

声明JavaScript变量

在计算机科学中, data(数据)大于一切，因为它对于计算机的意义重大。JavaScript提供七种不同的data types(数据类型)，它们是undefined（未定义）, null（空）, boolean（布尔型）, string（字符串）, symbol(符号), number（数字）, and object（对象）。

举个例子, 计算机能够分辨不同的数字, 例如数字 12，但是"12"和"dog", 或"123 cats", 却是字母的集合。 计算机能够精确地运算数字， 但是无法处理字符串中的数字。

Variables（变量）允许计算机以动态的形式来存储和操作数据，通过操作指向数据的指针而不是数据本身来避免了内存泄露，以上的七种数据类型都可以存储到一个Variables（变量）中。

Variables（变量 ）类似于您在数学中使用的x和y变量，这意味着它们通过一个简单的名称来表示我们要引用的数据。 计算机中的Variables（变 量）与数学变量不同，因为它们可以在不同时间存储不同的值。

通过在变量的前面使用关键字var，我们告诉 JavaScript 来创建或者 declare（声明）一个变量,就像这样：

var ourName;

上面代码的意思是创建一个名为ourName的variable（变量），在JavaScript中我们使用;（分号）来结束一段声明。

Variable （变量）的名字可以由数字、字母、$ 或者 \_组成，但是不能包含空格或者以数字为首。

JavaScript 使用=为变量赋值

在JavaScript中，您可以使用=赋值运算符将值存储在变量中。

myVariable = 5;

把Number数字5赋给变量myVariable。

赋值过程是从右向左进行的。所有 = 赋值运算符右边的值都会被赋到左边的变量中。

myVar = 5;   
myNum = myVar;

数值 5 被赋给变量 myVar 中， 然后变量 myVar 又赋给变量 myNum ，这样子 myNum 变量中的值也是 5 了。

理解JavaScript变量的大小写敏感性

在 JavaScript 中所有的变量都是大小写敏感的。这意味着你要区分大写字母和小写字母。

MYVAR与MyVar和myvar 是截然不同的变量。这就有可能导致多个不同的变量却有着有相似的名字。正是由于以上原因所以强烈地建议你, 不要使用这一特性。（尽量避免设置名字相近难以区分的变量名）

JavaScript加法运算

Number 是JavaScript中的数据类型，表示数字数据。

现在让我们尝试在JavaScript中做加法运算。

JavaScript 中使用 + 运算符来让两个数字执行加法运算。

+/-/\*// 加减乘除

i=i+1 等同于 i++ (--)

JavaScript浮点数

我们可以把小数存储到变量中。小数也被称作 浮点数 。

**提示**  
不是所有的实数都可以用 浮点数 来表示。因为可能存在四舍五入的错误 (保留三位)

JavaScript取余运算%

使用%运算符来对两个数字进行**求余**运算。

JavaScript -=赋值操作

与 += 操作符类似，-= 操作符用来对一个变量进行减法赋值操作。

myVar = myVar - 5; 也可以写成这种形式：myVar -= 5;（\*/+/-//都一样）

# JavaScript 转义字符串中的引号

当你定义一个字符串时，你必须以单引号或双引号作为开头和结尾。那么当你需要在字符串中使用一个: " 或者 ' 时该怎么办呢?

在 JavaScript 中，你可以通过在引号前面使用 反斜杠 (\) 来转义引号。

var sampleStr = "Alan said, \"Peter is learning JavaScript\".";

这标志着提示 JavaScript 该单引号或双引号并不是字符串的结尾，而是出现在字符串内的字符。所以，如果你要打印字符串到控制台，你将得到：

Alan said, "Peter is learning JavaScript".

# JavaScript 使用单引号引用字符串

在 JavaScript 中的 字符串 要用单引号或双引号来包裹它，只要你在开始和结束都使用相同类型的引号，单引号和双引号的功能在JavaScript中是相同的。

"This string has \"double quotes\" in it"

当我们需要在字符串中使用与开头结尾相同的引号时，我们需要对引号进行 转义 。如果你有很多双引号的字符串，使用转义字符可能导致难以阅读。这时候可以使用单引号。

'This string has "double quotes" in it. And "probably" lots of them.'

