Chapter 5 第5章 UML 类图

UML 类图(Class Diagram)描述了类以及类之间的静态关系。与传统软件工程中的结构化分析使用的数据流模型不同,UML 类图不仅显示了信息的结构,还描述了对象的行为。以 UML 类图为基础的状态图、顺序图、协作图进一步描述了系统其他方面的动态特性。

本章将首先介绍 UML 类图,然后阐述类之间的关系,并解释 UML 中的边界类、控制类和实体类的概念,最后以学生成绩管理系统为例,阐述在 Rose 建模环境下创建 UML 类图的方法和步骤。

5.1 概述

作为一种语言,UML 定义了一系列的图以描述软件密集型系统。UML 图有着严格的语义和清晰的语法,这些图及其定义的语义和语法组成了一个标准,使得从事软件开发的所有相关人员都能够借助它们对软件系统的各个侧面进行描述。

类是具有相似结构、行为和关系的一组对象的描述符。类图是描述类以及类之间的关系的一种图。类图从静态角度表示软件系统,属于一种静态模型。类之间的关系有关联(Association)关系、泛化(Generalization)关系、依赖(Dependency)关系、实现(Realization)关系等。

5.2 类的定义

在 UML 中,类可以用划分为 3 个格子的长方形表示,上面是类名,中间是属性,下面是操作。类图就是由这些类框和表明这些类框之间的关系的连线组成的。图 5.1 是一个表示 Java 类的 UML 类图,以及由 Account 类生成的 Java 源代码(后续内容将讲述如何用 Rose 的正向工程根据 UML 类图自动生成 Java 源代码)。

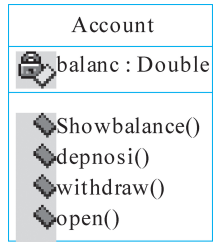

图 5.1 UML 类图的示例

```
1:  public class Account  {
2:    private Double balance;
3:    public Boolean showBalance( ) {  }
3:    public Boolean deposit( ) {  }
4:    public Boolean withdraw( ) {  }
5:    public Account open( ) {  }
6:  }
```

1. 类的属性

在 UML 中，类的属性的语法格式如下。

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

- 上面表示的属性格式中，除了用单撇号括起来的方括号表示一个具体的字符外，其他方括号均表示该项是可选项。
- 属性的可见性(visibility)表示。在 UML 中，＋、＃、－等符号分别表示 public、protected、private，而 Rose 中用的是图形符号。
- 多重性声明不是表示数组，而是表示该属性值有两个或多个，这些值可以是有序的，约束特性只是对该属性的一个声明。

2. 类的操作

在 UML 中，类的操作的定义语法格式如下(各项的含义与属性的说明相同)。

```
[可见性] 操作名[(参数列表)][:类型][{约束特性}]
```

5.3　关联关系

在 UML 中，类之间的语义连接被定义为关系。在软件系统中，各种动态行为的实现是由对象之间的交互产生的，所以类之间的关系建模就为类的对象之间的交互提供了实现支持。对象之间的交互可以与类之间的关系相对应，而这些关系又可以被映射到大多数的程序设计语言中，从而使得对象之间的交互能得到最终实现。

5.3.1　关联

关联(Association)表示两个类之间存在的某种语义上的联系，它是对具有共同的结构特性、行为特性、关系和语义的链接(Link)的描述，即与该关联连接

的类的对象之间的语义连接(称为链接)。注意:“关联”表示类与类之间的关系,而“链接”表示对象与对象之间的关系,即链接是关联的实例。

关联的表示方法就是在有关系的类之间画一条直线。关联可以是单向的,也可以是双向的。单向关联用带箭头的直线表示,双向关联用一条直线表示。单向关联表示从箭头端出发的类的对象,可以调用箭头指向端的类中的方法。在 Java 语言中,这个关联是可调用方法的这个类中的实例变量。单向关联的示例如图 5.2 所示。

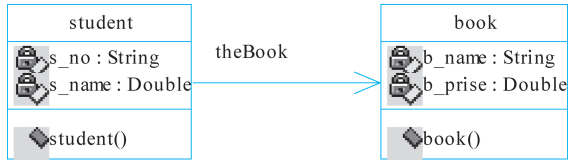

图 5.2 单向关联的示例

图 5.2 所示的两个类生成的 Java 源代码如下所示。

```
1.  public class student {
2.     private String s_no;
3.     private Double s_name;
4.     public book theBook;
5.     public student( ) {
6.     }
7.  }
1.  public class book {
2.     private String b_name;
3.     private Double b_prise;
4.     public book( ) {
5.     }
6.  }
```

从生成的 Java 代码中可以看出,在类 student 中,有一个属性 theBook(关联名),其类型为 book。而在类 book 中,却没有相应类型为 student 的属性。

