哈爾濱工程大學
Harbin Engineering University

软件工程基础 —— 第8章 需求建模：基于场景的方法

计算机学院 孟宇龙

目录

关键概念

- 活动图
- 域分析
- 正式用例
- 需求分析
- 需求建模
- 基于场景分析
- 泳道图
- UML模型
- 用例
- 用例异常

8.1 需求分析

- 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
- 需求分析让软件工程师（有时这个角色也被称作分析或建模师）：
  - 细化在前期需求工程任务中建立的基础需求
  - 建立描述用户场景、功能活动、问题类及它们之间的关系、系统和类行为以及数据流变换时等模型

8.1 需求分析

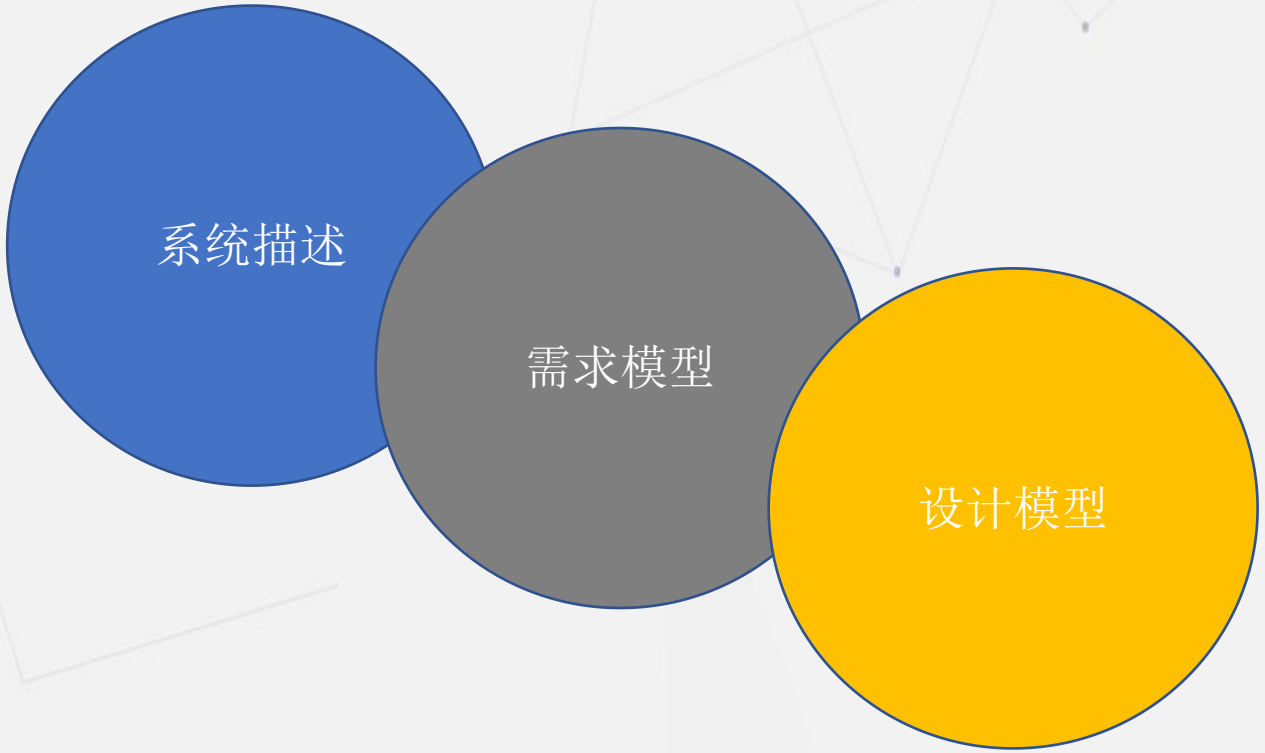
- 需求建模动作的结果为以下一种或几种
  - 场景模型
  - 面向类的模型
  - 基于行为和模式的模型
  - 面向流的模型
- 这些信息最终被转化为结构、接口和构件级的设计

8.1.1 总体目标和原理

- 需求必须实现三个主要目标：
  - 描述客户需要什么
  - 为软件设计奠定基础
  - 定义在软件完成后可以被确认的一组需求

8.1 需求分析

8.1.1 总体目标和原理



桥接

- **需求不是架构、设计和接口**
- **需求就是指需要什么**

8.1.2 分析的经验原则

- 模型应关注在问题域或业务域内可见的需求，抽象的级别应该相对高一些。（不试图解释系统将如何工作）
- 分析模型的每个元素都应能增加对软件需求的整体理解，并提供对信息域、功能和系统行为的深入理解。
- 关于基础和其他非功能的模型应推延到设计阶段再考虑。（例如数据库）
- 最小化整个系统内的关联。（减少在高层次的互联）
- 确认需求模型为所有利益相关者都带来价值。（给不同的人带来不同的利益）
- 尽可能保持模型简洁。（不使用不必要的图标和复杂的表示方法）

8.1.3 域分析

- 软件域分析是识别、分析和详细说明某个特定应用领域的公共需求，特别是那些在该应用域内被多个项目重复使用的需求……

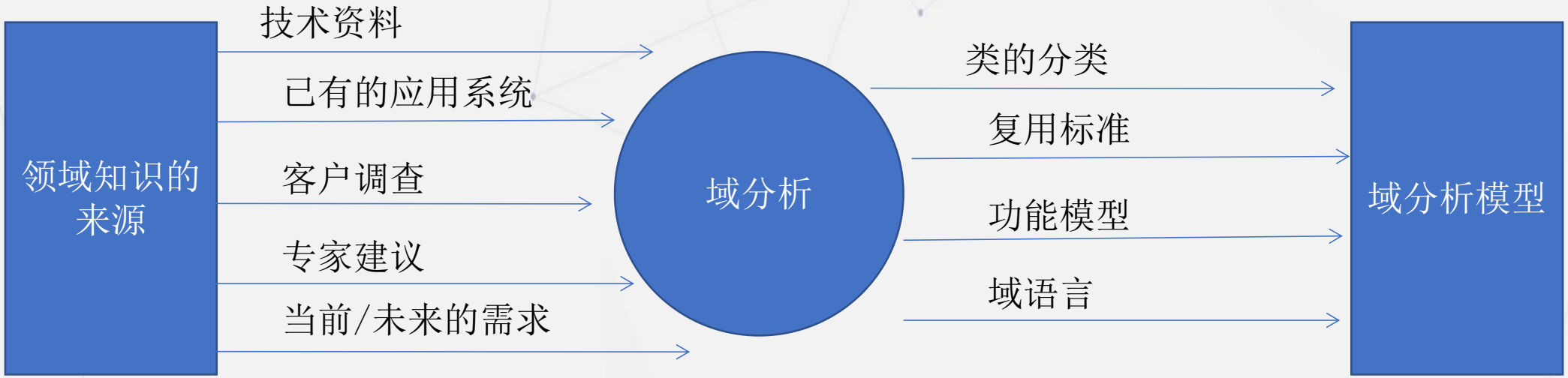
- [面向对象的域分析是]在某个特定应用领域，根据通用的对象、类、部件和框架，识别、分析和详细说明公共的、可复用的能力

