# [JavaScript]

老马-经典前端教程-JavaScript]

谦太祥和（北京）科技有限公司

# 1. 联系老马

老马联系方式

QQ：515154084

邮箱：malun666@126.com

微博：http://weibo.com/flydragon2010

百度传课：

https://chuanke.baidu.com/s5508922.html

微信：


# 2. 认识 JavaScript

## 2.1 JavaScript 是什么？

JavaScript 是世界上用的最多的脚本语言。脚本语言：不需要编译，直接运行时边解析边执行的语言。

它可以运行在浏览器端、服务器端、移动端上进行各种编程操作。

## 2.2 JavaScript 的历史(了解)

1995.2 月 Netscape 公司发布 LiveScript，后临时改为 JavaScript，为了赶上 Java 的热浪。

欧洲计算机制造商协会（ECMA）英文名称是 European Computer Manufacturers Association

1997 年，以 JavaScript 1.1 为基础。由来自 Netscape、Sun、微软、Borland 和其他一些对脚本编程感兴趣的公司的程序员组成的 TC39（ECMA 的小组） 锤炼出了 ECMA-262，也就是 ECMAScript1.0。

1998 年 6 月，ECMAScript 2.0 版发布。

1999 年 12 月，ECMAScript 3.0 版发布，成为 JavaScript 的通行标准，得到了广泛支持。

2007 年 10 月，ECMAScript 4.0 版草案发布：分歧太大，失败告终。
2009 年 12 月，ECMAScript 5.0 版正式发布
2015 年 6 月 17 日，ECMAScript 6 发布正式版本，即 ECMAScript 2015。

34 岁的系统程序员 Brendan Eich，1995 年 4 月，网景公司录用了他
1995 年 5 月，网景公司做出决策，未来的网页脚本语言必须"看上去与 Java 足够相似"，但是比 Java 简单，使得非专业的网页作者也能很快上手。
他只用 10 天时间就把 Javascript 设计出来。

JavaScript 的特点【不需要记住，了解即可，后面都会讲】
（1）借鉴 C 语言的基本语法；
（2）借鉴 Java 语言的数据类型和内存管理；
（3）借鉴 Scheme 语言，将函数提升到"第一等公民"（first class）的地位；
（4）借鉴 Self 语言，使用基于原型（prototype）的继承机制。

## 2.3 JavaScript 的组成

ECMAScript：JavaScript 的语法。
DOM：JavaScript 操作网页上的元素的 API
BOM：JavaScript 操作浏览器的部分功能的 API

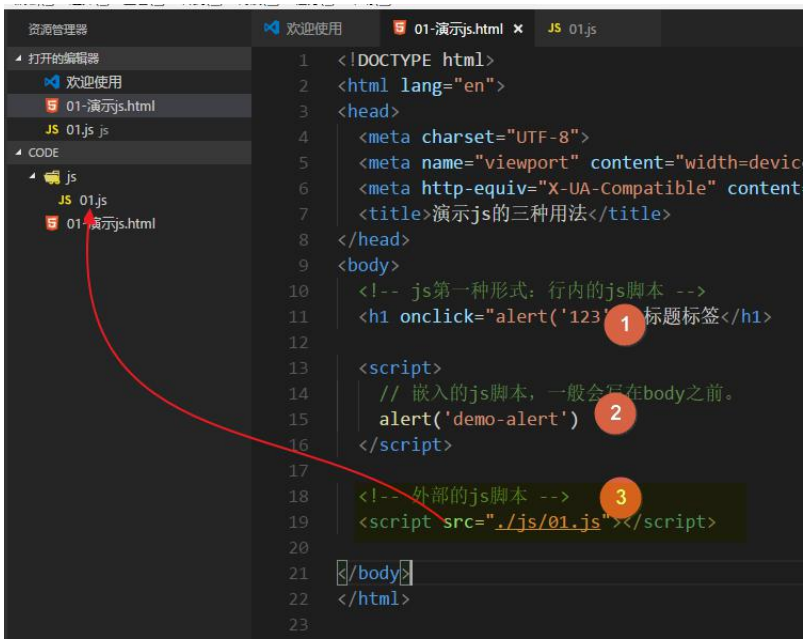
## 2.4 引入 JavaScript

**回顾 CSS 引入的三种方式：**

- 行内样式：写在标签上的 CSS
- 嵌入样式：style 标签
- 外部样式：link 标签引入的 CSS 文件

**JavaScript 的引入方式：**

- 行内 JavaScript 脚本
- Script 标签
- 外部脚本




## 2.5 Script 的属性

| 属性 | 值 | 描述 |
| --- | --- | --- |
| async | async | 立即异步下载脚本，但是不影响下载 HTML（仅适用于外部脚本）。 |
| charset | utf-8... | 规定在外部脚本文件中使用的字符编码。 |
| defer | defer | 规定是否对脚本执行进行延迟加载，直到页面加载为止。 |
| src | URL | 规定外部脚本文件的 URL。 |

```
<script type="text/javascript" defer="defer"
charset="utf-8" ></script>
```

```
<script type="text/javascript"  src="" async="async"
charset="utf-8" ></script>
<!--蓝色部分都可以直接省略-->
```

# 3. JavaScript 语法

## 3.1 JavaScript 的标识符规范

标识符也就是名字，比如：变量、函数、属性、参数等的名字。

标识符的规定：

- 第一个字符必须是字母、下划线或者美元符号$
- 其他的字符可以是字母、下划线、数字

```
例如:footerBar      bannerArea   _userName   $home2
错误例子:88Bar   &demo    address(Info)
```

## 3.2 直接量

直接量：就是在程序中直接使用的数据的值。

```
18  //数字
1.2//小数
"web.itcast.cn"//字符串
true   false//布尔值
{  a: 8; }   //
[] //数组
```

## 3.3 JavaScript 的注释

- 行内注释：

```
//注释内容
// 注释 2
```

- 多行注释，跟 CSS 一致。

```
/*
注释内容
*/
```

## 3.4 JavaScript 语句

JavaScript 语句就是开发人员让程序执行的命令。JavaScript 程序由语句组成。

JavaScript 会忽略多余的空格和换行符。

```
例如:var x = 19;
定义一个变量 x, 它的值是 19;
“=”号是赋值运算符, 把 19 的值给 x
var 定义一个变量的关键字
var y = x + 20;
```

每条语句用";"结束，非必须，但是我们要求必须添加";"号。

```
//例如:
var x,y;
x = 10;
y = 19;
y = y * x;
console.log(y);
```

## 3.5 变量

变量：源于数学的概念。变量，即是可以改变的量。也就是计算机内存中存储数据的标志符，变量的标识符指向的计算机存储空间可以存储相关的数据。

JavaScript 中的变量是弱类型的，可以存储任何数据类型的数据。

变量命名的语法： var a = 10; //var 是变量定义的关键字。语句的含义为：定义一个变量 a, 并且给 a 赋值 10，也就是 a 指向的计算机存储空间中存储了 10 这个值。

变量可以在声明的时候赋值，也可以稍后赋值。

```
例如: var  a;  //定义变量 a
  a= "1222";//给 a 赋值字符串。
```

变量可以随时改变存储的数据，甚至类型不一致都可以。

```
var a = 9;  a = "cn";//弱类型
```

弱类型：变量的类型可以随时改变，不强迫类型一旦确定不能更改。

```
如果定义多个变量用“,”隔开
例如:var  a, b = 9,  c = 123;//都是局部变量
特殊情况:var  a = b = c = 9;//var 只能作用到 a 上, b、c 都是全局变量。
```

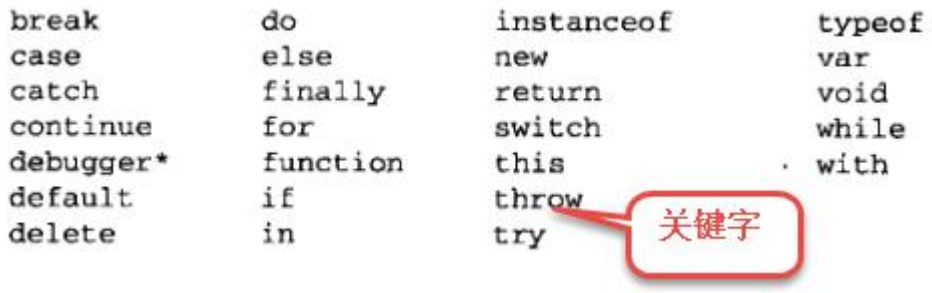
```
// 赋值语句
var x = 123; // 数字
var y = "3333"; // 字符串
var c = false; // 布尔值
var userName = []; // 数组
// 变量定义都通过 var 关键字
console.log(x)
```

```
// 先声明变量后进行赋值
```

```
var t;
t = 8 * 9; // 72 *代表的乘法
console.log(" 8 * 9 = " + t)
t = "字符串";
console.log(t);
// 定义多个变量
var a1, a2, a3, a4; // 声明了 4 个变量（局部变量）
a1 = 1;
a2 = "sss";
var b1 = 1, // 声明变量，顺便给变量赋值
b2 = 3,
b3 = "nihao";
// 绝对不推荐的一种
var c1 = c2 = c3 = c4 = 0; // 定义了一个 c1 局部变量。 c2 c3
c4 全局变量
```

## 3.6 关键字保留字

关键字：JavaScript 语言用于程序控制或者执行特定操作的英语单词。




保留字：ECMAScript 规范中，预留的某些词汇，以便于以后某个时间会用于关键字。

| | | | |
|---|---|---|---|
| abstract | enum | int | short |
| boolean | export | interface | static |
| byte | extends | long | super |
| char | final | native | synchronized |
| class | float | package | throws |
| const | goto | private | transient |
| debugger | implements | protected | volatile |
| double | import | public | |

保留字

# 4. JavaScript 数据类型

## 4.1 数据类型分类

```
JavaScript 的数据类型有:
数值类型(number)
123,   0.2   100.3 -4   -9.0
10e10   => 10 的 10 次方
布尔类型(boolean)
true   false
字符串类型(string)
'1dddd'  ,  “sssssssljldj”
未定义类型(undefined)
undefined
null 类型(本质上是一个特殊的 object)
null
var t = null;   // t 就是 null 类型
var m ;          //  m  == undefined
object 类型(引用类型)
function 类型(函数类型)
```

这些类型又分为简单类型和复杂类型。其中 object 和 function 为复杂类型，其他为简单类型。

## 4.2 typeof 获取数据类型

typeof 是一种**操作符**。操作就是：获取变量或者字面量的数据类型。
使用方法：

```
typeof(变量|直接量)
typeof 变量|直接量
```

```
Console was cleared
> typeof(6)
< "number"
> typeof "字符串"
< "string"
> typeof null
< "object"
> typeof t
< "undefined"
> var m = {}
< undefined
> typeof m
< "object"
> function f() {}
< undefined
> typeof f
< "function"
>
```

## 4.3 number 类型

- Number 类型，包括了所有的数字类型，包括：小数、整数、正负数、实数等。

```
1、整数: var   num = 10;
   合法的整数的范围:2-53 到 2 53 次方
2、小数(浮点数):  var  num2 = - 1.33;
   小数如果是 0 开头, 可以省略 0.例: var b = .33;
3、表示十六进制:以 0x 开头的数据。
   十六进制数字表示从 0 到 9, A 到 F
   var b = 0x4B;// =>十进制: 4*16+ 11
4、科学表示法的数字:
```

```
    8.88e22   //8.88*1022
    3.3e-22   //3.3*10-22
```

- 数据的表示范围

```
1、最大值:Number.MAX_VALUE
    1.7976931348623157e+308
2、最小值 Number.MIN_VALUE
    5e-324
3、超过以上的范围后:Infinity
正 Infinity
负 Infinity
```

- 非数字 NaN

NaN：not a number，非数字值，是数字类型，但是非常特殊的数字类型值。

```
parseInt("ss")//把 ss 字符串转成整数, 此时转换失败会返回 NaN
```

