1.11 本书的构成

本书大致分为两部分来讲解面向对象，前半部分介绍编程技术，后半部分介绍从编程技术派生出来的应用技术（图 1-5）。

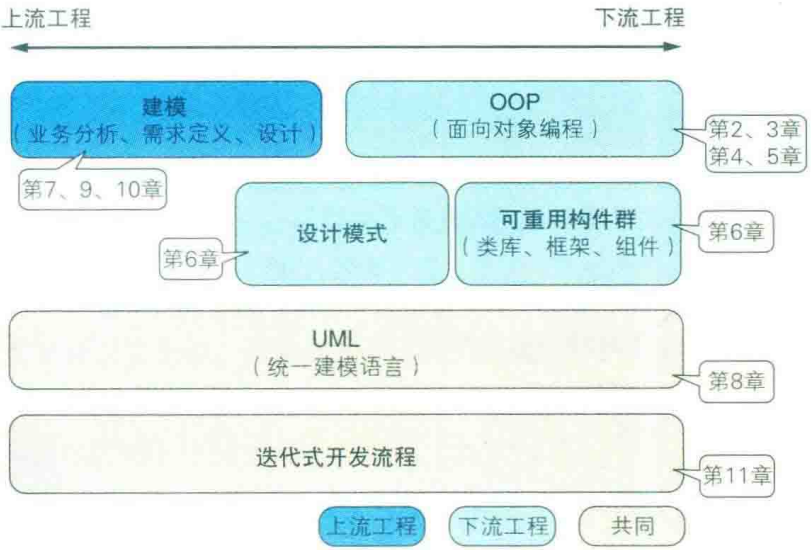

图 1-5 面向对象的全貌和本书的构成

前半部分包括第 2 章到第 6 章的内容，这一部分将为大家介绍编程技术的相关知识。首先在第 2 章指出现实世界和 OOP 结构是似是而非的，接着，在回顾了 OOP 之前的编程语言的历史（第 3 章）之后，再介绍 OOP 的基本结构——类、多态和继承（第 4 章）。然后介绍 OOP 运行时的机制（第 5 章），以及 OOP 的发展所带来的可重用构件群和设计模式这两种可重用技术（第 6 章）。

后半部分包括第 7 章到第 11 章的内容，这一部分将为大家介绍面向对象的应用技术。我们将首先介绍 OOP 结构在应用于上流工程时化为表示集合论和职责分配的归纳整理法的现象（第 7 章）。接着将介绍用图形表示软

件功能的 UML（第 8 章）、支持业务分析和需求定义的建模（第 9 章）、面向对象设计（第 10 章），以及与面向对象的普及共同发展的迭代式开发流程（第 11 章）。

之所以这样安排章节，是因为考虑到了逐个理解面向对象所涉及的各项技术时的步骤，而这也基本上符合面向对象演化的过程。

引言到此为止，下面就让我们一起进入具体内容的学习吧。

虽然笔者想在最开始的部分介绍面向对象的核心技术——编程，但为了能让大家正确理解面向对象，还是非常有必要提醒各位读者不要混淆面向对象的结构和现实世界的。

因此，笔者将从重新考虑以往常见的讲解——“面向对象是直接将现实世界表示为软件的技术”开始。接下来就让我们一起进入第 2 章吧。

本章的关键词

面向对象的三大要素、现实世界

第2章 似是而非：面向对象与现实世界

热身问答

在阅读正文之前，请挑战一下下面的问题来热热身吧。

问题

下列哪一项是“NODM”的正确解释？

A. 20 世纪 60 年代设计出来的 Simula 67 之前的编程语言的名称

B. 活跃于 20 世纪 70 年代的超级程序员的昵称

C. 20 世纪 80 年代日本作为国家工程推进的人工智能研究工程的简称

D. 20 世纪 90 年代在 IT 领域流行的表示企业系统发展方向的词语

D. 20 世纪 90 年代在 IT 领域流行的表示企业系统发展方向的词语

解 析

NODM 由网络（Network）、开源（Open source）、小型化（Down-sizing）和多媒体（MultiMedia）这 4 个单词的首字母组成[1]，该词在日本 20 世纪 90 年代前半期的 IT 领域广为流行。不过，随着互联网的普及，NODM 所代指的技术都逐渐成了通用技术，因此该词自然也就不再被使用了。

