第5章 面向对象方法与UML

- 面向对象的概念与开发方法
- **UML**简介
- **UML**的事物
- **UML**的关系
- **UML**的图
- 使用和扩展**UML**

5.1 面向对象的概念与开发方法

- 现实世界就是由各种对象组成的，如建筑物、人、汽车、动物、植物等。
- 复杂的对象可以由简单的对象组成。
- 在研究对象时主要考虑对象的属性和行为。
- 通常将属性及行为相同或相似的对象归为一类。
- 类可以看成是对象的抽象，代表了此类对象所具有的共有属性和行为。

5.1 面向对象的概念与开发方法

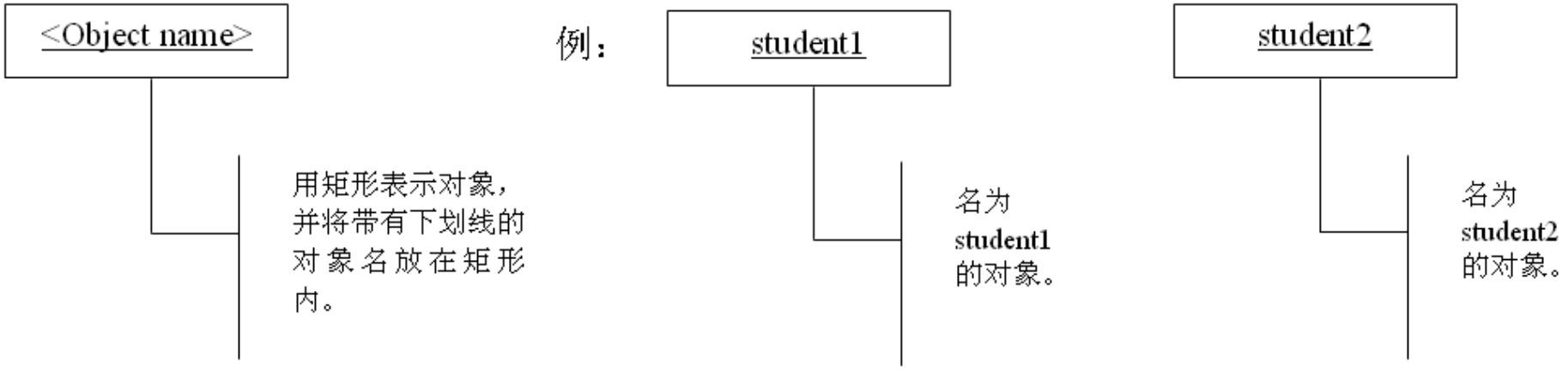
- **Coad和Yourdon给出了“面向对象”的一个定义：**

  面向对象＝对象＋类＋继承＋消息通信

- 如果一个系统是使用这样4个概念设计和实现的，则可认为这个系统是面向对象的。

对象

- 对象是包含现实世界物体特征的抽象实体，它反映了系统为之保存信息和（或）与它交互的能力。
- 例如，Student对象的数据可能有姓名、性别、出生日期、家庭住址、电话号码等，其操作可能是对这些数据值的赋值及更改。



对象的图形表示

对象

- 对象与后面讲的类具有几乎完全相同的表示形式，主要差别是对象的名字下面要加一条下划线。对象名有下列三种表示格式：

(1) 第一种格式是对象名在前，类名在后，中间用冒号连接。形如：

对象名：类名

(2) 第二种格式形如：

：类名

这种格式用于尚未给对象命名的情况，注意，类名前的冒号不能省略。

(3) 第三种格式形如：

对象名

对象

- 对象有两个层次的概念：

(1) 现实生活中对象指的是客观世界的实体。可以是可见的有形对象，如人、学生、汽车、房屋等；也可以是抽象的逻辑对象，如银行帐号，生日。

(2) 程序中的对象就是一组变量和相关方法的集合，其中变量表明对象的状态，方法表明对象所具有的行为。

对象

- 可以将程序中的对象分为5类：物理对象，角色，事件，交互，规格说明。

(1) 物理对象（Physical Objects）—— 物理对象是最易识别的对象，通常可以在问题领域的描述中找到，它们的属性可以标识和测量。

例如，大学课程注册系统中的学生对象；一个网络管理系统中各种网络物理资源对象（如开关、CPU和打印机）都是物理对象。

对象

(2) 角色（Roles）—— 一个实体的角色也可以抽象成一个单独的对象。角色对象的操作是由角色提供的技能。

- 例如，一个面向对象系统中通常有“管理器”对象，它履行协调系统资源的角色。一个窗口系统中通常有“窗口管理器”对象，它扮演协调鼠标器按钮和其他窗口操作的角色。特别地，一个实际的物理对象可能同时承担几个角色。
- 例如，一个退休教师同时扮演退休者和教师的角色。

对象

(3) 事件（**Events**）—— 一个事件是某种活动的一次“出现”。

- 例如“鼠标”事件。一个事件对象通常是一个数据实体，它管理“出现”的重要信息。事件对象的操作主要用于对数据的存取。
- 如“鼠标”事件对象有诸如光标坐标、左右键、单击，双击等信息。

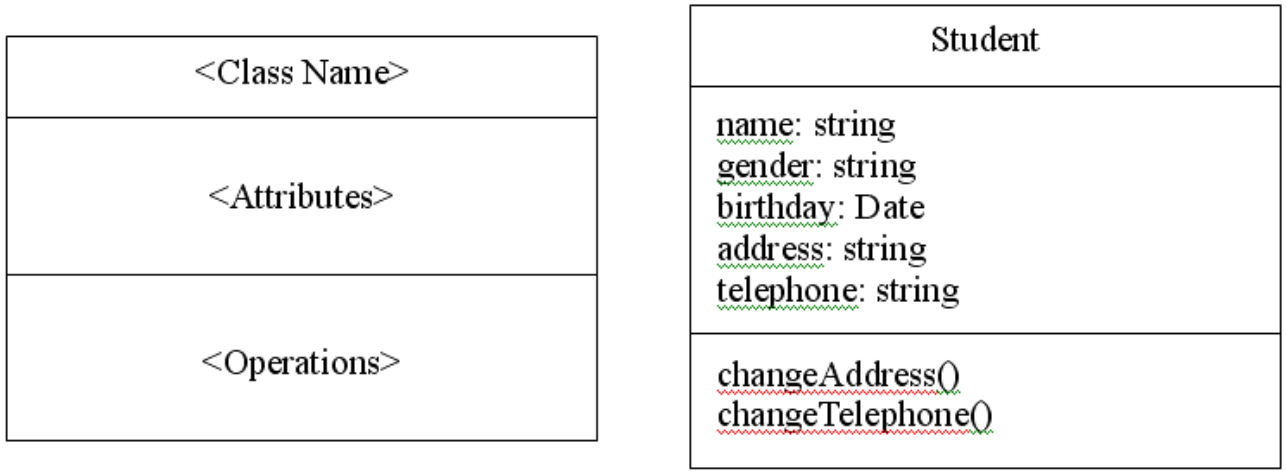
对象

(4) 交互（**Interactions**）—— 交互表示了在两个对象之间的关系，这种类型的对象类似于在数据库设计时所涉及的“关系”实体。

- 当实体之间是多对多的关系时，利用交互对象可将其简化为两个一对多的关系。
- 例如，在大学课程注册系统中，学生和课程之间的关系是多对多的关系，可设置一个“选课”交互对象来简化它们之间的关系。

类与封装

- **类。**可以将现实生活中的对象经过抽象，映射为程序中的对象。对象在程序中是通过一种抽象数据类型来描述的，这种抽象数据类型称为类（Class）。
- 为了让计算机创建对象，必须先提供对象的定义，也就是先定义对象所属的类。例如，可以将学生对象所属的类定义为Student。类的图形表示如图所示。



类的图形表示

类与封装

- **封装。**面向对象的封装特性与其抽象特性密切相关。封装是一种信息隐蔽技术，就是利用抽象数据类型将数据和基于数据的操作封装在一起。用户只能看到对象的封装界面信息，对象的内部细节对用户是隐蔽的。
- 封装的定义是：

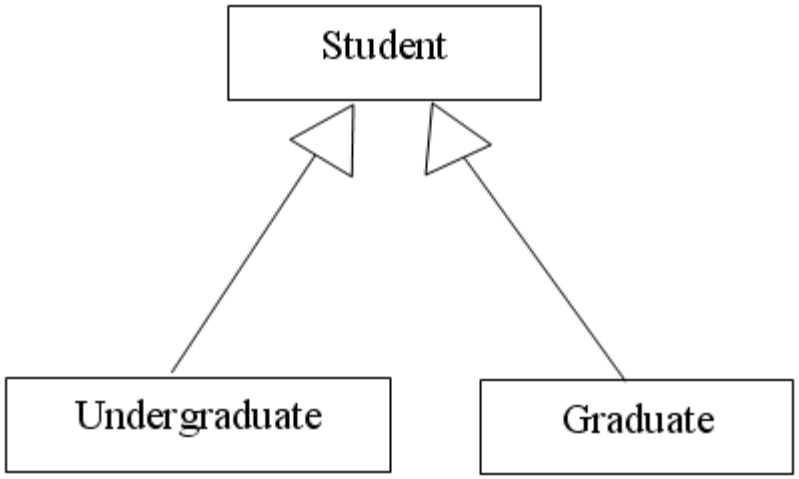
(1) 清楚的边界，所有对象的内部信息被限定在这个边界内；

(2) 接口，即对象向外界提供的方法，外界可以通过这些方法与对象进行交互；

(3) 受保护的内部实现，即软件对象功能的实现细节，实现细节不能从类外访问。

继承

- 继承。继承是一种联结类的层次模型，为类的重用提供了方便，它提供了明确表述不同类之间共性的方法。
- 我们将公共类称为超类(superclass)、父类（father class）、祖先（ancestor）或基类（base class），而从其继承的类称为子类(subclasses)、后代（deslendane）或导出类（derived class）。



类的继承关系

多态

- 根据为请求提供服务的对象不同可以得到不同的行为，这种现象称为多态。
- 在运行时对类进行实例化，并调用与实例化对象相应的方法，称为动态绑定、后期绑定或运行时绑定。相应地，如果方法的调用是在编译时确定的，则称为是静态绑定、前期绑定或编译时绑定。
- 通过在子类中覆盖父类的方法实现多态。

消息通信

- 消息是一个对象与另一个对象的通信单元，是要求某个对象执行类中定义的某个操作的规格说明。
- 发送给一个对象的消息定义了一个方法名和一个参数表（可能是空的），并指定某一个对象。
- 一个对象接收到消息，则调用消息中指定的方法，并将形式参数与参数表中相应的值结合起来。

面向对象的开发方法

- 面向对象软件开发方法的特征
  - **方法的唯一性**
    即方法是对软件开发过程所有阶段进行综合考虑而得到的。
  - **从生存期的一个阶段到下一个阶段的高度连续性，**即生存期后一阶段的成果只是在前一阶段成果的补充和修改。
  - 将面向对象分析(OOA)、面向对象设计(OOD)和面向对象程序设计(OOP)集成到生存期的相应阶段。

面向对象的开发方法

- Rumbaugh方法

  Rumbaugh和他的同事提出的对象模型化技术(OMT)用于分析、系统设计和对象级设计。分析活动建立三个模型:

➢对象模型(描述对象、类、层次和关系);

➢动态模型(描述对象和系统的行为);

➢功能模型(类似于高层的DFD，描述穿越系统的信息流)。

面向对象的开发方法

- Coad和Yourdon方法

Coad和Yourdon方法常常被认为是最容易学习的OOA方法。建模符号相当简单，其OOA过程如下：

(1) 使用“要找什么”准则标识对象；

(2) 定义对象之间的一般化/特殊化结构（又称为分类结构）；

(3) 定义对象之间的整体/部分结构（又称为组合结构）；

(4) 标识主题；

(5) 定义对象的属性及对象之间的实例连接；

(6) 定义服务及对象之间的消息连接。

面向对象的开发方法

- Booch方法

包含“微开发过程”和“宏开发过程”两个过程。

OOA 宏观开发过程如下：

- 标识类和对象；
- 标识类和对象的语义；
- 标识类和对象间的关系；
- 进行一系列精化；
- 实现类和对象。

面向对象的开发方法

- Jacobson方法

  也称为OOSE(面向对象软件工程)，其特点是特别强调使用用例——用以描述用户和产品或系统间如何交互的场景。

过程如下：

- 标识系统的用户和他们的整体责任
- 构造需求模型
- 构造分析模型

5.2 UML简介

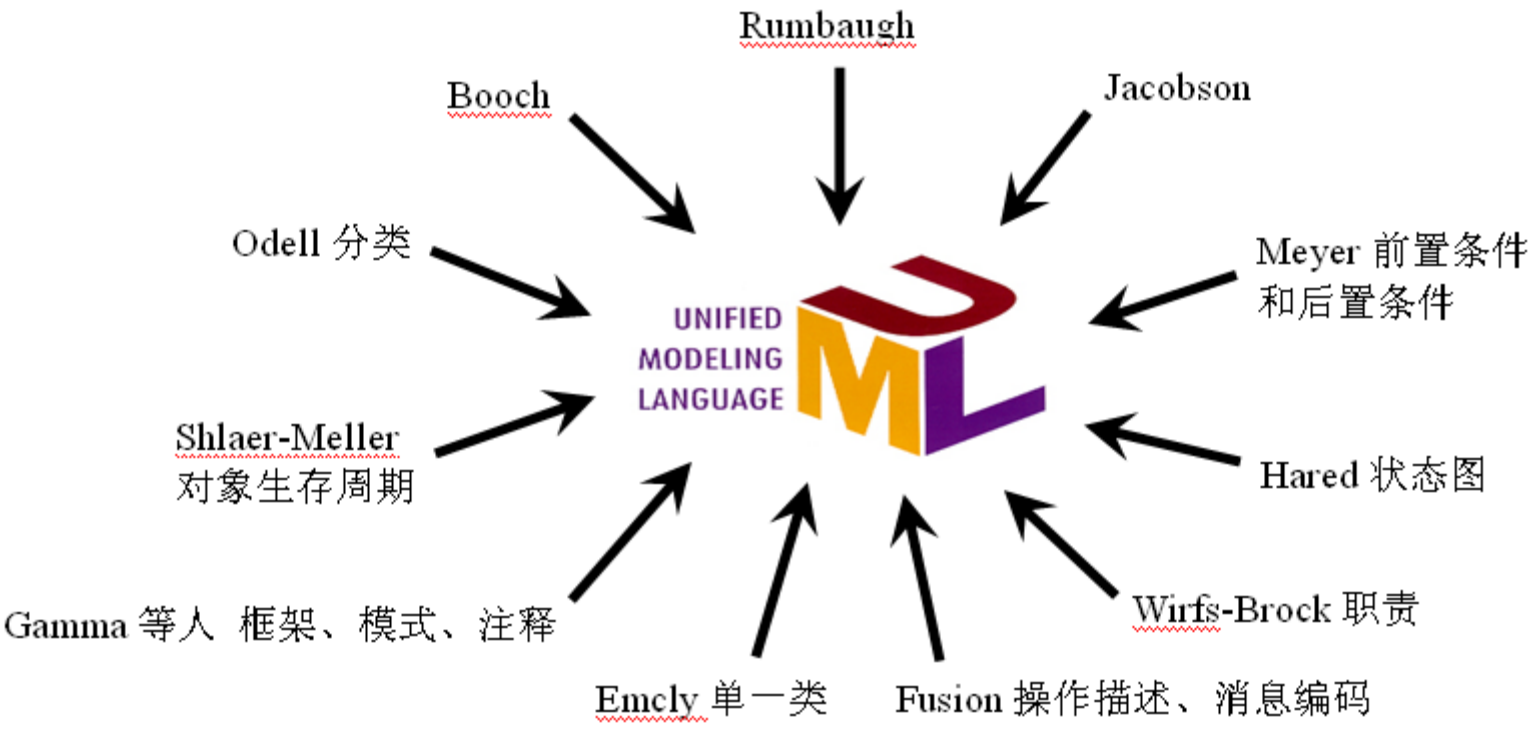
- 面向对象的建模语言很多，目前使用最广泛的是统一建模语言(UML，Unified Modeling Language)；
- 它将Booch、Rumbaugh和Jacobson等各自独立的OOA和OOD方法中最优秀的特色组合成一个统一的方法。

UML的产生和发展

UML(Unified Modeling Language)的概念于1996年由面向对象方法领域的三位著名专家Grady Booch，James Rumbaugh和Ivar Jacobson提出的。

