# JavaScript基础

一、网页、网站和应用程序

网页：单独的一个页面

网站：一些列相关的页面组成到一起

应用程序：可以和用户产生交互，并实现某种功能。例如百度脑图（思维导图，类似流程图一样），在web上，它的使用跟window一些软件一样。

二、演示JavaScript的强大

http://impress.github.io/impress.js/

http://naotu.baidu.com/

https://codecombat.com/

https://ide.codemao.cn/ 编程猫

blockly迷宫不需要翻墙

https://blockly.uieee.com/ 谷歌开源的游戏

三、JavaScript现在的意义(应用场景)

JavaScript 发展到现在几乎无所不能。

1. 网页特效

2. 服务端开发(Node.js)

3. 命令行工具(Node.js)

4. 桌面程序(Electron)

5. App(Cordova)

6. 控制硬件-物联网(Ruff)

7. 游戏开发(cocos2d-js)

四、JavaScript和HTML、CSS的区别

1. HTML：提供网页的结构，提供网页中的内容。结构

2. CSS: 用来美化网页。表现

3. JavaScript: 可以用来控制网页内容，给网页增加动态的效果。行为

五、JavaScript初体验

1、JavaScript的书写位置

（1）写在行内

|  |
| --- |
| <input type="button" value="按钮" onclick="alert(**'**Hello World**'**)" />  这里不能使用双引号，会跟html中的双引号冲突 |

（2）写在script标签中

|  |
| --- |
| <head>  <script>  alert('Hello World!');  </script>  </head>  会按顺序执行，先执行hello，然后加载body的内容 |

（3）写在外部js文件中，在页面引入

|  |
| --- |
| <script src="main.js"></script>  注意点  引用外部js文件的script标签中不可以写JavaScript代码  可以写，写了等等会被替换。 |

2、在F12上Network 中可以模仿网速，在等待服务器反馈结果的时候，网页是一片空白的

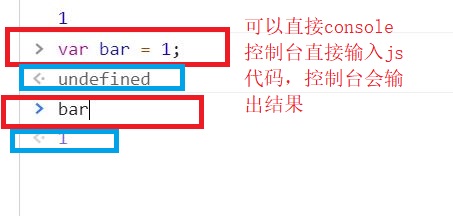
3、浏览器中JavaScript解释器也被叫做JavaScript引擎。解释一行执行一行，是一行行执行的。 CPU不认识javaScript，解释器就会翻译成CPU认识的。

4、在html中<script></script>标签如果放在head中的语句会按顺序执行，函数不会，然后在body中的其他元素要等script执行完，才绘制出来，**一般是放在body标签的底部，不是在body标签之外，不然会影响网页打开的速度。这个要验证一下。**

onclick = “alert(“hello!”)”，alert中的双引号会跟html中的双引号冲突？？？验证之后确实是这样。

<script></script>引入的js文件，会直接作为标签中的内容，会替代到原来的内容。

5、要避免在表达式写一个固定的值，尽量使用变量来代替，这样后续容易维护和修改。



六、计算机组成

1、软件

- **应用软件：浏览器(Chrome/IE/Firefox)**、QQ、Sublime、Word

- 系统软件：Windows、Linux、mac OSX

2、硬件

- 三大件：CPU、内存、硬盘 -- 主板

- 输入设备：鼠标、键盘、手写板、摄像头等

- 输出设备：显示器、打印机、投影仪等



七、变量

1、什么是变量

什么是变量

变量是计算机内存中存储数据的**标识符**，根据变量名称可以获取到**内存中**存储的数据。

为什么要使用变量

使用变量可以方便的获取或者修改内存中的数据。

2、如何使用变量

在JavaScript中声明变量的时候并没有确定变量的类型，它是**弱类型**语言，在执行代码的过程中，会确定变量的类型。

（1）var声明变量

|  |
| --- |
| var age;  **只是声明，不给它赋值，它是一个 undefined 类型，typeof age** |

（2）变量的赋值

|  |
| --- |
| var age;  age = 18; |

（3）同时声明多个变量

|  |
| --- |
| var age, name, sex;  age = 10;  name = 'zs'; |

（4）同时声明多个变量并赋值

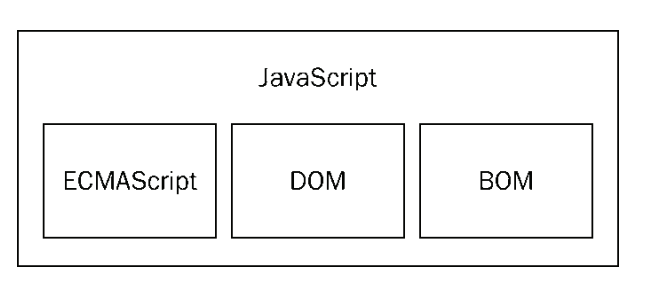
|  |
| --- |
| var age = 10, name = 'zs'; |

（5）如果在**同一个作用域**下，你**重复声明**一个 JavaScript 变量，它将不会丢失其值。

3、变量在内存中的存储

|  |
| --- |
| var age = 18;  1496981558575 |

八、JavaScript的组成



（1）ECMAScript - JavaScript的核心

ECMA 欧洲计算机制造联合会，定义了JavaScript的语法规范。

JavaScript的核心，描述了语言的基本语法和数据类型，**ECMAScript是一套标准**，定义了一种语言的标准与具体实现无关

（2）BOM - 浏览器对象模型

一套操作浏览器功能的API。

通过BOM可以操作浏览器窗口，比如：弹出框、控制浏览器跳转、获取分辨率等。

（3）DOM - 文档对象模型

一套操作页面元素的API

DOM可以把HTML看做是**文档树**，通过DOM提供的API可以对树上的节点进行操作。

4、var作用域

通过 var 声明的变量没有块级作用域。在语句块里声明的变量作用域是其所在的函数或者 script 标签内，你可以在语句块外面访问到它。换句话说，语句块不会生成一个新的作用域。

例如：

var x = 1;

{

var x = 2;

}

console.log(x); // 输出 2

该会输出 2，因为块中的 var x 语句与块前面的 var x 语句作用域相同，它们的作用域都是在script标签内。

5、变量提升

（1）变量声明，无论发生在何处，都在执行任何代码之前进行处理。在代码中的**任意位置**声明变量总是等效于在**代码开头**声明。在同一个作用域中。

|  |
| --- |
| function do\_something() {  console.log(bar); // undefined  var bar = 111;  console.log(bar); // 111  } |
| // is implicitly understood as:  function do\_something() {  var bar;  console.log(bar); // undefined  bar = 111;  console.log(bar); // 111  } |

（2）将赋值给未声明变量的值在执行**赋值**时将其隐式地创建为全局变量（它将成为全局对象的属性）。

|  |
| --- |
| function x() {  y = 1; // 在严格模式（strict mode）下会抛出 ReferenceError 异常  var z = 2;  }  x();  console.log(y); // 打印 "1"  console.log(z); // 抛出 ReferenceError: z 未在 x 外部声明 |

（3）声明变量在任何代码执行前创建，而非声明变量只有在执行**赋值**操作的时候才会被创建，没有赋值则没有

|  |
| --- |
| console.log(a); // 抛出ReferenceError。  console.log('still going...'); // 永不执行。 |

6、变量的命名规则和规范

（1）规则 - **必须**遵守的，不遵守会报错

- 由**字母**、**数字**、**下划线**、**$符号**组成，不能**以数字开头**

- 不能是关键字和保留字，例如：for、while。

- 区分大小写

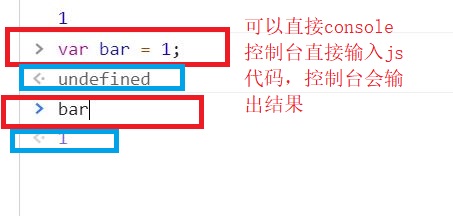
（2）规范 - **建议**遵守的，不遵守不会报错

- 变量名必须有意义

- 遵守驼峰命名法。首字母小写，后面单词的首字母需要大写。例如：userName、userPassword

- 下面哪些变量名不合法

|  |
| --- |
| a  **1**  age18  **18age**  name  $name  \_sex  **&sex**  theworld theWorld 合法的但是建议用驼峰命名法。 |



7、小案例

|  |
| --- |
| 不使用临时变量，交换两个数值变量的值 |
| var a,b;  a = 1;  b = 2;  console.log(a);  console.log(b);  a = a+b;  b = a - b;  a = a - b;  console.log(a);  console.log(b); |

九、数据类型

简单数据类：Number、String、Boolean、Undefined、Null

1、Number类型

数值字面量：数值的固定值的表示法

110 1024 60.5

（1）进制

|  |
| --- |
| 十进制  var num = 9;  进行算数计算时，八进制和十六进制表示的数值**最终都将被转换成十进制数值输出**。  十六进制  var num = 0xA;  数字序列范围：0~9以及A~F  八进制，**前导0零**  var num1 = 07; // 对应十进制的7  var num2 = 019; // 对应十进制的19  var num3 = 08; // 对应十进制的8  数字序列范围：0~7  如果字面值中的数值超出了范围，那么前导零将被忽略，后面的数值将被当作十进制数值解析 |

（2）浮点数

浮点数的精度问题

|  |
| --- |
| 浮点数  var n = 5e-324; // 科学计数法 5乘以10的-324次方  浮点数值的最高精度是 17 位小数，但在进行算术计算时其精确度远远不如整数  var result = 0.1 + 0.2; // 结果不是 0.3，而是：0.30000000000000004  console.log(0.07 \* 100);  0.1+0.3 = 0.4？？？为什么  **不要判断两个浮点数是否相等** |

（3）数值范围

|  |
| --- |
| 最小值：Number.MIN\_VALUE，这个值为： 5e-324  最大值：Number.MAX\_VALUE，这个值为： 1.7976931348623157e+308  无穷大：Infinity  无穷小：-Infinity |

（4）数值判断

|  |
| --- |
| NaN：not a number不是一个数，NaN 与任何值都不相等，包括他本身。  isNaN()，是一个判断 **数** 是不是属于非数字范围，有点绕，数值**12**不是非数字，返回**false**，字符**’abc’**属于非数字，返回**true**。它是自动会将类型进行转换的，并不是单纯看字面意思，如’10’是可以转换成数字，会返回false |

2、String类型

'abc' "abc"，这两种写法都可以，但是推荐使用单引号

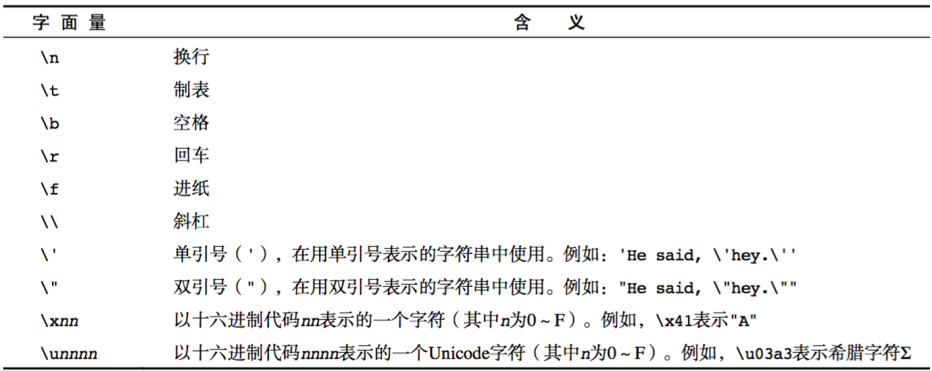
（1）字符串字面量：'程序猿'，'程序媛', "黑马程序猿"

思考：如何打印以下字符串。

我是一个"正直"的人

我很喜欢"黑马'程序猿'"

（2）转义符



\t 制表符在不同的系统中所占的空格可能是不一样的，一般是占4个，有可能是占2个。

（3）符串长度

length属性用来获取字符串的长度

var str = '黑马程序猿 Hello World';

console.log(str.length);

（4）字符串拼接

字符串拼接使用 + 连接

console.log(11 + 11);

console.log('hello' + ' world');

console.log('100' + '100');

console.log('11' + 11);

console.log('male:' + true);

1. 两边只要有一个是字符串，那么 + 就是字符串拼接功能，它会把另外一边的类型自动转换成字符串，然后再进行字符串的拼接。

2. 两边如果都是数字，那么就是算术功能。

3、Boolean类型

Boolean字面量： true和false，区分大小写。

计算机内部存储：true为1，false为0。

4、Undefined和Null

1. undefined表示一个声明了没有赋值的变量，变量只声明的时候值默认是undefined。

2. null表示一个空，变量的值如果想为null，必须手动设置。

3. 因为undefined类型与任意数类型进行“+”运算结果都是NaN。

5、复杂数据类型

Object。

十、获取变量的类型

typeof

|  |
| --- |
| var age = 18;  console.log(typeof age); // 'number'  **typeof返回的值是字符串类型**  var result = typeof age;  console.log(typeof result); |
| 注：  var a = 8;  console(**typeof a**+’’);得到的结果我们希望是字符串类型，输出 string  实际结果是 Number 这个字符串。  原因就是typeof a先运算得到Number然后再变成字符串的Number。  可以加小括号改变运算顺序console(typeof **(**a+’’**)**); |

十一、字面量

在源代码中一个固定值的表示法。根据这个值一眼看过去就知道是数值类型

数值字面量：8, 9, 10,

字符串字面量：'黑马程序员', "大前端"

布尔字面量：true，false

十二、注释

1、单行注释

用来描述下面一个或多行代码的作用

|  |
| --- |
| // 这是一个变量  var name = 'hm'; |

2、多行注释

用来注释多条代码

|  |
| --- |
| /\*  var age = 18;  var name = 'zs';  console.log(name, age);  \*/ |

十三、数据类型转换

如何使用谷歌浏览器，快速的查看数据类型？

**字符串**的颜色是**黑色**的，**数值类型**是**蓝色**的，**布尔类型**也是**蓝色**的，undefined和null是**灰色**的

1、转换成字符串类型

（1）toString()

|  |
| --- |
| var num = 5;  console.log(num.toString());  注：**null 和 undefined 类型没有toString()的方法。**  它不是改变num的性质，它的本质上还是Number类型。 |
| var age = 11;  var ageAsString = age.toString(); // 字符串"11"  var found = true;  var foundAsString = found.toString(); // 字符串"true"  数值、布尔值、对象和字符串值（没错，每个字符串也都有一个 toString() 方法，该方法返回**字符串的一个副本**）都有 toString() 方法 |

（2）String()

|  |
| --- |
| String()函数存在的意义：有些值没有toString()，这个时候可以使用String()。比如：undefined和null |
| 这个函数能够将**任何类型的值**转换为字符串  String() 函数遵循下列转换规则：  1、如果值有 toString() 方法，则调用该方法（没有参数）并返回相应的结果；  2、如果值是 null ，则返回 “null” ；  3、如果值是 undefined ，则返回 “undefined” 。  4、下面再看几个例子：  var value1 = 10;  var value2 = true;  var value3 = null;  var value4;  alert(String(value1)); // "10"  alert(String(value2)); // "true"  alert(String(value3)); // "null"  alert(String(value4)); // "undefined" |
| 因为 null 和 undefined 没有 toString() 方法，所以 String() 函数就返回了这**两个值的字面量**。 |