**涉及到 NaN 的所有的操作都会返回 NaN**

判断是否是 NaN 使用 isNaN 方法

```
isNaN(NaN);// => true
```

## 4.4 算数运算符

- +：加法。 10 + 10 = 20
- - 减法
- * 乘法
- / 除法
- % 求余（整除后的余数） 3%2 = 1

代码示例：

```
// 算数运算符:  +
var t = 10;
t = 10 + 290;
```

```
console.log(t); // => 300

// - 号

var m = 200;

console.log( t - m ); // t= 300, m = 200. ==> 100

// 乘法

console.log( t * m); // 300 *200 = ==>60000

// 除法

console.log( m / t); // 200 / 300 = 1.5

// 取余数

console.log( 7 % 4 );
```

## 4.5 算数运算符综合练习

1. 定义变量：a,b ,c，d

2. 变量 a = 10, b = 8, c = 9;

3. 变量 d 的值为 a 与 b 的 和。

4. 让变量 c 的值改变为：变量 d 和 变量 a 的乘机。

5. 让变量 c 的值翻倍。

6. 定义变量 f 为 abcd 的和，并输出 f 的值

```
// 定义变量: a,b ,c, d

var a, b, c, d;
```

```
// 变量 a = 10, b = 8, c = 9;
a = 10;
b = 8;
c = 9;
// 变量 d 的值为 a 与 b 的 和。
d = a + b;
// 让变量 c 的值改变为：变量 d 和 变量 a 的乘机。
c = d * a;
// 让变量 c 的值翻倍。
c = c * 2;
// 定义变量 f 为 abcd 的和，并输出 f 的值
var f = a + b + c + d;
console.log(f);
```

## 4.6 boolean 类型（布尔类型）

布尔类型只有两个值：真 true 和 假 false。注意区分大小写，都是小写。

```
var  x =  true;//真
var y =  false;//假

var z  =  10 > 9;  // z=>true

var m = (NaN == NaN);   // m => false;

var t = (1 === '1');  // t => false 三个等号比较值和类型。
```

```
var t = (1 == '1');    // t => true
```

## 4.7 字符串类型

字符串：就是由字符组成的串（羊肉）
定义一个字符串直接量： “字符串” '字符串'.
字符串都是用双引号或者单引号引起来。

```
var a = "1233"
a  = '123';
```

字符串直接量必须写在同一行。（EC5 中可以用/链接不同行）
toString()方法将其他类型转为字符串类型。

```
var  t = 19;
t.toString();  // => “19”
```

"+值 => 转字符串

```
var  t = 19;
t = t + ‘’;  // t => “19”
```

## 4.8 字面量转义符

在字符串中某些特殊的字符用转义符表示。
比如：如何在字符串中表示双引号？
var a = "ssss"sss\n\t \b ss";//错误
var a = "ssss\"ssssss";//转义符： \" => "

| 字面量 | |
|---|---|
| \n | 换行 |
| \t | 制表 |
| \b | 空格 |
| \r | 回车 |
| \f | 进纸 |
| \\ | 斜杠 |
| \' | 单引号（'） |
| \" | 双引号（"） |

## 4.9 Undefined 类型

Undefined 类型只用一个特殊的值：undefined，中文的意思就是未定义。

undefined 值出现的情况：

- 当声明了一个变量，而未赋值的时候则变量的值为 undefined。
- 当未声明变量，直接对变量操作的时候（函数的参数未定义直接使用），变量会初始化一个值：undefined
- 显式的给变量赋值 undefined
- 当调用一个对象的属性或者函数执行没有任何返回值时返回 undefined

## 4.10 空对象 null

Null 只是对象类型的一个值 null：空对象。
null 表示对象为空，或者是无对象。
typeof null=>"object",所以 null 是一个特殊的 object
使用方法：

```
var  a = null;//定义一个空对象。
null ==  undefined //true, 都表示空
null === undefined // false , 类型不同
```

# 5. JavaScript 运算符

运算符分类：

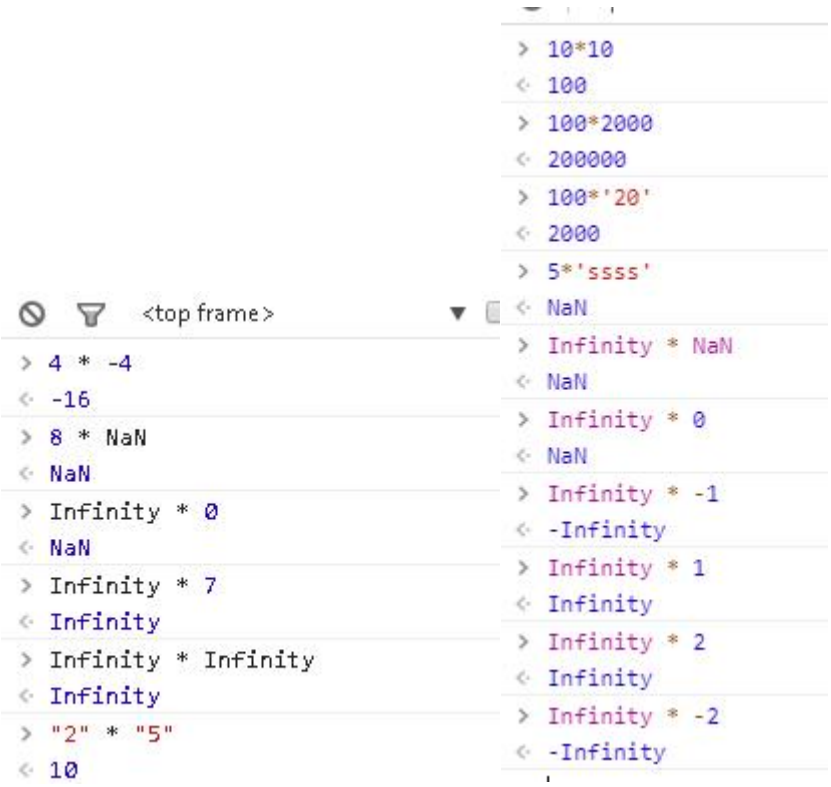
- 一元运算符
- 算数运算符
- 位操作符
- 关系操作符
- 相等运算符

## 5.1 算数运算符

### 5.1.1 乘法运算符

在处理特殊值的情况下，乘法操作符遵循下列特殊的规则：

- 如果操作数都是数值，执行常规的乘法计算，即两个正数或两个负数相乘的结果还是正数，而如果只有一个操作数有符号，那么结果就是负数。如果乘积超过了 ECMAScript 数值的表示范围，则返回 Infinity 或-Infinity；
- 如果有一个操作数是 NaN，则结果是 NaN；
- 如果是 Infinity 与 0 相乘，则结果是 NaN；
- 如果是 Infinity 与非 0 数值相乘，则结果是 Infinity 或-Infinity，取决于有符号操作数的符号；
- 如果是 Infinity 与 Infinity 相乘，则结果是 Infinity；
- 如果有一个操作数不是数值，则在后台调用 Number()将其转换为数值，然后再应用上面的规则。




## 5.1.2 除法运算符

与乘法操作符类似，除法操作符对特殊的值也有特殊的处理规则。这些规则如下：

- 如果操作数都是数值，执行常规的除法计算，即两个正数或两个负数相除的结果还是正数，而如果只有一个操作数有符号，那么结果就是负数。如果商超过了 ECMAScript 数值的表示范围，则返回 Infinity 或-Infinity；
- 如果有一个操作数是 NaN，则结果是 NaN；
- 如果是 Infinity 被 Infinity 除，则结果是 NaN；
- 如果是零被零除，则结果是 NaN；
- 如果是非零的有限数被零除，则结果是 Infinity 或-Infinity，取决于有符号操作数的符号；
- 如果是 Infinity 被任何非零数值除，则结果是 Infinity 或-Infinity，取决于有符号操作数的符号；

```
<script>
```

```
// 1、 如果是两个正常的数字运算，那就按照数学的除法运算得到结果。
var t0 = 9 / 3; // 3
var n1 = -999999e4000;
var n2 = 0.0000000000000000000000000000000000000000000000000000000001;
var t = n1 / n2;
console.log(t); // => -Infinity
// n1 / n2 的结果超出了 es 的数值的表示范围，那么结果就是 Infinity

// 2、 如果一个操作数是 NaN 那么结果一定是 NaN
var n3 = NaN;
var t2 = n1 / n3; // => NaN
var t3 = n3 / n2; // => NaN
// 3、 如果是 Infinity 和 Infinity 进行相除结果是 NaN
console.log(Infinity / Infinity); // => NaN
// 4、 0 / 0 ,0 被除，如果是 0 那么结果是 NaN，如果不是 0 那么结果我就是 Infinity
console.log( 0 / 0 ); // => NaN
console.log( '9/0=' + (-9 / 0 )); // => -Infinity
```

```
// 5、 如果 Infinity 除以任何数，结果都是 [-]Infinity
console.log('Infinity / 8 = ' + (Infinity / 8)); //=>
Infinity
</script>
```

### 5.1.3 求余运算符

- 如果操作数都是数值，执行常规的除法计算，返回除得的余数；
- 如果被除数是无穷大值而除数是有限大的数值，则结果是 NaN；
- 如果被除数是有限大的数值而除数是零，则结果是 NaN；
- 如果是 Infinity 被 Infinity 除，则结果是 NaN；
- 如果被除数是有限大的数值而除数是无穷大的数值，则结果是被除数；
- 如果被除数是零，则结果是零；
- 如果有一个操作数不是数值，则在后台调用 Number()将其转换为数值，然后再应用上面的规则。

```
> Infinity % 2
< NaN
> Infinity % 0
< NaN
> 333 % 0
< NaN
> Infinity % Infinity
< NaN
> 33 % Infinity
< 33
> 0 % 3
< 0
> "3"  %  2
< 1
```

### 5.1.4 自增自减运算符

自增运算符，用两个连续的+号。比如： ++i t++.
如果两个加号在变量前面，先进行自增运算再进行表达式运算。
不然，先进行表达式运算，后进行自增运算。

```
var i = 9;
```

```
i++; // 自增运算符，  自己增加 1

// i = i + 1;

console.log( i ); // =>10

// 自加运算符在前面的时候，先进行自加运算，再参与表达式的运算。

++i; // i = i + 1;

console.log( i ); // =>11

// var i2 = ++i + 3; // i = 11, ++i > 12 12+3=>15

// console.log(i2)

var i2 = (i++) + 3; // i = 11, i + 3 > 14 ,最后在让 i= i+1;

console.log(i2); // => 14

var t = 11;

t--;

// t = t -1; // 自减运算符 ， 自己减 1

console.log(t)
```

自减同自加，不再赘述。

## 5.1.5 加法运算符

- 加法的二义性：

加法运算分为： 字符串连接运算和算数加法运算。当运算的变量中有字符串的时候优先字符串连接，后考虑算法运算。

```
var  a = 1 + 2; // a =>3 算数运算加法

var a = "我是" + "老马";  // a = "我是老马" 字符串连接
```

- 一元加法运算

当操作数只有一个的时候，+法运算符，就是把操作数使用 Number 函数进行转成数值类型。
比如：

```
var a  =  1 + "22";  //=> a="122",字符串连接操作
var t = ‘23’;
var m = +t;  // m => 23 数字
// 等价于: m = Number(t);
```

## 5.1.6 减法运算符

减法运算符：只有数学减法运算。不具备字符串链接功能。
比如： var t = 3 - '2'; // => 1
减法运算符也具备将变量转换为数值类型的作用

- 如果两个操作符都是数值，则执行常规的算术减法操作并返回结果；
- 如果有一个操作数是 NaN，则结果是 NaN；
- 如果是 Infinity 减 Infinity，则结果是 NaN；
- 如果是-Infinity 减-Infinity，则结果是 NaN；
- 如果是 Infinity 减-Infinity，则结果是 Infinity；
- 如果是-Infinity 减 Infinity，则结果是-Infinity；
- 如果是+0 减+0，则结果是+0；
- 如果是+0 减–0，则结果是–0；

```
> 4-3
< 1
> NaN - 3
< NaN
> Infinity - Infinity
< NaN
> -Infinity - (-Infinity)
< NaN
> Infinity - (- Infinity)
< Infinity
> -Infinity - Infinity
< -Infinity
> 0-0
< 0
> 0-(-0)
< 0
> -0-0
< -0
```

