对象

基本格式

注意每条属性或者方法后面都有逗号！！！

这是对象！！！

```
var mrDeng = {
    name : "MrDeng",
    age : 40,
    sex : "male",
    health : 100,
    smoke : function (){        //注意对象中的语句结尾都是逗号！！！
        console.log('I am smoking !!! cool');
        this.health --;
    },
    drink : function (){
        console.log('I am drinking');
        this.health ++;
    }
}
```

谁调用this方法this就指向谁！

增删改查

```
mrDeng.name          //查
mrDeng.smoke();      //查
mrDeng.name++        //改
mrDeng.age = 30      //改
mrDeng.wife = 'xiaowang'   //增
//删
delete mrDeng.name
```

对象的构造方法

除了 var obj {} 之外

可以var obj = new object();（系统自带的构造函数）（也可以自己构造，形式和函数形式相同，首字母大写以区别普通函数）

这是类！！！

```
function Car(color){
    //var this = {
    //    __proto__:Person.prototype
    //};
    //在内部执行的操作
```

```
    //new之后执行的操作！！！
    this.color = color;
    this.name = "BMW";
    this.height = "1400"
    this.weight = 1000;
    this.health = 100;
    this.run = function(){
        health--;
    }
    //不能返回｛｝相当于捣乱系统机制，
    //而可以返回对象值，eg：123，数组，function
    //return this
}
var car = new Car('green');
var car = new Car('red');
//我感觉跟类差不多
```

原始值 包装类：

1. 为了便于操作“基本类型值”，JS 提供了 三个 特殊的引用类型：Boolean、Number、String。这些类型和其他引用类型相似，但同时 也具备 与各自基本类型相应的特殊行为。 实际上：每当读取一个基本类型值的时候， “后台就会创建一个对应的基本包装类型的对象”，从能能够调用一些方法来操作这些数据。
2. 引用类型和基本包装类型的主要区别就是对象的生存期;
3. 自动创建的基本包装类型的对象,则只存在于一行代码的执行瞬间,然后立即被销毁;

这意味着我们不能在运行时为基本类型值添加属性和方法

```
var arr = [1,2,3,4,5];
arr.length = 2;
console.log(arr);       //[1,2]
var str = ('asdfg');
arr.length = 2;
//new String('asdfg').length = 2;delete
//new String('asdfg')    //新的
console.log(str.length);       //4
//其实第五行没做任何改变
//str num
```

eg:

```
var str = "abc";
str += 1;       //str = "abc1"
var test = typeof(str);   //String
if (test.length == 6){    //True
        test.sign = "typeof的返回结果可能为String";
        //new String(test).sign = 'xxx' delete
```

```
}
//new String(test).sign
console.log(test.sign);


function Person(name,age,sex) {
    //var this = {}
    var a = 0;
    this.name = name;
    this.age = age;
    this.sex = sex;
    function sss() {
        a++;
        document.write(a);
    }
    this.say = sss;
    //return this
    //返回了函数，形成闭包
}
var oPerson = new Person();
oPerson.say();
oPerson.say();
var oPerson1 = new Person();
oPerson1.say();
```

打印结果 ： 121

```
var x = 1;
var y = z = 0;
function add(){
    return n = n + 1;
}
y = add(x);
function add(){
    return n = n + 3;
}
z = add(x);
//求x,y,z
//结果为 1 4 4
//后定义的函数会覆盖前面的
//从作用域中执行
```