Donald Firesmith

8.1.3 域分析

- 确定调查的领域
- 收集应用领域内的典型例子
- 分析例子中每一个应用
- 为对象开发一种分析模型

8.1.3 域分析



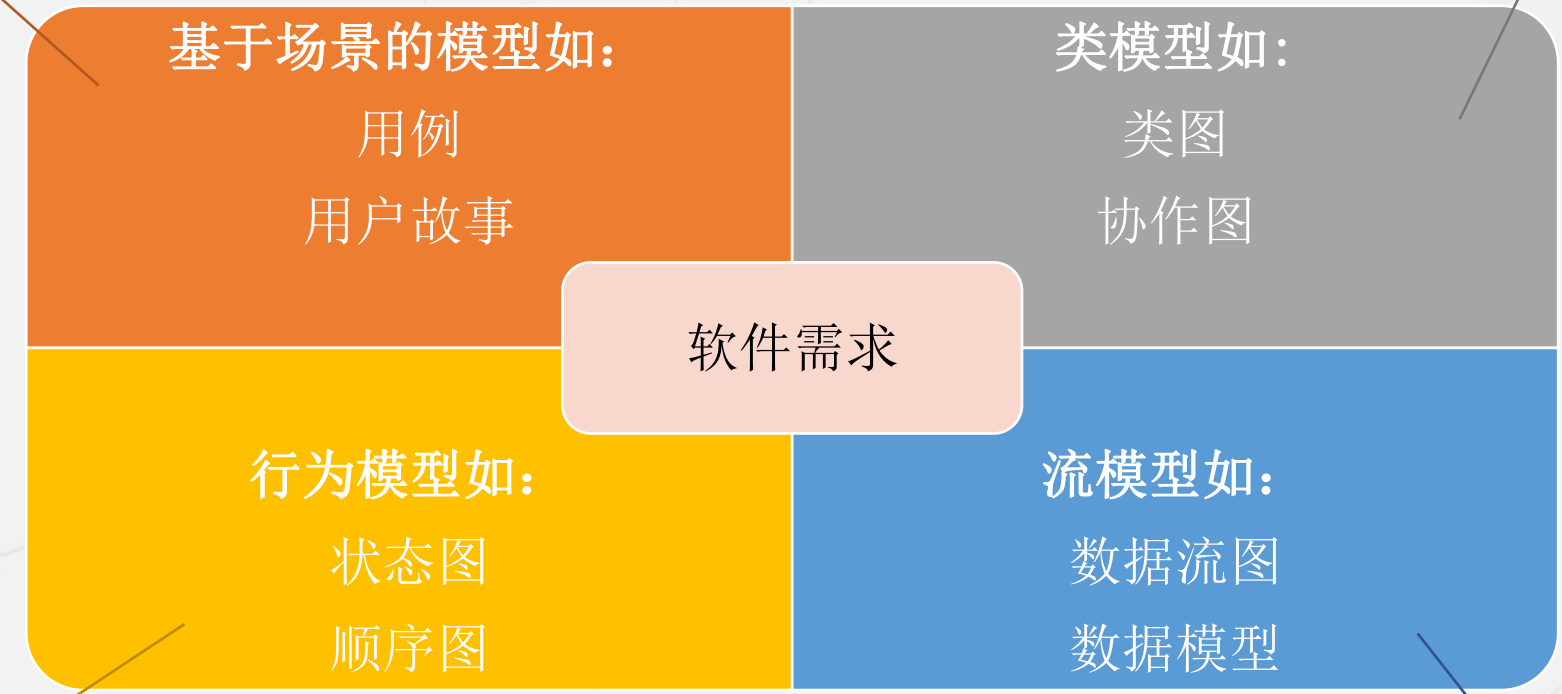
域分析的输入输出

8.1.4 需求建模的方法

结构化 VS 面向对象

从用户角度来看的系统

定义对象、属性和关系。



表示事件对系统状态的影响

表示数据在系统内是如何转换的

8.2 基于场景建模

在描述需求建模时常用哪些观点

“[用例]只是帮助定义系统之外（参与者）存在什么以及系统应完成什么（用例）。” Ivar Jacobson

(1) 我们应该写什么？

(2) 我们应该写多少？

(3) 我们编写说明应该多详细？

(4) 我们如何组织说明？

8.2.1 创建初始用例

编写什么？

- 起始和导出——提供了开始编写用例所需要的信息..
- 运用需求收集会议、QFD和其他需求工程机制
  - 确定利益相关者
  - 定义问题的范围
  - 说明整体的运行目标
  - 建立优先级顺序
  - 概述所有已知的功能需求
  - 描述系统将处理的信息(对象)。
- **开始开发一系列用例时，应列出特定参与者执行的功能或活动。**

8.2.1 创建初始用例

编写多少？

- 在与相关利益人进一步会话取得进步后，需求收集团队为每一个功能说明开发用例。
- 通常以一种非正式的叙述性的方式表达用例。
- 若要求更正式的方式，则以一种类似于被提及的结构化的格式重写相同的用例。

8.2.1 创建初始用例

- 一个描述系统“使用线程”的场景
- 责任人代表人或设备在系统运行时所扮演的角色
- 用户可以在给定的场景中扮演若干不同的角色
- 通常这个步骤连续的陈述不考虑其他可能的交互，有时被称作主场景（P102）

8.2.2 细化初始用例

- 参与者完成的主要任务和功能是什么？
- 参与者将获取、产生或改变哪些系统信息？
- 参与者必须通知系统有关外部环境的改变吗？
- 参与者希望从系统获取什么信息？
- 参与者希望得知会有意料之外的变更吗？

评审用例

- 用例首先用描述性风格编写，若需要正式形式，再映射到模板上。
- 每个主场景都应该进行评审和细化，以查看有没有可能有可替代的交互方式
  - 在这一步，参与者能做一些其他动作吗？
  - 在这一步，参与者有没有可能遇到一些其他错误条件？如果有？是什么错误条件？
  - 在这一步，参与者有没有可能遇到一些其他行为？如果有？是什么行为？

8.2.3 编写正式用例

- 用例图显示一组用例，参与者以及它们之间的关系。用于需求分析阶段，即确定“谁使用系统以及做什么”。
- 画好用例图是由软件需求到最终实现的第一步。

8.2.3 编写正式用例

- 用例图包括以下3方面内容。

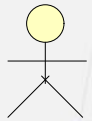
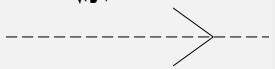
  （1）用例（Use Case）

  （2）参与者（Actor）

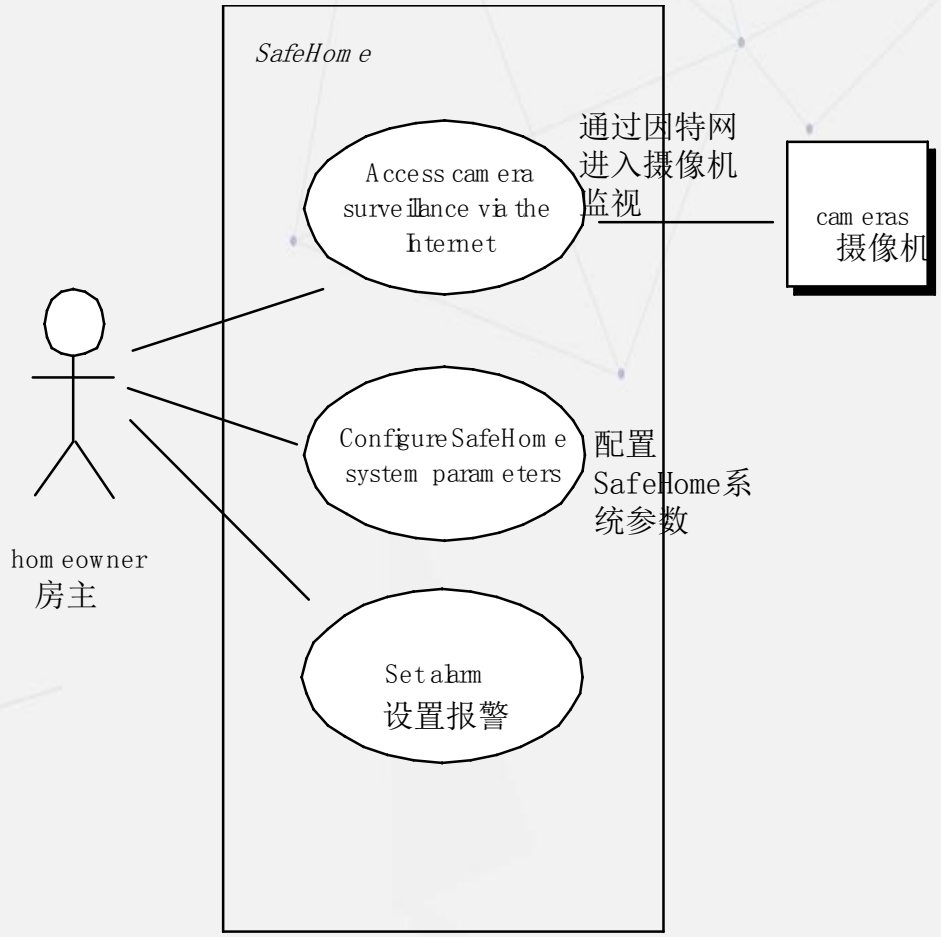
  （3）关系（依赖、泛化以及关联）

- 用例图的主要元素是用例和参与者：所谓用例是指对系统提供的功能(或系统的用途)的描述；参与者是指可能使用用例的人或外部系统。两者的关系是**“谁使用了哪个用例”**.
- 用例图着重于从系统外部参与者的角度来描述系统需要提供哪些功能，并且指明这些功能的参与者是谁。

8.2.3 编写正式用例

名称	图例	说明
角色 actor	 角色名称	代表与系统交互的实体。角色可以是用户、其它系统或者硬件设备。在用例图中以小人表示。
用例 use case	 用例名称	定义了系统执行的一系列活动，产生一个对特定角色可观测的结果。在用例图中，用例以椭圆表示。“一系列的活动”可以是系统执行的功能、数学计算、或其它产生一个结果的内部过程。活动是原子性的。活动的原子性可以决定用例的粒度。用例必须向角色提供反馈。
关联 association	 <<原型>>	表示用户和用例之间的交互关系。
依赖		用例与用例之间的依赖关系。用带箭头的虚线表示。

8.2.3 编写正式用例

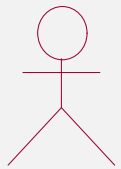

SafeHome
Access camera surveillance via the Internet
通过因特网进入摄像机监视
cameras
摄像机
Configure SafeHome system parameters
配置SafeHome系统参数
Set alarm
设置报警
homeowner
房主

异常处理

- 描述这样一种情景（失败或用户选择），该场景导致系统展示出某些不寻常的行为
- 使用头脑风暴来合理地完成每个用例中一系列的异常处理。
- 在这些用例中是否有某些具有“确认功能”的用例出现？
  - 在这些用例中是否有支持功能（或参与者）的应答失败？
  - 性能差的系统是否会导致无法预期或不正确的用户活动？
- 开始开发一系列用例时，应列出特定参与者执行的功能活动。
- 处理异常可能需要创建额外的用例。

8.2.3 编写正式用例

1、参与者（Actor）



Customer

(from Actor)

- 参与者（Actor）是系统外部的一个实体（可以是任何的事物或人），它以某种方式参与了用例的执行过程。
- 参与者通过向系统输入或请求系统输入某些事件来触发系统的执行。参与者是系统之外，透过系统边界与系统进行有意义交互的任何事物（人或事物）。
- 在处理参与者时，应考虑其参与系统的身份，而不是人名或工作名。
- 在UML中，参与者用人形图符表示。
- 但参与者未必是人，可以是一个外部系统。

参与者的识别思路

- 谁使用该系统
- 谁改变系统的数据
- 谁从系统获取信息
- 谁需要系统的支持以完成日常工作任务
- 谁负责维护、管理并保持系统正常运行
- 系统需要应付那些硬件设备
- 系统需要和那些外部系统交互
- 谁对系统运行产生的结果感兴趣

