

4. 建造者模式

目录

- 建造者模式
 - 模式动机
 - 模式定义
 - 模式结构
 - 时序图
 - 代码分析
 - 模式分析
 - 实例
 - 优点
 - 缺点
 - 适用环境
 - 模式应用
 - 模式扩展
 - 总结

4.1. 模式动机

无论是在现实世界中还是在软件系统中，都存在一些复杂的对象，它们拥有多个组成部分，如汽车，它包括车轮、方向盘、发送机等各种部件。而对于大多数用户而言，无须知道这些部件的装配细节，也几乎不会使用单独某个部件，而是使用一辆完整的汽车，可以通过建造者模式对其进行设计与描述，建造者模式可以将部件和其组装过程分开，一步一步创建一个复杂的对象。用户只需要指定复杂对象的类型就可以得到该对象，而无须知道其内部的具体构造细节。

在软件开发中，也存在大量类似汽车一样的复杂对象，它们拥有一系列成员属性，这些成员属性中有些是引用类型的成员对象。而且在这些复杂对象中，还可能存在一些限制条件，如某些属性没有赋值则复杂对象不能作为一个完整的产品使用；有些属性的赋值必须按照某个顺序，一个属性没有赋值之前，另一个属性可能无法赋值等。

复杂对象相当于一辆有待建造的汽车，而对象的属性相当于汽车的部件，建造产品的过程就相当于组合部件的过程。由于组合部件的过程很复杂，因此，这些部件的组合过程往往被“外部化”到一个称作建造者的对象里，建造者返还给客户端的是一个已经建造完毕的完整产品对象，而用户无须关心该对象所包含的属性以及它们的组装方式，这就是建造者模式的模式动机。

4.2. 模式定义

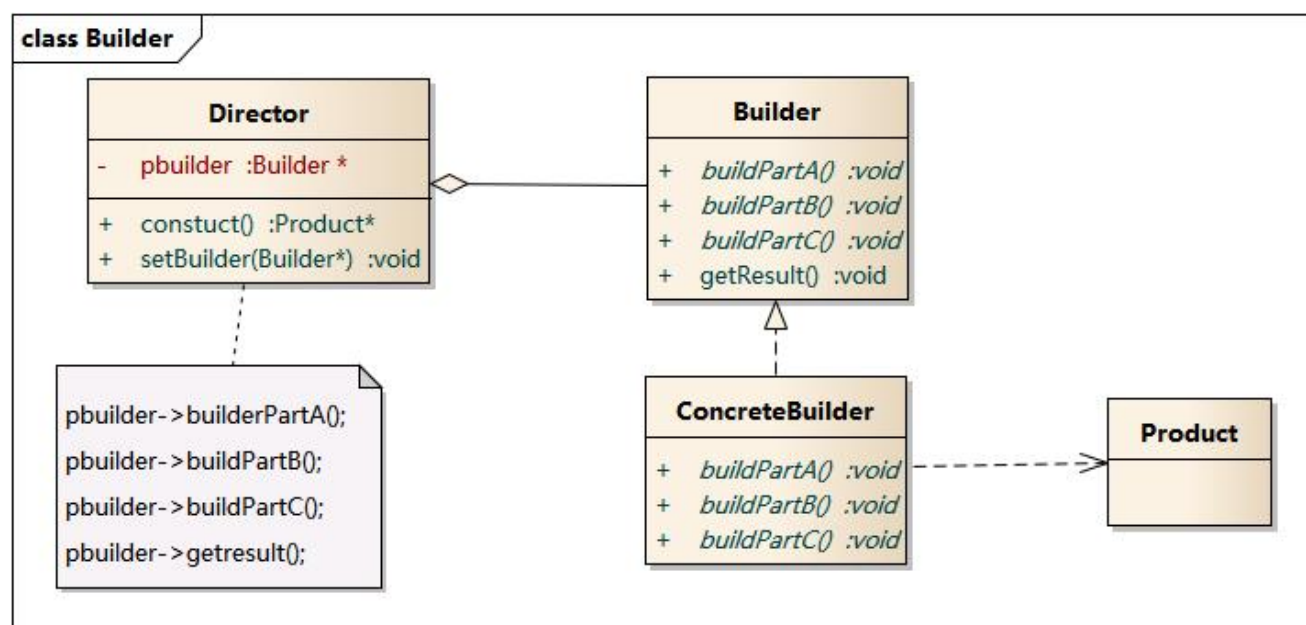
造者模式(Builder Pattern): 将一个复杂对象的构建与它的表示分离, 使得同样的构建过程可以创建不同的表示。

