第5章
关　　系

本章内容

- 依赖、泛化和关联关系
- 对简单依赖建模
- 对单继承建模
- 对结构关系建模
- 创建关系网

当建造抽象时，你就会发现类很少单独存在。相反，大多数类以几种方式相互协作。因此，当对系统建模时，不仅要识别形成系统词汇的事物，而且还必须对这些事物如何相互联系建模。

在面向对象的建模中，有3种特别重要的关系：依赖（*dependency*），它表示类之间的使用关系（包括精化、跟踪和绑定关系）；泛化（*generalization*），它把一般类连接到它的特殊类；关联（*association*），它表示对象之间的结构关系。其中的每一种关系都为组织抽象提供了不同的方法。【在第10章中讨论关系的高级特征。】

构造关系网与创建类之间职责的均衡分布是相似的。过分细致的设计，将导致关系混乱，使得模型不可理解；对设计考虑的过少，将会丢失系统中以一定方式协作的事物所含的许多有用信息。

5.1　入门

如果你正在建造一所房子，像墙、门、窗户、橱柜和照明灯具这样的事物将形成部分词汇。然而这些事物都不是单独存在的。墙要与别的墙相连接，门窗要安在墙上，形成分别供人们出入和采光的开口。橱柜和照明灯具自然要安在墙上和天花板上。把墙、门、窗户、橱柜和照明灯具组合在一起，就形成了像房间这样较高层次的事物。

在这些事物中，不仅能发现结构关系，而且也能发现其他种类的关系。例如，房子肯定有窗户，但窗户的种类可能有很多。可能有不能开的凸型大窗户和能开的小厨房窗户。一些窗户能上下开；而另一些窗户（像通向庭院的窗户）可以左右拉开。一些窗户仅有一块玻璃，另一些窗户有两块玻璃。无论它们多么不同，它们都具有一些基本的窗户要素：每个窗户都是墙上的一个开口，被设计用于采光和通气，有时还能过人。

在UML中，事物之间相互联系的方式，无论是逻辑上的还是物理上的，都被建模为关系。在面向对象的建模中，有3种最重要的关系：依赖、泛化和关联。

依赖（*dependency*）是使用关系。例如，水管依赖水加热器对它们所运送的水进行加热。

泛化（*generalization*）把一般类连接到较为特殊的类，也称为超类/子类关系或父/子关系。例如，凸窗是一种带有固定窗格的大窗户；通向庭院的窗是一种带有向两边开的窗格的窗户。

关联（*association*）是一种实例之间的结构关系。例如，房间是由墙和一些其他事物组成的，墙体可以镶嵌门窗，管道可以穿过墙体。

这3种关系覆盖了大部分事物之间相互协作的重要方式。显然，这3种关系也能很好地映射到大多数面向对象编程语言所提供的连接对象的方式。【另外几种关系，如实现和精化，在第10章中讨论。】

UML对每种关系都提供了一种图形表示，如图5-1所示。这种表示法允许脱离具体的编程语言而对关系进行可视化，在某种程度上可使你强调关系的最重要的部分：关系名、关系所连接的事物和关系的特性。

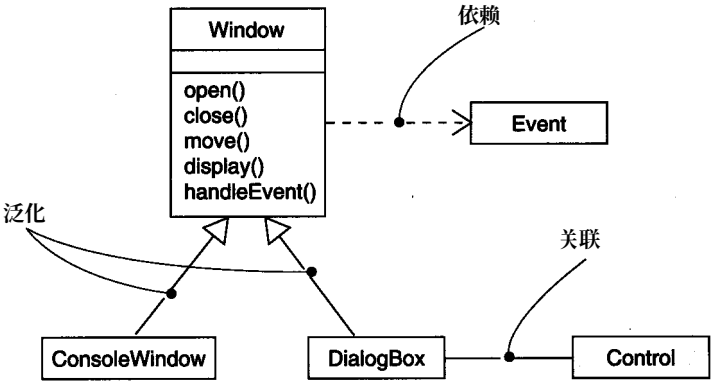

图5-1 关系

5.2 术语和概念

关系（*relationship*）是事物之间的联系。在面向对象的建模中，最重要的3种关系是依赖、泛化和关联。在图形上，把关系画成一条线，并用不同的线区别关系的种类。

