C. 恐怖分子能谈判，而方法学家不能谈判

解 析

UML 是 Unified Modeling Language（统一建模语言）的缩写，是用图形表示软件功能和内部结构的统一的表示方法。

在 UML 被制定之前就已经有了许多面向对象方法论，每种方法论的图形表示方法也是各式各样。主要的方法论就有 Booch 方法、OMT 方法、OOSE 方法、Coad/Yourdon 方法、Shlaer-Mellor 方法、Martin-Odell 方法、Fusion 方法等。提出这些方法论的方法学家都坚定地认为自己的方法论是最优秀的，这也就是为什么 OMG 会说“恐怖分子能谈判，而方法学家不能谈判”。

直到 20 世纪 90 年代中期，三位方法学家葛来迪·布区（Grady Booch）、詹姆士·兰宝（James Rumbaugh）和伊瓦尔·雅各布森（Ivar Jacobson）统一了表示方法，形成了 UML。后来 UML 被国际标准化组织 OMG 定为标准，这才终结了表示方法的混乱状况。

本章重点

本章将为大家介绍 UML。UML 是一个固定形式的世界标准，它将软件功能和内部结构表示为二维图形。如果用一句话来描述 UML，就可以说“UNL 是查看无形软件的工具”。在实际的系统中，程序有几十万行之多，规格说明书等文档也多达几百页。如果使用 UML 图，就可以从庞大的信息中提取出重要的部分，表示为逻辑清晰且直观的形式。

UML 覆盖了整个系统开发工程。这是因为 UML 除了作为面向对象的两方面（编程技术和归纳整理法）的共同成果使用之外，还加入了在面向对象之前就已经使用的图形表示。由于 UML 的适用范围很广，为了在实际开发中能够熟练使用，大家除了要记住各种图形的绘制方法之外，还要掌握各种图形的目的和用途。

8.1 UML 是表示软件功能和结构的图形的绘制方法

虽然 UML（Unified Modeling Language）中有“语言”（language）一词，但它实际上是一种表示软件功能和内部结构的图形的绘制方法。在面向对象领域，图形表示原本是为了表示使用 OOP 编写的程序的结构而被提出的。不过，随着面向对象被用于业务分析和需求定义等上流工程，UML 图形也开始作为上流工程的成果使用。

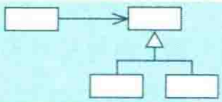
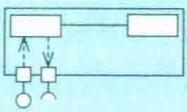
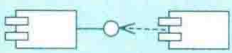
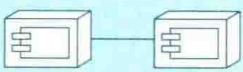
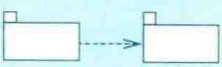
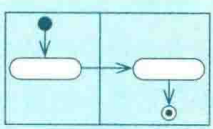
另外，UML 除了有表示类、继承等面向对象特有的结构的图形之外，还包含之前就已经在使用的流程图、状态迁移图等，因此，UML 是软件开发中图形表示的集大成者。

8.2 UML 有 13 种图形

UML 被面向对象相关的国际标准化组织 OMG[1] 定为标准。OMG 最开始采用的是 1997 年的 1.1 版本，之后 UML 不断完善，在 2011 年推出了 2.3 版本。

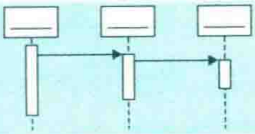
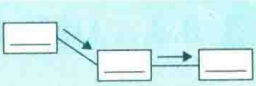
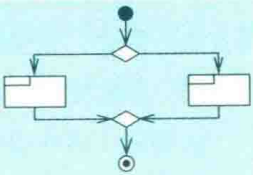
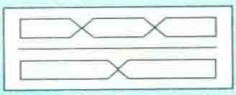
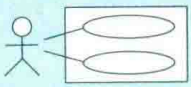
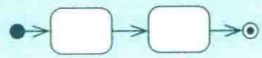
UML 2 中定义了 13 种图形，如表 8-1 所示。

