《面向对象设计与构造》
Lec14-模型及模型化设计

OO2022课程组

北京航空航天大学计算机学院

提纲

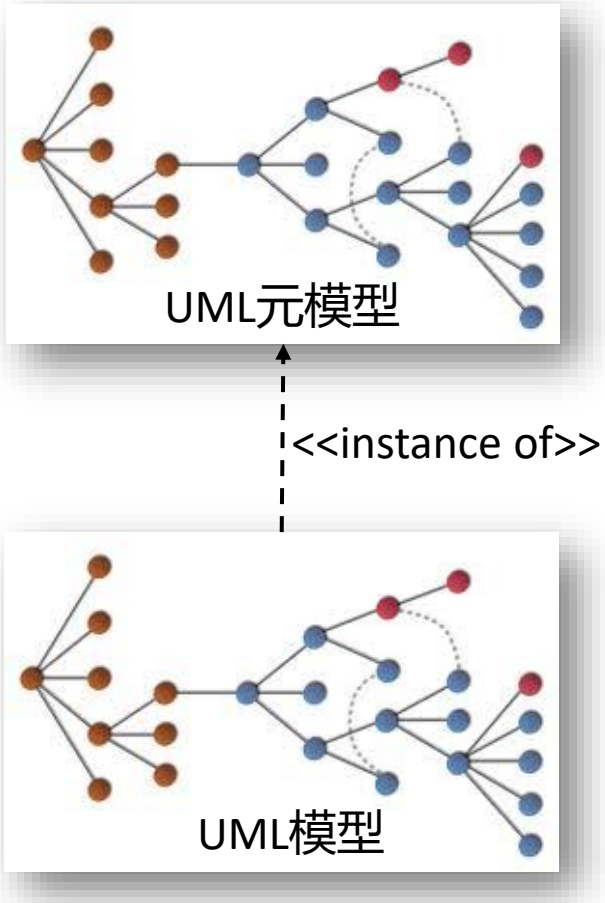
- 什么是模型
- 类模型的理解
- 建模任务
- UML模型图是如何定义的
- 模型化设计
- 作业解析

什么是模型

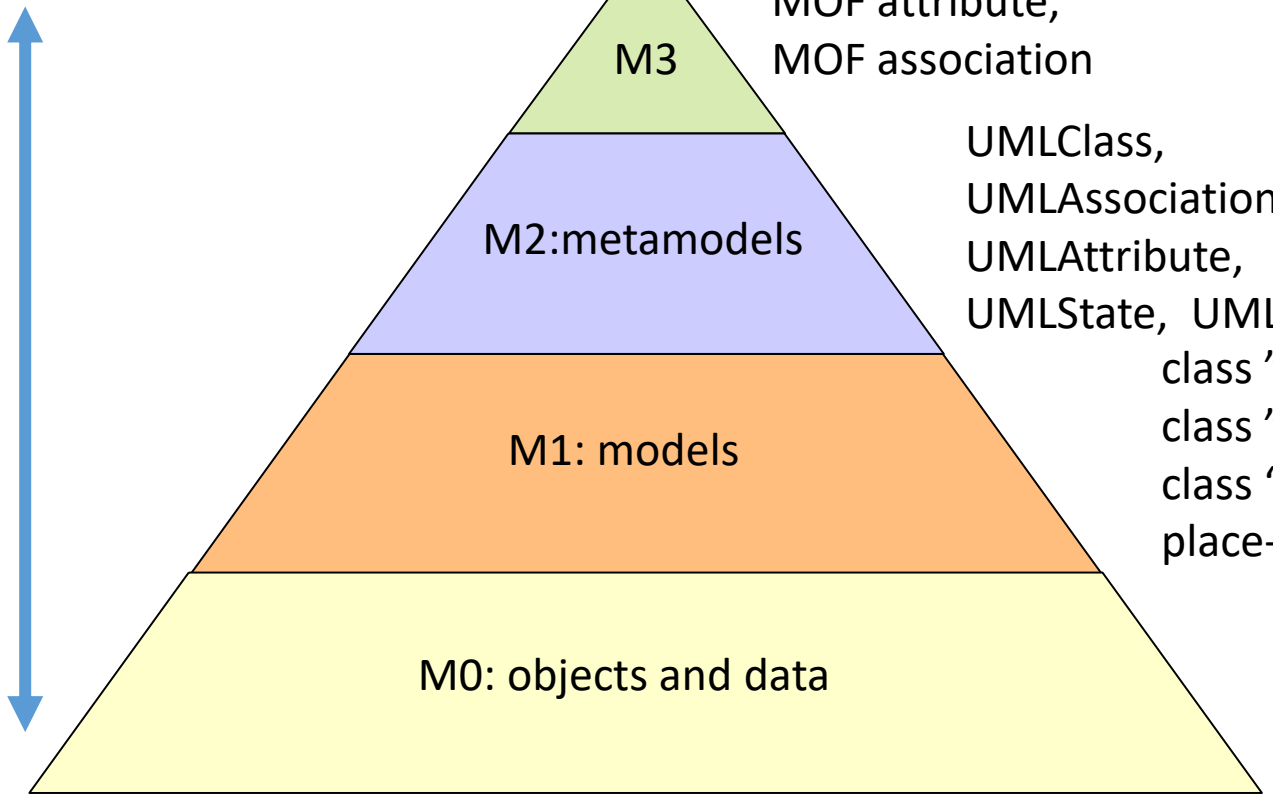
- 模型：对用来描述一个目标对象的元素及其关系、约束的统称
- UML模型：各种<UML***>类型的对象及其关系和相应的约束
- UML建模工具提供可视化图(diagram)，让开发者以‘画图’的方式来构建类型为<UML***>的各种对象，工具后台自动维护和管理这些对象之间的关系
- UML模型中的对象关系
  - 上下层关系：*member*， parent
  - 全局性的引用关系：type, source, target, …

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”, “OrderItem”,
place-order activity

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

什么是模型-以Java程序为例

- Level 0模型
  - 程序运行起来创建的诸多**对象**及其**连接**所形成的graph
  - 如Network-Group-Person对象
- Level 1模型
  - 代码中定义和使用的类、接口及其**连接**
  - MyNetwork, MyGroup, MyPerson
- Level2模型
  - Java语言内置的类型定义机制：关键词和句型
- Level 3模型?

