# 第 3 章

# 类图

使用面向对象的思想描述系统能够把复杂的系统简单化、直观化，这有利于用面向对象的程序设计语言实现系统，并且有利于未来对系统的维护。构成面向对象模型的基本元素有类、对象和类与类之间的关系等。类图是显示模型的静态关系，也是最常用的 UML 图，它显示出类、接口及它们之间的静态结构和关系，主要用于描述系统的结构化设计，这样可以更好地体现系统的分层结构，使得系统层次关系一目了然。

本章主要介绍类图、类与类之间的关系类图模型、抽象类以及接口等内容，通过本章的学习，读者可以充分了解类图的相关知识，并且能够掌握类与类之间的关系。

**本章学习内容：**

- 类图的概念
- 泛化关系
- 依赖关系
- 实现关系
- 关联关系

## 3.1 类图的概念

构建面向对象模型的基础是类、对象以及它们之间的关系，可以在不同类型的系统（例如商务软件、嵌入式系统和分布式系统等）中应用面向对象技术。不同系统中所描述的类是各种各样的，例如在某个商务信息系统中，包含的类可以是顾客、协议书、发票、债务等；在某个工程技术系统中，包含的类可以有传感器、显示器、I/O 卡、发动机等。

在面向对象的处理中类图处于核心地位，它提供了用于定义和使用对象的主要规则，同时类图是正向工程（将模型转化为代码）的主要资源，是逆向工程（将代码转化为模

型）的生成物。因此类图是任何面向对象系统的核心，进而也成了最常用的 UML 图。本节将详细介绍与它相关的知识，包括类图的概念、类的表示和如何定义一个类等相关内容。

### 3.1.1 类图概述

类图是描述类、接口以及它们之间关系的图，是一种静态模型，显示了系统中各个类的静态结构。类图根据系统中的类及各个类的关系描述系统的静态视图，可以用某种面向对象的语言实现类图中的类。

类图是面向对象系统建模中最常用和最基本的图之一，其他许多图（如状态图、协作图、组件图和配置图等）都是在类图的基础上进一步描述了系统其他方面的特性。类图可以包含类、接口、依赖关系、泛化关系、关联关系和实现关系等模型元素，另外在类图中也可以包含注释、约束、包或子系统。

UML 类图中常见的关系有以下几种：泛化、实现、关联、聚合、组合及依赖。其中聚合和组合关系都是关联关系的一种。这些关系的强弱顺序不同，排序结果为：泛化=实现>组合>聚合>关联>依赖。

类图用于对系统的静态视图（它用于描述系统的功能需求）建模，通常以如下所示的某种方式使用类图。

- **对系统的词汇建模** 在进行系统建模时，通常首先构造系统的基本词汇，以描述系统的边界。在对词汇进行建模时，通常需要判断哪些抽象是系统的一部分，哪些抽象位于系统边界之外。
- **对协作建模** 协作是一些协同工作的类、接口和其他元素的共同体，其中，元素协作时的功能强于它们单独工作时的功能之和。系统分析员可以用类图描述图形化系统中的类及它们之间的关系。
- **对数据库模式建模** 在很多情况下都需要在关系数据库中存储永久信息，这时可以使用类图对数据库模式进行建模。

图 3-1 列举了一个简单的类图示例，它起到一个引导的作用，目的在于使读者对类图有一个直观浅显的了解。

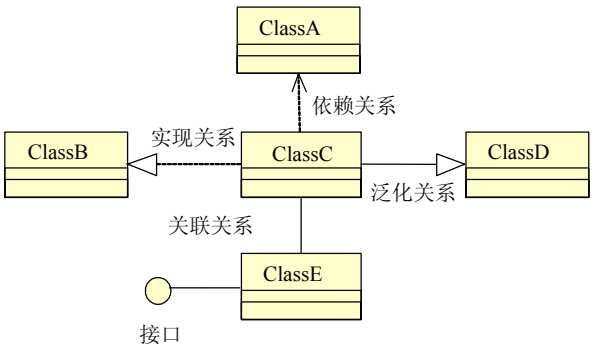



图 3-1 类图示例

类图通过分析用例和问题域，可以建立系统中的类，再把逻辑上相关的类封装成包，这样就可以直观清晰地展现出系统的层次关系。

但是使用类图时也需要遵循一些原则。其主要说明如下。

- **简化原则** 在项目的初始阶段不要使用所有的符号，只要能够有效表达就可以。

- **分层理解原则** 根据项目开发的不同阶段，使用不同层次的类图来进行表达方便理解，不要一开始就陷入到实现类图的细节当中。
- **关注关键点原则** 不要为每一个事物都画一个模型，只把精力放到关键的位置。

## 3.1.2 类

类是构成类图的基础，也是面向对象系统组织结构的核心。要使用类图，需要了解类和对象之间的区别。类是对资源的定义，它所包含的信息主要用来描述某种类型实体的特征，以及对该类型实体的使用方法。对象是具体的实体，它遵守类制定的规则。从软件的角度看，程序通常包含的是类的集合及类所定义的行为，而实际创建信息和管理信息的是遵守类的规则的对象。

类定义了一组具有状态和行为的对象，这些对象具有相同的属性、操作、关系和语义。其中属性和关系用来描述状态。属性通常用没有身份的数据值表示，如数字和字符串；关联则用有身份的对象之间的关系来表示。行为由操作来描述，方法是操作的实现。

为了支持对身份、属性和操作的定义，UML 规范采用一个具有 3 个预定义分栏的图标表示类，分栏中包含的信息有：名称（Name）、属性（Attribute）和操作（Operation），它们对应着类的基本元素，如图 3-2 所示。

当将类绘制在类图中时，名称分栏是必须出现的分栏，而属性分栏和操作分栏则可以出现或不出现。图 3-2 显示了所有的分栏，另外 3 种形式如图 3-3 所示。

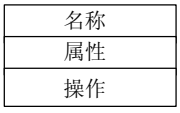



图 3-2 类的分栏

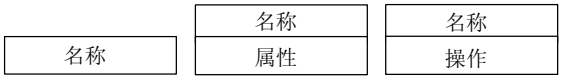



图 3-2 类的其他形式

当隐藏某个分栏时并非表明某个分栏不存在，只显示当前需要注意的分栏可以使图形更加直观清晰。

类在它的包含者（可以是包或者另一个类）内必须有唯一的名称。类对它的包含者来说是可见的，可见性规定了类能够怎样被位于可见者之外的类所使用。类的多重性说明了类可以具有多少个实例，通常情况下可以有 0 个或多个。

下面将详细介绍类的名称、属性和操作在类图中的具体表示方法和含义。

### 1. 名称

类名采用黑体字书写在名称分栏的中部，它通常表示为一个名词，既不带前缀，也不带后缀。为类命名时最好能够反映类所代表的问题域中的概念，并且要清楚准确，不能含糊不清。类名可分为简单名称和路径名称。简单名称只有类名没有前缀；路径名称中可以包含由类所在包的名称表示的前缀，如图 3-4 所示。


图 3-4 类的简单名称和路径名称

其中，Student 是类的名称，Person 是

Student 类所在包的名称。

### 2．属性

类的属性也称为特性，它描述了类在软件系统中代表的事物（即对象）所具备的特性，这些特性是该类的所有对象所共有的。类可以有任意数目的属性，也可以没有属性。在系统建模时只抽取那些对系统有用的特性作为类的属性，通过这些属性可以识别该类的对象。例如可以将学生姓名、学生编号、出生年月、所在班级、职位等特性作为 Student 类的属性。

从系统处理的角度来看，事物的特性中只有其值能被改变的那些才可以作为类的属性。UML 中描述类属性的语法格式如下所示：

```
［可见性］属性名［:类型］［=初始值］［{属性字符串}］
```

上述语法中属性包含 5 个部分：可见性、属性名、类型、初始值和属性字符串。除了属性名之外，其他内容都是可有可无的，可以根据需要选用上面列出的某些项。

❑ 可见性

可见性用于指定它所描述的属性能否被其他类访问，以及能以何种方式访问。在 UML 中并未规定默认的可见性，如果在属性的左边没有标识任何符号，表明该属性的可见性尚未定义，而并非取了默认的可见性。

最常用的可见性类型有 3 种，分别为公有（Public）、私有（Private）和被保护（Protected）类型。

- 被声明为 Public 的属性和操作可以在它所在类的外部被查看、使用和更新。在类里被声明为 Public 的属性和操作共同构成了类的公共接口。类的公共接口由可以被其他类访问及使用的属性和操作组成，这表示为公共接口是该类与其他类的联系的部分。类的公共接口应尽可能减少变化，以防止任何使用该类的地方有不必要的改变。
- Private 可见性是限制最为严格的可见性类型，只有包含 Private 元素的类本身才能使用 Private 属性中的数据，或者调用 Private 操作。
- 被声明为 Protected 的属性和操作可以被类的其他方法访问，也可以被任何相应继承类所声明的方法访问，但是非继承的类无法访问 Protected 属性和操作。即使用 Protected 声明的属性和操作只可以被该类和该类的子类使用，而其他类无法使用。

**注　意**

对于是否应该声明为 Public 属性是有不同的观点的。许多面向对象的设计者对 Public 属性有抱怨，因为这会将类的属性向系统的其余部分公开，就违反了面向对象的信息隐蔽的原则。因此，最好避免使用 Public 属性。

除了以上 3 种类型的可见性之外，其他类型的可见性可由程序设计语言进行定义。需要注意的是，公有和私有可见性一般在表达类图时是必需的。UML 中，Public 类型用符号“+”表示，Private 类型用符号“–”表示，Protected 类型用符号“#”表示。这几种类型符号在类中的表示如图 3-5 所示。

在图 3-5 中，属性 studentNo 和 studentBirth 是类 Student 的私有属性，studentName 是类的公有属性，studentClass 和 studentPosition 属性是类被保护的属性，这些属性的可见性是由它们的名称左边的符号指定的。