案例：库存管理系统

某汽车制造厂需要一套库存管理系统，该系统实现的业务：

（1）生产工人根据生产计划领取物料

（2）库存操作员根据生产系统的派单准备，交付给领料工人，余料即时归还库房

（3）库房管理人员定期盘点库存，通知供应商供货，对长期积存的货物，申请退货。

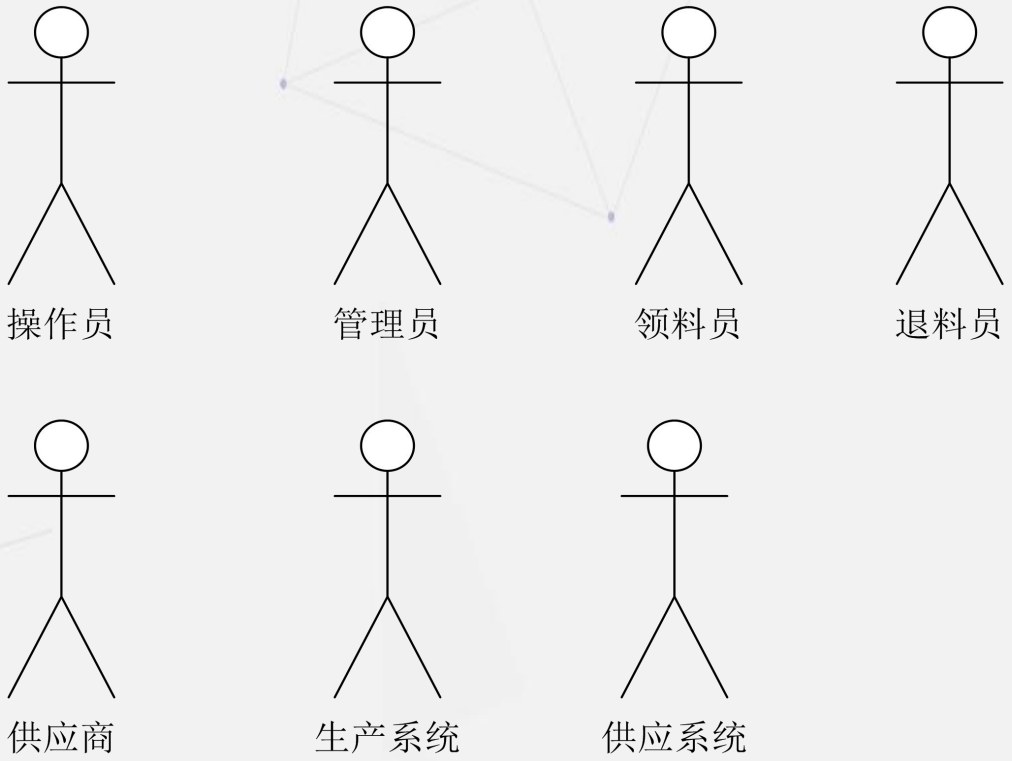
8.2.3 编写正式用例

识别思路

- 谁使用该系统 — 操作员，管理员
- 谁改变系统的数据 — 操作员，管理员
- 谁从系统获取信息 — 操作员，管理员
- 谁需要系统的支持以完成日常工作任务 — 领料员,退料员,操作员,管理员,供应商

- 谁负责维护、管理并保持系统正常运行 — 管理员
- 系统需要应付那些硬件设备
- 系统需要和那些外部系统交互 — 生产系统, 供应系统
- 谁对系统运行产生的结果感兴趣 — 操作员,管理员,领料员,退料员

库存管理系统的参与者



5.2.3 编写正式用例

•用例描述了系统的功能需求，是系统的一组动作序列的描述.

•用例的本质是用户与计算机之间的一次交互作用。在UML的概念中用例是系统作出的一系列动作,而参与者能够察觉到这一系列动作的结果。

•UML中用例用一个椭圆来表示，用例的名字可以写在椭圆的内部或下方。

2、用例（Use Case）

如何识别用例？

- 识别用例最好的办法就是从分析系统的参与者开始，先列出所有的参与者，再根据每个参与者列出与它有关的用例。在识别用例的过程中，通过以下的几个问题可以帮助识别用例：

  （1）参与者希望系统提供什么功能？

  （2）系统是否存储和检索信息？如果是，这个行为由哪个参与者触发?

  （3）当系统改变状态时，通知参与者吗？

  （4）存在影响系统的外部事件吗？

  （5）是哪个参与者通知系统这些事件？

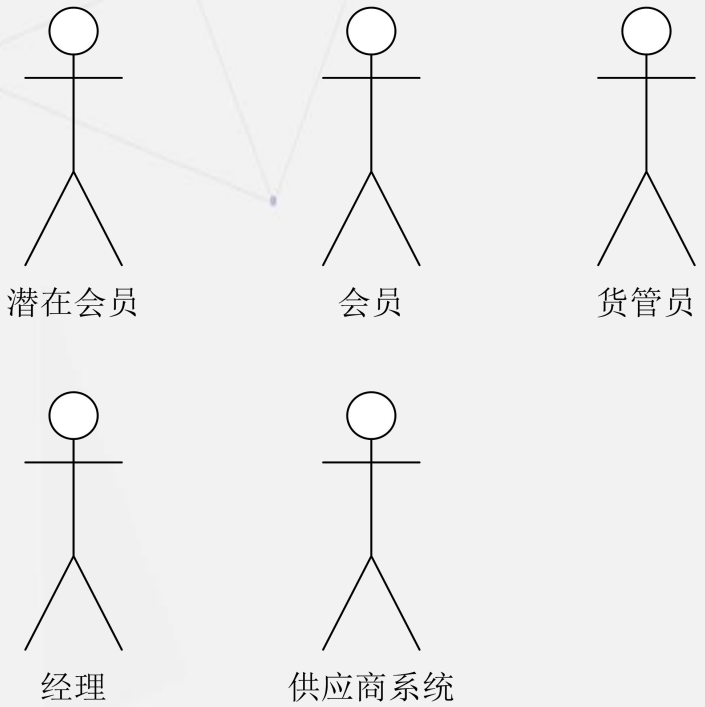
8.2.3 编写正式用例

- 用例分析是处于系统的需求分析阶段，这个阶段应该尽量的避免去考虑系统实现的细节问题。
- 也就是说，用例描述的是一个系统做什么，而不是怎么做。

案例2：零件销售系统

- 在基于Web的新系统中，顾客可以通过Internet进行购买。
- 顾客先预付一定金额存入内部账户中成为会员，然后才能购买零件。顾客可以根据自己所知道的零件的形状，大小、零件编号等指标，搜索出所需要的零件。结帐使用内部帐户支付。系统根据会员提供的送货地址和订购数量，从库存中搜索出离送货地址最近的供应商，通知供应商发货。
- 内部工作人员不定期地根据供应商方面的价格变动，对某些零件的销售价格进行更新。每个星期，各个供应商会把记录自己最新库存情况的Excel文件寄来，系统根据这些文件更新库存信息。
- 因简化的需要，以下因素略去不考虑：折扣，延迟交货...

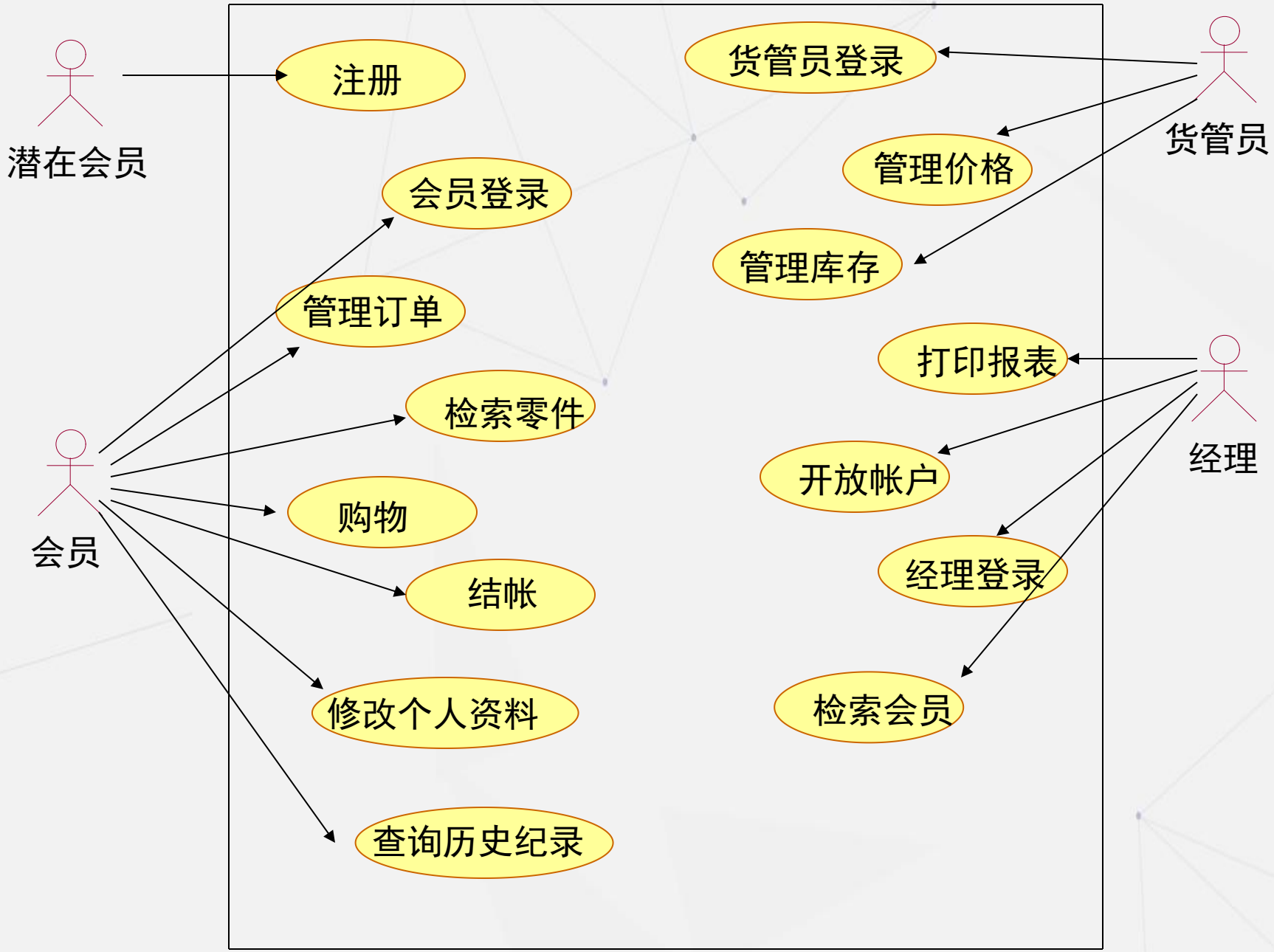
案例2：零件销售系统的参与者



案例2：零件销售系统


潜在会员
会员
货管员
经理
注册
会员登录
管理订单
检索零件
购物
结帐
修改个人资料
查询历史纪录
货管员登录
管理价格
管理库存
打印报表
开放帐户
经理登录
检索会员

