# 第 5 章

# 类图和对象图

## 5.1 类 的 定 义

在 UML 中，有两个图非常重要，一个是第 3 章中介绍的用例图，另一个就是本章将要介绍的类图。Rumbaugh 对类的定义是：类是具有相似结构、行为和关系的一组对象的描述符[24]。在 UML 中，类表示为划分成 3 个格子的长方形，如图 5.1 所示。

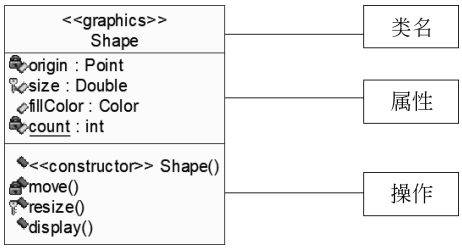


**图 5.1 UML 中表示类的符号**

在如图 5.1 所示的类中，类名是 Shape，共有 4 个属性，分别为 origin、size、fillColor 和 count。其中，属性 count 有一下画线，表示该属性是静态(static)属性。Shape 类有 shape()、move()、resize()和 display()4 个方法。其中，方法 shape()的版型为<<Constructor>>，表示该方法是构造方法，而 Shape 类是一个版型为<<Graphics>>的类。对于版型的定义在 5.5 节中还会介绍。

在定义类的时候，类的命名应尽量用应用领域中的术语，应明确、无歧义，以利于开发人员与用户之间的相互理解和交流。一般而言，类的名字是名词。在 UML 中，类的命名分为 simple name 和 path name 两种形式。其中，simple name 形式的类名就是简单的类的名字，而 path name 形式的类名还包括包名。例如，下面是 path name 形式的类名。

```
Banking::CheckingAccount
```

其中，Banking 是包名，CheckingAccount 是包 Banking 中的一个类。

### 5.1.1 类的属性

属性在类图标的属性分隔框中用文字串说明，UML 规范说明 1.5 版本中定义属性的格式为

```
[可见性]属性名[:类型]['['多重性[次序]']'][=初始值][{特性}]
```

根据详细程度的不同，每条属性可以包括属性的可见性、属性名称、类型、多重性、初始值和特性。其中，特性是用户对该属性性质的一个约束说明。例如，{只读}这样的特性说明该属性的值不能被修改。

上面表示属性的格式中，除了用‘’括起来的方括号表示的是一个具体的字符外，其他方括号表示该项是可选项。

**例 5.1**　属性声明的一些例子。

```
+size:Area=(100,100)
# visibility:Boolean=false
+default-size:Rectangle
-xptr:XwindowPtr
colors:Color[3]
points:Point[2..*ordered]
name:String[0..1]
```

需要说明的是，对属性可见性(Visibility)的表示，UML 和 Rose 采用不同的符号，UML 规范中规定的是用＋、＃、－等符号，而 Rose 中采用、、等图形符号标识(见图 5.1)。

对于例 5.1 中的 points 属性和 name 属性，需要注意它们的多重性部分。多重性声明并不是表示数组的意思。points 的多重性为 2..＊，表示该属性值有两个或多个，同时这些值之间是有序的(因为有 ordered 指明)。而 name 这个属性的多重性为[0..1]，表示 name 有可能有一个值，也有可能值为 null。特别需要注意的是，name:String[0..1]并不表示 name 是一个 String 数组。

从理论上讲，一个类可以有无限多个属性，但一般不可能把所有的属性都表示出来，因此在选取类的属性时应只考虑那些系统会用到的特征。原则上，由类的属性应能区分每个特定的对象。

### 5.1.2　类的操作

操作(Operation)用于修改、检索类的属性或执行某些动作，通常也称为功能。但是它们被约束在类的内部，只能作用到该类的对象上。操作在类图标的操作分隔框中用文字串说明，UML 规范说明 1.5 中规定操作的格式为

```
[可见性]操作名[(参数列表)][:返回类型][{特性}]
```

