

# 软工导论

---

## 概论

---

### 软件工程

计算机软件是指计算机系统中的**程序**及其**文档**

软件的特点：

- 软件是一种**逻辑实体**，而不是有形的系统元件，其开发成本和进度难以准确地估算。
- 软件是**被开发的或被设计的**，**没有明显的制造过程**，一旦开发成功，只需复制即可，但其维护的工作量大。
- 软件的使用**没有硬件**那样的机械磨损和老化问题。

软件工程定义：是应用计算机科学、数学及管理科学等原理开发软件的工程。软件工程借鉴传统工程的原则、方法，以提高质量、降低成本为目的。

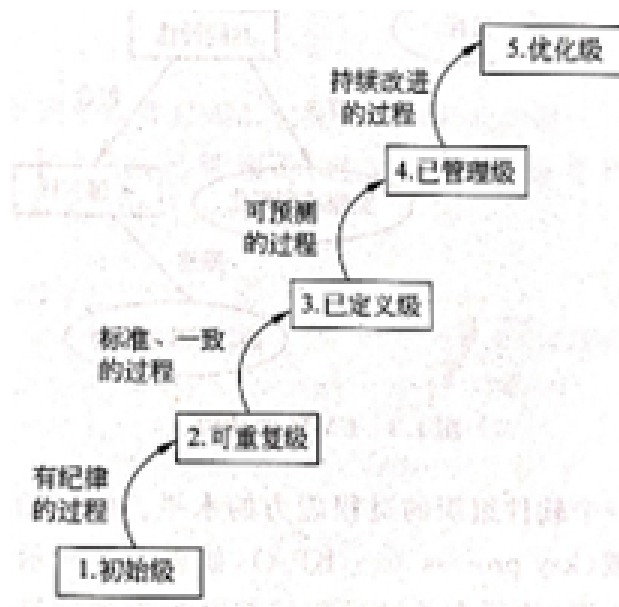
### 软件工程生存周期

软件工程生存周期大致可以分为6个阶段：

- 计算机系统工程
  - 计算机系统工程的任务是**确定待开发软件的总体要求和范围**，以及该软件与其他计算机系统元素之间的关系，进行成本估算，作出进度安排，并进行可行性分析。
- 需求分析
  - 需求分析主要解决待开发软件要**“做什么”**的问题，确定软件的功能、性能、数据、界面等要求，生成软件需求规约（也称软件需求规格说明）。
- 设计
  - 系统设计的任务是**设计软件系统的体系结构**，详细设计的任务是设计各个组成成分的实现细节，包括局部数据结构和算法。
- 编码
  - 编码阶段的任务是**用某种程序设计语言**，将设计的结果转换为可执行的程序代码。
- 测试
  - 测试阶段的任务是**发现并纠正软件中的错误和缺陷**。
- 运行和维护
  - 软件完成各种测试后，就可以交付使用，在软件运行期间，需对投入运行的软件进行维护，即当发现了软件中潜藏的错误或需要增加新的功能或使软件**适应外界环境的变化**等情况出现时，对软件进行修改。

### 能力成熟度模型 CMM (不考)

CMM模型定义了5个软件过程成熟度等级



- 初始级
  - 软件过程的特点是**无秩序的**，甚至是混乱的。几乎没有什么过程是经过妥善定义的，成功往往依赖于个人或小组的努力。
- 可重复级
  - 建立了**基本的项目管理过程**来跟踪成本、进度和功能特性。制定了必要的过程纪律，能重复早先类似应用项目取得的成功。
- 已定义级
  - 已将管理和工程活动两方面的软件过程**文档化、标准化**，并综合成该组织的标准软件过程。所有项目均使用经批准、剪裁的标准软件过程来开发和维护软件。
- 已管理级
  - 收集对软件过程和产品质量的详细度量值，**对软件过程和产品都有定量的理解和控制**。
- 优化级
  - 过程的量化反馈和先进的新思想、新技术促使过程**不断改进**。

## 软件过程模型

- 瀑布模型
  - 上一阶段的活动完成并经过评审才能开始下一阶段的活动，接受上一阶段活动的结果作为本阶段活动的输入，依据上一阶段活动的结果实施本阶段应完成的活动，对本阶段的活动进行评审。（适合对**软件需求明确**的情况下）
- 演化模型
  - 从结构初始的原型出发，逐步将其演化成最终软件产品的过程。演化模型特别适用于对软件需求缺乏准确认识的情况。（适合对**软件需求缺乏准确认识**的情况下）
- 增量模型
  - 将软件的开发过程分为若干个日程时间交错的线性序列，融合了瀑布模型的基本成分（重复地应用）和演化模型的迭代特征，特别适用于需求经常发生变化的软件开发。（适合**需求经常发生变化**的情况下）
- 原型模型
  - 开发人员和用户在“原型”上达成一致，缩短了开发周期，加快了工程进度，降低成本。（**不适合大规模开发**）
- 螺旋模型

- 将原型实现的迭代特征与瀑布模型中控制的和系统化的方面结合起来，不仅体现了这两种模型的优点，而且增加了风险分析。（**风险分析...**）
- 喷泉模型
  - 各个阶段没有明显的界限，开发人员可以同步进行开发，可以提高软件项目开发效率，节省开发时间，适应于面向对象的软件开发过程。（**面向对象、有较好的可移植性**）
- 基于构件的开发模型
  - 利用预先包装的构件来构造应用系统。（**模块较多、模块成熟**）
- 形式化方法模型
  - 易于发现需求的歧义性、不完整性和不一致性，易于对分析模型、设计模型和程序进行验证。（建立严谨的数学基础上，易发现需求的歧义与不完整）

增量模型与原型模型的区别：

- 增量模型：质量与最终版的基本一致
- 原型模型：前期无法使用

## 计算机辅助软件工程 CASE

CASE已被证明可以加快开发速度，提高应用软件生产率并保证应用软件的可靠品质。计算机专业人员利用计算机使他们的企业提高了效率,企业的各个部门通过使用计算机提高了生产率和效率,增强了企业的竞争力并使之带来了更多的利润。

## 习题

对于下列每一个过程模型,分别列举一个可以适用的具体软件项目,并说明在开发中如何应用该模型。

- 某项目需要在一种新型机器上,为一种已知语言开发一个普通的编译器。  
 选用模型：瀑布模型  
 选用分析:由于该项目的**语言是已知的,需求是明确的和稳定的**,整个系统属于中小规模,因此适合采用瀑布模型进行软件开发。
- 某公司需要给火车站开发一个交互式火车车次查询系统,这是火车站**首次使用该系统**。  
 选用模型：快速原型模型  
 选用分析：本项目的主要问题在于用户需求方面，该系统与最终用户的交互是十分关键的，但是在项目的初期，因为是首次使用该系统，**用户的需求基本上是不知道的**，因此适合采用快速原型法来确定需求，在需求确定的基础上再开发最终的系统。
- 某公司开发一个通用CAD软件产品，产品需求是逐步完善的，某些需求在一定范围内是明确的，某些需求需要进一步细化，但是迫于市场竞争的压力，产品需要**尽快上市**。  
 选用模型：增量模型  
 选用分析：通用CAD软件产品具有一定的成熟度，某些需求和软件系统结构是可以确定的。但是实现该产品所有功能需要比较长的开发周期，为了尽快上市可以采用增量模型实行多版本的发布策略，既可以很快占领市场又可以为后继版本的需求定义奠定基础。
- 某公司开发企业管理ERP系统，包括销售、库存、生产、财务、物流、人力资源等部分，在系统实施过程中**不同的企业具有一定的需求差异**。  
 选用模型：基于构件的开发模型  
 选用分析：企业ERP系统具有构件化的结构，在不同企业实施时应该尽量重用已有的组件。因此适合采用基于构件的开发模型开发该系统，在直接应用或修改使用的基础上，最终进行组件开发和系统集成。
- 某公司开发一个汽车防抱死刹车控制系统。  
 选用模型：形式化方法模型

选用分析：由于该系统对**安全性和可靠性要求极高**，需要在系统运行之前进行相关性能的检查，性能检查由复杂的数学运算实现，因此适合采用形式化方法开发该系统。

## 系统工程

---

### 系统工程的任务

- 识别用户的要求
  - 识别用户对基于计算机的系统的总体要求，标识系统的功能和性能范围，确定系统的功能、性能、约束和接口。
- 系统建模和模拟
  - 一个基于计算机的系统通常可考虑建立以下模型：硬件系统模型、软件系统模型、人机接口模型、数据模型。
- 成本估算及进度安排
  - 开发一个基于计算机的系统需要一定的资金投入和时间约束（交互日期），需进行成本估算，并作出进度安排。
- 可行性分析
  - 主要从经济、技术、法律等方面分析所给出的解决方案是否可行。
- 生成系统规格说明
  - 作为以后开发基于计算机的系统的依据。

### 可行性分析

- 经济可行性
  - 成本
  - 效益
  - 货币的时间价值
  - 投资的回收期
  - 纯收入
- 技术可行性
  - 风险分析
  - 资源分析
  - 技术分析
- 法律可行性
- 方法的选择和折衷