- 1996年6月和10月分别发布了UML0.9, UML0.91
- 1997年1月，UML1.0被提交给对象管理组织OMG
- 1997年9月，提交UML1.1，1997年11月被OMG采纳作为基于面向对象技术的标准建模语言
- 1998、2000、2001、2003、2005年分别发布了UML1.2、UML1.3、UML1.4、 UML1.5、 UML2.0
- 2011年发布了UML2.4，UML2.4.1
- 2013年发布了UML2.5

UML的产生和发展



UML 吸收了许多面向对象方法的优点

UML的特点

(1) 统一标准

UML不仅统一了Booch、OMT和OOSE等方法中的基本概念，还吸取了面向对象技术领域中其他流派的长处，其中也包括非OO方法的影响。已经成为OMG的标准。

(2) 面向对象

UML支持面向对象技术的主要概念，它提供了一批基本的表示模型元素的图形和方法，能简洁明了地表达面向对象的各种概念和模型元素。

UML的特点

(3) 可视化，表达能力强大

UML是一种图形化语言，用UML的模型图形能清晰地表示系统的逻辑模型或实现模型。UML还提供了语言的扩展机制，用户可以根据需要增加定义自己的构造型、标记值和约束等。

(4) 独立于过程

UML是系统建模的语言，不依赖特定的开发过程。

UML的特点

(5) 容易掌握使用

UML概念明确，建模表示法简洁明了，图形结构清晰，容易掌握使用。

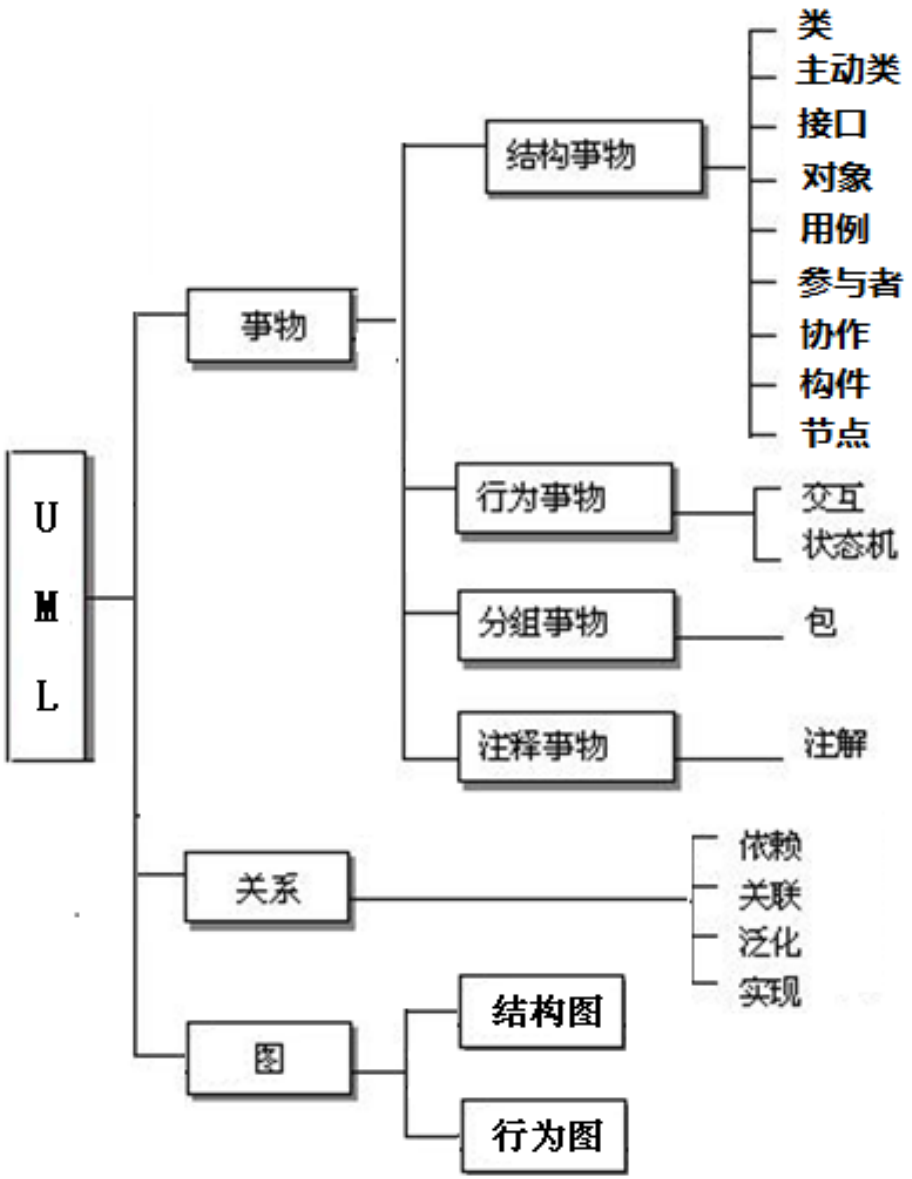
(6)与编程语言的关系

支持UML的一些CASE工具（如Rose）可以根据UML所建立的系统模型自动产生Java、C++ 等代码框架。

UML的基本模型

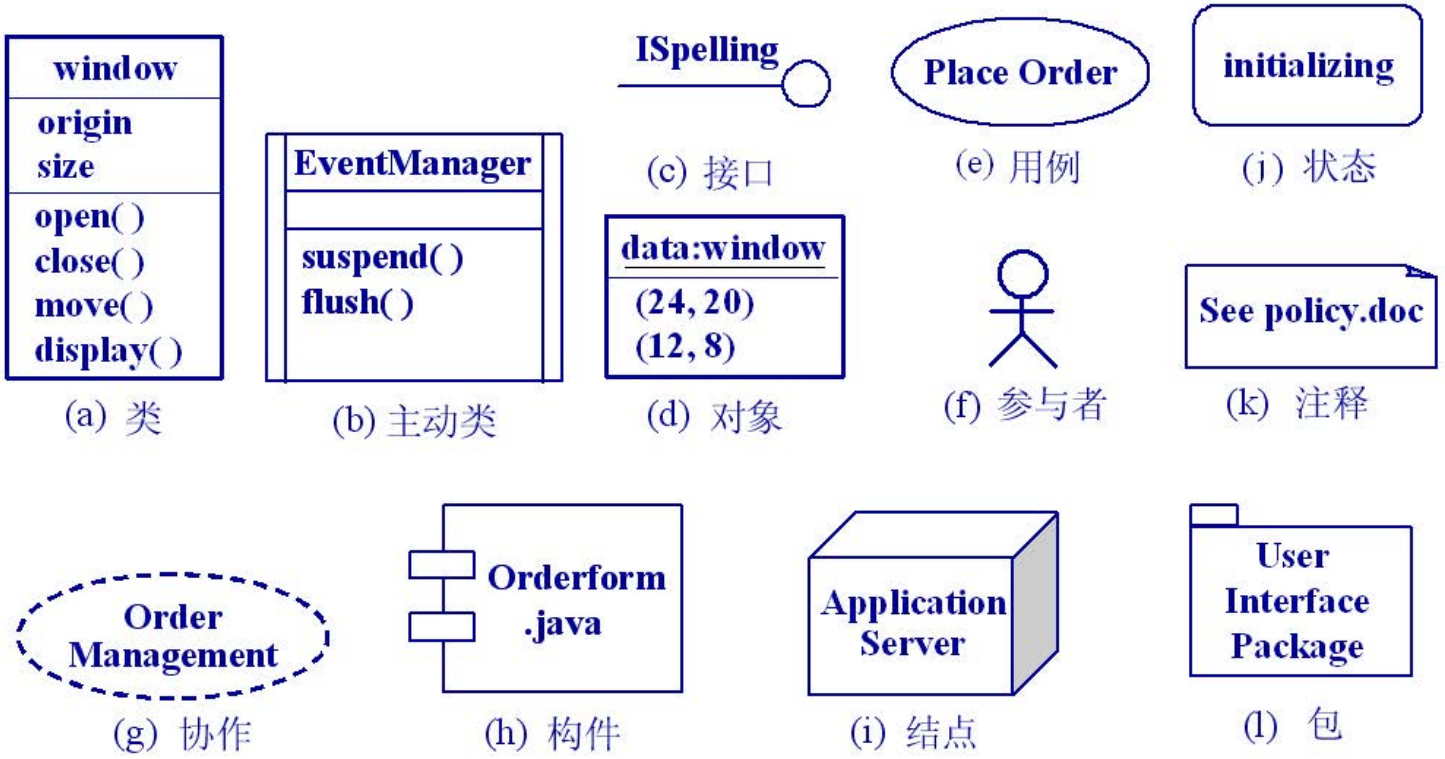
- UML符号为开发者或开发工具使用这些图形符号和文本语法为系统建模提供了标准。
- 这些图形符号和文字所表达的是应用级的模型，在语义上它是UML元模型的实例。
- UML模型由事物、关系和图组成 。

UML的基本模型


U
M
L
事物
结构事物
类
主动类
接口
对象
用例
参与者
协作
构件
节点
行为事物
交互
状态机
分组事物
包
注释事物
注解
关系
依赖
关联
泛化
实现
图
结构图
行为图

5.3 UML的事物

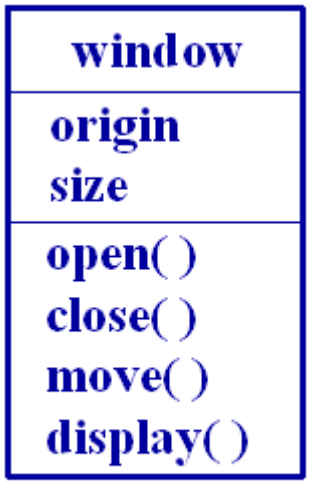
- 事物是对模型中最具代表性成分的抽象，在UML中，可以分为结构事物、行为事物、分组事物和注释事物4类。



结构事物

- 结构事物是UML模型的静态部分，主要用来描述概念的或物理的元素，包括类、主动类、接口、对象、用例、参与者、协作、构件和节点等。

（1）类（class）—— 类用带有类名、属性和操作的矩形框来表示。



(a) 类

结构事物

（2）主动类（active class）—— 主动类的实例应具有一个或多个进程或线程，能够启动控制活动。

(b) 主动类

结构事物

（3）接口（interface）—— 描述了一个类或构件的一组外部可用的服务（操作）集。

接口定义的是一组操作的描述，而不是操作的实现。

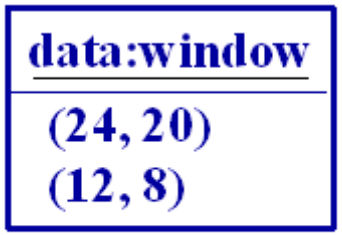
一般将接口画成从实现它的类或构件引出的圆圈，接口体现了使用与实现分离的原则。

(c) 接口

结构事物

（4）对象（object）—— 对象是类的实例，其名字下边加下划线，对象的属性值需明确给出。

（5）用例（use case）—— 也称用况，用于表示系统想要实现的行为，即描述一组动作序列（即场景）。而系统执行这组动作后将产生一个对特定参与者有价值的结果。



(d) 对象

(e) 用例

结构事物

（6）参与者（actor）—— 也称角色，是指与系统有信息交互关系的人、软件系统或硬件设备，在图形上用简化的小木头人表示。

(f) 参与者

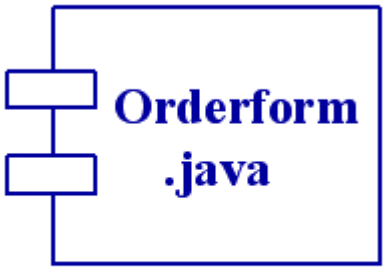
（7）协作（collaboration）—— 用例仅描述要实现的行为，不描述这些行为的实现。这种实现用协作描述。

协作定义交互，描述一组角色实体和其他实体如何通过协同工作来完成一个功能或行为。类可以参与几个协作。

(g) 协作

结构事物

（8）构件（component）—— 也称组件，是系统中物理的、可替代的部件。它通常是描述一些逻辑元素的物理包。



(h) 构件

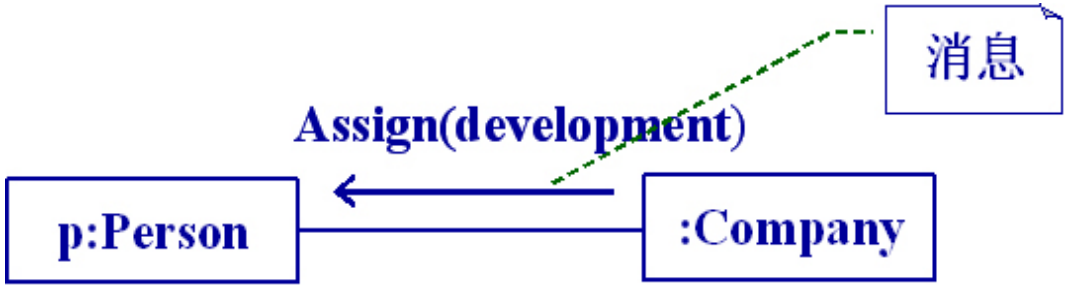
（9）节点（node）—— 是在运行时存在的物理元素。它代表一种可计算的资源，通常具有一定的记忆能力和处理能力。

(i) 节点

行为事物

- 行为事物是UML模型的动态部分，包括两类：

（1）交互（interaction）—— 交互由在特定的上下文环境中共同完成一定任务的一组对象之间传递的消息组成。如图所示。交互涉及的元素包括消息、动作序列（由一个消息所引起的行为）和链（对象间的连接）。



对象之间的交互

行为事物

（2）状态机（state machine）—— 描述了一个对象或一个交互在生存周期内响应事件所经历的状态序列，单个类或者一组类之间协作的行为都可以用状态机来描述。

状态机涉及到状态、变迁和活动，其中状态用圆角矩形来表示。

(j) 状态

分组事物

- 分组事物是UML模型的组织部分。它的作用是为了降低模型复杂性。
- UML中的分组事物是包（package）。
- 包是把模型元素组织成组的机制，结构事物、行为事物甚至其他分组事物都可以放进包内。

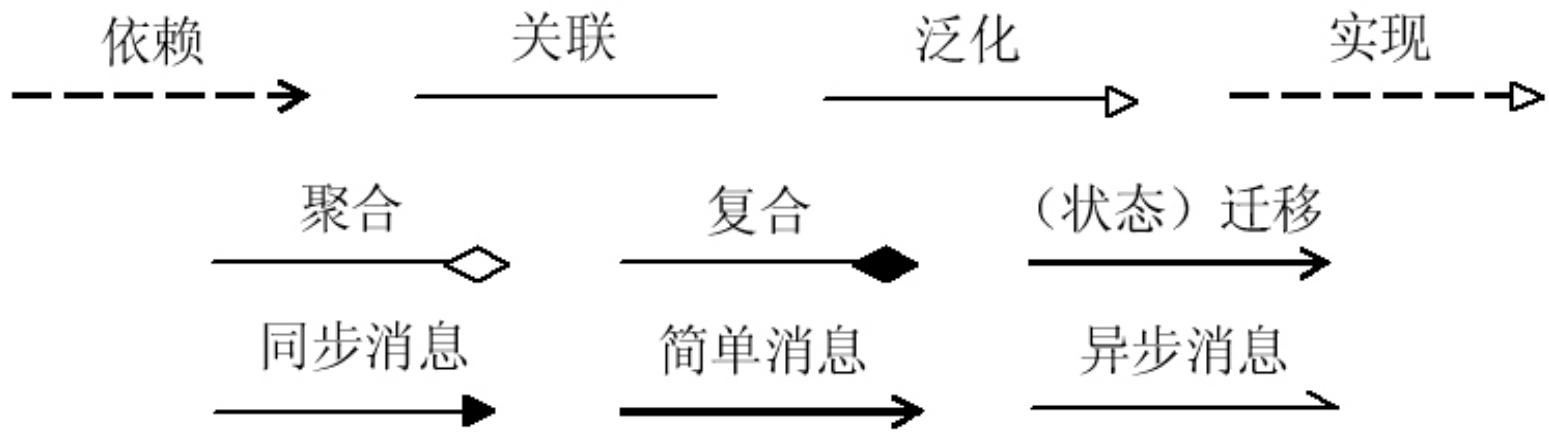
(1) 包

注释事物

- 注释事物是UML模型的解释部分，它们用来描述和标注模型的任何元素。
- 通常可以用注释修饰带有约束或者解释的图。

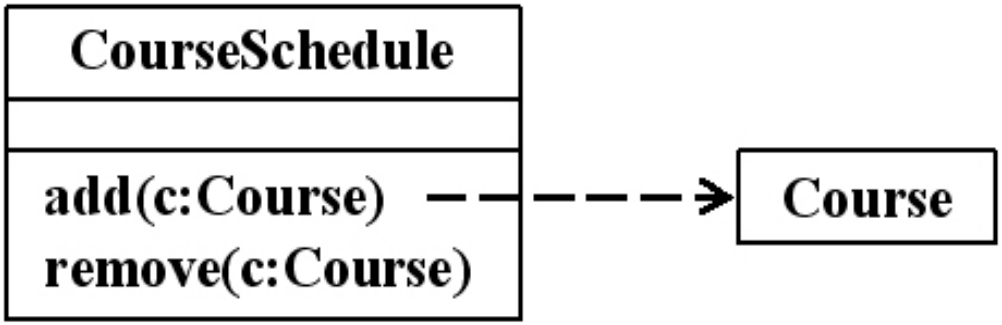
(k) 注释

5.4 UML的关系



依赖关系

- 依赖(Dependency)是两个事物之间的语义关系，其中一个事物发生变化会影响到另一个事物的语义，它用一个虚线箭头表示。
- 虚线箭头的方向从源事物指向目标事物，表示源事物依赖于目标事物。



