Java设计模式

Builder模式

创建个辅助类（一般使用内部类） `Builder` ，先将属性设置到 `Builder` 中，然后调用 `Builder#build` 方法生成实例：

```
public class Toy {
    private String head;
    private String body;
    private ArrayList legs;
    private ArrayList hands;

    public String getHead() {
        return head;
    }

    public void setHead(String head) {
        this.head = head;
    }

    public String getBody() {
        return body;
    }

    public void setBody(String body) {
        this.body = body;
    }

    public ArrayList getLegs() {
        return legs;
    }

    public void setLegs(ArrayList legs) {
        this.legs = legs;
    }

    public ArrayList getHands() {
        return hands;
    }

    public void setHands(ArrayList hands) {
        this.hands = hands;
    }

    static class Builder {
        private Toy toy;

        public Builder() {
            toy = new Toy();
        }

        public Builder setHead(String head) {
            toy.setHead(head);
            return this;
        }

        public Builder setBody(String body) {
            toy.setBody(body);
            return this;
        }
```

```
        public Builder setLegs(ArrayList legs) {
            toy.setLegs(legs);
            return this;
        }

        public Builder setHands(ArrayList hands) {
            toy.setHands(hands);
            return this;
        }

        public Toy build() {
            return toy;
        }
    }

    public static void main(String[] hh) {
        ArrayList hands = new ArrayList();
        hands.add("left");
        hands.add("right");

        ArrayList legs = new ArrayList();
        legs.add("left");
        legs.add("right");

        Toy toy = new Toy.Builder()
                .setBody("body")
                .setHands(hands)
                .setLegs(legs)
                .setHead("head")
                .build();
    }

}
```

策略模式

定义

- 策略模式定义了一系列的算法，并将每一个算法封装起来，而且使他们可以相互替换，让算法独立于使用它的客户而独立变化
- `行为参数化模式` 与 `策略模式` 比较相近