## 需求工程

---

定义：直到（但不包括）把软件分解为实际架构和构建之前的所有活动。

### 需求工程步骤

需求工程主要包括以下6个步骤：

- **需求获取**：系统分析人员通过与用户的交流、对现有系统的观察及对任务进行分析。
- **需求分析与协商**：分析每个需求与其他需求的关系以检查需求的一致性、重叠和遗漏的情况，并根据用户的需求对需求进行排序。

- **系统建模**：通过合适的工具和符号系统地描述需求。
- **需求规约**：给出对目标软件的各种需求。
- **需求验证**：对功能的正确性、完整性和清晰性以及其它需求给予评价。
- **需求管理**：对需求工程所有相关活动的规约和控制。

## 系统分析员的思想素质与能力（屁话）

- 能够熟练地掌握计算机硬、软件的专业知识，具有一定的系统开发经验。
- 善于进行抽象的思维和创造性的思维，善于把握抽象的概念，并把它们重新整理成为各种逻辑成分，并给出简明、清晰的描述。
- 善于从相互冲突或混淆的原始资料中抽出恰当的条目来。
- 善于进行调查研究，能够很快学习用户的专业领域知识，理解用户的环境条件。
- 能够倾听他人的意见，注意发挥其它人员的作用。
- 具有良好的书面和口头交流表达能力。

## 软件需求

定义：是指用户对目标软件系统在功能、行为、性能、设计约束等方面的期望。P35

在指定需求获取策略时的**3种主要考虑因素**：

- 功能需求
  - 考虑系统要做什么，在何时做，在何时及如何修改或升级
- 性能需求
  - 考虑软件开发的技术性指标
- 用户或人对因素
  - 考虑用户的类型

## 非功能性需求（不考）

定义：是指软件产品为满足用户业务需求而必须具有且除功能需求以外的特性。软件产品的非功能性需求包括系统的性能、可靠性、可维护性、可扩充性和对技术和对业务的适应性等。

例如在银行管理系统中，由于银行数据量的庞大以及对银行账户的管理需求，用户对系统的性能、可靠性、可维护性要求很高。安全性是对银行用户个人信息保密的基本要求；在使用系统时，由于用户庞大，要求能快速安全的执行要求，这就对系统的性能有高需求；银行的用户的变动比较大，需求高要求的系统维护。

## 需求工程的指导性原则（屁话）

- **在开始建立分析模型之前应当先理解问题**。如果问题没有很好理解就急于求成，常常会产生一个解决错误问题的完美的软件。
- **强力推荐使用原型**。这样做可以使用户了解将如何与计算机交互，而人们对软件质量的认识常常是基于对界面“友好性”的切身体会。
- **记录每一个需求的起源和原因**。这是建立对用户要求的可追溯性的第一步。
- **使用多个视图，建立系统的数据、功能和行为模型**。这样做可帮助分析员从多方面分析和理解问题，减少遗漏，识别可能的不一致之处。

- **给需求赋予优先级。**因为过短的时限会减少实现所有软件需求的可能性。因此，对需求排定一个优先次序，标识哪些需求先实现，哪些需求后实现。
- **注意消除歧义性。**因为大多数需求都是以自然语言描述，存在叙述的歧义性问题，会造成遗漏和误解。采用正式的技术评审是发现和消除歧义性的好方法。

## 需求获取的方法和策略

- 建立顺畅的通信途径
- 访谈与调查
- 亲身实践
- 会议
- 头脑风暴
- 概念建模
- 原型、仿真
- 自省
- 用户行为数据在线采集

## 设计工程

---

定义：软件设计是把软件需求变换成软件表示的过程。P47

### 软件设计的任务（不考）

- **数据/类设计：**将分析类模型变换成为类实现和软件实现所需要的数据结构。
- **体系结构设计：**定义了软件的整体结构，由软件部件、外部可见的属性和他们之间的关系组成。
- **接口设计：**描述了软件内部、软件和协作系统之间以及软件同人之间的通信方式。
- **部件级设计：**将软件体系结构的结构性元素变换为对软件部件的过程性描述。

### 软件设计原则

- 抽象与逐步求精
- 模块化
- 信息隐藏
- 功能独立

### 模块化设计

- **模块：**数据说明、可执行语句等程序对象的集合，是单独命名的，并且可以通过名字来访问的。
- **模块化：**把软件按照规定原则，划分为一个个较小的，相互独立的但又相互关联的部件。
- **模块化设计：**程序的编写不是开始就逐条录入计算机语句和指令，而是首先用主程序、[子程序](#)、子过程等框架把[软件](#)的主要结构和流程描述出来，并定义和调试好各个框架之间的输入、输出链接关系。

## 内聚

定义：内聚是一个模块**内部**各个元素**彼此结合**的**紧密程度的度量**。P51

一个内聚程度高的模块在理想状态下应当只做一件事情。内聚程度由低至高：

- **巧合内聚**：将几个模块中没有明确表现出独立功能的**相同程序代码段独立出来建立的模块**称巧合内聚模块。
- **逻辑内聚**：逻辑内聚是指完成一组逻辑相关任务的模块，调用该模块时，由**传送给模块的控制性参数**来确定该模块应执行哪一种功能。
- **时间内聚**：时间内聚是指一个模块中的所有任务必须**在同一时间段内执行**。
- **过程内聚**：过程内聚是指一个模块完成多个任务，这些任务必须在**指定的过程执行**。
- **通信内聚**：通信内聚是指一个模块内所有处理元素都**集中在某个数据结构的一块区域中**。
- **顺序内聚**：顺序内聚是指一个模块**完成多个功能，这些功能又必须顺序执行**。
- **功能内聚**：功能内聚是指一个模块中**各个部分都是为完成一项具体功能而协同工作，紧密联系、不可分割**。

## 耦合

定义：耦合是指模块之间的相对独立性（**互相连接的紧密程度**）的度量。

耦合程度由高到低：

- **内容耦合**：当一个模块**直接修改或操作另一个模块的数据**，或者直接转入另一个模块时就发生了内容耦合。此时，被修改的模块完全依赖于修改它的模块。如果发生下列情形，两个模块之间就发生了内容耦合：
  - a) 一个模块直接访问另一个模块的内部数据
  - b) 一个模块不通过正常入口转到另一模块内部
  - c) 两个模块有一部分程序代码重叠(只可能出现在汇编语言中)
  - d) 一个模块有多个入口。
- **公共耦合**：若一组模块都访问同一个**公共数据环境**，则它们之间的耦合就称为公共耦合。公共的数据环境可以是全局数据结构、共享的通信区、内存的公共覆盖区等。
- **外部耦合**：一组模块都访问**同一全局简单变量**而不是同一全局数据结构，而且不是通过参数表传递该全局变量的信息，则称之为外部耦合。I/O将模块耦合到特定的设备、格式、通信协议上。
- **控制耦合**：如果一个模块通过**传送开关、标志、名字等控制信息**，**明显地控制选择另一模块的功能**，就是控制耦合。模块A实现了计算平均分和计算最高分的功能，而具体执行哪种操作由模块B传递的控制参数决定。**逻辑内聚的模块调用就是典型的控制耦合**。
- **标记耦合**：一组模块通过**参数表传递记录信息**，就是标记耦合。这个记录是某一数据结构的子结构，而不是简单变量。其实传递的是这个数据结构的地址。**模块A向模块B传递数组类型数据**
- **数据耦合**：一个模块访问另一个模块时，**彼此之间是通过简单数据参数**(不是控制参数、公共数据结构或外部变量)来**交换输入、输出信息**的。模块A实现两个数的加法操作，模块B实现数字的初始化，模块B将数字传递给模块A，由模块A完成加法。
- **非直接耦合**：两个模块之间**没有直接关系**，它们之间的联系完全是通过**主模块的控制和调用来实现的**。

一个好的模块应该是**高内聚、低耦合**的。

## 信息隐蔽

信息隐蔽：在设计和确定模块时，使得一个模块内包含信息(过程或数据)，对于不需要这些信息的其他模块来说,是不能访问的。在面向对象方法中，信息隐蔽是通过**对象的封装性**来实现的。

信息隐蔽的概念与模块的独立性直接相关

## 模块的独立性

- 模块独立性：
  - 模块独立性指每个模块**只完成系统要求的独立的子功能,并且与其他模块的联系最少且接口简单**
  - 模块独立性是指模块内部各部分及模块间的关系的一种衡量标准，由**内聚和耦合**来度量。
- 优点：
  - 具有独立的模块的软件**比较容易开发出来**。这是由于能够分割功能，而且接口可以简化，当许多人分工合作开发同一个软件时，这个优点尤其重要。
  - 独立的模块**比较容易测试和维护**。这是因为相对来说，修改设计和程序需要的工作量比较小，错误传播范围小，需要扩充功能时能够"插入"模块。总之，模块独立是优秀设计的关键，而设计又是决定软件质量的关键环节。

