

「学习笔记」JavaScript基础

原创 饭老板 前端fan 2020-09-06

收录于话题

#前端入门 4 #JavaScript基础 1



「学习笔记」JavaScript基础

前言

最近一直在跟着黑马教程学习JavaScript内容，遂把这一阶段的学习内容整理成笔记，巩固所

学知识，同时也会参考一些博客，书籍上的内容，查漏补缺，给自己充充电🔋🔋🔋🔋

文章内容如有错误，欢迎指正批评✓✓

工欲善其事，必先利其器，为了提高开发效率，选用VScode。

- **管理-设置-常用设置-字体** Consolas, '微软雅黑 Light', monospace
- **Chinese** 汉化Vscode
- **Prettier** 格式化代码(缩进2格)
- **Auto Rename Tag** 同步修改标签
- **HTML CSS Support** Html文档的CSS支持
- **HTML Snippets** 自动输入Html标签
- **JavaScript (ES6) code snippets** ES6语法支持
- **open in browser** 浏览器打开

编程基础

编程基础

「**计算机语言**」分为机器语言，汇编语言，高级语言。计算机内部最终执行的都是机器语言，由 **0** 和 **1** 这样的二进制数构成。

「**数据存储单位**」8bit(比特) = 1B(Byte)字节 千字节1KB = 1024B

「**翻译器**」高级语言编写的程序不能被计算机识别，需要经过转换，将源代码程序翻译成 **机器语言** 才能运行。浏览器里面的js解释器就是这样的一个翻译器。

「程序运行」

- 打开某个程序时，先从硬盘中把程序的代码加载到内存中
- CPU执行内存中的代码
- 注意：之所以要内存的一个重要原因，是因为cpu运行太快了，如果只从硬盘中读数据，会浪费cpu性能，所以，才使用存取速度更快的内存来保存运行时的数据。（内存是电，硬盘是机械）

初识JavaScript

「创始人」 布兰登·艾奇(Brendan Eich),起初命名为 **LiveScript** 后来与Sun公司合作改名为 **JavaScript** 。

「**JavaScript**」 运行在客户端的脚本语言，不需要编译，由js解释器(js引擎)逐行解释执行。Node.js也可以用于服务器端编程。

「**JavaScript组成**」 ECMAScript(JavaScript语法)、DOM(文档对象模型)、BOM(浏览器对象模型)

JavaScript的作用

- 表单动态校验(密码强度检测)
- 网页特效
- 服务端开发(Node.js)
- 桌面程序(Electron)、App(Cordova)、控制硬件-物联网(Ruff)、游戏开发(cocos2d-js)

「JavaScript书写位置」

JS有3种书写位置，分别为行内、内嵌和外部。

1. 行内式



```
<input type="button" value="点我试试"
onclick="alert('Hello World')" />
```

2. 内嵌式



```
<script>
    alert('Hello World~!');
</script>
```

3. 外部式



引用外部js文件

```
<script src = "my.js"></script>
```

「注释」

1. 单行注释



```
// 我是单行注释 （快捷键 ctrl + / ）
```

2. 多行注释



```
/*  
    获取用户年龄和姓名  
    并通过提示框显示出来  
    点击vscode左下角管理- 键盘快捷方式- 切换块注释  
    （默认快捷键 alt + shift + a）修改为 （ctrl + shift + /）  
*/
```

变量

变量的概念

「变量」是程序在内存中申请的一块用于存放数据的空间。变量是用于存放数据的容器，可以通过变量名获取数据，甚至修改数据。



变量的使用

「1. 声明变量」



```
// 1. 声明变量  
var num; // 声明一个 名称为 num 的变量
```

var是一个JS关键字，用来声明变量(variable变量的意思)。num是我们定义的变量名，可以通过变量名来访问内存中分配的空间。

「2. 赋值」



```
num = 10; // 给 num 这个变量赋值为 10
```

「3. 变量的初始化」

声明一个变量并赋值， 我们称之为变量的初始化。



```
var num = 10; // 声明变量并赋值为10
```

「4. 变量语法扩展」



```
// 1. 一个变量被重新赋值后，它原有的值会被覆盖掉，变量值以最后一次赋的值为准。  
var num = 10;  
num = 11;  
  
// 2. 同时声明多个变量(只需要写一个var, 多个变量名之间用英文逗号隔开)  
var num = 10, age = 15, name = 'fan';
```

声明变量特殊情况

情况	说明	结果
var age ; console.log (age);	只声明 不赋值	undefined
console.log(age)	不声明 不赋值 直接使用	报错
age = 10; console.log (age);	不声明 只赋值	10

「5. 变量命名规范」



数据类型

分为两类：简单数据类型(Number,String,Boolean,Undefined,Null)和复杂数据类型(object)。

简单数据类型	说明	默认值
Number	数字型,包含整型值和浮点型值	0
String	字符串型	""
Boolean	布尔值型	false
Undefined	var a;声明了变量a但是没有赋值,此时a = undefined	undefined
Null	var a = null;声明了变量a为空值	null

Number数字型

「数字型进制」



```
// 1.在JS中八进制前面加0, 十六进制前面加 0x
var num1 = 07;    // 对应十进制的7
// 2.十六进制数字序列范围: 0~9以及A~F
var num = 0xA;
```

「数字型范围」JavaScript中的数值有最大值和最小值

- 最大值: `Number.MAX_VALUE` , 值为: 1.7976931348623157e+308
- 最小值: `Number.MIN_VALUE` , 值为: 5e-32

- 特殊值：**Infinity** 无穷大 **-Infinity** 无穷小 **NaN** 代表一个非数字
- `isNaN()`:用来判断一个变量是否为非数字的类型。非数字型为`true`,数字型为`false`。

String字符串型



```
// 1.字符串型可以是引号中的任意文本，语法为 单引号 和 双引号
var msg = '我的名字叫';
var name = "fan";
```

「1. 字符串转义符」都是以 \ 开头，详细如下

转义符	说明
\n	换行符，n是 newline 的意思
\\	斜杆 \
\'	单引号 ‘
\"	双引号 "
\t	tab 缩进
\b	空格,b是 blank 的意思

「2. 字符串长度」

字符串是由若干字符组成的，这些字符的数量就是字符串的长度。



```
// 1.字符串型可以是引号中的任意文本，语法为 单引号 和 双引号
var msg = '我是帅气的饭老板';
console.log(msg.length); // 显示 8
```

「3. 字符串拼接」

多个字符串之间可以使用 + 进行拼接，其拼接方式为 字符串 + 任何类型 = 拼接之后的新字符串。

拼接前会把与字符串相加的任何类型转成字符串，再拼接成一个新的字符串



```
//1.1 字符串 "相加"
alert('hello' + ' ' + 'world'); // hello world
//1.2 数值字符串 "相加"
alert('100' + '100'); // 100100
```

//1.3 数值字符串 + 数值

```
alert('11' + 12); // 1112 +号口诀：数值相加，字符相连
```

// 1.4 字符串拼接加强

```
var age = 18;
```

```
alert("饭老板今年" + age + "岁了");
```

布尔型Boolean

布尔类型有两个值：true 和 false，其中 true 表示真（对），而 false 表示假（错）。

布尔型和数字型相加的时候，true 的值为 1，false 的值为 0。



```
console.log(true + 1) // 2
```

```
console.log(false + 1) // 1
```

