# JavaScript基础之对象与内置对象(三)

## ✍JavaScript帝国之行🔥

| 内容 | 地址 |
| --- | --- |
| JavaScript基础大总结(一) 🔥 | https://blog.csdn.net/Augenstern_QXL/article/details/119249534 |
| JavaScript基础之函数与作用域(二) 🔥 | https://blog.csdn.net/Augenstern_QXL/article/details/119250991 |
| JavaScript基础之对象与内置对象(三) 🔥 | https://blog.csdn.net/Augenstern_QXL/article/details/119250137 |
| JavaScript进阶之DOM技术(四) 🔥 | https://blog.csdn.net/Augenstern_QXL/article/details/115416921 |
| JavaScript进阶之BOM技术(五) 🔥 | https://blog.csdn.net/Augenstern_QXL/article/details/115406408 |
| JavaScript提高之面向对象(六) 🔥 | https://blog.csdn.net/Augenstern_QXL/article/details/115219073 |
| JavaScript提高之ES6(七) 🔥 | https://blog.csdn.net/Augenstern_QXL/article/details/115344398 |

## ✍目录总览


## 1、对象🔥

在 JavaScript 中，对象是一组无序的相关属性和方法的集合，所有的事物都是对象，例如字符串、数值、数组、函数等。

对象是由属性和方法组成的：

- 属性：事物的**特征**，**在对象中用属性**来表示（常用名词）
- 方法：事物的**行为**，**在对象中用方法**来表示（常用动词）

## 1.1、创建对象🔥

在 JavaScript 中，现阶段我们可以采用三种方式创建对象（object）：

- 利用字面量创建对象
- 利用 new Object创建对象
- 利用构造函数创建对象

## ①利用字面量创建对象🔥

对象字面量：就是花括号 `{ }` 里面包含了表达这个具体事物（对象）的属性和方法

`{ }` 里面采取键值对的形式表示

- 键：相当于属性名
- 值：相当于属性值，可以是任意类型的值（数字类型、字符串类型、布尔类型，函数类型等）

```
var star = {
    name : 'pink',
    age : 18,
    sex : '男',
    sayHi : function(){
        alert('大家好啊~');
    }
};
// 多个属性或者方法中间用逗号隔开
// 方法冒号后面跟的是一个匿名函数
```

### 🔥对象的调用

- 对象里面的属性调用 : **对象.属性名** ， 这个小点 . 就理解为“ **的** ”
- 对象里面属性的另一种调用方式 : **对象['属性名']**，注意方括号里面的属性必须**加引号**，我们后面会用
- 对象里面的方法调用：**对象.方法名()** ，注意这个方法名字后面**一定加括号**

```
console.log(star.name)      // 调用名字属性
console.log(star['name'])   // 调用名字属性
star.sayHi();               // 调用 sayHi 方法,注意，一定不要忘记带后面的括号
```

### 🔥变量、属性、函数、方法总结

- 变量：单独声明赋值，单独存在
- 属性：对象里面的变量称为属性，不需要声明，用来描述该对象的特征
- 函数：单独存在的，通过==“函数名()”==的方式就可以调用
- 方法：对象里面的函数称为方法，方法不需要声明，使用==“对象.方法名()”==的方式就可以调用，方法用来描述该对象的行为和功能。

## ②利用 new Object 创建对象 🔥

跟之前的 `new Array()` 原理一致： `var 对象名 = new Object();`

使用的格式：**对象.属性 = 值**

```
var obj = new Object(); //创建了一个空的对象
obj.name = '张三丰';
obj.age = 18;
obj.sex = '男';
obj.sayHi = function() {
    console.log('hi~');
}

//1.我们是利用等号赋值的方法添加对象
//2.每个属性和方法之间用分号结束
console.log(obj.name);
console.log(obj['sex']);
obj.sayHi();
```

## ③利用构造函数创建对象 🔥

**构造函数** ： 是一种特殊的函数，主要用来初始化对象，**即为对象成员变量赋初始值**，它总与 new 运算符一起使用。我们可以把对象中一些公共的属性和方法抽取出来，然后封装到这个函数里面。

在 js 中，使用构造函数要时要注意以下两点：

- 构造函数用于创建某一类对象，其**首字母要大写**
- 构造函数要和 new **一起使用**才有意义

```
//构造函数的语法格式
function 构造函数名() {
```

