“十二五”普通高等教育本科国家级规划教材

北京高等教育精品教材

21世纪软件工程专业规划教材

软件工程导论（第6版）

第9章　面向对象方法学引论

张海藩，牟永敏编著　　清华大学出版社

第9章 面向对象方法学引论

面向对象技术强调在软件开发过程中面向客观世界或问题域中的事物，采用人类在认识客观世界的过程中普遍运用的思维方法，直观、自然地描述客观世界中的有关事物。

面向对象的分析方法是利用面向对象的信息建模概念，如实体、关系、属性等，同时运用封装、继承、多态等机制来构造模拟现实系统的方法。

传统的结构化设计方法的基本点是面向过程，系统被分解成若干个过程。而面向对象的方法是采用构造模型的观点，在系统的开发过程中，各个步骤的共同的目标是建造一个问题域的模型。在面向对象的设计中，初始元素是对象，然后将具有共同特征的对象归纳成类，组织类之间的等级关系，构造类库。在应用时，在类库中选择相应的类。

主要内容

9.1 面向对象方法学概述

9.2 面向对象的概念

9.3 面向对象模型

9.4 对象模型

9.5 动态模型

9.6 功能模型

9.7 3种模型之间的关系

主要内容

9.1 面向对象方法学概述

9.1.1. 面向对象方法学的要点

面向对象方法学的要点面向对象方法学的出发点和基本原则，是尽可能模拟人类习惯的思维方式，使开发软件的方法与过程尽可能接近人类认识世界解决问题的方法与过程，也就是使描述问题的问题空间(也称为问题域)与实现解法的解空间(也称为求解域)在结构上尽可能一致。

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。对象与传统的数据有本质区别，它不是被动地等待外界对它施加操作，相反，它是进行处理的主体。必须发消息请求对象主动地执行它的某些操作，处理它的私有数据，而不能从外界直接对它的私有数据进行操作。

9.1 面向对象方法学概述

面向对象方法具有下述4个要点：

(1)面向对象的软件系统是由对象组成的，软件中的任何元素都是对象，复杂的软件对象由比较简单的对象组合而成。

(2)把所有对象都划分成各种对象类(简称为类,class)，每个对象类都定义了一组数据和一组方法。数据用于表示对象的静态属性，是对象的状态信息。

(3)按照子类(或称为派生类)与父类(或称为基类)的关系，把若干个对象类组成一个层次结构的系统(也称为类等级)。

(4)对象彼此之间仅能通过传递消息互相联系。

面向对象的方法学可以用下列方程来概括：OO=objects+classes+Inheritance+communication with messages，即，既使用对象又使用类和继承等机制，且对象之间仅能通过传递消息实现彼此通信。

9.1 面向对象方法学概述

9.1.2. 面向对象方法学的优点

1. 与人类习惯的思维方法一致

- 面向对象的软件技术以对象为核心，用这种技术开发出的软件系统由对象组成。对象是由描述内部状态表示静态属性的数据，以及可以对这些数据施加的操作(对象的动态行为)，封装在一起所构成的统一体。
- 面向对象的设计方法基本原理是，使用现实世界的概念抽象地思考问题从而自然地解决问题。
- 面向对象方法学的基本原则是按照人类习惯的思维方法建立问题域的模型，开发出尽可能直观、自然地表现求解方法的软件系统。面向对象的软件系统中使用的对象，是对客观世界中实体的抽象。

9.1 面向对象方法学概述

2. 稳定性好

面向对象的软件系统的结构是根据问题领域的模型建立起来的，而不是基于对系统应完成的功能的分解，所以，当对系统的功能需求变化时并不会引起软件结构的整体变化，往往仅需要作一些局部性的修改。由于现实世界中的实体是相对稳定的，因此，以对象为中心构造的软件系统也是比较**稳定**的。

3. 可重用性好

对象固有的封装性和信息隐藏机制，使得对象的内部实现与外界隔离，具有较强的独立性。对象是比较理想的模块和可重用的软件成分。

面向对象的软件技术在利用可重用的软件成分构造新的软件系统时，有很大的灵活性。有两种方法可以重复使用一个对象类：一种方法是创建该类的实例，从而直接使用它；另一种方法是从它派生出一个满足当前需要的新类。

9.1 面向对象方法学概述

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一个微型程序，有自己的数据、操作、功能和用途，因此，可以把一个大型软件产品分解成一系列本质上相互独立的小产品来处理，这就不仅降低了开发的技术难度，而且也使得对开发工作的管理比较容易。

5. 可维护性好

(1) 面向对象的软件稳定性比较好。

(2) 面向对象的软件比较容易修改。

(3) 面向对象的软件比较容易理解。

(4) 易于测试和调试。

主要内容

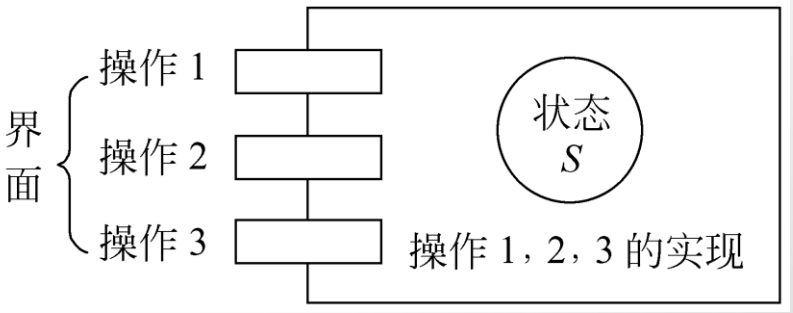
9.2 面向对象的概念

9.2.1. 对象

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。对象可以作的操作表示它的动态行为，在面向对象分析和面向对象设计中，通常把对象的操作称为服务或方法。

1. 对象的形象表示

右图为对象的形象表示，形象地描绘了具有3个操作的对象。



9.2 面向对象的概念

1. 对象的形象表示

一个对象很像一台录音机。实现对象操作的代码和数据是隐藏在对象内部的，一个对象好像是一个黑盒子，表示它内部状态的数据和实现各个操作的代码及局部数据，都被封装在这个黑盒子内部，在外面是看不见的，更不能从外面去访问或修改这些数据或代码。

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，不仅使得对象的使用变得非常简单、方便，而且具有很高的安全性和可靠性。