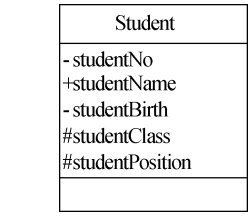



图 3-5 类变量的表示

❑ 属性名

类的属性是类定义的一部分，每个属性都应有唯一的属性名，以标识该属性，并以此区别其他属性。属性名通常由描述所属类的特性的名词或名词短语组成，单字属性名小写，如果属性名包含了多个单词，则这些单词可以合并，且从第二个单词起，每个单词的首字母都应是大写形式，图 3-5 中属性可见性的右边表示属性名。

❑ 类型

每个属性都应指定其所属的数据类型。常用的数据类型有整型、实型、布尔型、枚举类型等。这些类型在不同的编程语言中可能有不同的定义，可以在 UML 中使用目标语言中的类型表达式，这在软件开发的实施阶段是非常有用的。

除了上面的类型外，属性的数据类型还可以使用系统中的其他类，或者用户自定义的数据类型。类的属性定义之后，类的所有对象的状态由其属性的特定值所决定。

❑ 初始值

开发人员可以为属性设置初始值，设置初始值可以防止因漏掉某些取值而破坏系统的完整性，并且为用户提供易用性。为 Student 类的有关属性指定数据类型和初始值后，效果如图 3-6 所示。

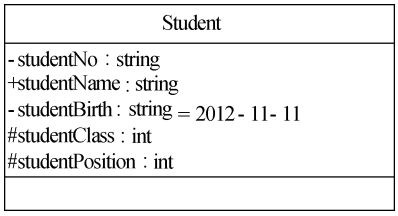



图 3-6 属性的数据类型和初始值

从图 3-6 中可以看出属性的数据类型之间要用冒号分隔，数据类型与初始值之间用等号分隔。使用 Microsoft Visio 画图时，冒号和等号都是该软件自动添加的。

❑ 属性字符串

描述类属性的语法格式中的最后一项是属性字符串。属性字符串用来指定关于属性的其他信息，任何希望添加属性定义字符串但又没有合适地方可以加入的规则都可以放在属性字符串里。

除了上面介绍外，还有一种类型的属性，它能被所属类的所有对象共享，这就是类的作用域属性，或者叫作类变量（例如，Java 类中的静态变量）。这类属性在类图中表示时要在属性名的下面加一条下画线。例如开发人员可以将 Student 类中的 studentPosition 属性更改为类变量或者重新添加一个新的类变量。

属性可以代表一个以上的对象，实际上属性能代表其类型的任意数目的对象。在程序设计时属性用一个数组来实现，体现了面向对象中对象之间关联的多重性，多重性指允许用户指定属性实际上代表一组对象集合，而且能够应用于内置属性及关联属性。图 3-7 列出了属性对应的多重性，由于一名学生可以借阅多本图书，所以一个 Student 类可以对应多个 Book 类。

### 3．操作

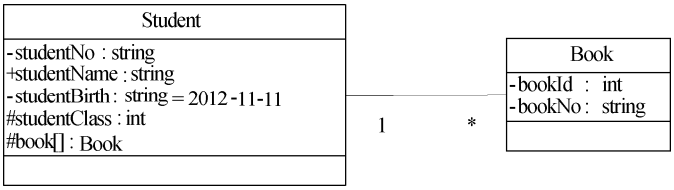



图 3-7 属性对应的多重性

属性仅仅描述了要处理的数据，而操作则描述了处理数据的具体方法。类的操作是对其所属对象的行为的抽象，相当于一个服务的实现，且该服务可以由类的任何对象请求以影响其行为。属性是描述对象特征的值，操作用于操纵属性或执行其他动作。操作可以看作是类的接口，通过该接口可以实现内、外信息的交互，操作的具体实现被称作方法。

操作由返回值类型、名称和参数表进行描述，它们一起被称为操作签名。某个类的操作只能作用于该类的对象，一个类可以有任意数量的操作或者根本没有操作。UML 中用于描述操作的语法形式如下：

```
[可见性] 操作名 [(参数表)] [:返回类型] [{属性字符串}]
```

上述语法形式中有可见性、参数表和返回类型等内容，其具体说明如下。

类操作的可见性类型包括公有（Public）、私有（Private）、受保护（Protected）和包内公有（Package）几种类型，UML 类图中它们可以分别用“+”“–”“#”和“～”来表示。如果某一对象能够访问操作所在的包，那么该对象就可以调用可见性为公有的操作；可见性为私有的操作只能被其所在类的对象访问；子类的对象可以调用父类中可见性为公有的操作；可见性为包内公有的操作可以被其所在包的对象访问。

在为系统建模时操作名通常用来描述类的行为的动词或者动词短语，操作名的第一个字母通常使用小写形式，当操作名包含多个单词时，这些单词要合并起来，并且从第二个单词起所有单词的首字母都是大写形式。

参数用来指定提供给操作以完成工作的信息，它是可选的，即操作可以有参数，也可以没有参数。如果参数表中包含多个参数时，各参数之间需要使用逗号隔开。当参数具有默认值时，如果操作的调用者没有为该参数提供相应的值，那么该参数将自动具有指定的默认值。例如 Student 类中 deleteStudent 操作表示根据学生的编号删除一个 Student 对象，执行该操作时只需要知道学生编号信息即可，如图 3-8 所示。

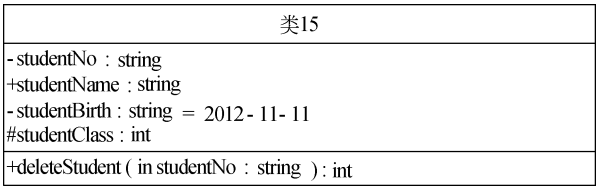



图 3-8 操作中的参数

操作除了具有名称与参数外，还可以有返回类型。返回类型被指定在操作名称尾端的冒号之后，它指定了该操作返回的对象类型，如图 3-9 所示。如果某个操作返回值时可以不注明返回值的类型，但是在具体的编程语言中可能需要添加关键字 void 来表示无

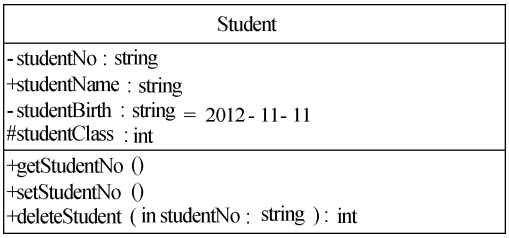



图 3-9 类操作的返回类型

返回值的情况。

除了图 3-9 中提供的每一个参数名及其数据类型外，还可以指定参数子句 in、out 或者 inout。其中，in 是默认的参数子句，通过值传递的参数使用 in 参数子句，或者不使用任何参数子句。通过值传递参数意味着把数据的副本发送到操作，因而操作不会改变值的主备份。如果希望修改传递到操作的参数值的主备份，需要使用 inout 类型的参数子句标记参数，这意味着值通过引用传递，操作中任何对参数值的修改也就是对变量主备份的修改。除此之外，还有一种 out 参数子句，使用该参数子句时，值不是被传递给操作，而是由操作把值返回给参数。

当需要在操作的定义中添加一些预定义元素之外的信息时，可以将它们作为属性字符串。

#### 4．职责

所谓的职责是指类或者其他元素的契约或义务，可以在类标记中操作分栏的下面另加一个分栏，用于说明类的职责。相关人员在创建类时，需要声明该类的所有对象具有相同的状态和相同的行为，这些属性和操作正是要完成类的职责。描述类的职责可以使用一个短语、一个句子或者若干句子。

#### 5．约束和注释

在类的标记中说明类的职责是消除二义性的一种非形式化的方法，而使用约束则是一种形式化的方法。约束指定了类应该满足的一个或者多个规则。约束在 UML 规范中是用由大括号括起来的文本表示的。图 3-10 所示为 Teacher 类所添加的约束。

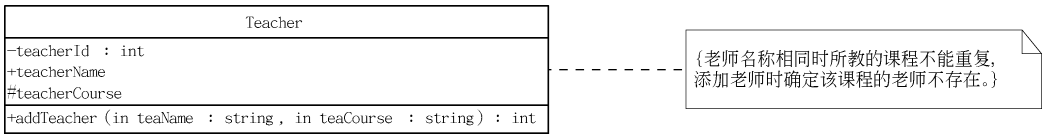



图 3-10　为 **Teacher** 类所添加的约束

除此约束外还可以在类图中使用注释，以便为类添加更多的说明信息，注释可以包含文本和图形。图 3-11 所示为 Teacher 类所添加的注释。

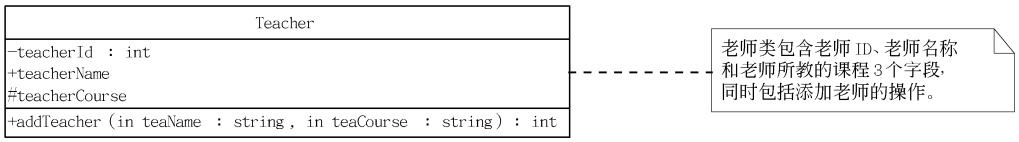



图 3-11　为 **Teacher** 类所添加的注释

### 3.1.3　定义类

由于类是构成类图的基础，所以在构造类图之前首先要定义类，也就是将系统要处理的数据抽象为类的属性，将处理数据的方法抽象为类的操作。要准确地定义类需要对

问题域有透彻准确的理解。在定义类时通常应当使用问题域中的概念，并且类的名字要用类实际代表的事物进行命名。

通过自我提问并回答下列问题，将有助于在建模时准确地定义类。

- 在要解决的问题中有没有必须存储或处理的数据，如果有，那么这些数据可能就需要抽象为类，这里的数据可以是系统中出现的概念、事件，或者仅在某一时刻出现的事务。
- 有没有外部系统，如果有，可以将外部系统抽象为类，该类可以是本系统所包含的类，也可以是能与本系统进行交互的类。
- 有没有模板、类库或者组件等，如果有，这些可以作为类。
- 系统中有什么角色，这些角色可以抽象为类，例如用户、客户等。
- 系统中有没有被控制的设备，如果有，那么在系统中应该有与这些设备对应的类，以便能够通过这些类控制相应的设备。

