第四部分 软件设计

- 4.1 软件工程开发方法与软件设计
- 4.2 体系结构设计
- 4.3 类/数据建模与设计
- 4.4 行为建模与设计
- 4.5 物理建模与设计

4.4 行为建模与设计

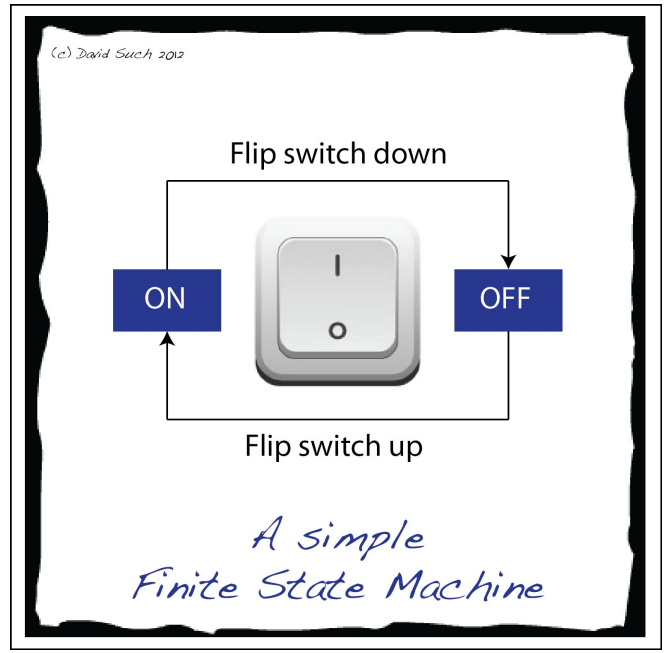
- 状态图 (Statechart Diagram)
- 顺序图 (Sequence Diagram)
- 协作图 (Collaboration Diagram)
- 活动图 (Activity Diagram)

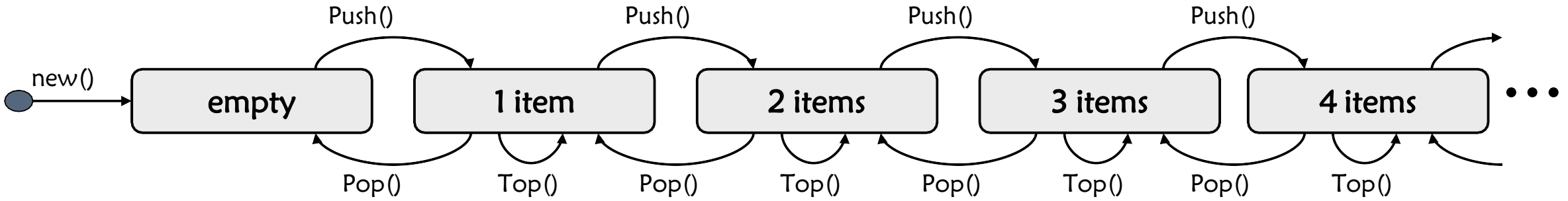
状态图（Statechart Diagram）

- 状态图(Statechart Diagram)描述了一个特定对象的所有可能状态以及由于各种事件的发生而引起的状态之间的转移。
- UML的状态图
  - 主要用于建立类的一个对象在其生存期间的动态行为，表现一个对象所经历的状态序列，引起状态转移的事件(Event)，以及因状态转移而伴随的动作(Action)。
- 状态图适合于描述跨越多个用例的单个对象的行为，而不适合描述多个对象之间的行为协作，因此，常常将状态图与其它技术组合使用。

“状态”建模

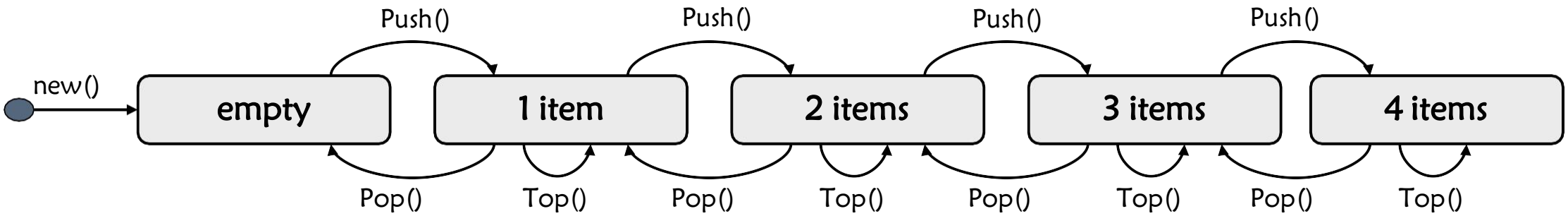
- 所有的对象都有“状态”
  - 对象存在或者不存在
    - 对象不存在也是一种状态
  - 如果对象存在，则具有相应表示其属性的值
  - 每一种状态表示一种可能的状态赋值
- 例如：栈





这个状态模型表示什么？

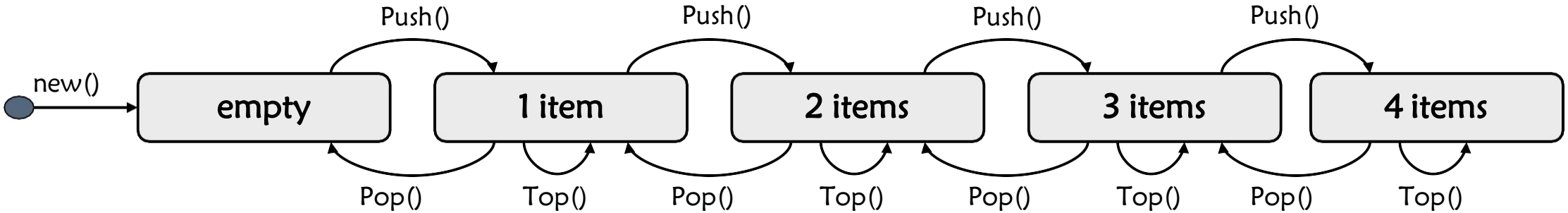
- 有限数量的状态 (所有的属性取值为有限的范围)
  - 例如，一个最大容量为4的栈



- 模型可以表示动作序列（状态变化）
  - 例如. new();Push();Push();Top();Pop();Push()…
  - 例如 new();Push();Pop();Push();Pop()…

这个状态模型表示什么？

- 有限数量的状态 (所有的属性取值为有限的范围)
  - 例如，一个最大容量为4的栈



- 模型可以过滤不符合的动作序列（状态变化）
  - 例如 Push()
  - 例如 new();Push();Pop(); Pop()…
  - 例如 new();Push(); Push(); Push(); Push(); Push();

状态空间

- 问1：一个人的一生，可以用年龄来建状态机吗？应该怎么建呢？

状态空间

- 对于大部分对象而言，状态空间是非常庞大的
  - 状态空间大小是对象每个属性取值空间的乘积加1（1表示初始状态）
    - 例如. 具有5个布尔值属性的对象有 2^5+1 个状态
      - 01011←每个位数都能选择是0还是1，最后加一个1表示是初始状态
    - 例如. 具有5个整数值属性的对象有 $(maxint)^5+1$ 个状态
      - 每个位数都能选择（maxint）个整数，总共5个位数
    - 例如. 具有5个实数值属性的对象具有？？个状态

状态的抽象表示

例如. 对于年龄, 我们经常选择以下的范围:

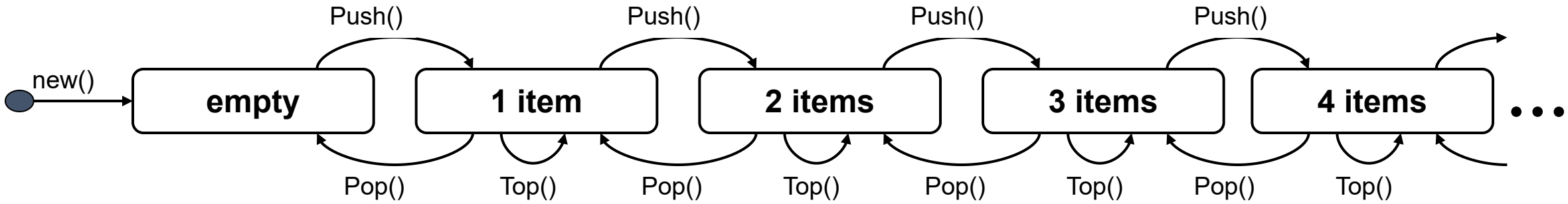
- age < 18; 18 ⩽ age ⩽ 65; age > 65

例如. 对于费用信息，我们更关注的约束划分为:

- cost ⩽ budget, cost=0, cost > budget, cost > (budget + 10%)

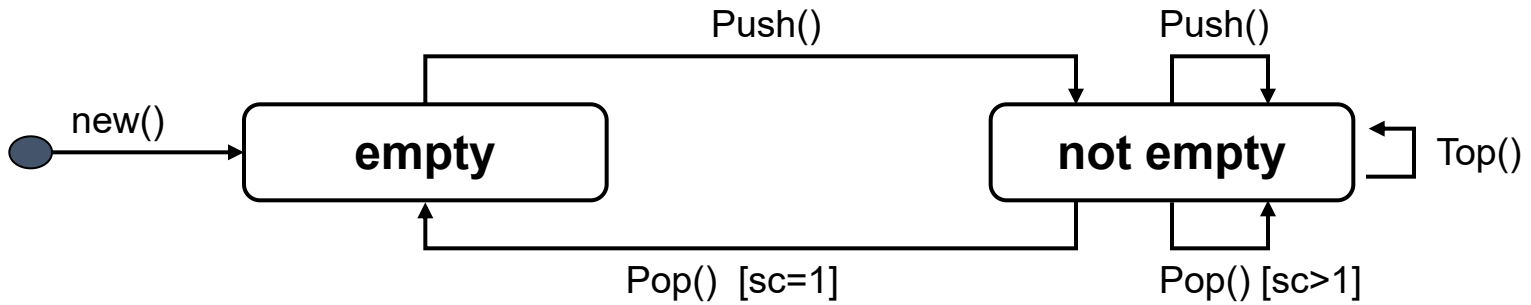
- 但往往状态空间中的局部更有探究的价值
  - 有一些状态是不可能出现的状态
  - 整数或实数值属性往往只在一定范围内取值
  - 通常，我们只关注特定约束下的对象及其行为

模型建立的过程——状态空间的分解



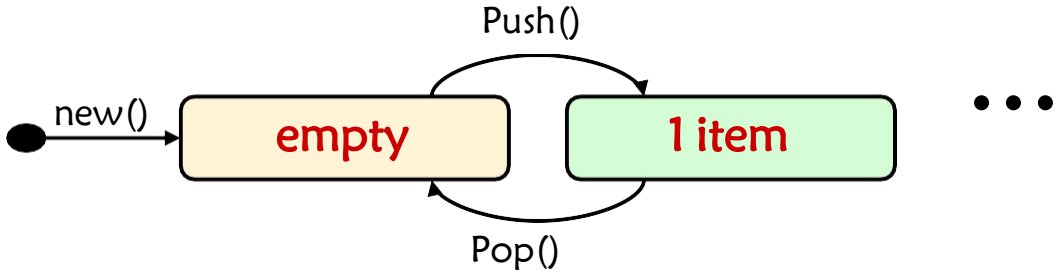
转换成为“空”状态

转换成为“非空“状态



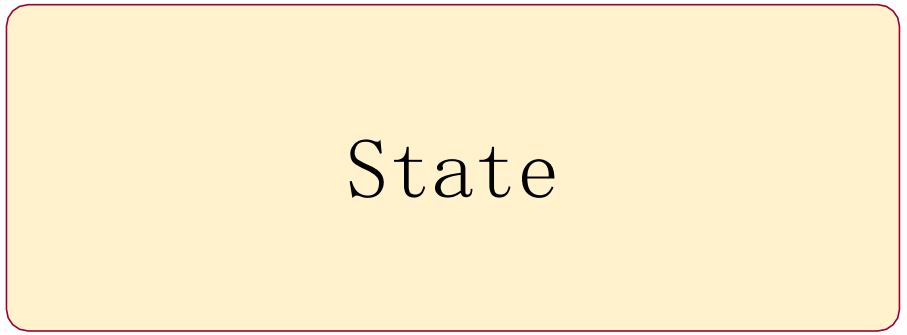
状态图建模

- 建模元素
  - 状态
  - 状态转移
  - 事件
- 特殊的状态
  - 初始状态、结束状态
  - 组合状态
  - 历史状态
- 状态图的绘制



状态图建模

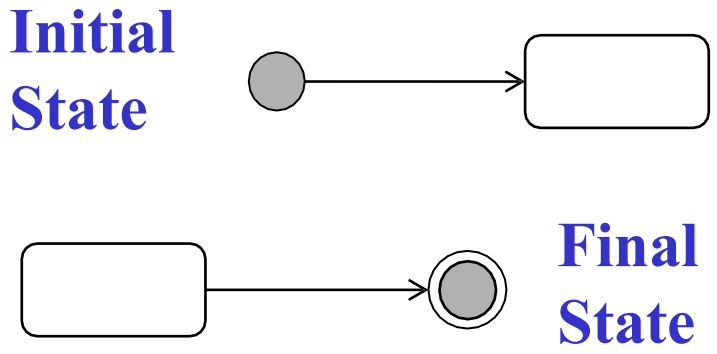
- 状态(State)
  - 一个对象在生命期中满足某些条件、执行一些行为或者等待一个事件时的存在条件。
- 实体对象都具有状态，状态是对象执行了一系列活动的结果。当某个事件发生后，对象的状态将发生变化。