这种表示业界趋势的词语通常被称为“潮词”（buzzword）。20 世纪 90 年代后半期到 21 世纪初，“面向对象”也作为一个潮词被经常使用，其含义是“能比以往更大幅度地提高生产率和可维护性的技术”或者“直接将现实世界的情形表示为软件的创新技术”。

另外，20 世纪 80 年代日本作为国家工程推进的人工智能研究工程的名称是“新一代计算机技术研究所”（Institute for New Generation Computer Technology），简称为“ICOT”。据笔者所知，并未听过有名为 NODM 的编程语言或著名程序员。

① 也有说“O”指面向对象（Object oriented），“M”指多厂商（Multivendor）。

本章重点

本章将重新考虑之前在介绍面向对象时常用的说法——“面向对象是直接将现实世界表示为软件的技术”。从结论上来讲，虽然面向对象编程的结构和现实世界的情形有一些相似点，但两者在本质上存在很大差别。

正如笔者在第 1 章中介绍的那样，面向对象是用来轻松地进行软件开发的综合技术，其核心是编程技术。我们将在第 3 章到第 6 章具体了解该编程技术，而为了防止造成混乱，在此我们先要准确理解为什么面向对象和现实世界是似是而非的。

2.1 如果只理解概念，就容易混乱

面向对象经常被解释为“直接将现实世界表示为软件的技术”，即使不像上面这样明确断言，但就像笔者在第 1 章中提到的“比喻滥用”那样，许多面向对象的介绍也都很容易引起误解。

如果大家对这种介绍信以为真，就可能会认为计算机中是像营业员对象和顾客对象交易商品对象那样编写软件的。实际上，不管在什么情况下，这样编写出来的软件基本上都无法正常运行。站在面向对象直接表示现实世界的角度，是无法很好地解释设计模式和类库等从编程派生出来的技术的。如果将所有内容都对应到现实世界来掌握概念，势必就会感觉面向对象难以理解，十分混乱。

这种混乱是影响正确理解面向对象的最大障碍。因此，在介绍各种技术之前，本章将首先说明面向对象和现实世界实际上是似是而非的。

2.2 对照现实世界介绍面向对象

接下来，我们首先像以往那样对照着现实世界来解释面向对象编程的三大要素——类（封装）、多态和继承。

请大家在阅读的同时考虑一下有没有什么奇怪的地方。之后我们会介绍面向对象编程的结构与现实世界有何不同，在此先让我们一起来思考一下。

另外，讲解中会用到 Java 示例代码，但由于本书的目的并不是讲解 Java 的语言规范，所以并不会对其进行详细介绍。不了解 Java 的读者可以试着根据注释来理解代码的内容。

2.3　类指类型，实例指具体的物

我们先从介绍类开始吧。

类是面向对象的最基本的结构，与其对应的概念是**实例**[1]，大家最好同时记住这两个概念。类的英文是 class，含义是“种类”“类型”等“同类物品的集合”。实例的英文是 instance，含义是“具体的物”。类指类型，实例则指具体的物，二者的关系就相当于数学集合论中的集合和元素一样。

这样的关系在现实世界中也很常见。

狗：斑点狗、柴犬、牧羊犬……
国家：中国、日本、韩国、美国、英国……
歌手：迈克尔·杰克逊、矢泽永吉……

接下来，我们再来介绍一下编程。

在面向对象中，我们通过定义类来编写程序。当程序运行时，从定义的类创建实例，它们之间互相作用，从而实现软件的功能。这与在现实世界中，通过人与物的相互作用来完成工作的原理非常相似。

我们来看一个具体示例，使用 Java 编写在前面的例子中列举的狗类。我们首先对狗类进行定义（代码清单 2.1）。

① “实例”有时也称为“对象”，但本书中统一称为“实例”。

代码清单2.1 狗类的示例代码

```
// 狗类的定义
class Dog {
  String name; // 名称
  Dog(String name){ // 构造函数（创建实例时执行）
    this.name = name;
  }
  String cry(){ // 被命令时发出“汪”的叫声
    return "汪";
  }
}
```

使用该狗类可以创建出两只狗，并能让其发出“汪”的叫声（代码清单 2.2）。

代码清单2.2 让两只狗叫的示例代码

```
// 创建两只狗
Dog pochi = new Dog("斑点狗");
Dog taro = new Dog("柴犬");
// 让斑点狗和柴犬叫，并输出结果
System.out.println(pochi.cry()); } (1)
System.out.println(taro.cry());
```