Undefined 和 Null

一个变量声明后没有赋值会有一个默认值undefined(如果相连或者相加时，注意结果💎💎)



```
var variable;
```

```
console.log(variable); // undefined
```

```
console.log("你好" + variable); // 你好undefined
```

```
console.log(11 + variable); // NaN
```

```
console.log(true + variable); // NaN
```

一个变量声明并赋值null,里面存的值为空



```
var var2 = null;
```

```
console.log(var2); // null
```

```
console.log("你好" + var2); // 你好null
```

```
console.log(11 + var2); // 11
```

```
console.log(true + var2); // 1
```

获取变量类型及转换

◦ 检测变量的数据类型 `typeof`



```
var num = 10;
console.log(typeof num)//结果为 number
```

- **字面量** :是源代码中一个固定值的表示法，就是字面量如何去表达这个值。通过数据的格式特征可以判断数据的类型
- 有数字字面量:8,9,10
 - 字符串字面量:'饭老板', "前端开发"
 - 布尔字面量:true,false

「数据类型转换」

◦ 转换为字符串

方式	说明	案例
toString()	转成字符串	var num=1; alert(num.toString())
String()	强制转换	var num=1; alert(String(num))
加号拼接字符串	和字符串拼接的结果都是字符串	var num=1; alert(num+'我是字符串')

◦ 转换为数字型

方式	说明	案例
parseInt(String)函数	将string类型转成整数型	parseInt('11')
parseFloat(String)函数	将string类型转成浮点型	parseFloat('11.2')
Number() 强制转换函数	将string类型强制转换为数值型	Number('12')
js隐式转换 (- * /)	利用算数运算隐式转换为数值型	'12' - 0

◦ 转换为布尔型

代表空、否定的值会被转换为false，如''、0、NaN、null、undefined 其余值都会被转换为true

方式	说明	案例
Boolean()	其他类型转换为布尔值	Boolean('true')

关键字和保留字

指开发人员为变量、属性、函数、参数取得名字。标识符不能是关键字或保留字。

「标识符」

「关键字」指 JS本身已经使用了的字，不能再用它们充当变量名、方法名

包
括：break、case、catch、continue、default、delete、do、else、finally、for、function、if、in、instanceof、new、return、switch、this、throw、try、typeof、var、void、while、with 等。

「保留字」实际上就是预留的“关键字”，意思是现在虽然还不是关键字，但是未来可能会成为关键字，同样不能使用它们当变量名或方法名。

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

注意：如果将保留字用作变量名或函数名，那么除非将来的浏览器实现了该保留字，否则很可能收不到任何错误消息。当浏览器将其实现后，该单词将被看做关键字，如此将出现关键字错误。

运算符与流程控制

运算符(操作符)

「运算符」是用于实现赋值、比较和执行算数运算等功能的符号。常用运算符分类如下

- 算数运算符
- 递增和递减运算符
- 比较运算符
- 逻辑运算符
- 赋值运算符

「算数运算符」

运算符	描述	案例

+	加	10+20=30
-	减	10-20=-10
*	乘	10*20=200
/	除	10/20=0.5
%	取余(取模)	返回除法的余数9%2=1

。浮点数的精度问题



```
var result = 0.1 + 0.2;    // 结果不是 0.3, 而是 : 0.30000000000000004
console.log(0.07 * 100);   // 结果不是 7, 而是 : 7.000000000000001
```

- 浮点数值最高精度是17位小数，但是在进行算数运算时其精确度远远不如整数,所以不要直接判断两个浮点数是否相等！

。表达式与返回值

- 表达式：由数字、运算符和变量组成的式子。
- 返回值：每一个表达式经过相应的运算之后，会有一个最终结果，称为表达式的返回值

「递增和递减运算符」

递增和递减运算符必须配合变量使用。

。递增运算符



```
var num = 10;
alert(++num + 10);    // 21 使用口诀：先自加，后返回值

var num1 = 10;
alert(10 + num1++);   // 20 使用口诀：先返回原值，后自加
```



```
var num = 1;
var num2 = ++num + num++; // num = 2
console.log(num2); // 4

var num = 1;
var num1 = 1;
```

```
var num2 = num++ + num1++; // 1 + 1

console.log(num2); // 2


var num = 1;

var num2 = num++ + num++; // 1 + 2

console.log(num2); // 3
```

「比较运算符」

运算符	描述	案例	结果
<	小于号	1<2	true
>	大于号	1>2	false
>=	大于等于号(大于或者等于)	2 >= 2	true
<=	小于等于号(小于或者等于)	3 <= 2	false
==	判等号(会转型)	15 == '15'	true
!=	不等号	37 != 37	false
=== !==	全等 全不等(要求值和数据类型都一致)	37 === '37'	false

「逻辑运算符」

逻辑运算符是用来进行布尔值运算的运算符

短路运算:当有多个表达式（值）时,左边的表达式值可以确定结果时,就不再继续运算右边的表达式的值;

运算符	描述	案例	特点
&&	"逻辑与",简称"与" and	true && false	两边都是 true才返回 true
	"逻辑或",简称"或" or	true	有真为真
!	"逻辑非",简称"非" not	!true	取反

「赋值运算符」

运算符	描述	案例
=	直接赋值	var userName = 'fan'
+= -=	加减一个数后再赋值	var age=5; age+=5
= /= %=	乘、除、取模后再赋值	var age=5; age=5

「运算符优先级」

优先级	运算符	顺序
1	小括号	()

2	一元运算符	! ++ --
3	算数运算符	先* / % 后+ -
4	关系运算符	> >= < <=
5	相等运算符	== != === !==
6	逻辑运算符	先&& 后
7	赋值运算符	=
8	逗号运算符	,

流程控制

「流程控制」在一个程序执行的过程中，各条代码的执行顺序对程序的结果是有直接影响的。很多时候我们要通过控制代码的执行顺序来实现我们要完成的功能。流程控制主要有三种结构，分别是顺序结构、分支结构和循环结构，代表三种代码执行的顺序。

「分支流程控制」



```
// 1. 条件成立执行的代码语句
if (条件表达式) {
}

// 2.if else 语句
if (条件表达式) {
    // [如果] 条件成立执行的代码
} else {
    // [否则] 执行的代码
}

// 3. if else if 语句(多分支语句)
// 适合于检查多重条件。
if (条件表达式1) {
    语句1;
} else if (条件表达式2) {
    语句2;
} else if (条件表达式3) {
    语句3;
    ....
} else {
    // 上述条件都不成立执行此处代码
}
```

「三元表达式」



//如果表达式1为 true , 则返回表达式2的值, 如果表达式1为 false, 则返回表达式3的值
表达式1 ? 表达式2 : 表达式3;

「switch分支流程控制」它用于基于不同的条件来执行不同的代码。当要针对变量设置一系列的特定值的选项时, 就可以使用 switch。



```
switch( 表达式 ){  
    case value1:  
        // 表达式 等于 value1 时要执行的代码  
        break;  
    case value2:  
        // 表达式 等于 value2 时要执行的代码  
        break;  
    default:  
        // 表达式 不等于任何一个 value 时要执行的代码  
}
```

循环与代码规范

循环

运算符	描述
初始化变量	通常被用于初始化一个计数器,该表达式可以使用var关键字声明新的变量, 这个变量帮我们来记录次数。
条件表达式	用于确定每一次循环是否能被执行, 如果结果是true就继续循环, 否则退出循环
操作表达式	每次循环的最后都要执行的表达式。通常用于更新计数器变量的值



```
for( 初始化变量; 条件表达式; 操作表达式 ){  
    //循环体  
}
```

「执行流程」

- 1. 初始化变量, 初始化操作在整个 for 循环只会执行一次。

2. 执行条件表达式，如果为true，则执行循环体语句，否则退出循环，循环结束。
3. 执行操作表达式,此时第一轮结束。
4. 第二轮开始，直接去执行条件表达式（不再初始化变量），如果为 true，则去执行循环体语句，否则退出循环。
5. 继续执行操作表达式，第二轮结束。.....

