# 高级6

### 工厂模式

```
function crearePerson(opts){
    var person = {
        name: opts.name||"zhangrui"
    };
    person.sayName =  function(){
        alert(this.name);
    }
    return person;
}

var p1 = crearePerson({name:"zhangrui"});
var p2 = crearePerson({name:"jirengu"});
p1.sayName();
p2.sayName();
```

### 构造函数模式

```
function Person(name, age){
    this.name = name;
    this.age = age;
}
Person.prototype.sayName = function(){
     alert(this.name);
}

// 调用
var student = new Person("zhangrui", 32);
student.sayName();
```

### 混合模式

```
// 就是在原有的对象上面增加、覆盖对象的行为。
var Person = function(name, age){
    this.name = name;
    this.age = age;
};
Person.prototype.sayName = function(){
    alert(this.name);
}
var Student = function(name, age, score) {
    Person.call(this, name, age);
    this.score = score;
}
```

```
function inherit(superClass, slefClass){
    var _prototype = Object.create(superClass.prototype);
    _prototype.constructor = slefClass;
    slefClass.prototype = _prototype;
}
inherit(Person, Student);
Student.prototype.sayScore = function(){
    alert(this.score);
}


//调用
var student = new Student("zhangrui", 32, 100);
student.sayName();
student.sayScore();
```

**模块模式**

```
var student = (function(){
    var name = "zhangrui";
    function sayName(){
        alert(name);
    }
    return {
        name: name,
        sayName: sayName
    }
})()

// 调用
student.sayName();
```

**单例模式**

```
// 构造函数的实例只有一个 可以节约内存
// 一般是通过闭包存储内部实例，通过接口访问内部实例。

var People = (function(){
    var instance;
    function init() {
        this.a = 1;
        this.b = 2;
    }
    return {
        createPeople: function(){
            if(!instance){
                instance = init();
            }
```

```
            return instance;
        }
    }
})();


var obj1 = People.createPeople();
var obj2 = People.createPeople();
obj1 === obj2 // true
```

**发布订阅模式**

```
var EventCenter =(function(){
    // 事件池
    var events = {};

    // 事件挂载
    function on(evt, handler){
    events[evt] = events[evt] || [];
    events[evt].push({
        handler:handler
    });
    }

    // 事件触发
    function fire(evt, args){
        if(!events[evt]){
            return;
        }
        for(var i=0;i<events[evt].length;i++){
            events[evt][i].handler(args);
        }
    }

    //  事件移除
    function off(evt){
        delete events[evt]
    }
    return {
        on: on,
        fire: fire,
        off: off
    }
})();


// 调用
EventCenter.on("my_event", function(data){
```

```
    console.log("my_event被监控");
})
EventCenter.on("my_event2", function(data){
    console.log("my_event2被监控");
})
EventCenter.fire("my_event");
```

# 使用发布订阅模式写一个事件管理器

### Publish_Subscribe_Mode.js

```
var Event = (function() {
    var event = {};

    function on(evt, handler) {
        event[evt] = event[evt] || [];
        event[evt].push({
            handler: handler
        });
    }

    function fire(evt, args) {
        if (!event[evt]) {
            return;
        }
        for (var i = 0; i < event[evt].length; i++) {
            event[evt][i].handler(args);
        }
    }

    function off(evt) {
        event[evt] = [];
        //      console.log(event);
    }
    return {
        on: on,
        fire: fire,
        off: off
    };
})();



Event.on('change', function(val) {
    console.log('change...  now val is ' + val);
});
Event.fire('change', '饥人谷');
Event.off('changer');
Event.fire('change', '饥人谷');
```

## index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>发布订阅模式</title>
</head>
<body>
<script src="Publish_Subscribe_Mode.js"></script>
</body>
</html>
```