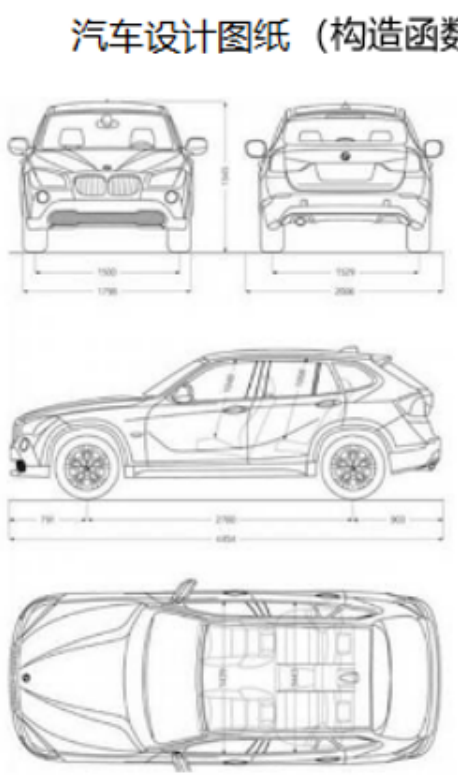
```
    this.属性 = 值;
    this.方法 = function() {}
}
new 构造函数名();
```

```
//1. 构造函数名字首字母要大写
//2. 构造函数不需要return就可以返回结果
//3. 调用构造函数必须使用 new
//4. 我们只要new Star() 调用函数就创建了一个对象
//5. 我们的属性和方法前面必须加this
function Star(uname,age,sex) {
    this.name = uname;
    this.age = age;
    this.sex = sex;
    this.sing = function(sang){
        console.log(sang);
    }
}
var ldh = new Star('刘德华',18,'男');
console.log(typeof ldh) // object对象，调用函数返回的是对象

console.log(ldh.name);
console.log(ldh['sex']);
ldh.sing('冰雨');
//把冰雨传给了sang

var zxy = new Star('张学友',19,'男');
```

- 构造函数名字首字母要大写
- 函数内的属性和方法前面需要添加 this ，表示当前对象的属性和方法。
- 构造函数中不需要 return 返回结果。
- 当我们创建对象的时候，必须用 new 来调用构造函数。

### 构造函数和对象

- 构造函数，如 Stars()，抽象了对象的公共部分，封装到了函数里面，它泛指某一大类（class）
- 创建对象，如 new Stars()，特指某一个，通过 new 关键字创建对象的过程我们也称为对象实例化







### new关键字

new 在执行时会做四件事:

1. 在内存中创建一个新的空对象。
2. 让 this 指向这个新的对象。
3. 执行构造函数里面的代码，给这个新对象添加属性和方法
4. 返回这个新对象（所以构造函数里面不需要return）

## 1.2、遍历对象的属性

- `for...in` 语句用于对数组或者对象的属性进行循环操作

语法如下

```
for(变量 in 对象名字){
    // 在此执行代码
}
```

语法中的变量是自定义的，它需要符合命名规范，通常我们会将这个变量写为 k 或者 key。

```
for(var k in obj) {
    console.log(k);     //这里的 k 是属性名
    console.log(obj[k]);//这里的 obj[k] 是属性值
}
```

```
var obj = {
    name: '秦sir',
    age: 18,
    sex: '男',
    fn:function() {};
};
console.log(obj.name);
console.log(obj.age);
console.log(obj.sex);

//for in 遍历我们的对象
//for (变量 in 对象){}
//我们使用for in 里面的变量 我们喜欢写k 或者key
for(var k in obj){
    console.log(k); // k 变量 输出得到的是属性名
    console.log(obj[k]); // obj[k] 得到的是属性值
}
```

## 2、内置对象 🔥

- JavaScript 中的对象分为3种：自定义对象 、内置对象、 浏览器对象
- 内置对象就是指 JS 语言自带的一些对象，这些对象供开发者使用，并提供了一些常用的或是最基本而必要的功能
- JavaScript 提供了多个内置对象：Math、 Date 、Array、String等

### 2.1、查文档

学习一个内置对象的使用，只要学会其常用成员的使用即可，我们可以通过查文档学习，可以通过MDN/W3C来查询

MDN: https://developer.mozilla.org/zh-CN/

#### 2.1.1、如何学习对象中的方法

1. 查阅该方法的功能
2. 查看里面参数的意义和类型
3. 查看返回值的意义和类型
4. 通过 demo 进行测试

## 3、Math对象 🔥

Math 对象不是构造函数，它具有数学常数和函数的属性和方法。跟数学相关的运算（求绝对值，取整、最大值等）可以使用 Math 中的成员。

```
// Math数学对象，不是一个构造函数，所以我们不需要new 来调用，而是直接使用里面的属性和方法即可

Math.PI                 // 圆周率
Math.floor()            // 向下取整
Math.ceil()             // 向上取整
Math.round()            // 四舍五入版 就近取整    注意 -3.5    结果是  -3
Math.abs()              // 绝对值
Math.max()/Math.min()   // 求最大和最小值
```

