软件工程：第八章 面向对象方法学概述

导学目标

- 了解面向对象方法学和传统方法学的区别以及思想
- 掌握面向对象的概念，如对象、实例、消息等
- 了解面向对象建模的目的、基本原则以及三种模型
- 掌握对象模型之间类图的基本符号和关系
- 掌握动态模型中的相关基础知识
- 掌握功能模型用例图的基本知识以及用例建模的主要工作
- 了解面向对象中三种模型相互之间的关系

第一节 面向对象方法学概述

两种方法学

- 传统软件工程方法学适用于中小型软件产品开发
- 面向对象软件工程方法学适用于大型软件产品开发，面向对象技术已成为当前最好的开发技术

8.1.1 面向对象方法学要点

1、面向对象方法学要点

- **面向对象方法学的出发点和原则是尽可能模拟人类的思维方式，使开发软件的方法与过程尽可能的接近人类认识世界，解决问题的方法与过程**
- 客观世界中的实体既有静态的属性，又有动态的行为
- 与传统方法相反，面向对象是一种以数据或信息为主线，**把数据和处理相结合**的方法
- 面向对象方法把对象作为由数据及可以施加在这些数据上的操作所构成的统一体
- 面向对象方法是一种新的思维方法，把**程序看做是相互协作又彼此独立的对象集合**

2、面向对象有4个要点

1. 认为客观世界是由各种对象组成的，任何事物都是对象，复杂对象由简单对象以某种方案组合而成
    - **一切皆对象**
2. 把所有对象都划分成各种对象类，每个对象类都定义了一组数据和一组方法
3. 按照子类与父类的关系，把若干个对象组合成一个层次结构的系统
    - **继承关系**
4. 对象彼此之间仅能通过传递消息互相联系
    - **封装**

总结：面向对象的方法学方程式

OO = 对象 + 类 + 继承 + 传递消息实现通信

8.1.2 面向对象方法学的优点

1. 与人类习惯的思维方式一致
    - 传统的面向过程的方法**以算法为核心**，将数据和过程独立，忽略了数据和操作之间的内在联系
    - 面向对象的软件技术**以对象为核心**，把描述事务静态属性的数据结构和表示事务动态行为操作放在一起构成整体
    - 例子：比如人有姓名、年龄属性，有吃饭、跑步行为
2. 稳定性好
    - **传统**的软件开发方法以算法为核心，开发过程基于功能分析和功能分解，系统结构紧密依赖于功能需求。当功能需求发生变化时，将引起软件结构的整体修改，因此是**不稳定**的
    - 面向对象的软件开发方法以对象为核心，不是基于功能分解的，当用户需求变化时，只需要做局部修改，因此是稳定的
3. 可重用性好
    - **传统的软件重用技术**是利用标准函数库来构建新系统，但是标准函数缺乏必要的柔性，不能适应不同应用场合的不同需要。标准函数库必须运行在相应的数据结构上，如果要重用这样的模块，则相应的数据也必须重用
    - **面向对象重用技术**具有很强的自含性，将数据和操作平等对待。对象固有的封装性和信息隐藏机制，使其具有独立性
    - 有两种方法可以实现可重复使用一个对象类
        - 创建该类的实例
        - 从该类派生出一个满足当前需要的新类
4. 较易开发大型软件产品
    - 用面向对象方法学开发软件时，构成软件系统的每个对象就像是微型程序，有自己的数据、操作、功能和用途
    - 可以把一个大型软件产品分解成本质上相互独立的小产品来处理
5. 可维护性好
    - 面向对象软件的稳定性要好
    - 面向对象的软件比较容易修改
    - 面向对象的软件比较容易理解
    - 易于测试和调试