双向关联表示关联中的两个对象可以互相调用其方法。双向关联的示例如图 5.3 所示。

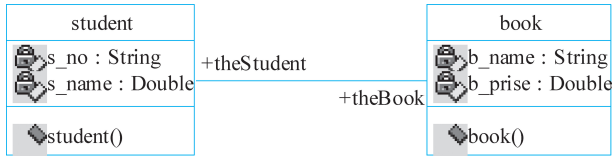

图 5.3 双向关联的示例

在类 student 中，有一个属性 theBook，其类型为 book；在类 book 中，也有一个属性 theStudent，其类型为 student。在 UML 中，关联的两端都是角色，而且都可以命名。如果用 Java 语言实现图 5.3 中的双向关联关系，则角色的名称应与各个类中的实例变量的名称相同。图 5.3 所示的两个类生成的 Java 源代码如下。

```
1.  public class student {
2.  private String s_no;
3.  private Double s_name;
4.  public book theBook;
5.  public student( ) {
6.  }
7.  }
1.  public class book {
2.  private String b_name;
3.  private Double b_prise;
4.  public book theStudent;
5.  public book( ) {
6.  }
7.  }
```

5.3.2　关联类

关联也可以有自己的属性和操作，此时，这个关联称为关联类（Association Class）。关联类的可视化表示方式与一般类相同，但是要用一条虚线把关联类和对应的关联线连接起来。如图 5.4 所示，contract 是一个关联类，contract 类中的属性 salary 描述的是 company 类和 person 类之间的关联的属性，而不是其他类的属性。

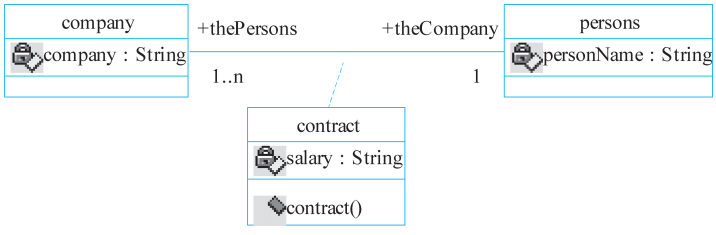

图 5.4　关联类的示例

由于指定了关联角色的名字，所以生成的 Java 语言源代码直接使用了关联角色名作为声明的变量名。另外，指定关联的 thePerson 端的多重性为 1..n，所

以 thePerson 是类型为 persons 的数组。图 5.4 所示的两个类生成的 Java 源代码如下。

```
1.  public class company {
2.     private String company companyName;
3.     public persons thePersons[ ];
4.  }
1.  public class persons {
2.     private String personName;
3.     public company theCompany;
4.  }
1.  public class contract {
2.     private String salary;
3.  }
```

5.3.3 多重性

一个类的关联的任何一个连接点称为关联端(Association End),与类有关的许多信息都附在它的端点上。关联端有名字(角色名)和可见性等特性,最重要的特性是多重性(Multiplicity)。多重性是对象之间关联的一个重要方面,它说明了某个类有多少个对象可以和另一个类的单个对象关联。例如,学校中的一门课程如果由一名教师讲授,那么课程和教师之间就是一对一(one-to-one)的关联;如果一门课程由多名教师讲授,那么课程与教师之间就是一对多(one-to-many)的关联。

在 UML 中,使用“*”代表许多(more)和多个(many)。在一种语义环境中,两个点代表 or 关系,例如,“1..*”代表一个或多个。在另一种语义环境中,or 用逗号表示,例如“5,10”代表 5 或者 10。多重性的符号表示如下所示(默认值是 1)。

- “0..1”——表示 0 或者 1。
- “0..*”或“*”——表示 0 个或者多个。
- “1..*”——表示 1 个或者多个。
- “3..16”——表示 3～16。
- “1,3,16”——表示或者 1,或者 3,或者 16。

在关联的两端可以写上一个数值范围,表示该类有多少个对象可以与对方的一个对象连接,即多重性。最普通的关联是一对类元之间的二元关联,包含一对对象,用一条连接两个类的连线表示,连线上有相互关联的角色名,而多重性则加在各个端点上。

在 Java 语言的实现方式中，多重性声明是一个多值的实例变量。例如，一个公司可以雇佣多个职员，而一个职员可以为多个公司工作，并且假定一个人最多在 5 个公司工作。对于变量的多个值，如果没有固定的上限，就表示只为一个公司工作的人；对于为 5 个公司工作的人，就会转换成一个带有 5 个元素的对象数组（参照图 5.4）。

5.3.4　递归关联

递归关联（Reflexive Association）是一个类与其自身的关联，及一个类的两个对象之间的关系。递归关联虽然只有一个被关联的类，但是有两个关联端，每个关联端的角色不同。图 5.5 是递归关联的示例。