圆角矩形
指对象执行某项活动或等待某个事件时的条件

状态种类

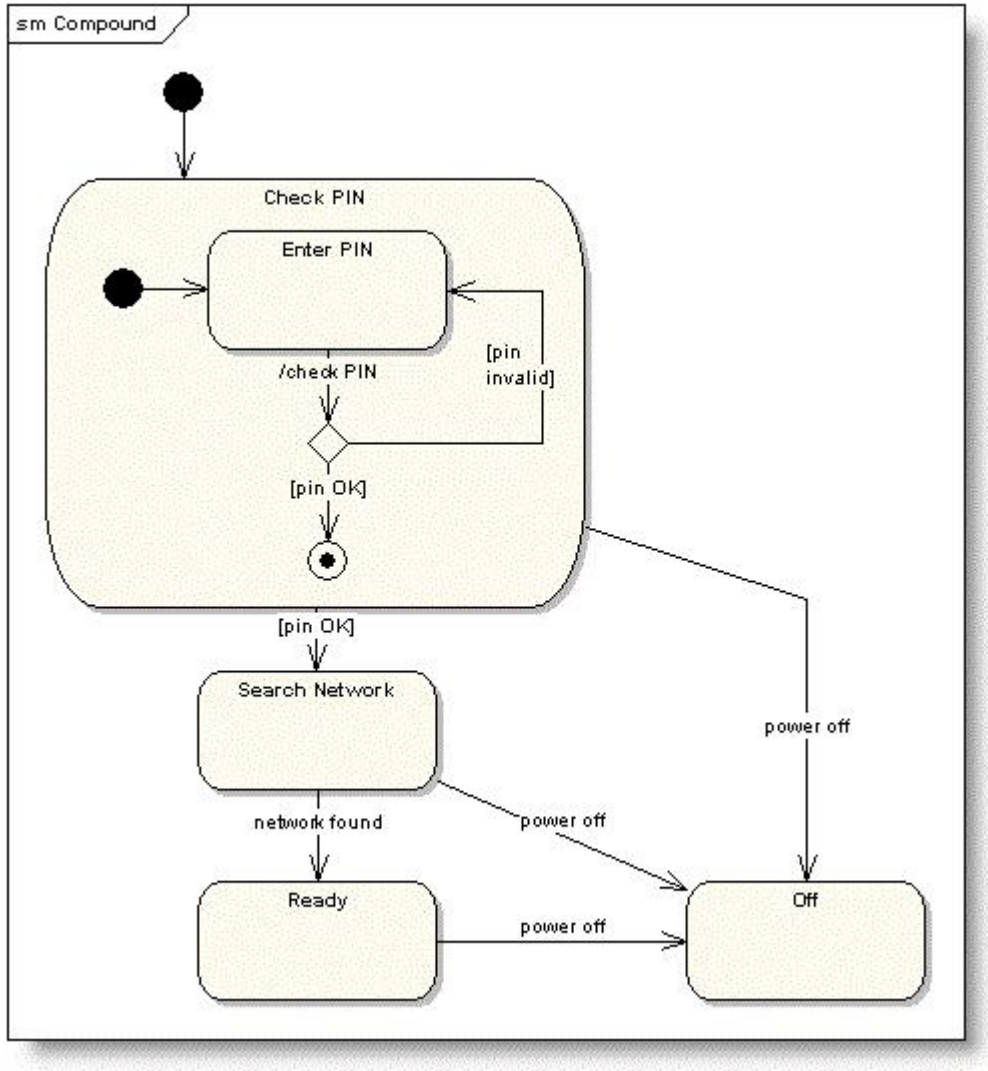
- 初态、终态



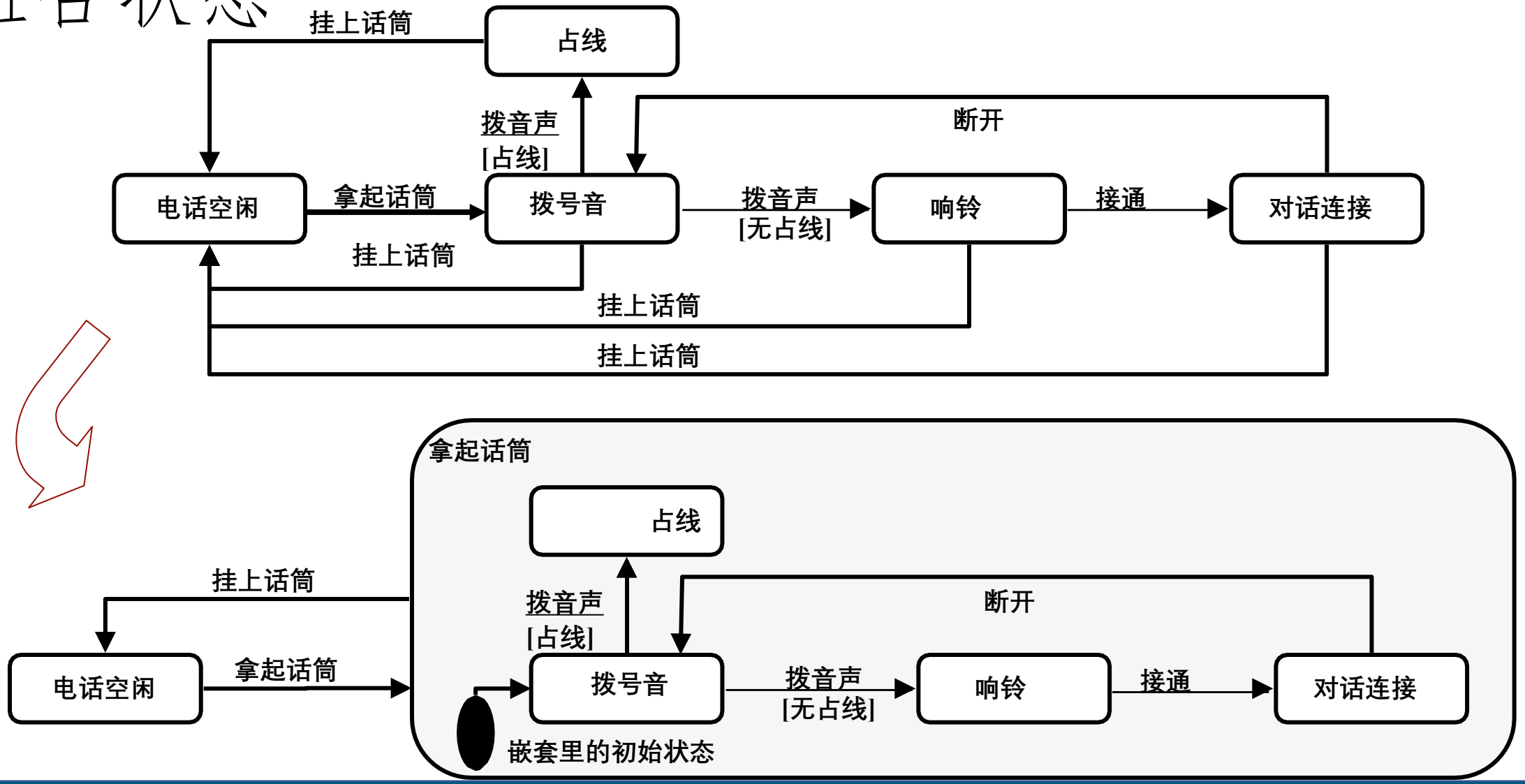
一个状态图只能有一个初始状态
一个状态图可以有多个结束状态

状态种类

- 组合状态：可以通过状态嵌套的方式简化图表
  - ✓ 一个组合状态可以包含一个或多个状态
  - ✓ 组合状态可以实现从不同抽象层次去体现状态图
  - ✓ 嵌套在另外一个状态中的状态称之为子状态（sub-state），一个含有子状态的状态被称作组合状态（Compound States）
  - ✓ 【Check PIN】是组合状态，【Enter PIN】是子状态。



组合状态

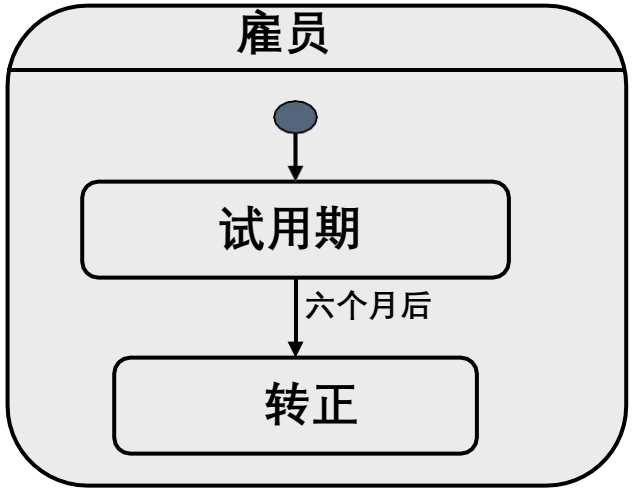

挂上话筒
占线
拨音声
[占线]
断开
电话空闲
拿起话筒
拨号音
拨音声
[无占线]
响铃
接通
对话连接
挂上话筒
挂上话筒
挂上话筒
拿起话筒
占线
挂上话筒
拨音声
[占线]
断开
电话空闲
拿起话筒
拨号音
拨音声
[无占线]
响铃
接通
对话连接
嵌套里的初始状态

组合状态

“OR”的组合状态

- 处于组合状态时只能满足其中一个子状态

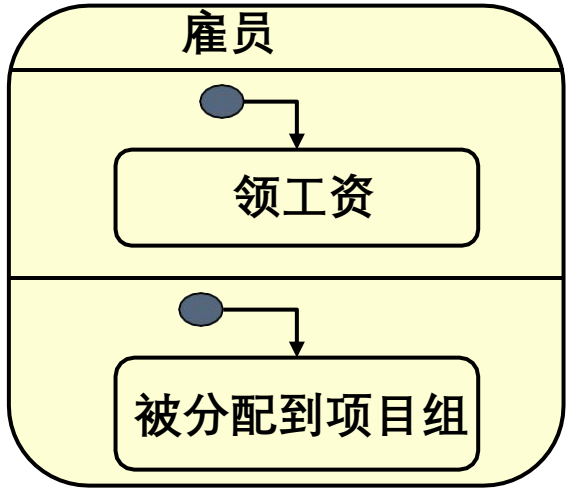
雇员要过了试用期才能转正



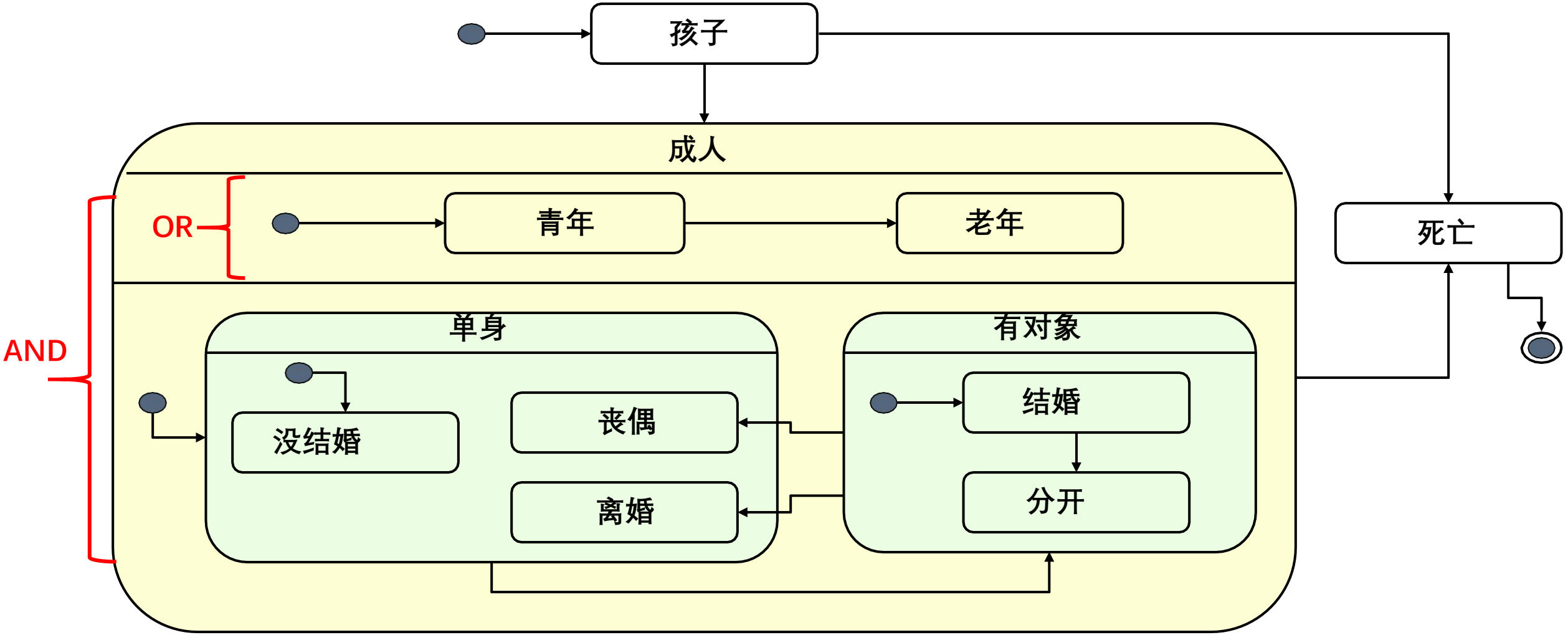
“AND”的组合状态(并发状态)

- 处于组合状态时，满足所有的子状态
- 通常，AND的子状态会进一步嵌套为OR的子状态

雇员既可以领工资，同时也可以被分配到项目组

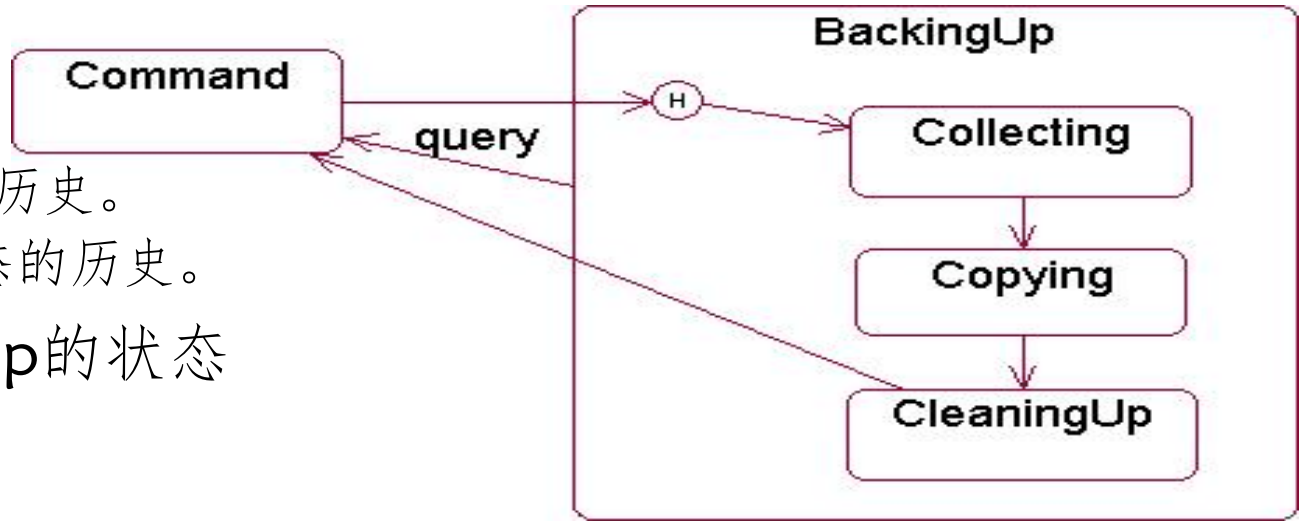


组合状态



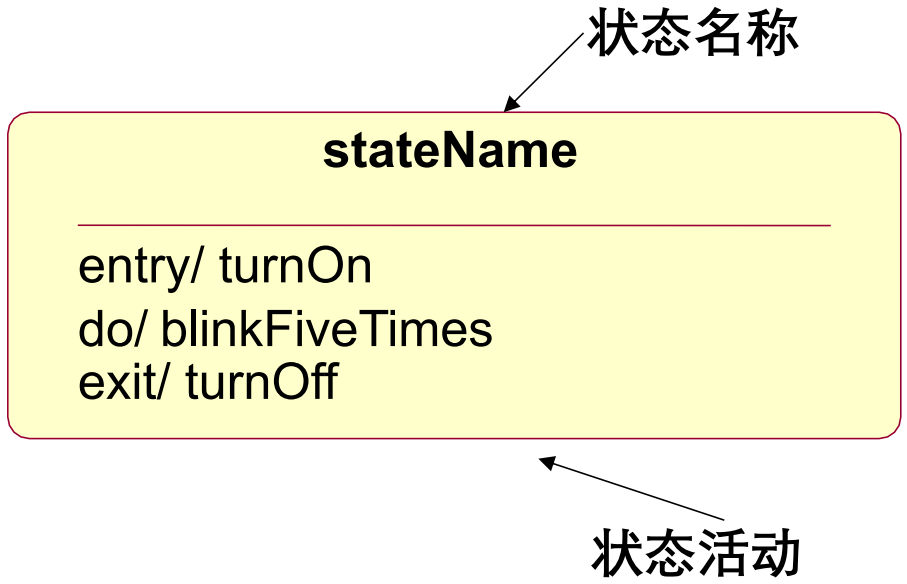
状态种类

- 历史状态
  - 使用历史状态，可以记住从组合状态中退出时所处的子状态，当再次进入组合状态时，可直接进入到这个子状态，而不是再次从组合状态的初态开始。
    - H和H*的区别：
      - H只记住最外层的组合状态的历史。
      - H*可记住任何深度的组合状态的历史。
  - 例：这里的H只记录Backing Up的状态



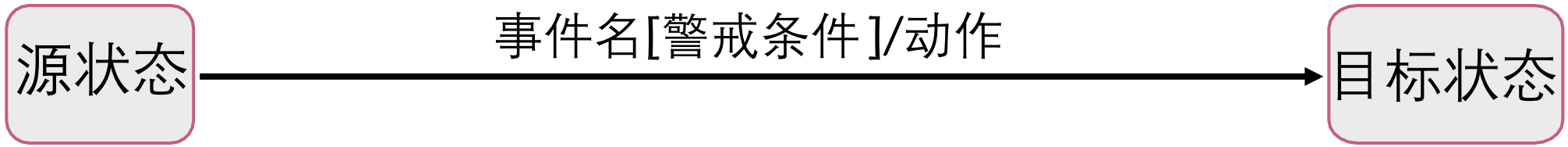
状态活动（activities）

- 状态相关的活动类型
  - do/activity
    - 只要处于这个状态，某个活动就会一直执行，直到离开这个状态
  - entry/action and exit/action
    - 当进入（/离开）某个状态时执行的动作
  - include/stateDiagramName
    - 调用另一个状态图，形成嵌套的状态图

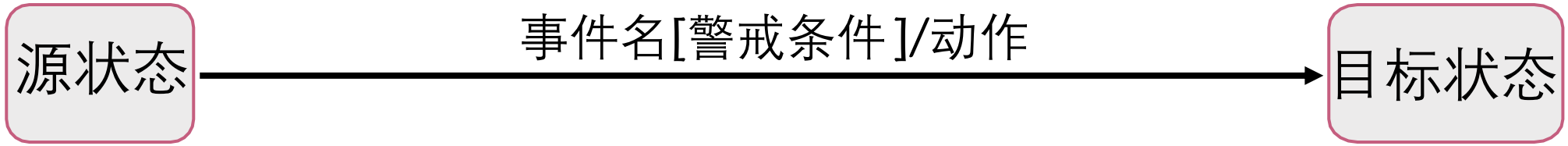


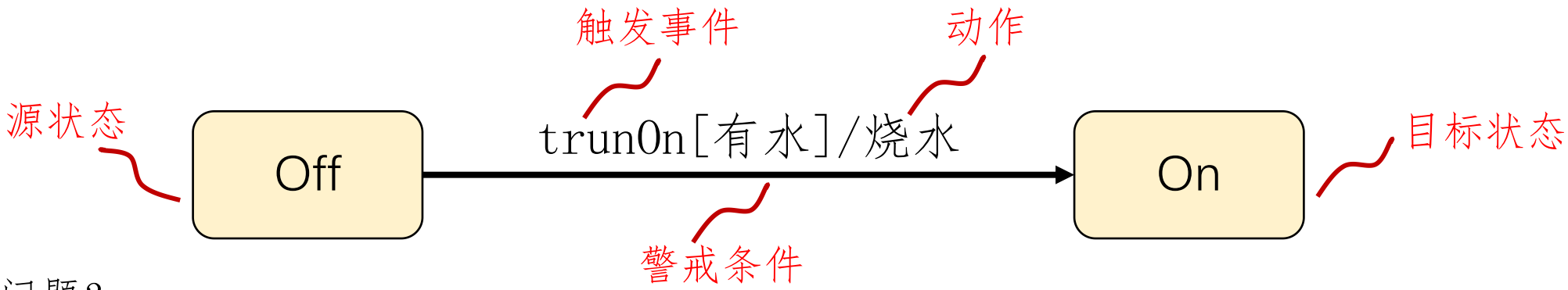
状态迁移(Transitions)

- 迁移包括五部分：
  - 转换是状态图的一个组成部分，表示一个状态到另一个状态的移动。
  - 状态之间的转移通常是由事件触发的，此时应在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。
- 警戒条件：
  - 一个true或false测试表明是否需要进行转换
  - 当事件发生时，只有在保护条件（警戒条件）为真时才发生转换



