javascript设计模式

1. 原型模式

使用克隆的原型模式

作用：既然原型模式是通过克隆来创建对象的，那么很自然地会想到，如果需要一个跟某个对象一模一样的对象，就可以使用原型模式。

例子：某种飞机拥有分身技能，当它使用分身技能的时候，要在页面中创建一些跟它一模一样的飞机。

原型模式的实现关键，是语言本身是否提供了 clone 方法。ECMAScript 5提供了 Object.create方法，可以用来克隆对象。

代码：var plane1 = Object.create( plane );

1. 命令模式

使用用面向对象的方式实现命令模式

## 客户创建命令；调用者执行该命令；接收者在命令执行时执行相应操作。

作用：命令模式的意图是把请求封装为对象，从而分离请求的发起者和请求的接收者（执行者）之间的耦合关系。在命令被执行之前，可以预先往命令对象中植入命令的接收者。命令模式，通过command对象，连接接收者和调用者，降低了两者的耦合度，便于代码的修改和维护。命令模式提高了代码的模块化程度，但降低了代码的可阅读性（毕竟谁都不愿意看个代码的具体实现要逐层找半天，从调用者，到接收者），增加了代码的难度，所以只有在需要把操作和调用分离时，或需要对操作进行规范化处理时再用它。

代码：

var Tv = { //接收者-receiving,接收者对象提供实现具体的功能的方法

open:function(){

console.log( '打开电视机' );

},

close:function(){

console.log( '关闭电视机' );

}

};

var OpenTvCommand = function( receiver ){ //调用者-invoking,用来实现客户对象与接收者之间的低耦合，也就是弱化按钮之类的用户界面元素与其功能之间的耦合。

this.receiver = receiver; //预先植入的命令接收者被当成对象的属性保存起来

};

OpenTvCommand.prototype.execute = function(){

this.receiver.open(); // 执行命令，打开电视机

};

OpenTvCommand.prototype.undo = function(){

this.receiver.close(); // 撤销命令，关闭电视机

};

var setCommand = function( command ){ //客户-client,客户对象就是页面上的控件（能绑定鼠标键盘等事件的dom元素）

document.getElementById( 'execute' ).onclick = function(){

command.execute(); // 输出：打开电视机

};

document.getElementById( 'undo' ).onclick = function(){

command.undo(); // 输出：关闭电视机

};

};

setCommand( new OpenTvCommand( Tv ) );

使用闭包方式实现命令模式

var Tv = {

open:function(){

console.log( '打开电视机' );

},

close:function(){

console.log( '关闭电视机' );

}

};

var createCommand = function( receiver ){ //命令接收者会被封闭在闭包形成的环境中

var execute = function(){

return receiver.open(); // 执行命令，打开电视机

};

var undo = function(){

return receiver.close(); // 执行命令，关闭电视机

};

return {

execute:execute,

undo:undo

};

};

var setCommand = function( command ){

document.getElementById( 'execute' ).onclick = function(){

command.execute(); // 输出：打开电视机

};

document.getElementById( 'undo' ).onclick = function(){

command.undo(); // 输出：关闭电视机

};

};

setCommand( createCommand( Tv ) );