表 8-1 UML 2 中定义的 13 种图形

No.	中文名称	英文名称	用途	图形
1	类图	Class Diagram	表示类的规格和类之间的关系	
2	复合结构图	Composite Structure Diagram	表示具有整体－部分结构的类的运行时结构	
3	组件图	Component Diagram	表示文件和数据库、进程和线程等软件的实现结构	
4	部署图	Deployment Diagram	表示硬件、网络等系统的物理结构	
5	对象图	Object Diagram	表示实例之间的关系	
6	包图	Package Diagram	表示包之间的关系	
7	活动图	Activity Diagram	表示一系列处理中的控制流程	

① 有全世界的几百家企业参加的非营利性的标准化组织。除了 UML 之外，该组织还制定了分布式对象通信的标准 CORBA（Common Object Request Broker Architecture，通用对象请求代理架构）的规范。

（续）

No.	中文名称	英文名称	用　途	图　形
8	时序图	Sequence Diagram	将实例之间的相互作用表示为时间序列	
9	通信图	Communication Diagram	将实例之间的相互作用表示为组织结构	
10	交互概览图	Interaction Overview Diagram	将根据不同条件执行不同动作的时序图放到活动图中进行表示	
11	定时图	Timing Diagram	采用带数字刻度的时间轴来表示实例之间的状态迁移和相互作用	
12	用例图	Use Case Diagram	表示系统提供的功能和使用者之间的关系	
13	状态机图	State Machine Diagram	表示实例的状态变化	

之所以定义这么多图形，是因为设想了其广泛的用途。从将画面上输入的信息存储到数据库中进行使用的商业应用程序，到在个人计算机上运行的单机应用程序、驱动控制机器和电器产品的嵌入式软件等，各种领域中都可以使用这些图形。另外，这些图形还对应于从业务分析到需求定义、设计的整个软件开发工程。

在UML出现之前的很多开发方法论都是对使用面向对象开发系统时的操作步骤、思想等进行系统的汇总。不过，当时根据开发方法论的不同，绘制的图形形式也不同。因此，如果使用的方法论不同，就无法共享需求规格说明书和设计信息。

为了解决这种状况，在 20 世纪 90 年代后半期，三位主要的开发方法论的提出者葛来迪·布区、詹姆士·兰宝和伊瓦尔·雅各布森对图形表示进行了统一，最终提出 UML。因此，UML 的名称中使用了“Unified”（统一）一词。另外，这三位关系密切，被称为**三友**（朋友三人）。

8.3 UML 的使用方法大致分为三种

UML 因为适用范围广，所以定义了许多种图形，但并未规定具体的使用方法。因此，在实际情况下，从众多图形中选择哪一种来使用，以及如何使用，都由使用者来判断。

正如第 7 章中介绍的那样，面向对象是作为编程语言出现并发展至今的，在被应用到上流工程后，又化为表示集合论和职责分配的归纳整理法。UML 是作为编程技术和归纳整理法的共同成果使用的。但实际上，即使同一种图形，根据使用情况的不同，其使用方法也要稍微改变一下。因此，我们将 UML 的使用方法分成两种情况来理解，这样会更容易一些。

另外，UML 中还有一些图形是过去就在使用的，与 OOP 和集合论并没有直接关系。正是因为这些图形被引入了 UML，所以有时也会被作为面向对象的一部分进行介绍。

接下来，我们将介绍一些具有代表性的 UML 图的使用方法，为了明确用途，这里分为以下三种情况进行介绍。

> <UML 的使用方法 >
> 之一：表示 OOP 程序的结构和动作。
> 之二：表示归纳整理法的成果。
> 之三：表示面向对象无法表示的信息。

8.4 UML 的使用方法之一：表示程序结构和动作

我们先来介绍一下 UML 作为表示程序的技术时的相关内容。

当然，程序是使用编程语言编写的。正如第 3 章中介绍的那样，编程语言在向着更容易理解、更容易预防错误的方向进化。不过，程序的最终目的还是驱动计算机。计算机从程序开头逐个字符进行读取和解释，因此，从本质上来说，程序是一维信息。

