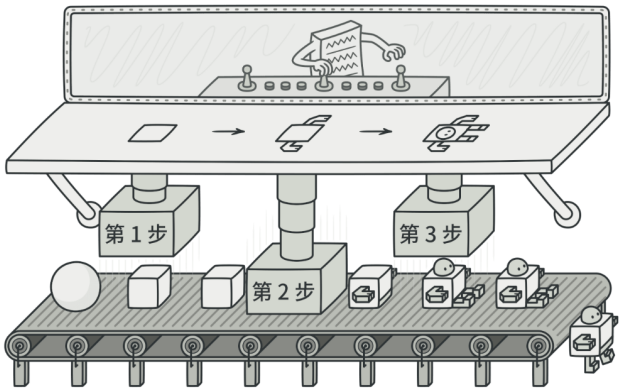
生成器模式（Builder）

生成器（Builder）

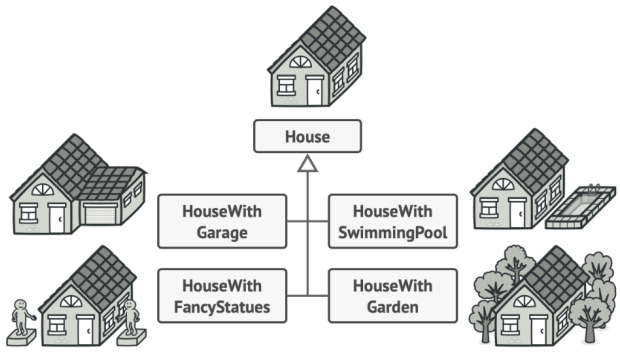
意图

生成器模式是一种创建型设计模式， 使你能够分步骤创建复杂对象。 该模式允许你使用相同的创建代码生成不同类型和形式的对象。



问题

假设有这样一个复杂对象， 在对其进行构造时需要对诸多成员变量和嵌套对象进行繁复的初始化工作。 这些初始化代码通常深藏于一个包含众多参数且让人基本看不懂的构造函数中； 甚至还有更糟糕的情况， 那就是这些代码散落在客户端代码的多个位置。

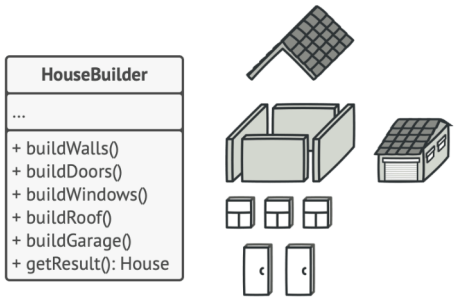


如果为每种可能的对象都创建一个子类，这可能会导致程序变得过于复杂。

> 最简单的方法是扩展 **房屋** 基类， 然后创建一系列涵盖所有参数组合的子类。 但最终任何新增的参数 （例如门廊类型） 都会让这个层次结构更加复杂。
>
> 另一种方法则无需生成子类，可以在房屋基类中创建一个包括所有可能参数的超级构造函数， 并用它来控制房屋对象。 这种方法确实可以避免生成子类， 但绝大部分的参数都没有使用，这使得**对于构造函数的调用十分不简洁**。

解决方案

生成器模式建议将对象构造代码从产品类中抽取出来， 并将其放在一个名为生成器的独立对象中。



生成器模式让你能够分步骤创建复杂对象。生成器不允许其他对象访问正在创建中的产品。

该模式会将对象构造过程划分为一组步骤， 比如 buildWalls 创建墙壁和 buildDoor 创建房门创建房门等。 每次创建对象时， 你都需要通过生成器对象执行一系列步骤。 重点在于你无需调用所有步骤， 而只需调用创建特定对象配置所需的那些步骤即可。

