

软件工程期中复习

版本：2023-11-23

最近更新日期：2023-11-23

考试题型与提纲

题型：

1. 简答题（50'，包含绘制UML图以及其他概念）
2. 设计题（40'，给定一段文字进行场景设计和画图）
3. 论述题（10'）

说明：

★为重点。如果加粗小点，则该小点为重点，如果加粗大点，则该部分都重要；
加粗为提炼

第一部分 软件过程

1. 软件

- 定义★：软件是能够**完成预定功能和性能**，并对**相应数据进行加工**的程序和**描述数据及其操作**的文档。
- 组成：软件=程序+文本+数据；程序=算法+数据结构
- 软件和硬件具有完全不同的特性★：**软件不会“磨损”**，但是**软件退化**的确存在。
- 软件维护要应对变更请求，比硬件维护更为复杂。

2. 软件工程

- 定义★：软件工程是**指导计算机软件开发和维护**的工程学科。它将**系统化的、规范的、可量化的方法应用于软件的开发、运行和维护**，即将**工程化方法应用于软件**。
- 出现原因：软件危机
- 目的：
 - **克服软件危机**
 - 用工程化的方法和规范来进行软件的开发，以**提高软件开发的成功率**
 - 作好软件开发的**培训工作**
 - 以**较低的成本开发出高质量的软件**

3. 软件工程方法学

- 定义：通常把**软件生命周期过程中使用的一整套技术**的集合称为软件工程方法学，也称为范型。
- 组成（三要素）：方法(如何做)、工具(支撑平台)和过程(工作步骤)。

4. 软件设计思路（3种）★：

- (1) 用户故事 --> 确定类（通过找出名词以及两轮筛选） --> 确定属性和操作（CRC卡，半头脑风暴）

(2) 活动图 --> 用例图 --> 用例描述 --> 将用例描述中的名词标出 --> ER图 --> 类图

(3) UI设计 --> ER图 --> 接口

5. 软件危机

- 定义：软件危机是指是指**落后的软件生产方式无法满足迅速增长的计算机软件需求**，从而导致软件开发与维护过程中出现一系列严重问题的现象。
- 表现：
 - 软件**开发成本**和**进度难于估计**
 - 用户对“已完成”的软件**不满意**
 - 软件**质量不可靠**
 - 软件的可**维护性差**
 - 软件**文档资料不足**
 - **软件成本**在系统总成本中所占的比例逐年**上升**
 - 软件**开发生产率**提高的速度**跟不上经济发展的速度**和**计算机应用迅速普及的速度**
- 产生原因：
 - 软件是逻辑产品而不是物理产品，进度和质量难于评价，开发过程难于管理和控制；
 - 软件规模过于庞大，程序的复杂性随程序规模的增长而呈指数增长；
 - 开发过程中或多或少地采用了错误的技术和方法(如忽视需求分析、认为开发软件就是写程序、轻视软件维护等)。

6. 软件工程生命周期★

- 定义：软件的生命周期(Software Life cycle Model)：软件的**产生直到报废或停止使用**的生命周期。
- 阶段：
 - 软件定义
 - 问题定义阶段：需要解决的问题是什么
 - 可行性研究阶段：上个阶段所确定的问题是否有行得通的解决办法
 - 需求分析阶段：确定系统必须完成哪些工作
 - 软件开发
 - 软件设计
 - 概要设计阶段：怎样实现目标系统
 - 详细设计阶段：应该怎样具体地实现这个系统
 - 编码和单元测试阶段：写出正确的容易理解、容易维护的程序模块
 - 综合测试阶段：通过各种类型的测试（及相应的调试）使软件达到预定要求
 - 软件运行维护阶段：通过各种必要维护活动使得系统持久地满足用户需求

7. 软件工程过程

- 概念★：软件开发中所**遵循的路线图**。在开发产品或构建系统时，遵循一系列可预测的步骤（即路线图）是非常重要的，它**有助于及时交付高质量的产品**。
- 重要性：软件过程提高了软件工程活动的稳定性、可控性和有组织性
- 通用过程模型的五种框架活动★：**沟通、策划、建模、构建、部署**
- 常见惯用过程模型★：**瀑布模型、原型模型、螺旋模型、增量过程模型、统一过程模型**
- 瀑布模型（普通瀑布模型，V模型（是瀑布模型的改进，在开发的同时引入测试））★
 - 应用：**适用于功能、性能明确、完整、无重大变化的软件系统**的开发。例如操作系统、编译系统、数据库管理系统等系统软件的开发。应用有一定的局限性。
 - 前提：需求必须是**准确定义和相对稳定的**。

○ 演化过程模型★

- 说明：**迭代的过程模型**，每次迭代产生软件的一个更完整的版本。
- 原型模型
 - 定义：原型是预期系统的一个可执行版本，反映了系统性质的一个选定的子集。一个原型**不必满足目标软件的所有约束**，其目的是**能快速、低成本地构建原型**。
 - 应用：适合于**用户需求不清、需求经常变化、系统规模不是很大也不太复杂**的情况。
- 螺旋模型
 - 定义：**演进式，风险驱动**的软件开发模型。结合了原型开发的迭代性质和瀑布模型的可控性和系统性特点。螺旋模型将开发过程分为几个螺旋周期，每个螺旋周期大致和瀑布模型相符合。
 - 应用：适用于**庞大、复杂并且具有高风险**的系统。
- 增量过程模型
 - 定义：增量模型融合了瀑布模型的基本成分和原型实现的迭代特征，该模型**采用随着日程时间的进展而交错的线性序列**，每一个线性序列产生软件的一个可运行的“增量”。早期的增量可以看做是最终产品的片段版本。