建造者模式是一步一步创建一个复杂的对象, 它允许用户只通过指定复杂对象的类型和内容就可以构建它们, 用户不需要知道内部的具体构建细节。建造者模式属于对象创建型模式。根据中文翻译的不同, 建造者模式又可以称为生成器模式。

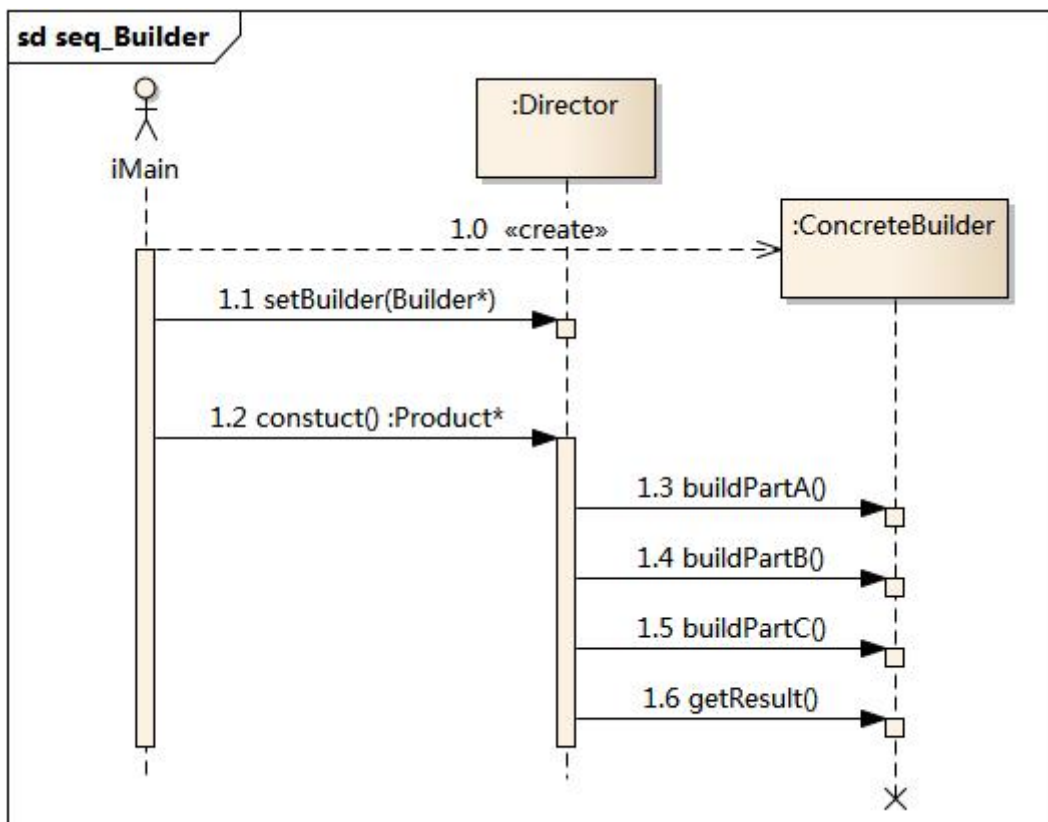
4.3. 模式结构

建造者模式包含如下角色:

- Builder: 抽象建造者
- ConcreteBuilder: 具体建造者
- Director: 指挥者
- Product: 产品角色



4.4. 时序图



4.5. 代码分析

```

1  #include <iostream>
2  #include "ConcreteBuilder.h"
3  #include "Director.h"
4  #include "Builder.h"
5  #include "Product.h"
6
7  using namespace std;
8
9  int main(int argc, char *argv[])
10 {
11     ConcreteBuilder * builder = new ConcreteBuilder();
12     Director director;
13     director.setBuilder(builder);
14     Product * pd = director.constuct();
15     pd->show();
16
17     delete builder;
18     delete pd;
19     return 0;
20 }
  