1. 依赖

依赖（*dependency*）是一种使用关系，它说明一个事物（如类`Event`）规格说明的变化可能影响到使用它的另一个事物（如类`Window`），但反之未必。在图形上，把依赖画成一条有向的虚线，指向被依赖的事物。当要指明一个事物使用另一个事物时，就使用依赖。

在大多数情况下，在类的语境中用依赖指明一个类把另一个类作为它的操作的特征标记中的参数，参见图5-2。这的确是一种使用关系，如果被使用的类发生变化，那么另一个类的操作也会受到影响，因为这个被使用的类此时可能表现出不同的接口或行为。在UML中，也可以在很多其他的事物之间创建依赖，特别是注解和包。【第6章讨论注释；第12章讨论包。】

注释 依赖可以带有一个名称，但很少使用，除非模型有很多依赖，并且要引用它们或作出区别。在一般情况下，用构造型区别依赖的不同含义。【在第10章中讨论不同种类的依赖；在第6章中讨论构造型。】

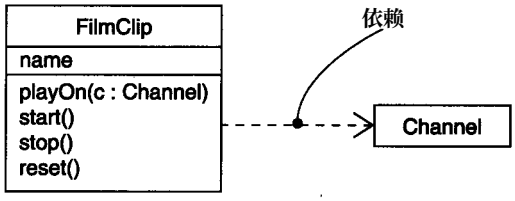

图5-2 依赖

2. 泛化

泛化（*generalization*）是一般事物（称为超类或父类）和该事物的较为特殊的种类（称为子类或儿子）之间的关系。有时也称泛化为“ is-a-kind-of ”关系：一个事物（如类`BayWindow`）是更一般的事物（如类`Window`）的“一个种类”。泛化意味着子类的对象可以被用在父类的对象可能出现的任何地方，但反过来不这样。换句话说，泛化意味着子类可以替换父类。子类继承父类的特性，特别是父类的属性和操作。通常（但不总是），子类除了具有父类的属性和操作外，还具有别的属性和操作。若子类的操作与父类的操作的特征标记相同，则它重载父类的操作，这称为多态性。在图形上，把泛化画为一条带有空心大箭头的有向实线，指向父类，如图 5-3所示。当要表示父/子关系时，就使用泛化。

一个类可以有0个、1个或多个父类。把没有父类且最少有一个子类的类称为根类或基类；把没有子类的类称为叶子类。可以说只有一个父类的类使用了单继承；也可以说有多个父类的类使用了多继承。

在大多数情况下，用类和接口间的泛化指明继承关系。在 UML中，也可以在其他的事物间创建泛化，最明显的例子是包。【在第12章中讨论包。】

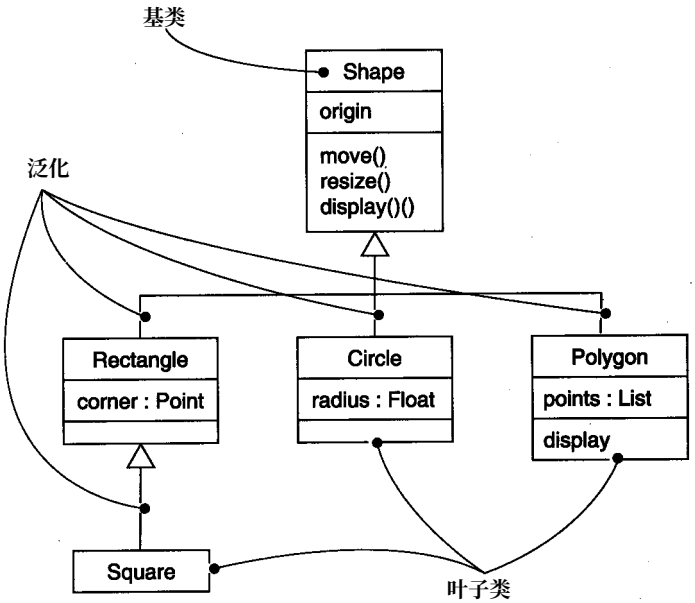

图5-3 泛化

注释 泛化可以有名称，但很少使用。除非模型有很多泛化，而且要引用它们或要加以区分，才需要使用泛化的名称。

3. 关联

关联（*association*）是一种结构关系，它指明一个事物的对象与另一个事物的对象间的联系。给定一个连接两个类的关联，可以从一个类的对象导航到另一个类的对象，反之亦然。关联的两端都连到同一个类是合法的。这意味着，从一个类的给定对象能连接到该类的其他对象。恰好连接两个类的关联叫做二元关联。尽管不普遍，但可以有连接多于两个类的关联，这种关联叫做n元关联。在图形上，把关联画成一条连接相同类或不同类的实线。当要表示结构关系时，就使用关联。【关联和依赖可以是自反的，但泛化关系不是这样，这在第10章中讨论。】