○ 统一过程模型★

- 特点：以**用例和风险驱动**，以**架构为核心**，**迭代并且增量**；由UML方法和工具支持，广泛应用于各类**面向对象项目**。
- 主要阶段：①**起始阶段**②**细化阶段**③**构建阶段**④**转换阶段**⑤**生产阶段**

○ 示例：

软件类型与要求	适合的开发过程
客户不太清楚待开发的系统需要提供什么服务	原型
开发团队了解待开发软件的相关领域知识,尽管此系统庞大，但其较已经开发的系统差异并不大。	瀑布
软件的功能是把读入的浮点致开平方，所得到的结果应该精确到小数点后4位。	瀑布
开发一个已发布软件的新版本，公司规定了严格的完成期限，并对外公布。	增量
汽车防抱死刹车控制系统	螺旋
大学记账系统，准备替换一个已存在的系统	瀑布
一个位于火车站的交互式火车车次查询系统	原型

8. 敏捷开发★

- 说明：因为软件危机的出现，所以有的敏捷开发；通常采用**增量模型和原型模型**
- 定义：是一种软件开发方法论，可以**应对客户快速变更的需求**。它强调**以人为核心**，采用**迭代**的方式，**循序渐进**的开发软件。
- 敏捷开发宣言
 - **个体与交互胜过过程与工具**
 - **可用的软件胜过完备的文档**

- **客户协作胜过合同谈判**
 - **响应变化胜过遵循计划**
 - Scrum方法（并列争球法）
 - 定义：并列争球法使用**迭代**的方法，按**需求的优先级别来实现产品**。多个自组织和自治的小组**并行地递增实现产品**。协调是通过简短的日常情况会议来进行，就像橄榄球中的“并列争球”。
 - 冲刺：工作任务在相对较短的时间盒的期限内完成（每30天迭代一次或**4-6周一次**），**冲刺中进行的工作适应当前的问题**，由Scrum团队规定并进行实时修改。
 - XP方法（极限编程法）
 - 说明：使用得**最为广泛**的敏捷过程
 - 定义：XP是一种轻量级（敏捷）、高效、低风险、柔性、可预测的、科学的软件开发方式。它由**价值观、原则、实践和行为**4个部分组成，彼此相互依赖、关联，并通过行为贯穿于整个生存周期。
 - 时间：2-4周一次
-

第二部分 建模

1. 需求工程

- 定义：致力于不断**理解需求的大量任务和技术**
- 步骤
 - 根据利益相关者**制定访谈计划**
 - **实施并记录访谈内容**
 - 分析并**进一步需求诱导**
 - 利用Axure等原型工具开发原型，进行**需求迭代与确认**
- 任务：起始，获取，细化，协商，规格说明，确认和管理

2. 需求建模

- 方法
 - 面向**数据流**的分析方法
 - 面向**数据（过程）**的分析方法
 - 面向**对象**的分析方法

3. 需求分析

- 产生软件工作特征的**规格说明**
- 指明软件和其他系统元素的**接口**
- 规定软件必须满足的**约束**

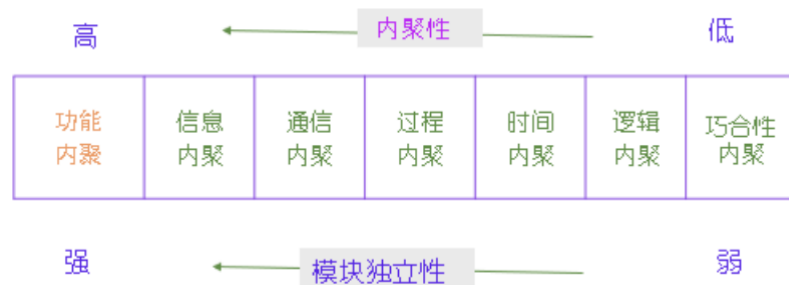
4. 设计建模原则：

- 设计应可追溯到需求模型。
- 始终考虑要构建系统的体系结构。
- 数据设计与处理功能设计同等重要。
- 接口（内部和外部）的设计必须谨慎。
- 用户界面设计应适应最终用户的需求。
- 构件级设计应在功能上独立。
- 构件应彼此松耦合，并应与外部环境松耦合。
- 设计表示（模型）应易于理解。
- 设计应迭代式开发。

- 设计模型的创建并不排除采用敏捷方法的可能性。

5. 软件工程设计

- 内容
 - 数据/类设计**——将分析类模型变换为软件实现类模型及数据结构
 - 体系结构设计**——描述软件主要构造元素之间的关系
 - 接口设计**——描述软件元素、硬件元素和终端用户间如何通信
 - 构件级设计**——将软件体系结构的构造元素变换为对软件构件的过程性描述
- 模块化★
 - 定义：模块化是软件的**单一属性**，它使程序能被**智能化地管理**。
 - 原因
 - 模块化是为了使一个复杂的大型程序能被人的智力所管理，软件应该具备的惟一属性。
 - 如果一个大型程序仅由一个模块组成，它将很难被人所理解。
- 衡量指标★：**内聚性和耦合性**
- 内聚性



- 定义★：标志一个模块内各个元素彼此结合的**紧密程度**。
- 分类

类型	定义	评价	解决方案
偶然/巧合性内聚	彼此间即使有关系， 关系 也是很 松散 的	①模块内各元素之间没有实质性联系②可理解性差，可维护性产生退化③模块是不可重用的	模块分解
逻辑内聚	逻辑 上属于 相同或相似 的一类	①接口难以理解，造成整体上不易理解②完成多个操作的代码互相纠缠在一起，即使局部功能的修改有时也会影响全局，导致严重的维护问题③难以重用	模块分解
时间内聚	一个模块包含的任务必须在 同一段时间内执行	①模块内操作之间的关系很弱，与其他模块的操作却有很强的关联②时间内聚的模块不太可能重用	

