第一章 UML概述

主讲教师：徐丙凤

第一章 UML概述

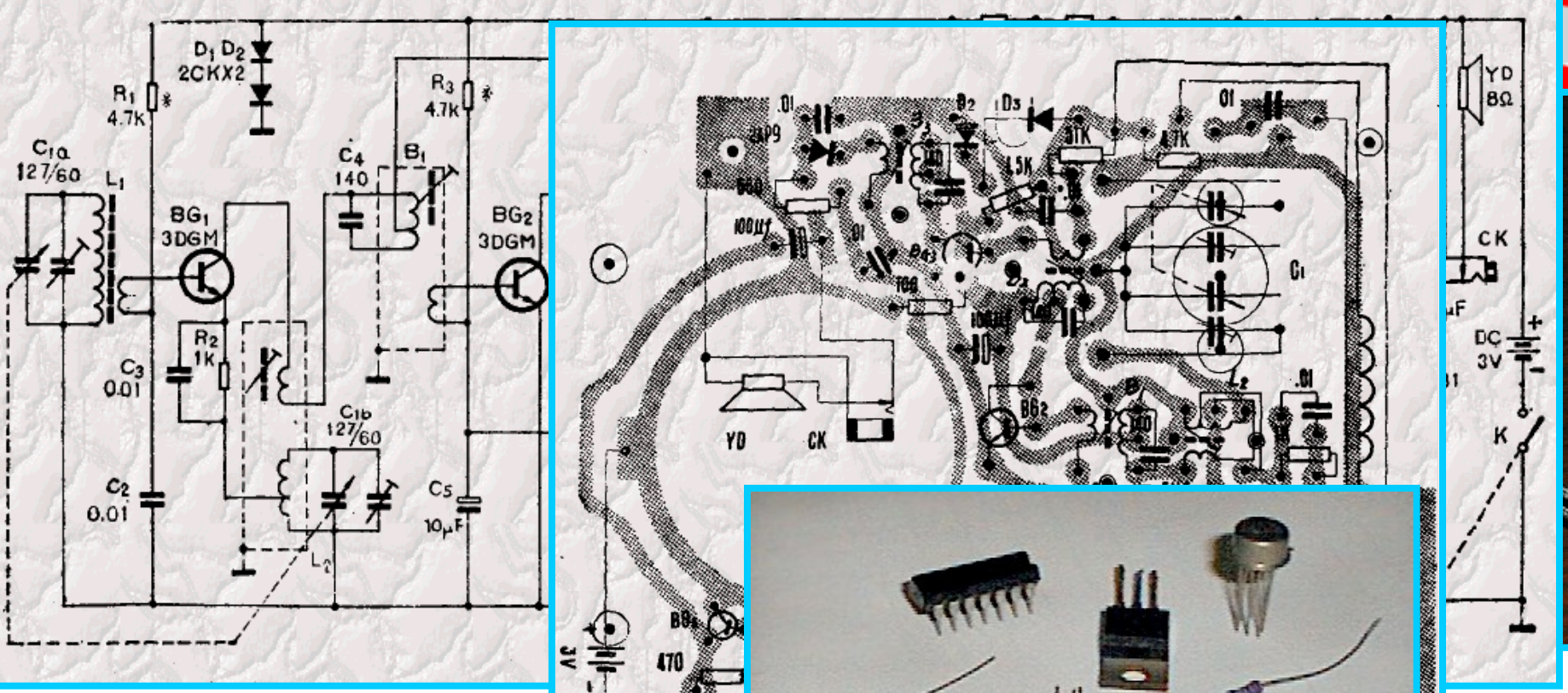
- 1.1 开发软件为什么需要模型
- 1.2 面向对象模型
- 1.3 统一建模语言UML
- 1.4 总结

1.1 模型

- 为了更好的了解一个过程或事物，人们通常根据所研究对象的某些特征 (形状、结构或行为等) 建立相关的模型(Model)
- 模型是从一个特定视点对系统进行的抽象
  - 它可以是实物模型，例如建筑模型、教学模型、玩具等
  - 也可以是抽象数字或图示模型，例如数学公式或图形等

1.1 模型

- 模型的目的不是复制真实的原物，而是帮助人们更好的理解复杂事物的本质，反映过程或事物内部各种因素之间的相互关系
- 模型是对复杂事物进行的有目的简化和抽象
- 在开发软件的过程中同样需要建立各种各样的软件模型



2.然后，设计 体

管符号、电阻电容

3.最后，设计电路板，调试，定型，生产

– 电路图就是模型，是现实世界的简化

为什么要建模

- 模型是对现实世界的简化。
  - 在成熟的工业生产领域，建模的方法得到了广泛应用
- 软件建模
  - 更好理解开发的系统
  - 保障软件质量
  - …

为什么要建模

- 软件是产品而非“程序”
- 对它的要求和所有其他工业产品一样
  - 使用者和制造者分离
  - 质量要求、文档、维护
- 软件产品的生产和其他工业产品的生产也是一样的
  - 生产：团队、工具的使用（Compiler,..）,技术复用
  - 先设计，再生产→ 建模

1.2 开发软件为什么需要模型

- 在开发软件的过程中，开发者在动手编写程序之前需要研究和分析软件的诸多复杂和纷乱的问题
  - 用户需求的准确描述问题
  - 功能与功能之间的关系问题
  - 软件的质量和性能问题
  - 软件的结构组成问题
  - 建立几十个甚至几百个程序或组件之间的关联问题等等
- 在这个复杂的开发过程中，我们最关注的还是开发者之间的交流问题

1.2 开发软件为什么需要模型

- 软件开发中能否消除技术人员与非技术人员(用户)之间、使用不同技术的开发人员之间、不同功能使用者之间的等等交流障碍是软件开发成功的关键。
- 直观的软件模型将有助于软件工程师与他们进行有效的交流。

1.2 开发软件为什么需要模型

- 软件设计者可以通过建立需求模型来实现技术人员与非技术人员(用户)之间的交流
- 在软件的设计中，设计人员首先要把描述系统功能需求的自然语言形式转化为软件程序的逻辑形式，在这个转化过程中，设计人员要借助许多模型来完成最终的程序设计模型
- 在软件的实施、测试和部署中，模型为不同领域的技术人员在软件和硬件的实施、测试和部署中提供有效的交流平台
- 软件模型是最有效的软件文档保存形式，软件模型在开发团队人员的培训、学习和知识的传递和传播等方面起着非常重要的作用

1.2 开发软件为什么需要模型

- 软件开发中需要建立
  - 需求(Requirement)模型
  - 问题域 (Domain) 模型
  - 设计(Design)模型
  - 实施模型
  - 测试模型
  - 部署模型
- 在系统开发生命周期中，需要从多角度来建立模型才能全面、准确地分析和设计软件系统

如何建模

- 建模通过对客观事务建立一种抽象的方法用以表示事物并获得对事物本身的理解。同时把这种理解概念化，用逻辑组织起来，以表达所观察的对象的内部结构和工作原理。
- 建模包含两个问题：
  - 怎么建模：即从哪些抽象角度认识和描述这个世界
  - 模型是什么：多个不同的抽象角度对问题域的描述

模型怎么建

- 怎么建：即从哪些抽象角度认识和描述这个世界
  - 面向对象和面向过程是两个不同的建模方法论。
  - 每个人对事物认识都有自己的抽象角度。

- 面向过程和面向对象的抽象角度是不同的
  - 面向过程：找出需要处理的原始数据，通过多道加工转化为需要的数据。
  - 面向对象：把事物分解成对象及对象之间的联系，使问题简单化。

第一章 导言

- 1.1 开发软件为什么需要模型
- 1.2 面向对象模型
- 1.3 统一建模语言UML
- 1.4 总结

- 在面向对象的程序设计方法出现之前，传统的程序设计方法大都是面向过程的。面向过程的程序设计结构清晰，它在历史上为缓解软件危机做出了贡献。面向过程的程序设计方法是以功能分析为基础的，它强调自顶向下的功能分解，并或多或少地把功能和数据进行了分离。

- 面向过程的方法存在如下问题：

（1）软件系统是围绕着如何实现一定的功能来进行的，当功能中的静态和动态行为发生变化需要修改时，修改工作颇为困难。

（2）程序员对客观世界的认知，与程序设计之间存在着鸿沟。

（3）面向过程的程序设计导致模块间的控制作用只能通过上下之间的调用关系来进行，这样会造成信息传递路径过长、效率低、易受干扰甚至出现错误。

（4）面向过程的方法开发出来的系统往往难以维护，因为所有的函数都必须知道数据结构。

（5）自顶向下功能分解的分析方法大大限制和降低了软件的易复用性，导致对同样对象的大量的重复性工作，从而降低了开发人员的生产率。

面向对象建模

- 面向对象的软件开发方法涉及从面向对象分析（OOA）→面向对象设计（OOD）→面向对象程序设计或编码（OOP）→面向对象测试（OOT）等一系列特定阶段。
- 面向对象设计方法期望获得一种独立于语言的设计描述，以求达到从客观世界中的事物原型到软件系统间的尽可能的平滑过渡。

面向对象建模

- 面向对象的方法把功能和数据看作是高度统一的，优点有：

（1）更好地诠释了软件度量中“高内聚，低耦合”的评价准则。

（2）能较好地处理软件的规模和复杂性不断增加所带来的问题。

（3）更适合于控制关系复杂的系统。

（4）面向对象系统通过对象间的协作来完成任务，更容易管理。

（5）使用各种直接模仿应用域中实体的抽象和对象，使得规约和设计更加完整。

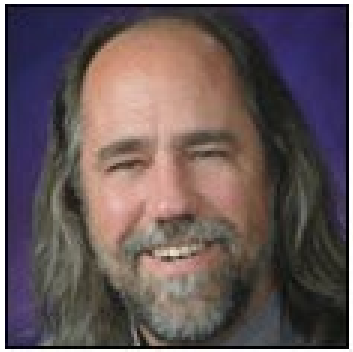
（6）围绕对象和类进行局部化，提高了规约、设计和代码的易扩展性、易维护性和易复用性。

（7）简化了开发者的工作，提高了软件和文档的质量。

面向对象建模

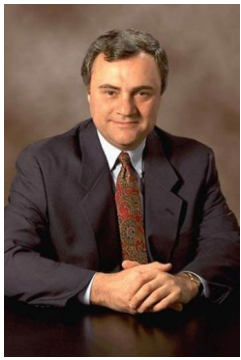
面向对象建模语言问世于20世纪70年代中期。

Booch是面向对象方法**最早**的倡导者之一，他提出了**面向对象软件工程**的概念。**Booch**在其**OOAda**中提出了面向对象开发的**4**个模型：**逻辑视图、物理视图及其相应的静态和动态语义**。



Grady Booch
IBM Fellow

Jacobson



Rumbaugh

- **Jacobson**于**1994**年提出了面向对象的软件工程（**OOSE**）方法，该方法的最大特点是面向用例。
- **Rumbaugh**等人提出了**OMT**方法。**OMT**方法中，系统是通过对象模型、动态模型和功能模型来描述的。

第一章 导言

- 1.1 开发软件为什么需要模型
- 1.2 面向对象模型
- 1.3 统一建模语言UML
- 1.4 总结

1.3 统一建模语言UML

- UML（Unified Modeling Language，统一建模语言），是一种能够描述问题、描述解决方案、起到沟通作用的语言。它是一种用文本、图形和符号的集合来描述现实生活中各类事物、活动及其之间关系的语言。
- UML是一种很好的工具，可以贯穿软件开发周期中的每一个阶段，适用于数据建模、业务建模、对象建模和组件建模。UML使开发人员专注于建立产品的模型和结构，而不是选用什么程序语言和算法实现。当模型建立之后，模型可以被UML工具转化成指定的程序语言代码。

什么是统一建模语言

- **UML-Unified Modeling Language**
  - 将现实世界映射成软件世界的一种图形化描述语言。
  - 组合了当前最好的面向对象软件建模方法
- 三位主要贡献者

Grady Booch

Booch方法论

Ivar Jacobson

OMT方法论

Jim Rumbaugh

OOSE方法论

- UML是一种Language（语言）
- UML是一种Modeling(建模)Language
- UML是Unified(统一)Modeling Language
- 已进入全面应用阶段的事实标准
- 应用领域正在逐渐扩展，包括嵌入式系统建模、流程建模等多个领域。
- 成为“生产式编程”的重要支持技术：MDA、可执行UML等。

UML能为我们做什么

- UML可以做软件需求分析
- UML可以做软件开发设计
- UML可以做系统部署设计
- UML也适用非软件领域的系统建模如企业机构或业务过程，以及处理复杂数据的信息系统、具有实时要求的工业系统或工业过程等。

UML的历史及发展

- 软件工程领域在1995年至1997年取得了前所未有的进展，其成果超过软件工程领域过去15年来的成就总和。
- 其中最重要的、具有划时代重大意义的成果之一就是统一建模语言UML的出现。
- 在世界范围内，至少在近10年内，UML将是面向对象技术领域内占主导地位的标准建模语言