# JavaScript 转义字符串中的特殊符号

字符串中的转义序列

引号不是字符串中唯一的可以被转义字符。下面是常见的转义序列列表:

| **Code** | **Output** |
| --- | --- |
| \' | 单引号 |
| \" | 双引号 |
| \\ | 反斜杠符 |
| \n | 换行符 |
| \r | 回车符 |
| \t | 制表符 |
| \b | 退格符 |
| \f | 换页符 |

注意，如果你想要显示一个反斜杠就必须要转义它。

# JavaScript字符串连接方式

在 JavaScript 中，当 + 运算符与 字符串 一起使用的时候，它被称作 连接 运算符。你可以通过和其他字符串连接 来创建一个新的字符串。

**举个栗子**

'My name is Alan,' + ' I concatenate.'

**注意**  
注意空格。连接操作不会添加两个字符串之外的空格，所以想加上空格的话，你需要自己在字符串里面添加。

# JavaScript 获取字符串的长度

你可以通过在字符串变量或字符串后面写上 .length 来获得变量中string （字符串）值的长度。

"Alan Peter".length; // 10

例如，我们创建了一个变量 var firstName = "Charles"，我们就可以通过使用 firstName.length 来获得 "Charles" 字符串的长度

# JavaScript 使用中括号获取字符串中的第一个字符

括号符号是一种在字符串中的特定索引处获取字符的方法。

() ：小括号。

[] ：中括号

{}：大括号

JavaScript中只有字符串类型，没有字符类型。

如果要获取字符串中的某个字符，我们需要通过[索引] 来获得对应的字符。

大多数现代编程语言，如JavaScript，不同于人类从1开始计数。它们是从0开始计数，这被称为 基于零 的索引。

例如, 在单词 "Charles" 中索引0上的字符为 "C"，所以在 var firstName = "Charles" 中，你可以使用 firstName[0] 来获得第一个位置上的字符

# JavaScript 使用中括号索引查找字符串中的第N个字符

你也可以使用 [索引]来获得一个字符串中的某个位置的字符。

请记住，计算机是从 0 开始计数，所以获取第一个字符的索引实际上是[0]。

# JavaScript 使用中括号索引查找字符串中的最后一个字符

为了得到一个字符串的最后一个字符，你可以获取该字符串的长度并减一来作为最后一个字符的索引。

例如，在 var firstName = "Charles" 中，你可以这样操作 firstName[firstName.length - 1] 来得到字符串的最后的一个字符。

Javascript 数组操作

通过 数组，我们可以将多个数据存储于同一个变量中。

通过左括号[开始定义一个数组，以右括号]结束定义，并把每个数据条目之间用逗号隔开，就像这样：  
var sandwich = ["peanut butter", "jelly", "bread"]。

JavaScript 使用索引修改数组中的数据

与字符串的数据不可变不同，数组的数据是可变的，并且可以自由地改变。

**例如**

var ourArray = [3,2,1];  
ourArray[0] = 1; // ourArray等于 [1,2,1]

JavaScript 使用索引操作多维数组

可以把 多维 数组理解成是一个 **数组中的数组**。当使用[]去访问数组的时候，第一个[index]访问的是该数组中的第N个子数组，第二个[index]访问的是第N个子数组的第N个元素。

**例如**

var arr = [  
[1,2,3],  
[4,5,6],  
[7,8,9],  
[[10,11,12], 13, 14]  
];  
arr[0]; // 等于 [1,2,3]  
arr[1][2]; // 等于 6  
arr[3][0][1]; // 等于 11

# JavaScript push()函数追加数组数据

将数据追加到一个数组末尾的最简单的方法是通过 push() 函数。

.push() 允许有一个或多个参数，并把它“push”到数组的末尾。

var arr = [1,2,3];  
arr.push(4);  
// 现在arr的值为 [1,2,3,4]

JavaScript shift()函数移出数组第一个数据

pop() 函数用来移出数组中最后一个元素。如果想要移出第一个元素要怎么办呢？

.shift() 就是专门用来处理这类型需求的。它的工作原理类似 .pop()，但它移除的是第一个元素，而不是最后一个。

JavaScript unshift()函数移入数据到数组第一位

你不仅可以 shift（移出）数组中的第一个元素，你也可以 unshift（移入）一个元素到数组的头部。

.unshift() 函数用起来就像 .push() 函数一样, 但不是在数组的末尾添加元素，而是在数组的头部添加元素。

JavaScript 函数定义

在 JavaScript 中，我们可以把代码的重复部分抽取出来，放到一个*函数*（functions）中。

这是一个函数（function）的例子：

function functionName() {  
console.log("Hello World");  
}

你可以通过函数名称functionName加上后面的小括号来调用这个函数（function），就像这样：

functionName();

