16 状态模式

状态模式是一种虽然不是简单到一目了然甚至增加代码量的模式，但是一旦你明白了精髓，以后一定会感谢它带给你的好处。

关键是区分事物内部的状态，内部状态的改变往往会带来事物的行为改变

初识状态模式

有这样一个场景，开关控制电灯，然后通过点击开关来切换电灯状态

第一个例子：电灯程序

```
/**
 * 电灯程序
 */
// 一般写法-简单状态机
class Light{
    constructor(){
        this.button = null;
        this.state = 'off'
    }

    init(){
        const button = document.createElement('button');
        const self = this;
        button.innerHTML = '开关';
        this.button = document.body.appendChild(button);
        this.button.onclick = function(){
            self.buttonWasPressed();
        }
    }

    buttonWasPressed(){
        if(this.state === 'off'){
            this.state = 'on';
            console.log('开灯')
        } else if(this.state === 'on') {
            this.state = 'off';
            console.log('关灯');
        }
    }
```

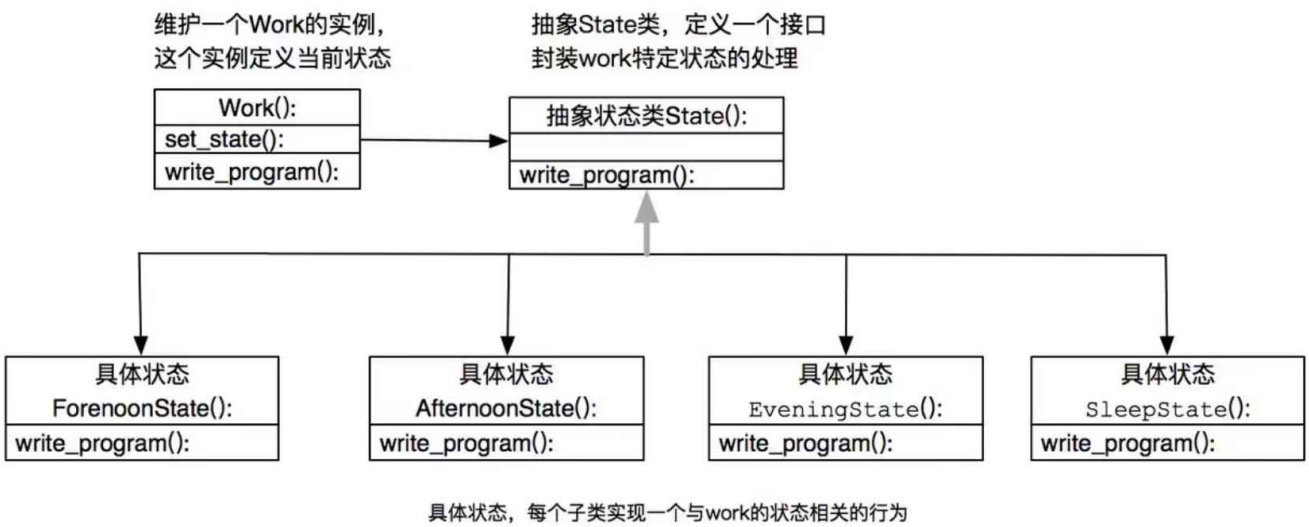
```
30
31 }
```

我们编写了一个状态机，逻辑简单且缜密，但是世界上的电灯并非只有一种，例如有一种调节灯，第一次打开弱光，第二次打开强光，第三次关灯，那我们必须改动buttonWasPressed函数才能实现了。我们总结下上述程序的缺点：

- buttonWasPressed函数违反开闭原则，每次新增light的状态这里都要改动
- buttonWasPressed封装了所有和状态有关的行为，未来可能变得非常巨大，尤其是内部业务逻辑复杂时
- 状态的切换操作不明显，仅仅为对state变量赋值，实际开发中必须读完buttonWasPressed的逻辑代码才能注意
- 状态之间的切换变成了函数中if-else逻辑的维护，难以阅读和维护

状态模式改进程序

一般我们谈到对象封装，都会优先封装对象的行为而非状态，但在状态模式则相反，关键是把状态本身封装为单独的类，跟此状态有关的行为封装到对应的类中，当button被按下时只需要把请求委托给具体的状态类来处理就好，它们负责渲染自己。同时可以把状态的切换规则分布在状态类中，消除大量条件分支语句



具体状态，每个子类实现一个与work的状态相关的行为

我们看一下根据状态模式改造之后的样子：

```
1 // 状态模式改造
2 class OffLightState {
3     constructor(light){
4         this.light;
5     }
```

```
6
7       buttonWasPressed(){
8           // off状态的逻辑
9           console.log('弱光');
10          this.light.setState(this.light.weakLightState);
11      }
12  }
13
14  class WeakLightState {
15      constructor(light){
16          this.light;
17      }
18
19      buttonWasPressed(){
20          console.log('强光');
21          this.light.setState(this.light.strongLightState);
22      }
23  }
24
25  class StrongLightState {
26      constructor(light){
27          this.light;
28      }
29
30      buttonWasPressed(){
31          console.log('关闭');
32          this.light.setState(this.light.offLightState);
33      }
34  }
35
36  // 改写Light类
37  class Light{
38      constructor(){
39          this.button = null;
40          this.offLightState = new OffLightState(this);
41          this.weakLightState = new WeakLightState(this);
42          this.strongLightState = new StrongLightState(this);
43      }
44
45      init(){
46          const button = document.createElement('button');
47          const self = this;
48          button.innerHTML = '开关';
49          // 不再通过字符串来记录state，而是通过具体的状态对象
50          this.currState = this.offLightState;
51          this.button = document.body.appendChild(button);
52          this.button.onclick = function(){
```

```
53              // 委托给具体状态对象执行
54              self.currState.buttonWasPressed();
55          }
56      }
57
58      setState(state){
59          // 提供状态切换方法，状态切换逻辑在具体状态类中，Context不需要关心
60          this.currState = state;
61      }
62
63  }
```

有状态相关的逻辑变化时我们只需要修改对象的状态类或者新增状态类就可以了

状态模式的定义

Gof中的定义：允许一个对象在其内部状态改变时改变他的行为，对象看起来似乎修改了它的类

- 将状态封装为独立的类，将请求委托给状态对象
- 客户角度来看，使用的对象会在不同状态下有不同的行为，这个对象看起来是从不同的类中实例化而来的

缺少抽象类的变通方式