## 5.2 布尔相关的运算符

### 5.2.1 关系运算符

关系运算符返回的都是 boolean 类型结果。

- 大于 >

```
  2  > 3 //false
"32" > "223"  // true,字符串比较的是字符串的编码大小。
```

- 小于 <

```
2 < 3 //true
```

- 小于等于 <=

```
var  a = 22, b = 22;  a<=b//true
```

- 大于等于 >=

```
 22 >22 //true                    33>=22  //true
```

- 逻辑运算符的规

- 如果两个操作数都是数值，则执行数值比较。
- 如果两个操作数都是字符串，则比较两个字符串对应的字符编码值。
- 如果一个操作数是数值，则将另一个操作数转换为一个数值，然后执行
- 如果一个操作数是对象，则调用这个对象的 valueOf()方法，用得到的行比较。如果对象没有 valueOf()方法，则调用 toString()方法，并的规则执行比较。
- 如果一个操作数是布尔值，则先将其转换为数值，然后再执行比较。

```
> 4 > 8
< false
> 10 > 0
< true
> 9 <= 9
< true
> 'a' < 'A'
< false
> 'A' > 'aA'
< false
> 1  >  '2'
< false
> 10  > '2'
< true
> '10' > '2'
< false
> 2 >  false
< true
> -1 > false
< false
> Number(false)
< 0
```

| | | |
|---|---|---|
| a | A | 2 t m 2 |
| A | 1 | t 2 |

a 97
A 65
a > A

var a1 = abbcAffc;
var a2 = abbcaffc;
a = 97
A =65
a >A
a1 <a2; // true

Ascii 码表地址：
https://baike.baidu.com/item/ASCII/309296?fr=aladdin&fromid=19660475&fromtitle=ascii%E7%A0%81%E8%A1%A8

## 5.2.2 逻辑操作符

- 逻辑非

逻辑非的符号用“！”取相反的意思。

```
比如: var  a = true;    a = !a; //a = false;
```

- 逻辑与

逻辑与是用两个“&&”符号运算

```
var a = true, b = true;   a&&b;//true
```

同时为真时才返回 true，其他 false

- 逻辑或

用两个竖线连接两个变量运算（||）

```
var  a = true, b = true;  a||b //true
```

两个变量只要有一个为真，那么就返回 true。

- &&短路表达式 ***

由于进行逻辑运算的变量不一定都是 Boolean 类型

非 Boolean 类型的&&逻辑运算返回第一个为假值或者最后一个真值的操作数（直接量、表达式）的值，可以不是 Boolean 类型。

比如：

```
89  &&  (9 < 3)      // => true
0    &&  98           // => 0, 注意不是 false
null &&  1            // => null
```

- 类似短路表达式，||运算符可以返回第一个真值或者最后一个假值。

|| 类型安全返回处理

有的时候为了防止变量为空，可以设置一个保底的结果。

```
t = max || max-width || 500;
```

```
> 1 && ''
< ""
> Boolean("")
< false
> // 返回第一个假值，或者最后一个真值
< undefined
> 1 && true && 'ssss' && 0 && 20
< 0
> 1 && true && 'ssss' && 2 && 20
< 20
> a && (+a)
✖ ▶Uncaught ReferenceError: a is not defined
    at <anonymous>:1:1
> var a = "22";
< undefined
> a && (+a)
< 22
> a = null;
< null
> a && (+a)
< null
> 7 || "ssss" || 0 || null
< 7
> 0 || "" || 7 || "ssss" || 0 || null
< 7
```

## 5.2.3 相等运算符

- 数值相等 ==

类型可以不相同，值相等就可以。

```
var a = 3,   b = "3";    a==b;//true
```

- 不等于 !=

值不等就返回 true，值相等就直接返回 false

```
3!=4 //true
3 != "3" //false
```

特殊情况：

```
NaN 跟任何其他变量都不等，甚至不等于自己。
NaN != NaN//true
null ==  undefined  都是空的意思。
```

- 全等运算符 ===

全等就是数值相等而且类型相同。

```
  55 =="55" //true
  55 === "55" //false,值转换后相等，但是类型不同。
```

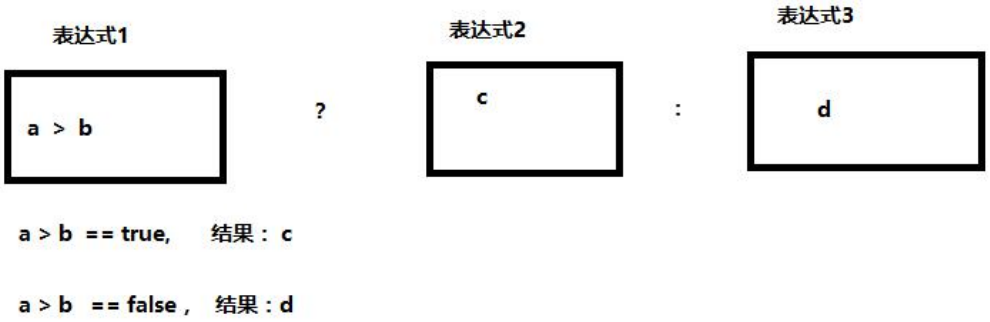
- 不全等运算符 !==

只要类型不同或者值不等那么就返回 true

## 5.2.4 条件运算符

条件运算符：  判断值表达式 ？true 返回值 1 ：false 返回值 2。

```
var  b =  a > 3 ?  2  :  1;  //语义:a 大于 3 吗？如果大于返回 2，如果不大于返回 1;
var  b =  判断表达式  ?  值 1  : 值 2;
当判断表达式为 true 的时候，返回值 1。
```




a > b == true,　结果：c

a > b == false，结果：d

## 5.3 赋值预算符和逗号运算符

### 5.3.1 赋值运算符

等号，就是最常用的赋值运算符。

```
var a = 9;
var a = 9, b = 3, c =4;
```

赋值和算数运算符结合简写

```
a  = a + 10;//当+-*/运算符跟自己进行运算时可用简写成
a += 10;// 跟 a = a + 10 效果一样。
a *= 2;   a -= 3;  a/= 3;   4%=2;
```

### 5.3.2 逗号运算符

逗号运算符，可用让 js 在一条语句中执行多次操作。

比如：

```
 var a = 2, b = 3, c =4;//一次性定义 3 个变量，并分别给变量赋值。
```

逗号运算符在用于赋值操作的时候，永远只返回最后一个值

比如：

```
a = (2,3,4);//a=4
```

## 5.4 综合练习

题目：

var a = '1',  b = 3, c = true;
求下面的值：

```
 a > b  => false  转数字比较
 a >= c => false 转成字符比较
 !(b) => false
!!a  => true
a+b =>  '13'
b + c => 4
b - a  => 3-1 =>2
b && a  => true
!(a || b)  => false

a > b && c < a =>false
+c  =>1
++a+c => 3
++b + 'ssss' =>'4ssss'
6 / 0  => Infinity
NaN * 0  =>NaN
b > c ?  ++a: c;  =>3
```

# 6. JavaScript 类型转换

JavaScript 类型之间可以相互转换，以下是本节要讲的三种类型的相互转换。

- 字符串与各种类型转换
- 数字跟各种类型转换
- boolean 类型跟各种类型转换

## 6.1 显示类型转换

### 6.1.1 数值类型的转换

- **Number 函数转数值类型**

Number(mix)函数，可以将任意类型的参数 mix 转换为数值类型

- 如果是布尔值，true 和 false 分别被转换为 1 和 0
- Number(true)// 1
- 如果是数字值，返回本身。
- 如果是 null，返回 0.
- 如果是 undefined，返回 NaN。
- 如果是字符串，遵循以下规则：
  - 如果字符串中只包含数字，则将其转换为十进制（忽略前导 0）
  - 如果字符串中包含有效的浮点格式，将其转换为浮点数值（忽略前导 0）
  - 如果是空字符串，将其转换为 0
  - 如果字符串中包含非以上格式，则将其转换为 NaN

```
单加法操作跟 Number 函数效果一致:
var  b, a = false;
b =  +a;// b =  0  => b= Number(a);
b =  + "123"  //  b = 123 ,把字符串转为数字
```

- **parseInt 转换数字**

parseInt(string，radix)函数，将字符串转换为整数类型的数值。它也有一定的规则：

- 忽略字符串前面的空格，直至找到第一个非空字符
- 如果第一个字符不是数字符号或者负号，返回 NaN
- 如果第一个字符是数字，则继续解析直至字符串解析完毕或者遇到一个非数字符号为止
- 如果上步解析的结果以 0 开头，则将其当作八进制来解析；如果以 0x 开头，则将其当作十六进制来解析

- **parseFloat(string)函数**

  将字符串转换为浮点数类型的数值。其他规则同 parseInt。


## 6.1.2 布尔类型的转换

Boolean(mix)函数，将任何类型的值转换为布尔值。

以下值会被转换为 false：
**false、”"（空字符串）、0、NaN、null、undefined，其余任何值都会被转换为 true。**

```
Var t = Boolean(123); // t => true
!!两次取饭跟 Boolean 方法效果一致
var  a = 3,b = 0,  c = "";
!!a;  // true
!!b;  //false
!!c;  //false
```

## 6.1.3 字符串类型的转换

String(mix)函数，将任何类型的值转换为字符串，其规则为：

```
如果有 toString()方法，则调用该方法并返回结果
var  a = 4;  a.toString();//"4"
如果是 null，返回”null”
a  = null;  a.toString();// "null"
如果是 undefined，返回”undefined”
```

也可以之间用变量+"的方法转换类型为字符串类型，同 String 函数。

```
例如：  a + “”;  // =>String(a);
```

## 6.2 隐式类型转换

隐式类型转换：就是当表达式进行运算时，如果类型不一致，JavaScript 引擎会自动根据规则把类型进行转换后再进行运输。

```
var b = +a;// 相当于 b = Number(a);
var b = !!a; // b = Boolean(a);
var b =  a + "";// String(a)
var c = 1 + true; // c=>2
var t = true + '';  // t => true
```

# 7. JavaScript 语句

JavaScript 程序由基本的语句组成。默认情况下语句是从第一条往后顺序执行。为了应对复杂需求，出现了分支语句、循环语句、声明语句、跳转语句等。
语句是编程的核心，要练的非常熟练！！！
JavaScript 语句分为：

- 表达式语句
- 空语句与复合语句
- 声明语句
- 分支语句（条件语句）
- 循环语句
- 跳转语句

## 7.1 表达式语句

```
123;  'sss';   true;   false;   null;  undefined;
12+true;    a+b;   Number(true);
```

## 7.2 单行语句

```
   var  a = 9;   // 单行的赋值语句
    b  =  a  * 9;   //  单行语句
```

## 7.3 复合语句（语句块）

复合语句，就是多条语句用花括号括起来，组成的整体就是一个语句块，也称为复合语句，在 JavaScript 中会把语句块当成一条语句执行。

```
{
   var  a = 2;
   a++;
   b = !!a;
}   // 语句块末尾不需要加分号。
```

## 7.4 声明语句

### 7.4.1 变量声明语句

关键字 var

语法格式：var 变量 1[=值 1][, 变量 2[=值 2]]….;

例如：

```
var a ,   // a =>undefined
 b =2,
 c = 3;
```

### 7.4.2 函数声明语句

```
function f() {  // function 是关键字，标注：当前为函数声明
    var a = 123;
    a *= 3;
    return a;
}   // 函数的定义用一个语句块包裹。
```

## 7.5 if 语句

### 7.5.1 简单分支 if 语句

If 语句是：条件分支语句。
语法：

```
if (表达式)  语句;          // 第一种
if(表达式)                  // 第二种
    语句;
if(表达式) {                // 第三种，推荐。
    语句块;
}                           // 语句块结尾不需要加分号。
```

语义：当表达式为真的时候，执行后面的语句。

```
demo:
    if( 233 * 2 > 456) {
        a = 4;
    }
```

注意：当 if 语句的执行语句只有单行的时候，可以省略花括号，但是建议不要省略。

### 7.5.2 if-else 语句

如果我需要实现：如果小明的年龄小于 18 岁，那么不给小明啤酒喝，否则给小明啤酒喝?
else：否则的意思

```
if else 语句的格式:
if(表达式){
    语句;
}else {
    语句;
}
```

```
var age = 10;
// 如果小明的年龄大于 18，给他喝啤酒，否则不给他喝啤酒。
if(age > 18) { // 如果条件表达式为真值，那么执行紧跟的语句块，否则执行 else 语句块
    console.log('小明，大于 18 岁了，可以喝啤酒');
} else {
    console.log('小明，小于 18，不能喝酒！ ');
}
```

