链滴

《Head First 设计模式》：状态模式

作者：jingqueyimu

原文链接：https://ld246.com/article/1600957372921

来源网站：链滴

正文

一、定义

状态模式允许对象在内部状态改变时改变它的行为，对象看起来好像修改了它的类。

要点：

- 状态模式允许一个对象基于内部状态而拥有不同的行为。
- 状态模式将状态封装成为独立的类，并将动作委托到代表当前状态的对象。
- 通过将每个状态封装进一个类，我们把以后需要做的任何改变局部化了。

二、实现步骤

1、创建状态接口

```
/**
 * 状态接口
 */
public interface State {

    /**
     * 根据状态进行处理的方法
     */
    public void handle();
}
```

2、在持有状态的类中，将请求委托给状态类

```
/**
 * 持有状态的上下文类
 */
public class Context {

    private State state;

    public State getState() {
        return state;
    }

    public void setState(State state) {
        this.state = state;
    }

    /**
     * 接收请求，并将请求委托给状态类
     */
    public void request() {
        state.handle();
```

```
    }
}
```

3、创建具体的状态，并实现状态接口

（1）具体状态A

```
/**
 * 具体状态A
 */
public class ConcreteStateA implements State {

    Context context;

    public ConcreteStateA() {
        context = new Context();
    }

    @Override
    public void handle() {
        // 实现该状态下相应的行为
        System.out.println("Context is in A state, and start to do something...");
        context.setState(this);
    }
}
```

（2）具体状态B

```
/**
 * 具体状态B
 */
public class ConcreteStateB implements State {

    Context context;

    public ConcreteStateB() {
        context = new Context();
    }

    @Override
    public void handle() {
        // 实现该状态下相应的行为
        System.out.println("Context is in B state, and start to do something...");
        context.setState(this);
    }
}
```

4、通过改变状态，来改变上下文类的行为

```
public class Test {

    public static void main(String[] args) {
```

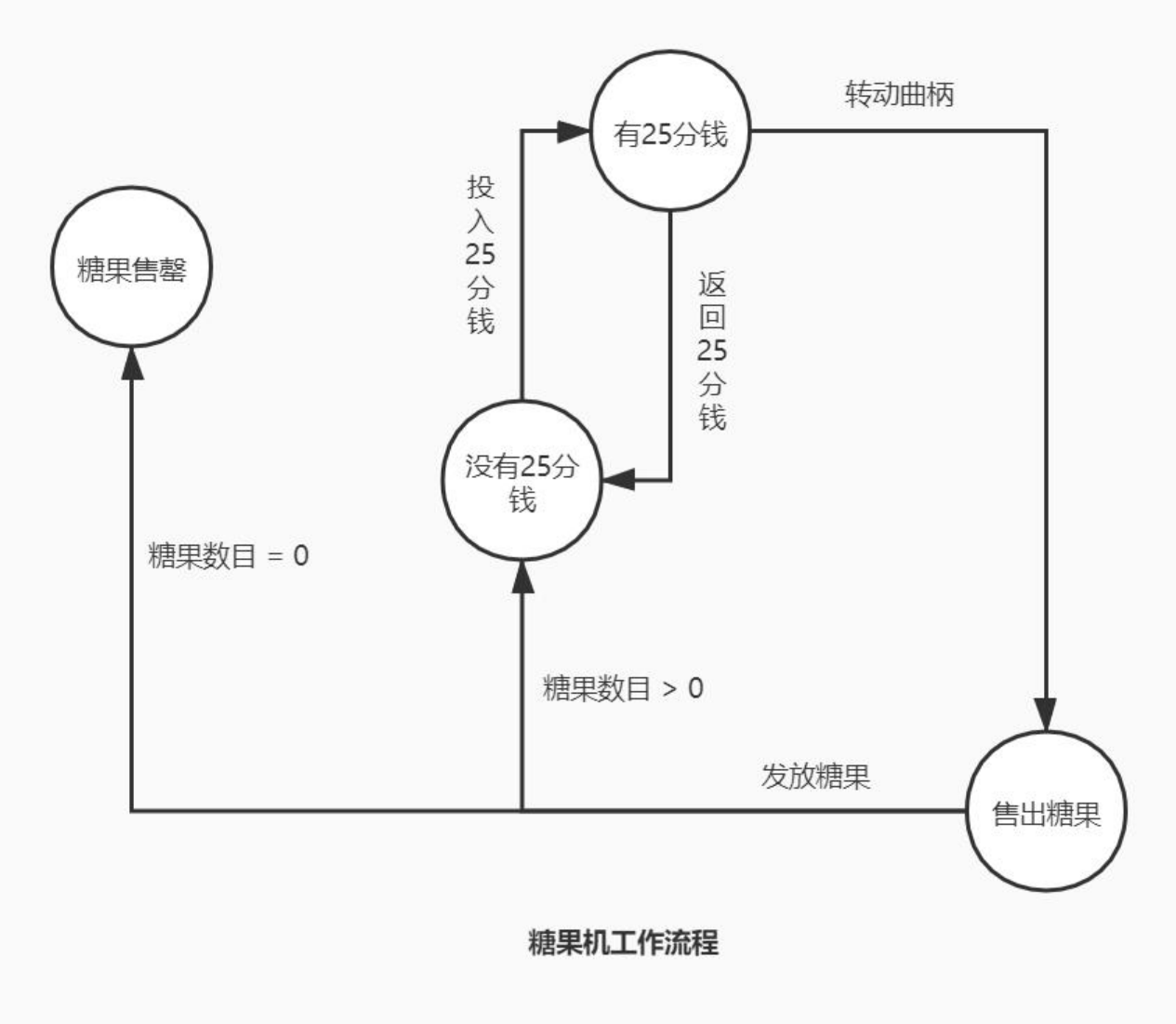
```
        // 上下文
        Context context = new Context();
        // 状态
        State stateA = new ConcreteStateA();
        State stateB = new ConcreteStateB();
        // 通过状态改变行为
        context.setState(stateA);
        context.request();
        context.setState(stateB);
        context.request();
    }
}
```

