简述

创建者模式又叫建造者模式，是将一个复杂的对象的**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。创建者模式隐藏了复杂对象的创建过程，它把复杂对象的创建过程加以抽象，通过的创建具有复合属性的对象

模式中角色

- 指挥者（Director）直接和客户（Client）进行需求沟通
- 建造者 （Builder）定义生成实例的接口，供指挥者调用
- 具体建造者（ConcreteBuilder），完成建造者接口的具体实现方法

代码

```
public abstract class Builder {

    public abstract void makeTitle(String title);      //编写标题

    public abstract void makeString(String string);        //编写字符串

    public abstract void makeItem(String item);           //编写条目

    public abstract String close();                    //完成编写文档

}

```

```java
public class TextBuilder extends Builder {

    private StringBuffer buff = new StringBuffer();

    @Override
    public void makeTitle(String title) {
        buff.append("标题开始");
        buff.append("\n");
        buff.append(title);
        buff.append("\n");
        buff.append("标题结束\n");
    }

    @Override
    public void makeString(String string) {
        buff.append("string开始");
        buff.append("\n");
        buff.append(string);
        buff.append("\n");
        buff.append("string结束\n");
    }

    @Override
    public void makeItem(String item) {
        buff.append("item开始");
        buff.append("\n");
        buff.append(item);
        buff.append("\n");
        buff.append("item结束\n");
    }

    @Override
    public String close() {
        return buff.toString();
    }
}

public class HTMLBuilder extends Builder{

    private StringBuffer buff = new StringBuffer();

    @Override
    public void makeTitle(String title) {
        buff.append("<title>");
        buff.append(title);
        buff.append("</title>\n");
    }

    @Override
    public void makeString(String string) {
        buff.append("<string>");
        buff.append(string);
        buff.append("</string>\n");
    }
```

```
67
68      @Override
69      public void makeItem(String item) {
70          buff.append("<item>");
71          buff.append(item);
72          buff.append("</item>\n");
73      }
74
75      @Override
76      public String close() {
77          return buff.toString();
78      }
79
80  }
81
82  public class Dirctor {
83
84      private Builder builder;
85
86      public Dirctor(Builder builder){
87          this.builder = builder;
88      }
89
90      public void editDoc(){
91          builder.makeTitle("编写标题");
92          builder.makeString("编写字符串");
93          builder.makeItem("编写条目");
94          String result = builder.close();//编写结束
95          System.out.println(result);
96      }
97  }
98
99  public class Main {
100     public static void main(String[] args) {
101
102         Dirctor d = new Dirctor(new TextBuilder());
103         d.editDoc();
104
105         System.out.println("--------------");
106
107         d = new Dirctor(new HTMLBuilder());
108         d.editDoc();
109
110     }
111 }
```

优缺点

优点

- 使用建造者模式可以使客户端不必知道产品内部组成的细节。
- 具体的建造者类之间是相互独立的，这有利于系统的扩展。

- 具体的建造者相互独立，因此可以对建造的过程逐步细化，而不会对其他模块产生任何影响

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
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

建造者模式与抽象工厂模式的比较

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
汽车

扩展:构建一个类的实例

常规模式

```
@Getter
@Setter
public class Computer {
    private String mainboard;
    private String cpu;
    private String disk;
    private String memory;
    private String ssd;

     public Computer(String mainboard, String cpu, String disk, String memory, String ssd) {
         this.mainboard = mainboard;
         this.cpu = cpu;
         this.disk = disk;
         this.memory = memory;
         this.ssd = ssd;
     }

     public static void main(String[] args) {
         new Computer("主板", "cpu", "硬盘", "内存", "固态硬盘");
     }
}
```

建造者模式

```
@Setter
@Getter
public class NewComputer {
    private String mainboard;
    private String cpu;
    private String disk;
    private String memory;
    private String ssd;
```

```
9
10      private NewComputer(){}
11
12      private NewComputer(Builder builder) {
13          cpu = builder.cpu;
14          disk = builder.disk;
15          memory = builder.memory;
16          mainboard = builder.mainboard;
17          ssd = builder.ssd;
18      }
19
20      public final static class Builder{
21          private String mainboard;
22          private String cpu;
23          private String disk;
24          private String memory;
25          private String ssd;
26
27          public Builder(){}
28
29          public Builder cpu(String cpu){
30              this.cpu = cpu;
31              return this;
32          }
33
34          public Builder mainboard(String mainboard){
35              this.mainboard = mainboard;
36              return this;
37          }
38
39          public Builder disk(String disk){
40              this.disk = disk;
41              return this;
42          }
43
44          public Builder memory(String memory){
45              this.memory = memory;
46              return this;
47          }
48
49          public Builder ssd(String ssd){
50              this.ssd = ssd;
51              return this;
52          }
53
54          public NewComputer build(){
55              return new NewComputer(this);
56          }
57      }
58  }
```

客户端调用代码

```
1  public class NewMain {
2      public static void main(String[] args) {
```

```
3
4           NewComputer newComputer = new NewComputer.Builder()
5                   .cpu("cpu")
6                   .disk("disk")
7                   .build();
8
9       }
10  }
```

总结

一般的套路：优点是比较简单，开发效率高，缺点是如果参数真的很多的话鬼知道每个对应的是什么意思啊

Builder模式：优点是可以将构造器的setter方法名取成类似注释的方式，这样我们可以很清晰的知道刚才究竟设置的什么值