第二节 面向对象的概念

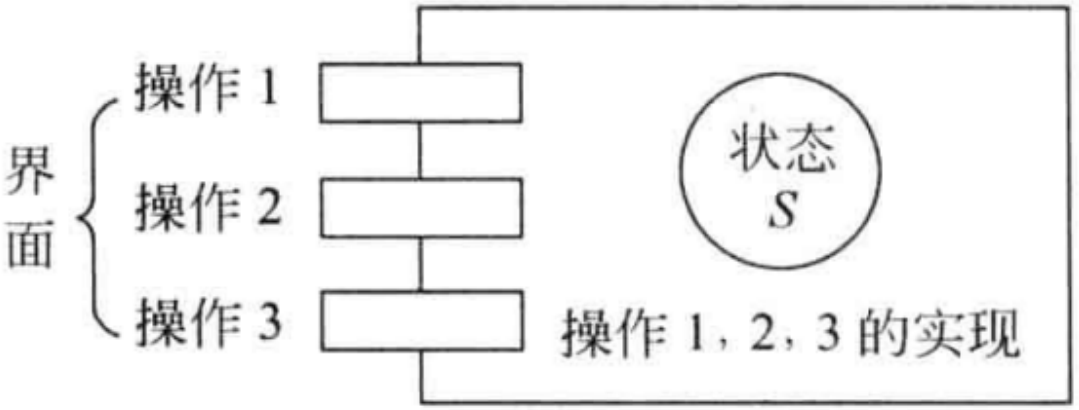
8.2.1 对象

1、基本认识

- 对象可以是具体的物理实体的抽象(如汽车、狗)，也可以是人为的概念(如上学、考试)或者是任何有明确边界和意义的东西
- 对象是对问题域中某个实体的抽象
- 客观世界中的实体既有静态属性，又有动态行为。因此面向对象方法学中的对象是由**描述该对象属性的数据**以及可以**对这些数据施加的所有操作**封装在一起构成的统一体**【对象 = 静态属性 + 动态操作】**
- 通常把对象中的操作叫做服务或者方法

- 对象与外界的界面也就是该对象向公众开放的操作
- 使用对象时，只需知道它向外界提供的接口形式而无须知道它的内部实现算法

2、对象的形象表示



3、定义

- 定义1：对象是具有相同状态的一组操作的集合
- 定义2：对象是对问题域中某个东西的抽象，也就是对象是属性值和操作的封装
- 定义3：对象::= <ID，MS，DS，MI>，形式化定义
    - ID:对象的标识或名字　MS:对象中的操作的集合
    - DS:对象的数据结构　　MI:对象受理的消息名集合
    - 即：对象 = 对象名 + 操作 + 数据结构 + 消息
- 总之，对象中的数据表示对象的状态，一个对象的状态只能该对象的操作来改变

4、特点

- 以数据为中心
- 对象是主动的
- 实现了数据封装
- 本质上具有并行性
- 模块独立性好

8.2.2 类

- 类是对相同属性和行为的一个或者多个对象的描述
- 示例：圆类
    - 具有相同的属性：圆心坐标、半径、颜色等
    - 具有相同的操作：显示自己、放大缩小半径等

8.2.3 实例

- 实例是由某个特定类所描述的一个具体的对象，
- 示例：狗就是动物类的一个实例

8.2.4 消息

- 消息就是要求某个对象执行在定义它的那个类中所定义的某个操作的规格说明
- 消息由三部分组成
  - 接受消息的对象
  - 消息名
  - 0个或多个变元
- 示例：MyCircle.Show(GREEN)
  - MyCircle: 接受消息的对象的名字
  - Show: 消息名
  - GREEN: 消息的变元

8.2.5 方法

- 对象所能够执行的操作
- 示例：MyCircle.Show(GREEN)

8.2.6 属性

- 就是类中所定义的数据，它是对客观世界实体所具有性质的抽象
- 示例：Circle类定义的圆心坐标、半径、颜色等

8.2.7 封装

- 封装就是信息隐蔽，在面向对象的程序中，把数据和实现操作的代码集中起来放在对象内部，外部是不可以直接访问和修改数据的
- 对象具有封装性的条件
  - 有一个清晰的边界
  - 有确定的接口
  - 受保护的内部实现

8.2.8 继承

- 从广义上讲，继承是指能够直接获得已有的性质和特征，而不必重复定义他们
- 在面向对象的技术中，继承是子类自动的共享基类中定义的数据和方法的机制
- 继承具有传递性

8.2.9 多态性

- 在面向对象的软件技术中，多态性是指子类对象可以像父类对象那样使用，同样的消息既可以发送给父类对象，也可以发送给子类对象

8.2.10 重载

- 函数重载：在同一个作用域内的若干个参数特征不同函数可以使用相同的函数名
- 运算符重载：同一个运算符可以施加在不同类型的操作数上面

第三节 面向对象建模

1、为什么要建模?