通过自我提问并回答以上列出的问题，有助于在建模时发现需要定义的类。但定义类的基本依据仍然是系统的需求规格说明，应当认真分析系统的需求规格说明，进而确定需要为系统定义哪些类。另外分析用例和问题域完成后可以建立系统中的类，再把逻辑上相关的类封装成包，这样就可以直观清晰地展现出系统的层次关系。

### 3.1.4 接口

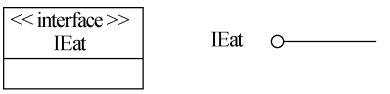
对 Java 或 C#等高级语言不陌生的读者一定知道：一个类只能有一个父类（即该类只能继承一个类），但是如果用户想要继承两个或两个以上的类时应该怎么办？很简单，可以使用接口（Interface）。

接口是对对象行为的描述，但是它并没有给出对象的实现和状态。且接口是一组没有相应方法实现的操作，非常类似于仅包含抽象方法的抽象类。接口中只包含操作而不包含属性，且接口没有对外界可见的关联。

一个类可以实现多个接口，使用接口可避免许多与多重继承相关的问题，因此使用接口比使用抽象类要安全得多。如在 Java 和 C#等新型编程语言中允许类实现多个接口，但只能继承一个通用类或抽象类。

接口通常被描述为抽象操作，即只是用操作名、参数表和返回类型说明接口的行为，而操作的实现部分将出现在使用该接口的元素中。可以将接口想成非常简单的协议，它规定了实现该接口时必须实现的操作。接口的具体实现过程、方法对调用该接口的对象而言是透明的。在进行系统建模时，接口起到十分重要的作用，因为模型元素之间的协作是通过接口进行的。相关人员可以为类、组件和包定义接口，利用接口说明类、组件和包能够支持的行为。一个结构良好的系统通常都定义了比较规范的接口。

UML 中接口可以使用构造型的类表示，也可以使用一个“球形”来表示，图 3-12 演示了实现的两种方法。




构造型表示法　　球形表示法

图 3-12　接口的两种表示方法

接口与抽象类一样，都不能实例化为对象。在

UML 中接口可以使用一个带有名称的小圆圈来进行表示，并且可以通过一条 Realize（实现关系）线与实现它的类相连接，如图 3-13 所示。

如果使用构造型表示接口，则由于实现接口的类与接口之间是依赖关系，所以用一端带有箭头的虚线表示显示这个实现的关系，如图 3-14 所示。

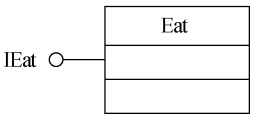



图 3-13　类实现了某个接口

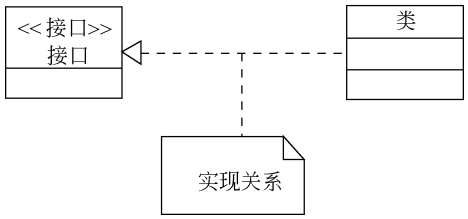



图 3-14　类实现接口的构造型表示法

如果某个接口是在一个特定类中实现的，则使用该接口的类仅依赖于特定接口中的操作，而不依赖于接口实现类中的其他部分。如果类实现了接口，但未实现该接口指定的所有操作，那么此类必须声明为抽象的。使用接口可以很好地将类所需要的行为与该行为如何被实现完全分开。

## 3.2　泛化关系

类与类之间的关系有多种，如依赖关系、实现关系和泛化关系等。泛化描述了一般事物与该事物的特殊种类之间的关系。在解决复杂问题时通常需要将具有共同特性的元素抽象成类别，并通过增加其内容而进一步分类。例如，车可以分为火车、汽车、摩托车等。它们也可以表示为泛化关系，下面将详细介绍与泛化相关的知识。

### 3.2.1　泛化的含义和用途

应用程序中通常会包含大量紧密相关的类，如果一个类 A 的所有属性和操作能被另一个类 B 所继承，则类 B 不仅可以包含自己独有的属性和操作，而且可以包含类 A 中的属性和操作，这种机制就是泛化（Generalization）。

UML 中继承是泛化的关键。父类与子类各自代表不同的内容，父类描述具有一般性的类型，而此类型的子类则描述该类型中的特殊类型。从另外一种方法来说，泛化是一种继承关系，表示一般与特殊的关系，它指定了子类如何泛化父类的所有特征和行为，例如老虎是动物的一种，既有老虎的特性也有动物的共性。

泛化关系是一种存在于一般元素和特殊元素之间的分类关系。这里的特殊元素不仅包含一般元素的特征，而且包含其独有的特征。凡是可以使用一般元素的场合都可以用特殊元素的一个实例代替，反之则不行。

泛化关系只使用在类型上，而不用于具体的实例。泛化关系描述了“is a kind of”（是……的一种）的关系。例如，金丝猴、猕猴都是猴子的一种，东北虎是老虎的一种。在采用面向对象思想和方法的地方，一般元素被称为超类或者父类，而特殊元素被称作子类。

UML 规定，泛化关系用一个末端带有空心三角形箭头的直线表示，有箭头的一端指

向父类。图 3-15 演示了一个简单的泛化关系，其中，Monkey 类表示父类或超类，该类包含 Golden Monkey 和 Macaque 两个子类，这两个子类不仅继承了父类中的所有属性和操作，同时也可以拥有自己特定的属性和操作。

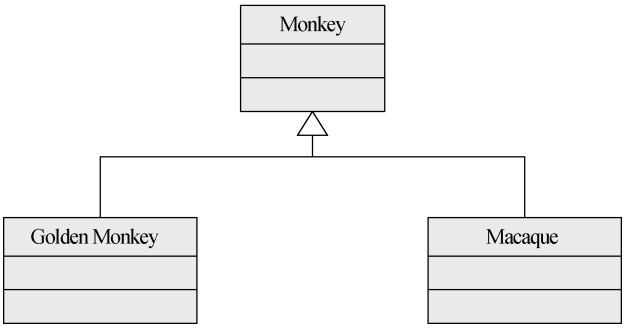



图 3-15 泛化关系示例

泛化主要有两个用途，第一个用途是当变量被声明承载某个给定类的值时，可使用类的实例作为值，这被称作可替代性原则。该原则表明无论何时祖先被声明了，其后代的一个实例就可以被使用。例如，如果猴子父类 Monkey 被声明，那么一个金丝猴或者猕猴的对象就是一个合法的值。第二个用途是通过泛化使多态操作成为可能，即操作的实现是由它们所使用的对象的类决定的，而不是由调用者所确定的。

## 3.2.2 泛化的层次与多重继承

泛化可能跨越多个层次。一个子类的超类也可以是另一个超类的子类。图 3-16 所示为具体层次结构的泛化。

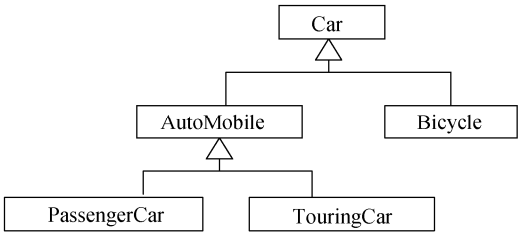



图 3-16 层次结构的泛化效果

在图 3-16 中，AutoMobile 类是 Car 类的子类，不仅如此，AutoMobile 类还是类 PassengerCar 和类 TouringCar 的超类，这就显示出了泛化的层次结构。子类和超类这两个术语是相对的，它们描述的是一个类在特定泛化关系中所扮演的角色，而不是类自身的内在特性。在该图中 Bicycle 表示 Car 类中的另外一个类，也可以使用 3 个点表示省略号，如果为 3 个点时，表明 Car 类除了图中所显示的子类 AutoMobile 外，还可以拥有其他子类。

对泛化层次图中的一个类而言，从它开始向上遍历到根时经历的所有类都是其祖先，从它开始向下遍历时遇到的所有类都是其后代。这里的“上”和“下”分别表示“更一般的类”和“更特殊的类”。

面向对象设计的最佳原则之一是避免紧密耦合的类，使在一个类改变时不必改变一系列相关的其他类。由于泛化使用子类可以看见父类内部的大部分内容，使得子类紧密耦合于父类，所以泛化是类关系中最强的耦合形式。因此使用泛化的基本原则是：只有在一个类确定是另外一个类的特殊类型时才使用泛化。

多重继承在 UML 中的正式术语称为多重泛化。多重泛化使同一个子类不仅可以像图 3-16 中的 Car 类那样具有多个子类，而且可以拥有多个父类，即一个类可以从多个父类派生而来。例如，坦克是一种武器，但它同时也可作为一种车来使用。多重泛化在 UML

中的表示方法如图 3-17 所示。

在图 3-17 中，一个子类带有两个指向超类的箭头。通过 Vehicle 类的 drive、reverse、park、start 和 stop 操作确定了属于 Vehicle 类的行驶功能，通过 Weapon 类的 load、aim 和 fire 操作确定了属于 Weapon 类的破坏功能。ram 和 radio 操作则是 Tank 类独有的。

虽然 UML 支持多重泛化，但是通常情况下实际应用中的泛化使用并不多。其主要原因在于如果两个父类具有重叠的属性和操作时，多重继承里的父类会存在错综复杂的问题。因此，多重继承在面向对象的系统开发中已经被禁止，而当今流行的一些开发语言，如 Java 和 C#都不支持多重继承。

