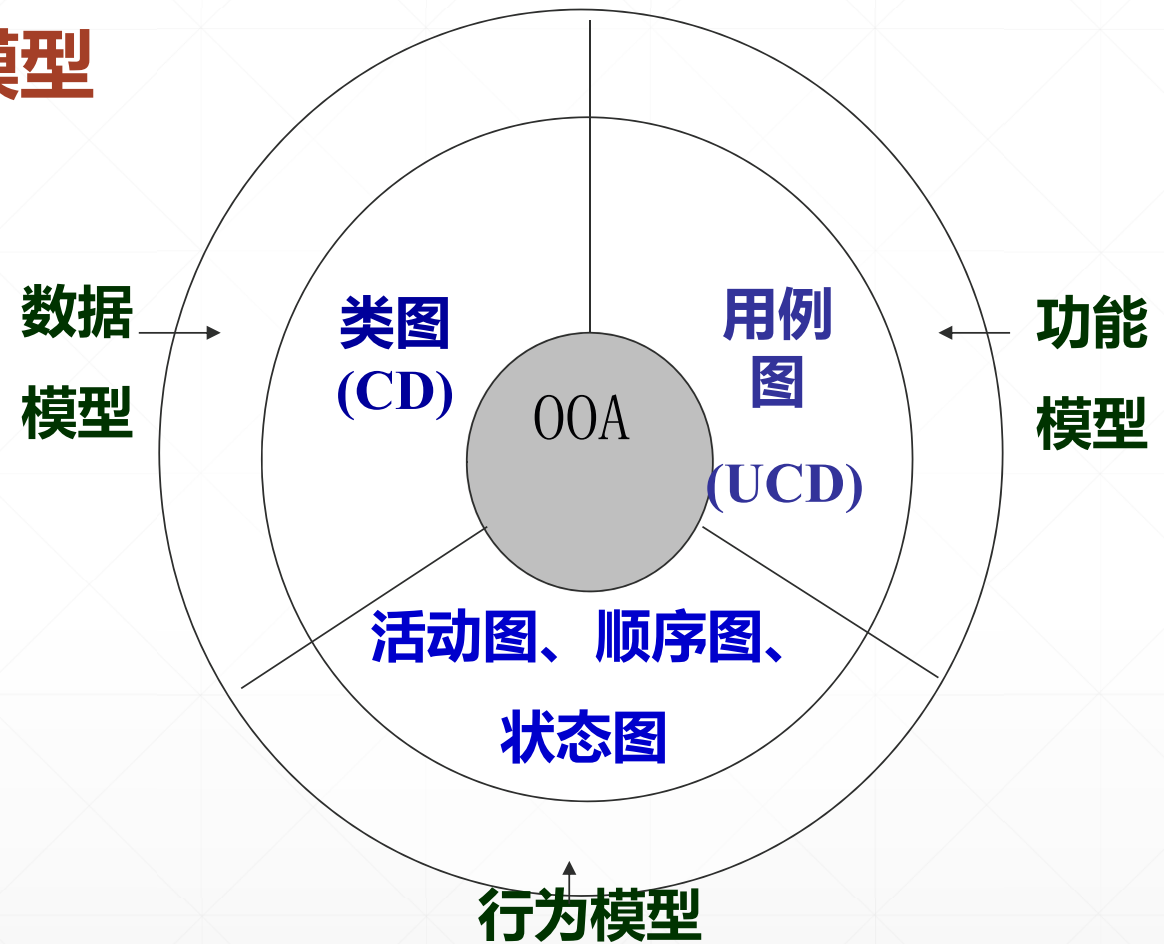
几种著名的面向对象方法

- Booch 方法 (1991)
- Coad — Yourdon 方法 (1991)
- Rumbaugh 方法 (简称 OMT) (object Modeling Technology , 1991)
- Jacobson 方法 (简称 OOSE , 1992)
- 由 Rumbaugh 、 Booch 、 Jacobson 提出的统一建模语言 (简称 UML) (Unify Modeling Language , 1994)