其中，方括号表示该项是可选项，而{特性}是一个文字串，说明该操作的一些有关信息。例如，{query}这样的特性说明表示该操作不会修改系统的状态。操作名、参数列表和返回类型组成操作接口。接口与第 2 章中所介绍的操作的特征标记(Signature)的概念很相似，但也有细微的差别。操作的特征标记一半只包括操作名和参数列表，而不包括返回类型，但接口是包括返回类型的。

**例 5.2**　操作声明的一些例子。

```
+display():Location
```

```
+hide()
#create()
-attachXWindow(xwin:XwindowPtr)
```

需要说明的是，对操作可见性(Visibility)的表示，UML 和 Rose 采用不同的符号，UML 规范中规定的是用＋、＃、－等符号，而 Rose 中采用、、等图形符号标识(见图 5.1)。

## 5.2 类之间的关系

一般来说，类之间的关系有关联、聚集、组合、泛化、依赖等，下面将对这些关系进行详细说明。

### 5.2.1 关联

关联(Association)是模型元素间的一种语义联系，它是对具有共同的结构特征、行为特性、关系和语义的链(Link)的描述。

在上面的定义中，需要注意链这个概念，链是一个实例，就像对象是类的实例一样，链是关联的实例，关联表示的是类与类之间的关系，而链表示的是对象与对象之间的关系。

在类图中，关联用一条把类连接在一起的实线表示，如图 5.2 所示。

一个关联可以有两个或多个关联端，每个关联端连接到一个类。关联也可以有方向，可以是单向关联或双向关联。图 5.2 表示的是双向关联，图 5.3 表示的是从类 A 到类 B 的单向关联。


图 5.2 类之间的关联关系


图 5.3 类之间的单向关联关系

关联是类图中非常重要的一种关系，这里以实现时相对应的 Java 代码来帮助理解关联关系。可以在 Rose 中创建如图 5.3 所示的类图，并用 Rose 生成 Java 代码，代码如下。

类 A 的代码：

```
public class A
{
    public B theB;
    /**
     *@ roseuid 3DAFBF0F01FC
     */
    public A()
    {
    }
}
```

类 B 的代码：

```
public class B
{
    /**
     *@ roseuid 3DAFBF0F01A2
     */
    public B()
    {
    }
}
```

从上面的代码中可以看到，在类 A 中有一个属性 theB，其类型为 B，而在类 B 中没有相应的类型为 A 的属性。如果把这个单向关联改为双向关联，则生成的类 B 的代码中，会有相应的类型为 A 的属性。

在上面的代码中，分别有类 A 和类 B 的构造方法生成。Rose 在生成代码时，默认情况下会生成构造方法。在这个例子中，采用系统的默认配置，即要求生成构造方法。如果不想构造方法，可以对 Rose 的 Tools Options Java 的 Class 选项的 GenerateDefaultConstructor 属性进行设置。如果设置为 FLASE 即要求不生成构造方法（该属性的默认值为 TRUE）。

另外，代码中有类似@roseuid 3DAFBF0F01FC 这样的语句，称作代码标识号。它的作用是标识代码中的类、操作和其他模型元素。在双向工程（正向工程和逆向工程）中，可以使代码和模型同步。

**1. 关联名**

可以给关联加上关联名，来描述关联的作用。如图 5.4 所示是使用关联名的一个例子，其中，公司（Company）类和人（Person）类之间的关联如果不使用关联名，则可以有多种解释，如 Person 类可以表示公司的客户、雇员或所有者等。但如果在关联上加上雇佣（Employs）这个关系名，则表示公司类和人类之间是雇佣（Employs）关系，显然这样语义上更加明确。一般说来，关联名通常是动词或动词短语。

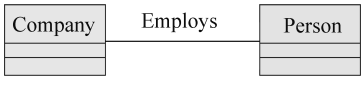


**图 5.4　使用关联名的关联示例**

当然，在一个类图中，并不需要给每个关联都加上关联名，给关联命名的原则应该是该命名有助于理解该模型。事实上，一个关联如果表示的意思已经很明确了，再给它加上关联名，反而会使类图变乱，只会起到画蛇添足的作用。