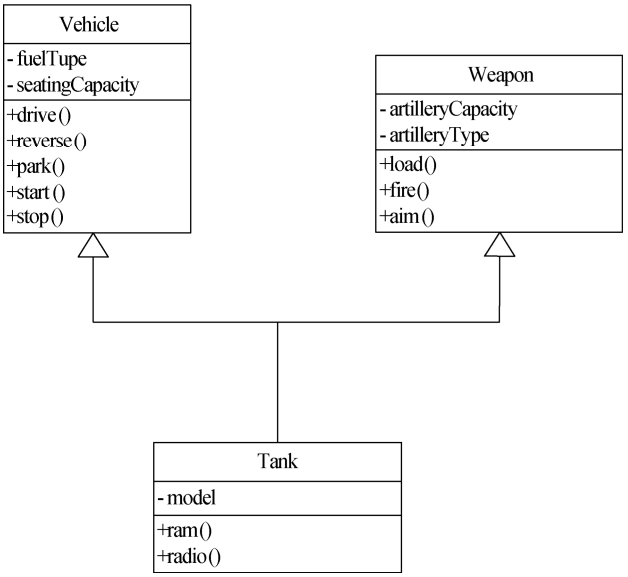



图 3-17　多重泛化示例

## 3.2.3　泛化约束

泛化约束用于表明泛化有一个与其相关的约束，带有约束条件的泛化也被称为受限泛化。泛化建模约束有两种情况，如果有多个泛化使用相同的约束，可以绘制虚线穿过两个泛化，并且在花括号（{...}）中标注约束名。如果只有一个泛化，或者多个泛化共享关联的空箭头部分，就只需在朝向空箭头的花括号中建模约束即可，如图 3-18 所示。

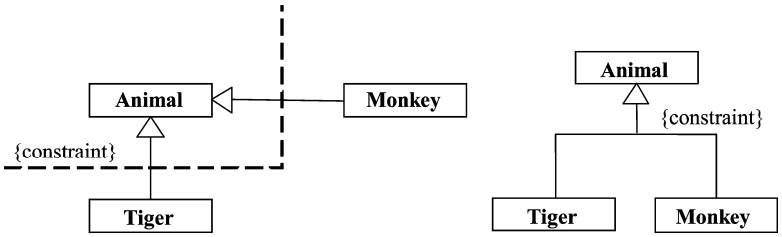



图 3-18　泛化约束示例

泛化约束包含 4 种：不完全约束（Incomplete Constraint）、完全约束（Complete Constraint）、解体约束（Disjoint Constraint）和重叠约束（Overlapping Constraint）。

- **不完全约束**　表示类图中没有完全显示出泛化的类，这种约束可以让读者知道类图中显示的内容仅仅是实际内容的一部分，其余内容可能位于其他类图中，如图 3-19 所示。
- **完全约束**　与不完全约束相对的是完全约束，当类图中存在完全约束时，表示类图中显示了全部内容，如图 3-20 所示。

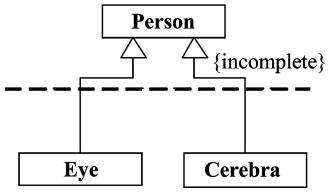



图 3-19 不完全约束

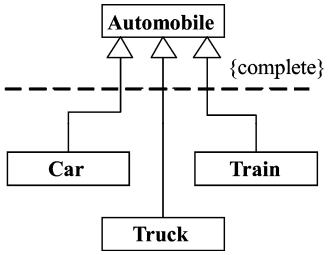



图 3-20 完全约束

- **解体约束** 表示紧靠约束下面的泛化类不能有子类化为通用的类，它比前两种约束更加复杂，如图 3-21 所示。从图中可以看出，根超类 OS 有两个子类 Windows 和 Linux。解体约束表示 Windows 和 Linux 类都不能共享其他的子类。在该图中，类 Windows 和 Linux 都有各自的子类，但不能从 Windows NT 类到 Linux 类绘制一个泛化关联，由于解体约束的存在，Windows NT 类不能同时继承 Windows 和 Linux 类。
- **泛化约束** 与解体约束的作用相反的泛化约束，即重叠约束。该类型的约束表示两个子类可以共享相同的子类。如图 3-22 所示，Database 类有两个子类 Relational 和 OLAP，它们共享相同的类 DataWarehouse。

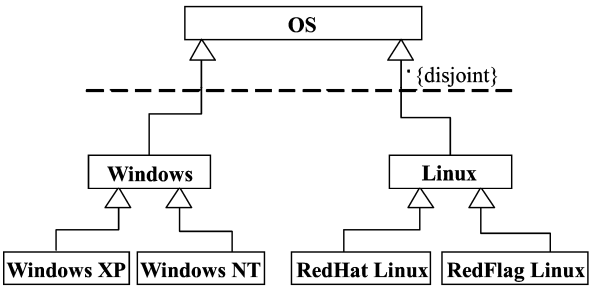



图 3-21 解体约束

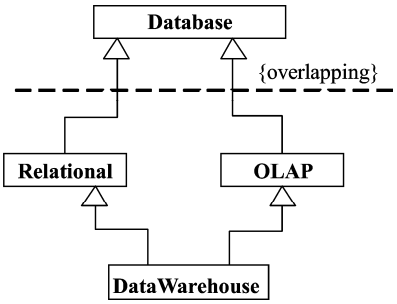



图 3-22 泛化约束

## 3.3 依赖关系和实现关系

模型元素之间的依赖关系描述的是它们之间语义上的关系。当两个元素处于依赖关系中时，其中一个元素的改变可能会影响或提供消息给另一个元素，即一个元素以某种形式依赖于另一元素。在 UML 模型中，模型元素之间的依赖关系表示某一元素以某种形式依赖于其他元素。从某种意义上说，关联关系、泛化关系和实现关系都属于依赖关系，但是它们都有其特殊的语义，因而被作为独立的关系在建模时使用。

### 3.3.1 依赖关系

依赖关系用一个一端带有箭头的虚线表示，在实际建模时可以使用一个构造型的关

键字来区分依赖关系的种类。例如图 3-23 中表示 Person 类依赖于 Computer 类。

UML 规范中定义了 4 种基本的依赖类型，它们分别是使用（Usage）依赖、抽象（Abstraction）依赖、授权（Permission）依赖和绑定（Binding）依赖，下面对它们分别进行介绍。

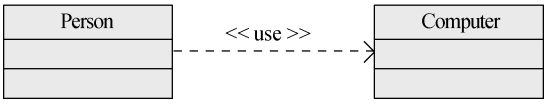



图 3-23 带有构造型的依赖关系

### 1．使用依赖

使用依赖用于表示一种元素使用其他元素提供的服务以实现它的行为，表 3-1 列出了 5 种使用依赖关系。

表 3-1 使用依赖

| 依赖关系 | 说　明 | 关键字 |
|---|---|---|
| 使用 | 用于声明使用某个模型元素需要用到已存在的另一个模型元素，这样才能实现使用者的功能，包括调用、参数、实例化和发送 | use |
| 调用 | 用于声明一个类调用其他类的操作方法 | call |
| 参数 | 用于声明一个操作与其参数之间的关系 | parameter |
| 实例化 | 用于声明使用一个类的方法创建了另一个类的实例 | instantiate |
| 发送 | 用于声明信号发送者和信号接收者之间的关系 | send |

在实际建模过程中，表 3-1 中的使用依赖使用最多，调用依赖和参数依赖一般很少使用，实例化依赖用于说明依赖元素会创建被依赖元素的实例，发送依赖用于说明依赖元素会把信号发送给被依赖元素。以下 3 种情况需要建模使用依赖关系。

- 客户类的操作需要提供者类的参数。
- 客户类的操作在实现中需要使用提供者类的对象。
- 客户类的操作返回提供者类型的值。

### 2．抽象依赖

抽象依赖包括 3 种：跟踪、精化和派生。它们的具体说明如下。

- **跟踪（Trace）依赖**　用于描述不同模型中元素之间的连接关系，但是没有映射精确。这些模型一般分属于开发过程中的不同阶段。跟踪依赖缺少详细的语义，它主要用来追溯跨模型的系统要求，以及跟踪模型中会影响其他模型的模型所发生的变化。
- **精化（Refine）依赖**　用于表示一个概念的两种形式之间的关系，这种概念位于不同的开发阶段或者处于不同的抽象层次。这两种形式的概念并不会在最终的模型中共存，其中的一个一般是另一个的不完善的形式。
- **派生（Derive）依赖**　用于声明一个实例可以从另一个实例导出。

### 3．授权依赖

授权依赖用于表示一个事物访问另一个事物的能力，被依赖元素通过规定依赖元素的权限，可以控制和限制对其进行访问的方法。常用的授权依赖关系有 3 种，其具体说明如表 3-2 所示。

表 3-2 授权依赖

| 依赖关系 | 说　明 | 关键字 |
|---|---|---|
| 访问 | 用于说明允许一个包访问另一个包 | access |
| 导入 | 用于说明允许一个包访问另一个包，并为被访问包的组成部分增加别名 | import |
| 友元 | 用于说明允许一个元素访问另一个元素，无论被访问的元素是否具有可见性 | friend |

#### 4．绑定依赖

绑定依赖用于为模板参数提供值，以创建一个新的模型元素，表示绑定依赖的关键字为 bind。绑定依赖是具有精确语义的高度结构化的关系，可通过取代模板备份中的参数实现。

UML 2.0 中还添加了一个被称作 substitution（替代）依赖性的新概念，它是 realization 依赖性的一种类型，即它是实现类元的另外一种方法。在 substitution 依赖关系中，作为客户的一方取代了作为提供者的类元。在需要对系统进行定制的时候，这种依赖概念尤其好用。图 3-24 演示了 substitution 的使用方法。

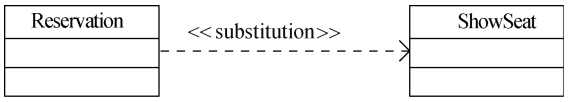



图 3-24 substitution 依赖性

在图 3-24 中主要用来预定演出座位，在系统中任何需要订座的地方都可以使用 Reservation 类来代替 ShowSeat 类，因此 ShowSeat 类必须遵从 Reservation 类确定的接口。