类模型的理解

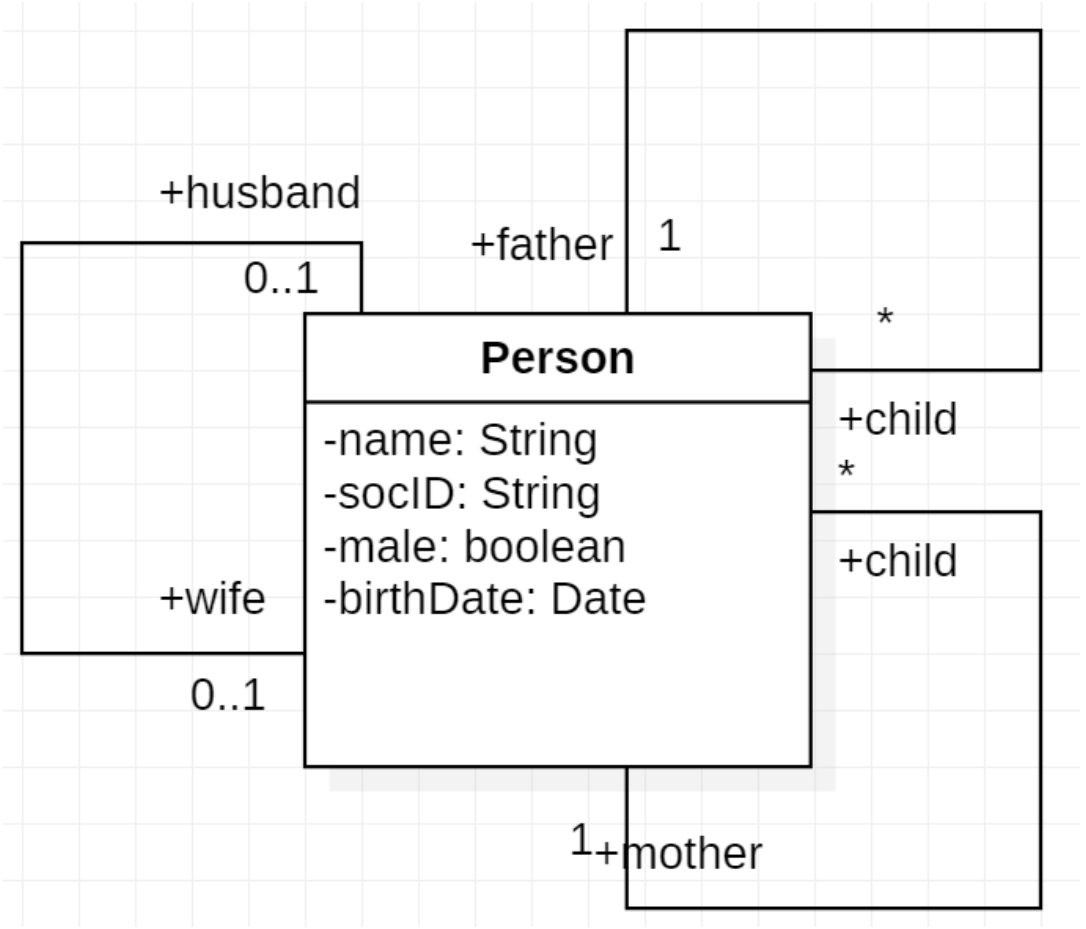
- 继承关系与接口实现关系的区别
  - 继承在语义上意味着获得父类所**拥有**的内容
  - 接口实现在语义上意味着实现了接口所定义的操作
- UML在类(UMLClass)和接口(UMLInterface)之间有复杂的全组合继承关系
  - 类继承类，类继承接口
  - 接口继承接口，接口继承类
- Java则进行了限制
  - 类继承类，接口继承接口，互相不可交叉

类模型的理解

- 关联关系(Association)
  - 定义**两组对象**之间的关系
  - 双向关系：两个UMLAssociationEnd对象，地位相同
  - 导航(navigable)反映建模者意图，即一个对象是否需要另外一个对象的支持或服务
  - 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

类模型的理解

- 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?

