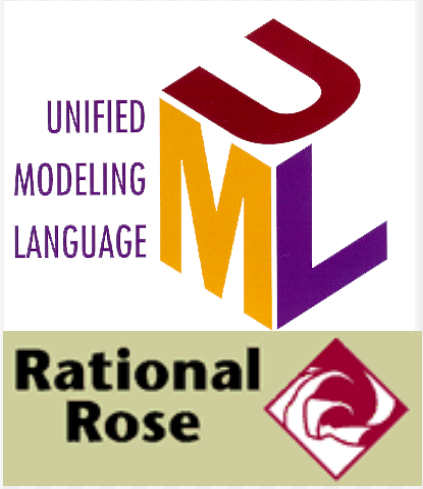
UML 及其建模工具的使用



内容提要

- UML的基本概念
- UML建模
  - 静态建模
  - 动态建模
  - 物理架构
  - 建模步骤

火龙果•整理
uml.org.cn

一、统一的建模语言方法

由Rumbaugh 、 Booch 、 Jacobson 提出的统一建模语言 **(Unify Modeing Language**简称**UML)**

UML是一种定义良好，易于表达，功能强大且普遍实用的建模语言。

UML的产生

- 1988年到1992年是面向对象方法学蓬勃发展的时期，人们从各自的经历和软件开发的经验提出了各种面向对象的开发方法，代表的有：
  - Sally Shlaer 和 Steve Mellor以信息模型化方法作为基础，并为目标系统增设了状态模型和过程模型；
  - Peter Coad 和 Ed Yourdon则在信息模型化、面向对象的程序设计语言和基于知识的系统的基础上，建立了他们的OOA和OOD，主要工具是类与对象图、对象状态图和服务图；
  - HP公司的Fusion开发方法。

UML的产生

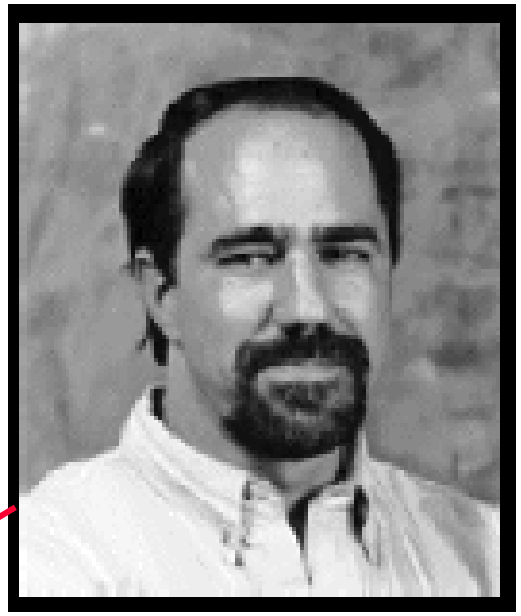
- Wirfs-Brock的职责驱动设计(Responsibility-Driven Design)，也称类-职责-协作Class-Responsibility-Collaboration (CRC) cards，用类所承担的责任来描述系统，利用责任把封装的概念带到分析与设计活动中去；
- Grady Booch在Rational软件公司开发Ada系统作了许多构件(Component)，并以此由底向上构筑大型软件系统，即OOD方法；
- Jim Rumbaugh在通用电子(General Electric)领导一个研究小组，提出了对象建模技术(OMT)方法，通过面向对象的三种模型：对象模型、动态模型和功能模型，从不同角度对系统进行描述；

UML的产生

- Ivar Jacobson和他的 Objectory 公司开发了OOSE（Object Oriented Software Engineering)面向对象的软件工程，利用Use Cases来表达系统要求。
- 1994年任职于Rational公司的Grady Booch首先联合Jim Rumbaugh加盟Rational软件公司开始了统一OO方法学和工具的历程。以融合Booch和OMT方法的UML开发开始。1995年10月UML0.8发布。1995年秋，Ivar Jacobson和他的 Objectory 公司加盟Rational，UML中加入了OOSE方法，使其有可能最集中地包容当今最适用的各种OO方法。1996年，UML0.9版本发布，1997年1月，UML1.0被提交给OMG组织，作为软件建模语言的候选，1997年11月7日，UML1.1正式被OMG组织采纳为业界标准。UML经历了1.2，1.3，1.4，目前UML2.0版本正在制定。

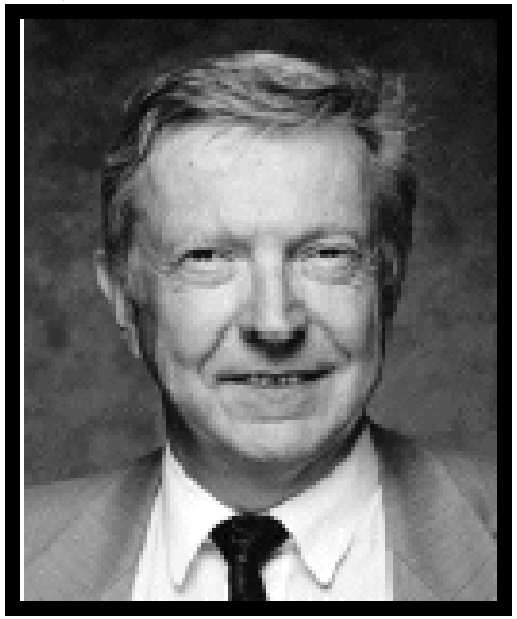
Rational三剑客

Jim Rumbaugh



Grady Booch

Ivar Jacobson



UML的基本概念[1]

- UML简介
- UML的目标
- UML概念范围

UML的基本概念[2]—UML简介

- UML(Unified Modeling Language)是一种构建软件系统和文档的通用可视化建模语言。
- UML能与所有的开发方法一同使用，可用于软件开发的整个生命周期。
- UML能表达系统的静态结构和动态信息，并能管理复杂的系统模型，便于软件团队之间的合作开发。
- UML不是编程语言，但支持UML语言的工具可以提供从UML到各种编程语言的代码生成，也可以提供从现有程序逆向构建UML模型。

UML的基本概念[3]—UML简介

- UML并不是万能的，它是一种离散的建模语言，对于特定的领域，比如：GUI、VLSI电路设计或基于规则的人工智能，用特定的语言和工具可能更合适。

UML的基本概念[4]— UML的目标

- 最重要目标：UML是所有建模人员可以使用的通用建模语言。它包含主流建模方法的概念，从而可以替代现有的软件分析和设计方法，比如：OMT,Booch,OOSE等。
- UML不是完整的开发方法，它不包括逐步的开发流程，但它提供所有必要的概念，具备足够的表达能力。
- UML的另一个目标是：能尽量简洁地表达系统的模型。

- UML的目标是：

- 1．易于使用，表达能力强，进行可视化建模；
- 2．与具体的软件开发过程无关，可应用于任何软件开发的过程；
- 3．简单、可扩展。扩展无需对核心概念进行修改；
- 4．为面向对象的设计和开发过程中涌现出的高级概念（例如协作、框架、组件）提供支持；
- 5．吸收当代最好的软件工程实践经验；
- 6．可升级，具有广阔的适用性和可用性；
- 7．有利于面向对象工具的市场成长。

UML语言包含三方面内容：

1．UML基本图素：它是构成UML模型图的基本元素。

- 2．UML模型图：它由UML基本图素按照UML建模规则构成。

- 3．UML建模规则：UML模型图必须按特定的规则有机地组合而成,从而构成一个有机的、完整的UML模型图（well-formed UML diagram）。

一、基本图素分为四类：

- 结构模型图素、
- 行为模型图素、
- 成组模型图素、
- 其他模型图素、
- 公用机制。

UML的基本概念[5]— UML概念范围

- UML概念可以划分为以下范围：
  - 系统需求
  - 静态结构
  - 动态行为
  - 交互行为
  - 物理实现
  - 各种图之间的关系
  - 模型组织
  - 扩展机制

UML的基本概念[6]— UML概念范围

- 系统需求
  - 用例视图(Use Cases View)从外部用户的角度来描述系统的行为，它将系统功能划分为对用户有意义的事务，这些事务被称为用例，用户被称为执行者，用例视图也就是描述活动者在各个用例中的参与情况，它指导所有的行为视图。

UML的基本概念[7]— UML概念范围

- 静态结构:
  - 静态视图(Static View)，一个模型必须首先定义各种事物的内部特征和相互之间的关系，应用概念建模成类，类描述事物的属性和以及在这些属性上的操作。类之间可以存在不同的关系，比如泛化（继承）、关联和依赖等，静态视图表示成类图，静态视图在某一时刻的快照称为对象图。

UML的基本概念[8]— UML概念范围

- 动态行为：
  - 状态机视图(State Machine View)，通过对每个类的对象的生命周期进行建模，描述了对象时间上的动态行为。状态机是由状态和迁移组成的图，状态机通常附属于类，描述类实例对接受事件的响应。
  - 活动视图(Activity View)是利用状态机对运算和工作流进行建模的特殊形式。活动图的状态代表了运算执行的状态，而非一般对象的状态，活动图和流程图很相似，不过它支持并发。

UML的基本概念[9]— UML概念范围

- 交互行为：
  - 交互视图(Interaction View)，对象通过交互来实现行为，交互视图通过协作来进行建模，协作具有结构和行为两个方面，结构包含为行为方面而定义的一系列角色和关系，行为方面是绑定于角色的对象间的一系列交换的消息，这些消息在协作中称为交互，消息序列可用两种图来表示：顺序图（重点在消息的时间顺序）和协作图（重点在交换消息的对象间的关系）。

UML的基本概念[10]－ UML概念范围

- 物理实现：
  - 物理视图(Physical View)，许多系统模型独立于最终的实现，在实现方面，必须充分考虑系统的重用性和性能。UML有两种视图来表示系统的实现：实现视图和部署视图，实现视图将可重用的系统片段打包成组件，部署视图描述系统运行时资源的物理分布，这些资源称为结点。

UML的基本概念[11]－ UML概念范围

- 各种图之间的关系
  - 静态视图（类图，对象图），物理视图（实现视图，部署视图）是描述系统的静态结构。
  - 用例图是描述系统的外部视图。
  - 活动图描述系统的外部/内部视图。
  - 交互视图（顺序图，协作图）描述系统的内部视图。
  - 状态图描述单个类的动态行为。

UML的基本概念[12]－ UML概念范围

- 模型组织
  - 模型管理视图(Model Management View)，任何大系统必须划分为较小的单元，以使人们能在某一时刻只接触有限的信息，不影响团队间的并行工作。模型是利用包(Package)和包的依赖来进行管理的。包是UML模型中通用的层次组织结构，包上的依赖总结了包内容的依赖关系。

UML的基本概念[13]－ UML概念范围

- 扩展机制
  - 扩展机制(Extension Mechanisms)，UML能满足绝大部分系统建模的需要，但任何语言都不是万能的，它必须考虑一定的扩展机制，UML的扩展机制包括约束、标签值和原型。这些扩展机制可以用来为特定领域剪裁UML的配置，这样带来一些好处：根据自身需要来使用建模语言。

基本图素

- 一个模型必须首先定义各种事物的内部特征和相互之间的关系，下面介绍一些基本的模型元素：

分类：共12个

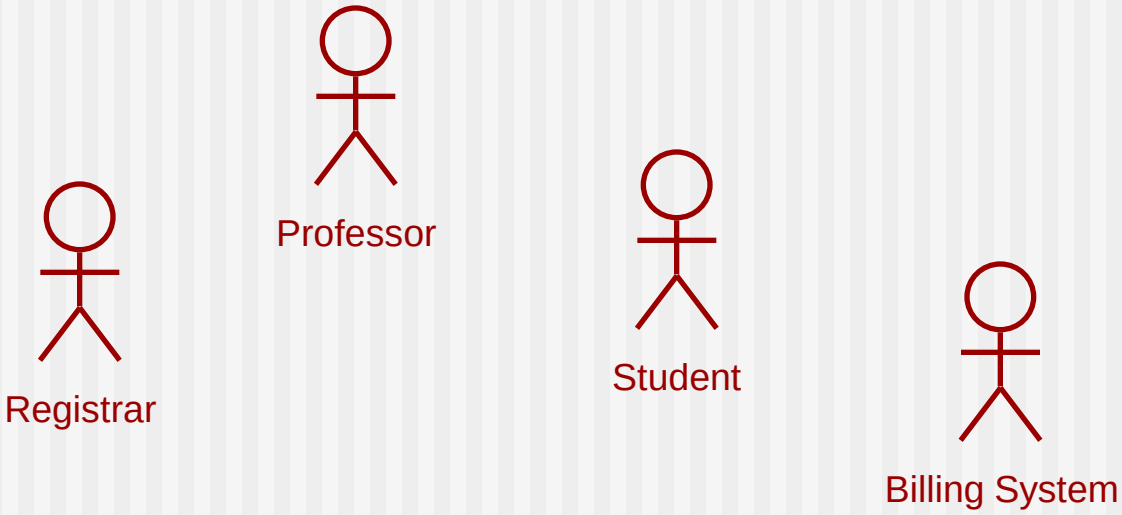
- 类（Class）
- 接口（Interface）
- 包（Package）
- 角色（Actor）
- 用例（Use Cases）
- 组件（Component）
- 结点（Node）
- 状态（state）……

关系：

- 关联（Association）
- 泛化（Generalization）
- 依赖（Dependency）
- 实现（Realization）
- 约束（Constraint)

基本图素[1]—角色

- 角色是与系统、子系统或类交互的外部人员，进程或事务。在运行时，具体人员会充当系统的多个执行者，不同用户可能会成为一个执行者。



- 根据上下文的不同而具有不同的含义

基本图素[2]—用例

- 用例是系统提供的外部可感知的功能单元，用例的目的是定义清晰的系统行为，但不解释系统的内部结构。
- 用例可以与角色关联，也可以参与其他的多种关系，比如扩展、泛化和包含等。
- 用户的动态部分用交互视图来描述，比如顺序图、协作图。
- 用例用椭圆来表示，用例名标在椭圆下方，用实线与同自身通信的用户相连。

