|3.3 Template Method模式中的登场角色

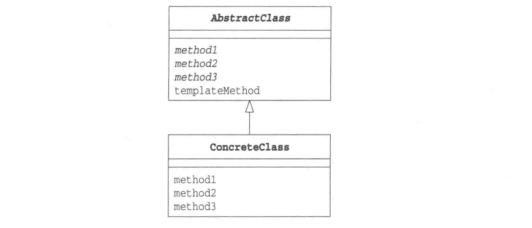
【将具体处理交给子类】

在Template Method模式中有以下登场角色。

◆AbstractClass（抽象类）AbstractClass角色不仅负责实现模板方法，还负责声明在模板方法中所使用到的抽象方法。这些抽象方法由子类ConcreteClass角色负责实现。在示例程序中，由AbstractDisplay类扮演此角色。

·ConcreteClass（具体类）该角色负责具体实现AbstractClass角色中定义的抽象方法。这里实现的方法将会在AbstractClass角色的模板方法中被调用。在示例程序中，由CharDisplay类和stringDisplay类扮演此角色。

图3-4展示了这两种Template Method模式的类图。



|3.4拓展思路的要点|可以使逻辑处理通用化

使用Template Method模式究竟能带来什么好处呢？这里，它的优点是由于在父类的模板方法中编写了算法，因此无需在每个子类中再编写算法。

例如，我们没使用Template Method模式，而是使用文本编辑器的复制和粘贴功能编写了多个ConcreteClass角色。此时，会出现ConcreteClass1、ConcreteClass2、ConcreteClass3等很多相似的类。编写完成后立即发现了Bug还好，但如果是过一段时间才发现在Concreteclass1中有Bug，该怎么办呢？这时，我们就必须将这个Bug的修改反映到所有的ConcreteClass角色中才行。

关于这一点，如果是使用Template Method模式进行编程，当我们在模板方法中发现Bug时，只需要修改模板方法即可解决问题。

父类与子类之间的协作

在Template Method模式中，父类和子类是紧密联系、共同工作的。因此，在子类中实现父类中声明的抽象方法时，必须要理解这些抽象方法被调用的时机。在看不到父类的源代码的情况下，想要编写出子类是非常困难的。

父类与子类的一致性

在示例程序中，不论是CharDisplay的实例还是stringDisplay的实例，都是先保存在AbstractDisplay类型的变量中，然后再来调用display方法的。

使用父类类型的变量保存子类实例的优点是，即使没有用instanceof等指定子类的种类，程序也能正常工作。

无论在父类类型的变量中保存哪个子类的实例，程序都可以正常工作，这种原则称为里氏替换原则”（The Liskov Substitution Principle，LSP）。当然，LSP并非仅限于Template Method模式，它是通用的继承原则。|3.5相关的设计模式

◆Factory Method模式（第4章）Factory Method 模式是将Template Method模式用于生成实例的一个典型例子。

Strategy模式（第10章）在Template Method模式中，可以使用继承改变程序的行为。这是因为Template Method模式在父类中定义程序行为的框架，在子类中决定具体的处理。

与此相对的是Strategy模式，它可以使用委托改变程序的行为。与Template Method模式中改变部分程序行为不同的是，Strategy模式用于替换整个算法。

3.6延伸阅读：类的层次与抽象类

|父类对子类的要求

我们在理解类的层次时，通常是站在子类的角度进行思考的。也就是说，很容易着眼于以下几点。

·在子类中可以使用父类中定义的方法

·可以通过在子类中增加方法以实现新的功能

·在子类中重写父类的方法可以改变程序的行为

现在，让我们稍微改变一下立场，站在父类的角度进行思考。在父类中，我们声明了抽象方法，而将该方法的实现交给了子类。换言之，就程序而言，声明抽象方法是希望达到以下目的。

·期待子类去实现抽象方法·要求子类去实现抽象方法

也就是说，子类具有实现在父类中所声明的抽象方法的责任。因此，这种责任被称为“子类责任”（subclass responsibility）。

|抽象类的意义

对于抽象类，我们是无法生成其实例的。在初学抽象类时，有人会有这样的疑问：“无法生成实例的类到底有什么作用呢？”在学完了Template Method模式后，大家应该能够稍微理解抽象类的意义了吧。由于在抽象方法中并没有编写具体的实现，所以我们无法知道在抽象方法中到底进行了什么样的处理。但是我们可以决定抽象方法的名字，然后通过调用使用了抽象方法的模板方法去编写处理。虽然具体的处理内容是由子类决定的，不过在抽象类阶段确定处理的流程非常重要。

|父类与子类之间的协作

父类与子类的相互协作支撑起了整个程序。虽然将更多方法的实现放在父类中会让子类变得更轻松，但是同时也降低了子类的灵活性；反之，如果父类中实现的方法过少，子类就会变得臃肿不堪，而且还会导致各子类间的代码出现重复。

在Template Method模式中，处理的流程被定义在父类中，而具体的处理则交给了子类。但是对于“如何划分处理的级别，哪些处理需要由父类完成，哪些处理需要交给子类负责”并没有定式，这些都需要由负责程序设计的开发人员来决定。

|3.7本章所学知识

在本章中，我们学习了在父类中定义处理的流程，在子类中实现具体处理内容的Template Method模式。此外，我们还分析了抽象类的意义和子类的责任。

在下一章中，我们将学习Factory Method模式——将Template Method模式用于生成实例的模式。