|7.3Builder模式中的登场角色

【组装复杂的实例】

Builder模式中有以下登场角色。

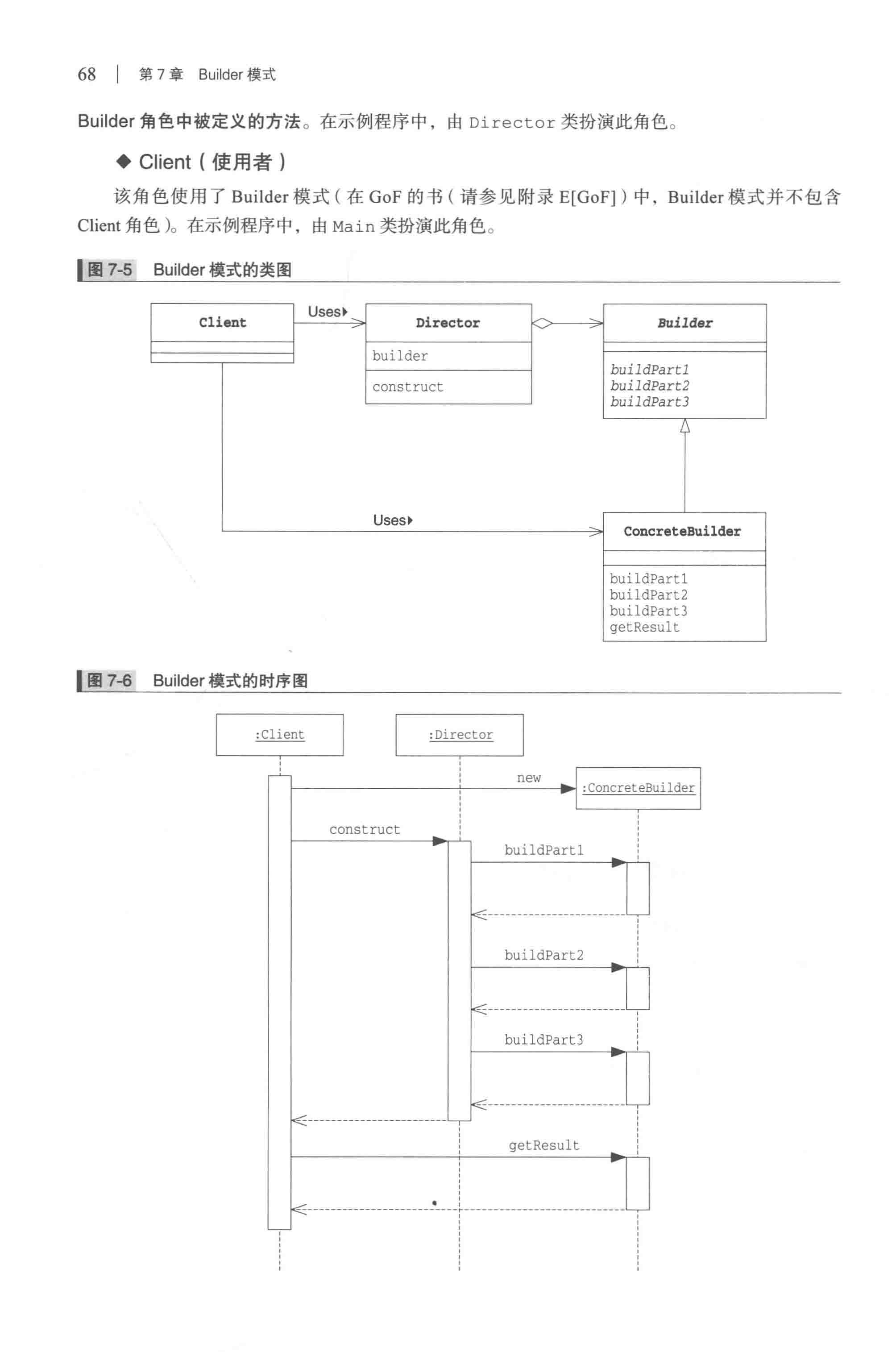
·Builder（建造者）Builder角色负责定义用于生成实例的接口（API）。Builder角色中准备了用于生成实例的方法。