除了这种基本形式外，还有4种应用于关联的修饰。

• 名称

关联可以有一个名称，用以描述该关系的性质。为了消除名称含义的歧义，可提供一个指引读名称方向的三角形，给名称一个方向，如图 5-4所示。【不要把名称方向和关联导航相混淆，在第 10章中讨论这方面的问题。】

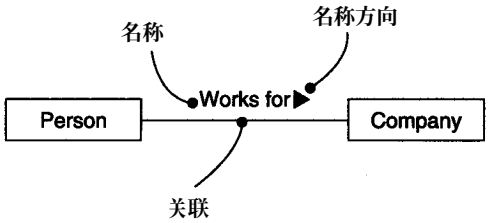

图5-4 关联的名称

注释 虽然关联可以有名称，但通常不需要给出名称。当要明确地给关联提供角色名时，或当一个模型有很多关联，且需要对这些关联进行查阅或区别时，则要给出关联的名称。特别是在有多个关联连接相同类的情况下，更要给出关联的名称。

• 角色

当一个类处于关联的某一端时，该类就在这个关系中扮演了一个特定的角色；角色是关联中靠近它的一端的类对另外一端的类呈现的职责。可以显式地命名类在关联中所扮演的角色。在图5-5中，扮演employee角色的类Person与扮演employer角色的类Company相关联。【角色与接口的语义相关，这在第11章中讨论。】

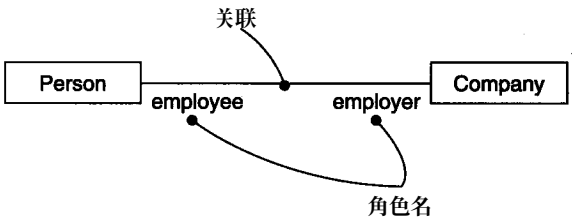

图5-5 角色

注释 相同类可以在其他的关联中扮演相同或不同的角色。

• 多重性

关联表示了对象间的结构关系。在很多建模问题中，说明一个关联的实例中有多少个相互连接的对象是很重要的。这个“多少”被称为关联角色的多重性，把它写成一个表示取值范围的表达式或写成一个具体值，如图 5-6所示。指定关联的一端的多重性，就是说明：在关联另一端的类的每个对象要求在本端的类必须有多少个对象。可以精确地表示多重性为 1（1）、0或1（0..1）、很多（0..*）、1个或很多（1..*）。甚至可以精确地指定多重性为一个数值（如 3）。【关联的实例称为链，这在第15章中讨论。】

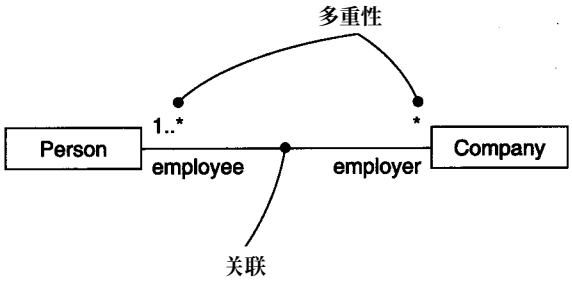

图5-6 多重性

注释 可以使用像0..1、3..4、6..* 这样的列表来描述更为复杂的多重性，该例的含义是“除了2或5外的任意数目的对象”。

• 聚合

两个类之间的简单关联表示了两个同等地位类之间的结构关系，这意味着这两个类在概念上是同级别的，一个类并不比另一个类更重要。有时要对一个“整体/部分”关系建模，其中一个类描述了一个较大的事物（“整体”），它由较小的事物（“部分”）组成。把这种关系称为聚合，它描述了“has-a”关系，意思是整体对象拥有部分对象。其实聚合是一种特殊的关联，它被表示为在整体的一端用一个空心菱形修饰的简单关联，如图 5-7所示。【聚合有一些重要的变种，这在第10章中讨论。】