类型	定义	评价	解决方案
过程内聚	模块内的处理元素是相关的，而且必须以 特定次序执行 ，即使两者之间没有数据进行传递	①比时间内聚好，至少操作之间是过程关联的②仍是弱连接，不太可能重用模块	分割为单独的模块，每个模块执行一个操作
通信内聚	在同一个 数据结构 上操作	①模块中各操作紧密相连，比过程内聚更好②不能重用	分成多个模块，每个模块执行一个操作
顺序/信息内聚	如果一个模块内的各个 成分和同一个功能密切相关 ，而且这些处理必须 顺序执行 ，一个成分的输出作为另一个成分的输入	根据数据流图划分模块时，通常得到顺序内聚的模块，这种模块彼此间的连接往往比较简单	
功能内聚	如果模块内所有处理元素属于一个 整体 ，完成一个单一的功能	①模块可重用，应尽可能重用②可隔离错误，维护更容易③扩充产品功能时更容易	

- 说明：对于内聚，禁止使用偶然性和逻辑性内聚，限制使用时间性内聚，少用过程性内聚和通信性内聚，提倡使用顺序内聚和功能内聚

○ 耦合性



- 定义★：耦合性是程序结构中各个模块之间**相互依赖**的度量，它取决于各个模块之间**接口的复杂程度、调用模块的方式以及哪些信息通过接口**。
- 分类：非直接耦合/完全独立、数据耦合、控制耦合、公共环境耦合、内容耦合

分类	定义	评价
----	----	----

分类	定义	评价
非直接耦合/ 完全独立	两个模块中的每一个都能 独立地工作 而不需要另一个模块的存在，那么它们完全独立。	在一个软件系统中不可能所有模块之间都没有任何连接。
数据耦合	两个模块彼此间 通过参数交换信息 ，而且交换的信息仅仅是 数据	①系统中至少必须存在这种耦合。一般说来，一个系统内可以只包含数据耦合②数据耦合是理想的目标。维护更容易，对一个模块的修改不会是另一个模块产生退化错误
标记耦合/ 数据结构耦合/ 特征耦合	几个模块间 数据结构作为参数 进行传递， 共享数据结构	被调用的模块可使用的数据多于它确实需要的数据，这将导致对数据的访问失去控制，从而给计算机犯罪提供了机会
控制耦合	两个模块彼此间传递的信息中有 控制信息	①控制耦合往往是多余的，把模块适当分解之后通常可以用数据耦合代替它②被调用的模块需知道调用模块的内部结构和逻辑，降低了重用的可能性
公共环境耦合	两个或多个模块通过一个 公共数据环境 相互作用；公共环境可以是全程变量、共享的通信区、内存的公共覆盖区、任何存储介质上的文件、物理设备等等	①一旦公共数据有变化，与之有关的模块都应随之而修改,增加了维护的工作量及难度②潜在危险很大。模块暴露出必需要更多的数据，难以控制数据存取，而且会导致计算机犯罪

分类	定义	评价
内容耦合	最高程度 的耦合是内容耦合	发生情况：①一个模块访问另一个模块的内部数据②一个模块不通过正常入口转到另一个模块的内部③两个模块有一部分程序代码重叠④一个模块有多个入口

- 说明：对于耦合，尽量使用非直接耦合、数据耦合和特征耦合，少用控制耦合和外部耦合限制使用公共耦合，完全不用内容耦合
- 三个基本类：
 - 实体类：**分析类**在设计中会被精化为实体类
 - 边界类：在设计中被用于创建用户在实用软件时所观看并交互的**接口**
 - 控制类：用于**管理**
 - 实体对象的创建或更新；
 - 从实体对象获得信息后的边界对象实例化；
 - 各对象间的复杂交流；
 - 对象间或用户与应用间数据交流的确定性
- 特征★：①完整性②充分性③原始性④高内聚性⑤低耦合性

6. 体系结构设计

- 定义：体系结构并非可运行的软件。确切地说，它是一种表达，使能够：
 - 在满足既定的需求方面下，**分析设计有效性**；
 - 在设计变更相对容易的阶段，**考虑体系结构可能的选择方案**；
 - **降低**与软件构造相关的**风险**
- 体系结构风格
 - 定义
 - 完成系统需要的某种功能的一组**构件**（例如，数据库、计算模块）；
 - 能使构件间实现“通信、合作和协调”的一组**连接件**；
 - 定义构件如何集成为系统的**约束**；
 - 语义模型，能使设计者通过**分析系统组成成分的已知属性**来理解系统的整体性质。
 - 分类★
 - 以数据为中心的体系结构
 - 数据流体系结构
 - 调用和返回体系结构
 - 面向对象体系结构
 - 层次体系结构

7. 构件级设计

- 构件★
 - 构件是计算机软件中的一个**模块化的构造块**。统中**模块化的、可部署的和可替换的部件**，该部件**封装**了实现并对外提供一组**接口**。
 - 构件存在于软件体系结构中，因而构件在完成所建系统的**需求和目标**的过程中**起着重要作用**。
- 设计基于类的构件的基本设计原则
 - 开闭原则
 - 里氏替换原则

