《面向对象设计与构造》
Lec14-多视角的模型化设计

OO2023课程组

北京航空航天大学计算机学院

提纲

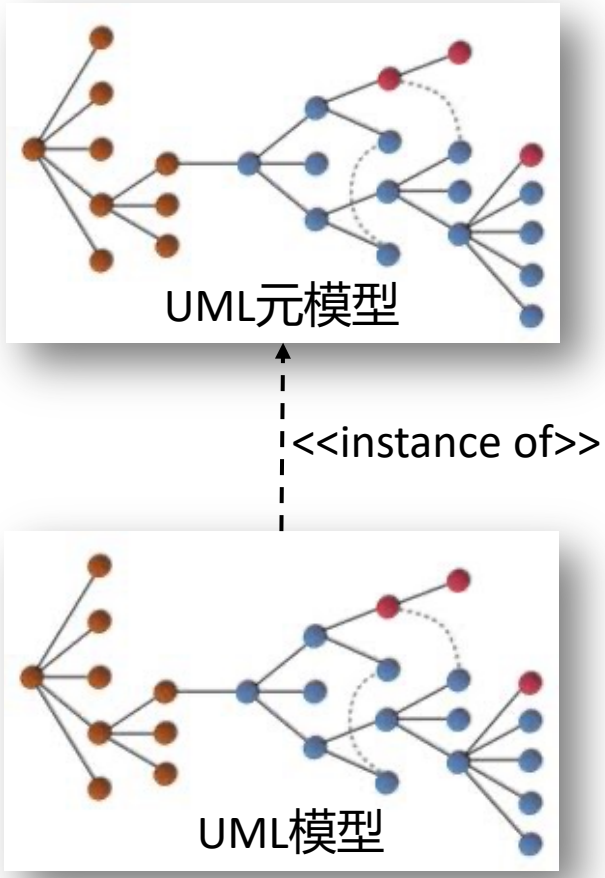
- 什么是模型
- 模型化设计
- 类模型设计
- 类图的表示结构
- 状态图的表示结构
- 状态评测规则
- 作业解析

什么是模型

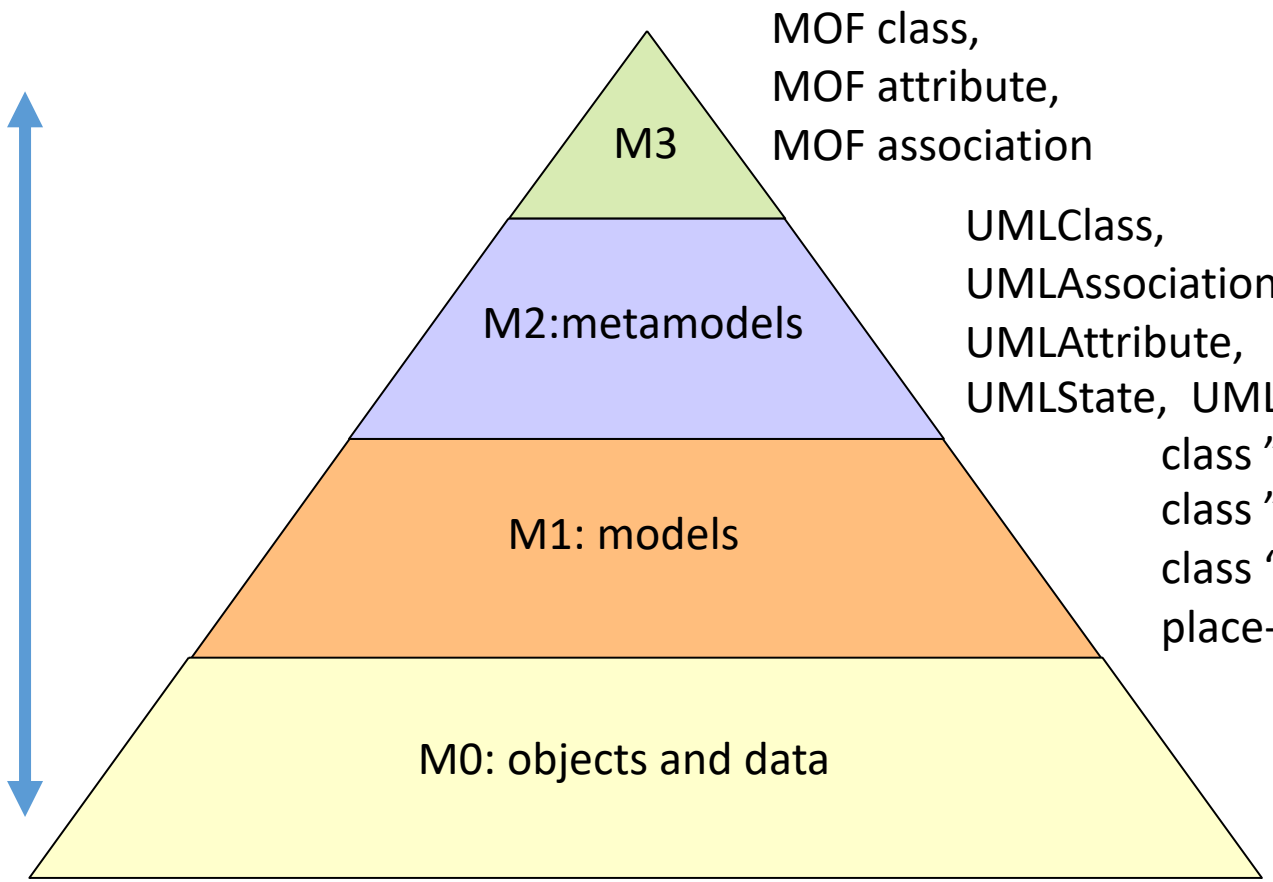
- 模型：对用来描述一个目标对象的元素及其关系、约束的统称
- UML模型：各种<UML***>类型的对象及其关系和相应的约束
- UML建模工具提供可视化图(diagram)，让开发者以‘画图’的方式来构建类型为<UML***>的各种对象，工具后台自动维护和管理这些对象之间的关系
- UML模型中的对象关系
  - 上下层关系：*member*，parent
  - 全局性的引用关系：type, source, target, …

什么是模型

- 模型必须使用一套统一的数据结构来加以表示
  - 所有的类型其实都是事先定义好的：UML****
- 模型就是一个把若干对象连接起来的图(graph)
  - 可视化层的节点对象
  - 可视化层的连接边对象
  - 都是一种UML***类型
  - 这些对象之间存在member-parent或者ref**关系**
- 为了定义{UML***}，UML语言还定义了诸多中间结构，把这些类型元素连接起来，形成一个更高层次的图(graph)：元模型(meta-model)



什么是模型



MOF class,
MOF attribute,
MOF association

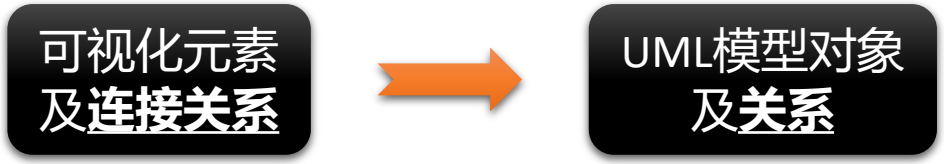
UMLClass,
UMLAssociation
UMLAttribute,
UMLState, UMLTransition, …

class ”Customer”,
class ”Company”,
class “Order”, “Computer”,
place-order activity

customer ‘老王’ places
an order of computer
‘Lenovo T450s’ from
company ‘JD.com’

