第20章

# 操作型模式介绍

在编写一个 Java 方法时，你完成的是整个工作中级别高于单行代码的一个基本单元。这些方法牵涉到整体设计、架构以及测试计划。编写方法是面向对象编程的中心环节。但反过来说，尽管方法是软件设计的核心，我们还是难以解释方法到底是什么，方法又是如何工作的。追本溯源，还是因为许多开发人员常常混淆了方法与操作的含义。进一步讲，算法和多态的概念比方法更加抽象，但最终它们还是通过方法来实现。

对术语“算法”、“多态”、“方法”与“操作”具有清晰的认识，有助于你理解多种设计模式。尤其是状态模式、策略模式和解释器模式，都是通过实现多个类中的方法来完成操作的。但是，只有当我们对方法和操作的理解达成共识时，这样的表达才有用。

## 操作和方法

关于类，有很多相关的术语。我们尤其要注意甄别“操作”和“方法”。UML 定义了操作和方法之间的区别。如下所示：

- 操作是一个服务的规格说明，它可以被类的实例调用。
- 方法是操作的实现。

注意，这意味着操作是方法的一种抽象。

操作表示类做了什么，还表示服务提供的接口。不同的类可能用不同的方法实现同样的操作。例如，很多类用自己的方式实现 `toString()`操作。每个类都通过方法来实现操作。方法就是（包含）让操作能够工作的代码。

搞清楚方法和操作的定义，有助于理清许多设计模式的结构。设计模式来源于类和方法，因此，能够在诸多设计模式中看到操作的身影，也就不足为奇了。例如在合成模式中，叶子节点与合成节点皆实现了共同的操作。在代理模式中，代理对象和目标对象具有相同的操作，因而，代理对象可以管理对目标对象的访问。

**挑战 20.1**

使用术语“操作”和“方法”来解释职责链模式是如何实现操作的。

答案参见第 345 页

在 Java 语言中，方法的声明包括方法头和方法体两部分。方法体是一系列的指令，可以通过调用方法的签名来执行。方法头包括方法的返回类型以及方法签名，还可能包含访问修饰符与异常语句。方法头的格式如下：

*modifiers return-type signature throws-clause*

**挑战 20.2**

尽可能多地写出 9 种 Java 访问修饰符。

答案参见第 345 页

## 签名

表面看来，操作的意义和签名相似。两个词都指的是接口的方法。当实现一个方法后，就可以使用方法的签名来调用它。*Java™ Language Specification*（由 Arnold 和 Gosling 在 1998 年

编写）一书的 8.4.2 节中说：

*方法的签名包括方法名、传入参数的数量以及类型。*

注意，方法的签名不包括返回类型，尽管当一个方法重载了另一个方法的声明时，如果它们的返回类型不同时，会发生编译错误。

**挑战 20.3**

即使 `Bitmap.clone()`通常返回 `Bitmap` 类的实例，但它返回的却是 `Object` 类型。请问当其返回类型被声明为 `Bitmap` 时，是否依然能通过编译？

答案参见第 346 页

方法签名代表了客户调用的方法。操作是可以被请求的服务规格。术语“签名”和“操作”的意思很接近，但是它们并不是一个词。这两个术语的区别主要在于它们的不同应用场景。当涉及不同类中的方法拥有相同的接口时，使用术语“操作”。当涉及如何将 Java 方法调用匹配到接收对象的方法上时，使用术语“签名”。签名依赖于方法的名字和参数，但是不依赖于方法的返回类型。

## 异常

在 *Illness as Metaphor* 一书中，Susan Sontag 提到：“每个人生来就具有两种身份，一谓健康，一谓疾病。”这个隐喻不仅适合于人，也适合于方法：正常情况下，方法会正常返回。但是，方法也可能抛出异常，或者在其内部调用其他方法时抛出异常。当程序正常返回时，会返回上次调用结束的入口点。出现异常时，则应用其他规则。

出现异常时，Java 运行时必须找到 `try/catch` 语句来匹配异常。`try/catch` 语句可能会出现在抛出异常的方法中，也可能出现在调用当前方法的方法中，还可能出现在调用之前方法的方法中。如果在方法调用的堆栈中没有找到 `try/catch` 语句，程序会停止并且崩溃。

任何方法都可以使用 `throw` 语句来抛出异常，例如：

```
throw new Exception("Good Luck!");
```

如果你的应用程序在调用方法时抛出了一个非预期的异常，程序可能会突然终止。为避免这类行为发生，需要从架构层面捕获和处理这类异常。或许，你会觉得声明所有可能的异常是件麻烦事儿。例如，C#不要求声明异常；C++允许异常的声明，但不要求在编译时进行检查。

**挑战 20.4**

与 Java 不同，C#不要求方法声明任何它可能抛出的异常。这种方式是否是对 Java 的一种改进？

答案参见第 346 页

## 算法和多态

算法和多态是编程中非常重要的概念，但我们却很难表达这两个术语的含义。如果你想向某人展示方法，可以直接拿代码来讲解。偶尔，某个方法也可能包含完整的算法，但算法通常都是作用于一些方法的。在 *Introduction to Algorithms*（由 Cormen、Leiserson 和 Rivest 在 1990 年编写）一书中提到：