每次调用函数时它会打印出消息的“Hello World”到开发的控制台上。所有的大括号之间的代码将在每次函数调用时执行。

JavaScript 函数全局变量定义

在 JavaScript 中， 作用域 影响着变量的作用范围。在函数外定义的变量具有 全局 作用域。这意味着，具有全局作用域的变量可以在代码的任何地方被调用。

没有使用var关键字定义的变量，会被自动创建在全局作用域中，变成全局变量。当在代码其他地方无意间定义了一个变量，刚好变量名与全局变量相同，这时候会产生意想不到的后果，可能导致你的代码出现BUG。因此你应该总是使用var关键字来声明你的变量。

任务

在函数外声明一个 全局 变量 myGlobal，并给它一个初始值 10

在函数 fun1 的内部，**不**使用 var 关键字来声明 oopsGlobal，并赋值为 5。

# JavaScript 函数全局变量与局部变量差异

一个程序中有可能具有相同名称的 局部 变量 和 全局 变量。在这种情况下，局部 变量将会优先于 全局 变量。

下面为例：

var someVar = "Hat";  
function myFun() {  
var someVar = "Head";  
return someVar;  
}

函数 myFun 将会返回 "Head"，因为 局部变量 优先级更高。

JavaScript 函数使用return返回值

我们可以把数据通过函数的 *参数* 来传入函数，也可以使用 return 语句把数据从一个函数中传出来。

**举个栗子**

function plusThree(num) {  
return num + 3;  
}  
var answer = plusThree(5); // 8

plusThree 带有一个为 num 的 *参数* 并且返回（returns）一个等于 num + 3 的值。

# JavaScript 队列

在计算机科学中 队列（queue）是一个抽象的数据结构，队列中的数据条目都是有秩序的。新的条目会被加到 队列 的末尾，旧的条目会从 队列 的头部被移出。

写一个函数 queue ，用一个数组arr和一个数字item作为参数。数字item添加到数组的结尾，然后移出数组的第一个元素，最后队列函数应该返回被删除的元素。

JavaScript 布尔boolean数据类型

布尔（Boolean）是另一种数据类型

。布尔 值要么是true 要么是false。它非常像电路开关， true 是“开”，false 是“关”。这两种状态是互斥的。

**注意**  
Boolean 值绝不会写作被引号包裹起来的形式。字符串 的 "true" 和 "false" 不是 布尔值，在 JavaScript 中也没有特殊含义。（布尔值不是字符串）

JavaScript if语句

If 语句用于在代码中做条件判断。关键字 if 告诉 JavaScript 在小括号中的条件为真的情况下去执行定义在大括号里面的代码。这种条件被称为 Boolean 条件，因为他们只可能是 true（真）或 false（假）。

当条件的计算结果为 true，程序执行大括号内的语句。当布尔条件的计算结果为 false，大括号内的代码将不会执行。

**伪代码**

if(*条件为真*){  
*语句被执行*  
}

**示例**

function test (myCondition) {  
if (myCondition) {  
 return "It was true";  
}  
return "It was false";  
}  
test(true);// returns "It was true"  
test(false); // returns "It was false"

当 test 被调用，并且传递进来的参数值为 true，if 语句会计算 myCondition 的结果，看它是真还是假。如果条件为 true，函数会返回 "It was true"。当 test 被调用，并且传递进来的参数值为 false，myCondition *不* 为 true，并且不执行大括号后面的语句，函数返回 "It was false"。

在 JavaScript 中，有很多 *相互比较的操作*。所有这些操作符都返回一个 true 或 false 值。

最基本的运算符是相等运算符：==。相等运算符比较两个值，如果它们是同等，返回 true，如果它们不等，返回 false。b必须注意相等运算符不同于赋值运算符（=），赋值运算符是把等号右边的值赋给左边的变量。

function equalityTest(myVal) {  
if (myVal == 10) {  
 return "Equal";  
}  
 return "Not Equal";  
}

如果 myVal 等于 10，相等运算符会返回 true，因此大括号里面的代码会被执行，函数将返回 "Equal"。否则，函数返回 "Not Equal"。

在 JavaScript 中，为了让两个不同的 数据类型（例如 数字 和 字符串）的值可以作比较，它必须把一种类型转换为另一种类型。然而一旦这样做，它可以像下面这样来比较：