模型化设计

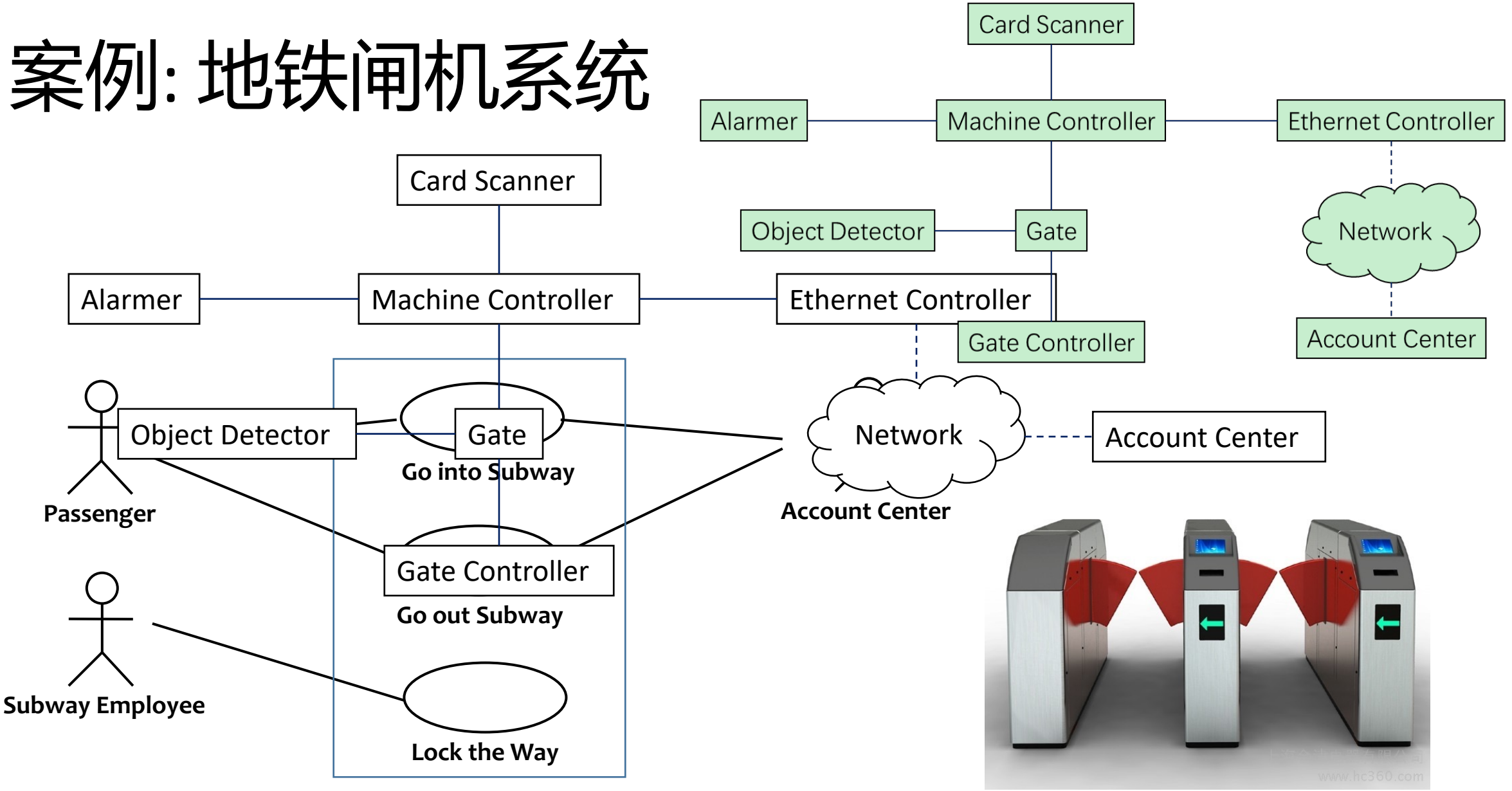
- 使用系统化的模型语言来表示设计结果，进而开展设计思考
- UML采用了视图与模型相分离的设计
  - 在提供的diagram中表达相应的元素和关系
  - 建模工具维护UML模型
- UML提供了四种模型视图
  - 功能视图
  - 结构视图
  - 行为视图
  - 部署视图



模型化设计

- UML功能视图(use case diagram)支持的元素及关系表达
  - 用例(use case)：系统提供给用户的功能及其交互场景
    - Precondition, postcondition, flow of events
  - 执行者(actor)：为系统执行提供输入激励或者记录系统执行结果的相关对象
    - 自然人、设备、其他系统等
  - actor与use case之间的关系
    - 哪些用户为这个用例提供输入？
    - 哪些用户关心这个用例的执行结果？
  - use case与use case之间的关系
    - 依赖关系
    - 抽象层次关系

案例: 地铁闸机系统


Card Scanner
Alarmer
Machine Controller
Ethernet Controller
Network
Object Detector
Gate
Gate Controller
Account Center
Passenger
Go into Subway
Go out Subway
Subway Employee
Lock the Way

模型化设计

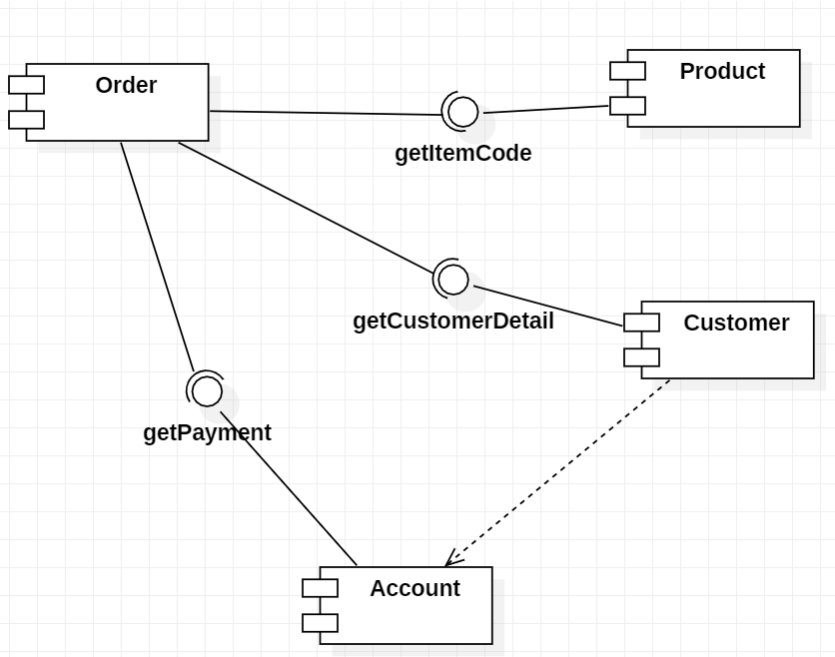
- 结构视图
  - 组件图
  - 类图
- 类图支持的元素及关系表达
  - 类、接口
    - 属性、操作
  - 关联关系
  - 继承关系
  - 接口实现关系

模型化设计

- 我的系统为什么需要这个类？
- 从与用户交互功能场景角度
  - 边界类
  - 实体类
  - 控制类
- 从数据封装与处理角度
  - 映射到功能需求中的数据项
  - 类中所封装数据项之间的聚合特性
- 从层次关系角度
  - 容器类
  - 控制策略类
  - 归一化类/接口

模型化设计

- **组件图**在系统架构设计中具有重要地位
  - 组件有多种类型
  - 软件组件、硬件组件、资源组件、网络组件等
- 每个组件定义两类接口
  - Provided Interface
  - Required Interface
- 组件之间有功能依赖关系
- 每个组件都可以使用一副类图来描述其设计逻辑