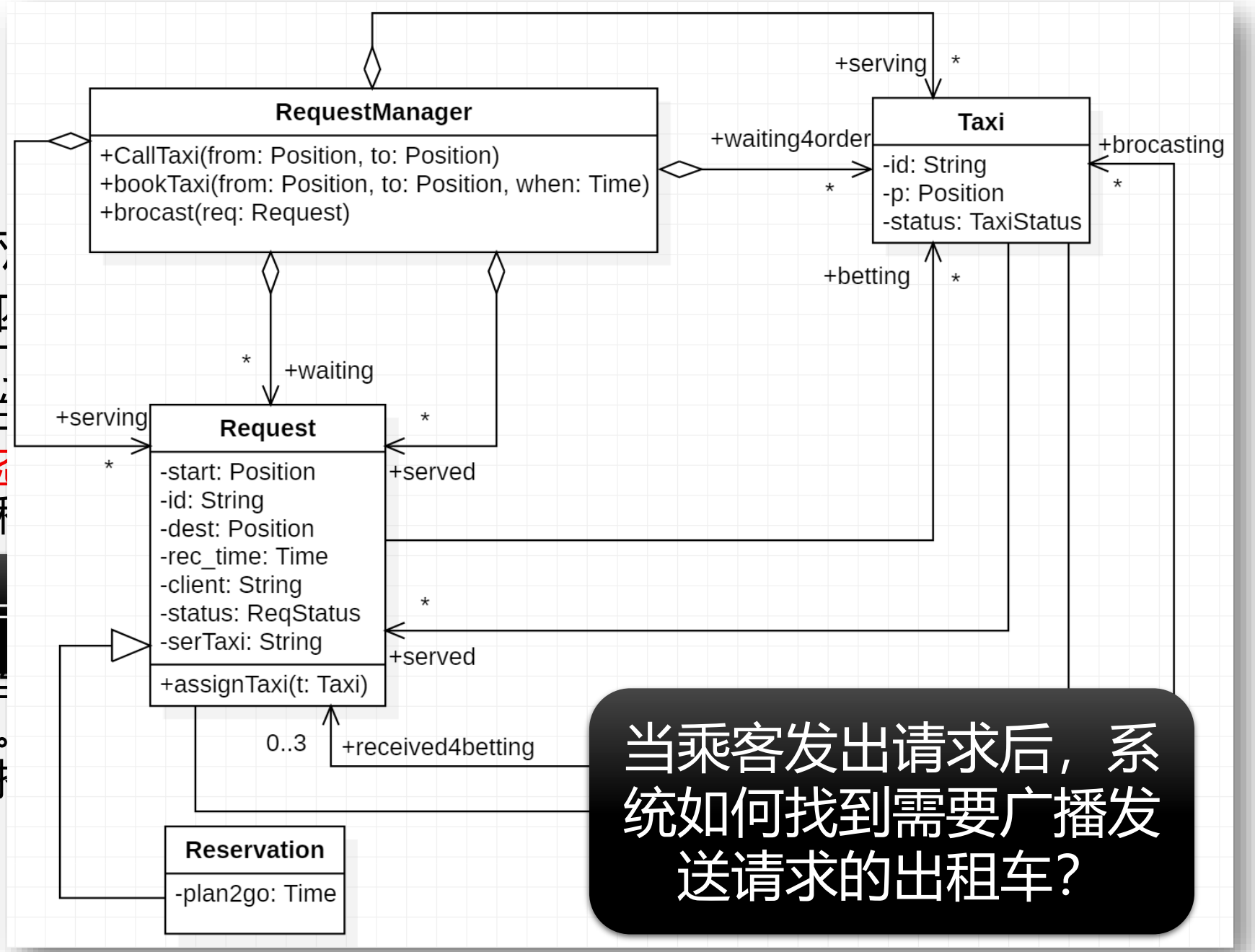
建模的核心

- 对诸多细节进行分析，提取共性结果，进行抽象
  - 设出租车的状态有四种：服务状态（把乘客送往目的地过程中所处的状态），接单状态（被派遣了乘客请求后，去接乘客过程中所处的状态），等待服务状态（无服务可接单的空车运行状态）和停止服务状态（无服务、不接单）。出租车在等待服务状态时可以获得派单，从而变成接单状态，一旦接到乘客，即变为服务状态。出租车只能从等待服务状态变为停止服务状态。
  - 系统一旦收到乘客的叫车请求，会设置一个抢单时间窗口，在时间窗口内系统向处于等待服务状态的出租车广播发送乘客的叫车请求。系统同时最多向一个出租车发送三个叫车请求。在抢单时间窗口关闭时，系统在抢单的出租车中，随机选择一辆派单。如果没有出租车抢单，则系统提示用户无出租车响应。一辆出租车可以对广播收到的请求进行抢单，但每辆车最多只能被选中并指派一单。

建模的核心

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**！



当乘客发出请求后，系统如何找到需要广播发送请求的出租车？

建模的核心

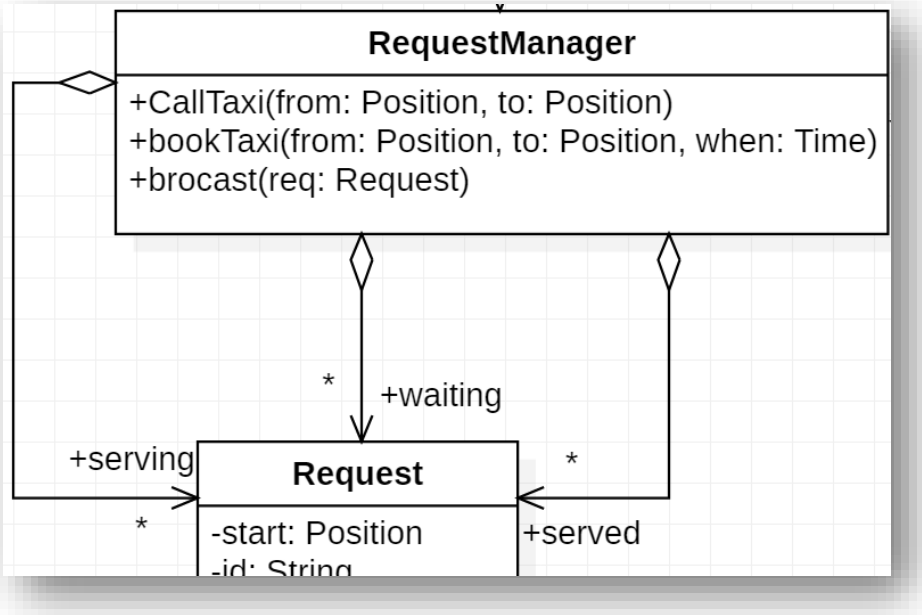
- 类图是UML建模的核心和基础
- 广泛采用的“入门级”方法：名词识别法
  - 具有明确的概念内涵和相应数据内容的名词
- 容易出现混淆的概念
  - 类-名词：这类名词往往蕴含着多维数据，如请求、出租车等
  - 属性-名词：这类名词往往蕴含着单维数据(但可能多例)，如目的地点、出发时刻等
  - 关联角色：这类名词往往蕴含着组合层次和对象实例分类，比如抢单出租车、等待服务出租车
  - 控制策略/机制：这类名词往往是对一种动态控制机制或策略的概括描述，如优先级调度、抢单时间窗口等

建模的核心

- 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
    - 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

建模的核心

- 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
  - 不同的类往往关心的是这些实例对象的一个子集，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



建模的核心

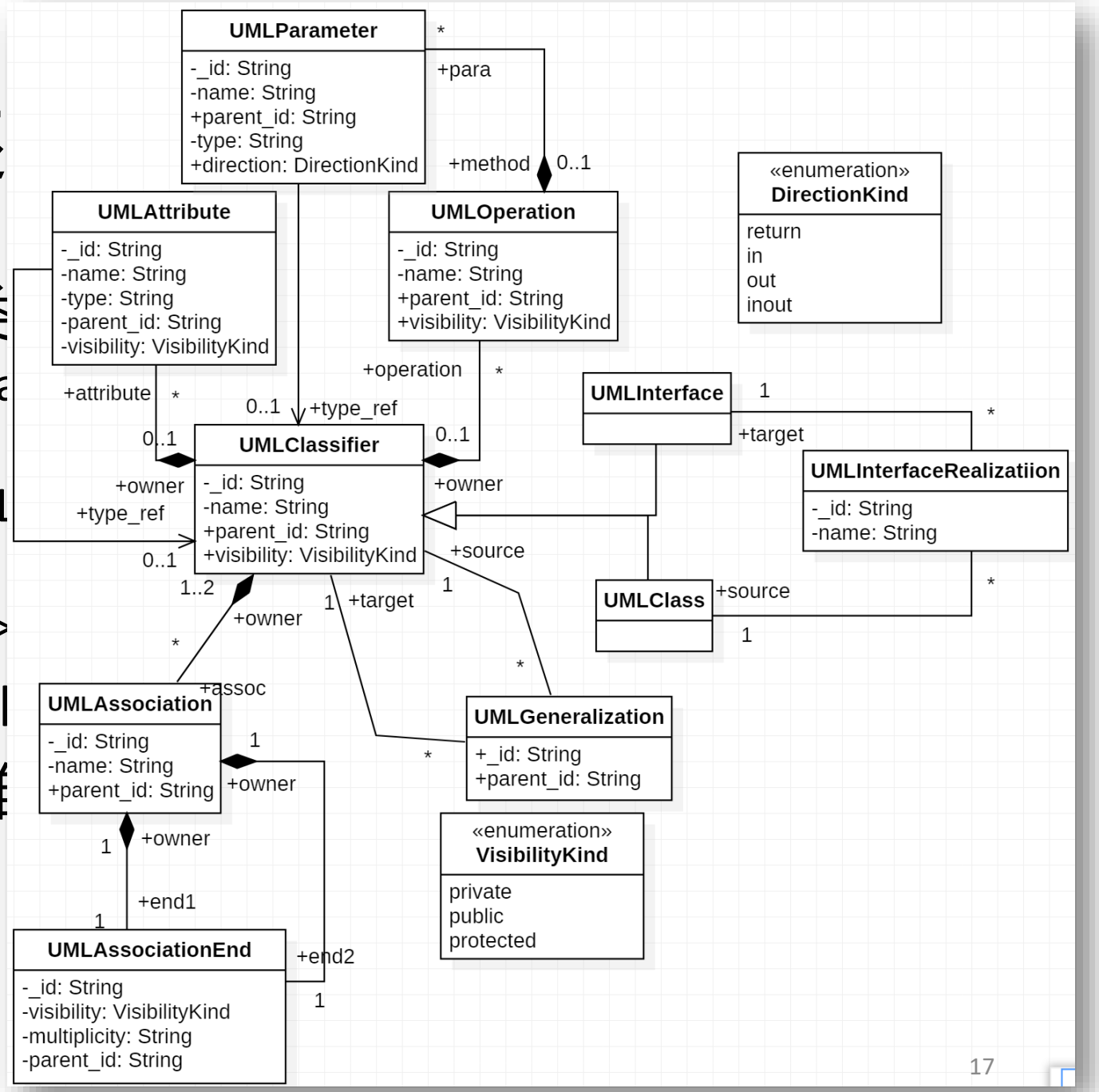
- 继承的识别与处理
  - 问题域描述中往往会出现多种形态的实体描述：请求、即时请求、预约请求，服务中请求、已服务请求等
  - 其中有些实体描述其实是按照角色的分类描述
  - 从继承角度，核心是分析不同的实体描述是否在数据上有显著不同
    - 请求，即时请求，预约请求
    - **抓住数据抽象这个本质！**
- 一个实体需要管理或记录哪些数据根本上来自于系统的领域需求
- 如果一个实体只关心它的行为，不关心数据，说明应该识别为一个接口，比识别为类带来更多灵活性