「**双重for循环**」循环嵌套是指在一个循环语句中再定义一个循环语句的语法结构，例如在for循环语句中，可以再嵌套一个for 循环，这样的 for 循环语句我们称之为双重for循环。



```
for (外循环的初始; 外循环的条件; 外循环的操作表达式) {  
    for (内循环的初始; 内循环的条件; 内循环的操作表达式) {  
        需执行的代码;  
    }  
}
```

//for循环打印九九乘法表

```
var str = "";  
for (var i = 1; i <= 9; i++) {  
    for (var j = 1; j <= i; j++) {  
        str += j + "x" + i + "=" + j * i + "\t";  
    }  
    str += "\n";  
}  
console.log(str);
```

「while循环」



```
while (条件表达式) {  
    // 循环体代码  
}
```

// 1. 先执行条件表达式，如果结果为 true，则执行循环体代码；
// 如果为 false，则退出循环，执行后面代码
// 2. 执行循环体代码
// 3. 循环体代码执行完毕后，程序会继续判断执行条件表达式，



```
//计算1-100的累加和  
var i = 1;  
var sum = 0;  
while (i <= 100) {
```

```
    sum += i;
    i++;
}
console.log(sum);
```

「do-while循环」



```
do {
    // 循环体代码 - 条件表达式为 true 时重复执行循环体代码
} while(条件表达式);

// 先执行一次循环体代码，再执行条件表达式
// 计算100以内的偶数累加和

var i = 1;
var sum = 0;

do {
    if (i % 2 == 0) {
        sum += i;
    }
    i++;
} while (i <= 100);

console.log(sum);
```

「continue、break」

continue 关键字用于立即跳出本次循环，继续下一次循环（本次循环体中 continue 之后的代码就会少执行一次）。

break 关键字用于立即跳出整个循环（循环结束）。

代码规范

1. 标识符命名规范

- 变量、函数的命名必须要有意义
- 变量的名称一般用名词
- 函数的名称一般用动词

2. 操作符规范



```
// 操作符的左右两侧各保留一个空格

for (var i = 1; i <= 5; i++) {
```



```
if (i == 3) {  
    break; // 直接退出整个 for 循环，跳到整个for循环下面的语句  
}  
console.log('我正在吃第' + i + '个包子呢');
```

3. 单行注释规范



```
for (var i = 1; i <= 5; i++) {  
    if (i == 3) {  
        break; // 单行注释前面注意有个空格  
    }  
    console.log('我正在吃第' + i + '个包子呢');
```

4. 其他规范



```
//关键词 操作符空格  
if (true) {}  
for (var i = 0; i<=10; i++) {}
```

数组与函数

数组

「1. 数组的概念」 一组数据的集合，其中的每个数据被称作 **元素**，在数组中可以存放任意类型的元素。数组是一种将一组数据存储在单个变量名下的优雅方式。

「2. 创建数组」

- 利用new关键字创建数组；



```
var 数组名 = new Array([n]); //[]代表可选 若写n，则代表数组的长度  
var arr = new Array(); //创建了一个名为 arr 的空数组
```

- 利用数组字面量创建数组



```
// 1. 使用数组字面量方式创建空的数组
var 数组名 = []; // 若写n, 则代表数组的长度

// 2. 使用数组字面量方式创建带初始值的数组
// 3. 声明数组并赋值称为数组的初始化
var arr = ['1', '2', '3', '4'];
var arr2 = ['fan', true, 17.5]; // 数组中可以存放任意类型的数据
```

「3. 访问数组元素」

索引(下标): 用来访问数组元素的序号。索引从 0 开始



```
// 定义数组
var arrStus = [1, 2, 3];
// 获取数组中的第2个元素
alert(arrStus[1]);
// 如果访问数组时没有和索引值对应的元素(数组越界),
// 返回值为undefined
```

「4. 遍历数组」

把数组中的元素从头到尾都访问一次。



```
// 数组的长度, 默认情况下等于元素的个数
// 当我们数组里面的元素发生了变化, length属性跟着一起变
// 如果设置的length属性值大于数组的元素个数, 则会在数组末尾出现空白元素;
// 如果设置的length属性值小于数组的元素个数, 则会把超过该值的数组元素删除
var arr = ["red", "blue", "green"];
for (var i = 0; i < arr.length; i++) {
    console.log(arr[i]);
}

arr.length = 2;
```

```
console.log(arr);// red blue
```

「4. 数组中新增元素」

数组中可以通过以下方式在数组的末尾插入新元素；



```
// 1. 数组[数组.length] = 新数据;  
arr = [] //arr.length = 0;  
for (var i = 0; i < 10; i++) {  
    arr[arr.length] = '0';  
}  
console.log(arr);
```

「5. 案例」



```
// 1. 筛选数组 大于10的元素选出来放到新数组中  
var arr = [2, 0, 6, 1, 77, 0, 52, 0, 25, 7];  
var newArr = [];  
var j = 0;  
for (var i = 0; i < arr.length; i++){  
    if (arr[i] >= 10) {  
        newArr[j] = arr[i];  
        j++;  
    }  
}  
console.log(newArr);  
//第二种方法 利用数组长度的可变性  
for (var i = 0; i < arr.length; i++){  
    if (arr[i] >= 10) {  
        newArr[j] = arr[i];  
        j++;  
    }  
}
```



```
// 2. 翻转数组  
// 把旧数组索引号第4个取过来 (arr.length - 1), 给新数组索引号第0个元素 (newArr.length)  
// 我们采取 递减的方式 i--  
var arr = ['red', 'green', 'blue', 'pink', 'purple'];  
var newArr = [];
```

```
for(var i = arr.length - 1; i >= 0; i--) {  
    newArr[newArr.length] = arr[i]  
}  
console.log(newArr);
```



// 3. 数组转换为字符串 用"|" 或其他符号分割

// 需要一个新变量用于存放转换完的字符串str

//遍历取出数据加到str后面然后加上分隔符

```
var arr = ['red', 'green', 'blue', 'pink', 'purple'];  
var str = '';  
for(var i = 0; i < arr.length; i++) {  
    str += arr[i] + '|';  
}  
console.log(str);
```