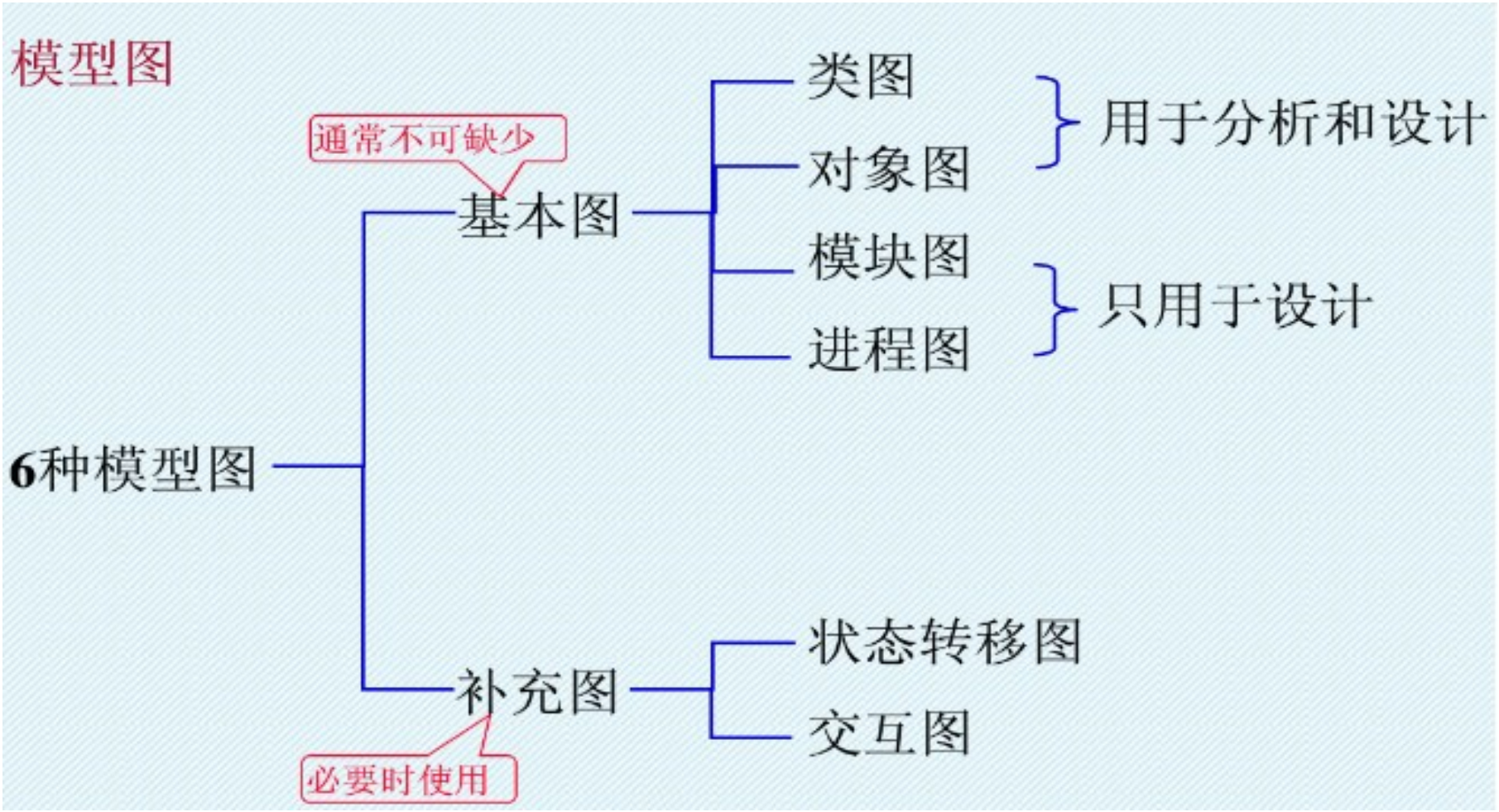
1.4 UML的发展史

- 面向对象的编程语言发展得比较成熟以后，人们开始将面向对象扩展到分析和设计领域
- 面向对象的建模语言
  - 问世于20世纪70年代中期
  - 1989年－1994年，10种→50多种
  - 面向对象技术的方法大战

UML历史—Booch1993方法

- 1991年，Grady Booch
  - 识别类和对象，识别其语义、关系，实现
- 丰富的符号体系
  - 逻辑—静态视图
    - 类图、对象图
  - 逻辑—动态视图
    - 状态转移图、交互图、时态图
  - 物理—静态视图
    - 模块图、进程图
- 特点
  - 强于设计、弱于分析
  - 偏向于系统的静态描述，动态描述支持较少
  - 比较关注于系统设计和构造阶段，不关注需求

- **Booch方法**

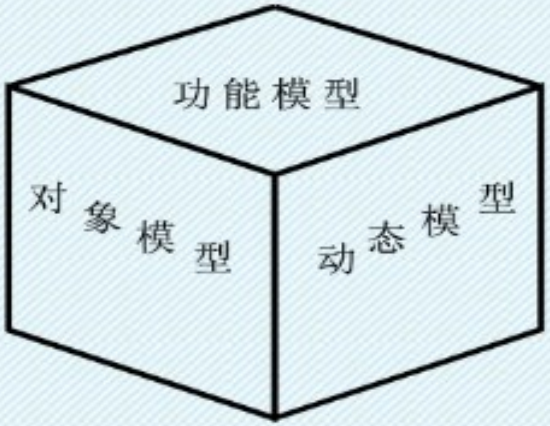

模型图
6种模型图
基本图
通常不可缺少
类图
对象图
模块图
进程图
用于分析和设计
只用于设计
补充图
必要时使用
状态转移图
交互图

UML历史－OMT-2方法

- 1991，James Rumbaugh等
  - 系统建模
    - 对象模型－对象的静态结构及关系（对象图）
    - 动态模型－对象的时间变化（状态图）
    - 功能模型－系统实现的功能（数据流图）
  - 概念和符号用于分析、设计和实现全过程
  - 开发过程分为四个阶段：分析，系统设计、对象设计、实现
- 特点
  - 分析能力很强
  - 适合于分析和描述以数据密集型信息系统

- **OMT方法**

三个模型



过程：

分析（面向对象）
系统设计（传统方法）
对象设计（面向对象）
实现

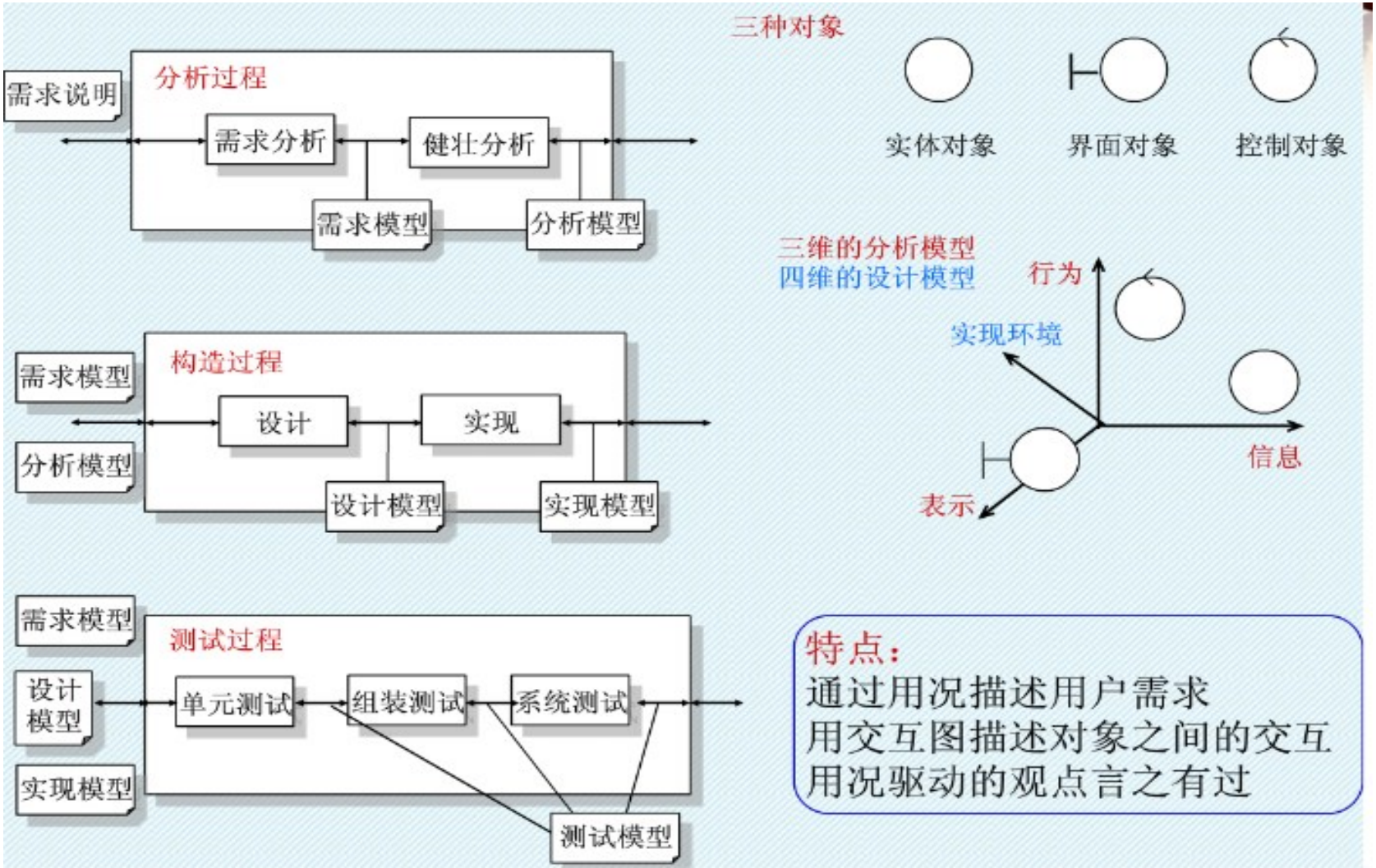
特点：

概念严谨，阐述清楚
过程具体，可操作性强
包含了许多非OO的内容
提出若干扩充概念，偏于复杂

UML历史-OOSE方法

- 1992，Ivar Jacobson
  - 面向用例（Use Case）在用例中引入了外部角色的概念
  - 涉及到整个软件生命周期
- Use Case贯穿始终，驱动其它模型的开发
- 使用的5种其它系统模型
  - 领域对象模型—根据领域来表示Use case
  - 分析模型—通过分析来构造
  - 设计模型—通过设计来具体化use case
  - 实现模型—实现use case模型
  - 测试模型—用来测试具体化的use case模型
- 特点
  - 强于工程和项目的分析阶段

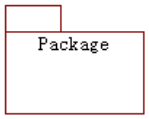
- **OOSE方法**


三种对象
实体对象
界面对象
控制对象
分析过程
需求说明
需求分析
健壮分析
需求模型
分析模型
三维的分析模型
四维的设计模型
行为
实现环境
信息
表示
构造过程
需求模型
分析模型
设计
实现
设计模型
实现模型
测试过程
需求模型
设计模型
实现模型
单元测试
组装测试
系统测试
测试模型
特点：
通过用况描述用户需求
用交互图描述对象之间的交互
用况驱动的观点言之有过

UML历史—统一的尝试

- 大批关于面向对象方法，各有自己的一套概念、定义、表示法、术语和适用的开发过程。
- 总的来说各个作者所使用的概念大同小异。
- 统一的初期尝试，是Coleman及同事对OMT、Booch、CRC方法使用的概念进行融合。
- 由于这项工作没有这些方法的原作者参与，实际上仅仅形成了一种新方法。