求一元二次方程根

基本图素[3]—系统

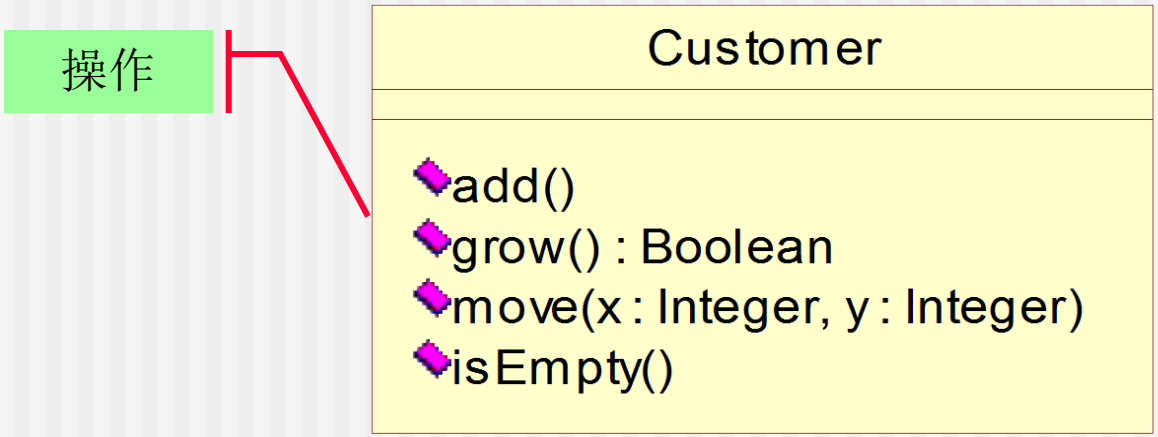
- 系统也叫做系统边界，用于界定系统功能范围。
- 它用一个矩形框表示。描述该系统功能的用例都置于其中，而描述的与系统交互的角色都置于其外。系统（边界）常常省略不画。

基本图素[4]—类

- 类是具有相同属性、操作和关系的对象集合的总称。通常在UML中类被画成矩形，包括三个部分：名称、属性和操作。下面是可选的。
  - 名称：每个类都必须有一个名字，用来区分其它的类。类名是一个字符串，称为简单名字。路径名字是在类名前加包含类的包名为前缀。例如Wall、java::awt::Wall都是合法的类名。
  - 属性：类可以有任意多个属性，也可以没有属性。在类图中属性只要写上名字就可以了，也可以在属性名后跟上类型甚至缺省取值 。
  - 操作：操作是类的任意一个实例对象都可以调用的，并可能影响该对象行为的实现。

基本图素[4]—类

Customer
name
address
phone
类名
属性
Customer
name : String
address : String
phone : Long = 6688777


操作
Customer
add()
grow() : Boolean
move(x : Integer, y : Integer)
isEmpty()

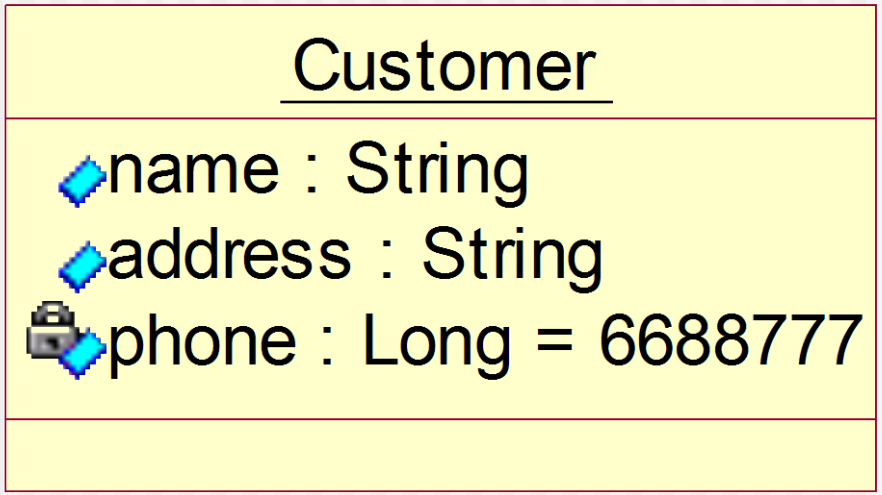
基本图素[5]—对象

1、对象是类的实例。

2、对象图素中对象名的下面加下划线，对象图素所使用的符号与类图素基本相同。二者的区别在于对象图素展示的是类的实例，而不是类本身。

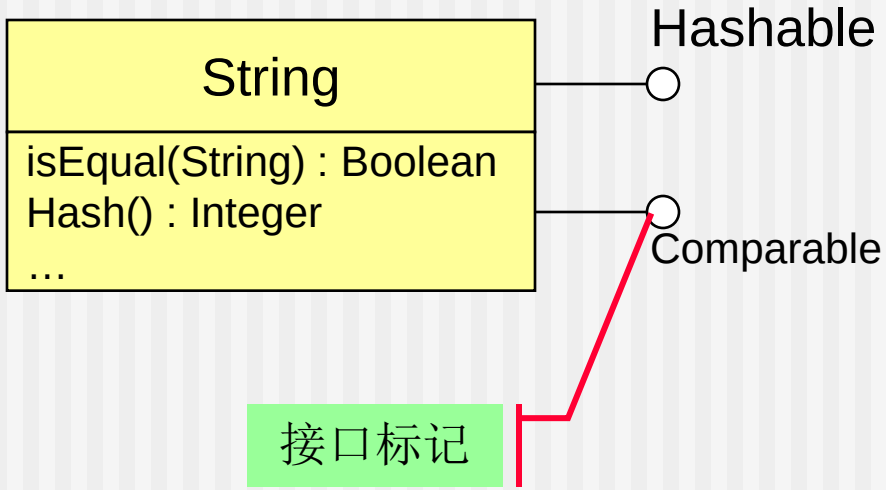
3、对象图素除第一格外，其他格是可选的。

对象常常出现在活动图、序列图和协作图中。


Customer
name : String
address : String
phone : Long = 6688777

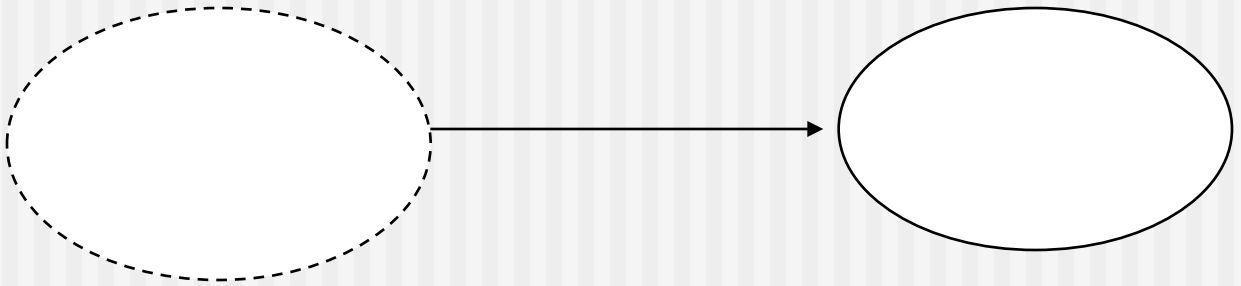
基本图素[6]—接口

- 接口是未给出实现的对象行为的描述，接口包含操作，但没有属性，一个或多个类可以实现接口，每个类实现接口的操作。
- 思考：为什么会有接口？



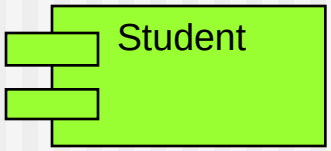
基本图素[7]—协作

- 协作实质代表一组模型元素协同完成对应用例的功能。
- 协作与对应的用例存在着一种所谓的实现关系。
- 协作与对应的用例分离带来的好处是为描述一个用例可以用不同的协作实现提供了可能。


求一元二次方程根
求一元二次方程根

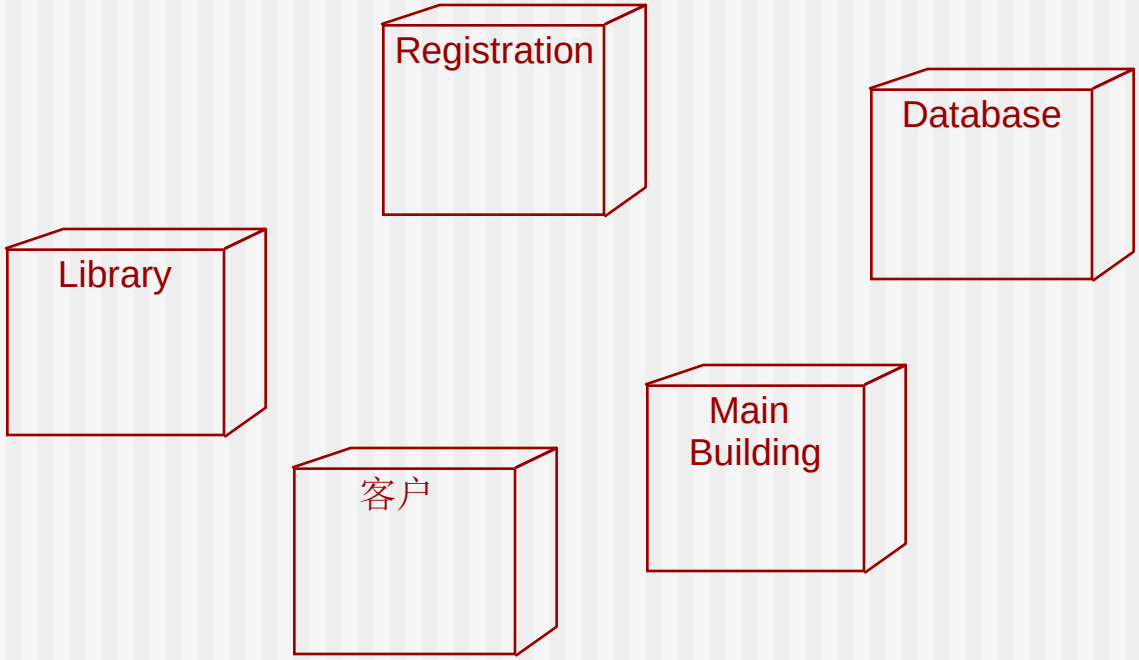
基本图素[8]—组件

- 组件是可重用的系统片段，具有良好定义接口的物理实现单元。每个组件包含了系统设计中某些类的实现。
- 组件设计的原则：良好的组件不直接依赖于其它组件，而是依赖于其它组件所支持的接口。这样的好处是系统中的组件可以被支持相同接口的组件所取代。
- 一个组件可能是源代码、可执行程序或动态库。



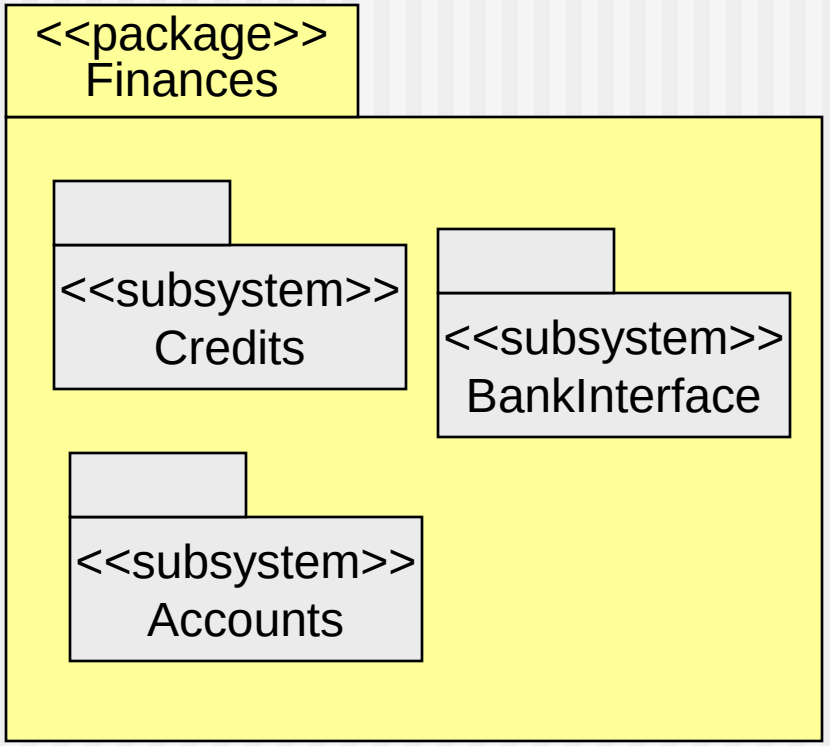
基本图素[9]—结点

- 结点代表系统运行时的物理对象，结点通常拥有运算能力，它可以容纳对象和组件实例。



基本图素[10]—包

- 任何大系统都必须划分为较小的单元，以便人们在某一时刻可以和有限的信息工作，使团队的工作不相互影响。
- 包可以包含各种模型元素和其它的包，包之间还可能存在一定的依赖。