### 3.3.2 实现关系

实现关系（Realization）用于规定规格说明与其实现之间的关系，它通常用在接口和实现该接口的类之间，以及用例和实现该用例的协作之间。换一种说法，实现关系指定两个实体之间的一个合同，一个实体定义一个合同，而另一个实体保证履行该合同。使用 Java 应用程序进行建模时，实现关系可直接用 implements 关键字来表示。

UML 中将实现关系表示为末端带有空心三角形的虚线，带有空心三角形的那一端指向被实现元素。除此之外，还可将接口表示为一个小圆圈，并和实现该接口的类用一条线段连接起来。图 3-25 演示了一个简单的实现关系。

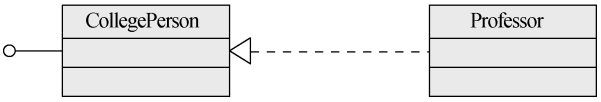



图 3-25 实现关系示例图

泛化关系与实现关系是有异同点的，它们都可以将一般描述和具体描述联系起来，但是泛化关系是将同一语义层上的元素连接起来，并且通常在同一模型内，而实现关系则将不同语义层的元素连接起来，并且通常建立在不同的模型内。在不同的发展阶段可能有不同数目的类等级存在，这些类等级的元素通过实现关系联系在一起。

## 3.4 关联关系

对象之间也需要定义通信手段，UML 规范中对象之间的通信手段就称为关系。类图

中的关联定义了对象之间的关系准则，在应用程序创建和使用关系时，关联提供了维护关系完整性的规则。类关系的强弱基于该关系所涉及的各类间彼此的依赖程度。彼此相互依赖较强的两个类称为紧密耦合。在这种情况下，一个类的改变极可能影响到另一个类。紧密耦合通常是个坏事。

### 3.4.1 二元关联

关联意味着类实际上以属性的形式包含对其他类的一个或多个对象的引用。在确定了参与关联的类之后，就可以对关联进行建模了。只有两个类参与的关联可以称为二元关联；多于两个类参与的关联，即为 n 元关联。在类图中二元关联定义了两个类的对象之间的关系准则，关联定义了什么是允许的，什么是不允许的。如果两个类在类图中具有关联关系，那么在对象图中这两个类的相应对象所具有的关系被称为链。关联描述的是规则，而链描述的是事实。图 3-26 演示了 Person 类和 Car 类之间的关联关系。Person 类定义了人对象及其功能，Car 类则定义了小汽车对象及其功能，两者之间的关联是一种单一类型的关系，存在于两者的对象之间，解释了这些对象需要通信的原因。

一个完整的关联包括类之间关联关系的直线和两个关联端点。图 3-27 演示了关联的组成。其中直线及关联名称定义了该关系的标志和目的，关联端点定义了参与关联的对象所应遵循的规则。在 UML 规范中关联端点是一个元类，它拥有自己的属性，例如多重性、约束、角色等。

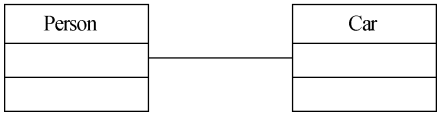



图 3-26 关联的简单示例

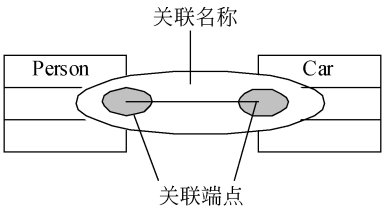



图 3-27 关联的组成

#### 1. 关联的名称

关联的名称表达了关联的内容，含义确切的名称使人更容易理解，如果名称含糊不清，就容易引起误解和争论，导致建模开销的增加和建模效率的降低。一般情况下，使用一个动词或者动词短语命名关联关系。图 3-28 显示的是同一关联的两个不同的名称，即“holds”和“is holded by”。

在命名关联关系时存在如下假定：如果要从相反的方向理解该关联，只需将关联名称的意义反过来理解。例如图 3-28 中的关联可以理解为“Person 对象拥有 Car 对象”，如果从相反的方向理解也是可以的，即“Car 对象被 Person 对象拥有”。因而对于图 3-28 中的关联，只需建立其中一个模型即可。

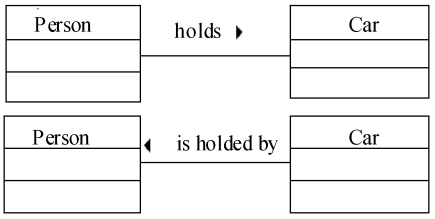



图 3-28 关联的名称

通常情况下，人们喜欢从左到右地阅读，所以当希望读者从右向左阅读时，应使用某种方法告诉读者，这时就可以使用方向指示符。可以将方向指示符放在关联名称的某一侧，以向读者说明应如何理解关联名称。图 3-28 中两个关联名称都使用方向指示符，该指示符指的是两个黑三角。事实上第一个不必使用，因为该名称的阅读顺序符合人们的阅读习惯；只有在阅读顺序不符合人们的阅读习惯时，才有必要使用方向指示符。

对关联进行命名是为了清晰而简洁地说明对象间的关系，同时可以用于指导对对象之间的通信方式进行定义，也决定每个对象在通信中所扮演的角色。

### 2．关联的端点

为了定义对象在关联中所扮演的角色，UML 将关联中的每个端点都作为具有相应规则的独立实体。因而，在“holds”关联中 Person 对象的参与跟 Car 对象的参与是不同的。

每个关联端点都包含了如下的内容：端点上的对象在关联中扮演什么角色，有多少对象可以参与关联，对象之间是否按一定的顺序进行排列，是否可以用对象的一些特征对该对象进行访问，以及一个端点的对象是否可以访问另一个端点的对象等。

关联端点可以包含诸如角色、多重性、定序、约束、限定符、导航性、可变性等特征中的部分或者全部。

### 3．关联中的角色

角色是关联关系中一个类对另一个类所表现出来的职责，任何关联关系中都涉及到与此关联有关的角色，也就是与此关联相连的类的对象所扮演的角色。在图 3-29 中，人在“enjoy”这一关联关系中扮演的是观众这一角色；演出是演员表演的结果，因而 Performance 对象所扮演的角色就是演员。

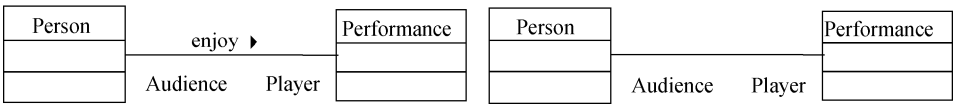



图 3-29　关联中的角色

与关联名称相比，角色名称从另外一个角度描述了不同类型的对象是如何参与关联的。关联中的角色通常用字符串命名，角色可以是名词或名词短语，以解释对象是如何参与关系的。类图中角色名通常放在与此角色有关的关联关系（代表关联关系的直线）的末端，并且紧挨着使用该角色的类。角色名不是类的组成部分，一个类可以在不同的关联中扮演不同的角色。

由于角色名称和关联名称都被用来描述关系的目的，所以角色名称可以代替关联名称，或者两者同时使用。例如图 3-29 中前面的模型同时使用了关联名称和角色名称，后面的模型只使用了角色名称，这两种表示关联的方法都是可行的。

与关联的名称不同，位于关联端点的角色名可以生成代码。每个对象都需要保存一个参考值，该参考值指向一个或者多个关联的对象。在对象中，参考值是一个属性值，如果只有一个关联，就只有一个属性来保存参考值。在生成的代码中，属性使用参考对象的角色名命名。

### 4．可见性

相关人员可以使用可见性符号修饰角色名称，以说明该角色名称可以被谁访问，如图 3-30 所示。

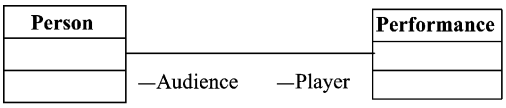



图 3-30 关联中的可见性

图 3-30 中类 Performance 的参考值指向角色名称“–Audience”，该角色名称前面的“–”表示可见性类型为 Private，这说明类 Performance 包含一个私有属性，它保存了一个参考值指向 Person 对象。

**提　示**

由于图 3-30 中属性的可见性为 Private，所以要想访问该属性，则需要使用一个可见性不是 Private 的操作。另外在 UML 2.0 版本中关联端点中已不再使用可见性的概念。

### 5．多重性

关联的多重性指的是有多少对象可以参与关联，它可用来表达一个取值范围、特定值、无限定的范围或者一组离散值。在 UML 中多重性是用由数字标识的范围来表示的，其格式为“mininum..maxinum”，其中 mininum 和 maxinum 都表示 int 类型。例如 0..9，它所表示的范围的下限为 0，上限为 9，下限和上限用两个圆点进行分隔，该范围表示所描述实体可能发生的次数是 0 到 9 中的某一个值。

多重性也可以使用符号“*”来表示一个没有上限或者说上限为无穷大的范围。例如，范围 0..*表示所有的非负整数。下限和上限都相同的范围可以简写为一个数字。例如，范围 2..2 可以用数字 2 来代替。

除上面介绍的表示外，多重性还可以用另外一种形式来表示，即用一个由范围和单个数字组成的列表来表示，列表中的元素通常以升序形式排列。例如，有一个实体是可选的，但如果发生的话就必须至少发生两次以上，那么在建模时就可以用多重性 0,3..*来表示。

赋给一个关联端点的多重性表示在该端点可以有多个对象与另一个端点的一个对象关联。例如，图 3-31 中所示的关联具有多重性，它表示一个人可以拥有 0 辆或者多辆小汽车。

### 6．定序

在关联中使用多重性时可能会有多个对象参与关联，当有多个对象时，还可以使用定序约束，定序就是指将一组对象按一定的顺序排列。UML 规范中布尔标记值 ordered 用于说明是否要对对象进行排序。要指出参与关联的一组对象需要按一定的顺序排列，只需将关键字{ordered}置于关联端点处就可以了。例如图 3-32 中，一个 Person 对象可以拥有多个 Car 对象，这些 Car 对象被要求按照一定的顺序进行排列。如果对象不需要按照一定的顺序进行排列，那就可以省略关键字{ordered}。