一种可视化建模语言，能描述开发需要的各种视图，并以此为基础组建系统。

3.12 面向对象的软件开发模型

- **数据模型（对象模型）：**
描述系统数据结构的对象模型;
- **行为模型（动态模型）**
描述系统控制结构
- **功能模型**
描述系统功能。



一个典型的软件系统使用**数据结构**（对象模型），**执行操作**（动态模型），并且完成**数据值的变化**（功能模型）。

3.13 功能模型——用例图

(1) 用例图的基本图形符号

- 用例建模用于描述系统需求，把系统当作黑盒，从用户的角度，描述系统的场景。主要图形元素有以下几个：



Student

- 参与者**：是指外部用户或外部实体在系统中扮演的角色。可以是**一个人**、**一个计算机子系统**、**硬件设备**或者**时间**等角色



Purchase Ticket

- 用例**：对**一组**动作序列的描述，系统通过执行**这一组动作序列**为参与者**产生**一个可观察的结果。用例名往往用动宾结构命名。

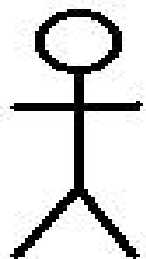
- 执行关联**：参与者（Actor）执行用例（Use Case）之间的关系

(2) 用例建模的过程

- 建立用例模型的顺序是：
 1. 确定谁会直接使用该系统。这些都是参与者(Actor)。
 2. 选取其中一个参与者。
 3. 定义该参与者希望系统做什么，参与者希望系统做的每件事成为一个用例。
 4. 对每件事来说，何时参与者会使用系统，通常会发生什么，这就是用例的基本过程。
 5. 描述该用例的基本过程。
 6. 考虑一些可变情况，把他们创建为扩展用例。
 7. 复审不同用例的描述，找出其中的相同点，抽出相同点作为共同的用例。
 8. 重复步骤2~7找出每一个用例。
-

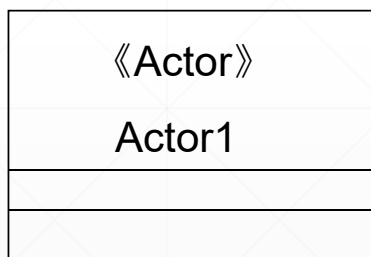
(3) 参与者

- 参与者 (actor) 是指系统以外的、需要使用系统或与系统交互的东西，包括人、设备、外部系统等。
- 参与者的三种表示形式

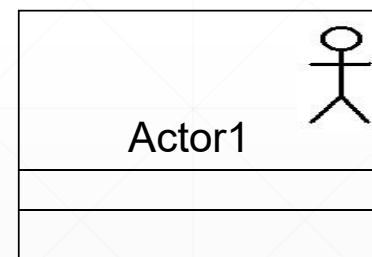


Actor 1

Icon形式



Label形式



Decoration形式

如何确定参与者?

- 在获取用例前首先要确定系统的参与者，开发人员可以通过回答以下的问题来寻找系统的参与者。
 - (1) 谁将使用该系统的主要功能?
 - (2) 谁将需要该系统的支持以完成其工作?
 - (3) 谁将需要维护、管理该系统，以及保持该系统处于工作状态?
 - (4) 系统需要处理哪些硬件设备?
 - (5) 与该系统交互的是什么系统?
 - (6) 谁或什么系统对本系统产生的结果感兴趣?
-

如何确定参与者?

示例：饮料自动售货系统

在饮料自动售卖机中，最先想到的参与者是**顾客**。

供应商向自动贩卖机添加饮料。
收银员从自动贩卖机收钱。

谁是参与者



每一种参与者具有自己的 use case

(4) 用例 (Use Case)

○ “用例” 的定义

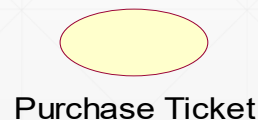
- 对一组动作序列的描述，系统通过执行这一组动作序列为参与者产生一个可观察的结果

○ 用例特征

- 说明了系统具有的一种行为模式
- 说明了一个参与者与系统执行的一个相关的事件序列
- 提供了一种获取系统需求的方法
- 提供了一种与最终的用户和领域专家进行沟通的方法
- 提供了一种测试系统的方法