模型化设计

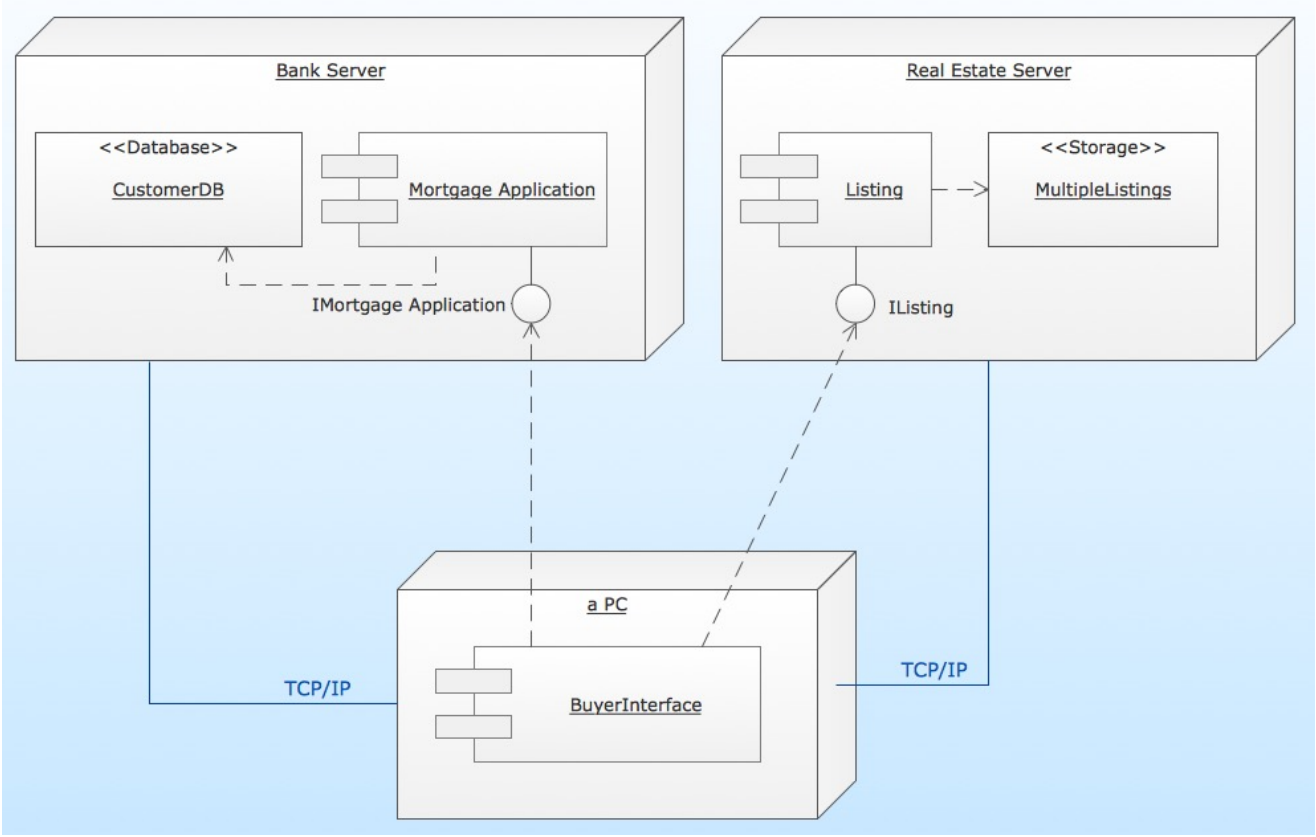
- 行为视图
  - 顺序图
  - 状态图
  - 活动图
- 顺序图围绕一个功能场景(如use case)，从**对象交互**角度给出相应use case的设计方案
  - **确定哪些对象参与交互**
  - 识别类应提供的操作
  - 识别类之间的关联关系
- 顺序图采用消息交互机制
  - 消息与对象操作相关联

模型化设计

- 状态图针对具有一定状态复杂度的类来专门设计其行为
  - 多种状态和转移关系，运行时会随输入动态改变其状态
  - 不同状态之间在属性取值上必须严格分离
- 一般多用于描述控制行为：针对被控对象的状态来实施相应的控制行为，并维护其状态更新
  - 电梯调度(电梯对象的状态空间)：根据电梯状态来分配乘客请求
- 活动图使用控制流+数据流来描述一个功能（用例）或一个业务场景的执行流程
  - 可表示同步和异步行为
  - 可表示并发和同步控制

模型化设计

- 部署视图支持的元素及其关系
  - 部署节点：提供运行所需的资源
  - 组件：一个部署单位，提供相对完整的业务功能和相应数据管理功能
  - 部署节点与部署节点：依赖和通信交互关系
  - 部署节点与组件：节点为组件提供运行时资源
- 部署图展示系统的部署安排和拓扑结构



模型化设计的思维方法

- 抽象思维
  - 按数据或行为提取抽象，形成抽象层次结构
- 分类思维
  - 概念分类、对象分类
  - 建立抽象类型层次或对象分类处理容器
- 层次思维
  - 按数据管理或从属层次建立关联结构
  - 按照行为实施层次建立分层设计
- 分段思维
  - 按业务处理流程分段处理，建立顺序结构或层次结构
- 映射思维
  - 按数据间的因果关系建立映射关联，实现数据的快速检索

抽象、分类、层次会在**数据结构**上形成**层次**
抽象、层次、分段会在**处理流程**上形成**层次**
分类、层次、映射会在**数据管理**上形成**层次**

模型化设计的思维方法

- 抽象是建模中的最重要方法
- 忽略细节，抓住本质
- 几乎每个UML模型图都需要使用这种思考方法
  - 识别类、属性、操作、关联和继承等
  - 识别接口和接口实现关系
  - 识别状态、迁移
  - 识别消息连接关系

模型化设计的思维方法

- 分类是最常用的一种抽象方法
- 类图
  - 识别类和接口，建立它们的继承关系
  - 建立多重关联，按照角色和特征进行分类化对象管理
- 状态图
  - 识别类的多个状态
  - 按照状态来设计类的行为
- 按照软件的处理逻辑把实例化对象分类/分组，有效降低处理的逻辑复杂性

模型化设计的思维方法

- 按照输入到输出处理过程，区分活动段，按段来识别相应数据抽象和行为抽象
- 在段之间建立数据流关系，形成协同结构
- 系统设计中必然涉及诸多数据，如何管理这些数据是一个不能忽视的问题
  - 建立数据管理**层次**
  - 结合数据分组构建多叉管理**层次**
  - 管理层次往往和协同结构**一致**
- 过深的数据管理层次显著加大数据检索的代价
  - **跨层次**间建立映射结构，快速检索和更新

类模型设计

- 类图是UML建模的核心和基础
- 广泛采用的“入门级”方法：名词识别法
  - 具有明确的概念内涵和相应数据内容的名词
- 容易出现混淆的概念
  - 类-名词：这类名词往往蕴含着多维数据，如请求、出租车等
  - 属性-名词：这类名词往往蕴含着单维数据(但可能多例)，如目的地点、出发时刻等
  - 关联角色：这类名词往往蕴含着组合层次和对象实例分类，如抢单出租车、等待服务出租车
  - 控制策略/机制：这类名词往往是对一种动态控制机制或策略的概括描述，如优先级调度、抢单时间窗口等