基本图素[10]—包

- 具有独立的说明和实现部分的包，它代表了与系统其它部分具有清晰接口的清晰单元，它通常代表了系统在功能或实现范围上的划分。

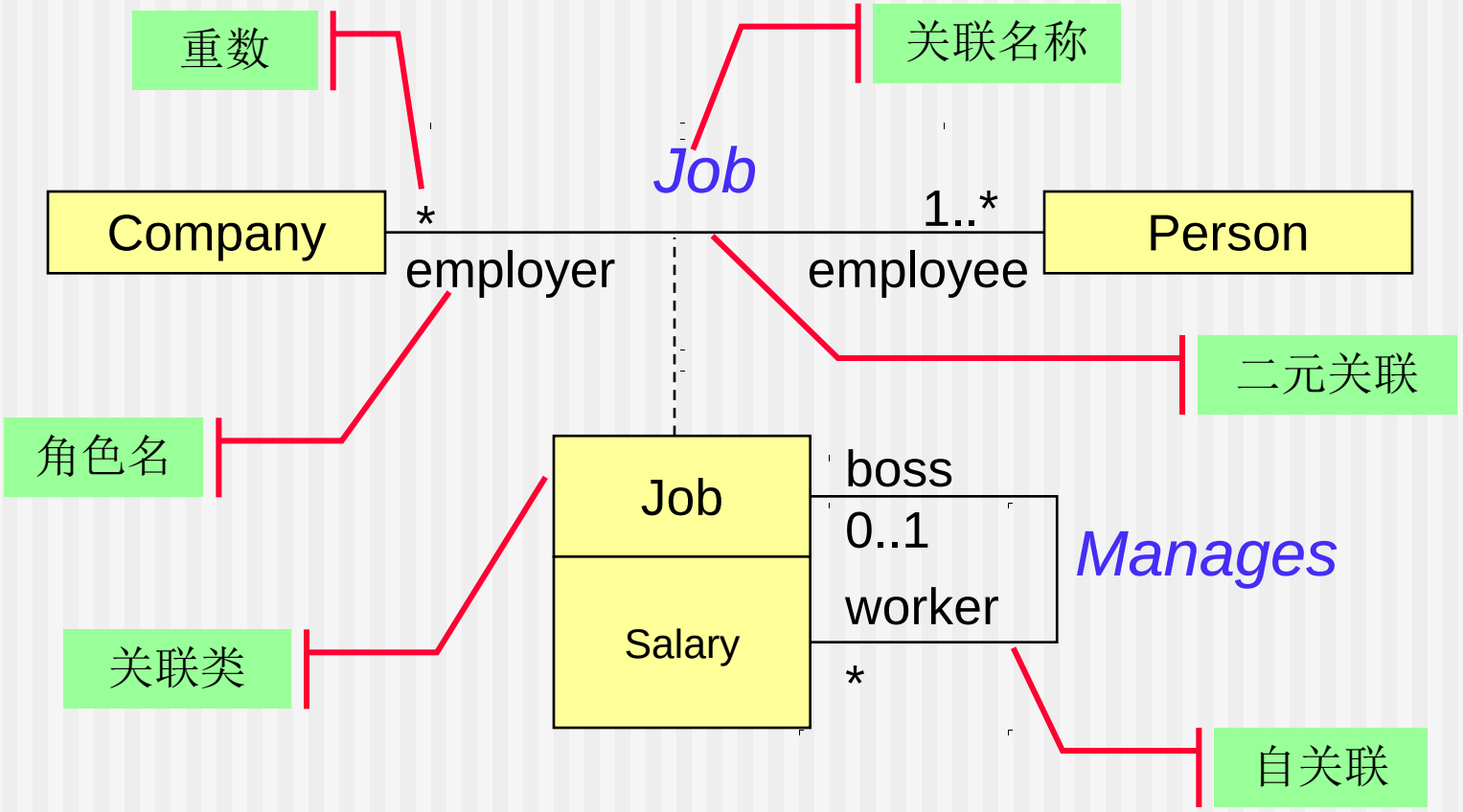
基本图素—消息

- 消息代表软件系统内两个对象中一个对象向另一个对象发出的执行某种操作的请求。
- 序列图中的消息有简单、异步、同步之分
- 1、简单消息是一个对象到另一个对象的转移。
- 2、如一个对象发送了同步消息，必须等待对方的回答后才继续自己的操作。
- 3、相反，如一个对象发送了异步消息，不必等待对方的回答就继续自己的操作。

基本图素[11]—关系—关联

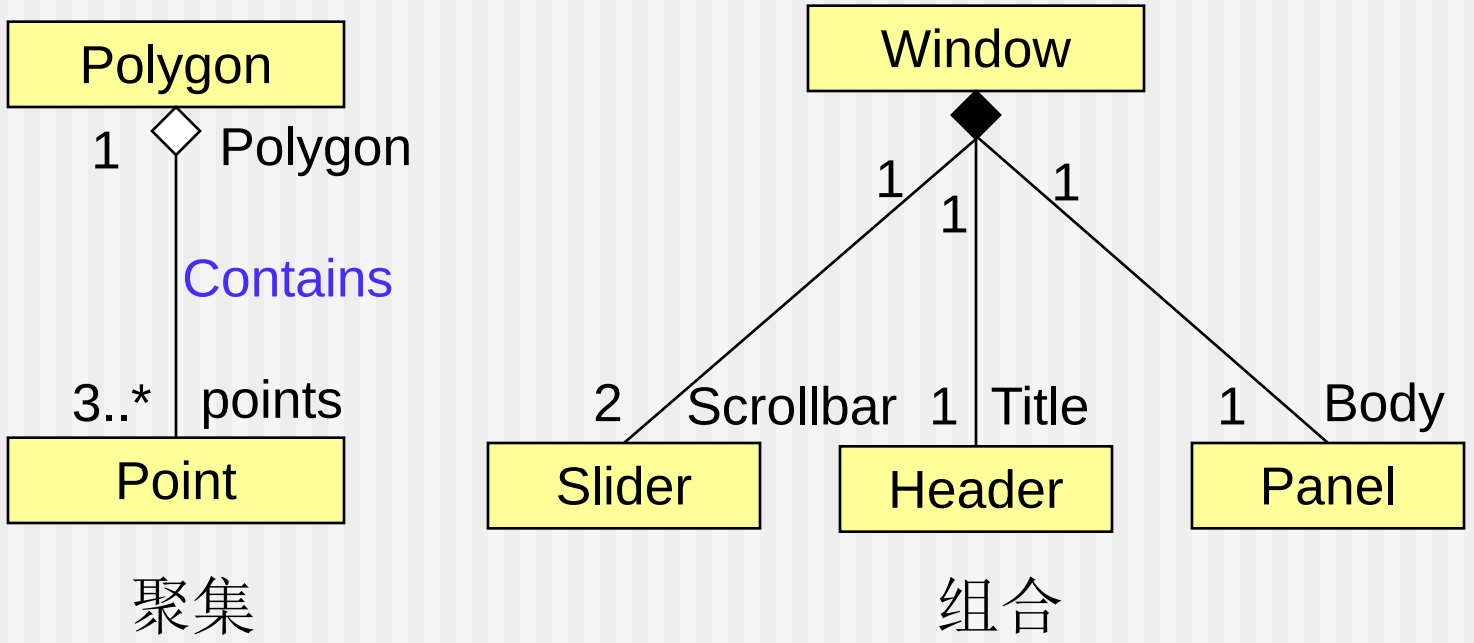
- 关联描述了系统中对象和其它实例之间的离散的连接，关联是有序的，它允许重复，关联的实例是链。
- 关联至对象的连接点称为关联端点，很多信息被附在关联端点上，它拥有角色名、重数（多少个类的实例可以关联于另一个类的实例），可见性等。
- 关联有自己的名称，可以拥有自己的属性，这时关联本身也是类，称为关联类。

基本图素[11]—关系—关联


重数
关联名称
Job
Company
*
employer
1..*
employee
Person
二元关联
角色名
Job
Salary
boss
0..1
worker
Manages
*
关联类
自关联

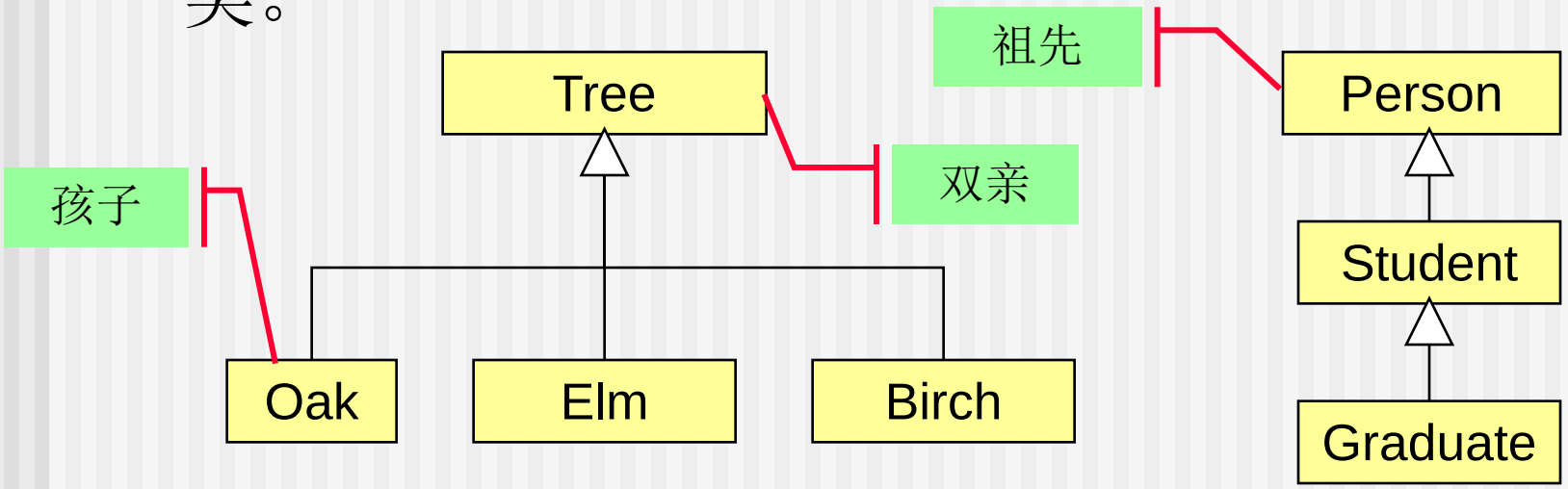
基本图素[11]—关系—关联

- 聚集（Aggregation）用来表达整体—部分关系的关联。组合（Composition）是一种聚集，是关联更强的形式。



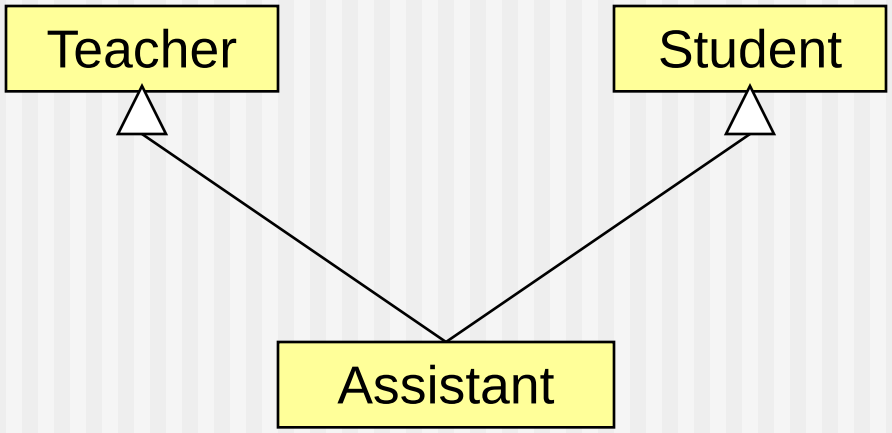
基本图素[11]—关系—泛化

- 泛化是一般化和具体化之间的一种关系。
- 继承就是一种泛化关系，更一般化的描述称为双亲，双亲的双亲称为祖先，更具体化的描述称为孩子，在类的范畴，双亲对应超类，孩子对应子类。