// 4. 数组转换为字符串 用"|" 或其他符号分割

// 需要一个新变量用于存放转换完的字符串str

//遍历取出数据加到str后面然后加上分隔符

```
var arr = ['red', 'green', 'blue', 'pink', 'purple'];  
var str = '';  
for(var i = 0; i < arr.length; i++) {  
    str += arr[i] + '|';  
}  
console.log(str);
```

冒泡排序



```
function sort(arr) {  
  for(var i = 0; i < arr.length - 1; i++) {  
    for(var j = 0; j < arr.length - 1 - i; j++) {  
      if (arr[j] > arr[j+1]){  
        var temp = arr[j];  
        arr[j] = arr[j+1];  
        arr[j+1] = temp;  
      }  
    }  
  }  
  return arr;  
}  
var arr1 = sort([1,4,2,9]);  
console.log(arr1); // 1 2 4 9
```

函数

「1. 函数的概念」

封装了一段可被重复调用执行的代码块，通过函数可以实现大量代码的重复使用。函数是一种数据类型。

「2. 函数的使用」

- 声明函数



1. 通过function关键字定义函数 -- 命名函数

```
function 函数名() {  
    //函数体代码  
}  
// 1.1 function 是声明函数的关键字, 必须小写  
// 1.2 函数名 命名为动词形式 例: getSum
```

2. 通过函数表达式定义函数 --- 匿名函数

```
var fn = function() {};  
// 2.1 fn是变量名, 不是函数名  
// 2.2 fn是变量, 只不过变量存储的是函数  
// 2.3 函数表达式创建的函数可以通过 变量名(); 来调用  
// 2.4 函数表达式也可以定义形参和调用传入实参。
```



匿名函数使用的第2种方式 -- 匿名函数自调用

```
(function(){  
    alert(123);  
})();
```

◦ 调用函数



函数名(); // 函数声明后调用才会执行函数体代码

◦ 函数的封装

- 函数的封装是把一个或者多个功能通过函数的方式封装起来, 对外只提供一个简单的函数接口。



```
/*  
    例用封装函数计算1-100累加和  
*/  
function getSum() {  
    var sumNum = 0; // 准备一个变量, 保存累加和  
    for (var i = 1; i <= 100; i++) {  
        sumNum += i; // 把每个数值 都累加 到变量中  
    }  
    alert(sumNum);  
}  
// 调用函数  
getSum();
```

「3. 函数的参数」

- 形参：函数定义时候，传递的参数(实参值传递给形参,不用声明的变量)
- 实参：函数调用时候，传递的参数



```
//带参数的函数声明
function 函数名(形参1,形参2,形参3...) {
    //函数体
}
// 带参数的函数调用
函数名(实参1,实参2,实参3...);
```

- 「函数形参和实参数量不匹配时」

参数个数	说明
实参个数等于形参个数	输出正确结果
实参个数多于形参个数	只取到形参的个数
实参个数小于形参	多的形参定义为undefined,结果为NaN



```
function getSum(a, b, c) {
    return a + b + c;
}
// js中形参的默认值是undefined。
// 调用函数
var n = getSum(1, 2); // n = NaN
var n = getSum(1, 2, 3, 4); //1 + 2 +3 = 6
```

「4. 函数的返回值」

返回值：函数调用整体代表的数据；函数执行完成后可以通过return语句将指定数据返回

。



```
// 声明函数
function 函数名() {
    ...
    return 需要返回的值;
// 1. 函数遇到return会停止执行，并返回指定的值
// 1. 如果函数没有return 返回的值是undefined
```

```
}  
// 调用函数  
函数名(); //此时调用函数就可以得到函数体内return的值
```

break,continue,return的区别

- break: 结束当前的循环体 (如for、while)
- continue: 跳出本次循环, 继续执行下次循环
- return: 不仅可以退出 (函数体内) 循环, 还能够返回return语句中的值, 同时还可以 结束当前的函数体内的代码



//避免踩坑 return只能结束函数体内的代码

```
function breakDown() {  
  for (var i = 0; i < 10; i++) {  
    if (i == 5) {  
      return 1;  
    }  
    console.log(i);  
  }  
}  
breakDown();
```

//避免踩坑2 函数如果有return 则返回的是 return 后面的值 ;

// return d,a,b; 返回的是b的值

//如果函数没有 return语句, 则返回undefined



「5. arguments的使用」

当不确定有多少个参数传递的时候, 可以用 arguments 来获取。JS中, arguments 实际上它是当前函数的一个内置对象。所有函数都内置了一个 arguments 对象, arguments 对象中存储了传递的所有实参。arguments展示形式是一个伪数组, 因此可以进行遍历。

- 伪数组具有以下特点 :
 - 具有length属性
 - 按索引方式存储数据
 - 不具有数组的push, pop等方法




```
function fn() {  
    //arguments 里面存储了所有传递过来的实参  
    console.log(arguments); // [1,2,3...]  
    console.log(arguments[1]); // 2  
    console.log(arguments.length); // 3  
  
    //我们可以按照数组的方式 遍历argument  
}  
fn(1, 2, 3);
```



```
// 用伪数组 实现求最大值  
function getMax() {  
    var max = arguments[0];  
    for (var i = 1; i < arguments.length; i++) {  
        if (arguments[i] > arguments[0]) {  
            max = arguments[i];  
        }  
    }  
    return max;  
}  
var result = getMax(1,3,77,5,85)  
colsole.log(result);
```

作用域

作用域

「作用域」一段程序代码中所用到的名字并不总是有效和可靠的，而限定这个名字的可用性代码范围就是这个名字的作用域。

- 作用域的使用提高了程序逻辑的局部性，增强了程序的可靠性，减少了名字冲突。
- ES6之前作用域有两种 **全局作用域** 和 **局部作用域** (函数作用域)

「**全局作用域**」作用于所有代码执行的环境(整个 script 标签内部)或者一个独立的js文件。

「**局部作用域**」作用于函数内部的代码环境，就是局部作用域。因为跟函数有关系，所以也被称为 **函数作用域**。

「JS没有块级作用域」

- 块作用域由 {} 包括
- 在其他编程语言，if语句中，循环语句创建的变量，仅仅只能在本if语句，本循环语句中使用，如下❖❖❖❖



```
if(true){  
    int num = 123;  
    System.out.print(num); //123  
}  
System.out.print(num); //报错
```

- 以上java代码会报错，因为代码中 {}是一块作用域，其中声明的变量num，在{}之外不能使用，而JavaScript代码则不会报错
- Js中没有块级作用域（在ES6之前）



```
if(true){  
    var num = 123;  
    console.log(num); // 123  
}  
console.log(num); // 123
```

变量的作用域

在JavaScript中，根据作用域的不同，变量可以分为两种:❖❖

- **全局变量**
- **局部变量**

「**全局变量**」在全局作用域下声明的变量(在函数外部定义的变量)

- 全局变量在代码的任何位置都可以使用
- 在全局作用域下 var 声明的变量 是全局变量
- 特殊情况下，在函数内不使用var声明的变量也是全局变量(不建议使用)。

「**局部变量**」在局部作用域下声明的变量(在函数内部定义的变量)