类模型设计

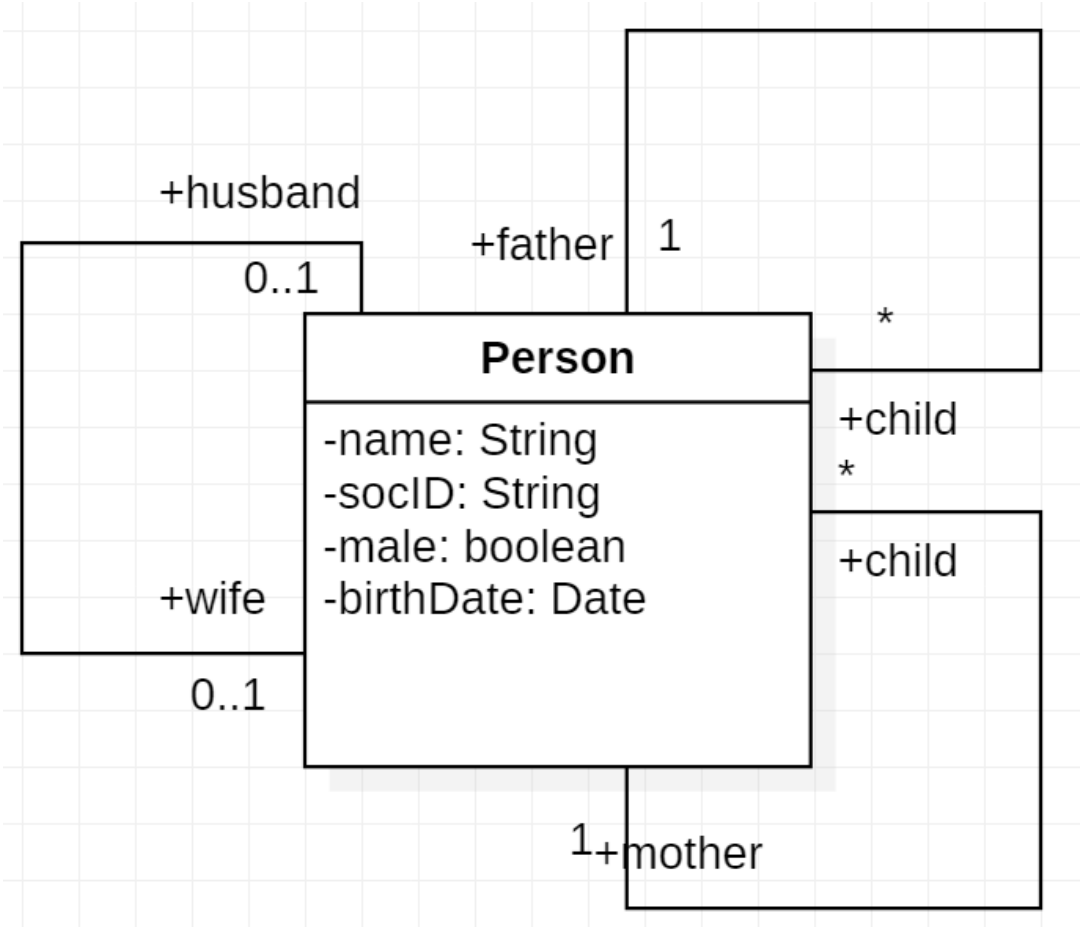
- 属性识别
  - 从问题域角度，要完成相应的功能，需要记录和管理的相关数据
  - 出租车需要管理哪些数据？
  - 请求需要管理哪些数据？
- 操作识别
  - 从所识别的数据角度来识别对数据的处理
  - 从系统用户与系统的交互事件角度，分配相应的职责
    - 打车系统中RequestManager类的操作
  - 系统事件处理往往对应着需求描述中的一些策略机制概述

类模型的设计

- 关联关系(Association)
  - 定义**两组对象**之间的关系
  - 通过UMLAssociation来表达
    - 双向关系：两个UMLAssociationEnd对象，地位相同
  - 通过属性类型引用来表达
    - 单向关系：由属性所在类指向属性类型
  - 聚合特性反映end1端连接对象与end2端连接对象之间的关系特性
    - none：两边都不是容器对象
    - shared：容器对象与元素对象关系，且共享管理元素对象
    - composite：容器对象与元素对象关系，且独享管理元素对象
- 任意两个类之间可以建立多个关联关系，互相独立