- 依赖倒置原则
- 接口分离原则
- 发布/复用等价性原则
- 共同封装原则
- 共同复用原则

8. 用户体验设计

- 步骤：确定任务、屏幕布局、开发原型、结果评估
- 黄金规则★
 - 把**控制权交给用户**
 - **减轻用户的记忆负担**
 - 保持**界面一致**

9. 软件质量★

- 功能性因素
 - 正确性：软件按照需求**正确执行任务**的能力
 - 健壮性（鲁棒性）：异常情况下，软件能够**正常运行**的能力
 - 可靠性：在一定环境下，在给定时间内，系统**不发生故障**的概率
- 非功能性因素
 - 性能：时间、空间复杂度
 - 兼容性：不同产品（或者新老产品）**相互交换信息**的能力
 - 安全性：**信息安全**，防止系统被非法入侵的能力
 - 可移植性：软件不经修改或稍加修改就**可以运行到不同软硬件环境**的能力
 - 易用性：用户**使用软件的容易程度**
 - 清晰性：所有**工作成果易读、易理解**，可以提高团队开发效率，降低维护代价
 - 可扩展性：反应软件**适应“变化”**的能力

第三部分 UML图★

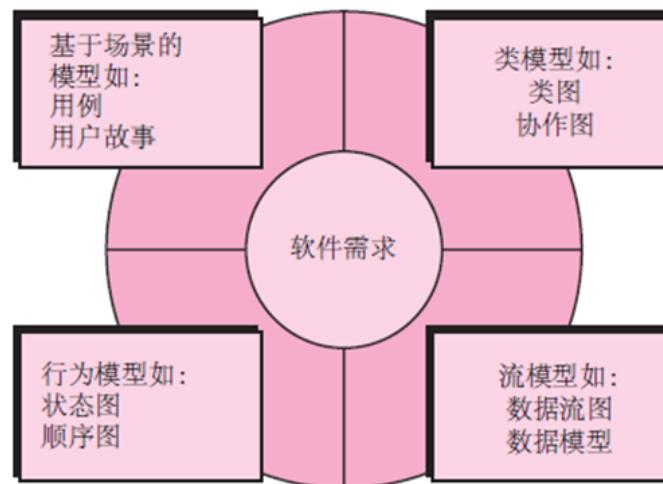
1. 分类与要求

分类	要求与注意事项
用例图	基本元素正确，内容基本达标；关联箭头可有可无；主执行者在左，辅执行者在右
类图	类与类之间的关系正确，基本元素正确；关联建议带箭头；依赖是关联的一种形式；信息的主要来源于文字描述和用例描述
顺序图 (时序图)	与协作图关系紧密，可相互转换
协作图 (通信图)	基本元素正确
状态图	基本元素正确

分类	要求与注意事项
活动图&泳道图	一定有终点；分支内不写条件而是写在迁移上
构件图	基本元素正确
部署图	非重点
ER图	基本元素正确
数据流图	基本元素正确

2. UML图介绍

○ 分类



- 场景模型：出自各种系统“参与者”观点的需求
- 数据模型：描述问题信息域模型
- 面向类的模型：表示面向对象类（属性和操作）的模型
- 面向流程的模型：描述功能元素在系统中运行时怎样进行数据变换
- 行为模型：描述如何将软件行为看作是外部“事件”的后续

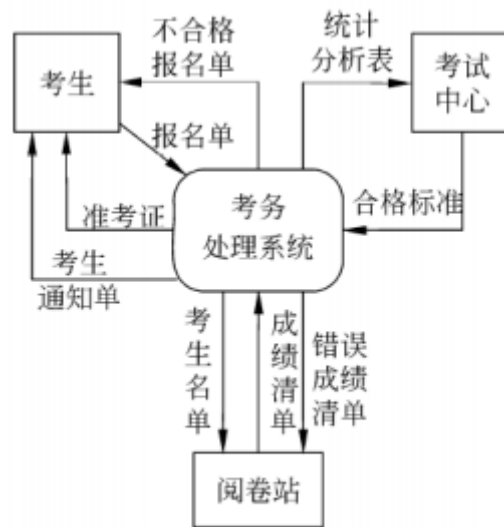
○ 元素

类	是对一组具有相同属性、相同操作、相同关系和相同语义的对象的描述	
对象		
接口	是描述了一个类或构件的一个服务的操作集	
协作	定义了一个交互，它是由一组共同工作以提供某种协作行为的角色和其他元素构成的一个群体	
用例	是对一组动作序列的描述	
主动类	对象至少拥有一个进程或线程的类	
构件	是系统中物理的、可替代的部件	
参与者	在系统外部与系统直接交互的人或事物	

节点	是在运行时存在的物理元素	
交互	它由在特定语境中共同完成一定任务的一组对象间交换的消息组成	
状态机	它描述了一个对象或一个交互在生命期内响应事件所经历的状态序列	
包	把元素组织成组的机制	
注释事物	是UML模型的解释部分	
依赖	一条可能有方向的虚线	
关联	一条实线，可能有方向	
泛化	一条带有空心箭头的实线	
实现	一条带有空心箭头的虚线	

○ 数据流图

- 示例



(a) 顶层图

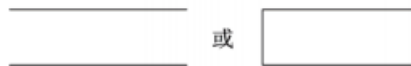
■ 图形元素



(a) 外部实体 (External Agent)



(b) 加工 (Process)

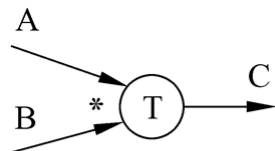


(c) 数据存储 (Data Store)

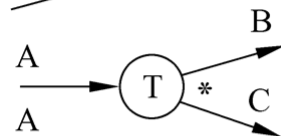


(d) 数据流 (Data Flow)