1 ==1// true  
 1 ==2// false  
 1 == '1' // true  
"3"==3// true

JavaScript 全等于比较运算符===

全等（===）是相对于相等操作符（==）的一种操作符。与相等操作符不同的是全等比较严格，它会同时比较元素的值和 数据类型。

**举个例子**

3 === 3 // true  
3 === '3' // false

3 是一个 数字 类型的，而'3' 是一个 字符 类型的，所以3不全等于'3'。

JavaScript 不等于比较运算符!=

不相等运算符（!=）与相等运算符是相反的。在不相等运算符中，如果条件结果在相等运算符中为“false”则会被判断为返回 true。与相等运算符类似，不相等运算符在比较的时候也会转换值的数据类型。

!== 不同类型不比较，且无结果，同类型才比较。

**例如**

1 != 2// true   
1 != "1"// false   
1 != '1'// false   
1 != true // false   
0 != false// false

JavaScript 不全等于比较运算符!==

不全等运算符（!==）与全等运算符是相反的。这意味着“不全等”并返回 false 的地方，用全等运算会返回 true，反之亦然。全等运算符不会转换值的数据类型。

**例如**

3 !== 3 // false  
3 !== '3' // true  
4 !== 3 // true

# JavaScript 大于比较运算符> 大于等于运算符

使用大于运算符（>）来比较两个数字。如果大于运算符左边的数字大于右边的数字，将会返回 true。否则，它返回 false。

与相等运算符一样，大于运算符在比较的时候，会转换值的数据类型。

# JavaScript 使用逻辑与运算符&&比较数值

有时你需要在一次判断中做多个操作。运算符的左边和右边结果必须都是 true，逻辑与 运算符（&&）才会返回 true。

同样的效果可以通过if语句的嵌套来实现：

if (num > 5) {  
if (num < 10) {  
return "Yes";  
}  
}  
return "No";

只有当 num 的值在6和9之间（包括6和9）才会返回 "Yes"。相同的逻辑可被写为：

if (num > 5 && num < 10) {  
return "Yes";  
}  
return "No";

#### 举个栗子

5 > 3 // true  
 7 > '3' // true  
 2 > 3 // false  
'1' > 9// false

#### 举个栗子

6>=6// true  
 7>= '3' // true  
 2>=3// false  
'7' >=9// false

JavaScript 使用逻辑或运算符||比较数值

逻辑或运算符左右两边任何一个操作数是true，逻辑或 运算符 (||) 返回 true。反之，返回 false。

举个栗子

if (num > 10) {  
return "No";  
}  
if (num < 5) {  
return "No";  
}  
return "Yes";

只有当num大于等于5或小于等于10时，函数返回"Yes"。相同的逻辑可以简写成：

if (num > 10 || num < 5) {  
return "No";  
}  
return "Yes";

JavaScript else语句

当if语句的条件为真，大括号里的代码执行，那如果条件为假呢？

这时候需要写一个else语句，当条件为假时执行相应的代码。

if (num > 10) {  
return "Bigger than 10";  
} else {  
return "10 or Less";  
}

JavaScript else if语句

如果你有多个条件语句，你可以通过else if语句把 if语句链起来。

if (num > 15) {  
return "Bigger than 15";  
} else if (num < 5) {  
return "Smaller than 5";  
} else {  
return "Between 5 and 15";  
}

Javascript 逻辑运算综合实战

在高尔夫golf游戏中，每个洞都有自己的标准杆数par，代表着距离。根据你把球打进洞所挥杆的次数strokes，可以计算出你的高尔夫水平。

函数将会传送2个参数，分别是标准杆数par 和 挥杆次数strokes ，根据下面的表格返回正确的水平段位。

| **Strokes** | **Return** |
| --- | --- |
| 1 | "Hole-in-one!" |
| <= par - 2 | "Eagle" |
| par - 1 | "Birdie" |
| par | "Par" |
| par + 1 | "Bogey" |
| par + 2 | "Double Bogey" |
| >= par + 3 | "Go Home!" |

par 和 strokes 必须是数字而且是正数。

JavaScript 使用switch语句进行多选项选择

如果你有非常多的选项需要选择，可以使用switch语句。根据不同的参数值会匹配上不同的case分支，语句会从第一个匹配的case分支开始执行，直到碰到break就结束。