对象内部的数据只能通过对象的公有方法(如C++的公有成员函数)来访问或处理，这就保证了对这些数据的访问或处理，在任何时候都是使用统一的方法进行的。

9.2 面向对象的概念

2.对象的定义

人们从不同角度给出对象的不同定义如下:(含义相同)

(1) **定义1:** 对象是具有相同状态的一组操作的集合。

这个定义主要是从面向对象程序设计的角度看“对象”。

(2) **定义2:** 对象是对问题域中某个东西的抽象，这种抽象反映了系统保存有关这个东西的信息或与它交互的能力。也就是说，对象是对属性值和操作的封装。

这个定义着重从信息模拟的角度看待“对象”。

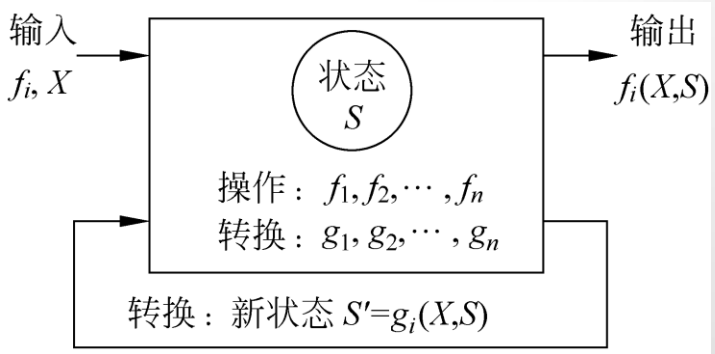
(3) **定义3:** 对象::=〈ID,MS,DS,MI〉。其中，ID是对象的标识或名字，MS是对象中的操作集合，DS是对象的数据结构，MI是对象受理的消息名集合(即对外接口)。

9.2 面向对象的概念

2. 对象的定义

对象是封装了数据结构及可以施加在这些数据结构上的操作的封装体，这个封装体有可以唯一地标识它的名字，而且向外界提供一组服务(即公有的操作)。对象中的数据表示对象的状态，一个对象的状态只能由该对象的操作来改变。

从动态角度或对象的实现机制来看，对象是一台自动机。具有内部状态S，操作$f_i\ (i=1, 2, \cdots, n)$，且与操作f_i对应的状态转换函数为$g_i\ (i=1, 2, \cdots, n)$的一个对象，可用右图所示的自动机来模拟。



9.2 面向对象的概念

3. 对象的特点

对象有如下一些基本特点。

(1) **以数据为中心**。操作围绕对其数据所需要做的处理来设置，不设置与这些数据无关的操作，而且操作的结果往往与当时所处的状态(数据的值)有关。

(2) **对象是主动的**。它是进行处理的主体。不能从外部直接加工它的私有数据，必须通过它的公有接口向对象发消息，请求它执行它的某个操作，处理它的私有数据。

(3) **实现了数据封装**。对象好像是一只黑盒子，它的私有数据完全被封装在盒子内部，对外是隐藏的、不可见的，对私有数据的访问或处理只能通过公有的操作进行。

9.2 面向对象的概念

3. 对象的特点

(4) 本质上具有并行性。对象是描述其内部状态的数据及可以对这些数据施加的全部操作的集合。不同对象各自独立地处理自身的数据，彼此通过发消息传递信息完成通信。

(5) 模块独立性好。对象内部各种元素彼此结合得很紧密，内聚性相当强。由于完成对象功能所需要的元素(数据和方法)基本上都被封装在对象内部，它与外界的联系自然就比较少，因此，对象之间的耦合通常比较松。

9.2 面向对象的概念

9.2.2. 其他概念

1.类（class）

在面向对象的软件技术中，“类”就是对具有相同数据和相同操作的一组相似对象的定义，也就是说，类是对具有相同属性和行为的一个或多个对象的描述，通常在这种描述中也包括对怎样创建该类的新对象的说明。

左图是3个圆心位置、半径大小和颜色均不相同的圆，是3个不同的对象。但是，它们都有相同的数据(圆心坐标、半径、颜色)和相同的操作。因此，它们是同一类事物，可以用“Circle类”来定义。

9.2 面向对象的概念

2. 实例（instance）

- **实例**就是由某个特定的类所描述的一个具体的对象。
- “对象” 既可以指一个具体的对象，也可以泛指一般的对象，但是，“实例”必然是指一个具体的对象。

3. 消息（massage）

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。通常，一个消息由接收消息的对象、消息选择符(也称为消息名）、 零个或多个变元3部分组成。

9.2 面向对象的概念

3.消息（massage）

例如，MyCircle是一个半径为4cm、圆心位于(100，200)的Circle类的对象，也就是Circle类的一个**实例**，当要求它以绿颜色在屏幕上显示自己时，在C++语言中应该向它发下列**消息**：

```
MyCircle.Show(GREEN);
```

其中，MyCircle是接收消息的对象的名字,Show是消息选择符(即消息名)，圆括号内的GREEN是消息的变元。当MyCircle接收到这个消息后，将执行在Circle类中所定义的Show操作。