**注意**：上面的方法必须带括号

```
console.log(Math.PI);
console.log(Math.max(1,99,3)); // 99
```

**练习：封装自己的数学对象**

利用对象封装自己的数学对象，里面有PI 最大值 和最小值

```
var myMath = {
    PI: 3.141592653,
    max: function() {
        var max = arguments[0];
        for (var i = 1; i < arguments.length; i++) {
            if (arguments[i] > max) {
                max = arguments[i];
            }
        }
        return max;
    },
    min: function() {
        var min = arguments[0];
        for (var i = 1; i < arguments.length; i++) {
            if (arguments[i] < min) {
                min = arguments[i];
            }
        }
        return min;
    }
}
console.log(myMath.PI);
console.log(myMath.max(1, 5, 9));
console.log(myMath.min(1, 5, 9));
```

## 3.Math绝对值和三个取整方法 🔥

- `Math.abs()` 取绝对值
- 三个取整方法:
  - `Math.floor()` ：向下取整
  - `Math.ceil()` ：向上取整
  - `Matg.round()` ：四舍五入，其他数字都是四舍五入，但是5特殊，它往大了取

```
//1.绝对值方法
console.log(Math.abs(1));  // 1
console.log(Math.abs(-1)); // 1
console.log(Math.abs('-1')); // 1 隐式转换，会把字符串 -1 转换为数字型
//2.三个取整方法
console.log(Math.floor(1.1)); // 1 向下取整，向最小的取值 floor-地板
console.log(Math.floor(1.9)); //1

console.log(Math.ceil(1.1)); //2 向上取整，向最大的取值 ceil-天花板
console.log(Math.ceil(1.9)); //2

//四舍五入 其他数字都是四舍五入，但是5特殊，它往大了取

console.log(Math.round(1.1)); //1 四舍五入
console.log(Math.round(1.5)); //2
console.log(Math.round(1.9)); //2
console.log(Math.round(-1.1)); // -1
console.log(Math.round(-1.5)); // -1
```

## 4.随机数方法random() 🔥

- random() 方法可以随机返回一个小数，其取值范围是 [0，1)，左闭右开 0 <= x < 1
- 得到一个两数之间的随机整数，包括第一个数，不包括第二个数

```
// 得到两个数之间的随机整数，并且包含这两个整数
function getRandom(min,max) {
    return Math.floor(Math.random() * (max - min + 1)) + min;
}
console.log(getRandom(1,10));
```

### 1.随机点名

```
var arr = ['张三', '李四','王五','秦六'];
console.log(arr[getRandom(0,arr.length - 1)]);
```

### 2.猜数字游戏

**案例：猜数字游戏**

程序随机生成一个 1~ 10 之间的数字，并让用户输入一个数字，

1. 如果大于该数字，就提示，数字大了，继续猜；
2. 如果小于该数字，就提示数字小了，继续猜；
3. 如果等于该数字，就提示猜对了， 结束程序。

```
function getRandom(min,max) {
    return Math.floor(Math.random() * (max - min + 1)) + min;
}
var random = getRandom(1,10);
while(true) { //死循环 ， 需要退出循环条件
     var num = prompt('请输入1~10之间的一个整数:');
     if(num > random) {
         alert('你猜大了');
     }else if (num < random) {
         alert('你猜小了');
     }else {
         alert('你猜中了');
         break; //退出整个循环
     }
}
```

## 4、Data()日期对象 🔥

- Date 对象和 Math 对象不一样，他是一个构造函数，所以我们需要实例化后才能使用
- Date 实例用来处理日期和时间

## 4.1、Date()方法的使用 🔥

### 4.1.1、获取当前时间必须实例化 🔥

```
var now = new Date();
console.log(now);
```

### 4.1.2、Date()构造函数的参数 🔥

如果括号里面有时间，就返回参数里面的时间。例如日期格式字符串为 `‘2019-5-1’` ，可以写成 `new Date('2019-5-1')` 或者 `new Date('2019/5/1')`

- 如果Date()不写参数，就返回当前时间
- 如果Date()里面写参数，就返回括号里面输入的时间

```
// 1.如果没有参数，返回当前系统的当前时间
var now = new Date();
console.log(now);


// 2.参数常用的写法 数字型 2019,10,1   字符串型 '2019-10-1 8:8:8' 时分秒
// 如果Date()里面写参数，就返回括号里面输入的时间
var data = new Date(2019,10,1);
console.log(data);  // 返回的是11月不是10月

var data2 = new Date('2019-10-1 8:8:8');
console.log(data2);
```

## 4.2、日期格式化 🔥

我们想要 2019-8-8 8:8:8 格式的日期，要怎么办?