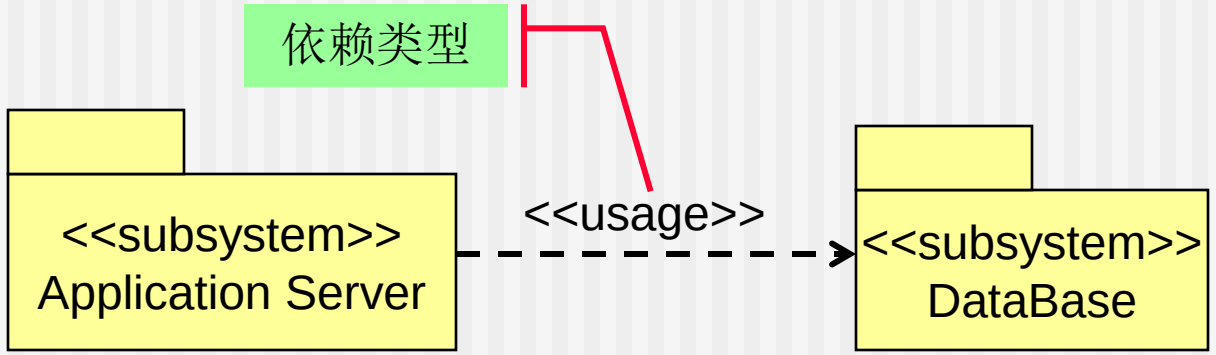
基本图素[11]—关系—泛化

- 多重继承：一个孩子可以从多个双亲继承属性和方法。多重继承可能存在冲突，因为被继承的双亲可能存在相同的类声明，这时，最好显式解决冲突问题。



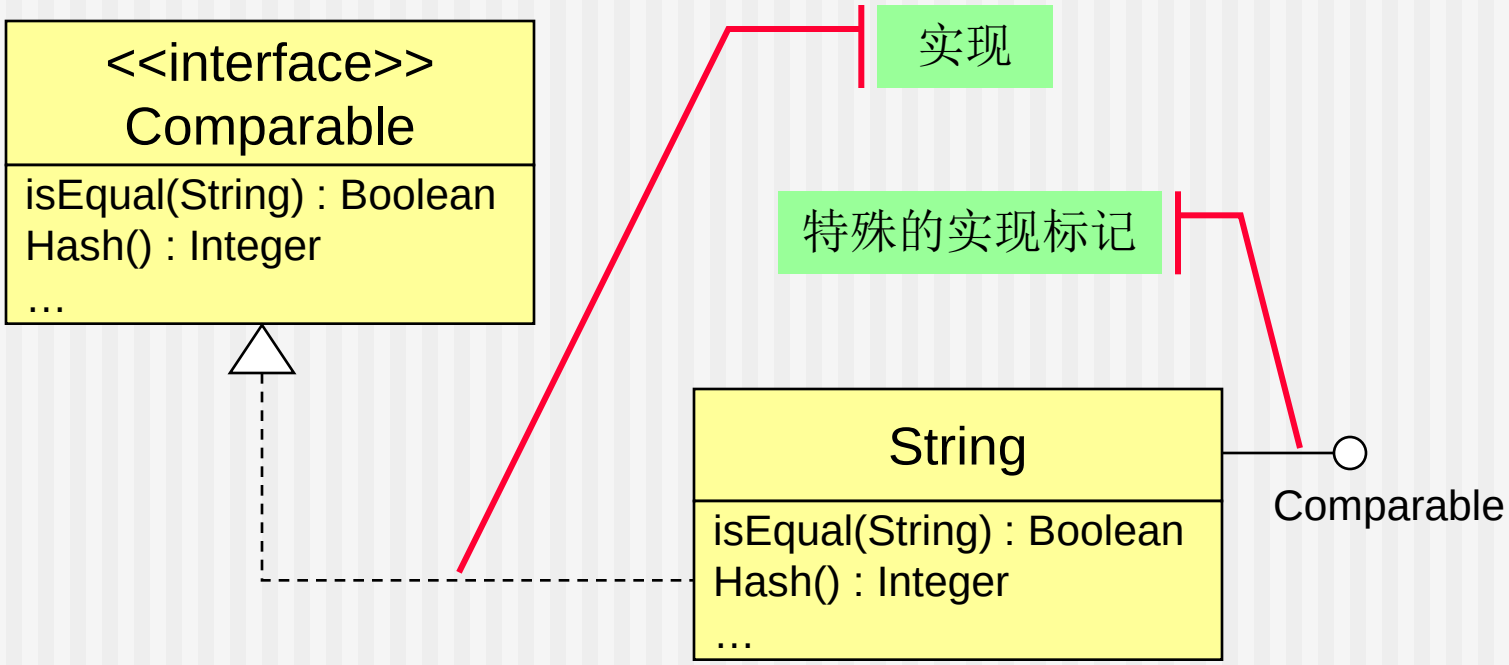
基本图素[11]—关系—依赖

- 依赖指明两个或两个以上模型元素之间的关系。
- 依赖有很多种类，比如：实现（realize）、使用（usage）、实例化（instantiate）、调用（call），派生（derive）、访问（access）、引入（import）、友元（friend）等等。