模块的独立程度可以由两个定性标准度量：**内聚和耦合**。

## 结构化程序设计方法

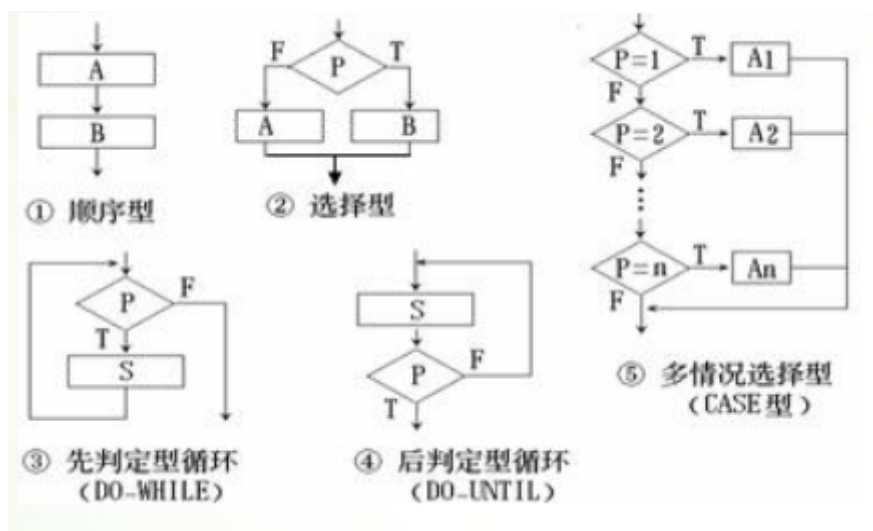
详细描述处理过程常用三种工具：图形、表格和语言。

- 图形：程序流程图、N-S图、PAD图
- 表格：判定表
- 语言：过程设计语言（PDL）

## 图形表示法

### 程序流程图

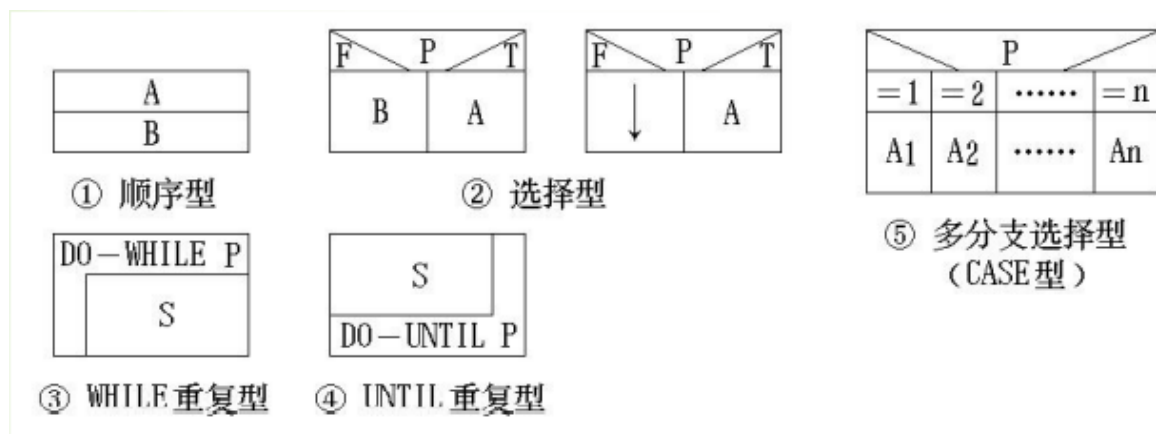
为使用流程图描述结构化程序，必须限制流程图只能使用五种基本控制结构。





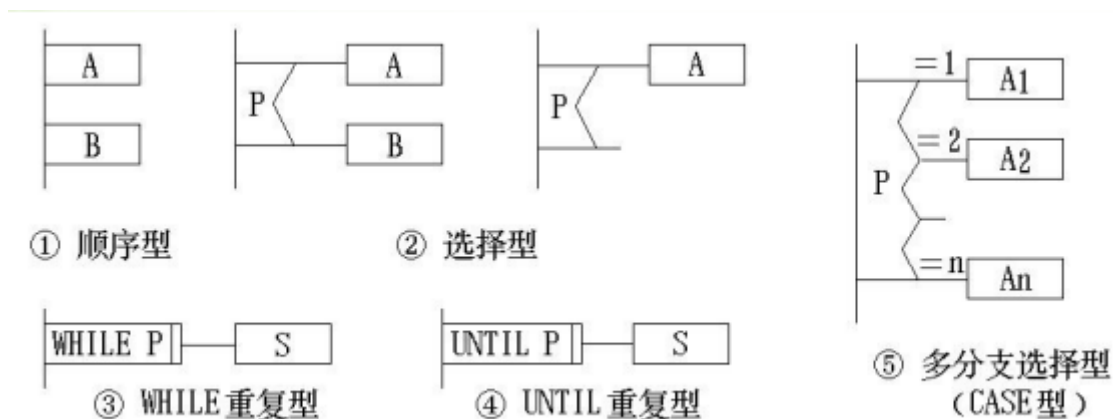
## N-S图

Nassi和Shneiderman 提出了一种符合结构化程序设计原则的图形描述工具，叫做**盒图**，也叫做N-S图。



## PAD图

PAD是Problem Analysis Diagram的缩写，由程序流程图演化而来。



程序流程图就是一般我们接触的图，N-S图形状为一个正方形，PAD图在一条线上。





## 结构化分析与设计

**结构化分析与设计方法**是一种面向数据流的传统软件开发方法。它以**数据流**为中心构建软件的分析模型和设计模型。

### 数据流图 DFD

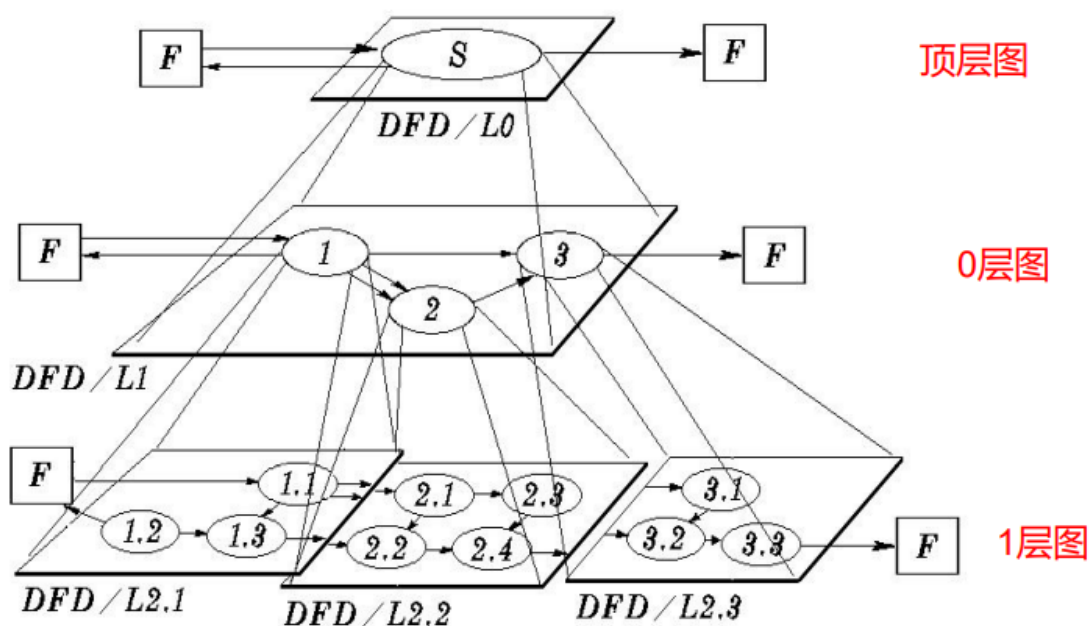
主要思想：数据流图描述输入数据流到输出数据流的变换（即加工），用于对系统的功能建模。PPT 05 考虑考前看一遍例题

数据流图的基本符号：

表 3-1 数据流图中的基本图形符号	
符号	说明
	圆或椭圆。表示加工,也称数据处理,它对数据流进行某些操作或变换。应在圆圈内写上加工名字(一般为动词短语),在分层数据流图中,加工还应编号
	矩形。表示数据流的源点和终点,是软件系统外部环境中的实体(包括人员、组织或其他软件系统),统称外部实体。应在方框内写上相应的名称
	双杠。表示文件或称数据存储,暂时保存数据,它可以是数据库文件或任何形式的数据组织。指向存储的数据可理解为写入,从存储引出的数据可理解为读出。要用名词或名词短语进行命名
	箭头。表示数据流,由一组成分固定的数据项组成。数据流必须有流向,箭头表示数据流动的方向,即在加工之间、加工与源点/终点之间、加工与数据存储之间流动。除与数据存储之间的数据流不用命名外,其他数据流均需用名词或名词短语命名

绘图步骤:

- 确定系统的外部项
  - 确定源和宿
- 画出顶层图 (顶层图一般没有文件, 只有一个加工)
- 自顶向下逐层分解 直到基本加工



## 分层的数据流图 (注意编号)

守恒及方法:

- 父图和子图平衡
  - 父图和子图平衡是指任何一张DFD 子图边界上的输入输出数据流必须与其父图中对应加工的输入输出数据流保持一致。
- 数据守恒
  - 一个加工所有输出数据流中的数据, 必须能从该加工的输入数据流中直接获得, 或者能通过该加工的处理而产生
  - 加工未使用其输入数据流中的某些数据项。这表明这些未用到的数据项是多余的, 可以从输入数据流中删去
  - 如果父图输出是B, 子图是B1、B2, 同时 $B=B1+B2$ 。这也是正确的。

- 局部文件
  - 考虑分层数据流中一个文件应画在哪些 DFD 中，而不该画在哪些 DFD 中

注意事项：

- 千万**不要**试图在数据流图中表现**分支条件或循环**，这样会造成混乱，画不出正确的数据流图。
- 检查分层细化时是否保持信息的连续性，即当把一个处理分解成一系列处理时，分解前和分解后的一系列输入输出数据流必须相同，这条规则也成为数据流图的数据平衡原则
- 两个加工可以直接用数据流相连，**两个源不能直接用数据流相连**
- 一个加工的输出数据流**不能**与该加工的输入数据流**同名**
- 每个加工至少有一个输入数据流和一个输出数据流
- 在整套分层数据流中，每个文件应至少有一个加工读该文件，有另一个加工写该文件
- 分层数据流图中得每个数据流和文件都必须命名（除了流入或流出文件的数据流），并且与数据字典一致

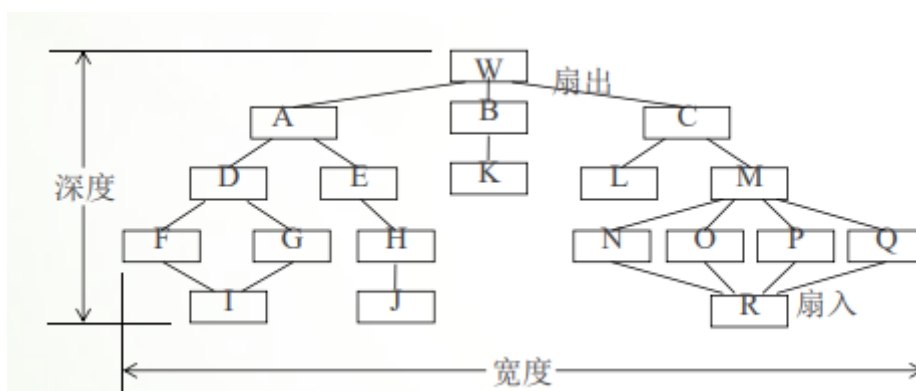
## 结构图

结构图的基本成分：

- 模块
  - 指具有一定功能的可以用模块名调用的一组程序语句，如函数、子程序等
- 调用
  - 用从一个模块指向另一个模块的箭头 来表示，其含义是前者调用了后者
- 数据
  - 模块调用时需传递的参数可通过在 调用箭头旁附加一个小箭头和数据名来表示

结构图的几个概念：

- 深度：模块结构图中控制的层数，例如图中所示的结构图的深度是5。
- 宽度：模块结构图中同一层次上模块总数的最大值，例如图中所示的结构图的宽度为7。
- 扇出(fan out)：该模块直接调用的模块数目。例如图中模块M的扇出是4，模块A的是2，模块B的扇出是1。（出度）
- 扇入(fan in)：能直接调用该模块的模块数目。例如图中模块G的扇入是1，模块I的扇入是2，模块R的扇入是4。（入度）



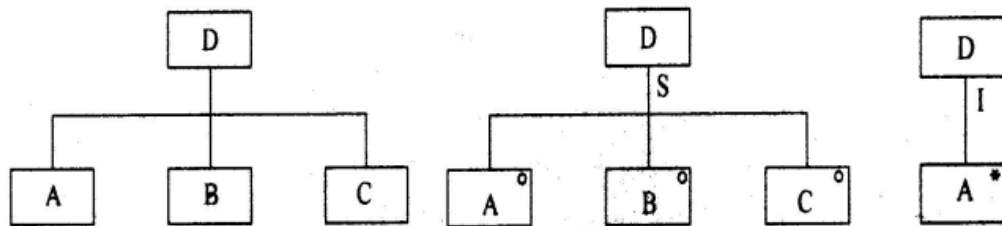
## 面向数据结构的分析与设计

面向数据结构的方法是以**数据结构为中心**，从输入/输出的数据结构导出程序结构的一种软件需求分析与设计的方法。

特点如下：

- 以信息对象及其操作作为核心进行需求分析
- 认为复合信息对象具有层次结构，并且可按顺序，选择，重复 3 种结构分解为成员对象信息
- 提供由层次信息结映射为程序结构的机制，从而为软件设计奠定良好的基础

## 6.2 Jackson 图的三种结构。



三种元素类型：顺序元素、选择元素、重复元素。

顺序元素：一个顺序元素由一个或多个从左到右的元素组成。

## 面向对象方法基础

UML是什么？

- UML 是一种 Language (语言)
- UML 是一种 Modeling (建模) Language
- UML 是 Unified (统一) Modeling Language
- UML 是一种统一的、标准化的建模语言，UML 是一种应用面很广泛的建模语言

我们学习过的图：

- 静态视图（类图）
- 用况视图（用况图）
- 状态机视图（状态机图）
- 活动视图（活动图）
- 交互视图（顺序图）

UML包含的四种事物：

- 结构事物：类、接口、用例、主动类、构件和结点等
- 动作（行为）事物：状态等
- 组织（分组）事物：包
- 注释事物：给建模者提供信息，提供关于任意信息的文本说明，但没有语义作用

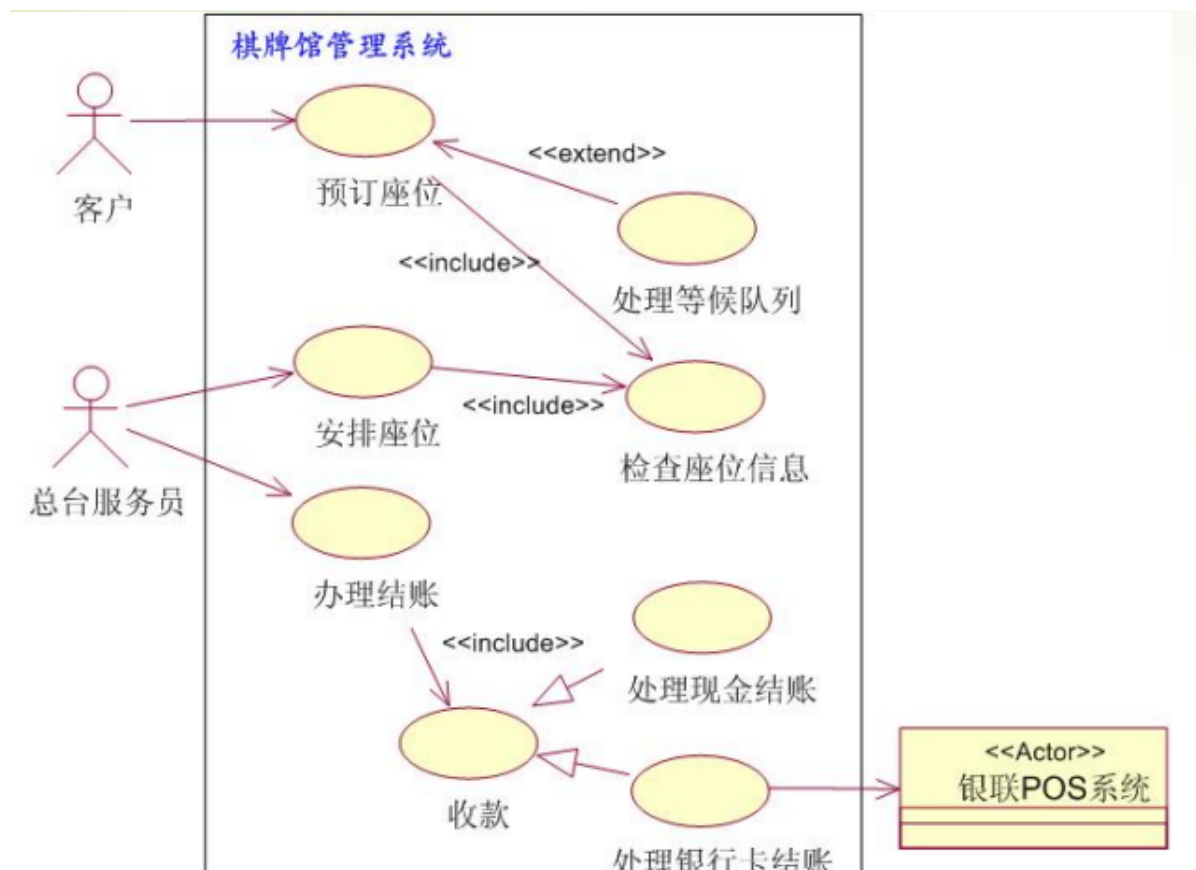
## 面向对象建模

### 用况图

用况建模是用于描述一个系统应该做什么的建模技术。用况图展示了各类**外部执行者**与**系统所提供的用况之间的连接**。一个用况是系统所提供的一个功能的描述。

- 描述动作者和使用案例之间的关系
- 用于表现系统根据需求所提供的功能

一个典型的用况图：



用况图的4种关系：

用况图中的 4 种关系：关联、包含、扩展、泛化

关系类型	说明	表示符号
关联	执行者与用况之间的关系	—————
包含	用况与用况之间的关系	-----<<包含>>----->
扩展	用况与用况之间的关系	-----<<扩展>>----->
泛化	执行者之间，或用况之间的关系	—————>