类模型的设计

- 人有父母、子女、配偶
- 中国现在实施新的生育政策，一对夫妻允许生三个孩子
- 中国实施一夫一妻制
- 如何用关联关系把其中的概念和关系表示出来？

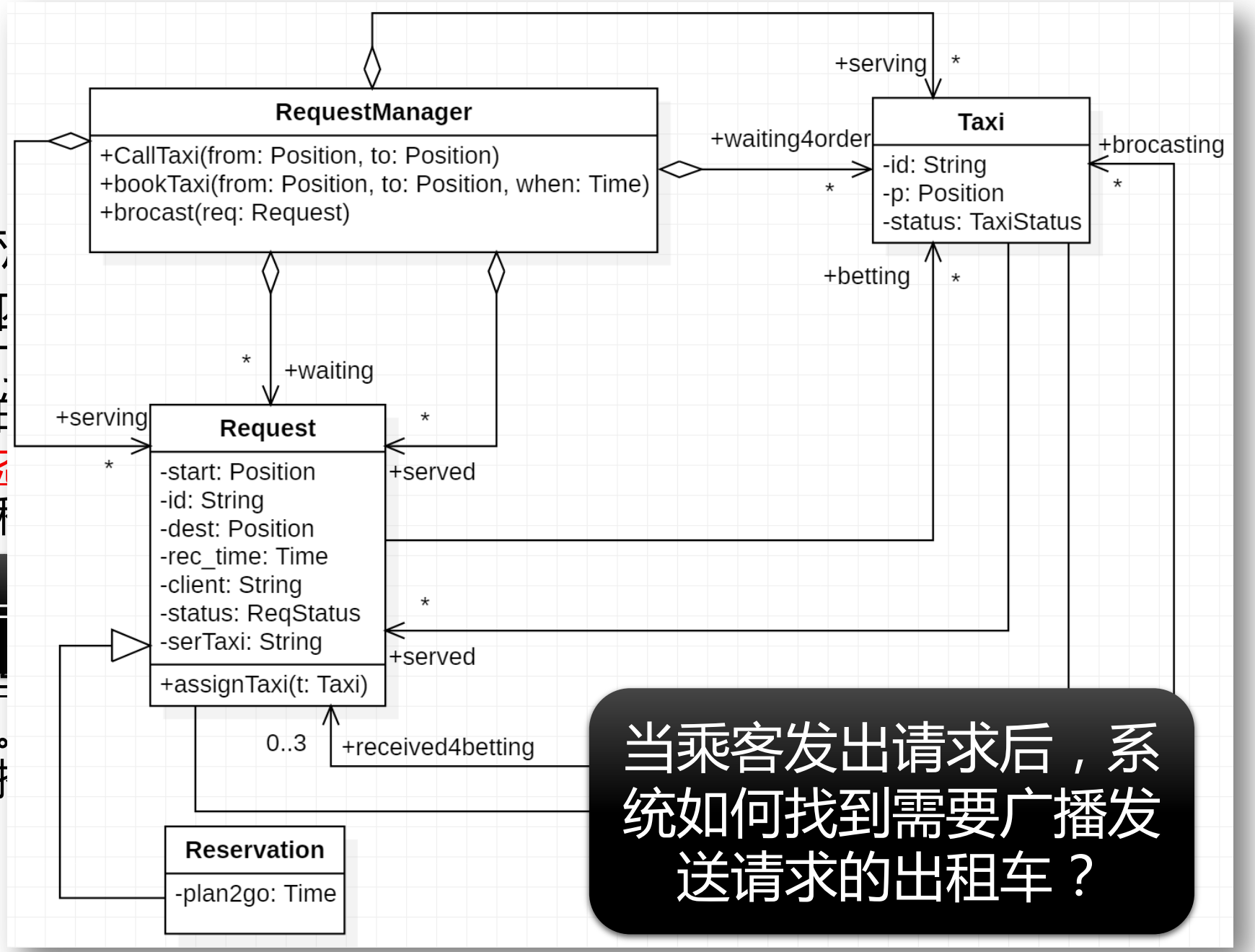


如何表达人的兄弟姐妹关系？

类模型设计

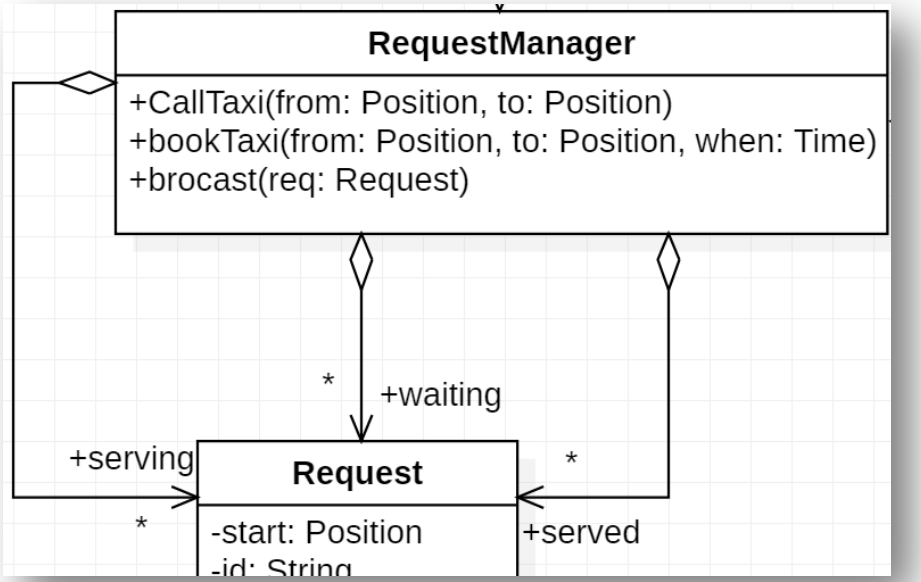
流程式描述

抢单时间窗口其实是一个控制机制，本质上是要在Request与Taxi之间建立关系。而所谓时间窗口无非是关联关系实例的**建立时刻**和**拆除时刻**！



类模型设计

- 关联的识别和处理
  - 从模型角度，只要识别了一个类，意味着就能构造该类的大量实例对象
  - 对这些实例对象的不同子集的管理策略往往有差异，因而特别表示出来
  - 通过对象分类/分组，可以有效简化系统设计
  - 关联角色容易被误识别为特殊的类
    - 等待处理的请求 vs 请求
- 关联是为了让一个类使用对方来管理数据或完成自己的行为
  - 关联一般实现为属性数据

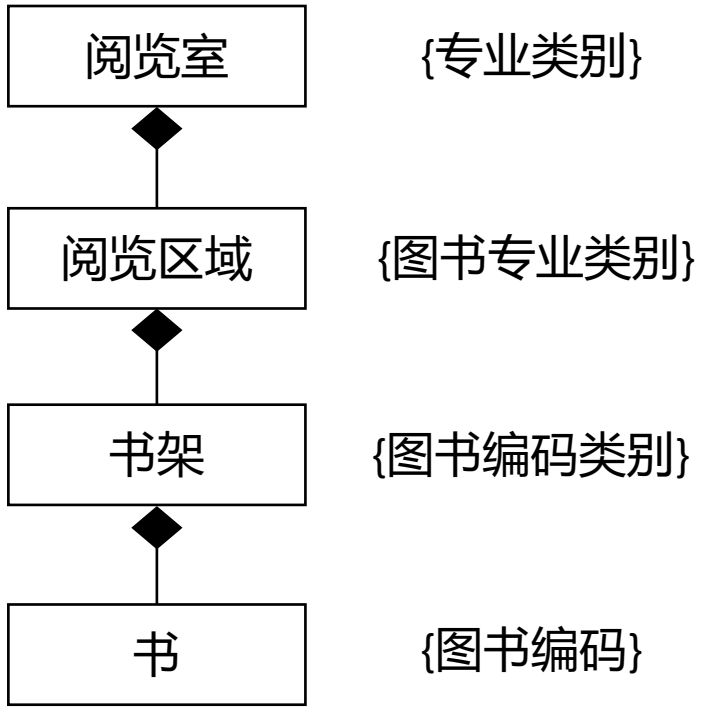


类模型设计

- 继承的识别与处理
  - 问题域描述中往往会出现多种形态的实体描述：请求、即时请求、预约请求，服务中请求、已服务请求等
  - 其中有些实体描述其实是按照角色的分类描述
  - 从继承角度，核心是分析不同的实体描述是否在数据上有显著不同
    - 请求，即时请求，预约请求
    - **抓住数据抽象这个本质！**
- 一个实体需要管理或记录哪些数据，取决于系统的领域需求
- 如果一个实体只关心它的行为，不关心数据，说明应该识别为一个接口，比识别为类带来更多灵活性

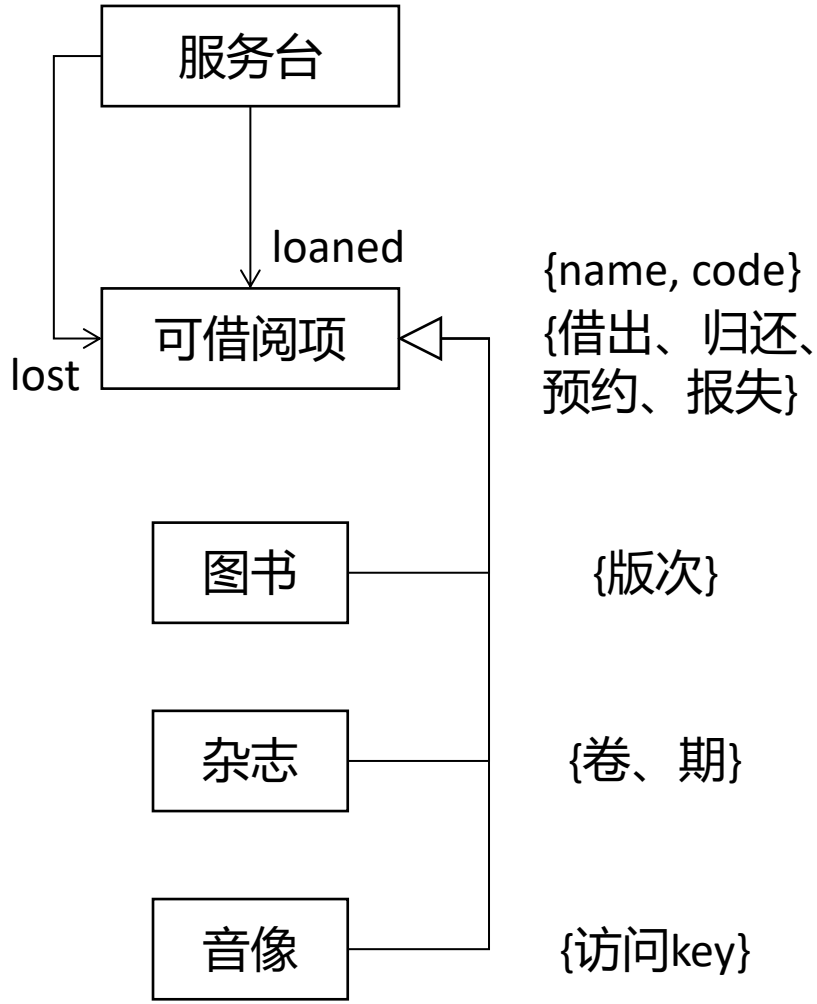
UML类图的表达结构(1)

