

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：

```
1  /**
2   * 菜单程序例子
3   */
4  var button1 = document.getElementById('button1');
5  var button2 = document.getElementById('button2');
6  var button3 = document.getElementById('button3');
7
8  // 定义setCommand函数 - 负责安装命令 - 封装不变的部分
```

```
9 var setCommand = function(button, command){
10     button.onclick = function(){
11         command.execute();
12     }
13 }
14
15 // 负责编写按钮点击逻辑的程序员实现逻辑功能
16 var MenuBar = {
17     refresh: function(){
18         console.log('刷新菜单目录');
19     }
20 }
21
22 var SubMenu = {
23     add: function(){
24         console.log('增加子菜单');
25     },
26     delete: function(){
27         console.log('删除子菜单');
28     }
29 }
30
31 // 封装行为到命令类中
32 var RefreshMenuBarCommnad = function(receiver){
33     this.receiver = receiver;
34 }
35
36 var AddSubMenuBarCommnad = function(receiver){
37     this.receiver = receiver;
38 }
39
40 var DeleteSubMenuBarCommnad = function(receiver){
41     this.receiver = receiver;
42 }
43
44 RefreshMenuBarCommnad.prototype.execute = function(){
45     this.receiver.refresh();
46 }
47
48 AddSubMenuBarCommnad.prototype.execute = function(){
49     this.receiver.add();
50 }
51
52 DeleteSubMenuBarCommnad.prototype.execute = function(){
53     this.receiver.delete();
54 }
55
```

```

56 // 把命令接收者传入command对象，并且安装command到button上
57 var refreshMenuBarCommnad = new RefreshMenuBarCommnad(MenuBar);
58 var addSubMenuBarCommnad = new AddSubMenuBarCommnad(SubMenu);
59 var deleteSubMenuBarCommnad = new DeleteSubMenuBarCommnad(SubMenu);
60
61 setCommand(button1, refreshMenuBarCommnad);
62 setCommand(button2, addSubMenuBarCommnad);
63 setCommand(button3, deleteSubMenuBarCommnad);

```

Javascript中的命令模式

上面的命令模式可能让你感觉奇怪，无非是把对象的某个方法取了execute的名字，还引入了command和receiver这两个把事情复杂化的对象，用下面的代码也可以实现相同的功能：

```

1  /**
2   * 简单的实现
3   */
4
5  var bindClick = function(button, func){
6      button.onclick = func;
7  }
8
9  var MenuBar = {
10     refresh: function(){
11         console.log('刷新')
12     }
13 }
14
15 var SubMenu = {
16     add: function() {},
17     delete: function() {}
18 }
19
20 bindClick(button1, MenuBar.refresh);
21 bindClick(button2, SubMenu.add);

```

上述的做法是正确的，因为第二节中是模拟面向对象语言的命令模式的实现，将请求封装在command对象的execute方法中并通过传递command对象来调用命令，客户不关心事情具体如何进行的



命令模式的由来，其实是回调函数（callback）的一个面向对象的替代品

所以在把函数作为一等对象的JS中，跟策略模式一样，命令模式本身就已经融入到了JS语言中，运算块不必封装在command的execute方法中，也可以封装在普通函数中来自然的四处传递

这里我们再来看看基于闭包实现的命令模式，将接收者封闭在闭包环境中，执行命令其实就是执行存下来的execute函数：

```
1  /**
2   * 基于闭包的实现
3   */
4  var setCommand = function(button,command){
5      button.onclick = function(){
6          command.execute();
7      }
8  }
9
10 var MenuBar = {
11     refresh: function(){
12         console.log('刷新')
13     }
14 }
15
16 var RefreshMenuBarCommnad = function(receiver){
17     return {
18         execute: function(){
19             receiver.refresh();
20         }
21     }
22 }
23
24 var refreshMenuBarCommnad = RefreshMenuBarCommnad(MenuBar);
25 setCommand(button1, refreshMenuBarCommnad);
```

撤销命令

命令模式有一个关键作用是很方便地给命令对象增加撤销操作。类似客人通过电话取消订餐一样。

先看下我们的目标，给Animate类增加动画，来让小球水平移动，一个input输入框输入移动后的位置，一个button按钮触发移动，接着还需要让小球还原到开始之前的位置，我们看一下基础实现：

```
1  /**
2   * 小球移动Demo
3   */
4
```

```

5 var ball = document.getElementById('ball');
6 var pos = document.getElementById('pos');
7 var moveBtn = document.getElementById('moveBtn');
8
9 moveBtn.onclick = function(){
10     var animate = new Animate(ball);
11     animate.start('left',pos.value, 1000, 'strongEaseOut');
12 }

```

如果希望撤回，就输入一个-200再点击button，但是似乎有点蠢，我们看一下如何借助命令模式+撤销功能来实现：

```

1  /**
2   * 命令模式+撤销
3   */
4  var ball = document.getElementById('ball');
5  var pos = document.getElementById('pos');
6  var moveBtn = document.getElementById('moveBtn');
7  var cancelBtn = document.getElementById('cancelBtn');
8
9  var MoveCommand = function(receiver, pos){
10     this.receiver = receiver;
11     this.pos = pos;
12     this.oldPos = null;
13 }
14
15 MoveCommand.prototype.execute = function(){
16     this.receiver.start('left',this.pos, 1000, 'strongEaseOut');
17     this.oldPos = this.receiver.dom.getBoundingClientRect()[this.receiver.proper
18 }
19
20 MoveCommand.prototype.undo = function(){
21     this.receiver.start('left',this.oldPos, 1000, 'strongEaseOut');
22 }
23
24 var moveCommand;
25
26 moveBtn.onclick = function(){
27     var animate = new Animate(ball);
28     moveCommand = new MoveCommand(animate, pos.value);
29     moveCommand.execute();
30 }
31
32 cancelBtn.onclick = function(){
33     moveCommand.undo();

```

上述方法通过命令模式实现了撤销，需要撤销时记录的原始位置在执行execute时候就已经记录下来了，只需要提供一个undo方法即可