# 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)

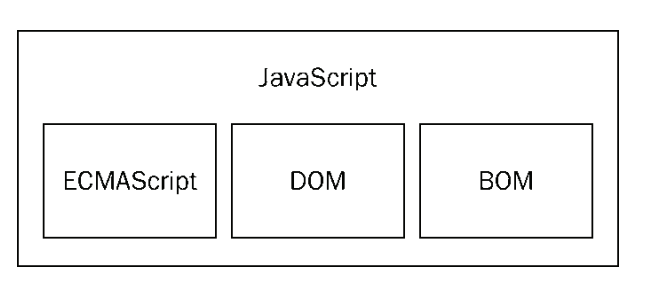
四、JavaScript和HTML、CSS的区别

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。

六、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。 |

七、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



八、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 |

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

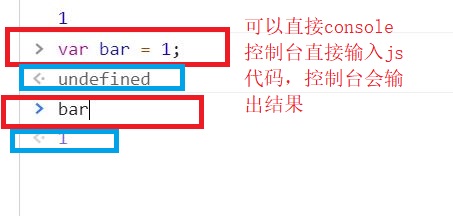
（2）规范 - **建议**遵守的，不遵守不会报错

- 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

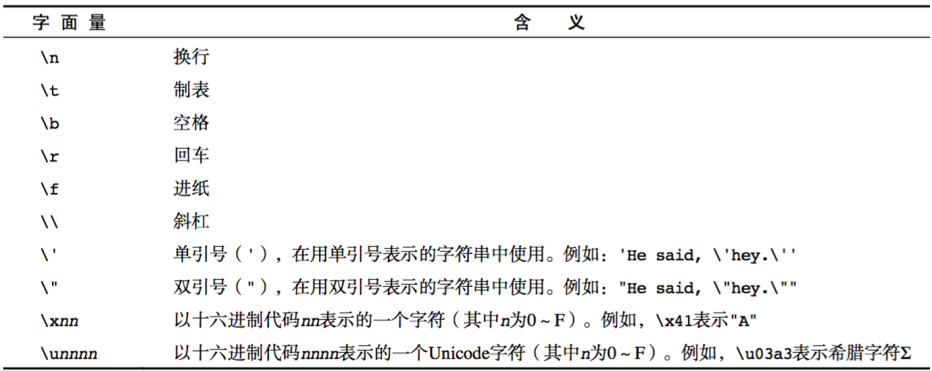
（1）字符串字面量：'程序猿'，'程序媛', "黑马程序猿"

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。

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'  type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’)//NaNsss |

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。

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 |

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

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。

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后边的，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。