这是一个伪代码案例：myTest(1) 应该有一个值为 "alpha" myTest(2) 应该有一个值为 "beta" myTest(3) 应该有一个值为 "gamma" myTest(4) 应该有一个值为 "delta"

function myTest(val) {

var answer = "";

switch (val) {

case 1:

answer = "alpha"

break;

case 2:

answer = "beta";

break;

case 3:

answer = "gamma";

break;

case 4:

answer = "delta";

break;

}

return answer;

}

myTest(1);测试case 值使用严格等于，break关键字告诉javascript停止执行语句。如果没有break关键字，下一个语句会继续执行。

JavaScript 在switch语句中添加default语句

在switch 语句中你可能无法用case来指定所有情况，这时你可以添加default语句。再也找不到case匹配的时候default语句会执行，非常类似于if/else组合中的else语句。

default语句应该是最后一个case。

switch (num) {  
case value1:  
statement1;  
break;  
case value2:  
statement2;  
break;  
...  
default:  
defaultStatement;  
}

# JavaScript switch语句中的多个相同选项判断

如果switch语句中的case分支的break 语句漏掉了，后面的 case语句会一直执行直到遇到break。如果你有多个输入值和输出值一样，可以试试下面的switch语句：

switch(val) {  
case 1:  
case 2:  
case 3:  
result = "1, 2, or 3";  
break;  
case 4:  
result = "4 alone";  
}

分支1、2、3将会产生相同的输出结果。

JavaScript 直接在函数中返回boolean值

所有的比较操作符返回的都是一个boolean值，要么是 true 要么是false 。

使用 if/else语句来做比较然后返回true或false已经成为大家的共识:

function isEqual(a,b) {  
if (a === b) {  
return true;  
} else {  
return false;  
}  
}

因为=== 总是返回 true 或 false，所以我们可以直接返回比较的结果：

function isEqual(a,b) {  
return a === b;  
}

JavaScript 在函数中使用return跳出函数

当代码执行到return语句时，函数返回一个结果就结束运行了，return后面的语句根本不会执行。

举个栗子

function myFun() {  
console.log("Hello");  
return "World";  
console.log("byebye")  
}  
myFun();

上面的代码输出"Hello"到控制台、返回 "World"，但没有输出"byebye"，因为函数遇到return语句就退出了。

JavaScript 对象操作

你之前可能听说过对象 object 。

对象和数组很相似，数组是通过索引来访问和修改数据，对象是通过属性来访问和修改数据的。

这是一个示例对象：

var cat = {  
"name": "Whiskers",  
"legs": 4,  
"tails": 1,  
"enemies": ["Water", "Dogs"]  
};

对象适合用来存储结构化数据，就和真实世界的对象一模一样，比如一只猫。

JavaScript 使用点操作符.读取对象属性

有两种方式访问对象属性，一个是点操作符(.)，一个是中括号操作符([])。

当你知道属性的名称的时候，使用点操作符。

这是一个使用点操作符读取对象属性的例子：

var myObj = {  
prop1: "val1",  
prop2: "val2"  
};  
var prop1val = myObj.prop1; // val1  
var prop2val = myObj.prop2; // val2

JavaScript 使用[]读取对象属性

第二种访问对象的方式就是中括号操作符([])，如果你想访问的属性的名称有一个空格，这时你只能使用中括号操作符([])。

这是一个使用中括号操作符([])读取对象属性的例子：

var myObj = {  
"Space Name": "Kirk",  
"More Space": "Spock"  
};  
myObj["Space Name"]; // Kirk  
myObj['More Space']; // Spock

提示：属性名称中如果有空格，必须把属性名称用单引号或双引号包裹起来。

JavaScript 使用变量访问对象属性

中括号操作符的另一个使用方式是用变量来访问一个属性。当你需要遍历对象的属性列表或查表时，这种方式极为有用。

这有一个使用变量来访问属性的例子：

var someProp = "propName";  
var myObj = {  
propName: "Some Value"  
}  
myObj[someProp]; // "Some Value"

还有更多：

var myDog = "Hunter";  
var dogs = {  
Fido: "Mutt",

Hunter: "Doberman",

Snoopie: "Beagle"  
}  
var breed = dogs[myDog]; // "Hunter"  
console.log(breed)// "Doberman"

提示：当我们通过变量名访问属性的时候，不需要给变量名包裹引号。因为实际上我们使用的是变量的值，而不是变量的名称。

JavaScript 更新对象属性

当你创建了一个对象后，你可以用点操作符或中括号操作符来更新对象的属性。

举个例子，让我们看看 ourDog:

var ourDog = {  
"name": "Camper",  
"legs": 4,  
"tails": 1,  
"friends": ["everything!"]  
};