注意：

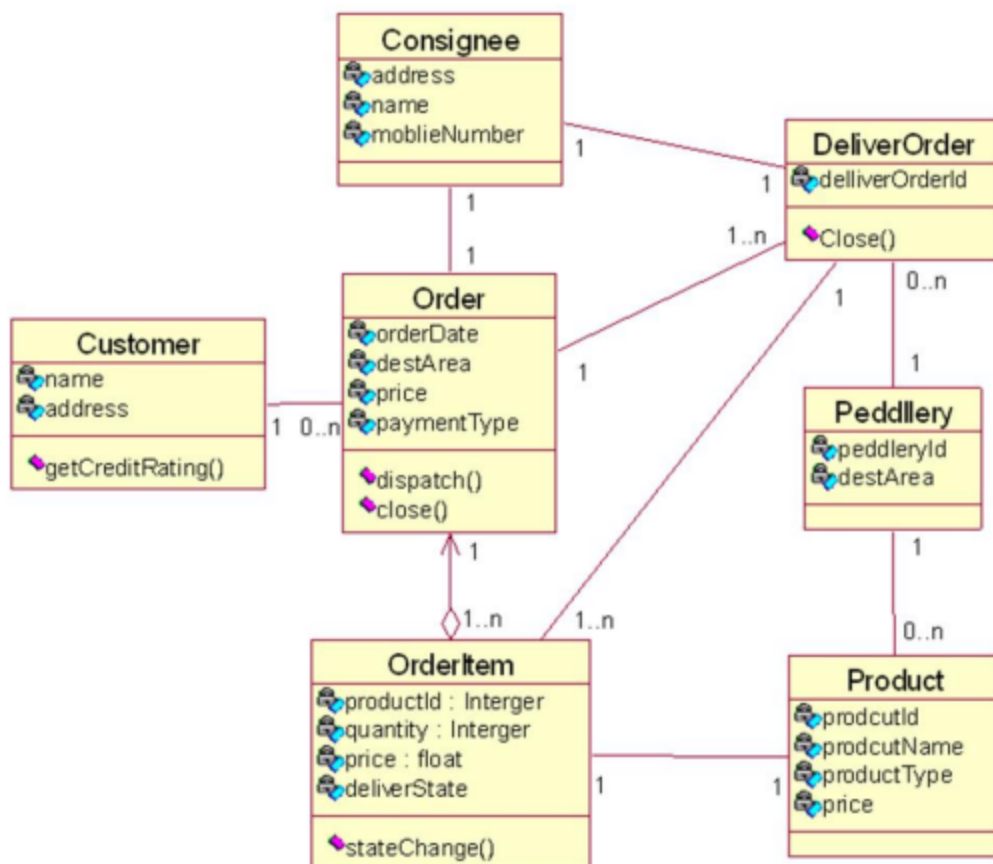
- 扩展是扩展的模块指向原模块（子指向父）
- 泛化（继承）：子类指向父类

## 类图

类和对象模型的基本模型元素有类、对象以及它们之间的关系。系统中的类和对象模型描述了系统的静态结构，在UML中用类图 and 对象图来表示。011类图题目如何做考虑复习。

类图由系统中使用的类以及它们之间的关系组成

一个典型的类图：



如何表示一个类图？

- **名称**：每个类都有一个唯一的名称，通常采用CamelCase（俗称驼峰格式，**每个单词的首字母都要大写**，其他均以小写字母出现）考点之一
- **属性（成员）**：是已被命名的类的特性，它描述该类实例中包含的信息
- **操作（方法）**：是类所提供的服务，它可以由类的任何对象请求以影响其行为。
- 属性名和操作名也通常采用CamelCase格式表示，只不过**首字母通常为小写**

类图的属性

类属性的语法为：

[可见性] 属性名 [: 类型] [=初始值] [{属性字符串}]



可见性表格

符号	种类	语义
+	Public	任何能看到这个类的类能够看到该属性
#	Protected	这个类或者它的任何子孙类都能看到该属性
-	Private	只有这个类能看到该属性
~	Package	在同一个包种的任何类能看到该属性

类图中类的重数

用来指出该属性可能的值的个数以及它们的排列次序和唯一性。值的个数写在方括号([ ])中，格式为 [minimum..maximum]。当重数缺省时，隐含表示重数为1。当值的个数是单一值（如值的个数是3）时，可写成 [3..3]或简写成[3]。

排列次序和唯一性写在 花括号({ })中，可使用的关键字如下表所示，其默认值是set，即无序且值元素唯一。

特征字符串

用来**明确地指明**该属性**可能的候选值**，如 {红，黄，绿} 指出该属 性可枚举的值只能是红、黄、绿。

类属性

表示被这个类的所有实例对象共享该属性的值（**static**）。类属性是这个类的名字空间中的全局变量。**类属性用下划线来指明。**

类之间的关系

关 系	功 能	符 号
关联	类实例间连接的描述	
依赖	二个模型元素之间的一种关系	
泛化	更特殊描述与更一般描述之间的一种关系，用于继承和多态性类型声明	
实现	规约（ <b>specification</b> ）与它的实现之间的关系	