○ 图形表示

- 用椭圆形表示



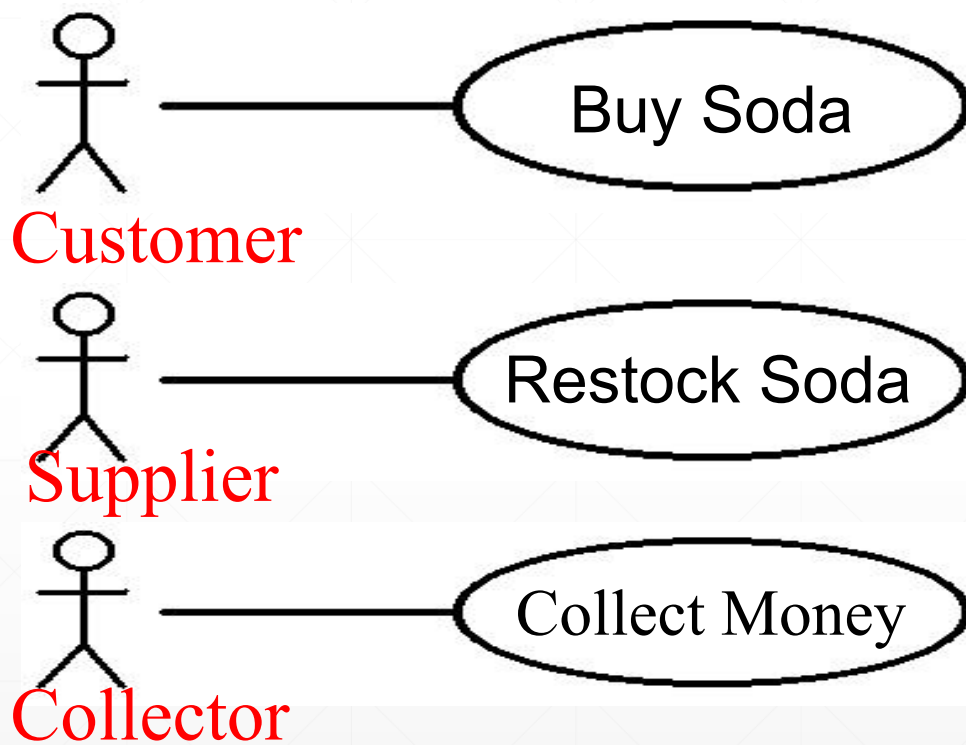
怎么获取用例？

- 参与者希望用户执行什么任务？
- 参与者在系统中访问哪些信息（创建、存储、修改、删除等）？
- 需要将哪些外界信息提供给系统
- 需要将系统的什么事情告诉参与者
- 如何维护系统



怎么获取用例?

每一种参与者具有自己的 use case



系统和关联

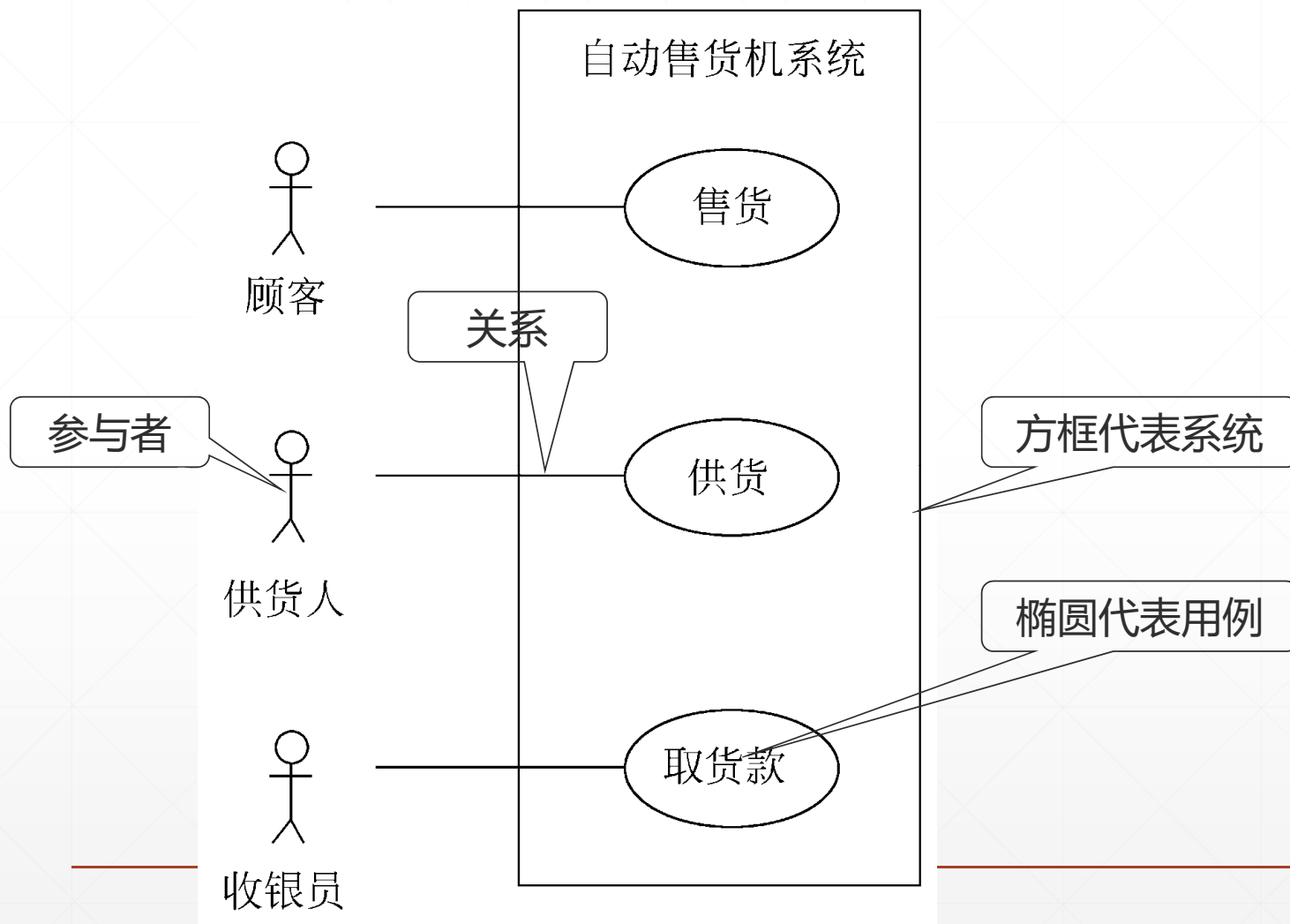
- **系统：**用于界定系统功能范围，描述该系统功能的用例都置于其中，而描述外部实体的参与者都置于其外。