UML类图及其描述内容

- UML类图提供了一个描述类及其关系的视图
  - 顶层：UMLClass，UMLInterface，UMLAssociation，UMLGeneralization，UMLInterfaceRealization
  - 下一层：UMLAttribute，UMLOperation，UMLParameter，UMLAssociationEnd
  - 再下一层：{<property, value>}
- 可以从输入的{<property, value>}来构造相应的graph
- 在graph上可以进行查询和推理

UML类图及其描述

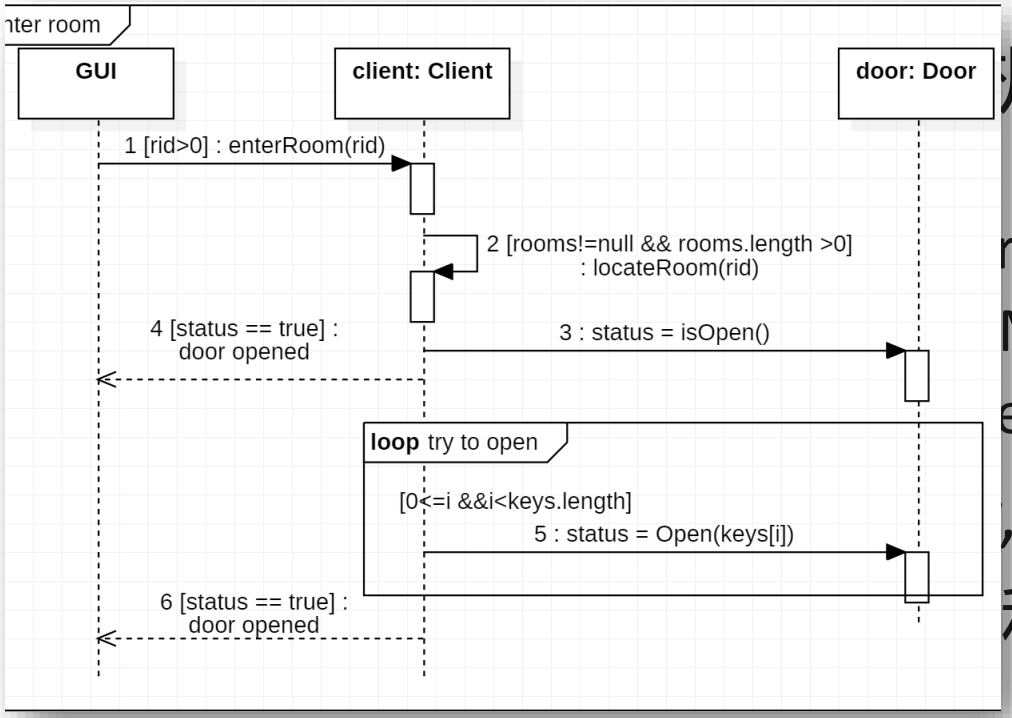
- UML类图提供了一个描述类
  - 顶层：UMLClass，UMLInterfa
    UMLInterfaceRealization
  - 下一层：UMLAttribute，UML
    UMLAssociationEnd
  - 再下一层：{<property, value>
- 可以从输入的{<property, val
- 在graph上可以进行查询和推