UML 将程序表示为二维图形。由于人们一般通过视觉来获取信息，并进行识别、理解和记忆，所以这种图形表示非常适合人脑。从这一点来看，UML 确实可以说是查看无形软件的工具（图 8-1）。

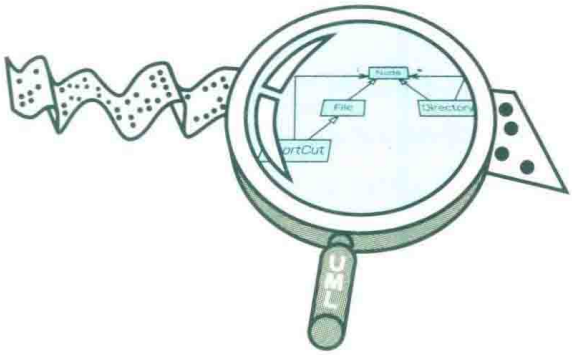


图 8–1　通过将程序表示为图形，人们就可以“查看”

在 UML 定义的图形中，表示程序结构和动作的具有代表性的图形有类图、时序图和通信图。下面我们就来介绍一下这三种图形。

8.5 类图表示 OOP 程序的结构

类图表示以类为基本单位的 OOP 程序的结构。

由于 OOP 之前的程序结构以子程序（函数）为基本单位，所以能够使用如图 8-2 所示的结构化图形来表示。

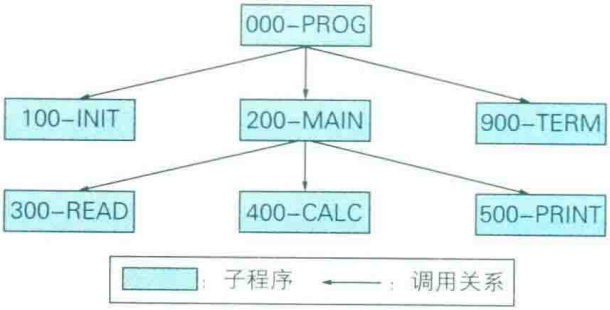


图 8-2　结构化图形示例

但是 OOP 中的一些结构之前并不存在，比如使用类定义的实例变量会引用其他类的实例、通过继承直接借用其他类的定义信息等。因此，结构化图形无法准确表示 OOP 程序的结构。而为了用图形来表示 OOP 特有的功能，类图便应运而生。

图 8-3 是一个简单的类图示例，展示了操作文件系统的程序的一部分。在图 8-3 中，长方形表示类，连接长方形的线表示类之间的关系。实例的引用和继承等关系通过箭头形状来区分。如果大家理解了类图规则，就可以从该图中读出以下内容。

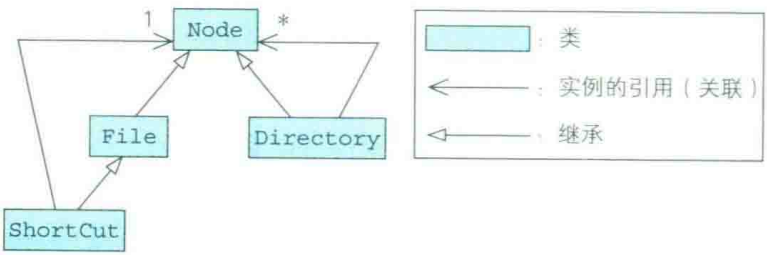


图 8-3　类图示例

- File、Directory 两个类是 Node 类的子类
- ShortCut 类是 File 的子类
- ShortCut 类的实例持有一个 Node 类（或者其子类）的实例
- Directory 类的实例可以持有多个 Node 类（或者其子类）的实例