代码清单 2.2 中的 (1) 处编写的 cry() 称为**方法**。调用该方法与给对方发消息分配工作的情形相似，所以称为**消息传递**（message passing）。

执行代码清单 2.2，系统控制台上会显示两次“汪”[1]。

这就像跟现实世界的狗说“抬爪子！”一样（图 2-1）。

① 实际上，为了编译并执行代码清单 2.2 的代码，我们需要在“主人”类中进行记述，这里省略了此内容。

图 2-1　使用比喻来介绍消息传递

2.4　多态让消息的发送方法通用

第二个要素是多态。

也许大家对**多态**（polymorphism）一词不太熟悉，其英文含义为“变为各种形式”，通常翻译为“多态”“多相”等。

如果用一句话来表示该结构，就是“让向相似的类发送消息的方法通用的结构”，也可以说是“发送消息时不关心对方具体是哪一个类的实例的结构”。

将该结构比作现实世界，可以做出如下说明：在前文的示例中，当向狗发送 cry 的消息时，会得到“汪”的响应；如果接收该消息的是婴儿，则会“哇”地哭；如果是乌鸦，则会发出“呱”的叫声（图 2-2）。

图 2-2　使用比喻来介绍多态

让我们来试着编写一下程序。首先定义全体共用的 Animal（动物）类（代码清单 2.3）。

代码清单2.3 动物类

```
class Animal {  //   动物类
  abstract String cry();  // 在此不定义具体的叫声
}
```

由于动物分为很多种，所以我们无法定义具体的叫声。因此，这里只定义接收 cry 消息，而不定义具体的动作。接下来，我们定义婴儿、狗和乌鸦等具体的动物作为各个类（代码清单 2.4）。

代码清单2.4 定义婴儿、狗和乌鸦类

```
class Baby extends Animal {  // 婴儿类（继承动物）
  String cry(){
    return "哇";  // "哇" 地哭
  }
}
class Dog extends Animal {  // 狗类（继承动物）
  String cry(){
    return "汪";  // "汪" 地叫
  }
}
class Crow extends Animal {  // 乌鸦类（继承动物）
  String cry(){
    return "呱";  // "呱" 地叫
  }
}
```

在代码清单 2.4 中声明类名的地方，extends Animal 是继承的声明，表示该类是一种动物（关于继承，我们将在下一节详细介绍）。

这样，我们就完成了多态的准备工作。

多态是一种方便消息发送者的手法。在上面的例子中，不管对方是谁，

指示者都可以发出同一个指示——cry。

下面，我们来定义教练（Trainer）类，不管对方是谁，都向其发送cry的指示（代码清单2.5）。

代码清单2.5 教练类

```
class Trainer {  // 定义教练类
  void execute(Animal animal){ // 参数是动物
    // 让对方哭（叫），并输出结果
    System.out.println(animal.cry());
  }
}
```

仅此而已，非常简单。

这里的重点在于，execute方法的参数是Animal（动物）类的实例。这样一来，不管对方是婴儿还是狗、乌鸦，都没有关系。不仅如此，即使对方是相扑冠军、职业摔跤手，或者是狮子、鲸鱼、蛇，只要对方是动物，就都可以应对。这就是可以应对所有动物的终极教练。

2.5 继承对共同点和不同点进行系统的分类和整理

最后一个要素是继承。

用一句话来表示**继承**，就是“系统地整理物的种类的共同点和不同点的结构”。在面向对象中，“物的种类”就是类。因此，换一种表达方式，也可以说继承是“整理相似事物的类的共同点和不同点的结构”。

我们在前面介绍过，类和实例相当于集合和元素，而继承则相当于全集和子集。

现实世界中也可以看到很多这样的关系，如图2-3中的动物分类。动物分为哺乳类、鸟类、昆虫类、爬虫类、两栖类和鱼类等，这些类又可以细分为很多种。

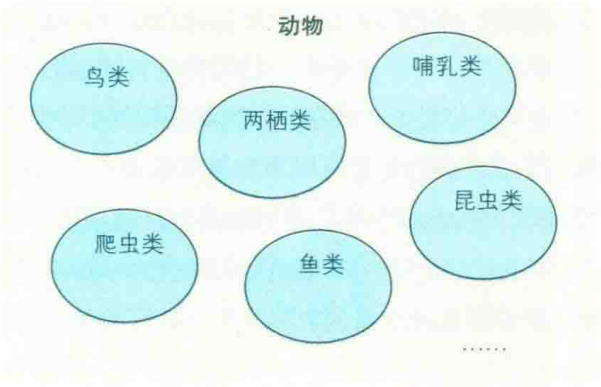

图 2-3　动物分类

在面向对象中，全集被称为**超类**，子集被称为**子类**（这与数学集合论中的超集和子集的称呼相同）。

除了动物的分类体系之外，这种继承关系还可以应用于现实世界中的各种情况，比如将医生分为内科医生、外科医生、眼科医生和牙科医生，将公司职员分为营业人员、技术人员和办公人员等（图 2-4）。