9.2 面向对象的概念

4.方法（method）

方法就是对象所能执行的操作，也就是类中所定义的服务。方法描述了对象执行操作的算法，响应消息的方法。在C++语言中把方法称为成员函数。

5.属性（attribute）

属性就是类中所定义的数据，它是对客观世界实体所具有的性质的抽象。类的每个实例都有自己特有的属性值。在C++语言中把属性称为数据成员。

例如，Circle类中定义的代表圆心坐标、半径、颜色等的数据成员，就是圆的属性。

9.2 面向对象的概念

6. 封装（encapsulation）

在面向对象的程序中，**封装**是指把数据和实现操作的代码集中起来放在对象内部。

对象具有封装性的条件如下：

（1）有一个清晰的边界。所有私有数据和实现操作的代码都被封装在这个边界内，从外面看不见更不能直接访问。

（2）有确定的接口（即协议）。这些接口就是对象可以接受的消息，只能通过向对象发送消息来使用它。

（3）受保护的内部实现。实现对象功能的细节（私有数据和代码）不能在定义该对象的类的范围外访问。

封装就是信息隐藏，通过封装对外界隐藏对象的实现细节。

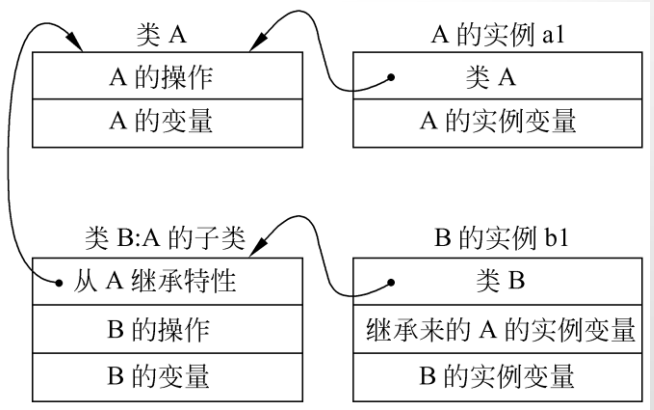
9.2 面向对象的概念

7.继承（inheritance）

在面向对象的软件技术中，**继承**是子类自动地共享基类中定义的数据和方法的机制。

面向对象软件技术把类组成一个层次结构的系统(类等级)：一个类的上层可以有父类，下层可以有子类。这种层次结构系统的一个重要性质是继承性，一个类直接继承其父类的全部描述(数据和操作)。

右图为实现继承机制的原理，图中以A、B两个类为例，其中B类是从A类派生出来的子类，它除了具有自己定义的特性(数据和操作)之外，还从父类A继承特性。



9.2 面向对象的概念

7.继承（inheritance）

- 继承具有传递性，即一个类实际上继承了它所在的类等级中在它上层的全部基类的所有描述。
- 当类等级为树形结构时，类的继承是单继承；当允许一个类有多个父类时，类的继承是多重继承。多重继承的类可以组合多个父类的性质构成所需的性质，使用多重继承时要注意避免二义性。
- 当需要扩充原有的功能时，派生类的方法可以调用其基类的方法，并在此基础上增加必要的程序代码；当需要完全改变原有操作的算法时，可以在派生类中实现一个与基类方法同名而算法不同的方法；当需要增加新的功能时，可以在派生类中实现一个新的方法。
- 有了继承性可以用把已有的一般性的解加以具体化的办法，来达到软件重用的目的。

9.2 面向对象的概念

8. 多态性（polymorphism）

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。即，在类等级的不同层次中可以共享(公用)一个行为(方法)的名字，然而不同层次中的每个类却各自按自己的需要来实现这个行为。

多态性机制不仅增加了面向对象软件系统的灵活性，进一步减少了信息冗余，而且显著提高了软件的可重用性和可扩充性。

9. 重载（overloading）

有两种**重载**：函数重载是指在同一作用域内的若干个参数特征不同的函数可以使用相同的函数名字；运算符重载是指同一个运算符可以施加于不同类型的操作数上面。

重载进一步提高了面向对象系统的灵活性和可读性。

主要内容

9.3 面向对象建模

- 所谓**模型**，就是为了理解事物而对事物作出的一种抽象，是对事物的一种无歧义的书面描述。通常，模型由一组图示符号和组织这些符号的规则组成，利用它们来定义和描述问题域中的术语和概念。
- 模型可以帮助人们思考问题、定义术语、在选择术语时作出适当的假设，并且有助于保持定义和假设的一致性。
- 对于因过分复杂而不能直接理解的系统，特别需要建立模型，建模的目的主要是为了减少复杂性。
- 面向对象方法最基本的原则，是按照人们习惯的思维方式，用面向对象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。

9.3 面向对象建模

- 用面向对象方法开发软件，通常需要建立3种形式的模型，它们分别是描述系统数据结构的**对象模型**，描述系统控制结构的**动态模型**和描述系统功能的**功能模型**。
- 一个典型的软件系统使用数据结构(对象模型)，执行操作(动态模型)，并且完成数据值的变化(功能模型)。
- 对任何大系统来说，上述3种模型都是必不可少的。用面向对象方法开发软件，在任何情况下，对象模型始终都是最重要、最基本、最核心的。
- 在面向对象分析过程中，构造出完全独立于实现的应用域模型；在面向对象设计过程中，把求解域的结构逐渐加入到模型中；在实现阶段，把应用域和求解域的结构都编成程序代码并进行严格的测试验证。

主要内容

9.1 面向对象方法学概述

9.2 面向对象的概念

9.3 面向对象模型

9.4 对象模型

9.5 动态模型

9.6 功能模型

9.7 3种模型之间的关系

9.4 对象模型

- 对象模型表示静态的、结构化的系统的“数据”性质。它是对模拟客观世界实体的对象以及对象彼此间的关系的映射，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。
- 建立对象模型，需要用适当的建模语言来表达模型，建模语言由记号（即模型中使用的符号）和使用记号的规则（语法、语义和语用）组成。
- 1997年11月，国际对象管理组织OMG批准把UML 1.1作为基于面向对象技术的标准建模语言。
- 通常，使用UML提供的类图来建立对象模型。

9.4 对象模型

9.4.1. 类图的基本符号

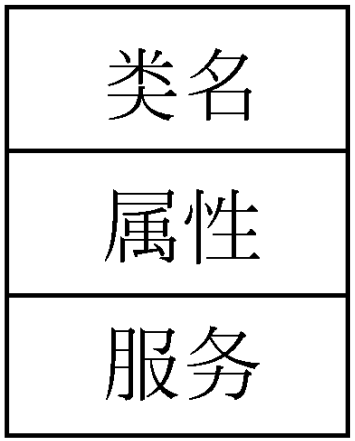
类图描述类及类与类之间的静态关系。类图是一种静态模型，它是创建其他UML图的基础。一个系统可以由多张类图来描述，一个类也可以出现在几张类图中。

1. 定义类

为类命名时应该遵守以下几条准则：

- 使用标准术语；
- 使用具有确切含义的名词；
- 必要时用名词短语作名字。

右图表示类的图形符号。



9.4 对象模型

2. 定义属性

UML描述属性的语法格式如下：

可见性属性名： 类型名=初值{性质串}

- 属性的可见性（即可访问性）通常有下述3种：公有的(public)、私有的(private)和保护的(protected)，分别用加号(+)、减号(-)和井号(#)表示。**注意，没有默认的可见性。**
- 属性名和类型名之间用冒号(:)分隔。类型名表示该属性的数据类型，它可以是基本数据类型，也可以是用户自定义的类型。
- 在创建类的实例时应给其属性赋值，如果给某个属性定义了初值，则该初值可作为创建实例时这个属性的默认值。类型名和初值之间用等号（=）隔开。