（3）拼接字符串方式

num + ""，当 + 两边一个操作符是字符串类型，一个操作符是其它类型的时候，会先把其它类型转换成字符串再进行字符串拼接，返回字符串。

2、转换成数值类型

（1）Number()

|  |
| --- |
| 如果是 Boolean 值， true 和 false 将分别被转换为 1 和 0。  如果是数字值，只是简单的传入和返回。  如果是 null 值，返回 0。  如果是 undefined ，返回 NaN 。 |
| 如果是**字符串**，遵循下列规则：  如果字符串中只包含数字（包括前面带正号或负号的情况），则将其转换为十进制数值，即 “1”会变成 1， “123” 会变成 123，而 “011” 会变成 11（注意：前导的零被忽略了）；  如果字符串中包含有效的浮点格式，如 “1.1” ，则将其转换为对应的浮点数值（同样，也会忽略前导零）；  如果字符串中包含有效的十六进制格式，例如开头是 “0xf” ，则将其转换为相同大小的十进制整数值；  如果字符串是空的（不包含任何字符），则将其转换为 **0**；parseInt转换为 **NaN**。  如果字符串中包含除上述格式之外的字符，则将其转换为 NaN。 |
| var num1 = Number("Hello world!"); //NaN  var num2 = Number(""); //0  var num3 = Number("000011"); //11  var num4 = Number(true); //1  var num5=Number(‘1.1.1’)//NaN |

由于 Number() 函数在转换字符串时比较复杂而且不够合理，因此在处理整数的时候更常用的是 parseInt() 函数。

（2）parseInt()，最终输出的结果都是十进制。

|  |
| --- |
| var num1 = parseInt("12.3abc"); // 返回12，如果第一个字符是数字会解析知道遇到非数字结束，小数点不是数字字符。  var num2 = parseInt("abc123"); // 返回NaN，如果第一个字符**不是数字**或者**符号（负号，负数**）就返回NaN。也就是说，用 parseInt() 转换**空字符串**会返回 **NaN** （ 而Number() 对空字符返回 0）  如果是 null 值，返回 NaN。  如果是 undefined ，返回 NaN。  如果是布尔类型，返回 NaN。 |
| 特殊情况：  1、parseInt() 能够识别出各种整数格式，即十进制、八进制（0开头）和十六进制数。也就是说，不带参数前提下。  （1）如果字符串以**“0x”**开头且后跟**数字字符**(如果强制按十进制解析，识别不了“0x”，这里没有指定，会可以自动识别)，就会将其当作一个十六进制整数。  注意十六进制，可以不带前面的“0x”，在一个字符一个字符判断，最后结果显示成十进制的   |  | | --- | | parseInt(“10blue”); 返回10。  parseInt(“0x10blue”); 返回267，自动识别为十六进制。  parseInt(“0x10blue”,10),用十进制去解析，识别不了“x”，返回 0。  parseInt(“10blue”,16)，返回267，可以不带前导“0x”。  parseInt(“0x10blue”,16)返回 267。 |   （2）如果字符串以 “0” 开头且后跟数字字符，则会将其当作一个八进制数来解析。  （3）parseInt()函数在转换字符串时，更多的是看其是否符合数值模式。它会**忽略**字符串前面的**空格**，直至找到第一个**非空格字符**。  （4）为了更好地理解 parseInt() 函数的转换规则，下面给出一些例子：  var num1 = parseInt("1234blue"); // 1234  var num2 = parseInt(""); // NaN  var num3 = parseInt("0xA"); // 10（十六进制数）  var num4 = parseInt(22.5); // 22  var num5 = parseInt("070"); // 56（八进制数）  var num6 = parseInt("70"); // 70（十进制数）  var num7 = parseInt("0xf"); // 15（十六进制数）  parseInt(0x11,16)--->输出是23，原因是0x11会转换成17，十六进制17表示十进制23  parseInt(“0x11”,16)--->输出是17  2、在使用 parseInt() 解析像八进制字面量的字符串时，ECMAScript 3 和 5 存在分歧。例如：  //ECMAScript 3 认为是 56（八进制），ECMAScript 5 认为是 70（十进制）  var num = parseInt("070");  在 ECMAScript 3 JavaScript 引擎中， “070” 被当成八进制字面量，因此转换后的值是十进制的 56。而在 ECMAScript 5 JavaScript 引擎中， parseInt() 已经**不具有解析八进制值**的能力，因此前导的零会被认为无效，从而将这个值当成 “70” ，结果就得到十进制的 70。在 ECMAScript 5 中，即使是在非严格模式下也会如此。  3、为了消除在使用 parseInt() 函数时可能导致的上述困惑，可以为这个函数提供第二个参数：转换时使用的基数（即多少进制）。如果知道要解析的值是十六进制格式的字符串，那么指定基数 16 作为第二个参数，可以保证得到正确的结果，例如：  var num = parseInt("0xAF", 16); //175  实际上，如果指定了 16 作为第二个参数，字符串可以不带前面的 “0x” ，如下所示：  var num1 = parseInt("AF", 16); //175  var num2 = parseInt("AF"); //NaN  这个例子中的第一个转换成功了，而第二个则失败了。  差别在于第一个转换传入了基数，明确告诉 parseInt() 要解析一个十六进制格式的字符串；而第二个转换发现第一个字符不是数字字符，因此就自动终止了。  4、指定基数会影响到转换的输出结果。例如：  var num1 = parseInt("10", 2); //2 （按二进制解析）  var num2 = parseInt("10", 8); //8 （按八进制解析）  var num3 = parseInt("10", 10); //10 （按十进制解析）  var num4 = parseInt("10", 16); //16 （按十六进制解析）  不指定基数意味着让 parseInt() 决定如何解析输入的字符串，因此为了避免错误的解析，我们建议无论在什么情况下都明确指定基数。  多数情况下，我们要解析的都是十进制数值，因此始终将 10 作为第二个参数是非常必要的。 |

（3）parseFloat()

|  |
| --- |
| parseFloat()和parseInt非常相似，不同之处在与  parseFloat会解析第一个小点**.** 遇到第二个小点**.**或者非数字结束。  它始终都会忽略前导的零。  只解析十进制值  如果解析的内容里只有整数，解析成整数。  var num1 = parseFloat("1234blue"); //1234 （整数）  var num2 = parseFloat("0xA"); //0  var num3 = parseFloat("22.5"); //22.5  var num4 = parseFloat("22.34.5"); //22.34  var num5 = parseFloat("0908.5"); //908.5  var num6 = parseFloat("3.125e7"); //31250000 |

（4）+，- 取正、取负运算

|  |
| --- |
| 1、如果是布尔类型  var test\_bool = true;  console.log(-test\_bool);输出1或者-1。   1. 纯整数可以(可以仅仅包含一个小数点，忽略前导0) 2. 非纯整数输出 NaN   console.log(str - 0);  如果一边是数值类型，一边是字符串，会先把字符串转换成数值类型再进行数值的减法运算，如果字符串转换数值类型失败，此时返回NaN。  适用范围：  **跟+ - 运算一样的。**  console.log(str + 0);  如果有一边是数值类型，一边是字符串，会先把数值类型转换成字符串，再进行字符串的拼接。 |

3、转换成布尔类型

（1）Boolean()

0(**不能写字符零**)、''(**空字符串，有空格就不算是空字符串**)、null、undefined、NaN 会转换成false。其它都会转换成true。

十四、操作符

1、运算符 operator

表达式由操作数和操作符组成，会有一个结果，如5 + 6 = 11

（1）算术运算符

|  |
| --- |
| +、-、\*、/、% 属于二元操作符 |
| 5 / 0 = -Infinity  5 % 0 = NaN |

（2）一元运算符

一元运算符：只有一个操作数的运算符。

5 + 6两个操作数的运算符 二元运算符。

++自身加1。

-- 自身减1。

增量运算符将其操作数递增（加 1）并返回一个值。

前置++，**先执行a自身加1，然后再返回表达式结果。**

|  |
| --- |
| var num1 = 5;  ++ num1;  var num2 = 6;  console.log(num1 + ++num2);输出13 |

后置++，**先返回表达式的结果a，再执行a自身加1。**

|  |
| --- |
| var num1 = 5;  num1 ++;  var num2 = 6  console.log(num1 + num2 ++);输出12 |

练习

|  |
| --- |
| var a = 1; var b = ++a + ++a; console.log(b); 输出 5  var a = 1; var b = a++ + ++a; console.log(b); 输出 4  var a = 1; var b = a++ + a++; console.log(b); 输出 3  var a = 1; var b = ++a + a++; console.log(b); 输出 4  a的最终是3 |

（3）逻辑运算符(布尔运算符，二元逻辑运算符)，通常会与“关系运算符”一起使用，这种情况就只有布尔类型。

|  |
| --- |
| && 与。两个操作数同时为true，结果为true，否则都是false  || 或。两个操作数有一个为true，结果为true，否则为false  ! 非。取反，**具有布尔类型隐式转换功能**，详情见布尔类型隐式转换。 |
| ||或运算符，是一个布尔类型的运算符。  两边是布尔类型的话，  如果有一边是true，返回true；  如果两个都是false，返回false。  两边也可以是其他的类型，会先转换成布尔类型。  如果第一个“运算数”转换成“布尔类型”，是true，则直接返回这个运算数。  如果第一个“运算数”转换成“布尔类型”，是false，则直接返回第二个运算数。  当运用在给变量赋值时  alert(1&&2); //输出2  alert(1||0); //输出1  知识点：  ****&&运算符规则：a&&b，如果 a 为true，直接返回 b；如果 a 为false 那么直接返回 a。****  ****|| 运算符规则：a || b，如果 a 为true，直接返回 a；如果 a 为false 那么直接返回 b。****  ****&& 优先级高于 ||****  alert((1 && 3 || 0) && 4); //输出4  alert(1 && 3 || 0 && 4); //输出 3  alert(0 && 3 || 1 && 4); //输出 4  alert(1 && 3 || 1 && 4); //输出 3 |

（4）关系运算符(比较运算符)

|  |
| --- |
| <、>、>=、<=、==、!=、===、!==，**返回一个布尔类型**的结果。  var a = 10;  var b = '10';  console.log(a == b);  /\*结果是输出true，因为b会先转换为Number类型，在继续比较是否相等\*/  == 判断的是变量的值是否相等。  === 先判断类型，在判断变量的值是否相等，类型不一样，则直接返回false。  能直接比较两字符串吗？？？ |

（5）运算符的优先级

|  |
| --- |
| 优先级从高到底  1. () 优先级最高  2. 一元运算符：++ -- !  3. 算术运算符：**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： = |
| 练习  // 练习1：  4 >= 6 || '人' != '阿凡达' && !(12 \* 2 == 144) && true  分析如下：  4 >= 6 || '人' != '阿凡达' && !(false) && true**先化简括号的内容**  4 >= 6 || '人' != '阿凡达' && true&& true  (4 >= 6) || ('人' != '阿凡达' && true&& true) 或运算优先级最低，可以先划分为两部分  false || true  // 练习2：  var num = 10;  5 == num / 2 && (2 + 2 \* num).toString() === '22'  分析如下：  5 == num / 2 && 22.toString() === '22'  (5 == num / 2 )&&( 22.toString() === '22')  true && true |

（6）三元运算符

|  |
| --- |
| 表达式1 **?** 表达式2 **:** 表达式3  是对if……else语句的一种简化写法。  表达式1是布尔类型的表达式，最终会返回一个布尔类型的值。  当值为true时，返回表达式2的值。  当值为false时，返回表达式3的值。  var num1 = 11;  var num2 = 22;  var max = num1 > num2 ? num1 : num2;  console.log(max); |

（7）逗号运算符

逗号操作符：对它的每个操作数求值（从左到右），并**返回最后一个操作数的值**。

|  |
| --- |
| var x = 1;  x = (x++, x);  console.log(x);  // expected output: 2  x = (2, 3);  console.log(x);  // expected output: 3 |

十五、表达式和语句

1、表达式

一个表达式可以产生一个值，有可能是运算、函数调用、有可能是字面量。表达式可以放在任何需要值的地方。

|  |
| --- |
| 理解使用(5)和5的区别  console.log( (5).toString() ); // 5  console.log( 5.toString() ); // Unexpected token ILLEGAL  (5)是一个表达式，它会返回一个Number对象，而5是一个常量，不会返回一个Number对象。 |

2、语句

语句可以理解为一个行为，循环语句和判断语句就是典型的语句。一个程序有很多个语句组成，一般情况下;分割一个一个的语句

十六、流程控制

1、程序的三种基本结构

（1）顺序结构

从上到下执行的代码就是顺序结构，程序默认就是由上到下顺序执行的

（2）分支结构，根据不同的情况，执行对应代码

if语句

语法结构

|  |
| --- |
| if (/\* 条件表达式 \*/) {  // 执行语句  }  if (/\* 条件表达式 \*/){  // 成立执行语句  } else {  // 否则执行语句  }  第三种情况，只要有一个 if 的条件符合就只执行那一个，剩下的不会执行。  if (/\* 条件1 \*/){  // 成立执行语句  } else if (/\* 条件2 \*/){  // 成立执行语句  } else if (/\* 条件3 \*/){  // 成立执行语句  } else {  // 最后默认执行语句  }  推荐使用 === 全相等 |

案例

|  |
| --- |
| 求两个数的最大数  判断一个数是偶数还是奇数  分数转换，把百分制转换成ABCDE <60 E 60-70 D 70-80 C 80-90 B 90 - 100 A  function maxNum(a,b)  {  if(a > b)  {  return a;  }  else  {  return b;  }  }  console.log(maxNum('4',5));  function evenOrOdd(num)  {  if((typeof num) != 'number')  {  console.log(typeof num);  console.log('它是非数字');  return;  }  if(num % 2 === 0)  {  console.log('是偶数');  }  else  {  console.log('是奇数');  }  }  evenOrOdd('1');  var score = 59;  if(score >= 90 && score <= 100) {  console.log('A');  } else if(score >= 80 && score < 90) {  console.log('B');  } else if(score >= 70 && score < 80) {  console.log('C');  } else if(score >= 60 && score < 70) {  console.log('D');  } else {  console.log('E');  }  var score = 59;  if(score >= 90 && score <= 100) {  console.log('A');  } else if(score >= 80) { 上一个条件是>=90，在else就隐含着 <90这个条件了，可以省略  console.log('B');  } else if(score >= 70) {  console.log('C');  } else if(score >= 60) {  console.log('D');  } else {  console.log('E');  }  还可以改成先判断<60、<70...反过来判断 |

|  |
| --- |
| 判断一个年份是闰年还是平年  闰年：能被4整除，但不能被100整除的年份 或者 能被400整除的年份  var year=2013;  if( (year % 4 ===0 && year % 100 !== 0) || (year % 400 === 0) ) {  console.log(year + '是闰年');  }  else {  console.log(year + '是平年');  }  判断一个人的年龄是否满18岁(是否成年) |

switch语句

|  |
| --- |
| switch (**expression**) {  case **常量1**:这里不能写表达式，或者范围。  语句;  **break;**  case 常量2:  语句;  break;  …  case 常量n:  语句;  break;  default:  语句;  break;  } |
| 1、break可以省略，如果省略，代码会继续执行下一个case  2、switch 语句在比较值时使用的是**全等操作符 === ，**因此不会发生类型转换（例如，字符串'10' **不等于**数值 10）  3、default 放到 case 之间吗？  可以啊！JavaScript 会在它找不到匹配项时跳回到那个 default ：**记得在default后面加上break语句，否则会执行其他case，这个比较容易忘记！！！**  4、不能做到范围性的判断，只能做全相等性的判断。if能做到范围性判断。 |