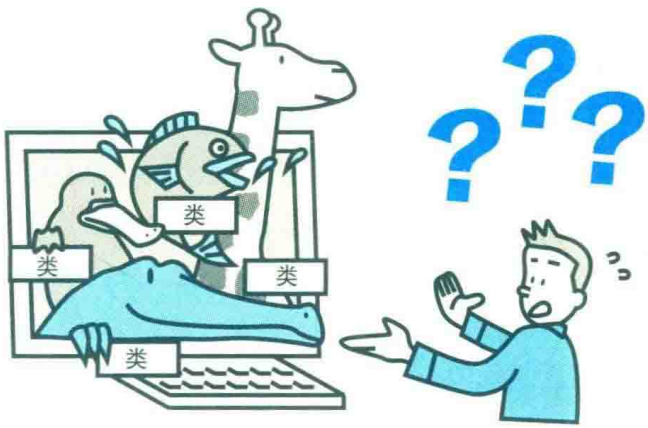

图 2-4　使用比喻来介绍继承

我们接着来介绍一下编程。

Java 等面向对象编程语言中可以直接描述继承。具体来说，在超类中

定义全集共同的性质，在子类中定义子集特有的性质，这样我们就可以不用重复定义非常相似的、只存在略微不同的类的共同点和不同点了。像这样将共同的性质定义在超类中，再创建子类就很轻松了。

我们来试着写一下动物分类的程序。如代码清单2.6所示，这里定义了Animal（动物）、Mammal（哺乳类）和Bird（鸟类）三个类。Mammal和Bird类定义中的extends Animal是继承Animal类的声明。这样一来，Animal类中定义的性质（这里是move和cry两个方法）也会自动定义到Mammal类和Bird类中。

代码清单2.6 继承的示例代码

```
// 动物类
class Animal {
  void move(){ /* 省略逻辑处理 */ }  // 行动
  String cry(){ /* 省略逻辑处理 */ }  // 哭（叫）
}
// 哺乳类（继承动物类）
class Mammal extends Animal {
  void bear(){ /* 省略逻辑处理 */ }  // 生育
}
// 鸟类（继承动物类）
class Bird extends Animal {
  void fly(){ /* 省略逻辑处理 */ }  // 飞
}
```

也就是说，“行动”“哭（叫）”等共同的性质定义在动物类中，而特有的“生育”性质则定义在哺乳类中，“飞”的性质定义在鸟类中。

2.6 面向对象和现实世界是似是而非的

至此，我们参照现实世界，介绍了类、多态和继承等面向对象编程的三大要素。大家感觉如何？

对于陌生的术语，我们进行了抽象的说明，并通过现实世界中大家身

边的例子进行了讲解。最后，我们还展示了使用这些例子编写的示例代码，并对相关的语言规范进行了介绍。

以上就是讲解面向对象编程结构的常见流程。

这样讲解的优点是易于直观理解，能在脑海中留下印象。不过，如果只是这样讲解，那么听众在实际编写程序时就会难以理解怎样编码更方便。另外，如果说“面向对象是以对象为中心的开发方法，与之前的思想完全不同”，或者说“需要经过多年学习才可以熟练掌握面向对象”等，那么第一次听到这些话的人肯定会感觉很迷茫吧。

实际上，面向对象和现实世界是似是而非的。下面我们就来具体地看一看它们到底有何不同。

2.7 现实世界的人和物不是由类创建的

首先是类和实例。我们在前面介绍过，类指类型，而实例指具体的物，并举出了现实世界的例子，即狗是类，斑点狗和柴犬是实例。到此为止并没有什么问题，但接下来就出现问题了。

