**介绍**

职责链模式（Chain of responsibility）是使多个对象都有机会处理请求，从而避免请求的发送者和接受者之间的耦合关系。将这个对象连成一条链，并沿着这条链传递该请求，直到有一个对象处理他为止。

也就是说，请求以后，从第一个对象开始，链中收到请求的对象要么亲自处理它，要么转发给链中的下一个候选者。提交请求的对象并不明确知道哪一个对象将会处理它——也就是该请求有一个隐式的接受者（implicit receiver）。根据运行时刻，任一候选者都可以响应相应的请求，候选者的数目是任意的，你可以在运行时刻决定哪些候选者参与到链中。

**正文**

对于JavaScript实现，我们可以利用其原型特性来实现职责链模式。

var NO\_TOPIC = -1;  
var Topic;  
  
function Handler(s, t) {  
 this.successor = s || null;  
 this.topic = t || 0;  
}  
  
Handler.prototype = {  
 handle: function () {  
 if (this.successor) {  
 this.successor.handle()  
 }  
 },  
 has: function () {  
 return this.topic != NO\_TOPIC;  
 }  
};

Handler只是接受2个参数，第一个是继任者（用于将处理请求传下去），第二个是传递层级（可以用于控制在某个层级下是否执行某个操作，也可以不用），Handler原型暴露了一个handle方法，这是实现该模式的重点，先来看看如何使用上述代码。

var app = new Handler({  
 handle: function () {  
 console.log('app handle');  
 }  
 }, 3);  
  
 var dialog = new Handler(app, 1);  
  
 var button = new Handler(dialog, 2);  
  
 button.handle();

改代码通过原型特性，调用代码从button.handle()->dialog.handle()->app.handle()->参数里的handle()，前三个都是调用原型的handle，最后才查找到传入的参数里的handle，然后输出结果，也就是说其实只有最后一层才处理。

那如何做到调用的时候，只让dialog的这个对象进行处理呢？其实可以定义dialog实例对象的handle方法就可以了，但需要在new button的之前来做，代码如下：

var app = new Handler({  
 handle: function () {  
 console.log('app handle');  
 }  
 }, 3);  
  
 var dialog = new Handler(app, 1);  
 dialog.handle = function () {  
 console.log('dialog before ...')  
 // 这里做具体的处理操作  
 console.log('dialog after ...')  
 };  
  
 var button = new Handler(dialog, 2);  
  
 button.handle();

该代码的执行结果即时dialog.handle里的处理结果，而不再是给app传入的参数里定义的handle的执行操作。

那能不能做到自身处理完以后，然后在让继任者继续处理呢？答案是肯定的，但是在调用的handle以后，需要利用原型的特性调用如下代码：

Handler.prototype.handle.call(this);

该句话的意思说，调用原型的handle方法，来继续调用其继任者（也就是successor ）的handle方法，以下代码表现为：button/dialog/app三个对象定义的handle都会执行。

var app = new Handler({  
 handle: function () {  
 console.log('app handle');  
 }  
}, 3);  
  
var dialog = new Handler(app, 1);  
dialog.handle = function () {  
 console.log('dialog before ...')  
 // 这里做具体的处理操作  
 Handler.prototype.handle.call(this); //继续往上走  
 console.log('dialog after ...')  
};  
  
var button = new Handler(dialog, 2);  
button.handle = function () {  
 console.log('button before ...')  
 // 这里做具体的处理操作  
 Handler.prototype.handle.call(this);  
 console.log('button after ...')  
};  
  
button.handle();

通过代码的运行结果我们可以看出，如果想先自身处理，然后再调用继任者处理的话，就在末尾执行Handler.prototype.handle.call(this);代码，如果想先处理继任者的代码，就在开头执行Handler.prototype.handle.call(this);代码。

**总结**

职责链模式经常和组合模式一起使用，这样一个构件的父构件可以作为其继任者。

同时，DOM里的事件冒泡机制也和此好像有点类似，比如点击一个按钮以后，如果不阻止冒泡，其click事件将一直向父元素冒泡，利用这个机制也可以处理很多相关的问题，比如本系列设计模式享元模式里的《例1：事件集中管理》的示例代码。