第一章 软件建模与UML

主讲教师：徐丙凤

第一章 导言

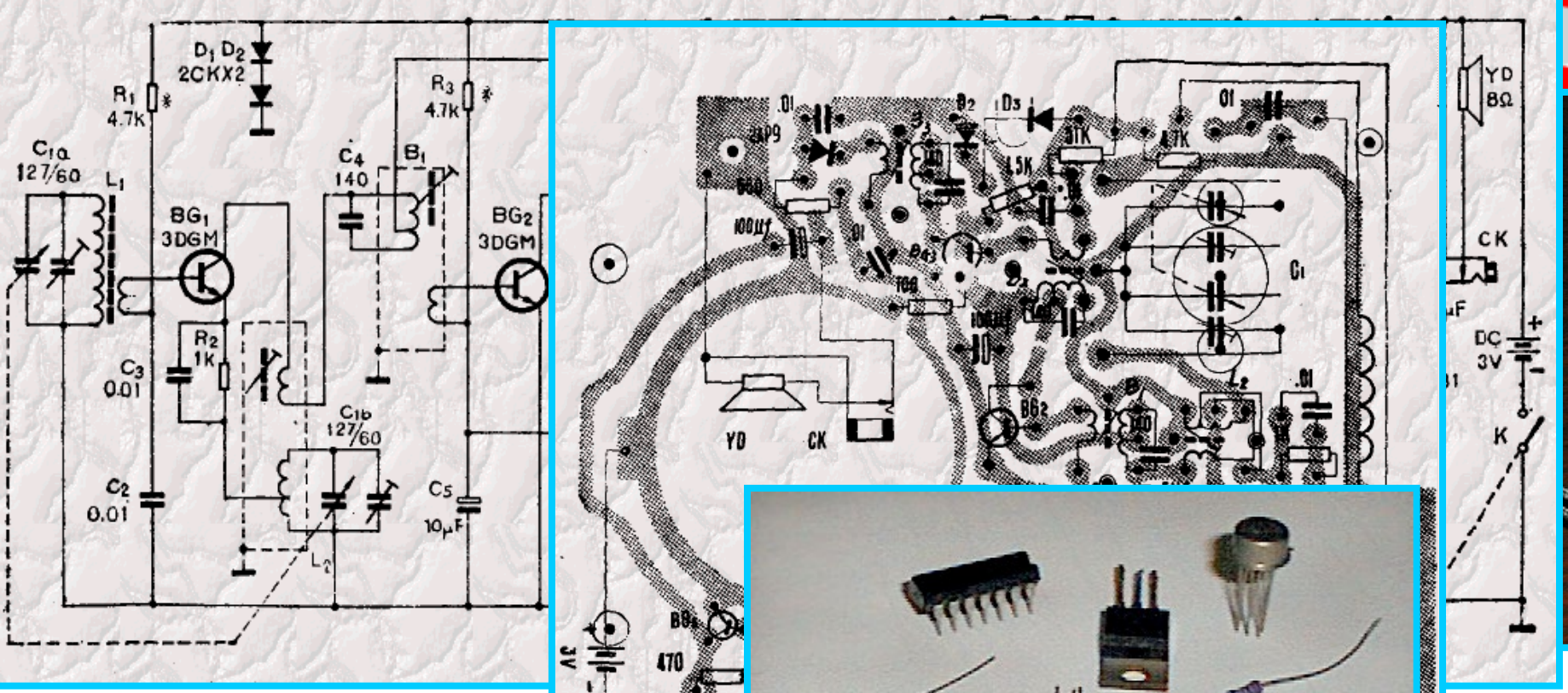
- 1.1 开发软件为什么需要模型
- 1.2 面向对象模型
- 1.3 统一建模语言UML
- 1.4 UML建模工具
- 1.5 总结

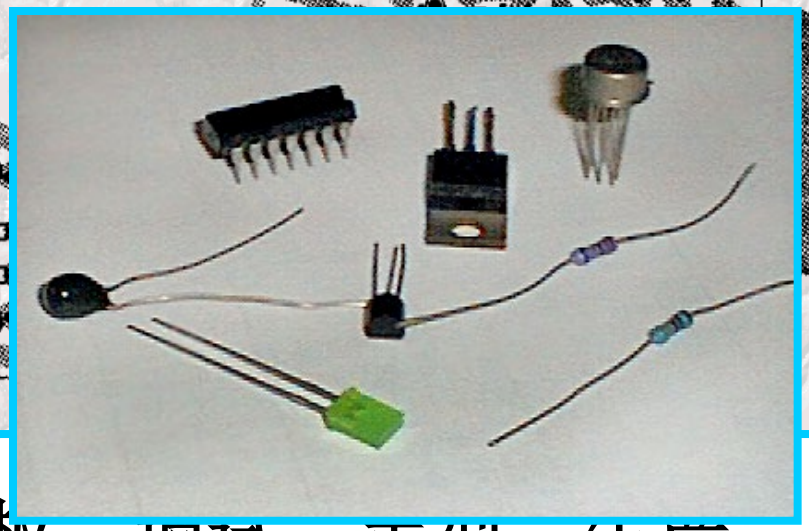
1.1 模型

- 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 UML建模工具
- 1.5 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。

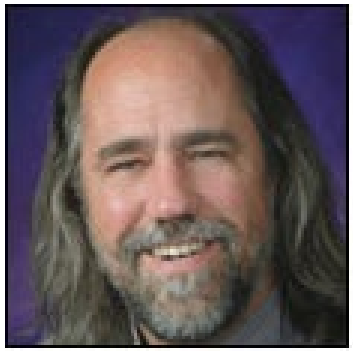
（6）围绕对象和类进行局部化，提高了规约、设计和代码的易扩展性、易维护性和易复用性。

（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

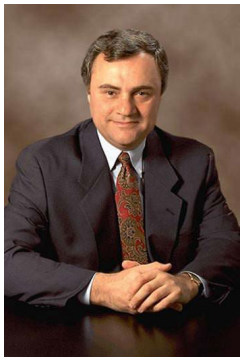
面向对象建模语言问世于20世纪70年代中期。

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 UML建模工具
- 1.5 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