我们来看一下代码清单 2.1 和代码清单 2.2。在面向对象的世界中是先创建类，然后再创建实例的。

在现实世界中也是如此吗？

差异应该是很大的。比如狗的生育，公狗和母狗在发情期亲密接触后，母狗怀孕而产下狗崽，狗崽并不是从预先定义的“狗”类创建的[1]。

在面向对象和现实世界中，对类的定位有很大不同。在面向对象中，类是用来创建实例的结构，实例只属于一个类。而在现实世界中，首先有具体的物（实例），然后再根据观察该物的人的立场和兴趣而采用不同的基准进行分类（类）。以笔者自己为例，在公司就是“员工”，与客户打交道

① 为了介绍类和实例，使用“章鱼小丸子模具”和“章鱼小丸子”这种生产设备和生产物品的比喻会更恰当吧。

时就是“技术人员”，而回到家就是“父亲”，走在街上时就是“路人”[1]。

此外还存在其他不同之处。在面向对象的世界中，实例固定属于唯一的类，即使经过一段时间，也无法变为属于其他类。由婴儿类创建的实例无论经过多少年，也无法成长为青年、中年和老年。

而现实世界中的人和物会随着时间的流逝而成长或老化，因此其分类也会发生变化。还是以笔者为例进行说明，笔者在年轻时被分类为“少年”“年轻人”，而最近已经完全是“大叔”了[2]。

像这样，即使是类和实例这种最基本的结构，在面向对象和现实世界中也存在很大的不同。

2.8　现实世界的人和物并不只是根据消息来行动

下面我们接着来思考一下消息传递。在前面介绍多态时我们曾介绍过，如果发送“哭（叫）”的消息，婴儿就会“哇”地哭，狗会“汪”地叫，乌鸦会“呱”地叫。乍一听，大家可能觉得没什么问题。

不过，这真的是在描述现实世界吗？如果我们向身边的婴儿、饲养的狗以及公园里的乌鸦发送“哭（叫）”的消息，可能得不到任何响应。即使对方是能进行正常交流的成年人，可能也会无动于衷。

现实世界中的人是根据自己的判断来行动的，虽然有时也会听从他人的指示，但这种情况只占很小的一部分。吃饭、不吃饭、睡觉、起床、上厕所、扔垃圾、不要喝酒、安静工作……如果大家的所有行为都出于别人的命令，并且大家完全不可以违背这些命令，那么现实世界岂不是会变得非常糟糕。

另外，在面向对象的世界中，我们需要事先为所有的行动都准备好方法，并通过消息传递来调用方法，这样事物才会开始运作。在这个世界中，

① 像这样，某个实例（无继承关系）可以属于多个类的现象称为“多重分类”。在面向对象语言中基本不支持这种多重分类。

② 像这样，某个实例可以变为属于其他类的现象称为“动态分类”。在面向对象语言中基本不支持这种动态分类。

“不违背指示”是最基本的[①]。

这样看来，“消息传递结构能够直接表示现实世界”的说法还是稍微有些夸张了吧。

2.9 明确定义为编程结构

实际上，类、多态和继承应该被明确定义为能提高软件的可维护性和可重用性的结构。类用于将变量和子程序汇总在一起，创建独立性高的构件；多态和继承用于消除重复代码，创建通用性强的构件。另外，实例能在运行时将实例变量在堆区展开，可以轻松地确保类的逻辑。通过熟练掌握这些结构，我们就可以创建出比以往更易于维护的、可重用的软件。

为了避免误解，这里必须明确一点，那就是仿照现实世界进行的讲解只是一种比喻，大家应该按照编程结构来理解。另外，关于这些结构在编程上的功能，我们将在第 4 章中详细讲述。关于运行时的结构，我们将在第 5 章中具体介绍。

2.10 软件并不会直接表示现实世界

根据到目前为止的讲解，有的读者可能会想：“我现在明白面向对象和现实世界不一样了，但它们又的确非常相似，难道不能采用适当的形式来表示现实世界吗？”

实际上，这里还存在一个应该讨论的内容，那就是系统化的范围，即软件能在多大程度上涵盖现实世界的工作。

从结论上讲，几乎所有的软件都只能涵盖人类工作的一部分。计算机擅长做**记忆工作和固定的工作**，能记录并在瞬间提取出大量的信息，能正确执行工资计算或者利息计算等具有固定步骤的处理。由于软件仅涵盖这种性质的工作，所以即使使用了计算机，许多工作也还是需要由人来完成。

① 面向对象仅拥有这种被动的特性，而代理技术却可以打破这种限制，通过主动进行动作来构建软件。目前“面向代理”的相关研究正在进行中，不过至今尚未出现具体实现该概念的编程语言。