依赖关系

依赖的种类

依赖	功　　能	关键字
访问	源包（如用户界面包）被赋予了可访问目标包（如业务对象包）的权限，并可引用目标包中的元素。	access
绑定	目标类是模板类（如<数据类型参数化为 T>栈），源类是将指定值代换模板参数而生成的特定类（如<实参为 int>栈）。	Bind
调用	强调源类中的操作(如矩形类的 draw)调用了定义在目标类中的操作(如像素点类的 draw)。	call
友元	目标类（如二叉树）视源类（如 Iterator）为其友元，允许源类访问目标类的所有私有成员。（UML2.0 中没有）	friend
派生	源事物（如年龄）可以从目标事物（如出生年月）通过计算导出。	derive
创建	源类（如链表类）可创建目标类（如链表结点类）的实例。	create
细化	同一模型元素的不同详细程度或不同语义层次的规格说明，源（如详细配置图）比目标（如概要配置图）更为详细。	refine
实例化	强调源类的实例（如链表）创建了目标类的实例（如链表结点）,而且还做了初始化和满足约束的工作。	instantiate

依赖关系

依赖的种类

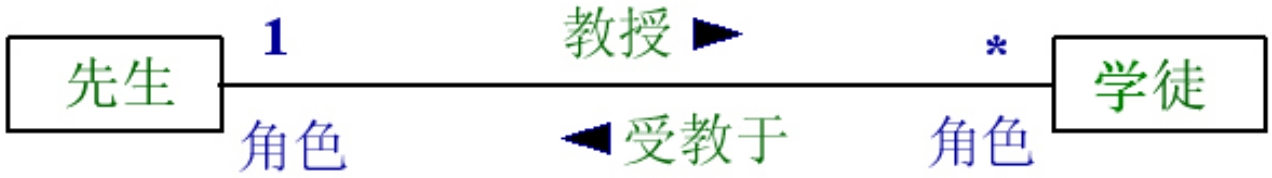
依赖	功　　能	关键字
允许	允许源事物（如电梯调度器）访问或处理目标事物（如中断向量表）的内容。	**permit**
实现	一个规格说明（如栈的接口）和其具体实现（该栈的类实现）之间的映射关系	**realize**
发送	一个信号发送者（如电梯控制器）与信号接收者（如楼层管理器）之间的关系。	**send**
替换	表明源类可以支持目标类的接口，并可以在类型声明为目标类的地方取代目标类。继承性和多态性都可支持这种关系。	**substitute**
使用	强调源事物（如电梯调度器）想要正确地履行职责（包括调用、创建、实例化、发送等），则要求目标事物（如决策表）存在。	**use(s)**
追踪	它连接两个模型元素，表明目标是源的历史上的前驱。如交互和协作就是从用例导出的。	**trace**

关联关系

- 关联(association)是一种结构关系，它描述了两个或多个类的实例之间的连接关系，是一种特殊的依赖。
- 关联分为普通关联、限定关联、关联类，以及聚合与复合。

关联关系——普通关联

- 普通关联是最常见的关联关系，只要类与类之间存在连接关系就可以用普通关联表示。普通关联又分为二元关联和多元关联。
- 二元关联描述两个类之间的关联，用两个类之间的一条直线来表示，直线上可写上关联名。



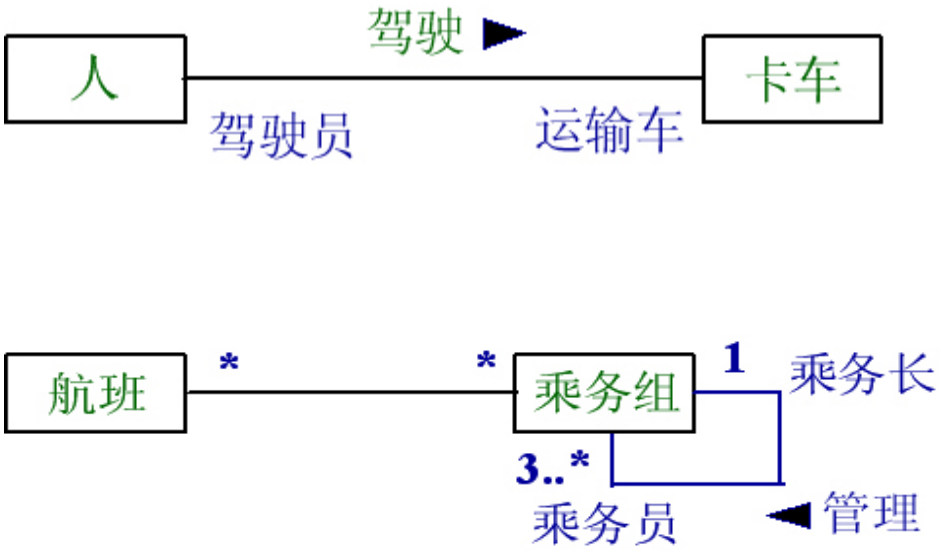
关联关系——普通关联

- 多重性（multiplicity）：多重性表明在一个关联的两端连接的类实例个数的对应关系，即一端的类的多少个实例对象可以与另一端的类的一个实例相关。
- 如果图中没有明确标出关联的多重性，则默认的多重性为1。

1	—— 1 个实例
0..1	—— 0 到 1 个实例
0..* 或 *	—— 0 到多个实例
1+ 或 1..*	—— 1 到多个实例

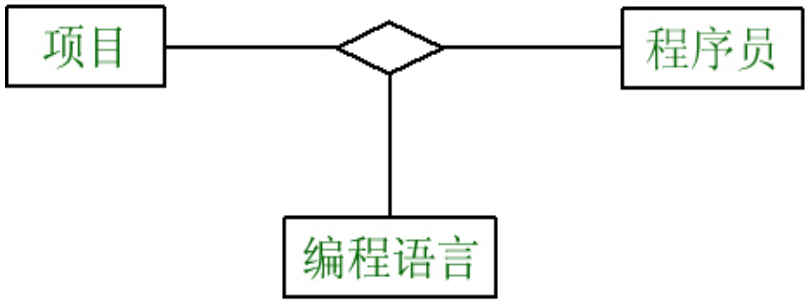
关联关系——普通关联

- 角色：关联端点上还可以附加角色名，表示类的实例在这个关联中扮演的角色。UML还允许一个类与它自身关联。



关联关系——普通关联

- 多元关联：多元关联是指3个或3个以上类之间的关联。
- 多元关联由一个菱形，以及由菱形引出的通向各个相关类的直线组成，关联名可标在菱形的旁边，在关联的端点也可以标上多重性等信息。

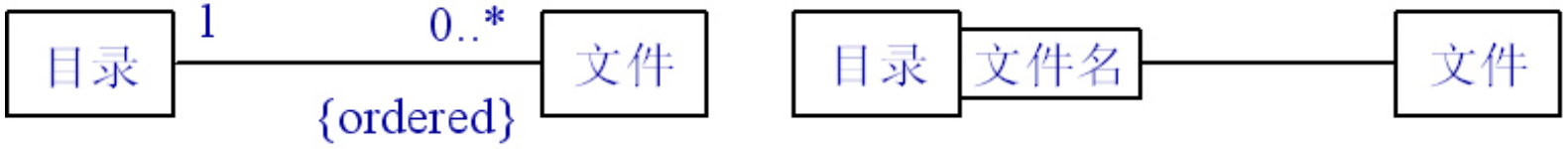


关联的链

程序员	编程语言	项目
宫力	C++	CAD
周斌	Java	网站
陈健	C++	CAD
陈健	C++	MIS
.....		

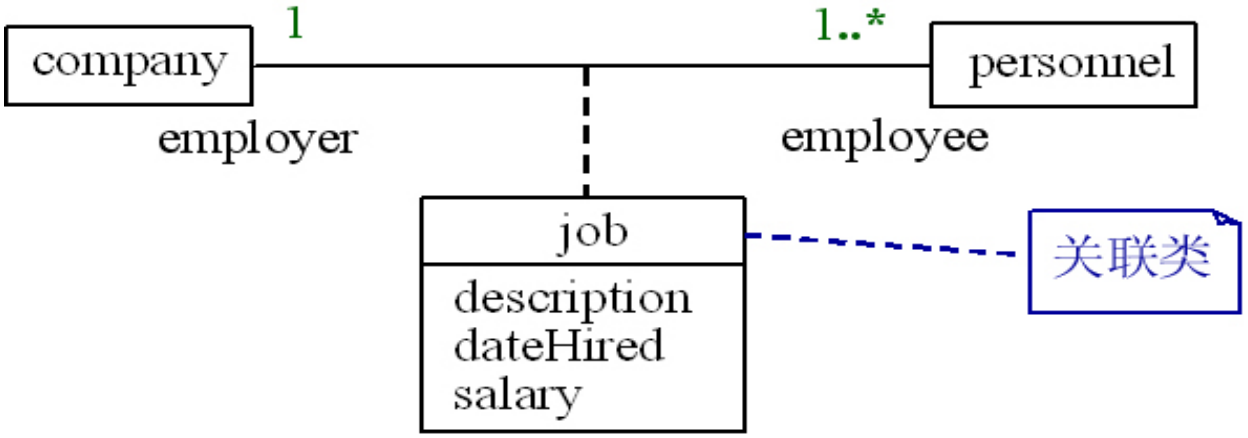
关联关系——限定关联

- 限定关联通常用在一对多或多对多的关联关系中，可以把模型中的多重性从一对多变成一对一，或将多对多简化成多对一。
- 在类图中把限定词（qualifier）放在关联关系末端的一个小方框内。



关联关系——关联类

- 在关联关系比较简单的情况下，关联关系的语义用关联关系的名字来概括。
- 但在某些情况下，需要对关联关系的语义做详细的定义、存储和访问，为此可以建立关联类（association class），用来描述关联的属性。
- 关联中的每个链与关联类的一个实例相联系。关联类通过一条虚线与关联连接。



关联关系——聚合

- 聚合（Aggregation）也称为聚集，是一种特殊的关联。它描述了整体和部分之间的结构关系。
- 两种特殊的聚合关系：共享聚合（shared aggregation）和复合聚合（composition aggregation）。
- 如果在聚合关系中处于部分方的实例可同时参与多个处于整体方实例的构成，则该聚合称为共享聚合。

共享聚合

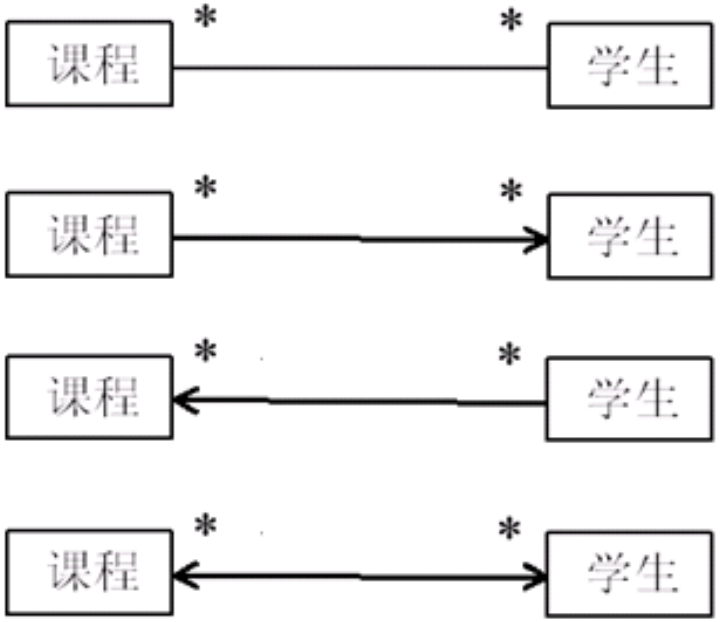
关联关系——聚合

- 如果部分类完全隶属于整体类，部分类需要与整体类共存，一旦整体类不存在了，则部分类也会随之消失，或失去存在价值，则这种聚合称为复合聚合。

复合聚合

关联关系——导航

- 导航（navigability）是关联关系的一种特性，它通过在关联的一个端点上加箭头来表示导航的方向。



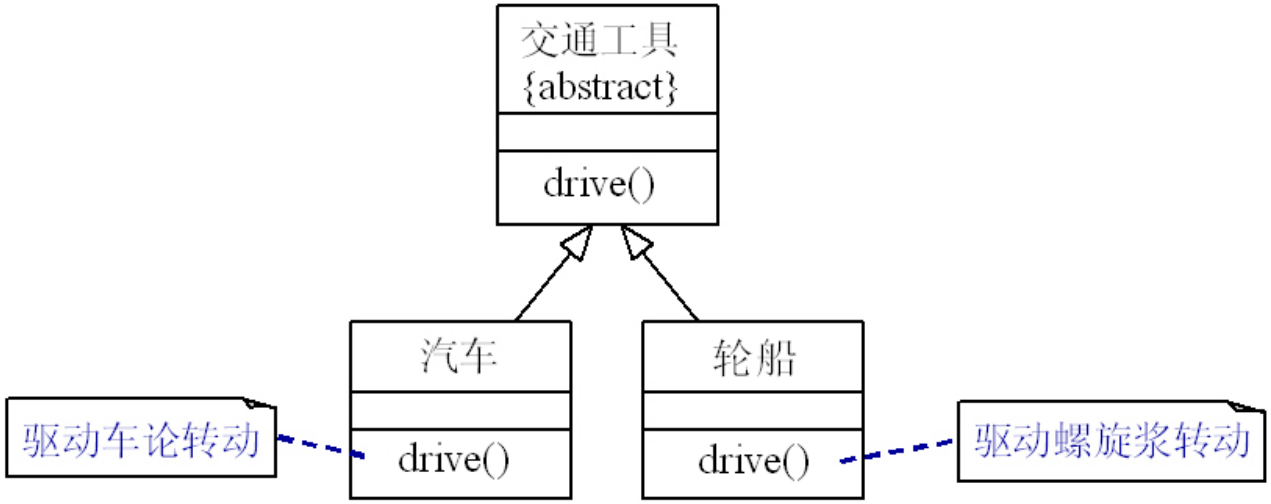
导航

泛化关系

- 泛化(generalization)关系就是一般类和特殊类之间的继承关系。
- 在UML中，一般类亦称泛化类，特殊类亦称特化类。
- 泛化针对类型而不针对实例，因为一个类可以继承另一个类，但一个对象不能继承另一个对象。
- 泛化可进一步划分成普通泛化和受限泛化两类。

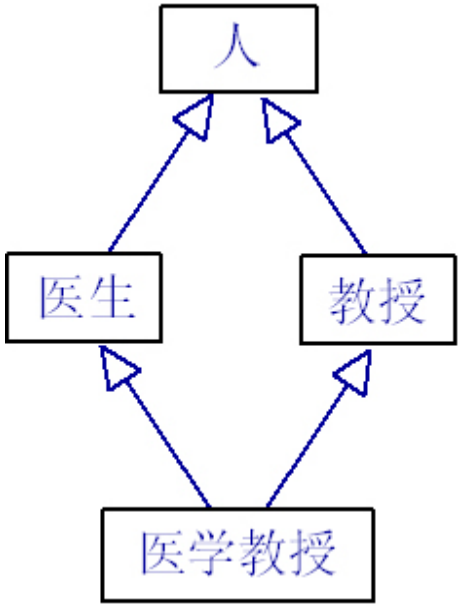
泛化关系——普通泛化

- 普通泛化与前面讲过的继承基本相同。但在泛化关系中常遇到抽象类。
- 一般称没有具体对象的类为抽象类。抽象类通常作为父类，用于描述其他类（子类）的公共属性和行为。