案例2：零件销售系统


潜在会员
注册
会员
会员登录
管理订单
检索零件
购物
结帐
修改个人资料
查询历史纪录
货管员登录
管理价格
管理库存
货管员
打印报表
开放帐户
经理登录
检索会员
经理

关系？

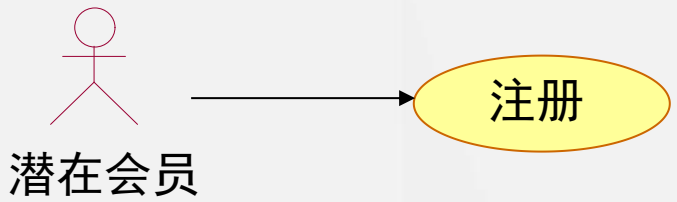
- **参与者与用例之间**
  - 关联关系
- **用例与用例之间**
  - 包含关系 (include)
  - 扩展关系 (extend)
  - 泛化关系 (generalization)
- **参与者与参与者之间**
  - 泛化关系 (generalization)

关系

关系—参与者与用例之间

- 关联关系

描述参与者与使用用例之间的关系。在UML中，关系用实线表示，实线可以有箭头，也可以没有箭头。



例：参与者与用例通过关联相连。

用例间的关系——包含关系

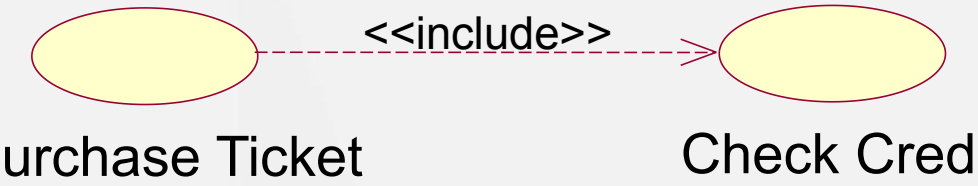
1）包含关系(include)

包含关系中一个用例总是使用另一个用例的功能

如果两个以上用例有大量一致的功能，则可以将这个功能分解到另一个用例中。

一个用例的功能太多时，可以用包含关系建模两个小用例。

包含关系中基用例本身是不完整的。



用例“Check Credit” 检查输入的信用卡号是否有效，信用卡是否有足够的资金。

用例间的关系——扩展关系

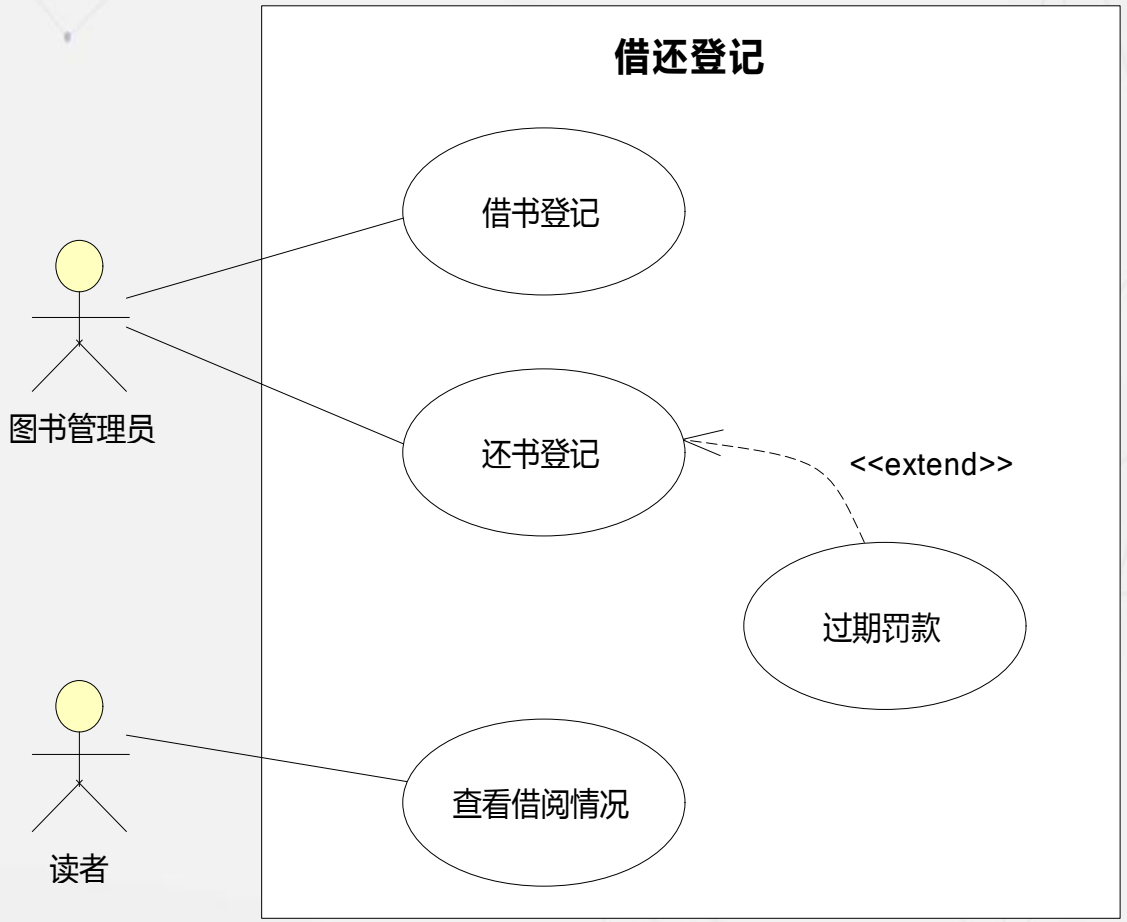
2）扩展关系(extend)

a. 扩展关系允许一个用例（可选）扩展另一个用例的功能。

b. 当某个新用例在原来的用例基础上增加了新的步骤序列，则原用例被称作基用例，这种关系被称为扩展关系。

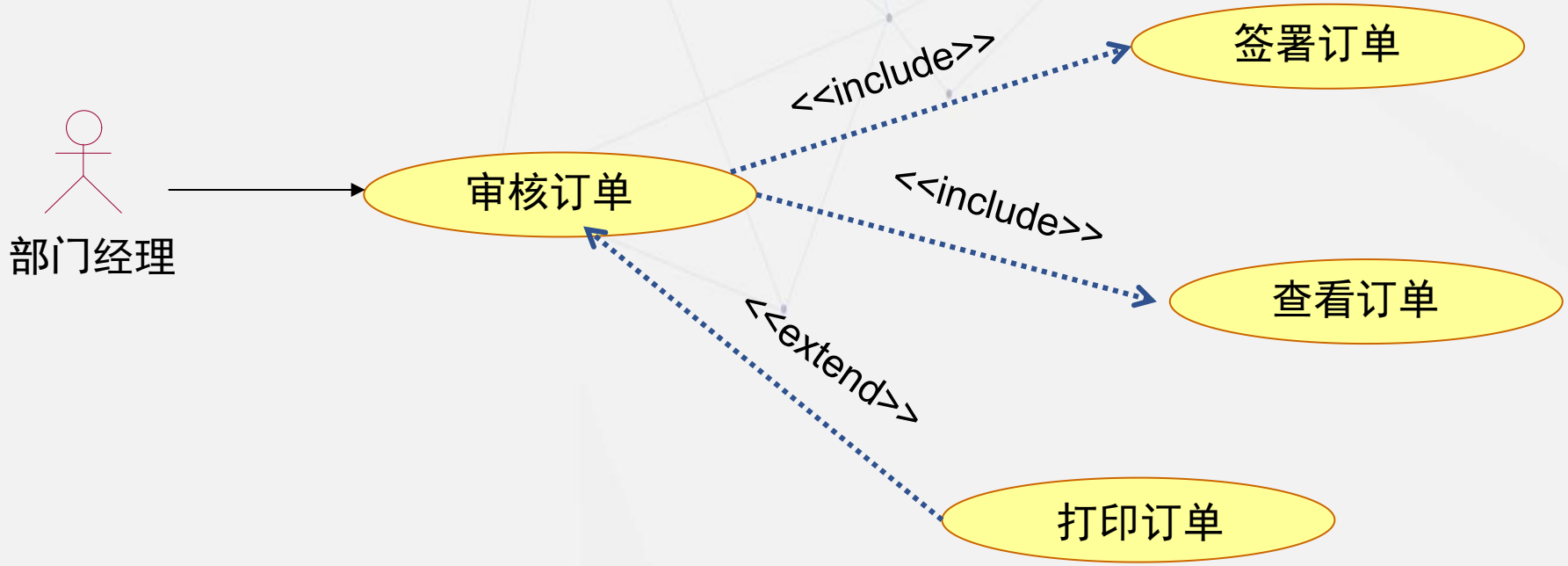
c. 基用例可以单独存在，但在一定的条件下，他的行为可以被另一个用例的行为延伸。扩展只能发生在基用例的序列中某个特定的点上，这个点叫扩展点。