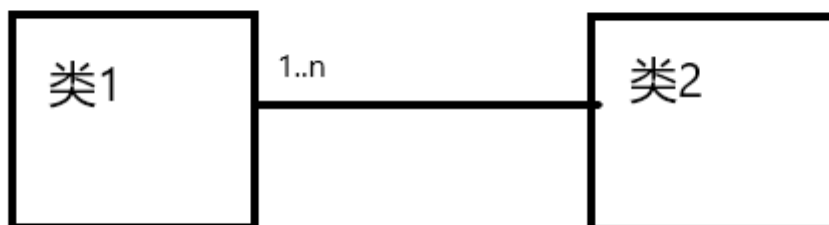
注意：

- 泛化和实现都是**子类指向父类，实现指向接口**

## 关联

关联描述了系统中对象之间或其他实例之间的连接。关联的种类主要有二元关联，多元关联，受限关联，聚集（aggregation）和组合（composition）。

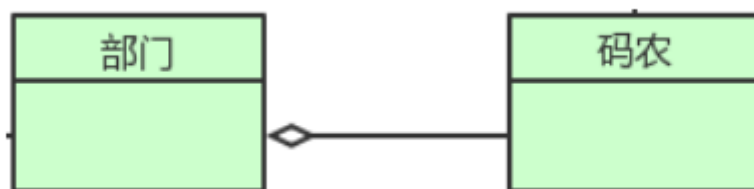
重数：



对于类2的1个实例，类1有1..n个实例与之对应。

### 聚集（聚合）

聚集是表示整体-部分关系的一种关联，它的“部分”对象可以是任意“整体”对象的一部分。公司和个人之间是聚集关系。公司没了，人还在（关联性较弱）。



### 组合

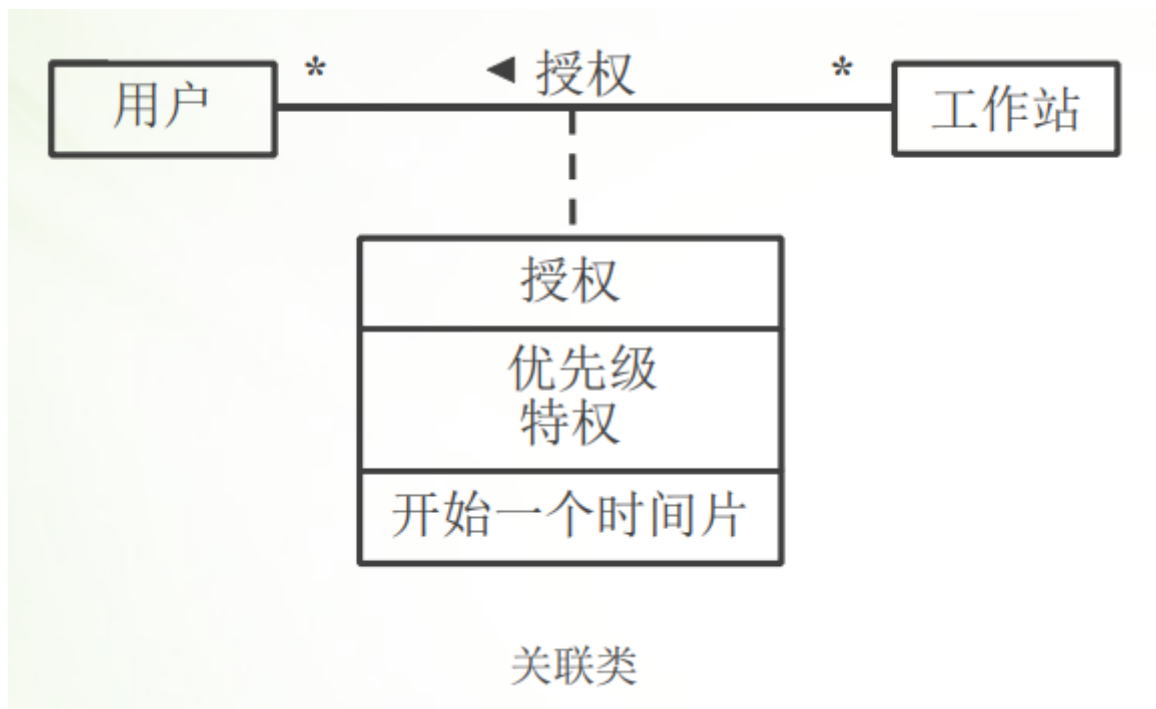
组合是一种**更强形式**的关联，代表整体的组合对象有管理它的部分对象的特有责任，如部分对象的分配和解除分配。组合关联具有强的物主身份，即“整体”对象拥有“部分”对象，**“部分”对象生存在“整体”对象中**。程序窗口和其中的按钮，程序没了按钮也就没了。





## 关联类

UML中可以把关联定义成类，该关联的每个链都是这个类的实例，使用虚线链接。



## 依赖

是一种使用的关系，即一个类的实现需要另一个类的协助。

可以简单的理解，就是一个类A使用到了另一个类B，而这种使用关系是具有偶然性的、临时性的、非常弱的，但是B类的变化会影响到A；比如某人要过河，需要借用一条船，此时人与船之间的关系就是依赖。

## 类图制作

- 筛选出系统外的概念，如“小王”、“人”
- 对**开发系统本身的名词**进行剔除，如“个人图书管理系统”
- 对于一些描述需求时使用的词进行剔除，如“功能”、“新书籍”、“信息”
- 找到明显重要的类，如“书籍”类。并对其中的属性成员与操作方法进行获取
- 寻找子类，并产生泛化关系，如“计算机类书籍”和“非计算机类书籍”
- 分析重数，如一名学生选择一名导师，一名导师可以有很多学生

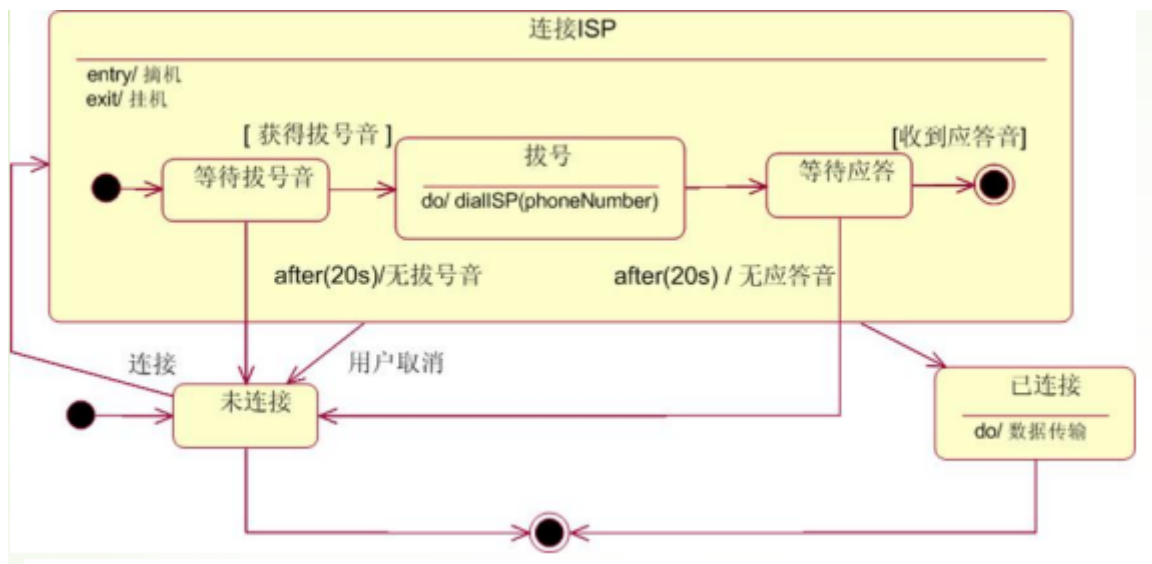
## 状态图

状态是指在对象生命周期中满足某些条件、执行某些活动或等待某些事件的一个条件和状况。

“人”就是一个类，而“你”、“我”、“张三”等都是“人”这个类的一个实例，“站着”、“躺着”等都是对象的一个状态。

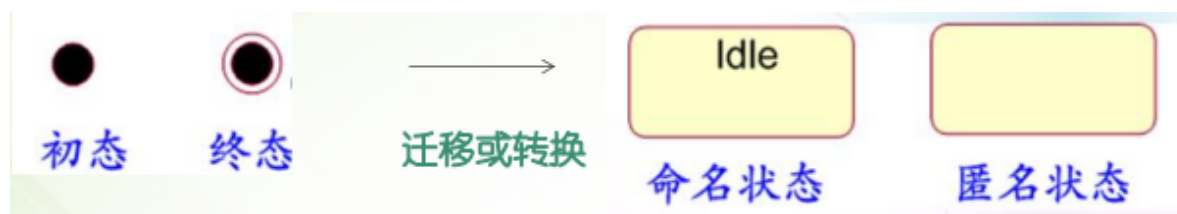
状态机图**描述从一个状态迁移到另一种状态的控制流程**。常用来对系统的动态特征进行建模。在大多数情况下，它用来对反应性对象（外部事件触发对象，对象接收到事件后产生响应）的行为建模。

一个典型的状态图：



## 状态机图具体定义

状态机图的基本符号：



转换的五要素：



- 源状态：即受转换影响的状态
- 目标状态：当转换完成后对象的状态
- 事件：用来为**转换定义一个事件**，包括调用、改变、信号、时间四类事件
- 监护条件：布尔表达式，**决定是否激活转换**
- 动作：**转换激活时的操作**

复杂转换：

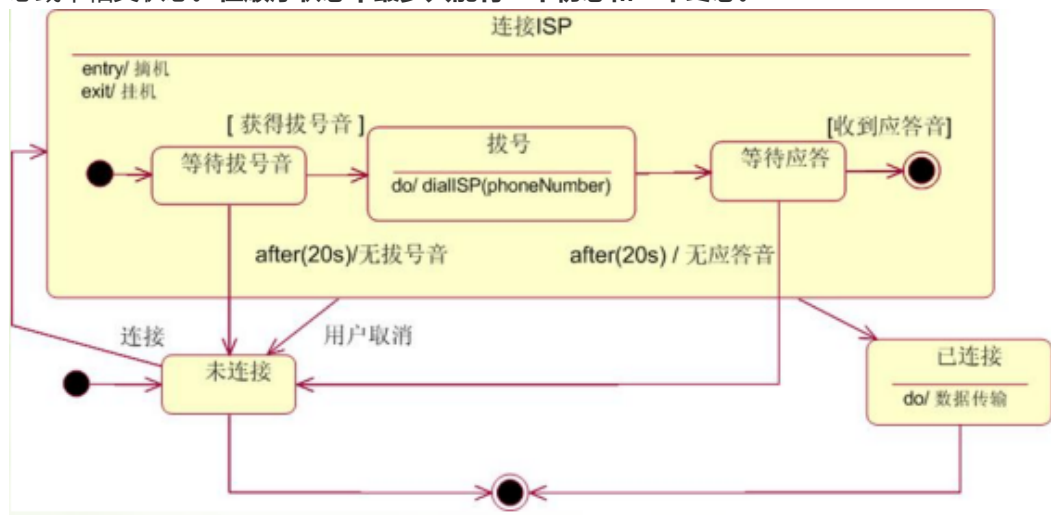
- 外部迁移：**改变对象状态**的迁移。状态和状态、初态（终态）与状态之间的箭头
- 内部迁移：**状态不变的情况下**，事件引发的动作
- 进入和退出转换：当进入一个状态时，执行某个动作；或当退出某个状态时，执行什么动作。这时就可以使用 进入和退出转换来表示。语法：entry/活动 exit/活动