- **关联：**连接参与者和用例，表示参与者所代表的系统外部实体与该用例所描述的系统需求有关。



自动售货系统 初步的用例图



• 建立用例模型的顺序是：

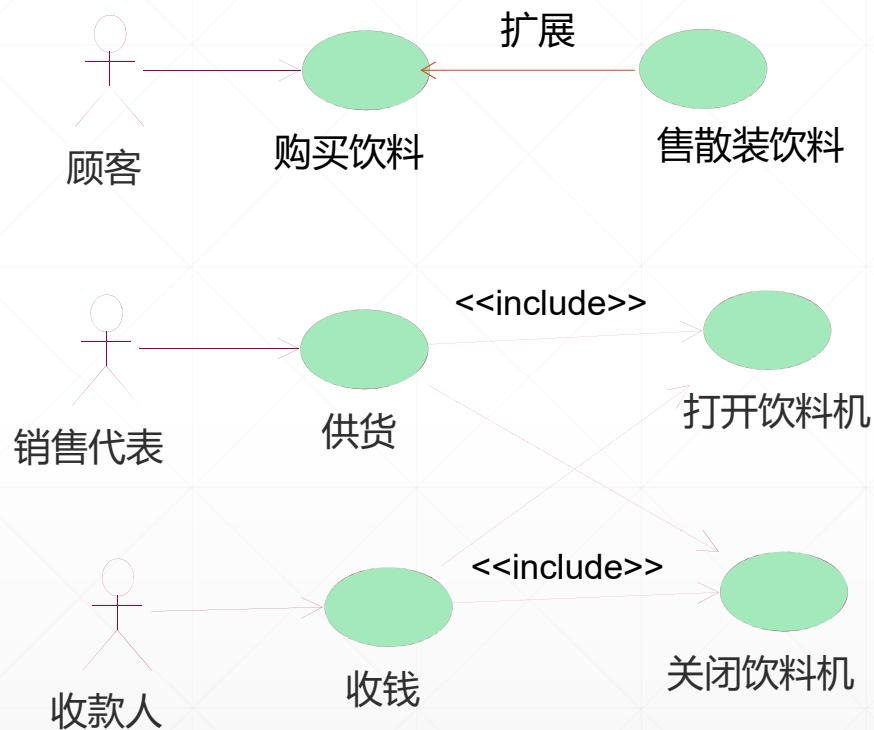
1. 确定谁会**直接使用**该系统。这些都是参与者(Actor)。
2. 选取其中一个参与者。
3. 定义该参与者**希望系统做什么**，参与者希望系统做的每件事成为一个用例。
4. 对每件事来说，**何时**参与者会使用系统，通常会**发生什么**，这就是用例的基本过程。
5. 描述该用例的**基本过程**。