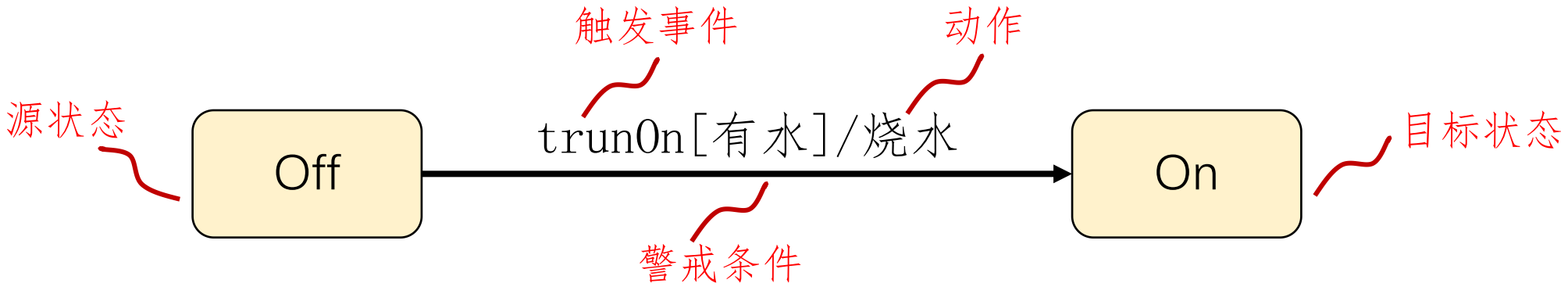
状态迁移(Transitions)

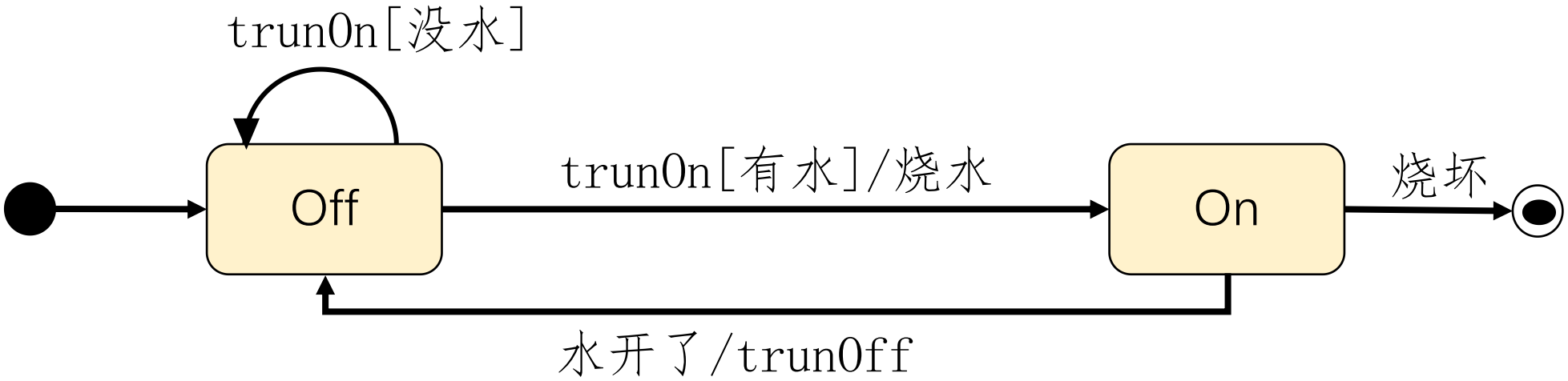




有什么问题？

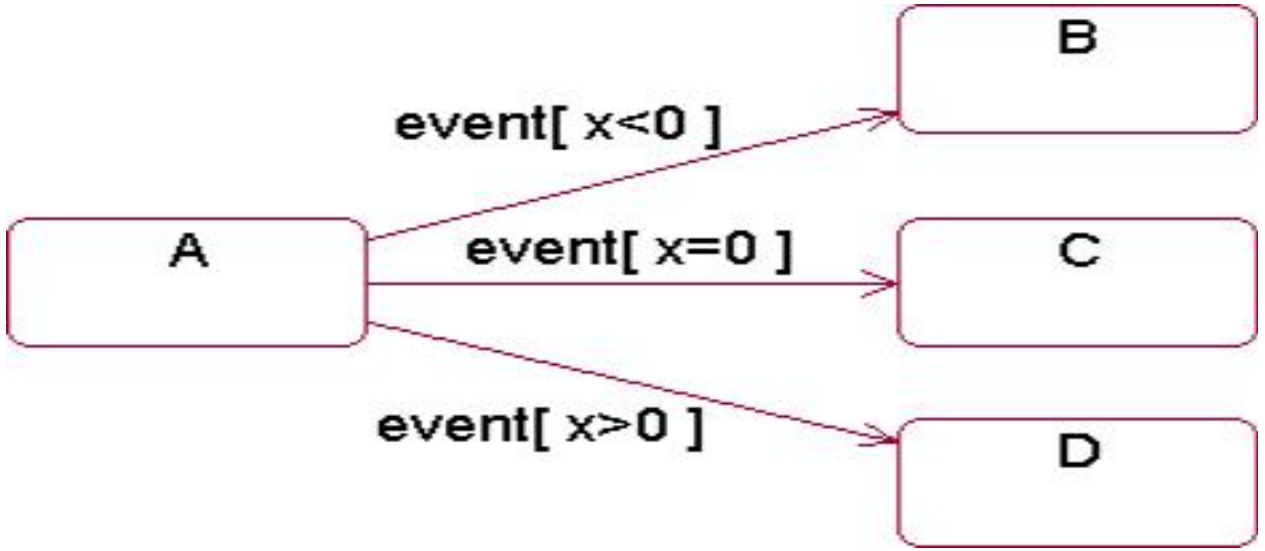
状态迁移(Transitions)





状态迁移(Transitions)

- 对于给定的状态，最终只能产生一个迁移，因此从相同的状态出来的、事件相同的几个迁移之间的条件应该是互斥的。

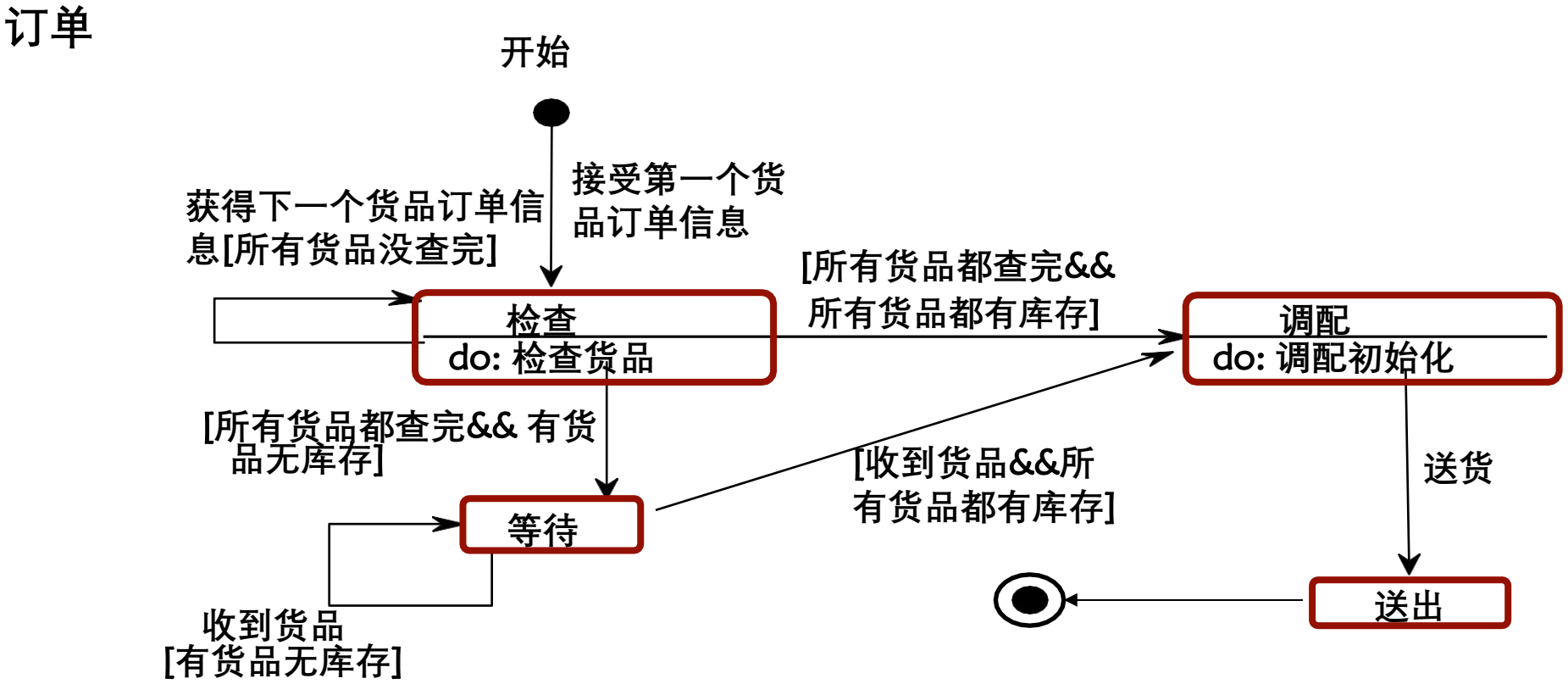


状态中的事件(Event)

- **事件(Event)**
  - An event is the specification of a noteworthy occurrence that has a location in time and space.
  - 事件是一件有意义、值得关注的现象。

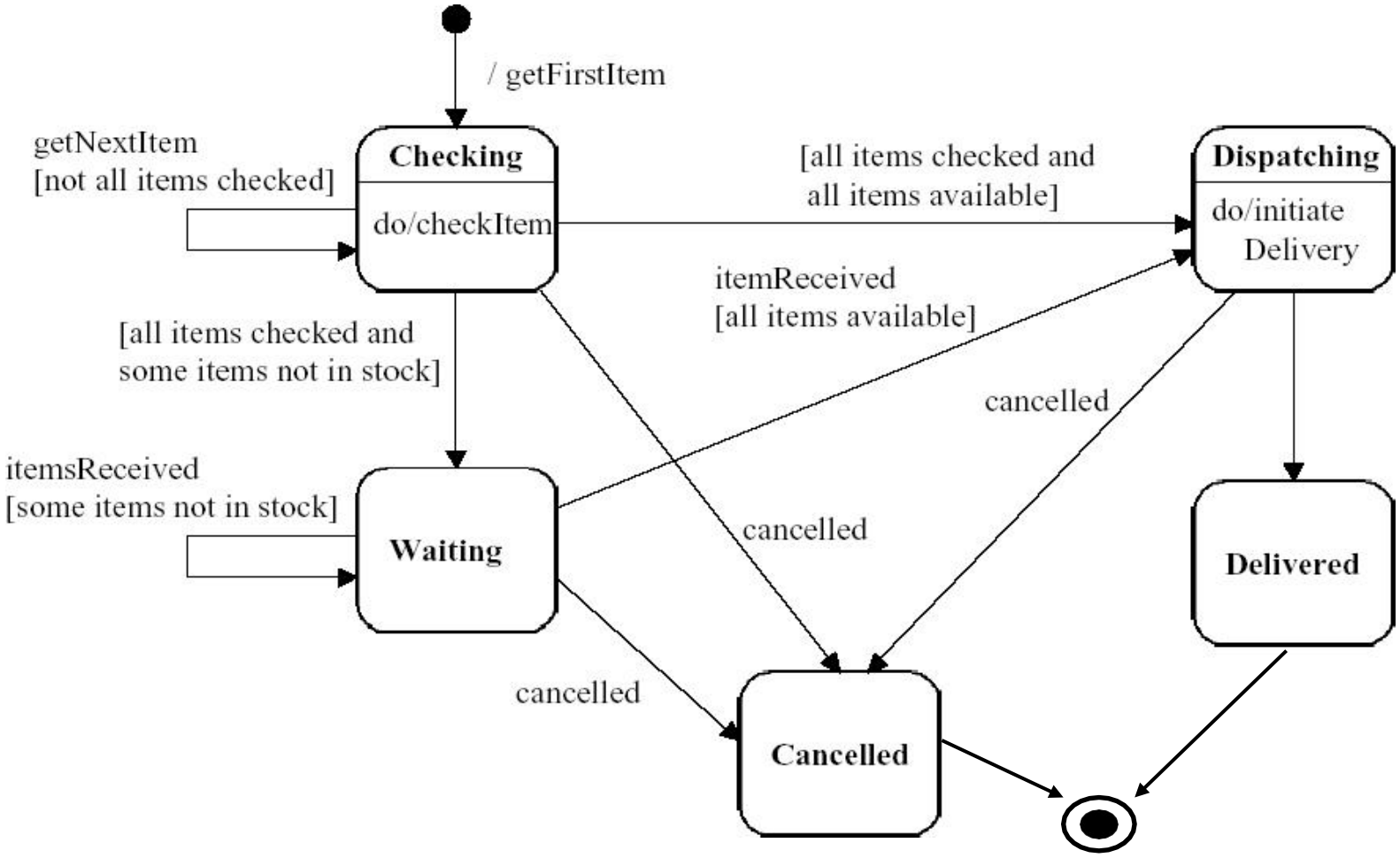
- **UML中事件分为四类**
  - Call Event (调用某个操作)
  - Change Event：when (temperature > 120)
  - Signal event (触发事件)
  - Time event：after 5 seconds

订单处理状态图

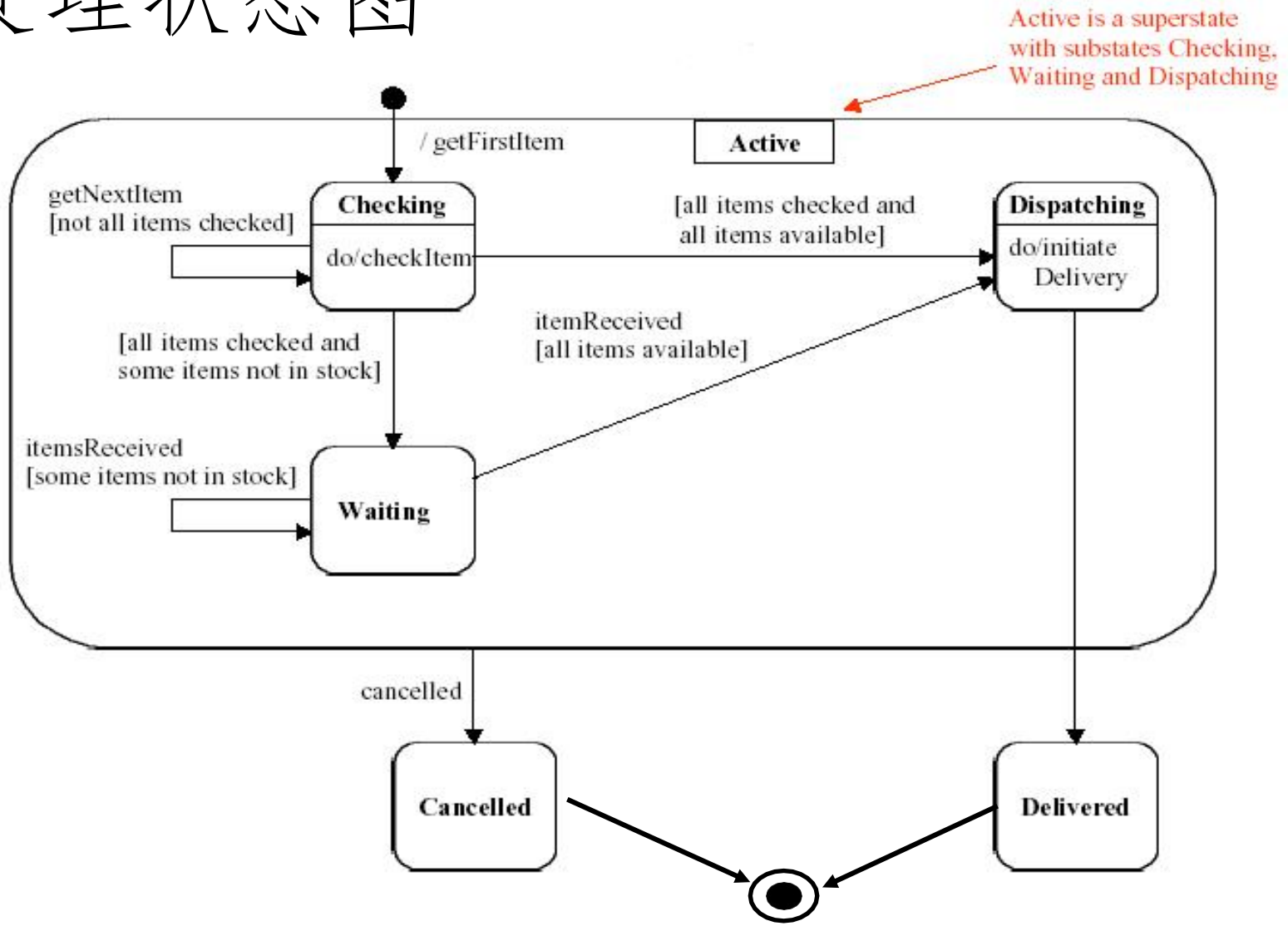


如果要加入一个cancel功能呢？

订单处理状态图


/ getFirstItem
getNextItem
[not all items checked]
Checking
do/checkItem
[all items checked and
all items available]
Dispatching
do/initiate
Delivery
itemReceived
[all items available]
[all items checked and
some items not in stock]
cancelled
itemsReceived
[some items not in stock]
Waiting
cancelled
Delivered
cancelled
Cancelled

订单处理状态图


Active is a superstate
with substates Checking,
Waiting and Dispatching
/ getFirstItem
Active
getNextItem
[not all items checked]
Checking
do/checkItem
[all items checked and
all items available]
Dispatching
do/initiate
Delivery
itemReceived
[all items available]
[all items checked and
some items not in stock]
itemsReceived
[some items not in stock]
Waiting
cancelled
Cancelled
Delivered

状态图建模风格

- 建模风格1：把初态放置在左上角；把终态放置在右下角
- 建模风格2：用过去式命名转移事件
- 建模风格3：警戒条件不要重叠
- 建模风格4：不要把警戒条件置于初始转移上