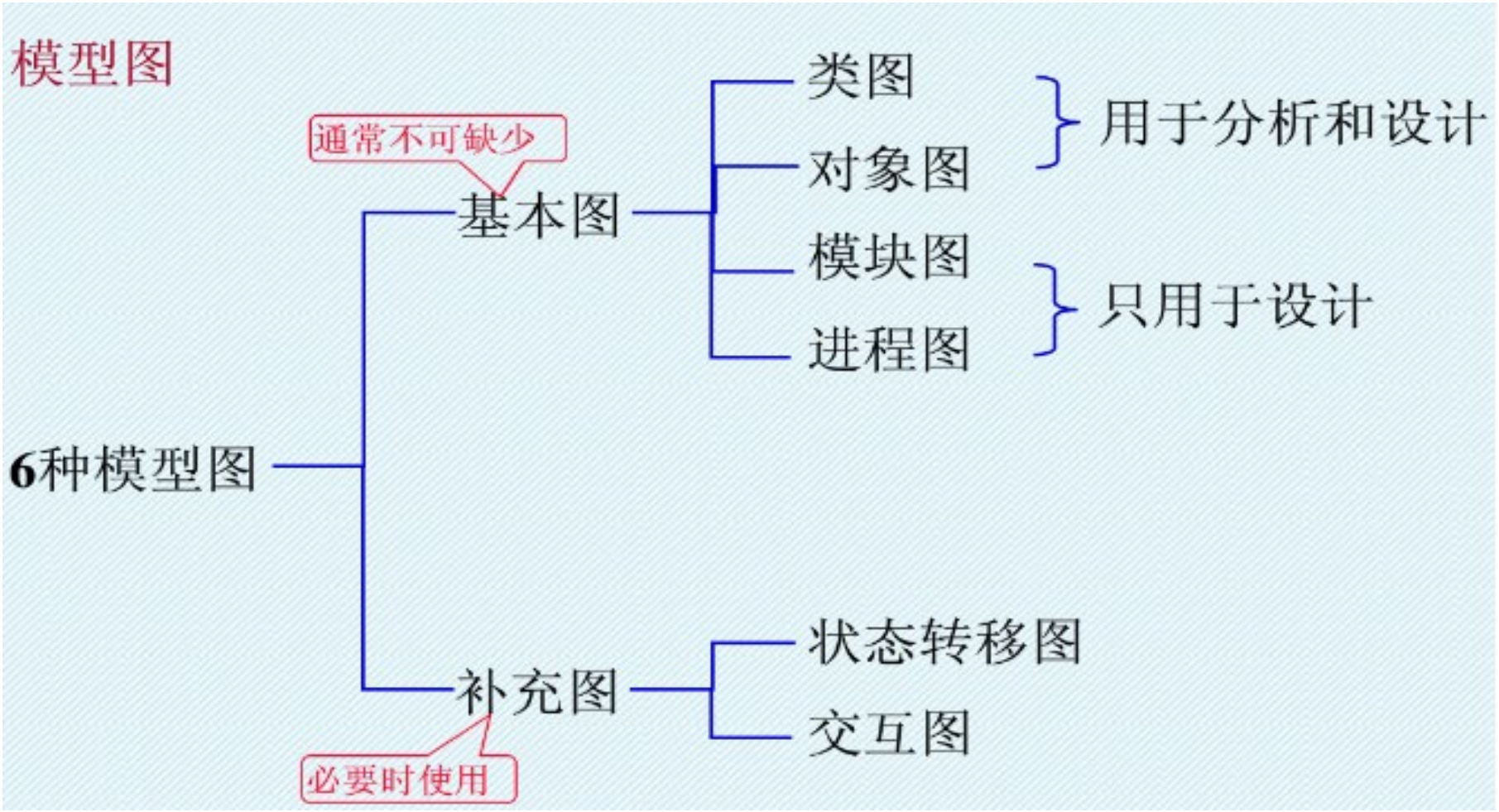
1.4 UML的发展史

- 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**

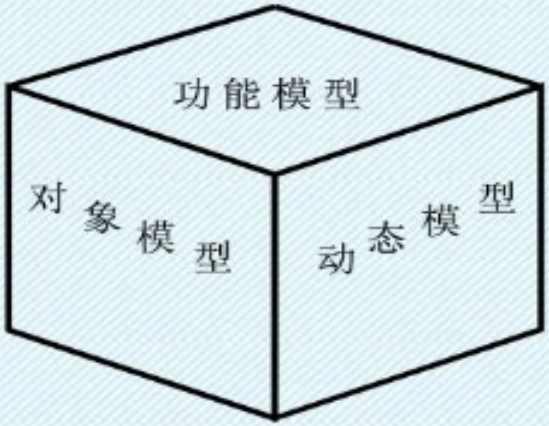

模型图
6种模型图
基本图
通常不可缺少
类图
对象图
模块图
进程图
用于分析和设计
只用于设计
补充图
必要时使用
状态转移图
交互图

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

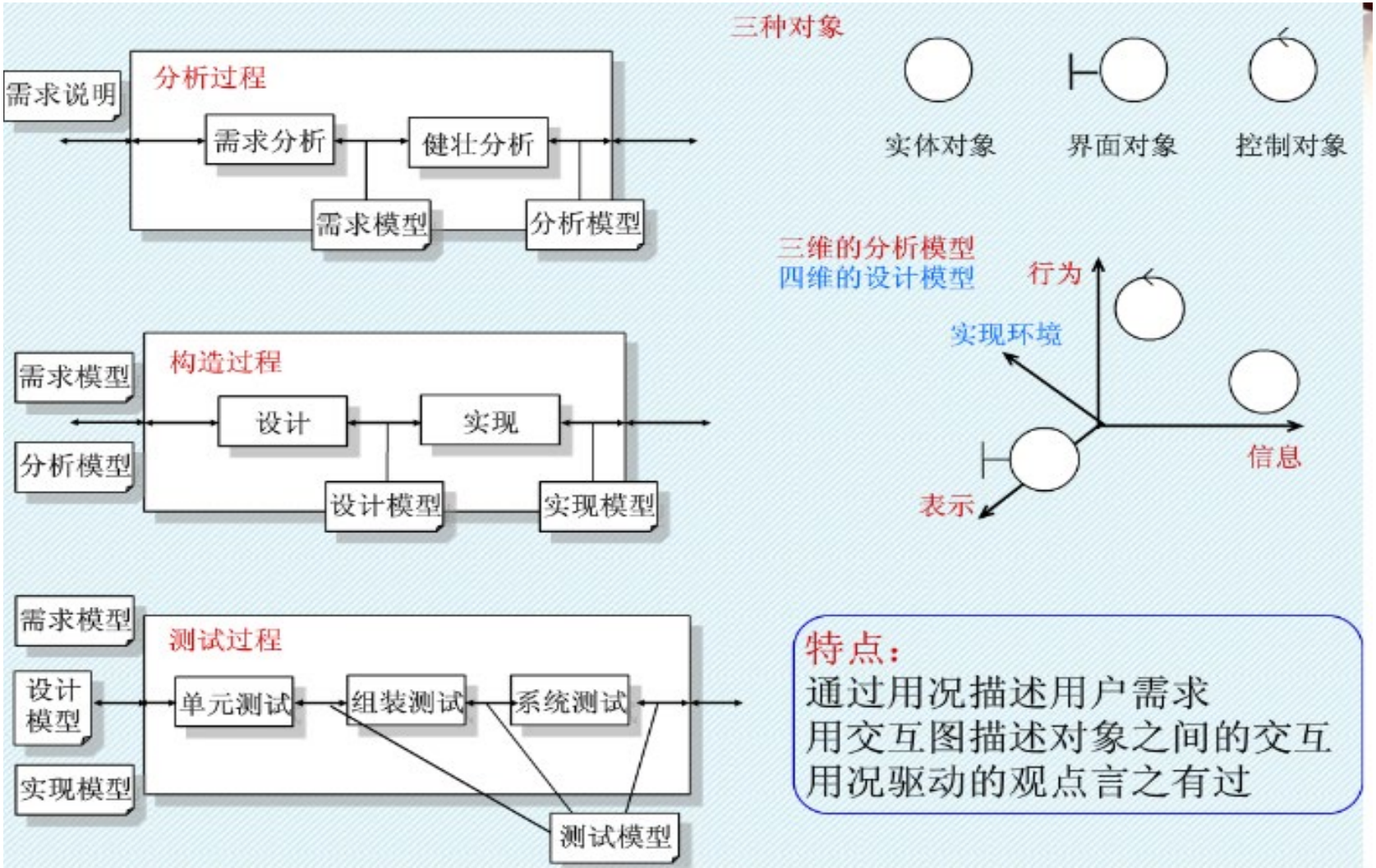
特点：

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

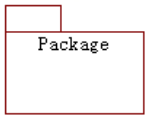
- **OOSE方法**


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

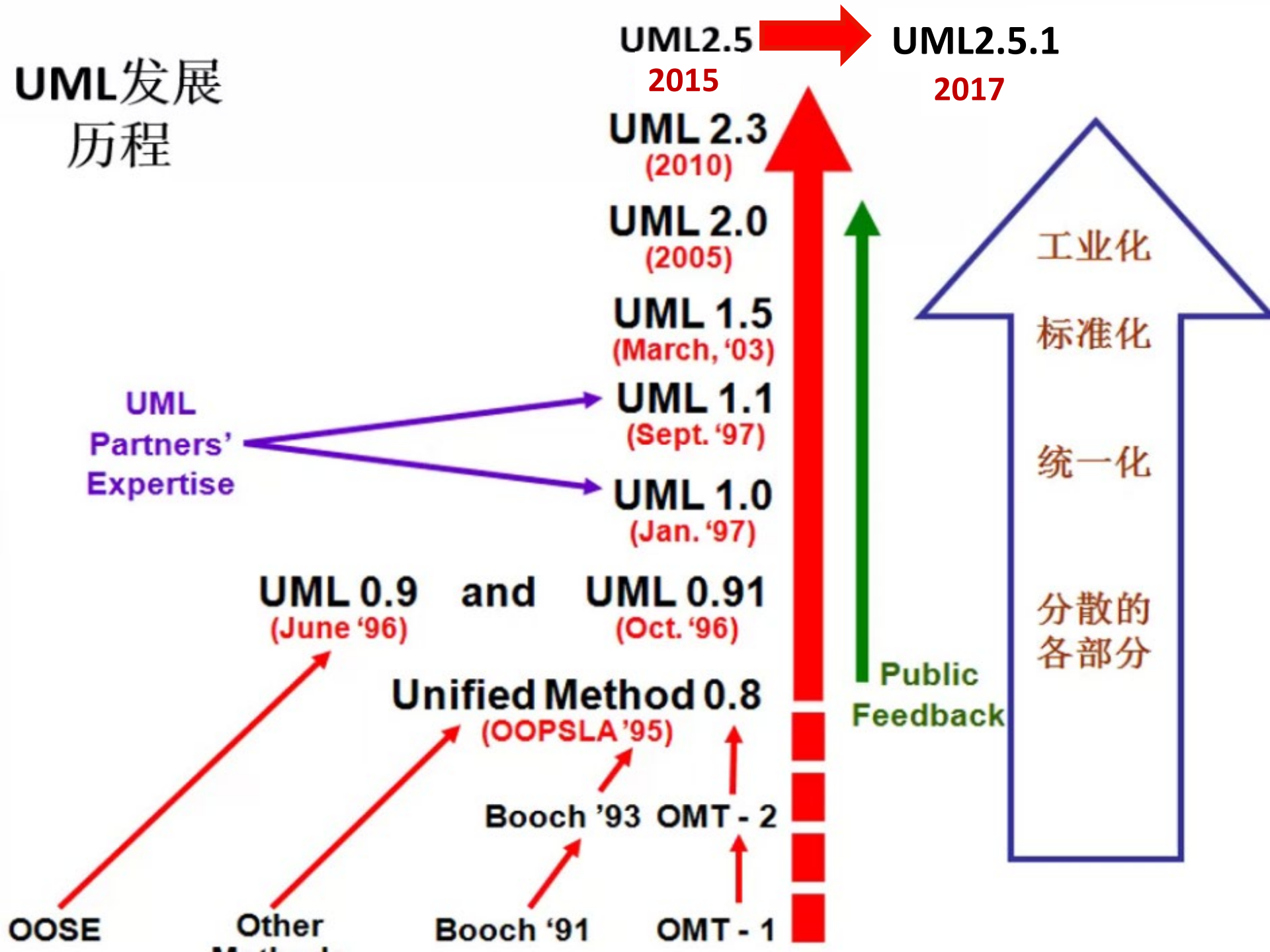
- UML对各个流派的建模符号和概念进行了归纳，并提供了非常严谨的定义和描述，成为真正的建模语言。
- Booch、Jacobson、Rumbaugh三人的表示法在UML中占了主要部分。
- 但还是有很多部分来自各方
  - 接口来自Microsoft
  - 

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

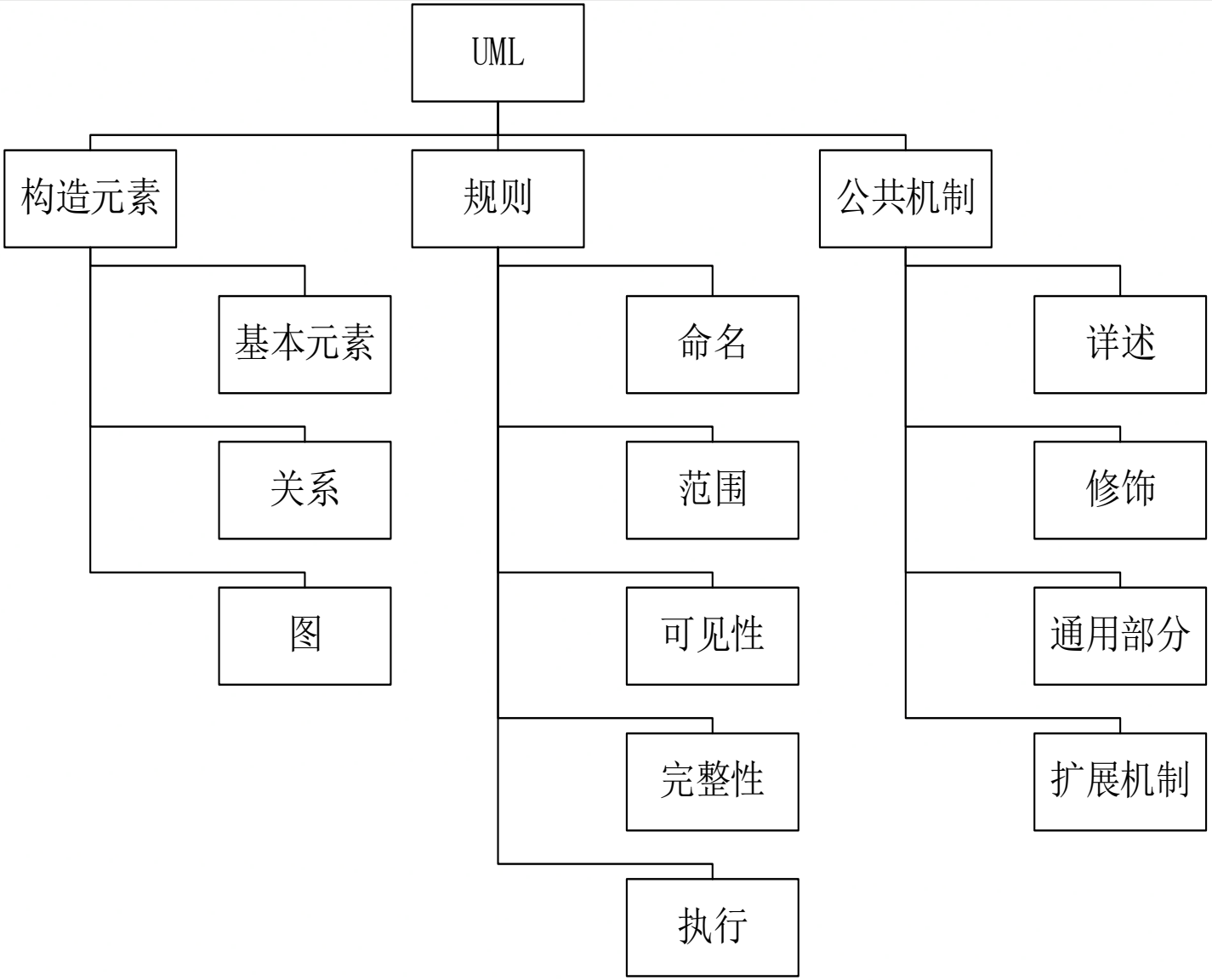
UML语言是一门设计语言，这种语言由一些构造元素、规则和公共机制构成。

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）构造元素包括基本元素、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
基本元素
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语言的基本元素

- UML定义了4种基本的面向对象的元素
  - 结构元素
  - 行为元素
  - 分组元素
  - 注释元素

- 结构元素是UML模型中的名词部分，定义了业务或软件系统中的某个物理元素，负责描述事物的静态特征，往往构成模型的静态部分。在UML规范中，一共定义了七种结构元素，分别是类和对象、接口、协作、用例、主动类、构件和节点。