前面已经介绍过在系统实现时关联被定义为保存了参考值的属性，该参考值指向一组参与关联的对象。在为对象规定了定序约束之后，对象必须按照一定的顺序排列，因

此实现关联标准时必须考虑关联的标准，以及如何在保持正确顺序的前提下，向队列中添加对象或者从队列中删除对象。

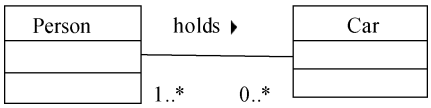



图 3-31 关联的多重性

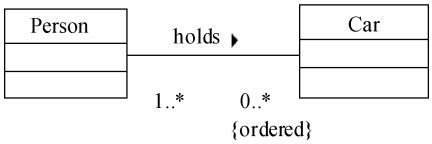



图 3-32 关联端点的定序约束

### 7．约束

UML 定义了 3 种扩展机制：标记值、原型和约束。约束定义了附加于模型元素之上的限制条件，保证了模型元素在系统生命周期中的完整性。约束的格式实际上是一个文本字符串（使用特定的语言表达），几乎可以被附加到模型中的任何元素上。约束使用的语言可以是 OCL、某种编程语言，甚至也可以是自然语言，如英文、中文等。

在关联端点上，约束可以被附加到{ordered}特性字符串里，例如图 3-33 中，ordered 后面添加了一个约束，该约束限定与 Person 对象关联的一个 Car 对象的价格不能超过 $100 000。

约束规定了实现关联端点时必须遵守的一些规则。如前所述，关联是使用包含参考对象的属性来表现的，在系统实现时需要编写一些方法以创建或者改变参考值，关联端点的约束就是在这些方法中实现的。

关联端点上的约束还可以用于限定哪些对象可以参与关联。例如，某国为了保护本国的汽车制造业，规定本国公民只能购买国产的小汽车，而不能购买市场上的非国产小汽车，这时可以在模型中使用约束，用布尔值 homemade 来表示，如图 3-34 所示。

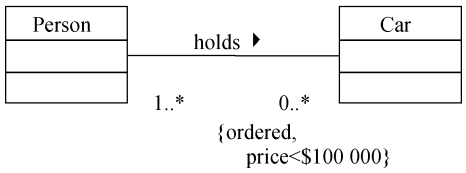



图 3-33 关联端点上的约束

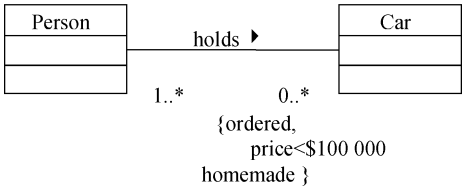



图 3-34 关联中的约束

**提　示**

约束条件的作用对象是靠近它的关联端点的类，在模型中使用约束时，要使约束条件靠近它所作用的类。在不熟悉 OCL 之前，可以使用自然语言表示约束。

### 8．限定符

限定符定义了被参考对象的一个属性，并且可以将该属性作为直接访问被参考对象的关键字。当需要使用某些信息作为关键字来识别对象集合中的一个对象时，可以使用限定符，使用限定符的关联被称为受限关联。

限定符提供了一种切实可行的实现直接访问对象的方法。要建立限定符的模型，首

先必须确定希望直接访问的对象的类型，以及提供被访问的对象的类型，限定符被放在希望实现直接访问的对象附近。

在现实系统中，限定符和用作对象标识的属性之间通常是密切联系的。例如图 3-35 中，Class 具有每个学生的信息，每个学生都有唯一的标识。但是该类图中并没有清楚地指出每个学生的编号是否是唯一的。

为了能够在类图中描述这一约束，建模者通常将用作标识的属性 stuID 作为类 Class 的一个限定符，如图 3-36 所示。对于识别对象身份这类问题来说，没有必要在数据模型中引入一个充当标识的属性，而应该用限定符来描述对象的标识。

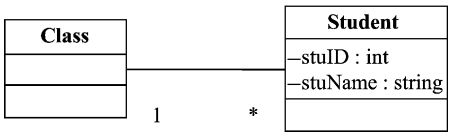



图 3–35 关联示例

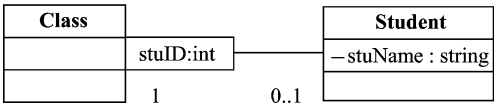



图 3–36 受限关联

### 9．导航性

导航性用来描述一个对象通过链进行导航访问另一个对象，也就是说，对一个关联端点设置导航属性意味着本端的对象可以被另一端的对象访问。导航性使用置于关联端点的箭头表示。如果存在箭头就表示该关联端点是可导航的，反之则不成立。例如图 3-37 中“holds”关联靠近类 Car 的端点的导航性被设置为真，UML 使用一个指向 Car 的箭头表示，这意味着另一端的 Person 对象可以访问 Car 对象。

所以，如果两个关联端点都是可导航的，就应该在关联的两个端点处都放置箭头，但在这种情况下，大多数建模工具采用了默认的 0 表示方法，即两个箭头都不显示。原因是：大多数关联都是双向的，因而，除非特别声明，一般都把代表导航性的箭头省略了。但是如果采用默认表示方法，在生成代码时，指向关联对象的参考值将被作为对象属性实现，并且会有一些操作负责处理该属性，操作和属性最终被写成代码，其中自然也包括了作为关联端点一部分的导航性，这样势必会增加代码量，并且增加编码和维护方面的开销。

**注　意**

千万不要把导航箭头和方向指示符混淆了，前者一般被置于关联直线的尾部，而方向指示符则置于关联名称的左侧或右侧。

### 10．可变性

可变性允许建模者对属于某个关联的链进行操作，默认情况是允许任何形式的编辑，例如，添加、删除等。在 UML 中可变性的默认值可以不在模型中表现出来。但是如果需要对可变性做些限定，则需要将可变性的取值放在特性字符串中，和定序以及约束放在一起。在预定义的可变性选项中，{frozen}表示链一旦被建立，就不能移动或者改变，如果应用程序只允许创建新链而不允许删除链，则可以使用{addOnly}选项。

图 3-38 所示为类 Contract 和类 Company 之间的关联模型。它表示某大学和某建筑

公司签订合同，由建筑公司负责建造该大学的图书馆，合同是两者之间的法定关系，为了避免意想不到的错误删除，在该关联的 Contract 端点上设置了{frozen}特性。

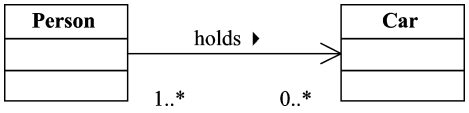



图 3-37 导航性

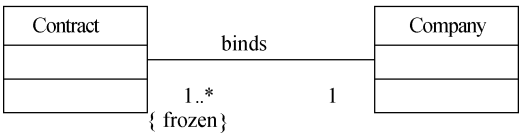



图 3-38 可变性

## 3.4.2 关联类

有时关联本身会引进新类，在想要显示一个类涉及到两个类的复杂情况中，关联类就显得特别重要。关联类就是与一个关联关系相连的类，它并不位于表示关联关系的直线两端，而是对应一个实际的关联，用关联类表示该关联的附加信息。关联中的每个连接与关联类中的一个对象相对应。

虽然类的属性描述了类实例所具有的特性，但有时却需要将对象的有关信息和对象之间的链接放在一起，而不是放在不同的类中。图 3-39 演示了 Student 和 Course 之间的 Elect 关联。

关联类是一种将数据值和链接关联在一起的手段，使用关联类可以增加模型的灵活性，并能够增强系统的易维护性，因此应该在模型中尽量使用关联类。UML 中关联类是一 种模型元素，它同时具有关联和类的特性。

关联类和其他的类非常相似，两者之间的区别就在于对它们的使用需求不同。一般的类描述的都是某个实体，即看得见摸得着的东西，而关联类描述的则是关系。它可以像关联那样将两个类连接在一起，也可以像类一样具有属性，其属性用来存储相应关联的信息。

如果用户需要记录学生所选课程的成绩，再使用图 3-39 就不能符合其要求了。重新以学生、课程和成绩为例，课程的得分并不是学生本来就有的，只有在学生选修了某门课程后才会有所选课程的得分，也就是说，课程的得分可以将学生和课程关联起来。图 3-40 所示的关联类用来存储学生选修的某一门课程的成绩，该关联类代替了图 3-39 中的关联关系。关联类的名字可以写在关联的旁边，也可以放在类标志的名称分栏当中，关联类的标志要用一条虚线与它所代表的关联连接起来。

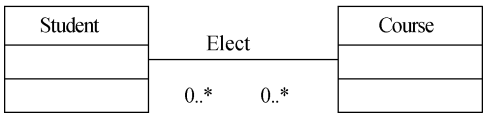



图 3-39 Elect（选课）关联

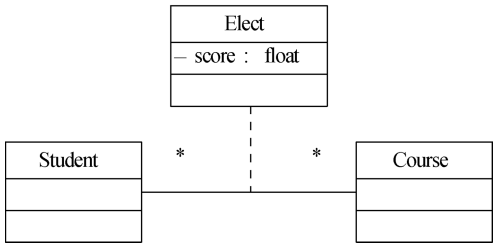



图 3-40 关联类

假设要求每个学生必须明确登记所选的课程，那么每个登记项中应包含所选课程的得分及其授课学期。可以认为班级是由若干名选修同一课程的学生组成的，将班级定义

为登记项的集合，即班级是由特定学期选修相同课程的学生组成的。如图 3-41 所示，通过一个用来识别对应于特定类的登记项的关联可以描述这种情况。

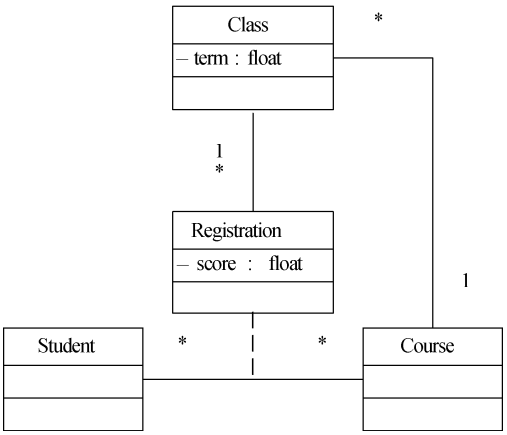



图 3-41 参与关联的关联类

## 3.4.3 或关联与反身关联

前面已经介绍过一个类可以参与多个关联关系，图 3-42 所示是保险业务的类图。个人可以同保险公司签订保险合同，其他公司也可以同保险公司签订保险合同，但是个人持有的合同不同于一般公司持有的合同，也就是说，个人与保险合同的关联关系不能跟公司与保险合同的关联关系同时发生。当这两个关联不能同时并存时，应该怎样表示呢？