- 要解决问题之前首先要理解问题，建模就是为了帮助我们更好的去理解问题

2、什么是模型?

- 模型是由一组图示符号和组成这些符号的规则组成，利用他们来定义和描述问题域中的术语和概念。

3、建模的目的

- 主要是为了减少复杂性

模型提供了组织大量信息的一种有效机制

面向对象方法最基本的原则是按照人类习惯的方式，用面向对象的观点建立问题域的模型，开发出尽可能自然地表现求解方法的软件

4、用面向对象技术开发软件时，需要建立三种模型

- **对象模型：描述系统的数据结构**
- **动态模型：描述系统的控制结构**
- **功能模型：描述系统的功能**

对任何大系统来说，上述三种模型都是必不可少的，但是在不同的应用问题中，三种模型的重要程度可能会有所不同

用面向对象的方法开发软件时，在任何情况下，**对象模型始终是最基本、最重要、最核心的**

第四节 对象模型

8.4.1 类图的基本符号

1、概念

- 对象模型表示静态的、结构化的系统的数据性质。它是模拟客观世界实体的对象以及对象之间的映射关系，描述了系统的静态结构
- 对象模型为建立动态模型和功能模型提供了实质性的框架
- Booch、Rumbaugh和Jacobson于1996年设计出统一建模语言UML0.9
- 1997年11月，国际对象管理组织OMG批准把UML1.1作为基于面向对象的标准设计语言
- **通常使用UML提供的类图来建立对象模型**

2、类图的基本符号

- 类图描述类以及类与类之间的静态关系
- 类图是一种静态模型，是创建其他UML图的基础
- 一个系统可以由多张类图来描述
- 一个类可以出现在多张类图中

3、定义类

- **UML中类的表示**
- 类名应该遵循的准则
  - 使用标准术语
  - 使用具有确切含义的名词
  - 必要时使用名词短语作名字
- 总之，类名应该是富于描述性、简洁、而且无二义性的

4、定义属性

- UML属性的描述格式
  - ```
    可见性 属性名：类型= 初值{性质串}
    ```
- **属性的可见性及表示**
  - **公有的(public)：** `+`
  - **私有的(private):** `-`
  - **保护的(protected)：** `#`
- **注意：没有默认可见性**
- 属性名和类型之间用`：`分隔
- 类型和初值之间用`=`隔开
- `{ }`括起来的性质串明确列出该属性的所有取值
- 示例：

```
- 成绩：float = 100
```

5、定义服务

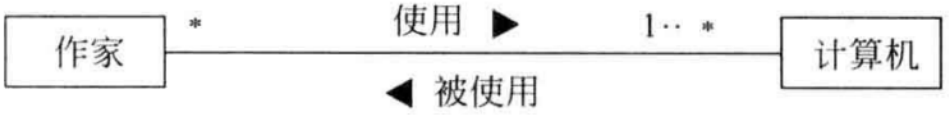
- UML操作的描述格式
  - ```
    可见性 操作名(参数表)：返回值类型{性质串}
    ```
- 参数表是用逗号分隔的形式参数的序列，语法如下：
  - ```
    参数名：类型= 参数值
    ```
    ```

8.4.2 表示关系的符号

类与类之间通常有**关联、泛化(继承)、依赖和细化**4种关系

1、关联

- 表示两个类的对象之间存在某种语义上的联系
- **(1)普通关联**
  - 最常见的关联关系，只要在类与类中存在连接关系就可以使用
  - 普通关联的符号是连接两个类之间的直线



  - 通常关联是双向的
  - 重数的表示方法：(默认重数是1)