**2. 关联的角色**

关联两端的类可以某种角色参与关联。例如，在图 5.5 中，公司类以雇佣者（employer）的角色、人类以被雇佣者（employee）的角色参与关联，employer 和 employee 称为角色名。如果在关联上没有标出角色名，则隐含地用类的名称作为角

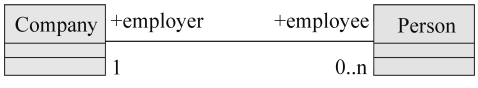


**图 5.5　关联的角色示例**

色名。

角色还具有多重性，表示可以有多少个对象参与该关联。在图 5.5 中，雇主(公司)可以雇佣多个雇员，表示为 0..n；雇员只能被一家雇主雇佣，表示为 1。

在 UML 中，多重性可以用下面的格式表示。

```
0..1
0..*(也可以表示为 0..n)
1  (1..1 的简写)
1..*(也可以表示为 1..n)
*   (即 0..n)
7
3,6..9
0  (0..0 的简写)(表示没有实例参与关联，一般不用)
```

可以看到，多重性是用非负整数的一个子集表示的。

**3. 关联类**

关联本身也可以有特性，通过关联类(association class)可以进一步描述关联的属性、操作以及其他信息。关联类通过一条虚线与关联连接。图 5.6 中的合同(Contract)类是一个关联类，合同类中有属性工资(salary)，这个属性描述的是公司类和人类之间的关联的属性，而不是描述公司类(Company)或人类(Person)的属性。

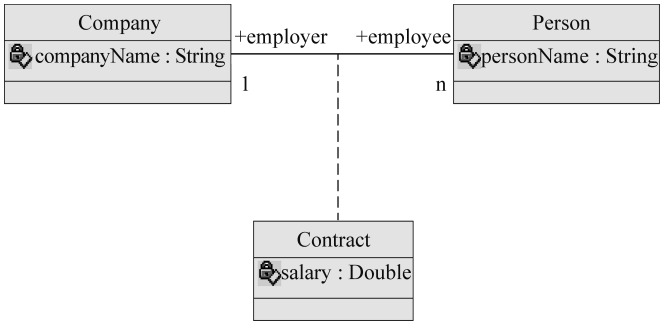


**图 5.6　使用关联类的关联**

为了有助于理解关联类，这里也用 Rose 生成相应的 Java 代码，共 3 个类，如下。

类 Company 的代码：

```
public class Company
{
    private String companyName;
    public Person employee[];
}
```

类 Person 的代码：

```
public class Person
{
```

```
    Private String personName;
    protected Company employer;
}
```

类 Contract 的代码：

```
public class Contract
{
    private Double salary;
}
```

由于指定了关联角色的名字，所以生成的代码中就直接用关联，角色名作为所声明的变量的名字，如 employee、employer 等。另外，employer 的可见性是 protected，也是在生成的代码中体现出来的。

因为指定关联的 employee 端的多重性为 $n$，所以在生成的代码中，employee 是类型为 Person 的数组。

另外，可以发现所生成的 Java 代码中都没有构造方法。这是因为在生成代码前，已经把 Rose 的 Tools→Options→Java 的 Class 选项的 GenerateDefaultConstructor 属性设置为 FALSE，即要求生成代码时不生成类的默认构造方法。

**4. 关联的约束**

对于关联可以加上一些约束，以加强关联的含义。如图 5.7 所示是两个关联之间存在异或约束的例子，即账户（Account）类或者人（Person）类有关联，或者与法人（Corporation）类有关联，但不能同时与 Person 类和 Corporation 类都有关联。