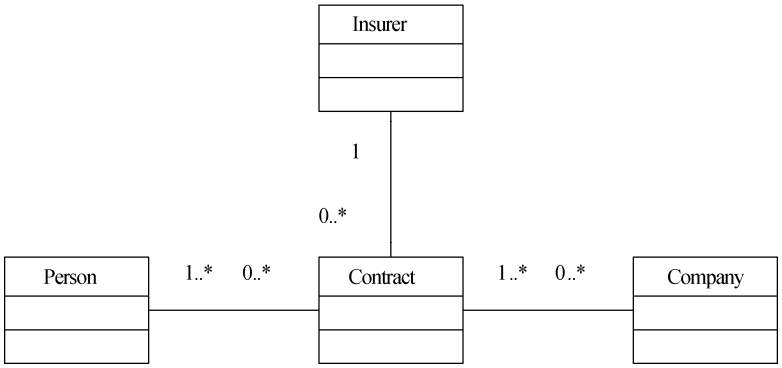



图 3-42 保险业务类图

答案很简单，UML 提供了一种或关联来建模这样的关联关系，或关联是指对多个关联附加约束条件，使类中的对象一次只能参与一个关联关系。或关联的表示方法如图 3-43 所示，当两个关联不能同时发生时，用一条虚线连接这两个关联，并且虚线的中间带有{OR}关键字。

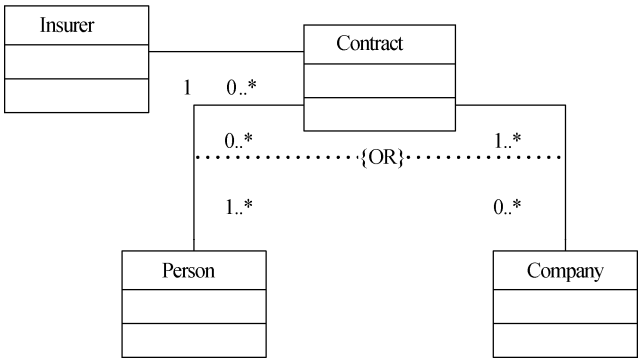



图 3-43 或关联示例

或关联以及前面介绍的其他关联都涉及到了多个类，但是有时候，参与关联的对象属于同一个类，这种关联被称为反身关联。例如不同的飞机场通过航线关联起来，用 Airport 类表示机场，那么 Airport 对象之间的关联关系就只涉及到了一个类。

当关联关系存在于两个不同的类之间时，关联直线从其中的一个类连接到另一个类，而如果参与关联的对象属于同一个类，那么关联直线的起点和终点都是该类，如图 3-44 所示。

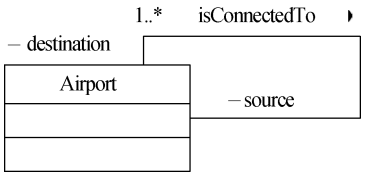


图 3–44　反身关联

图 3-44 中该关联只涉及了一个类 Airport。反身关联通常要使用角色名称。在二元关联中描述一个关联时需要使用类的名称，但在反身关联中，只使用类表达关联的意义可能比较模糊，而使用角色名则会更清晰一些。

## 3.4.4　聚合关系

聚合（Aggregation）关系是在关联之上进一步的紧密耦合，用来表明一个类实际上拥有但可能共享另一个类的对象。聚合关系是一种特殊的关联关系，它表示整体与部分的关系，且部分可以离开整体而单独存在。在聚合关系中一个类是整体，它由一个或者多个部分类组成，当整体类不存在时部分类仍能存在，但是当它们聚集在一起时，就用于组成相应的整体类。例如车和轮胎就可以看作是聚合关系，车为整体，轮胎为部分，轮胎离开车后仍然可以存在。

在表示聚合关系时，需要在关联实线的连接整体类那一端添加一个菱形，图 3-45 所示演示了一个简单的聚合关系。

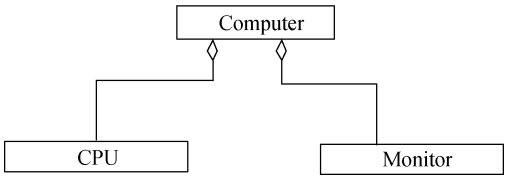


图 3–45　聚合关系示例

在图 3-45 中，CPU 类和 Monitor 类与 Computer 类之间的关系远比关联关系更强。CPU 类和 Monitor 类都可以单独存在，但是当它们组成 Computer 类时，就会变为整个计算机的组成部分。

**提　示**

由于聚合关联的部分类可以独立存在，这意味着当整体类销毁时部分类仍可以存在。如果部分类被销毁，整体类也将能够继续存在。

## 3.4.5　组合关系

在类的众多关系中，再加强一步的耦合是组合关系，组合关系也是一种特殊的关联关系，在某种情况下也可以说它是一种特殊的聚合关系。组合关系是比聚合关系还要强的关系，它要求普通的聚合关系中代表整体的对象负责代表部分对象的生命周期。

组合关系和聚合关系很相似，都是整体与部分的关联关系，但是它们之间的不同处在于部分不能够离开整体而单独存在，当整体类被销毁时，部分类将同时被销毁。例如公司和部门是整体和部分的关系，没有公司就不存在部门。

组合关系所表达的内涵是为组成类的内在部分建模。表示组成关系的符号与聚合关系类似，但是端末的菱形是实心的。图 3-46 所示演示了一个简单的组合关系示例图。

图 3-46 中代表数据库的整体类 DBEmployee 由表 TableEmployee 和表 Employee 组成，这些关联使用组成关系表示。如果数据库不存在了，数据库中的表也就不存在了。

组合关系还可以进行嵌套，如图 3-47 所示。

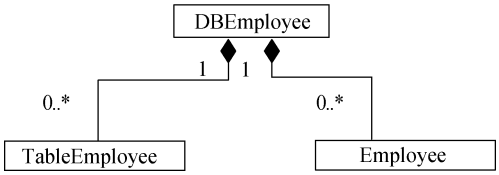



图 3-46 组合关系示例图

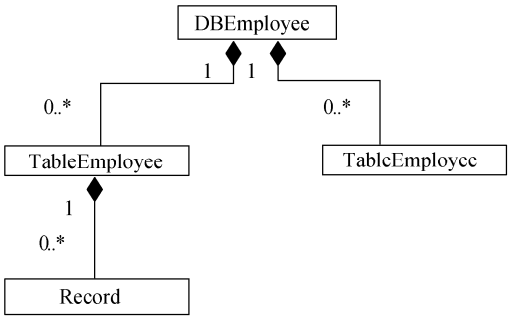



图 3-47 组合关联的嵌套

图 3-47 中添加了 Record 类，可以将该类作为 TableEmployee 的部分类。该图也说明表 TableEmployee 中有 0 个或者 0 个以上的记录，也表达了记录不能离开表单独存在这一客观情况。

## 3.5 实例：创建 BBS 论坛类图

构建类图模型就是要表达类图及类图之间的关系，以便于理解系统的静态逻辑。类图模型的构造是一个迭代的过程，需要反复进行，通过分析用例模型和系统的需求规格说明可以初步构造系统的类图模型，随着系统分析和设计的逐步深入，类图将会越来越完善。

### 3.5.1 创建实体类

从 2.3.1 节中的需求分析和 2.3.3 节中的用例图可以将论坛系统划分为 10 个类，它们分别是：管理员、版主、会员用户、普通用户、版块、提出建议、帖子、请求信息、回复信息、新手手册。

#### 1. 管理员类

管理员用于记录管理员的基本信息和登录时间，它是与整个系统相关的核心类。管理员类中可以包含多个属性和操作，如属性包括管理员姓名、账号、登录时间和联系电话等，而操作可以包括添加版块、删除版块、关闭版块、添加会员、删除会员，以及提出建议等。图 3-48 所示为管理员类的类图。

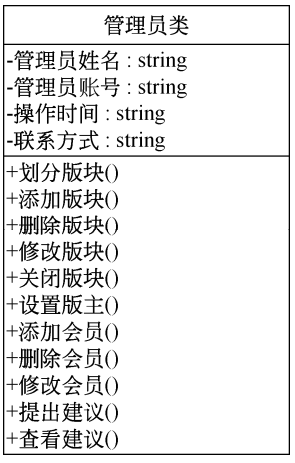



图 3-48 管理员类的类图

**注　意**

图 3-48 以及下面其他相关的类图中，属性和操作仅仅列出了一些常用的信息，并不是所有的类图中都将属性和操作一一列出。

### 2．版主类

版主类用于记录版主的基本信息和与该版主有关的版块，版主在管理版块的同时，也会保留会员身份。像管理员类一样，版主类中也可以包含多个属性和操作，如属性包含版主账号、版主姓名和版主级别等，操作则包括设置热门帖子、设置精华帖子等。图 3-49 所示为版主类的类图。

### 3．会员用户类

会员类记录与会员相关的基本信息和操作，该类中可以包含会员名称、会员账号、会员等级、会员的发帖数量、回帖数量，以及登录时间等属性内容，也可以包含会员发帖、会员回帖和浏览帖子等操作。图 3-50 所示为会员类的类图。