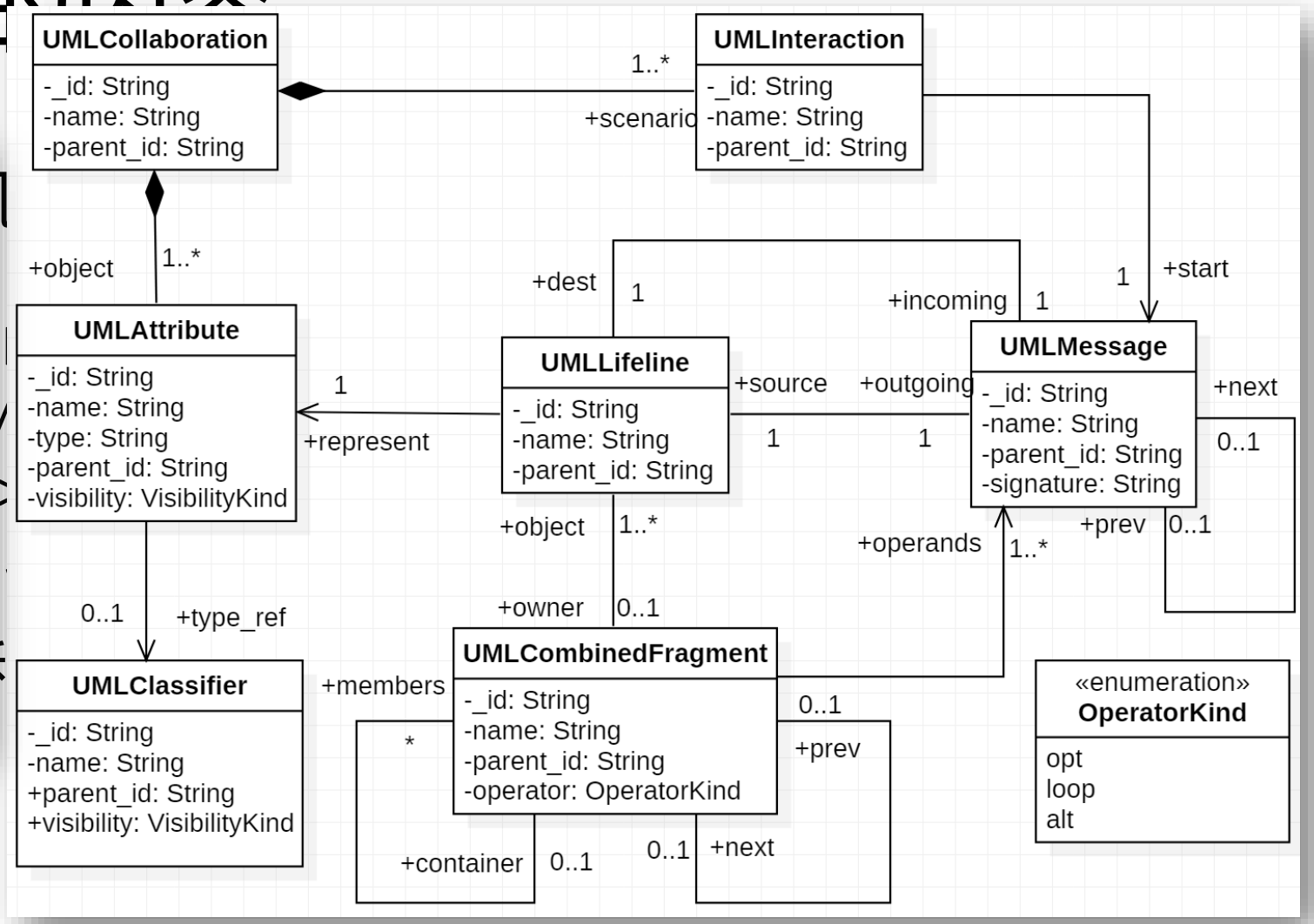


UML顺序图描述的内容

- 顺序图描述了基于消息机制的对象协作关系，应具有明确的协作主题
  - 顶层：协作对象(UMLAttribute)和交互模型(UMLInteraction)
  - 下一层：UMLLifeline，UMLMessage，UMLCombinedFragment
  - 最下层：{<property, value>}
- 可以从输入的{<property, value>}来构造相应的graph
- 可以在graph上进行对象和消息的相关推理分析