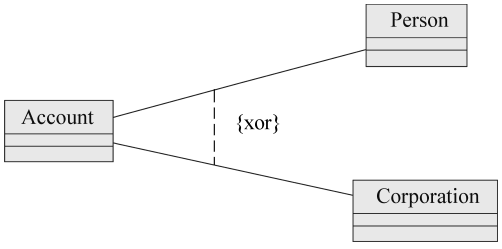


**图 5.7　带约束的关联**

约束是 UML 中的三种扩展机制之一，另外两种扩展机制是版型（Stereotype）和标记值（tagged value）。

当然，约束不仅可以作用在关联这个建模元素上，也可以作用在其他建模元素上。

**5. 限定关联**

在关联端紧靠源类图标处可以有限定符，带有限定符的关联称为限定关联。限定符的作用就是在给定关联一端的一个对象和限定符值以后，可确定另一端的一个对象或对象集。

使用限定符的例子如图 5.8 所示。

图 5.8 表示的意思是，一个 Person 可以在 Bank 中有多个 account，但给定了一个 account 值后，就可以对应一个 Person 值，或者对应的 Person 值为 NULL，因为 Person

端的多重性为 0..1。这里的多重性表示的是 Person 和(Bank,account)之间的关系,而不是 Person 和 Bank 之间的关系。即

```
(Bank,account)-> 0个或者1个 Person
Person-> 多个(Bank, account)
```

但图 5.8 中并没有说明 Person 类和 Bank 之间是一对多的关系还是一对一的关系,既可能一个 Person 只对应一个 Bank,也可能一个 Person 对应多个 Bank。如果一定要明确一个 Person 对应的是一个 Bank 还是多个 Bank,则需要在 Person 类和 Bank 类之间另外增加关联来描述。如图 5.9 所示表示一个 Person 可以对应一个或多个 Bank。

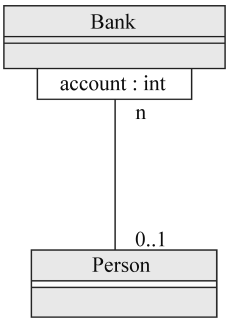


图 5.8 限定符和限定关联

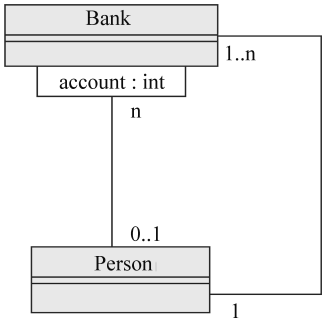


图 5.9 限定关联和一般关联

需要注意的是,限定符是关联的属性,而不是类的属性。也就是说,在具体实现图 5.8 中的结构时,account 这个属性有可能是 Person 类中的一个属性,也可能是 Bank 类中的一个属性(当然,这里在 Bank 类中包含 account 属性并不好),也可能是在其他类中有一个 account 属性。

限定符这个概念在设计软件时非常有用,如果一个应用系统需要根据关键字对一个数据集做查询操作,则经常会用到限定关联。引入限定符的一个目的就是把多重性从 $n$ 降为 0..1,这样如果做查询操作,则返回的对象至多是一个,而不会是一个对象集。如果查询操作的结果是单个对象,则这个查询操作的效率会较高。所以在使用限定符时,如果限定符另一端的多重性仍为 $n$,则引入这个限定符的作用就不是很大。因为查询结果仍然还是一个结果集,所以也可以根据多重性来判断一个限定符的设计是否合理。

**6. 关联的种类**

按照关联所连接的类和数量,类之间的关联可分为自返关联、二元关联和 $N$ 元关联共三种关联。

自返关联又称为递归关联,是一个类与自身的关联,即同一个类的两个对象间的关系。自返关联虽然只有一个被关联的类,但有两个关联端,每个关联端的角色不同。自返关联的例子如图 5.10 所示。

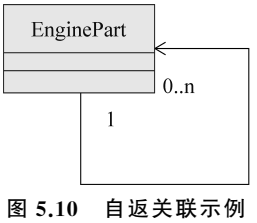


图 5.10 自返关联示例

对于图 5.10 中的类,在 Rose 中所生成的 Java 代码如下。

类 EnginePart 的代码如下:

```
public class EnginePart
{
    public EnginePart theEnginePart[];

    /**
     *@ roseuid 3E920390281
     */
    public EnginePart()
    {
    }
}
```