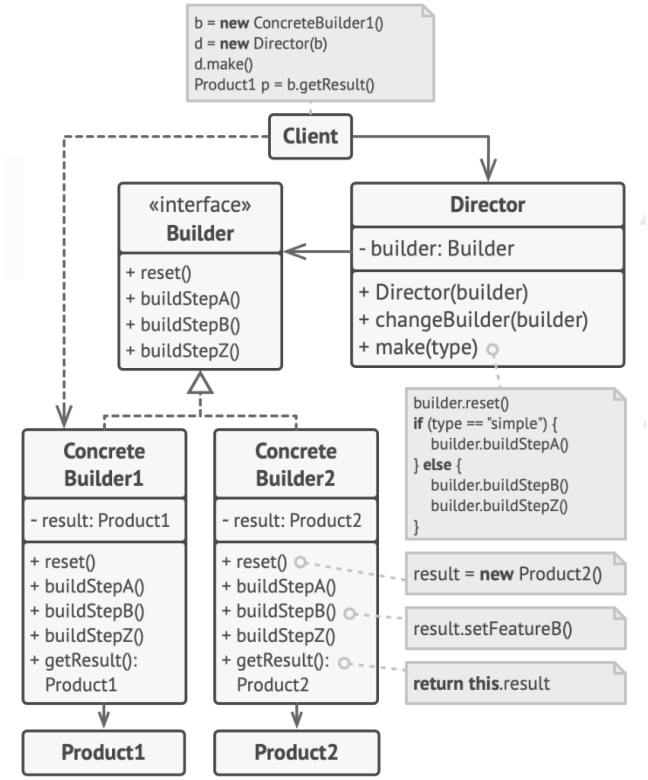
主管

可以进一步将用于创建产品的一系列生成器步骤调用抽取成为单独的主管类。 主管类可定义创建步骤的执行顺序， 而生成器则提供这些步骤的实现。

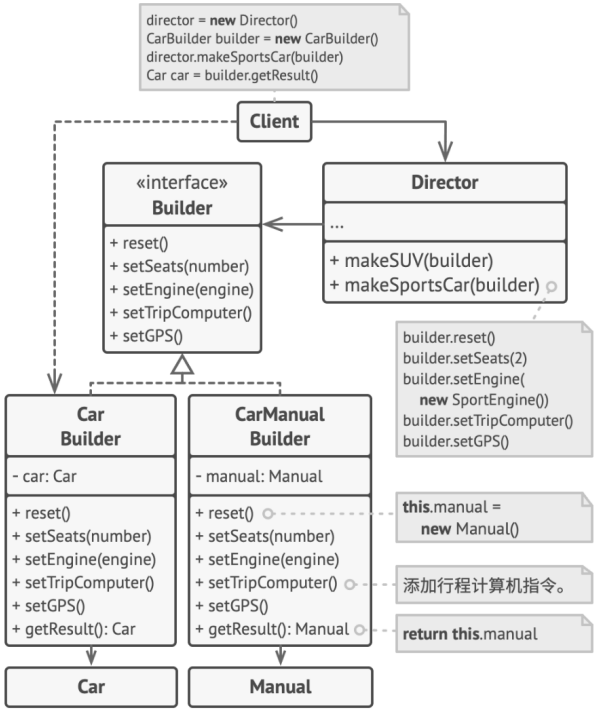
严格来说， 程序中并不一定需要主管类 ，客户端代码可直接以特定顺序调用创建步骤。不

过，`主管类中非常适合放入各种例行构造流程， 以便在程序中反复使用。

此外， 对于客户端代码来说， 主管类完全隐藏了产品构造细节。 客户端只需要将一个生成器与主管类关联， 然后使用主管类来构造产品， 就能从生成器处获得构造结果了。



伪代码



分步骤制造汽车并制作对应型号用户使用手册的示例

生成器模式适合的应用场景

1. 使用生成器模式可避免 “重叠构造函数 （telescoping constructor）” 的出现。

```
class Pizza {
public:
    Pizza(int size) { }
    Pizza(int size, boolean cheese) { }
    Pizza(int size, boolean cheese, boolean pepperoni) {}
};
```

2. 当希望使用代码创建不同形式的产品 （例如石头或木头房屋） 时， 可使用生成器模式。

3. 使用生成器构造组合树或其他复杂对象。

生成器模式优缺点

优点：

（1）可以分步创建对象， 暂缓创建步骤或递归运行创建步骤。

（2）生成不同形式的产品时， 可以复用相同的制造代码。

（3）单一职责原则，可以将复杂构造代码从产品的业务逻辑中分离出来。

缺点：
由于该模式需要新增多个类， 因此代码整体复杂程度会有所增加。

与其它模式的关系

在许多设计工作的初期都会使用工厂方法 模式 （较为简单， 而且可以更方便地通过子类进行定制）， 随后演化为使用抽象工厂模式 、 原型模式 或生成器模式 （更灵活但更加复杂）。

生成器 重点关注如何分步生成复杂对象。 抽象工厂 专门用于生产一系列相关对象。 抽象工厂 会马上返回产品， 生成器 则允许你在获取产品前执行一些额外构造步骤。

你可以在创建**复杂组合模式**树时使用生成器 ， 因为这可使其构造步骤以递归的方式运行。

你可以结合使用生成器 和桥接模式 ： 主管类负责抽象工作， 各种不同的生成器负责实现工作。

抽象工厂 、 生成器 和 原型 都可以用单例模式来实现。