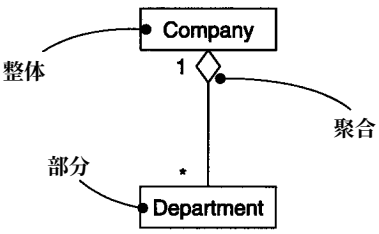

图5-7 聚合

注释 这种简单形式的聚合的含义完全是概念性的。空心菱形只是把整体和部分区别开来。这意味着简单聚合没有改变在整体与部分之间整个关联的导航含义，在整体和部分

的生命期内的链也是如此。

4. 其他特性

简单而未加修饰的依赖、泛化、带有名称、多重性和角色的关联是创建抽象时所需要的最常见的特征。事实上，对于所建的大多数模型，这3种关系的基本形式足以表达关系的最重要的语义。然而，有时需要可视化或详述其他性征，如组合聚合、导航、判别式、关联类、特殊种类的依赖和特殊种类的泛化。这些以及很多其他的特性都可以用UML表达，但它们都被作为高级概念处理。【在第10章中讨论高级关系概念。】

依赖、泛化和关联都是定义在类这一级的静态事物。在UML中，通常在类图中对这些关系进行可视化。【在第8章中讨论类图。】

当你开始在对象级建模时，特别是开始解决这些对象的动态协作时，你将遇到其他两种关系：链（它是关联的实例，描述可能发送消息的对象间的连接）和转换（它是状态机中不同状态间的连接）。【在第15章中讨论链，在第21章中讨论转换。】

5.3 普通建模技术

5.3.1 对简单依赖建模

最普通的依赖关系是两个类之间的连接，其中的一个类只是使用另一个类作为它的操作参数。

为了对这种使用关系建模，要做如下的工作：

- 创建一个依赖，从含有操作的类指向被该操作当做参数使用的类。

例如，图5-8显示了取自一个管理大学中学生选课和教师任课的系统中的一组类。该图显示了一个从CourseSchedule到Course的依赖，因为Course被用作CourseSchedule的操作add和remove的参数。

如果像本图这样给出了一个操作的全部特征标记，通常就不需要给出这个依赖，这是因为对类的使用情况已经显式地写在特征标记中。然而有时要显示这样的依赖，特别是当省略了操作的特征标记，或模型描述了被使用类的其他关系时。

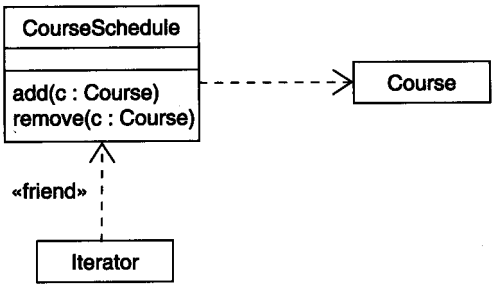

图5-8 依赖关系

本图还显示了另一个依赖关系，这个依赖没有涉及用在操作中的类，而是对C++的一种习惯用法进行建模。这个起自Iterator的依赖表明Iterator使用CourseSchedule；但CourseSchedule不知道有关Iterator的任何信息。用一个构造型来标记这个依赖，这表明它不是一个简单的依赖，而仅表示一个友元（如在C++中）。【在第10章中讨论其他关系构造型。】

5.3.2 对单继承建模

在对系统的词汇建模中，经常会遇到在结构或行为上与其他的类相似的类。可以把这样的每一个类建模为独立的、不相关的抽象。但更好的方法是提取所有共同的结构特征和行为特征，并把它们提升到更一般的类中，特殊类从中继承这些特征。

为了对继承关系建模，要做如下的工作：

- 给定一组类，寻找两个或两个以上的类的共同职责、属性和操作。
- 把这些共同的职责、属性和操作提升到更一般的类中。如果需要，创建一个新类，用以分配这些元素（但要小心不要引入过多的层次）。
- 画出从每个特殊类到它的更一般的父类的泛化关系，用以表示更特殊的类继承更一般的类。