状态图的检查表

- 绘图风格
  - 每个状态的命名应该是唯一的，意义明确的
  - 只对行为复杂的状态使用组合状态建模
  - 不要在一个图中包含太多细节
  - 使用警戒条件时要特殊注意不要引入二义性
    - 状态图应该具有确定性（除非特殊原因）
- 下述情况不适宜使用状态图：
  - 当大部分的状态转移为“当这个状态完成时”
  - 有很多来自对象自身发出的触发事件

4.4 行为建模与设计

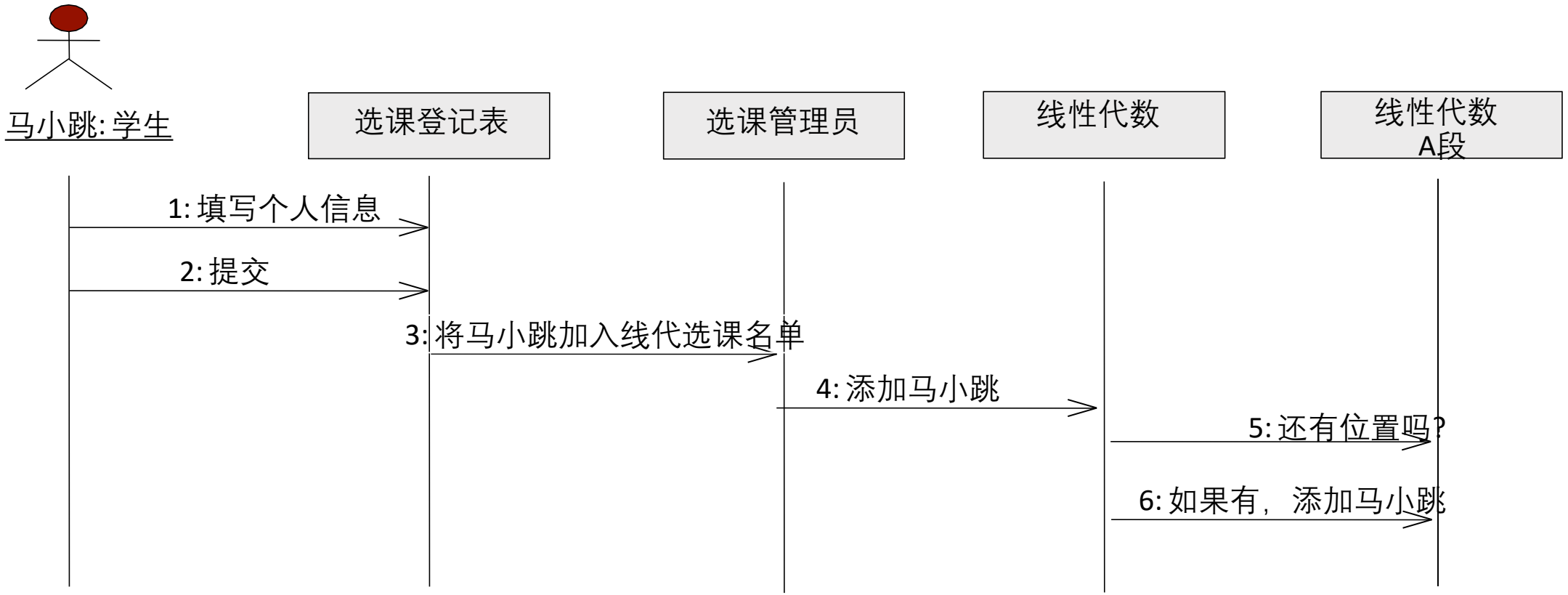
- 状态图（Statechart Diagram）
- 顺序图（Sequence Diagram）
- 协作图（Collaboration Diagram）
- 活动图（Activity Diagram）

顺序图

- 顺序图是强调消息时间顺序的交互图
- 顺序图描述了对象之间传送消息的时间顺序，表示用例中的行为顺序
- 顺序图将交互关系表示为一个二维图。其中，纵轴是时间轴，时间沿竖线向下延伸。横轴代表了在协作中各独立的对象

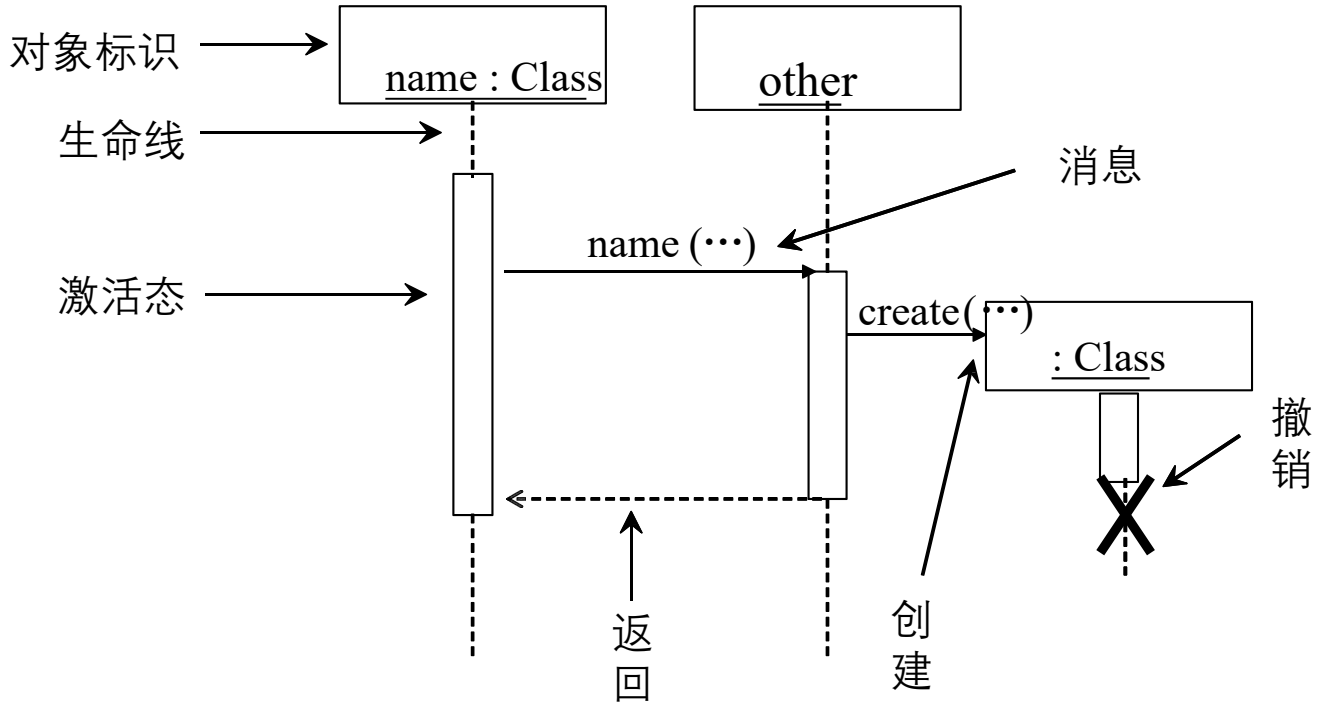
顺序图举例（Sequence Diagram）

- 顺序图用来刻画系统实现某个功能的必要步骤



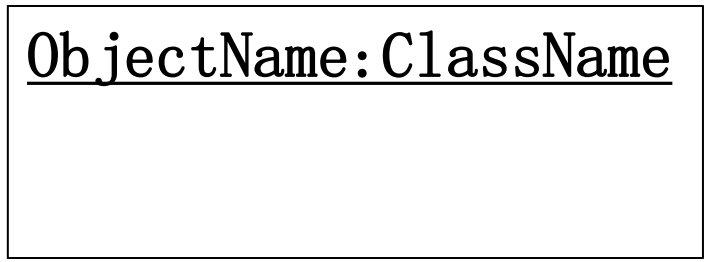
顺序图的组成

- 对象（Object）：类的实例，以某种角色参与交互，可以是人，物，其他系统或者子系统
- 生命线（Lifeline）：表示对象存在的时间
- 激活（Activation）：表示对象进行操作的时间片段
- 消息（Message）：对象方法的调用

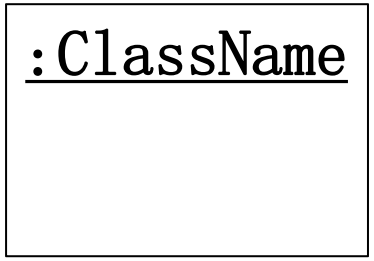


对象（Object）

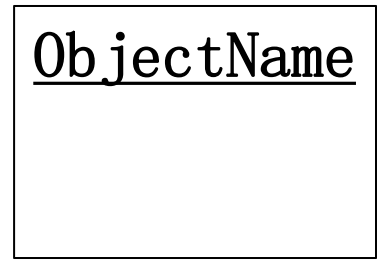
- 对象可以是系统的参与者或者任何有效的系统对象。对象是类的实例，它使用包围名称的矩形框来标记。名称带下划线。顺序图中对象的标记符如下



显示对象名和类名



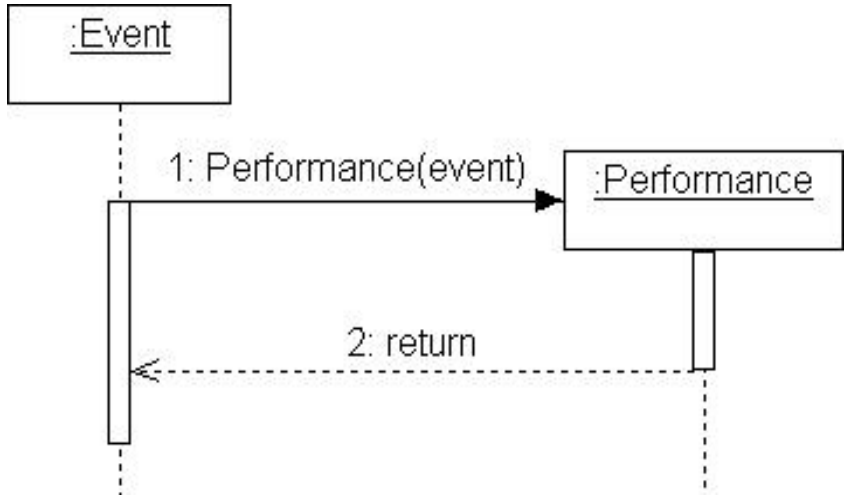
只显示类名

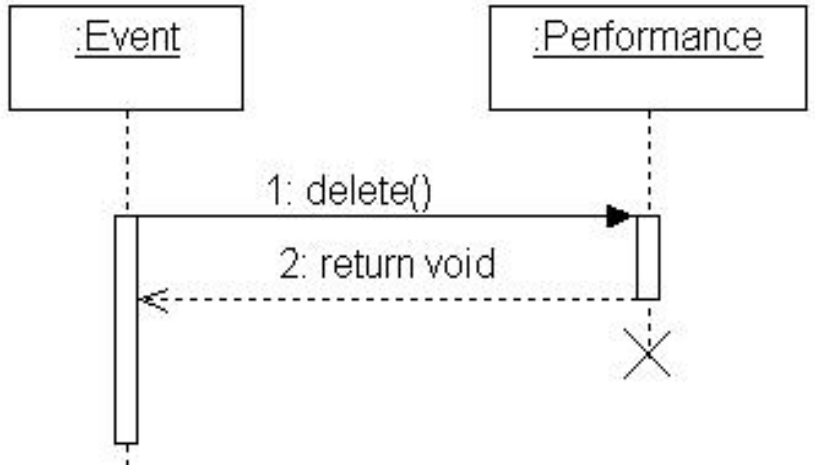


只显示对象名

对象（Object）

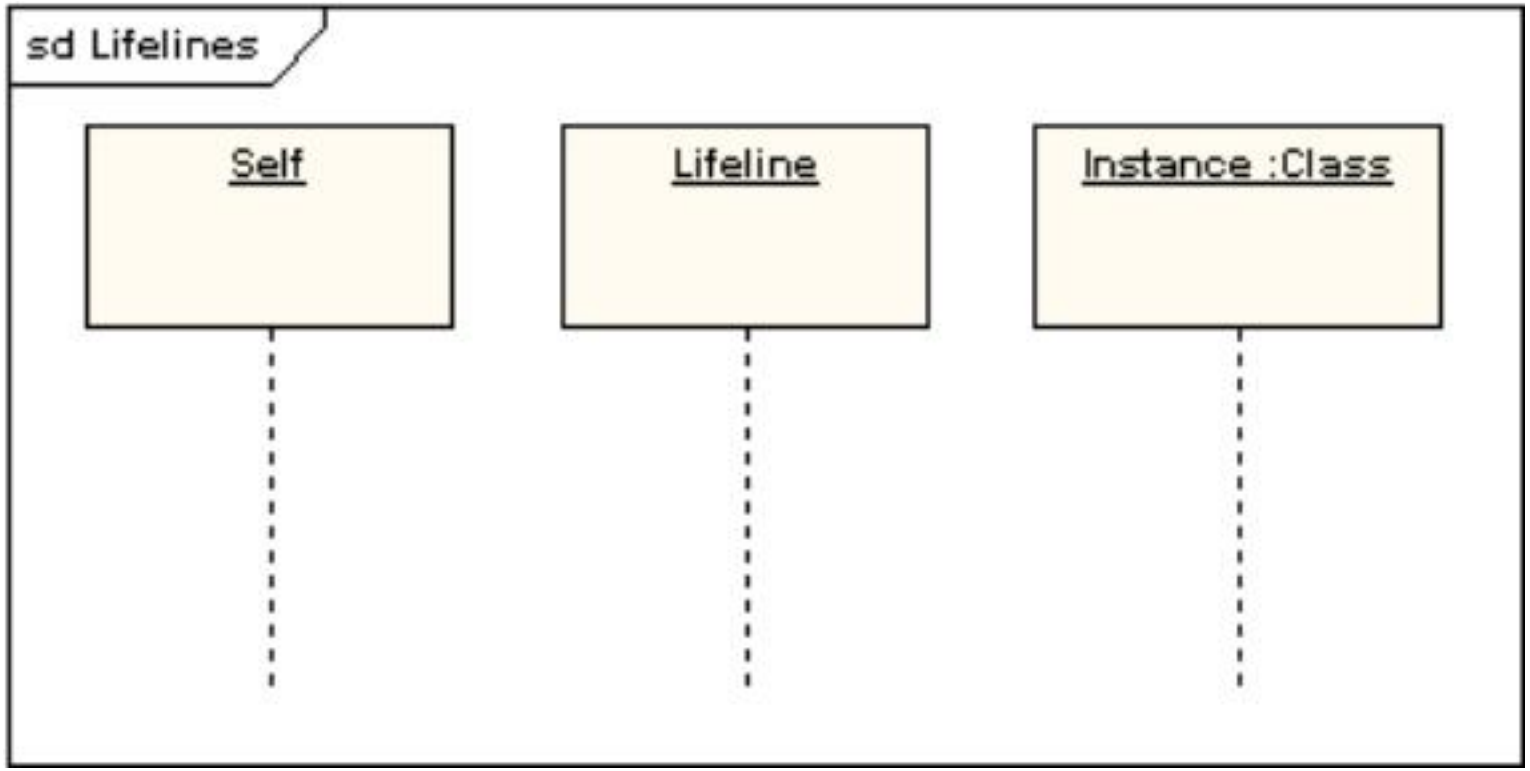
- 对象的创建与撤销
  - 如果对象位于顺序图的顶部，说明在交互开始之前该对象已经存在了。如果对象是在交互的过程中创建的，新建的对象在图中的位置较低。
  - 对象在创建消息发生之后才能存在，对象的生命线也是在创建消息之后才存在的。
  - 如果要撤销一个对象，只要在其生命线终止点放置一个“X”符号即可，该点通常是对删除或取消消息的回应。





生命线（Lifeline）

- 生命线在顺序图中表示为从对象图标向下延伸的一条虚线，表示对象存在的时间

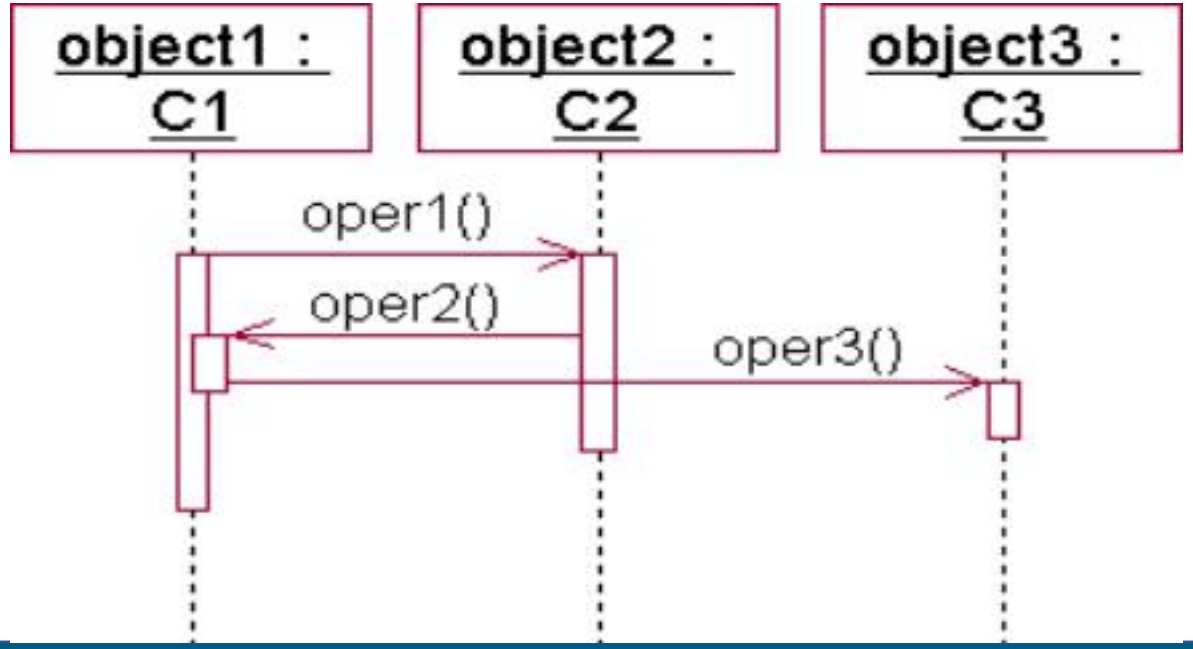


激活（Activation）

- 控制焦点/激活期(Focus of Control/Activation)表示对象进行操作的时间片段
  - 激活表示该对象被占用以完成某个任务，去激活指的则是对象处于空闲状态、在等待消息。
  - 在UML中，为了表示对象是激活的，可以将该对象的生命线拓宽成为矩形。其中的矩形称为激活条或控制期，对象就是在激活条的顶部被激活的，对象在完成自己的工作后被去激活。