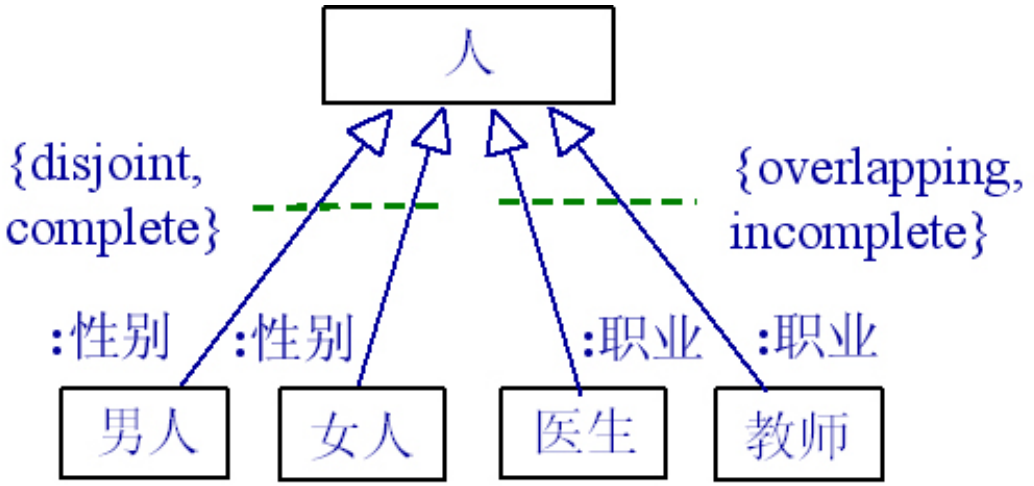
泛化关系——普通泛化

- 普通泛化可以分为多重继承和单继承。多重继承是指一个子类可同时继承多个上层父类。



泛化关系——受限泛化

- 受限泛化关系是指泛化具有约束条件。
- 一般有4种约束：交叠（overlapping）、不相交（disjoint）、完全（complete）和不完全（incomplate）。

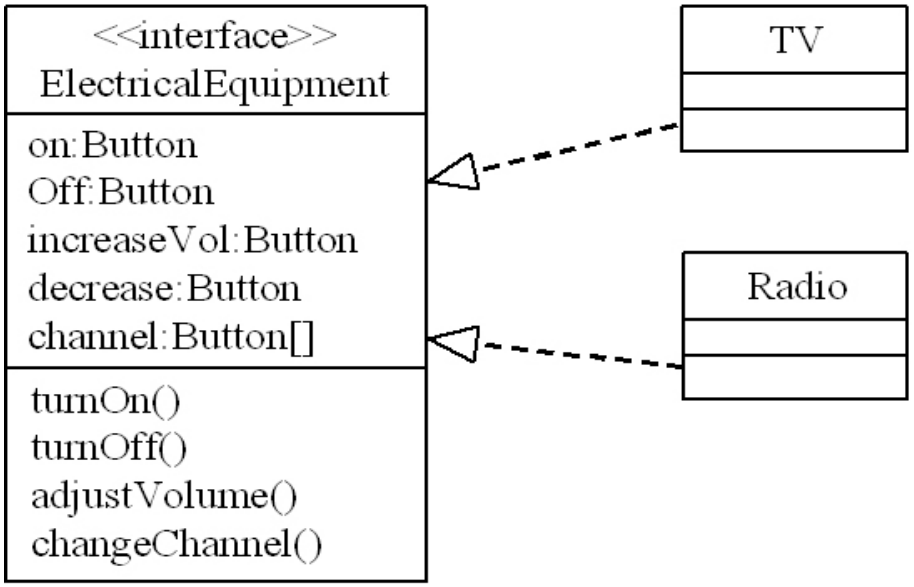


实现关系

- 实现(implement)是泛化关系和依赖关系的结合，也是类之间的语义关系，通常在以下两种情况出现实现关系：

(1)接口和实现它们的类或构件之间；

(2)用例和实现它们的协作之间。



5.5 UML的图

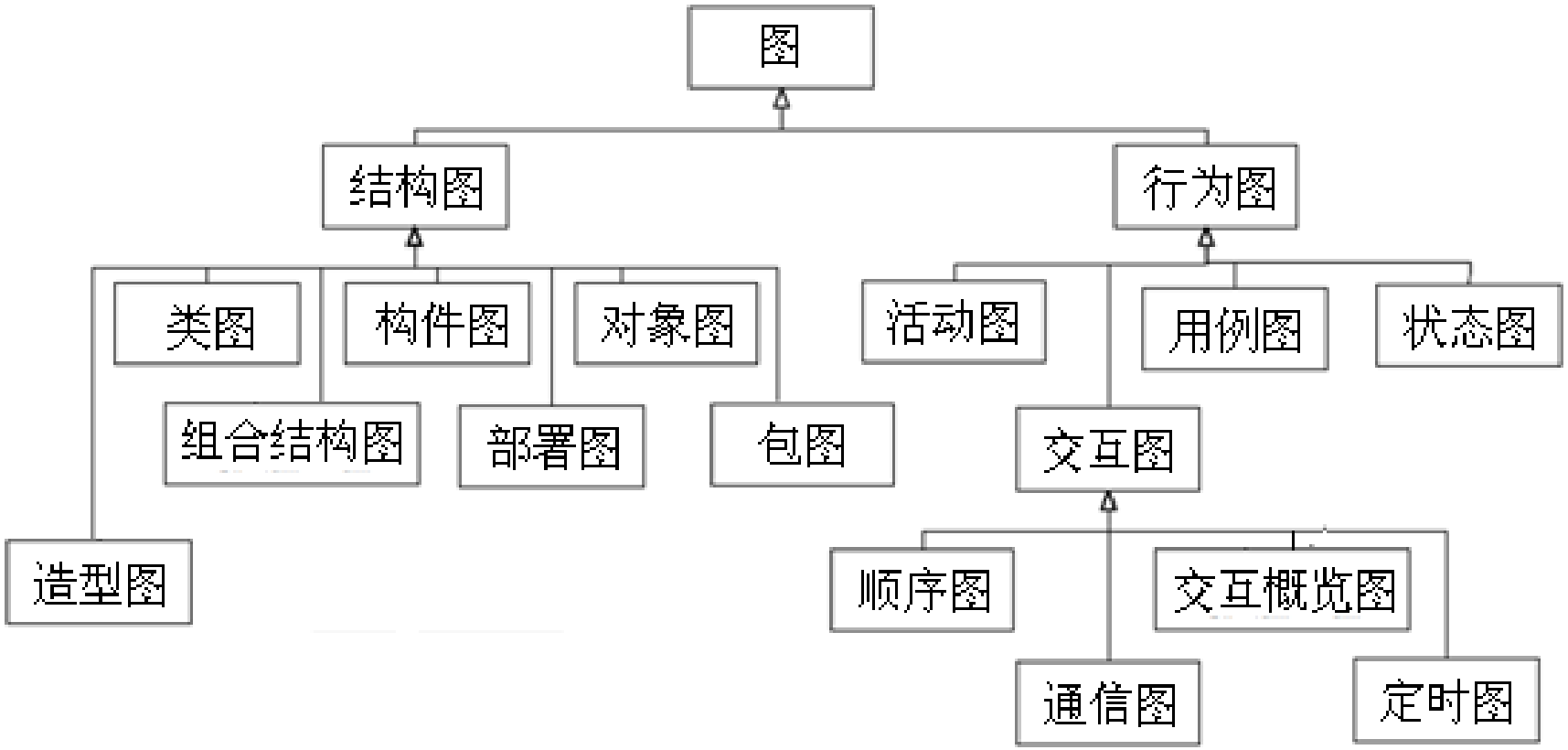


图
结构图
行为图
类图
构件图
对象图
活动图
用例图
状态图
组合结构图
部署图
包图
交互图
造型图
顺序图
交互概览图
通信图
定时图

用例图

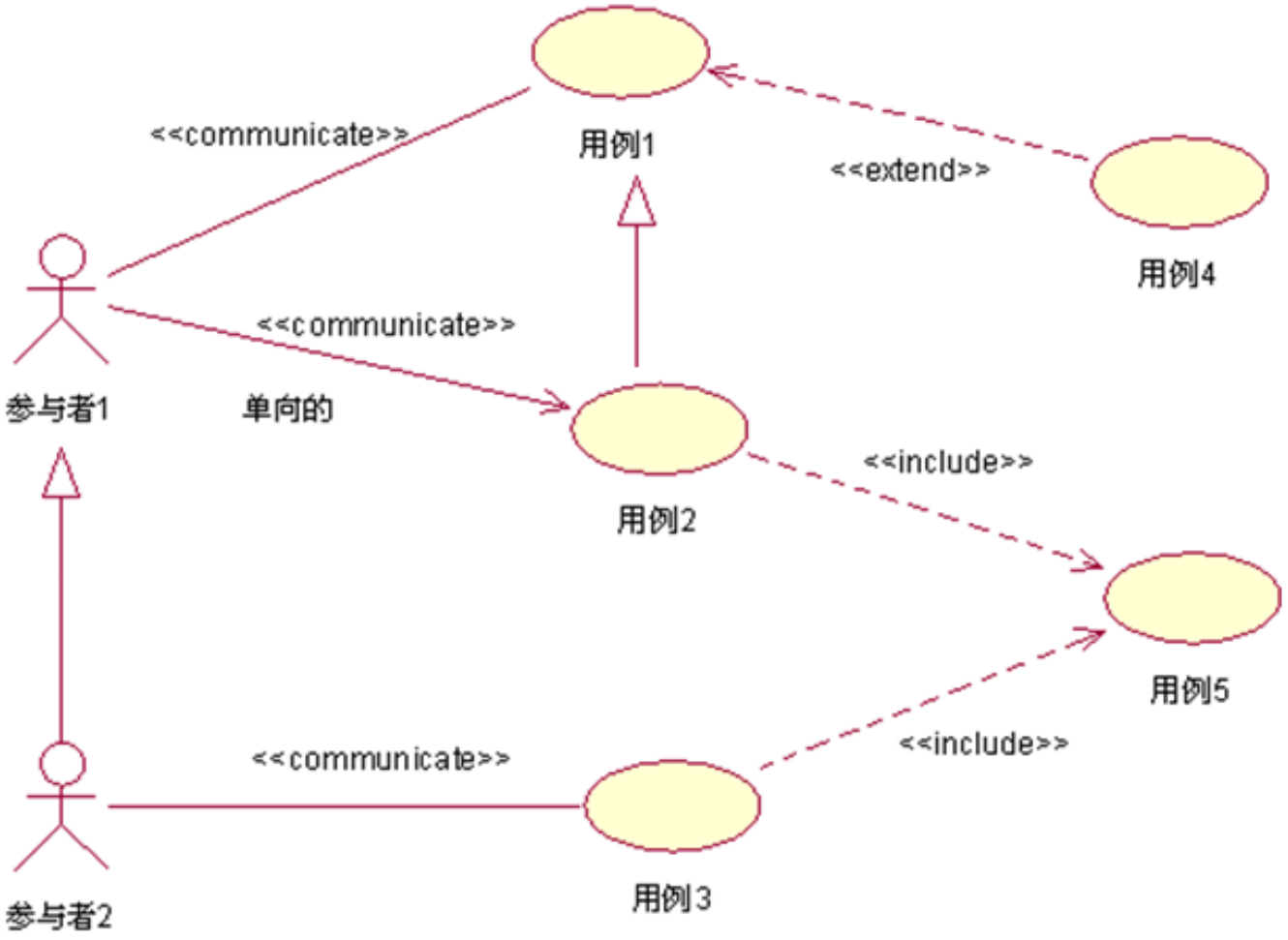
1. 用例模型

用例模型描述的是外部执行者(actor)所理解的系统功能。用例模型用于需求分析阶段，它的建立是系统开发者和用户反复讨论的结果，描述了开发者和用户对需求规格达成的共识。

在UML中，一个用例模型由若干个用例图来描述，用例图的主要元素是用例和执行者。

用例图是包括执行者、由系统边界（一个矩形）封闭的一组用例，执行者和用例之间的关联、用例间关系以及执行者的泛化的图。

用例图



用例图的建模元素

用例图

2. 用例之间的关系

用例之间可以有泛化、扩展、使用（包含）三种关系。

(1) 泛化关系

- 用例泛化是指一个用例可以被特别列举为一个或多个子用例。

用例图

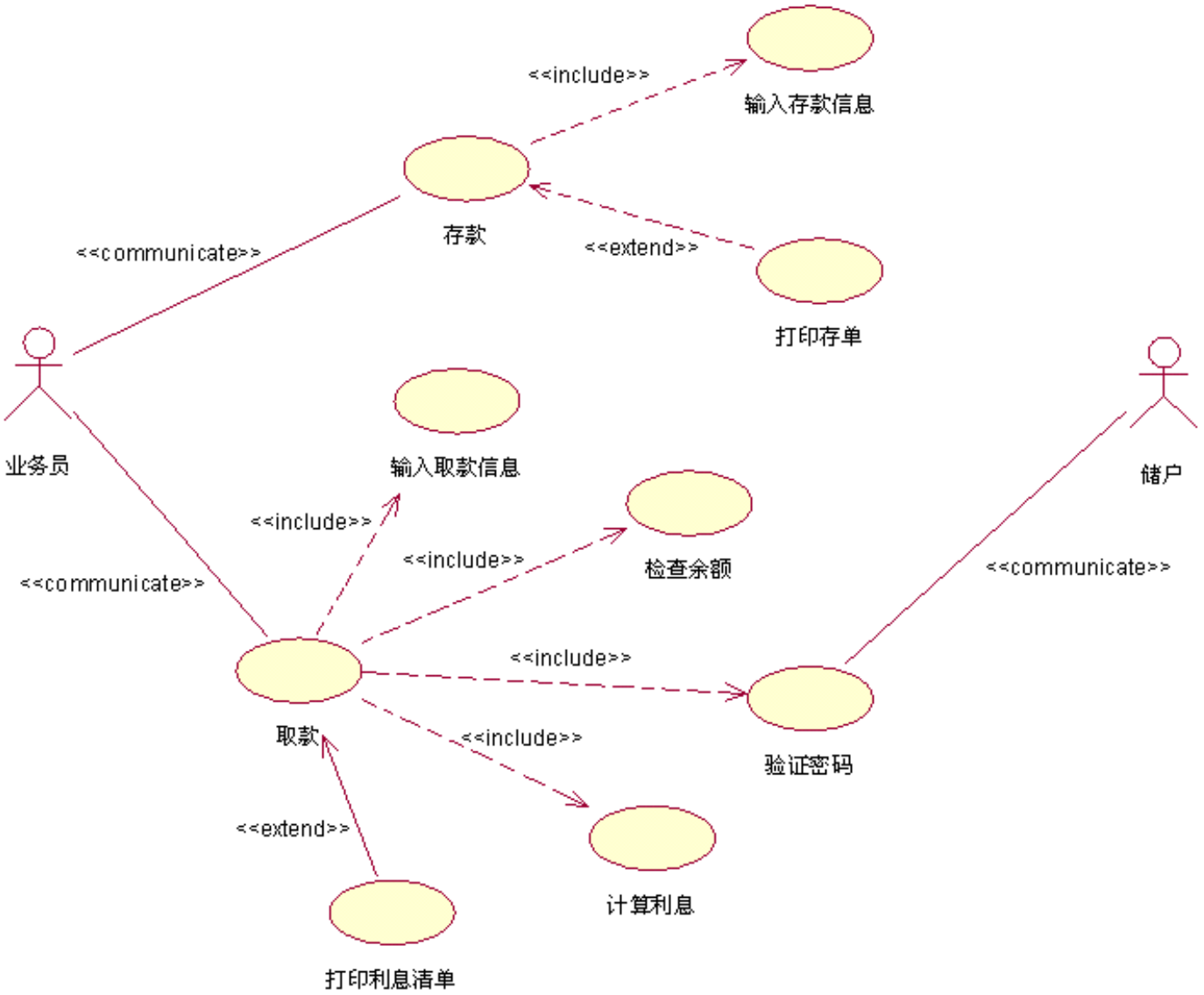
(2) 扩展关系

- 向一个用例中加入一些新的动作后构成了另一个用例，这两个用例之间的关系就是扩展关系，后者通过继承前者的一些行为得来，通常把后者称为扩展用例。

(3) 使用（包含）关系

- 当一个用例使用另一个用例时，这两个用例之间就构成了使用关系。
- 当有一大块相似的动作存在于几个用例，又不想重复描述该动作，将重复的部分分离为一个用例，两用例间关系称为使用关系。