用例扩展

- 建立用例模型的顺序是：

1. 确定谁会直接使用该系统。这些都是参与者(Actor)。
 2. 选取其中一个参与者。
 3. 定义该参与者希望系统做什么，参与者希望系统做的每件事成为一个用例。
 4. 对每件事来说，何时参与者会使用系统，通常会发生什么，这就是用例的基本过程。
 5. 描述该用例的基本过程。
 6. 考虑一些可变情况，把他们创建为扩展用例。
 7. 复审不同用例的描述，找出其中的相同点，抽出相同点作为共同的用例。
 8. 重复步骤2~7找出每一个用例。
-

自动售货系统 用例扩展后的用例图



6. 考虑一些可变情况，把他们创建为扩展用例。

7. 复审不同用例的描述，找出其中的相同点，**抽出相同点**作为共同的用例。

8. 重复步骤2~7找出每一个用例。

3.14 用例之间的关系



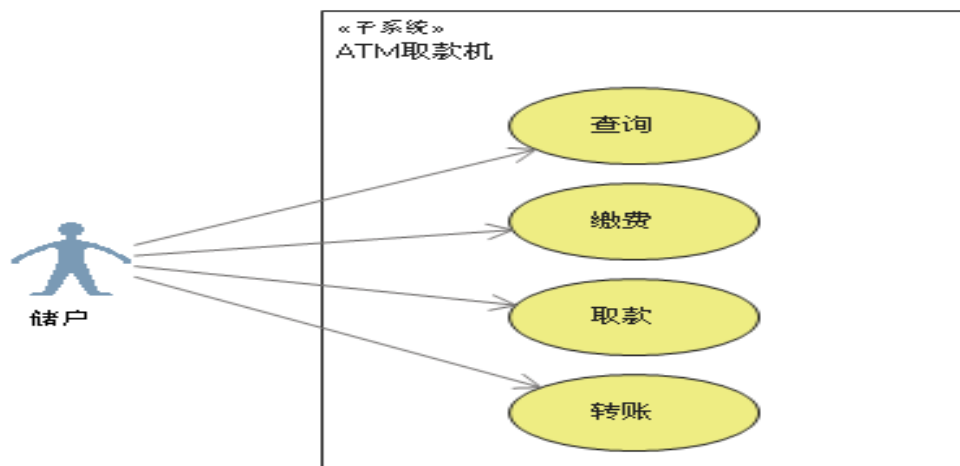
(1) 关联(Association)