图 8-3 的类图对应的程序源代码如图 8-4 所示。

```
// Node类
public abstract class Node {
  private String name;
  protected Node(String name) {
    this.name = name;
  }
  public String getName() {
    return this.name;
  }
  public void setName(String name) {
    this.name = name;
  }
}
// File类
public class File extends Node {
  public File(String name) {
    super(name);
  }
}
// ShortCut类
public class ShortCut extends File {
  private Node linkedNode;
  public ShortCut(Node node, String name) {
    super(name);
    this.linkedNode = node;
  }
  public ShortCut(Node node) {
    super("Short Cut to " + node.getName());
  }
  public Node getLinkedNode() {
    return this.linkedNode;
  }
}
// Directory类
public class Directory extends Node {
  private java.util.List children;
  public Directory(String name) {
    super(name);
    this.children = new java.util.ArrayList();
  }
  public void add(Node node) {
    this.children.add(node);
  }
  public java.util.Iterator getChildren() {
    return this.children.iterator();
  }
  public java.util.Iterator getDescendants() {
~下略~
```

图 8-4　OOP 的源代码示例

我们来比较一下图 8-3 和图 8-4。

首先，信息量的差别很明显。图 8-4 中编写了所有的命令，所以可以编译，也可以嵌入到一部分应用程序中运行。另外，图 8-4 中记述了图 8-3 的类图中表示的类名以及它们之间的关系。

但是，在对整体的把握上，信息量少的类图更占优势。反之，如果从图 8-4 来读取图 8-3 表示的简洁的内容，并在头脑中进行组织，太多信息量反而是一种灾难。

另外，使用二维图形表示信息还有一个好处，就是方便人们记忆。由于人的大脑更容易记住图形，所以使用像图 8-3 那样的图形来表示，人们就很容易结合图形中的位置关系来记忆相关内容，比如“左下方是 ShortCut 类”“File 和 Directory 两个类的上面有一个超类”，等等。

UML 的效果正是如此。通过将无形的软件表示为二维图形，可以很好地帮助人们对整体进行理解和掌握。

记忆力好的读者可能会发现图 8-3 使用了第 6 章介绍的 Composite 模式。有一个词语叫“模式识别”，使用图形来表示，人们也容易注意到设计模式。像这样，UML 还担负着促进思想重用的作用。

8.6 使用时序图和通信图表示动作

接下来，我们介绍一下时序图和通信图[1]。

前面介绍的类图表示的是源代码信息，而这两种图形则表示程序运行时的动作。也可以说，类图表示静态信息，时序图和通信图表示动态信息。

在传统编程语言中，表示程序运行结构的图形并不是特别需要。这是因为结构化语言中以子程序为单位编写并运行程序，使用结构化图形和流程图基本上就可以表示静态信息和动态信息。