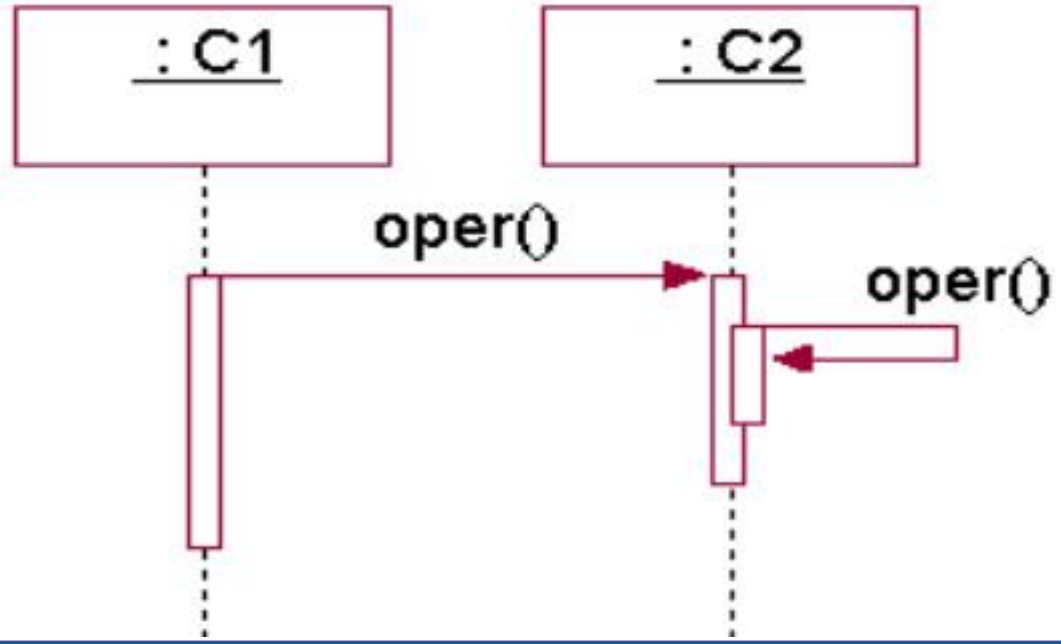
激活（Activation）

- 控制焦点/激活期(Focus of Control/Activation)的嵌套
  - 嵌套的FOC可以更精确地说明消息的开始和结束位置。



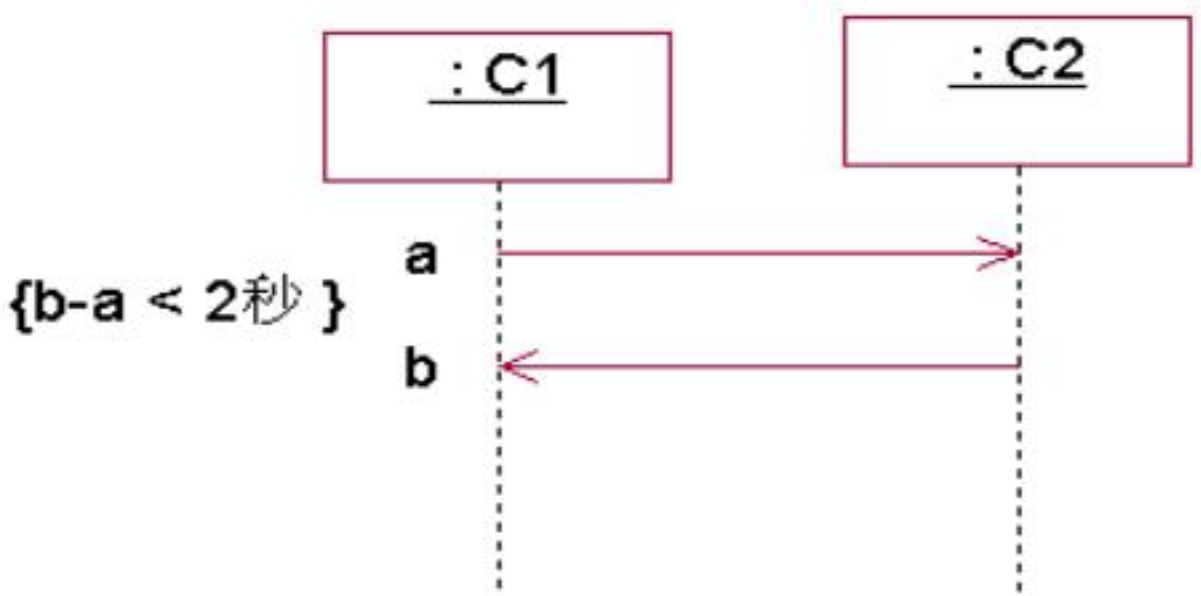
激活（Activation）

- 控制焦点/激活期(Focus of Control/Activation)的嵌套
  - 嵌套的FOC可以表示递归。



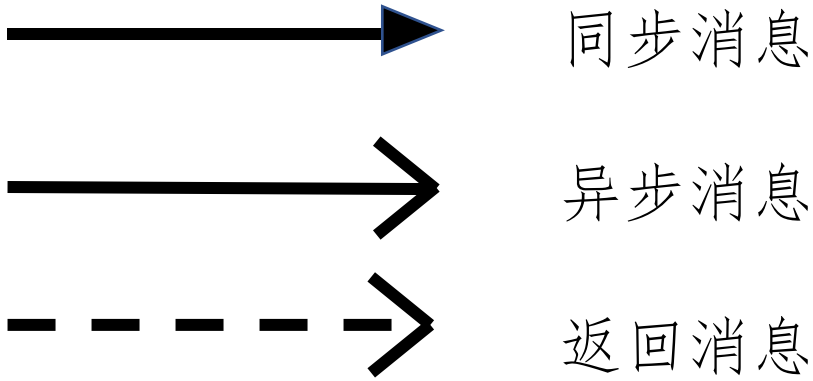
激活（Activation）

- 顺序图中时间约束的表示
  - 用约束(constraint)来表示



消息（Message）

- 消息(Message)用于描述对象间的交互操作和值传递过程，在UML中，消息使用箭头来表示，箭头的类型表示了消息的类型。
  - 对象之间的消息只能单路通信、消息用箭头表示
- 消息类型:
  - Synchronous　同步消息（调用消息）
  - Asynchronous　异步消息
  - Return　返回消息
  - Time-out　超时等待
  - Self-message　自关联消息



消息（Message）

- 同步消息（调用消息）
  - 把调用的消息发给接受者、等待接受者放弃或者返回信息、接受者返回信息之前不能发送任何别的消息、并且工作流程被中断。

- 异步消息
  - 简单的说就是把消息发给接受者、不用等待接受者的反馈、可以给别的对象发消息异步消息可以并发工作

消息（Message）

- 用户登陆电影可视化系统，输入用户名和密码，发出登陆请求是同步消息还是异步消息？
- 微信公众号向用户推送一篇文章，推送是同步消息还是异步消息

消息（Message）

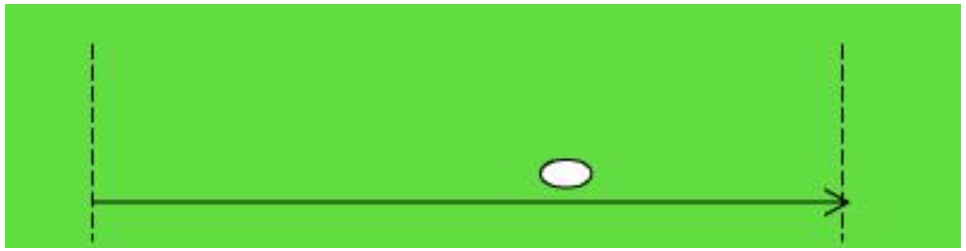
- 我去银行办理业务，可能会有两种方式：
  1. 选择排队等候
  2. 另种选择取一个小纸条上面有我的号码，等到排到我这一号时由柜台的人通知我轮到我去办理业务了

消息（Message）

- 返回消息
  - 发送给对象的异步消息或调用消息、对象给的反馈、称作返回消息
  - 如果是过程调用的返回、返回消息是隐含的、所以返回消息可以不用画出来
  - 如果是非过程的、返回消息要明确的表示出来！

消息（Message）

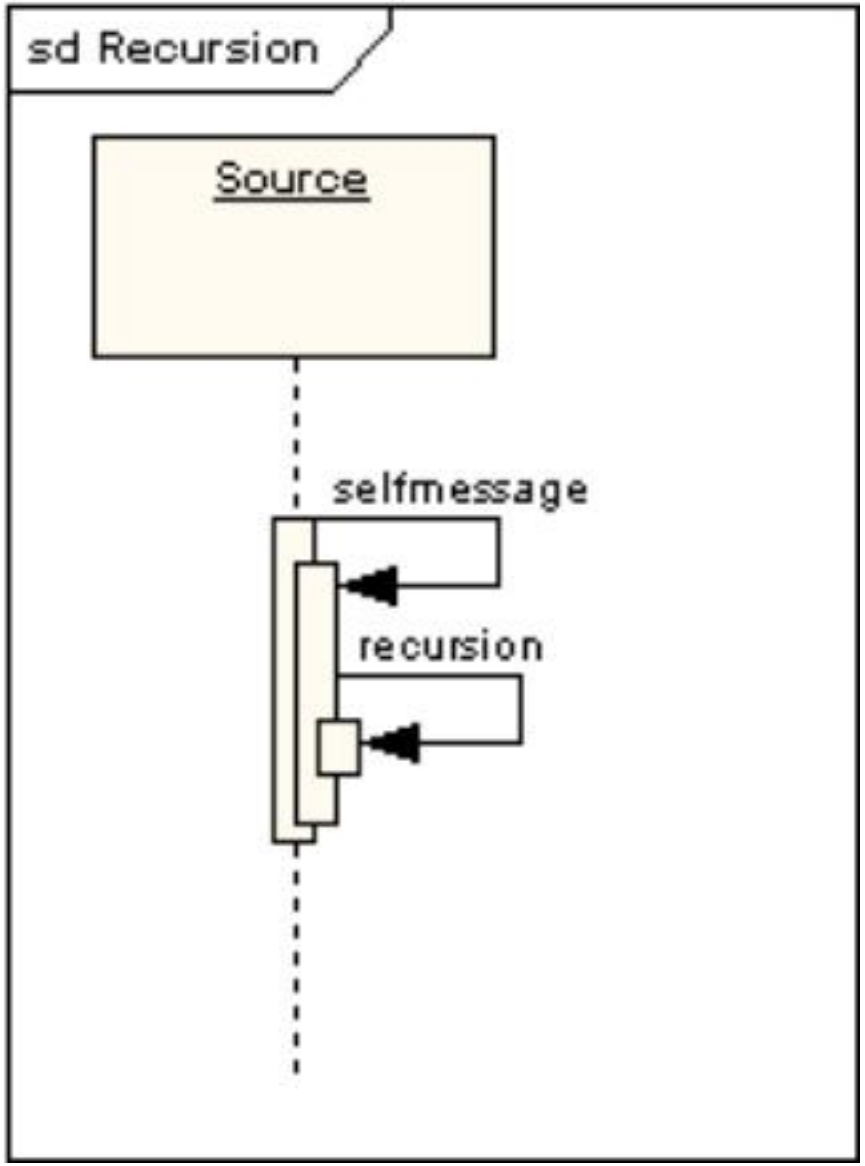
- 超时消息
  - 接受者在指定时间内无法接受此消息、则发送者放弃这个消息



- 能举出在电影可视化系统中超时消息的例子吗？

消息（Message）

- 自关联消息
  - 消息发送者给自己发消息



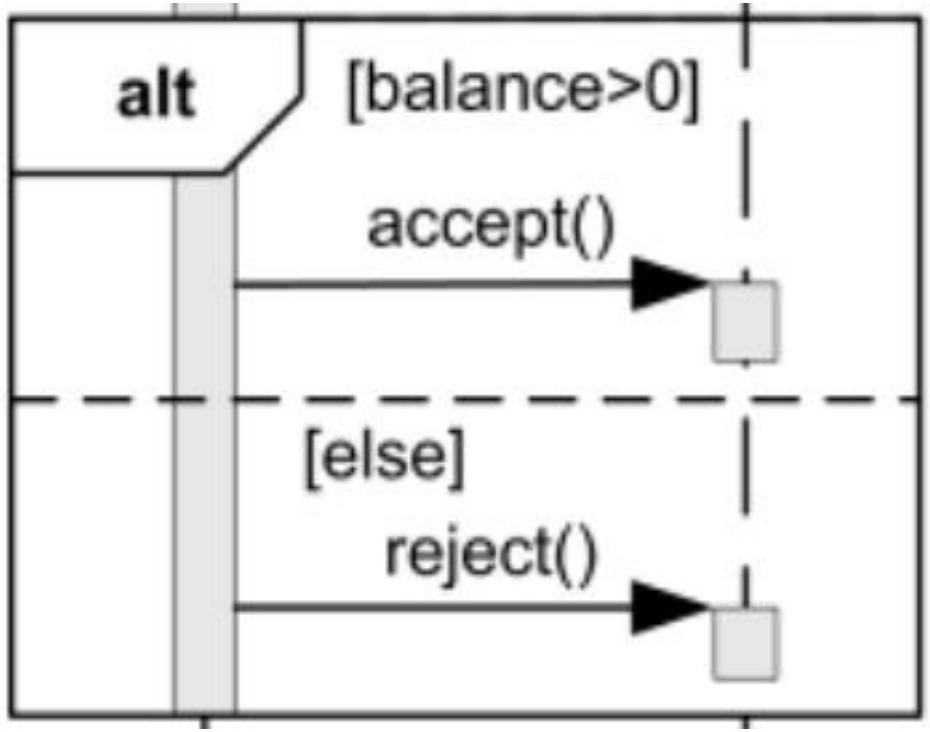
复合片段

- 一个复杂的顺序图可以划分为几个小块，每一个小块称为一个复合片段。每个复合片段由一个大方框包围，其名称显示在方框左上角的间隔区内，表示该顺序图的信息。
  - alt:多条路径，条件为真时执行。if/else -> (alt)[condition] 通过水平虚线分割不同情形。
  - opt:任选，仅当条件为真时执行。If -> (opt)[condition]
  - par:并行，每一片段都并发执行
  - loop:循环，片段可多次执行

复合片段

- if/else -> (alt)[condition]，多条路径，条件为真时执行

Alternatives

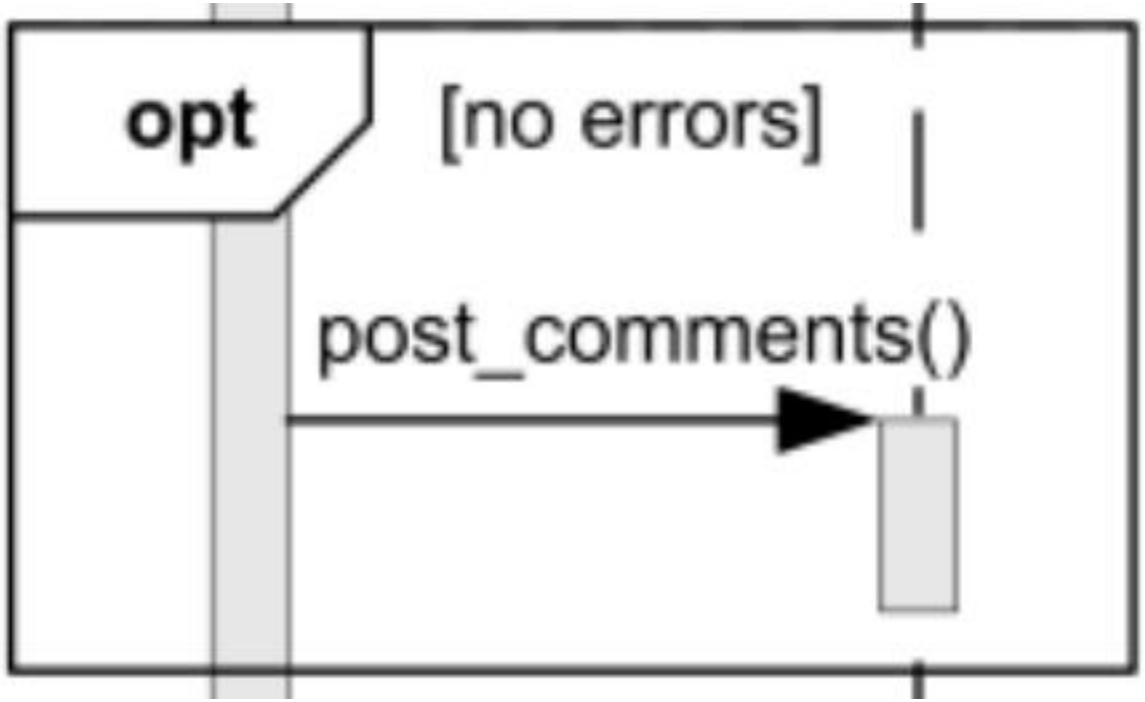


Call accept() if balance > 0, call reject() otherwise.