## 7.5.3 if else if 语句

如果有多个条件，分别对应执行多个操作的时候怎么用 if 和 else 实现呢？
比如： a = 1[2,3--7]; 输出 a= 1,那么输出星期一，2=>星期二.....
语法格式：

```
if(表达式) {
    语句;
}else if(表达式) {
    语句;
}else if {
```

```
    语句;
} else {
    语句;
}
```

代码：

```
var num = 5;
if( num == 1 ) {
console.log('星期一');
} else if( num == 2 ) {
console.log('星期二');
} else if( num == 3 ) {
console.log('星期三');
} else if( num == 4 ) {
console.log('星期四');
} else if( num == 5 ) {
console.log('星期五');
} else if( num == 6 ) {
console.log('星期六');
} else {
console.log('星期日');
}
```

### 7.5.4 If 语句嵌套

If 语句内部的语句块可以嵌套其他语句及 if 语句

语法格式：

```
if(表达式) {
    if(表达式 2) {
      if(表达式 3) {
        console.log(1);
      }
    }
}
```

案例：

如果变量 a 是数字类型，那么判断其是否大于 9，如果大于 9 则输出'大于 9'，否则输出:'非法类型'。

```
// 如果变量 a 是数字类型，那么判断其是否大于 9，如果大于 9 则输出'大于 9'，否则输出:'非法类型'
// 第一步：判断变量 a 是否是数字
var a = 7;
// 判断变量 a 的类型，使用 typeof 可以获取它的类型
if( typeof(a) === "number" ) {
// 判断 a 是否大于 9
   if( a > 9 ) {
      console.log('大于 9');
   } else {
      console.log('非法类型');
   }
}
// 第二步：判断他的大小。
```

## 7.5.5 if 语句练习题目

备注：Math.random()方法会随机返回 0-1 之间的小数
声明两个变量 a，b,随机赋值 0-10 之间的数，然后如果 a > b 那么输出 a，如果 a = b 输出 =号，如果 a< b 输出 老马
给定一个数值变量（1-7 之间）如果是 6、7 则输出周末，否则输出工作日.
判断用户名是否为空。
判断一个变量是不是字符串，如果是字符串请将字符串转为数值类型并返回结果。否则，直接把变量转成 boolean 类型输出结果。

```
<script>
// 第一、声明两个变量 a，b,随机赋值 0-10 之间的数，然后如果 a >
b 那么输出 a，如果 a = b 输出=号，如果 a< b 输出 老马
var a, b;
// Math.random() 可以随机生成一个 0-1 之间的小数。
a = Math.random() * 10; // *10 后，生成就是 0 -10 之间的一个
随机数。
```

```
b = Math.random() * 10;
```

```
if( a > b ) {
   console.log(a);
} else if( a == b ) {
   console.log('=');
} else {
   console.log('老马');
}
```

```
// 2、给定一个数值变量（1-7 之间）如果是 6、7 则输出周末，否则输出工作日.
var a = 7;
// if( a == 6 ) {
// console.log('周末'); // a== 6 和 a == 7 的 if 的代码段，一致
// } else if( a == 7 ) {
// console.log('周末');
// } else {
// console.log('工作日');
// }

// 判断合并
if( a==6 || a==7 ) {
console.log('周末');
} else {
console.log('工作日');
}
```

```
//3、判断用户名是否为空
// var userName = undefined; // undefined '' 'malun' null
// var userName = ""; // undefined '' 'malun' null
```

```
var userName = "ssss"; // undefined '' 'malun' null
```

```
// null == undefined => true
// if( userName == null || userName == undefined ) { //
userName == null || userName == undefined
// }
if( userName == null || userName.length == 0 ) {
console.log('空! ')
}
```

```
// 4、 判断一个变量是不是字符串，如果是字符串请将字符串转为数值类型并返回结果。否则，直接把变量转成 boolean 类型输出结果。
var t = '1243';
if( typeof(t) === "string" ) {
console.log(Number(t));
} else {
// console.log(Boolean(t));
console.log(!!t);
}
```

## 7.6 While 语句

### 7.6.1 while 语句

如何实现用 js 计算 1 到 100 的和呢？
1+2+3+4....?
while 就是循环语句的一种。

```
语法:
while(表达式) {
    语句;
}//语义:当满足条件时,循环执行语句,直到表达式不成立


demo:
var i = 0,sum = 0;
while( i <= 50){
    sum += i;
    i++;
}
```

案例 1：输出 10 个（0-100 之间）随机数
Math.random() // => 0-1 之间的一个小时。
0-100 的随机数，那么就需要 *100

```
var i = 1; // 循环的索引。
// 当 满足这个判断表达式为真值的时候,不断进行循环执行语句块。
while(i <= 100) {
    // 循环体语句块。
    t += i; // t =>1+2=>3 => 6
    i++; // i =>2 , i => 3 => 4
}
console.log(t);
// 第一步: 判断 判断表达式是否为 真值(1,true、非空字符串等)
// 第二步: 如果是真值 那么执行 while 的语句块,否则结束 while 语句的循环,继续执行后面的语句。
// 继续重复第一步和第二步。
```

案例 2：实现 1 到 100 的阶乘?

```
var i = 1, result = 1;
while( i <= 10 ) {
    result *= i;
    i++;
}
```

## 7.6.2 do-while 语句

do-whie 语句就是 while 语句的变种。
语法格式

```
do {
 语句;
} while(表达式);
```

do while 语句就是先执行一个语句块然后在做判断，如果表达式成立继续下一次的语句的执行。
do while 语句较少使用。

```
var i = 0; // 循环的索引
do {
    console.log(Math.random() * 100);
    i++; // 让索引加 1
    console.log(i);
} while( i < 10 );
console.log('循环执行完成');
```

# 7.7 for 循环语句

for 循环语句是前测试语句，其实本质是 while 循环语句的变种。

```
语法
for(初始化语句;判断语句;循环结束执行语句) {
    循环体执行的语句;
}
```

for 语句，现在执行初始化语句，只执行一次。
然后根据 判断语句是否为 true，如果是 true 则执行循环体，最后执行 结束执行语句，在进行执行判断语句，如果 false，则 for 循环结束。

```
// 循环的特点：
// 1. 都有一个索引变量，。 要么 0，要么 1。 i，k,j 一般都用这些变量名做索引变量。
// 2. 在循环体内或者每次循环的时候都要对循环的索引+1
// 3. 都有一个判断表达式进行退出的出口。

// for( 索引的赋值表达式; 判断表达式; 索引后续处理表达式 ) {
// // for 循环的语句块
// }
// 第一：先执行（索引赋值表达式），而且只执行一次。
// 第二：执行判断表达式，如果是真值，那么执行语句块。否则结束当前 for 循环语句。
// 第三：执行完语句块后，执行索引后续处理表达式。
// 第四：继续重复 第二和第三步骤。
```

```
// 赋值表达式：执行一次。 var i = 1;
// 判断表达式是每次循环的开始， 然后执行语句块， 最后执行 索引后续处理表达式+1.
for(var i = 1; i <= 10; i++) {
    // 循环体：每次循环都会执行一次。
    console.log(Math.random() * 100);
}
```

循环案例：

```
//1. 求从 1 到 100 的和
var result = 0; // 放我们最终的计算结果。
for(var i = 1; i <= 100; i++) {
    result += i;
}

console.log(result);
```

```
// 2. 求 10 的阶乘
var result2 = 1, k;
for( k = 2; k <= 10; k++ ) {
    result2 *= k;
}
console.log(result2);
```

综合案例：

1、求 100 以内，可以对 3 求余为 0 的正整数的个数。

2、斐波那契数列，又称黄金分割数列，指的是这样一个数列：0、1、1、2、3、5、8、13、21、34、……在数学上，斐波纳契数列以如下被以递归的方法定义：F（0）=0，F（1）=1，F（n）=F(n-1)+F(n-2)（n≥2，n∈N*）在现代物理、准晶体结构、化学等领域，斐波纳契数列都有直接的应用。

规律：前面两数的和等于后面的值。

请用 JavaScript 实现，求 F(n)的值，n>=3;

规定：f（1） =1， f（2） =1 ，f（3）=2... f(n)

分别用 for 循环和 while 循环实现

```
var fn1 = 1, // 第一个数
fn2 = 1, // 第二个数
n = 20, // 求 f（20）的值，n=20
i, // 循环索引
result; // 最终的结果
for( i = 3; i <= n; i++) {
   // f(n) = f(n-1) + f(n-2)
   result = fn1 + fn2;
   // 2 = 1 + 1 第一次循环
   // 3 = 2 + 1 第二次循环
   // 5 = 3 + 2 第三次循环
   // 8 = 5 + 3 第四次循环 ....
   // 下一次循环的时候，fn1 fn2 怎么变？
   // 1. fn1 变成 了当前的 result
   // 2. fn2 变成了 当前的 fn1
   fn2 = fn1;
```

```
    fn1 = result;
    console.log(result + ' ');
}
console.log(result);
```

## 7.8 continue 语句

continue 语句也是为了精确控制循环语句。
js 遇到 continue 语句后，立即结束当前循环轮次，直接把程序跳转到下一个循环的轮次。

```
for(var i =0; i<10;i++) {
    if(i %7 == 0) {
        continue;   //立即执行 结束执行语句，并执行下次循环
    }
}
```

案例：计算 1-100 的和，去除掉对 3 取余为 0 的数字。

```
// 计算 1-100 的和，去除掉对 3 取余为 0 的数字。
var i = 1, result = 0;

for(i = 1; i<= 100; i++) {
if( i % 3 == 0 ) {
    // 不能让当前的 i 参与和的运算了。略过当前的循环轮次。
    continue; // 继续下一轮的循环，当前循环轮次结束。
    }
    result += i;
```

```
    }
console.log(result);
```

## 7.9 break 语句

break 语句可以用于精确控制循环语句跳出语句块。
当 js 遇到 break 后，会立即结束当前的循环语句，并强制立即执行循环后面的语句；

```
for(var i =0; i<10;i++) {
    if(i %7 == 0) {
        break;   //立即结束循环
    }
}
```

## 7.10 Switch-case 语句

switch 语句用于基于不同的条件来执行不同的动作。

**语法**

```
switch(n)
{
case 1:
  执行代码块 1
  break;
case 2:
  执行代码块 2
  break;
default:
  n 与 case 1 和 case 2 不同时执行的代码
}
```

default 关键词来规定匹配不存在时做的事情。

把给一个数字 1-7 之间，根据数字返回相应的星期，  1：星期一，2：星期二...

```
var num = 5;
switch( num ) {
```

```
    case 1: // 当 num === 1 的时候，执行冒号后面的语句。
        console.log('星期一');
        break; // 跳出当前的 switch 语句
    case 2:
        console.log('星期二');
        break;
    case 3:
        console.log('星期三');
        break;
    case 4:
        console.log('星期四');
        break;
    case 5:
        console.log('星期五');
        break;
    case 6:
        console.log('星期六');
        break;
    // case 8:
    // case 9:
    // case 10:
    // console.log('error');
    // break;
    default:
        console.log('星期日');
}
```

# 8. 函数基础

## 8.1 函数的定义

函数由数学中定义中引申来。
函数就是，一个功能对应的所有语句的集合。
函数关键字：function
函数不能定义在循环、分支等语句中。

```
语法:
```

```
function  函数名(参数 1, 参数 2....) {
    函数体;
    // JavaScript 语句
}
函数名();//调用函数执行。
// 标识符:变量名、函数名、属性名、类名、参数名。
// 必须以字母、下划线、或者$开头, 后面可以跟着数字、字母、下划线
```

代码：

```
// 第一:  基础函数定义
// 函数定义可以放在函数调用之前和之后都可以。
// 函数的关键字: function
function add() {
    // 函数体
    console.log('my first function! ');
}

// 函数的调用语句。
add();
```

```
// 第二: 带参数的函数
// 定义一个函数，实现求两个数的乘积
function mult(a, b) { // a \ b 形参。
    // a * b;
    console.log(a * b);
```