而使用 OOP 编写的程序在运行时会从类创建实例进行动作。正如第 4 章中介绍的那样，OOP 程序可以从一个类创建很多个实例。另外，类图表

① UML 1 中称为协作图。

示的是类之间的关系，并不表示类中定义的方法的调用关系。像这样，由于仅通过类图无法表示程序运行时的动作，所以 UML 中提供了这两种图形。

我们先来介绍一下时序图（图 8-5）。

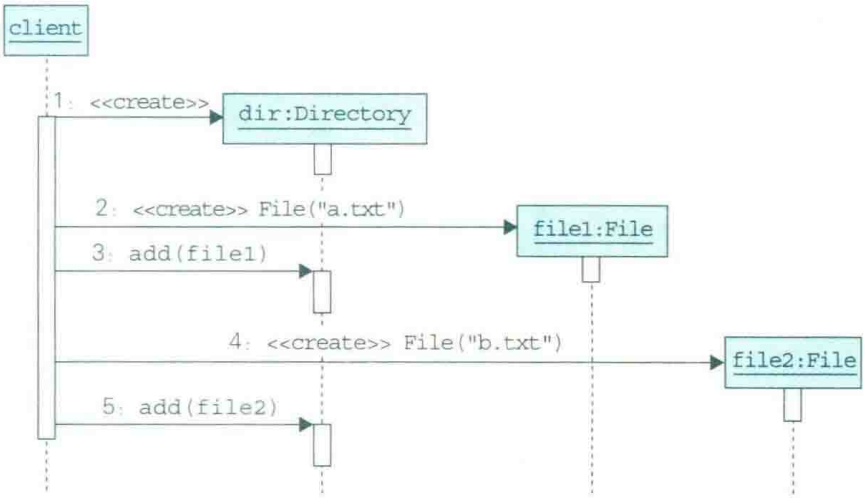


图 8-5　时序图将实例之间的相互作用表示为时间序列

时序图中的“时序”（sequence）有“连续”“顺序”的含义，因为是将方法调用表示为时间序列，所以才如此命名。由于 OOP 中的方法会指定对象实例来调用，所以时序图表示的是实例之间的相互作用。

在图 8-5 中，纵轴表示时间，长方形中是实例的名称。横向箭头表示方法调用，箭头上面是调用的方法名称和参数。之所以在箭头上面写上方法名，是因为在 OOP 中，一个类中可以定义多个方法，如果仅用线连接实例，就无法判断调用的是哪一个方法。

下面我们来看表示程序动作的另一种图形——通信图。

通信图表示的信息与时序图基本上是一样的，区别在于通信图的表示方法以实例的关系为中心。也可以认为这种图表示第 5 章中介绍的实例在内存中的配置（图 8-6）。

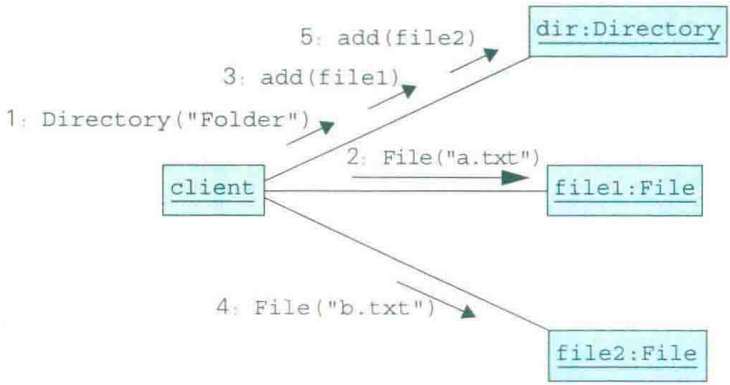


图 8–6　通信图表示实例之间的相互作用

实际的程序在计算机中以闪电般的速度运行，而使用时序图和通信图的二维图形来表示，我们就可以一目了然地“查看”程序运行时的动作。

这些图还可以被灵活应用在其他方面，比如作为编写程序逻辑之前的设计图、用于展示的资料，或者完成的程序的说明资料。

到这里为止，我们介绍了表示 OOP 程序的三种图形，下面来简单地总结一下这三种图形的特征。

< 类图 >

表示类的定义信息和类之间的关系。

< 时序图 >

将运行时的实例之间的方法调用表示为时间序列。

< 通信图 >

将运行时的实例之间的方法调用以实例关系为中心进行表示。

8.7　UML 的使用方法之二：表示归纳整理法的成果

下面我们换一个话题，来介绍一下作为归纳整理法的成果使用的 UML。

正如第 7 章中介绍的那样，面向对象在被用于上流工程后，化为了表示集合论和职责分配的归纳整理法。因此，UML 也可以用来表示现实世界和计算机系统中管理的信息的结构，以及进行了职责分配的人或组织协作完成整个工作的情况。反之，也可以说正是因为有了 UML 的图形表示，作为归纳整理法的面向对象技术才得以广泛使用。