### 4．普通用户类

普通用户类即没有注册的用户类，在该类中没有固定的信息，所以也没有明确记录用户信息的属性。但是如果用户注册成为会员时，则会记录用户申请的会员号，注册成功后能够顺利转为会员。图 3-51 所示为普通用户类的类图。

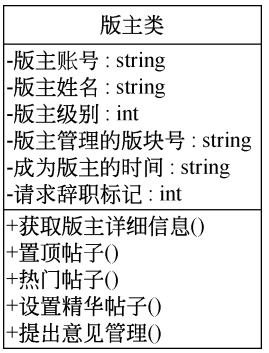



图 3-49　版主类的类图


图 3-50　会员类的类图


图 3-51　普通用户类的类图

### 5．版块类

版块类记录了与版块相关的基本信息和相关操作，还记录了当前版块是否能够关闭，如果关闭了则不能发表帖子。另外，在版块相关操作中还有显示版块的详细信息，例如单击某个版块的链接时，会自动调用操作内容显示版块详情。图 3-52 所示为版块类的类图。

### 6．提出建议类

提出建议类记录了会员用户和普通用户建议的基本信息，如用户提出意见的时间、

提出建议者的账号、建议属性和建议记录等内容。图 3-53 所示为建议类的类图。

在图 3-52 中，单击某个版块链接显示详细内容时调用相应的操作，单击某个版块后，管理员可以根据自己的需要设置需要关闭版块的标记操作，当设置或取消某个版块标记后，会自动调用该操作更新关闭版块列表。

### 7．帖子类

帖子类中可以包含多个属性与操作，如帖子属性中包含帖子 ID、帖子的点击次数和帖子作者的账号等；帖子操作中可以包含帖子的详细信息和帖子列表等。图 3-54 所示为帖子类的类图。

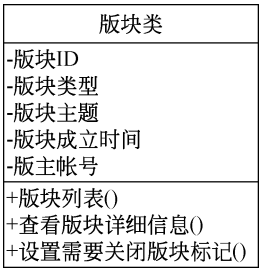



图 3-52 版块类的类图

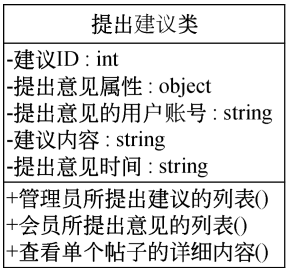



图 3-53 建议类的类图

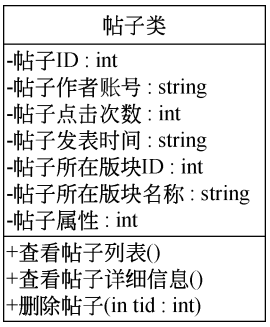



图 3-54 帖子类的类图

### 8．请求信息类

请求信息包含属性和操作两部分，属性部分记录了请求信息的类型，用户可以根据请求类型的选择来调用相应的操作，调用操作完成后，则自动调用设置请求标记，请求信息类的类图如图 3-55 所示。

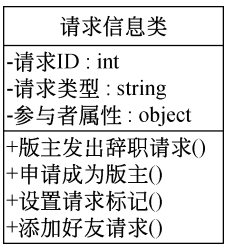



图 3-55 请求信息类的类图

### 9．回复信息类

回复信息类是与请求信息类相反的一个过程，该类会根据回复类型来选择调用哪个操作，调用完毕后会自动设置回复标记记录结果，图 3-56 所示为回复信息类的类图。

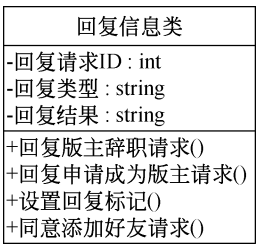



图 3-56 回复信息类的类图

### 10．新手手册类

论坛系统中新手手册只有一份，因此该类中只需要记录形成时间和更新时间即可，不需要再记录其他的详细信息。与该类相关的类图不再具体显示。

## 3.5.2 创建类与类之间的关系图

类与类之间可以存在多种关系，如泛化关系、依赖关系、组合关系和聚合关系等。

前面已经介绍过与论坛系统相关的 10 个类，图 3-57 所示为这些类之间的关系图。由于之前已经列出了大多数类的属性和操作，所以该关系图中不再显示相关属性和操作，而直接使用相关的类。

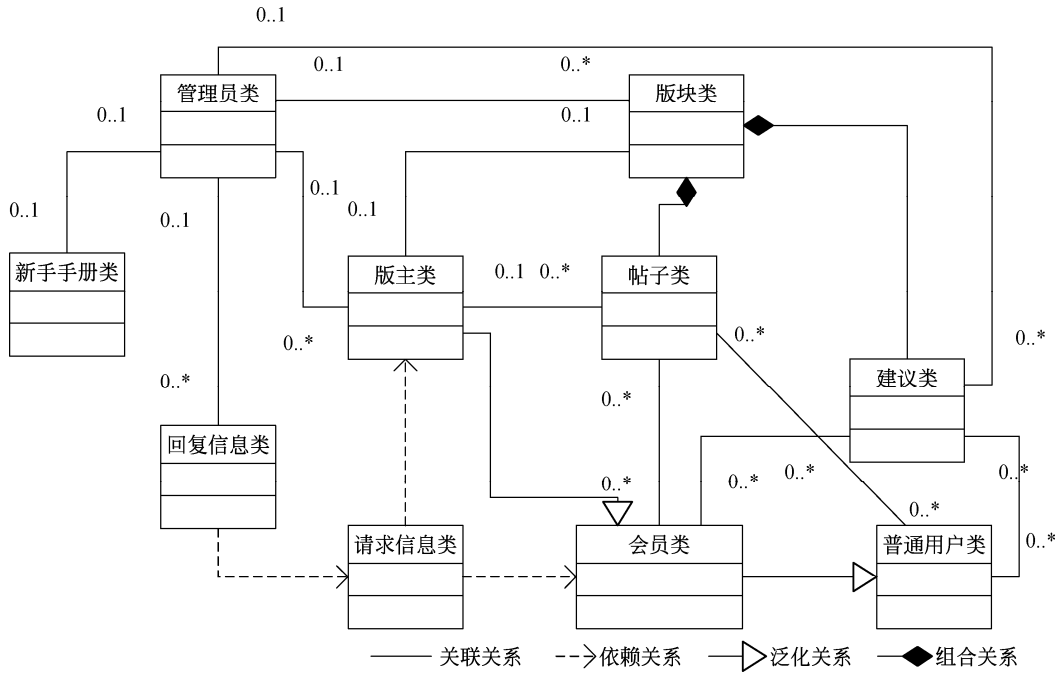



图 3-57　类与类之间的关系类图

从图 3-57 中可以看到管理员类与建议类存在一对多的关联关系、管理员类与版块类存在一对多的关系、版块类与帖子类是组合关系，以及建议类与版块类是组合关系等。下面只挑选几种常见的类关系进行介绍。

- **管理员类对版主类**　一对多的关联关系，管理员可以管理多个版主，而系统管理员只能有一个。
- **管理员类对回复信息类**　一对多的关联关系，管理员可以接收多个用户的请求信息，并对这些信息进行回复。
- **帖子类对版块类**　组合关系，帖子是构成版块的重要部分，它对版块来说是必不可少的。
- **建议类对版块类**　组合关系，管理员可以向会员和版主提出意见，而版块内需要有接收建议的地方，可以说建议是版块的一部分。
- **回复信息类对请求信息类**　依赖关系，回复信息依赖于请求信息类，请求信息类发生变化，则回复信息也会发生变化。
- **请求信息类对版主类**　依赖关系，请求信息的操作依赖于版主类的对象，如果对象发生变化，则请求信息也发生变化，因此请求信息依赖于版主类。
- **请求信息对会员类**　依赖关系，请求信息的操作也依赖于会员类的对象，如果会员对象发生变化，则请求信息也会发生变化，因此请求信息依赖于会员类。
- **版主类对会员类**　泛化关系。
- **会员类对普通用户类**　泛化关系。

## 3.6 思考与练习

### 一、填空题

1．_____是面向对象系统建模中最常用和最基本的图之一。

2．泛化约束可以分为不完全约束、完全约束、_____和重叠约束。

3.UML规范中定义了4种基本的依赖类型，它们分别是_____、抽象依赖、绑定依赖和授权依赖。

4．_____用来描述整体与部分，但是部分不能够离开整体而单独存在，当整体类被销毁时，部分类将同时被销毁。

5．组合关系和_____都是一种特殊的关联关系，它们都描述了整体与部分的关系。

### 二、选择题

1．下面关于依赖关系的说法，选项_____是正确的。

A．依赖关系的 4 种类型包括绑定依赖和调用依赖

B．依赖关系的 4 种类型包括抽象依赖和调用依赖

C．依赖关系用一个一端带有箭头的虚线表示

D．依赖关系使用一个一端带有箭头的实线表示

2．关于 UML 类图中的关系，下面说法不正确的是_____。

A．聚合关系和组合关系是特殊的关联关系，它们都描述了整体与部分的关系

B．UML 中的类图关系只有 3 种：泛化关系、关联关系和依赖关系

C．UML 中的常用的类图关系有泛化关系、关联关系、依赖关系和实现关系

D．UML 类图中常用关系的强弱顺序为：泛化=实现>组合>聚合>关联>依赖

3．类定义了一组具有状态和行为的对象，这些对象具有相同的属性、操作、关系和语义。其中属性和_____用来描述状态。

A．依赖

B．操作

C．关系

D．语句

4.定序是指将一组对象按一定的顺序排列，要指出参与关联的一组对象需要按一定的顺序排列，只需将关键字_____置于关联端点处就行了。

A．{ordered}

B．{orderer}

C．{OR}

D．{incomplete}

### 三、问答题

1．类图中的主要元素是什么？

2．类与类之间的主要关系有几种？它们的含义是什么？

3．简述使用类图时要遵循的基本原则。

4．简述聚合关系和组合关系的相同点和不同点。