- 层次化数据管理
  - 架构设计中经常涉及的逻辑结构
  - 使用容器管理，访问效率
  - 分层管理，delegation降低复杂度
- 类图可以使用多级关联来表达
  - 阅览室划分为多个区域
  - 每个区域由多个书架组成
  - 每个书架码放多本书
  - 同一个区域的书架拥有相同图书专业编码
  - 每本书有更加具体的专业信息编码



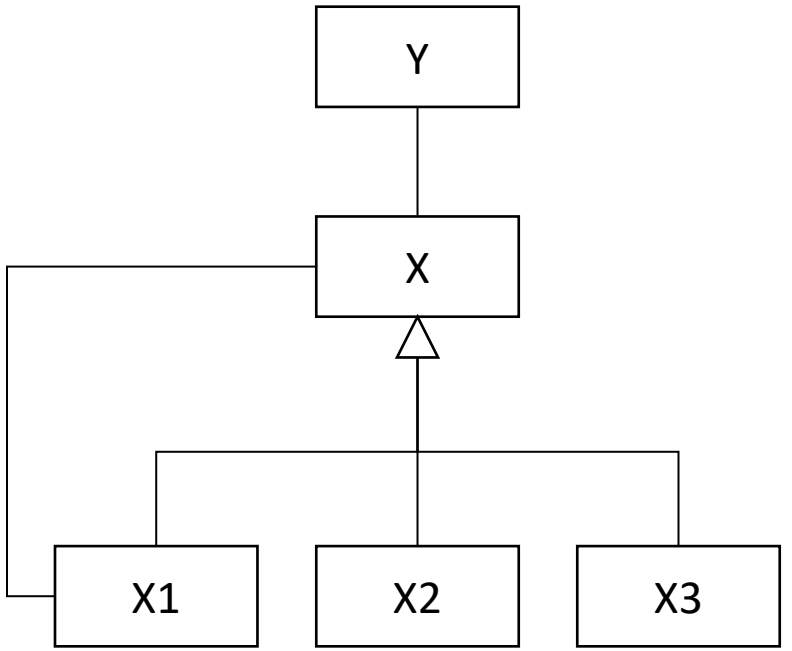
UML类图的表达结构(2)

- 数据/行为的归纳/抽象表示
  - 通过顶层的行为归纳表示可以概括下层的多种行为**实现**或**重写**
  - 归一化设计的核心
- 类图可以用接口实现或继承+关联来表达
  - 图书馆提供了图书、杂志和音像媒体（如ISO电子文件）的借阅服务
  - 图书和杂志可以借出，音像媒体只能在馆内借阅
  - 音像媒体借阅时提供一个电脑终端访问key，登录终端电脑通过key来播放和阅读



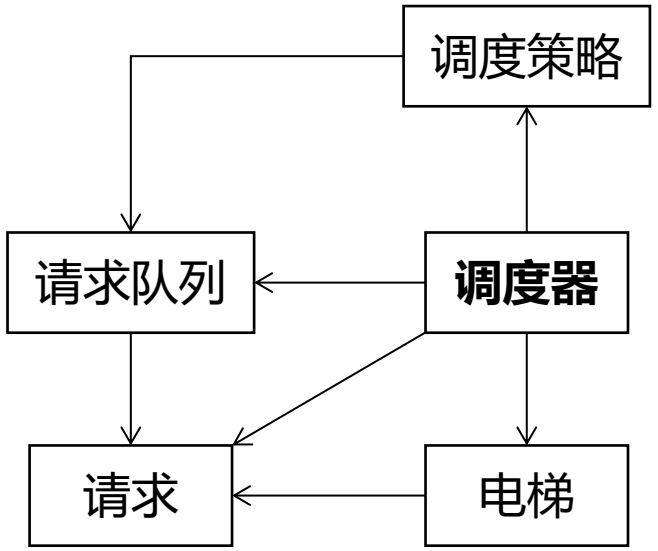
UML类图的表达结构(3)

- 数据内在具有递归结构特征
  - 文件目录下管理着多种类型的文件（如文本、图片、音频等）
  - 文件目录也是一种文件类型
- 类图可以采用关联+继承+递归关联来表示
  - 多项式由项组成
  - 项由因子组成
  - 因子包括常数因子、变量因子和表达式因子三种
  - 表达式因子管理着一个多项式



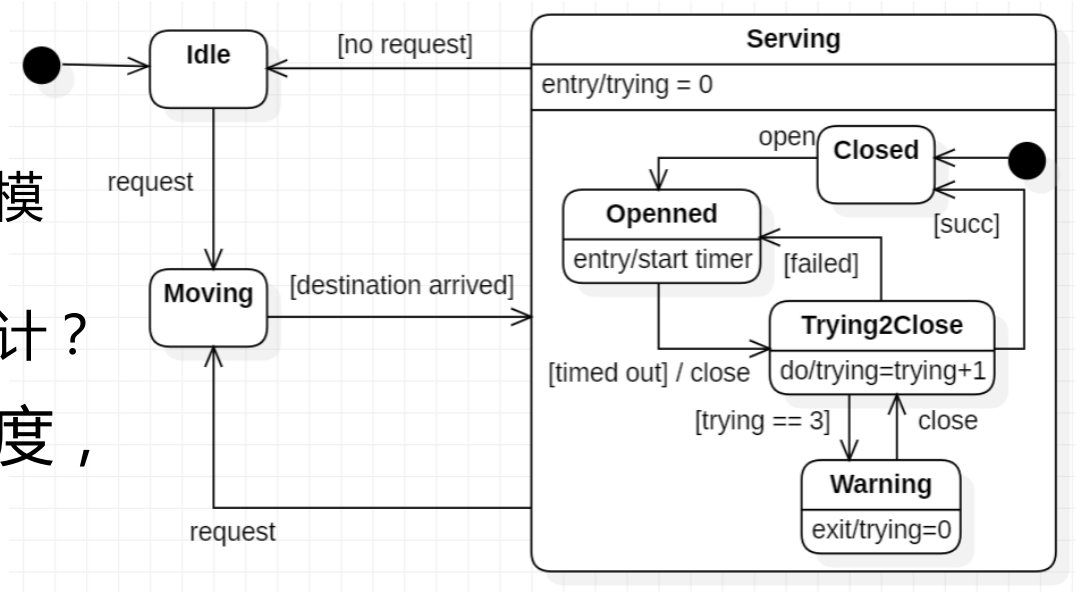
UML类图的表达结构(4)

- 很多软件涉及控制策略和行为
  - 往往具有全局性特征
  - 如网络通信中的路由控制
- 类图通过控制类+多个关联来表示
  - 调度是电梯系统的核心控制逻辑
  - 动态方式针对电梯状态和当前待处理的请求来确定下一个分配给电梯的请求
  - 调度可以配置多种调度策略，从待处理的请求队列中挑选合适的请求