```
0 .. 1          表示0到1个对象

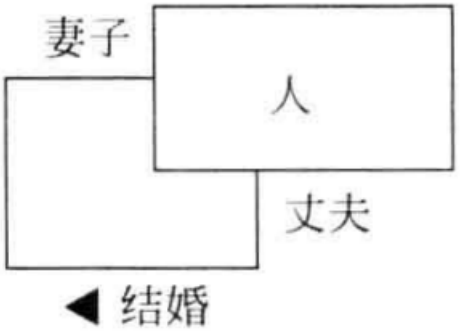
0 .. *或*        表示0到多个对象

1+或1 .. *       表示1到多个对象

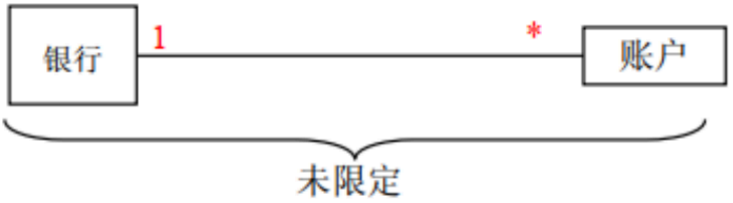
1 .. 15         表示1到15个对象

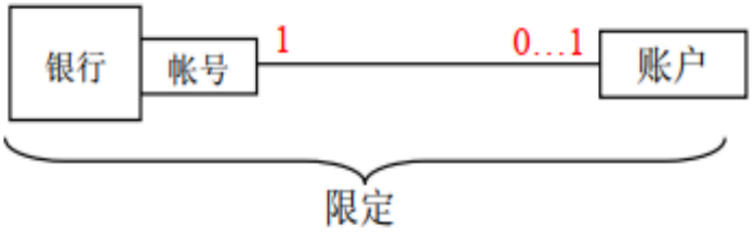
3               表示3个对象
```

- (2)关联的角色
  - 在任何关联中都会涉及参与此关联的对象所扮演的角色
  - 示例：

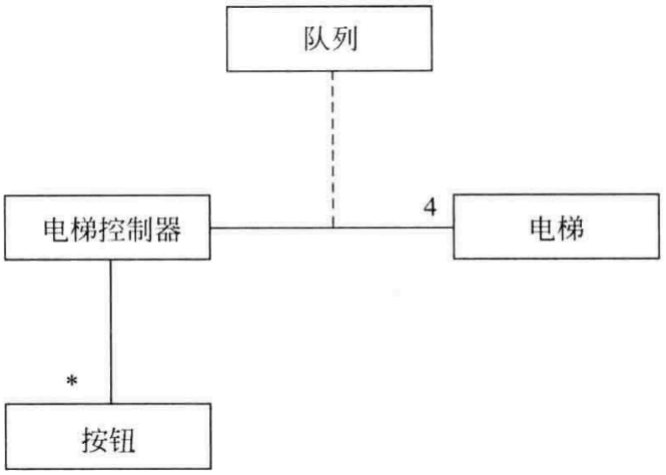


- **(3)限定关联**
  - 通常用于一对多或者多对多的关联关系中，将其转化为一对一或者多对一
  - 示例：



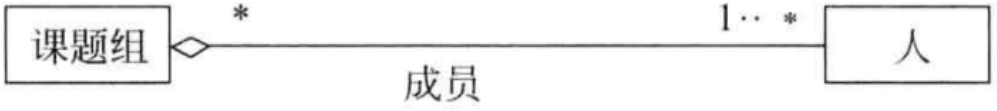


- **(4)关联类**
  - 为了说明关联的性质，可能需要一些附加信息，可以引入关联类来记录这些信息
  - 关联类通过一条虚线与关联连接
  - 关联类与一般类一样，具有属性、操作和关联
  - 示例：

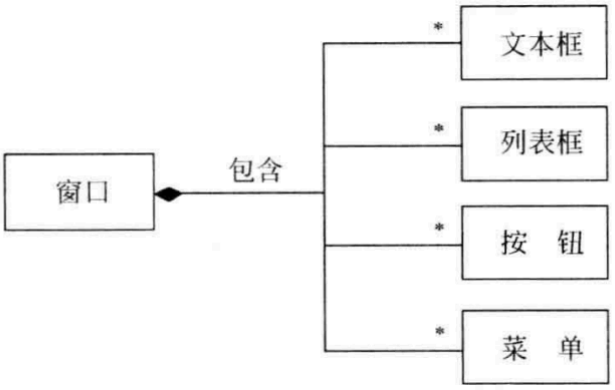


2、聚集

- 是关联的特例；分为两种
- **(1)聚合**
  - 类与类之间是“has a”的关系，整体与部分关系，较弱



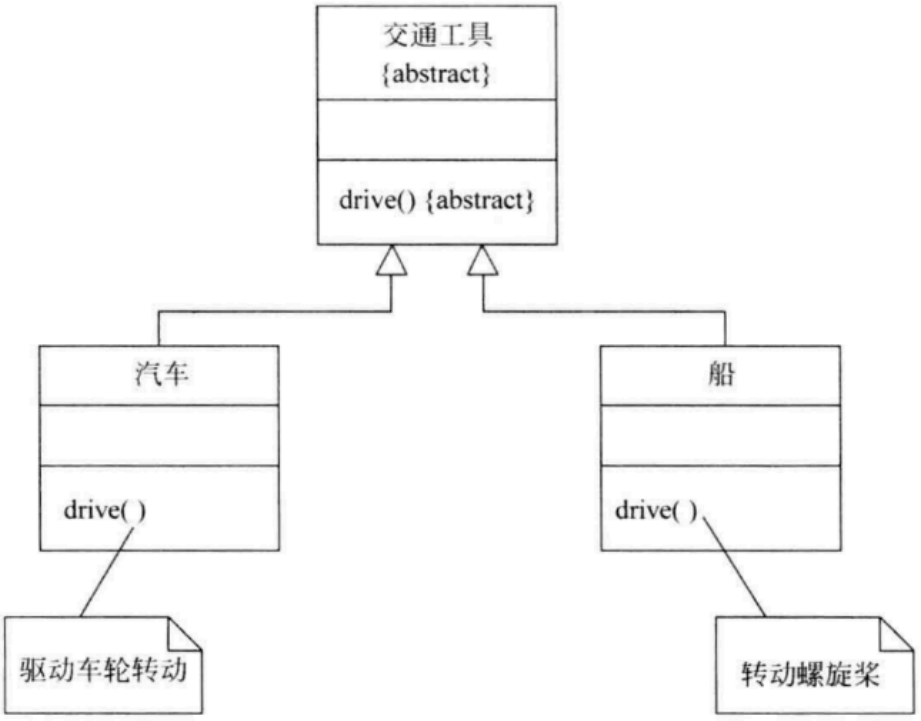
  - 菱形端代表整体事物类，代表部分事物类可属于整体事物类
  - **聚合关系中代表部分事物对象与代表聚合事物对象生存期无关，删除聚合对象不一定删除代表部分事物对象。**
    - (聚合中的整体和部分是可以分割的)
- **(2)组合**
  - 是“contains-a”的关系，整体与部分关系较强，部分完全隶属于整体



- **组合中删除组合对象，同时也就删除代表部分事物对象**