例子：

|  |
| --- |
| var score = 33;  score = parseInt(score / 10);/\*js中的除法跟正常计算器得到的结果是一样的，跟C不一样\*/  switch (score) {  case 10:  case 9:  console.log('A');  break;  case 8:  console.log('B');  break;  case 7:  console.log('C');  break;  case 6:  console.log('D');  break;  default:  console.log('E');  break;  } |

布尔类型的隐式转换

**流程控制语句 if**会把后面的值隐式转换成布尔类型。

|  |
| --- |
| 转换为true：非空字符串、非0数字、true、任何对象。  转换成false：空字符串、0、false、null、undefined。 |

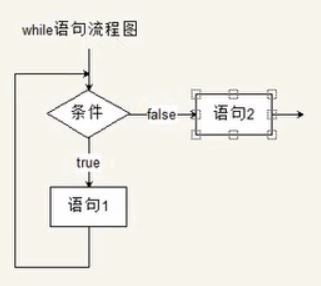
|  |
| --- |
| var message;  // 会自动把message转换成false，因为message属于undefined类型。  **if (message)** {  // todo...  } |

|  |
| --- |
| var booleanSwitch = '11';  var result = **!!**booleanSwitch; 隐式转换成布尔类型  console.log(result);//输出 true  console.log(typeof result);//输出 boolean  console.log(typeof booleanSwitch);//输出 string |

（3）循环结构，循环结构：重复做一件事情

在javascript中，循环语句有三种，**while**、**do..while**、**for**循环。

while语句



基本语法：

|  |
| --- |
| // 当循环条件为true时，执行循环体，  // 当循环条件为false时，结束循环。  while (循环条件) {  //循环体  }  写死循环会导致，网页崩溃。 |

代码示例：

|  |
| --- |
| // 计算1-100之间所有数的和  var i = 1;  var sum = 0;// 初始化变量，这个是1-100  // 判断条件  while (i <= 100) {  sum += i;// 循环体  i++;// 自增  }  console.log(sum);  初始值不一样，一些循环的代码有点区别 这个是0-99  var num=0;  var sum=0;  while(num < 100) {  num++;  sum +=num;  }  console.log(sum); |

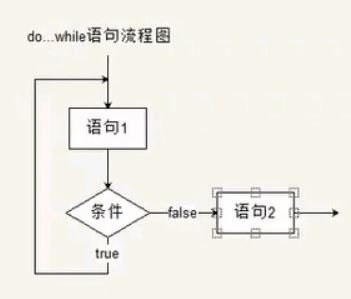
案例

|  |
| --- |
| 打印100以内所有偶数的和  var num=0;  var sum=0;  while(num < 100) {  num++;  if(num % 2 === 0)  {  sum += num;  }  }  console.log(sum); |

do...while语句

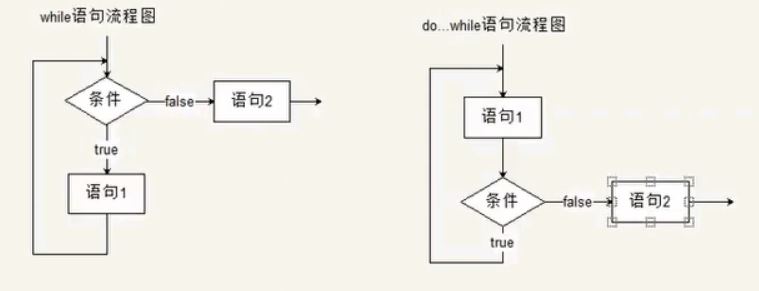
do..while循环和while循环非常像，二者经常可以相互替代，但是do..while的特点是不管条件成不成立，都会执行一次。

基础语法：



|  |
| --- |
| do {  // 循环体;  } while (循环条件); |

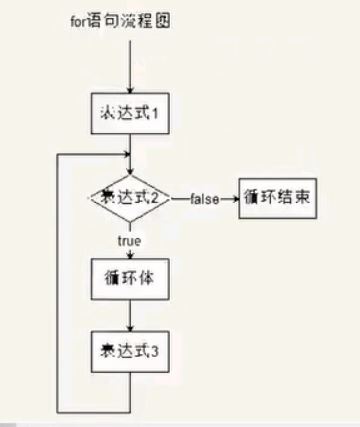
|  |
| --- |
| // 初始化变量  var i = 1;  var sum = 0;  do {  sum += i;//循环体  i++;//自增  } while (i <= 100);//循环条件 |
| 使用do-while循环：输出询问“我爱你，嫁给我吧？”，选择“你喜欢我吗？(y/n):"，如果输入为y则打印”我们形影不离“，若输入为n,则继续询问  prompt函数的用法  第一个参数，是用于提示的字符串。  第二个参数，在输入框中提示用户输入。  返回值：就是我们刚才在输入框中输入的内容。  var result;  var str='请输入yes/no';  do {  result = prompt('你到底是谁',str);  } while (result !== ‘yes’); |



for语句

while和do...while一般用来解决**无法确认**次数的循环。for循环一般在循环**次数确定**的时候比较方便

for循环语法：



|  |
| --- |
| // for循环的表达式之间用的是;号分隔的，千万不要写成,  for (初始化表达式1; 判断表达式2; 自增表达式3) {  // 循环体4  }  for( ; ; )里边的表达式是可以省略的，可以在别的地方写，但是中的条件最好省略，会导致死循环。  执行顺序：**1243**---->**243**-----> **243**(直到循环条件变成false)  1. 初始化表达式  2. 判断表达式  3. 自增表达式  4. 循环体 |

打印正方形、打印直角三角形、打印9\*9乘法表

|  |
| --- |
| 2  var start = '';  for (var i = 1; i <= 10; i++) {  for (var j=1; j <= 10; j++) {  start += '\* ';  }  start += '\n';  }  console.log(start);  /\*在字符串中的\n，跟单独输出的\n在控制台上怎么有点不一样。  解析： 一句console.log就会占一个行\*/  var start = '';  for (var i = 1; i <= 10; i++) {  for (var j=i; j <= 10; j++) {  start += '\* ';  }  start += '\n';  }  console.log(start);  /\*输出上三角 for (var j=1; j <= 10-i; j++)\*/  /\*输出乘法表\*/  var start ='';  var space = '';  for (var i = 1; i < 10; i++) {  for (var j = i; j < 10; j++) {  /\*if (i\*j < 10) {  space = ' ';  } else {  space = ' ';  }  start += (i + ' \* ' + j + ' = ' + i\*j + space);\*/  start += (i + ' \* ' + j + ' = ' + i\*j + '\t');/\*直接改成制表符就ok了\*/  }  start +='\n';  }  console.log(start); |
| （1）求1-100之间所有数的乘积  （2）计算1-100之间不能被7整除的数的和  （3）本金10000元存入银行，年利率是千分之三，每过1年，将本金和利息相加作为新的本金。计算5年后，获得的本金是多少？  （4）有个人想知道，一年之内一对兔子能繁殖多少对？于是就筑了一道围墙把一对兔子关在里面。已知一对兔子每个月可以生一对小兔子，而一对兔子从出生后第3个月起每月生一对小兔子。假如一年内没有发生死亡现象，那么，一对兔子一年内（12个月）能繁殖成多少对？（兔子的规律为数列，1，1，2，3，5，8，13，21）   |  | | --- | | var sum=1;  for (var i=1; i<=100; i++) {  sum \*=i;  }  console.log(sum);  var sum=1;  for (var i=1; i<=100; i++) {  if (i % 7 !== 0) {  sum += i;  }  }  console.log(sum);  本金 = 本金 + 本金 x 利率（本金 += 本金 x 利率）  var originMoney = 10000;  var sum =0;  var year =5;  sum = originMoney;  for (var i = 1; i <= 5; i++) {  sum \*= (1 + 0.003);  }  console.log(sum);  2  var n1 = 1;  var n2 = 1;  var n3;  var month = 12;  /\*因为最少是需要两个已知的量，所以从第三个月开始\*/  for (var i = 3; i <= month; i++) {  n3 = n1 + n2;  n1 = n2;  n2 = n3;  }  console.log(n3); | |

continue和break

（1）break:立即跳出整个循环，即循环结束，开始执行循环后面的内容。

（2）continue:立即结束当前循环，开始继续下一次循环（跳到i++的地方）

案例：

求200-300之间第一个能被7整数的数（break）

|  |
| --- |
| var sum = 0;  for (var i = 200; i <= 300; i++) {  if (i % 2 == 0) {  **continue;**  }  sum += i;  }  console.log(sum); |

十七、调试

1、过去调试JavaScript的方式

（1）alert()，这个遇到循环，调试非常麻烦，需要点击n次，需要关闭整个网页才行。

（2）console.log()

2、断点调试

断点调试是指自己在程序的某一行设置一个断点，调试时，程序运行到这一行就会停住，然后你可以一步一步往下调试，调试过程中可以看各个变量当前的值，出错的话，调试到出错的代码行即显示错误，停下。

设置断点后，可以把鼠标移到代码的变量上，可以看到变量的值。可以根据蓝色的背景（代表着即将要执行）看到程序执行顺序。

（1）调试步骤

|  |
| --- |
| 浏览器中按F12 --> sources --> 找到需要调试的文件 --> 在程序的某一行设置断点 --> **程序只会执行到断点的前面的一个语句** |

（2）调试中的相关操作

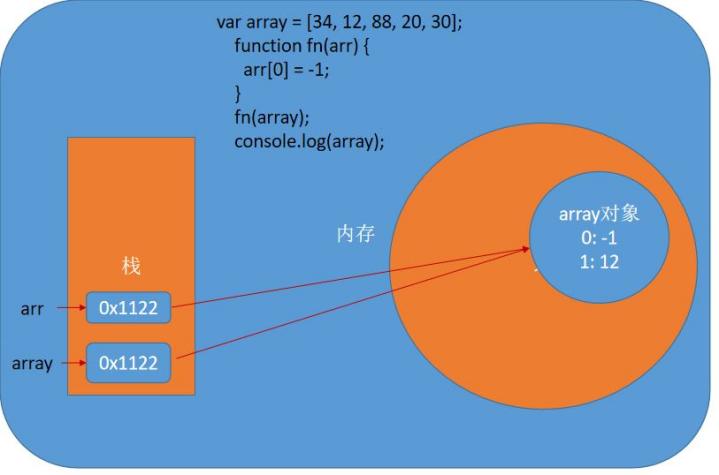
|  |
| --- |
| **Watch:** 监视，通过watch可以监视变量的值的变化，非常的常用。需要自己添加变量的名字  **F10:** 程序单步执行，让程序一行一行的执行，这个时候，观察watch中变量的值的变化。  F8：跳到下一个断点处，如果后面没有断点了，则程序执行结束。 |

tips: 监视变量，不要监视表达式，因为监视了表达式，那么这个表达式也会执行。

十八、数组

1、数组的定义

属于**复杂数据类型**，存储**在堆中**，在**栈中**存放的是它的指针。通过数组字面量创建数组（其实这是一个简写的方式，后续会有new来创建）



当fn(array)执行的时候，fn函数会在“栈”中创建arr空间，然后把array的值复制给arr空间中，这样他们就指向“堆”中的同一个对象了。

数组是一个有序的列表，可以在数组中存放任意的数据，并且数组的长度可以动态的调整。数组是一种数据类型，它可以存储多个数据。

数组中存储的数据类型可以不一致。但是不推荐这么做，它是具有一定的类型意义，例如存储学生的名字，一般用字符串，学号就使用数值等。

通过数组字面量创建数组（其实这是一个简写的方式，后续会有new来创建），数组的字面量 **[ ]** ，一看到这个中括号就知道是数组。

|  |
| --- |
| // 创建一个空数组  var arr1 = **[]**;  // 创建一个包含3个数值的数组，多个数组项以逗号隔开  var arr2 = [1, 3, 4];  // 创建一个包含2个字符串的数组  var arr3 = ['a', 'c'];  // 可以通过数组的length属性获取数组的长度，数组的长度就是数组元素的个数  console.log(arr3.length);  // 可以设置length属性改变数组中元素的个数，数组的长度可以动态的改变。  arr3.length = 0;  设置为0 之后，所有的元素都会变为 undefined ，相当于是被删除了，清空了数组。  2 |

2、获取数组元素

数组的取值，数组中的第一项的索引是**从 0 开始**的

|  |
| --- |
| // 格式：数组名[下标] 下标又称索引  // 功能：获取数组对应下标的那个值，如果下标不存在，则返回undefined。  var arr = ['red',, 'green', 'blue'];  arr[0]; // red  arr[2]; // blue  arr[**3**]; // 这个数组的最大下标为2,因此返回undefined |

3、数组中新增元素

|  |
| --- |
| // 格式：数组名[下标/索引] = 值;  // 如果下标有对应的值，会把原来的值覆盖，如果下标不存在，会给数组新增一个元素。  var arr = ["red", "green", "blue"];  // 把red替换成了yellow  arr[0] = "yellow";  // 给数组新增加了一个pink的值  arr[3] = "pink";  arr[6] = 'xxx';//接着连续给值，那些跳过的值是 undefined，为什么浏览器显示 empty？  2 |

4、数组的遍历

遍历：遍及所有，对数组的每一个元素都访问一次就叫遍历。有一些数组自带的方法是有兼容性问题。使用的时候需要注意

（1）直接使用for去遍历数组

数组遍历的基本语法：

|  |
| --- |
| for(var i = 0; i < **arr.length**; i++) {  // 数组遍历的固定结构  } |

（2）every()

every() 方法测试一个数组内的所有元素是否都能通过某个指定函数的测试。它**返回一个布尔值**。

注意：若收到一个空数组，此方法在一切情况下都会返回 true。

|  |
| --- |
| function isBigEnough(element, index, array) {  return element >= 10;  }  console.log([12, 5, 8, 130, 44].**every**(isBigEnough)); // false  console.log([12, 54, 18, 130, 44].every(isBigEnough)); // true  console.log([].every(isBigEnough)); // true |