二元关联是两个类之间的关联，对于二元关联，前面已经举了很多例子，这里就不再举例说明了。

*N* 元关联是在三个或三个以上类之间的关联。*N* 元关联的例子如图 5.11 所示，Player、Team 和 Year 这三个类之间存在三元关联，而 Record 类是关联类。*N* 元关联中多重性的意义是：在其他 *N*－1 个实例值确定的情况下，关联实例元组的个数。如在图 5.11 中，多重性表示的意思是在某个具体年份(Year)和运动队(Team)中，可以有多个运动员(Player)；一个运动员在某一个年份中，可以在多个运动队服役；同一个运动员在同一个运动队中可以服役多年。

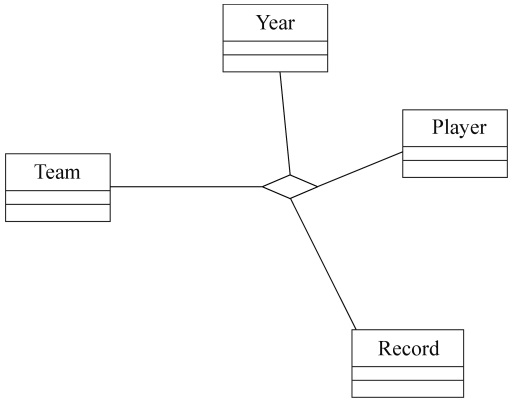


**图 5.11　*N* 元关联示例**

*N* 元关联没有限定符的概念，也没有聚集、组合等概念(在 5.2.2 节中将介绍聚集、组合的概念)。

需要说明的是，在 UML 的规范说明中，有 *N* 元关联这个建模元素，用菱形表示。在 Rose 2003 中，并不能直接表示 *N* 元关联，但可以在类图中创建一个类的版型来模拟画出 *N* 元关联(图 5.11 即是在 Rose 2003 中增加了一个表示 *N* 元关联的版型后画出的)。至于如何在 Rose 2003 中加入用户自己要用的版型，这涉及 Rose 的扩展机制，在第 17 章中介绍 Rose 2003 开发工具时再详细讨论。

### 5.2.2 聚集和组合

聚集(Aggregation)是一种特殊形式的关联。聚集表示类之间整体与部分的关系。在对系统进行分析和设计时,需求描述中的“包含”“组成”“分为…部分”等词常常意味着存在聚集关系。

组合(Composition)表示的也是类之间的整体与部分的关系,但组合关系中的整体与部分具有同样的生存期。也就是说,组合是一种特殊形式的聚集。

如图 5.12 和图 5.13 所示分别是聚集关系和组合关系的例子。

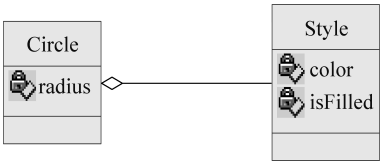


图 5.12 聚集关系

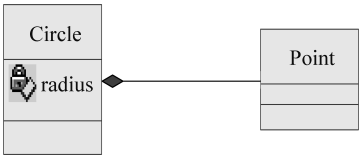


图 5.13 组合关系

图 5.12 中的圆(Circle)类和样式(Style)类之间是聚集关系。一个圆可以有颜色、是否填充这些样式(Style)方面的属性,可以用一个 Style 对象表示这些属性,但同一个 Style 对象也可以表示别的对象如三角形(Triangle)的一些样式方面的属性,也就是说,Style 对象可以用于不同的地方。如果 Circle 这个对象不存在了,不一定意味着 Style 这个对象也不存在了。

图 5.13 中的 Circle 类和 Point 类之间是组合关系。一个圆可以由半径和圆心确定,如果圆不存在了,那么表示这个圆的圆心也就不存在了,所以 Circle 类和 Point 类是组合关系。

聚集关系的实例是传递的、反对称的,也就是说,聚集关系的实例之间存在偏序关系,即聚集关系的实例之间不能形成环。需要注意的是,这里说的是聚集关系的实例(即链)不能形成环,而不是说聚集关系不能形成环。事实上,聚集关系可以形成环。