UML的统一

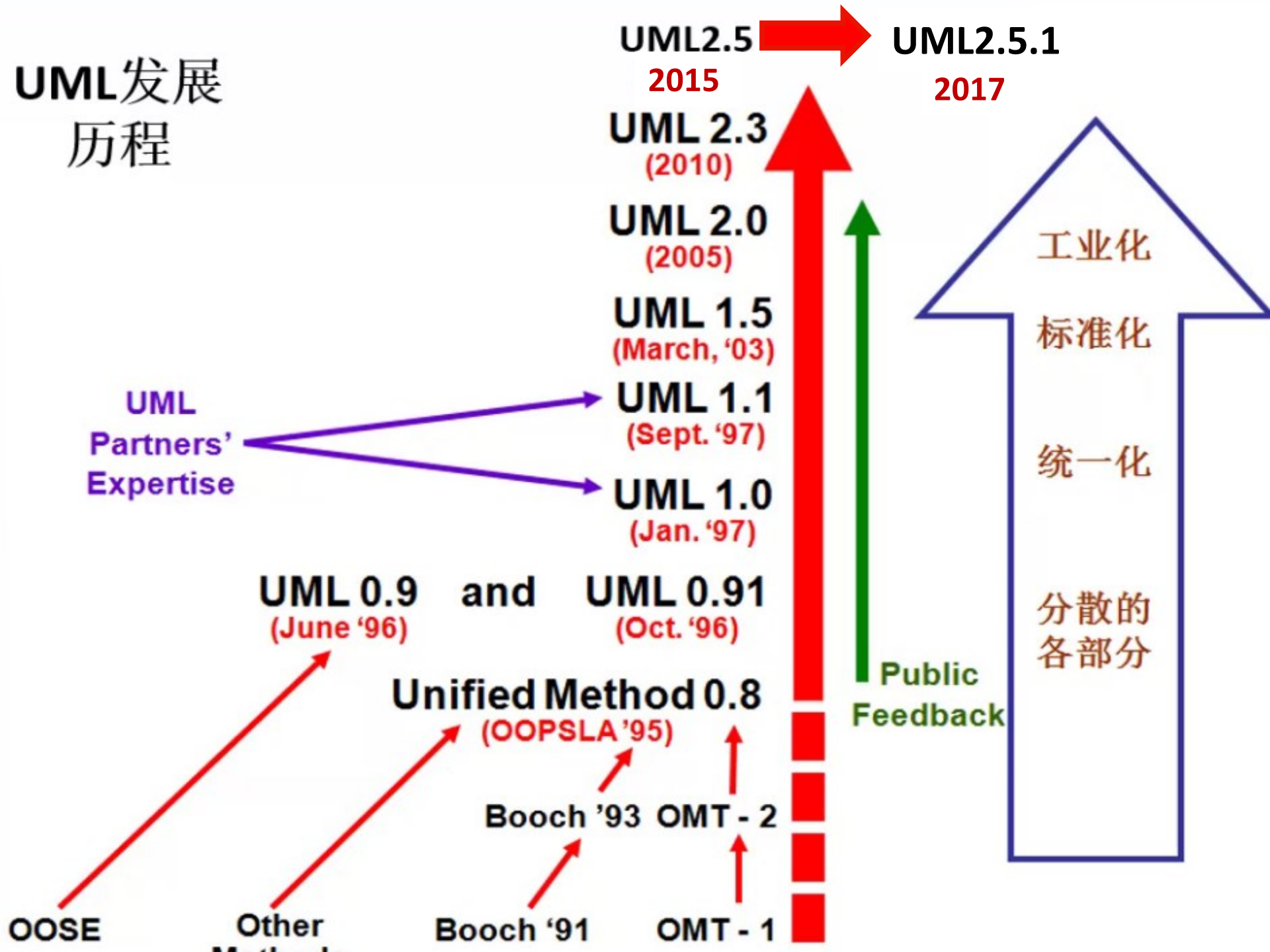
- UML对各个流派的建模符号和概念进行了归纳，并提供了非常严谨的定义和描述，成为真正的建模语言。
- Booch、Jacobson、Rumbaugh三人的表示法在UML中占了主要部分。
- 但还是有很多部分来自各方
  - 接口来自Microsoft
  - 

包的符号来自Apple Macintosh
  - 活动图来自James Odell
  - 状态图来自David Harel

UML历史

- 1994年10月，Grady Booch和Jim Rumbaugh首先将Booch93和OMT-2统一起来。
- 1995年10月发布了第一个公开版本，称之为统一方法UM 0.8（Unified Method）
- 1995年秋，OOSE 的创始人Ivar Jacobson加盟到这一工作，并力图把OOSE方法也统一进来。
- 1995年，与OMG达成协议以使得UML成为一个标准
- 1996年6月和10月，分别发布了两个新版本，UML0.9和UML0.91，并重新命名为UML
- 1997年7月UML1.0版本被提供给对象管理工作组（OMG），11月被OMG采纳为业界标准
- 2005年，OMG发布了UML2.0

UML发展历程


UML2.5
2015
UML2.5.1
2017
UML 2.3
(2010)
UML 2.0
(2005)
UML 1.5
(March, '03)
UML 1.1
(Sept. '97)
UML 1.0
(Jan. '97)
UML
Partners'
Expertise
UML 0.9 and UML 0.91
(June '96) (Oct. '96)
Unified Method 0.8
(OOPSLA '95)
Booch '93
OMT - 2
OOSE
Other
Booch '91
OMT - 1
Public
Feedback
工业化
标准化
统一化
分散的
各部分

UML的特点

标准建模语言UML的主要特点可以归结为三点：

- UML统一了Booch、OMT和OOSE等方法中的基本概念和符号。
- UML吸取了面向对象领域中各种优秀的思想，其中也包括非OO方法的影响。
- UML在演变过程中还提出了一些新的概念。在UML标准中新添加了模板（Stereotypes）、职责（Responsibilities）、扩展机制（Extensibility Mechanisms）、线程（Threads）、过程（Processes）、分布式（Distribution）、并发（Concurrency）、模式（Patterns）、合作（Collaborations）、活动图（Activity Diagram）等概念，并清晰地区分类型（Type）、类（Class）、实例（Instance）、细化（Refinement）、接口（Interfaces）和组件（Components）等概念。

UML语言的构成

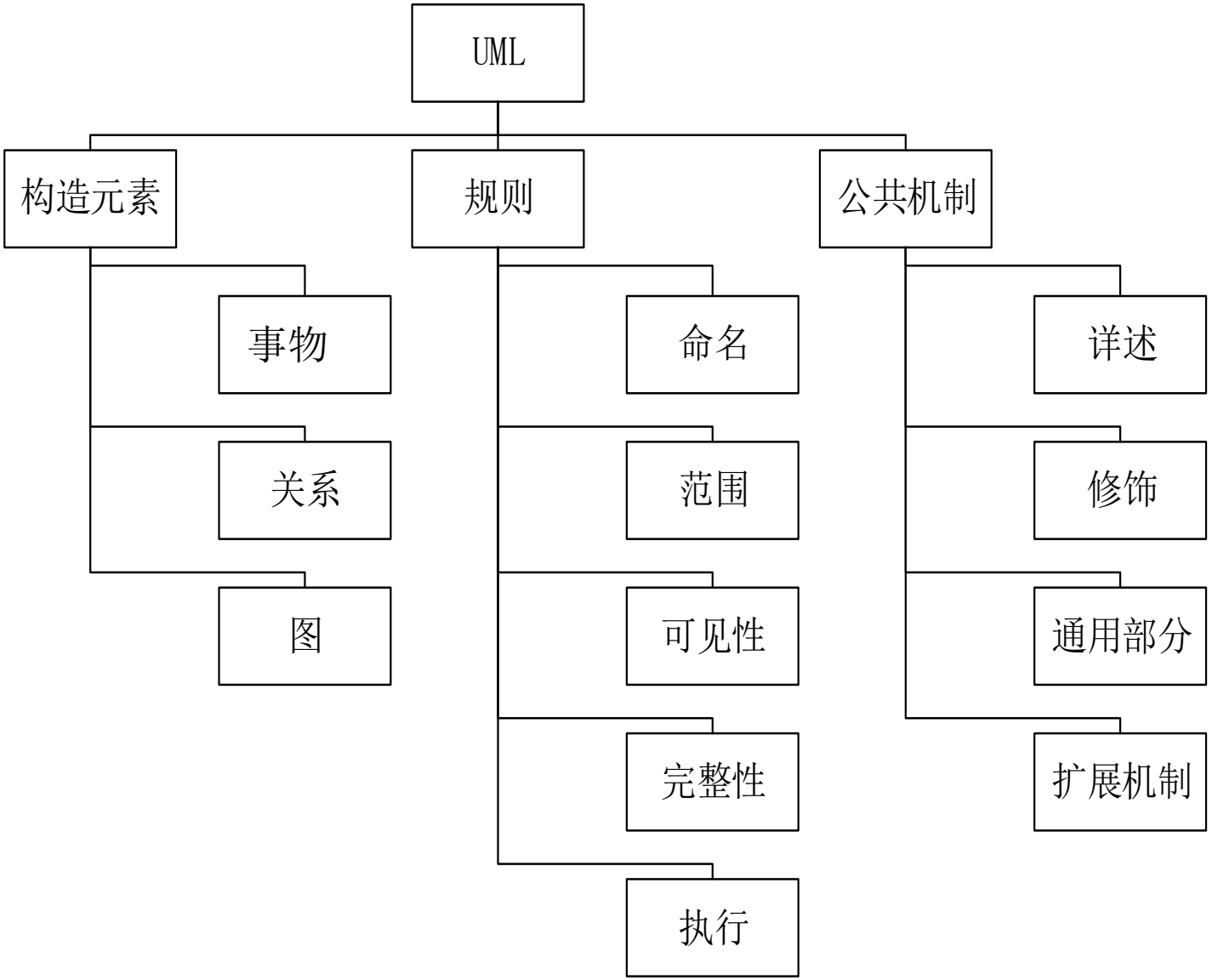
UML语言是一门设计语言，这种语言由一些构造元素、规则和公共机制构成。

构造元素描述事物的基本成分，这些基本成分按某种规则关联在一起，组成图；同时，这些基本元素都遵循通用规则，即公共机制。

（1）构造元素包括事物、关系和图。这3种元素描述了软件系统或业务系统中的某个事物或事物间的关系。构造元素应该具有命名、范围、可见性、完整性和执行等属性。

（2）规则是对软件系统或业务系统中的某些事物的约束或规定。

（3）公共机制包括详述、修饰、通用划分、扩展机制。公共机制指适用于软件系统或业务系统中每个事物的方法或规则。


UML
构造元素
规则
公共机制
事物
关系
图
命名
范围
可见性
完整性
执行
详述
修饰
通用部分
扩展机制

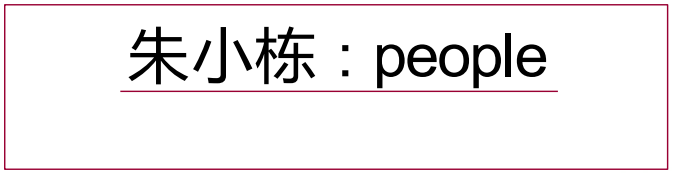
UML语言的事物

- UML定义了4种基本的面向对象的事物
  - 构件事物
  - 行为事物
  - 分组事物
  - 注释事物

- 构件事物是UML模型的静态部分、描述概念或物理元素。在UML规范中，一共定义了七种构件事物，分别是类和对象、接口、协作、用例、构件和节点。

1.类和对象

类的表示方法



对象的表示方法

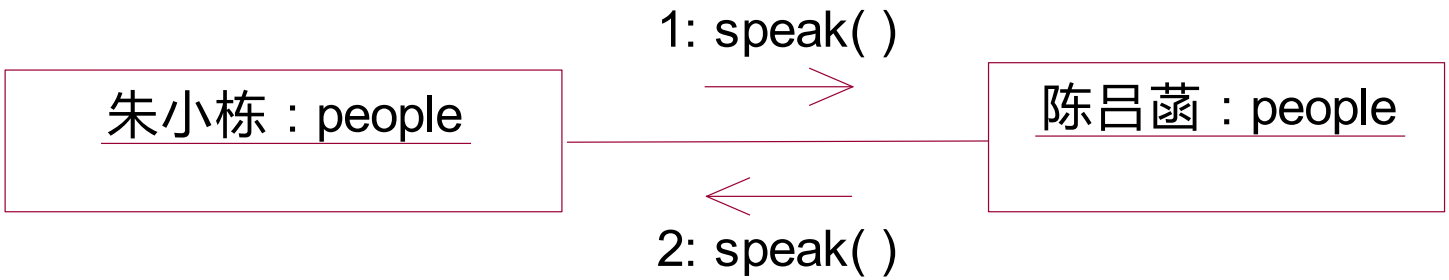
2.接口

接口是指为类或组件提供特定服务的一组操作的集合。一个接口描述了类或组件对外可见的动作。一个接口可以实现类或组件的全部动作，也可以实现一部分。

接口分为供给接口和需求接口两种，供给接口只能向其它类(或构件)提供服务，需求接口表示类(或构件)使用其它类(或构件)提供的服务。

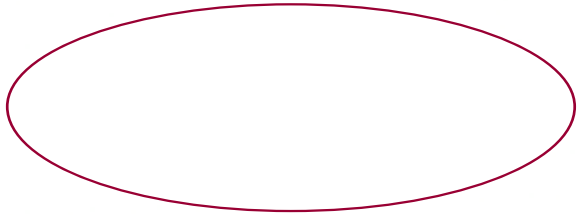
3.协作

协作描述了一组事物间的相互作用集合。通常来说，这些协作行为大于所有元素的行为的总和。一个类可以参与到多个协作中，在协作中表现了系统构成模式的实现。



4.用例

用例是著名的大师UML创始人之一Ivar Jacobson首先提出的，用于表示系统所提供的服务，它定义了系统是如何被参与者所使用的，它描述的是参与者为了使用系统所提供的某一完整功能而与系统之间发生的一段对话。用例是对组动作序列的抽象描述。