（3）find()

find() 方法返回数组中满足提供的测试函数的**第一个元素的值**。否则返回 undefined。

|  |
| --- |
| // 它大概是根据这个函数返回的值，看是true还是false，再返回元素  function big10 (element) {  return element > 10;  }  const array1 = [5, 12, 8, 130, 44];  // const found = array1.find(element => element > 10);  const found = array1.**find**(big10);  console.log(found); |

（4）findIndex() 方法，它返回数组中找到的元素的索引，而不是其值。

（5）forEach()

forEach() 方法对数组的每个元素执行一次给定的函数。forEach() 被调用时，不会改变原数组

**注意：** 除了抛出异常以外，没有办法中止或跳出 forEach() 循环。如果你需要中止或跳出循环，forEach() 方法不是应当使用的工具。

若你需要提前终止循环，你可以使用：

* 一个简单的 [for](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Statements/for) 循环
* [for...of](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Statements/for...of) / [for...in](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Statements/for...in) 循环
* [Array.prototype.every()](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Array/every)
* [Array.prototype.some()](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Array/some)
* [Array.prototype.find()](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Array/find)
* [Array.prototype.findIndex()](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Array/findIndex)

这些数组方法则可以对数组元素判断，以便确定是否需要继续遍历：

* [every()](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Array/every)
* [some()](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Array/some)
* [find()](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Array/find)
* [findIndex()](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Array/findIndex)

|  |
| --- |
| function add (element) {  element++;  console.log(element);  }  const array1 = [5, 12, 8, 130, 44];  array1.**forEach**(add);  console.log(array1); |

（6）map()

map() 方法创建一个新数组，其结果是该数组中的每个元素是调用一次提供的函数后的返回值。

|  |
| --- |
| const array1 = [1, 4, 9, 16];  // pass a function to map  const map1 = array1.**map**(x => x \* 2);  console.log(array1);  console.log(map1);  // expected output: Array [2, 8, 18, 32] |

（7）some()

some() 方法测试数组中是不是至少有1个元素通过了被提供的函数测试。它返回的是一个Boolean类型的值。

**注意：**如果用一个空数组进行测试，在任何情况下它返回的都是false。

some() 为数组中的每一个元素执行一次 callback 函数，直到找到一个使得 callback 返回一个“真值”（即可转换为布尔值 true 的值）。

|  |
| --- |
| function test(element, index) {  if (element % 2 === 0) {  console.log(index);  return true;  }  }  const array = [1, 2, 3, 4, 5];  console.log(array.some(test));  // expected output: true |

5、数组.length的用法

当需要**动态的往新数组**里**添加元素**的时候可以这样使用

要求将数组中的0项去掉，将不为0的值存入一个新的数组，生成新的数组。

|  |
| --- |
| var num1 = [11,22,33,56,0,87,0];  var newArr = [];/\*此时的length = 0\*/  for (var i = 0; i < num1.length; i++ ) {  if (num1[i] !== 0) {  /\*此时的length = 0，即是newArr[0]\*/  newArr[newArr.length] = num1[i];  /\*执行完这条语句，length = 1,相当是一个自增量，不用另外定义一个变量\*/  }  }  console.log(num1);  console.log(newArr);  console.log(newArr.length); |

6、将字符串数组用|或其他符号分割，意思将数组中的字符拼接成一个字符串，中间插入符号作为分割。

|  |
| --- |
| /\*如果你在 hi 后边加分隔符，那么处理最后一个比较麻烦，直接在第二个元素前加就不用去处理了\*/  var names = ['hi','hello','world','a','b'];  var str = names[0];  var seperator = '|';  for (var i = 1; i < names.length; i++) {  str += seperator + names[i];  }  console.log(str); |

（1）翻转数组

|  |
| --- |
| var temp;  var nums = [11,22,33,0,44,55,66 ];  console.log(nums);  for (var i = 0; i < nums.length; i++) {//这里不用循环那么多次的，大概一半就ok，还没有优化，其实下边这句break就处理了这个事了  if (i >= nums.length-1-i) {  break;  }  temp = nums[i];  nums[i] = nums[nums.length-1-i];  nums[nums.length-1-i] = temp;  }  console.log(nums);  //反向输出数组到新的数组  for (var i = 0; i < nums.length; i++) {  console.log(nums[nums.length-1-i]);  newArry[newAr.length ]  } |

（2）\*冒泡排序

|  |
| --- |
| 2 |
| /\*冒泡排序法\*/  var temp = 0;  var numArray = [11,23,44,6,88,99,90];  console.log(numArray);  for (var i = 0; i < numArray.length-1; i++) {  **var isSort = true;**  for (var j = 0; j < numArray.length-1-i; j++) {  if (numArray[j] >= numArray[j+1]) {  **isSort = false;**  temp = numArray[j+1];  numArray[j+1] = numArray[j];  numArray[j] = temp;  }  }  **if (isSort) {**  **break;**  **}**  }  console.log(numArray);  如果不添加红色部分，那么有可能在数组在有序后，可能会继续进行不必要的比较，因为已经没有可以交换的数据了。 |

（3）求一组数中的最大值和最小值，以及所在位置

|  |
| --- |
| var nums = [11,22,33,0,44,55,66 ];  var max = nums[0];  var min = nums[0];  var minIndex **= 1**;  var maxIndex **= 1**;  /\*这里 i 的值可以从 1 开始，因为 0 已经没必要比了\*/  for (var i = **1**; i < nums.length; i++) {  if (max < nums[i]) {  max = nums[i];  maxIndex = i+1;  }  if (min > nums[i]) {  min = nums[i];  minIndex = i+1;  }  }  console.log('最大值：' + max + ' 位置：' + maxIndex);  console.log('最小值：' + min + ' 位置：' + minIndex);  /\*以上这段求最大值、最小值代码中有bug，当第一个数是最大值，或者最值时，它的位置会显示为undefined。  因为没有执行到位置的赋值语句。  解决方式，就是给maxIndex、minIndex初始化为 1 \*/ |

7、二维数组

|  |
| --- |
| // 二维数组，每一项元素还是一个数组，数组里可以是基本数据类型、复杂数据类型  var arr = [];  arr[0] = [1,2,3];  arr[1] = [4,5,6];  arr[2] = [7,8,9];  arr['hello'] = ['hi']; // 可以是字符串，数组本身是对象，给它添加了属性  for (var i = 0; i < arr.length; i++) {  for (var j = 0; j < arr[i].length; j++) {  console.log(arr[i][j]);  }  }  console.dir(arr); |

8、array.forEach()

forEach是 ES5 中**操作数组**的一种方法，**主要功能是遍历数组**，其实说穿了，就是 for 循环的加强版，该语句**需要一个回调函数，作为参数**。

回调函数的形参，依次为：

value：遍历的数组内容；index：对应的数组索引，array：数组本身。

|  |
| --- |
| var arr = ['a', 'b', '123'];  arr.forEach (function (value, index, array) {  console.log(value);  console.log(index);  console.log(array);  });  1 |

十九、函数

1、什么是函数

把一段相对独立的具有特定功能的代码块封装起来，形成一个独立实体，就是函数，起个名字（函数名），在后续开发中可以反复调用。

函数的作用就是封装一段代码，将来可以重复使用，函数自己是**不会自动运行**的。

在html中引入外部的js：<script type="text/javascript" src="index.js"></script> ，当html从上到下执行到这里的引入时候，会把js文件中的代码，相当于是替换到 <script></script>中。

1. 函数声明

（1）函数声明的最重要的一个特征是**函数声明提升**，意思是在执行代码之前会先读取函数声明，所以函数声明可以放在调用函数语句之后。

|  |
| --- |
| **function** 函数名() {  // 函数体  }  **函数的命名最好是动词** |

（2）函数表达式

|  |
| --- |
| 函数表达式的语法形式如下：  **var fn = function () {**  **// 函数体**  **}**  这种形式看起来像常规的变量赋值，先创建一个**匿名函数**，然后赋值给变量functionName。我们知道在JS中如果要想使用一个变量，必须先给这个变量赋值，所以函数表达式也不例外，在调用之前，必须先给它赋值。否则会报错。  sayHi(); //在这里调用会报错  var sayHi = function(){  console.log("Hi");  }  sayHi(); //Hi |

函数声明和函数表达式都是用来定义一个函数的，区别就是函数声明存在函数声明，可以在声明前调用。而函数表达式则必须在赋值语句之后再调用。

3、函数的调用

调用函数的语法

|  |
| --- |
| **函数名(参数);** |

4、函数的参数

（1）形参和实参

1. 形式参数：在声明一个函数的时候，为了函数的功能更加灵活，有些值是固定不了的，对于这些固定不了的值。我们可以给函数设置参数。这个参数没有具体的值，仅仅起到一个占位置的作用，我们通常称之为形式参数，也叫形参。**形参：形式上的参数**

2. 实际参数：如果函数在声明时，设置了形参，那么在函数调用的时候就需要传入对应的参数，我们把传入的参数叫做实际参数，也叫实参。

|  |
| --- |
| // 函数内部是一个封闭的环境，可以通过参数的方式，把外部的值传递给函数内部  // 带参数的函数声明，形参之间使用**逗号**隔开。  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**);  形参1 = 实参1  形参2 = 实参2 |

|  |
| --- |
| var x = 5, y = 6;  fn(x,y);  function fn(a, b) {  console.log(a + b);  }  // x,y实参，有具体的值。函数执行的时候会把x,y复制一份给函数内部的a和b，函数内部的值是复制的新值，**无法修改外部的x,y。不能对他们进行交换。**  **数组是例外，数组作为形参，里边的值可以被修改。下边冒泡排序可以验证。** |

（2）\*求3个数中的最大值

|  |
| --- |
| /\*求N个数的最大值？N是一个数组\*/  function getMaxOfN(array) {  var max = array[0];  for (var i = 1; i < array.length; i++) {  if (max < array[i]) {  max = array[i];  }  }  console.log(max);  }  var num = [1,2,3,44];  getMaxOfN(num); |

（3）\*判断一个数是否是素数(又叫质数，只能被1和自身整数的数)

|  |
| --- |
| //求一个数是不是素数  function isPrimeNumber(num) {  //先进行假设，它是一个素数  var isPrime = true;  for (var i = 2; i < num; i++) { //记得这里可以进行优化的，不用算全部的数？？？  if (num % i === 0) {  isPrime = false;  }  }  if (isPrime) {  console.log(num + '是素数');  } else {  console.log(num + '不是素数');  }  }  isPrimeNumber(24); |

5、函数的返回值

当函数执行完的时候。我们期望函数给我一些反馈（比如计算的结果返回进行后续的运算），这个时候可以让函数返回一些东西。也就是返回值。函数通过**return**返回一个返回值。

|  |
| --- |
| //声明一个带返回值的函数  function 函数名(形参1, 形参2, 形参3...) {  //函数体  **return 返回值**;  }  //可以通过变量来接收这个返回值  var 变量 = 函数名(实参1, 实参2, 实参3...); |

函数的调用结果就是返回值，因此我们可以直接对函数调用结果进行操作。

|  |
| --- |
| （1）函数没有显示的使用 return语句 ，那么函数有默认的返回值：undefined。  （2）函数使用 return语句，那么跟再return后面的值，就成了函数的返回值。  （3）函数使用 return语句，但是return后面没有任何值，那么函数的返回值也是：undefined。  函数使用return语句后，这个函数会在执行完 return 语句之后停止并立即退出，也就是说return后面的所有其他代码都不会再执行。 |

（1）\*求一个数 n 的阶乘

|  |
| --- |
| function getMultiple(n) {  var result = 1;  for (var i = 1; i <= n; i++) {  result = result \* i;  }  return result;  }  var n = 5;  console.log('5! = '+ getMultiple(n)); |

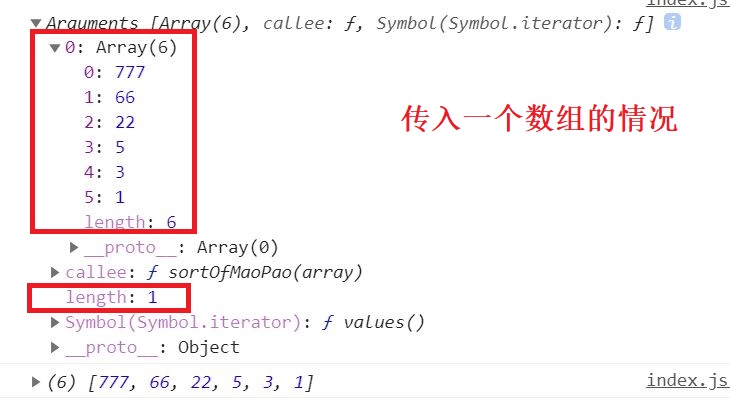
（2）\*求1!+2!+3!+...+n!