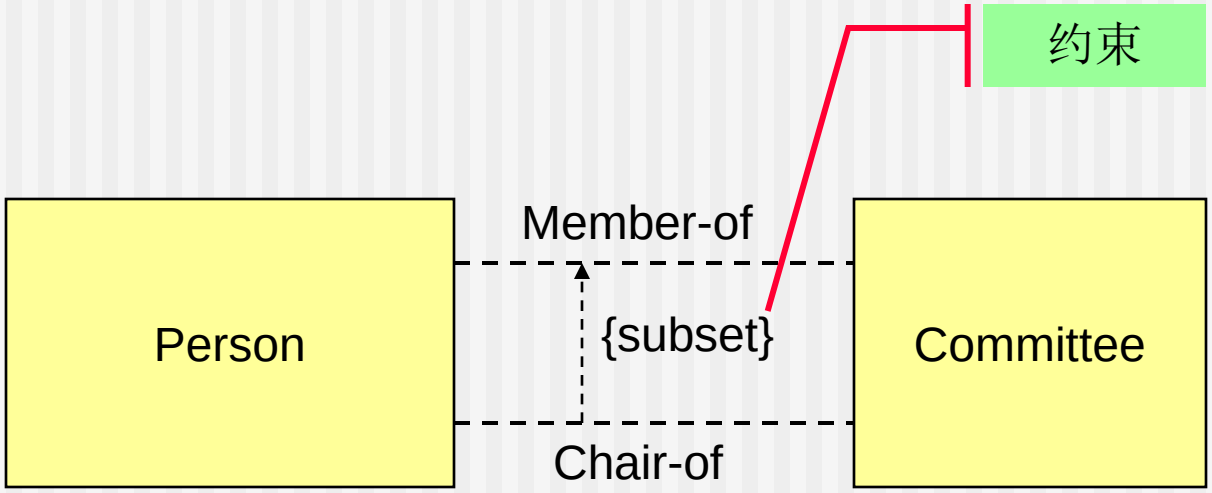
基本图素[11]—关系—实现

- 实现是依赖的一种，但由于它具有特殊意义，所以将它独立讲述。实现是连接说明和实现之间的关系。



基本图素[11]—关系—约束

- 约束用来表示各种限制，如关联路径上的限制，和属性特征检测（存在、所有）。

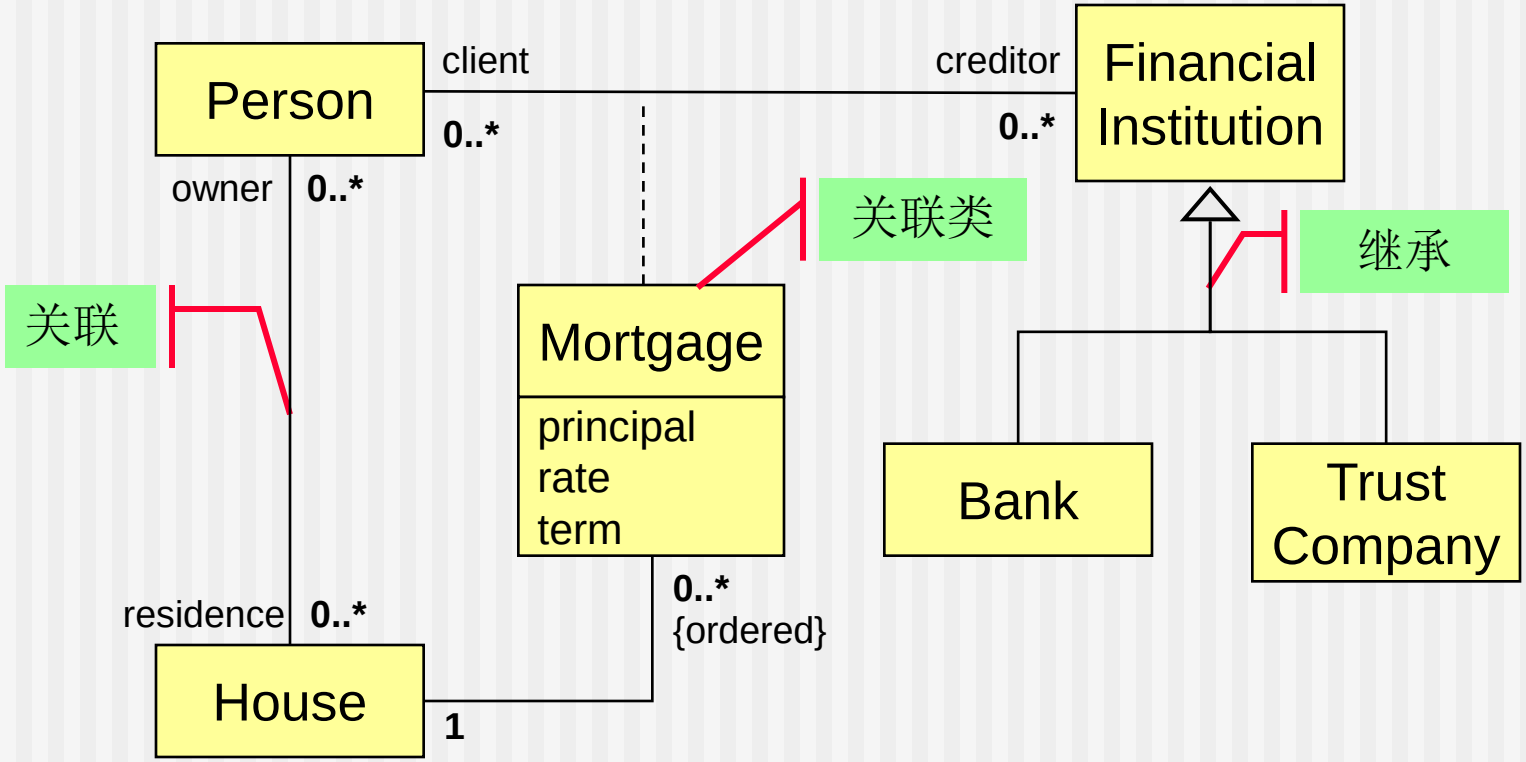


模型图

- 静态视图：
- 类图
- 对象图
- 部署图
- 组件图

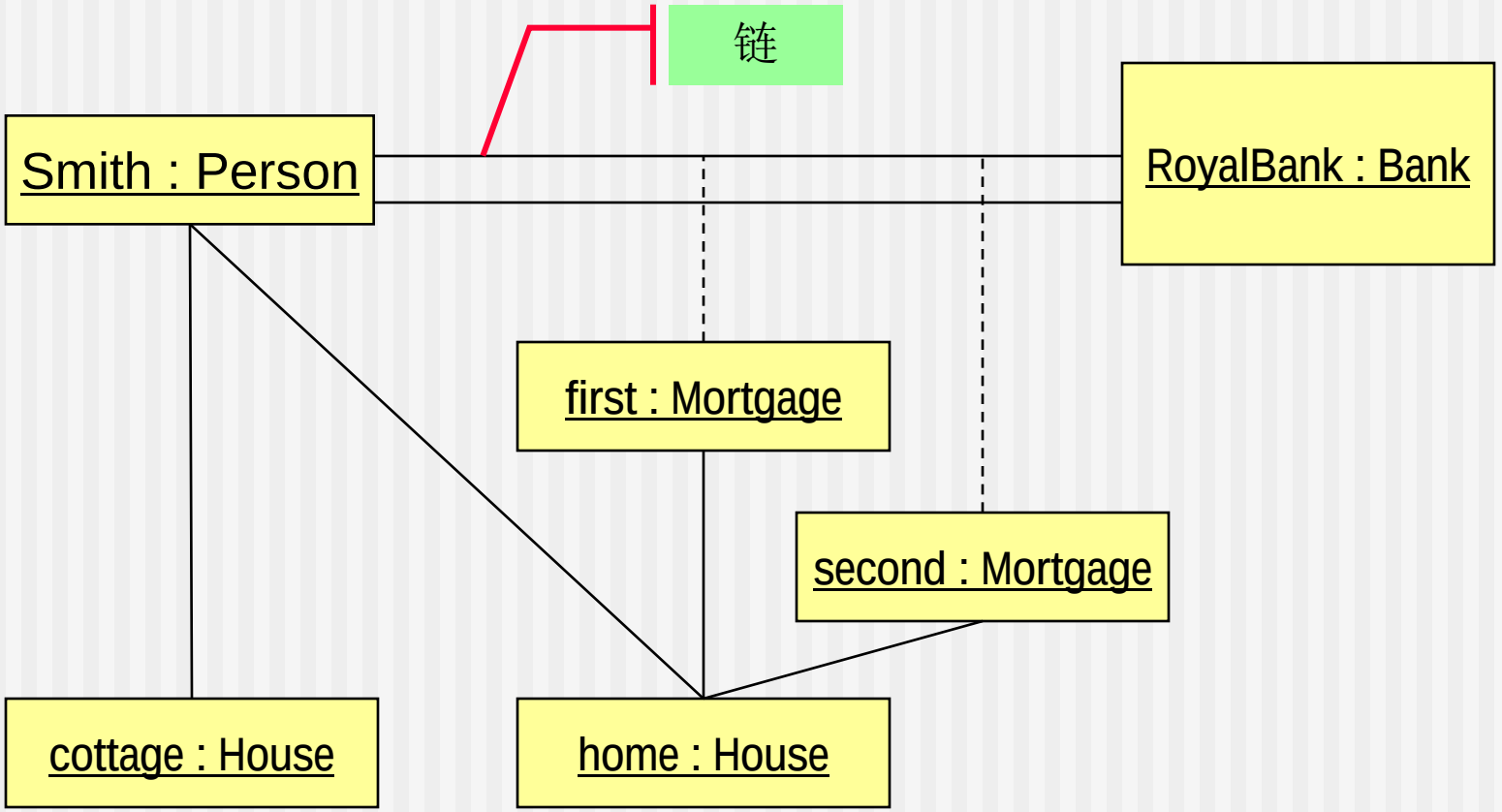
静态建模[1]—类图

- 静态视图是UML的基础，静态视图表示为类图，主要是描述类和类之间的关系。



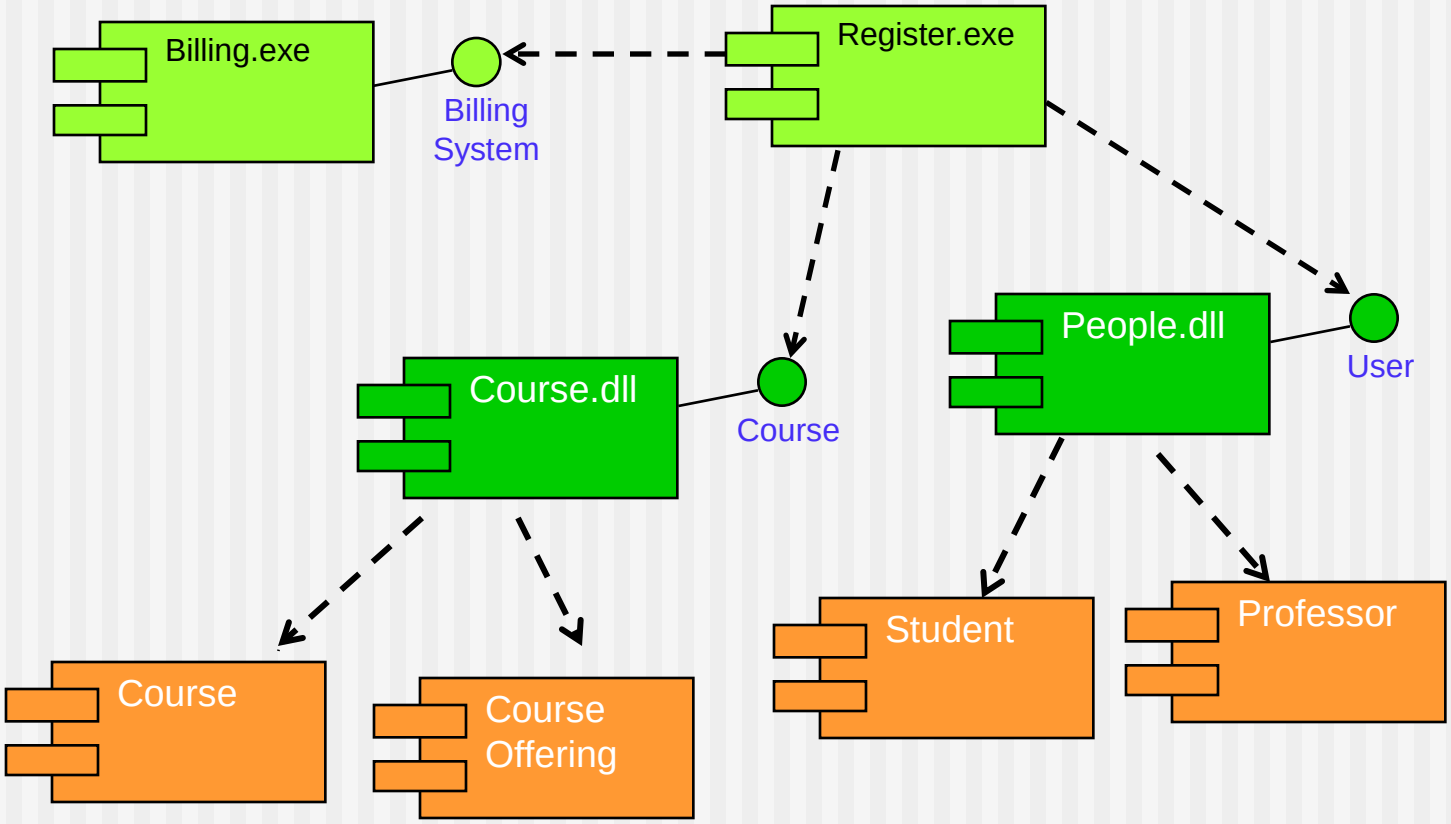
静态建模[2]—对象图

- 对象图是系统在某一时刻的快照。



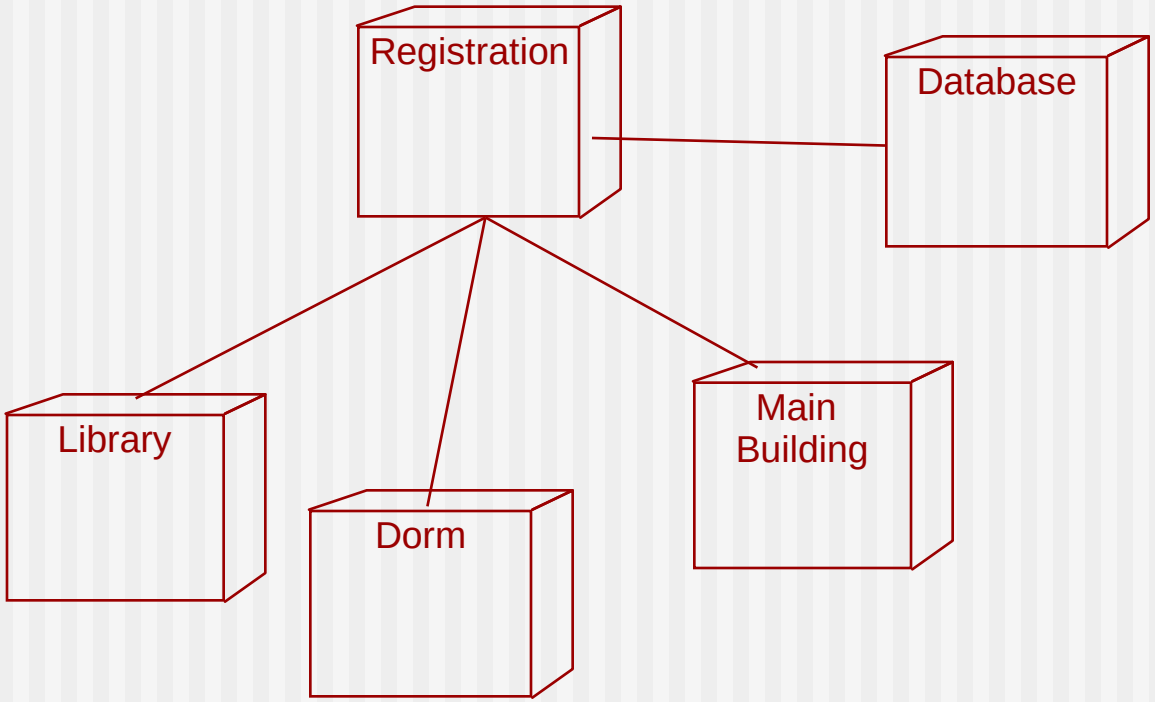
静态建模[3]—组件图

- 组件图描述可重用的系统组件以及组件之间的依赖。



静态建模[3]—部署图

- 部署图描述系统资源在运行时的物理分布，系统资源成为结点。

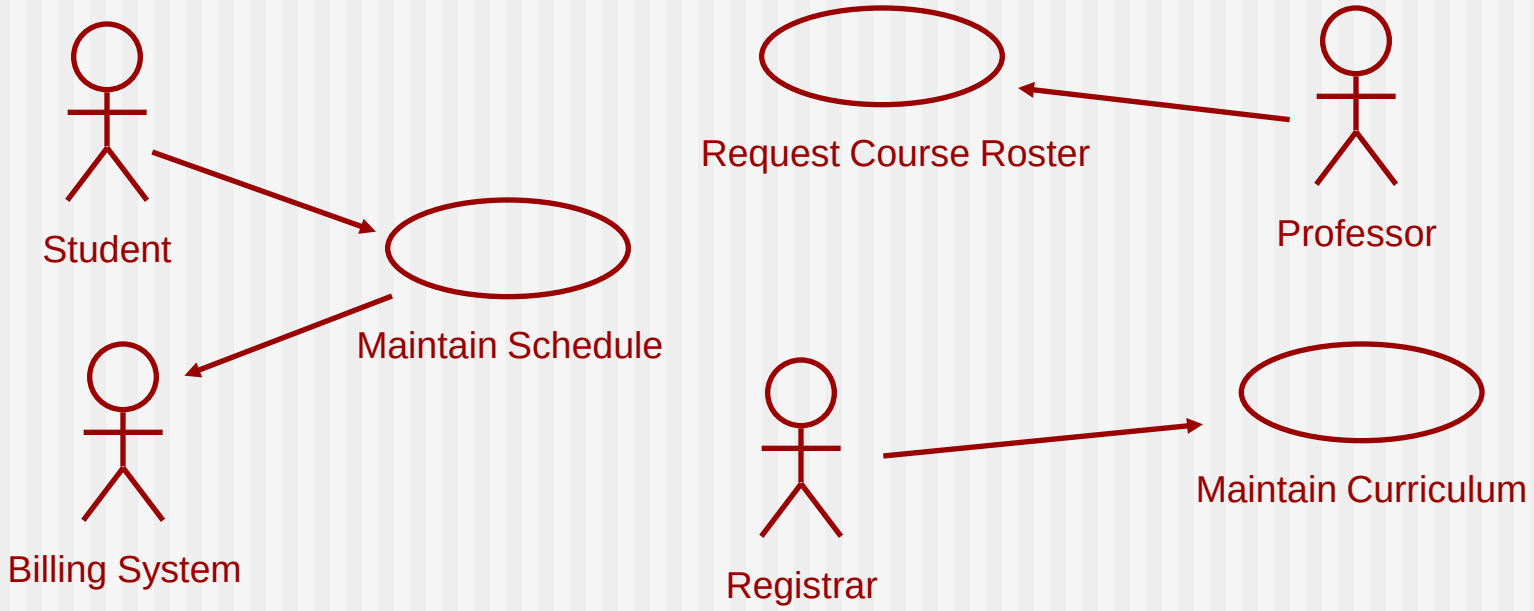


动态建模[1]

- 用例图：从外界对系统的感知
- 状态图：对象自身的活动
- 活动图
- 序列图：} 对象之间的关系
- 协作图

动态建模[1]—用例图

- 用例图描述执行者在各个用例中的参与情况。

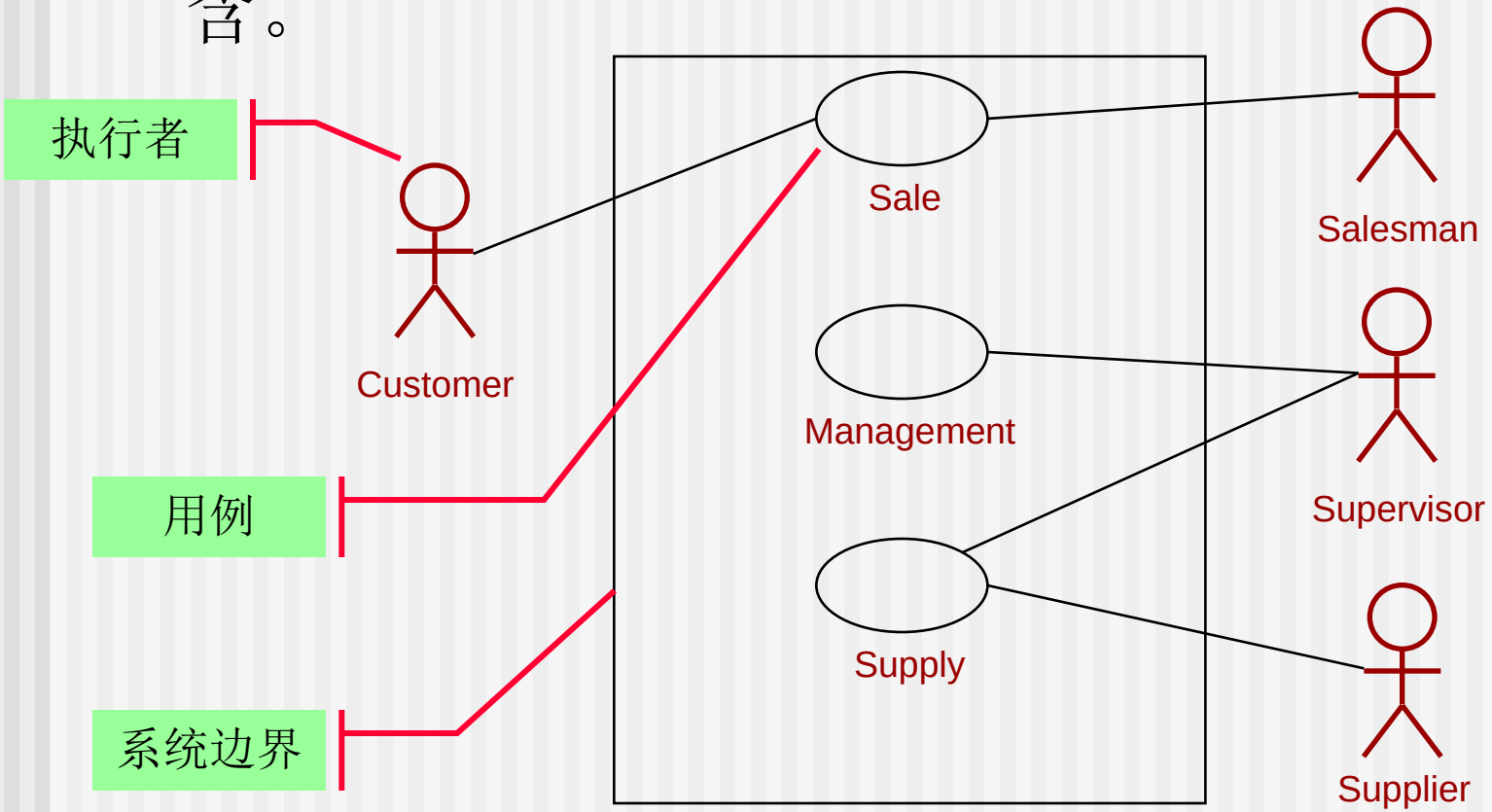


动态建模[1]—用例图

- 用例图描述各个执行者在各个用例中的参与情况，描述系统为用户所感知的外部视图。
- 用例图的功能：
  - 捕获系统用户需求
  - 描述系统边界
  - 指明系统外部行为
  - 指导系统开发者的功能开发
  - 系统建模的起点，指导所有的类图和交互图的设计
  - 产生测试用例，用户文档
  - 估计项目大小和进度。