```

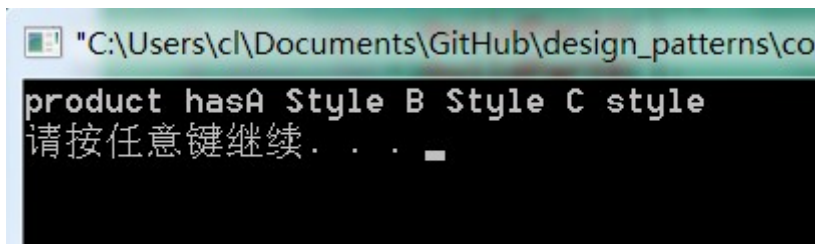
```
1  ////////////////////////////////////////////
2  //  ConcreteBuilder.cpp
3  //  Implementation of the Class ConcreteBuilder
4  //  Created on:      02-十月-2014 15:57:03
5  //  Original author: colin
6  ////////////////////////////////////////////
7  #include "ConcreteBuilder.h"
8
9  ConcreteBuilder::ConcreteBuilder(){
10 }
11
12
13 ConcreteBuilder::~ConcreteBuilder(){
14 }
15
16 void ConcreteBuilder::buildPartA(){
17     m_prod->setA("A Style "); // 不同的建造者，可以实现不同产品的建造
18 }
19
20 void ConcreteBuilder::buildPartB(){
21     m_prod->setB("B Style ");
22 }
23
24 void ConcreteBuilder::buildPartC(){
25     m_prod->setC("C style ");
26 }
27
28
29
30
31
32
33
```

```

1  //////////////////////////////////////
2  //  Director.cpp
3  //  Implementation of the Class Director
4  //  Created on:      02-十月-2014 15:57:01
5  //  Original author: colin
6  //////////////////////////////////////
7
8  #include "Director.h"
9
10 Director::Director(){
11 }
12
13 Director::~Director(){
14 }
15
16 Product* Director::constuct(){
17     m_pbuilder->buildPartA();
18     m_pbuilder->buildPartB();
19     m_pbuilder->buildPartC();
20
21     return m_pbuilder->getResult();
22 }
23
24 void Director::setBuilder(Builder* buider){
25     m_pbuilder = buider;
26 }
27

```

运行结果：



4.6. 模式分析

抽象建造者类中定义了产品的创建方法和返回方法;