状态分为简单状态和复合状态。**包含子状态**的状态称为**复合状态**。

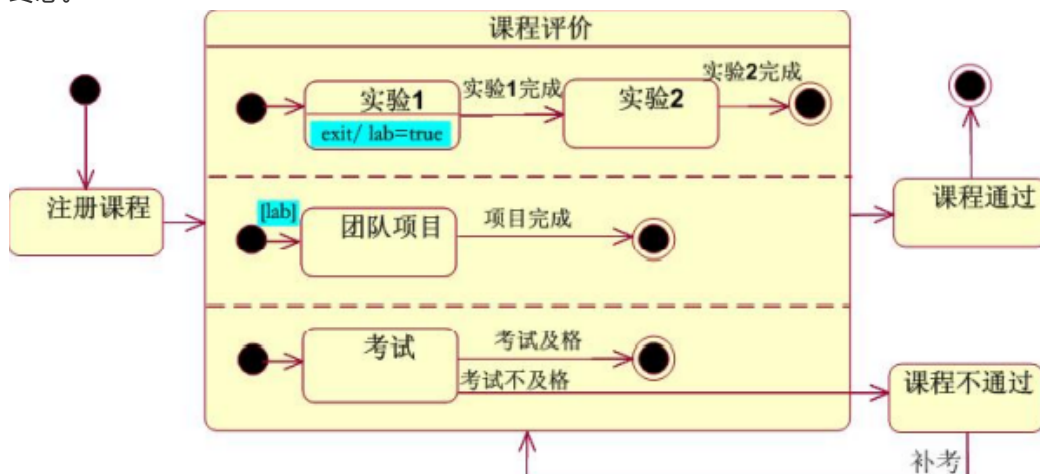
复合状态包含顺序状态、并发状态、历史状态。

- 顺序状态

- 如果一个组成状态的子状态对应的对象在其生命周期内的**任何时刻都只能处于一个子状态**，也就是说状态机图中**多个子状态是互斥的**，不能同时存在，这种子状态被称为顺序状态或互斥状态或不相交状态。**在顺序状态中最多只能有一个初态和一个终态。**



- 并发复合状态
  - 有时组成状态有两个或多个并发的子状态，此时称组成状态的子状态为并发子状态。如果并发子状态中有一个子状态机比其它并发子状态**先到达终态**，先到的将**等待**，直到所有子状态到达终态。



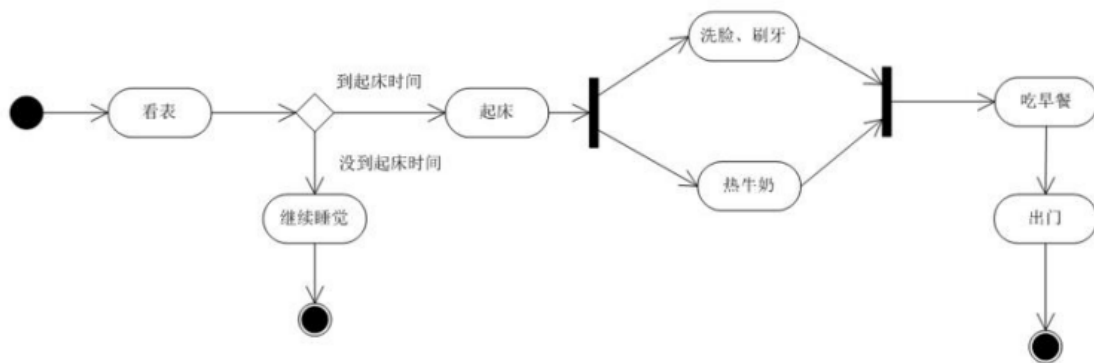
- 历史状态
  - 回到历史状态的状态时，其中的内容是不变的，仍然保留原来的信息。在状态右下方加入以下图标。



## 活动图

活动图描述的是从一个活动到另一个活动的控制流，描述活动的顺序，**活动表示处理事务的动作和状态。**

一个典型的活动图：



活动图分析：

**开始与结束：**活动图只能有一个起点，但可以有多多个终点的。

活动开始：●

活动结束：●

**分支判断：**菱形代表分支判断，这个与流程图的分支判断是一样的。上面的分支判断意思是：睡醒后看时间是否到上班时间，如果没到，继续睡觉；如果到了，起床洗漱。

**分叉与汇合：**分叉与汇合必须组合使用，表示并发动作。分叉表示一个活动完成后产生后续的多多个并行的活动；汇合表示多个活动全部完成后再进行下一个活动。分叉是一个指入多个指出，汇合是多个指入汇集后，一个指出。分叉与汇合间的活动是并行执行的，最后都执行完后统一汇合进入下一活动。

分叉：➤

汇合：➤

- **活动：**活动用圆角矩形表示，活动间的控制流用实体箭头表示。活动表示流程中的动作，活动可大可小，活动可以继续拆分成更细小的活动。

活动：

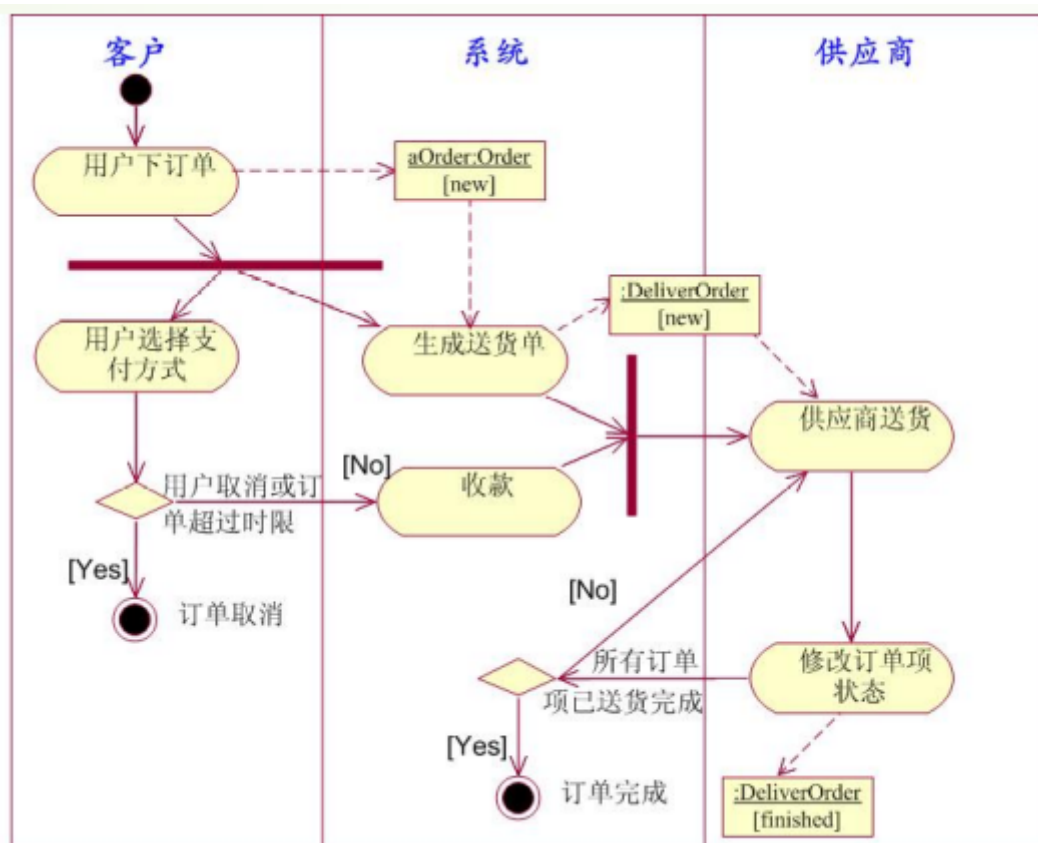


注意：

- 活动图只能有一个起点，但是可以有多个终点。
- 分叉和汇合必须组合使用，其主要目的是代表并发。只有并发出去的任务都完成了，汇合后才可进入下一活动
- 对于活动图的**文档性成果物**（市场需求文档），要用**长方形框**进行表示，为对象。对象可以作为下一个活动的输入，也可是活动的一个或多个输出。

