私有属性和方法：在函数内部（构造函数），定义的变量和方法；

特权属性和方法：在函数内部，定义的this.xxx=xxx。new以后可以通过实例获得；

共有静态属性和方法：函数名.xxx=xxx，可以通过函数名获得，但不能通过实例获得；

共有属性和方法：函数名.prototype.xxx=xxx，可以通过实例获得。

单体模式：单体是一个用来划分命名空间并将一批相关的属性和方法组织在一起的对象，如果他可以被实例化，那么他只能被实例化一次。

直接用对象字面量：var obj = {}

或者：

```
var functionGroup  =newfunction myGroup(){
    this.name ='Darren';
    this.getName =function(){
        returnthis.name
    }
    this.method1 =function(){}
    ...
}
```

单例模式：单例就是保证一个类只有一个实例，实现的方法一般是先判断实例存在与否，如果存在直接返回，如果不存在就创建了再返回，这就确保了一个类只有一个实例对象。

```
var single = (function(){
    var unique;

    function getInstance(){
       // 如果该实例存在，则直接返回，否则就对其实例化
        if( unique === undefined ){
            unique = new Construct();
        }
        return unique;
    }

    function Construct(){
        // ... 生成单例的构造函数的代码
    }

    return {
        getInstance : getInstance
    }
})();
```

通过single.getInstance()来获取到单例。

工厂模式：不用new，通过工厂方法

简单工厂，抽象工厂（必须先被继承，然后用子类）

桥接模式：用来弱化它与使用它的类和对象之间的耦合，就是将抽象与现实隔离开来。最常见和实际的应用场合之一是事件监听回调函数。在监听和回调函数要做的事情之间做一个桥梁。

装饰者模式：为对象添加功能，寄生继承

var a = {};

a.decorations = {};

a.decorations.add = function(){this.xxx=xxx}

new a.decorations.add()

组合模式：

门面模式：门面模式展现的是一个简化的接口。

```
//_model.util是一个命名空间
_myModel.util.Event = {
    getEvent:function(e){
        return e|| window.event;
    },
```

适配器模式：将一个类的接口转换成客户希望的另外一个接口。适配器模式使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。

观察者：也就是发布订阅模式

定义对象间的一种一对多的依赖关系，以便当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并自动刷新，也被称为是发布订阅模式。

发布订阅模式的流程如下：

1. 确定谁是发布者(比如我的博客)。
2. 然后给发布者添加一个缓存列表，用于存放回调函数来通知订阅者。
3. 发布消息，发布者需要遍历这个缓存列表，依次触发里面存放的订阅者回调函数。
4、退订（比如不想再接收到这些订阅的信息了，就可以取消掉）

```
var pubsub = {};  // 定义发布者

(function (q) {

    var list = [], //回调函数存放的数组，也就是记录有多少人订阅了我们东西
    subUid = -1;

    // 发布消息,遍历订阅者
  q.publish = function (type, content) {
        // type 为文章类型，content为文章内容

    // 如果没有人订阅，直接返回
    if (!list[type]) {

            return false;
        }

        setTimeout(function () {
            var subscribers = list[type],
                len = subscribers ? subscribers.length : 0;

            while (len--) {
                // 将内容注入到订阅者那里
                subscribers[len].func(type, content);
            }
        }, 0);
```

```
            return true;

        };
        //订阅方法，由订阅者来执行
    q.subscribe = function (type, func) {
            // 如果之前没有订阅过
        if (!list[type]) {
                list[type] = [];
            }

            // token相当于订阅者的id，这样的话如果退订，我们就可以针对它来知道
是谁退订了。
        var token = (++subUid).toString();
            // 每订阅一个，就把它存入到我们的数组中去
            list[type].push({
                token: token,
                func: func
            });
            return token;
        };
        //退订方法
    q.unsubscribe = function (token) {
            for (var m in list) {
                if (list[m]) {
                    for (var i = 0, j = list[m].length; i < j; i++) {
                        if (list[m][i].token === token) {
                            list[m].splice(i, 1);
                            return token;
                        }
                    }
                }
            }
            return false;
        };

} (pubsub));
```

```
//将订阅赋值给一个变量，以便退订
var girlA = pubsub.subscribe('js类的文章', function (type, content) {
    console.log('girlA订阅的'+type + ": 内容内容为：" + content);
});
var girlB = pubsub.subscribe('js类的文章', function (type, content) {
    console.log('girlB订阅的'+type + ": 内容内容为：" + content);
});
var girlC = pubsub.subscribe('js类的文章', function (type, content) {
    console.log('girlC订阅的'+type + ": 内容内容为：" + content);
});

//发布通知
pubsub.publish('js类的文章', '关于js的内容');
// 输出:
// girlC订阅的js类的文章： 内容内容为：关于js的内容
// test3.html:78 girlB订阅的js类的文章： 内容内容为：关于js的内容
// test3.html:75 girlA订阅的js类的文章： 内容内容为：关于js的内容


//girlA退订了关于js类的文章
setTimeout(function () {
    pubsub.unsubscribe(girlA);
}, 0);

//再发布一次，验证一下是否还能够输出信息
pubsub.publish('js类的文章', "关于js的第二篇文章");
// 输出:
// girlB订阅的js类的文章： 内容内容为：关于js的第二篇文章
// girlC订阅的js类的文章： 内容内容为：关于js的第二篇文章
```