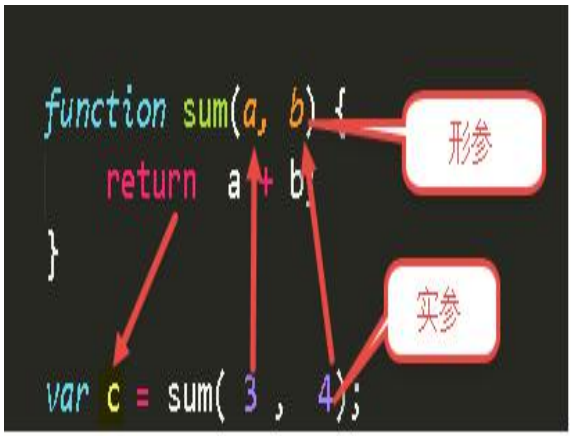
```
    return a * b;// 把 a 和 b 的相乘=结果返回到函数调用者的地方。
}


// 函数返回的结果:  直接赋值给了变量 t
// 2 \4 是实际传递的参数，实参。
var t = mult(2, 4); // 执行 mult 函数。给函数体传递 2 和 4 两个参数。
console.log('t = ' + t);
```

## 8.2 Return 语句

return 标识函数的结束。
当程序遇到 return 时，函数立即结束并返回相应的值。否则函数执行到}结束。
如果函数没有任何的返回值，JavaScript 会默认返回 undefined

## 8.3 参数

形参和实参匹配传递
形参就是：函数定义的参数。
实参：就是函数执行传递的参数。
形参和实参个数不匹配的时候
优先从左向右进行匹配
实参个数可以少于形参，也可以多于形参
所有的参数都会放到 arguments




## 8.4 匿名函数

匿名函数就是没有函数名的函数。一般会把匿名函数赋值一个变量进行使用，或者匿名函数声明完了后立即调用自己。

匿名函数可以作为一个变量来使用。
匿名还是可以作为一个参数进行传递。

```
// 匿名函数
var f = function(a) {
   return a *a;
};
// 因为我们用 var 声明的一个变量 f，f 的类型是 function
// f 变量跟其他变量一样使用，可以作为其他函数的参数使用。
console.log(typeof f); // => function
console.log(f(3));


console.log(typeof f2);
console.log(f2(9));
// 函数的声明
```

```
function f2(a) {
   return a * a;
}
// 区别：  变量声明的函数（匿名函数）不能进行函数提升。
```

## 8.5 函数练习

```
// 实现 1 到 n 的和。
function sum(n) {
   // 实现 1 到 n 的计算
   // -9, 0 ==0 'ssss' "0" == 0 null !=0 undefined != 0
false ..
   // 如果用户传来的数据不是数字类型：返回 NaN
   // 如果用户传来的数据不是大于 1 的数字：返回 0;
   if( typeof(n) != 'number') {
      return NaN; // 直接返回 NaN。如果函数中执行遇到了 return
   语句，函数立即结束。
   }

   if( n < 1 ) {
   // return; // 直接写 return 和没有返回值的函数，都会返回一
个默认的 undefined
      return 0;
```

```
    }
```

```
// 大于 1 的情况下
    var i = 1, result = 0;
    while( i <= n ) {
        result += i;
        i++;
    }
```

```
    return result;// 把计算的和返回。
}
var t = sum(7); // => 1+ 2+ 3+..7
console.log(t);
```

# 9. 对象类型

## 9.1 Object 对象类型

对象是 JavaScript 中一组属性和方法的无序集。
对象一般是针对一功能载体的描述。比如：狗对象、猫对象。狗对象：拥有常见 age、name、color、type 属性，还有奔跑 run（）等行为。
JavaScript 中的对象是动态的，可以随时添加属性和删除属性。
除了字符串、数值类型、布尔类型、null、undefined 之外的都是对象类型。
对象是引用类型（后面还会讲）

## 9.2 Object 对象创建方式

Object 类型是我们用的最多的引用类型，我们接触到的大部分都是 Object 类型的实例。
创建 Object 类型实例的方法：

第一种：new 运算符创建对象

```
var a = new Object();//new 运算符，创建一个 Object 对象。
a.age = 18;
a.name = "itt";
```

第二种：对象字面量

```
var a = {
    name :  "itct",
    show : "http://www.ss.cn"
};
a.age = 19;
```

参考代码：

```
// 创建 Object 对象的两种方式
// 第一种：使用 new 操作符创建对象。
var t = new Object(); // 创建了一个 Object 对象 t。
// new：会创建一个对象，然后对象执行 Object 函数。最后返回一个对象实例
// js 是一个动态语言，可以随时添加属性和方法。
t.age = 19; // 年龄 19
t.name = 'jeck'; // 名字
t.run = function () { // 匿名函数
console.log('奔跑的方法...');
};
t["demo"] = "demo2";
```

```
// 对象怎么使用自己的属性和方法？
```

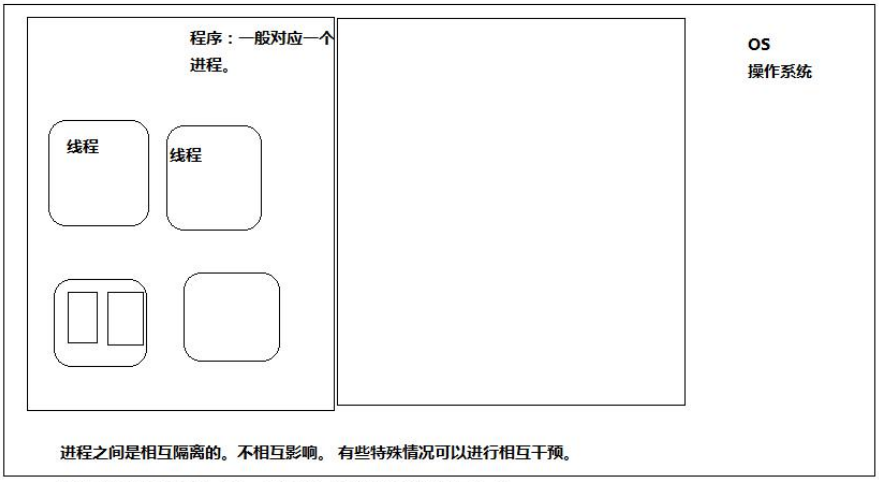
```
// 第一种方式：使用点的方式。
console.log(t.age); // => 19
// 第二种方式：使用中括号的方式。
console.log(t['name']); // => ject
```

字面量创建对象的方式

```
// 创建对象的第二种方式：字面量的方式
// 也称为使用 JSON 对象的方式。
var t = { // 创建一个对象 t
    age: 19,
    name: 'laoma',
    run: function() {
        console.log('老马跑路了！ ');
    }
};

t.color = 'yellow';
console.log(t.age);
t.run(); // 执行 t 对象中的 run 方法。如果一个对象的属性是函数，
我们就称为方法。
t.age = 10;
console.log(t.age);
```

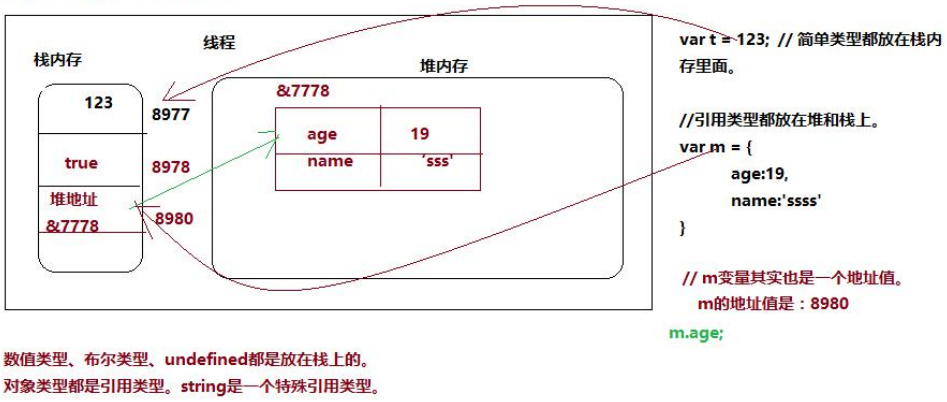
# 9.3 引用类型

引用类型是 JavaScript 中复杂类型，是一种数据结构。
JavaScrip 中所有的东西都是对象。
object 就是最长用的引用类型
引用类型在内存中模型：
栈内存
堆内存
简单类型只存储在栈上（boolean、number）




线程是执行代码的最小单位。我们所有的代码都在线程里面执行

线程的内存分为： 栈内存和 堆内存




数值类型、布尔类型、undefined都是放在栈上的。
对象类型都是引用类型。string是一个特殊引用类型。

# 9.4 删除属性

JavaScript 的对象非常灵活，可以随时删除属性。
删除属性的运算符是 delete 关键字
语法结构：

```
delete object.property
delete object['property']
```

注意以下几点：

- delete 操作符会从某个对象上移除指定属性。成功删除的时候回返回 true，否则返回 false。
- 如果你删除的属性在对象上不存在，那么 delete 将不会起作用，但仍会返回 true
- 如果 delete 操作符删除成功，则被删除的属性将从所属的对象上彻底消失。
- 任何使用 var 声明的属性不能从全局作用域或函数的作用域中删除。
- 除了在全局作用域中的函数不能被删除，在对象(object)中的函数是能够用 delete 操作删除的。

```
var t = {}; // t= new Object();
t.age = 19; // 添加一个属性 age
console.log(t.age); // => 19

// 通过 delete 操作符 删除属性
delete t.age; // 删除 t 的自定义 age 属性。
// debugger t["age"];
console.log(t.age); // => undefined

// 通过 in 运算符 可以检测属性是否属于对象的自定义属性,返回的结果是 true 或者 false
t.name = '123';
console.log('name' in t); // => true
delete t['name']; // 删除了自定义属性
console.log('name' in t); // => false
```

## 9.5 检测属性

JavaScript 的对象检测属性用 in 运算符

```
var t = {};
t.age = 19;
‘age’ in t; // =>true
delete t.age;
‘age’ in t; // =>false
```

## 9.6 枚举自定义属性

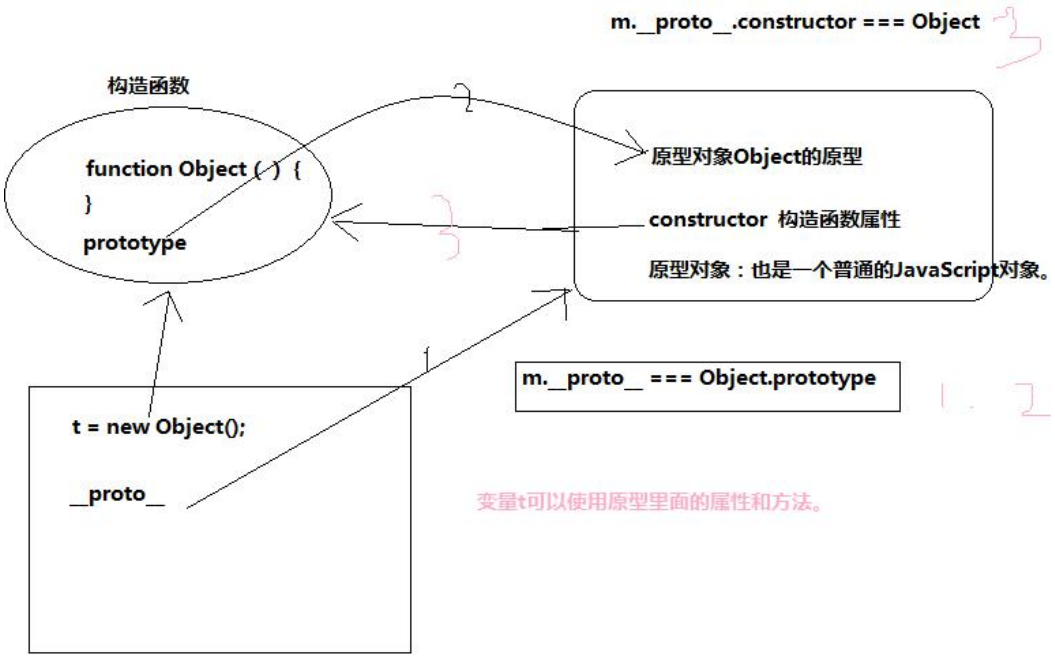
JavaScript 的对象可以使用 for in 循环遍历对象中的所有属性。

```
var t = {};
 t.age = 20;
for(var  i in t) {  console.log(i);}
```