  - (组合关系系中中的整体和部分是不可分割的，是共存亡)

3、泛化

- 就是指的继承关系，也就是类间“一般到特殊”的关系
- 示例：



4、依赖

- 描述两个模型元素(类、用例等)之间的语义连接关系
- 独立模型元素的变化必然会影响依赖它的模型元素
- 在UML的类图中，用带箭头的虚线连接有依赖关系的两个类，箭头指向独立的类
- 示例：

5、细化

- 对同一事物在不同抽象层次上描述时，这些描述关系为细化关系
- 假设两个模型元素A，B描述同一事物，如果B是在A的基础上更详细的描述，则称B细化了A

- 虚线箭头指向被细化的类
- 示例：

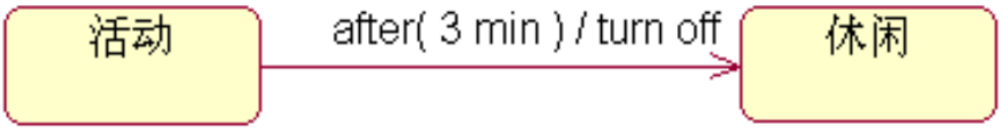
第五节 动态模型

- 一旦对象模型建立好了以后，就需要考察对象的动态行为
- 所有对象都具有自己的生命周期
- 各个对象相互间触发就形成了一系列的状态变化
- 用UML提供的**状态图**来描绘对象的状态、触发状态转换的事件以及对象的行为(对事件的响应)

状态转换图

> 3.6.1 处

- 回顾
  - 表示一个对象(或模型元素)生存史，显示触发状态转移的事件和因状态改变导致的动作
- 三种状态
  - 初态
  - 终态
  - 中间状态
- 活动
  - 语法格式： 事件名/动作表达式
  - 三种事件：
    - entry事件：入口活动
    - do事件：内部执行的活动
    - exit事件：出口活动
  - 动作表达式：描述应该做的具体动作
- 状态转换
  - 语法格式：事件说明[守卫条件]/动作表达式
  - 事件说明的语法：事件名(参数表)
  - 守卫条件是一个布尔表达式
  - 如果同时使用事件说明和守卫条件，则当且仅当事件发生且布尔表达式为真时，状态转换才发生；如果只有守卫条件而没有事件说明，只要守卫条件发生，则状态转换就发生
  - 动作表达式是一个过程表达式，当状态转换开始时执行该表达式
  - 示例：after事件名，3min参数表，turn off 动作表达式



第六节 功能模型

功能模型指明了系统应该做什么，更直接的反映了用户对目标系统的需求