三、举个栗子

1、背景

万能糖果公司打算使用 Java 来实现糖果机的控制器。他们希望设计能够尽量有弹性而且好维护，因将来可能要为糖果机增加更多的行为。

糖果机的工作流程如下：



糖果机工作流程

2、实现

（1）创建状态接口，并定义相应的糖果机行为

```
/**
 * 状态接口
 */
public interface State {

    /**
     * 投入25分钱
     */
    public void insertQuarter();

    /**
     * 退回25分钱
     */
    public void ejectQuarter();

    /**
     * 转动曲柄
     */
    public void turnCrank();

    /**
     * 发放糖果
     */
    public void dispense();
}
```

（2）创建糖果机

将传递给糖果机的请求，委托给状态类。

```
/**
 * 糖果机
 */
public class GumballMachine {

    State soldOutState;
    State noQuarterState;
    State hasQuarterState;
    State soldState;

    State state = soldOutState;
    int gumballCount = 0;

    public GumballMachine(int initGumballCount) {
        soldOutState = new SoldOutState(this);
        noQuarterState = new NoQuarterState(this);
        hasQuarterState = new HasQuarterState(this);
        soldState = new SoldState(this);
        // 初始化糖果数量
```

```
        this.gumballCount = initGumballCount;
        // 初始化糖果机状态
        if (initGumballCount > 0) {
            state = noQuarterState;
        } else {
            state = soldOutState;
        }
    }

    /**
     * 投入25分钱
     */
    public void insertQuarter() {
        state.insertQuarter();
    }

    /**
     * 退回25分钱
     */
    public void ejectQuarter() {
        state.ejectQuarter();
    }

    /**
     * 转动曲柄
     */
    public void turnCrank() {
        state.turnCrank();
        state.dispense();
    }

    /**
     * 发放糖果
     */
    public void releaseBall() {
        System.out.println("A gumball comes rolling out the slot...");
        if (gumballCount > 0) {
            gumballCount = gumballCount -1;
        }
    }

    public void setState(State state) {
        this.state = state;
    }

    public int getGumballCount() {
        return gumballCount;
    }

    public State getSoldOutState() {
        return soldOutState;
    }

    public State getNoQuarterState() {
```

```
        return noQuarterState;
    }

    public State getHasQuarterState() {
        return hasQuarterState;
    }

    public State getSoldState() {
        return soldState;
    }
}
```