- 局部变量只能在函数内部使用

- 在函数内部 `var` 声明的变量是局部变量
- 函数的形参实际上就是局部变量

「全局变量和局部变量的区别」

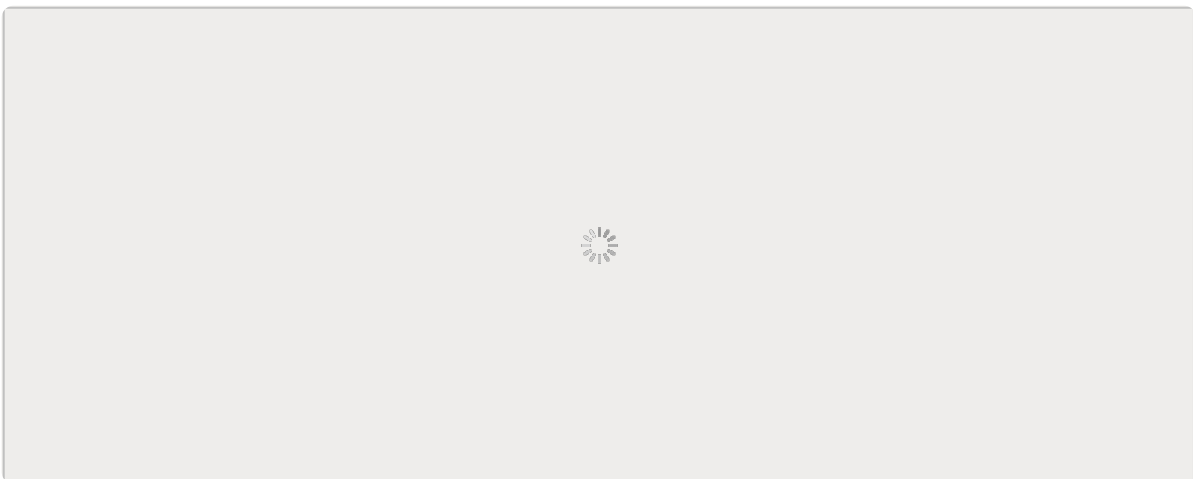
- **全局变量**：在任何一个地方都可以使用，只有在浏览器关闭时才会销毁，因此比较占内存
- **局部变量**：旨在函数内部使用，当其所在的代码块被执行时，才会被初始化；当代码块运行结束后，就会被销毁，因此更节省内存空间。

作用域链

「作用域链」只要是代码都在一个作用域中，写在函数内部的局部作用域，未卸载仍和行数内部即在全局作用域中；如果函数中还有函数，那么在这个作用域中又可以诞生一个作用域；根据「**内部函数可以访问外部函数变量**」的这种机制，用链式查找决定哪些数据能被内部函数访问，就称作作用域链。



```
function f1() {  
    var num = 123;  
    function f2() {  
        console.log( num );  
    }  
    f2();  
}  
var num = 456;  
f1();
```



作用域链 采取就近原则的方式来查找变量最终的值



```
var a = 1;
```

```
function fn1() {  
    var a = 2;  
    var b = '22';  
    fn2();  
    function fn2() {  
        var a = 3;  
        fn3();  
        function fn3() {  
            var a = 4;  
            console.log(a); //a的值 4  
            console.log(b); //b的值 '22'  
        }  
    }  
}  
fn1();
```



预解析

「**预解析相关概念**」JavaScript代码是由浏览器中的JavaScript解析器来执行的。JavaScript解析器在运行JavaScript代码的时候分为两步:预解析和代码执行。

- 「**预解析**」在当前作用域下，JS代码执行之前，浏览器会默认把带有 var 和 function 声明的变量在内存中进行提前声明或定义。
- 「**代码执行**」从上往下执行JS语句

预解析会把变量和函数的声明在代码执行之前完成，预解析也叫做变量、函数提升。

「**变量预解析(变量提升)**」变量的声明会被提升到当前作用域的最上面，变量的赋值不提升。



```
console.log(num); // 结果是多少？
```

```
var num = 10; // ?
```

相当于

```
var num;
```

```
console.log(num); // 结果是 undefined
```

```
num = 10;
```



结果： `undefined`

注意： 变量提升只提升声明，不提升赋值。

「函数预解析(函数提升)」 函数的声明会被提升到当前作用域的最上面，但是不会调用函数。



```
fn();
```

```
function fn() {  
    console.log('打印');
```

```
}
```



结果： 控制台打印字符串 --- "打印"

注意： 函数声明代表函数整体，所以函数提升后，函数名代表整个函数，但是函数并没有被调用！

「函数表达式声明函数问题」



函数表达式创建函数，会执行变量提升，此时接收函数的变量名无法正确的调用

```
fn();
```

```
var fn = function(){  
    console.log("想不到吧");  
}
```



结果: 报错提示 `"fn is not a function"`

解释： 该段代码执行之前，会做变量声明提升，fn在提升之后的值

是`undefined`；而fn调用是在fn被赋值为函数体之前，

此时fn的值是`undefined`，所以无法被调用。



预解析案例1

```
var num = 10;
fun();

function fun(){
    console.log(num);
    var num = 20;
}
```

相当于执行了以下操作 结果打印 `undefined`

```
var num;

function fun(){
    var num;
    console.log(num);
    num = 20;
}
num = 10;
fun();
```



预解析案例2

```
var a = 18;
f1();

function f1(){
    var b = 9;
    console.log(a);
    console.log(b);
    var a = '123';
}
```

相当于执行了以下操作 结果为 `undefined 9`

```
var a;
function f1(){
    var b;
    var a;
    b = 9;
    console.log(a);
    console.log(b);
    a = '123';
}
a = 18;
f1();
```



预解析案例3

```
f1();
console.log(c);
console.log(b);
console.log(a);

function f1() {
  var a = b = c = 9;
  console.log(a);
  console.log(b);
  console.log(c);
}
```

相当于执行了以下操作 结果为 9 9 9 9 9 "报错--a is not defined"

```
function f1() {
  var a;
  a = b = c = 9;
  //相当于 var a = 9; b=9; c=9; b和c 直接赋值，没有var声明，当全局变量看。
  // 差异：集体声明 var a = 9,b = 9, c = 9;
  console.log(a);
  console.log(b);
  console.log(c);
}
f1();
console.log(c);
console.log(b);
console.log(a);
```

对象

对象的概念

「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，在对象中常用方法来表示（常用动词）

「为什么需要对象」

- 保存一个值时，可以使用变量，保存多个值（一组值）时，可以使用数组，如果保存一个的完整信息呢？
- 为了更好地存储一组数据，对象应用而生；对象中为每项数据设置了属性名称，可以访问数据更语义化，数据结构清晰，表意明显，方便开发者使用。



```
var obj = {  
  "name": "fan",  
  "sex": "male",  
  "age": 18,  
  "height": 155  
}
```

创建对象的三种方式

「1. 利用字面量创建对象」使用对象字面量创建对象：

- 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）



```
// star 就是创建的对象  
var star = {  
  name : 'pink',  
  age : 18,  
  sex : '男',  
  sayHi : function() {  
    alert('大家好啊');  
  }  
};
```