复合片段

- If -> (opt)[condition]，任选，仅当条件为真时执行

Option

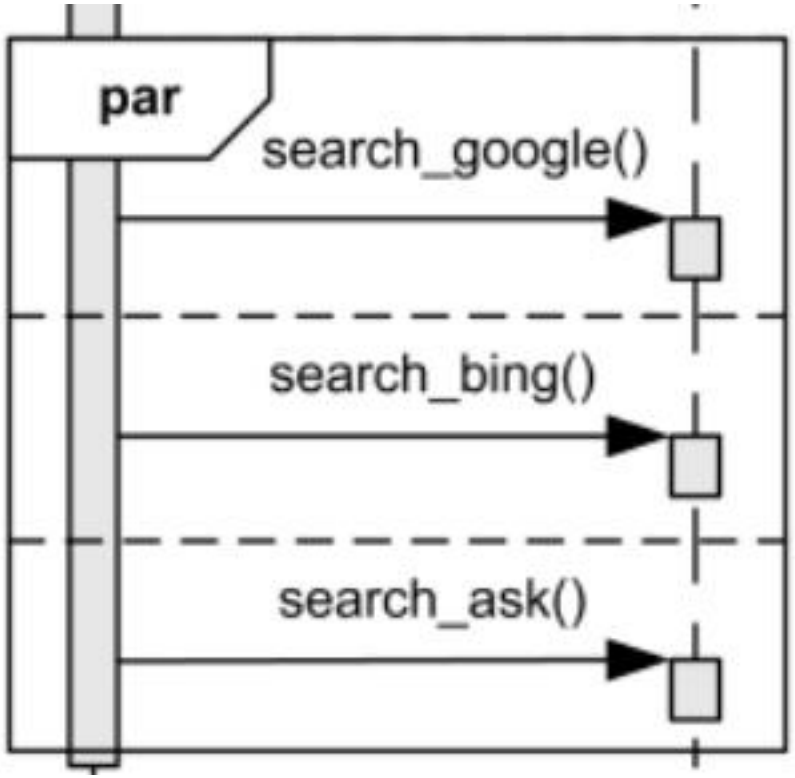


Post comments if there were no errors.

复合片段

- 并行，每一片段都并发执行

Parallel

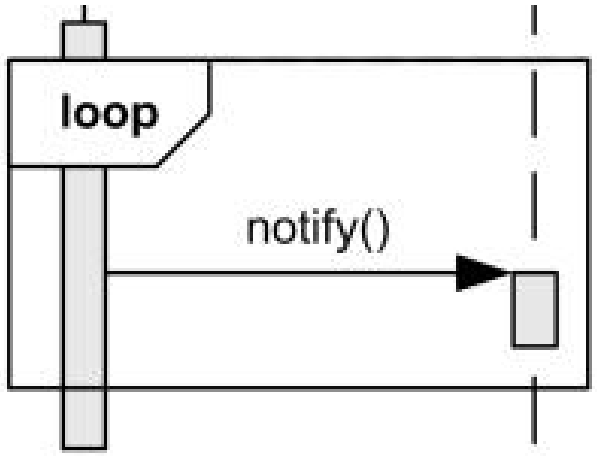


Search Google, Bing and Ask in any order, possibly parallel.

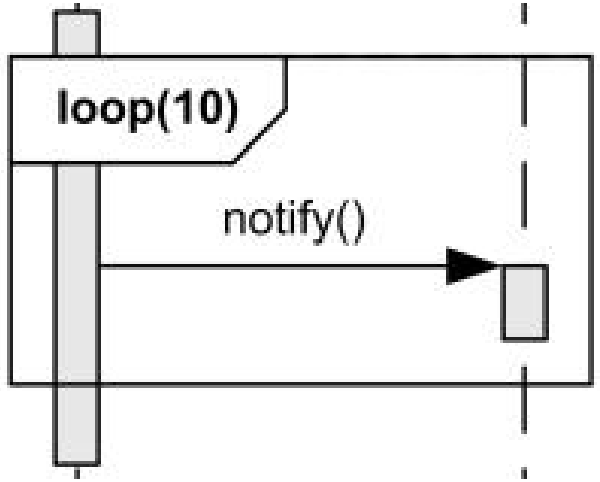
复合片段

- 循环，片段可多次执行

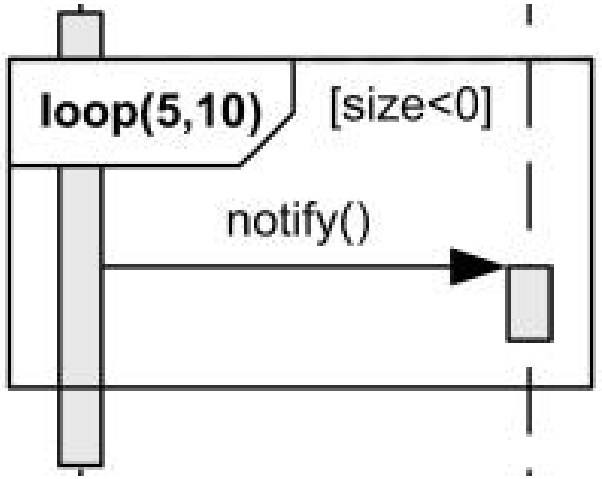
Loop



Potentially infinite loop.

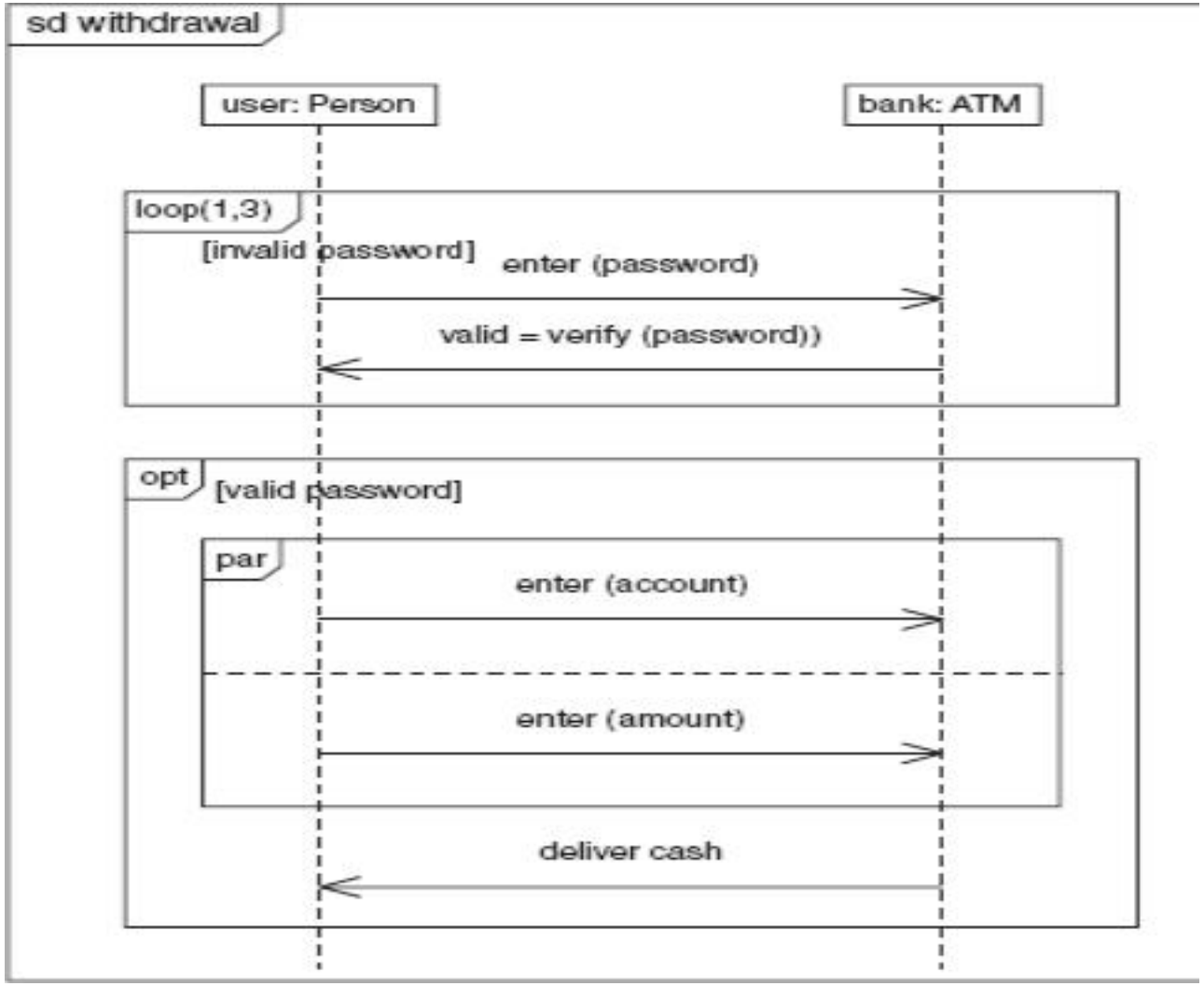


Loop to execute exactly 10 times.



the loop is expected to execute minimum 5 times and no more than 10 times.

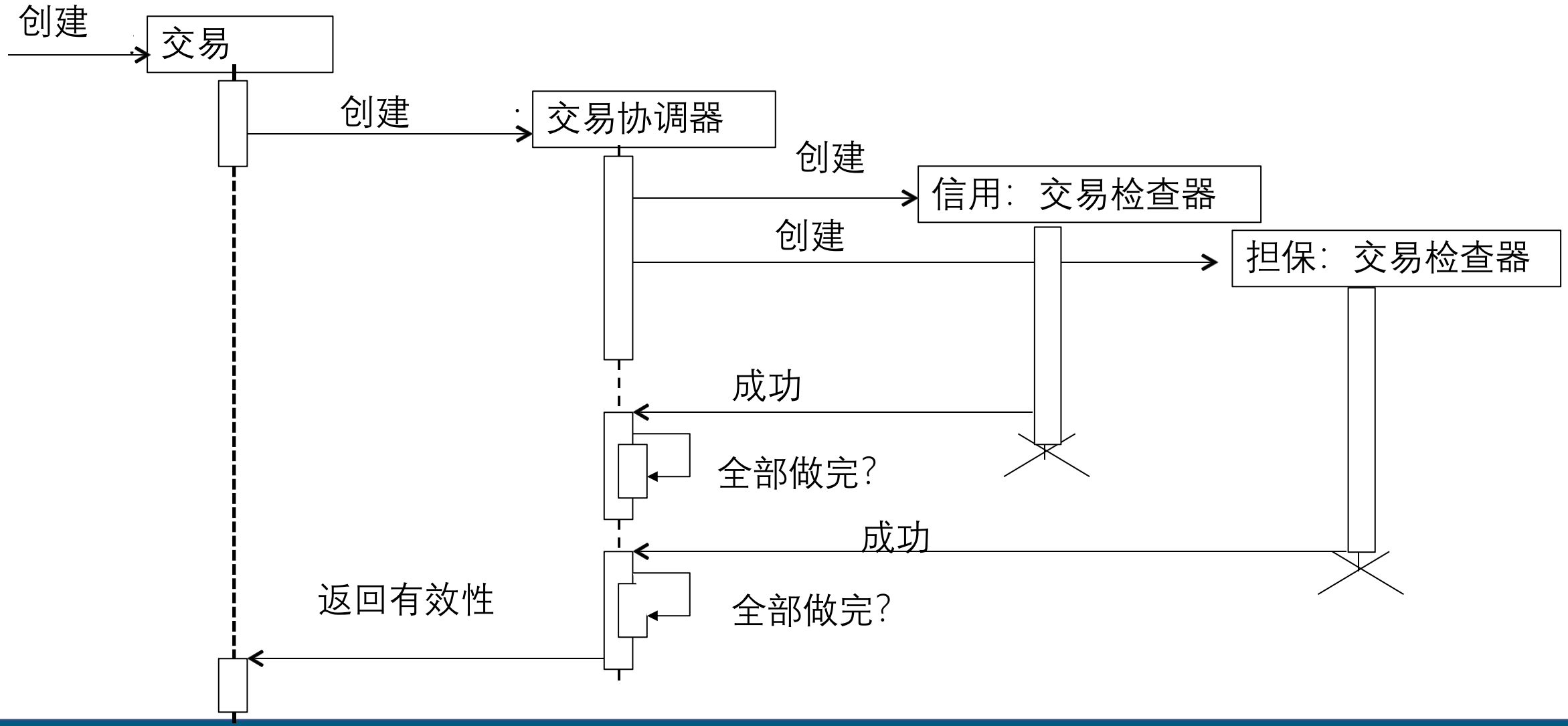
复合片段的例子


sd withdrawal
user: Person
bank: ATM
loop(1,3)
[invalid password]
enter (password)
valid = verify (password))
opt
[valid password]
par
enter (account)
enter (amount)
deliver cash

绘制顺序图步骤

1. 在顺序图顶端绘制矩形框，定义参与交互的类实例（对象）名；
2. 在每个对象下面绘制竖直虚线，表示该对象的生命线；
3. 在对象间添加箭头表示各种类型的消息，跟踪对象间的控制流；
4. 生命线加竖直矩形定义对象激活期，表明对象正在执行某操作；
5. 根据需要添加框的组合与关联，表示复杂的控制结构。

例:银行系统的交易验证



顺序图建模意义

- 通过顺序图描述算法逻辑
- 高质量的顺序图是代码的抽象
- 顺序图是与语言无关的表示方式
- 可以绘制顺序图来描述业务逻辑
- 可以通过团队协作完成顺序图的绘制
- 可以在同一页浏览多个对象和类的行为

4.4 行为建模与设计

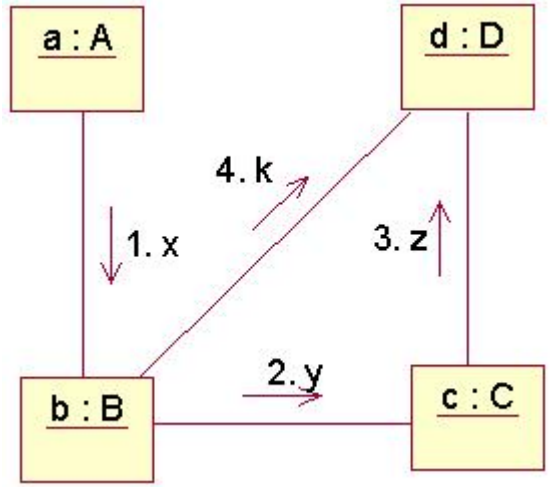
- 状态图（Statechart Diagram）
- 顺序图（Sequence Diagram）
- 协作图（Collaboration Diagram）
- 活动图（Activity Diagram）

协作图

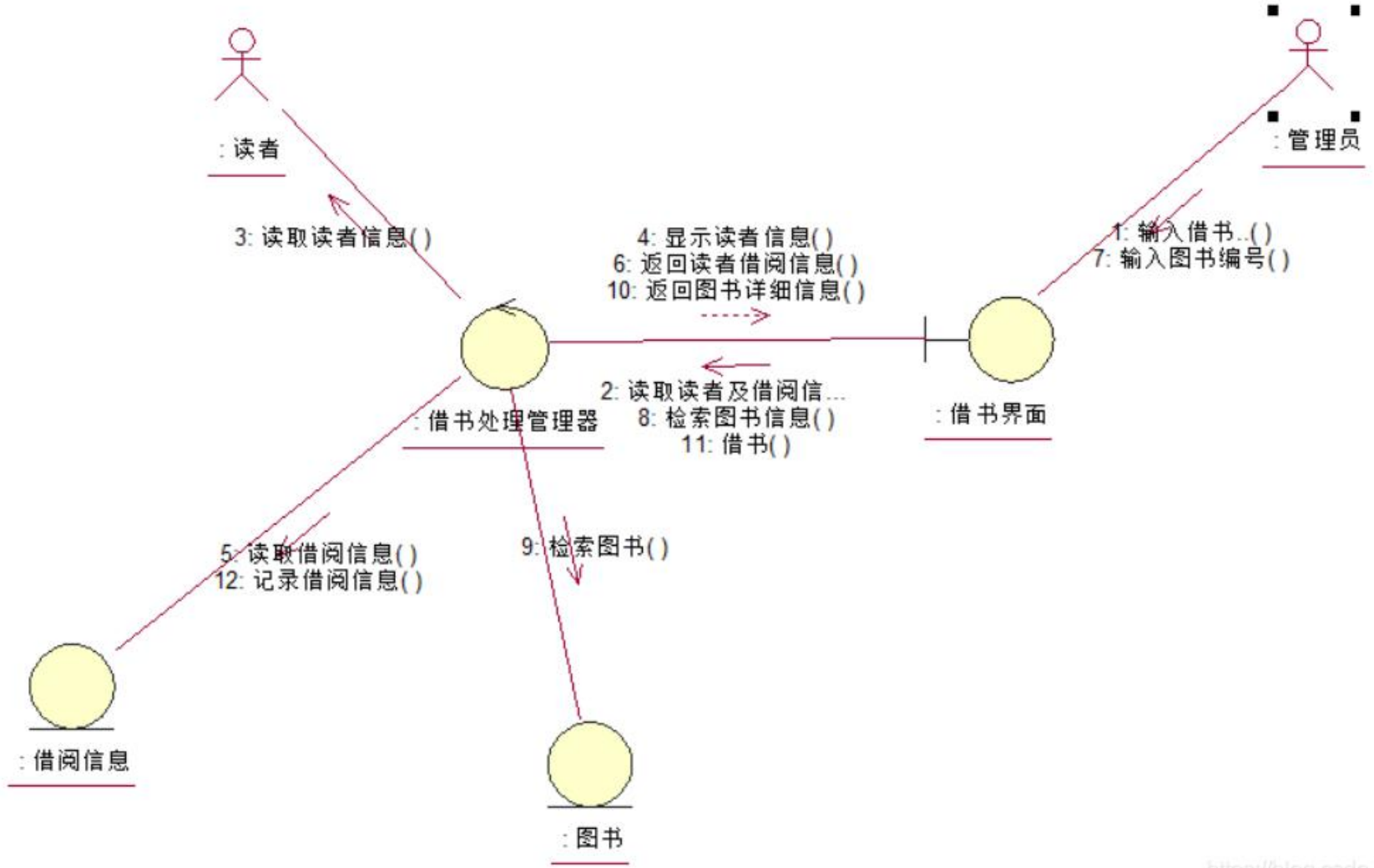
- 协作图是交互图的另一种表现形式，它强调参加交互的各对象的组织。主要用于描述一组相互合作的对象间的交互和链接。
- 协作图只对相互间有交互作用的对象和这些对象间的关系建模，而忽略了其他对象和关联。
- 顺序图主要描述对象间消息发送的时间顺序，而协作图侧重于描述交互对象之间的链接关系，而不专门突出这些消息发送的时间顺序。
- 协作图不像时序图一样具备时间维，为了表示消息的时间顺序，通常要为消息加一个数字前缀。