（3）创建具体的状态，并实现状态接口

```
/**
 * 未投入25分钱状态
 */
public class NoQuarterState implements State {

    GumballMachine gumballMachine;

    public NoQuarterState(GumballMachine gumballMachine) {
        this.gumballMachine = gumballMachine;
    }

    @Override
    public void insertQuarter() {
        // 投入25分钱，并转到已投入25分钱状态
        System.out.println("You inserted a quarter");
        gumballMachine.setState(gumballMachine.getHasQuarterState());
    }

    @Override
    public void ejectQuarter() {
        // 当前为未投入25分钱状态，不能退回25分钱
        System.out.println("You haven't inserted a quarter");
    }

    @Override
    public void turnCrank() {
        // 当前为未投入25分钱状态，不能转动曲柄
        System.out.println("You truned, but there's no quarter");
    }

    @Override
    public void dispense() {
        // 当前为未投入25分钱状态，不能发放糖果
        System.out.println("You need to pay first");
    }
}

/**
 * 已投入25分钱状态
 */
```

```
public class HasQuarterState implements State {

    GumballMachine gumballMachine;

    public HasQuarterState(GumballMachine gumballMachine) {
        this.gumballMachine = gumballMachine;
    }

    @Override
    public void insertQuarter() {
        // 当前为已投入25分钱状态，不能再次投入
        System.out.println("You can't insert another quarter");
    }

    @Override
    public void ejectQuarter() {
        // 退回25分钱，并将状态转到未投入25分钱状态
        System.out.println("Quarter returned");
        gumballMachine.setState(gumballMachine.getNoQuarterState());
    }

    @Override
    public void turnCrank() {
        // 转动曲柄，并将状态转为售出状态
        System.out.println("You turned...");
        gumballMachine.setState(gumballMachine.getSoldState());
    }

    @Override
    public void dispense() {
        // 当前为已投入25分钱状态，还未转动曲柄，不能发放糖果
        System.out.println("No gumball dispensed");
    }
}

/**
 * 售出状态
 */
public class SoldState implements State {

    GumballMachine gumballMachine;

    public SoldState(GumballMachine gumballMachine) {
        this.gumballMachine = gumballMachine;
    }

    @Override
    public void insertQuarter() {
        // 当前为售出状态，不能再次投入25分钱
        System.out.println("Please wait, we're already giving you a gumball");
    }

    @Override
    public void ejectQuarter() {
```

```
        // 当前为售出状态，不能退回25分钱
        System.out.println("Sorry, you already truned the crank");
    }

    @Override
    public void turnCrank() {
        // 当前为售出状态，不能再次转动曲柄
        System.out.println("Turning twice doesn't get you another gumball!");
    }

    @Override
    public void dispense() {
        // 发放糖果
        gumballMachine.releaseBall();
        if (gumballMachine.getGumballCount() > 0) {
            gumballMachine.setState(gumballMachine.getNoQuarterState());
        } else {
            System.out.println("Oops, out of gumballs!");
            gumballMachine.setState(gumballMachine.getSoldOutState());
        }
    }
}

/**
 * 售罄状态
 */
public class SoldOutState implements State {

    GumballMachine gumballMachine;

    public SoldOutState(GumballMachine gumballMachine) {
        this.gumballMachine = gumballMachine;
    }

    @Override
    public void insertQuarter() {
        // 当前为售罄状态，不能投入25分钱
        System.out.println("You can't insert a quarter, the machine is sold out");
    }

    @Override
    public void ejectQuarter() {
        // 当前为售罄状态，不能要求退回25分钱
        System.out.println("You can't eject, you haven't inserted a quarter yet");
    }

    @Override
    public void turnCrank() {
        // 当前为售罄状态，不能转动曲柄
        System.out.println("You turned, but there are no gumballs");
    }

    @Override
    public void dispense() {
```

```
        // 当前为售罄状态，不能发放糖果
        System.out.println("No gumball dispensed");
    }
}
```

（4）操作糖果机

```
public class Test {

    public static void main(String[] args) {
        // 糖果机
        GumballMachine gumballMachine = new GumballMachine(5);
        // 正常操作
        gumballMachine.insertQuarter();
        gumballMachine.turnCrank();
        System.out.println("-----------------------");
        // 异常操作
        gumballMachine.insertQuarter();
        gumballMachine.ejectQuarter();
        gumballMachine.turnCrank();
    }
}
```