第十二课 状态模式

学习目录

- **使用场景**
- **使用方式**
- **优缺点**

一．使用场景

大家可以顾名思义，状态模式其实就是对状态的一种封装，它区别于之前我们说过的其他的设计模式对行为的封装。它是通过改变对象状态来达到改变对象行为的目的，这是状态模式的主要作用。

因此状态模式它需要把状态单独封装成对象，当本体对象有请求时，本体对象会把请求给到状态对象，状态对象改变状态从而改变本体对象的行为。

比如我们用微信和好友进行视频通话，这其中视频通话我们可以看成准备视频通话、正在视频通话、视频通话结束这几种状态，这几种状态的改变可以决定视频通话行为的改变。

状态模式的核心思想是把对象的状态封装成单独的对象，通过状态的改变来改变对象的行为。

二．使用方式

我们可以先来用普通编程思路编写一个视频通话的代码。

普通视频通话

```
function VideoCallMethod(state){
```

```
    this.state = state;
    this.$button = null;
}
VideoCallMethod.prototype.init = function(){
    this.$button = document.createElement('button');
    this.$button.innerHTML = '准备视频通话';
    document.body.appendChild(this.$button);
    this.eventCenter();
}
VideoCallMethod.prototype.eventCenter = function(){
    var that = this;
    this.$button.onclick = function(){
        that.setFn();
    }
}
VideoCallMethod.prototype.setFn = function(){
    if(this.state == 'ready'){
        this.$button.innerHTML = '准备视频通话';
        this.state = 'calling';
    }else if(this.state == 'calling'){
        this.$button.innerHTML = '正在视频通话';
        this.state = 'end';
```

```
        }else if(this.state == 'end'){
            this.$button.innerHTML = '视频通话结束';
            this.state = 'ready';
        }
    }
    var v1 = new VideoCallMethod('ready');
    v1.init();
```

在这个例子中，通过改变状态已经可以达到改变对象行为的目标了，但是在 setFn 方法中出现了 3 个条件判断分支，现在还好，不需要太多状态记录，但是随着程序的扩展或者应用在其他地方的话，这些条件分支可能还会增加，这样一方面代码的可读性变差，一方面代码的维护难度也增加了。

因此需要使用状态模式对每个状态单独封装成独立的对象，通过调用这些独立的对象来减少条件分支带来的问题。

对象属性状态模式视频通话

```
    var stateObj = {
        'ready':{
            setState:function(){
                this.$button.innerHTML = '准备视频通话';
                this.state = stateObj.calling;
                console.log('准备视频通话');
            }
```

```
    },
    'calling':{
        setState:function(){
            this.$button.innerHTML = '正在视频通话';
            this.state = stateObj.end;
            console.log('正在视频通话');
        }
    },
    'end':{
        setState:function(){
            this.$button.innerHTML = '视频通话结束';
            this.state = stateObj.ready;
            console.log('视频通话结束');
        }
    }
};

function VideoCallMethod(){
    this.state = stateObj.ready;
    this.$button = null;
}
VideoCallMethod.prototype.init = function(){
```

```
        this.$button = document.createElement('button');
        this.$button.innerHTML = '准备视频通话';
        document.body.appendChild(this.$button);
        this.eventCenter();
    }
    VideoCallMethod.prototype.eventCenter = function(){
        var that = this;
        this.$button.onclick = function(){
            that.state.setState.call(that);
        }
    }

    var v1 = new VideoCallMethod();
    v1.init();
```

这里的状态模式是通过对象属性的形式来保存变量状态，还可以通过把状态封装成单独的构造函数来保存变量状态。

构造函数状态模式视频通话

```
    function Ready(){}
    Ready.prototype.setState = function(){
        this.$button.innerHTML = '准备视频通话';
        this.state = this.callingState;
```

```
    console.log('准备视频通话');
}

function Calling(){}
Calling.prototype.setState = function(){
    this.$button.innerHTML = '正在视频通话';
        this.state = this.endState;
        console.log('正在视频通话');
}

function End(){}
End.prototype.setState = function(){
    this.$button.innerHTML = '视频通话结束';
    this.state = this.readyState;
    console.log('视频通话结束');
}

function VideoCallMethod(){
    this.readyState = new Ready();
    this.callingState = new Calling();
    this.endState = new End();
    this.state = this.readyState;
```

```
        this.$button = null;
    }
    VideoCallMethod.prototype.init = function(){
        this.$button = document.createElement('button');
        this.$button.innerHTML = '准备视频通话';
        document.body.appendChild(this.$button);
        this.eventCenter();
    }
    VideoCallMethod.prototype.eventCenter = function(){
        var that = this;
        this.$button.onclick = function(){
            that.state.setState.call(that);
        }
    }

    var v1 = new VideoCallMethod();
    v1.init();
```

状态模式与策略模式相同与区别

1.两种模式都是封装出来多个独立对象，上下文对象根据需要把执行请求委托给指定的策略类或者状态类来执行。

2.状态模式侧重点在于一开始就确定了，他们是为了切换状态而准备的，调用者不需要

知道状态类的具体细节也可以用状态来改变行为。

3.策略类是互不干扰和互不影响的独立对象，调用者可以根据具体需要来调用指定的策略类，调用者需要知道策略类的相关作用。

三．优缺点

1.状态模式可以把状态单独封装成对象来管理，避免了状态变化带来的上下文条件分支判断语句。

2.状态模式是对状态的封装，通过状态的改变来改变行为，把请求委托给定义的状态对象来执行，实现了请求与状态的独立。

3.状态模式需要单独维护多个状态类，这些类可能随着业务逻辑继续增加，因此维护成本也会增加，状态类之间相互之间的联系也会变得复杂。

4.状态模式如果构建的状态类过多，也会让代码看上去没有太强的逻辑性，可能会在之后逻辑代码增多的情况下让代码变得难以维护。

谢谢观看！

我是星星课堂老师：周小周