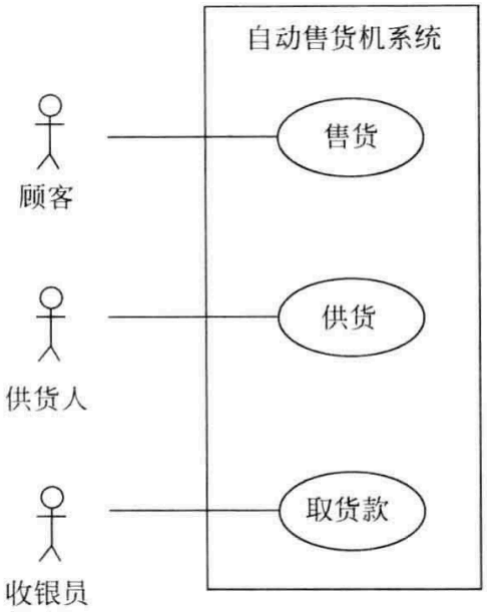
通常，功能模型由一组数据流图组成

建立功能模型有助于软件开发人员更深入的理解问题域，改进和完善自己的设计

UML提供的用例图是进行需求分析和建立功能模型的强有力工具，在UML 中把用用例图建立起来的系统模型称为用例模型

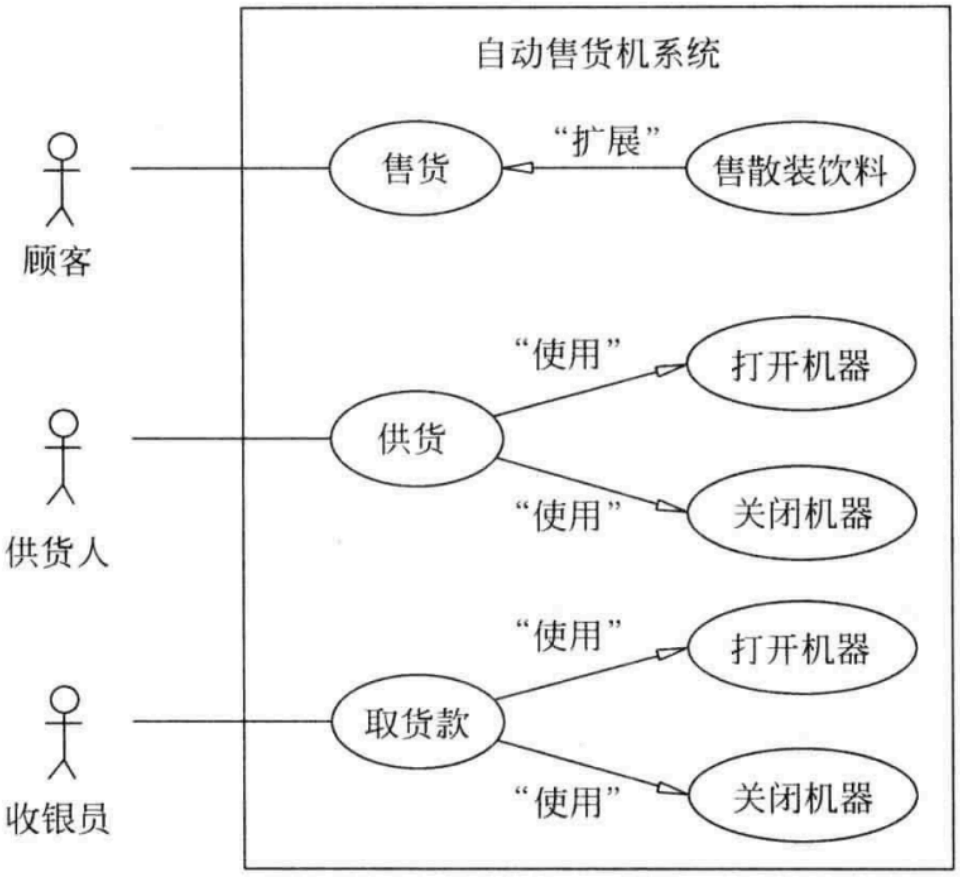
1、用例图

- 一幅用例图包含的模型元素有系统、行为者、用例及用例之间的关系
- 用例图中的元素组成
  - 方框代表系统
  - 椭圆代表用例
  - 线条人代表行为者
  - 连线代表关系



- 系统
  - 是一个提供用例的黑盒子，内部如何工作，用例如何实现，对于建立用户模型来说都是不重要的
  - 边线表示系统的边界，用于划定系统的功能范围
- 用例
  - 一个用例是可以被行为者感受到的、系统的一个完整的功能
  - 用例具有下述特征：
    - 用例代表某些用户可见的功能，实现一个具体的用户目标
    - 用例总是被行为者启动的，并向行为者提供可识别的值
    - 用例必须是完整的
  - 用例是一个类，它代表一类功能，而不是使用该功能的一个具体实例
  - 用例的实例是系统的一种实际使用方法，通常称之为脚本
- 行为者

    - 行为者是指与系统交付的人或者其他系统，它代表外部实体
    - 行为者代表的是一种角色，而不是具体的人或物
    - 在用例图中，用直线连接行为者和用例，表示两者之间交换消息，叫通信联系
- 用例之间的关系
    - 主要有扩展和使用两种关系，它们是**泛化**关系的两种不同形式
    - 扩展关系：
        - 向一个用例中添加一些动作后构成了另一个用例，这两个用例之间的关系就叫做扩展关系
    - 使用关系：
        - 当一个用例使用另一个用例时，两个用例之间就构成了使用关系
    - 使用与扩展的异同
        - 描述一般行为变化时使用扩展关系；在两个或者多个用例中重复描述又想避免这种重复，可以选择使用关系