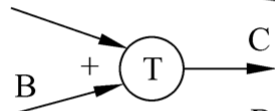
■ 符号



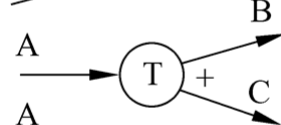
数据 A 和 B 同时输入才能变换成数据 C



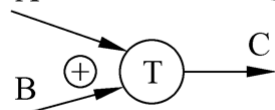
数据 A 变换成 B 和 C



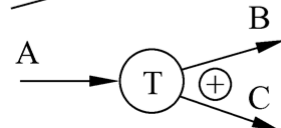
数据 A 或 B, 或 A 和 B 同时输入变换成 C



数据 A 变换成 B 或 C, 或 B 和 C



只有数据 A 或只有数据 B(但不能 A、B 同时) 输入时变换成 C



数据 A 变换成 B 或 C, 但不能变换成 B 和 C

■ 要求

- 一个加工可以有多个输入数据流和多个输出数据流, 但至少有一个输入数据流和一个输出数据流。
- DFD中的每个数据流用一个定义明确的名字表示。除了流向数据存储或从数据存储流出的数据流不必命名外, 每个数据流都必须有一个合适的名字, 以反映该数据流

的含义。

- 数据流的流向

- 从一个加工流向另一个加工
- 从加工流向数据存储（写）
- 从数据存储流向加工（读）
- 从外部实体流向加工（输入）
- 从加工流向外部实体（输出）

- 常见错误

- 黑洞：加工有输入无输出
- 白洞：加工有输出无输入
- 灰洞：加工的输入不足以产生输出

- 数据字典

- 定义：对数据的**数据项、数据结构、数据流、数据存储、处理逻辑**等进行定义和描述，是对系统中使用的所有数据元素的定义的集合。

- 符号

表 6-1 在数据字典的定义式中出现的符号

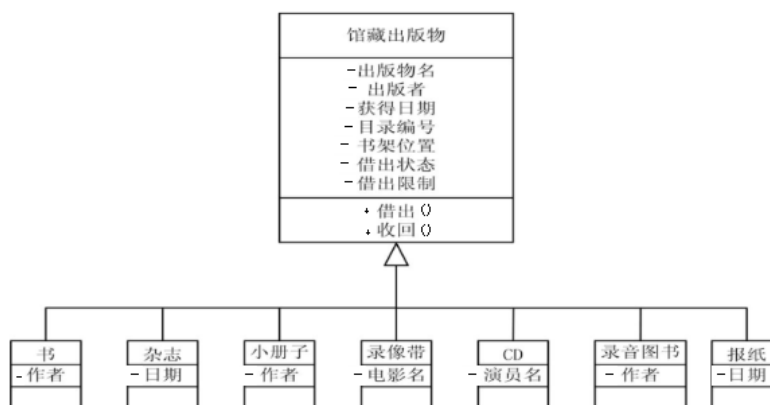
符 号	含 义	举例及说明
=	被定义为	
+	与	$x = a + b$ ，表示 x 由 a 和 b 组成
[... ...]	或	$x = [a b]$ ，表示 x 由 a 或 b 组成
{...}	重复	$x = \{a\}$ ，表示 x 由 0 个或多个 a 组成
$m\{\dots\}n$ 或 $\{\dots\}_m^n$	重复	$x = 2\{a\}5$ 或 $x = \{a\}_2^5$ ，表示 x 中最少出现 2 次 a ，最多出现 5 次 a 。5、2 为重复次数的上、下限
(...)	可选	$x = (a)$ 表示 a 可在 x 中出现，也可不出现
"..."	基本数据元素	$x = "a"$ ，表示 x 是取值为字符 a 的数据元素
..	连接符	$x = 1..9$ ，表示 x 可取 1~9 中的任意一个值

- 示例

- 电话号码=[分机号|外线号码]
- 分机号=7201...7299
- 外线号码=9+[市话号码|长话号码]
- 长话号码=区号+市话号码
- 区号=100...300
- 市话号码=局号+分局号
- 局号=[455|466|888|552]
- 分局号=4{0...9}4

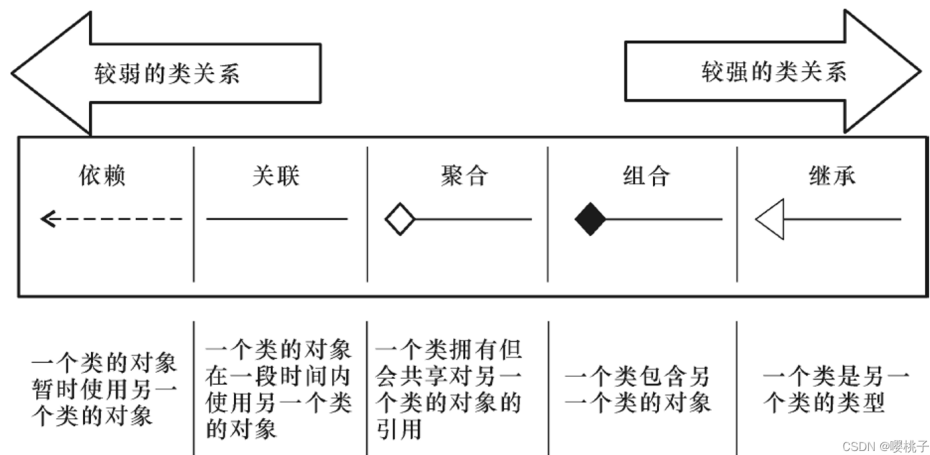
- 类图

- 示例



- 类间关系：①依赖②关联③聚合④组合⑤泛化（继承）⑥实现

■ 类关系由弱到强次序及表示



■ 基于类建模

■ 分析类

- **外部实体** (例如, 其他系统、设备、人员), 产生或使用基于计算机系统的信息。
- **事物** (例如, 报告、显示、字母、信号), 问题信息域的一部分。
- **偶发事件或事件** (例如, 所有权转移或完成机器人的一组移动动作), 在系统操作环境内发生。
- **角色** (例如: 经理、工程师、销售人员), 由和系统交互的人员扮演。
- **组织单元** (例如, 部门、组、团队), 和某个应用系统相关。
- **场地** (例如: 制造车间或码头), 建立问题的环境和系统的整体功能。
- **结构** (例如: 传感器、四轮交通工具、计算机), 定义了对象的类或与对象相关的类。