用例名称

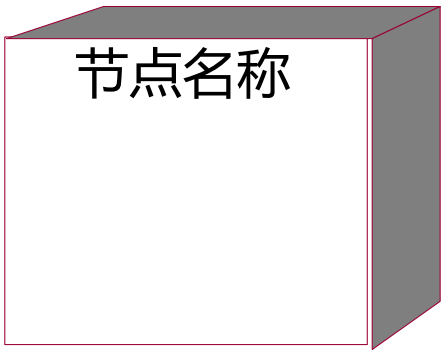
UML语言的基本元素

5.构件

构件也称组件，是定义了良好接口的物理实现单元，它是系统中物理的、可替代的部件。它遵循且提供一组接口的实现，每个构件体现了系统设计中特定类的实现。良好定义的构件不直接依赖于其他构件而依赖于构件所支持的接口。在这种情况下，系统中的一个构件可以被支持正确接口的其他构件所替代。在每个系统中都有不同类型的部署构件，如 JavaBean、DLL、Applet和可执行exe文件等。构件通常采用带有2个小方框的矩型表示。

6.节点

节点是系统在运行时切实存在的物理对象，表示某种可计算资源，这些资源往往具有一定的存储能力和处理能力，如PC机、打印机、服务器等都是节点。一个构件集可以驻留在一个节点内，也可以从一个节点迁移到另一个节点。在UML中，用一个立方体表示一个节点。



课堂提问

- 下列事物不属于UML构件事物的是（　）。

  A. 组件

  B. 类

  C. 节点

  D. 状态机

答案：D

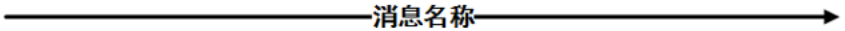
UML语言的基本元素

- **行为事物**

行为事物是指 UML模型的相关动态行为，是UML模型的动态部分，它可以用来描述跨越时间和空间的行为。行为元素在模型中通常使用动词来进行表示，如“注册”、“登录”、“购买”等动作。行为元素可以划分为两类，分别是交互和状态机。

1．交互

交互是指在特定的语境中，一组对象为共同完成一定任务，在进行的一系列消息交换的过程中，所形成的消息机制。因此，在交互中，不仅包括一组对象、对象间的普通连接，还包括连接对象间的消息，以及消息发出的动作形成的有序的序列。交互的表示法很简单，用一条有向直线来表示对象间的交互，并在有向直线上面标有消息名称。



交互的表示方法

2．状态机

状态机是一个描述类的对象所有可能的生命历程的模型，因此状态机可用于描述一个对象或一个交互在其生命周期内响应时间所经历的状态的序列。当对象探测到一个外部事件后，它依照当前的状态做出反应，这种反应包括执行一个相关动作或转换到一个新的状态中去。单个类的状态变化或多个类之间的协作过程都可以用状态机来描述，利用状态机可以精确地描述行为。在UML模型中，将状态表示为一个圆角矩形，并在矩形内标识状态名称。

课堂提问

- 下列叙述中，属于UML行为事物的是（　）。

A. 状态

B. 协作

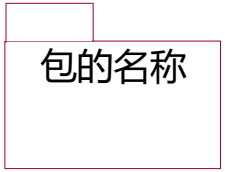
C. 交互

D. 用例

答案：C

- **分组事物**

分组事物是UML中对模型中的各种组成部分进行事物分组的一种机制。可以把分组事物当成是一个“盒子”，那么不同的“盒子”就存放不同的模型，从而模型在其中被分解。对于一个中大型的软件系统而言，通常会包含大量的类、接口、交互，因此也就会存在大量的结构元素、行为元素。为了能有效地对这些元素进行分类和管理，就需要对其进行分组。在UML中，提供了“包（Package）”来实现这一目标。表示“包（Package）”的图形符号，与windows中表示文件夹的图符很相似。包的作用与文件夹的作用也相似，



- **注释事物**

注释事物是UML模型的解释部分，用于进一步说明UML模型中的其他任何组成部分。可以用注释事物来描述、说明和标注整个UML模型中的任何元素。

注解是依附于某个元素或一组建模元素之上，对这个或这组建模元素进行约束或解释的简单注释符号。注解的一般形式是简单的文本说明，可以帮助我们更加详细地解释要说明的模型元素所代表的内容。注释元素的表示方法如图所示。

课堂提问

- UML中的事物包括构件事物、分组事物、注释事物和（　　）。

A. 实体事物

B. 边界事物

C. 控制事物

D. 行为事物

答案：D

UML中的关系

UML中有4种关系，包括：

- 依赖
- 关联
- 泛化
- 实现

1. 依赖关系

依赖关系指的是两个事物之间的一种语义关系，当其中一个事物（独立事物）发生变化就会影响另外一个事物（依赖事物）的语义。如下图所示，反映了元素X依赖于元素Y。

(独立元素)- - - - - - - - - - - -＞(依赖元素)

本质上说，关联和泛化以及实现关系都属于依赖关系的一种，但是它们有更特别的语义，因此分别定义了其的名字和详细的语义。

2. 关联关系

关联指明了一个对象与另一个对象间的关系。在图形上，关联用一条实线表示。在关联关系中，有两种比较特殊的关系，它们是聚合关系和组合关系。

1）关联关系的表示

关联关系是聚合关系和组合关系的统称，是比较抽象的关系；聚合关系和组合关系是更具体的关系。在UML中，使用一条实线来表示关联关系，如图所示。

关联关系

2）聚合关系

- 聚合是一种特殊形式的关联。聚合表示类之间的关系是整体与部分的关系。聚合关系是一种松散的对象间关系。
- 如：计算机和它的外围设备（如音箱）就是一例。一台计算机和它的外设之间只是很松散地结合在一起。这些外设可有可无，可以与其他计算机共享，而且没有任何意义表明它由一台特定的计算机所“拥有”——这就是聚合。
- 聚合的表示如下图所示，菱形箭头为空心箭头，菱形端表示事物的整体，另一端表示事物的部分。如计算机就是整体，外设就是部分。

聚合关系

3）组合关系

- 如果发现“部分”类的存在，是完全依赖于“整体”类的，那么就应该使用“组合”关系来描述。
- 组合关系是一种非常强的对象间关系，例如，树和它的树叶之间的关系。某棵树是和它的叶子紧密联系在一起，叶子完全属于这树，它们不能被其它的树所分享，并且当树死掉，叶子也会随之死去——这就是组合。
- 组合是一种强的聚合关系。组合的表示如下图所示，菱形箭头为实心箭头。

组合关系

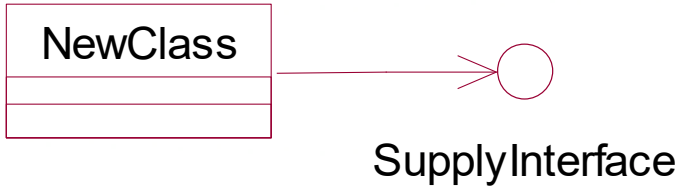
3.泛化关系

- 泛化关系是事物之间的一种特殊/一般关系，特殊元素（子元素）的对象可替代一般元素（父元素）的对象，也就是面向对象中的继承关系。
- 通过继承，子元素具有父元素的全部结构和行为，并允许在此基础上再拥有自身特定的结构和行为。
- 在系统开发过程中，泛化关系的使用并没有什么特殊的地方，只要注意能清楚明了地刻画出系统相关元素之间所存在的继承关系就行了。

鱼和鲫鱼之间的泛化关系

4. 实现关系

- 实现关系也是UML元素之间的一种语义关系，它描述了一组操作的规约和一组对操作的具体实现之间的语义关系。在系统的开发中，通常在两个地方需要使用实现关系，一种是用在接口和实现接口的类或构件之间，另一种是用在用例和实现用例的协作之间。当类或构件实现接口时，表示该类或构件履行了在接口中规定的操作。下图描述的是类对接口的实现关系。