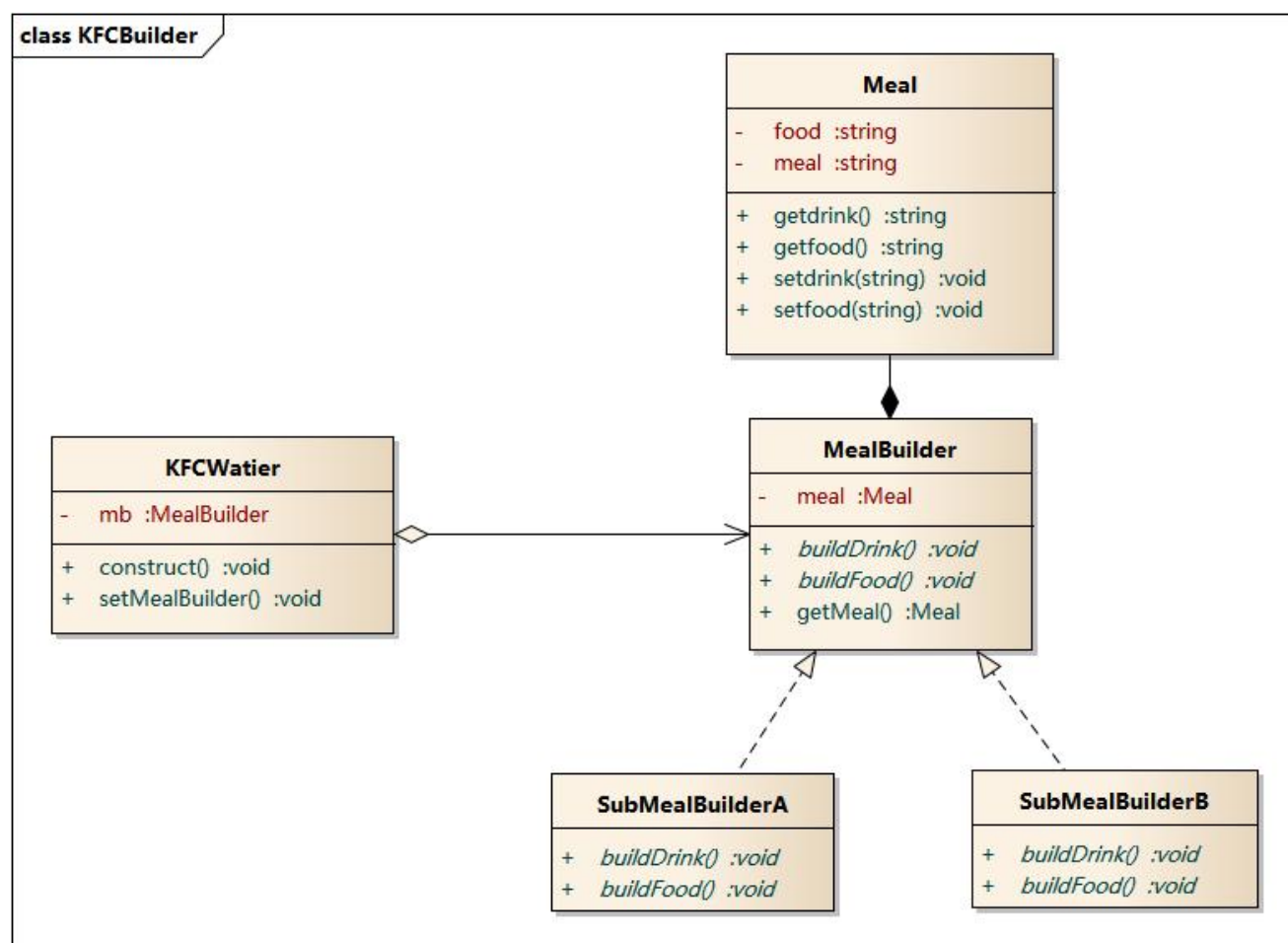
建造者模式的结构中还引入了一个指挥者类Director，该类的作用主要有两个：一方面它隔离了客户与生产过程；另一方面它负责控制产品的生成过程。指挥者针对抽象建造者编程，客户端只需要知道具体建造者的类型，即可通过指挥者类调用建造者的相关方法，返回一个完整的产品对象

在客户端代码中，无须关心产品对象的具体组装过程，只需确定具体建造者的类型即可，建造者模式将复杂对象的构建与对象的表现分离开来，这样使得同样的构建过程可以创建出不同的表现。

4.7. 实例

实例：KFC套餐

建造者模式可以用于描述KFC如何创建套餐：套餐是一个复杂对象，它一般包含主食（如汉堡、鸡肉卷等）和饮料（如果汁、可乐等）等组成部分，不同的套餐有不同的组成部分，而KFC的服务员可以根据顾客的要求，一步一步装配这些组成部分，构造一份完整的套餐，然后返回给顾客。



4.8. 优点

- 在建造者模式中，客户端不必知道产品内部组成的细节，将产品本身与产品的创建过程解耦，使得相同的创建过程可以创建不同的产品对象。
- 每一个具体建造者都相对独立，而与其他的具体建造者无关，因此可以很方便地替换具体建造者或增加新的具体建造者，用户使用不同的具体建造者即可得到不同的产品对象。
- 可以更加精细地控制产品的创建过程。将复杂产品的创建步骤分解在不同的方法中，使得创建过程更加清晰，也更方便使用程序来控制创建过程。
- 增加新的具体建造者无须修改原有类库的代码，指挥者类针对抽象建造者类编程，系统扩展方便，符合“开闭原则”。

4.9. 缺点

- 建造者模式所创建的产品一般具有较多的共同点，其组成部分相似，如果产品之间的差异性很大，则不适合使用建造者模式，因此其使用范围受到一定的限制。
- 如果产品的内部变化复杂，可能会导致需要定义很多具体建造者类来实现这种变化，导致系统变得很庞大。