在类图中,使用聚集关系和组合关系的好处是简化了对象的定义,同时支持分析和设计时类的重用。

聚集和组合是类图中很重要的两个概念,但也是比较容易混淆的概念,在实际运用时往往很难确定是用聚集关系还是组合关系。事实上,在设计类图时,设计人员是根据需求分析描述的上下文来确定是使用聚集关系还是组合关系。对于同一个设计,可能采用聚集关系和采用组合关系都是可以的,不同的只是采用哪种关系更贴切些。

下面列出聚集和组合之间的一些区别。

聚集关系,也称为“has-a”关系,组合关系也称为“contains-a”关系。

聚集关系表示事物的整体/部分关系较弱的情况,组合关系表示事物的整体/部分关系较强的情况。

在聚集关系中,代表部分事物的对象可以属于多个聚集对象,可以为多个聚集对象所共享,而且可以随时改变它所从属的聚集对象。代表部分事物的对象与代表聚集事物对象的生存期无关,一旦删除了它的一个聚集对象,不一定也就随即删除代表部分事物的对

象。在组合关系中，代表整体事物的对象负责创建和删除代表部分事物的对象，代表部分事物的对象只属于一个组合对象。一旦删除了组合对象，也就随即删除了相应的代表部分事物的对象。

### 5.2.3　泛化关系

泛化(Generalization)定义了一般元素和特殊元素之间的分类关系，如果从面向对象程序设计语言的角度来说，类与类之间的泛化关系就是平常所说的类与类之间的继承关系。

泛化关系也称为“a-kind-of”关系。在 UML 中，泛化关系不仅是类与类之间才有，像用例、参与者、关联、包、构件(Component)、数据类型(Data Type)、接口(Interface)、节点(Node)、信号(Signal)、子系统(Subsystem)、状态(State)、事件(Event)、协作(Collaboration)等这些建模元素之间也可以有泛化关系。

UML 中用一头为空心三角形的连线表示泛化关系。如图 5.14 所示是类之间泛化关系的例子。

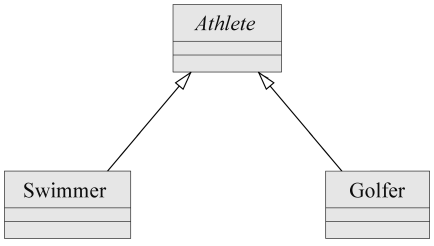

**图 5.14　泛化关系示例**

在图 5.14 中，Swimmer 类和 Golfer 类是对 Athlete 类的泛化，其中，Athlete 类的名字用斜体表示，表示该类是一个抽象类，而 Swimmer 类和 Golfer 类的名字没有用斜体，表示这两个类是具体类。

### 5.2.4　依赖关系

假设有两个元素 X、Y，如果修改元素 X 的定义可能会导致对另一个元素 Y 的定义的修改，则称元素 Y 依赖于元素 X。

对于类而言，依赖(Dependency)关系可能由各种原因引起，如一个类向另一个类发送消息，或者一个类是另一个类的数据成员类型，或者一个类是另一个类的操作的参数类型等。如图 5.15 所示是类之间依赖关系的例子，其中，Schedule 类中的 add()操作和 remove()操作都有类型为 Course 的参数，因此 Schedule 类依赖于 Course 类。

有时依赖关系和关联关系比较难区分。事实上，如果类 A 和类 B 之间有关联关系，那么类 A 和类 B 之间也就有依赖关系了。但如果两个类之间有关联关系，那么一般只要表示出关联关系即可，不用再表示这两个类之间还有依赖关系。而且，如果在一个类图中有过多的依赖关系，反而会使类图难以理解。

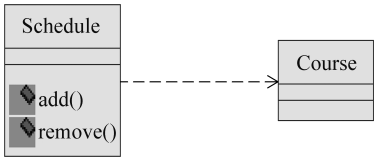

**图 5.15　依赖关系示例**

与关联关系不一样的是，依赖关系本身不生成专门的实现代码。

另外，与泛化关系类似，依赖关系也不只是限于类之间，其他建模元素，如用例与用例之间，包与包之间也可以有依赖关系。