需要获取日期指定的部分，所以我们要手动的得到这种格式

| 方法名 | 说明 | 代码 |
| --- | --- | --- |
| getFullYear() | 获取当年 | dObj.getFullYear() |
| getMonth() | 获取当月(0-11) | dObj.getMonth() |
| getDate() | 获取当天日期 | dObj.getDate() |
| getDay() | 获取星期几(周日0到周六6) | dObj.getDay() |
| getHours() | 获取当前小时 | dObj.getHours() |
| getMinutes() | 获取当前小时 | dObj.getMinutes() |
| getSeconds() | 获取当前秒钟 | dObj.gerSeconds() |

```
var date = new Date();
console.log(date.getFullYear()); // 返回当前日期的年 2019
console.log(date.getMonth() + 1);  //返回的月份小一个月 记得月份 +1
console.log(date.getDate); //返回的是几号
console.log(date.getDay());  //周一返回1 周6返回六 周日返回0



// 写一个 2019年 5月 1日 星期三
var date = new Date();
var year =  date.getFullYear();
var month = date.getMonth() + 1;
var dates = date.getDate();
console.log('今天是' + year +'年' + month + '月' + dates +'日' );

// 封装一个函数返回当前的时分秒 格式 08:08:08
function getTimer() {
    var time = new Date();
    var h = time.getHours();
    h = h < 10 ? '0' + h : h;
    var m = time.getMinutes();
    m = m < 10 ? '0' + m : m;
    var s = time.getSeconds();
    s = s < 10 ? '0' + s : s;
    return h + ':' + m + ':' + s;
}
console.log(getTimer());
```

## 4.3、获取日期的总的毫秒形式 🔥

- `date.valueOf()` ： 得到现在时间距离1970.1.1总的毫秒数
- `date.getTime()` ： 得到现在时间距离1970.1.1总的毫秒数

```
// 获取Date总的毫秒数 不是当前时间的毫秒数 而是距离1970年1月1号过了多少毫秒数

// 实例化Date对象
var date = new Date();

// 1 .通过 valueOf()  getTime() 用于获取对象的原始值
console.log(date.valueOf());  //得到现在时间距离1970.1.1总的毫秒数
console.log(date.getTime());

// 2.简单的写法
var date1 = +new Date();  // +new Date()返回的就是总的毫秒数,
console.log(date1);

// 3. HTML5中提供的方法 获得总的毫秒数 有兼容性问题
console.log(Date.now());
```

## 🔥倒计时效果

做一个倒计时效果




```
function countDown(time) {
    var nowTime = +new Date(); //没有参数，返回的是当前时间总的毫秒数
    var inputTime = +new Date(time); // 有参数，返回的是用户输入时间的总毫秒数
    var times = (inputTime - nowTime) / 1000; //times就是剩余时间的总的秒数

    var d = parseInt(times / 60 / 60 / 24); //天数
    d < 10 ? '0' + d : d;
    var h = parseInt(times / 60 / 60 % 24); //小时
    h < 10 ? '0' + h : h;
    var m = parseInt(times / 60 % 60); //分
    m < 10 ? '0' + m : m;
    var s = parseInt(times % 60); //秒
    s < 10 ? '0' + s : s;
    return d + '天' + h + '时' + m + '分' + s + '秒';
}
console.log(countDown('2020-11-09 18:29:00'));
var date = new Date;
console.log(date); //现在时间
```

# 5、数组对象 🔥

## 5.1、数组对象的创建 🔥

创建数组对象的两种方式

- 字面量方式
- new Array()

## 5.2、检测是否为数组 🔥

- instanceof 运算符，可以判断一个对象是否属于某种类型
- `Array.isArray()` 用于判断一个对象是否为数组，isArray() 是 HTML5 中提供的方法

```
var arr = [1, 23];
var obj = {};
console.log(arr instanceof Array); // true
console.log(obj instanceof Array); // false
console.log(Array.isArray(arr));   // true
console.log(Array.isArray(obj));   // false
```

## 5.3、添加删除数组元素 🔥

| 方法名 | 说明 | 返回值 |
| --- | --- | --- |
| push(参数1…) | 末尾添加一个或多个元素，注意修改原数组 | 并返回新的长度 |
| pop() | 删除数组最后一个元素 | 返回它删除的元素的值 |
| unshift(参数1…) | 向数组的开头添加一个或更多元素，注意修改原数组 | 并返回新的长度 |
| shift() | 删除数组的第一个元素，数组长度减1，无参数，修改原数组 | 并返回第一个元素 |

```
// 1.push() 在我们数组的末尾，添加一个或者多个数组元素 push 推
var arr = [1, 2, 3];
arr.push(4, '秦晓');
console.log(arr);
console.log(arr.push(4, '秦晓'));
console.log(arr);
// push 完毕之后，返回结果是新数组的长度


// 2. unshift 在我们数组的开头 添加一个或者多个数组元素
arr.unshift('red');
console.log(arr);

// pop() 它可以删除数组的最后一个元素，一次只能删除一个元素
arr.pop(); //不加参数
// shift() 它剋删除数组的第一个元素,一次只能删除一个元素
arr.shift(); //不加参数
```