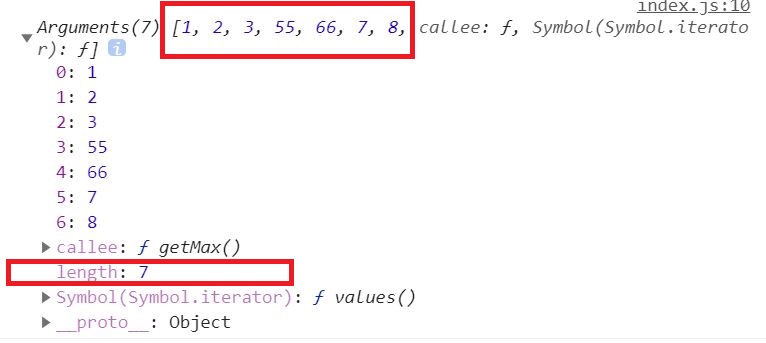
|  |
| --- |
| function getMultipleN(n) {  var result = 1;  var sum = 0;  for (var i = 1; i <= n; i++) {  result = result \* i;  sum = sum + result;  }  return sum;  }  var n = getMultipleN(5);  console.log(n); |

6、arguments的使用

JavaScript中，arguments对象是比较特别的一个对象，实际上是当前函数的一个内置属性。也就是说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的实参。arguments是一个伪数组，因此及可以进行遍历。跟数组的用法比较类似。

我们可以通过 argument 获取函数调用时候的实参。可以方便获取到动态的形参。





（1）不确定参数的个数，计算最大值

|  |
| --- |
| function getMax() {  console.log(arguments);  var max = arguments[0];  for (var i = 1; i < arguments.length; i++) {  if (max < arguments[i]) {  max = arguments[i];  }  }  return max;  }  var max = getMax(1,2,3,55,66,7,8);  console.log(max); |

（2）\*求斐波那契数列Fibonacci中的第n个数是多少？如:1 1 2 3 5 8 13 21...只要我们知道前面的两个数是多少，后面的数都可以求。用递归也可以做。

|  |
| --- |
| /\*n的值默认为大于2的数，否则需要先进行判断\*/  function Fab(n) {  var n1 = 1, n2 = 1, n3;  for (var i = 3; i <= n; i++) {  n3 = n2 + n1;  n1 = n2;  n2 = n3;  }  return n3;  }  var n = 9;  console.log(Fab(n)); |
| 用递归方式  function getFab(n) {  if (n === 1 || n === 2) {  return 1;  } else {  return getFab(n - 1) + getFab(n - 2);  }  }  console.log(getFab(9));  /\*输出 34\*/ |

（3）\*冒泡对数组排序，从小到大

|  |
| --- |
| /\*冒泡排序\*/  function sortOfMaoPao(array) {  var temp;  var isSort = true;//先假设已经排好顺序了，如果发送交换则说明没有  console.log(arguments);  for (var i = 0; i < array.length - 1; i++) {  for (var j = 0; j < array.length-1-i; j++) {  if(array[j] > array[j+1]) {  temp = array[j+1];  array[j+1] = array[j];  array[j] = temp;  isSort = false;  }  }  if (isSort) {  break;  }  }  }  var array = [1,22,3,5,66,777];  sortOfMaoPao(array);  console.log(array); |

（4）\*输入某年某月某日，判断这一天是这一年的第几天？

|  |
| --- |
| function getDays(year,month,day) {  var days = day;  /\*计算的时候，当前月份不要进行计算\*/  for (var i = 1; i < month; i++) {  switch(i) {  case 4:  case 6:  case 9:  case 11:  days += 30;  break;  case 2:  if ((year % 4 ===0 && year % 100 !== 0)||(year % 400 === 0)) {  days += 29;  } else {  days += 28;  }  break;  default:  days += 31;  break;  }  }  return days;  }  console.log(getDays(2001,3,29));  //输出 88 |

二十、函数其他

1、匿名函数

匿名函数：**没有名字的函数**，它**不能独立存在**，直接写会报错。就得按照以下的两个方法去做。

匿名函数如何使用：

（1）将匿名函数赋值给一个变量，这样就可以通过变量进行调用。

（2）匿名函数自调用。

（1）自调用函数

匿名函数不能通过直接调用来执行，因此可以通过匿名函数的自调用的方式来执行，当书写完成之后**立即进行调用**。

同时定义了变量、匿名函数或者闭包函数，并且该变量定义在了这两类函数之前，如果变量后面省略了分号，js就会自动解析为匿名函数进行自调，会被认为是变量名而不是方法，从而会导致报错；所以**变量后面以及闭包函数后面都需要加上分号**。

|  |
| --- |
| (function () {  alert(123);  })();  前面的一个括号是函数表达式？？？，后边的小括号是对函数进行调用。 |

（2）自调用函数容易出现的问题

|  |
| --- |
| <script type="text/javascript">  (function () {  console.log(1);  })()  (function () {  console.log(2);  })()  </script>  1  属于js 中不可忽略分号的场景，因为在语法分析阶段就出现了错误。  如果你补全第一个自动执行函数的分号，这段代码将会正常运行。  当然，也不一定使用分号，你还可以在每个自执行函数前面加上!、以及其他普通算术运算符+ - \* /。  另外你会发现一些第三方类库大多都会在开头写上 ! 或者; ，就是为了避免在一些压缩合并后的 js 出现上面的错误。  !(**function**(){*// some code*  })() |
| 第一个函数执行完之后会返回undefined  <script type="text/javascript">  undefined(function () {  console.log(2);  })()  </script>  此时代码可能会进行优化，把空格直接去除  通过加上分号，告诉解析器这个语句已经结束了，不再与后边产生关系。  当写多个连续的自调用函数的时候，要写上分号，别人的代码一般是加上自调用函数的前边：**; (function () {})()**，加在前边，表示前面的语句已经结束了。前面没有内容，表示那是一条空语句 |

|  |
| --- |
| <script type="text/javascript">  var fn = function () {  console.log(1);  }**;**  **;**(function (){  console.log(2);  })()  </script>  2  添加分号解决，具体待理解？？？ |

（3）自调用函数带参数window、undefined

开始看到很多js函数里都带一个undefined的参数，很是疑惑，后来查了查，原来是这样，函数参数是undefined，那这个undefined是个变量标识符还是值呢？

未给函数形参传值的时候，该形参的值是undefined。所以上面的形参undefined是一个**局部变量标识符**，该变**量值是undefined**。注意，undefined不是**JS保留字和关键字，说明可以作为变量名**。

假如我们定义了一个函数

|  |
| --- |
| function a(){  if(arg1===undefined)  alert("arg1未定义");  } |

有在某一个地方，由于什么原因，存在一个变量**undefined="xx"(undefined不是保留字)**。这样当我们调用函数a();时就不会输出"arg1未定义",但事实上arg1确实未定义。

但当我们这样定义函数a

|  |
| --- |
| function a(undefined){    if(arg1===undefined)  alert("arg1未定义");  } |

在调用a();由于未传入参数，所以函数a的参数undefined等价于未定义，再判断arg1===undefined时就会返回true，得到正确结果。

所以，给函数在原有基础上增加一个undefined的参数，从某种程度上防止的undefined被变量污染。当然这个undefined也可以用其他标示符定义，但那样就不直观，就不容易理解了。

（4）自调用函数传递参数

|  |
| --- |
| var inputList = document.getElementsByTagName('input');  for (var i = 0, len = inputList.length; i < len; i++) {  (function (**i**) { // 形式参数  inputList[i].onclick = function () {  console.log(i);  }  })**(i)**;  } |

2、函数是一种数据类型

|  |
| --- |
| function fn() {}  console.log(typeof fn);//输出 function 类型。 |

（1）函数作为参数

因为函数也是一种类型，可以把函数作为两一个函数的参数，在另一个函数中调用，现在还没有合适的场景来进行运用。

|  |
| --- |
| var fn = function () {  console.log('函数作为形参');  }  function test(func) {  func();  }  test(fn); |

（2）函数做为返回值

因为函数是一种类型，所以可以把函数可以作为返回值从函数内部返回。

|  |
| --- |
| function fn(b) {  var a = 10;  return function () {  alert(a+b);  }  }  var test = fn(10);//此函数返回的结果是一个函数。能否认为是传回来函数的地址？？？  test(); |

3、回调函数

（1）定义：一个函数被**作为参数**传递给**另一个函数**（在这里我们把另一个函数叫做“otherFunction”），回调函数在otherFunction中被调用。

（2）特点：

注意的很重要的一点是回调函数并**不会马上被执行**。它会在包含它的函数内的某个特定时间点被“回调”。

二十一、代码规范

|  |
| --- |
| 1.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.变量规范  操作符的前后要有空格  var name = 'zs';  3.注释规范  // 这里是注释  4.空格规范  if (true) {    }  for (var i = 0; i <= 100; i++) {    }  5.换行规范  var arr = [1, 2, 3, 4];  if (a > b) {    }  for (var i = 0; i < 10; i++) {    }  function fn() {    } |

二十二、作用域

作用域：变量可以起作用的范围

（1）全局变量

在任何地方都可以访问到的变量就是全局变量，对应全局作用域，在script或者一个独立的javaScript文件中。

|  |
| --- |
| 不使用var声明的变量是全局变量，不推荐使用。  变量退出作用域之后会销毁，全局变量关闭网页或浏览器才会销毁。 |

在js中定义的全局变量是挂在window下的，而window的属性也一样，那么这两者有什么区别呢？

a = 2;

console.log(a);          // 2

console.log(window.a);   // 2

其实这两者还是有小小的区别的，全局变量是不能通过delete操作符删除的，而直接定义在window上的属性是可以删除的。

那么为什么全局变量不能删除呢？

因为全局变量也是个对象，这个对象时通过叫做PropertyDescriptor来定义的。

可以在浏览器中调用Object.getOwnPropertyDescriptor(对象, 属性)来查看

举例：

var test = 2;

window.test; // 2

Object.getOwnPropertyDescriptor(window, "test") //Object {value: 2, writable: true, enumerable: true, configurable: false}

大家可以看到最后一行有configurable（这个方法用于获取defineProperty方法设置的property 特性）的特性，这个特性决定了该属性是不能被删除的。

var age = 22；

　　window.color = 'red';

　　delete window.age; // IE<9时报错  其他返回false

　　delete window.color； //IE<9报错  其他返回true

　　alert（window.age）//22

　　alert（window.color）//red

|  |
| --- |
| <script type="text/javascript">  var a = 123;  window.b = 'b';  console.log('window.a a');  console.log(window.a,a);  console.log(delete window.a);  console.log(delete window.b);  </script>  2 |

（2）局部变量

只在固定的代码片段内可访问到的变量，最常见的例如函数内部。对应局部作用域(函数作用域)，任何一个函数的内部都有一个局部作用域。

（3）块级作用域

任何一对花括号**｛ ｝**中的语句集都属于一个块，在这之中定义的 **let 变量**在代码块外都是不可见的，**var 变量不存在块级作用域**，我们称之为块级作用域。

|  |
| --- |
| {  **var** num = 5;  }  console.log(num);  {  **let** num1 = 5;  }  console.log(num1);  2 |

二十二、作用域链

只有函数可以制造作用域结构， 那么只要是代码，就至少有一个作用域, 即全局作用域。凡是代码中有函数，那么这个函数就构成另一个作用域。如果函数中还有函数，那么在这个作用域中就又可以诞生一个作用域。

将这样的所有的作用域列出来，可以有一个结构: 函数**内指向函数外**的链式结构。就称作，作用域链。

|  |
| --- |
| // 案例1：  function f1() {  function f2() {  }  }  var num = 456;  function f3() {  function f4() {  }  }  06-1 |

|  |
| --- |
| var num =10;// 全局作用域 -- 0级作用域  function f1() {  var num = 100;// 局部作用域 -- 1级作用域  function f2() {  console.log(num);// 局部作用域 -- 2级作用域  /\*打印的num会找当前作用域，然后再找最近的作用域中寻找变量的值\*/  }  f2();  }  f1();  06-2 |

二十三、预解析

JavaScript代码的执行是由浏览器中的JavaScript解析器来执行的。JavaScript解析器执行JavaScript代码的时候，分为两个过程：**预解析过程**和**代码执行过程。**

1、预解析过程：

（1）把**变量的声明**（var num; 其赋值不会提升）提升到当前作用域的最前面，只会提升声明，不会提升赋值。

（2）把**函数的声明**提升到当前作用域的最前面，只会提升声明，不会提升调用。

（3）先提升**var**，在提升**function**。

（4）在解析过程中如果函数和变量的名字相同，此时**函数优先**。见案例2

**注：**匿名函数，或者函数表达式，它们是不会提升

2、JavaScript的执行过程

|  |
| --- |
| // 案例1  var a = 25;  function abc() {  alert(a);  var a = 10;  }  abc();  预解析  var a;  function abc() {  var a;//进行局部预解析  alert(a);  a = 10;  }  a = 25;  abc();  2  // 案例2  console.log(a);  function a() {  console.log('aaaaa');  }  var a = 1;  console.log(a);  预解析  var a;  function a() {  console.log('aaaaa');  }  console.log(a);  a = 1;  console.log(a);  //在解析过程中如果函数和变量的名字相同，此时函数优先  2 |

|  |
| --- |
| //2、-----------------------------------  var a = 18;  f1();  function f1() {  var b = 9;  console.log(a);  console.log(b);  var a = '123';  } |
| 预解析  var a;  function f1() {  var b;  var a;  b = 9;  console.log(a);  console.log(b);  a = '123';  }  a = 18;  f1();  2 |

|  |
| --- |
| // 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);  } |
| function f1() {  var a;  **a = b = c = 9;// b、c隐式声明为全局变量**  console.log(a);  console.log(b);  console.log(c);  }  f1();  console.log(c);  console.log(b);  console.log(a);  2 |

二十四、对象

|  |
| --- |
| function printPerson(name, age, sex....) {  }  // 函数的参数如果特别多的话，可以使用对象简化  function printPerson(person) {  console.log(person.name);  ……  }  这样子参数比较多，不好操作。 |

1、什么是对象

现实生活中：万物皆对象，对象是一个具体的事物，要具体到某个人，某辆车，某个椅子等等，一个具体的事物就会有行为和特征。

车是一类事物，门口停的那辆车才是对象。

特征：红色、四个轮子。

行为：驾驶、刹车。

2、JavaScript中的对象

JavaScript中的对象其实就是生活中对象的一个抽象。

JavaScript的对象是**无序属性（属性不是变量）**的集合。

其属性可以包含基本值、对象或函数。对象就是一组没有顺序的值。我们可以把JavaScript中的对象想象成**键值对**，其中值可以是**数据和函数**。

对象的行为和特征。

特征---属性。

行为---方法。

3、对象字面量

|  |
| --- |
| var student = {  name: 'lxf',  age: 25,  sex: '男',  playGame: function() {  console.log('playing game');  }  };  console.log(student.age); |

4、访问对象的属性或者方法

（1）对象封装一组属性和方法。this在方法中表示的是当前的对象

如何访问属性：对象.属性名

如何访问方法：对象.方法

访问属性的另外一种方式：console.log(**dog['name']**); 类似于数组，里面使用属性字符串。

（2）函数与方法的区别

函数是独立存在的，通过函数名进行调用。

方法要通过对象的调用才能执行。

|  |
| --- |
| 字面量方式创建对象。  var student = {  name: 'lxf',  age: 25,  sex: '男',  hobbies: **['play','eating','running'],//可以是数组**  playGame: function() {  console.log(this.name+'playing game');  }  };  console.log(student.age);  student.playGame();  console.log(student['name']);  console.log(student['hobbies']);  2 |

5、对象创建方式

（1）对象字面量方式创建，其实是new Object() 的简写。

就是上边所示的方式。

（2）new Object()方式

new是关键字，Object()是构造函数，通过new的方式来调用构造函数。第一步创建了一个空的对象。

|  |
| --- |
| var person = new Object();//创建一个空的对象，调用构造函数必须通过new的方式，会在内存中创建一个对象，普通函数直接通过函数名的方式。  person.name = 'lisi';//给对象动态添加属性，方法，JavaScript动态特性  person.age = 35;  person.job = 'actor';  person.sayHi = function() {  console.log(this.name + ' Hello,everyBody');  }  console.log(person.sex);//打印一个不存在属性，输出undefined |

如果要创建多个类似的对象，那么上边的两个方式都不太行得通。相同的属性要写多次。

（3）工厂函数创建对象，相当于把创建的函数包装一下

|  |
| --- |
| function createPerson(name, age, job) {  var person = new Object();  person.name = name;  person.age = age;  person.job = job;  person.sayHi = function(){  console.log(this.name + ' Hello,everyBody');  }  return person;  }  var p1 = createPerson('张三', 22, 'actor'); |