d. 扩展关系中基用例本身是完整的。



包含关系与扩展关系的区别

包含关系与扩展关系的区别



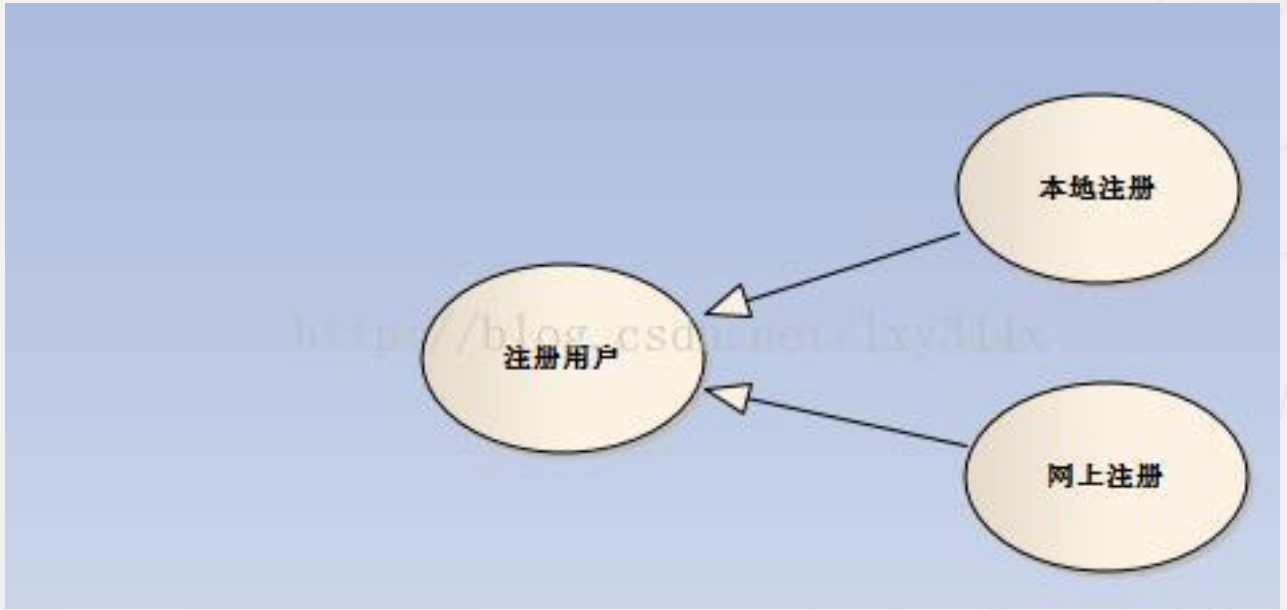
联系：都是从现有的用例中抽取出公共的那部分信息，作为一个单独的用例，然后通后过不同的方法来重用这个公共的用例，以减少模型维护的工作量。

区别：扩展关系中基本用例的基本流执行时，扩展用例不一定执行，即扩展用例只有在基本用例满足某种条件的时候才会执行。包含关系中基本用例的基本流执行时，包含用例一定会执行。

用例间的关系——泛化关系

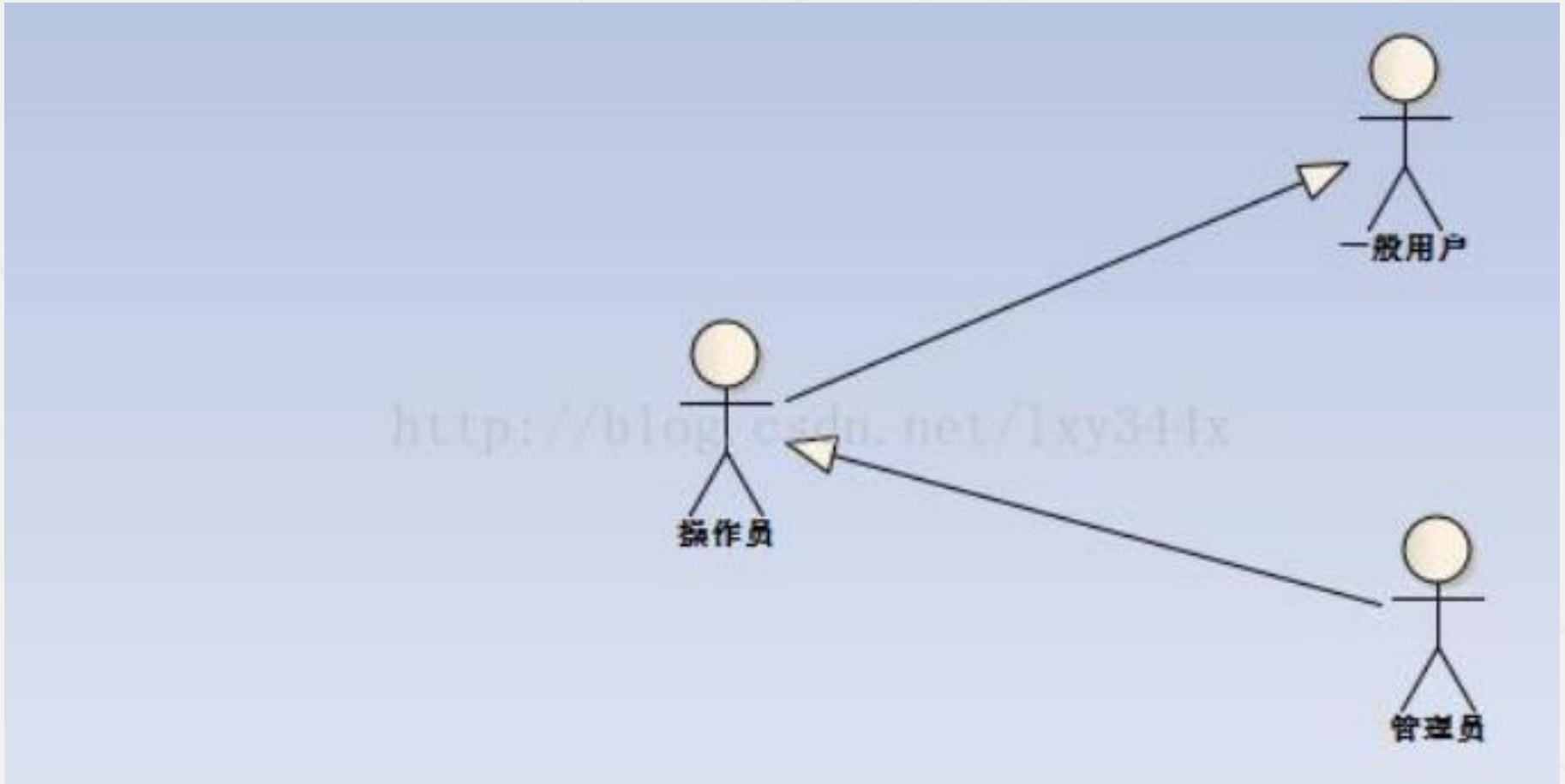
3）泛化关系（也称类属或概括关系）

泛化关系其实是子类与父类的关系。象类之间的泛化关系一样，用例和参与者也可以继承另一个用例和参与者。



用例图里的关系

参与者与参与者之间：泛化关系



区别

包含以及扩展过程与泛化过程类似，但三者对用例关系的优化侧重点是不同的。如下：

- 泛化侧重表示子用例间的互斥性；
- 包含侧重表示被包含用例对Actor提供服务的间接性；
- 扩展侧重表示扩展用例的触发不定性；详述如下：

用例图里的事件流

用例的描述——事件流

- 建立实际的系统，还需要更多的细节，这些细节写在事件流文档中。
- 事件流是通过文字描述一个用例的行为，说明用例的逻辑流程。发起用例的参与者是谁，用例的前置条件是什么，主事件流，其他事件流和完成后的后置条件是什么，从用例中获益的参与者是谁。
- 事件流包括：简要说明、前置条件、主事件流、其他事件流和后置条件。
  - 简要说明：每个用例应有一个相关说明，描述该用例的作用。
  - 前置条件（前提条件）：列出开始用例之前必须满足的条件。
  - 主事件流：显示用例从开始到结束的完整的正常流程。
  - 其他事件流：显示异常条件或错误。
  - 后置条件（事后条件）：用例结束后系统应具备的状态。

一般用例的格式

用例名称
用例编号
创建人　　创建时间
版本号　　修改时间
主要参与者
次要参与者
简要描述
触发条件
前置条件
事件流
后置条件
可选事件流
例外
非功能性需求
假设
备注
补充性规格说明