UML顺序图描述的内容



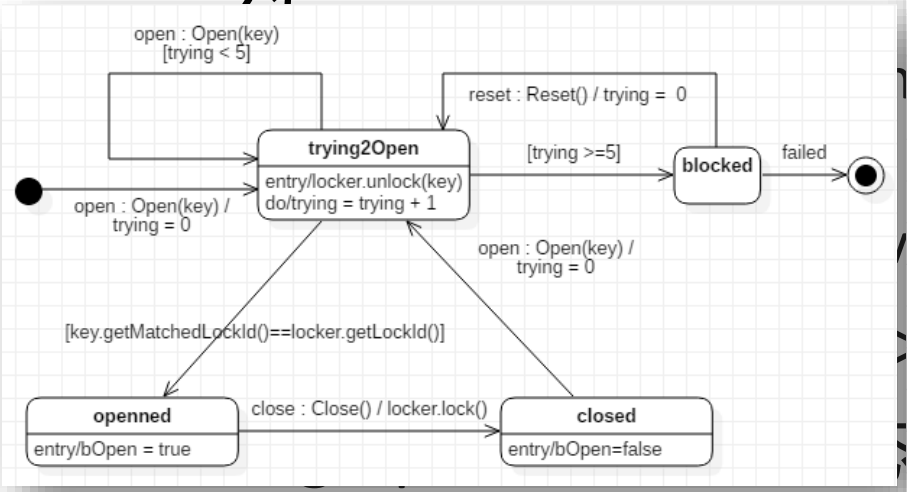


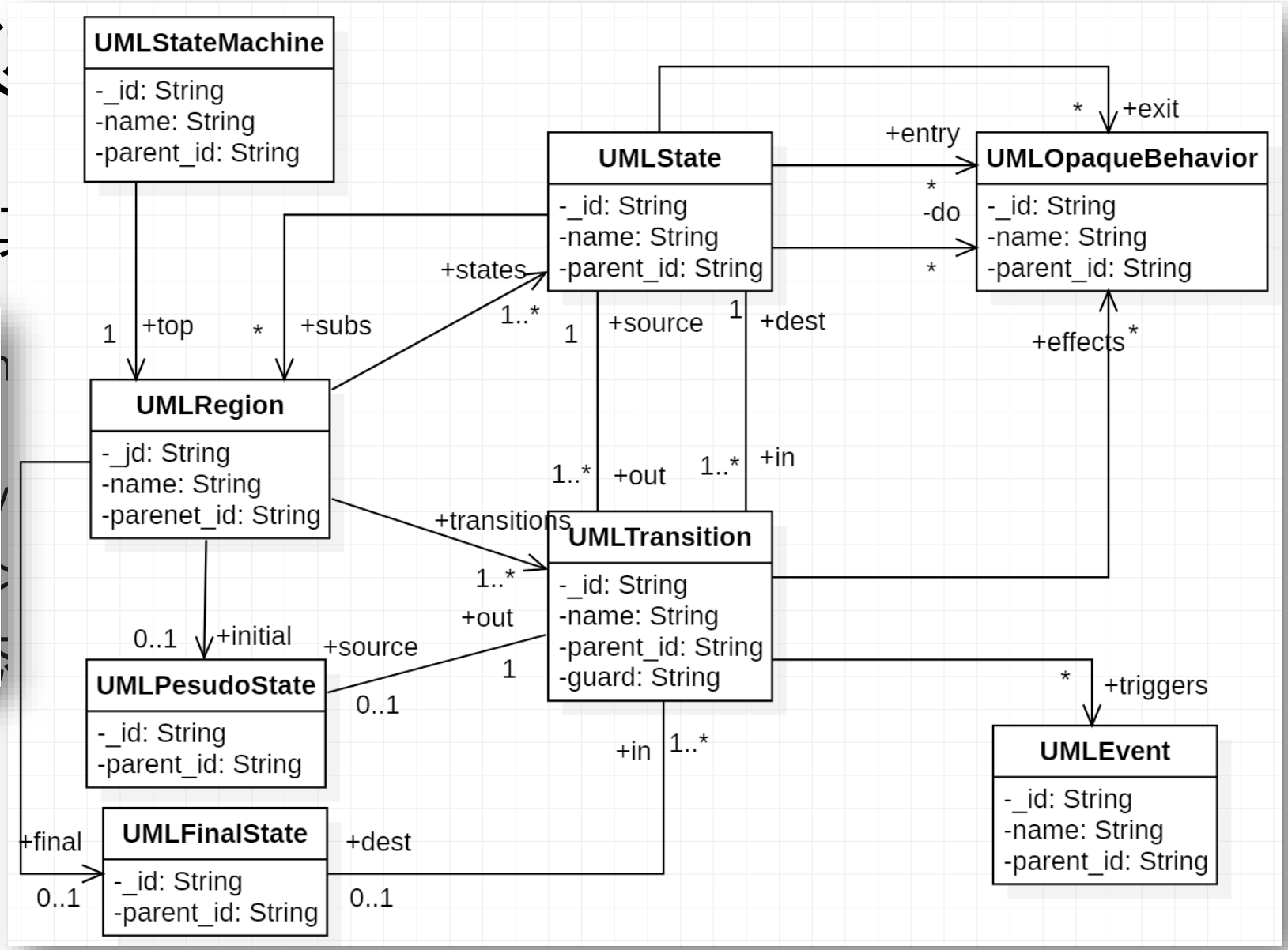
UML状态图描述的内容

- 状态图描述了状态及其关系，一般用来描述一个特定类/组件的行为
  - 顶层：UMLStateMachine，UMLRegion
  - 下一层：UMLState, UMLTransition, UMLEvent, UMLOpaqueBehavior
  - 最下层：{<property, value>}
- 依据{<property, value>}可以构造出相应的graph
- 在graph可以进行状态和迁移行为的推理分析

UML状态图描述

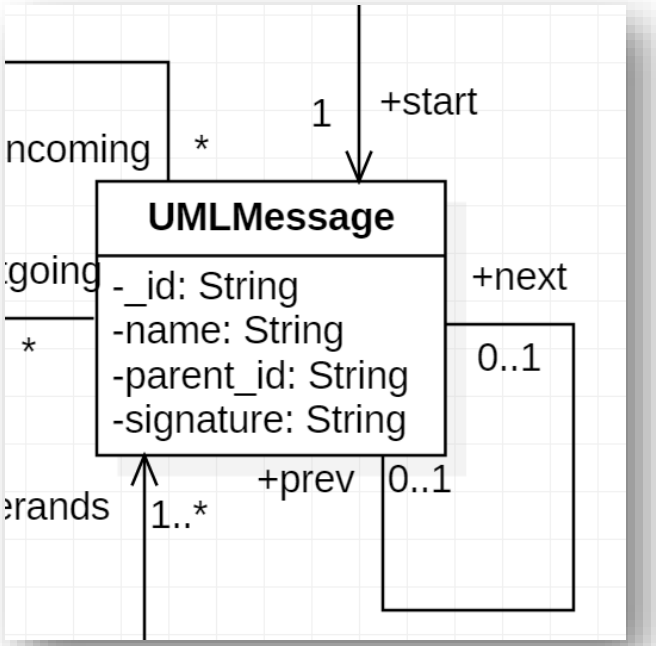
- 状态图描述了状态及其为

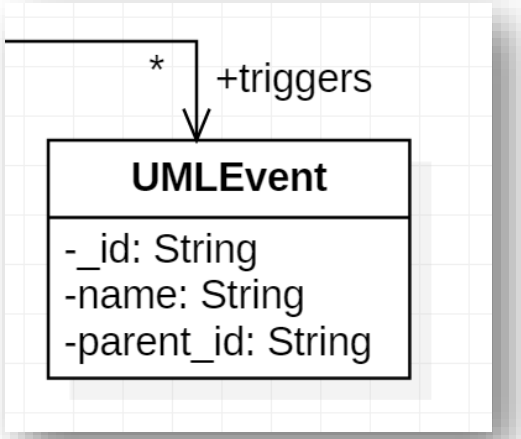




讨论

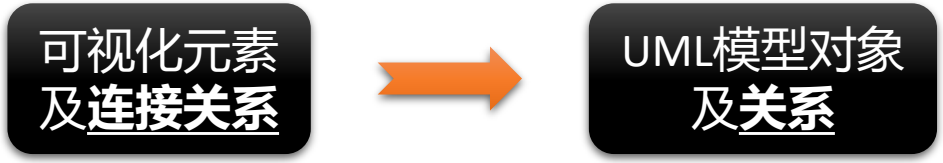
- 相比较于UML语言定义，前面所述的UMLMessage，UMLEvent等在属性数据上都做了很大的简化处理
  - 既可以支撑本单元的作业设计实现，又便于理解
- 现在有如下要求
  - UMLMessage本质上是调用或请求一个对象实例（UMLAttribute）运行时（UMLLifeline）的方法，我们要求这个消息一定要和类图中所定义的操作匹配上
  - UMLEvent本质上是用来描述触发一个对象状态迁移的事件，我们要求这个事件也必须和类图中所定义的操作匹配上
- 请根据这两个要求来改进UMLMessage和UMLEvent的属性表示