（4）自定义构造函数

比工厂模式少了两行代码。这个不是普通函数，具有语义性，要new。**它跟new 在一起的时候是构造函数，不在一起使用时是普通函数。**

|  |
| --- |
| //帕斯卡命名，第一个单词的第一个字母大写，后续的每一个单词的第一个字母都大写。  function Person(name, age, job){  this.name = name;//动态的给对象增加成员，this指向了当前对象，谁创建了就指向谁  this.age = age;  this.job = job;  this.sayHi = function(){  console.log('Hello,everyBody');  }  }  var p1 = new Person('张三', 22, 'actor'); |

（5）给对象添加属性

Object[name] = xxx;这样也能添加属性？？？

6、属性和方法，两者可以统称为：**成员**

|  |
| --- |
| 如果一个变量属于一个对象所有，那么该变量就可以称之为该对象的一个属性，属性一般是名词，用来描述事物的特征  如果一个函数属于一个对象所有，那么该函数就可以称之为该对象的一个方法，方法是动词，描述事物的行为和功能  对象的属性是一个字符串的类型。  // 这种方式是直接更改数组元素，数组会发生变化。  function noRepeat1(array) {  var obj = {};  for (var i = 0; i < array.length; i++) {  if (!obj[array[i] + **(typeof(array[i]))**]) {// 实际上是对象的属性都是是字符串，如果array是数字，他会隐式转换为字符串  obj[array[i] + **(typeof(array[i]))**] = 1;// 红色加粗部分如果去掉，array是数字，他会隐式转换为字符串  } else {  array.splice(i, 1);  i--;//因为此时的数组已经发生变化了，元素都往前移动了一位。  }  }  return;  }  这个方法实现的是数组的去重，什么样的类型都可以去掉 |

7、new关键字

构造函数，是一种特殊的函数。主要用来在创建对象时初始化对象，即为对象成员变量赋初始值，总与new运算符一起使用在创建对象的语句中。

（1）构造函数用于创建一类对象，首字母要大写。

（2）构造函数要和new一起使用才有意义。

（3）new在执行时会做四件事情

|  |
| --- |
| 1、new会在内存中**创建**一个**新的空对象**。  2、new 会让构造函数中的 **this** 指向刚刚创建的对象。  3、执行构造函数，在构造函数中设置属性和方法（当然也可以做其他的事情，如console.log(xxx);）  4、new会**返回这个新的当前对象**,构造函数并没有写返回值，是内部帮我们处理这个事，让我们的代码可以更简洁。 |

8、this详解

（1）this 出现在以下位置：

1 函数中 --- this 指向Window

2 方法中 --- this 指向的是这个方法的所属对象

3 构造函数中 --- this 就是构造函数创建的对象

4 事件处理函数中的 this --- 指向事件源，谁调用的该事件就指向谁。

|  |
| --- |
| // 普通函数  function fn() {  console.log(this);  }  fn();  // 方法中  var student = {  name: 'xiao',  fn: function () {  console.log(this);  }  }  student.fn();  // 构造函数中  function Fn() {  this.name = 'lxf',  console.log(this);  }  var temp = new Fn();  Fn();//当成普通函数来使用 |
| 23 |

|  |
| --- |
| JavaScript中的this指向问题，有时候会让人难以捉摸，随着学习的深入，我们可以逐渐了解。  现在我们需要掌握函数内部的this几个特点  1. 函数在**定义**的时候**this是不确定**的，只有在调用的时候才可以确定。  2. 一般函数直接执行，内部**this指向全局window**。  3. 函数作为一个对象的**方法**，被该对象所调用，那么this指向的是该对象。  4. 构造函数中的this其实是一个隐式对象，类似一个初始化的模型，所有方法和属性都挂载到了这个隐式对象身上，后续通过new关键字来调用，从而实现实例化。 |

9、遍历对象的属性

（1）通过for..in语法可以遍历一个对象。

|  |
| --- |
| var obj = {};  for (var i = 0; i < 10; i++) {  obj['a' + i] = i \* 2;  }  //以上是动态的给对象添加属性。  for(var key in obj) {//key 就是属性的名字，是一个字符串  console.log(key + "==" + obj[key]);  } |

（2）删除对象的属性。

|  |
| --- |
| function fun() {  this.name = 'mm';  }  var obj = new fun();  console.log(obj.name); // 输出mm  delete obj.name;  console.log(obj.name); // 输出undefined |

|  |
| --- |
| function testForIn() {  this.name = 'lxf',  this.age = 22,  this.say = function () {  console.log(this.name + ' hi');  }  }  var test = new testForIn();  for (var key in test) {  console.log(test[key]);  }  delete test.name;  console.log('执行delete name');  for (var key in test) {  console.log(test[key]);  } |
| 2 |

二十五、简单类型和复杂类型的区别

简单数据类型、基本数据类型、值类型。

Number、String、Boolean、Null、Undefined

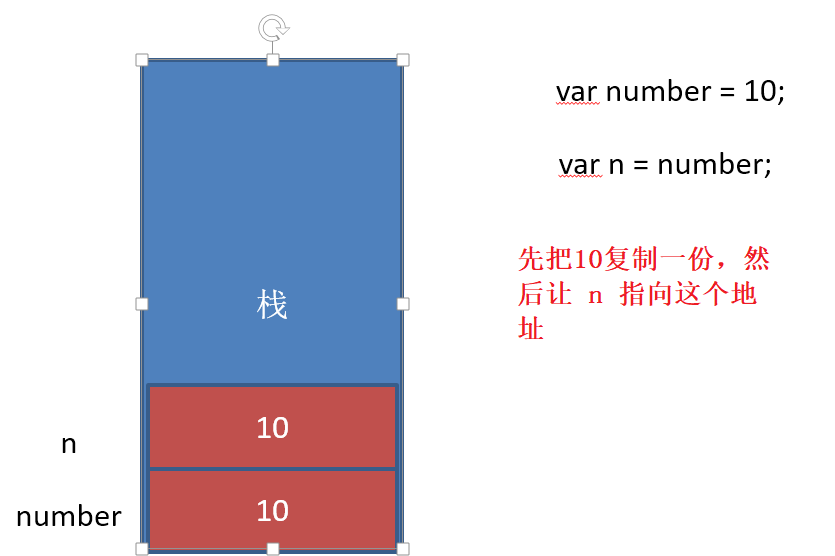
复杂数据类型、引用类型。

Object、数组也是复杂数据类型

1、堆和栈

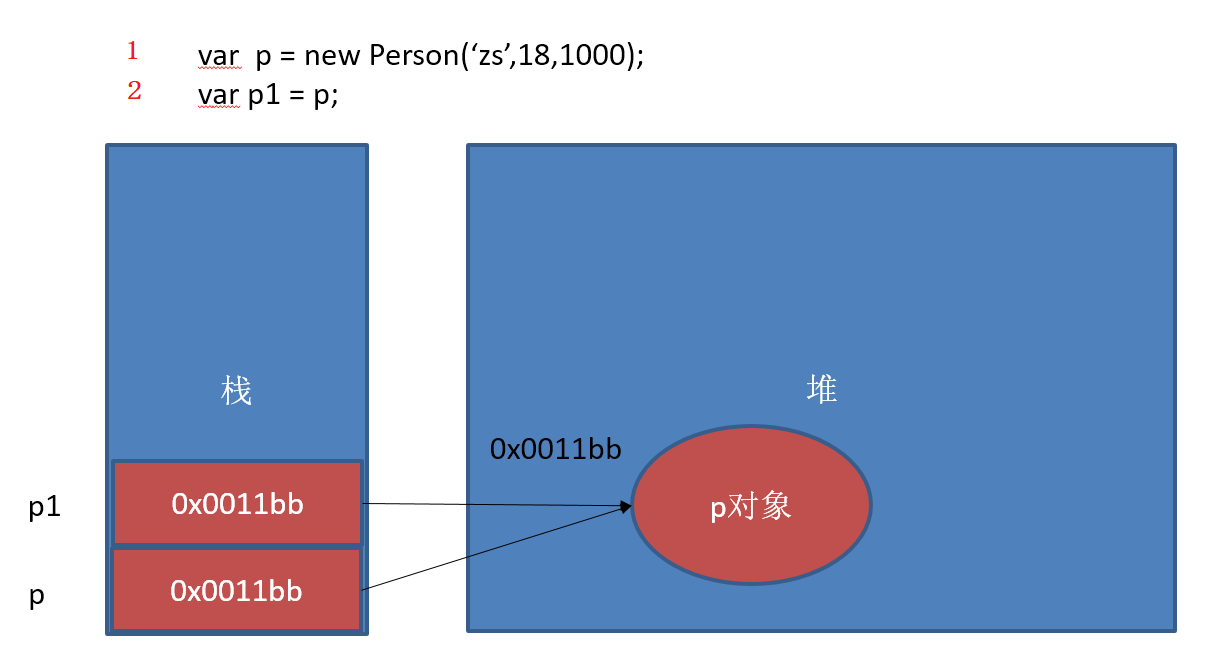
|  |
| --- |
| 堆栈空间分配区别：  　　1、栈（操作系统）：由操作系统自动分配、释放，存放函数的参数值，局部变量的值等。  2、堆（操作系统）：存储复杂类型(对象)，一般由程序员手动分配、释放， 若程序员不释放，由垃圾回收机制回收。  注意：JavaScript中没有堆和栈的概念，此处我们用堆和栈来讲解，目的方便理解和方便以后的学习。 |

2、**基本类型**在内存中的存储



注：如果他们没写在函数中，有可能是在静态存储区的。待验证？？？

3、**复杂类型**在内存中的存储，所有的对象都是基于 Object ，所以都是复杂类型。



解析

（1）当 1 执行的时候，new Person()会在“堆”中开辟出空间。

（2）然后在“栈”开辟空间中把“堆”的中内存的地址记录下来。

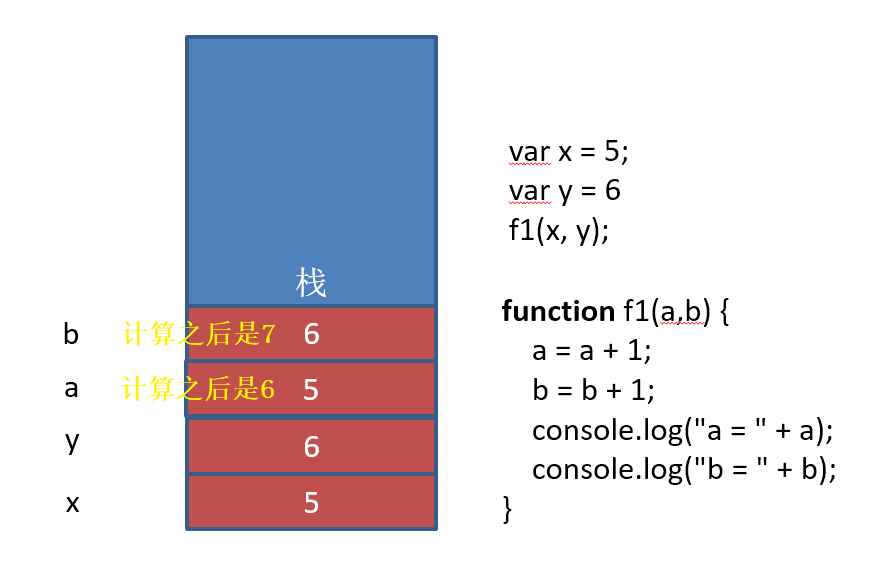
（3）变量 p 会指向“栈”中的地址。

所以访问对象要经过**两次寻址**，第一次找到“栈”的地址，第二次根据“栈”中的地址，找到在“堆”中的位置

当p1.name = ‘ls’;

那么p2.name的值是多少？答案是 ls ，他们两个指的地址是同一个。

4、基本类型作为函数的参数



这里的x、y两个变量应该是全局变量，不应该在栈中的，应该是在静态存储区的。待验证一下。？？？

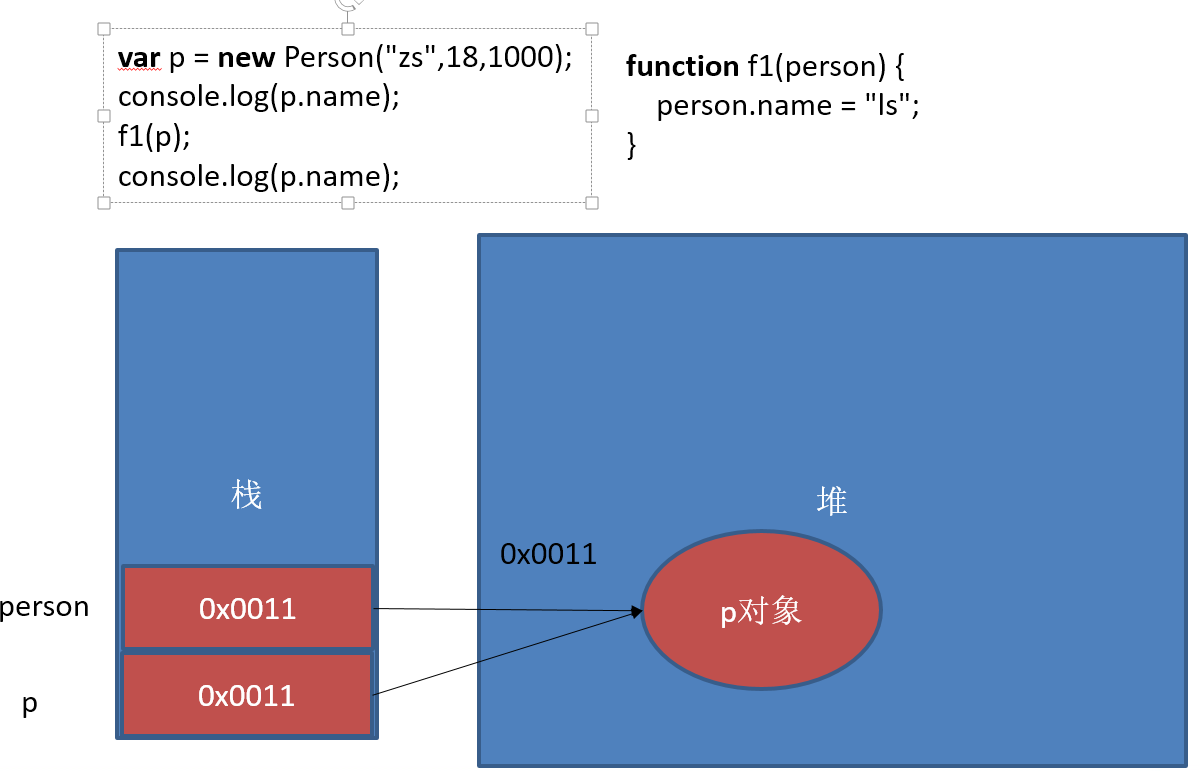
（1）首先开辟一个在栈中的空间，用来存储数值10；然后再让变量x指向这个空间。

（2）在开辟一块空间，把20存储进去，让变量y指向那个空间。

（3）在函数内部要开辟一个局部变量区域，变量的名字叫a，在内存中开辟一块空间，把变量x的值10 赋值到a的空间去。

（4）a、b在函数执行完毕之后，它们的空间会被释放掉。

5、复杂类型作为函数的参数



函数中先在栈中开辟一个空间，然后把p的值复制到空间去。

person就指向了这个区域。这样他们俩的地址一样，指向了堆中的同一个对象。

1. 练习

|  |
| --- |
| // 下面代码输出的结果?  function Person(name,age,salary) {  this.name = name;  this.age = age;  this.salary = salary;  }  function f1(person) {  person.name = "ls";//现在的地址跟下面一句话的指针的地址的不同的  person = new Person("aa",18,10);//这里在堆中创建了一个新的内存空间，在栈的地址也会随着更改为新对象在堆中的地址。是person这个局部变量的在栈中的值被改变了，不会影响到p。函数退出后，person也会被释放掉。  }  var p = new Person("zs",18,1000);  console.log(p.name);**//输出 zs**  f1(p);  console.log(p.name);**//输出 ls** |
| //1.  var num1 = 10;  var num2 = num1;  num1 = 20;  console.log(num1);**//输出 20**  console.log(num2);**//输出 10**  //2.  var num = 50;  function f1(num) {  num = 60;  console.log(num);  }  f1(num); **//输出 60**  console.log(num); **//输出 50**  **//3**  var num = 50;  function f1() {  num = 60;  console.log(num);  }  f1(num); **//输出 60**  console.log(num); **//输出 60** |