1.类和对象

类的表示方法

对象的表示方法

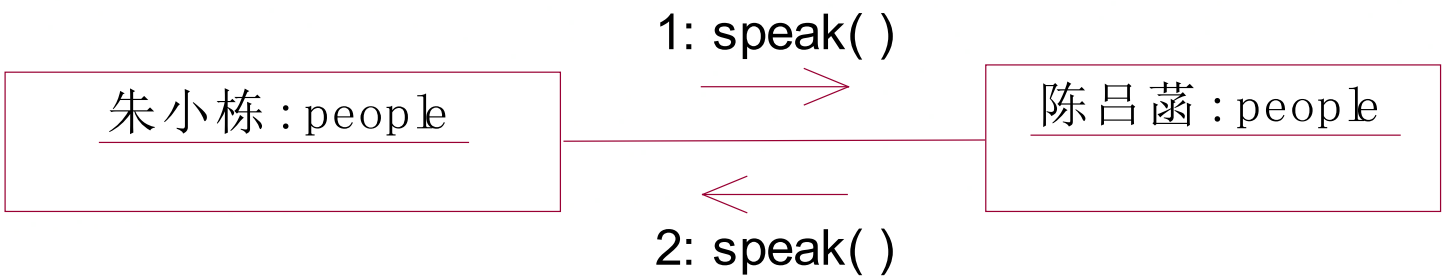
2.接口

接口由一组对操作的定义组成。但是，接口对操作的具体实现不做描述。接口用于描述一个类或构件的一种服务的操作集。它描述了元素的外部可见操作。一个接口可以描述一个类或构件的全部行为或部分行为。接口很少单独存在，往往依赖于实现接口的类或构件。

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。

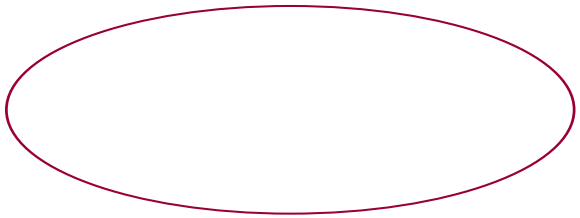
3.协作

协作用于对一个交互过程的定义，它是由一组共同工作以提供协作行为的角色和其他元素构成的一个整体。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。从本质上说，协作就是用例的实现。



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

5.主动类

主动类的对象，也称主动对象，能够自动地启动控制活动，因为主动对象本身至少拥有一个进程或线程，每个主动对象都有它自己的事件驱动控制线程，控制线程与其他主动对象并行执行。被主动对象所调用的对象称为被动对象。它们只在被调用时接受控制，而当它们返回时将控制放弃。被动对象被动地等待其他对象向它发出请求，这些对象所描述的元素的行为与其他元素的行为并发。主动类的可视化表示类似于一般类的表示，特殊的地方在于其外框为粗线。在许多UML的工具中，主动类的表示和一般类的表示并无区别。

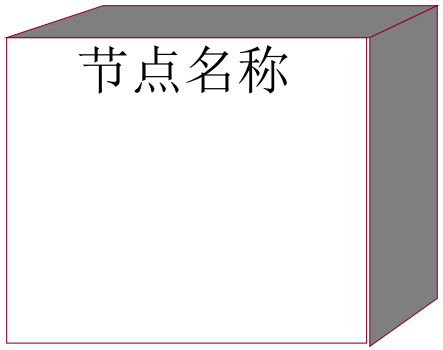
UML语言的基本元素

6.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。

7.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。



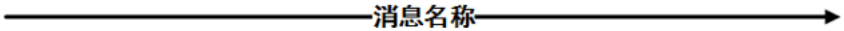
UML语言的基本元素

- **行为元素**

行为元素是指 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。



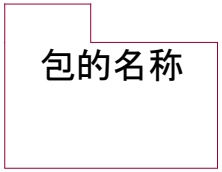
交互的表示方法

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米描述，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。

• 分组元素

分组元素是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，



• 注释元素

注释元素是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。

关系元素

元素关系种类	关系变种	UML表示法	关键字或符号	元素关系种类	关系变种	UML表示法	关键字或符号
抽象	派生	依赖关系	《derive》	导入	私有	依赖关系	《access》
	显现		《manifest》		公有		《import》
	实现	实现关系	虚线加空心三角	信息流			《flow》
	精化	依赖关系	《refine》	包含并			《merge》
	跟踪		《trace》	许可			《permit》
关联		关联关系	实线	协议符合			未指定
绑定		依赖关系	《bind》(参数表)	替换		依赖关系	《substitute》
部署			《deploy》	使用	调用		《call》
扩展	Extend		《extend》（扩展点)		创建		《create》
扩展	extension	扩展关系	实线加实心三角		实例化		《instantiate》
泛化		泛化关系	实线加空间三角		职责		《responsibility》
包含		依赖关系	《include》		发送		《send》

1. 关联关系

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

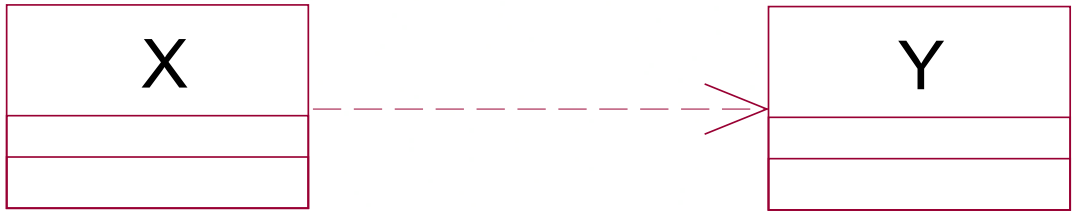
4.泛化关系

- 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

5. 依赖关系

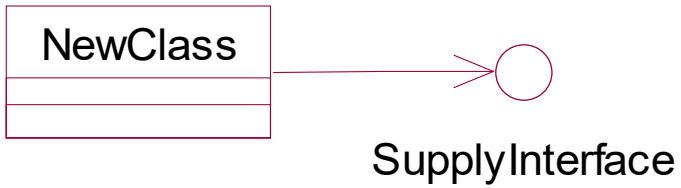
依赖关系指的是两个事物之间的一种语义关系，当其中一个事物（独立事物）发生变化就会影响另外一个事物（依赖事物）的语义。如下图所示，反映了元素X依赖于元素Y。



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。

6. 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。



7. 扩展关系

- 扩展表示把一个构造型附加到一个元类上，使得元类的定义中包括这个构造型。它是一种UML提供的底层的扩展机制，与用例之间的扩展（Extend）关系是不同的。在UML中，用一个带箭头的实线表示，如下图所示。

扩展关系的表示方法

图和视图

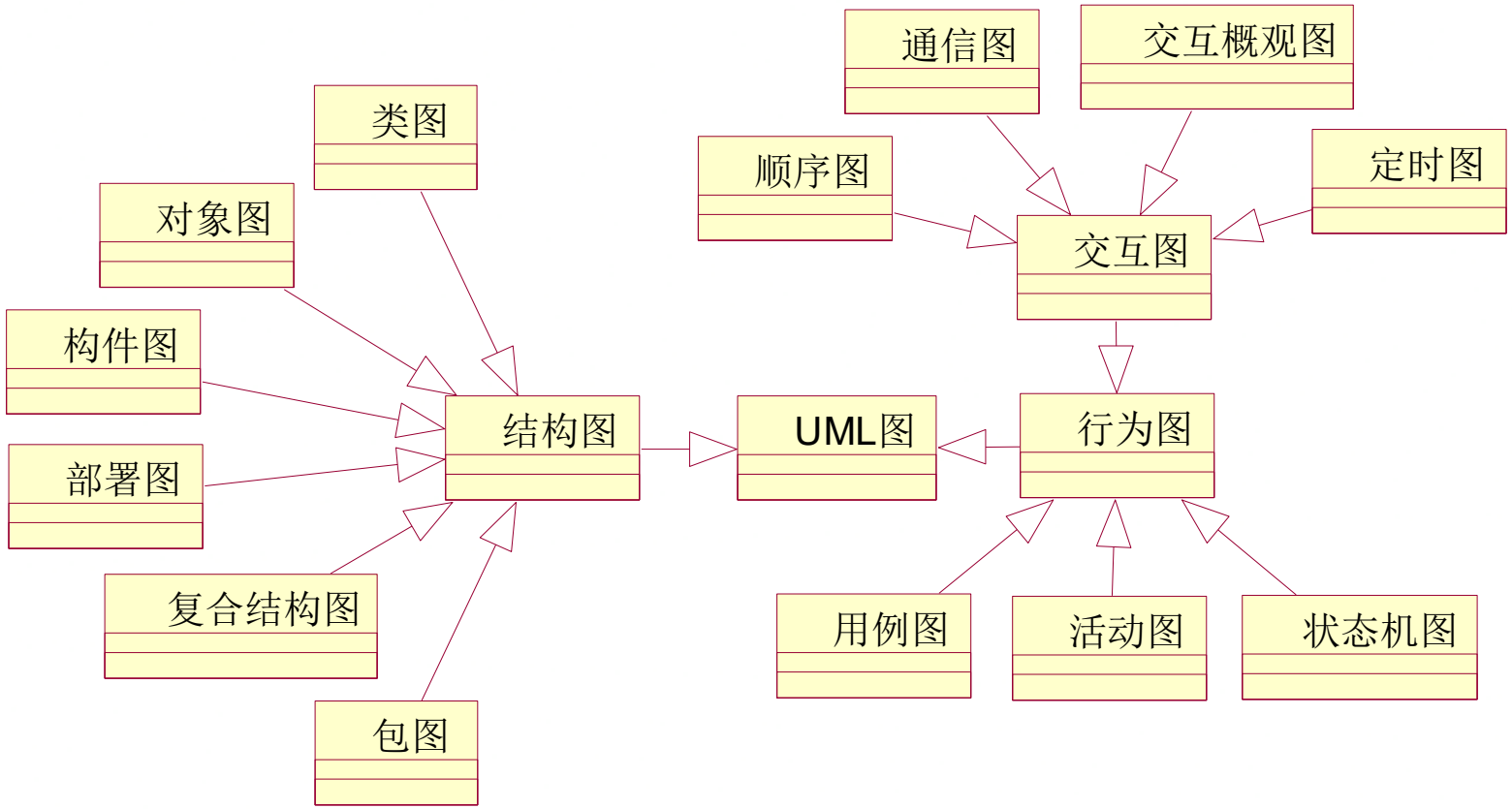
- UML是用模型来描述系统的结构或静态特征以及行为或动态特征的，它从不同的视角为系统的架构建模形成系统的不同视图。视图并不是图，它是表达系统某方面特征的UML建模构件的子集。在每一类视图中使用一种或两种特定的图来可视化地表示视图中的各种概念。也就是说，视图是由一个或多个图组成的对系统某个角度的抽象。

- **在UML 2.0中共定义了13种图，比UML 1.0新增了3种。下表列出了这13种图的功能。**

图名	功能	备注
类图	描述类、类的特性以及类之间的关系	UML 1原有
对象图	描述一个时间点上系统中各个对象的一个快照	UML 1非正式图
复合结构图	描述类的运行时刻的分解	UML 2.0新增
构件图	描述构件的结构与连接	UML 1原有
部署图	描述在各个节点上的部署	UML 1原有
包图	描述编译时的层次结构	UML中非正式图
用例图	描述用户与系统如何交互	UML 1原有
活动图	描述过程行为与并行行为	UML 1原有
状态机图	描述事件如何改变对象生命周期	UML 1原有
顺序图	描述对象之间的交互，重点在强调顺序	UML 1原有
通信图	描述对象之间的交互，重点在于连接	UML 1中的协作图
定时图	描述对象之间的交互，重点在于定时	UML 2.0 新增
交互概观图	是一种顺序图与活动图的混合	UML 2.0新增