例如，图5-9显示了取自一个贸易应用中的一组类。可以发现从类CashAccount、Stock、Bond和Property到更一般的名为Security的类的泛化关系。Security是父类，CashAccount、Stock、Bond和Property都是子类。每一个这样的特殊的子类都是一种Security。要注意Security包括两个操作：presentValue和history。由于Security是这4个类的父类，因此CashAccount、Stock、Bond和Property都继承了这两个操作，同时也继承了可能在本图中省略了的Security的其他任何属性和操作。

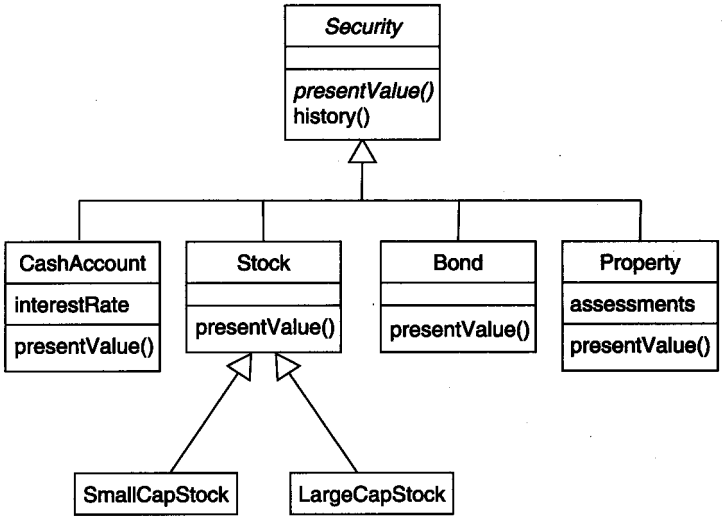

图5-9 继承关系

你可能已注意到Security和presentValue的写法与其他类的写法有所不同，这样做是

有原因的。当建造像图5-9那样的层次时，经常会遇到不完全的或你不想让它有任何对象的非叶子类。通常把这样的类称为抽象类（*abstract class*）。在UML中，通过把类名写为斜体，以指明这个类是抽象的，例如类Security就是如此。这种规定也适用于操作，如presentValue，这意味着这样的操作提供了一种特征标记，但它是不完全的，因此必须在较低的抽象层次用一定的方法实现。事实上，如图5-9所示，Security的4个直接子类都是具体的（即非抽象的），并且也提供了操作presentValue的具体实现。【在第9章中讨论抽象类和抽象操作。】

一般/特殊的层次不必仅限于两层。事实上，像图5-9中那样，多于两层的继承的层次是很常见的。SmallCapStock和LargeCapStock是Stock的两个子类，Stock是Security的子类。由于Security没有父类，因此它是一个基类。由于SmallCapStock和LargeCapStock都没有子类，因此它们是叶子类。Stock既有父类也有子类，因此它既不是根类也不是叶子类。

尽管在本例中没有体现，但实际上也可以为一个类创建多个父类。这种方法称为多继承，这意味着这个类继承它的各个父类的所有属性、操作和关联。【在第10章中讨论多继承。】

当然，在继承格中不能有任何循环，即一个给定的类不能是它自己的父类。

5.3.3 对结构关系建模

当用依赖或泛化关系建模时，是对表示了不同重要级别或不同抽象级别的类建模。给定两个类间的一个依赖，则一个类依赖另一个类，但后者没有前者的任何信息。给定两个类间的一个泛化关系，则子类从它的父类继承，但父类没有任何子类所特有的信息。简而言之，依赖和泛化关系都是单向的。

当用关联关系建模时，是在对相互同等的两个类建模。给定两个类间的一个关联，则这两个类在某些方面相互依赖，从两边都可以导航。然而，依赖是使用关系，泛化是“ is-a-kind-of ”关系，关联描述了类的对象间相互作用的结构路径。【关联在缺省的情况下是双向的，也可以限制它们的方向，在第10章中讨论这些问题。】

为了对结构关系建模，要做如下的工作：

- 对于每一对类，如果需要从一个类的对象到另一个类的对象导航，就要在这两个类之间说明一个关联。这是关联的数据驱动观点。
- 对于每一对类，如果一个类的对象要与另一个类中不作为其操作的参数的对象相互交互，就要在这两个类间说明一个关联。这是关联的行为驱动观点。
- 对于这样的每一个关联，要说明其多重性（特别是当多重性不为 *时，其中*是缺省）和角色名（特别是在有助于解释模型的情况下）。
- 如果关联中的一个类与另一端的类相比，前者较有整体的结构性或组织性，后者看起来像它的部分，则在靠近整体的一端对该关联进行修饰，把它标记为聚合。