课堂提问

- UML的关系不包括（　）。

A. 抽象

B. 实现

C. 依赖

D. 关联

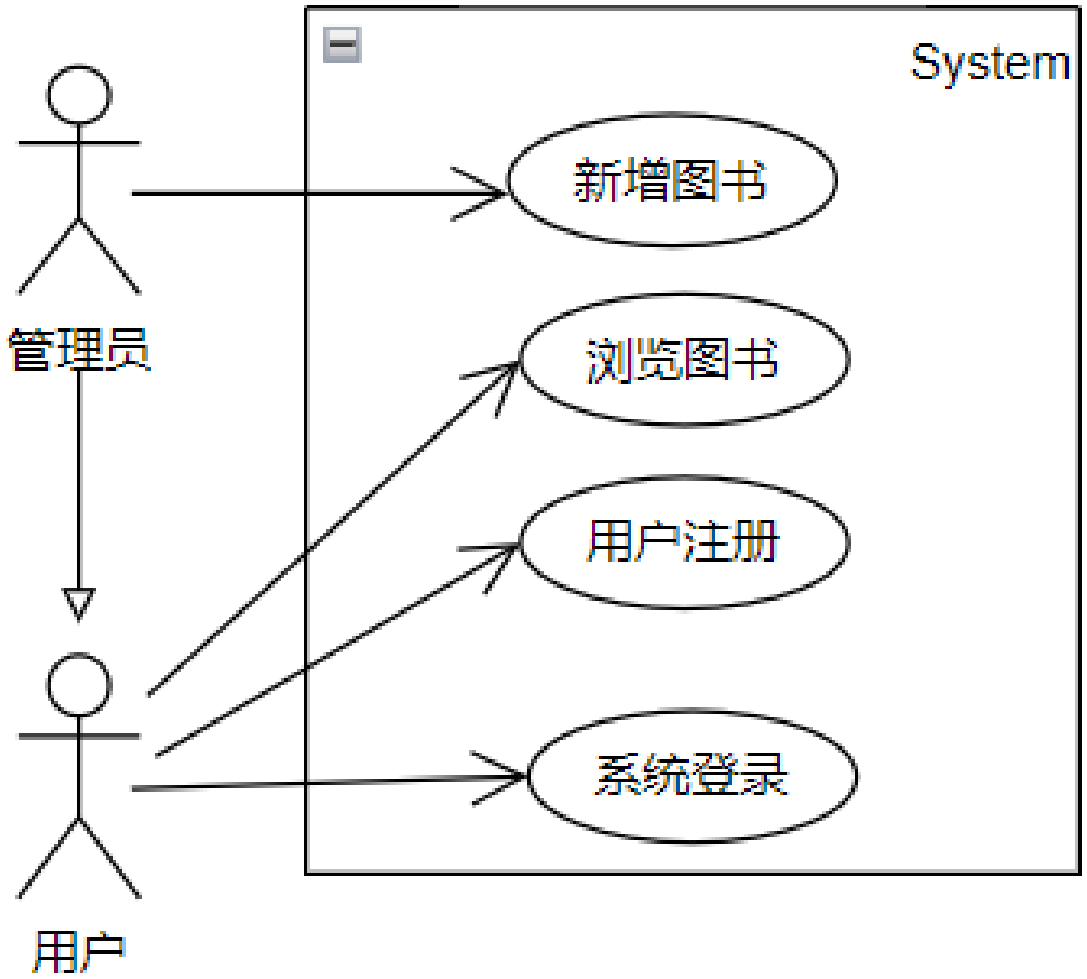
答案：A

UML2.5的图

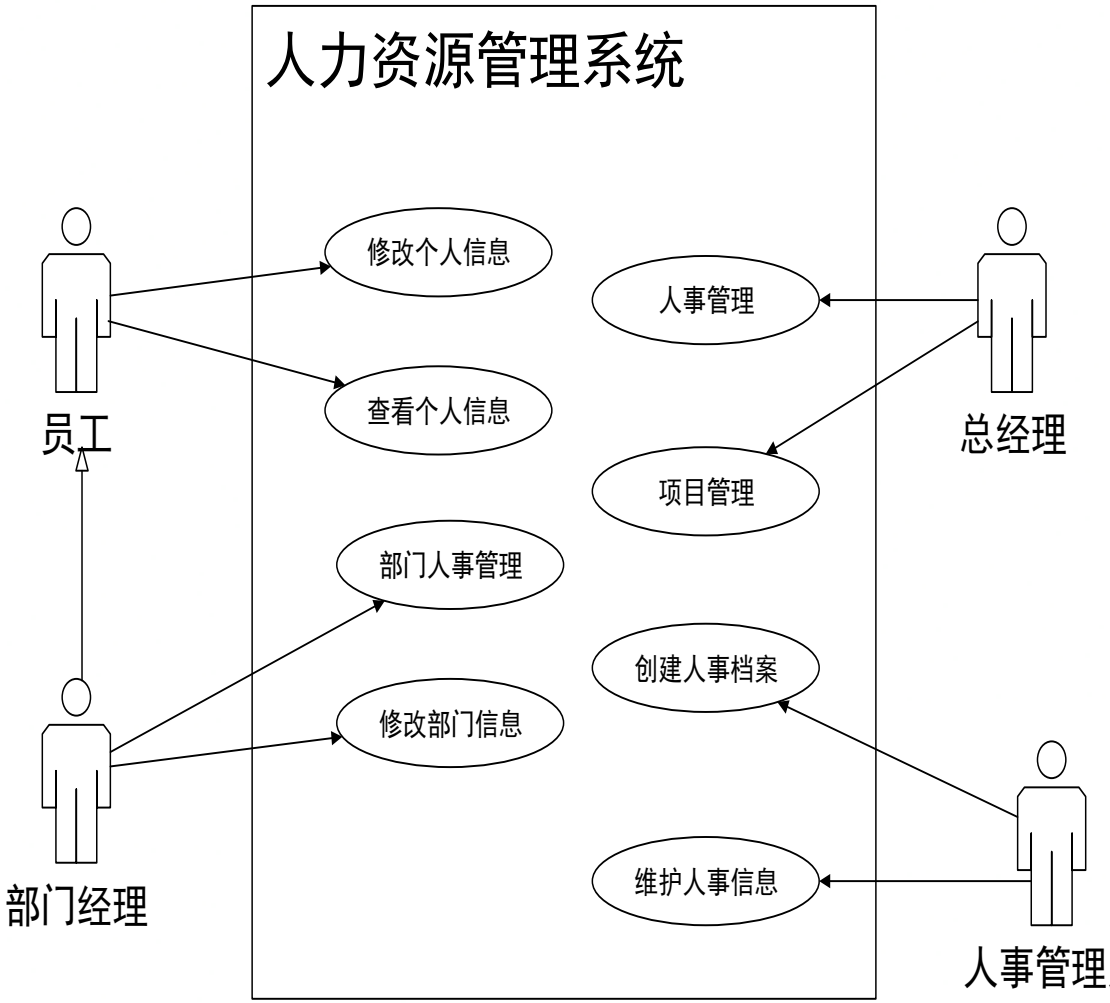
- UML中的图是描述UML视图内容的图形。对比UML 1.x，UML 2.0增加了“包图”、“组合结构图”、“交互概览图”和“定时图”，一共13种不同的图，通过它们的相互组合提供被建模系统的所有视图。UML 2.0在可视化建模方面进行了许多改革和创新。它可以描述现今软件系统中存在的许多技术，例如模型驱动架构（MDA）和面向服务的架构（SOA）。UML2.5中一共有14种模型图（新增了概要图）。

1. 用例图

- 用例图是从用户角度描述系统功能，并指出各功能的操作者。用例图是UML中最简单也是最复杂的一种图。说它简单是因为它采用了面向对象的思想，基于用户角度来描述系统，绘制非常容易，图形表示直观并且容易理解。说它复杂是因为用例图往往不容易控制，要么过于复杂，要么过于简单。用例图展示了一组用例、参与者以及它们之间的关系。

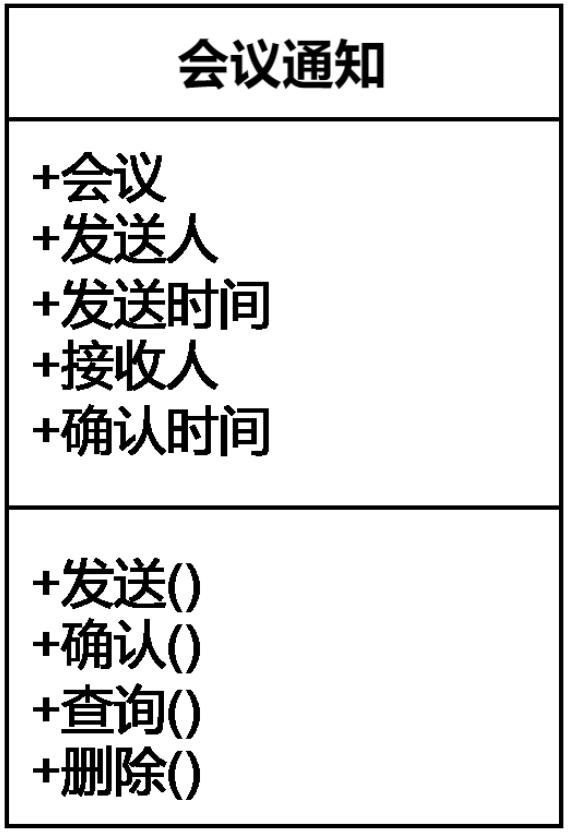

System
新增图书
浏览图书
用户注册
系统登录
管理员
用户

- 如图所示是一个人力资源管理系统的用例视图。这是一个简单的用例视图，但却包含了系统、用户和各种用户在这个系统中做什么事情等信息。



2. 类图

- 类图是UML面向对象中最常用的一种图，类图可以帮助我们更直观地了解一个系统的体系结构。通过关系和类表示的类图，可以图形化地描述一个系统的设计部分。

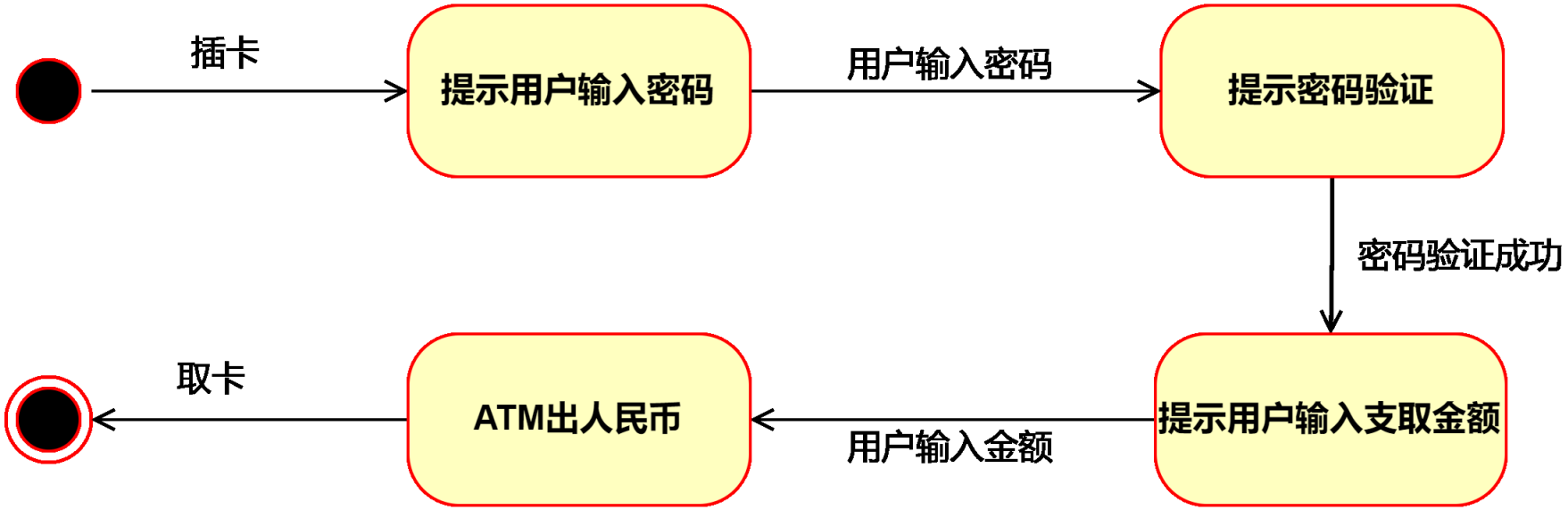


3. 对象图

- 在面向对象中对象图是类图的实例，几乎使用与类图完全相同的标识。它们的不同点在于对象图显示类的多个对象实例，而不是实例的类。一个对象图是类图的一个实例。由于对象存在生命周期，因此对象图只能在系统的某一时间段存在。

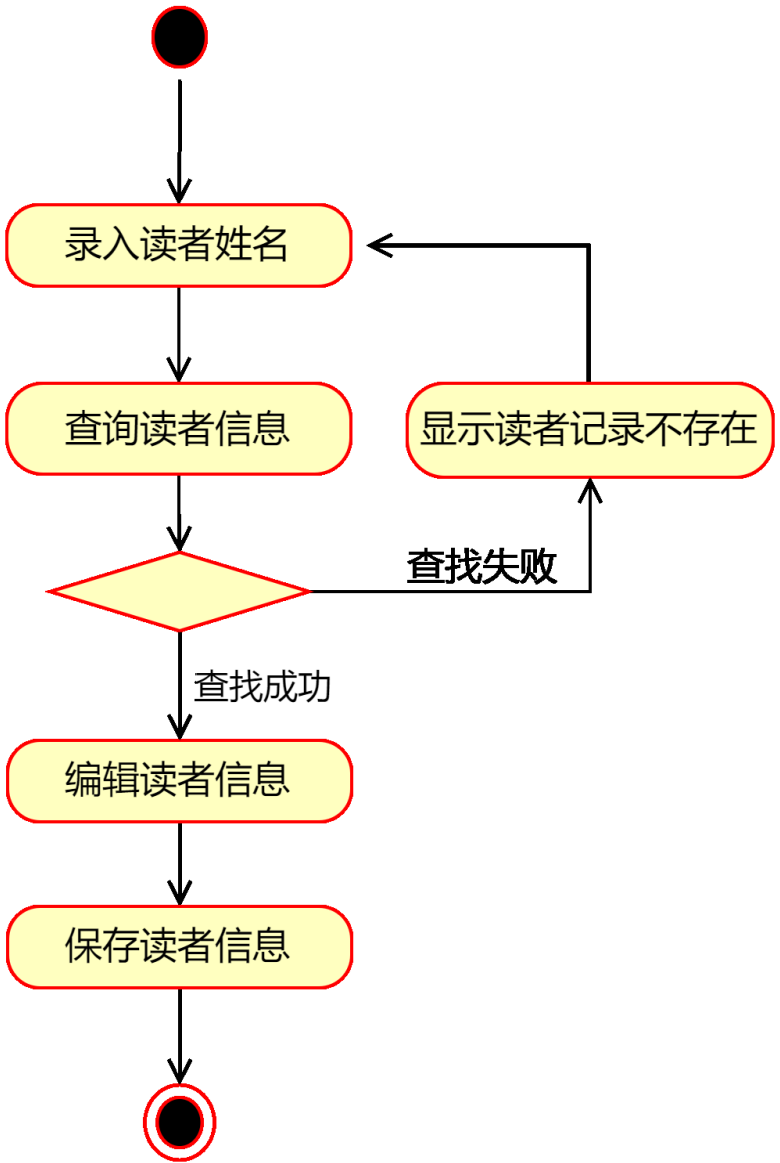
4. 状态机图

- 状态机图用于描述一个实体基于事件反应的动态行为，显示了该实体如何根据当前所处的状态对不同的事件做出反应的。



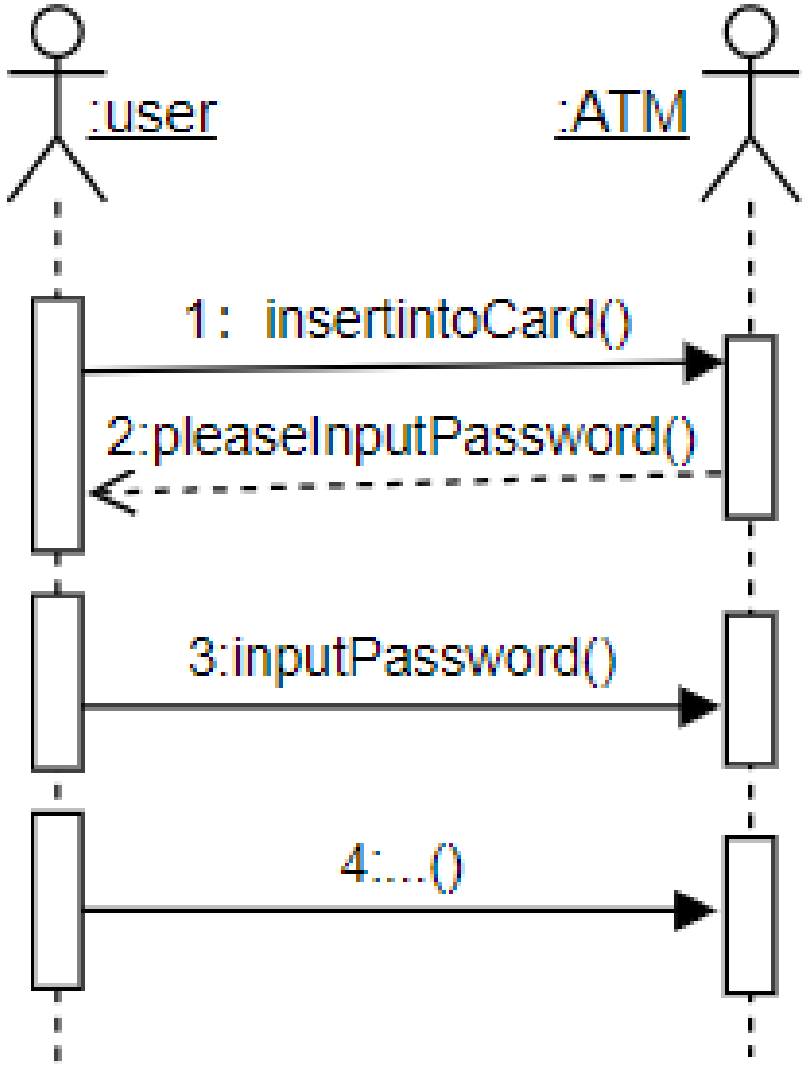
5. 活动图

- 在面向对象中活动图记录了单个操作、方法的逻辑，或者单个业务流程的逻辑，描述系统中各种活动的执行顺序，通常用于描述一个操作中所要进行的各项活动的执行流程。同时，它也常被用来描述一个用例的处理流程，或者某种交互流程。
- 活动图由一些活动组成，图中同时包括了对这些活动的说明。当一个活动执行完毕之后，将沿着控制转移箭头转向下一个活动。活动图中还可以方便地描述控制转移的条件以及并行执行等要求。
- 活动图是比较常用的一种图，接近于流程图。在UML 2.0中，活动图增加了许多新特性。例如泳道可以划分成层次，增加丰富的同步表达能力，在活动图中引入对象等特性。


