1 Builder 模式的核心思想

- Builder 模式注重过程，它关注与产品的组装过程
- 而抽象工厂注重结果，是对结果的抽象，客户端不知道过程

2 第一种 Builder 模式

- 首先定义一个抽象的 Builder 类：规定了子类必须实例化的方法

```
public abstract class PersonBuilder {
    Person person;
    public abstract void buildHead();
    public abstract void buildBody();
    public abstract void buildArmLeft();
    public abstract void buildArmRight();
    public abstract void buildLegLeft();
    public abstract void buildLegRight();
}
```

- 然后实例化一个具体的 Builder 实例

```
public class PersonThinBuilder extends PersonBuilder {
    public PersonThinBuilder() {
        person = new Person();
    }
    @Override
    public void buildHead() {
        person.setHead("小头");
    }
    // 其它代码略
```

- 最后实例化一个 Director(指挥者)，抽象创建过程

```
public class PersonDirector {
    private PersonBuilder pb;
    public PersonDirector(PersonBuilder pb) {
        this.pb = pb;
    }
    //这里就是创建者模式的核心创建过程，但是这个确定的过程完全可以直接放在 PersonBuilder 类中啊
```

```
    public Person createPerson() {
        pb.buildArmLeft();
        pb.buildArmRight();
        pb.buildBody();
        pb.buildHead();
        pb.buildLegLeft();
        pb.buildLegRight();
        return pb.person;
    }
}
@Test
public void test() {
    //客户端程序
    PersonDirector pd = new PersonDirector(new PersonThinBuilder());
    Person p = pd.createPerson();
    System.out.println(p);
}
```

3 第一种模式的困惑

- 这个模式在指挥者中定义了产品创建的过程，完全可以不需要这个指挥者，将这个过程直接放入创建者中。这各指挥者起什么作用呢？
- 我觉得这一种模式就是抽象工厂（PersonBuilder ）和模板模式（createPerson）的结合体，而且这个模板模式的代码很冗余
- **这种模式在客户端仍然没体现过程**

4 第二种 Builder 模式

```
public class PersonBuilder0 {
    private Person person;
    public PersonBuilder0() {
        person = new Person();
    }
    public PersonBuilder0 buildBody(String body) {
        person.setBody(body);
        return this;
    }
    public PersonBuilder0 buildHead(String head) {
        person.setHead(head);
        return this;
    }
    public PersonBuilder0 buildArmLeft(String leftArm) {
        person.setArmLeft(leftArm);
        return this;
    }
    public PersonBuilder0 buildArmRight(String rightArm) {
        person.setArmRight(rightArm);
        return this;
    }
    public PersonBuilder0 buildLegLeft(String leftLeg) {
        person.setLegLeft(leftLeg);
```

```
            return this;
        }
        public PersonBuilder0 buildLegRight(String rightLeg) {
            person.setLegRight(rightLeg);
            return this;
        }
        public Person build() {
            return person;
        }
    }
    @Test
    public void test() {
        Person p = new PersonBuilder0().buildArmLeft("大左胳膊").buildArmLeft("小右胳
膊").build();
        System.out.println(p);//没有手和头
    }
```

5 第二种 Builder 模式见解

- 很明显第二种模式客户端的是很体现过程的
- 而且灵活些也很强，客户端可以任意的组装，形成不同的产品
- 并且如果 Person 中的组件很复杂，也可以结合抽象工厂模式建立一个 Person 组件的抽象工厂，客户通过抽象工厂方便的选择想要的组件

6 参考

大话设计模式 Head First 设计模式 设计模式 github 源码地址