■ 潜在类

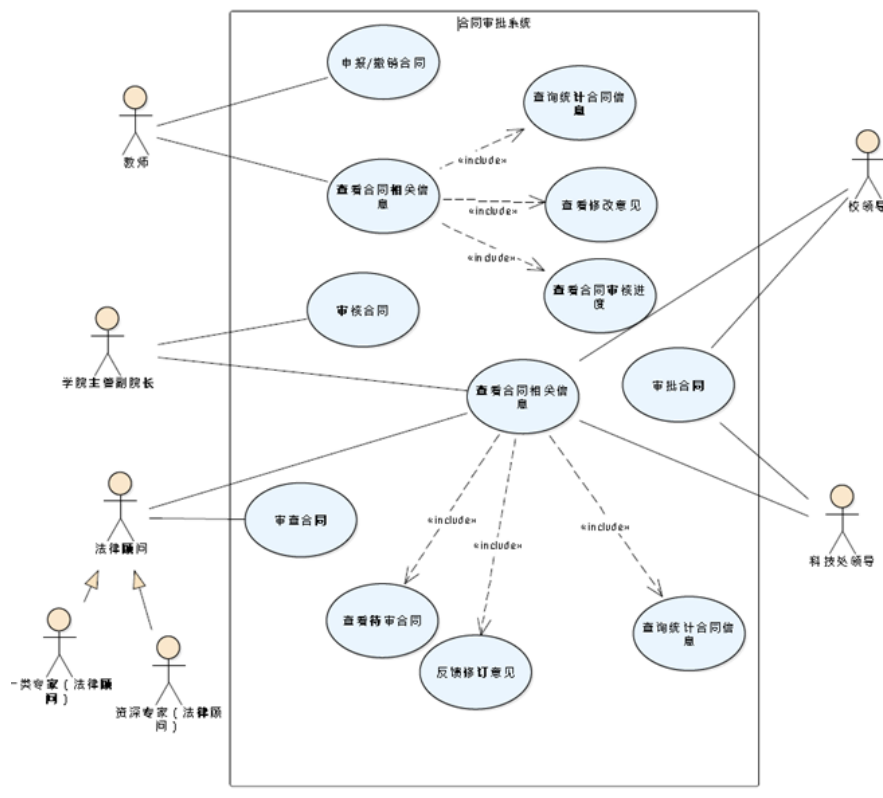
- **保留信息**。只有记录潜在类的信息才能保证系统正常工作, 在这种分析过程中的潜在类是有用的。
- **所需服务**。潜在类必须具有一组可确认的操作, 这组操作能用某种方式改变类的属性值。
- **多个属性**。在需求分析过程中, 焦点应在于“主”信息; 事实上, 只有一个属性的类可能在设计中有用, 但是在分析活动阶段, 最好把它作为另一个类的某个属性。
- **公共属性**。可以为潜在类定义一组属性, 这些属性适用于类的所有实例。
- **公共操作**。可以为潜在类定义一组操作, 这些操作适用于类的所有实例。
- **必要需求**。在问题空间中出现的外部实体, 和任何系统解决方案运行时所必需的生产或消费信息, 几乎都被定义为需求模型中的类。

○ 对象图

- 定义: **对象图是类图的实例**, 几乎使用与类图完全相同的标识。他们的不同点在于对象图显示类的多个对象实例, 而不是实际的类

○ 用例图

- 示例



■ 基本元素：①用例②参与者

■ 关系

■ 关联关系

■ 包含关系：箭头出发的用例为**基用例**；**包含用例（客户用例）**执行，则被包含用例（提供者用例）必须执行

■ 扩展关系：箭头指向的用例为**基础用例**；没有基础用例，扩展用例也是完整的用例；基础用例被执行时，一般不会涉及扩展用例，**只有特定的条件发生，扩展用例才可能被执行**，这是与包含关系的差别

■ 泛化关系

■ 用例描述

■ **简要描述（说明）**：对用例的角色、目的的简要描述；

■ **前置（前提）条件**：执行用例之前系统必须要处于的状态，或者要满足的条件；

■ **基本事件流**：描述该用例的基本流程，指每个流程都“正常”运作时所发生的事情，没有任何备选流和异常流，而只有最有可能发生的事件流；

■ **其他事件流**：表示这个行为或流程是**可选的或备选的**，**并不是总要执行它们**；

■ **异常事件流**：表示发生了某些非正常的事情所要执行的流程；

■ **后置（事后）条件**：用例一旦执行后系统所处的状态；

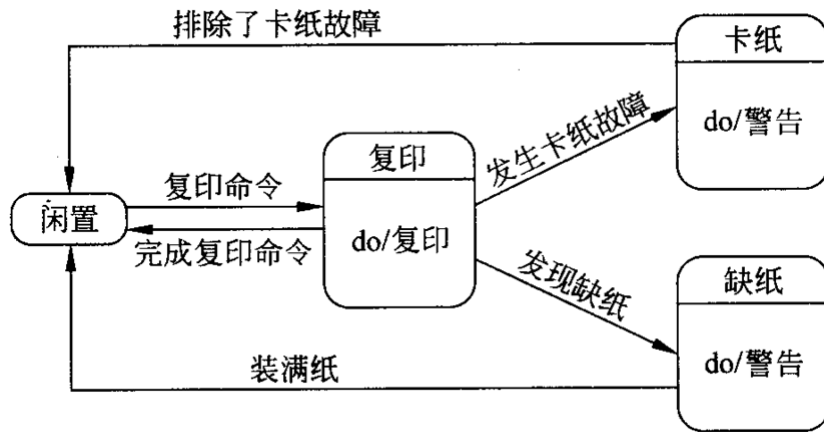
■ 示例

用例名称： 网站公告发布
用例标识号： 202
参与者： 负责人
简要说明： 负责人用来填写和修改家教网站首页的公告，公告最终显示在家教网站的首页上。
前置条件： 负责人已经登陆家教网站管理系统