动态建模[1]—用例图

- 用例可以参与多种关系：关联、扩展、泛化和包含。



动态建模[2]—状态机图

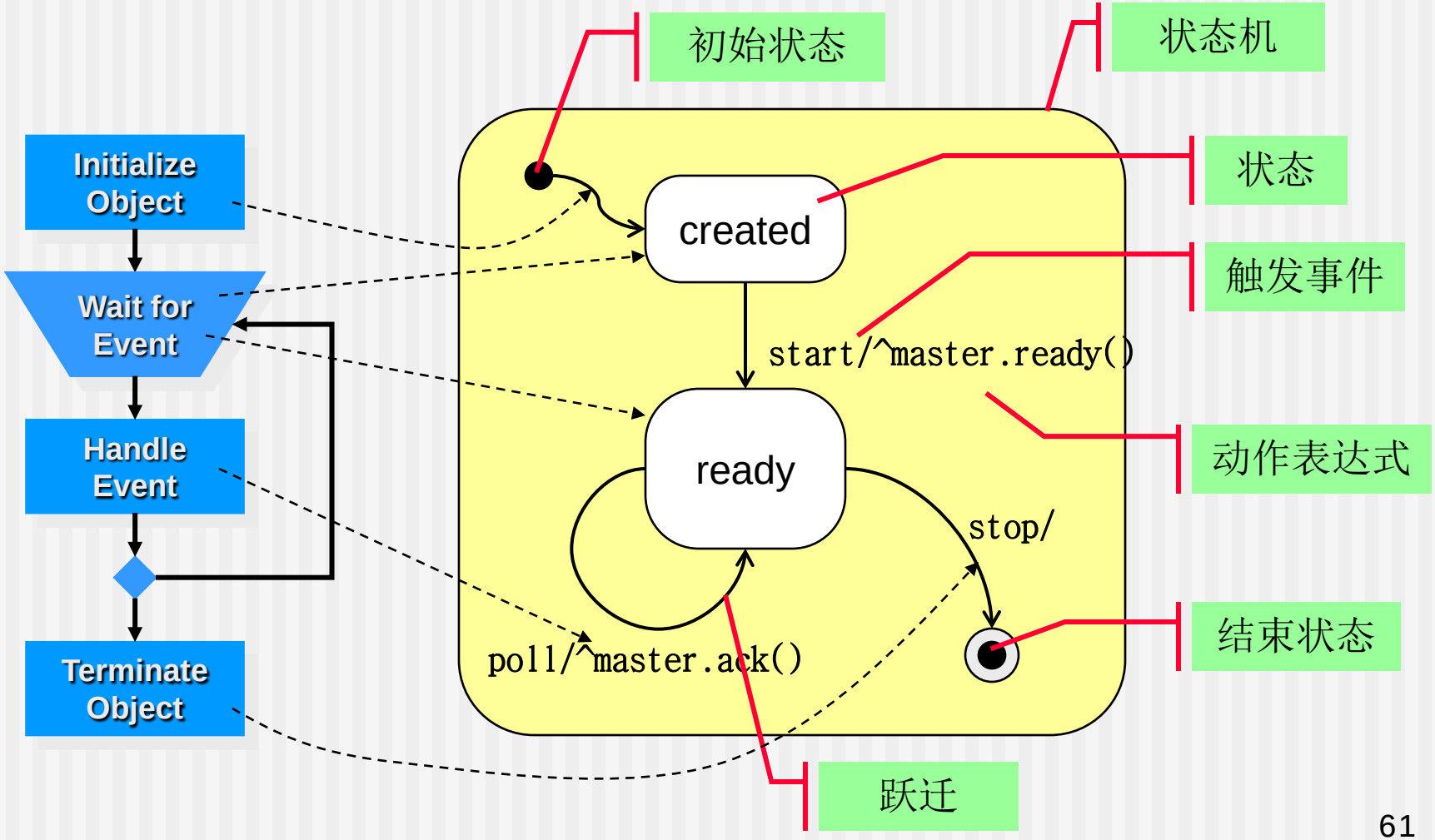
- 状态机图是对单个类的对象的生命周期进行建模，描述了对象时间上的动态行为，每个对象被认为是事件驱动的孤立实体。
- 事件表达对象间的调用、显式信号、值的改变或时间的推移。
  - 调用事件、变更事件、信号事件、时间事件
- 状态描述对象生命周期的一段时间，可以是等待其它事件时所处的时间，或是执行某一活动时所处的时间，状态分为简单状态和复合状态。

动态建模[2]—状态机图

- 状态机图是由状态和跃迁组成的图，通常状态机附属于类，描述类实例对接受事件的响应。
- 跃迁定义对象对某一事件发生的反应，通常，迁移具有触发事件、跃迁条件、动作和目标状态。
- 跃迁的种类有外部跃迁和内部跃迁。外部跃迁是最普通的跃迁，会发生状态改变；内部跃迁不发生状态改变。
- 跃迁有两个隐式动作：进入动作和退出动作。无论何时进入和退出时都要执行，这方便进入时进行初始化工作，退出时进行资源的释放工作。

- 它强调对象行为的事件顺序，强调对象对外部事件的响应及相应的状态变迁。
- 通过状态图可以了解一个对象所能到达的所有状态以及对象涉及的事件（消息、超时、错误、条件满足）对对象状态的影响。

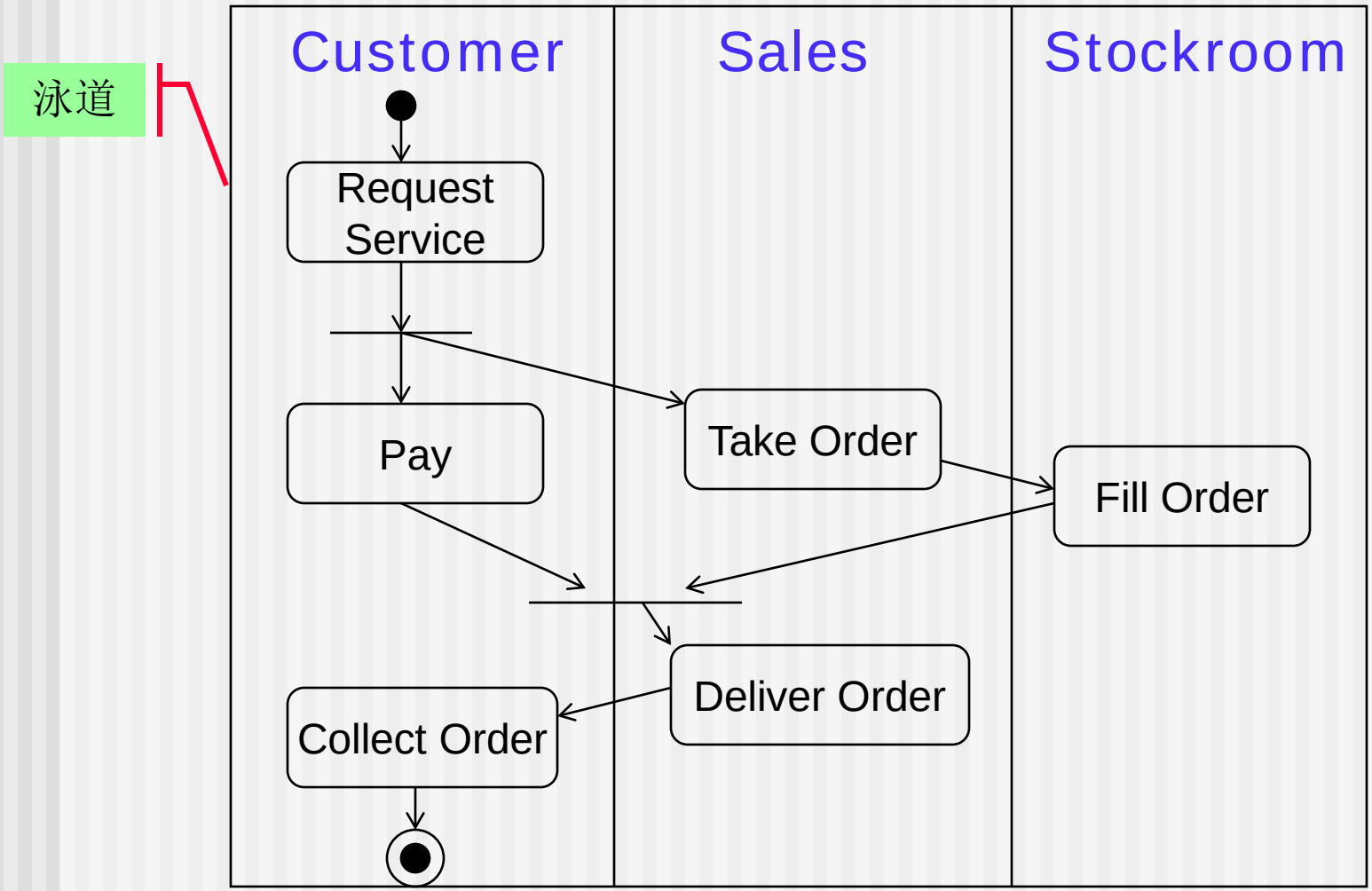
动态建模[2]—状态机图


初始状态
状态机
Initialize Object
Wait for Event
Handle Event
Terminate Object
created
状态
触发事件
start/^master.ready()
ready
动作表达式
stop/
poll/^master.ack()
结束状态
跃迁

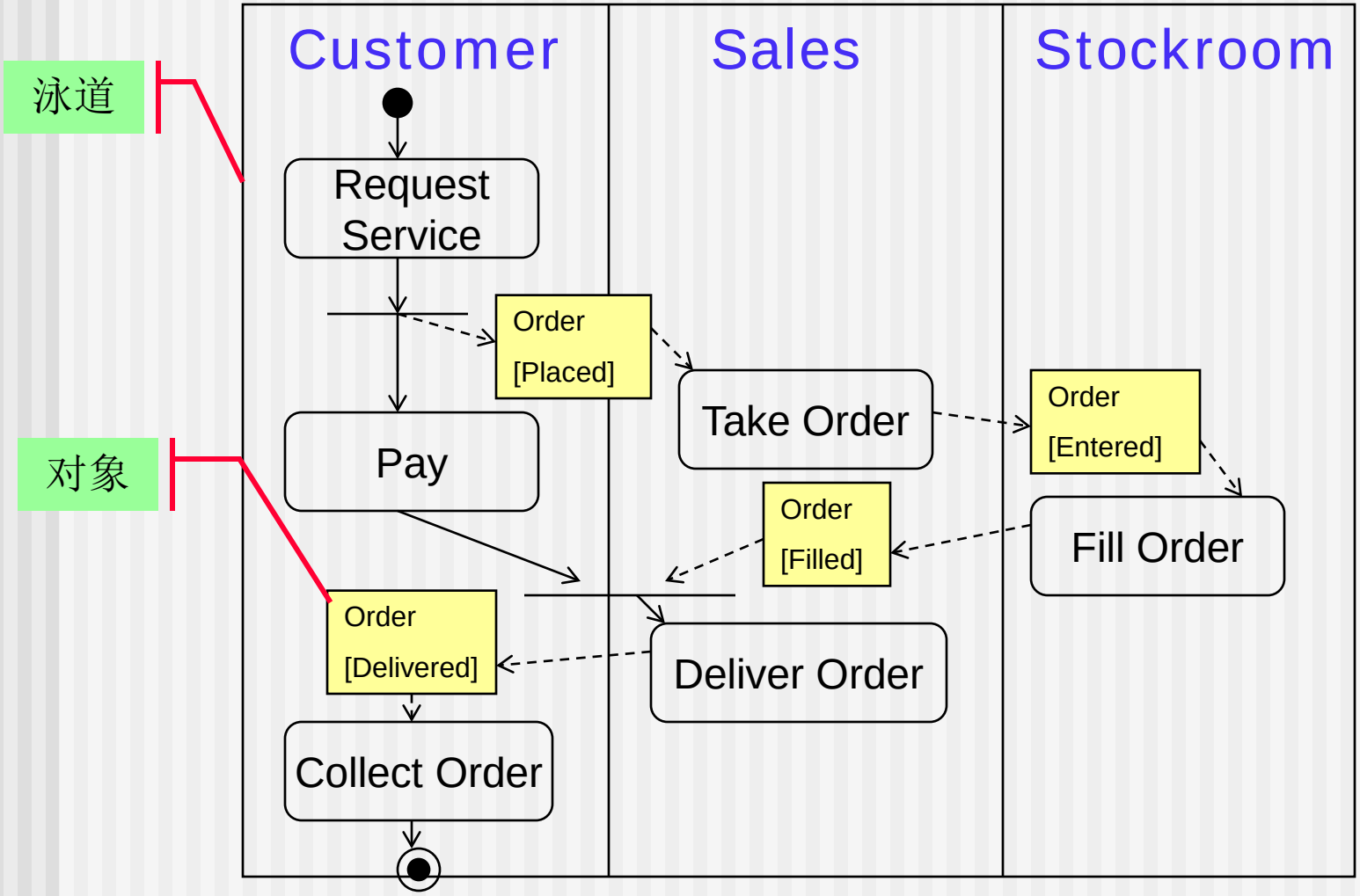
动态建模[3]—活动图

- 活动图是用状态机对工作流进行建模的特殊形式，它和流程图很类似，不过它支持并发控制。
- 活动图一般不描述所有的运算细节，它显示活动的流，但不显示执行活动的对象。
- 活动图处于系统的外部和内部视图之间，所以它可以作为设计的起点，为了完成设计，每个活动必须扩展成一个和多个操作，每个操作被指派给特定的对象来实现。
- 带有生命线的活动图和无生命线的活动图。

动态建模[3]—活动图


泳道
Customer
Sales
Stockroom
Request Service
Pay
Take Order
Fill Order
Deliver Order
Collect Order