两个事件流的例子……

8.3 补充用例的UML模型

- 很多基于文本的需求建模情景不能简明扼要地传递信息
- 需要能从大量的UML图形模型中选择和是的表达方式

（1）活动图

（2）泳道图

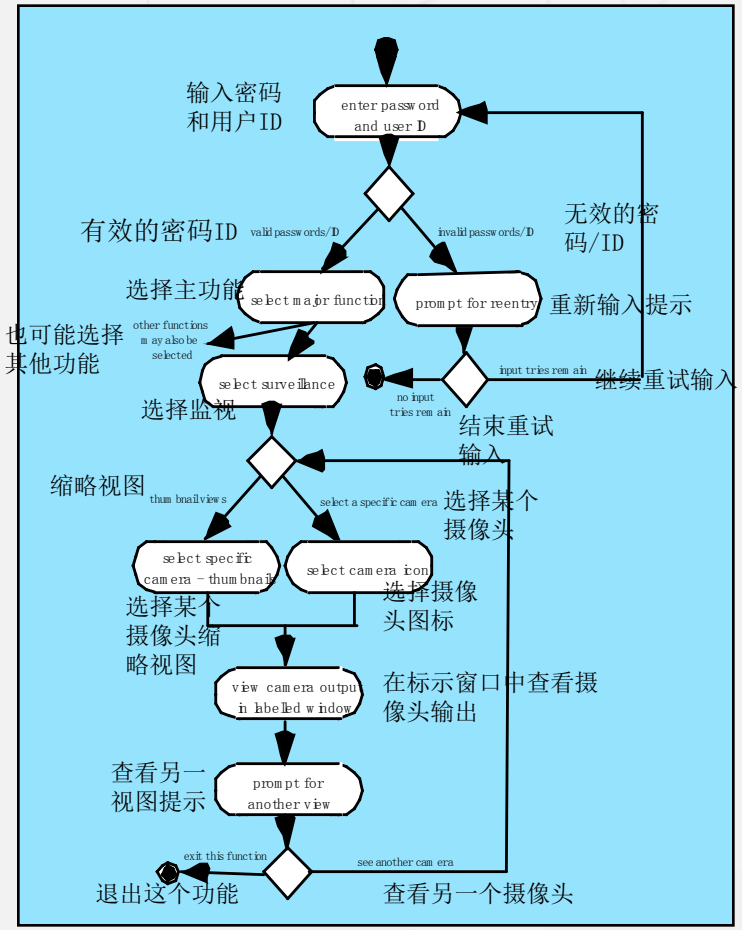
8.3.1 活动图

类似流程图，补充用例

- 在统一建模语言UML中，活动图用于描述系统行为，在需求阶段，可以配合用例图说明复杂的交互过程。
- 活动图由开始点、活动、转换和结束点组成。一个活动图只能包含一个开始点，可以有多个结束点。开始点、活动、结束点之间通过转换连接。

8.3.1 活动图

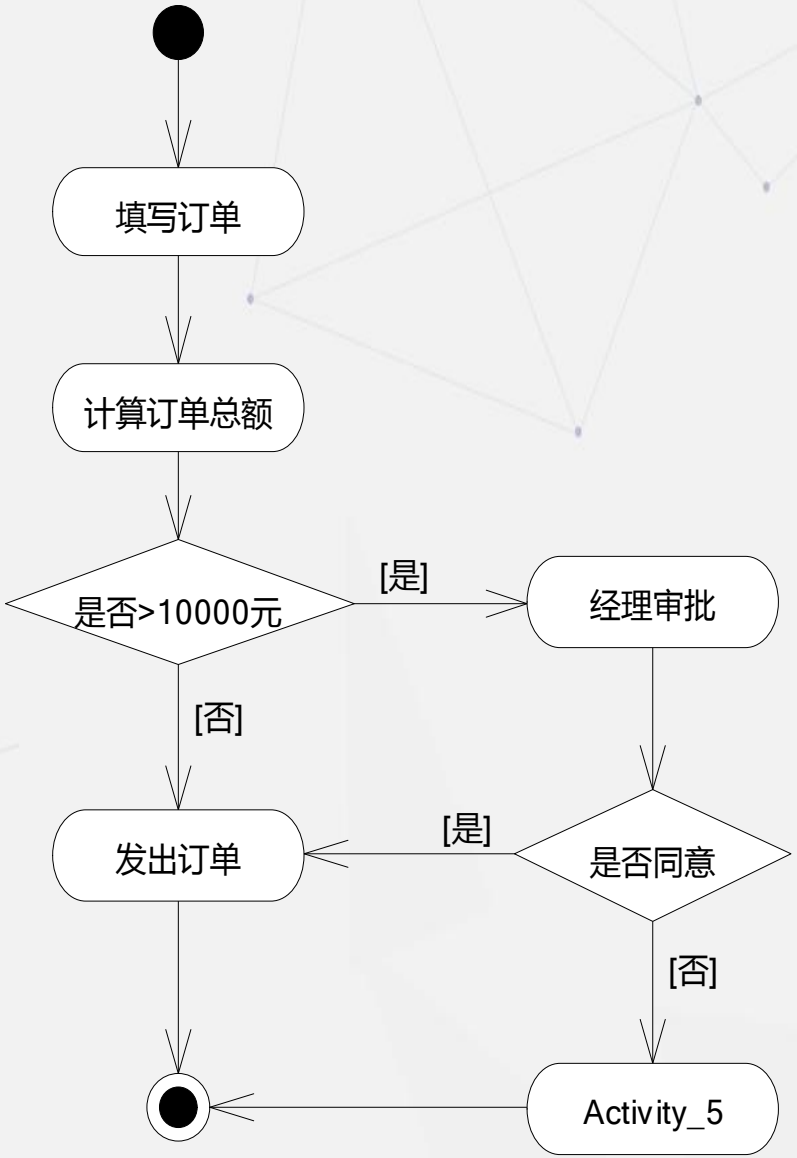
类似流程图，补充用例



8.3.1 活动图

- 学习过C语言或别的程序设计语言的读者一定接触过流程图，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，它描述系统的活动、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）**

8.3.1 活动图



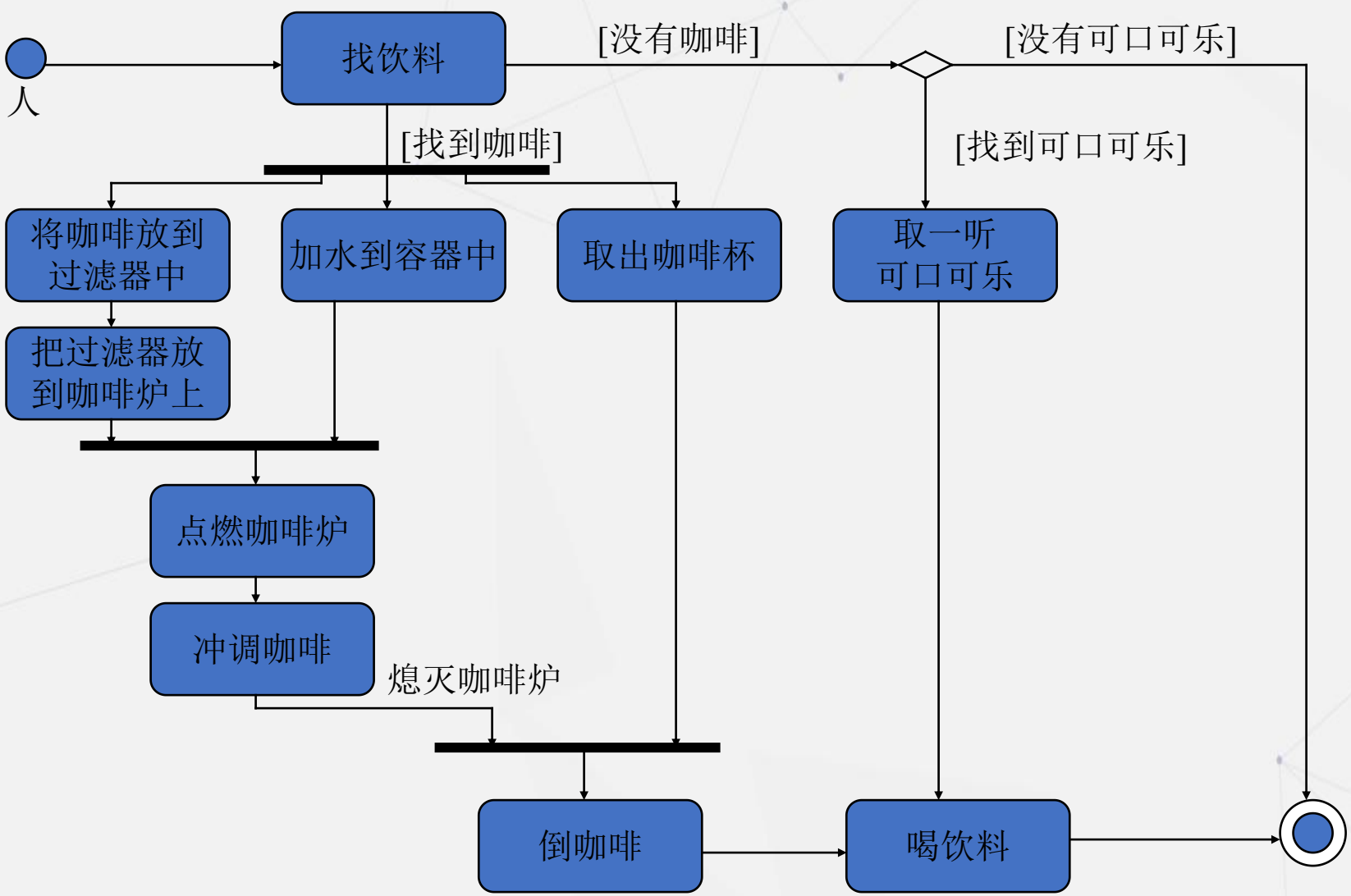
活动图的基本概念和组成

- 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。

几个活动图的基本概念

- **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。
- **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。
- **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。

几个活动图的基本概念


人
找饮料
[没有咖啡]
[没有可口可乐]
[找到咖啡]
[找到可口可乐]
将咖啡放到过滤器中
加水到容器中
取出咖啡杯
取一听可口可乐
把过滤器放到咖啡炉上
点燃咖啡炉
冲调咖啡
熄灭咖啡炉
倒咖啡
喝饮料

