第十四课 组件继承

学习目录

- **基类组件**
- **扩展类子组件**
- **钩子函数**

一．基类组件

基类组件实际在 js 中也称为基础父类构造函数，这个父类构造函数中包含了我们需要开发的各种组件的统一属性和方法，同时可以提供钩子函数方便不同类型的组件来构建各自的执行方法，所以基类组件往往是各种复杂组件的基础，也是最重要的基础组件。

其实大多数情况下基类组件并不复杂，甚至很简单，因为许多子组件功能各不相同，分离提出的相同结构的代码并不会特别多，以我们的模态框为例，除了新建模态框方法、构建模态框基础结构、显示模态框、隐藏模态框之外，其他的衍生模态框产品很难在有相同的结构代码，因此基类组件往往比较简单。

另外，比如在设计模式的模快方法模式中，基类组件的模块方法中只是定义了子类组件各种原型对象方法的执行顺序，在原型对象上也只是提供了子类组件需要根据不同子组件业务逻辑改写的实例方法与挂钩函数。

我们这里并不深入讲解模块方法模式，而是专门针对之前我们学到的继承的知识，来讲解如何利用继承来编写基于基类组件，同时拥有某些不同方法的子组件。

因此我们为了让子组件继承能够尽可能的少写代码，多复用统一代码，就要对父组件中进行一定的改进。

改进基类组件

```
Model.prototype.create = function(){
    var wrap = document.getElementById('wrap');
    var model = document.createElement('div');
    model.setAttribute('class','model');
    var str = '';
    str += '<div class="header"><div class="title">'+this.settingsOption.title+'</div>'
    str += '<div class="close">x</div></div>'
    str += '<div class="content">'+this.commonContent()+'</div>'
    str += '<div class="footer"><span class="confirmBtn">确定</span><span class="cancelBtn">取消</span></div>';
    model.innerHTML = str;
    wrap.appendChild(model);
    return model;
}
```

二．扩展类子组件

构建表单子组件，我们想要扩展一个内容区域为表单的子组件，同时提供一个获取表单数据的方法，在用户操作确定按钮触发的自定义事件中进行数据处理与业务逻辑交互。

可能子组件有很多个，因此确定获取到这两个数据之后要做的事情是不一样的，每个子组件对数据的交互操作会有不同，有可能提交的接口也有不同，因此最好不要把 ajax 数据发送的请求也放在子组件的实例方法中，通过自定义事件可以更加灵活的编写业务代码，这也符合 js 设计模式中的单一职责原则，也就是每个对象或者函数只做某一种功能，尽可能的保证函数或者对象之间的低耦合，也有利于对象或者函数的进一步开发与维护。

```
function FormModel(){
    Model.apply(this);
}

FormModel.prototype = new Model();
FormModel.prototype.constructor = FormModel;

FormModel.prototype.commonContent = function(){
    var str = '';;
    str += '<div><div>反馈意见：<input type="text" class="questionTitle" /></div>';
    str += '<div class="questionWrap" style="padding-top:10px;">具体问题：<textarea cols="30" class="questionContent" />请输入具体问题</textarea></div></div>';
    return str;
}

FormModel.prototype.getquestionData = function(){
    var questionTitle = document.getElementsByClassName('questionTitle')[0].value;
    var questionContent = document.getElementsByClassName('questionContent')[0].value;
    return {
        questionTitle:questionTitle,
        questionContent:questionContent
    }
}

var fm1 = new FormModel();
fm1.init();
var openModelBtn2 = document.getElementById('openModelBtn2');
openModelBtn2.onclick = function(){
    fm1.show();
}
fm1.modelDom.addEventListener('confirmEvent',function(){
    console.log(fm1.getquestionData());
});
```

三．钩子函数

钩子函数其实应用层面并不复杂，他为不同子组件提供了更加灵活的方法执行条件，子组件可以手动改写钩子函数条件来决定是否需要修改调用某些父组件提供的实例方法。

在基类组件中增加一个钩子函数，默认返回 true，保证各个子组件在不修改基类组件的钩子函数的时候让确定按钮的自定义事件可以正常执行调用。

```
Model.prototype.event = function(){
    var that = this;

    var confirmBtn = this.modelDom.getElementsByClassName('confirmBtn')[0];
    var confirmEvent = new CustomEvent('confirmEvent',{
        detail:{
            params : '确定参数'
        }
    })
    confirmBtn.onclick = function(){
        if(that.hasFormData()){
            that.modelDom.dispatchEvent(confirmEvent);
        }
    }

    var cancelBtn = this.modelDom.getElementsByClassName('cancelBtn')[0];
    var cancelEvent = new CustomEvent('cancelEvent',{
        detail:{
            params : '取消参数'
        }
    })
    cancelBtn.onclick = function(){
        that.modelDom.dispatchEvent(cancelEvent);
    }

    var close = this.modelDom.getElementsByClassName('close')[0];
    close.onclick = function(){
        that.close();
    }

}

Model.prototype.show = function(){···

Model.prototype.close = function(){···

Model.prototype.hasFormData = function(){
    return true;
}
```

在子组件中修改父组件的钩子函数，当表单的内容为空的时候不会执行调用父类的确定按钮的自定义事件，同时给出表单输入提示。

```
FormModel.prototype.hasFormData = function(){
    if(this.getquestionData().questionTitle && this.getquestionData().questionContent){
        return true;
    }else{
        alert('请输入反馈意见与具体问题');
    }
}
```

谢谢观看！

我是星星课堂老师：周小周