图和视图

- 此外，从使用的角度来看，将UML的13种图分为结构模型（也称为静态模型）和行为模型（也称为动态模型）两大类，但这里讲的结构、行为其含义与前面所说的是有一定区别的，前者是从定义角度，后者则是从使用角度。



图和视图

- UML中的各种组件和概念之间没有明显的划分界限，但为方便起见，我们用视图来划分这些概念和组件。视图只是表达系统某一方面特征的UML建模组件的子集。在最上一层，视图被划分成三个视图域：结构分类、动态行为和模型管理。
- 结构分类描述了系统中的结构成员及其相互关系。类元包括类、用例、构件和节点。类元为研究系统动态行为奠定了基础。类元视图包括静态视图、用例视图、实现视图和部署视图。
- 动态行为描述了系统随时间变化的行为。行为用从静态视图中抽取的瞬间值的变化来描述。动态行为视图包括状态机视图、活动视图和交互视图。

- 下表列出了UML的视图和视图所包括的图以及与每种图有关的主要概念。不能把这张表看成是一套死板的规则，应将其视为对UML常规使用方法的指导，因为UML允许使用混合视图。

主要的域	视图	图	主要概念
结构	静态视图	类图	类、关联、泛化、依赖关系、实现、接口
	用例视图	用例图	用例、参与者、关联、扩展、包括、用例泛化
	实现视图	构件图	构件、接口、依赖关系、实现
	部署视图	部署图	节点、构件、连接、位置
动态	状态机视图	状态机图	状态、事件、转换、活动、动作
	活动视图	活动图	状态、活动、完成转换、分叉、结合
	交互视图	顺序图	交互、对象、消息、激活
		协作图	协作、交互、协作角色、消息
模型管理	模型管理视图	包图	包、子系统、模型
可扩展性	所有	所有	构造型、标记值、约束

- UML还包括多种具有扩展能力的组件，这些组件包括约束、构造型和标记值，它们适用于所有的视图元素。
- 将这些总结起来，在UML中主要包括的视图为静态视图、用例视图、交互视图、实现视图、状态机视图、活动视图、部署视图和模型管理视图。物理视图对应用自身的实现结构建模，例如系统的构件组织和建立在运行节点上的配置。由于实现视图和部署视图都是反映了系统中的类映射成物理构件和节点的机制，可以将其归纳为物理视图。

1. 静态视图

静态视图是对在应用领域中的各种概念以及与系统实现相关的各种内部概念进行的建模。静态视图主要由类与类之间的关系构成，这些关系包括关联、泛化和依赖关系，我们又把依赖关系具体再分为使用和实现关系。可以从以下三个方面来了解静态视图在UML中的作用。

(1) 静态视图是UML的基础。

(2) 静态视图构造了这些概念对象的基本结构。

(3) 静态视图也是建立其他动态视图的基础。

（1）静态视图是UML的基础。

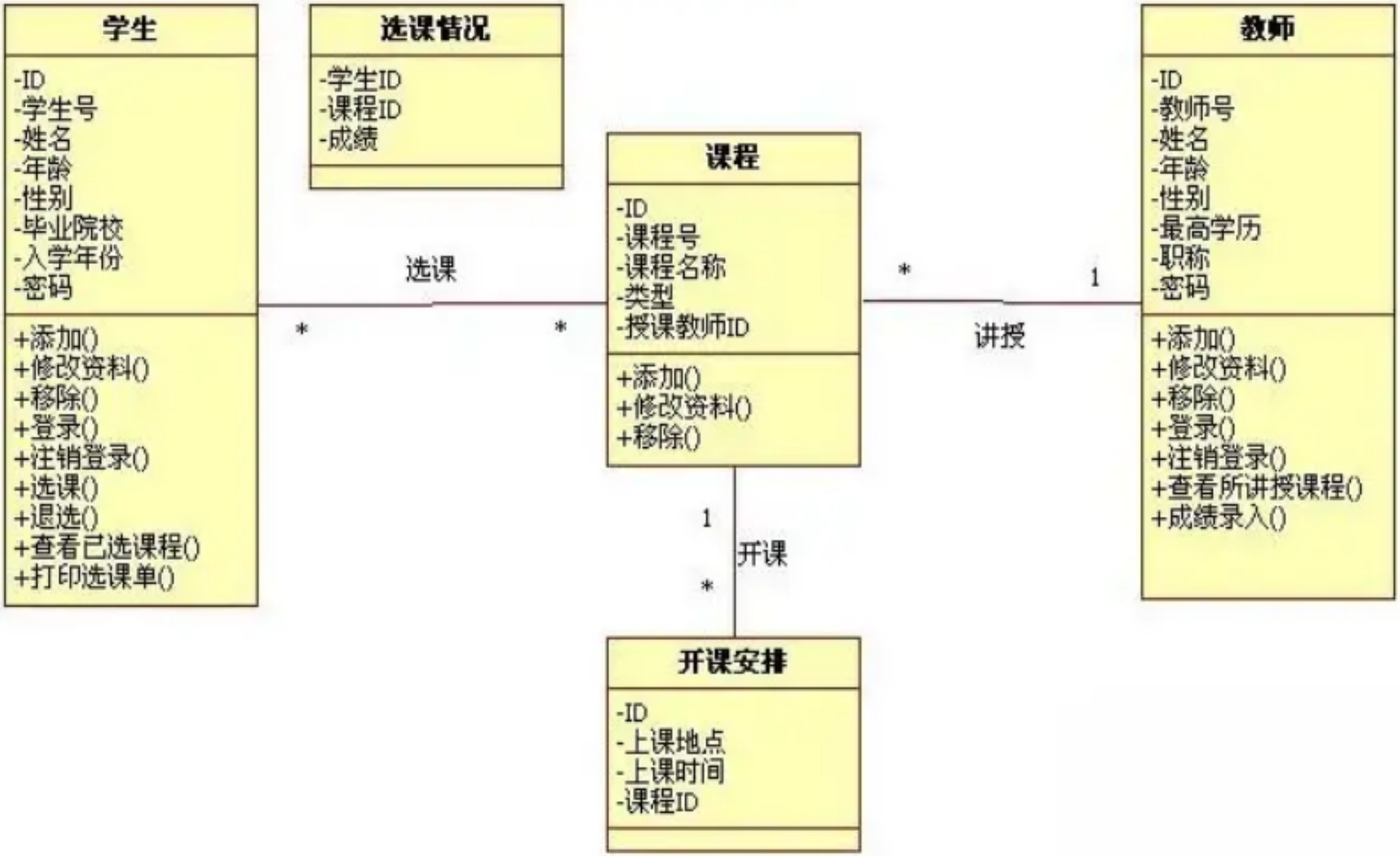
模型中静态视图的元素代表的是现实系统应用中有意义的概念，这些系统应用中的各种概念包括真实世界中的概念、抽象的概念、实现方面的概念和计算机领域的概念。例如，一个仓库管理系统由各种概念构成，如仓库、仓库中的物料、仓库管理员、物料领取者、物料信息等。静态视图描绘的是客观现实世界的基本认知元素，是建立一个系统中所需概念的集合。

（2）静态视图构造了这些概念对象的基本结构。

静态视图不仅包括所有的对象数据结构，同时也包括了对数据的操作。根据面向对象的观点，数据和对数据的操作是紧密相关的，数据和对数据的操作可量化为类。例如，仓库中的物料对象可以携带数据，如物料的供货商、物料的编号、物料的进价，并且物料对象还包含了对物料的基本信息的操作，如物料的出库和入库等。

（3）静态视图也是建立其他动态视图的基础。

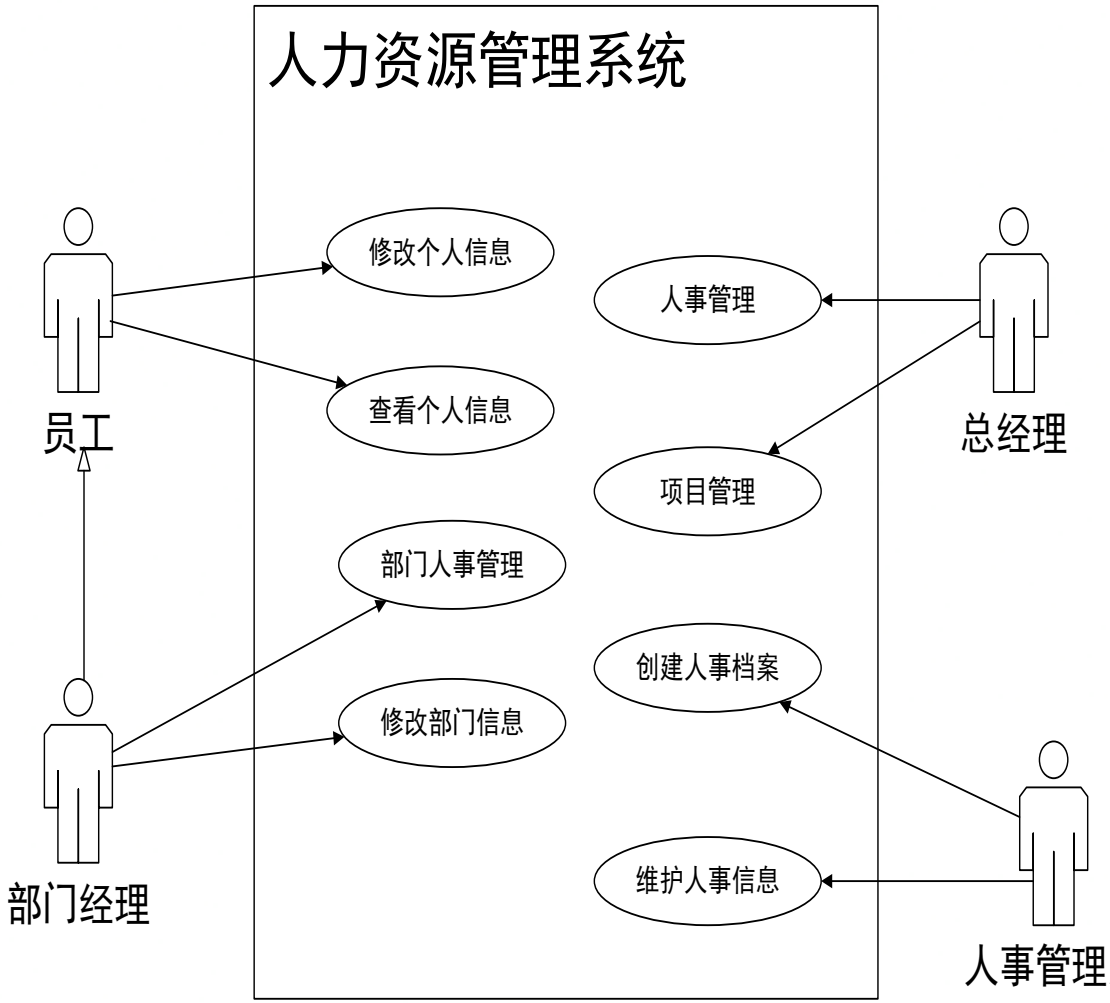
静态视图将具体的数据操作使用离散的模型元素进行描述，尽管它不包括对具体动态行为细节的描述，但是它们是类所拥有并使用的元素，使用和数据同样的描述方式，只是在标识上进行区分。我们要建立的基础是说清楚什么在进行交互作用。如果无法说清楚交互作用是怎样进行的，那么也无从构建静态视图。