录入读者姓名
查询读者信息
显示读者记录不存在
查找失败
查找成功
编辑读者信息
保存读者信息

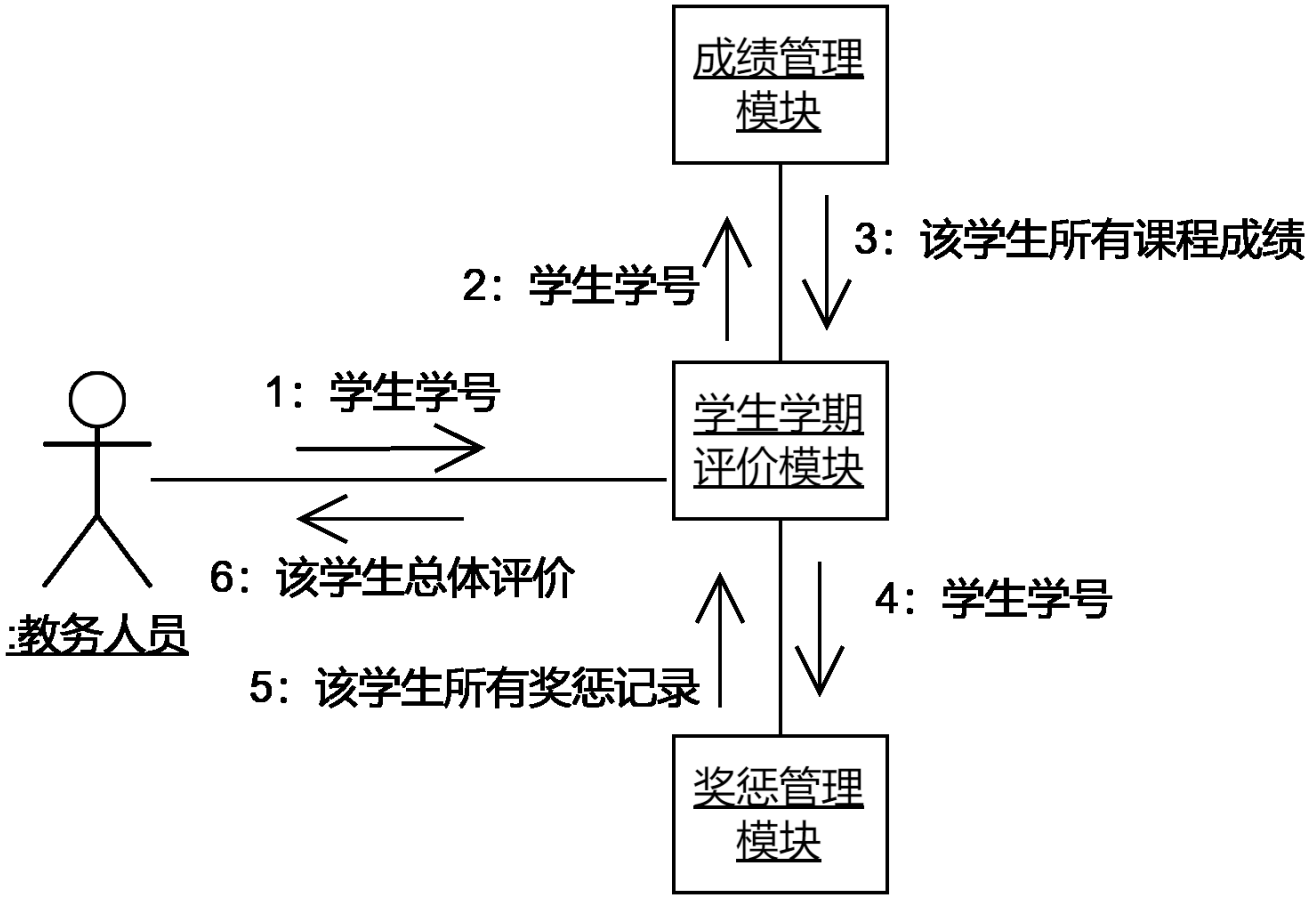
6. 顺序图

- 顺序图描述了对象之间动态的交互关系，主要体现对象之间进行消息传递的时间顺序。顺序图由一组对象构成，每个对象分别带有一条竖线，称作对象的生命线，它代表时间轴，时间沿竖线向下延伸，顺序图描述了这些对象随着时间的推移相互之间交换消息的过程。消息利用从一个对象的生命线指向另一个对象的生命线的水平箭头表示。图中还可以根据需要增加有关时间的说明和其他注释。


:user
:ATM
1: insertintoCard()
2:pleaseInputPassword()
3:inputPassword()
4:...()

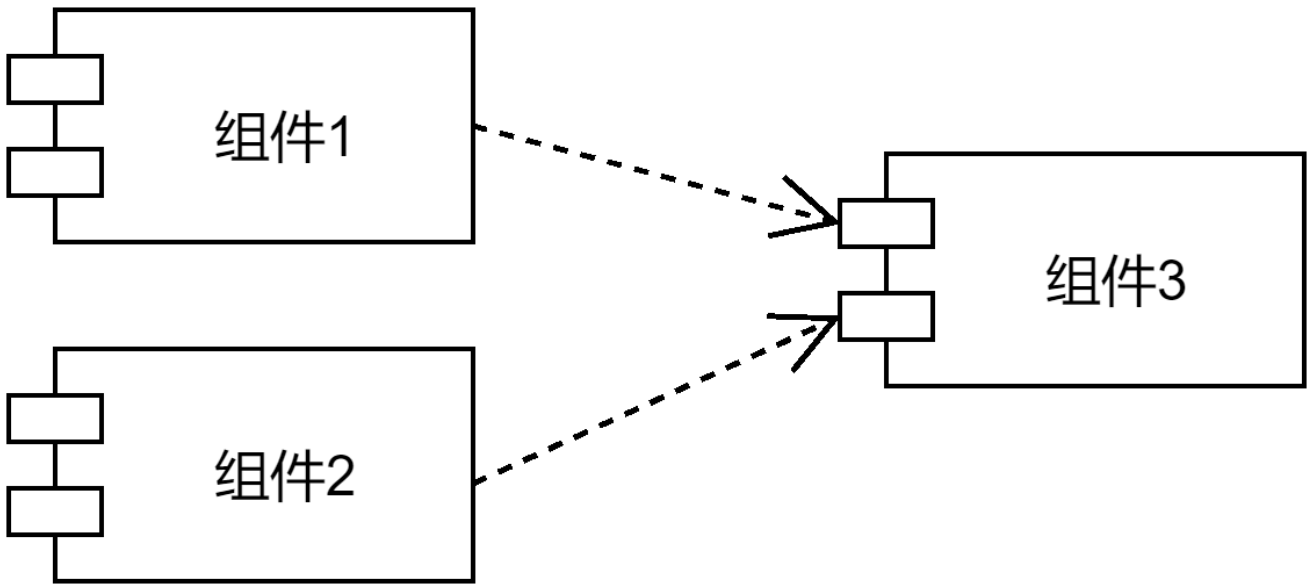
7. 通信图

- 在面向对象中通信图用于显示组件及其交互关系的空间组织结构，它并不侧重于交互的顺序。通信图显示了交互中各个对象之间的组织交互关系以及对象彼此之间的链接。与顺序图不同，通信图显示的是对象之间的关系，而且通信图没有将时间作为一个单独的维度，因此序列号就决定了消息及并发线程的顺序。它用带有编号的箭头来描述特定的方案，以显示在整个方案过程中消息的移动情况。通信图主要用于描绘对象之间消息的移动情况来反映具体的方案，显示对象及其交互关系的空间组织结构，而非交互的顺序。


成绩管理模块
3：该学生所有课程成绩
2：学生学号
1：学生学号
学生学期评价模块
6：该学生总体评价
:教务人员
4：学生学号
5：该学生所有奖惩记录
奖惩管理模块

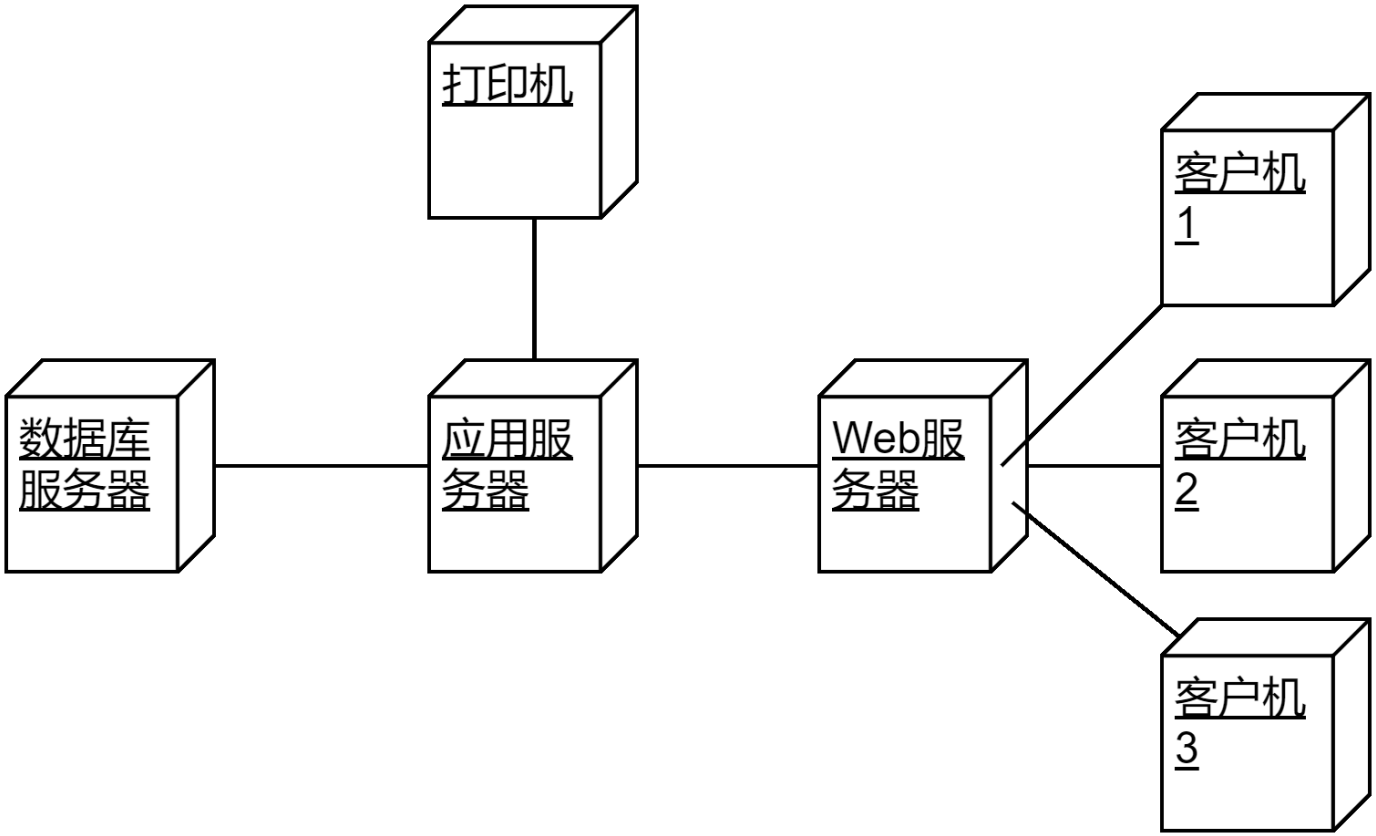
8. 构件图

- 构件图，也称为组件图。构件图描述代码部件的物理结构及各部件之间的依赖关系，构件图有助于分析和理解部件之间的相互影响程度。从构件图中，可以了解各软件组件（如源代码文件或动态链接库）之间的编译器和运行时依赖关系。使用构件图可以将系统划分为内聚组件并显示代码自身的结构。