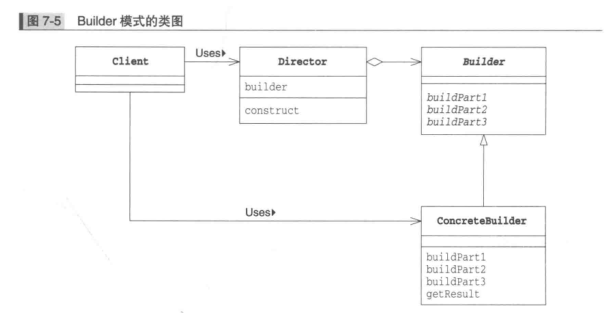
在示例程序中，由Builder类扮演此角色。

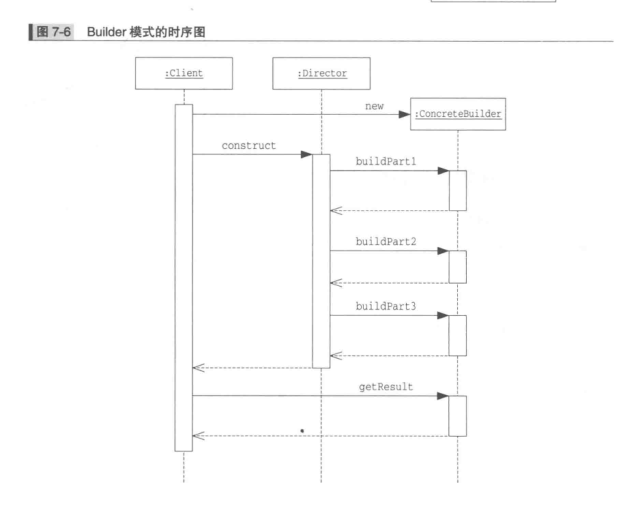
·ConcreteBuilder（具体的建造者）ConcreteBuilder角色是负责实现Builder角色的接口的类（API）。这里定义了在生成实例时实际被调用的方法。此外，在ConcreteBuilder角色中还定义了获取最终生成结果的方法。在示例程序中，由TextBuilder类和HTMLBuilder类扮演此角色。

◆Director（监工）Director角色负责使用Builder角色的接口（API）来生成实例。它并不依赖于ConcreteBuilder角色。为了确保不论ConcreteBuilder角色是如何被定义的，Director角色都能正常工作，它只调用在Builder角色中被定义的方法。在示例程序中，由Director 类扮演此角色。

·Client（使用者）该角色使用了Builder模式（在GoF的书（请参见附录E[GoF]）中，Builder模式并不包含Client角色）。在示例程序中，由Main类扮演此角色。







|7.4相关的设计模式

·Template Method模式（第3章）在Builder模式中，Director角色控制Builder角色。

在Template Method模式中，父类控制子类。①

·Composite模式（第11章）有些情况下Builder模式生成的实例构成了Composite模式。

◆Abstract Factory模式（第8章）Builder 模式和Abstract Factory模式都用于生成复杂的实例。

◆Facade模式（第15章）在Builder模式中，Director角色通过组合Builder角色中的复杂方法向外部提供可以简单生成实例的接口（API）（相当于示例程序中的construct方法）。

Facade模式中的Facade角色则是通过组合内部模块向外部提供可以简单调用的接口（API）。

|7.5拓展思路的要点

|谁知道什么

在面向对象编程中，“谁知道什么”是非常重要的。也就是说，我们需要在编程时注意哪个类可以使用哪个方法以及使用这个方法到底好不好。

请大家再回忆一下示例程序。

Main 类并不知道（没有调用）Builder类，它只是调用了Direct类的construct方法。

这样，Director类就会开始工作（Main类对此一无所知），并完成文档的编写。

另一方面，Director类知道Builder类，它调用Builder类的方法来编写文档，但是它并不知道它“真正”使用的是哪个类。也就是说它并不知道它所使用的类到底是TextBuilder类、HTMLBuilder类还是其他Builder类的子类。不过也没有必要知道，因为Director类只使用了Builder类的方法，而Builder类的子类都已经实现了那些方法。

Director类不知道自己使用的究竟是Builder类的哪个子类也好。这是因为“只有不知道子类才能替换”。不论是将TextBuilder的实例传递给Director，还是将HTMLBuilder类的实例传递给Director，它都可以正常工作，原因正是Director类不知道Builder类的具体的子类。

正是因为不知道才能够替换，正是因为可以替换，组件才具有高价值。作为设计人员，我们必须时刻关注这种“可替换性”。

①这里的控制指的是方法的调用顺序的控制。在Builder模式中，Director决定了Builder角色中方法的调用顺序，而在Template Method模式中，父类决定了子类方法的调用顺序。一—译者注70|第7章Builder模式

设计时能够决定的事情和不能决定的事情

在Builder类中，需要声明编辑文档（实现功能）所必需的所有方法。Director类中使用的方法都是Builder类提供的。因此，在Builder类中应当定义哪些方法是非常重要的。

而且，Builder类还必须能够应对将来子类可能增加的需求。在示例程序中，我们只编写了支持纯文本文档的子类和支持HTML文件的子类。但是将来可能还会希望能够编写其他形式（例如xxxx形式）的文档。那时候，到底能不能编写出支持xxxX形式的xxXXBuilder类呢？应该不需要新的方法吧？

虽然类的设计者并不是神仙，他们无法准确地预测到将来可能发生的变化。但是，我们还是有必要让设计出的类能够尽可能灵活地应对近期可能发生的变化。

|代码的阅读方法和修改方法

在编程时，虽然有时需要从零开始编写代码，但更多时候我们都是在现有代码的基础上进行增加和修改。

这时，我们需要先阅读现有代码。不过，只是阅读抽象类的代码是无法获取很多信息的（虽然可以从方法名中获得线索）。

让我们再回顾一下示例程序。即使理解了Builder抽象类，也无法理解程序整体。至少必须在阅读了Director的代码后才能理解Builder类的使用方法（Builder类的方法的调用方法）。然后再去看看TextBuilder类和HTMLBui1der类的代码，就可以明白调用Builder类的方法后具体会进行什么样的处理。

如果没有理解各个类的角色就动手增加和修改代码，在判断到底应该修改哪个类时，就会很容易出错。例如，如果修改Builder类，那么就会对Director类中调用Builder类方法的地方和Builder类的子类产生影响。或是如果不小心修改了Director类，在其内部调用了TextBuilder类的特有的方法，则会导致其失去作为可复用组件的独立性，而且当将子类替换为HTMLBuilder时，程序可能会无法正常工作。