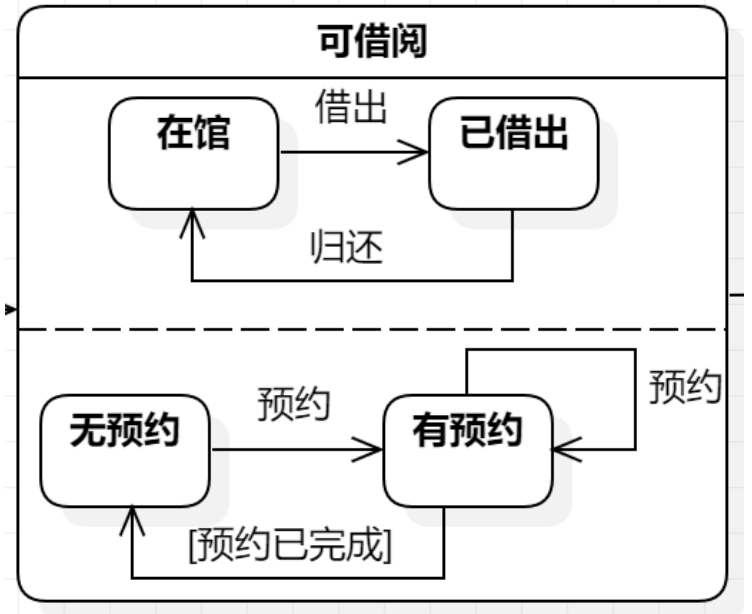
UML状态图的表达结构(1)

- 对象是状态化存在
- 状态表示细致程度取决于建模意图
  - 电梯状态建模：空闲、运动、服务停靠，表明建模者不关心运动方向
  - ➔这样的状态模型能否服务于电梯调度策略的设计？
- 引入复合状态，不增加上层状态迁移的复杂度，同时增加更细致的状态及迁移行为表达能力
  - 以服务停靠状态为例
  - 需要关注开门和关门，以及门开达到一定时间段后自动关门
  - 如果连续3次关门无法成功，则触发警告



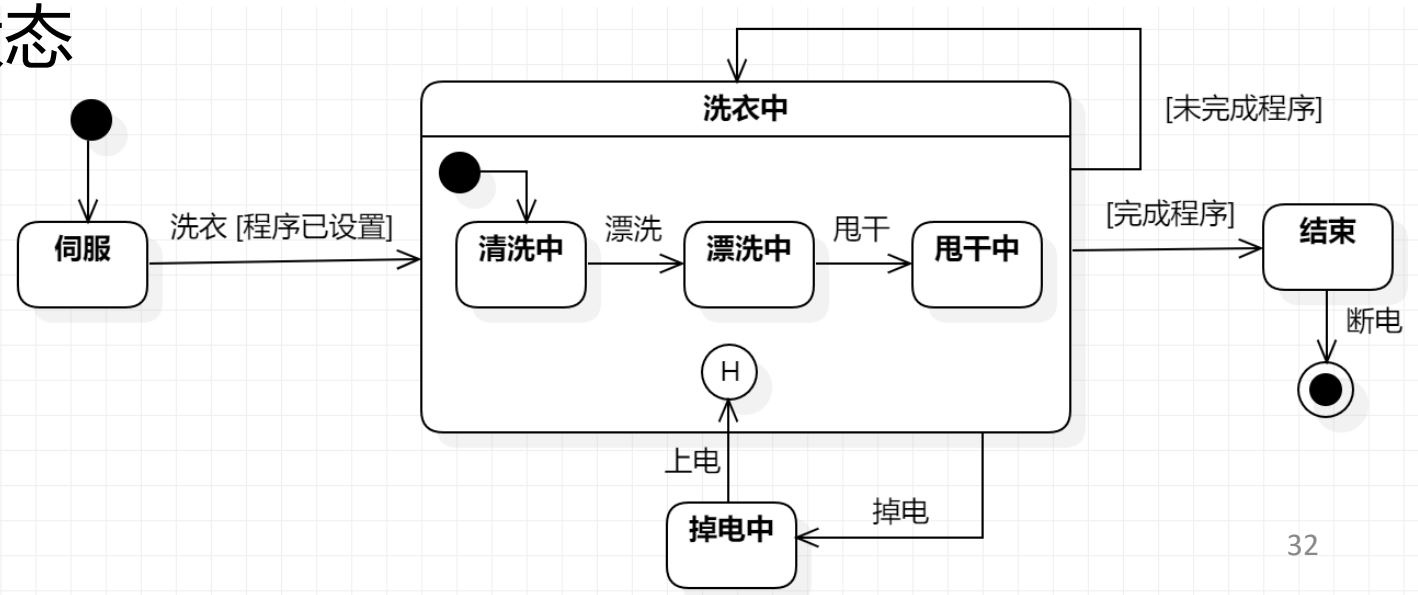
UML状态图的表达结构(2)

- 根据业务需要，一个对象可能需要具备表达多个维度状态的能力
  - 书的状态维度：预约维度、借出维度与损坏维度
  - 出租车的状态维度：服务状态维度与车况舒适度维度
- UML类图提供了平行状态表达能力，针对一个复合状态表示其多维度子状态
  - 一个对象在任意时刻一定处于每个维度下的一个状态
  - 为状态增加UMLRegion即可建立这样的表达



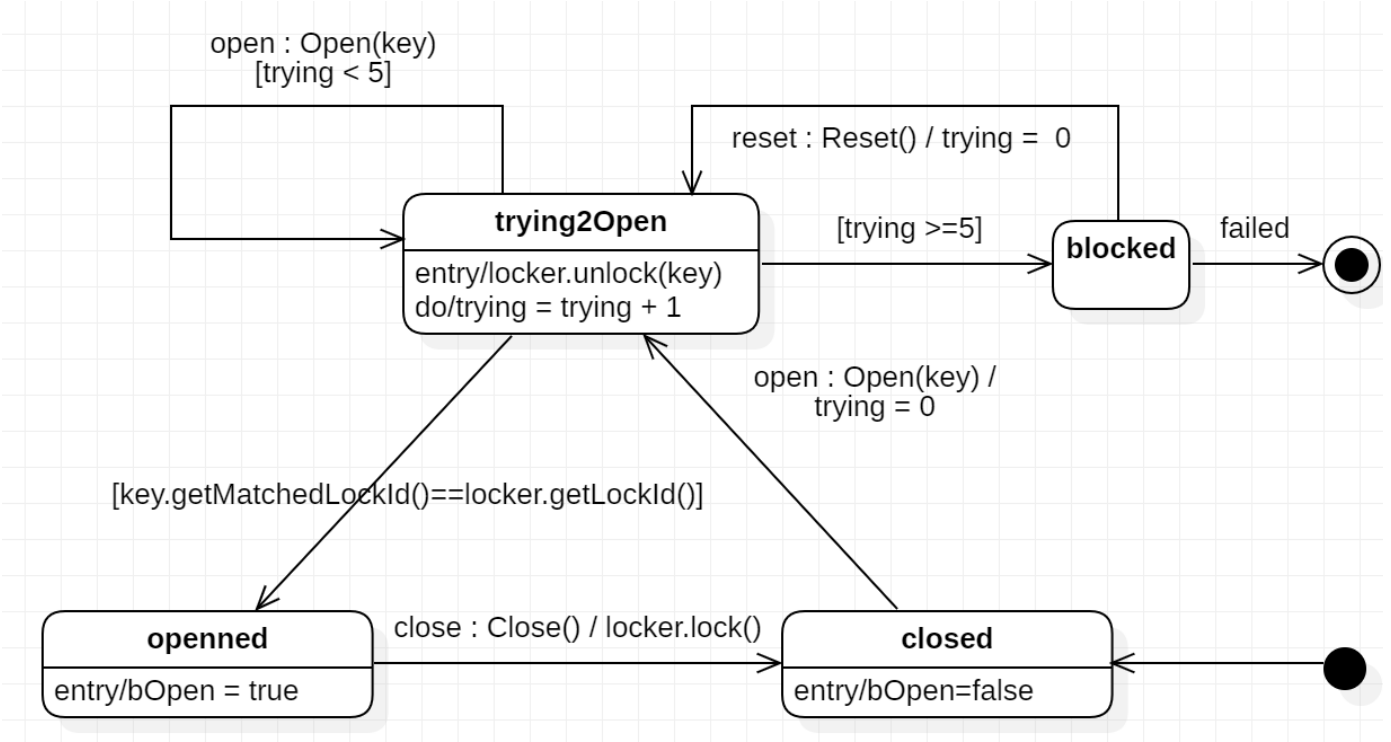
UML状态图的表达结构(3)