学生
-ID
-学生号
-姓名
-年龄
-性别
-毕业院校
-入学年份
-密码
+添加()
+修改资料()
+移除()
+登录()
+注销登录()
+选课()
+退选()
+查看已选课程()
+打印选课单()
选课情况
-学生ID
-课程ID
-成绩
选课
*
*
课程
-ID
-课程号
-课程名称
-类型
-授课教师ID
+添加()
+修改资料()
+移除()
*
1
讲授
教师
-ID
-教师号
-姓名
-年龄
-性别
-最高学历
-职称
-密码
+添加()
+修改资料()
+移除()
+登录()
+注销登录()
+查看所讲授课程()
+成绩录入()
1
开课
*
开课安排
-ID
-上课地点
-上课时间
-课程ID

2. 用例视图

- 用例视图描述了系统的参与者与系统进行交互的功能，是参与者所能观察和使用到的系统功能的模型图。
- 一个用例是系统的一个功能单元，是系统参与者与系统之间进行的一次交互作用。
- 用例模型的用途是标识出系统中的用例和参与者之间的联系，并确定什么样的参与者执行了哪个用例。
- 用例使用系统与一个或多个参与者之间的一系列消息来描述系统的交互作用。系统参与者可以是人，也可以是外部系统或外部子系统等。

- 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。

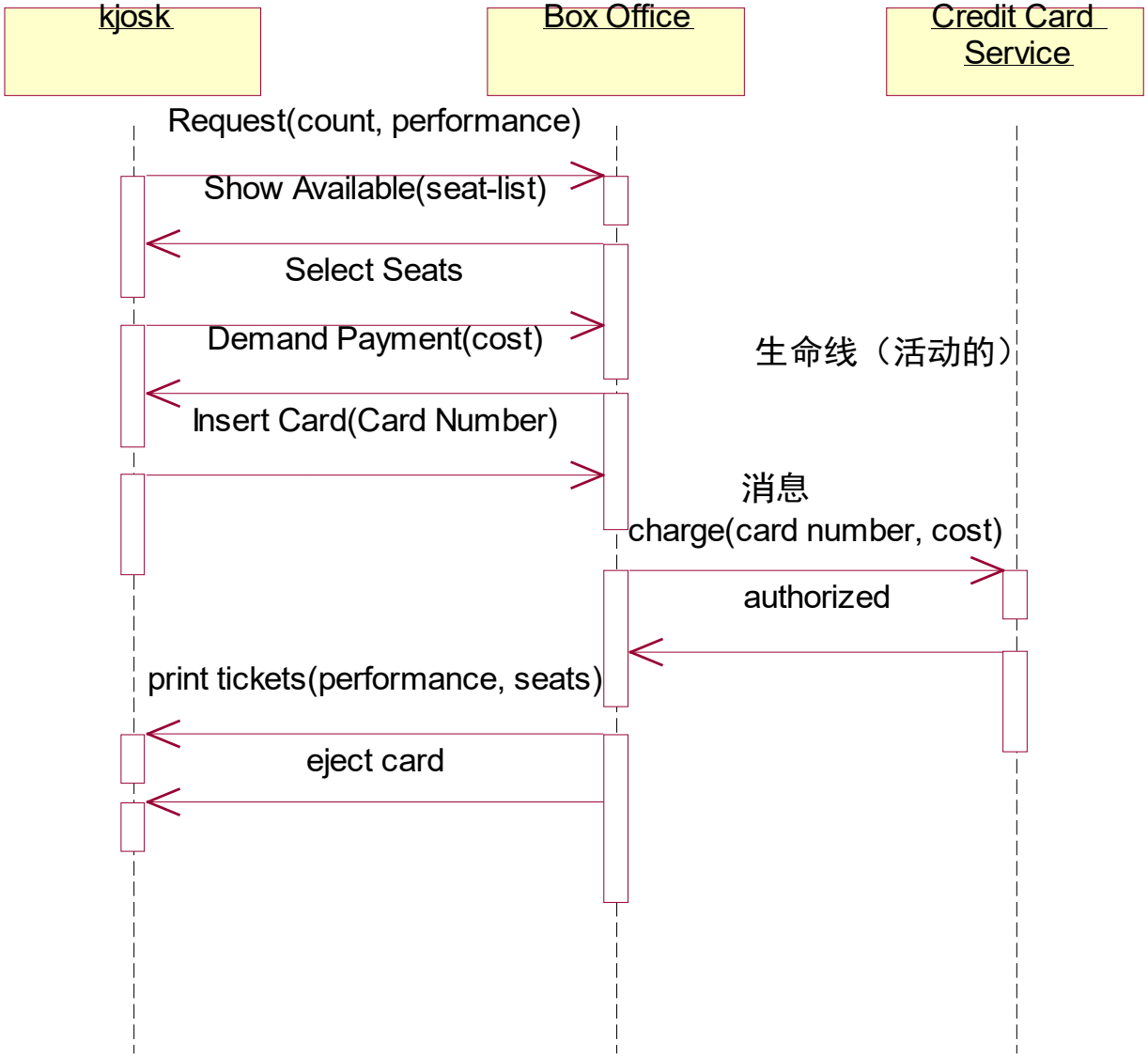


3. 交互视图

- 交互视图描述了执行系统功能的各个角色之间相互传递消息的顺序关系，是描绘系统中各种角色或功能交互的模型。
- 交互视图显示了跨越多个对象的系统控制流程。
- 通过不同对象间的相互作用来描述系统的行为，是通过两种方式进行的，一种方式是以独立的对象为中心进行描述，另一种方式是以相互作用的一组对象为中心进行描述。
- 交互视图可运用两种形式来表示： 顺 序 图和协作图，它们各有自己的侧重点。

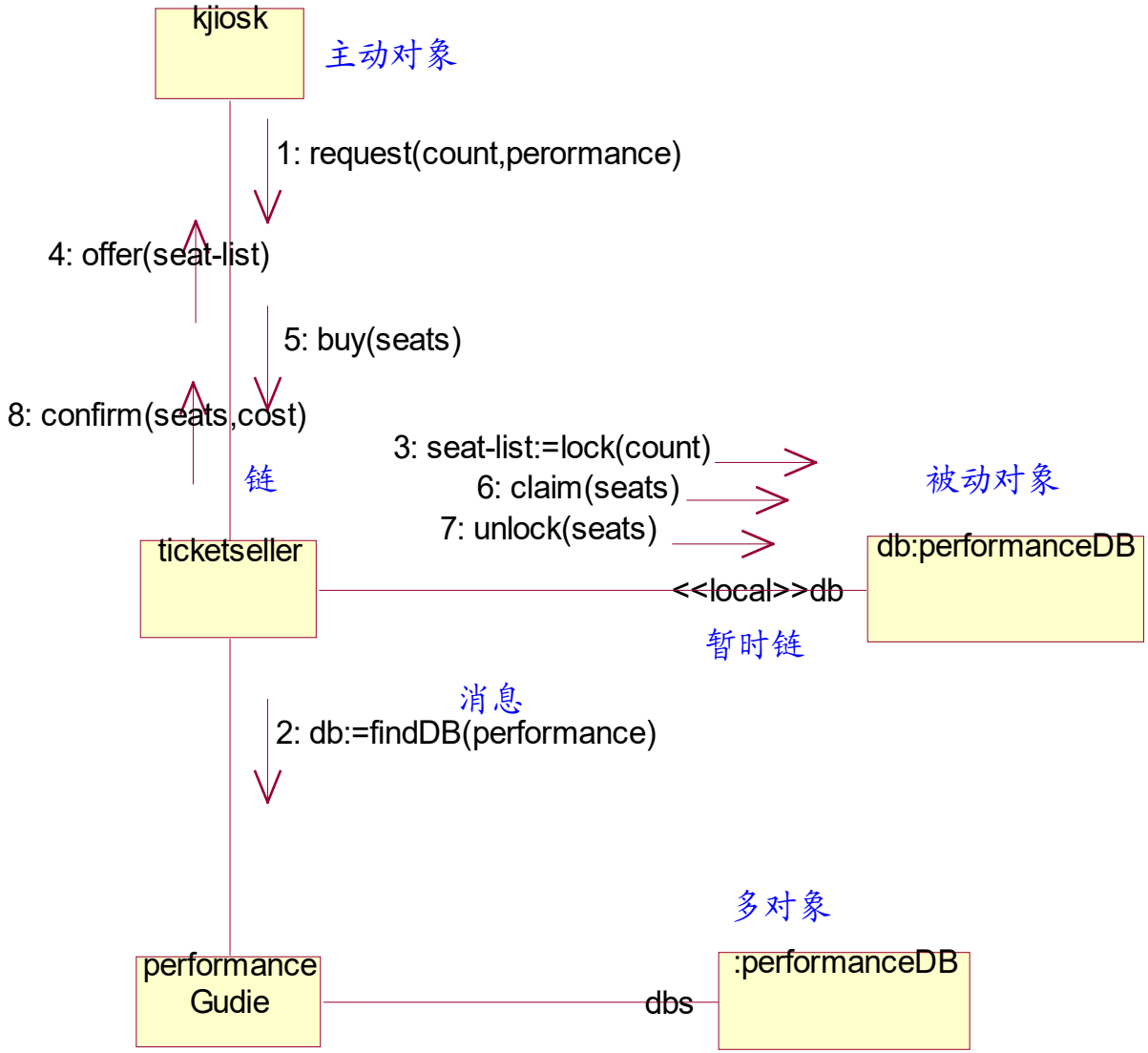
顺序图

- 顺序图表示了对象之间传送消息的时间顺序
- 每一个类元角色用一条生命线来表示—即用垂直线代表整个交互过程中对象的生命期。生命线之间的箭头连线代表消息。
- 顺序图可以用来进行一个场景说明—即一个事务的历史过程
- 顺序图的一个用途是用来表示用例中的行为顺序
- 当执行一个用例行为时，顺序图中的每条消息对应了一个类操作或状态机中引起转换的触发事件


kjosk
Box Office
Credit Card Service
Request(count, performance)
Show Available(seat-list)
Select Seats
Demand Payment(cost)
Insert Card(Card Number)
生命线（活动的）
消息
charge(card number, cost)
authorized
print tickets(performance, seats)
eject card

协作图

- 协作图对在一次交互中有意义的对象和对象间的链建模
- 类元角色描述了一个对象，关联角色描述了协作关系中的一个链。
- 协作图用几何排列来表示交互作用中的各角色。
- 附在类元角色上的箭头代表消息。消息的发生顺序用消息箭头处的编号来说明。