二十六、内置对象

JavaScript中的对象分为3种：**内置对象**、**自定义对象**、**浏览器对象**(它不属于ECMAscript)。

JavaScript 提供多个内置对象：Math/Array/Date....就是系统已经提供给我们使用的

对象只是带有**属性**和**方法**的特殊数据类型。学习一个内置对象的使用，只要学会其常用的成员的使用（通过查文档学习），可以通过**MDN**/W3C来查询，内置对象的方法很多，我们只需要知道内置对象提供的常用方法，使用的时候查询文档。

1、MDN

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

**Mozilla 开发者网络**（MDN）提供有关开放网络技术（Open Web）的信息，包括 HTML、CSS 和万维网及 HTML5 应用的 API。

通过查询MDN学习Math对象的random()方法的使用。

如何学习一个方法？

（1）方法的功能。

（2）参数的意义和类型。

（3）返回值意义和类。

（4）demo进行测试。

2、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，都是以静态成员的方式提供

跟数学相关的运算来找Math中的成员（求绝对值，取整）

演示：Math.PI、Math.random()、Math.floor()/Math.ceil()、Math.round()、Math.abs()、Math.max()

|  |
| --- |
| Math.PI // 圆周率  Math.random() // 生成随机数  Math.floor()/Math.ceil() // 向下取整/向上取整  Math.round() // 取整，四舍五入  Math.abs() // 绝对值  Math.max()/Math.min() // 求最大和最小值  Math.sin()/Math.cos() // 正弦/余弦  Math.power()/Math.sqrt() // 求指数次幂/求平方根 |

1. 求10-20之间的随机数 [ 10, 20 ]之间的整数

|  |
| --- |
| var random(min, max) {  return Math.floor(Math.random()\*(max - min + 1) + min);  }  解析： Math.random()\*(max - min + 1) 它的区间是 [ 0, 11)  加上 min 区间为 [ 11 , 21)  取值最大不能到达21，只能为 20.9多，向下取整后就是 20，不会超过 |

1. 随机生成颜色RGB

|  |
| --- |
| function getRandomColor(min, max) {  var r = random(min, max);  var g = random(min, max);  var b = random(min, max);  return 'rgb(' + r + ',' + g + ',' + b + ')';  }  console.log(getRandomColor(0,255));  1 |

1. 模拟实现max()/min()

|  |
| --- |
| function max() {  var max = arguments[0];  for (var i = 1; i < arguments.length; i++) {  if (max < arguments[i]) {  max = arguments[i];  }  }  return max;  }  console.log(max(1,2,3,4444,55,6,7000));  // 现在只是一个函数，要把它变成方法，还要放到对象中去。  var myMath = {  max: max,  }  console.log(myMath.max(2,3,4,5,6,3));  1 |

Math对象和Date对象的区别

Math不是一个构造函数，里面提供的是静态成员

var myMath = {

PI:3.14,

Max: function () {

}

}

静态成员

myMath.PI

myMath.max()

3、Date对象

Date是一个构造函数，首先要通过 new Date() 来创建日期实例对象。实例成员。

GMT：格林威治时间，是世界标准时间。它是零时区。

GMT+0800 (中国标准时间)，中国是八时区，快8个小时。

计算机存储的是距离1970-1-1相差的毫秒数，它之前的日期用负数表示，之后的用正数。

（1）日期构造函数 Date()

|  |
| --- |
| 1、空构造函数，获取的是当前时间的对象  var date = new Date();  console.log(date);  console.log(date.valueOf());//一般不推荐使用这个，不是在代码中显示调用，它是对象隐式转换的时候，内部自己调用的，使用getTime代替 |
| 2、构造函数中传入毫秒值  var date = new Date(1616317589644);  console.log(date);  console.log(date.getTime());  时间就会停在那，不会动了。  1 |
| 3、可以传入日期形式的字符串  /\*var date = new Date('2021-1-1 1:1:1');\*/  var date = new Date('2021/1/1 1:1:1');  console.log(date);  console.log(date.getTime());  1 |
| 4、可以传入数字，**要注意月份，传入1，表示的是2月份**  var date = new Date(1988, 1, 2);  console.log(date); |

（2）获取日期的毫秒形式

|  |
| --- |
| var now = new Date();  // valueOf用于获取对象的原始值  console.log(date.valueOf())  // HTML5中提供的方法，有兼容性问题，静态成员，这里怎么**理解？？？**  var now = Date.now();  // 不支持HTML5的浏览器，可以用下面这种方式  var now = + new Date(); // 这里是取正，相当于把时间转换成数字，隐式调用 Date对象的valueOf() |

（3）日期格式化方法，作为了解，实际很少使用

|  |
| --- |
| toString() // 转换成字符串  valueOf() // 获取毫秒值  // 下面格式化日期的方法，在不同浏览器可能表现不一致，一般不用，下边的方法基本是不用的  toDateString()  toTimeString()  toLocaleDateString()//本地化，根据当前的操作系统的语言，返回对应的日期格式  toLocaleTimeString() |

（4）获取日期指定部分

|  |
| --- |
| getTime() // 返回毫秒数和valueOf()结果一样  getMilliseconds()  getSeconds() // 返回0-59  getMinutes() // 返回0-59  getHours() // 返回0-23  getDay() // 返回星期几 0是周日、6是周六  getDate() // 返回当前月的第几天  getMonth() // 返回月份，\*\*\*从0开始\*\*\*，这个就要稍微进行记忆，返回3，表示4月  getFullYear() //返回4位的年份 如 2016 |

4、案例

（1）写一个函数，格式化日期对象，返回yyyy-MM-dd HH:mm:ss的形式

**实例 instanceof Date** 表示这个实例是不是通过 new Date()构造函数创建出来的，是的话返回 true，否则返回 false。

|  |
| --- |
| /\*function getFormatDate() {  var date = new Date();  return date.getFullYear() + '-' + date.getMonth()+1 + '-' + date.getDate() + ' ' + date.getHours() + ':' + date.getMinutes() + ':' + date.getSeconds();  }  console.log(getFormatDate());  以上的写法比较简单，问题也比较多  1、无法设置时间  2、打印出来的数据会有对不齐的问题，如1跟10，如果是01跟10就可以\*/  function getFormatDate(date) {  if (!(date instanceof Date)) {// 感叹号，取反的优先级比较高  console.error('输入的参数不是Date的实例');  return;// 如果没有返回，在正常的情况下会继续执行，输出 getFullYear函数错误，因为数组对象没有这个函数。  }  var year = date.getFullYear(),  month = date.getMonth() + 1,  day = date.getDate(),  hour = date.getHours(),  minute = date.getMinutes(),  second = date.getSeconds();  month = month < 10 ? '0' + month : month;  day = day < 10 ? '0' + day : day;  hour = hour < 10 ? '0' + hour : hour;  minute = minute < 10 ? '0' + minute : minute;  second = second < 10 ? '0' + second : second;  return year + '-' + month + '-' + day + ' ' + hour + ':' + minute + ':' + second;  }  var date = new Date();  console.log(getFormatDate(date));  var d = new Date(1999,2,3);  console.log(getFormatDate(d));  var da = [];  console.log(getFormatDate(da)); |
| 1  2 |

（2）计算时间差，返回相差的天/时/分/秒

|  |
| --- |
| 问题点   1. 两个日期对象如何相减。   var d1 = new Date();  var d2 = new Date(2021,4,1);  // 隐含d2.valueOf() - d1.valueOf()  console.log(d2 - d1); // 相差的毫秒数   1. 函数如何返回多个值。   通过返回一个对象  function fn() {  Return {  name:xx;  age:xxx;  };  }   1. 如何计算相差的 天/时/分/秒。   相差的总的秒数 = 毫秒 / 1000  相差的天数 = 毫秒 / 1000 / 60 / 60 / 24  相差的小时数 = 毫秒 / 1000 / 60 / 60 % 24  相差的分钟数 = 毫秒 / 1000 / 60 % 60  相差的秒数 = 毫秒 / 1000 % 60 ----> 一共要走多少圈，一圈60秒  看小时需要走多少圈，例如3603秒，此时3603/3600=1.0008 % 24 = 1.0008，此时剩下大约1小时，0.0008是还有其他的秒数导致的。还要在继续理解一下，初步大概理解了。  进一步的理解：例如分钟：有 7300 秒，7300 / 60 向下取整121分钟，121 % 60 = 1，  时分秒它们三是独立的个体，不要联想在一起，你就算出是多少分钟，然后分针最后在哪就行了，在 0- 59之间。 |
| function diffDate(date1, date2) {  if (!(date1 instanceof Date) || !(date2 instanceof Date)) {  console.error('参数有误，非日期对象');  }  var diff = (date2 - date1) / 1000;  console.log(diff);  var day = Math.floor(diff / 60 / 60 / 24),  hour = Math.floor(diff / 60 / 60 % 24),  minute = Math.floor(diff / 60 % 60),  second = Math.floor(diff % 60);  return {  day: day,  hour: hour,  minute: minute,  second: second  }  }  var date1 = new Date(2021,2,22,11,0,10);  var date2 = new Date(2021,2,24,11,0,0);  var result = diffDate(date1,date2);  console.log(result);  1 |

5、Array对象

1. 创建数组对象的两种方式，字面量是new Array()的简写，**他们之间没有区别**。

|  |
| --- |
| 1 字面量方式  var array = [];// 创建空数组  var num = [1,2,3]; |
| 2 new Array()  var array = new Array();// 创建空数组  var num = new Array(1,2,3); |

（2）如何判断一个变量是不是“数组对象”。

在函数中添加对传入的参数进行判断，让程序能更好正确执行。函数的参数，如果要求是一个数组的话，可以用这种方式来进行判断。

|  |
| --- |
| 1 instanceof 这个没有兼容性问题。 |
| var array = new Array();  var num = new Array(1,2,3);  function fn(array) {  if (**!(array instanceof Array)**) {  console.error('参数error，不是数组');  **return;**  }  console.log('正常执行');  }  fn(array); |

|  |
| --- |
| 2 Array.isArray(数组)。HTML5中提供的方法，浏览器有兼容性问题，要支持html5的浏览器就支持这个方法，否则就不支持。 |
| var array = new Array();  var num = new Array(1,2,3);  function fn(array) {  if (**!Array.isArray(array)**) {  console.error('参数error，不是数组');  **return;**  }  console.log('正常执行');  }  fn(array); |

（3）toString()/valueOf()，每一个对象都具有这两个方法，只不过他们打印的值可能有点不同。

toString() 把数组转换成字符串，逗号分隔每一项。

valueOf() 返回数组对象本身。

（4）数组常用方法

|  |
| --- |
| 1 栈操作(先进后出)，相当于是弹夹  push()//返回数组添加后长度  pop() //从一个数组中删除并返回最后一个元素，修改length属性。  2 队列操作(先进先出)，相当于是排队  push()  shift() //从数组中删除第一个元素，并返回该元素的值。此方法更改数组的长度。后边元素会往前移动一个位置  unshift() //一个或多个元素添加到数组的开头，并返回该数组的新长度(该方法修改原有数组)。  3 排序方法  reverse() //方法将数组中元素的位置颠倒，并返回该数组。数组的第一个元素会变成最后一个，数组的最后一个元素变成第一个。该方法会改变原数组。  4 操作方法  concat() //把参数拼接到当前数组。该方法不会改变现有的数组，而仅仅会返回被连接数组的一个副本。  slice() //从当前数组中截取一个新的数组，不影响原来的数组，参数start从0开始，end从1开始。  如果省略 begin，则 slice 从索引 0 开始。  如果 end 被省略，则 slice 会一直提取到原数组末尾。  splice() //splice(index,count,ele1,...,ele2);  //index是起始索引，count表示从index开始要删除的个数，后面的参数表示从index开始插入的值。  //此时的数组已经发生变化了，元素都往前移动了一位。循环变量i 也要跟着改变。  5 位置方法  indexOf()、索引值大于或等于数组长度，意味着不会在数组里查找，返回-1。参数可以是任何基本类型  lastIndexOf() // 如果没找到返回-1，一个从前往后，一个从后往前。  6 迭代方法（内部已经自带循环）不会修改原数组(可选) 是html5提供的，会存在兼容性的问题。  every()、  filter()、方法创建一个新数组, 其包含通过所提供函数实现的测试的所有元素。意思就是可以过滤一些不符合条件的元素。  forEach()、  map()、  some()  7 方法将数组的所有元素连接到一个字符串中。  join() // 它是返回一个新的连接后的字符串，原来的数组不会有变化。如果separator是空字符串("")，则所有元素之间都没有任何字符 |