### 👉筛选数组

有一个包含工资的数组[1500,1200,2000,2100,1800],要求把数组中工资超过2000的删除，剩余的放到新数组里面

```
var arr = [1500, 1200, 2000, 2100, 1800];
var newArr = [];
for (var i = 0; i < arr.length; i++) {
    if (arr[i] < 2000) {
        newArr.push(arr[i]);
    }
}
console.log(newArr);

```

## 5.4、数组排序🔥

| 方法名 | 说明 | 是否修改原数组 |
| --- | --- | --- |
| reverse() | 颠倒数组中元素的顺序，无参数 | 该方法会改变原来的数组，返回新数组 |
| sort() | 对数组的元素进行排序 | 该方法会改变原来的数组，返回新数组 |

```
// 1.翻转数组
var arr = ['pink','red','blue'];
arr.reverse();
console.log(arr);

// 2.数组排序(冒泡排序)
var arr1 = [3,4,7,1];
arr1.sort();
console.log(arr1);

// 对于双位数
var arr = [1,64,9,61];
arr.sort(function(a,b) {
     return b - a;  //降序的排列
     return a - b; //升序
    }
 )
```

## 5.5、数组索引🔥

| 方法名 | 说明 | 返回值 |
| --- | --- | --- |
| indexOf() | 数组中查找给定元素的第一个索引 | 如果存在返回索引号，如果不存在，则返回-1 |
| lastIndexOf() | 在数组的最后一个索引，从后向前索引 | 如果存在返回索引号，如果不存在，则返回-1 |

```
//返回数组元素索引号方法 indexOf(数组元素)  作用就是返回该数组元素的索引号
//它只发返回第一个满足条件的索引号
//如果找不到元素，则返回-1
var arr = ['red','green','blue','pink','blue'];
console.log(arr.indexOf('blue'));  // 2

console.log(arr.lastIndexOf('blue')); // 4
```

### 5.5.1、🔥数组去重

**案例： 数组去重（重点案例）**

有一个数组['c', 'a', 'z', 'a', 'x', 'a', 'x', 'c', 'b']，要求去除数组中重复的元素。

分析：把旧数组里面不重复的元素选取出来放到新数组中，重复的元素只保留一个，放到新数组中去重。

核心算法：我们遍历旧数组，然后拿着旧数组元素去查询新数组，如果该元素在新数组里面没有出现过，我们就添加，否则不添加。

我们怎么知道该元素没有存在？ 利用 新数组.indexOf(数组元素) 如果返回是 -1 就说明 新数组里面没有改元素

```
// 封装一个去重的函数 unique 独一无二的
function unique(arr) {
    var newArr = [];
    for (var i = 0; i < arr.length; i++) {
        if (newArr.indexOf(arr[i]) === -1) {
            newArr.push(arr[i]);
        }
    }
    return newArr;
}
var demo = unique(['c', 'a', 'z', 'a', 'x', 'a', 'x', 'c', 'b']);
console.log(demo);
```

## 5.6、数组转化为字符串🔥

| 方法名 | 说明 | 返回值 |
| --- | --- | --- |
| toString() | 把数组转换成字符串，逗号分隔每一项 | 返回一个字符串 |
| join('分隔符') | 方法用于把数组中的所有元素转换为一个字符串 | 返回一个字符串 |

```
// 1.toString() 将我们的数组转换为字符串
var arr = [1, 2, 3];
console.log(arr.toString()); // 1,2,3
// 2.join('分隔符')
var arr1 = ['green', 'blue', 'red'];
console.log(arr1.join()); // 不写默认用逗号分割
console.log(arr1.join('-')); //  green-blue-red
console.log(arr1.join('&')); // green&blue&red
```

## 5.7、其他方法

| 方法名 | 说明 | 返回值 |
| --- | --- | --- |
| concat() | 连接两个或多个数组 不影响原数组 | 返回一个新的数组 |
| slice() | 数组截取slice(begin,end) | 返回被截取项目的新数组 |
| splice() | 数组删除splice(第几个开始要删除的个数) | 返回被删除项目的新数组，这个会影响原数组 |