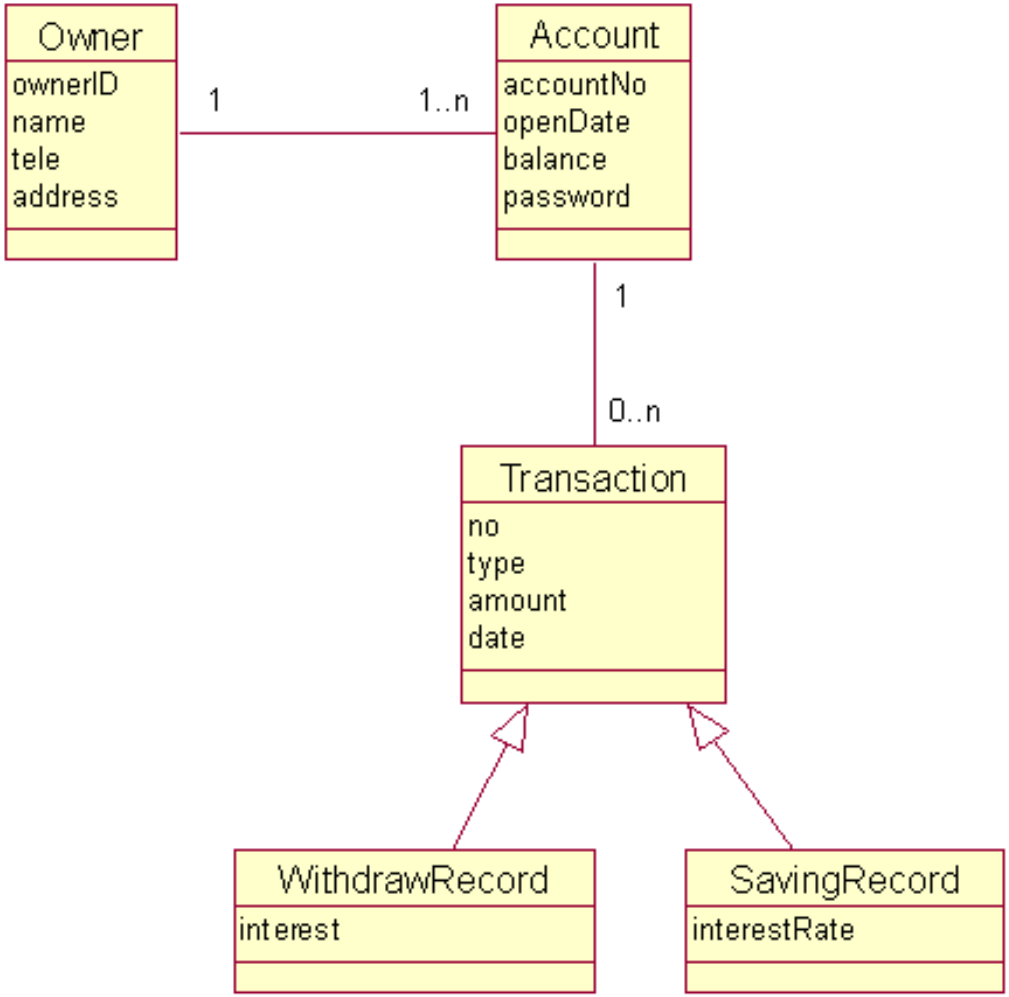
银行储蓄系统的用例图


<<include>>
输入存款信息
存款
<<communicate>>
<<extend>>
打印存单
业务员
输入取款信息
储户
<<include>>
<<communicate>>
<<include>>
检查余额
<<communicate>>
<<include>>
取款
<<include>>
验证密码
<<extend>>
计算利息
打印利息清单

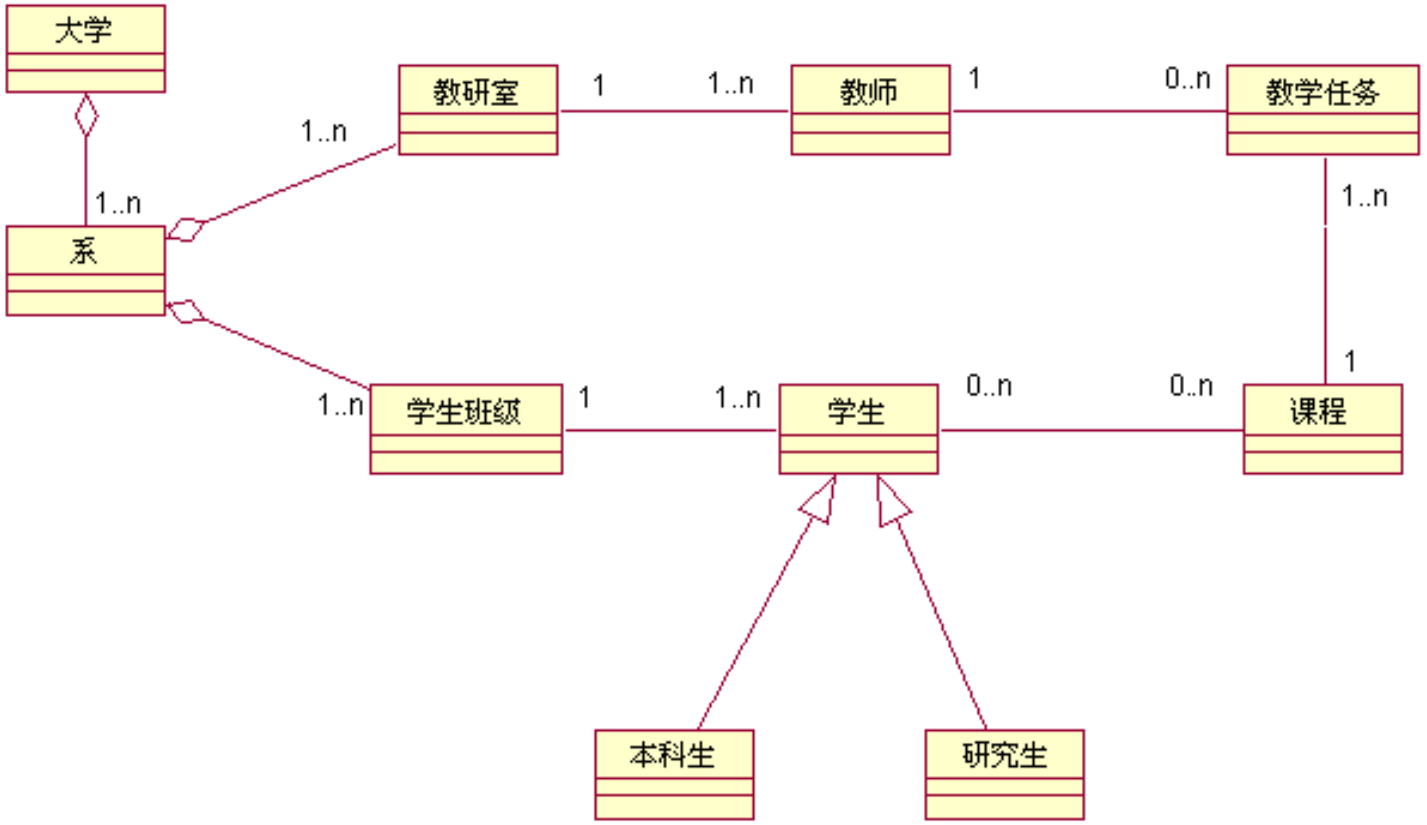
类图

- 类图描述类和类与类之间的静态关系，它是从静态角度表示系统的，因此类图属于一种静态模型。类图是构建其他图的基础，没有类图就没有状态图、协作图等其他图，也就无法表示系统其他方面的特性。
- 类图显示了类（及其接口）、类的内部结构以及与其他类的联系。联系是指类元之间的联系，在类的建模中可以使用关联、聚合和泛化（继承）关系。