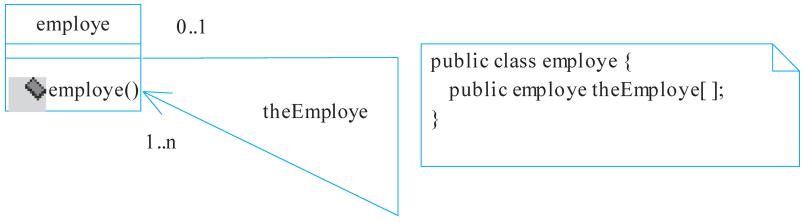

图 5.5　递归关联的示例

5.3.5　关联的约束

对关联可以添加一些约束，以加强关联的语义。图 5.6 是两个关联之间约束的示例。

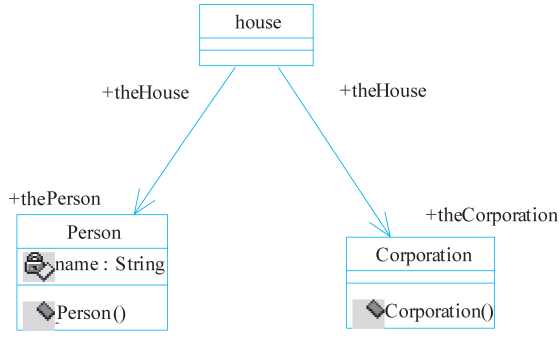

图 5.6　带约束的关联示例

house 类或者与 Person 类有关联，或者与 Corporation 类有关联，但是不能同时与这两个类都有关联。

5.4 聚集与组成关系

关联关系的基本形式是双向的，这表明关联关系两端的类的地位是平等的。关联关系又是一种结构关系，其中的角色代表一个类的对象在另一个类中的存在，即类中的对象是相互拥有的。当需要打破这种平等关系，强调类与对象之间的有向的拥有关系时，可以用聚集或组成关系对关联关系进行修饰，以表示类和对象之间的拥有关系。

5.4.1 聚集关系

聚集(Aggregation)是关联关系的一种，表示两个类之间的整体与部分的关系，表明聚集关系中的客户端以供应端的类的对象作为其一部分。聚集关系的客户端的类称为聚集类，聚集类的实例是聚集对象。位于聚集关系的供应端的类的实例是被聚集对象包含或拥有的部分。如果两个类具有聚集关系，则表示其中的聚集对象在物理上是由其他对象构造的，或逻辑上包含另一个对象，聚集对象具有其部分所有权。

聚集用端点带空心菱形的线段表示，空心菱形与聚集类连接，箭头方向是从部分指向整体。聚集关系构成了一个层次结构，表示整体的类位于层次结构的最顶部，以下依次是各个部分类。在对系统进行分析与设计时，需求描述中的“包含”“组成”“分成…部分”等词汇意味着存在聚集关系。

图 5.7 所示的 circle 类与 style 类是聚集关系。一个圆可以有颜色、是否填充等属性，可以用一个 style 对象表示这些属性。但是，同一个 style 对象也可以表示其他对象，如像三角形这样图形的样式的属性，即 style 对象可以用于不同的图形。如果 circle 这个对象不存在了，不一定就意味着 style 这个对象也不存在了。

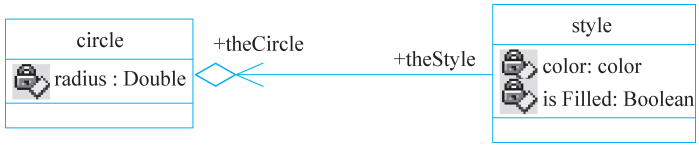

图 5.7 聚集关系的示例

以下为图 5.7 所示的聚集关系生成的 Java 源代码。

```
1.  public class circle {
2.     private double radius;
```

```
3.     public style theStyle;
4.  }
1.  public class style {
2.  private Color color;
3.  private Boolean isFilled;
4.  private circle theCircle;
5.  }
```

5.4.2　组成关系

组成(Composition)也表示类之间的整体与部分的关系，但是组合关系中的整体与部分具有相同的生命周期，即整体不存在了，部分也会随之消失。组合用端点带实心菱形的线段表示，实心菱形与组合类连接，箭头的方向是从部分指向整体。组合是一种特殊形式的聚集。在 Java 语言中，组成和聚集对应的 Java 源代码相同。

图 5.8 所示的 circle 类与 point 类是组合关系。一个圆可以由半径和圆心确定，但是如果圆不存在了，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。