表示参与者与用例之间的通信，任何一方都可发送或接受消息。

【箭头指向】：指向消息接收方



备注：参与者可以参与多个用例，由此形成子系统

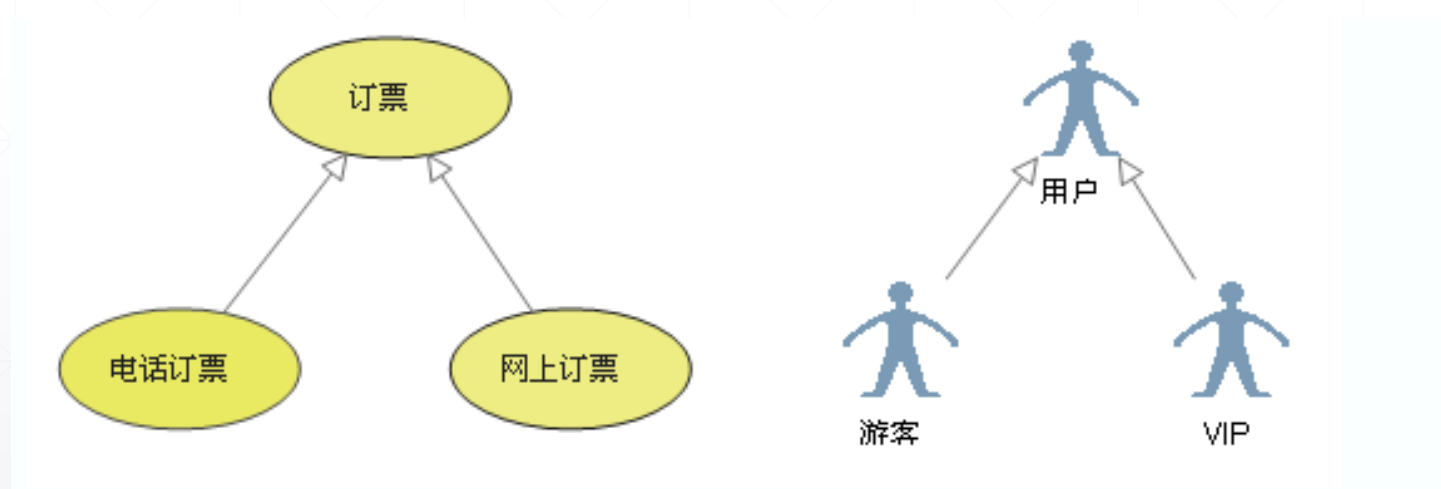


(2) 泛化(Inheritance)

就是通常理解的**继承关系**，**子**用例和**父**用例相似，但表现出更特别的行为；子用例将继承父用例的所有结构、行为和关系。子用例可以使用父用例的一段行为，也可以重载它。父用例通常是抽象的。

用例之间的**is a kind of** 关系，表示用例之间的场景共享；Actor之间的 **is a kind of** 关系，一般描述职责共享。

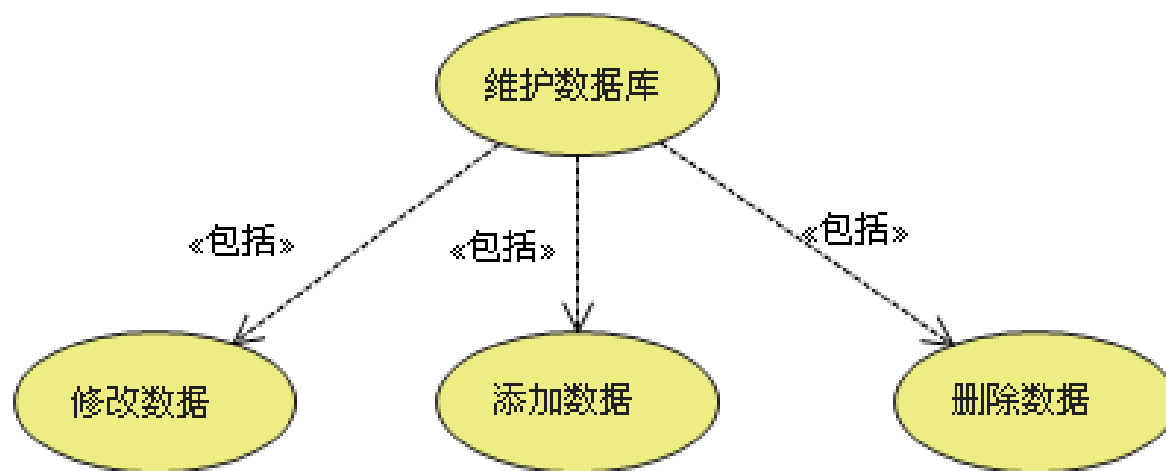
【箭头指向】：**指向父用例**



(3) 包含(Include)

包含关系用来把一个较复杂用例所表示的功能分解成较小的步骤。一个用例可以包含另外一个用例。

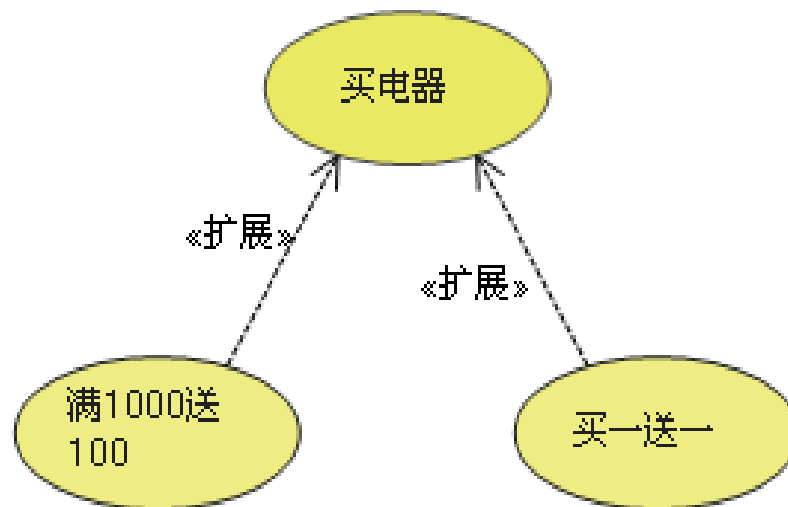
【箭头指向】：指向分解出来的功能用例



(4) 扩展(Extend)