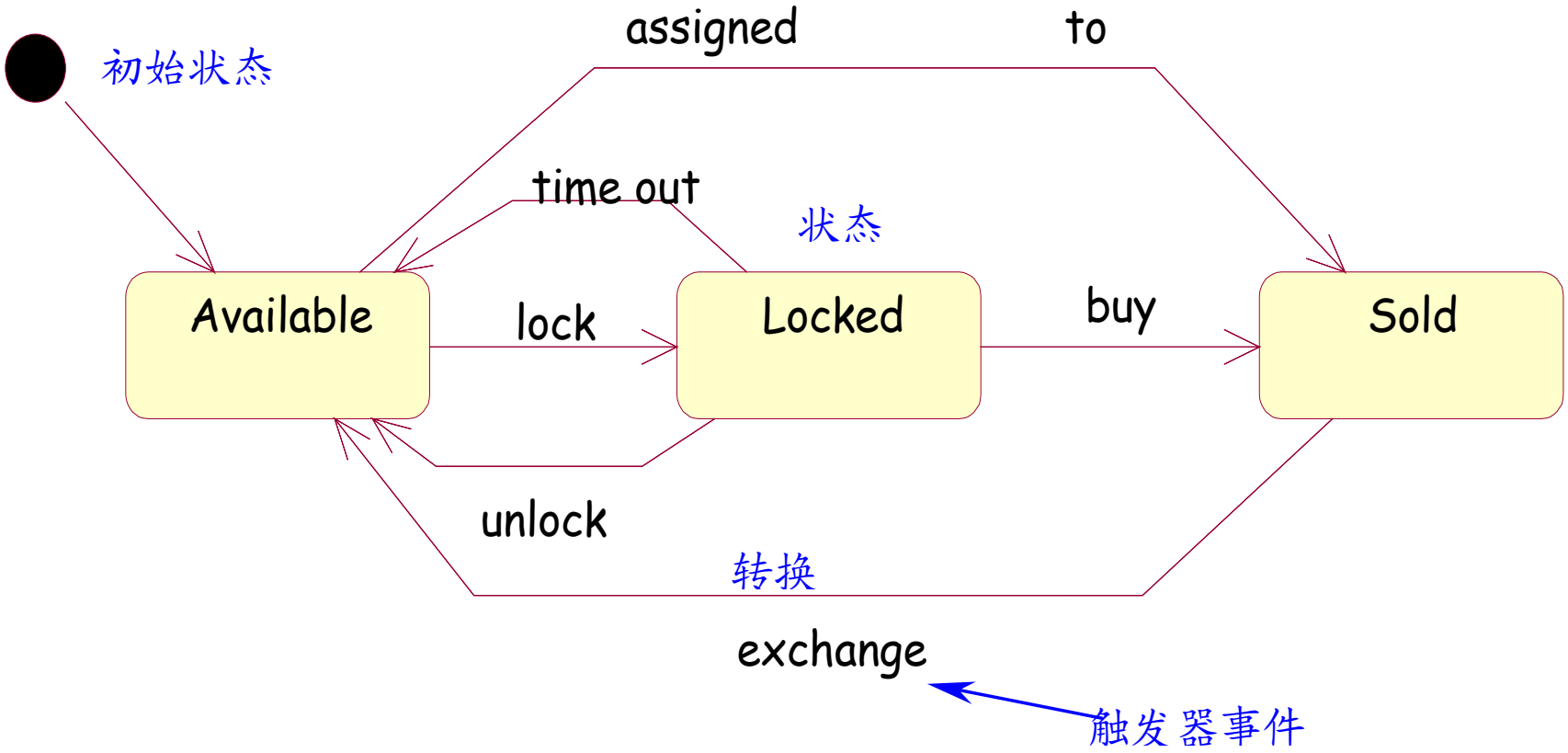

kjiosk
主动对象
1: request(count,perormance)
4: offer(seat-list)
5: buy(seats)
8: confirm(seats,cost)
链
3: seat-list:=lock(count)
6: claim(seats)
7: unlock(seats)
被动对象
ticketseller
db:performanceDB
<<local>>db
暂时链
消息
2: db:=findDB(performance)
多对象
performance Gudie
dbs
:performanceDB

顺序图 VS.协作图

- 顺序图和协作图都可以表示各对象间的交互关系，但它们的侧重点不同。
  - 顺序图用消息的几何排列关系来表达消息的时间顺序，各角色之间的相关关系是隐含的
  - 协作图用各个角色的几何排列图形来表示角色之间的关系，并用消息来说明这些关系。
- 在实际中可以根据需要选用这两种图。

4. 状态机视图

- 状态机视图是通过对象的各种状态建立模型来描述对象随时间变化的动态行为。它描述了一个对象自身具有的所有状态。
- 一个状态机由该对象的各种所处状态以及连接这些状态的符号组成。每个状态对一个对象在其生命期中满足某种条件的一个时间段建模。
- 当一个事件发生时，它会触发状态间的转换，导致对象从一种状态转化到另一种新的状态。与转换相关的活动执行时，转换也同时发生。
- 状态图可用于描述用户接口、设备控制器和其他具有反馈的子系统。它还可用于描述在生命期中跨越多个不同性质阶段的被动对象的行为 。


初始状态
assigned
to
time out
状态
Available
lock
Locked
buy
Sold
unlock
转换
exchange
触发器事件

5. 活动视图

- 活动视图是一种特殊形式的状态机视图，是状态机的一个变体，用来描述执行算法的工作流程中涉及的活动。
- 通常活动视图用于对计算流程和工作流程建模。活动视图中的状态表示计算过程中所处的各种状态。
- 活动视图使用活动图来体现。活动图中包含了描述对象活动或动作的状态以及对这些状态的控制。
- 活动图的用途
  - 对人类组织的现实世界中的工作流程建模
  - 对事物建模是活动图的主要用途，但活动图也可对软件系统中的活动建模
  - 活动图有助于理解系统高层活动的执行行为，而不涉及建立协作图所必须的消息传送细节

6. 物理视图

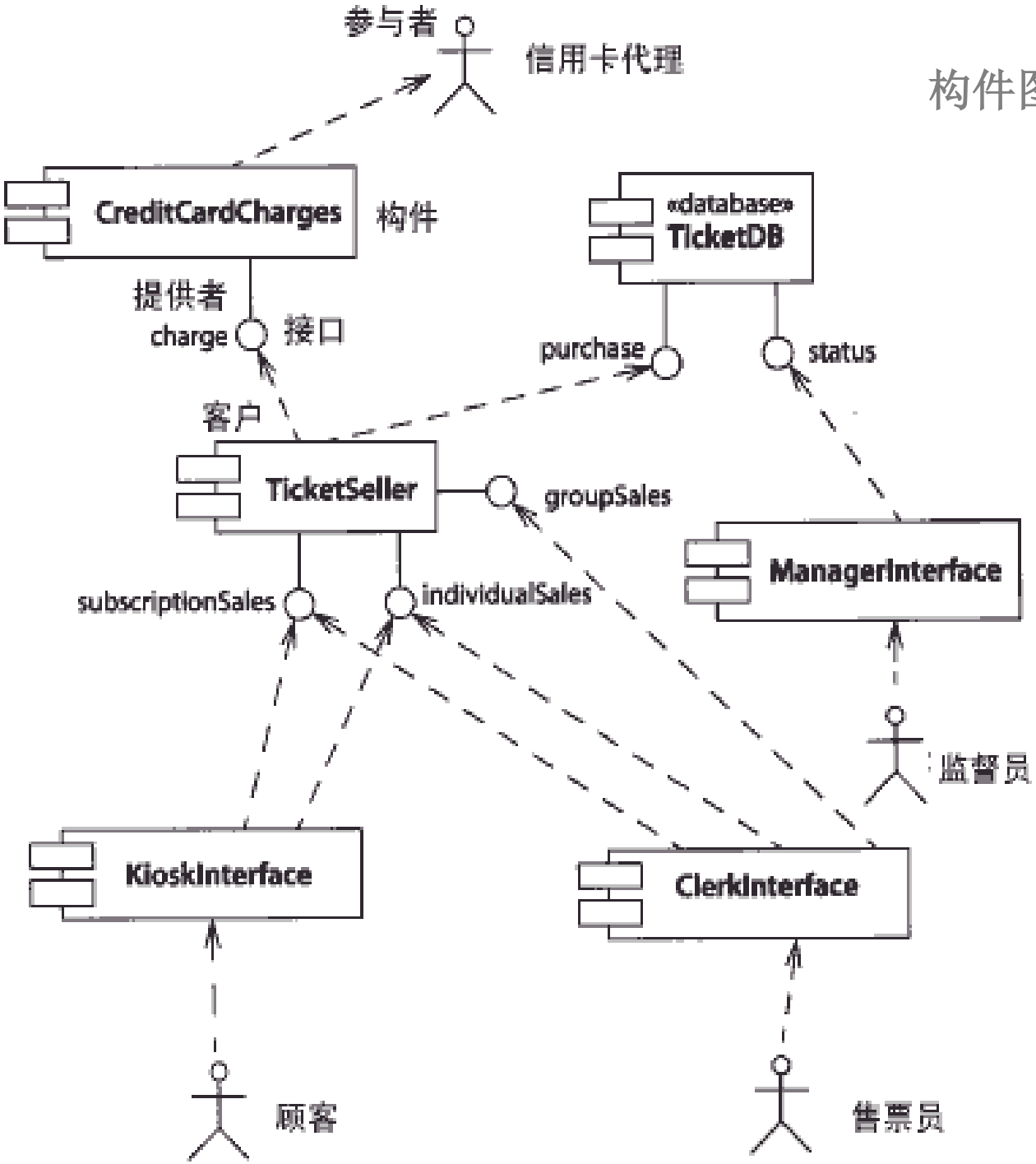
- 前面所提到物理视图包含两种视图，分别是实现视图和部署视图。物理视图是对应用自身的实现结构建模，如系统的构件组织情况以及运行节点的配置等。物理视图提供了将系统中的类映射成物理构件和节点的机制。
- 为了可重用性和可操作性的目的，系统实现方面的信息也很重要。实现视图将系统中可重用的块包装成为具有可替代性的物理单元，这些单元被称为构件。
- 实现视图用构件及构件间的接口和依赖关系来表示设计元素的具体实现。构件是系统高层的可重用的组成部件。

物理视图

- 物理视图对应用自身的实现结构建模
  - 例如系统的构件组织和建立在运行节点上的配置。
- 物理视图提供了将系统中的类映射成物理构件和节点的机制
- 物理视图有两种
  - 实现视图
  - 部署视图