2、用例建模

- 几乎在任何情况下都需要使用用例，通过用例可以获取用户需求，规划和控制项目
- 一个用例模型由若干幅用例图组成。创建用例模型的工作包括：
    - 定义系统
    - **寻找行为者和用例(关键工作)**
    - 描述用例
    - 定义用例之间的关系
    - 确定模型

- 寻找行为者：可以通过请用户回答问题的方法
    - 谁将使用系统的主要功能？
    - 谁需要借助系统的支持来完成日常工作？
    - 谁来维护和管理系统？
    - 系统控制哪些硬件设备？
    - 系统需要与哪些其他系统交互？
    - 哪些人或系统对本系统产生的结果感兴趣？
- 寻找用例：找到了行为者，可以通过行为者回答问题来获取用例
    - 行为者需要系统提供哪些功能？行为者自身需要做什么？
    - 行为者是否需要读取、创建、删除、修改或者存储系统中的某类信息
    - 系统中发生的事件需要通知行为者吗？行为者需要通知系统某些事件吗？从功能观点看，这些事件能做什么？
    - 行为者的日常工作是否因为系统的新功能而被简化或提高了效率？
    - 系统需要哪些输入输出？输入来自何处？输出到哪里？
    - 当前使用的系统存在的主要问题是什么？

第七节 三种模型之间的关系

1、三种模型相互补充、相互配合

- 功能模型：指明了系统应该做什么
- 动态模型：明确规定了什么时候做
- 对象模型：定义了做事情的实体

2、关系

1. 针对每个类建立的动态模型，描述了类实例的生命周期或运行周期
2. 状态转换驱使行为发生，这些行为在数据流图中被映射为处理，在用例图中被映射为用例，它们同时与类图中的服务相对应
3. 功能模型中的处理(用例)对应于对象模型中的类所提供的服务
4. 数据流图中的数据存储以及数据的源点/终点，通常是对象模型中的对象
5. 数据流图中的数据流往往是对象模型中的属性值，也可能是整个对象
6. 用例图中的行为者，可能是对象模型中的对象
7. 功能模型中的处理(用例)可能产生动态模型中的事件
8. 对象模型描述数据流图中的数据流、数据存储以及数据源点/终点的结构