组件1
组件2
组件3

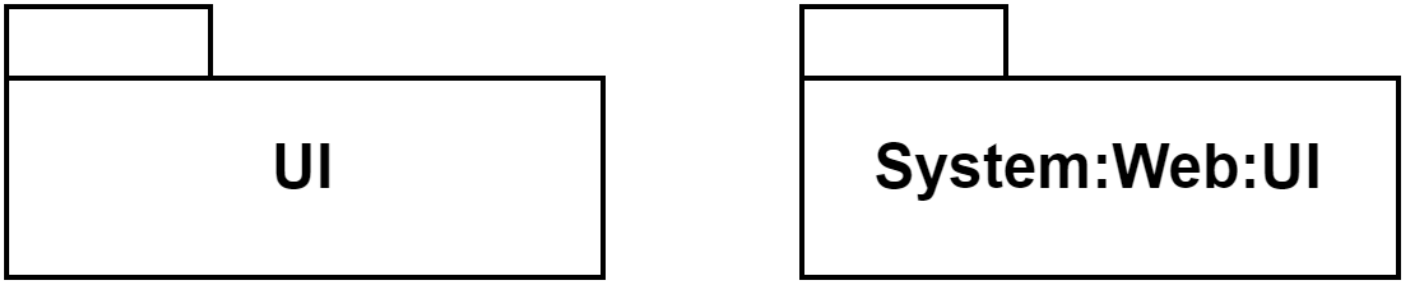
9. 部署图

- 部署图，也称为配置图，用于描述系统中硬件和软件的物理配置情况和系统体系结构。在部署图中，用结点表示实际的物理设备，如计算机和各种外部设备等，并根据它们之间的连接关系，将相应的结点连接起来，并说明其连接方式。在结点里面，说明分配给该结点上运行的可执行构件或对象，从而说明哪些软件单元被分配在哪些结点上运行。


打印机
客户机1
数据库服务器
应用服务器
Web服务器
客户机2
客户机3

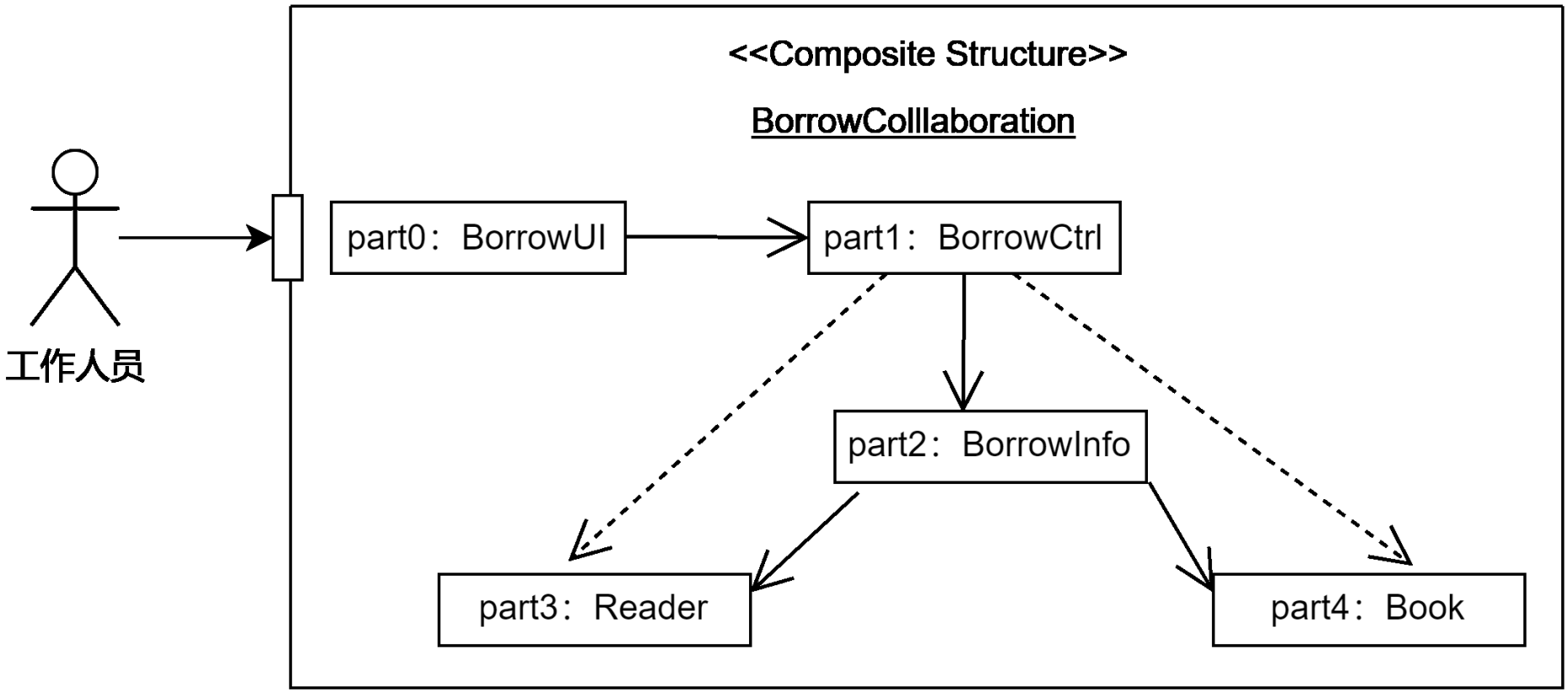
10.包图

- “包图”展现模型要素的基本组织单元，以及这些组织单元之间的依赖关系，包括引用关系（PackageImport）和扩展关系（PackageMerge）。在通用的建模工具中，一般可以用类图描述包图中的逻辑内容。



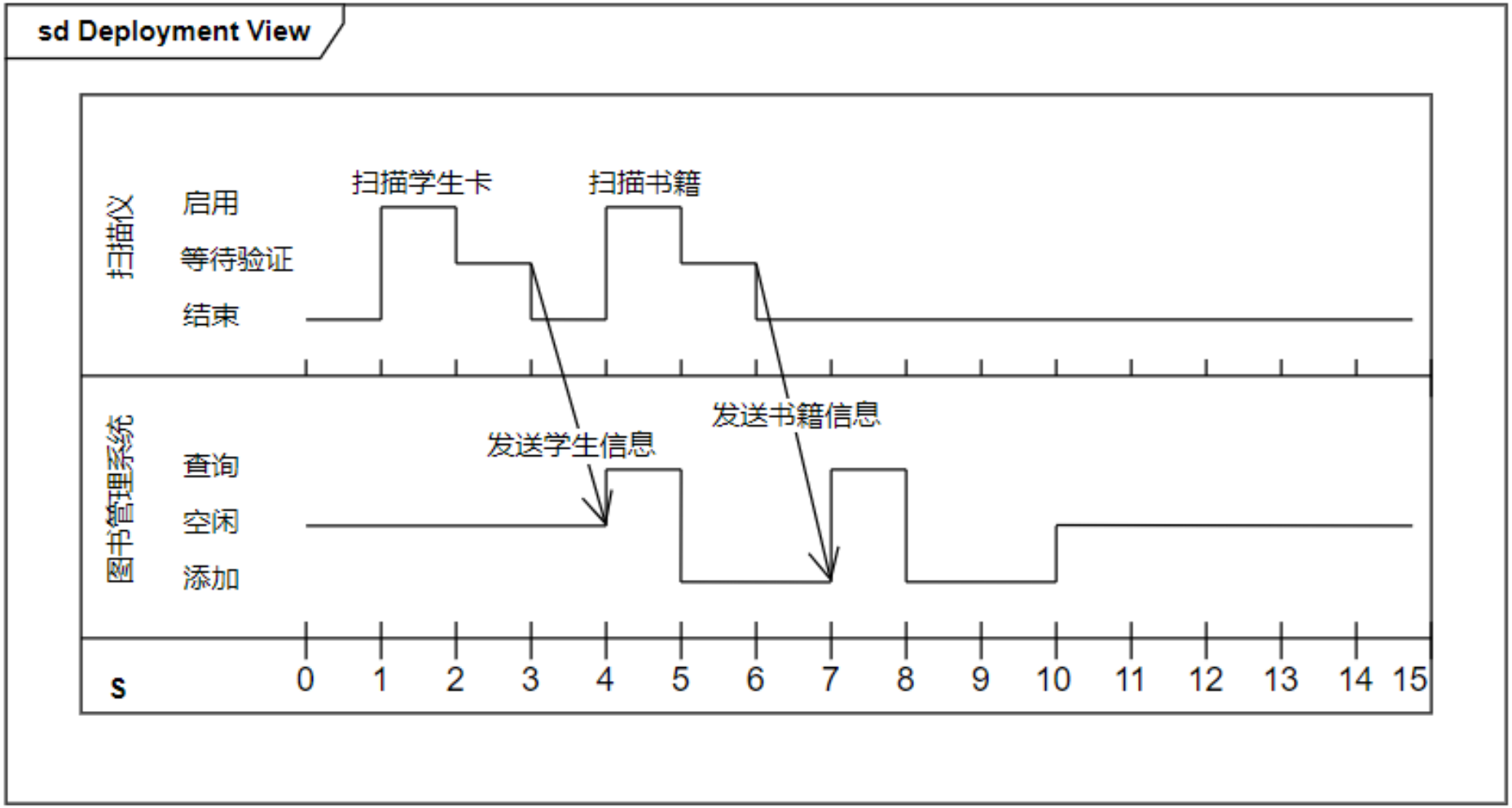
11. 组合结构图

- “组合结构图”用于描述系统中的某一部分（即“组合结构”）的内部内容，包括该部分与系统其他部分的交互点，这种图能够展示该部分内容“内部”参与者的配置情况。“组合结构图”中引入了一些重要的概念。例如，“端口”（Port），“端口”将组合结构与外部环境隔离，实现了双向的封装，既涵盖了该组合结构所提供的行为（PrrovidedInterface），同时也指出了该组合结构所需要的服务（RequiredInterface）；如“协议”（protocol），基于UML中的“协作”（collaboration）的概念，展示那些可复用的交互序列，其实质目的是描述那些可以在不同上下文环境中复用的协作模式。“协议”中所反映的任务由具体的“端口”承担。


<<Composite Structure>>
BorrowColllaboration
工作人员
part0：BorrowUI
part1：BorrowCtrl
part2：BorrowInfo
part3：Reader
part4：Book

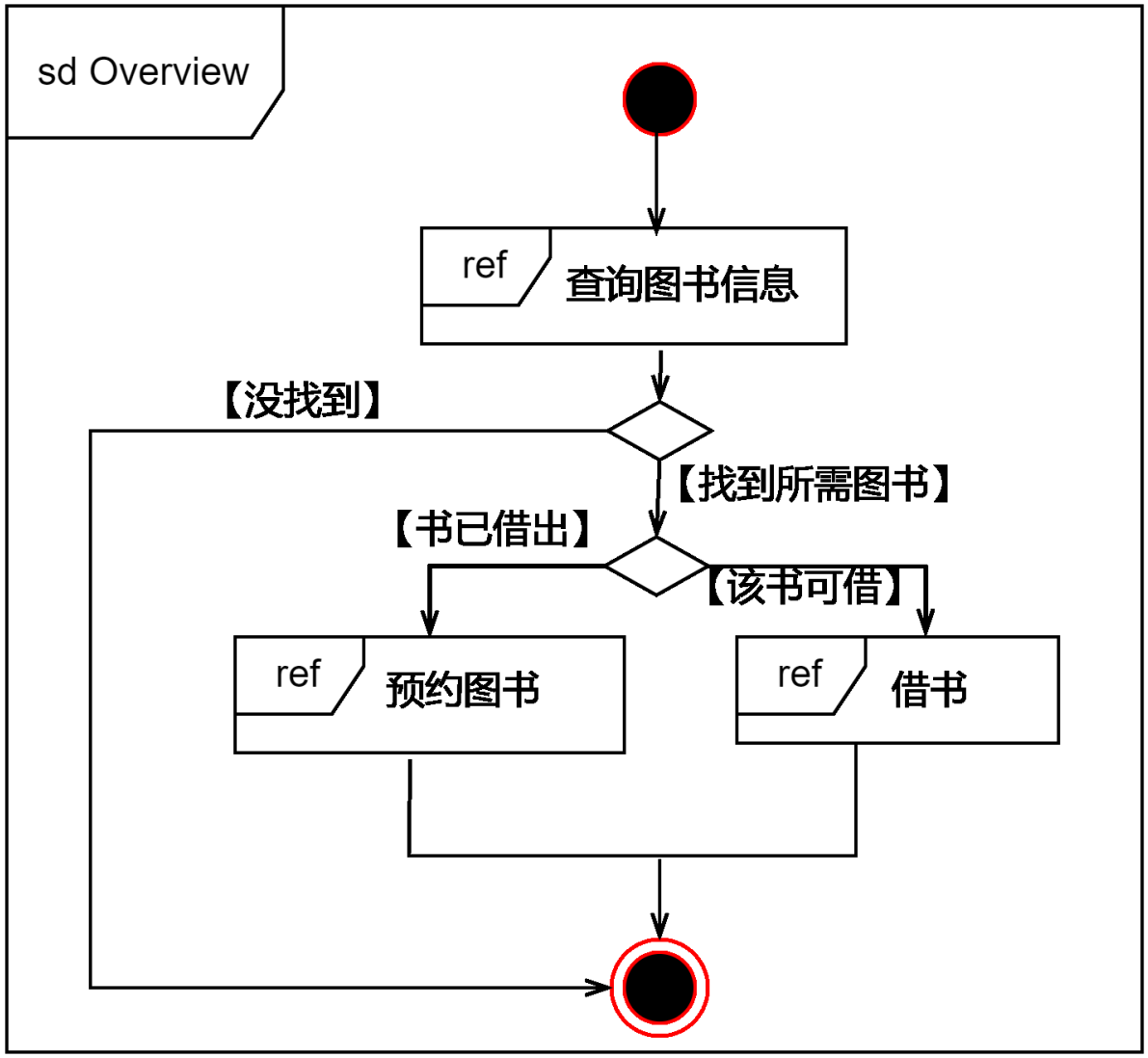
12. 定时图

- 定时图是一种可选的交互图，展示交互过程中的真实时间信息，具体描述对象状态变化的时间点以及维持特定状态的时间段。定时图是UML2.0新增的建模图，实质上，定时图是一种特殊的顺序图。对于一些实时性较强的系统功能进行建模时，则可用定时图进行建模。定时图是一个二维图，在纵轴方向，处在不同位置的水平线，表示对象处在不同的状态；横轴用来表示时间，时间由左向右延伸。定时图包含的基本元素有：对象、状态、表示状态的水平线、表示状态迁移的垂直线、表示时间的横轴和时间刻度。


sd Deployment View
扫描仪
启用
等待验证
结束
扫描学生卡
扫描书籍
图书管理系统
查询
空闲
添加
发送学生信息
发送书籍信息
S
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