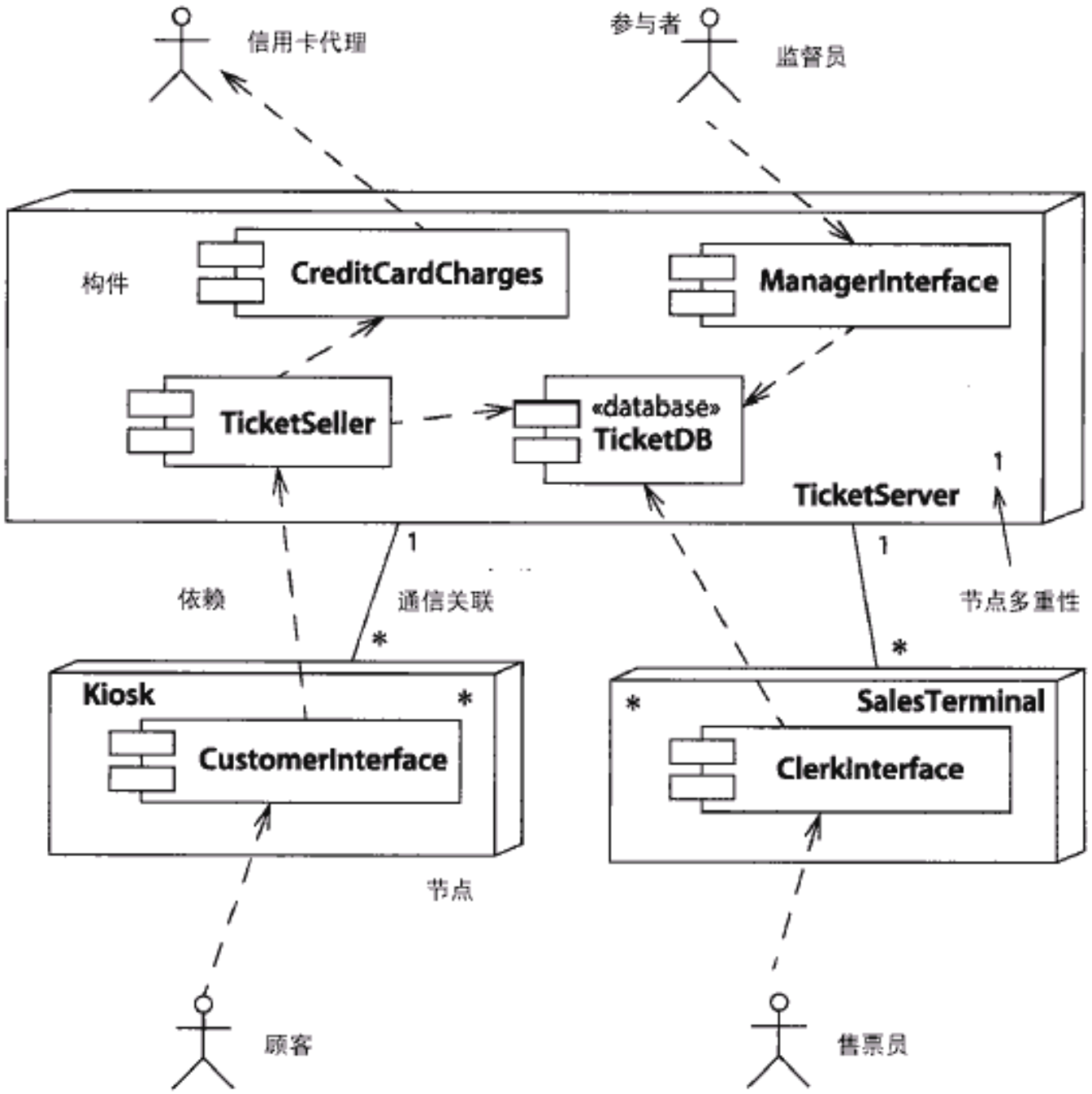
- 实现视图为系统的构件建模型，也包括各构件之间的依赖关系
  - 可以通过这些依赖关系来估计对系统构件的修改给系统可能带来的影响。
- 实现视图用构件图来表现
- 构件图表示了系统中的各种构件

构件图


参与者
信用卡代理
CreditCardCharges
构件
«database»
TicketDB
提供者
charge
接口
purchase
status
客户
TicketSeller
groupSales
ManagerInterface
subscriptionSales
individualSales
监督员
KioskInterface
ClerkInterface
顾客
售票员

部署视图

- 部署视图描述位于节点实例上的运行构件实例的安排。
- 节点是一组运行资源，如计算机、设备或存储器。
- 部署视图允许评估分配结果和资源分配
- 部署视图用部署图来表达


信用卡代理
参与者
监督员
构件
CreditCardCharges
ManagerInterface
TicketSeller
«database»
TicketDB
TicketServer
1
1
1
依赖
通信关联
节点多重性
*
*
Kiosk
*
CustomerInterface
*
SalesTerminal
ClerkInterface
节点
顾客
售票员

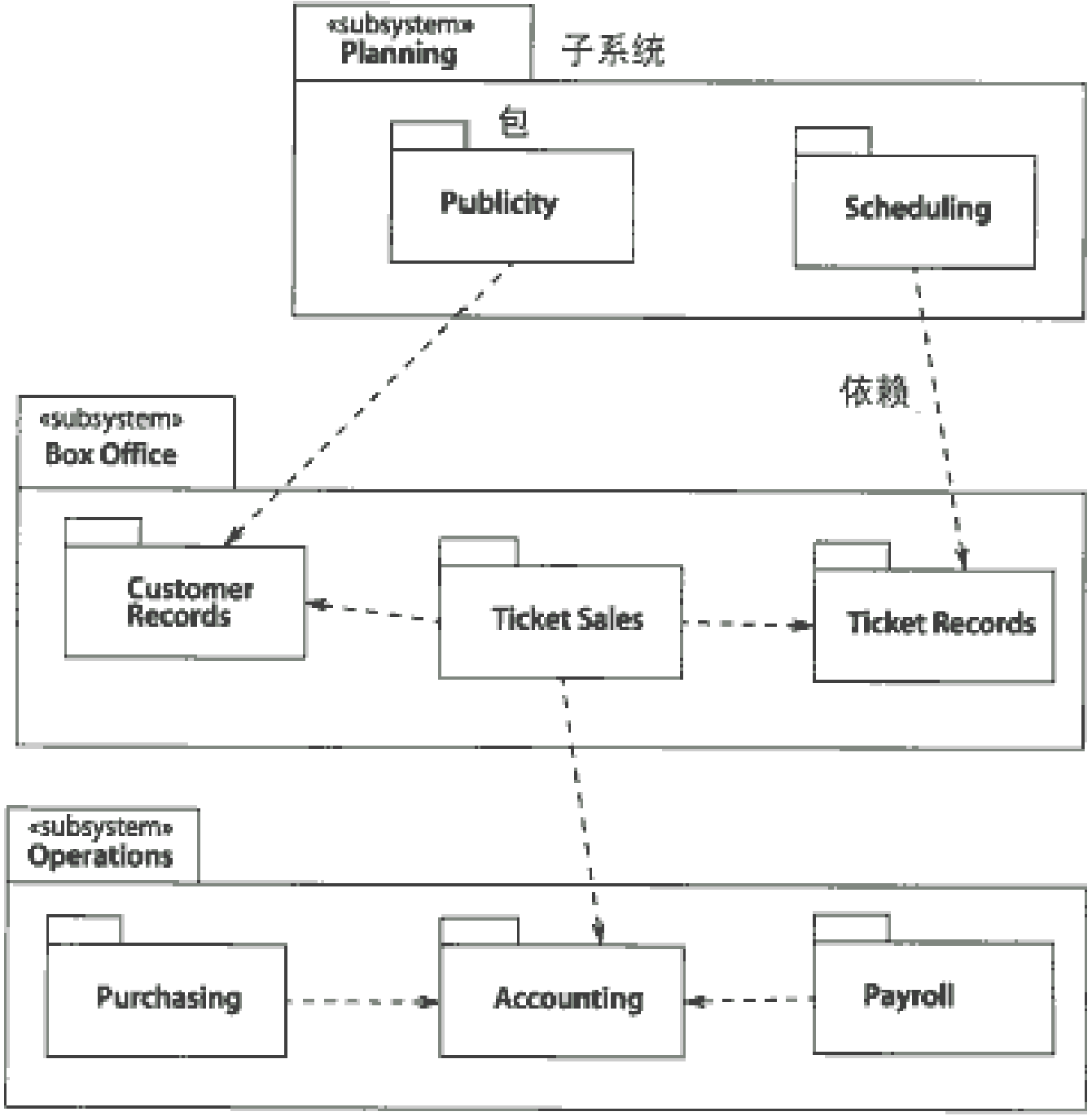
7. 模型管理视图

模型管理视图是对模型自身组织进行的建模，是由自身的一系列模型元素（如类、状态机和用例）构成的包所组成的模型。模型是从某一观点以一定的精确程度对系统所进行的完整描述。

模型是一种特殊的包。一个包还可以包含其他的包。整个系统的静态模型实际上可看成是系统最大的包，它直接或间接包含了模型中的所有元素内容。包是操作模型内容、存取控制和配置控制的基本单元。每一个模型元素包含或被包含于其他模型元素中。子系统是另一种特殊的包。

模型管理视图

- 模型管理视图对模型自身组织建模
  - 一系列由模型元素（如类、状态机和用例）构成的包组成了模型
  - 一个包(package)可能包含其他的包，因此，整个模型实际上可看成一个根包，它间接包含了模型中的所有内容
  - 包是操作模型内容、存取控制和配置控制的基本单元。
- 模型是从某一观点以一定的精确程度对系统所进行的完整描述
  - 从不同的视角出发，对同一系统可能会建立多个模型，例如有系统分析模型和系统设计模型之分。
  - 模型是一种特殊的包。
- 子系统是另一种特殊的包
  - 它代表了系统的一个部分，它有清晰的接口，这个接口可作为一个单独的构件来实现。
- 模型管理信息通常在类图中表达


«subsystem»
Planning
子系统
包
Publicity
Scheduling
依赖
«subsystem»
Box Office
Customer Records
Ticket Sales
Ticket Records
«subsystem»
Operations
Purchasing
Accounting
Payroll

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，如表3-4所示。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

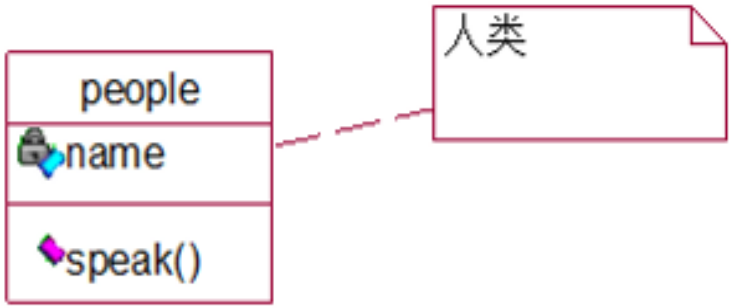
UML的通用机制

- 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”。

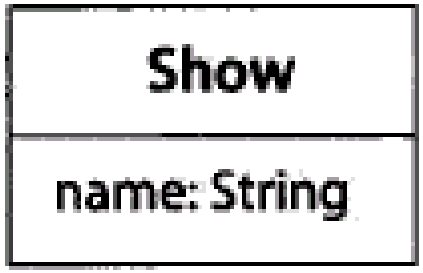
- 约束

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。

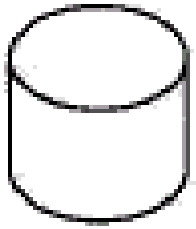
扩展组件

- 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。



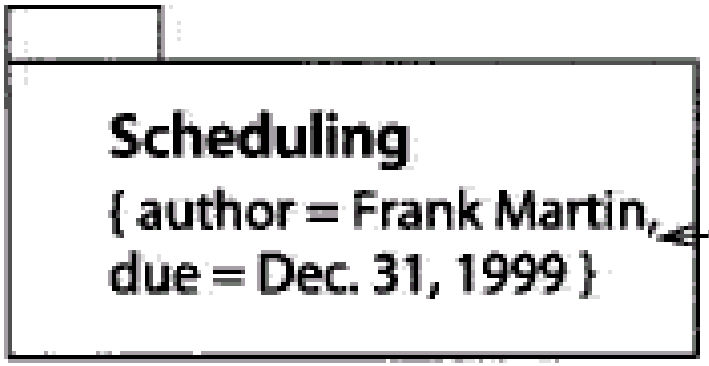
{names for one season must be unique} ← 约束

构造型



TicketDB

构造型图标



标记值

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 UML建模工具
- 1.5 总结

UML建模工具

- 能绘制UML图形的工具主要有很多，例如：

  Rational Rose（ Rational Software Architect ）

  Enterprise Architect

  MS Visio

  StarUML

  Draw.io

（1）Rational Rose

Rose主要是在开发过程中的各种语义、模块、对象以及流程，状态等描述比较好，主要体现在能够从各个方面和角度来分析和设计，使软件的开发蓝图更清晰，内部结构更加明朗，对系统的代码框架生成有很好的支持。

Rose主要是在开发过程中的各种语义、模块、对象以及流程，状态等描述比较好，主要体现在能够从各个方面和角度来分析和设计，使软件的开发蓝图更清晰，内部结构更加明朗，对系统的代码框架生成有很好的支持。