## 泳道

各个对象的泳道为大的长方形，其主要作用就是更加细化区分每个活动是属于哪个对象的。



上图中的虚线部分是对象流，后面题目基本没有，应该不重要。

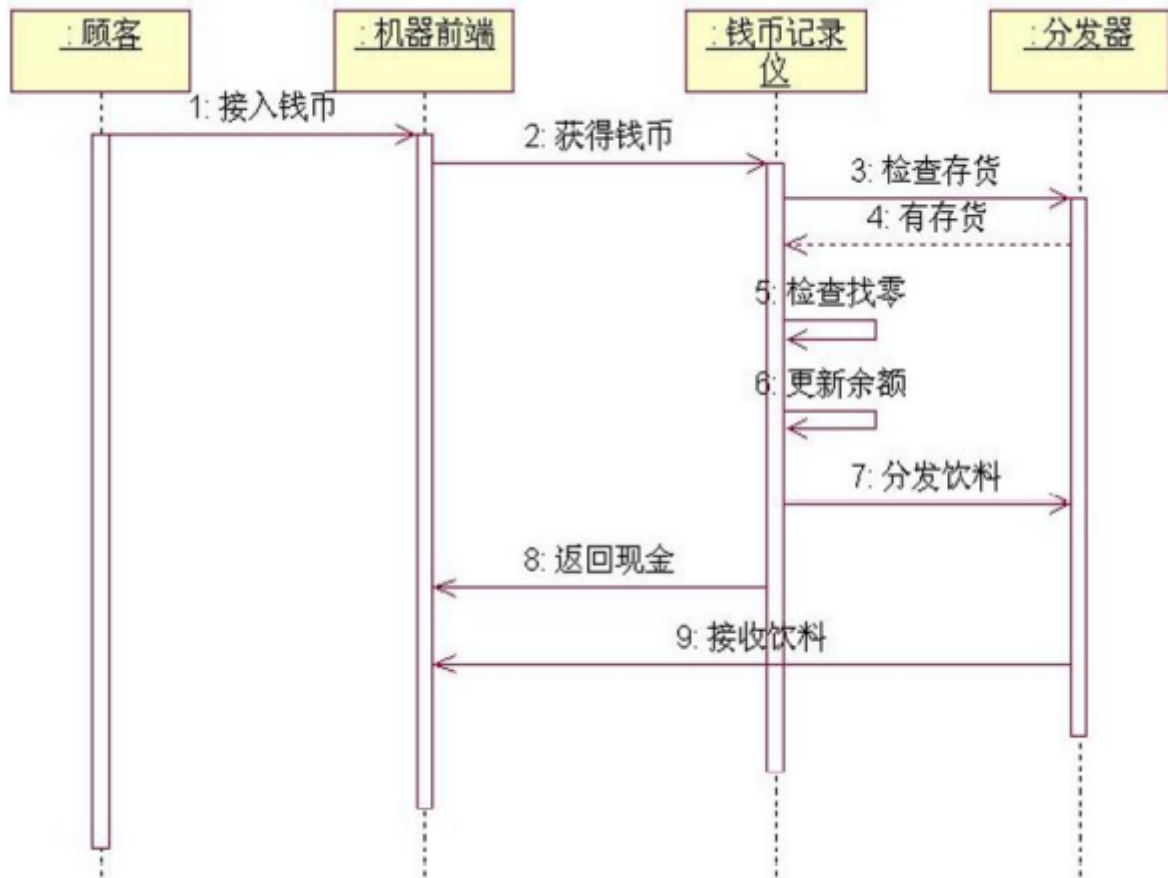
## 顺序图

顺序图也称为时序图、序列图，它描述了系统中的对象间通过消息进行的交互，**强调消息在时间轴上的先后顺序。**

顺序图常用来描述用例的实现，它表明了由哪些对象，通过消息相互协作来实现用例的功能，在顺序图中，标识了**消息发生交互的先后顺序。**

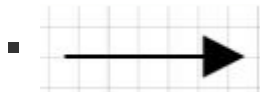
顺序图有两个坐标，垂直坐标表示时间（从上到下），水平坐标表示一组对象（用对象框表示）。

一个典型的顺序图：



## 顺序图的构成元素

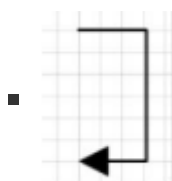
- 对象
  - 顺序图中的对象和对象图中的对象概念是一样的，都是类的实例。顺序图的对象可以是系统的参与者或者任何有效的系统对象
  - 对象之间的可发送消息
- 生命线
  - 生命线是一条垂直的虚线，表示顺序图中的对象在一段时间内的存在。时间从上到下流过
  - 生命线实际上显示了消息的顺序，在生命线之上的消息比在它之下的消息先发生
- 激活（控制焦点）
  - 在生命线上，当一个对象接收到一个消息时，该对象开始活动，称为激活
  - 激活画成对象生存线上的一个长方形框，是对对象操作的执行，它表示一个对象直接地或通过从属操作完成操作的过程。它对执行的持续时间和执行与调用者之间的控制关系进行建模。
- 消息
  - **同步消息**：表示发送对象必须等待接收对象完成消息的处理后才能继续执行



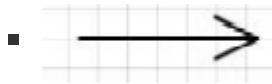
- **反身消息**：是消息的发送方和接收方是同一个对象



- **异步消息**：表示发送对象在消息发送后立即继续执行，而不必等待接收对象的返回



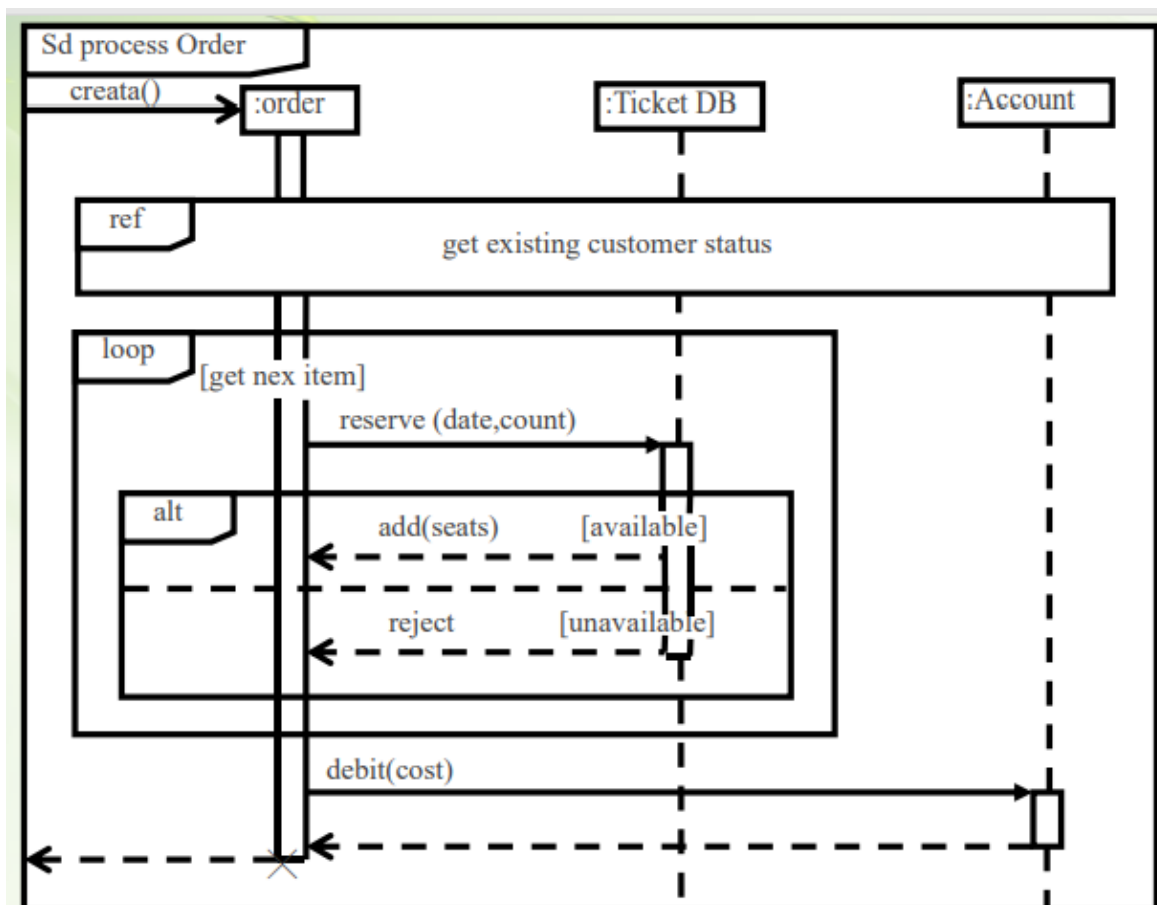
- **定时消息**：对消息附加时间约束条件，包括：发送时间、接收时间、已用时间等



## 结构化控制结构

前面的顺序图中描述的都是顺序的控制流，对于复杂的控制流可以用**组合片段** (combined fragment) 来表示。

关键字	含义
ref (引用)	对另一交互的引用
loop (循环)	它有一个子片段，当循环的警戒条件为真时执行子片段
alt (选择)	它有两个或多个子片段，当某个片段警戒条件为真时执行
opt (任选)	它是带单个子片段的特殊情况，即警戒条件为假时省略该子片段
par (并发)	它有两个或多个子片段，处于此片段时，所有子片段都并发地执行



实例6：带组合片段的顺序图

## 敏捷软件开发

- 什么是敏捷开发方法？

- 敏捷开发是一种基于更紧密的团队协作、能够有效应对快速变化需求、快速交付高质量软件的迭代和增量的新型软件开发方法。
- 有哪些有代表性的敏捷开发方法？
  - 以 Scrum 为基础的方法论（包括 Scrum、Scrum/XP 混合等）体系仍然居于主流地位，使用率最高。其他还有：看板方法、精益创业、极限编程等

—— JerryKim