转载请注明链接：https://blog.csdn.net/feather_wch/article/details/78647901

介绍状态模式，涉及到两个实例，第一个是用Java代码实现的基本状态模式，第二个是在Android中去切换“验证码发送按钮”的状态的实例。

> 特别鸣谢《Head First 设计模式》

状态模式

版本号：2018/09/06-1(20:00)

问题汇总

定义

1、状态模式的定义是什么?

> 允许一个对象在内部状态改变时改变自己的行为。这个对象就像是改变了他的类。

结构

2、状态模式的结构组成？分四部分。

> 1. State(interface)状态接口
> 2. Concrete State：具体状态
> 3. Context：持有各种状态的具体类。如下面实例中的GumballMachine。会在内部调用state的方法进行处理，如 `state.handle()` ，能在不同状态下完成不同的工作。

Java实例

3、State状态的具体实例

> 分为：状态接口、具体状态、持有状态的类、具体测试
> 1-状态接口

```
public interface State {
        public void insertQuarter();
        public void ejectQuarter();
        public void turnCrank();
        public void dispense();
}
```

> 2-具体状态: 这里有4种，继承自State。

```
public class HasQuarterState implements State{
        @Override
        public void insertQuarter() {
                System.out.println("HasQuarterState-insertQuarter ");
        }
        @Override
        public void ejectQuarter() {
                System.out.println("HasQuarterState-ejectQuarter ");
        }
        @Override
        public void turnCrank() {
                System.out.println("HasQuarterState-turnCrank ");
        }
        @Override
        public void dispense() {
                System.out.println("HasQuarterState-dispense ");
        }
}
public class NoQuarterState implements State{
        @Override
        public void insertQuarter() {
                System.out.println("NoQuarterState-insertQuarter ");
        }
        @Override
        public void ejectQuarter() {
                System.out.println("NoQuarterState-ejectQuarter ");
        }
        @Override
        public void turnCrank() {
                System.out.println("NoQuarterState-turnCrank ");
        }
        @Override
        public void dispense() {
                System.out.println("NoQuarterState-dispense ");
        }
}
public class SoldOutState implements State{
        @Override
        public void insertQuarter() {
                System.out.println("SoldOutState-insertQuarter ");
        }
        @Override
        public void ejectQuarter() {
                System.out.println("SoldOutState-ejectQuarter ");
        }
        @Override
        public void turnCrank() {
                System.out.println("SoldOutState-turnCrank ");
        }
        @Override
        public void dispense() {
                System.out.println("SoldOutState-dispense ");
        }
}
```

```
public class SoldState implements State{
        @Override
        public void insertQuarter() {
                System.out.println("SoldState-insertQuarter ");
        }
        @Override
        public void ejectQuarter() {
                System.out.println("SoldState-ejectQuarter ");
        }
        @Override
        public void turnCrank() {
                System.out.println("SoldState-turnCrank ");
        }
        @Override
        public void dispense() {
                System.out.println("SoldState-dispense ");
        }
}
```

> 3-Context: GumballMachine,持有状态。内部调用状态的方法。

```
public class GumballMachine {
        State state;

        public GumballMachine() {
                state = new SoldOutState();
        }
        public void insertQuarter() {
                state.insertQuarter();
        }
        public void ejectQuarter() {
                state.ejectQuarter();
        }
        public void turnCrank() {
                state.turnCrank();
                state.dispense();
        }
        public void setState(State state) {
                this.state = state;
        }
}
```

> 4-测试

```
GumballMachine gumballMachine = new GumballMachine();
// 售卖状态
gumballMachine.setState(new SoldOutState());
gumballMachine.insertQuarter();
gumballMachine.turnCrank();
//
gumballMachine.setState(new NoQuarterState());
gumballMachine.ejectQuarter();
gumballMachine.turnCrank();
```

状态模式和策略模式

1、State状态模式和Strategy策略模式有什么相同的地方?

> 结构类似

2、State状态模式和Strategy策略模式有什么不同的地方

> 1. State状态模式是一个对象的状态改变后，会改变自身的行为
> 2. Strategy是给一个对象封装好的算法族(行为)，可以更改算法族(行为)

实例二：Android中发送验证的状态机

1、验证码登陆界面，有一个发送验证码按钮。在没有输入时无法点击，输入号码后，点击发送，就会进入重新发送倒计时，以及多种状态。这就需要状态模式。

> 1-LoginSMSFragment有成员变量mState和Button
> 2-状态机

```
// 验证码的状态接口
private interface IDCodeState {
    public void changeButtonState();
}
// 还没有输入过手机号的State
private class NoNumberState implements IDCodeState{
    public void changeButtonState() {
          mSendCodeMsgButton.setClickable(false);
          mSendCodeMsgButton.setText("请输入手机号");
    }
}
// 处于可以发送验证码的状态
private class CanSendState implements IDCodeState{
    public void changeButtonState() {
        mSendCodeMsgButton.setClickable(true);
        mSendCodeMsgButton.setText("发送验证码");
    }
}
// 处于等待重新发送的状态
private class WaitReSendState implements  IDCodeState{
    public void changeButtonState() {
        mSendCodeMsgButton.setClickable(true);
        mSendCodeMsgButton.setText("重新发送(60s)");
    }
}
```

> 3-LoginSMSFragment内部使用状态模式

```
public class LoginSMSFragment extends SupportFragment{
    // 0. 验证码按钮的状态
    IDCodeState mState;
    // 1. 发送短信验证码的按钮
    private Button mSendCodeMsgButton;

    @Override
    public void onActivityCreated(@Nullable Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        // 使用状态
        mState = new NoNumberState();
        mState.changeButtonState();
    }
}
```

参考资料

1. 《Head First 设计模式》