9.4 对象模型

2. 定义属性

- 用花括号括起来的性质串明确地列出该属性所有可能的取值。枚举类型的属性往往用性质串列出可以选用的枚举值，不同枚举值之间用逗号分隔。也可以用性质串说明属性的其他性质，例如，约束说明{只读}表明该属性是只读属性。

例如，“发货单”类的属性“管理员”，在UML类图中如下面描述：

-管理员：String=“未定”

类的属性中还可以有一种能被该类所有对象共享的属性，称为类的作用域属性，也称为类变量。类变量在类图中表示为带下划线的属性。

例如，发货单类的类变量“货单数”，用来统计发货单的总数，在该类所有对象中这个属性的值都是一样的，下面是对这个属性的描述：-货单数： Integer

9.4 对象模型

3. 定义服务

服务也就是操作，UML描述操作的语法格式如下：

可见性操作名（参数表）： 返回值类型{性质串}

- 操作可见性的定义方法与属性相同（见(2)定义属性）。
- 参数表是用逗号分隔的形式参数的序列。描述一个参数的语法如下：

参数名： 类型名=默认值

- 当操作的调用者未提供实在参数时，该参数就使用默认值。
- 与属性类似，在类中可定义类作用域操作，在类图中表示为带下划线的操作。这种操作只能存取本类的类作用域属性。

9.4 对象模型

9.4.2. 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

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

只要在类与类之间存在连接关系就可以用普通关联表示。普通关联的图示符号是连接两个类之间的直线，如下图所示。

关联是双向的，可在每一个方向上为关联起一个名字(也可不起名字)。为避免混淆，在名字前面（或后面）加一个表示关联方向的黑三角。

9.4 对象模型

(1) 普通关联

在表示关联的直线两端可以写上重数（multiplicity），它表示该类有多少个对象与对方的一个对象连接。重数的表示方法通常有：

如果图中未明确标出关联的重数，则默认重数是1。

上图表示，一个作家可以使用1到多台计算机，一台计算机可被0至多个作家使用。

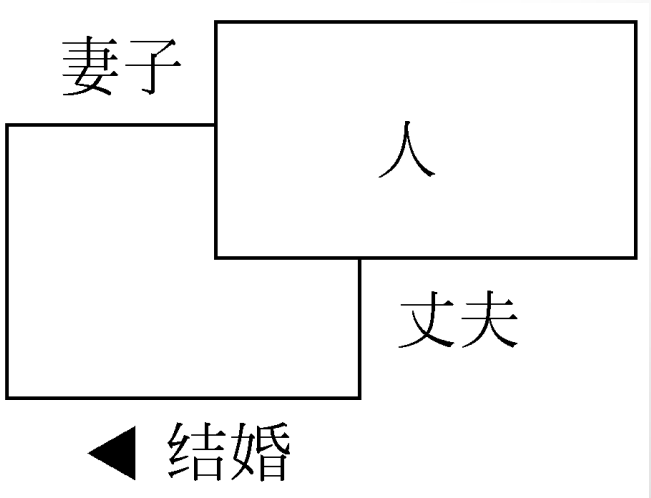
重数	表示
0 .. 1	0到1个对象
0 .. *或 *	0到多个对象
1 + 或1 .. *	1到多个对象
1 .. 15	1到15个对象
3	3个对象

9.4 对象模型

(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（即起的作用），在某些情况下显式标明角色名有助于别人理解类图。

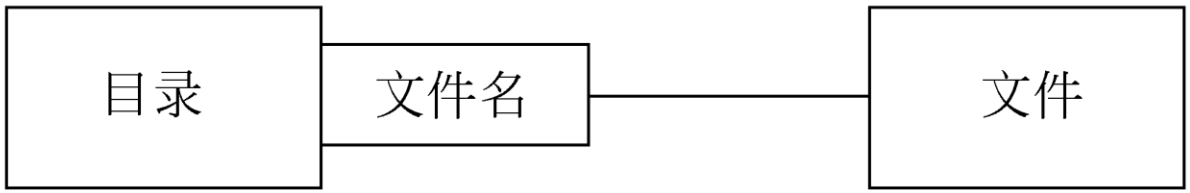
右图是一个递归关联（即一个类与它本身有关联关系）的例子。一个人与另一个人结婚，必然一个人扮演丈夫的角色，另一个人扮演妻子的角色。如果没有显式标出角色名，则意味着用类名作为角色名。



9.4 对象模型

(3) 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，可以把模型中的重数从一对多变成一对一，或从多对多简化成多对一。在类图中把限定词放在关联关系末端的一个小方框内。



上图利用限定词“文件名”表示了目录与文件之间的关系，可见，利用限定词把一对多关系简化成了一对一关系。

上图一个受限的关联限定提高了语义精确性，增强了查询能力。限定的语法表明，文件名在其目录内是唯一的。

9.4 对象模型

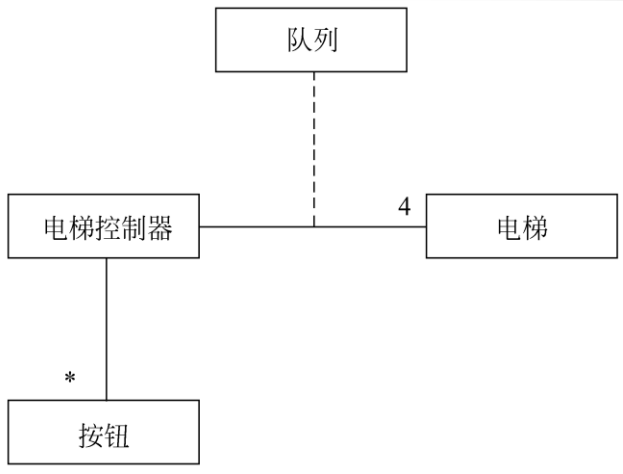
(4) 关联类

为了说明关联的性质，引入一个关联类来记录附加信息。关联中的每个连接与关联类的一个对象相联系。关联类通过一条虚线与关联连接。

关联类与一般的类一样，也有属性、操作和关联。

右图是一个电梯系统的类模型，队列就是电梯控制器类与电梯类的关联关系上的关联类。

一个电梯控制器控制着4台电梯，控制器和电梯之间的实际连接就有4个，每个连接都对应一个队列（对象），每个队列（对象）存储着来自控制器和电梯内部按钮的请求服务信息。



9.4 对象模型

2. 聚集

聚集也称为聚合，是关联的特例。聚集表示类与类之间的关系是整体与部分的关系。使用的“包含”、“组成”、“分为……部分”等字句，意味着存在聚集关系。有**共享聚集**和**组合聚集**两种特殊的聚集关系。