# 6、字符串对象🔥

## 6.1、基本包装类型🔥

为了方便操作基本数据类型，JavaScript 还提供了三个特殊的引用类型：String、Number和 Boolean。

**基本包装类型**就是把简单数据类型包装成为复杂数据类型，这样基本数据类型就有了属性和方法。

我们看看下面代码有什么问题吗?

```
var str = 'andy';
console.log(str.length);
```

按道理基本数据类型是没有属性和方法的，而对象才有属性和方法，但上面代码却可以执行，这是因为 js 会把基本数据类型包装为复杂数据类型，其执行过程如下：

```
// 1.生成临时变量,把简单类型包装为复杂数据类型
var temp = new String('andy');
```

```
// 2.赋值给我们声明的字符变量
str = temp;
// 3.销毁临时变量
temp = null;
```

## 6.2、字符串的不可变 🔥

指的是里面的值不可变，虽然看上去可以改变内容，但其实是地址变了，内存中新开辟了一个内存空间。

```
var str = 'abc';
str = 'hello';
// 当重新给 str 赋值的时候，常量'abc'不会被修改，依然在内存中
// 重新给字符串赋值，会重新在内存中开辟空间，这个特点就是字符串的不可变
// 由于字符串的不可变，在大量拼接字符串的时候会有效率问题
var str = '';
for(var i = 0; i < 10000;  i++){
    str += i;
}
console.log(str);
// 这个结果需要花费大量时间来显示，因为需要不断的开辟新的空间
```

## 6.3、根据字符返回位置 🔥

字符串所有的方法，都不会修改字符串本身(字符串是不可变的)，**操作完成会返回一个新的字符串**

| 方法名 | 说明 |
| --- | --- |
| indexOf('要查找的字符'，开始的位置) | 返回指定内容在元字符串中的位置，如果找不到就返回-1，开始的位置是index索引号 |
| lastIndexOf() | 从后往前找，只找第一个匹配的 |

```
// 字符串对象  根据字符返回位置  str.indexOf('要查找的字符', [起始的位置])
        var str = '改革春风吹满地，春天来了';
        console.log(str.indexOf('春')); //默认从0开始查找 ，结果为2
        console.log(str.indexOf('春', 3)); // 从索引号是 3的位置开始往后查找，结果是8
```

### 6.3.1、返回字符位置 🔥

查找字符串 “abcoefoxyozzopp” 中所有o出现的位置以及次数

- 核心算法：先查找第一个o出现的位置
- 然后 只要 indexOf返回的结果不是 -1 就继续往后查找
- 因为 indexOf 只能查找到第一个，所以后面的查找，一定是当前索引加1，从而继续查找

```
var str = "oabcoefoxyozzopp";
var index = str.indexOf('o');
var num = 0;
// console.log(index);
while (index !== -1) {
    console.log(index);
    num++;
    index = str.indexOf('o', index + 1);
}
console.log('o出现的次数是: ' + num);
```

## 6.4、根据位置返回字符 🔥

| 方法名 | 说明 | 使用 |
| --- | --- | --- |
| charAt(index) | 返回指定位置的字符(index字符串的索引号) | str.charAt(0) |
| charCodeAt(index) | 获取指定位置处字符的ASCII码(index索引号) | str.charCodeAt(0) |
| str[index] | 获取指定位置处字符 | HTML,IE8+支持和charAt()等效 |

### 🔥返回字符位置

判断一个字符串 “abcoefoxyozzopp” 中出现次数最多的字符，并统计其次数

- 核心算法：利用 charAt() 遍历这个字符串
- 把每个字符都存储给对象， 如果对象没有该属性，就为1，如果存在了就 +1

- 遍历对象，得到最大值和该字符

```
<script>
    // 有一个对象 来判断是否有该属性 对象['属性名']
    var o = {
        age: 18
    }
    if (o['sex']) {
        console.log('里面有该属性');

    } else {
        console.log('没有该属性');

    }

    //  判断一个字符串 'abcoefoxyozzopp' 中出现次数最多的字符，并统计其次数。
    // o.a = 1
    // o.b = 1
    // o.c = 1
    // o.o = 4
    // 核心算法：利用 charAt() 遍历这个字符串
    // 把每个字符都存储给对象， 如果对象没有该属性，就为1，如果存在了就 +1
    // 遍历对象，得到最大值和该字符
    var str = 'abcoefoxyozzopp';
    var o = {};
    for (var i = 0; i < str.length; i++) {
        var chars = str.charAt(i); // chars 是 字符串的每一个字符
        if (o[chars]) { // o[chars] 得到的是属性值
            o[chars]++;
        } else {
            o[chars] = 1;
        }
    }
    console.log(o);
    // 2. 遍历对象
    var max = 0;
    var ch = '';
    for (var k in o) {
        // k 得到是 属性名
        // o[k] 得到的是属性值
        if (o[k] > max) {
            max = o[k];
            ch = k;
        }
    }
    console.log(max);
    console.log('最多的字符是' + ch);
</script>
```