（5）排序方法sort();

|  |
| --- |
| （1）默认排序顺序是在将元素转换为字符串，然后比较它们的UTF-16代码单元值序列时构建的，utf-16中的英文字母和数字与ASCII码是一致的。  var array = [1,2,3,4,3,22,11,43];  console.log(array);  array.sort();  console.log(array);  1 |
| （2）arr.sort([compareFunction])   * 如果 compareFunction(a, b) 小于 0 ，那么 a 会被排列到 b 之前； * 如果 compareFunction(a, b) 等于 0 ， a 和 b 的相对位置不变。备注： ECMAScript 标准并不保证这一行为，而且也不是所有浏览器都会遵守（例如 Mozilla 在 2003 年之前的版本）； * 如果 compareFunction(a, b) 大于 0 ， b 会被排列到 a 之前。 * compareFunction(a, b) 必须总是对相同的输入返回相同的比较结果，否则排序的结果将是不确定的。   function compare(a, b) {  return a - b;//如果换成 b - a，那么数组从大到小排列。  }  var array = [1,2,3,4,3,22,11,43];  console.log(array);  array.sort(compare);  console.log(array);  1 |
| （3）对象可以按照某个属性排序：  function compare(a, b) {  return a.length - b.length;  }  // 按照字符串的字符个数  var array = ['a','abc','xd','ab','ddd'];  console.log(array);  array.sort(compare);  console.log(array);  1  另外的写法  var array = ['a','abc','xd','ab','ddd'];  console.log(array);  array.sort(function (a, b) {  return a.length - b.length;  });  console.log(array); |

（6）模拟sort方法排序

使用冒泡排序作为例子，通过F12设置断点的方式，可以看到参数是函数的一个调用过程。之前的冒泡排序只能将数组从大到小，或者从小到大排序，现在用户能自己决定比较方式。

|  |
| --- |
| function sortOfMaoPao(array, compare) {  var temp;  var isSort = true;  //第一层for是控制趟数是  for (var i = 0; i < array.length - 1; i++) {  //第二层for是进行排序  for (var j = 0; j < array.length - 1 - i; j++) {    **/\*if (array[j] > array[j+1]) { 这句话是决定了排序,这里写死了\*/**  **if (compare(array[j], array[j+1]) > 0) {**  temp = array[j+1];  array[j+1] = array[j];  array[j] = temp;  isSort = false;  }  }  if (isSort) {  break;  }  }  }  var array = [111,231,55,66,77,900];  // 把匿名函数赋值给一个变量就可以  sortOfMaoPao(array, **function (a,b) {**  **return b - a;**  **}**);  console.log(array);  1 |

（7）清空数组

|  |
| --- |
| // 方式1 推荐  arr = [];  // 方式2  arr.length = 0;  // 方式3 第一个参数，是从什么位置开始删除，写索引  // 第二个参数是，删除多少个元素  arr.splice(0, arr.length); |

6、案例

（1）将一个字符串数组输出为|分割的形式，比如“刘备|张飞|关羽”。使用两种方式实现

|  |
| --- |
| var array = [1,2,3,5];  var newarray = array.join('|');  console.log(array);  console.log(array.length);  console.log(newarray);  console.log(newarray.length);  1 |

（2）工资的数组[1500, 1200, 2000, 2100, 1800]，把工资超过2000的删除。

|  |
| --- |
| var array = [1500, 1200, 2000, 2100, 1800];  var newarray = array.filter(**function (item) {**  **// item就是数组中的每一个元素**  **return item < 2000;**  **}**);  console.log(newarray);  1 |

（3）["c", "a", "z", "a", "x", "a"]找到数组中每一个a出现的位置

|  |
| --- |
| var array = ["c", "a", "z", "a", "x", "a"];  var index = -1;  var indexArray = [];  do {  index = array.indexOf('a', index+1);  if (index !== -1)  indexArray.push(index);  } while (index !== -1);  console.log(indexArray); |

（4）编写一个方法去掉一个数组的重复元素

|  |
| --- |
| var num = [1,2,3,55,66,7,56,1,3,2];  var array = ['c', 'a', 'z', 'a', 'x', 'a'];  var newArray = new Array();  function clearRepeat(array) {  var tempArray = [];  var index;  for (var i = 0; i < array.length; i++) {  index = tempArray.indexOf(array[i]);  if(index === -1) {  tempArray.push(array[i]);  /\*tempArray[tempArray.length] = array[i];\*/  }  }  return tempArray;  }  console.log(array);  newArray = clearRepeat(array);  console.log(newArray); |

二十七、基本包装类型（String/Number/Boolean）

为了方便操作简单数据类型，JavaScript还提供了三个特殊的简单类型类型：String/Number/Boolean，都是构造函数。通过 new 之后它们的类型是 object。

|  |
| --- |
| // s1是基本类型，基本类型是没有方法的，对象才有属性和方法  var s1 = 'zhangsan';  var s2 = s1.substring(5);//输出san 把基本数据类型包装成复杂数据类型。  var str = new String('hi');  console.log(typeof str); // 输出 object是对象  console.log(str);  1  // 当调用s1.substring(5)的时候，先把s1包装成String类型的临时对象，再调用substring方法，最后销毁临时对象, 相当于：  var s1 = new String('zhangsan');  var s2 = s1.substring(5);  s1 = null; //销毁对象 null是内存中一块无效的区域。 s1就指向了null区域，刚刚new String的空间没有人引用了，等等浏览器在某个时间段去清空。给释放出来  将来我们一般情况下不会使用Number、Boolean基本数据类型包装。它会带来歧义。 |

|  |
| --- |
| var num = new Number('12');  console.log(num);  console.log(typeof num);  // 类型转换  var num1 = Number('11');  console.log(num1);  1  var bool = new Boolean(false);  var result = bool && true;  console.log(result); // 输出true。  1 |

1、String对象

（1）字符串的不可变

|  |
| --- |
| 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，不确定浏览器什么时候才会将他回收。  // 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，会不断开辟新的空间，运行的性能会慢。 |

（2）字符串对象的常用方法

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，跟数组是有差别的)，操作完成会返回一个新的字符串

使用以下方法的时候，可以不用通过new String的方式，能直接使用字面量的字符串，系统会自动帮我们创建一个临时基本包装对象。

他**可以像数组**一样来进行访问。

|  |
| --- |
| 1 字符方法  charAt() //获取指定位置处字符  charCodeAt() //获取指定位置处字符的ASCII码  str[0] //**HTML5**，IE8+支持 和charAt()等效  2 字符串操作方法  concat() //拼接字符串，等效于+，+更常用  slice() //从start位置开始，截取到end位置，end取不到  substring() //从start位置开始，截取到end位置， end取不到  substr() //从start位置开始，截取length个字符，如果忽略 length，则 substr 提取字符，直到字符串末尾。  3 位置方法  indexOf() //返回指定内容在元字符串中的位置  lastIndexOf() //从后往前找，只找第一个匹配的  4 去除空白  trim() //只能去除字符串**前后**的空白，中间的没有办法。  返回值：一个代表调用字符串两端去掉空白的新字符串。  5 大小写转换方法  to(Locale)UpperCase() //转换大写，极少数的国家的大小写可能不一致。一般不用加local  to(Locale)LowerCase() //转换小写  6 其它  search() //查找指定的子字符串，支持正则表达式，比indexOf不支持，功能强大  replace() // 只能替换第一个找到的循环，如果要全部，需要进行多次循环处理。  split() // 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  var str = 'a!!!coefox!!!yozzopp';  console.log(str);  console.log(str.length);  var array = str.split('!');  console.log(array);  1  **此时,应该用正则表达式来进行切割,还没有学到** |

2、案例

（1）截取字符串"我爱中华人民共和国"，中的"中华"。

|  |
| --- |
| var str = '我爱中华人民共和国';  var newArray = str.substr(2,2);  console.log(newArray);// 输出 中华 |

1. "abcoefoxyozzopp"查找字符串中所有o出现的位置

|  |
| --- |
| var str = 'abcoefoxyozzopp';  var index = -1;  do {  index = str.indexOf('o', index+1);  if (index !== -1) {  console.log(index);  }  } while(index !== -1);  //输出 3 6 9 12 |

1. 把字符串中所有的o替换成!

|  |
| --- |
| var str = 'abcoefoxyozzopp';  var index = -1;  do {  index = str.indexOf('o', index+1);  if (index !== -1) {  **str = str.replace('o','!');**  }  } while(index !== -1);  console.log(str);  //输出 abc!ef!xy!zz!pp |

（4）把字符串中的所有空白去掉' abc xyz a 123 '

方法1 可以把字符串中所有的空格字符，用replace替换成’’空字符串

方法2 使用split简化

|  |
| --- |
| 方法1，无法找到所有的空格，是替换成空字符引起，换成其他的字符串则能准确替换，暂时未解？？？  var str = ' abc xyz a 123 ';  function clearBlank(str) {  var index = -1;  do {  index = str.indexOf(' ',index+1);  console.log(index);  if (index !== -1) {  str = str.replace(' ','');  }  } while(index !== -1);  return str;// 返回的是字符串值  }  var temp = clearBlank(str);  console.log(temp); |
| 方法2  var str = ' abc xyz a 123 ';  function clearBlank(str) {  str = str.split(' ');//返回一个分割后的数组  if(str[0] == '') {  console.log('赋值为空\'\'');  }  str = str.join('');  return str;// 返回的是字符串值  }  var temp = clearBlank(str);  console.log(temp);  1 |

（5）判断一个字符串中出现次数最多的字符，统计这个次数

|  |
| --- |
| var str = 'abcoefoxyozzopp';  function getStrMost(str) {  var storage = {};  var item;  var ch;  var max = 0;//对象的属性是无序的  for (var i = 0; i < str.length; i++) {  item = str.charAt(i);  if (storage[item]) {//不能写成 storage[item] !== '',这句话一直是true  storage[item] ++;  } else {  storage[item] = 1;  }  }  console.log(storage);  for (var key in storage) {  if (max < storage[key]) {  max = storage[key];  ch = key;  }  }  console.log('出现次数最多的字符：' + ch + ' 次数：' + max);  }  getStrMost(str);  1 |

（6）获取url中?后面的内容，并转化成对象的形式。例如：http://www.itheima.com/login?name=zs&age=18&a=1&b=2

|  |
| --- |
| var str = 'http://www.itheima.com/login?name=zs&age=18&a=1&b=2';  function getUrlInfo(str) {  var index;  var arrayStr = [];  var temp;  var storage = {};  **index = str.indexOf('?') + 1;**  **str = str.substr(index);**  //上面的方法也可以通过用 ？ 进行分割为两字符串，然后取后边的字符串值就ok  /\***str = (str.split('?'))[1];**  **console.log(str);**\*/  arrayStr = str.split('&');// 此时数组里面的值是字符串  for (var i = 0; i < arrayStr.length; i++) {  temp = arrayStr[i].split('=');  storage[temp[0]] = temp[1];  }  return storage;  }  var o = getUrlInfo(str);  console.log(o);  1 |

1. 练习

给定一个字符串如：“abaasdffggghhjjkkgfddsssss3444343”问题如下：

1、 字符串的长度

2、 取出指定位置的字符，如：0,3,5,9等

3、 查找指定字符是否在以上字符串中存在，如：i，c ，b等

4、 替换指定的字符，如：g替换为22,ss替换为b等操作方法

5、 截取指定开始位置到结束位置的字符串，如：取得1-5的字符串

6、 找出以上字符串中出现次数最多的字符和出现的次数

7、 遍历字符串，并将遍历出的字符两头添加符号“@”输出至当前的文档页面。

|  |
| --- |
| var str = 'abaasdffggghhjjkkgfddsssss3444343';  function getNumMost(str) {  var item;  var o = {};//尽量要给变量进行初始化，否则可能识别不了。  var max = 0;  var ch;  for (var i = 0; i < str.length; i++) {  item = str.charAt(i);  if (o[item]) {//如果o没有初始化为一个空字符对象，这里会无法判断。  o[item]++;  } else {  o[item] = 1;  }  }  for (var key in o) {  if (max < o[key]) {  max = o[key];  ch = key;  }  }  console.log('出现次数最多的字符：'+ch+'次数为：'+max);  }  getNumMost(str);  console.log('@'+str+'@');  1 |

二十八、新增知识点

1、七种简单数据类型是无法像对象一样有属性的，通过String/Number/Boolean能创建出对应的对象。创建后打印出来的是对象，不是简单数据类型那样了

|  |
| --- |
| <script>  var str = 'str';  var num = 123;  str.name = 'hello';  num.name = 100;  console.log(str.name);// 输出undefined  console.log(num.name);// 输出undefined  // 通过构造函数创建对象才可以  var str = new String('str');  str.name = 'hi';  console.log(str.name);// 输出hi  </script> |

2、Js加法运算符（有字符串，转成字符串，没有全部转数字）

JavaScript允许使用对象，数组，null或undefined作为操作数。

注意：对数字和NaN进行相加求值等于NaN.

（1）转换规则

如果至少有一个操作数是对象，会被转换成原始值（字符串，数字或布尔）；

转换之后，如果至少有一个操作数是字符串类型，第二个操作数会被转换成字符串，并且会执行连接。

在其他的情况下，两个操作数都会**转换成数字**并执行算数加法运算。

如果两个操作数都是原始类型，运算符会检查是否至少有一个是字符串类型，如果是就执行连接操作。其他情况就都转换为数字并求合。

（2）对象类型转为原始类型

toString() ：返回对象的字符串表示。

valueOf() ：返回对象的字符串、数值或布尔值表示。通常与 toString() 方法的返回值 相同。

|  |
| --- |
| var str = 'str';  console.log(str.toString());  console.log(str.valueOf());    var num = 123;  console.log(num.toString());  console.log(num.valueOf());    var bool = true;  console.log(bool.toString());  console.log(bool.valueOf());    var obj = {  name: 'hello'  };  console.log(obj.toString());  console.log(obj.valueOf());    var arr = [1,2,3,4,'hi'];  console.log(arr.toString());  console.log(arr.valueOf()); |

当数组转换为原始类型，JavaScript会使用它的**join(',')**方法，例如[1,5,6]就是"1,5,6"。普通JavaScript对像{}的原始类型是"[object Object]"。

3、在for循环表达式中使用let声明表达式变量

因为let有自己的作用域块，所以在for循环表达式中使用let其实就等价于在代码块中使用let，也就是说

**（1）for( let i = 0; i< 5; i++) 这句话的圆括号之间，有一个隐藏的作用域**

**（2）for( let i = 0; i< 5; i++) { 循环体 } 在每次执行循环体之前，JS 引擎会把 i 在循环体的上下文中重新声明及初始化一次。**

|  |
| --- |
| var liList = document.querySelectorAll('li') // 共5个li  for(let i = 0;i<liList.length;i++){  liList[i].onclick = function(){  //这里的i是循环结束的i，因为监听事件是异步  console.log(i)//0 1 2 3 4  }  }  //上面代码其实就等价于  for (let i = 0;i<liList.length;i++){  let i = 隐藏作用域中的i（表达式中的i）  liList[i].onclick = function(){  console.log(i)  }  }  //你也可以在代码块中使用另一个let变量接受每一个i  for(let i = 0;i<liList.length;i++){  //这里能很直观的看出j有自己的作用域  let j = i;  liList[j].onclick = function(){  console.log(j)  }  }  //你也可以当成是声明了五个let块级作用域  {let i = 0}  {let i = 1}  {let i = 2}  {let i = 3}  {let i = 4} |

**因为let在代码块中都有自己的作用域，所以在for循环中的表达式中使用let它的每一个值都会单独存在一个独立的作用域中不会被覆盖掉。**