模型化设计

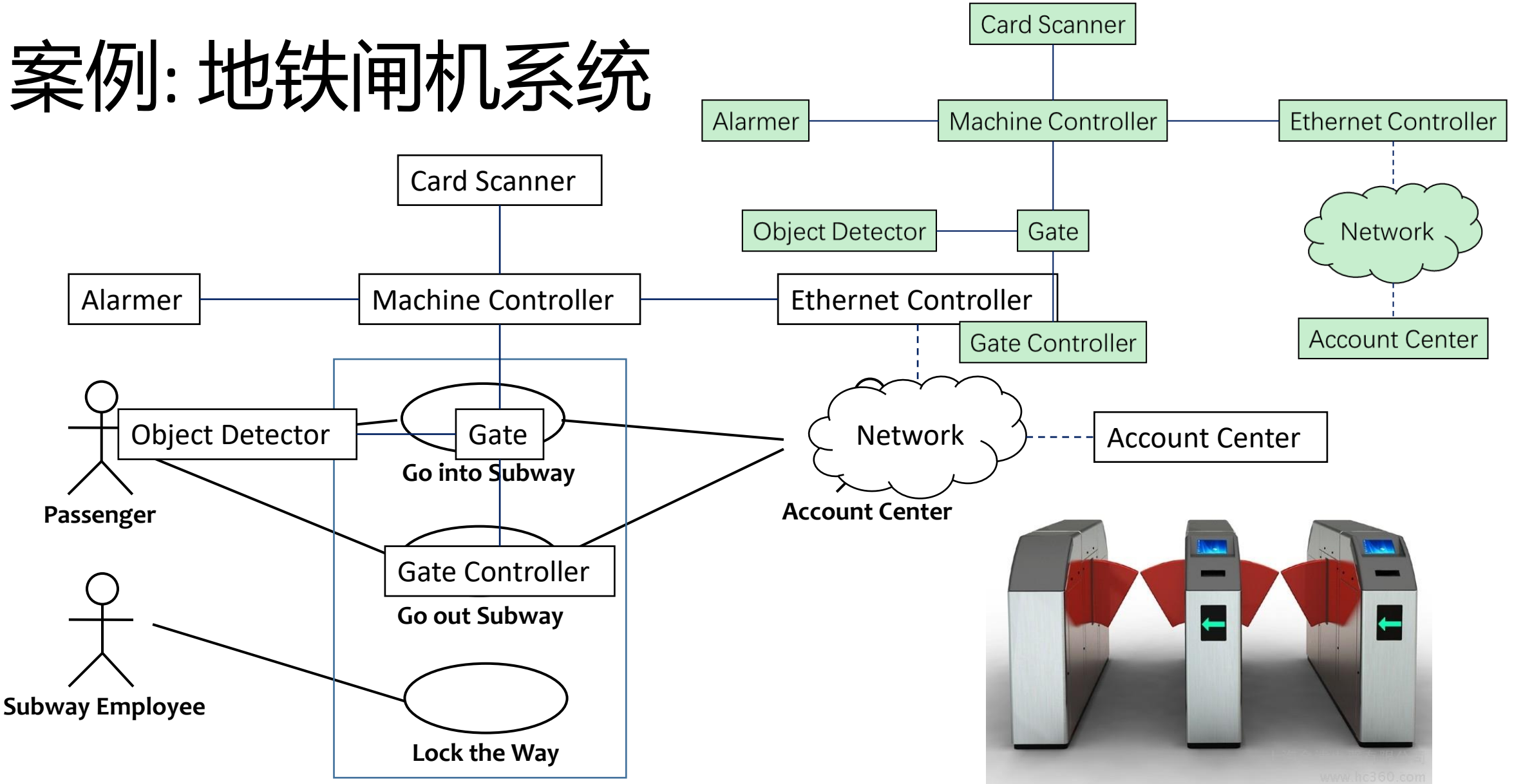
- 使用系统化的模型语言来表示设计结果，进而开展设计思考
  - UML
- UML采用了视图与模型相分离的设计
  - 在提供的diagram中表达相应的元素和关系
  - 建模工具维护UML模型
- UML提供了四种模型视图
  - 功能视图
  - 结构视图
  - 行为视图
  - 部署视图



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

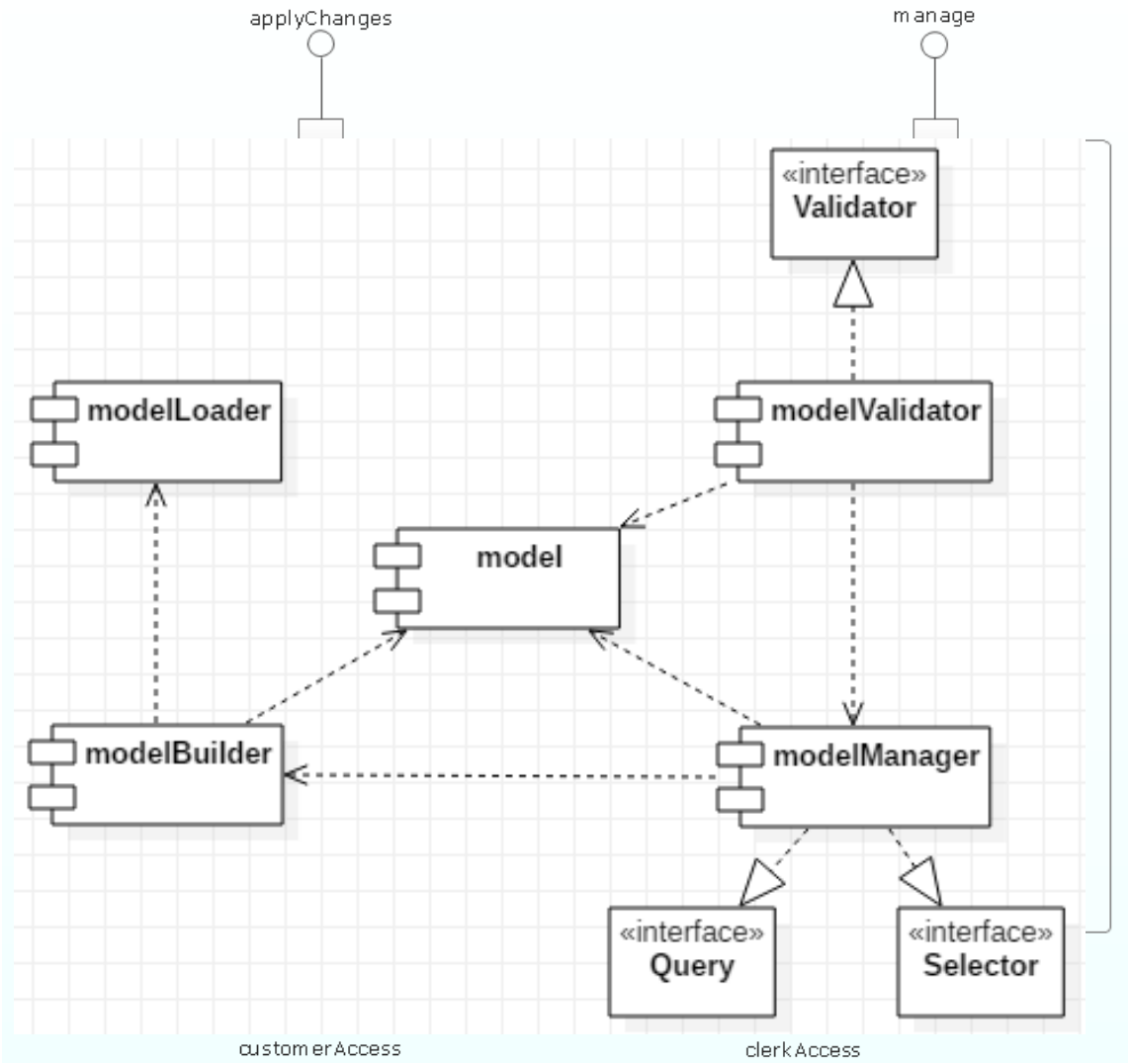
- 我的系统为什么需要这个类?
- 从与用户交互功能场景角度
  - 边界类
  - 实体类
  - 控制类
- 从数据封装与处理角度
  - 映射到功能需求中的数据项
  - 类中所封装数据项之间的聚合特性
- 从层次关系角度
  - 容器类
  - 控制策略类
  - 归一化类/接口