- 有些系统为了更好的服务客户，能够记录客户上次来访问系统时所访问的对象，再来访问时可以呈现上次退出时所看到的状态
  - 网上商城，客户可以浏览商品和下订单等，系统会记录每次用户关注的商品类别，下次进入系统时再次呈现相同类别的商品列表
- UML提供了H状态标记，自动记录退出一个复合状态时所处的具体状态，下次进入时自动恢复到H状态



UML状态图的表达结构(4)

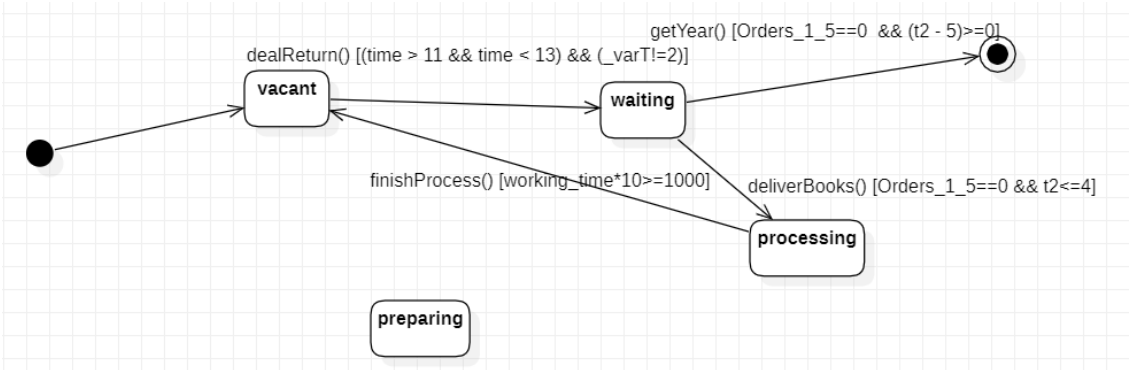
- 对象状态变化意味着对象取值变化
- 有些对象属性专门用于对象状态变化控制
  - 可用于guard逻辑表达
  - 可在迁移effect或者状态行为中进行修改
  - 如trying2open状态中涉及的trying控制变量



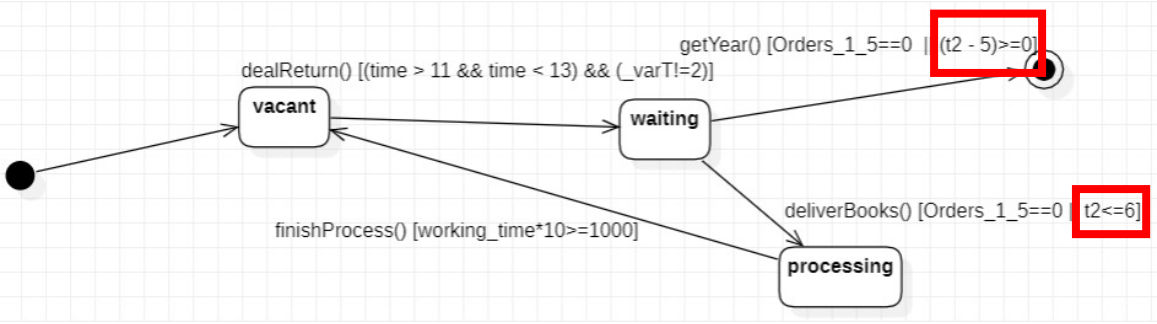
UML状态图评测规则

- R1：状态图中只能有一个起始状态和0到1个终止状态
- R2：起始状态只能有外出迁移，终止状态只能有进入迁移
- R3：除了起始状态的外出迁移，所有状态转移中Trigger和Guard至少要存在一个
- R4：任意一个迁移的Trigger必须**对应到**状态图所属类的一个方法
- R5：任意一个迁移的Guard所涉及的变量，都必须是所在类的成员变量
- R6：从一个状态转移到不同状态的Guard条件必须互斥（不论trigger是否相同）
- R7：任意一个状态s，必然存在一条迁移路径（起点为**起始状态**，终点为s）
- R8：任意一个状态s，必然存在一条路径迁移到终止状态（如果有）
- R9：针对每个状态s，从起始状态到s的所有简单路径都必须有解
  - 路径有解：该路径上所有转移的Guard在一起（逻辑与）有解
  - 简单路径：路径中所有迁移循环走0次或1次

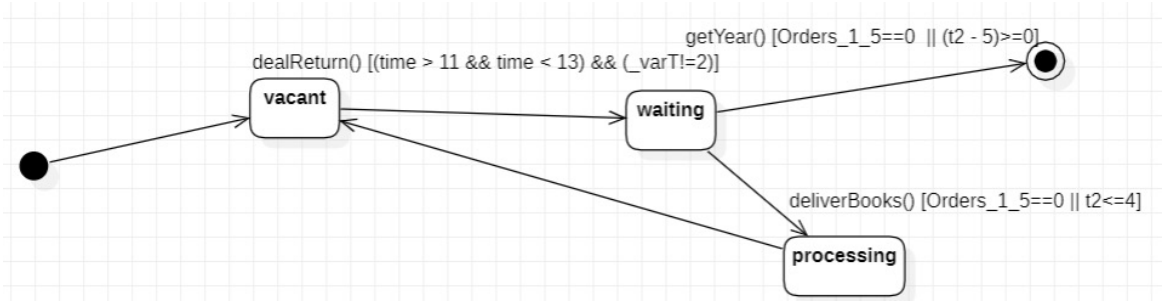
UML状态图评测示例



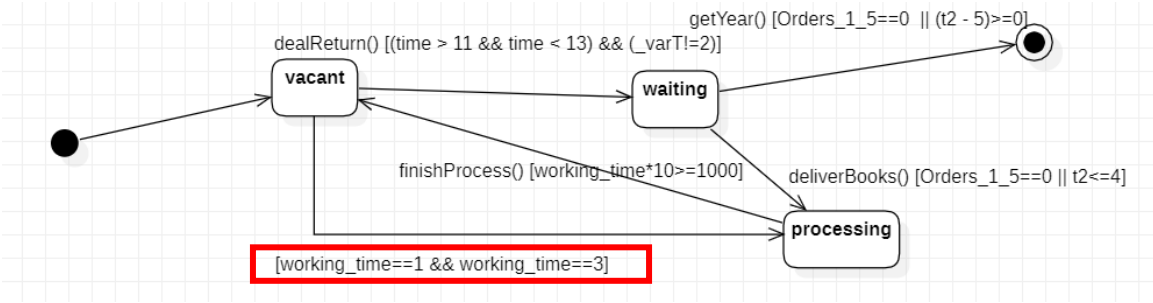
failed：preparing**状态不可达**



failed：waiting状态出发的两个转移的guard**不互斥**



failed：从processing到vacant的迁移**没有Trigger和Guard**



failed：从vacant到processing的**转移路径无解**

作业解析

- 继续迭代图书管理系统
  - 增加校际借阅
  - 增加新购图书的管理
  - 增加状态图评测
    - 且与类图要保持一致
    - 目前没有工具可以自动从代码逆向生成状态图！
- 本单元项目的显著特点
  - 业务流程“繁琐”
  - 涉及的数据多样化
  - 没什么算法的事

模型化设计去繁为简，建立抽象！