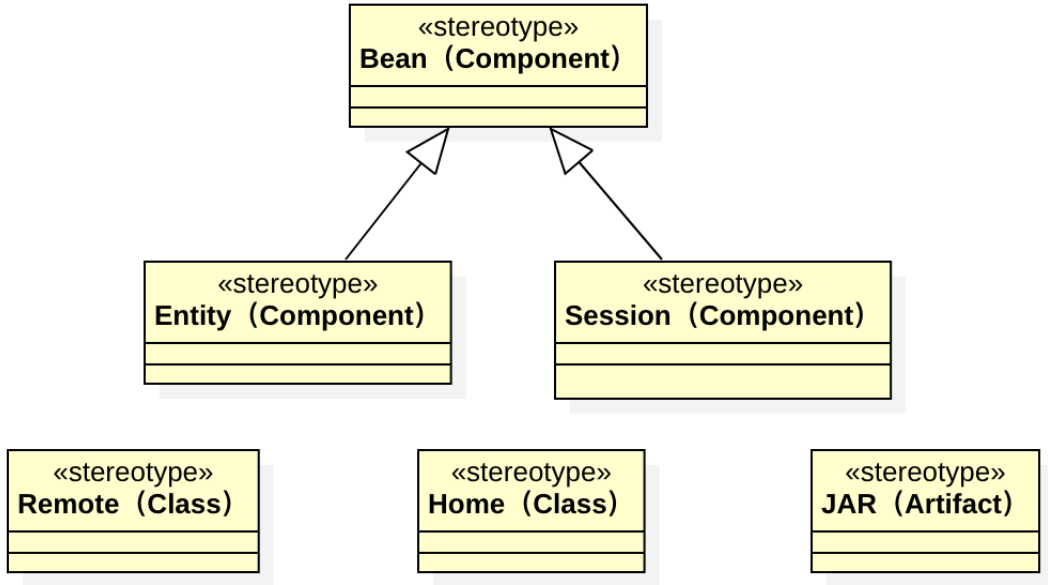
13. 交互概览图

- “交互概览图”（InteractionOverviewDiagram）与活动图类似，只是将活动图中的动作元素改为交互概览图的交互关系。如果概览图内的一个交互涉及时序，则使用顺序图；如果概览图中的另一个交互需要关注消息次序，则可以使用顺序图。交互概览图将系统内单独的交互结合起来，并针对每个特定交互使用最合理的表示法，以显示出它们是如何协同工作来实现系统的主要功能。顺序图、通信图和定时图主要关注特定交互的具体细节，而交互概览图则将各种不同的交互结合在一起，形成针对系统某种特定要点的交互整体图。


sd Overview
ref
查询图书信息
【没找到】
【找到所需图书】
【书已借出】
【该书可借】
ref
预约图书
ref
借书

14. 概要图

- 概要图，又称轮廓图，或者剖面图，是一种结构图，它通过定义自定义原型、标记值和约束类描述UML的轻量级扩展机制，用来辅助其他UML图。
- 概要图中主要包含的元素有概要、元素、构造型等，元素之间的关系有引用、应用、还有UML中常用的泛化等关系。



UML图分类

- 上述14种图可归纳为5类。

类型	包含
静态图	类图、对象图、包图、组合结构图、概要图
行为图	状态机图、活动图
用例图	用例图
交互图	顺序图、通信图、定时图、交互概览图
实现图	组件图、部署图

- 从应用的角度看，当采用面向对象技术设计系统时，首先是描述需求；其次根据需求建立系统的静态模型，以构造系统的结构；第三步是描述系统的行为。
- 其中在第一步与第二步中所建立的模型都是静态的，包括用例图、类图、包图、对象图、组合结构图、构件图和部署图等7个图形，是标准建模语言UML的静态建模机制。其中第三步中所建立的模型或者可以执行，或者表示执行时的时序状态或交互关系。它包括状态机图、活动图、顺序图、通信图、定时图和交互概览图等6个图形，是标准建模语言UML的动态建模机制。因此，标准建模语言UML的主要内容也可以归纳为静态建模机制和动态建模机制两大类。

UML的语言规则

在UML中，基本元素在使用时，应该遵守一系列规则，其中，最常用的3种语义规则如下：

（1）命名：也就是为事物、关系和图起名字。和任何语言一样，名字都是一个标识符。

（2）范围：指基本元素起作用的范围，相当于程序设计语言中的变量的“作用域”。

（3）可见性：UML元素可能属于一个类或包中，因此，所有元素都具有可见性这一属性。在UML中共定义了4种可见性。在面向对象的编程语言C++、Java和C#中，也常常讨论这些可见性。

可见性	规则	标准表示法
public	任一元素，若能访问包容器，就可以访问它	+
protected	只有包容器中的元素或包容器的后代才能够看到它	#
private	只有包容器中的元素才能够看得到它	-
package	只有声明在同一个包中的元素才能够看到该元素	~

UML的公共机制

- 在UML中，共有4种贯穿于整个统一建模语言并且一致应用的公共机制，这4种公共机制分别是规格说明、修饰、通用划分和扩展机制。通常会把规格说明、修饰和通用划分看作UML的通用机制。其中扩展机制可以再划分为构造型、标记值和约束。

UML的公共机制

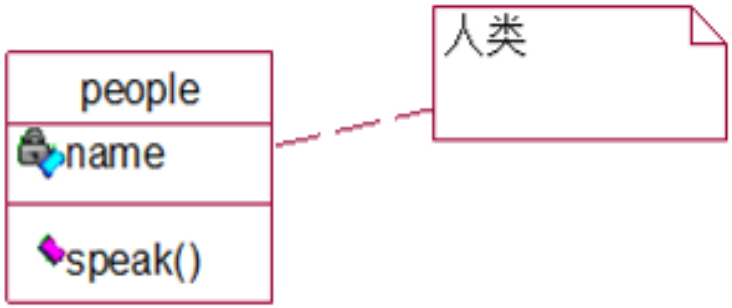
UML的通用机制

- 规格说明

模型元素作为一个对象本身也具有很多的属性，这些属性用来维护属于该模型元素的数据值。属性是使用名称和标记值的值来定义的。标记值指的是一种特定的类型，可以是布尔型、整型或字符型，也可以是某个类或接口的类型。

- 修饰

在UML的图形表示中，每一个模型元素都有一个基本符号，这个基本符号可视化地表达了模型元素最重要的信息。用户也可以把各种修饰细节加到这个符号上以扩展其含义。这种添加修饰细节的做法可以为图中的模型元素在一些视觉效果上发生一些变化。



注释实例

有数目关系的修饰实例

- 通用划分

通用划分是一种保证不同抽象概念层次的机制。一般采用两种方式进行通用划分：

（1）对类和对象的划分。指类是一个抽象而对象是这种抽象的一个实例化。

（2）对接口和实现的分离。接口和实现的分离是指接口声明了一个操作接口，但是却不实现其内容，而实现则表示了对该操作接口的具体实现，它负责如实地实现接口的完整语义。

UML的公共机制——扩展机制

- 虽然UML已经是一种功能较强、表现力非常丰富的建模语言，但是有时仍然难以在许多细节方面对模型进行准确的表达。所以，UML设计了一种简单的、通用的扩展机制，用户可以使用扩展机制对UML进行扩展和调整，以便使其与一个特定的方法、组织或用户相一致。扩展机制是对已有的UML语义按不同系统的特点合理地进行扩展的一种机制。
- 三种扩展机制：构造型、标记值和约束，使用这些扩展机制能够让UML满足各种开发领域的特别需要。其中，构造型扩充了UML的词汇表，允许针对不同的问题，从已有的基础上创建新的模型元素。标记值扩充了UML的模型元素的属性，允许在模型元素的规格中创建新的信息。约束扩充了UML模型元素的语义，允许添加新的限制条件或修改已有的限制条件。

- 构造型

（1）第一种表示法：创建一种新的UML元素符号的方法是，用符号“《》”把构造名字括起来，这是一种标准表示方法。如，《exception》就是新构造的元素。

（2）第二种表示方法：用符号“《》”把构造名字括起来，并为元素增加一个图标。

（3）第三种表示方法：直接用一个图标表示新的构造元素。

- 标记值

标记值是用来为元素添加新特征的。标记值的表示方法是用形如“{标记信息}”的字符串表示。标记信息通常由名称、分隔符和值组成。标记值是对元素属性的表示，因此，标记值放在UML元素中的，如，name=“邓小平”。

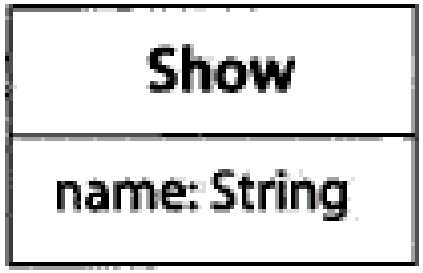
- 约束

约束机制用于扩展UML构造块的语义，允许建模者和设计人员增加新的规则和修改现有的规则。约束可以在UML工具中预定义，也可以在某个特定需要的时候再进行添加。约束可以表示在UML的规范表示中不能表示的语义关系。在定义约束信息的时候，应尽可能准确地去定义这些约束信息。

约束使用大括号和大括号内的字符串表达式表示，即约束的表现形式为｛约束的内容｝。约束可以附加在表元素、依赖关系或注释上。

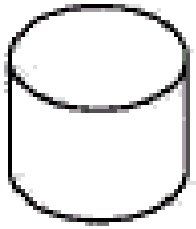
扩展组件

- UML包含三种主要的扩展组件
  - 约束是用某种形式化语言或自然语言表达的语义关系的文字说明。
  - 构造型是由建模者设计的新的模型元素，但是这个模型元素的设计要建立在UML已定义的模型元素基础上。
  - 标记值是附加到任何模型元素上的命名的信息块
- 这些组件提供了扩展UML模型元素语义的方法，同时不改变UML定义的元模型自身的语义。



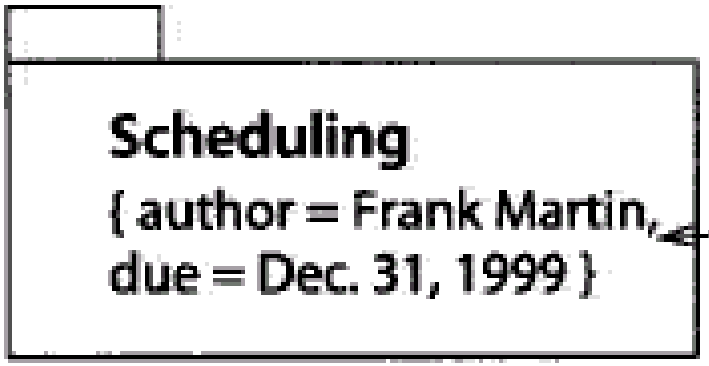
{names for one season must be unique} ← 约束

构造型



TicketDB

构造型图标



标记值

总结

- UML是一种建模语言，具有广泛的应用领域，它不仅可以应用于软件领域的建模，也可以用于非软件领域的建模
- UML不是一种开发方法，它是独立于任何软件开发方法之外的语言。利用它建模时，可遵循任何类型的建模过程
- UML特别适用于面向对象分析和设计的软件开发方法的建模
- 软件工程的研究和实践证明，在提高软件工程的质量，降低软件开发的风险，处理复杂的功能需求，建立有效的开发平台等诸多软件开发中的关键问题方面，UML建模是最有效的方法