用例名称：网站公告发布
基本事件流： 1. 负责人鼠标点击“修改公告”按钮 2. 系统出现一个文本框，显示着原来的公告内容 3. 负责人可以在文本框上修改公告，也可以完全删除，重新写新的公告 4. 负责人编辑完文本框，按“提交”按钮，首页公告就被修改 5. 用例终止
其他事件流A1： 在按“提交”按钮之前，负责人随时可以按“返回”按钮，文本框的任何修改内容都不会影响网站首页的公告
异常事件流： 1. 提示错误信息，负责人确认 2. 返回到管理系统主页面
后置条件： 网站首页的公告信息被修改
注释： 无

○ 状态图

■ 示例



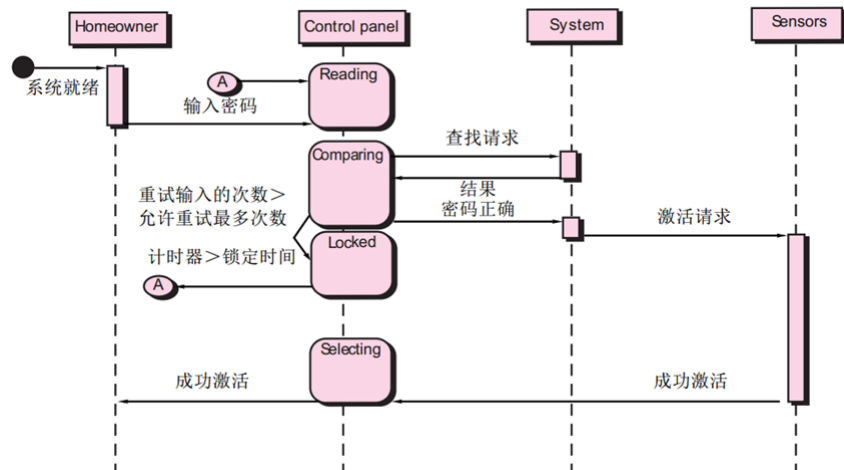
■ 三要素：①事件②状态③行为

■ 基本元素

状态	上格放置名称，下格说明处于该状态时，系统或对象要做的工作(见可选活动表)	<div>Enter Password</div> <pre>entry / set echo to star; password.reset() exit / set echo normal digit / handle character clear / password.reset() help / display help</pre>
转移	转移上标出触发转移的事件表达式。如果转移上未标明事件，则表示在源状态的内部活动执行完毕后自动触发转移	<div>消息(属性)[条件]/动作</div>
开始	初始状态(一个)	
结束	终态(可以多个)	

○ 顺序图

■ 示例（考虑交互）

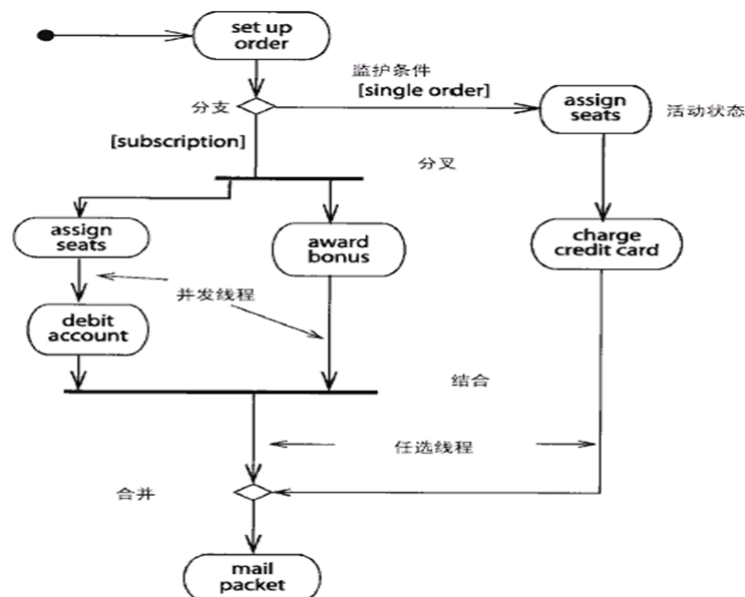


■ 基本元素

事物名称	解释	图
参与者	与系统、子系统或类发生交互作用的外部用户(参见用例图定义)。	
对象	顺序图的横轴上是与序列有关的对象。对象的表示方法是：矩形框中写有对象或类名，且名字下面有下划线。	
生命线	坐标轴纵向的虚线表示对象在序列中的执行情况(即发送和接收的消息，对象的活动)这条虚线称为对象的“生命线”。	
消息符号	消息用从一个对象的生命线到另一个对象生命线的箭头表示。箭头以时间顺序在图中从上到下排列。	同步 异步 简单 同步且立即返回