会把继承的属性也会遍历到。

## 9.7 对象的原型

每个对象都有自己的私有原型对象（除了 null）
每个对象都会从自己的私有原型对象上继承原型的方法和属性，可以直接对象可以直接使用。
对象调用一个属性或者方法的时候先搜索自己的属性和方法，如果没有那么就去搜索原型上的属性和方法，如果有就直接使用，如果没有就直接抛出错误。
对象的私有原型都是通过__proto__联系在一块，注意这不是一个标准的属性，但是所有浏览器都实现了这个属性。
构造函数就是构造对象的时候执行的函数。比如：Object
对象的原型可以通过它的构造函数的.prototype 获的。
后面讲原型和继承的时候还会详细讲




```
> var m  = new Object();
< undefined
> typeof Object
< "function"
> Object
< ƒ Object() { [native code] }
> m.__proto__ === Object.prototype
< true
> m.__proto__.constructor === Object
< true
> m.age = 19;
< 19
> m.hasOwnProperty('age')
< true
> m.hasOwnProperty('age222')
< false
>
```

## 9.8 Object 对象原型的方法

toString()转换成字符串
toLocalString()　转换成本地化对应的字符串
valueOf()　获取它的值。
hasOwnProperty()　判断属性是否是自己创建添加的。

## 9.9 对象练习

1.封装一个猫对象。猫拥有：颜色、重量、名字、ID 等属性，还拥有跑、跳、叫等行为。并调用猫猫的跑方法。
2.输出猫猫的所有的自定义属性的名称
3.封装一个数学对象，对象中包括：Pi 属性，获取 1-n 的和的方法，获取 n 的阶乘的方法。

```
    //1、 封装一个猫对象。猫拥有:颜色、重量、名字、ID 等属性, 还拥有跑、
跳、叫等行为。并调用猫猫的跑方法。
    // var cat = new Object();
    var cat = {
      color: 'red',
      weight: 20,
      name: 'kimi',
      ID: 1234,
      run: function() {
        // this 就相当于 cat 对象。在对象的方法中使用 this 代表当前的对
象。
        console.log(this.name + ' 奔跑方法');
      },
      jump: function() {
      }
    };

    cat.run();

    // 输出猫所有的自定义属性
    for(var k in cat) {
      console.log(k);
    }
```

```
// 2、封装一个数学对象，对象中包括:Pi 属性，获取 1-n 的和的方法，获取 n 的阶乘的方法。
var myMath = {
  PI: 3.1415926,
  sum: function(n) {
    var result = 0;
    // 实现 1 到 n 的和
    for(var i = 1; i <= n; i++) {
      result += i;
    }
    return result;
  },
  factorial: function(n) {
    // 求 1 到 n 的阶乘
    var result = 1;
    for(var i = 1; i<=n; i++) {
      result *= i;
    }
    return result;
  }
};

console.log(myMath.PI);
console.log(myMath.sum(100));
console.log(myMath.factorial(10));
```

# 10. JavaScript 中的数组 Array

数组对象的作用是：使用单独的变量名来存储一系列的值。
数组就是一堆数据的分组或者集合。

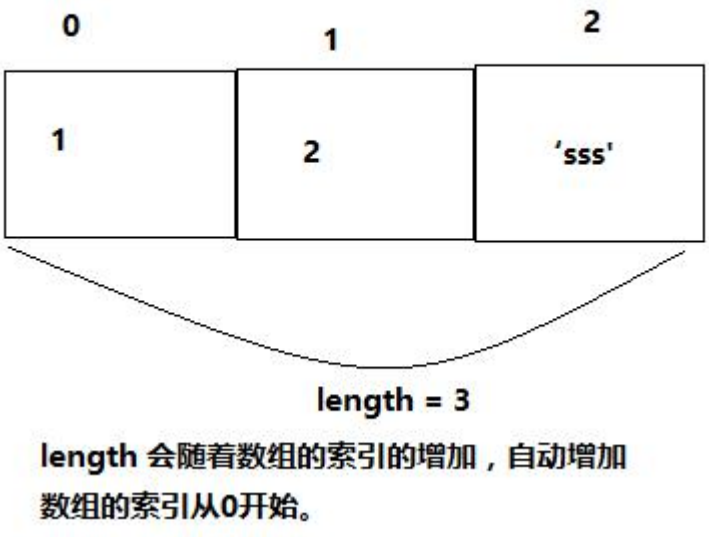
数组的对象是 Array，也是非常常用的引用类型
例如：[1, 3, ‘sss’,333]

# 10.1 创建数组

## 10.1.1 使用 new 操作符创建数组对象

```
var  arr =  new  Array();
arr[0] =  19;//数组的索引从 0 开始
arr[1] = "www.hamkd.com";
arr[2] = "www.aicoder.com";
```

数组中可以存储任何数据类型的数据（其他语言中数组的数据类型一旦确定只能存储特定类型的数据)
通过[]和索引来访问和设置数组的内容
数组的索引是从 0 开始！！！
JavaScript 中的数字的容量可以动态改变
数组的容量最大为：4294967295 个　(232-1)
求幂运算符：**　　（补充）




数组的构造函数可以传递数组的容量的参数

```
var arr = new Array();// 创建一个空数组
var  arr = new  Array(3);//定义数组的容量为 3 个
var arr = new Array("a","b","c");
```

//定义一个数字，有 a , b ,c 三个字符串
数组的长度可以通过 length 数组来获取。
length 是可读和可以进行设置，可以对数组进行截断操作。

```
var arr = new Array(); // 创建一个空数组类型;  []

// console.log(arr.toString());

arr[0] = 1; // length = 1

arr[1] = 2; // length = 2

arr[2] = 'sss'; // length = 3

console.log(arr); // [1, 2, 'sss']

console.log("length =" + arr.length);


// 构造函数创建数组传递参数

var arr2 = new Array(4); // 给构造函数传递数值类型，那么它
认为是创建容量为数值的数组

arr2[0] = 'sssss';

arr2[1] = 0;

arr2[2] = 0;

arr2[3] = 0;

arr2[4] = 0;

console.log(arr2);

// 创建数组的时候，顺便进行初始化数组的内容

var arr3 = new Array(1,2,3,'sssd', 'malun', true, [22,33]);

console.log(arr3);// length = 7

// 如果减小 length 值，就相当于截断了数组。
```

```
arr3.length = 2;

console.log( arr3 ); // => [1,2]
```

### 10.1.2 字面量创建数组

```
var arr = [];//创建一个空数组
var arr =  [1,2,3];//创建三个数字的数字。
var arr=[1, ,2];//中间的省略的是 undefined
var arr = [1 ,"dd",true, [1,3], { age: 19}, 33];
```

数组的元素的类型可以是任意类型。

```
var a1 = []; //创建空数组

// 创建一共复杂数组

var a2 = [1, 2, 'sss', true, {age:19}, null, undefined];

console.log(a2.toString());

console.log(a2);

var a3 = [2, ,3]; // => [2, undefined, 3]

console.log(a3);

// 这种方式最多，推荐这么使用。

console.log(a3[0]); // 数组的索引是从 0 开始。所有开发语言数
组都是从 0 开始。
```

## 10.2 遍历数组

length 属性
length 属性,如果是连续数组，那么 length 就是数组元素的个数
如果是稀疏数组，那么 length 不能代表元素的个数

for 循环方式遍历数组

```
// 如果是连续的数组，可以中使用 lenth 代表数组中的元素个数。
var a = [1,2,3,89, "slj", true]; // length = 6;
console.log('length = ' + a.length);
// 输出数组中的所有元素。
for(var i = 0; i < a.length; i++) {
    console.log(a[i]);
}
```

for in 循环的方式

```
// 如果是连续的数组，可以中使用 lenth 代表数组中的元素个数。
var a = [1,2,3,89, "slj", true]; // length = 6;
// 另外我们还可以使用 for in 的方式遍历数据。 也可以把继承自原型的属性也进行遍历。
// for in 遍历对象，拿到的是对象的属性名，而不是属性值。
for(var k in a) { // for in 遍历数组，k 值是数组的索引编号，不是数组的元素值。
// k 0 ,1 ,2
    console.log(a[k]); // k 是索引编号不是数组元素值
}
```

数组的索引是从 0 开始！！！不是从 1 开始

注意：从原型上继承来的属性也会被 for in 遍历，所以如果需要去掉这部分属性可以通过 hasOwnProperty（）进行过滤。

hasOwnProperty()只有当是对象自己属性才返回 true，如果是继承的返回 false。

```
> Object.prototype.temp = '222';
"222"
> var t = new Object();
undefined
> t
▶ {}
> t.age = 20;
20
> for(var i in t) {
    console.log(i);
  }
  age
  temp
undefined
```

```
> Array.prototype.myProp = 'laomao';
"laomao"
> var t = [];
undefined
> t
▶ []
> t = ["ma", 'nba', 234, true]
▶ (4) ["ma", "nba", 234, true]
> for(var k in t) {
    console.log(k);
  }
  0
  1
  2
  3
  myProp
  temp
```

继承原型上的属性也会被遍历。这不是我们想要的，我们只想遍历数组元素。

```
> for(var k in t) {
      if(t.hasOwnProperty(k)) {
          console.log(k);
      }
  }
  0
  1
  2
  3
undefined
> for(var k in t) {
      if(!t.hasOwnProperty(k)) {
          continue;
      }
      console.log(k);
  }
```

## 10.3 遍历数组案例

案例:一个数组，合法值为 1,2 或 3，其余为不合法，统计合法及不合法的个数。

```
// 案例:一个数组, 合法值为 1,2 或 3, 其余为不合法, 统计合法及不合法的个数。
    var t = [1, 4, 9, 'sss', 3, '2', 2, 3, 2, 1];
    var rigthfulNum = 0,  // 合法的个数
        illNum   = 0;     // 非法的个数
```

```
    //第一种：遍历数组中的每个元素，判断是否是合法，如果合法给
rigthfulNum += 1
    // for(var i = 0; i < t.length; i++) {
    //   if(t[i] === 1 || t[i] === 2 || t[i] === 3) {
    //     rigthfulNum += 1;
    //   } else {    // else 可以不用写。
    //     illNum += 1;
    //   }
    // }

    // console.log('合法:' + rigthfulNum);
    // console.log('非法:' + illNum);


    // // 第二种写法：只求 rightfulNum，illNum 通过 length -
rightfulNum
    // for(var i = 0; i < t.length; i++) {
    //   if(t[i] === 1 || t[i] === 2 || t[i] === 3) {
    //     rigthfulNum += 1;
    //   }
    // }
    // illNum = t.length - rigthfulNum;  // 通过数组的总长度-合法
的剩下的就非法的。
    // console.log('合法:' + rigthfulNum);
    // console.log('非法:' + illNum);

    // 第三种:for in
    // for(var k in t) {
    //   // console
```

```
//   switch(t[k]) {
//     case 1:
//       rigthfulNum += 1;
//       break;
//     case 2:
//       rigthfulNum += 1;
//       break;
//     case 3:
//       rigthfulNum += 1;
//       break;
//     default:
//       illNum +=1 ;
//   }
// }
// console.log('合法:' + rigthfulNum);
// console.log('非法:' + illNum);

// switch 优化版本
for(var k in t) {
  // console
  switch(t[k]) {
    case 1:
    case 2:
    case 3:
      rigthfulNum += 1;
      break;
    default:
      illNum +=1 ;
  }
```

```
    }
    console.log('合法:' + rigthfulNum);
    console.log('非法:' + illNum);
```

## 10.4 稀疏数组

数组中的数据的索引不一定都是连续的。非连续的数组就是稀疏数组。
比如：

```
var t = [1, 3];
t[100]= true;
// => t.length = 101;
// => t = [1, 3, undefinded*98, true]
// 实际上有数据的只有  1, 3, true  中间有 98 个 undefined
```

可以使用 for in 循环越过 undefined 的数据。
也就是数组的元素的个数不一定跟 length 相等


注意：稀疏数组尽量少用！！！！！或者就是不用！！！！

```
 var t = [1, 2];
   // 0 , 1
   t[100] = 'sslj';  // 索引到了 100,  length = 101
   console.log(t);
   console.log(t[10]); //undefined

   // for in 循环会越过稀疏数组种的 undefied 空值。
   for(var k in t) {
     console.log(t[k]);
```

```
}
```