(1) 共享聚集

如果在聚集关系中处于部分方的对象可同时参与多个处于整体方对象的构成，则该聚集称为**共享聚集**。下图中，一个课题组包含许多成员，每个成员又可以是另一个课题组的成员，则课题组和成员之间是共享聚集关系。一般聚集和共享聚集的关联关系用空心菱形表示。

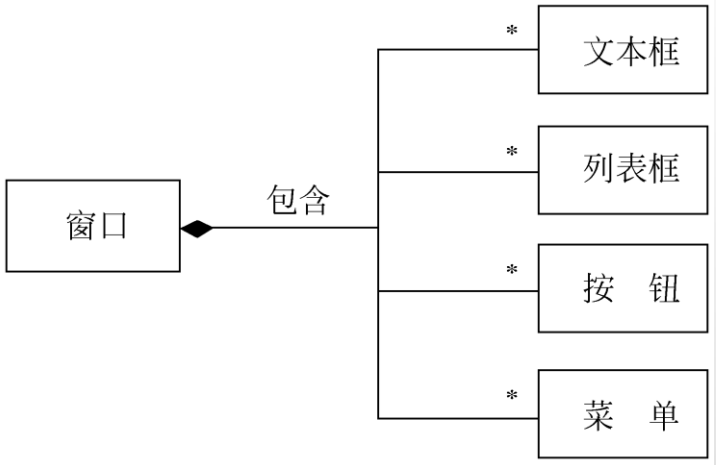
9.4 对象模型

2. 聚集

(2) 组合聚集

如果部分类完全隶属于整体类，部分与整体共存，整体不存在了部分也会随之消失（或失去存在价值了），则该聚集称为**组合聚集**（简称为组成）。

在屏幕上打开一个窗口，它就由文本框、列表框、按钮和菜单组成，一旦关闭了窗口，各个组成部分也同时消失，窗口和它的组成部分之间存在着组合聚集关系。右图是窗口的组成，组合聚集的组成关系用实心菱形表示。



9.4 对象模型

3.泛化

UML中的泛化关系就是通常所说的继承关系，它是通用元素和具体元素之间的一种分类关系。具体元素完全拥有通用元素的信息，并且还可以附加一些其他信息。

在UML中，用一端为空心三角形的连线表示泛化关系，三角形的顶角紧挨着通用元素。

泛化关系指出在类与类之间存在“一般--特殊”关系。泛化可进一步划分成普通泛化和受限泛化。

(1) 普通泛化

没有具体对象的类称为抽象类。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。图示抽象类时，在类名下方附加一个标记值{abstract}。