- 对象的使用
 - 对象的属性：对象中 **存储具体数据** 的“键值对”中的键称为对象的属性，即对象中存储具体数据的项。
 - 对象的方法：对象中 **存储函数** 的“键值对”中的“键”称为对象的方法，即对象中存储函数的项。

- 访问对象的属性：对象里面的属性调用：`对象.属性名`；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上引号。
- 调用对象的方法：`对象.方法名()`；
- 变量、属性、函数、方法总结：
 - ①变量：单独声明赋值，单独存在
 - ②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。
 - ③方法：方法是对象的一部分，函数不是对象的一部分，函数是单独封装操作的容器。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。
 - ④函数：单独存在的，通过"`函数名()`"的方式就可以调用。



```
console.log(star.name)      // 调用名字属性
console.log(star['name'])   // 调用名字属性

star.sayHi();
```

「2. 利用new Object创建对象」

◦ 创建空对象



通过内置构造函数`Object`创建对象，此时`andy`变量已经保存了创建出来的空对象

```
var andy = new Object();
```

◦ 给空对象添加属性和方法



通过对象操作属性和方法的方式，来为对象增加属性和方法

```
andy.name = 'pink';
andy.age = 18; // andy.age = 19修改对象属性
andy.sex = '男'; // andy.phoneNum = 110 添加属性
andy.sayHi = function() {
  alert('大家好');
}
obj.sayHi();调用对象的方法 //第二种写法 obj['sayHi']();

// Object()第一个字母大写；
//new Object() 需要new关键字，使用的格式:对象.属性 = 值
```

「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出来，然后封装到这个函数里面。

◦ 构造函数的封装格式：



```
function 构造函数名(形参1, 形参2, 形参3...) {  
    this.属性名1 = 参数1;  
    this.属性名2 = 参数2;  
    this.属性名3 = 参数3;  
    this.方法名 = 函数体;  
}
```

◦ 构造函数的调用格式



```
var obj = new 构造函数名(实参1, 实参2, 实参3);
```

// 以上代码中，obj即接收到构造函数创建出来的对象。

注意事项：

1. 构造函数约定首字母大写
 2. 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
 3. 构造函数中不需要return返回结果
 4. 但我们创建对象的时候，必须用new 来调用构造函数
-
1. 其他：构造函数如Stars(),抽取了对象的公共部分，封装到了函数里面，它泛指某一大类(class)
 2. 创建对象，如new Stars();特指某一个，利用new关键字创建对象的过程我们也称为对象实例化

◦ new关键字的作用(面试题)

- 1.在构造函数代码开始执行之前，创建一个空对象；
- 2.修改this的指向，把this指向创建出来的空对象；
- 3.执行构造函数内的代码，给这个新对象添加属性和方法
- 4.在函数完成之后，返回这个创建出来的新对象(所以构造函数里面不需要return)



// 工厂函数创建对象 这个把创建好的对象返回给函数调用处

```
function createPerson(name, age, job) {  
    var person = new Object();
```

```
person.name = name;
person.age = age;
person.job = job;
person.sayHi = function(){
    console.log('Hello,everyBody');
}
return person;
}
var p1 = createPerson('张三', 22, 'actor');
```

遍历对象



`for...in` 语句用于对数组或者对象的属性进行循环操作。

其语法如下：

```
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
}
```

内置对象

「**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**

前面两种对象是JS基础内容，属于ECMAScript;第三个浏览器对象属于JS独有的，JS API讲解内置对象就是指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非必要的功能(属性和方法),内置对象最大的优点就是帮助我们快速开发。

「**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。

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

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。

属性、方法名	功能
--------	----

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
Math.random()	获取范围在[0,1)内的随机值

- 获取指定范围的随机整数



```
function getRandom(min, max) {  
    return Math.floor(Math.random() * (max - min + 1)) + min;  
}
```

日期对象

Date 对象和 Math 对象不一样，Date是一个构造函数，所以使用时需要实例化后才能使用其中具体方法和属性。Date 实例用来处理日期和时间

◦ 使用Date实例化日期对象

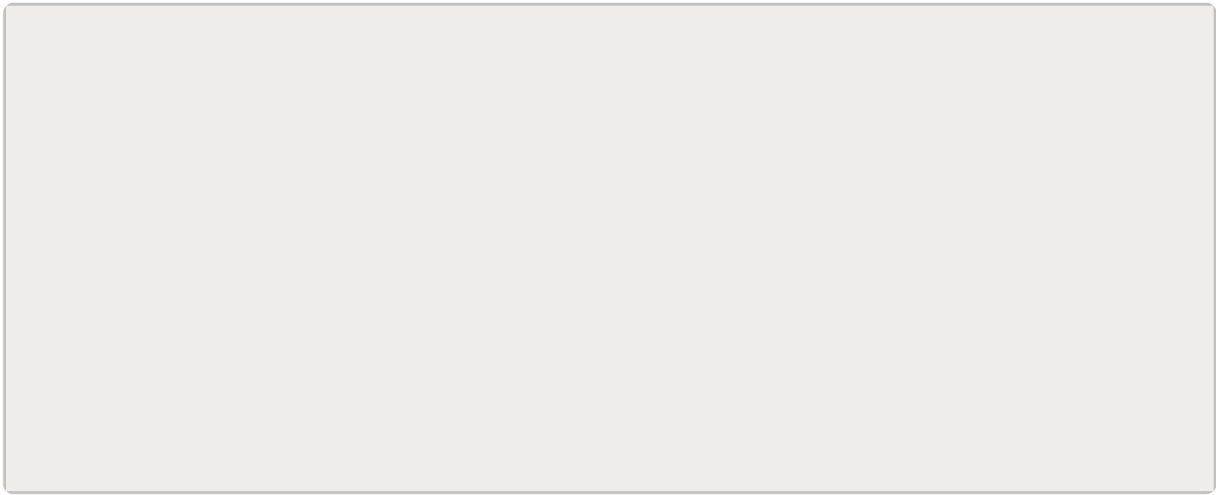
- 获取当前时间必须实例化
- 获取指定时间的日期对象



```
var now = new Date();  
var future = new Date('2020/10/1')  
// 注意：如果创建实例时并未传入参数，则得到的日期对象是当前时间对应的日期对象
```

◦ 续上

- 使用Date实例的方法和属性
- getMonth()方法获取到的月份 + 1 = 当月



```
//参数常用写法 数字型或者字符串型 '2019-10-1 8:8:8'
var date1 = new Date(2019,10,1);
```



```
//日期格式化
```

```
    // 格式化日期 年 月 日
var date = new Date();
console.log(date.getFullYear()); //返回当前日期的年 2020
console.log(date.getMonth() + 1); //月份 返回的月份小1个月，记得月份加1呦
console.log(date.getDate()); //返回的是 几号
console.log(date.getDay()); //周一返回的是1 周六返回的是6 周日返回的是0
//我们写一个 2020年 9月 6日 星期日
var year = date.getFullYear();
var month = date.getMonth() + 1;
var dates = date.getDate();
var day = date.getDay();
if (day == 0) {
    day = "星期日";
}
console.log("今天
是" + year + "年" + month + "月" + dates + "日" + day);
```



```
//格式化日期 时分秒
var date = new Date();
console.log(date.getHours()); //时
console.log(date.getMinutes()); //分
console.log(date.getSeconds()); //秒
```

```
//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();
    var h = h < 10 ? "0" + h : h;

    var m = time.getMinutes();
    var m = m < 10 ? "0" + m : m;

    var s = time.getSeconds();
    var s = s < 10 ? "0" + s : s;
    return h + ":" + m + ":" + s;
}
console.log(getTimer());
```

