9 命令模式

假设有一个快餐店，而我是服务员，那么客人点餐时候我的工作就是把客人需求写在清单上交给厨房，客人不需要关心他的菜是谁炒的，当时它还可以支持预定上菜或者撤销订单，只要有订单在就可以满足客人需求

> 💡 这些记录着订餐信息的清单，便是命令模式中的命令对象

命令模式的用途

命令模式是最简单和优雅的模式之一，其中的命令（command）指的是一个执行某些特定事情的指令

常见的应用场景：

- 需要向某些对象发送请求实现自己的诉求
- 但是不知道明确的接收者与被请求的操作是什么
- 松耦合的设计使得请求者和请求接收者可以解耦

相对于偏向过程化的请求调用，command对象拥有更长的生命周期，所以我们可以更加灵活的再任意时刻执行这个command，例如预定1个小时之后炒菜

例子-菜单程序

这里有一个需求：

- 编写一个用户界面，至少数十个Button
- 复杂，需要分工，一个程序员负责绘制按钮和UI，另一个负责编写按钮点击逻辑

这时候我们必须解开按钮和负责行为对象之间的耦合，不变的是click之后执行逻辑，变化的是执行的具体逻辑，这里我们就可以借助命令对象的帮助来尝试实现：

```
/**
 * 菜单程序例子
 */
var button1 = document.getElementById('button1');
var button2 = document.getElementById('button2');
var button3 = document.getElementById('button3');

// 定义setCommand函数 - 负责安装命令 - 封装不变的部分
```

```
var setCommand = function(button, command){
    button.onclick = function(){
        command.execute();
    }
}

// 负责编写按钮点击逻辑的程序员实现逻辑功能
var MenuBar = {
    refresh: function(){
        console.log('刷新菜单目录');
    }
}

var SubMenu = {
    add: function(){
        console.log('增加子菜单');
    },
    delete: function(){
        console.log('删除子菜单');
    }
}

// 封装行为到命令类中
var RefreshMenuBarCommnad = function(receiver){
    this.receiver = receiver;
}

var AddSubMenuBarCommnad = function(receiver){
    this.receiver = receiver;
}

var DeleteSubMenuBarCommnad = function(receiver){
    this.receiver = receiver;
}

RefreshMenuBarCommnad.prototype.execute = function(){
    this.receiver.refresh();
}

AddSubMenuBarCommnad.prototype.execute = function(){
    this.receiver.add();
}

DeleteSubMenuBarCommnad.prototype.execute = function(){
    this.receiver.delete();
}

```

```
// 把命令接收者传入command对象，并且安装command到button上
var refreshMenuBarCommnad = new RefreshMenuBarCommnad(MenuBar);
var addSubMenuBarCommnad = new AddSubMenuBarCommnad(SubMenu);
var deleteSubMenuBarCommnad = new DeleteSubMenuBarCommnad(SubMenu);

setCommand(button1, refreshMenuBarCommnad);
setCommand(button2, addSubMenuBarCommnad);
setCommand(button3, deleteSubMenuBarCommnad);
```