从使用的角度分析，Rational Rose易于使用，支持使用多种构件和多种语言的复杂系统建模；利用双向工程技术可以实现迭代式开发；团队管理特性支持大型、复杂的项目和大型而且通常队员分散在各个不同地方的开发团队。

（2）Enterprise Architect

Enterprise Architect ，简称EA，是澳大利亚Sparx Systems 公司的旗舰产品。

Enterprise Architect是一个完全的UML分析和设计工具，它能完成从需求收集经步骤分析、模型设计到测试和维护的整个软件开发过程。它基于多用户Windows平台的图形工具，可以帮助您设计健全可维护的软件。除此，它还包含特性灵活的高品质文档输出。用户指南可以在线获取。

（3）Visio

UML建模工具Visio 原来仅仅是一种画图工具，能够用来描述

各种图形，如 0才开始引

进软件分析 是目前最能

够用图形方 开发中的

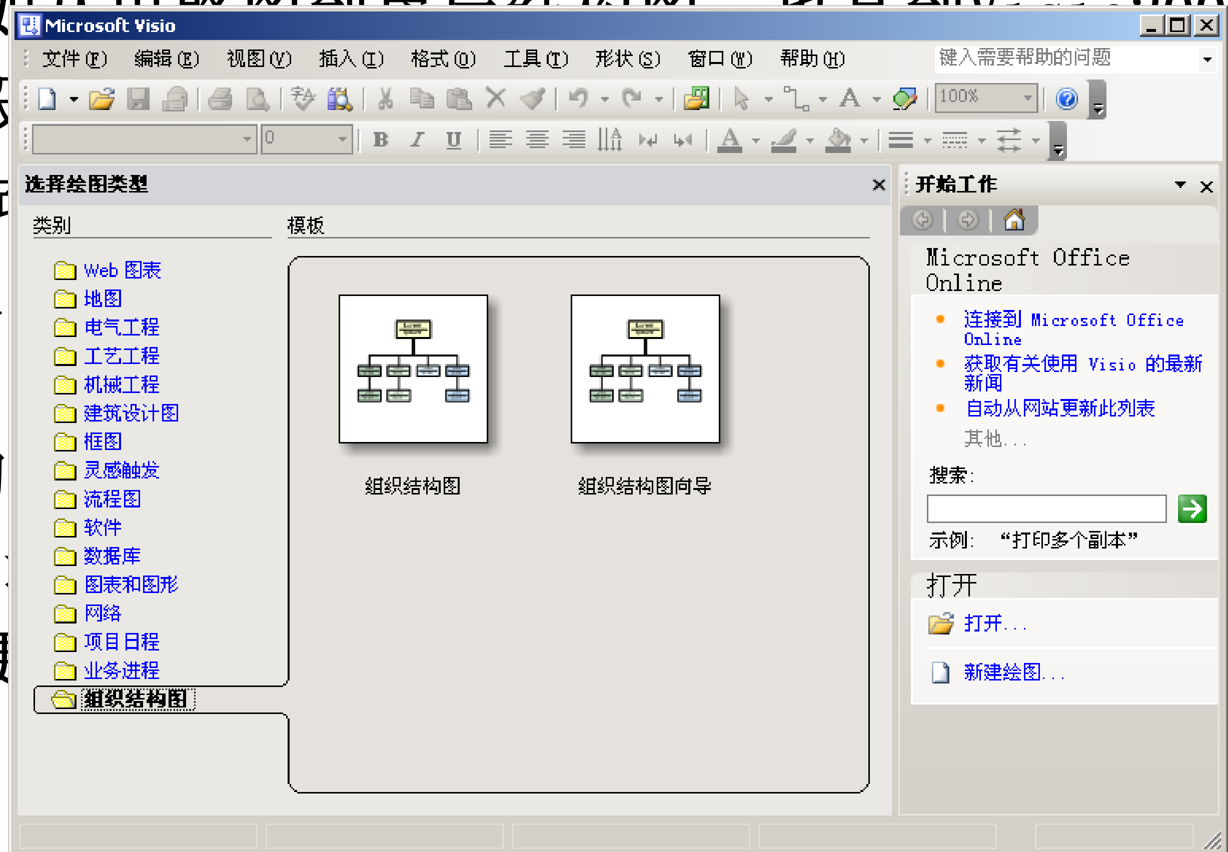
UML支持仅仅 e产品的能

够很好兼容。 当中。但是

对于代码的 +、MS SQL

Server 等（ 图形语义的

描述比较方 牵强。

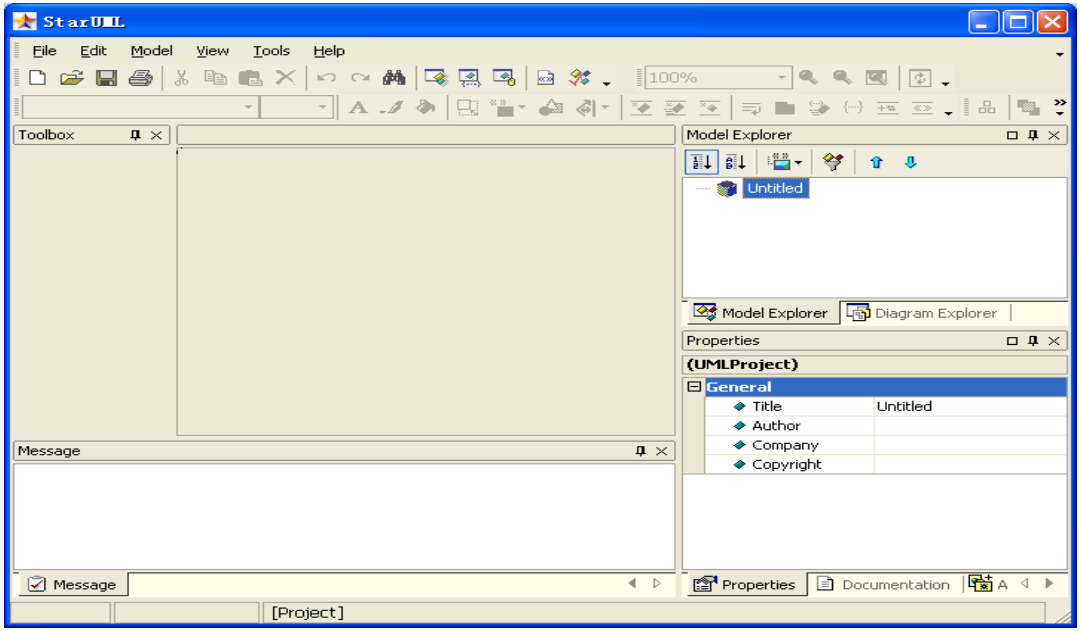


（4）StarUML

- StarUML（简称SU）是一款免费的开放源码的UML开发工具，由韩国公司主导开发出来的，可以直接到StarUML网站下载。

- StarUML适用于个人和小型团队。它提供了基本的UML图形绘制功能，如类图、对象图等。虽然功能相对简单，但StartUML具有良好的可扩展性，用户可以通过插件来扩展其功能。此外，它还支持代码生成和模型转换，使用户可以轻松地在UML图和代码之间进行转换。

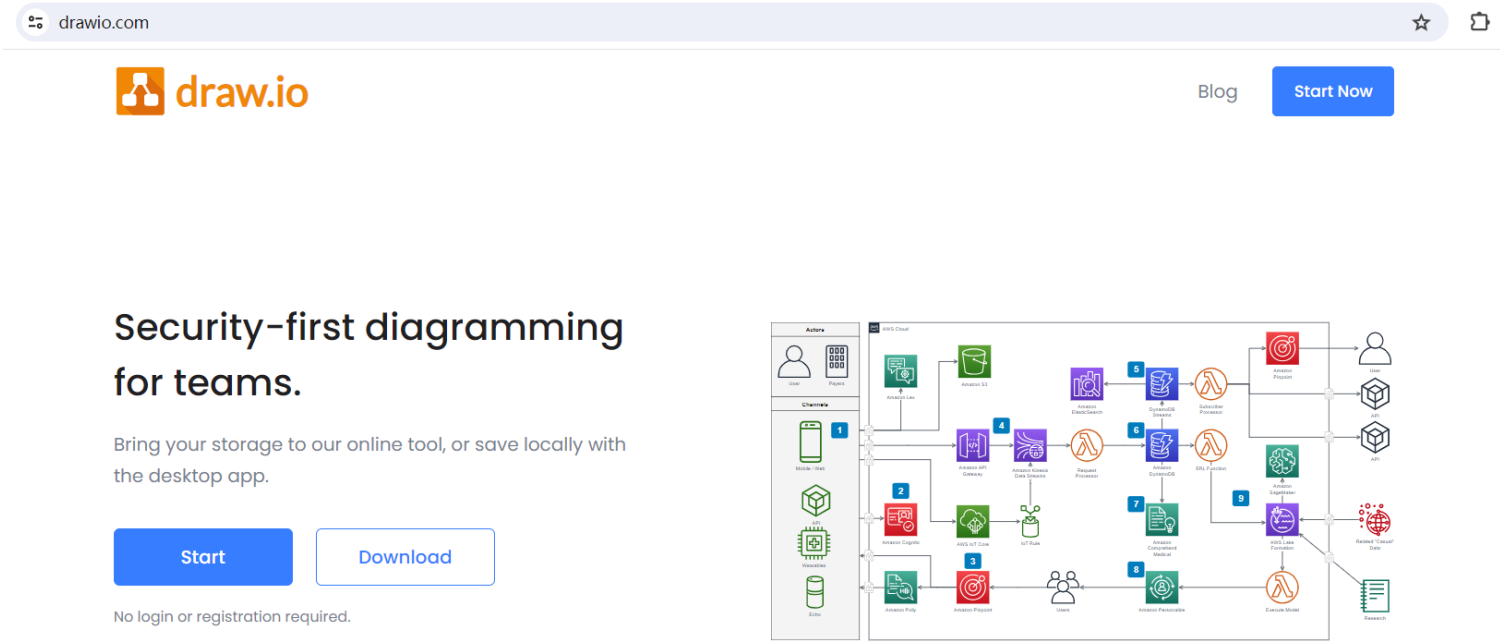
- UML 2.0分为两大类：结构图（Structure Diagram）和行为图（Behavior Diagram）共13种图。结构图用于对系统的静态结构建模，包括类图、组合结构图、构件图、部署图、对象图和包图；行为图用于对系统的动态行为建模，包括实例图、交互图（顺序图、通信图、交互概览图、定时图）、活动图和状态机图。StarUML可支持除对象图、包图、定时图和交互预览图之外的其他11种图的绘制。



(5) Draw.io是一个非常出色的免费、开源、简洁、方便的绘图软件，利用这款软件可以绘制出生动、有趣的图形，包括流程图、地图、网络架构图、UML用例图、流程图等等。它支持各种快捷键，免费提供了1000多张画图模板，图形形状丰富，还可以自定义图形工具栏。它还支持云存储和各种导出格式，包括图片格式和PDF格式等。

Draw.io软件内置了丰富的绘图资源，包括各种形状、图标、连接器和模板，能够满足绝大多数绘图需求。它还支持导入第三方图标资源，以满足更多的需求。使用Draw.io，可以通过直观易用的界面绘制图表，添加文本、图标、箭头和其他元素，轻松构建清晰明了的图示。它还支持图层功能，可以轻松管理和编辑不同元素，使得图表的修改和调整变得非常便捷。除了提供丰富的绘图工具和资源，Draw.io还具备方便的共享和协作功能。可以将绘图作品保存在云端，与他人共享和协作编辑，促进团队间的合作和沟通。

- **Draw.io：**
- **https://github.com/jgraph/drawio-desktop/releases/tag/v24.7.8**



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