接下来要介绍的还是类图、时序图和通信图。需要注意的是，在表示程序结构的情况下和在表示集合论、职责分配的情况下，即使这些图的绘制方法相同，含义也大不一样。

8.8 使用类图表示根据集合论进行整理的结果

UML 的类图可以作为根据集合论的思想对事物进行分类整理的成果使用。这里举一个简单的例子，将书籍作为全集，将中文书籍、翻译书籍和外文书籍作为子集[①]。

大家首先想到的集合论的图形表示是图 8-7 所示的包含图吧。

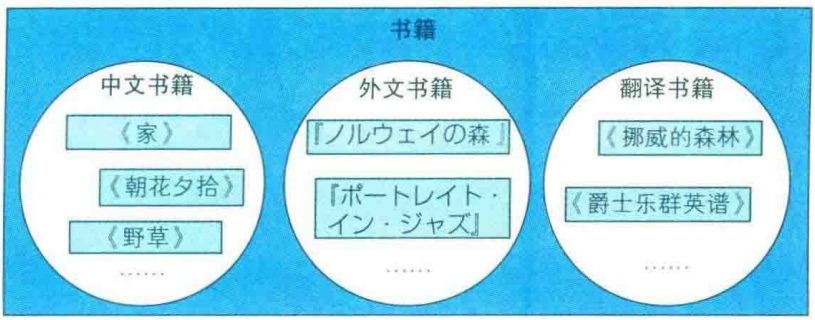


图 8-7 使用包含图的集合图形表示

用 UML 类图来表示图 8-7，如图 8-8 所示。类图中将集合表示为类，使用继承来表示全集和子集的关系。请大家看一下图 8-8 中表示“书

① 一般来说，翻译书籍包含在中文书籍中，这里为了方便，只将用中文编写的非翻译书籍作为中文书籍。

籍”“外文书籍”“翻译书籍”类的长方形的下半部分。这在 UML 中称为属性，是表示该类的性质的信息（在表示程序结构的情况下，该属性栏中写的是实例变量）。

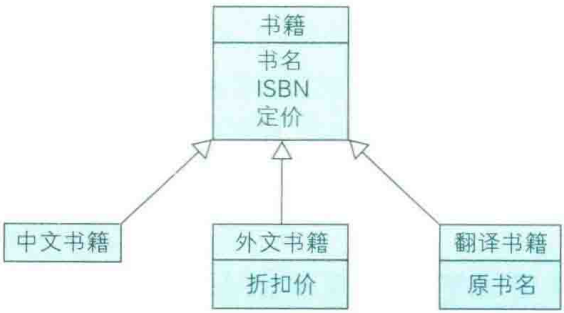


图 8-8　基于 UML 类图的集合的图形表示

在该示例中，“书籍”中定义了“书名”“ISBN”“定价”等属性。“书籍”是“中文书籍”“外文书籍”“翻译书籍”的全集，因此，这三个属性也都会被定义在书籍的所有子类中（通过声明 OOP 的继承结构，超类的定义信息会默认定义到子类中）。

这里，只有“外文书籍”中定义了“折扣价”属性，表示只有外文书籍可以打折销售，“翻译书籍”中除了“书名”之外，还有“原书名”属性。

文氏图中只列出了全集和子集的关系，以及集合中包含的元素，而使用 UML 的类图，除了全集和子集的关系之外，还能够简洁地表示集合中包含的元素的性质。

另外，文氏图中还列出了具体的书籍示例，这在 UML 中则使用**对象图**来表示。作为参考，我们也来看一下对象图的示例（图 8-9）。

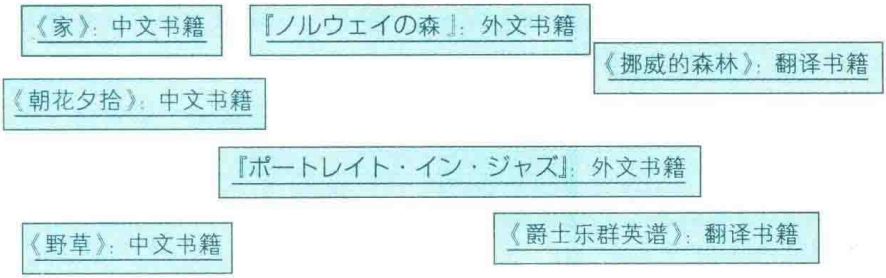


图 8–9　对象图示例

UML 的类图不仅表示集合，还可以表示集合中包含的元素之间的关系。

我们以书籍和作者之间的关系为例进行说明。使用文氏图进行表示的情况如图 8-10 所示。在翻译书籍中，书籍和作者之间存在原作者和译者两种关系，这里使用不同的线来表示[1]。