Case分析

- 本单元作业的用户交互功能
  - 模型对象查询(query)
    - 存在型查询：某个类中是否有某个方法/属性
    - 数量型查询：某个类中有多少个方法/属性
  - 基于关系的模型对象检索(selector)
    - this, {o|this→o}
    - this, {o|o→this}
  - 模型对象关系验证(validator)
    - validate***:boolean
    - validate(objSet, constraint):boolean
- 涉及容器类对象和对象连接关系
  - 设计相应的接口
  - 让相关的类实现这个接口

模型化设计

- 组件图支持的元素及其关系表达
  - 组件：提供相对完整功能的设计单位
  - 端口：组件对外的交互界面
  - 接口：组件对外的交互能力
  - 组件与组件
    - 依赖关系
  - 组件与接口
    - 提供的接口(provided interface)
    - 要求提供的接口(required interface)
  - 端口与接口
    - 通过端口提供接口操作



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
- AADL(Architecture Analysis & Design Language)是专门用于嵌入式分布式系统的架构设计模型语言
  - 高层次架构设计
  - www.aadl.info
- 本课程涉及的架构设计关注于软件模块层次的设计

模型化设计

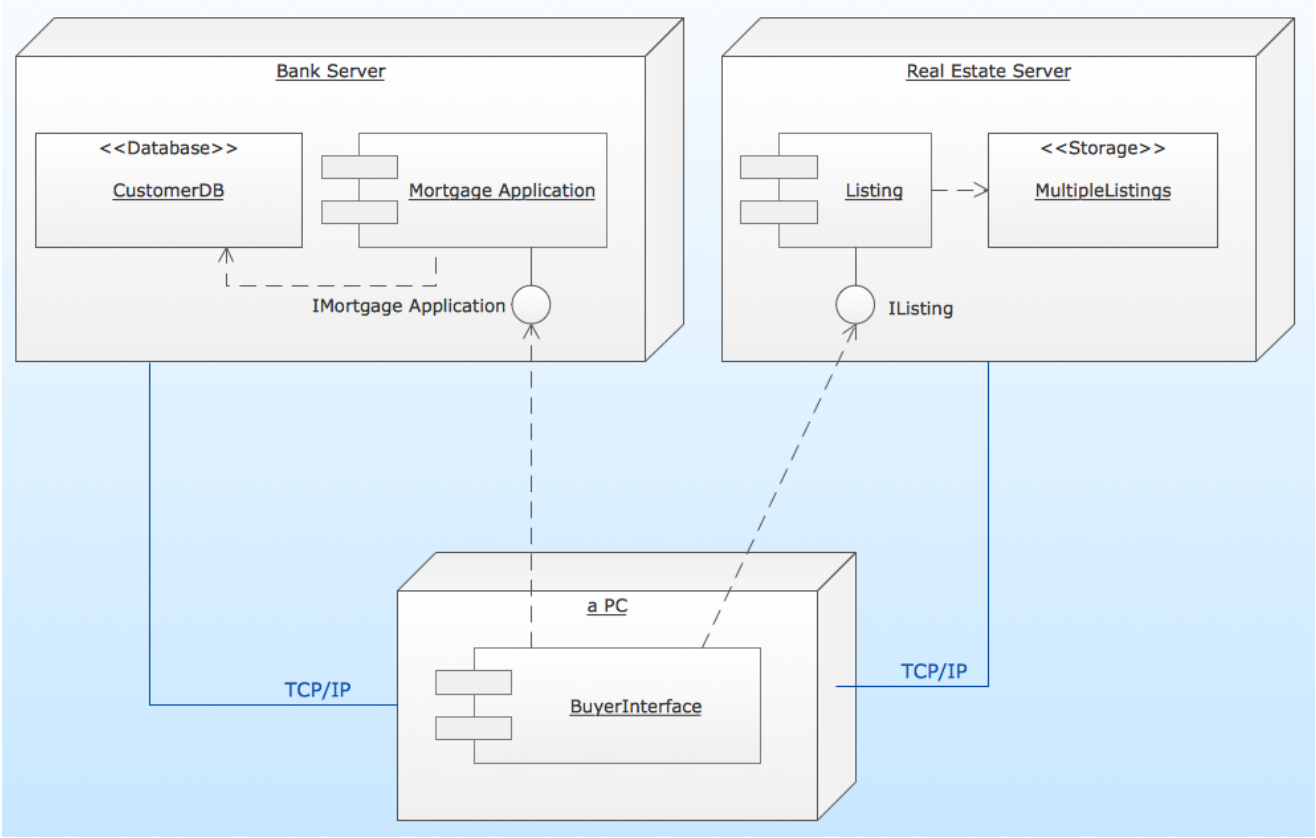
- 行为视图
  - 顺序图
  - 状态图
- 顺序图围绕一个功能场景(如use case)，从**对象交互**角度给出相应use case的设计方案
  - **确定哪些对象参与交互**
  - 识别类应提供的操作
  - 识别类之间的关联关系
- 顺序图采用消息交互机制
  - 消息与对象操作相关联

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
  - 在运行时会随输入动态改变其状态
- 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
  - 线程调度(线程对象的状态空间)
    - 根据线程对象的状态来调度请求来更改其状态
  - 电梯调度(电梯对象的状态空间)
    - 根据电梯对象的状态来分配乘客请求并改变其状态
- 不同状态之间在属性取值上必须严格分离

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
  - 按数据或行为提取抽象，形成类型与实例的层次
- 分类
  - 概念分类、对象分类
  - 分类是处理复杂问题的核心手段
- 层次
  - 按数据管理层次
- 分段
  - 按业务处理时间线，典型的分治法
- 映射
  - 按数据间的因果或者关联关系

抽象、分类、层次会在结构上**形成层次**

分段会在处理流程上**形成层次**

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
  - UML把类和接口抽象为UMLClassifier
- 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

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

模型化设计小结

- 一种设计方法
  - 建立模型为手段和目标
  - 模型提供多个视图
  - 分层和协同仍然是有效的方法
- 使用模型化设计需注意
  - 针对问题域思考目标“模型”是什么
  - 针对目标模型的内容思考需要建立哪几种视图
  - 使用UML（推荐）来按照视图建立相应的模型逻辑图

作业解析

- 增加对顺序图和状态图的解析
- 增加关于顺序图和状态图的查询命令
- 是否需要重构?
  - 统一查询接口
  - 统一模型对象管理
  - 建立跨层次的对象关系/映射

作业设计

- 顺序图查询命令
  - 对于一个UMLInteraction，有多少个participants?
  - 关注特定类型消息的查询
    - 创建消息、Found消息、Lost消息
    - 基本查询结构：UMLAttribute →UMLLifeline →message (src. dst)
- 状态图查询命令
  - 给定的statemachine(name)，context的name？拥有多少个state，多少个transition?
    - 包括初始状态和终止状态
  - 给定状态机模型和其中的一个状态，判断其是否是关键状态
    - 何为关键状态？---图论
  - 给定状态机模型中和其中两个状态，引起状态迁移的所有触发事件