怎样才能知道一个给定类的对象何时必须与另一个类的对象相互作用？答案是， CRC卡和用况分析非常有助于考虑结构性和行为性脚本。在两个或两个以上的类进行交互的地方指定一个关联。【在第16章中讨论用况。】

图5-10所显示的是取自一个学校的信息系统中的一组类。从该图的左下部开始，可以找到名称为Student、Course和Instructor的类。在Student和Course之间有一个关联，它描

述了学生听的课程。同时，还描述了每一名学生可以听任意门数的课程，并且每一门课程可以由任意名学生去听。

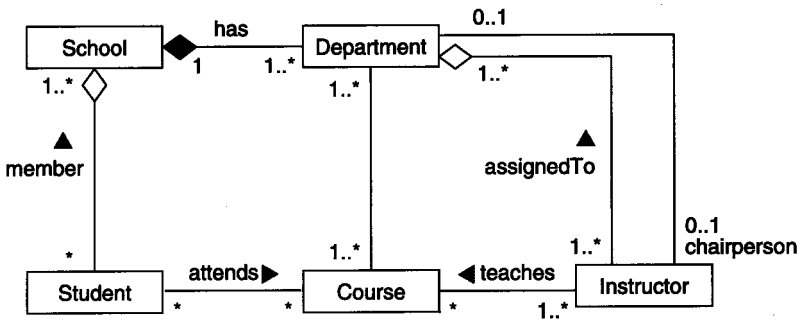

图5-10 结构关系

类似地，在Course和Instructor之间也可以找到一个关联，它描述了教师教的课程。每一门课至少有一名教师，并且每一名教师可以不教课或教多门课。

School、Student和Department间的关系有点不同。在这里可以看到聚合关系。一所学校可以没有学生，或有多名学生，一名学生可以是在一所或多所学校注册的学员，一所学校可以有一个或多个系，每个系只能属于一所学校。可以不用聚合修饰而用简单的关联，但通过说明School是整体，Student和Department是部分，可以解释在组织上哪个高于哪个。因此，学校在一定程度上由学生和学生所在的系来定义。类似地，实际上学生和系并不是与他们所属的学校无关，而是从他们的学校能得到他们的身份。【School和Department之间的聚合关系是组合聚合，在第10章中讨论这个问题。】

你还可以看到，在Department和Instructor之间有两个关联。其中的一个关联说明可以指派一个教师到一个或多个系中，并且一个系可以有一名或多名教师。由于在学校的组织结构中系比教师的层次要高，所以这要用聚合来建模。另一个关联表明一个系只能有一名教师是系主任。这种建模方式说明，一名教师最多是一个系的系主任，并且某些教师不是任何系的系主任。

> 注释 由于这个模型可能没有反映出你所处的现实情况，因此你可以把它作为一种例外。你所在的学校可能没有系。你的系主任可能不是教师，甚至学生也可以是教师。这不意味着这个模型是错误的，只是与你所在的学校的情况有所不同。不能孤立地建模，像这样的每一个模型都依赖于你打算怎样使用这些模型。

5.4 提示和技巧

在UML中对关系建模时，要遵循如下策略：

- 仅当被建模的关系不是结构关系时，才使用依赖。
- 仅当关系是“is-a-kind-of”关系时，才使用泛化。往往可以用聚合代替多继承。
- 小心不要引入循环的泛化关系。

- 一般要保持泛化关系的平衡；继承的层次不要太深（大约多于 5层就应该想一想），也不要太宽（代之以寻找可能的中间抽象类）。
- 在对象间有结构关系的地方，要以使用关联为主。

在UML中绘制关系时，要遵循如下策略：

- 要一致地使用直线或斜线。直线给出的可视化提示强调了相关事物之间的连接都集中到一个共同事物。在复杂的图中斜线则经常有更好的空间效果。在同一个图中使用两种线形，有助于把人们的注意力引导到不同的关系组上。
- 避免线段交叉。
- 仅显示对理解特定的成组事物必不可少的关系。应该省略多余的关系（特别是多余的关联）。