银行储蓄系统的核心类图

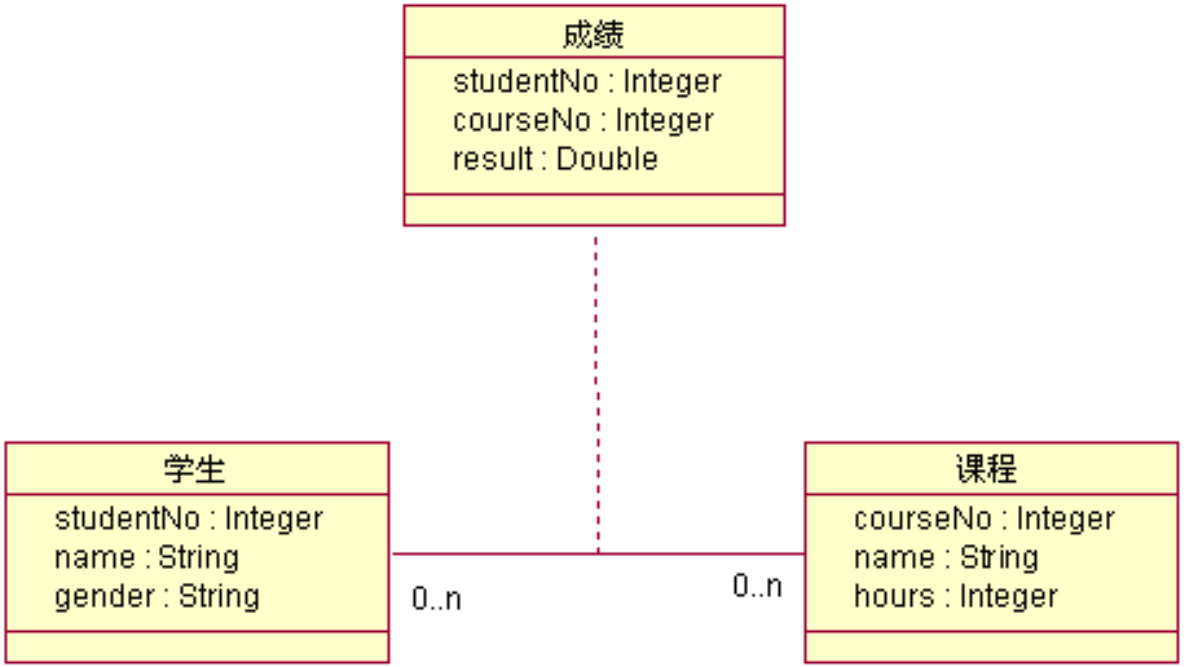


教学管理系统的类图



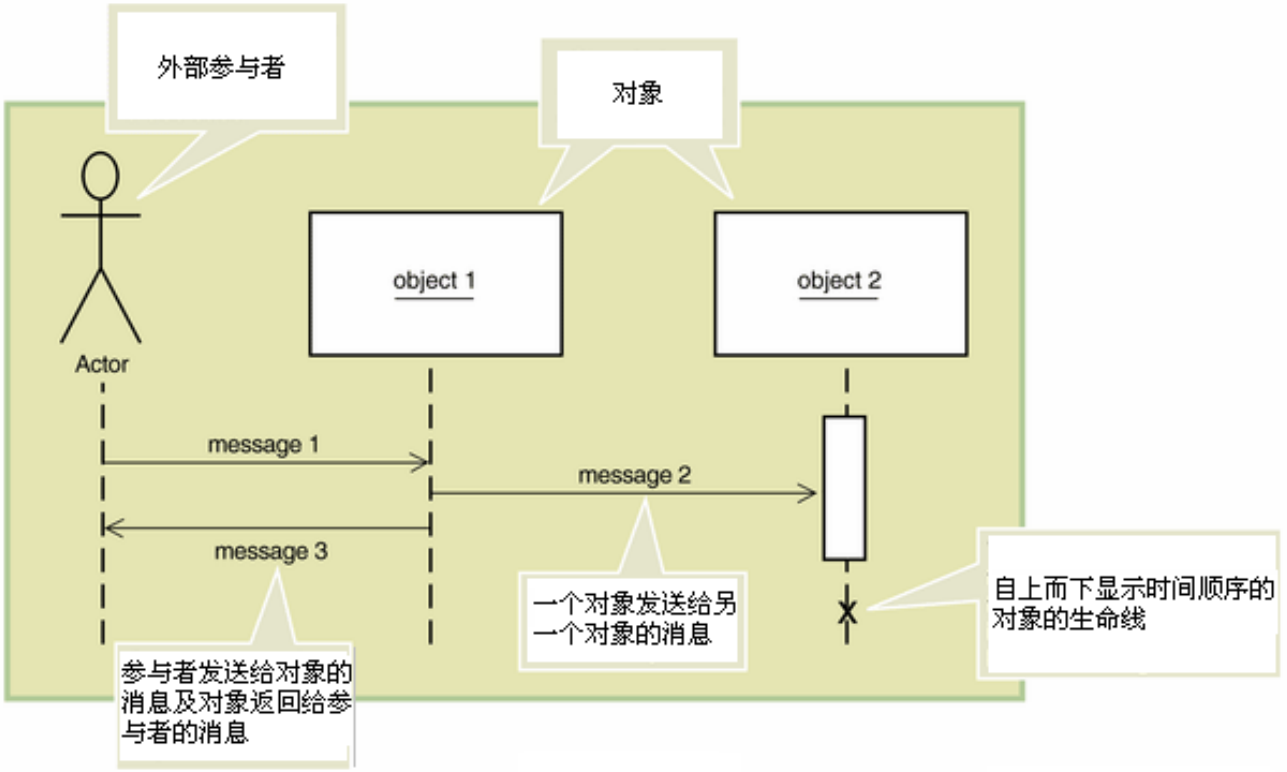
关联类

- 关联类是指表示其他类之间关联关系的类。当一个关联具有自己的属性并需要存储它们时，就需要用关联类建模。关联类用虚线连接在两个类之间的联系上。

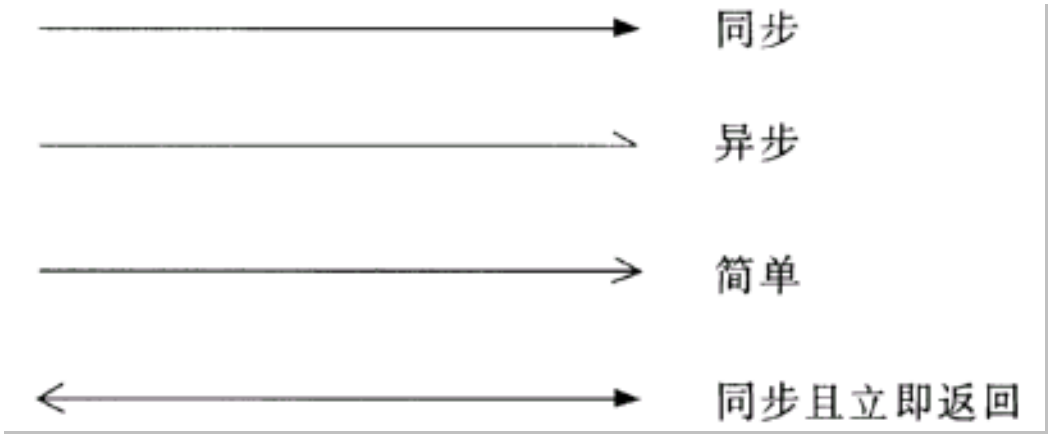


交互图

- UML中有两种类型的交互图：顺序图和协作图。
- 顺序图描述对象之间的动态交互关系，着重表现对象间消息传递的时间顺序。顺序图中的符号如下：



交互图

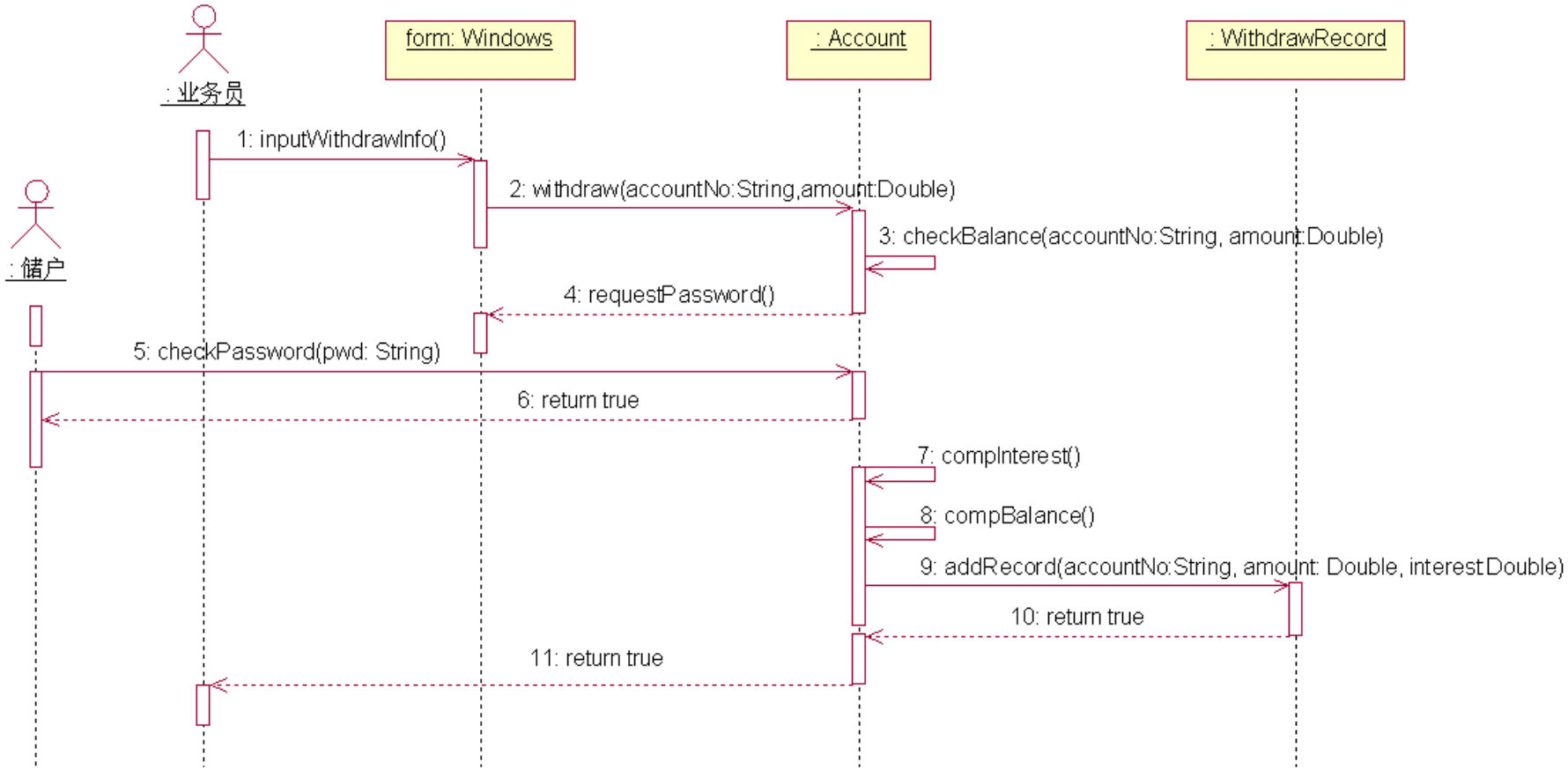


消息的类型

UML定义了三种消息:

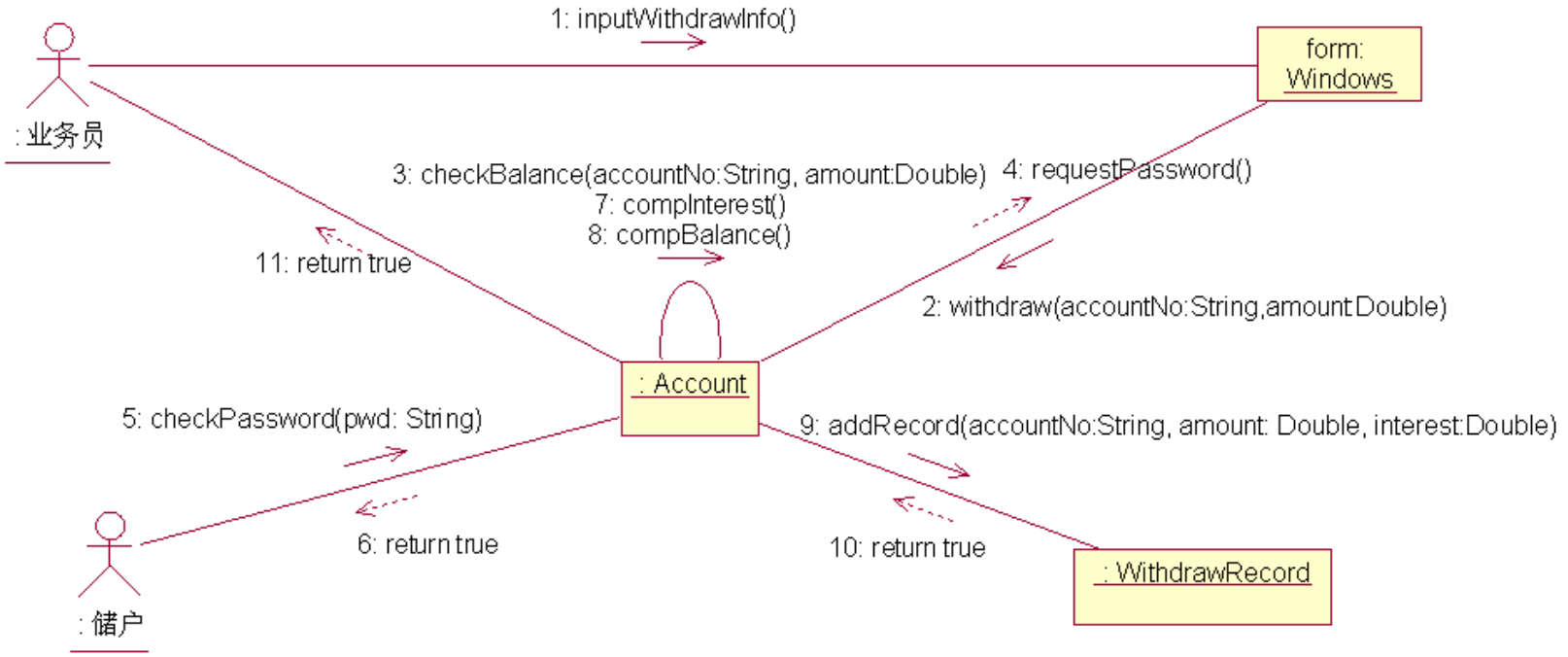
- 简单消息：表示简单的控制流，它只是表示控制从一个对象传给另一个对象，而没有描述通信的任何细节。
- 同步消息：表示嵌套的控制流，操作的调用是一种典型的同步消息。调用者发出消息后必须等待消息返回，只有当处理消息的操作执行完毕后，调用者才可以继续执行自己的操作。
- 异步消息：表示异步控制流，发送者发出消息后不用等待消息处理完就可以继续执行自己的操作。异步消息主要用于描述实时系统中的并发行为。

取款用例的顺序图


: 业务员
form: Windows
: Account
: WithdrawRecord
: 储户
1: inputWithdrawInfo()
2: withdraw(accountNo:String,amount:Double)
3: checkBalance(accountNo:String, amount:Double)
4: requestPassword()
5: checkPassword(pwd: String)
6: return true
7: compInterest()
8: compBalance()
9: addRecord(accountNo:String, amount: Double, interest:Double)
10: return true
11: return true

通信图

- 通信图是顺序图的一种变化形式，用于描述相互协作的对象间的交互关系和链接关系。



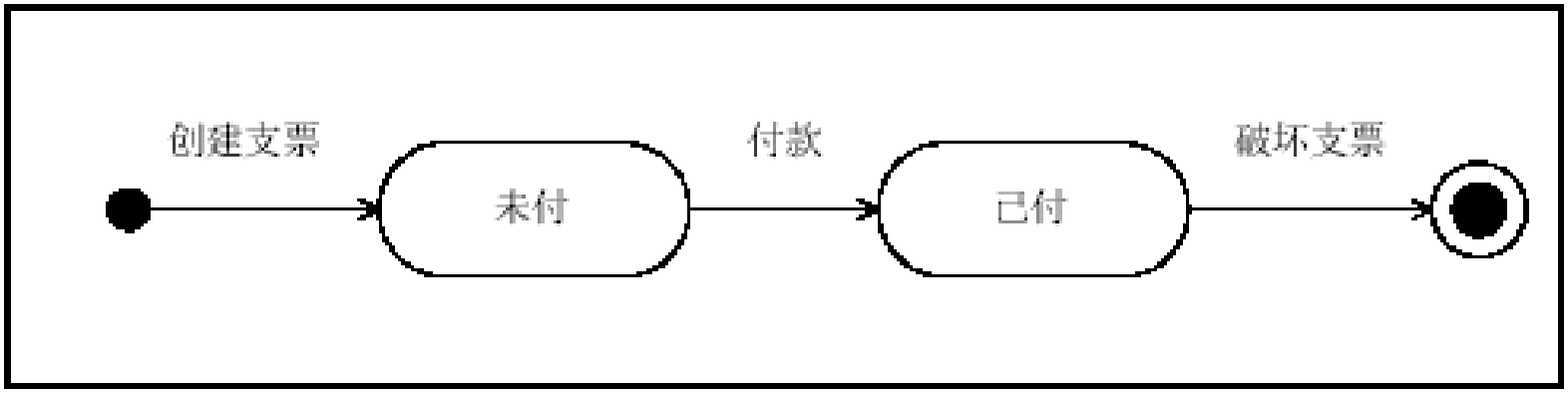
状态图

状态图描述一个特定对象的所有可能的状态以及引起状态转换的事件。大多数面向对象技术都用状态图表示单个对象在其生命期中的行为。一个状态图包括一系列状态、事件以及状态之间的转移。

1. 状态

- 所有对象都具有状态，状态是对象执行了一系列活动的结果。当某个事件发生后，对象的状态将发生变化。在状态图中定义的状态可能有：初态(初始状态)、终态(最终状态)、中间状态和复合状态。
- 在一张状态图中只能有一个初态，而终态则可以有多个。