动态建模[3]—带有对象流的活动图



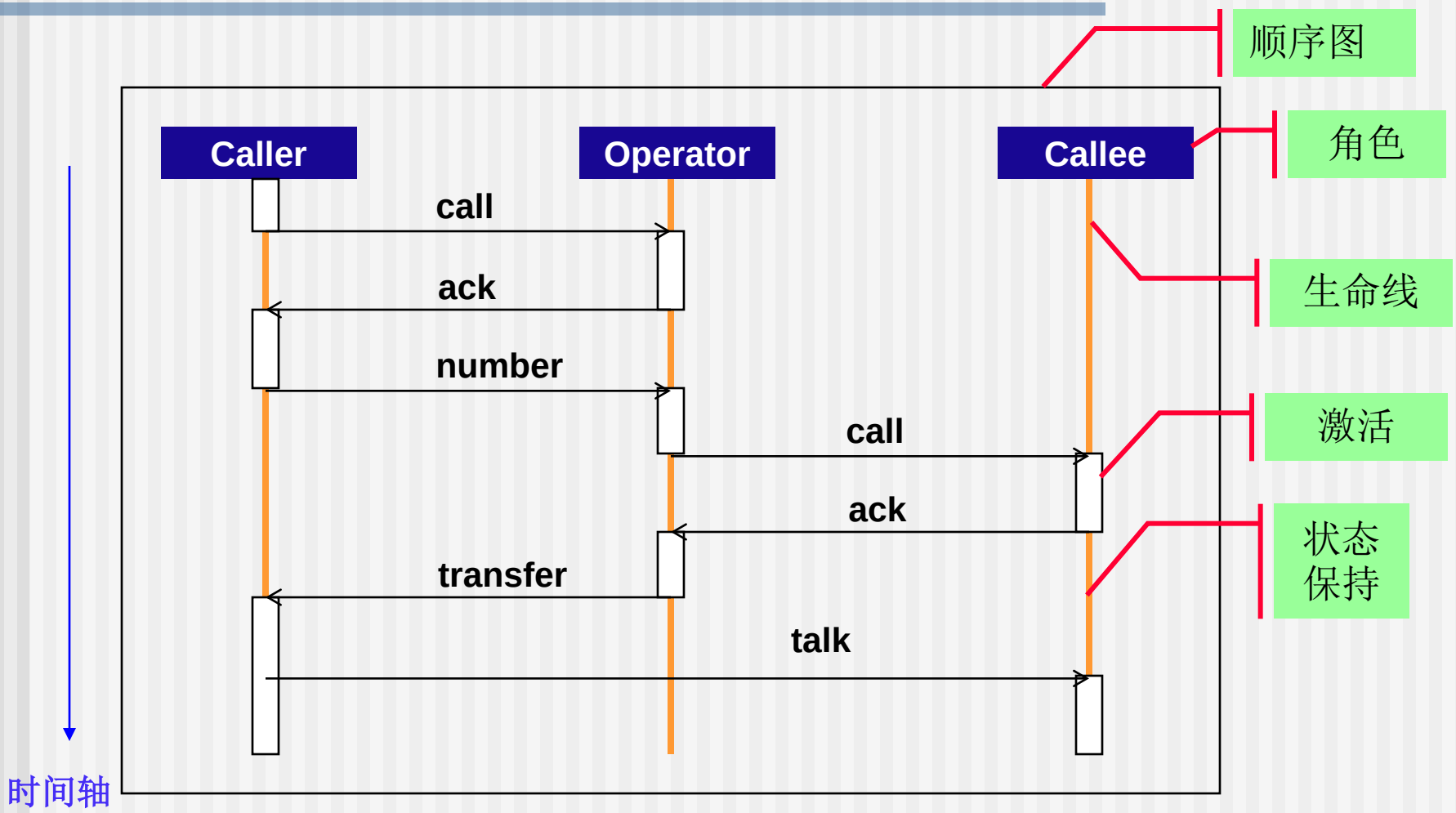
动态建模[4]—序列图

- 对象行为是通过交互来实现的，交互是对象间为完成某一目的而进行的一系列消息交换。
- 消息是对象间的单向通信，从发送者到接受者的携带信息的控制流。消息可能带有值参。
- 消息序列可用两种图表示：序列图（重点在消息的时间顺序）和协作图（重点在交换消息的对象间的关系）。对协作图来说，时间顺序可以从顺序号获得。

动态建模[4]—序列图

- 序列图用二维表来表示交互，纵向是时间轴，横向是参与的角色以及它们交换的消息。
- 角色的生命周期表现为生命线，一条垂直的线，在激活的时间段里是双线，在状态保持的时间里是虚线。
- 消息表示为从一条生命线出发到另一条生命线的有向线，从上而下，表示消息的时间顺序。

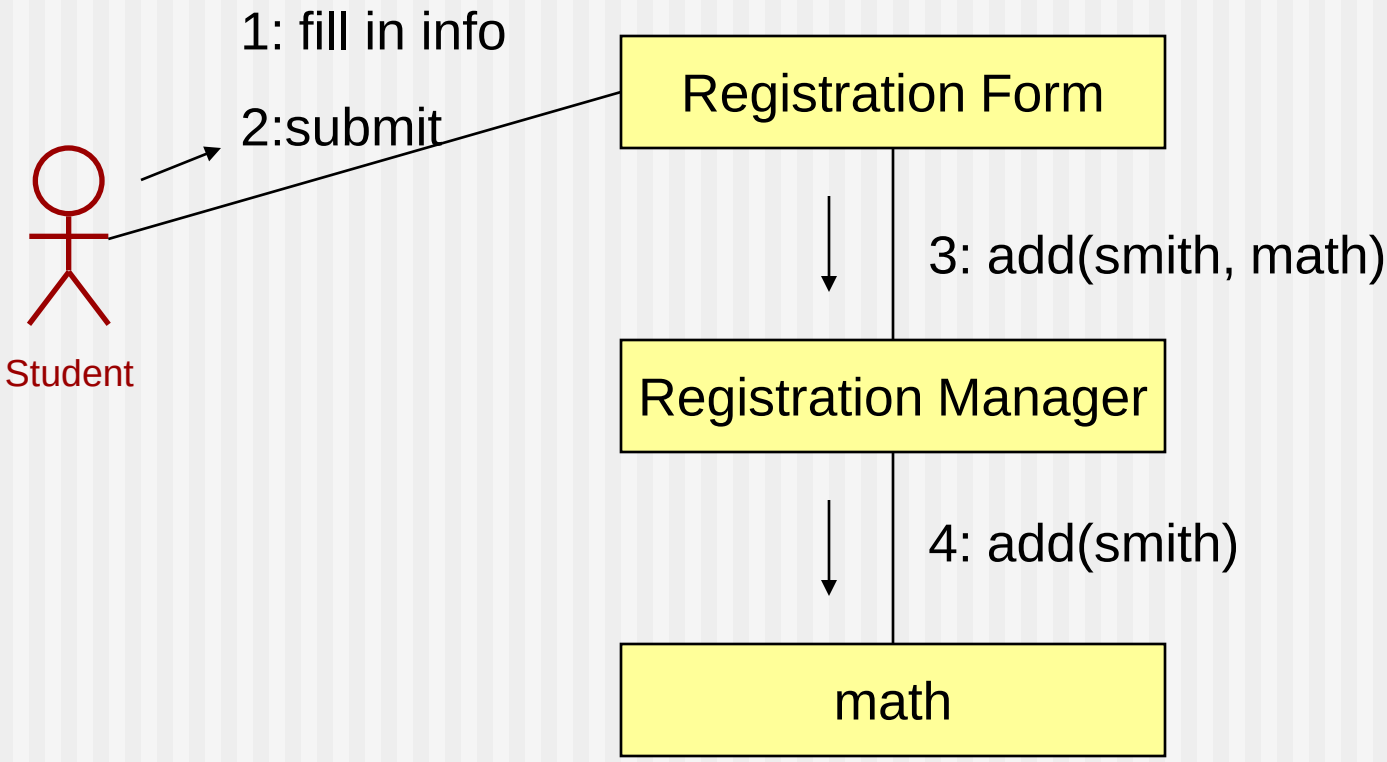
动态建模[12]—序列图


顺序图
角色
生命线
激活
状态
保持
Caller
Operator
Callee
call
ack
number
call
ack
transfer
talk
时间轴

动态建模[5]—协作图

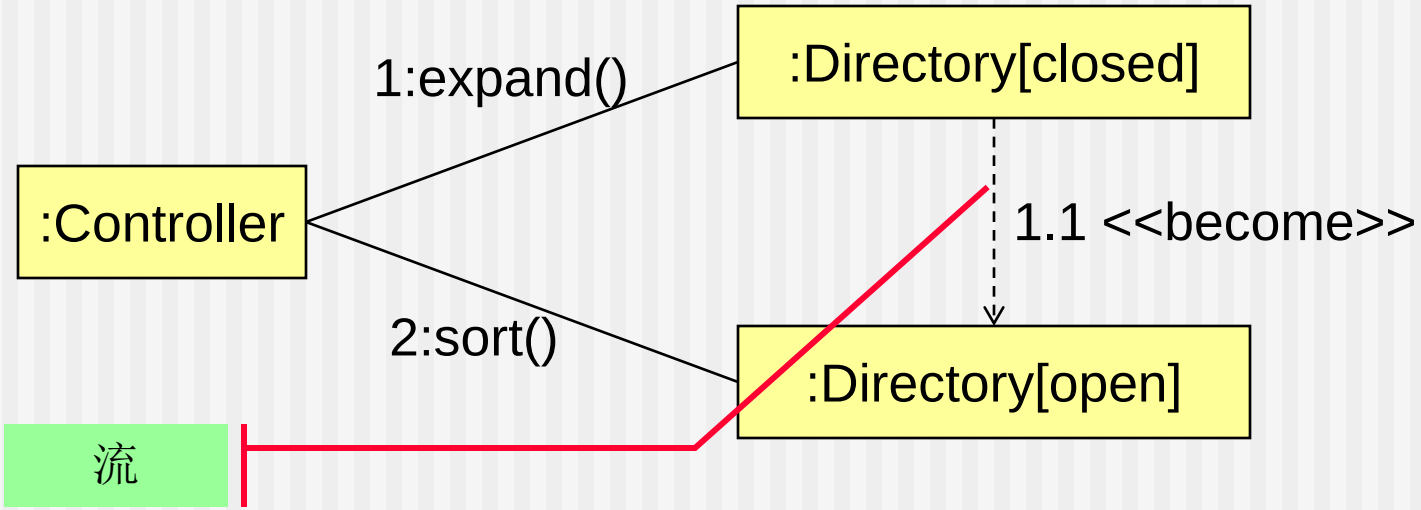
- 协作图包含分类角色和关联角色，当它实例化时，对象被绑定到分类角色，链被绑定到关联角色.关联角色还可能被各种暂时性的链来充当，如过程参数和局部过程变量，链可以指定暂时性的原型：<<parameter>>、<<local>>或自身调用<<self>>。
- 协作图对实现协作的对象和链进行建模，而忽略其他对象。

动态建模[5]—协作图


1: fill in info
2:submit
Registration Form
Student
3: add(smith, math)
Registration Manager
4: add(smith)
math

动态建模[5]—协作图

- 通常在一个协作图中每个对象分配一个符号，然而有时不同状态的对象需要显式指出，流将同一个对象的不同状态版本关系在一起，使用<<become>>原型。流的<<copy>>原型不太常用。

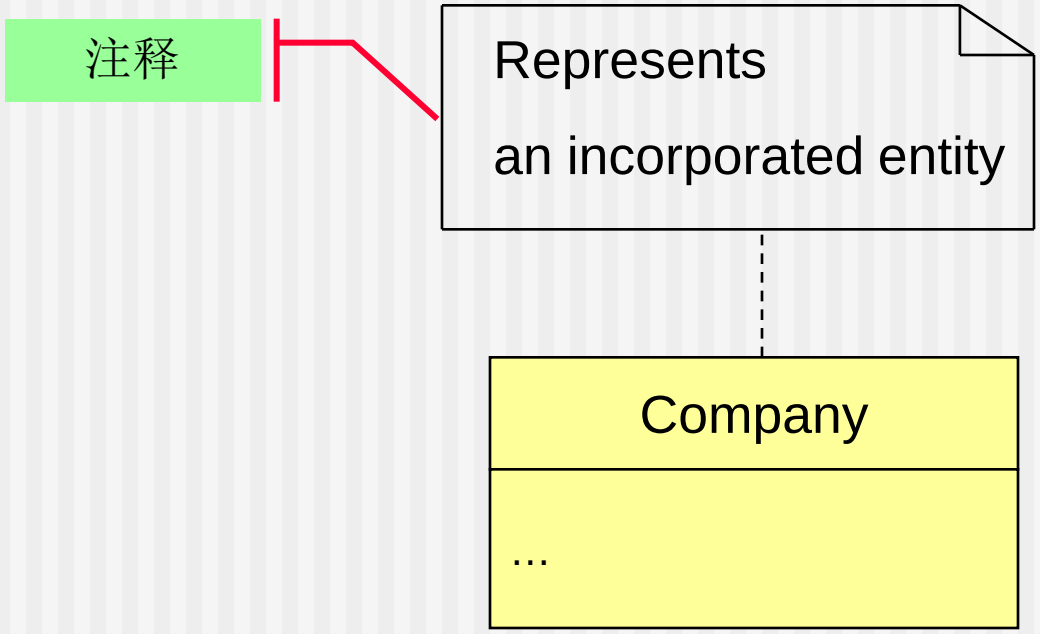


公用机制

- 注释
- 规范说明
- 修饰
- 扩充机制
- 公共划分

公用机制[1]—注释

- 注释用于解释设计的思路，便于理解。
- 一个好的模型应该有详尽的注释。



规范说明

- 软件模型必须是完整的，以便于软件系统的建造。也就是说，此模型必须具备足够的详细信息以供软件建造使用。这些构成完整模型的详细信息就是模型的规范说明（简称规范）。所有UML模型成员都有规范说明。
- 不同的模型成分规范说明的内容不同，这些规范说明的内容一般用属性名和属性值的形式来表达。UML中有许多预定义的属性，比如：文档(documentation)、持续性(persistence)和并发性(Concurrency)等。
- 属性一般作为模型成分附加说明，比如，用一些文字逐条列举类的功能，这种规范说明方式是非形式化的。