扩展关系是指用例功能的**延伸**，相当于为基础用例提供一个附加功能。由一个用例的**扩展点**可以扩展出另外一个用例。





【箭头指向】：**指向基础用例**



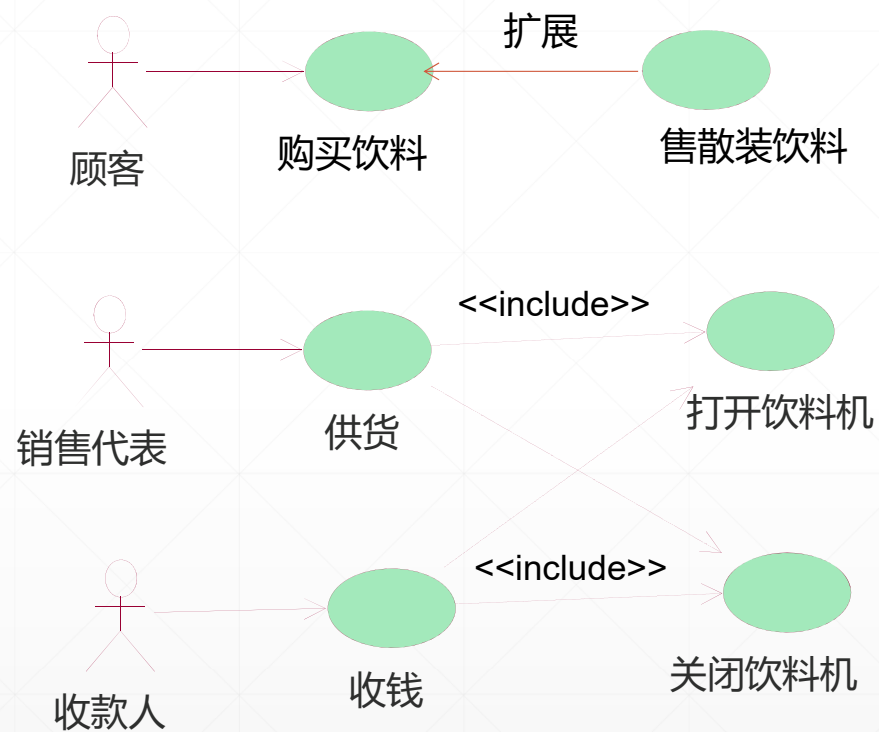
包含(include)、扩展(extend)的区别:

- 在扩展关系中, 一个基本用例执行时, 可以执行、也可以不执行扩展用例部分
 - 在包含关系中, 在执行基本用例时, 一定会执行包含用例部分
-

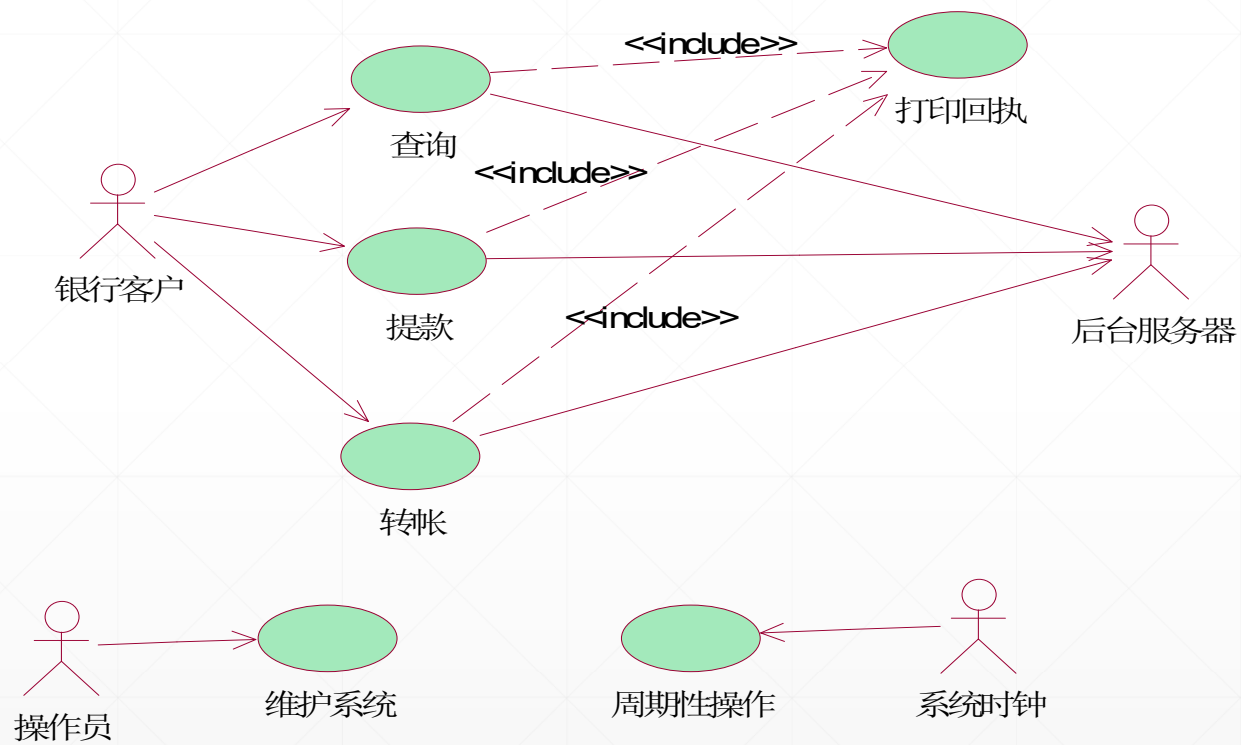
关联、包含、扩展、泛化的区别：

关系	功能	表示法
关联	参与者与其参与执行的用例之间的通信途径	
扩展	在基础用例上插入基础用例不能说明的扩展部分	
泛化	用例之间的一般和特殊关系，其中特殊用例继承了一般用例的特性并增加了新的特性	
包含	在基础用例上插入附加的行为，并且具有明确的描述	

自动售货系统 用例扩展后的用例图

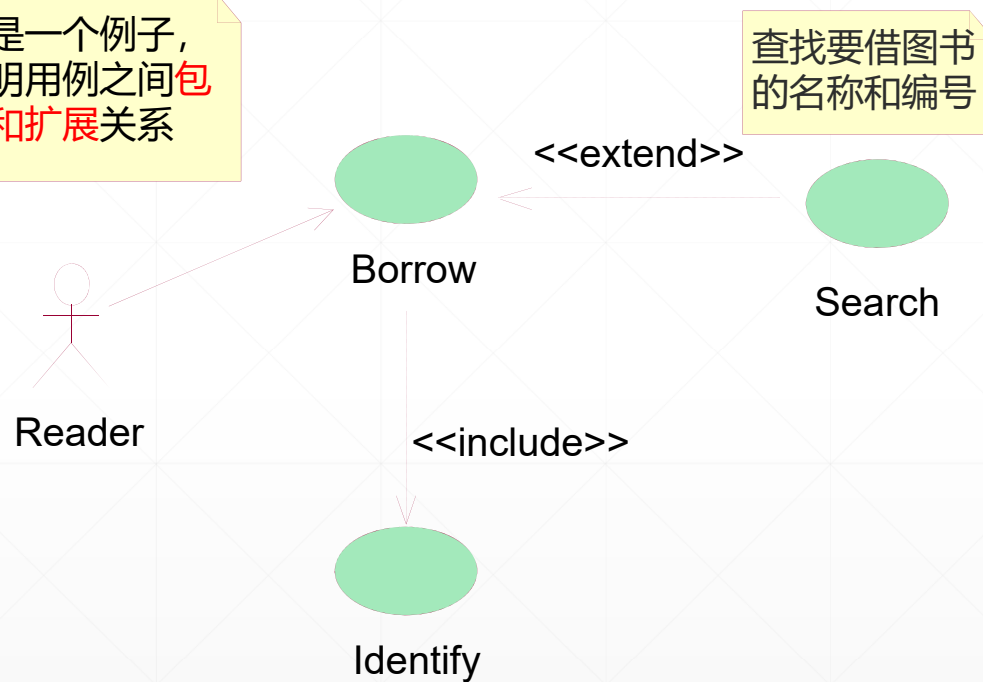


用例实例一：ATM系统用例图



用例实例二：图书借阅系统用例图

这是一个例子，
说明用例之间**包**
含和扩展关系





谢谢!