8.3 补充用例的UML模型

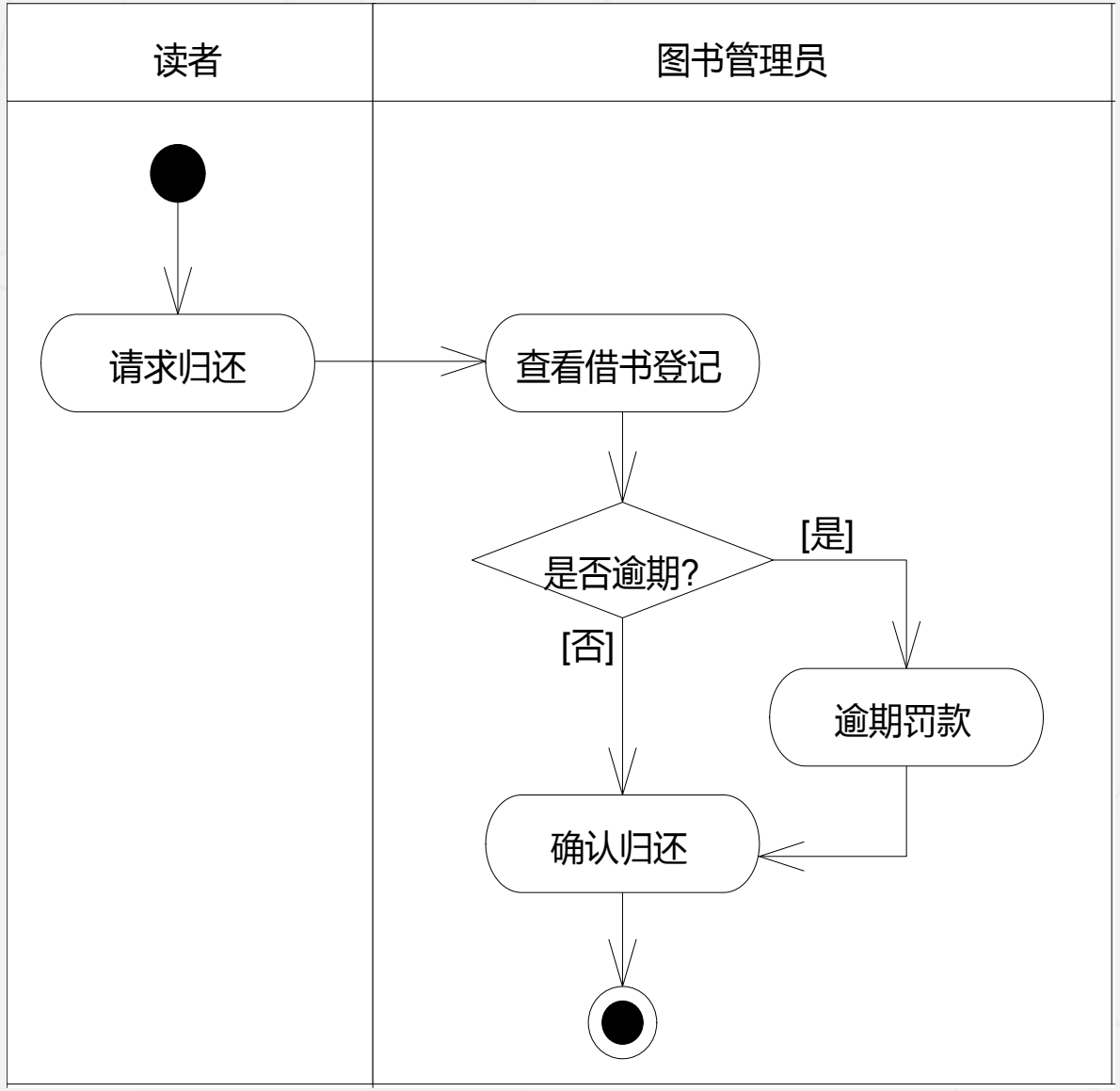
8.3.2 泳道图

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。

泳道图是活动图的一种变形

可让建模人员表示用例所描述的活动流，同时指示哪个参与者（如果在特定用例中涉及了多个参与者）或分析类是由活动矩形所描述的活动来负责。



一个例子……

8.3 补充用例的UML模型

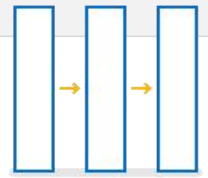
活动图用于对系统的动态行为建模。描述了从活动到活动的流。

优点

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。

缺点

- 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。



用途

- 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。

5.3 补充用例的UML模型

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。

如果涉及到多个对象，则进化为泳道图

活动图与流程图的区别

(1)流程图着重描述处理过程，它的主要控制结构是顺序、分支和循环，各个处理过程之间有严格的顺序和时间关系。而活动图描述的是对象活动的顺序关系所遵循的规则，它着重表现的是系统的行为，而非系统的处理过程。

(2)活动图能够表示并发活动的情形，而流程图不行。

(3)活动图是面向对象的，而流程图是面向过程的。

8.4 数据建模

实体/关系对是数据模型的基础

- 检查数据对象的独立处理
- 注意力集中在数据域
- 创建客户级抽象的模型
- 指出数据对象之间的相互联系

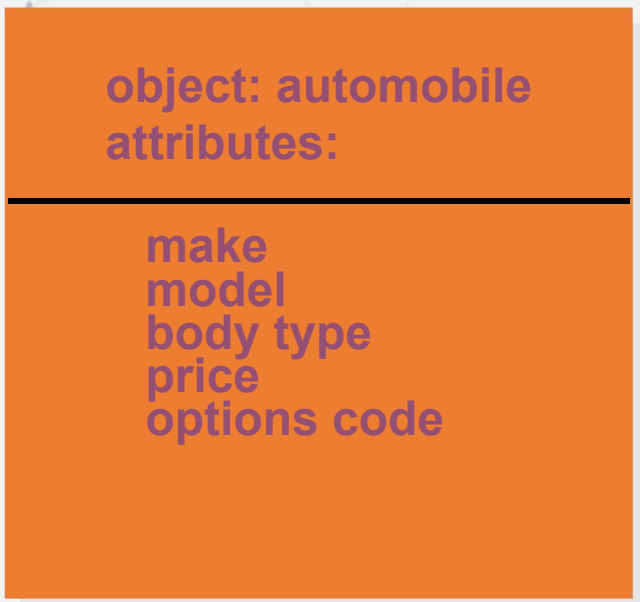
实体关系图ERD：数据对象、数据属性、数据关系

什么是数据对象

- 数据对象必须由软件理解的复合信息表示。
- 复合信息—具有若干不同的特征或属性的事物
- 可能是外部实体（例如产生或使用信息的任何东西），事物（例如报告或显示），偶发事件（例如电话呼叫）或事件（例如报警），角色（例如销售人员），组织单位（例如财务部），地点（例如仓库）或结构（例如文件）。
- 描述包括了数据对象及其所有属性。
- 数据对象只封装数据——在数据对象中没有操作数据的引用

8.4.2 数据属性

数据对象包含一组属性，它们充当一个方面、质量、特性或对象的描述



什么是关系

- 数据对象可以以多种不同的方式与另一个数据对象链接。
  - 在person和car之间建立联系，因为这两个对象之间是相互联系的。
    - 人 拥有 车
    - 汽车 驾车投保 人
- 关系“拥有”和“驾车投保”定义了person和car之间的相关连接。
- 一个关系中可以存在几个实例
- 对象之间可以以多种不同的方式联系

- 一对一关系（1：1）数据对象A 的一次出现只能关联到数据对象B的一次出现，反之亦然。一般也称为一 一对应关系；
- 一对多关系（1：N）数据对象A的一次出现能关联到数据对象B的一次或者多次出现，但是数据对象B的一次出现只能关联到数据对象A的一次出现。例如一个客户可以有一个或多个订票单，但是一个订票单只能属于一个用户。
- 多对多关系（N：N）数据对象A的一次出现能关联到数据对象B的一次或多次出现，数据对象B的一次出现也能关联到数据对象A的一次或多次出现。

8.4.3 数据关系

基数与形态

数据对象除了彼此有关系外，这个绑定在一起的关系和发生的次数也有关，也就是说一对相关的对象发生的次数与另一个对象发生的次数有关。于是产生了基数的概念。两个相互关联的对象的次数关联如下：