修饰

- UML模型图中的图素通常都有一个基本的结构，它描述模型成分最主要的特征。为了表达和建造系统的需要，基本图素表达时，UML提供了有选择地公开相关细节的方式。
- 表示的类的属性和方法的可见性图示出了。
- UML模型中类的成分带“-”表示私有的(Private)，带“+”表示公开的（Public），其他类可以访问，带“#”表示受保护的（Protected）。

扩充机制

- 当使用UML的基本图素难以有效地表达复杂事物时，就需要对UML进行某种形式的扩充，正如同人类的语言需要不断地扩充语汇，以描述各种新出现的事物一样。
- UML提供了这种扩充机制（extensibility mechanisms）。

建模规则

- UML的模型图不是UML基本图素的简单堆砌，它必须按特定的规则有机地组合而成，从而构成一个完整的UML模型图。
- UML建模规则包括：
  （1）命名：任何一个UML基本图素和模型图（统称UML成员）都必须命名。
  （2）作用域：UML成员所定义的内容起作用的上下文环境，如类名受所属包的约束。
  （3）可见性：UML成员能被其它成员引用的方式。
  （4）完整性：UML成员之间互相关联的合法性和一致性。
  （5）运行属性：UML成员在运行时的特性，如进程、线程、同步等。

UML建模

- 五个视图：
- 用例视图
- 设计视图
- 进程视图
- 组件视图
- 部署视图

逻辑视图

用例视图

- 用例视图用于描述系统应该具有的功能集。它是从系统的外部用户角度出发，对系统行为的抽象表示。它所描述的系统为用户或另一个系统提供服务 。
- 用例视图主要为用户、系统分析人员、系统开发人员和测试人员服务，这些人员利用用例视图交流，以达到各自的目的。
- 用户和系统分析人员通过用例视图达到系统功能上的共识，系统开发人员通过用例视图完成系统逻辑模型的建立，测试人员通过用例视图测试系统是否满足用户要求。

视图

- 用例视图只考虑系统应提供什么功能，对这些功能的内部运作情况不予考虑，为了揭示系统内部的设计和协作状况，要使用逻辑视图描述系统。

- 逻辑视图用来显示系统内部的功能是怎样设计的。逻辑视图通过类图、对象图描述系统类、对象和它们之间的关系。逻辑视图也称为结构模型视图或静态视图或系统结构建模。

- 原则上软件系统逻辑视图在与软件系统的实现平台无关。逻辑视图包含的模型图有：类图、对象图、交互图。

- 类图是构成逻辑视图的核心，通常用对象描述客观世界中某个具体的实体。建立类模型时，应尽量与应用领域的概念保持一致，使模型更符合客观实际、易修改、易理解和易交流。对象图、交互图描述系统的动态行为。

组件视图

- 当系统的逻辑视图被定义之后， 需要定义逻辑结构的物理实现。
- 这包括：逻辑结构对应的源代码文件、物理文件之间的关系、存放路径等。
- 组件视图主要描述系统部件的配置。通常情况下，组件图看起来似乎是系统物理组件图标的集合。

部署视图

- 提供给开发者、部署、集成者和测试者。在网络开发环境中，部署视图是系统模型必不可少的一部分。
- 部署视图用来描述软件产品在计算机硬件系统和网络上的安装、分发和分布。

- UML建模
- UML将软件的体系结构分解为四个不同的层次，分别是：用例视图、逻辑视图、组件视图、部署视图。其中每个视图分别关注软件开发的一个层次。视图由一种或多种模型图构成。

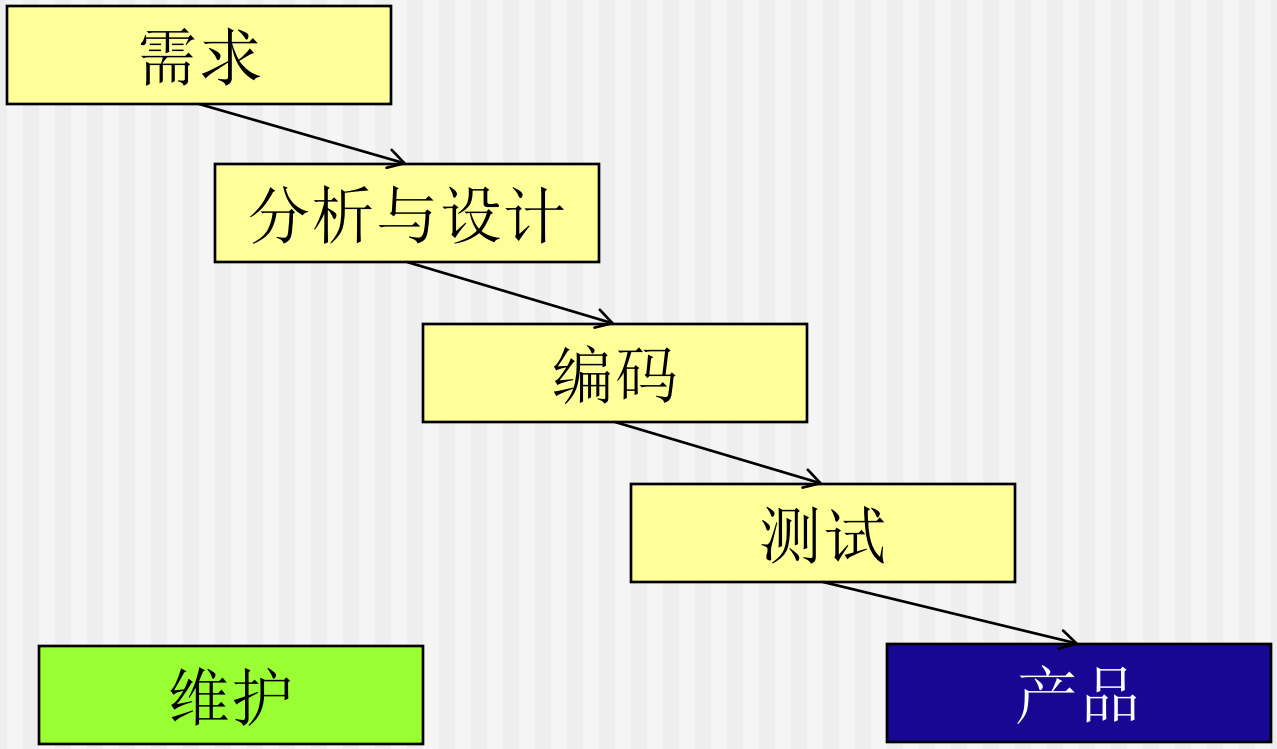
- 用例视图关注系统需求，逻辑视图关注系统的逻辑结构，组件视图关注系统定义逻辑结构的物理实现，部署视图关注系统中软件和硬件之间的物理关系。

建模步骤[1]

- UML是一种建模语言而不是方法，这是因为UML中没有过程的概念，而过程正是方法的一个重要组成部分。UML本身独立于过程，这意味着用户在使用UML进行建模时，可以选用任何适合的过程。
- 一般采用的建模过程有：瀑布开发模型、迭代递增开发模型。

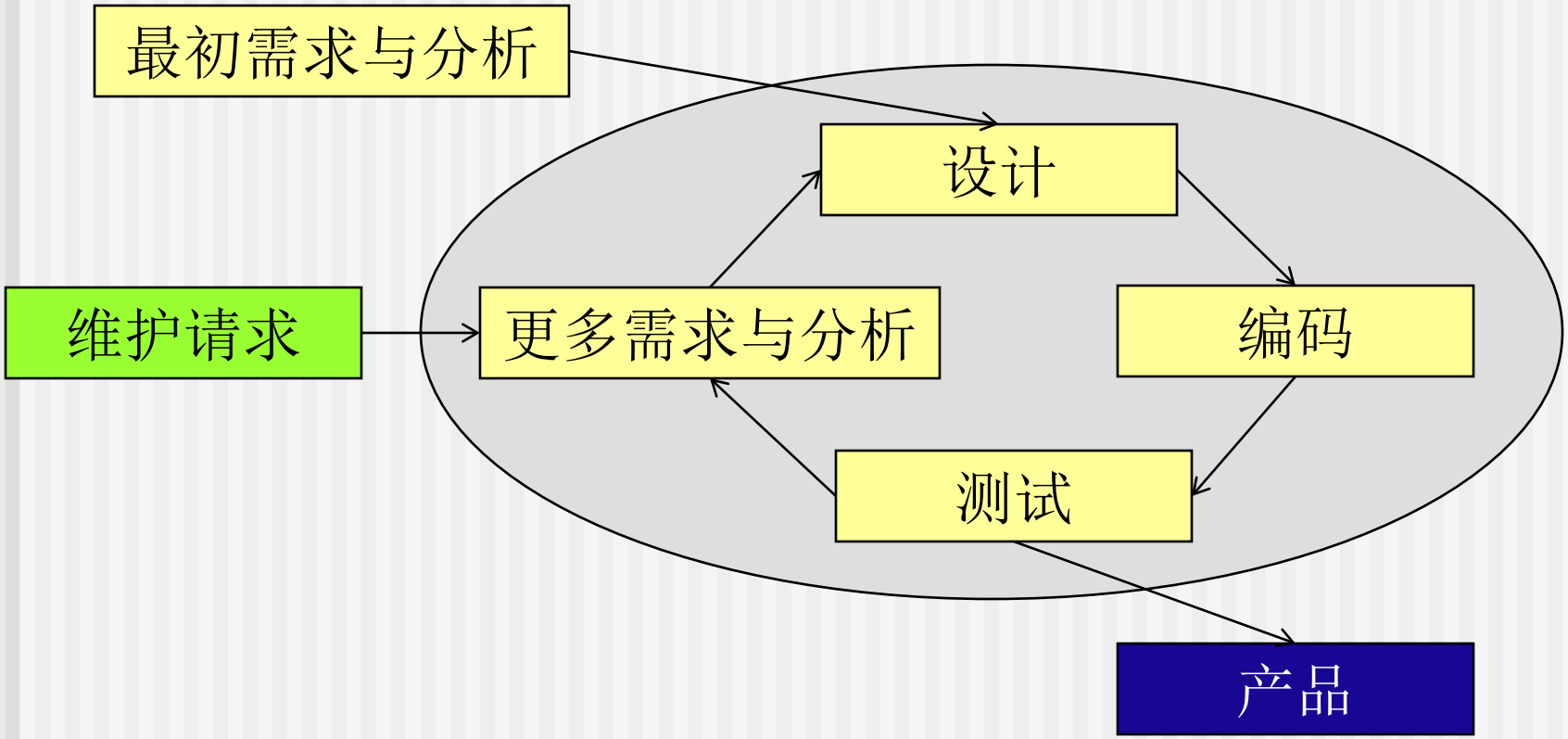
建模步骤[2]—瀑布开发模型

- 瀑布开发模型



建模步骤[3]—迭代递增开发模型

- 迭代递增开发模型



建模步骤[4]—UML建模过程

- 基于UML的系统开发采取增量迭代开发模型。

- [1] 需求　最初需求规格说明应当由代表系统最终用户的人员提供，内容包括系统基本功能需求和对计算机系统的要求。
- [2] 分析　分析的任务是找出系统的所有需求并加以描述，同时建立模型，以定义系统中的关键领域类，应由系统用户和开发人员合作完成。

  分析的第一步是定义用例，以描述所开发系统的外部功能需求。用例分析包括阅读和分析需求说明，此时需要与系统的潜在用户进行讨论。

建模步骤[5]—UML建模过程

- [3] 设计　设计阶段的任务是通过综合考虑所有的技术限制，以扩展和细化分析阶段的模型。
  - 设计阶段可以分为两个部分：结构设计是高层设计，其任务是定义包（子系统），包括包间的依赖性和主要通信机制。我们希望得到尽可能简单和清晰的结构，各部分之间的依赖尽可能的少，并尽可能的减少双向的依赖关系。 第二部分是详细设计，细化包的内容，使编程人员得到所有类的一个足够清晰的描述。

建模步骤[6]—UML建模过程

- 结构设计　一个设计良好的系统结构是系统可扩充和可变更的基础。包实际上是一些类的集合。类图中包括有助于用户从技术逻辑中分离出应用逻辑（领域类），从而减少它们之间的依赖性。
- 详细设计　详细设计的目的是通过创建新的类图、状态图和动态图（顺序图、协作图和活动图），描述新的技术类，并扩展和细化分析阶段的对象类。

建模步骤[7]—UML建模过程

- [4] 实现　构造或实现阶段是对类进行编程的过程。可以选择某种面向对象对象编程语言（如Java）作为实现系统的软件环境。Java很容易实现从逻辑视图到代码部件的映射，因为类到Java代码文件之间是一一映射关系。

  在实现阶段中，可以选取各种图的说明来辅助编程，比如：类图，状态图和动态图等。

建模步骤[8]—UML建模过程

- [5]测试和配置 完成系统编码后，需要对系统进行测试，它通常包括：单元测试、集成测试、系统测试和验收测试。

  在单元测试中使用类图和类的规格说明，对单独的类或一组类进行测试；在集成测试中，使用组件图和合作图，对各组件的合作情况进行测试；在系统测试中，使用用例图，以检验所开发的系统是否满足例图所描述的需求。

  系统的配置是实际地交付系统，包括文档和组成模型等。