结论：

// 1、 length 和数组的元素的个数不一定相等
// 2、 数组不一定是连续的
// 3、 使用 for in 循环可以对稀疏数组的空值 undefined 进行过滤。
// 4、 不要稀疏数组！！！！！

# 10.5 数组的常用方法

## 10.5.1 模拟数据结构栈

pop()从数组尾部弹出一个元素并返回弹出的元素。尾部是指索引最大的元素。数组的长度会减 1；
push(); //从数组的尾部压入一个元素，并返回数组的新长度 ， 数组长度+1
// 栈方法：可以对比弹夹

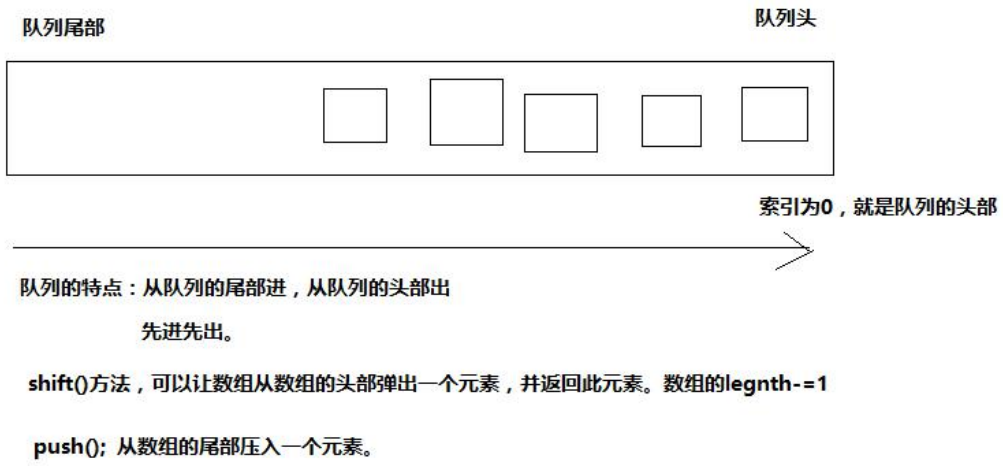
```
// 数组可以通过 push 和 pop 两个方法形成栈数据结构。
var t = ['a', 'b', 'c', 'd', 'e', 'f'];
console.log(t.length)
console.log(t);
t.pop();
```

```
console.log(t);
// 从数组的末尾弹出一个元素。并返回弹出的数组元素。数组的长度
-1
t.pop();
console.log(t);
// 压入一个元素。
var r = t.push(3);
console.log(r);
console.log(t);
```

栈是一种数据结构，特点就是 后入先出。

## 10.5.2 数组模拟队列

shift();//从数组的头部弹出一个元素，并返回此元素
unshift()//从数组的头部压入一个元素，并返回 length

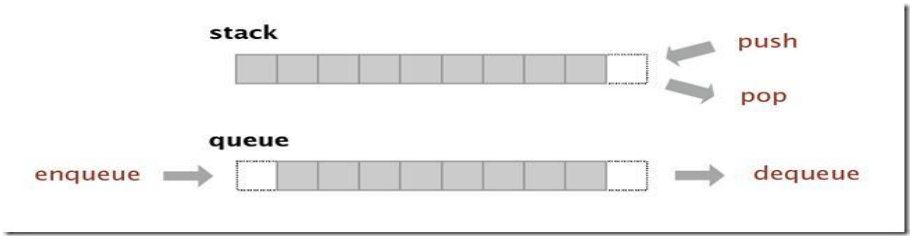



用 push 和 shift 方法可以将数组模拟成队列的数据结构。

```
// 用数组模拟队列数据结构
```

```
var t = [];
// 往队列里面放入 1， 2， 3 三个元素。
t.push(1);
t.push(2);
t.push(3);
console.log(t);


// 出队一个元素。 1
console.log(t.shift());
console.log(t);
// 再出队一个元素。
console.log(t.shift()); //2
console.log(t);
// 队列的特点就是先进先出。
```

栈和队列的数据结构总结：




栈数据结构： 先进后出。
队列数据结构： 先进先出。

## 10.5.3 排序方法

- reverse();//对原数组进行逆序
  Reverse 方法可以对原数组的数据进行逆序，原来的数组会收到影响。

```
 var t = ["a", "b", "c", "d", "e"];
 t.reverse();
 console.log(t);
// ["e", "d", "c", "b", "a"]
```

- sort()；//转成字符串排序

sort 方法对原数组进行排序，会对原数组有影响，默认从升序（从小到大）进行排序。排序比较的算法是利用字符串比较的算法，如果是其他类型的元素，会先转成字符串再进行比较。

```
// 对数组进行排序
var t = ['c', 'b', 'd', 'e', 'a'];
console.log(t);
t.sort(); // 对数组中的元素进行排序
console.log(t);
// => ["a", "b", "c", "d", "e"]


// 如果是数字进行排序
var arr = [33, 10, 1, 22, 12, 222, 30];
console.log(arr);
// sort 方法是按照字符串进行比较大小规则计算排序的。
// 如果数组中的元素不是字符串类型会转成字符串后进行比较。
arr.sort();
console.log(arr);
// =>  [1, 10, 12, 22, 222, 30, 33]
```

## 10.5.4 sort 排序高级应用

函数可以作为参数传递给其他函数进行使用。函数本身也是很一个对象。这就是函数式编程的要诀。

数组的 sort 方法可以接收一个类型为函数的参数，此函数接收两个参数，要求返回值为：负数， 0， 正数。如果返回负数代表第一个参数小于第二个参数，0 为相等，正数则大于

```
[1, 3, 20, 11, 9].sort(function (a, b) {
   if( a > b) {
      return 1;
 } else if( a == b) {
   return 0;
} else {
   return -1;
}
});
```

```
// 此函数接收两个参数，要求返回值为:负数， 0， 正数。如果返回负数代表第一个参数小于第二个参数，0 为相等，正数则大于
   // 定义一个变量，变量的类型的是 function
   var compareFun = function (a, b) {
     return a - b;  //  a ==b, a-b =0   a<b, a-b =负数
   };

   var m = [ 3, 20, 10 , 9, 11, 12];
   console.log(m);
   // 默认是转成字符串后比较大小。
   // m.sort();
   // console.log(m);

   // 利用函数式编程，sort 方法可以接受一个比较大小的函数，例如数值比较大小的方法进行排序。
   // m.sort(compareFun);
```

```
    // console.log(m);

    // 进一步优化。由于匿名函数只用一次，没有必要创建一个变量。直接可以吧匿名函数的表达式传递到 sort 函数里去就行。
    m.sort(function(a, b) {
      return a-b;
    });
    console.log(m);
```

## 10.5.5 数组的连接方法

- concat();

连接原数组的元素和传递的参数形成一个新数组并返回，不影响原来的数组。
如果传入的参数是数组，会把数组中的元素跟原数组的元素进行合并成一新数组。

```
[1, 2,3].concat(9,1,4);     // => [1, 2, 3, 9, 1, 4]
[1, 2,3].concat([9,true]); // => [1, 2, 3, 9, true]
[1, 2,3].concat([9,true, ['22',4, 9], 33])
//=> [1, 2, 3, 9, true, Array(3), 33]

   var t = [1, 2, 3];

    console.log(t);
    // 调用数组的链接方法,不会影响原来的数字，函数会返回一个新的拼接的数组。
    var newArr = t.concat('ss', true, 222);
    console.log(t);  // => [1, 2, 3]
    console.log(newArr);

    var newArr2 = t.concat(['laoma', 'beijing', 999]);
```

```
    console.log(newArr2);
```

- join();

可以把数组的元素（项）连接成字符串，接收一个参数为连接符号,默认是逗号,返回的结果为字符串。

```
[1, 2, 3].join();   //=> 1,2,3
[4,true, 3].join("-") ; // => 4-true-3
```

```
var t = [1, 2, 3];
// toString 方法会把数组转成字符，重写了原型的方法。
// 把数组中的元素都转成字符串然后用逗号分隔不同的元素。
console.log(t.toString());
// join 方法也可以将数组转成字符串，默认跟 toString 一样。
console.log(t.join());
// join 发方法可以传一个参数，用来分隔数组中的元素
console.log(t.join('-'));
console.log(t.join('|'));
```

结果：

```
1,2,3
1,2,3
1-2-3
1|2|3
```

## 10.5.6 slice 方法

slice()；//复制数组的一部分
截取数组的一个片段或者子数组
接收 1 个到 2 个参数。参数：截取数组起始索引和结束索引
如果只指定一个参数代表：从索引位置到数组结尾。
参数如果是负数代表从数组末尾计算索引位置。
此方法只能从数组前面往后面截取，如果第二个参数在第一个参数的前面则返回空数组[];

数组只能往后截取，不能向前截取，如果向前截取返回[]
此方法对原数组没有影响。

```
m = [1,2,3,4,5];
m.slice(2);  //=>[3, 4, 5]
m.slice(-3); //=>[3, 4, 5]
m.slice(3, 4);//=> [4]
m.slice(-3, -1);//=>[3, 4]
```

```
var t = [0,1,2,3,4,5];
console.log( t );
// slice: 复制数组的一部分。
// 传一个参数时候，是从参数的索引位置开始截取到数组的最后。
var a1 = t.slice(2); //从索引位置 2 开始截取到数组的最后。
// a1= [2, 3, 4, 5]
console.log( a1 );

// 传两个参数：从第一个参数作为索引位置开始，到第二个参数作为索引前面的那个元素结束，截取数组切片。对原数组没有任何影响。
var a2 = t.slice(2, 5); // a2=> [2,3,4]
console.log(a2);
console.log(t); // [0, 1, 2, 3, 4, 5] 对原数组没有影响
// 如果传递 的是负数，那么从数组结尾开始计算。 但是不要用。
// slice 方法只能往后截取，如果往前截取返回[];
var a3 = t.slice(-3, -1); // a3 => [3,4]
console.log(a3);
```

### 10.5.7 splice 方法

在原数组上进行插入或者删除数组元素，会影响原来数组。
返回的结果是删除的元素组成的数组。
参数：可以接受 1 个参数，2 个参数或者 2 个以上的参数。
第一个参数是删除数据的索引位置
第二个参数是要删除数组元素的个数
第三个参数开始是要插入到原数组中的元素，插入的位置从第一个参数所在的索引开始。

```
//删除数据
[1, 2, 3, 4, 5].splice(2);//=> [3, 4, 5]  原数组:[1,2]
[1, 2, 3, 4, 5].splice(3,2);// =>[4, 5]   原数组:[1,2,3]
[1, 2, 3, 4, 5].splice(-2); // =>[4, 5]   原数组:[1,2,3]
a = [1,2,3,4,5];
a.splice(3,2,33,'222',[99,98]);//=>[4,5]
//a=>[1, 2, 3, 33, "222", Array(2)]
//插入数据:
a=[1,2,3]; a.splice(1,0,5,6);//=>[]    a=[1,5,6,23]
//替换数据:
a=[1,2,3];a.splice(1,1,4);//=>[2]    a=[1,4,3]
```

## 10.6 数组案例

数组数据：[90, 8, 34,  2,  39, 87, 22, 10, 34]

1. 将数组内容进行反序
2. 求一个数据数组中的最小值及它的索引
3. 求一个数组中的数据的平均值，和。
4. 数组的数据进行排序
5. 给定一个数组，请去掉数组中的重复数据。
6. 冒泡排序算法

请参考：https://chuanke.baidu.com/v5508922-234347-1696511.html

# 11.包装类型与字符串

先看一个现象：

```
var t = 9.187;  // 编辑，设置
t.toString(); // => “9.187”
```

为什么一个数值类型有 toString 方法？

针对布尔类型、数值类型、字符串类型 JavaScript 都提供了对应的包装类型。当三种类型的变量在读取操作的时候，JavaScript 执行引擎会自动创建一个临时包装对象，帮助它可以访问包装对象的方法，使用完毕后立即销毁包装对象。

例如：

```
var t = 19;  // 创建数值类型的变量。
t.age = 19;// 创建临时对象，但是临时对象用完即毁
console.log(t.age); // undefined
// 临时创建了包装对象，并销毁了
```