一对一关系（1：1）

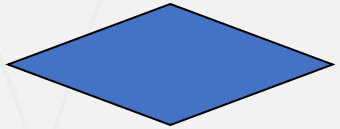
一对多关系（1：N）

多对多关系（N：N）

- 基数定义了在一个关系中可以参与对象关联的最大数目。
- 数据模型需要定义在一个给定的关系中数据对象出现的次数

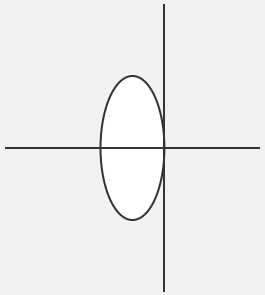
ERD图组成符号

数据对象

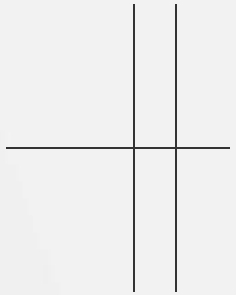


关系

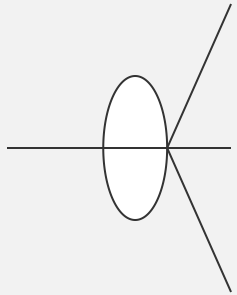
对象属性



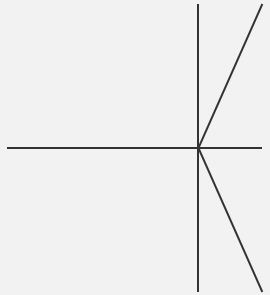
0：1



1：1



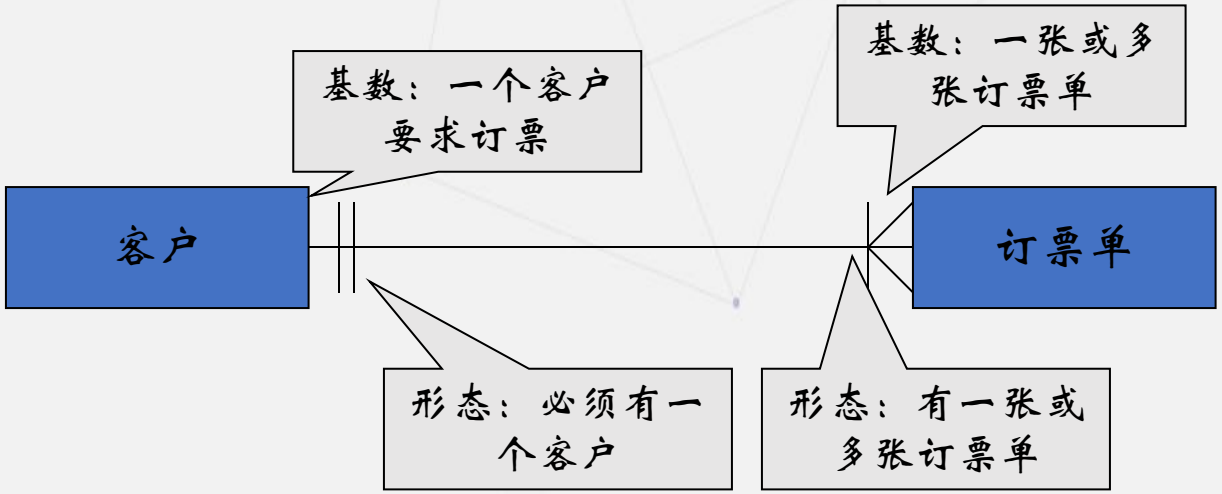
0：m



1:m

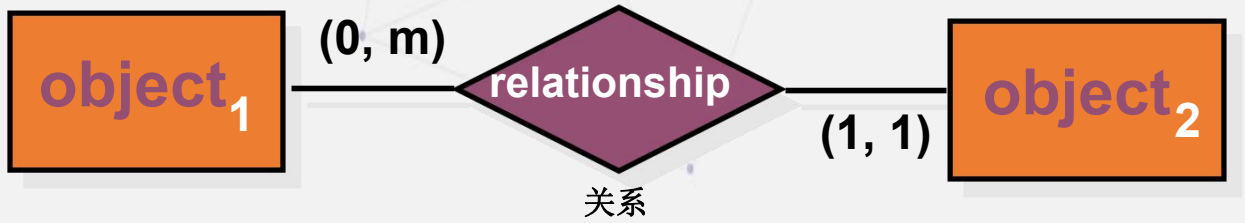
8.4 数据建模

8.4.3 数据关系



- 左边的短竖线第一条表示基数为1，即客户和订票单是1：N的关系；
- 第二条竖线表示形态为1，即发生订票时必须有一个客户；
- 右边的短竖线表示形态为1，即发生订票关系时必须填写一张或多张订票单
- 右边的三叉线是基数，表示1：N的关系，即可填写一张或多张订票单

One common form: 一种常见形式：



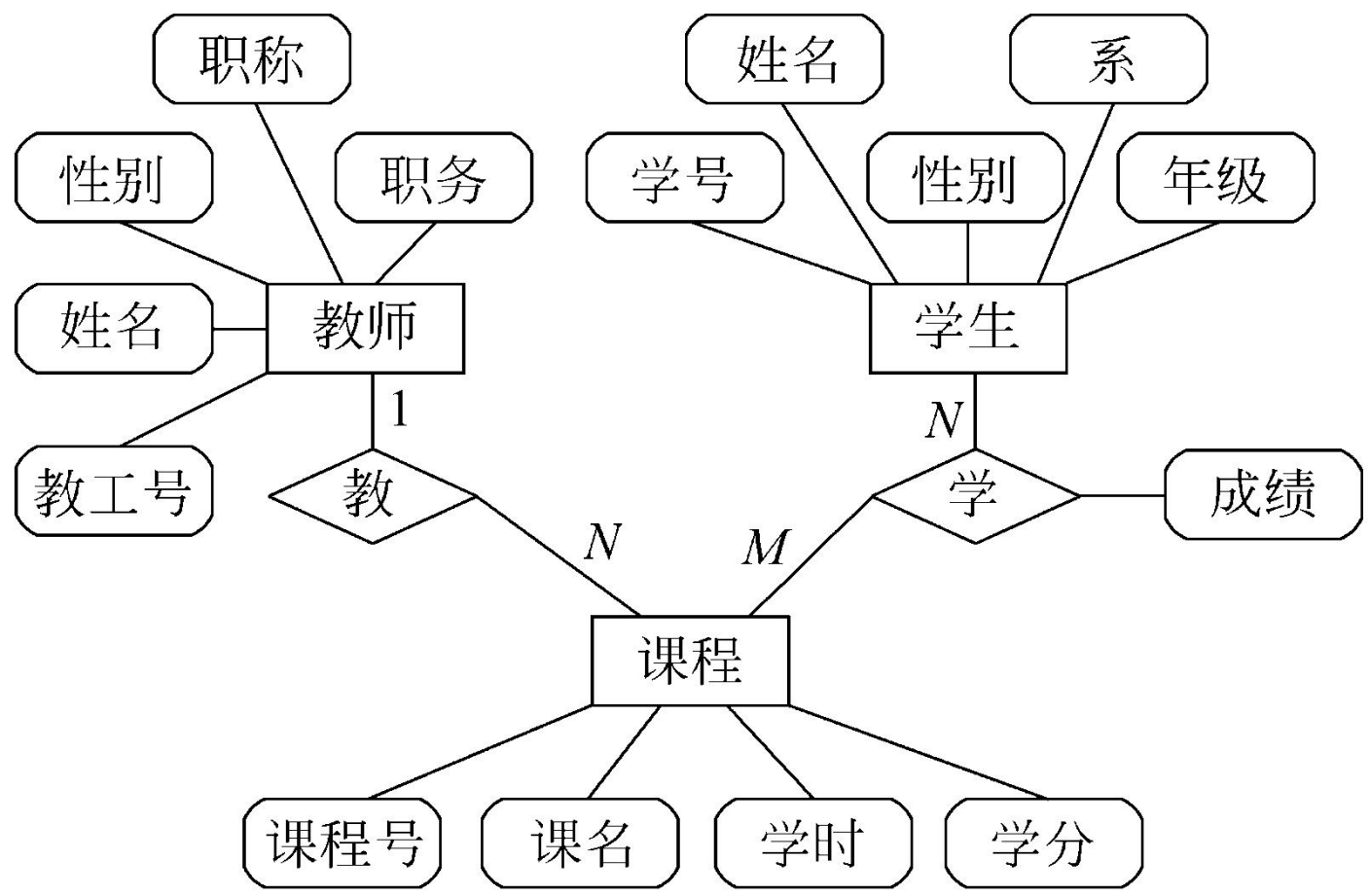
Another common form: 另一种常见形式：

8.4 数据建模

- 第1步——建模所有数据对象(实体) 和它们与另一个对象的“联系”
- 第2步——建模所有的实体和关系
- 第3步——建模所有实体、关系及进一步提供的属性

8.4 数据建模

建模

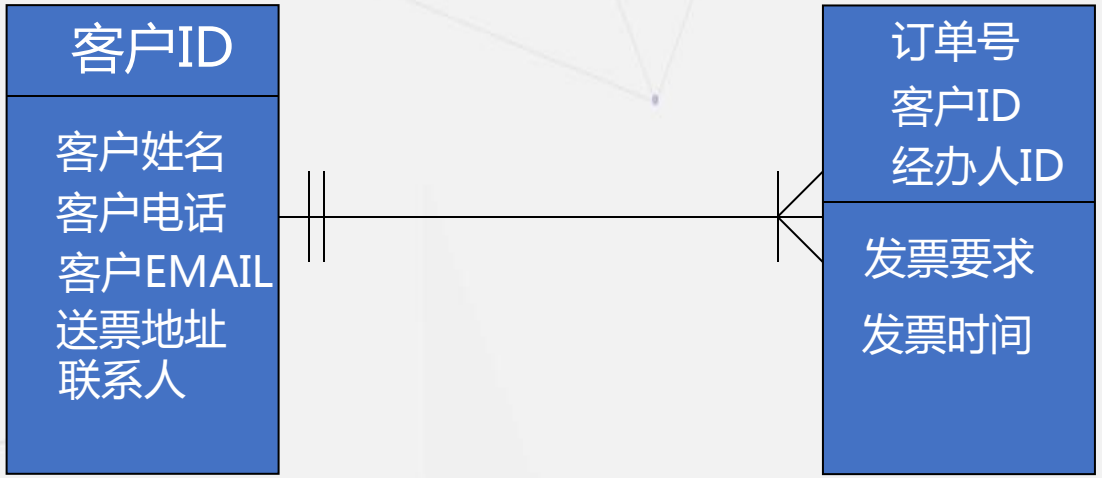


关键字

- 数据对象的属性组合中总有一个或几个属性是主要的，可以定义为数据实体的标识符。所谓主要的是指这些属性是经常要进行检索的属性，或者其它属性都依赖它而存在。
- 实体中某些关键字是继承其它对象的关键字而来的，这些关键字称为外部关键字。例如“演出”和“演出公司”是依靠演出公司的“演出公司ID”建立关联的。因为演出是由演出公司举办的。而演出对象中的属性演出公司名称来自于演出公司，是一个外部关键字。通过该外部关键字将两个对象联系起来。
- 在数据库中，确定关键字的另外一个好处是可以通过关键字建立索引。

8.4 数据建模

前述属性、基数、关键字等是ER图的基本元素。



5 建立数据模型的一般步骤

迭代

A 辩识数据实体：在需求过程中要收集业务过程中涉及到的事物，输入数据和输出数据。列出系统中的数据对象。

B 找出属性和主关键字：对每个数据对象列出系统 需要的属性，然后找出关键字，包括主关键字、外来关键字。

C 建 立 关 联 ：根据关键字建立关联。关联一般有两种，一般/具体；整体/部分。

D 建立ER图：对每个数据实体进行关联考察并确定其基数和形态，画出ER图。

E 评审数据模型 ：