协作图

- 协作图由对象、链接和消息组成。
- --使用实线表示两个对象间的链接
- --消息由标记在连接上方的带有标记的箭头表示



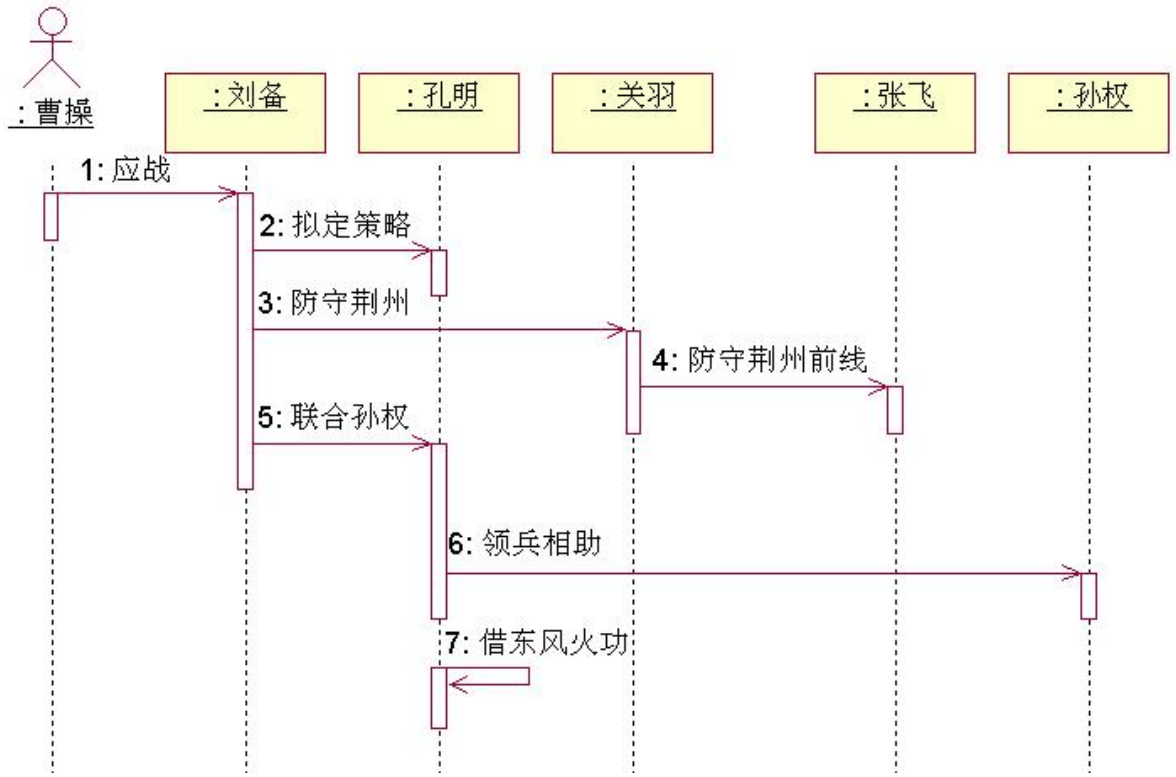
例子

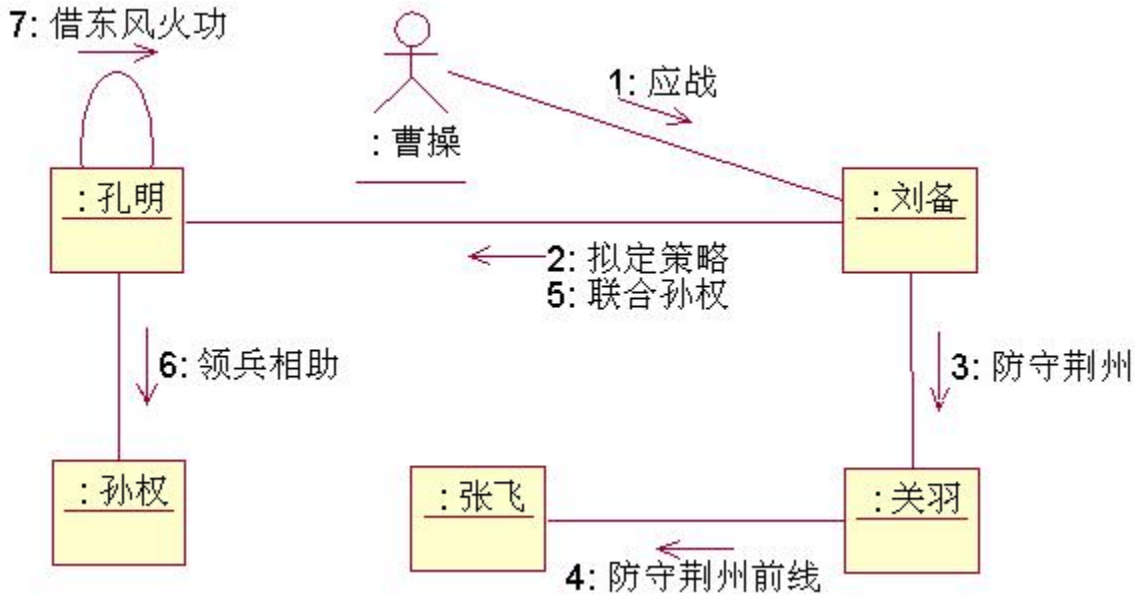


时序图与协作图比较

- 时序图和协作图都属于交互图，都用于描述系统中对象之间的动态关系。两者可以相互转换，但两者强调的重点不同。
- --当对象及其连接有利于理解对象之间的交互时,选择协作图;
- --当强调消息发送的时间顺序时,选择时序图。
- --时序图中有对象生命线和控制焦点，协作图中没有；协作图中有路径，并且协作图中的消息必须要有顺序号，但时序图中没有这两个特征。
- 实际应用中，一般采用时序图。

对比例子





(from https://www.jianshu.com/p/88aa96090105)

4.4 行为建模与设计

- 状态图（Statechart Diagram）
- 顺序图（Sequence Diagram）
- 协作图（Collaboration Diagram）
- 活动图（Activity Diagram）

活动图（Activity Diagram）

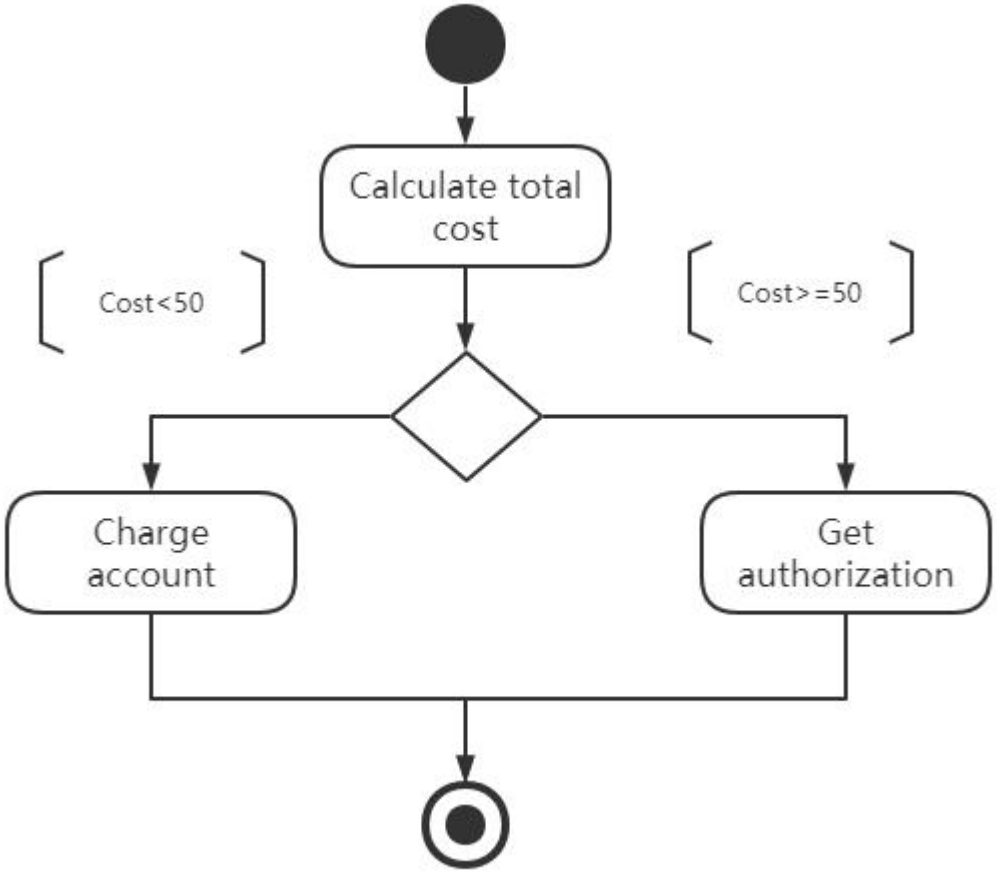
- 一般学习过c语言或别的程序设计语言的同学一定接触过流程图，因为流程图清晰的表达了程序的每一个步骤序列、过程、判定点和分支。
- 在UML里，活动图本质上就是流程图，描述系统的活动、判定点、分支等，可用于对系统的业务需求建模，因此它对于开发人员来说是一种重要的工具。
- UML活动图记录了单个操作或方法的逻辑，单个用户案例或者单个业务流程的逻辑。
- 也可以说，活动图是用图形化的方式描述事件流
- (即描述用例图中某个用例的逻辑流程)

活动图的基本概念和组成

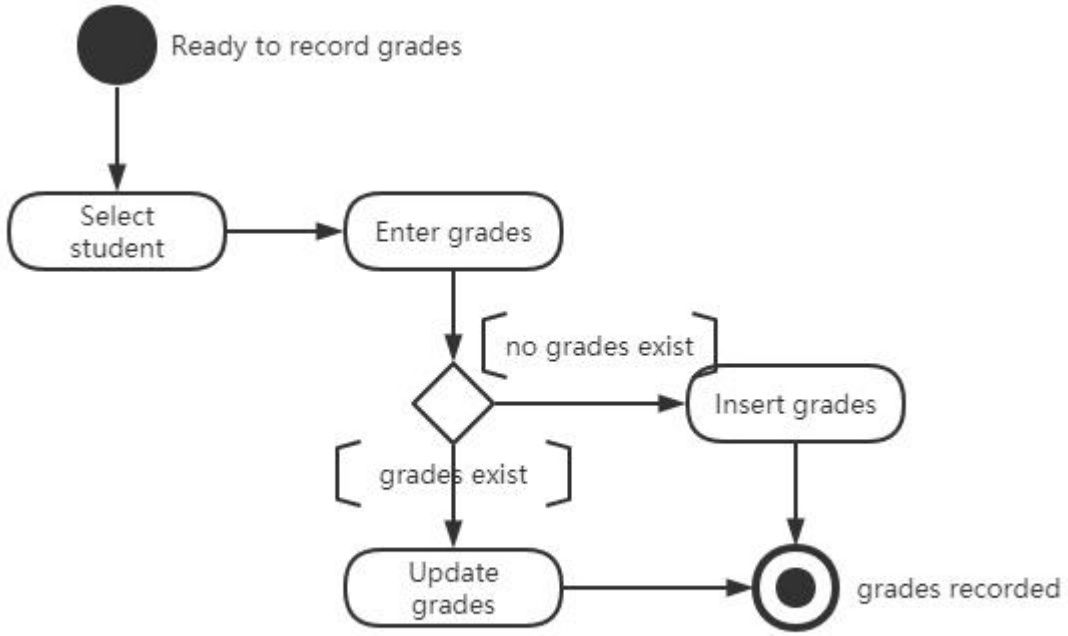
- 从系统内部视角来看，活动图反映的是系统功能所要完成的动作过程。它定义了工作流从何时开始、哪里开始、按什么顺序发生、最终在哪结束。
- 活动图由起始状态、终止状态、活动、状态转移、决策、守护条件、同步棒和泳道组成。
- 活动图的起始状态和终止状态的表示同状态图。
- 活动图中的活动用圆角四边形表示，内部文字说明采取的动作。动作间的转移用带有箭头的实线表示。

活动图的基本概念和组成

- 守护条件：用来约束转移，守护条件为真时转移才可以开始。
- 决策：活动图中的决策用一个菱形表示。分支表示一个触发事件在不同的触发条件下引起多个不同的转移 。
- 分支可以有一个进入转移和两个或多个输出转移。在每条输出转移上都有守护条件（即一个布尔表达式）保护，当且仅当守护条件的值为真时，该输出路径才有效。



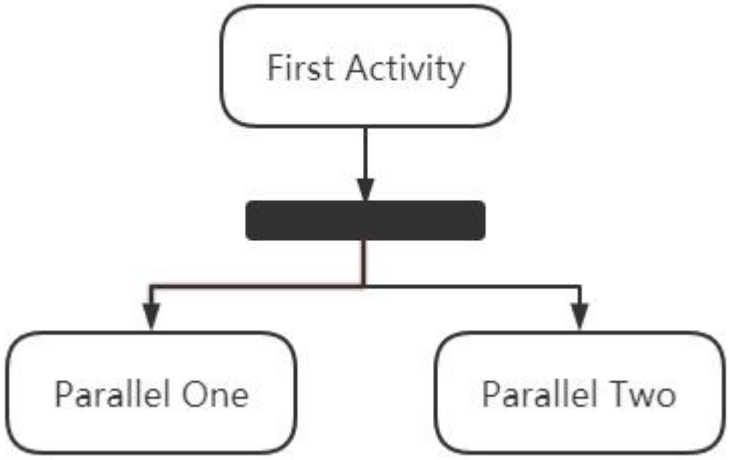
例子：“记录学生分数”的活动图

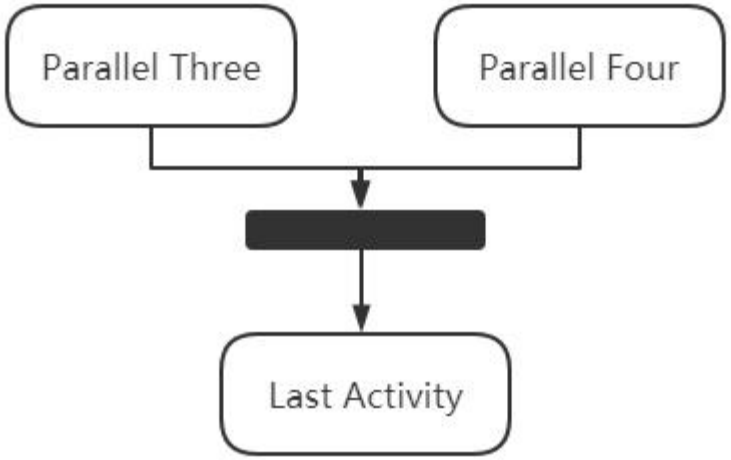


- 注:活动图与状态图的标记符非常相似，有时会让人混淆。其实，活动图是用来建模不同区域的工作如何彼此交互的；而状态图用来表示单个的对象，以及对象的行为如何改变其状态。