◦ 获取Date日期总的毫秒数(时间戳)

基于1970年1月1日(世界标准世界)起的毫秒数



```
// 实例化Date对象
var now = new Date();
// 1. 用于获取对象的原始值
console.log(now.valueOf())
console.log(now.getTime())
// 2. 简单写可以这么做 (最常用的)
var now = + new Date();
// 3. HTML5中提供的方法, 有兼容性问题
var now = Date.now();
```



倒计时案例：

1. 输入的时间减去现在的时间就是剩余的时间，即倒计时。
2. 用时间戳来做，用户输入时间总的毫秒数减去现在时间的总的毫秒数，得到的就是剩余时间的毫秒数
3. 把剩余时间总的毫秒数转换为天、时、分、秒（时间戳转换时分秒）
转换公式如下：

```
d = parseInt(总秒数/60/60/24) // 计算天数
h = parseInt(总秒数/60/60%24) // 计算小时
m = parseInt(总秒数/60%60); // 计算分钟
s = parseInt(总秒数%60); // 计算当前秒数
```



// 倒计时案例 封装函数实现

```
function countdown(time) {
    var nowTime = +new Date(); // 返回的是当前时间总的毫秒数
    var inputTime = +new Date(time); // 返回的是用户输入时间总的毫秒数
    var times = (inputTime - nowTime) / 1000; // times是剩余时间总的秒数
    var d = parseInt(times / 60 / 60 / 24); // 天
    d = d < 10 ? "0" + d : d;
    var h = parseInt((times / 60 / 60) % 24); // 时
    h = h < 10 ? "0" + h : h;
    var m = parseInt((times / 60) % 60); // 分
    m = m < 10 ? "0" + m : m;
    var s = parseInt(times % 60); // 当前的秒
    s = s < 10 ? "0" + s : s;
    return d + "天" + h + "时" + m + "分" + s + "秒";
}
console.log(countdown("2020-10-1 18:00:00"));
var date = new Date();
console.log(date);
```

数组对象

「创建数组的两种方式」

- 1. 字面量方式 `var arr = [1, "test", true];`
- 2. 实例化数组对象 `new Array() var arr = new Array();`
 - 注意：上面代码中arr创建出的的是一个空数组，如果需要使用构造函数Array创建非空数组，可以在创建数组时传入参数
 - 如果只传入一个参数(数字)，则参数规定了数组的长度。
 - 如果传入了多个参数，则参数称为数组的元素。

「检测是否为数组」

- 1. instanceof 运算符
 - instanceof 可以判断一个对象是否是某个构造函数的实例



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

```
console.log(obj instanceof Array); // false
```

◦ 2. Array.isArray()

- 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(Array.isArray(arr)); // true
console.log(Array.isArray(obj)); // false
```

◦ 3. 注意 typeof用法

- typeof 用于判断变量的类型



```
var arr = [1, 23];
console.log(typeof arr) // object 对象arr是构造函数的实例因此是对象数据类型
```

「添加删除数组元素的方法」

- 数组中有进行增加、删除元素的方法, 部分方法如下表💡💡



```
var arr = [1, 2, 3];
console.log(arr.push(4, 5)); // 5 向数组末尾添加元素
arr.pop(); // 删除数组最后一个值并返回
console.log(arr); // [1,2,3,4]

// 向数组的开头添加元素并返回数组长度
console.log(arr.unshift(5, 6)); // 6 数组变为[5,6,1,2,3,4]
console.log(arr.shift()); // 5 删除数组开头的元素并返回该值
```


「数组排序」

◦ 数组中有对数组本身排序的方法，部分方法如下表

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

◦ 注意：sort方法需要传入参数(函数)来设置升序、降序排序

- 如果传入“function(a,b){ return a-b;}”，则为升序
- 如果传入“function(a,b){ return b-a;}”，则为降序



```
// 踩坑 数组sort()排序(冒泡排序) return a - b 则升序
// 写法固定 参考如下
var arr1 = [13,4,77,1,7];
arr1.sort(function(a,b){
    return a-b;
});
console.log(arr1);
```

「数组索引方法」

◦ 数组中有获取数组指定元素索引值的方法，部分方法如下表



```
var arr = [1, 2, 3, 4, 5, 4, 1, 2];
// 查找元素2的索引
console.log(arr.indexOf(2)); // 1
// 查找元素1在数组中的最后一个索引
console.log(arr.lastIndexOf(1)); // 6
```

「数组转换为字符串」

◦ 数组中有把数组转化为字符串的方法，部分方法如下表

- 注意：join方法如果不传入参数，则按照“，”拼接元素



```
var arr = [1, 2, 3, 4];
var arr2 = arr;
var str = arr.toString(); // 将数组转换为字符串
console.log(str); // 1,2,3,4

var str2 = arr2.join("|"); // 按照键入字符将数组转换为字符串
console.log(str2);
```

「其他方法」



```
var arr = [1, 2, 3, 4];
var arr2 = [5, 6, 7, 8];
var arr3 = arr.concat(arr2);
console.log(arr3); // [1,2,3,4,5,6,7,8]

// slice(begin,end) 是一种左闭右开区间 [1,3)
// 从索引1开始截取，到索引3之前
var arr4 = arr.slice(1, 3);
console.log(arr4); // [2,3]

var arr5 = arr2.splice(0, 3);
console.log(arr5); // [5,6,7]
console.log(arr2); // [8]    splice()会影响原数组
```

字符串对象

「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。



```
// 下面代码有什么问题？  
var str = 'andy';  
console.log(str.length); // 4
```

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;
```

「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。
- 当重新给字符串变量赋值的时候，变量之前保存的字符串不会被修改，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

「根据字符返回位置」

- 字符串通过基本包装类型可以调用部分方法来操作字符串，以下是返回指定字符的位置的方法：



```
var str = "anndy";
console.log(str.indexOf("d")); // 3
// 指定从索引号为4的地方开始查找字符"d"
console.log(str.indexOf("d", 4)); // -1

console.log(str.lastIndexOf("n")); // 2
```

案例：查找字符串"abcfoxyozopp"中所有o出现的位置以及次数

1. 先查找第一个o出现的位置
2. 然后 只要indexOf 返回的结果不是 -1 就继续往后查找
3. 因为indexOf 只能查找到第一个，所以后面的查找，利用第二个参数，当前索引加1，从而继续查找



```
var str = "oabcfoxyozopp";
var index = str.indexOf("o");
var num = 0;
while (index !== -1) {
  console.log(index);
  num++;
  index = str.indexOf("o", index + 1);
}
```