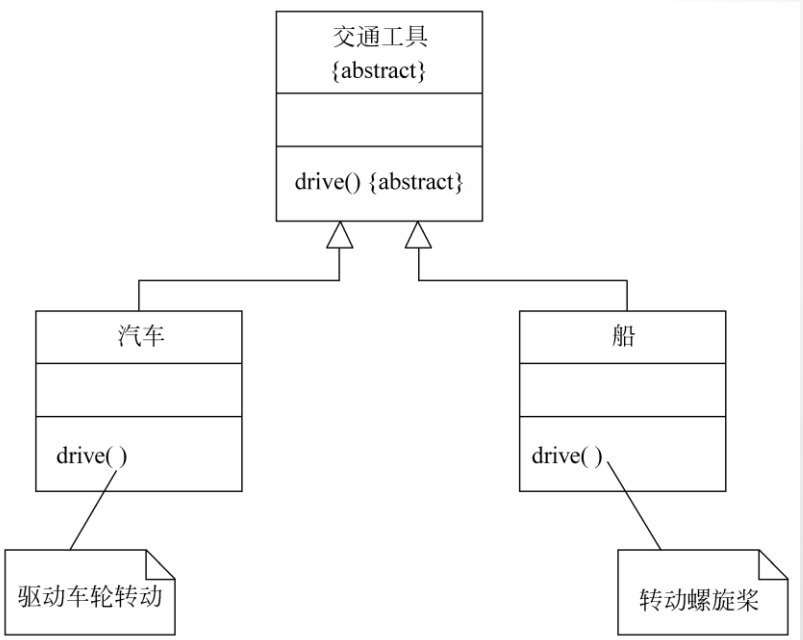
9.4 对象模型

3. 泛化

(1) 普通泛化

右图下方的两个折角矩形是模型元素“笔记”的符号，其中的文字是注释，分别说明两个子类的操作drive的功能。

抽象类通常都具有抽象操作。抽象操作仅用来指定该类的所有子类应具有哪些行为。抽象操作的图示方法与抽象类相似，在操作标记后面跟随一个性质串{abstract}。

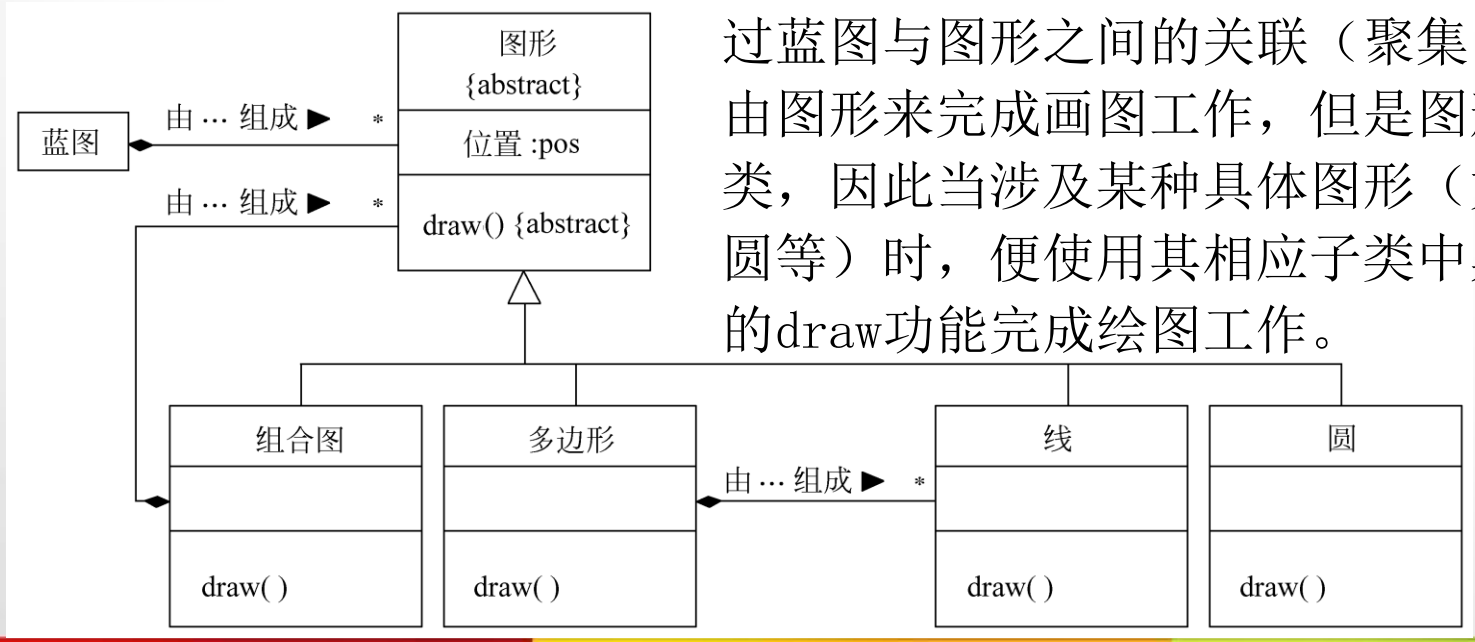


9.4 对象模型

3. 泛化

(1) 普通泛化

与抽象类相反的类是具体类，具体类有自己的对象，并且该类的操作都有具体的实现方法。



当客户要求画一幅蓝图时，系统便通过蓝图与图形之间的关联（聚集）关系，由图形来完成画图工作，但是图形是抽象类，因此当涉及某种具体图形（如直线、圆等）时，便使用其相应子类中具体实现的draw功能完成绘图工作。

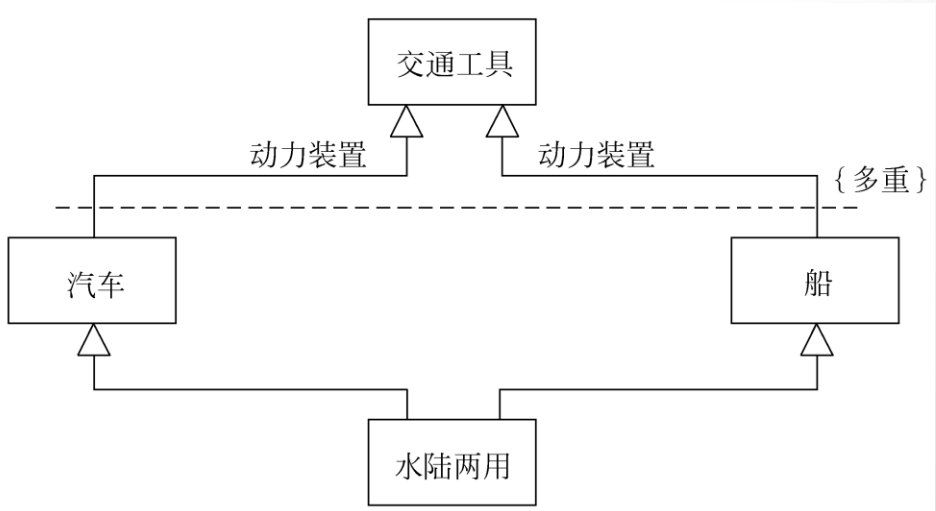
9.4 对象模型

3. 泛化

(2) 受限泛化

给泛化关系附加约束条件，以进一步说明该泛化关系的使用方法或扩充方法，这样的泛化关系称为受限泛化。预定义的约束有4种：**多重**、**不相交**、**完全**和**不完全**。这些约束都是语义约束。

多重继承指的是，一个子类可以同时多次继承同一个上层基类，右图中的水陆两用类继承了两次交通工具类。



9.4 对象模型

3. 泛化

(2) 受限泛化

与多重继承相反的是**不相交继承**，即一个子类不能多次继承同一个基类（这样的基类相当于C++语言中的虚基类）。如果图中没有指定{多重}约束，则是不相交继承，一般的继承都是不相交继承。

完全继承指的是父类的所有子类都已在类图中穷举出来了，图示符号是指定{完全}约束。

不完全继承与完全继承恰好相反，父类的子类并没有都穷举出来，随着对问题理解的深入，可不断补充和维护，这为日后系统的扩充和维护带来很大方便。不完全继承是一般情况下默认的继承关系。

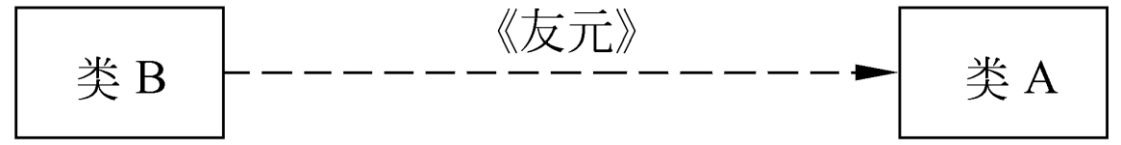
9.4 对象模型

4. 依赖和细化

(1) 依赖关系

依赖关系描述两个模型元素（类、用例等）之间的语义连接关系： 其中一个模型元素是独立的，另一个模型元素不是独立的，它依赖于独立的模型元素，如果独立的模型元素改变了，将影响依赖于它的模型元素。

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。在虚线上可以带一个版类标签，具体说明依赖的种类。



上图表示一个友元依赖关系，该关系使得B类的操作可以使用A类中私有的或保护的成员。

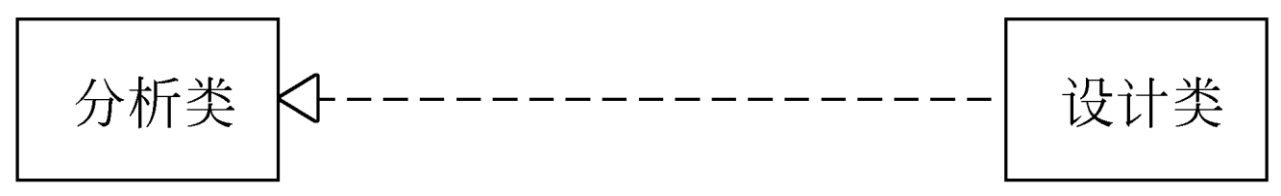
9.4 对象模型

4. 依赖和细化

(2) 细化关系

当对同一个事物在不同抽象层次上描述时，这些描述之间具有**细化关系**。

假设两个模型元素A和B描述同一个事物，它们的区别是抽象层次不同，如果B是在A的基础上的更详细的描述，则称B细化了A，或称A细化成了B。细化的图示符号为由元素B指向元素A的、一端为空心三角形的虚线（**注意**，不是实线），如下图所示。细化用来协调不同阶段模型之间的关系，表示各个开发阶段不同抽象层次的模型之间的相关性，常常用于跟踪模型的演变。