4.10. 适用环境

在以下情况下可以使用建造者模式：

- 需要生成的产品对象有复杂的内部结构，这些产品对象通常包含多个成员属性。
- 需要生成的产品对象的属性相互依赖，需要指定其生成顺序。
- 对象的创建过程独立于创建该对象的类。在建造者模式中引入了指挥者类，将创建过程封装在指挥者类中，而不在建造者类中。
- 隔离复杂对象的创建和使用，并使得相同的创建过程可以创建不同的产品。

4.11. 模式应用

在很多游戏软件中，地图包括天空、地面、背景等组成部分，人物角色包括人体、服装、装备等组成部分，可以使用建造者模式对其进行设计，通过不同的具体建造者创建不同类型的地图或人物。

4.12. 模式扩展

建造者模式的简化：

- 省略抽象建造者角色：如果系统中只需要一个具体建造者的话，可以省略掉抽象建造者。
- 省略指挥者角色：在具体建造者只有一个的情况下，如果抽象建造者角色已经被省略掉，那么还可以省略指挥者角色，让

Builder角色扮演指挥者与建造者双重角色。

建造者模式与抽象工厂模式的比较：

- 与抽象工厂模式相比，建造者模式返回一个组装好的完整产品，而抽象工厂模式返回一系列相关的产品，这些产品位于不同的产品等级结构，构成了一个产品族。
- 在抽象工厂模式中，客户端实例化工厂类，然后调用工厂方法获取所需产品对象，而在建造者模式中，客户端可以不直接调用建造者的相关方法，而是通过指挥者类来指导如何生成对象，包括对象的组装过程和建造步骤，它侧重于一步步构造一个复杂对象，返回一个完整的对象。
- 如果将抽象工厂模式看成汽车配件生产工厂，生产一个产品族的产品，那么建造者模式就是一个汽车组装工厂，通过对部件的组装可以返回一辆完整的汽车。

4.13. 总结

- 建造者模式将一个复杂对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示。建造者模式是一步一步创建一个复杂的对象，它允许用户只通过指定复杂对象的类型和内容就可以构建它们，用户不需要知道内部的具体构建细节。建造者模式属于对象创建型模式。
- 建造者模式包含如下四个角色：抽象建造者为创建一个产品对象的各个部件指定抽象接口；具体建造者实现了抽象建造者接口，实现各个部件的构造和装配方法，定义并明确它所创建的复杂对象，也可以提供一个方法返回创建好的复杂产品对象；产品角色是被构建的复杂对象，包含多个组成部件；指挥者负责安排复杂对象的建造次序，指挥者与抽象建造者之间存在关联关系，可以在其construct()建造方法中调用建造者对象的部件构造与装配方法，完成复杂对象的建造
- 在建造者模式的结构中引入了一个指挥者类，该类的作用主要有两个：一方面它隔离了客户与生产过程；另一方面它负责控制产品的生成过程。指挥者针对抽象建造者编程，客户端只需要知道具体建造者的类型，即可通过指挥者类调用建造者的相关方法，返回一个完整的产品对象。
- 建造者模式的主要优点在于客户端不必知道产品内部组成的细节，将产品本身与产品的创建过程解耦，使得相同的创建过程可以创建不同的产品对象，每一个具体建造者都相对独立，而与其他的具体建造者无关，因此可以很方便地替换具体建造者或增加新的具体建造者，符合“开闭原则”，还可以更加精细地控制产品的创建过程；其主要缺点在于由于建造者模式所创建的产品一般具有较多的共同点，其组成部分相似，因此其使用范围受到一定的限制，如果产品的内部变化复杂，可能会导致需要定义很多具体建造者类来实现这种变化，导致系统变得很庞大。
- 建造者模式适用情况包括：需要生成的产品对象有复杂的内部结构，这些产品对象通常包含多个成员属性；需要生成的产品对象的属性相互依赖，需要指定其生成顺序；对象的创建过程独立于创建该对象的类；隔离复杂对象的创建和使用，并使得相同的创建过程可以创建不同类型的产品。