「根据位置返回字符」

- 。字符串通过基本包装类型可以调用部分方法来操作字符串，以下是根据位置返回指定位置上的字符：





```
// 根据位置返回字符
// 1. charAt(index) 根据位置返回字符
var str = 'andy';
console.log(str.charAt(3)); // y
// 遍历所有的字符
for (var i = 0; i < str.length; i++) {
    console.log(str.charAt(i));
} // a n d y

// 2. charCodeAt(index)
// 返回相应索引号的字符ASCII值 目的：判断用户按下了那个键
console.log(str.charCodeAt(0)); // 97
// 3. str[index] H5 新增的
console.log(str[0]); // a
```

◦ 案例：判断一个字符串 'abcoefoxyozzopp' 中出现次数最多的字符，并统计其次数

1. 核心算法：利用 charAt() 遍历这个字符串
2. 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
3. 遍历对象，得到最大值和该字符 注意：在遍历的过程中，把字符串中的每个字符作为对象的属性存储在对象中，对应的属性值是该字符出现的次数



```
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);
```

「字符串操作方法」

- 字符串通过基本包装类型可以调用部分方法来操作字符串，以下是部分操作方法：



```
// 字符串操作方法
// 1. concat('字符串1','字符串2'....)
var str = 'andy';
console.log(str.concat('red')); // andyred

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

◦ 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，其使用格式如下：



字符串.replace(被替换的字符串, 要替换为的字符串);

◦ 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。
- 其使用格式如下：



字符串.split("分割字符")



```
// 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")); // bndyandy
// 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); // abc*ef*xy*zz*pp

// 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(",")); // [red,pink,blue]
var str3 = "red&pink&blue";
console.log(str3.split("&")); // [red,pink,blue]
```

简单数据类型和复杂数据类型

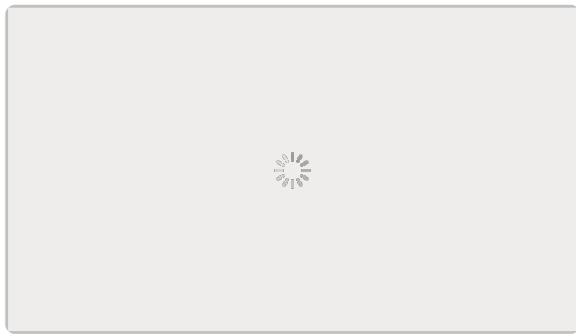
「简单类型（基本数据类型、值类型）」：在存储时变量中存储的是值本身，包括string, number, boolean, undefined, null

「复杂数据类型（引用类型）」：在存储时变量中存储的仅仅是地址（引用），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；

「堆栈」

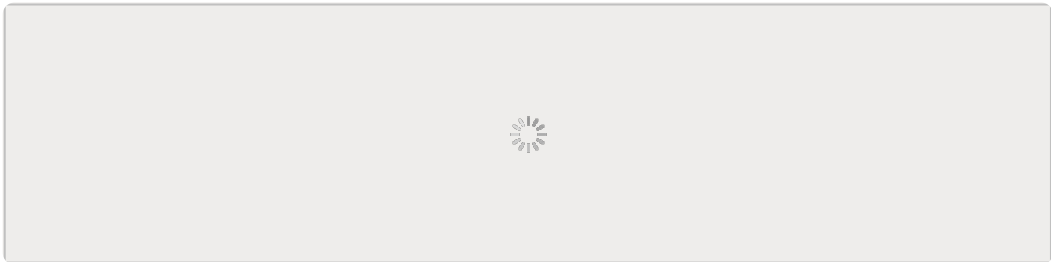
。堆栈空间分配区别：

- 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。



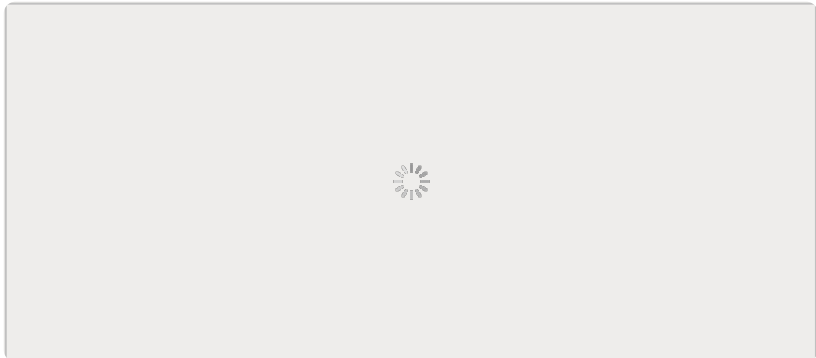
◦ 简单数据类型的存储方式

- 值类型变量的数据直接存放在变量（栈空间）中



◦ 复杂数据类型的存储方式

- 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



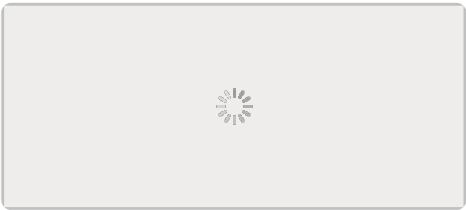
「简单类型传参」

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。



```
function fn(a) {  
  a++;  
  console.log(a);  
}  
var x = 10;  
fn(x);  
console.log(x);
```


运行结果如下



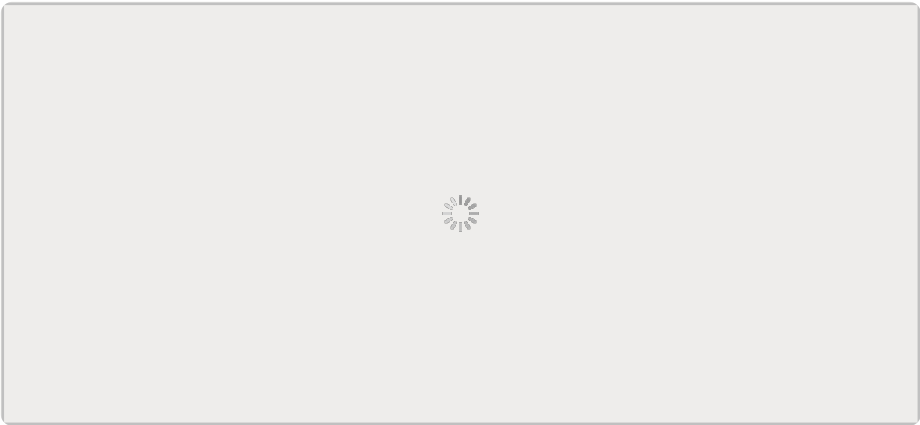
「复杂数据类型传参」

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。



```
function Person(name) {  
    this.name = name;  
}  
function f1(x) { // x = p  
    console.log(x.name); // 2. 这个输出什么 ?  
    x.name = "张学友";  
    console.log(x.name); // 3. 这个输出什么 ?  
}  
var p = new Person("刘德华");  
console.log(p.name);    // 1. 这个输出什么 ?  
f1(p);  
console.log(p.name);    // 4. 这个输出什么 ?
```

运行结果如下：



喜欢此内容的人还喜欢

安康发布新冠肺炎疫情防控温馨提示！安康发布新冠肺炎疫情防控温馨提示！安康日报



能当职业选手？浇你盆冷水！劝退警告来了！能当职业选手？浇你盆冷水！劝退警告共青团中央



【万字长文详解】Python库collections,【万字长文详解】Python库collections AI入门学习