主要内容

9.1 面向对象方法学概述

9.2 面向对象的概念

9.3 面向对象模型

9.4 对象模型

9.5 动态模型

9.6 功能模型

9.7 3种模型之间的关系

9.5 动态模型

动态模型表示瞬时的、行为化的系统的“控制”性质，它规定了对象模型中的对象的合法变化序列。

所有对象都具有自己的生命周期（或称为运行周期）。生命周期中的阶段就是对象的状态。**状态**是对对象属性值的一种抽象。各对象之间相互触发（即作用）就形成了一系列的状态变化。人们把一个触发行为称作一个事件。对象对事件的响应，取决于接受该触发的对象当时所处的状态，响应包括改变自己的状态或者又形成一个新的触发行为。

9.5 动态模型

状态有持续性，它占用一段时间间隔。状态与事件密不可分，一个事件分开两个状态，一个状态隔开两个事件。事件表示时刻，状态代表时间间隔。

通常，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）。

每个类的动态行为用一张状态图来描绘，各个类的状态图通过共享事件合并起来，从而构成系统的动态模型。也就是说，动态模型是基于事件共享而互相关联的一组状态图的集合。

主要内容

9.1 面向对象方法学概述

9.2 面向对象的概念

9.3 面向对象模型

9.4 对象模型

9.5 动态模型

9.6 功能模型

9.7 3种模型之间的关系

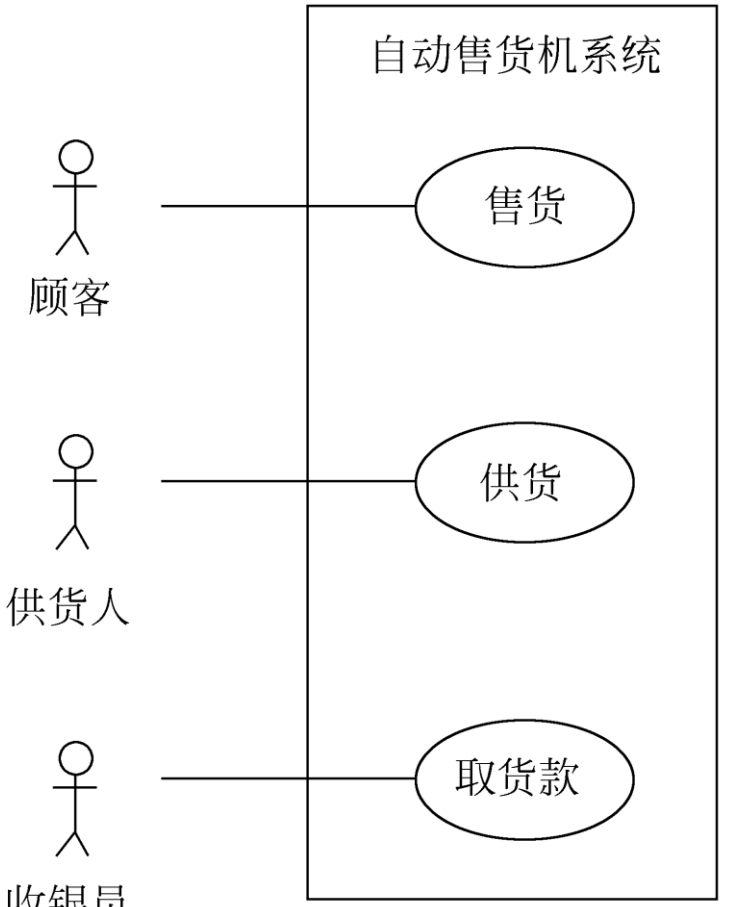
9.6 功能模型

- **功能模型**表示变化的系统的“功能”性质，它指明系统应该“做什么”，因此更直接地反映了用户对目标系统的需求。
- **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地理解问题域，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**。
- 使用**用例模型**代替传统的功能说明，往往能够更好地获取用户需求，它所回答的问题是“系统应该为每个（或每类）用户做什么”。
- **用例模型**描述的是外部行为者(actor）所理解的系统功能。用例模型的建立是系统开发者和用户反复讨论的结果，它描述了开发者和用户对需求规格所达成的共识。

9.6 功能模型

9.6.1. 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。右图是自动售货机系统的用例图。图中的方框代表系统，椭圆代表用例（售货、供货和取货款是自动售货机系统的典型用例），线条人代表行为者，它们之间的连线表示关系。



9.6 功能模型

1. 系统

系统被看作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例模型来说都是不重要的。

代表**系统**的方框的边线表示系统的边界，用于划定系统的功能范围，定义了系统所具有的功能。描述该系统功能的用例置于方框内，代表外部实体的行为者置于方框外。

2. 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。在UML中把**用例**定义成系统完成的一系列动作，动作的结果能被特定的行为者察觉到。这些动作除了完成系统内部的计算与工作外，还包括与一些行为者的通信。用例通过**关联**与行为者连接，**关联**指出一个用例与哪些行为者交互，这种交互是双向的。

9.6 功能模型

2. 用例

用例具有下述特征。

(1)用例代表某些用户可见的功能，实现一个具体的用户目标。

(2)用例总是被行为者启动的，并向行为者提供可识别的值。

(3)用例必须是完整的。

注意，用例是一个类，它代表一类功能而不是使用该功能的某个具体实例。用例的实例是系统的一种实际使用方法，通常把用例的实例称为**脚本**。脚本是系统的一次具体执行过程，例如，在自动售货机系统中，张三投入硬币购买矿泉水，系统收到钱后把矿泉水送出来，上述过程就是一个脚本；李四投币买可乐，但是可乐已卖完了，于是系统给出提示信息并把钱退还给李四，这个过程是另一个脚本。