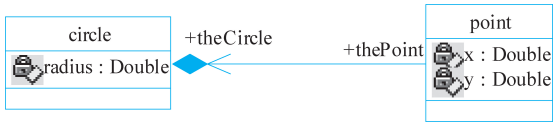

图 5.8　组合关系的示例

5.5　泛化关系

泛化(Generalization)关系描述类之间属性和操作的继承关系。当一个类的属性和操作被定义后，可以用泛化关系建立一个新导出的类，使得导出类自动具备基类已经具有的属性和操作，可以在任何基类出现的地方用其导出类代替它，但反之则不行。

泛化用从子类指向父类的箭头表示，指向父类的是一个空三角形。多个泛化关系可以用箭头线组成的树表示，每个分支指向一个子类。这种连接类型的含义是“is a kink of”(属于…中的一种)。另外，在父类中已经定义的属性和操作，在子类中不需要再定义。每种泛化元素都有一组继承特性。如果一个子类继承了它所有祖先的可继承特性，那么它的完整特性就包括继承特性和直接声明的特性。如果一个类只有一个父类，这样的继承关系称为单继承(Single

Inheritance)。UML 中的泛化可以直接映射为 Java 语言的关键字 extends，示例如图 5.9 所示。

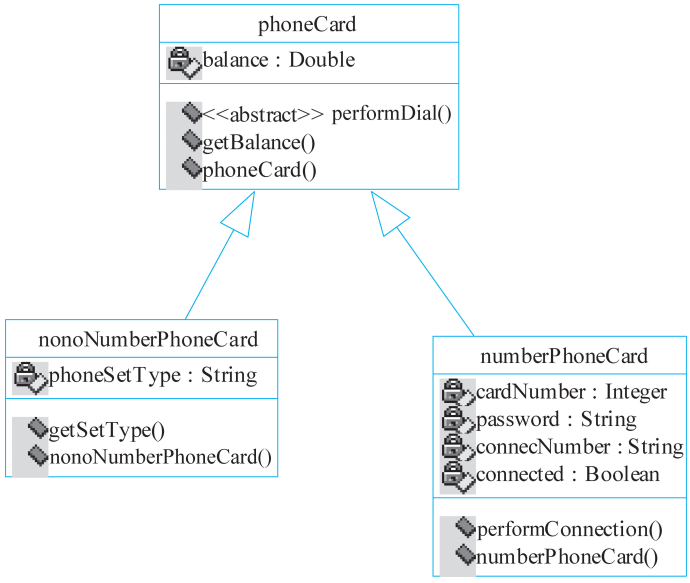

图 5.9　单继承关系的示例

以下为图 5.9 所示的继承结构生成的 Java 源代码。

```
1.  public class phoneCard {
2.     private Double balance;
3.     public phoneCard( )  {
4.     }
5.     public void performDial( )  {
6.     }
7.     public void getBalance( ) {
8.       return balance;
9.     }
10. }
1.  public class numberPhoneCard extends phoneCard {
2.  private Integer cardNumber;
3.  private String password;
4.  private String connecNumber;
5.  private Boolean connected;
6.  public numberPhoneCard( ) {
7.  }
8.  public void performConnection( ) {
```

```
9.
10. }
11. }
1.  public class nonoNumberPhoneCard extends phoneCard {
2.  private String phoneSetType;
3.  public nonoNumberPhoneCard( ) {
4.  }
5.  public void getSetType( ) {
6.  return phoneSetType;
7.  }
8.  }
```

5.6　依赖关系

在 UML 中，两个类之间的依赖(Dependency)关系表明其中的一个类(客户类)依赖于另一个类(供应类)提供的某些服务。UML 中的依赖关系被图形化地表示为一个带虚线的箭头，箭头指向的类是供应类(被依赖的类)，箭头出发点的类是客户类，示例如图 5.10 所示。

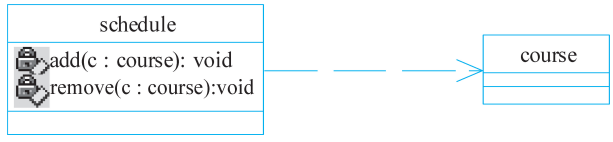

图 5.10　类之间的依赖关系

在图 5.10 中，course(课程)类是独立的供应者，schedule(课程表)类是依赖于课程类的客户类。课程表类的操作 add 和 remove 都使用了课程类中的数据，课程类是这两个操作的参数类型。一旦课程发生变化，课程表也要随之改变。

5.7　接口和实现关系

软件系统的内部由大量相互关联的类构成，当对其中一个类的局部进行修改时，不应影响其他类的工作。为了实现这一点，可以为类或类的集合设定一个外部的行为规范，只要对类或类的集合进行的修改不会改变这个行为规范，就可以保证其他类乃至整个系统能正常工作。这样的规范在 UML 中称为接口(Interface)。接口是一系列操作的集合，它指定了一个类或者一个部件能提供的服务。接口只能拥有操作，不能拥有属性。接口可以用一个类图标表示。在