## 6.5、字符串操作方法🔥

| 方法名 | 说明 |
| --- | --- |
| concat(str1,str2,str3…)🔥 | concat() 方法用于连接两个或对各字符串。拼接字符串🔥 |
| substr(start,length)🔥 | 从 start 位置开始(索引号), length 取的个数。🔥 |
| slice(start,end) | 从 start 位置开始，截取到 end 位置， end 取不到 (两个都是索引号) |
| substring(start,end) | 从 start 位置开始，截取到 end 位置， end 取不到 (基本和 slice 相同，但是不接受负) |

```
<script>
    // 1. concat('字符串1','字符串2'....)
    var str = 'andy';
    console.log(str.concat('red'));

    // 2. substr('截取的起始位置', '截取几个字符');
    var str1 = '改革春风吹满地';
    console.log(str1.substr(2, 2)); // 第一个2 是索引号的2    第二个2 是取几个字符
</script>
```

## 6.6、replace()方法🔥

replace() 方法用于在字符串中用一些字符替换另一些字符

其使用格式： `replace(被替换的字符,要替换为的字符串)`

```
<script>
    // 1. 替换字符 replace('被替换的字符', '替换为的字符')  它只会替换第一个字符
    var str = 'andyandy';
    console.log(str.replace('a', 'b'));
    // 有一个字符串 'abcoefoxyozzopp'  要求把里面所有的 o 替换为 *
    var str1 = 'abcoefoxyozzopp';
    while (str1.indexOf('o') !== -1) {
        str1 = str1.replace('o', '*');
    }
    console.log(str1);
</script>
```

## 6.7、split()方法 🔥

split() 方法用于切分字符串，它可以将字符串切分为数组。在切分完毕之后，返回的是一个新数组。

例如下面代码：

```
var str = 'a,b,c,d';
console.log(str.split(','));
// 返回的是一个数组 ['a', 'b', 'c', 'd']
```

```
<script>
// 2. 字符转换为数组 split('分隔符')    前面我们学过 join 把数组转换为字符串
    var str2 = 'red, pink, blue';
    console.log(str2.split(','));
    var str3 = 'red&pink&blue';
    console.log(str3.split('&'));
</script>
```

## 6.8、大小写转换

- `toUpperCase()` 转换大写
- `toLowerCase()` 转换小写

# 7、简单类型于复杂类型 🔥

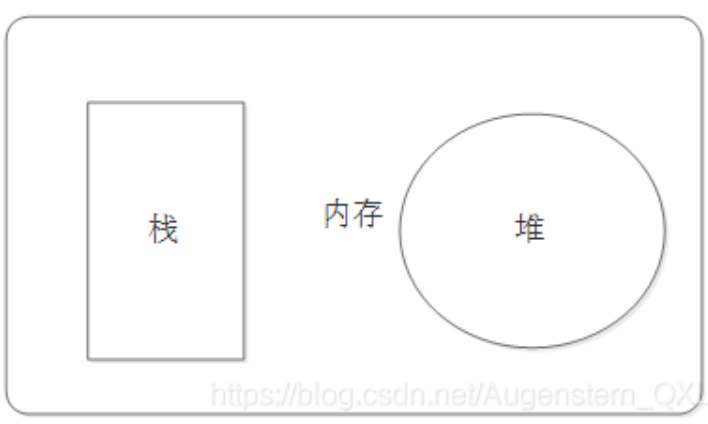
简单类型又叫做基本数据类型或者**值类型**，复杂类型又叫做**引用类型**。

- 值类型：简单数据类型/基本数据类型，在存储时变量中存储的是值本身，因此叫做值类型
  - string ，number，boolean，undefined，null
- 引用类型：复杂数据类型，在存储时变量中存储的仅仅是地址（引用），因此叫做引用数据类型
  - 通过 new 关键字创建的对象（系统对象、自定义对象），如 Object、Array、Date等