9.6 功能模型

3. 行为者

行为者是指与系统交互的人或其他系统，它代表外部实体。使用用例并且与系统交互的任何人或物都是行为者。

行为者代表一种角色，而不是某个具体的人或物。一个具体的人可以充当多种不同角色。

在用例图中用直线连接行为者和用例，表示两者之间交换信息，称为**通信联系**。行为者触发(激活)用例，并与用例交换信息。单个行为者可与多个用例联系；一个用例也可与多个行为者联系。

可以把行为者分成主行为者和副行为者，还可分成主动行为者和被动行为者。

9.6 功能模型

4.用例之间的关系

UML用例之间主要有**扩展**和**使用**两种关系，它们是泛化关系的两种不同形式。

（1）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是扩展关系，后者继承前者的一些行为，通常把后者称为扩展用例。

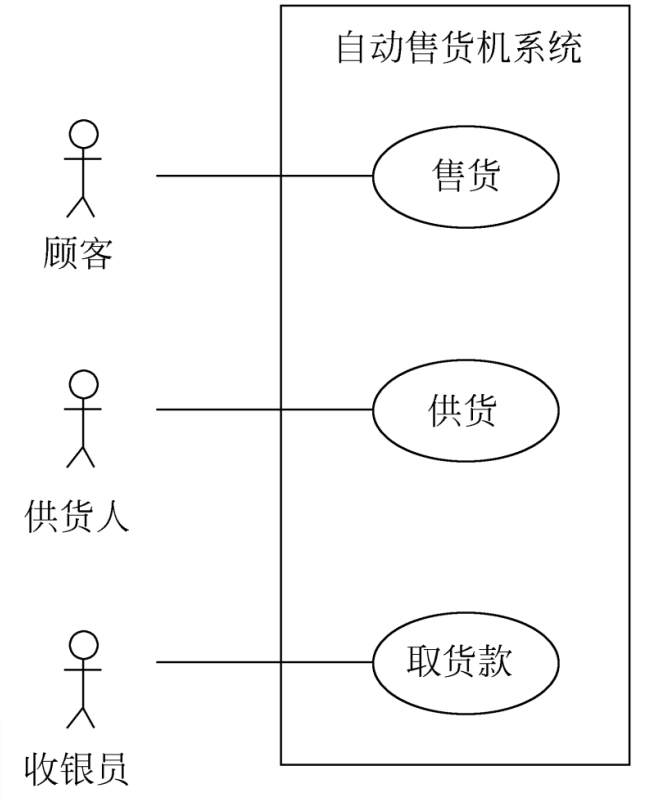
（2）使用关系

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。一般说来，如果在若干个用例中有某些相同的动作，则可以把这些相同的动作提取出来单独构成一个用例（称为抽象用例）。这样，当某个用例使用该抽象用例时，就好像这个用例包含了抽象用例中的所有动作。

9.6 功能模型

4. 用例之间的关系

右图为含扩展和使用关系的用例图。



注意扩展与使用之间的异同：这两种关系都意味着从几个用例中抽取那些公共的行为并放入一个单独的用例中。通常在描述一般行为的变化时采用扩展关系；在两个或多个用例中出现重复描述又想避免这种重复时，可以采用使用关系。

9.6 功能模型

9.6.2. 用例建模

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，描述用例，定义用例之间的关系，确认模型。其中，寻找行为者和用例是关键。

1. 寻找行为者

为获取用例首先要找出系统的行为者，可通过请系统的用户回答一些问题来发现行为者。下述问题有助于发现行为者。

- 谁将使用系统的主要功能（主行为者）？
- 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（副行为者）？
- 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（值）感兴趣？

9.6 功能模型

2. 寻找用例

一旦找到了行为者，就可以通过请每个行为者回答下述问题来获取用例。

- 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情吗？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？

9.6 功能模型

2. 寻找用例

还有一些不是针对具体行为者而是针对整个系统的问题，也能帮助建模者发现用例，例如：

- 系统需要哪些输入输出？输入来自何处？输出到哪里去？
- 当前使用的系统（可能是人工系统）存在的主要问题是什么？

注意，最后这两个问题并不意味着没有行为者也可以有用例，只是在获取用例时还不知道行为者是谁。事实上，**一个用例必须至少与一个行为者相关联。**

主要内容

9.7 3种模型之间的关系

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(即在何种状态下接受了什么事件的触发)做；**对象模型**则定义了做事情的实体。

在面向对象方法学中，对象模型是最基本最重要的，它为其他两种模型奠定了基础，人们依靠对象模型完成3种模型的集成。下面扼要地叙述3种模型之间的关系。

- 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。
- 状态转换驱使行为发生，这些行为在数据流图中被映射成处理，在用例图中被映射成用例，它们同时与类图中的服务相对应。

9.7 3种模型之间的关系

- 功能模型中的处理（或用例）对应于对象模型中的类所提供的服务。通常，复杂的处理（或用例）对应于复杂对象提供的服务，简单的处理（或用例）对应于更基本的对象提供的服务。有时一个处理（或用例）对应多个服务，也有一个服务对应多个处理（或用例）的时候。
- 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。
- 数据流图中的数据流，往往是对象模型中对象的属性值，也可能是整个对象。
- 用例图中的行为者，可能是对象模型中的对象。
- 功能模型中的处理（或用例）可能产生动态模型中的事件。
- 对象模型描述了数据流图中的数据流、数据存储以及数据源点/终点的结构。

本章小结

1. 面向对象范型明显优于结构化范型，使用面向对象范型能够开发出稳定性好、可重用性好和可维护性好的软件。

2. 面向对象方法学比较自然地模拟了人类认识客观世界的思维方式，在结构上尽可能一致。

3. 系统中每个对象都属于一个特定的对象类。类是对具有相同属性和行为的一组相似对象的定义。按照子类、父类的关系，把众多的类进一步组织成一个层次系统，处于下一层次上的类可以自动继承位于上一层次的类的属性和行为。

4. 用面向对象观点建立系统的模型，分别是描述系统静态结构的对象模型、描述系统控制结构的动态模型以及描述系统计算结构的功能模型。其中，对象模型是最基本、最核心、最重要的。

5. 统一建模语言UML是国际对象管理组织OMG批准的基于面向对象技术的标准建模语言。使用UML的类图来建立对象模型，使用UML的状态图来建立动态模型，使用数据流图或UML的用例图来建立功能模型。在UML中把用用例图建立起来的系统模型称为用例模型。

本章结束