第六课 模板方法模式

学习目录

- **使用场景**
- **使用方式**
- **优缺点**

一．使用场景

在之前讲解构造函数组件化的时候，我们基本上讲解的是一整套组件的流程，这是其中一种构建组件的方法，其实还可以利用继承的方式，实现由基类定义基本方法，由扩展类来继承基类，根据业务需要进行组件类的扩展方法。

这种方式可以构建一套简单的树形结构，最基础的是基类，同时根据扩展需要开枝散叶由子类在不改变基类结构的情况下进行更多的扩展，这种方式更加符合 js 基于原型继承的语言特点，因此可以用来做库与框架的组件化开发。

这种扩展方式要求的基类结构是复用的，基类还要指定子类只能在指定的位置进行扩展，其他方法必须按照基类来执行，这样才能保证所有的子类属于同一个基类，基类的结构才是真正的复用。

模板方法模式的核心思想是构建一个基类，子类继承这个基类根据不同需求进行扩展。

二．使用方式

```
function BaseFn(){
```

```
}
BaseFn.prototype.init = function(){
    this.firstFn();
    this.secendFn();
    if(this.needEditThird()){
        this.thirdFn();
    }
}
BaseFn.prototype.firstFn = function(){
    console.log('基类第一个方法');
}
BaseFn.prototype.secendFn = function(){
    throw new Error('必须由子类改写基类的第二个方法');
}
BaseFn.prototype.thirdFn = function(){
    console.log('子类决定是否改写基类的第三个方法');
}
BaseFn.prototype.needEditThird = function(){
    return true;
}
function SubFn(){
    BaseFn.apply(this);
```

```
}
SubFn.prototype = new BaseFn();
SubFn.prototype.secendFn = function(){
    console.log('子类改写基类的第二个方法');
}
SubFn.prototype.thirdFn = function(){
    console.log('子类改写基类的第三个方法');
}
SubFn.prototype.needEditThird = function(){
    return false;
}
var s1 = new SubFn();
s1.init();
```

这是模板方法模式的基本用法，它的使用还被用来构建某些 ui 组件，比如说模态框作为基类，在此基础上可以扩展出来信息提示模态框、操作按钮模态框等等扩展组件。

模态框基类、标题子类、按钮子类

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html;
```

```
charset=utf-8" />
        <title>第六课 模板方法模式</title>
        <style>
            .tip{
                width:300px;
                height:300px;
                background: orange;
            }
            .header{
                border-bottom:1px solid #ccc;
                height:45px;
            }
            .content{
                height:200px;
                border-bottom:1px solid #ccc;
            }
            .footer{
                height:55px;
            }
        </style>
    </head>
```

```
<body>
    <div class="wrap">
        <!-- <div class="tip">
            <div class="header">标题</div>
            <div class="content">内容</div>
            <div class="footer">底部</div>
        </div> -->
    </div>
    <script>
        //基类
        function Tip(){
            this.wrap = document.getElementsByClassName('wrap')[0];
            this.tip = document.createElement('div');
            this.header = document.createElement('div');
            this.content = document.createElement('div');
            this.footer = document.createElement('div');
            this.settings = {};
        }
        Tip.prototype.init = function(obj){
            this.settings = obj;
            this.createFn();
            this.eventCenter();
```

```
}
Tip.prototype.createFn = function(){
    this.tip.className = 'tip';
    this.header.className = 'header';
    this.content.className = 'content';
    this.footer.className = 'footer';

    this.header.innerHTML = '默认标题';
    this.content.innerHTML = this.settings.content;
    this.footer.innerHTML = '默认底部';

    this.tip.appendChild(this.header);
    this.tip.appendChild(this.content);
    this.tip.appendChild(this.footer);
    this.wrap.appendChild(this.tip);
}
Tip.prototype.eventCenter = function(){
    var that = this;
    if(this.footerEvent()){
        this.footerBtn.onclick = function(){
            that.settings.btnClickEvent();
        }
```

```
        }
    }
    Tip.prototype.footerEvent = function(){
        return false;
    }

    //标题
    function TitleTip(){
        Tip.call(this);
        this.title = document.createElement('h3');
    }
    TitleTip.prototype = new Tip();
    TitleTip.prototype.createFn = function(){
        Tip.prototype.createFn.call(this);
        this.header.innerHTML = '';
        this.title.innerHTML = this.settings.title;
        this.header.appendChild(this.title);
    }

    //按钮
    function BtnTip(){
        Tip.call(this);
```

```
    this.footerBtn = document.createElement('button');
}
BtnTip.prototype = new Tip();
BtnTip.prototype.createFn = function(){
    Tip.prototype.createFn.call(this);
    this.footer.innerHTML = '';
    this.footerBtn.innerHTML = this.settings.footerBtn;
    this.footer.appendChild(this.footerBtn);
}
BtnTip.prototype.footerEvent = function(){
    return true;
}

var t1 = new TitleTip();
var obj = {
    title:'标题',
    content:'内容'
};
t1.init(obj);

var b1 = new BtnTip();
var obj = {
```

```
            content:'内容',
            footerBtn:'按钮内容',
            btnClickEvent:function(){
                console.log('点击按钮');
            }
        };
        b1.init(obj);
    </script>
</body>
</html>
```

三．优缺点

1.模板方法模式把核心方法封装在了基类中，保证了方法的共享。

2.模板方法模式规定了子类定义方法的规则，对子类行为进行了约束，让子类可以保证稳定调用。

3.子类需要对父类方法进行重写和拓展，需要拓展人员熟悉基类结构。

谢谢观看！

我是星星课堂老师：周小周