状态图

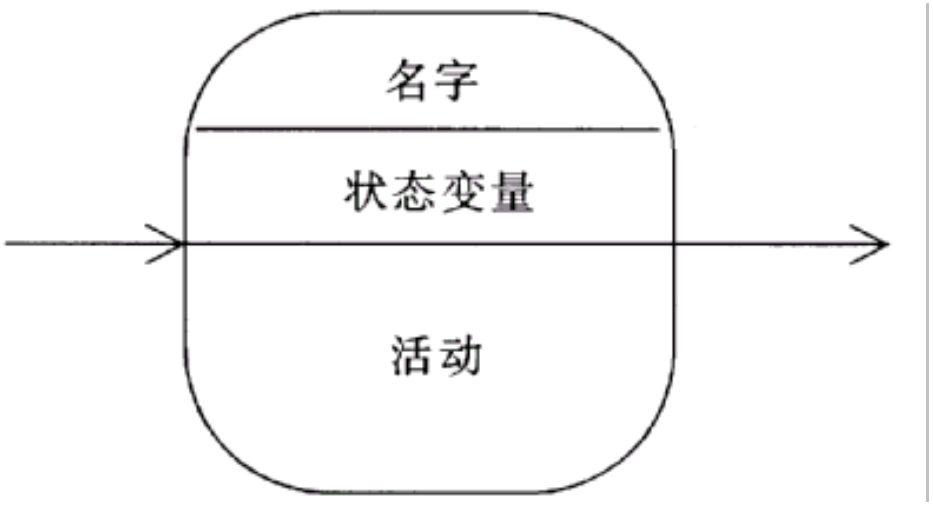


：支票对象的状态图。黑圆点代表支票对象的起点(对象刚创建)。黑圆点外加一个圆代表对象的终点(对象被删除)。状态间的箭头表示状态转移和引起状态改变的事件

支票对象的状态图

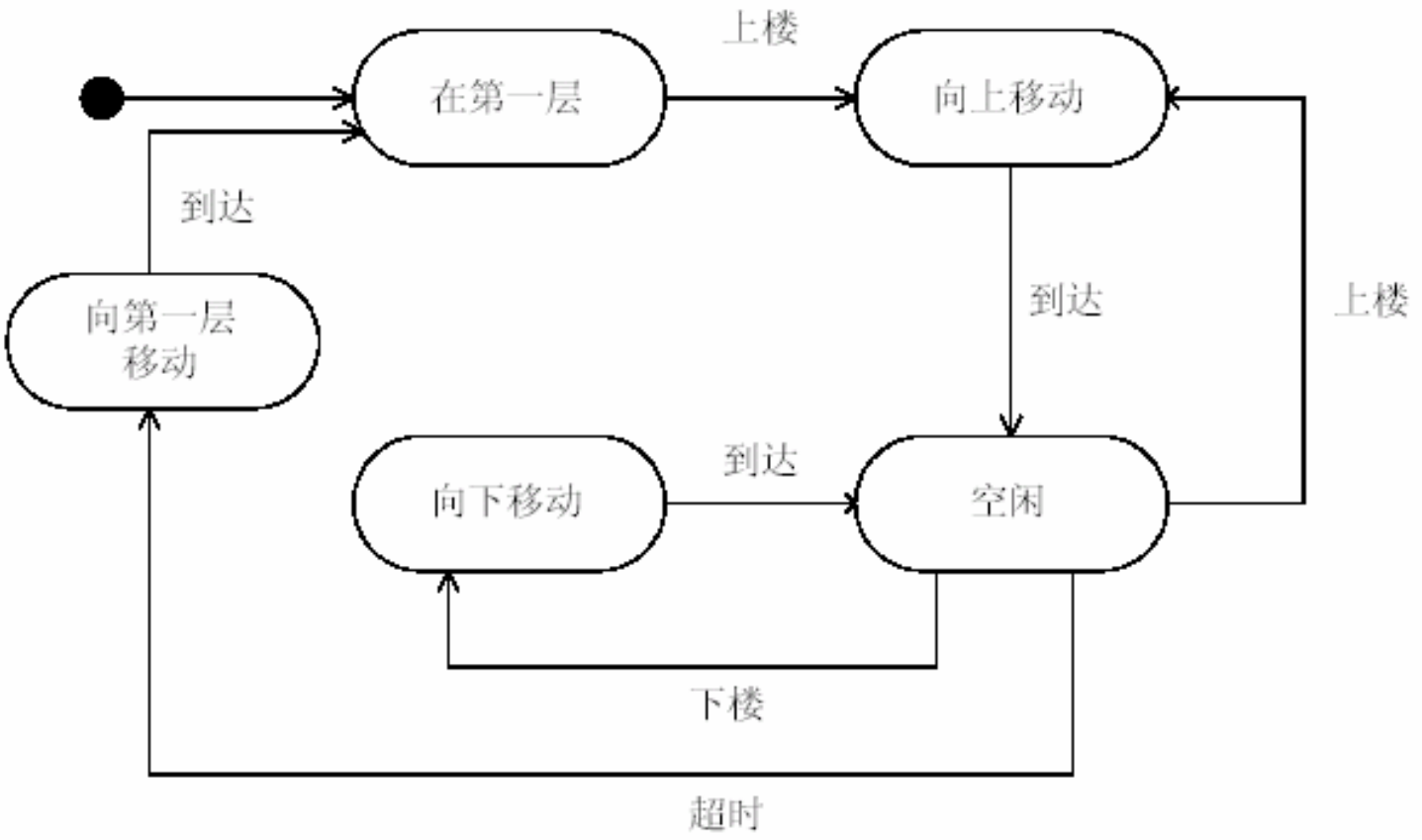
状态图

- 中间状态用圆角矩形表示，可能包含三个部分，第一部分为状态的名称；第二部分为状态变量的名字和值，这部分是可选的；第三部分是活动表，这部分也是可选的。



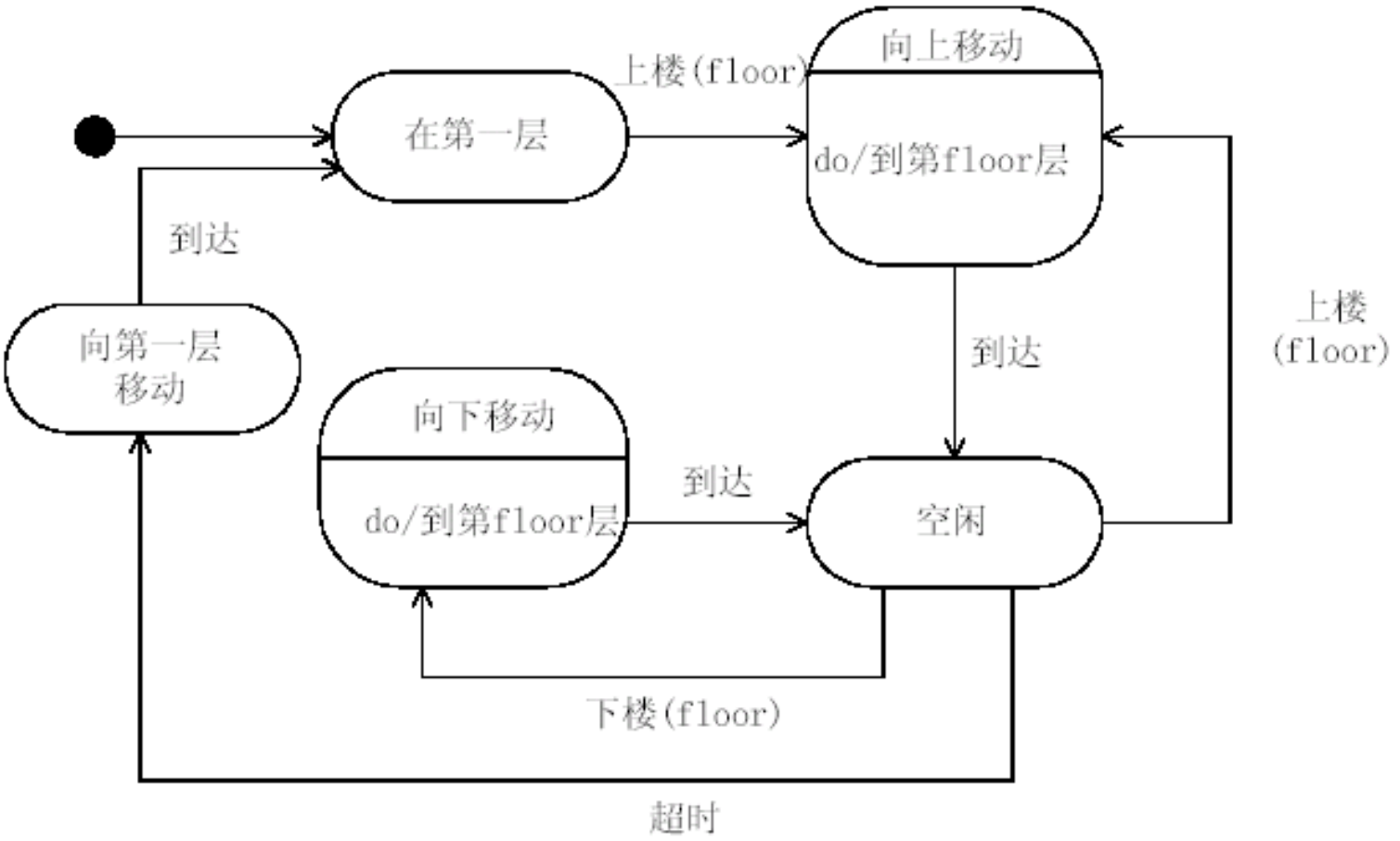
中间状态

状态图



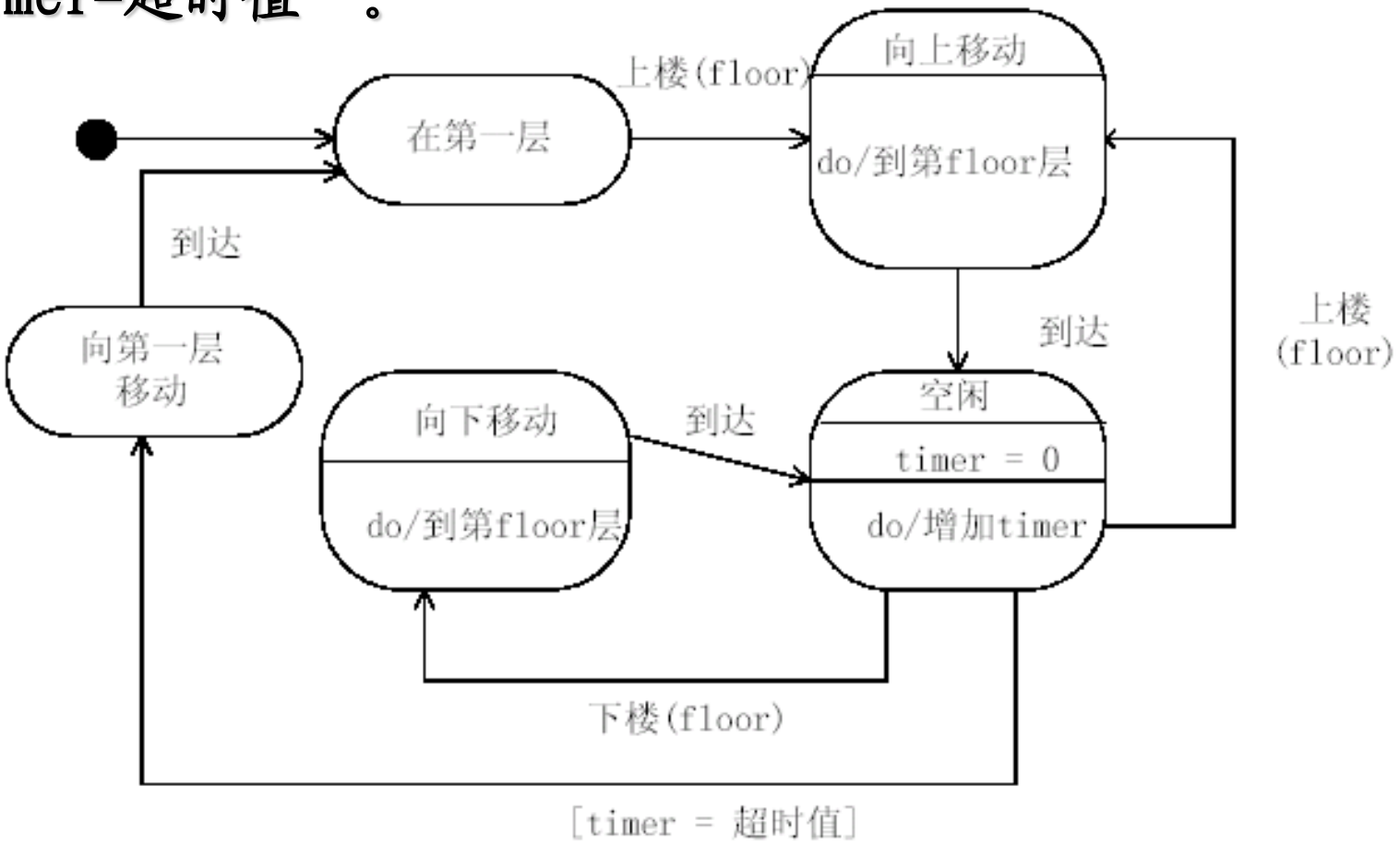
电梯的状态图(本状态图没有终点)

例:带有事件说明的状态转换的例子,在上楼及下楼事件中增加参数floor.



带有事件说明的状态转换

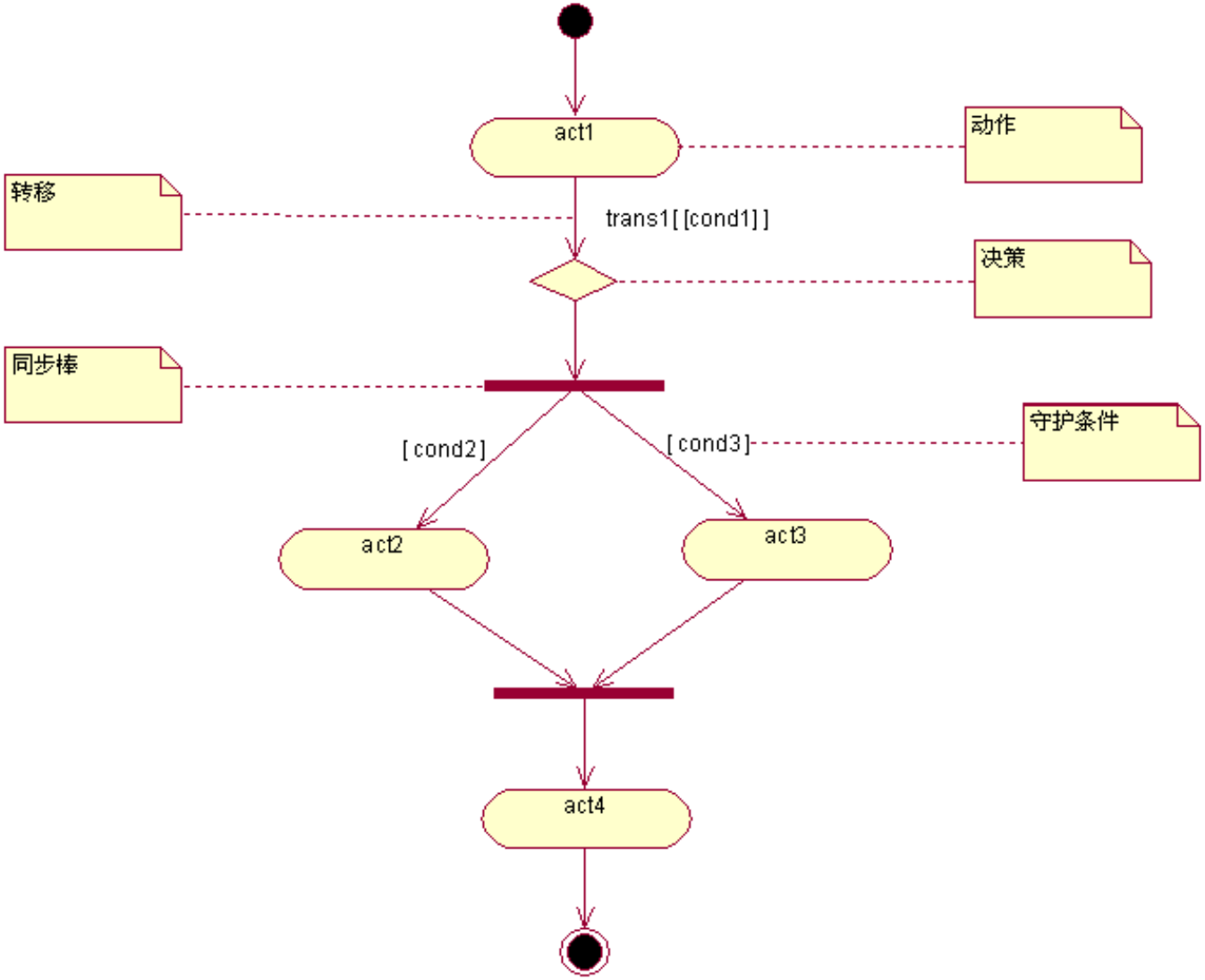
在“空闲”状态,将属性timer的值置0,然后连续递增timer的值,直到“上楼”或“下楼”事件发生,或守卫条件“timer=超时值”。



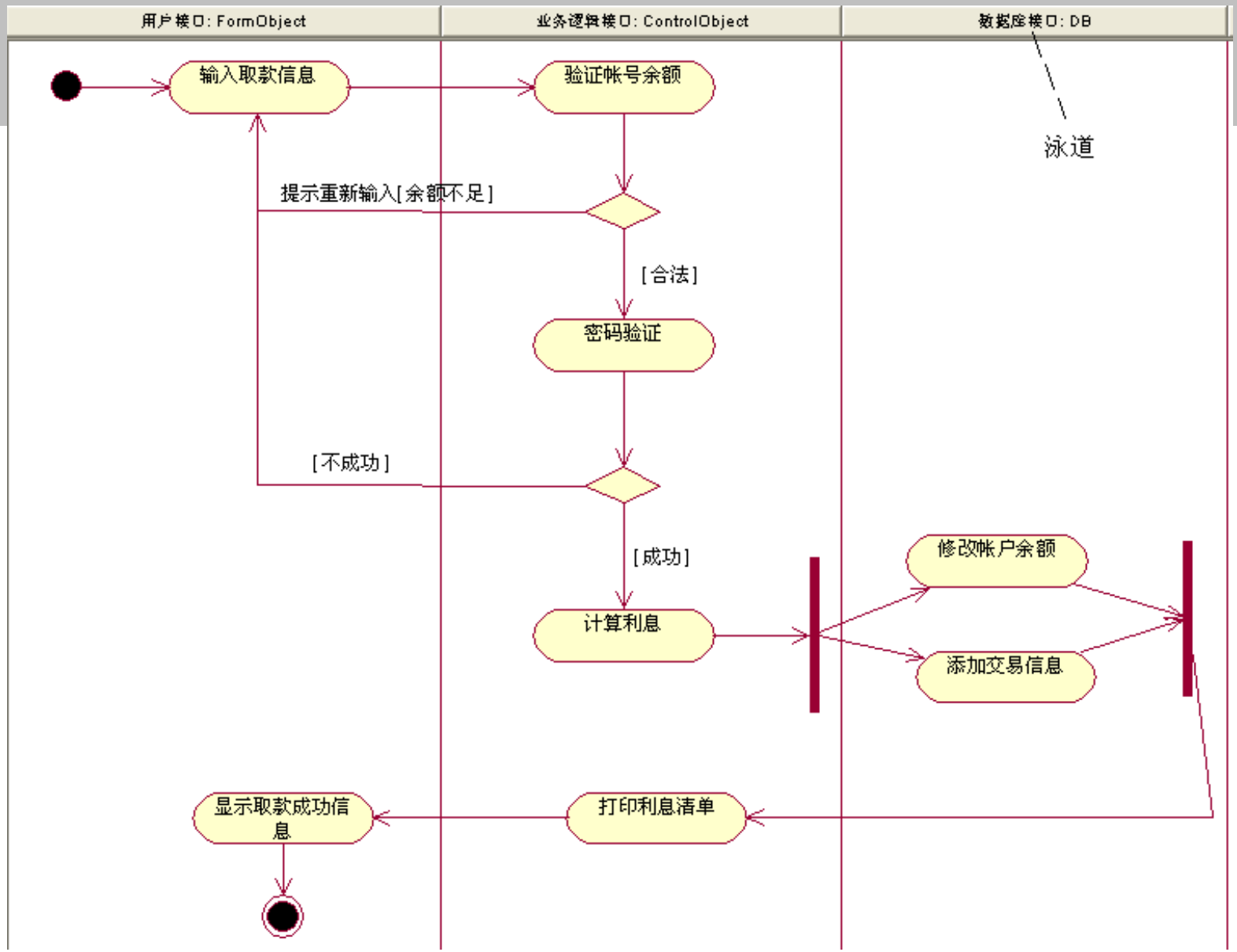
加上属性的状态转换

活动图

- 活动图用来捕捉用例的活动，使用框图的方式显示动作及其结果。
- 活动图是一个流图，描述了从活动到活动的流。
- 它是另一种描述交互的方式，它描述采取何种动作，动作的结果是什么(动作状态改变)，何时发生(动作序列)，以及在何处发生(泳道)。


act1
动作
转移
trans1[[cond1]]
决策
同步棒
[cond2]
[cond3]
守护条件
act2
act3
act4

活动图中的符号


用户接口: FormObject
业务逻辑接口: ControlObject
数据库接口: DB
泳道
输入取款信息
验证帐号余额
提示重新输入[余额不足]
[合法]
密码验证
[不成功]
[成功]
计算利息
修改帐户余额
添加交易信息
显示取款成功信息
打印利息清单