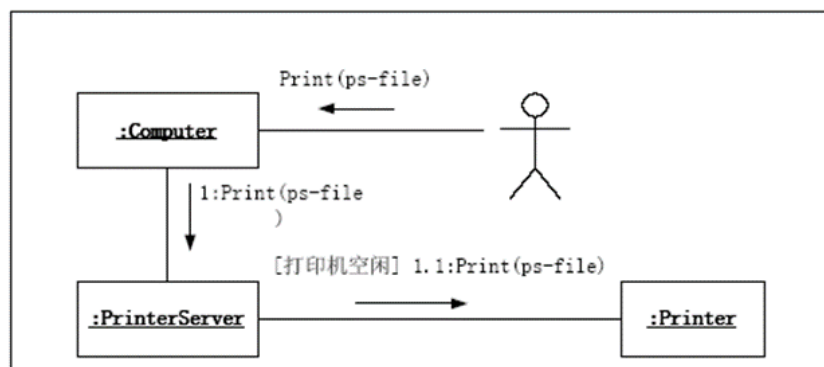
○ 活动图/泳道图

■ 示例（类似流程图）



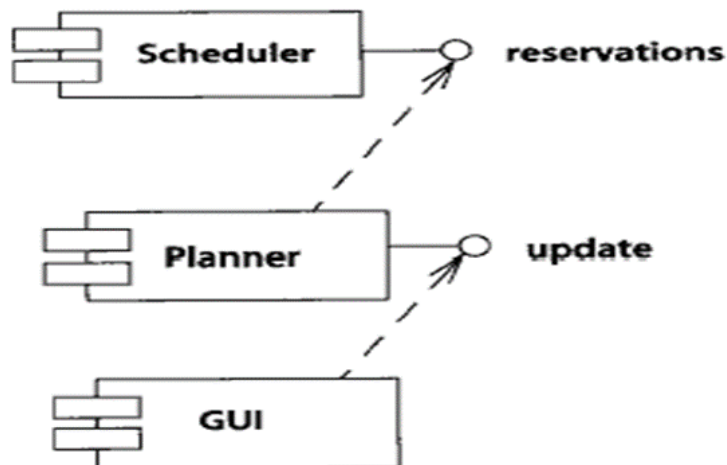
○ 协作图

■ 示例（描述协作关系；用途：表示一个类操作的实现）






○ 构件图

■ 示例



- 说明：图中“Planner计划者”构件向外提供一个“update更新”接口服务；同时，该构件要求外部接口提供一个“Reservations预定”服务。

■ 基本元素

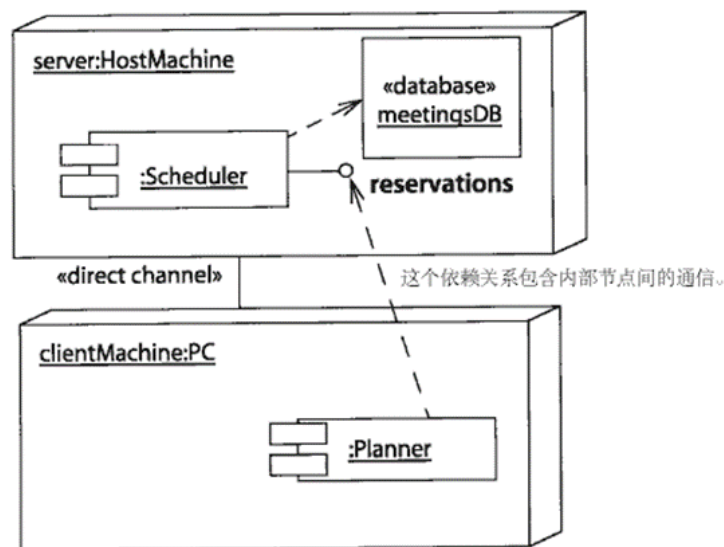
事物名称	含义	图例
构件	指系统中可替换的物理部分，构件名字(如图中的Dictionary)标在矩形中，提供了一组接口的实现。	 Dictionary
接口	外部可访问到的服务 (如图中的Spell-check)。	 Spell-check
构件实例	节点实例上的构件的一个实例，冒号后是该构件实例的名字(如图中的RoutingList)。	 :RoutingList

■ 关系：①实现关系②依赖关系

- 作用：构件为构造应用的软件单元，包括各构件之间的依赖关系，通过这些关系来估计对系统构件的修改给系统带来的影响

○ 部署图

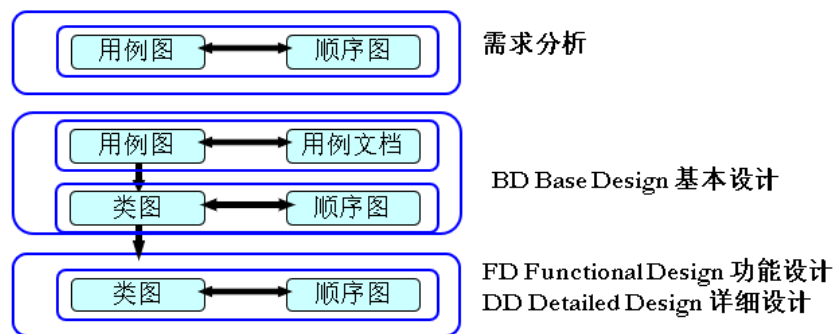
- 示例（描述位于节点实例上的运行构件实例的安排，节点是一组运行资源，如计算机、设备或存储器）



■ 基本元素

事物名称	解释	图例
节点	节点用一长方体表示，长方体中左上角的文字是节点的名字 (如图中的Joe'sMachine:PC) 。节点代表一个至少有存储空间和执行能力的计算资源。节点包括计算设备和(至少商业模型中的)人力资源或者机械处理资源，可以用描述符或实例代表。节点定义了运行时对象和构件实例(如图中的Planner构件实例)驻留的位置。	
构件	系统中可替换的物理部分。	
接口	外部可访问的服务。	
构件实例	构件的一个实例。	

3. 各UML之间的关系



主要图之间的关系