## 11.1 Boolean 类型的包装对象

Boolean 类型是引用类型

创建方法：

```
var t = new Boolean(false);  // t 是引用类型
!!t  => true   非空对象转成 boolean 类型是 true
typeof t ;//=> object
 t instanceof Boolean;//=>true

 var m = true;
 m.toString(); //=> true  ,临时创建包装类型
 typeof m ;// => boolean
```

创建的 true 值的 Boolean 类型在转成 boolean 的简单类型的时候却成了 false，所以会引起很大误解。所以一般不会这样使用！！！！

## 11.2 Number 包装类型

Number 类型是引用类型
创建方法：

```
var t = new Number(0);  // t 是引用类型
typeof t ;//=> object
 t instanceof Number;//=>true
```

Number 类型的常用的方法

```
toString();//重写了 object 的 toString 方法，可以接受进制的参数。
 var t = 2;
t.toString(2);// =>“10”输出二进制的数值 2
 t.toString();// => “2”  默认是 10 进制
 t.toString(8);// =>“2”输出 8 进制的数值 2

//toLocalString();// 只是将数值转成字符串
t.toLocalString();// => “2”
valueOf(); //返回自身
toFixed(); //接受一个参数，限定小数的位数。
// toFixed()有 bug，所以参考：
http://qkxue.net/info/109055/toFixed-bug
IE8 及之前的版本对：[-0.94 到-0.5]  [0.5-0.94]返回 0，不是 1
 a = 1.23; a.toFixed(1) ; => ”1.2”
 a = 2.3;  a.toFixed(3);   => “2.300”会自动补充 0
toExponential(3);
 按指数的形式输出字符串
 传入的参数是保留小数的位数
 a = 3; a.toExponential(3); // “3.000*e+0”
toPrecision();
接受一个参数。参数是输出数据的总位数。
```

```
它会自动根据情况调用 toExponential 或者 toFixed 方法
 a = 96;
 a.toPrecision(3);  //=>96.0
 a.toPrecision(1);  // => "1e+2"
```

# 11.3 字符串 Sting 包装类型

字符串包装类型 String。跟 Boolean 和 Number 的包装类型一样，在字符串被读取的时候会自动创建一个包装类型用于包装类型方法的调用，而且会立即销毁。

## 11.3.1 创建 String 包装类型

var a = new String("123");
length 属性返回字符串字符的个数。
字符串是不可变的。对字符串的所有操作都会返回一个新字符串，元字符串不变。

## 11.3.2 字符串包装类型常用方法

- charAt(index)方法:获取某个索引位置的字符
- charCodeAt(index)方法：获取索引位置的字符 的编码
- stringValue[index]；类似数组的使用，获取索引位置的字符，IE8 以上才支持。
- concat()方法：连接字符串,可以传入多个参数。

```
"22".concat('2',333, 9);  //=> "2223339"
```

- slice() 类似数组的用法。字符串切片。
  一个参数，从索引位置到最后。负数从结尾结算。 '12345".slice(2);=>"345"
  两个参数，返回从第一个参数索引位置到最后一个参数索引位置之前的一个字符

```
'12345'.slice(2,4);=>"34"
```

- replace();
  可以传入两个参数。第一个是要替换的原字符串，第二个是替换的新字符串
  对原字符串没有影响，返回新字符串。

```
"12345".replace('4',"ss");// => "123ss5"
```

  后面会介绍正则表达式
- substring()
  截取字符串，对原字符串没有影响
  可以接受两个参数或者一个参数。第一个参数：截取索引位置，第二个是结束的位置。

都是正数的情况跟 slice 保持一致。负数有点不同。如果出现负数，substring 都将其看作 0 处理。

- substr()

  截取字符串，跟 substring 区别就是第二个参数是截取字符串的长度

  如果第二个参数是负数直接看做 0 处理

  **第一个参数是负数从末位计算。【IE8 有 bug】不要用**

- trim()

  方法会从一个字符串的两端删除空白字符

  并不影响原字符串本身，它返回的是一个新的字符串

```
var orig = ' foo '; console.log(orig.trim()); // 'foo'
```

  IE9 以上才支持

- split()

split()方法使用指定的分隔符字符串将一个 String 对象分割成字符串数组，以将字符串分隔为子字符串，以确定每个拆分的位置。

```
语法 :str.split([separator[, limit]])
```

separator：指定表示每个拆分应发生的点的字符串
limit：一个整数，限定返回的分割片段数量；

各种情况分类：

① 找到分隔符后，将其从字符串中删除，并将子字符串的数组返回。

```
'abcdef'.split('c');// =>["ab", "def"]
```

② 如果没有找到或者省略了分隔符，则该数组包含一个由整个字符串组成的元素。

```
'abcdef'.split('k'); // => ["abcdef"]
```

③ 如果分隔符为空字符串，则将 str 转换为字符数组。

```
'abcdef'.split(''); // => ["a", "b", "c", "d", "e", "f"]
```

④ 如果分隔符出现在字符串的开始或结尾，或两者都分开，分别以空字符串开头，结尾或两者开始和结束。因此，如果字符串仅由一个分隔符实例组成，则该数组由两个空字符串组成

```
'abcdef'.split('a'); //=>["", "bcdef"]
'abcdef'.split('f'); //=>["abcde", ""]
'abcdef'.split('abcdef');//=>["", ""]
```

- indexOf() 方法返回调用字符串对象中第一次出现的指定值的索引，开始在 fromIndex 进行搜索。如果未找到该值，则返回-1

  **语法格式：** str.indexOf(searchValue[, fromIndex])

  **参数：**

  searchValue 一个字符串表示被查找的值。

fromIndex 可选表示调用该方法的字符串中开始查找的位置。可以是任意整数。默认值为 0。如果 fromIndex < 0 则查找整个字符串（如同传进了 0）。如果 fromIndex >= str.length，则该方法返回 -1，除非被查找的字符串是一个空字符串，此时返回 str.length。

**返回值：**

指定值的第一次出现的索引; 如果没有找到 -1。

**案例**

```
"Blue Whale".indexOf("Blue"); // returns 0
"Blue Whale".indexOf("Blute"); // returns -1
"Blue Whale".indexOf("Whale", 0); // returns 5
"Blue Whale".indexOf("Whale", 5); // returns 5
"Blue Whale".indexOf("", 9); // returns 9
"Blue Whale".indexOf("", 10); // returns 10
"Blue Whale".indexOf("", 11); // returns 10
lastIndexOf()跟 indexOf 方法类似。要查询的是最后一个字符。
```

## 11.4 综合练习

- 截取字符串 abcdefg 的 efg
- 请编写代码实现字符串逆序。至少 2 种方法。
- 判断一个字符串中出现次数最多的字符，统计这个次数

- 问题：

输入两个字符串，从第一个字符串中删除第二个字符串中的所有字符。不可以使用 replace
<!--例如：输入“They are students” 和“aeiou”-->
<!--则删除之后的第一个字符串变成 “Thy r stdnts”-->

# 12.单体对象及日期类型

## 12.1 日期类型

Date 是 JavaScript 中处理日期的对象。它的值是从 1970.1.1 午夜零时起的毫秒数。
世界协调时间：UTC
创建日期对象可以通过构造函数：

语法

```
new Date();   // 当前时间
new Date(value);
 new Date(dateString);
new Date(year, month[, day[, hour[, minutes[, seconds[,
milliseconds]]]]]);
```

案例：

```
var today = new Date();
var today = new Date(1453094034000);
var birthday = new Date("December 17, 1995");
var birthday = new Date("1995-12-17T03:24:00");
var birthday = new Date(1995,11,17,3,24,0);
0-11 数字表示 1-12 月: var a= new Date(2006,5,6) 结果是 2006-6-6
0-6 表示星期
```

## 12.2 日期对象的常用方法

getDate()
根据本地时间返回指定日期对象的月份中的第几天（1-31）。
getDay()
根据本地时间返回指定日期对象的星期中的第几天（0-6）。
getFullYear()
根据本地时间返回指定日期对象的年份（四位数年份时返回四位数字）。
getHours()
根据本地时间返回指定日期对象的小时（0-23）。
getMilliseconds()
根据本地时间返回指定日期对象的毫秒（0-999）。
getMinutes()
根据本地时间返回指定日期对象的分钟（0-59）。
getMonth()
根据本地时间返回指定日期对象的月份（0-11）。
getSeconds()
根据本地时间返回指定日期对象的秒数（0-59）。
getTime()
返回从 1970-1-1 00:00:00 UTC（协调世界时）到该日期经过的毫秒数，对于 1970-1-1 00:00:00 UTC 之前的时间返回负值

所有的 get 换成 set 就是设置方法

toDateString()
以人类易读（human-readable）的形式返回该日期对象日期部分的字符串。
toISOString()
把一个日期转换为符合 ISO 8601 扩展格式的字符串。
toGMTString()
返回一个基于 GMT (UT) 时区的字符串来表示该日期。请使用 toUTCString() 方法代替。
toLocaleDateString()
返回一个表示该日期对象日期部分的字符串，该字符串格式与系统设置的地区关联（locality sensitive）。
toLocaleString()
返回一个表示该日期对象的字符串，该字符串与系统设置的地区关联（locality sensitive）。覆盖了 Object.prototype.toLocaleString() 方法。
toLocaleTimeString()
返回一个表示该日期对象时间部分的字符串，该字符串格式与系统设置的地区关联（locality sensitive）。
toString()
返回一个表示该日期对象的字符串。覆盖了 Object.prototype.toString() 方法。
toTimeString()
以人类易读格式返回日期对象时间部分的字符串。
toUTCString()
把一个日期对象转换为一个以 UTC 时区计时的字符串。
valueOf()
获取原始的时间毫秒数

## 12.3 数学对象 Math

JavaScript 提供了 Math 内置对象方便我们进行数学运算
它具有数学常数和函数的属性和方法。

- 常用属性：

```
Math.PI; //圆周率=> 3.141592653589793
Math.E   //自然对数底，数学中 e 的值 2.718281828459045
Math.LN10; //10 的自然对数 ，约等于 2.303.
Math.LN2; // 2 的自然对数，约等于 0.693
Math.LOG2E;//以 2 为底 e 的对数，约等于 1.443.
Math.LOG10E;//以 10 为底 e 的对数，约等于 0.434.
Math.SQRT1_2;1/2 的平方根，约等于 0.707.
```

```
Math.SQRT2;2 的平方根,约等于 1.414.
```

- 常用方法：

```
max()与 min()。求一组数的最大值和最小值。
Math.min(10,9,8,22);//=>8
Math.max(10,9,22);//22
舍入方法
Math.ceil();        // 向上舍入  Math.ceil(2.34);//3
Math.floor();       // 向下舍入   Math.floor(2.3);//2
Math.round();  //四舍五入    Math.round(2.7);//3
Math.random();获取随机数(0-1)
Math.abs(num);//求 num 绝对值
Math.exp(num);//求 e 的 num 次幂
Math.log(num);//求 num 的自然数对数
Math.pow(num,power);//求 num 的 power 次方
Math.sqrt(num);//求 num 的平方根
三角函数 sin(), cos(), tan(),asin(), acos(), atan()参数是弧度(0-2
π)
```

## 12.4 全局对象

全局对象 Global，是 JavaScript 的兜底对象，所有没有对象的方法和属性都会归到全局对象上。
在浏览器端：window 对象实现了 Global 对象，所以在浏览器环境中全局对象就是 window 对象。但是：在其他环境中不是这样的，比如 Nodejs 环境。

常用方法：

```
isNaN
parseInt
parseFloat
encodeURI: 对 uri 进行编码, uir 中特殊符号保留。
encodeURIComponent:直接进行编码
```

```
decodeURI   decodeURIComponent

eval

函数会将传入的字符串当做 JavaScript 代码进行执行。
```

# 13. JavaScript 高级预告

高级内容会包括：

- 函数高级
- 作用域深入
- 原型链
- 闭包
- 垃圾回收机制
- 异常处理
- 正则表达式
- JavaScript 面向对象继承
- JavaScript 模块化开发
-

请大家继续关注老马的传课官网的 JavaScript 高级课程。

老马联系方式
QQ：515154084
邮箱：malun666@126.com
微博：http://weibo.com/flydragon2010

百度传课：
https://chuanke.baidu.com/s5508922.html

微信：