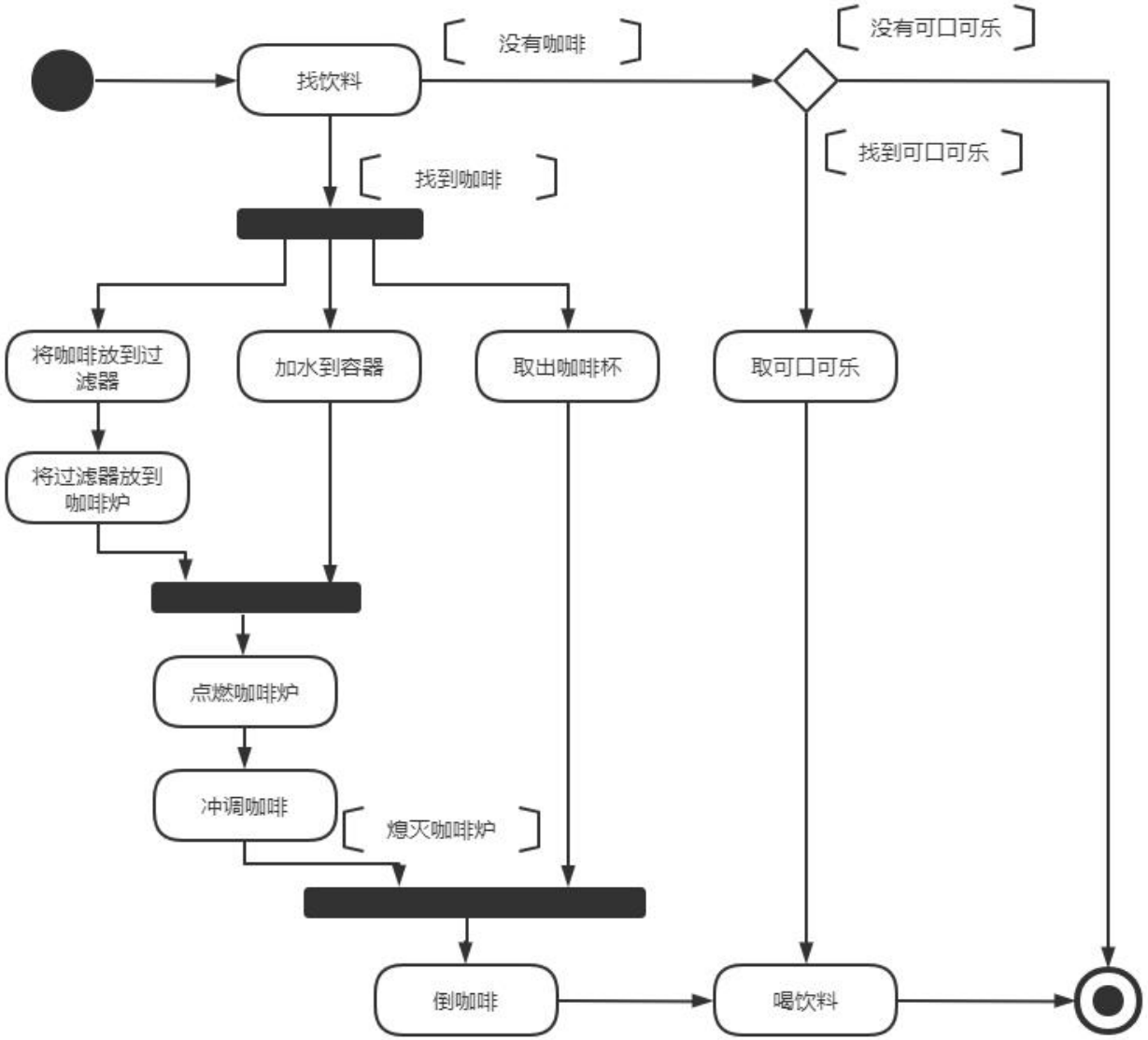
活动图的基本概念和组成（续）

- 同步棒：在建模过程中，可能会遇到对象在运行时存在两个或多个并发运行的控制流。所有的并行转移在合并前必须被执行。
- 在UML中，一条粗黑线表示将转移分解成两个或多个并发流，同样用粗黑线表示分支的合并。粗黑线称为同步棒。



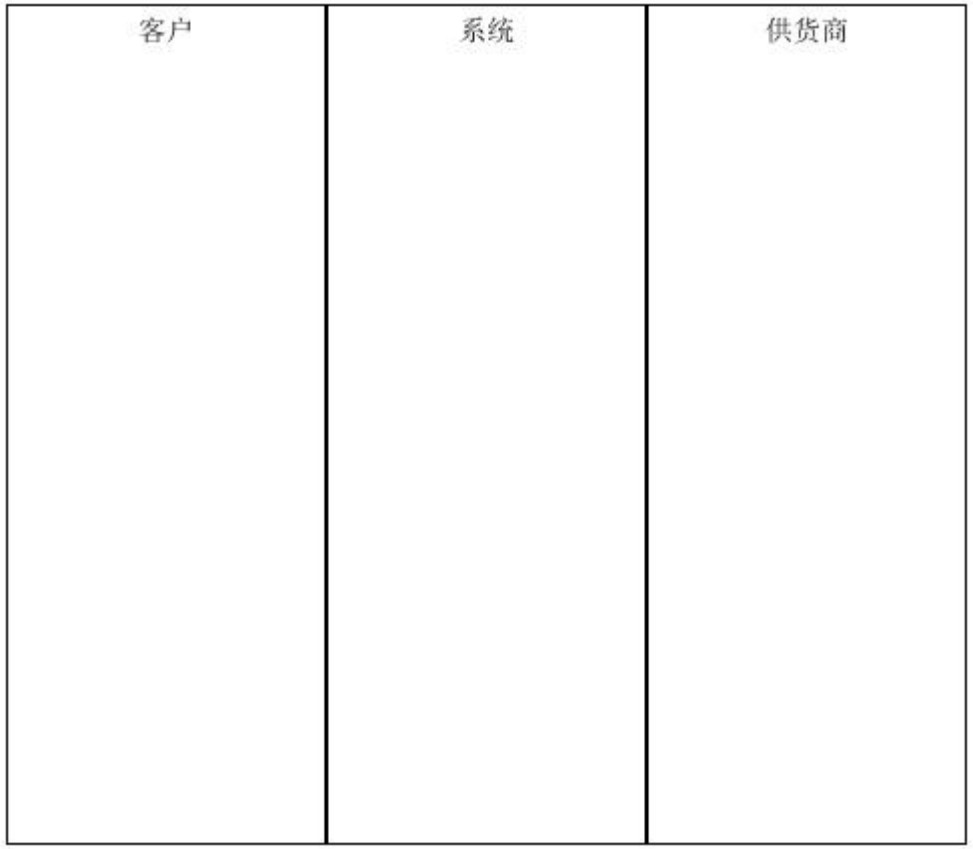


例子

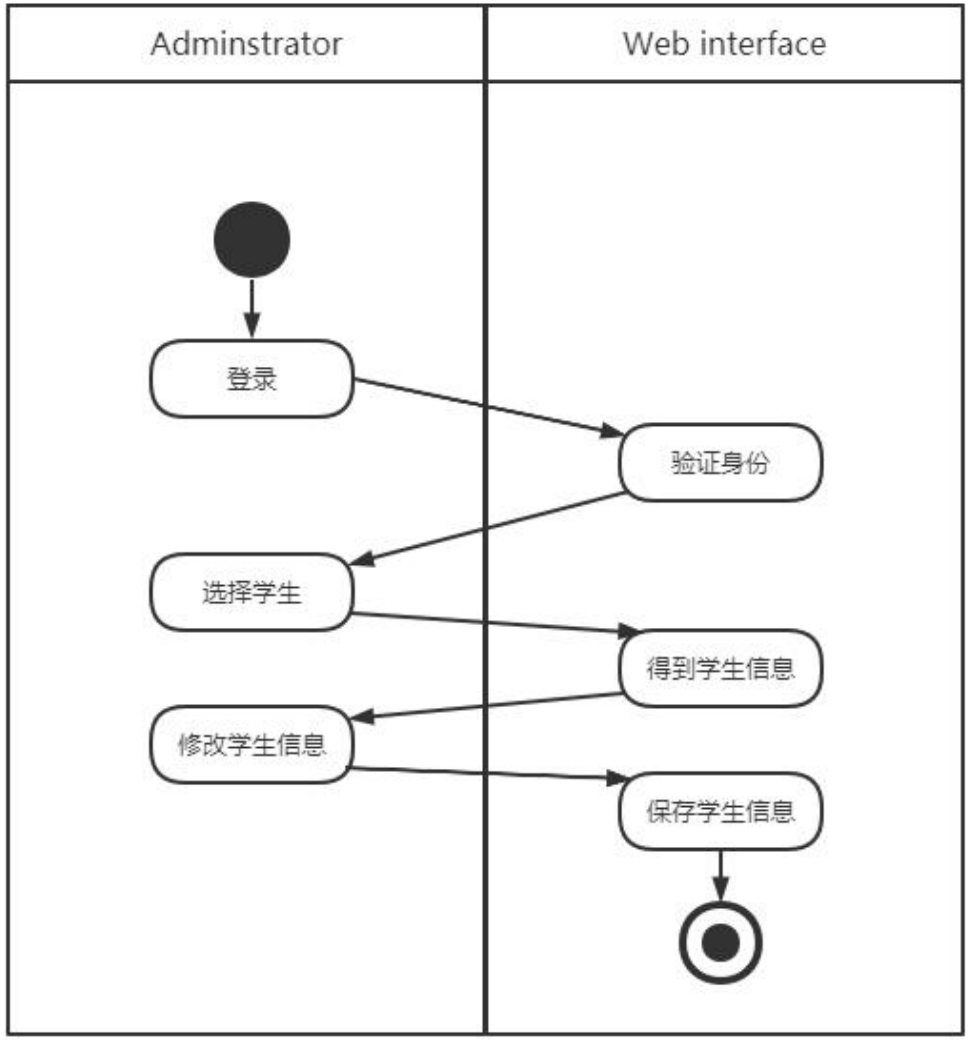

找饮料
没有咖啡
没有可口可乐
找到咖啡
找到可口可乐
将咖啡放到过滤器
加水到容器
取出咖啡杯
取可口可乐
将过滤器放到咖啡炉
点燃咖啡炉
冲调咖啡
熄灭咖啡炉
倒咖啡
喝饮料

活动图的基本概念和组成--泳道

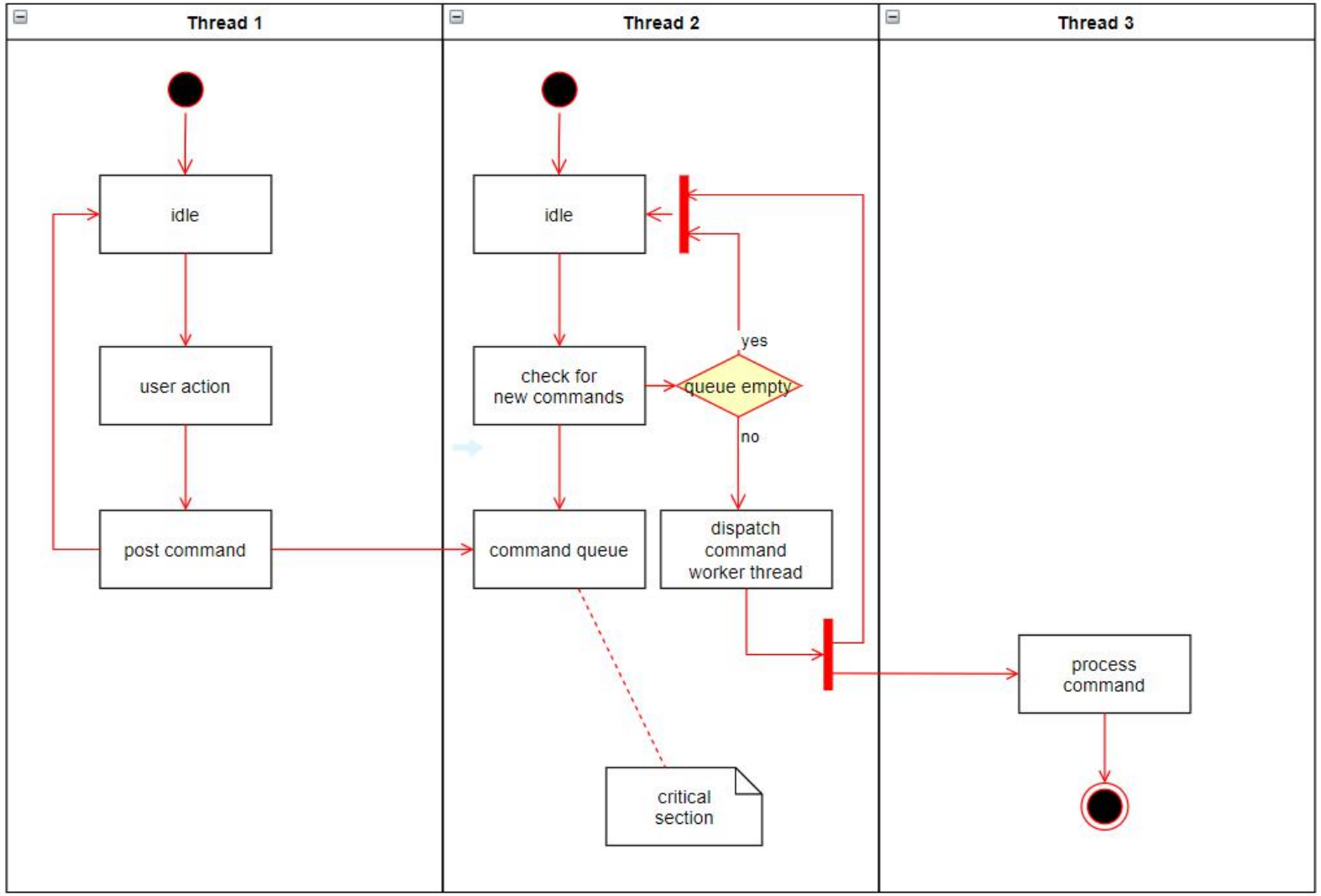
- 活动图告诉你发生了什么，但没有告诉你该项活动由谁来完成。在程序设计中，这意味着活动图没有描述出各个活动由哪个类来完成。泳道解决了这一问题。
- 泳道：用矩形框来表示，属于某个泳道的活动放在该矩形框内，将对象名放在矩形框的顶部，表示泳道中的活动由该对象负责。
- 泳道可以提高活动图的可读性，可用于建模某些复杂的活动图。



例子：“修改学生信息”


Adminstrator
Web interface
登录
验证身份
选择学生
得到学生信息
修改学生信息
保存学生信息

例子


Thread 1
Thread 2
Thread 3
idle
user action
post command
idle
check for
new commands
yes
queue empty
no
command queue
dispatch
command
worker thread
critical
section
process
command

活动图的优缺点

- 活动图最适合支持描述并行行为，这使之成为支持工作流建模的最好工具

- 但活动图最大的缺点是很难清楚地描述动作与对象之间的关系。

活动图的用途

- 活动图用于对系统的动态行为建模。活动图描述了从活动到活动的流。
- 在对一个系统建模时，通常有两种使用活动图的方式：
  - (1) 为工作流建模

    对工作流建模强调与系统进行交互的对象所观察到的活动。用于可视化、详述、构造和文档化开发系统所涉及的业务流程。
  - (2) 为对象的操作建模

    活动图本质上就是流程图，他描述系统的活动、判定点、分支等部分。因此，在UML中，可以把活动图作为流程图来使用，用于对系统的操作建模。

活动图的适用范围

- 对于以下情况可以使用活动图
  - (1)分析用例，即用图形化的方式描述用例的事件流；
  - (2)理解牵涉多个用例的工作流，即描述系统的业务流程；
  - (3)处理多线程应用。

第四部分 软件设计

- 4.1 软件工程开发方法与软件设计
- 4.2 体系结构设计
- 4.3 类/数据建模与设计
- 4.4 行为建模与设计
- 4.5 物理建模与设计

4.5 物理建模与设计

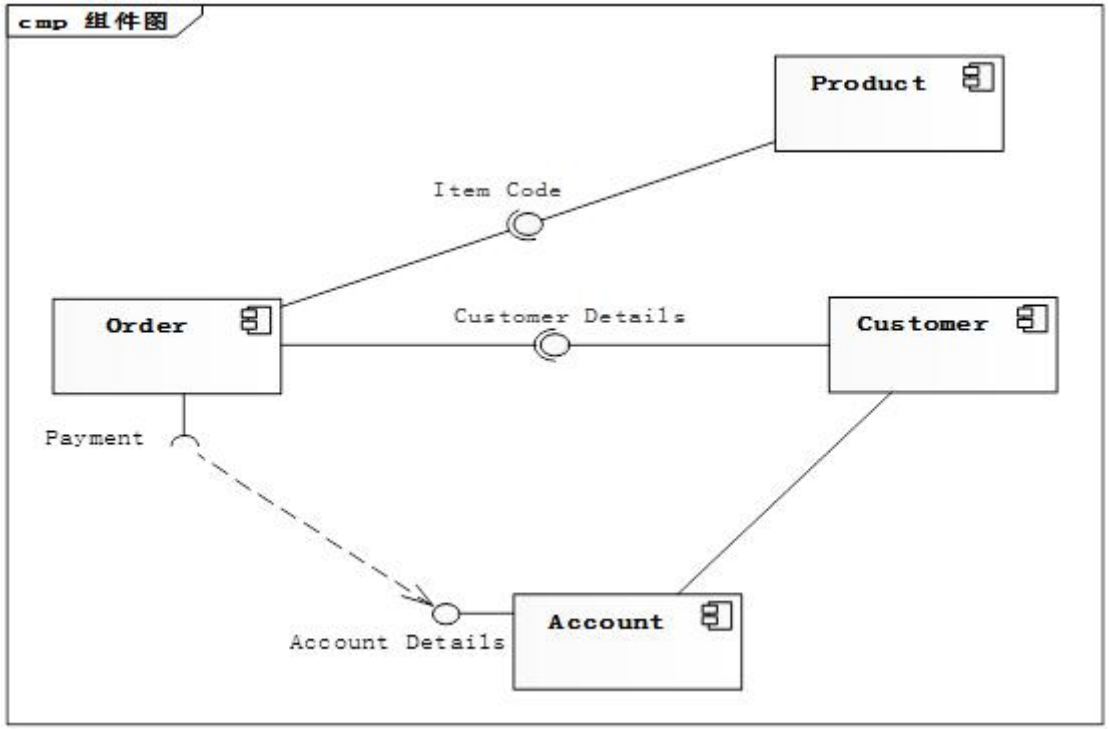
- 组件图（Component Diagram）

组件图

- 组件图是对面向对象系统的物理方面建模时使用的两种图之一，另一种图是配置图。目前，演示意义大于实际意义。
- 组件图和配置图统称系统的实现图。其中组件图显示代码本身的逻辑结构；配置图显示系统运行前的结构。
- 组件图描述软件组件以及组件之间的关系。组件是代码的软件模块，组件图则显示了代码的结构。
- 组件图画在组件视图(Component View)下面。
- 组件图显示了组件以及它们之间的管理信息

组件

- 组件是代码的软件模块，一般来说，就是一个实际文件。
- 组件包括以下类型：
- --源代码组件
- --二进制组件
- --可执行组件
- 组件的特点：
  - 组件是物理的
  - 组件是可替代的
  - 组件是系统的一部分



组件图的作用

- 每个组件体现了系统设计中特定类的实现。良好定义的组件不直接依赖于其他组件而依赖于组件所支持的接口
- 每个组件实现一些接口，并使用另一些接口。
- 组件图的用途是显示系统中的组件之间的依赖关系，以及组件的接口和调用关系。