> 算法是已经定义好的计算程序，将数据或者数据集作为输入，将产生的数据或者数据集作为输出。

算法是一个过程——一个有输入和输出的指令序列。单个的方法可能是一个算法：接收输入——参数列表——产生并且输出其返回值。然而在面向对象的程序中，很多算法需要多个方法来执行。例如第 5 章合成模式中的 isTree()算法，需要 4 个方法，如图 20.1 所示。

**挑战 20.5**

图 20.1 中描述了多少种算法、多少种操作以及多少个方法？

答案参见第 346 页

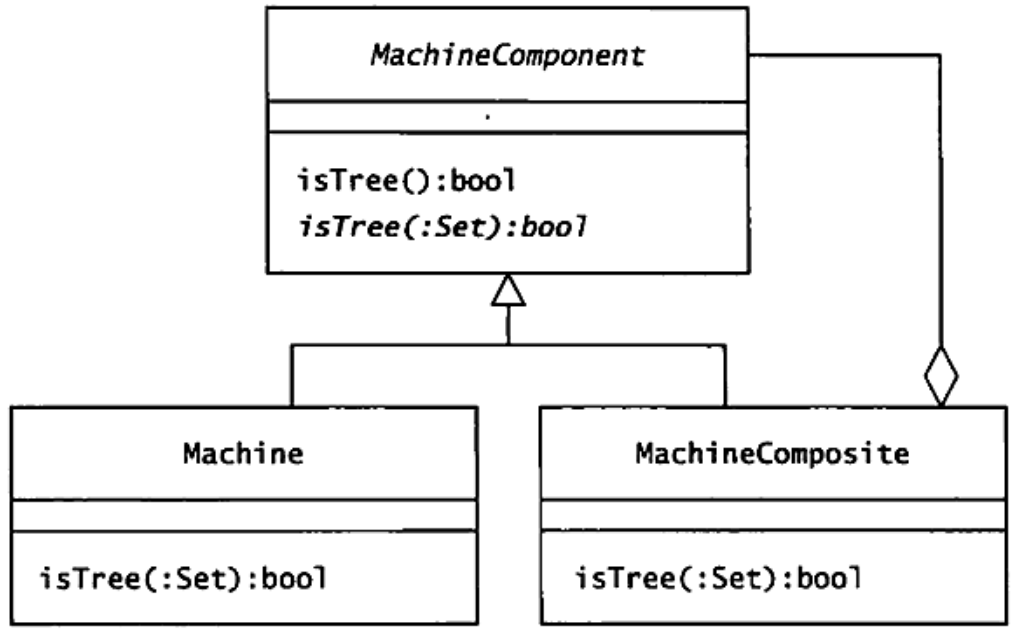


**图 20.1　isTree()中的 4 个方法相互协作，以判断一个 MachineComponent 实例是否是一棵树**

算法完成了一些事情，它可能是方法的一部分，也可能调用了很多方法。在面向对象的设计中，需要多个方法参与的算法通常依赖于多态，因为多态机制允许一个操作具有多个不同的实现。多态是一个既依赖于调用的操作，又依赖于调用接收者类型的一种方法调用的原则。例如，当执行到表达式 `m.isTree()`时，到底哪个方法会被执行呢？答案取决于它的依赖关系。如果 m 是 Machine 类的实例，Java 会调用 `Machine.isTree()`。如果 m 是 MachineComposite 类的实例，Java 会调用 `MachineComposite.isTree()`。通俗地讲，多态是指合适的对象调用合适的方法。这在某些情形下，正是设计模式使用多态的直接意图。

## 小结

尽管混淆操作、方法、签名和算法的做法普遍存在，但是，弄清楚这些术语有助于我们描述一些重要的概念。操作和签名一样，是服务的规格说明。操作意味着许多方法拥有相同的接口。签名意味着方法的查询规则。方法的定义包括它的签名——方法名和参数列表，以及访问修饰符、返回类型与方法体。方法拥有签名和操作的实现。

调用方法的常规做法是直接调用。结束方法调用的常规做法是使其返回。但是在出现未处理的异常时，任何方法都会停止执行。

算法是一个接收输入和产生输出的过程。方法接收输入，产生输出，并且包含了一个过程的方法体。因此有人将方法看做是一个算法。然而，一个算法过程可能包括多个操作和方法，或者属于其他方法的一部分。当你在讨论一个过程会产生一个结果时，最适合用算法来表达。

许多设计模式都会将一个操作分散设计在多个类中。你也可以说这些模式依赖于多态，即调用的方法取决于传入的对象类型。

## 超越常规的操作

不同类在实现同一个操作时采用不同的方式。换言之，Java 支持多态。这看似简单的想法却被多种设计模式所使用。表 20.1 列出了不同场景适用的模式

表 20.1　不同场景适用的模式

| 如果你的意图是 | 使用的模式 |
| --- | --- |
| • 在方法中实现算法，推迟对算法步骤的定义使得子类能够重新实现 | 模板方法模式 |
| • 将操作分散，使得每个类都能够表示不同的状态 | 状态模式 |
| • 封装操作，使得实现是可以互相替换的 | 策略模式 |
| • 用对象来封装方法调用 | 命令模式 |
| • 将操作分散，使得每个实现能够运用到不同类型的集合中 | 解释器模式 |

面向操作的模式适合于这类场景：设计中需要多个具有相同签名的方法。例如，模板方法模式允许子类重新实现及调整父类已经定义好的方法。