取款用例的活动图

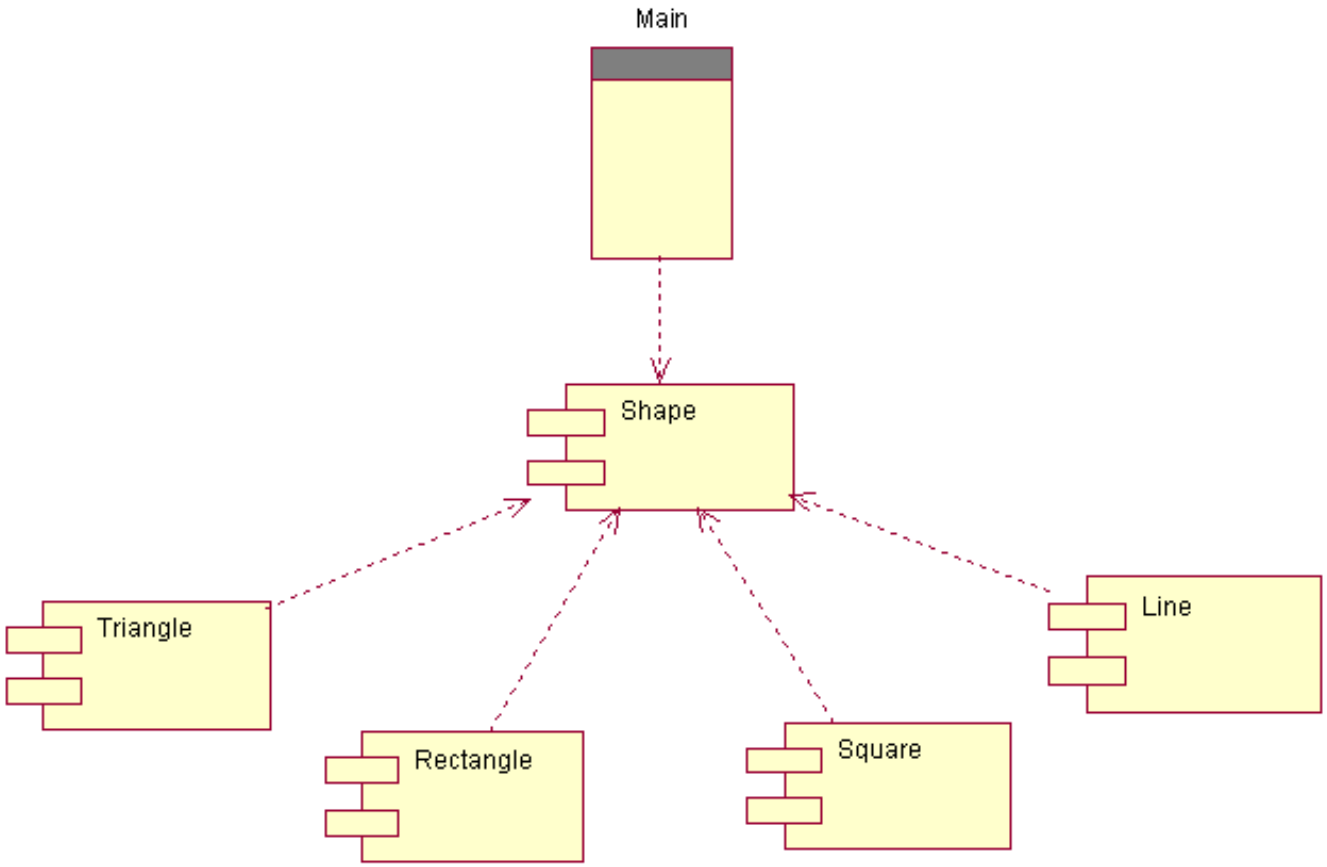
构件图

- 构件图描述软件构件及构件之间的依赖关系，显示代码的静态结构。
- 构件是逻辑架构中定义的概念和功能(例如，类、对象及它们之间的关系)在物理架构中的实现。典型情况下，构件是开发环境中的实现文件。

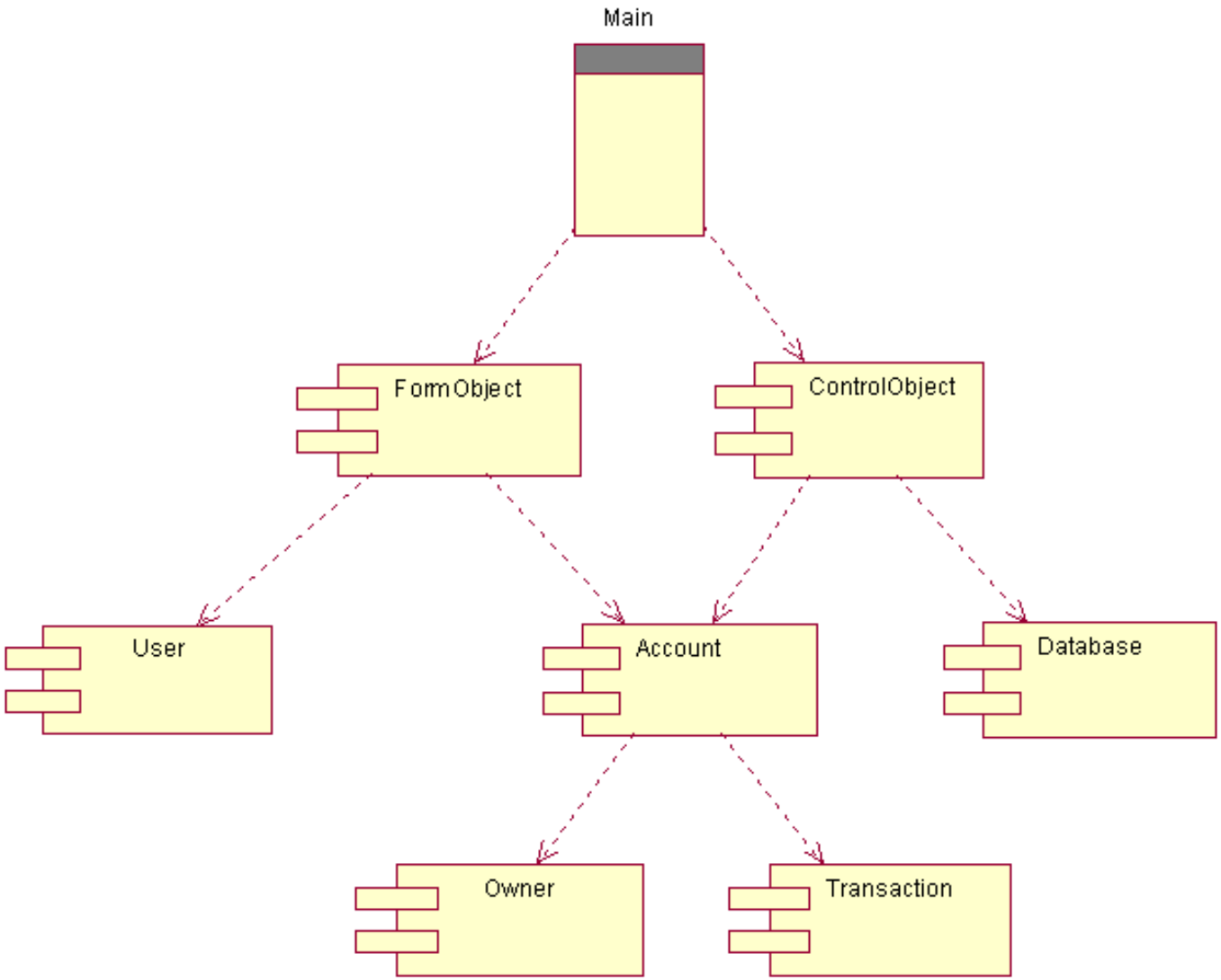
构件图

软件构件可以是下述的任何一种构件。

- 源构件：源构件仅在编译时才有意义。典型情况下，它是实现一个或多个类的源代码文件。
- 二进制构件：典型情况下，二进制构件是对象代码，它是源构件的编译结果。
- 可执行构件：可执行构件是一个可执行的程序文件，它是链接所有二进制构件所得到的结果。一个可执行构件代表在处理器(计算机)上运行的可执行单元。



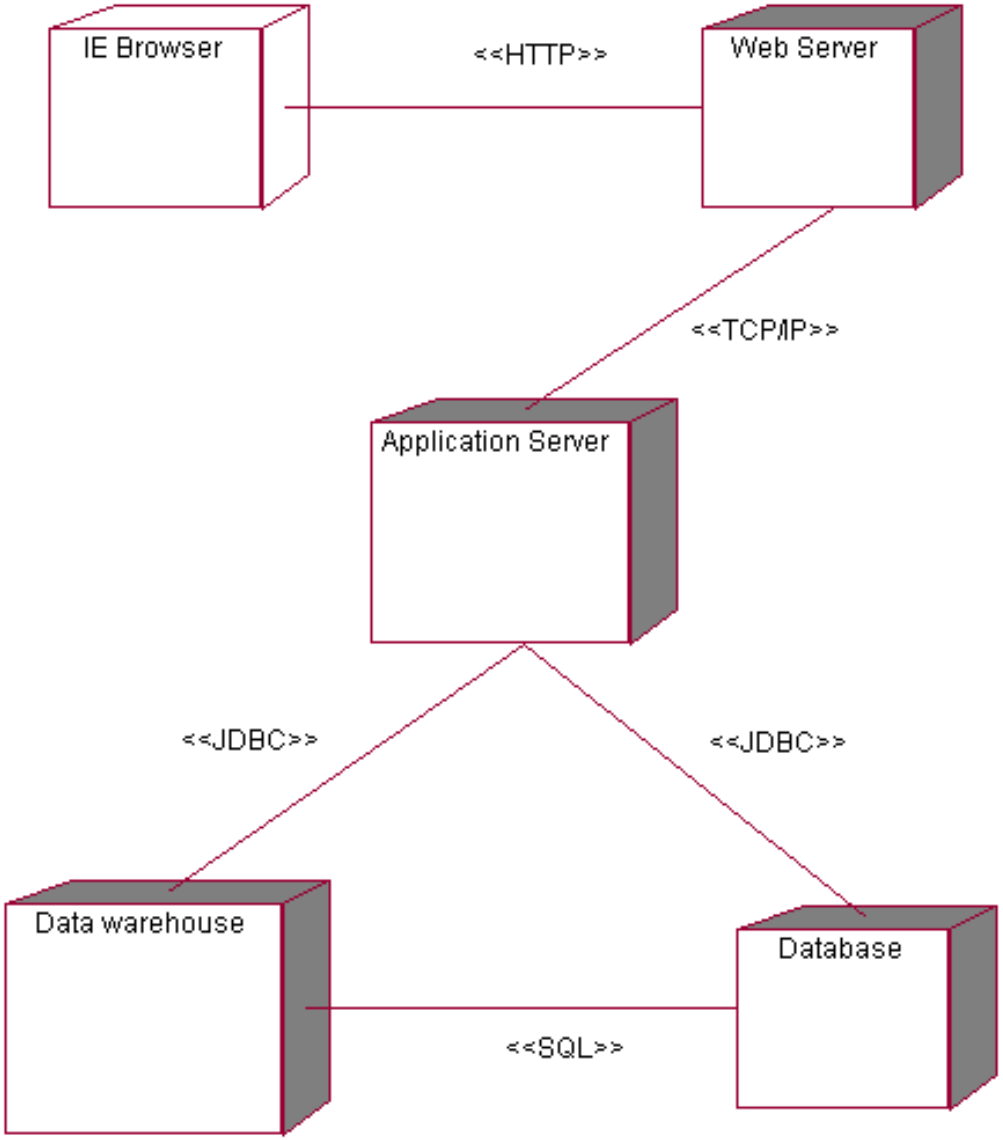
画图系统的构件图


Main
FormObject
ControlObject
User
Account
Database
Owner
Transaction

银行储蓄系统的构件图

部署图

- 部署图描述处理器、设备和连接，它显示系统硬件的物理拓扑结构及在此结构上执行的软件。
- 部署图可以显示计算节点的拓扑结构和通信路径、节点上运行的软件以及软件包含的逻辑单元。


IE Browser
<<HTTP>>
Web Server
<<TCP/IP>>
Application Server
<<JDBC>>
<<JDBC>>
Data warehouse
Database
<<SQL>>

典型的部署图

5.6 使用和扩展UML

使用UML的准则

1. 不要试图使用所有的图形和符号

应该根据项目的特点，选用最适用的图形和符号。一般来说，应该优先选用简单的图形和符号，例如，用例、类、关联、属性和继承等概念是最常用的。

2. 不要为每个事物都画一个模型

应该把精力集中于关键的领域。最好只画几张关键的图，经常使用并不断更新、修改这几张图。

5.6 使用和扩展UML

3. 应该分层次地画模型图

根据项目进展的不同阶段，用正确的观点画模型图。如果处于分析阶段，应该画概念层模型图；当开始着手进行软件设计时，应该画设计层模型图；当考察某个特定的实现方案时，则应画实现层模型图。

使用UML的最大危险是过早地陷入实现细节。为了避免这一危险，应该把重点放在概念层和说明层。

4. 模型应该具有协调性

模型必须在每个抽象层次内和不同的抽象层次之间协调。

5.6 使用和扩展UML

5. 模型和模型元素的大小应该适中

过于复杂的模型和模型元素难于理解也难于使用，这样的模型和模型元素很难生存下去。如果要建模的问题相当复杂，则可以把该问题分解成若干个子问题，分别为每个子问题建模，每个子模型构成原模型中的一个包，以降低建模的难度和模型的复杂性。

5.6 使用和扩展UML

扩展UML的机制

- 为避免使UML变得过于复杂，UML并没有吸收所有面向对象的建模技术和机制，而是设计了适当的扩展机制，使得它能很容易地适应某些特定的方法、机构或用户的需要。利用扩展机制，用户可以定义和使用自己的模型元素。

5.6 使用和扩展UML

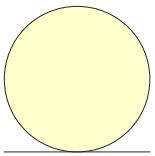
扩展UML的机制

- 扩展的基础是UML的模型元素，利用扩展机制可以给这些元素的变形加上新的语义。新语义可以有三种形式：重新定义，增加新语义或者对某种元素的使用增加一些限制。相应地，有下述三种扩展机制。

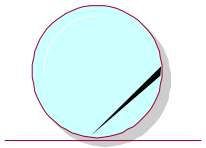
- 构造型 (stereotype)
- 标记值 (tagged value)
- 约束 (constraint)

1. 构造型 (stereotype)

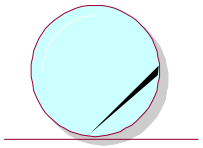
构造型是在一个已定义的模型元素的基础上构造的一种新的模型元素。构造型的信息内容和形式与已存在的基本模型元素相同，但是含义和使用不同。



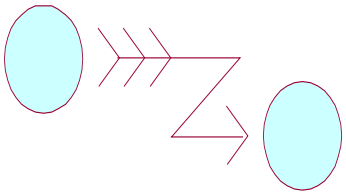
Provider



ProductType



Contract



EnterRecord

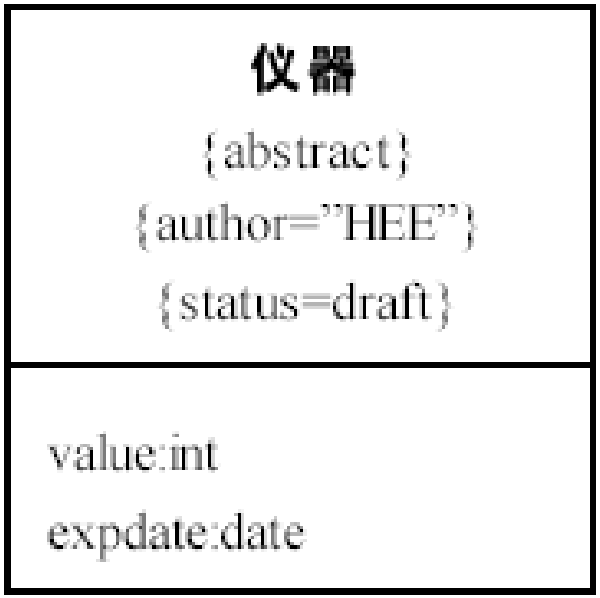

«database»
Reservations


communication
stereotype
«ethernet»
Kiosk
Server

2.标记值 (tagged value)

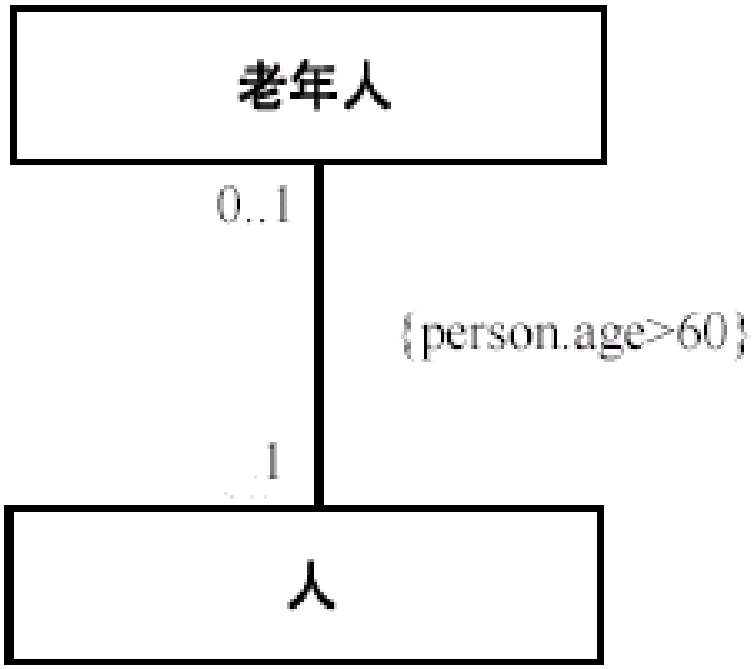
标记值可以用来存储元素的任意信息，对于存储项目管理信息尤其有用的，如元素的创建日期、开发状态、截止日期和测试状态。

标记值用字符串表示，字符串有标记名、等号和值。它们被规则地放置在大括弧内。



3.约束（constraint）

约束是用文字表达式表示的语义限制。约束用大括弧内的字符串表达式表示。约束可以附加在表元素、依赖关系，或注释上。



That's All!