## 7.1、堆和栈 🔥

堆栈空间分配区别：

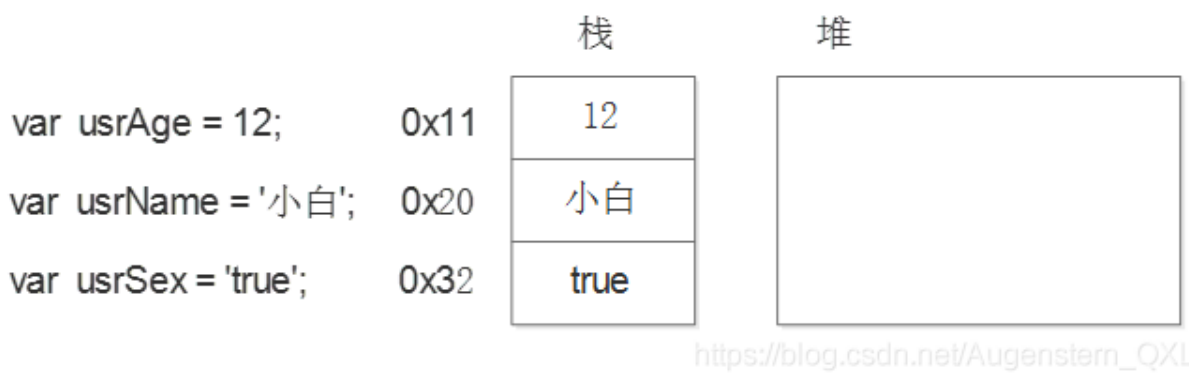
1. 栈（操作系统）：由操作系统自动分配释放存放函数的参数值、局部变量的值等。其操作方式类似于数据结构中的栈；
   - **简单数据类型存放到栈里面**
2. 堆（操作系统）：存储复杂类型(对象)，一般由程序员分配释放，若程序员不释放，由垃圾回收机制回收。
   - **复杂数据类型存放到堆里面**




注意：JavaScript中没有堆栈的概念，通过堆栈的方式，可以让大家更容易理解代码的一些执行方式，便于将来学习其他语言。

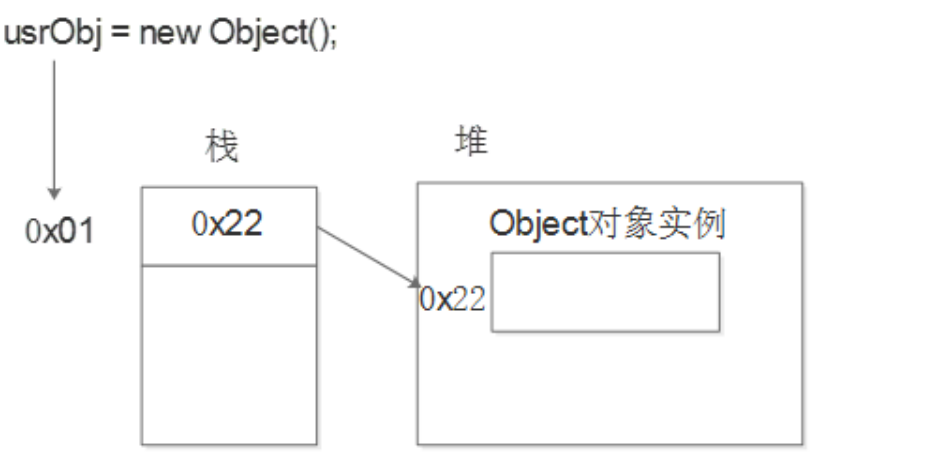
## 7.2、简单类型的内存分配🔥

- 值类型（简单数据类型）： string ，number，boolean，undefined，null
- 值类型变量的数据直接存放在变量（栈空间）中





- 引用类型（复杂数据类型）： 通过 new 关键字创建的对象（系统对象、自定义对象），如 Object、Array、Date等
- 引用类型变量（栈空间）里存放的是地址，真正的对象实例存放在堆空间中





```
<script>
    // 简单数据类型 null  返回的是一个空的对象  object
    var timer = null;
    console.log(typeof timer);
    // 如果有个变量我们以后打算存储为对象，暂时没想好放啥， 这个时候就给 null
    // 1. 简单数据类型 是存放在栈里面 里面直接开辟一个空间存放的是值
    // 2. 复杂数据类型 首先在栈里面存放地址 十六进制表示  然后这个地址指向堆里面的数据
</script>
```

## 7.3、简单类型传参🔥

函数的形参也可以看做是一个变量，当我们把一个值类型变量作为参数传给函数的形参时，其实是把变量在栈空间里的值复制了一份给形参，那么在方法内部对形参做任何修改，都不会影响到的外部变量。

```
<script>
    // 简单数据类型传参
    function fn(a) {
        a++;
        console.log(a);
    }
    var x = 10;
    fn(x);
    console.log(x);
</script>
```

## 7.4、复杂类型传参🔥

函数的形参也可以看做是一个变量，当我们把引用类型变量传给形参时，其实是把变量在栈空间里保存的堆地址复制给了形参，形参和实参其实保存的是同一个堆地址，所以操作的是同一个对象。

```
<script>
    // 复杂数据类型传参
    function Person(name) {
        this.name = name;
    }

    function f1(x) { // x = p
        console.log(x.name); // 2. 这个输出什么 ？  刘德华
        x.name = "张学友";
```

```
        console.log(x.name); // 3. 这个输出什么 ?   张学友
    }
    var p = new Person("刘德华");
    console.log(p.name); // 1. 这个输出什么 ?   刘德华
    f1(p);
    console.log(p.name); // 4. 这个输出什么 ?   张学友
</script>
```