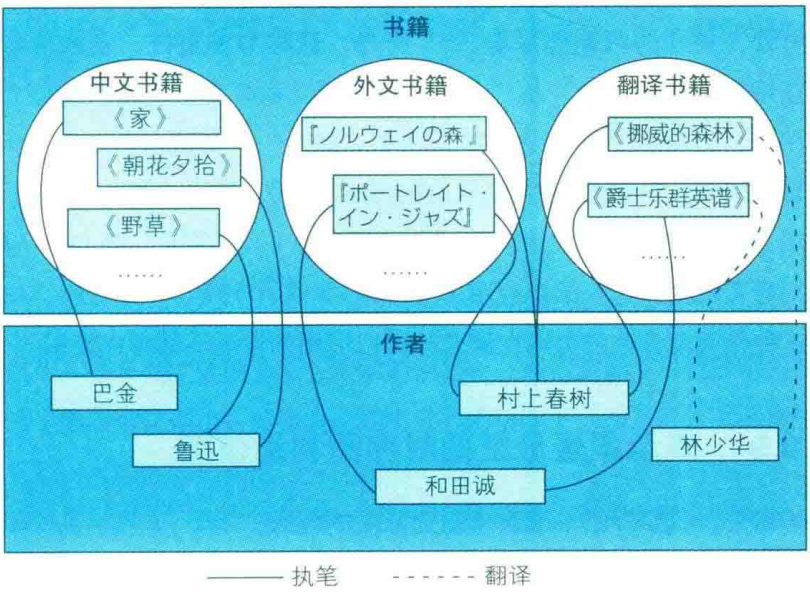


图 8–10　书籍和作者的关系（文氏图）

① 一般来说，当提到作者时，并不包含译者，但这里为了方便，使用“译者类”来包含译者。

而如果用 UML 来表示，则如图 8-11 所示。

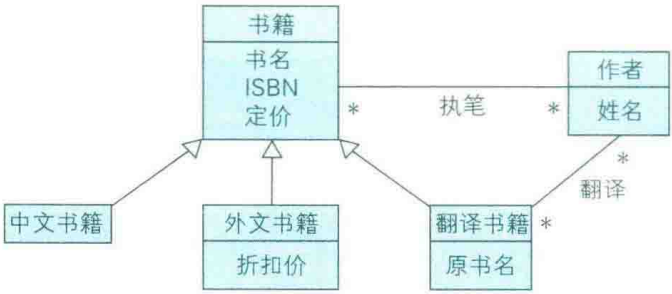


图 8-11　类和关联的表示

UML 中将集合的元素之间的关系称为**关联**[1]。这里在“作者”和“书籍”之间定义了“执笔”的关联，在“作者”和“翻译书籍”之间定义了“翻译”的关联。这样就显式地表示出所有书籍中都有作者，但只有翻译书籍中有译者。

另外，关联的两端都写着“*”符号，这称为**多重性**，表示某个元素可以连接多个元素。“*”是“多个”的意思，如果个数确定，则写上该数值。在图 8-11 中，一个作者会编写多本书籍，一本书籍也会由多个作者共同编写，因此，两端都写着表示“多个”的“*”（翻译书籍也是如此）。通过类图，我们就可以显式地表示这种多重连接（该“关联”将 OOP 程序运行时持有实例指针这一结构应用于集合论）。

这种作为集合论的类图在集合论可以适用的情况下都可以使用。

8.9　表示职责分配的时序图和通信图

接下来，我们介绍一下表示职责分配的图形。

前面介绍了表示程序动作的时序图和通信图，在上流工程中，这些图可以用来表示拥有固定职责的多个人或组织协作完成整个工作的情形。

这种职责分配的例子有很多，比如在医院里，医生、护士和药剂师等

① 准确来说，实例间的关系被称为“连接”，连接的集合被称为“关联”。