让我们更改它的名称为 "Happy Camper"，这有两种方式来更新对象的name属性：

ourDog.name = "Happy Camper";

ourDog["name"] = "Happy Camper";

# JavaScript 给对象添加属性

你也可以像更改属性一样给对象添加属性。

看看我们是如何给ourDog添加 "bark"属性：

ourDog.bark = "bow-wow";

或者

ourDog["bark"] = "bow-wow";

# JavaScript 删除对象属性

我们同样可以删除对象的属性，例如：

delete ourDog.bark;

JavaScript 使用对象进行查找值

对象和字典一样，可以用来存储键/值对。如果你的数据跟对象一样，你可以用对象来查找你想要的值，而不是使用switch或if/else语句。当你知道你的输入数据在某个范围时，这种查找方式极为有效。

这是简单的反向字母表：

var alpha = {  
1:"Z",  
2:"Y",  
3:"X",  
4:"W",  
...  
24:"C",  
25:"B",  
26:"A"  
};  
alpha[2]; // "Y"  
alpha[24]; // "C"  
  
var value = 2;  
alpha[value]; // "Y"

JavaScript 检查对象属性

有时检查一个对象属性是否存在是非常有用的，我们可以用.hasOwnProperty(propname)方法来检查对象是否有该属性。如果有返回true，反之返回 false。

**举例**

var myObj = {  
top: "hat",  
bottom: "pants"  
};  
myObj.hasOwnProperty("top");// true  
myObj.hasOwnProperty("middle"); // false

注意：如果你需要通过变量来访问对象的属性0值，请用中括号【】操作符，点操作符.不支持变量。

# JavaScript JSON操作

JavaScript Object Notation 简称 JSON，它使用JavaScript对象的格式来存储数据。JSON是灵活的，因为它允许 数据结构 是 字符串，数字，布尔值，字符串，和 对象的任意组合。

这里是一个JSON对象的示例：

var ourMusic = [  
{  
"artist": "Daft Punk",  
"title": "Homework",  
"release\_year": 1997,  
"formats": [   
"CD",   
"Cassette",   
"LP" ],  
"gold": true  
}  
];

这是一个对象数组，并且对象有各种关于专辑的 详细信息。它也有一个嵌套的 formarts 的数组。附加专辑记录可以被添加到数组的最上层。

**提示**  
数组中有多个 JSON

对象的时候，对象与对象之间要用逗号隔开。

JavaScript 获取JSON属性值

通过串联起来的点操作符或中括号操作符来访问JSON对象的嵌套属性。

下面是一个嵌套的JSON对象：

var ourStorage = {  
"desk": {  
"drawer": "stapler"  
},  
"cabinet": {  
"top drawer": {   
"folder1": "a file",  
"folder2": "secrets"  
},  
"bottom drawer": "soda"  
}  
}  
ourStorage.cabinet["top drawer"].folder2;// "secrets"  
ourStorage.desk.drawer; // "stapler"

JavaScript 获取JSON数组值

正如我们在前面的例子所见，JSON对象可以嵌套对象和数组。与访问嵌套对象一样，用中括号操作符同样可以访问嵌套数组。

下面是如何访问嵌套数组的例子：

var ourPets = {   
"cats": [  
"Meowzer",  
"Fluffy",  
"Kit-Cat"  
],  
"dogs": [  
"Spot",  
"Bowser",  
"Frankie"  
]  
};  
ourPets.cats[1]; // "Fluffy"  
ourPets.dogs[0]; // "Spot"

JavaScript JSON集合操作

右边有一个JSON对象，代表着你的专辑集。每一张专辑由一个唯一的id标识，并具有多种属性。但并非所有的专辑都有完整的信息。

写一个函数，它有个三个参数，id（专辑名称）、prop（专辑属性）、 value“属性内容”。

如果 value !='' 而且prop != 'tracks' ，collectionCopy[id][prop]=value;。

（添加属性‘prop’和元素内容‘value’）

如果 value !='' 而且prop == 'tracks' ，collectionCopy[id][prop].push(value);。（将元素内容放入已有的属性最后面）

如果 value == '' ，delete collectionCopy[id][prop];。（删除属性和元素内容）

谨记：函数返回的永远是整个对象。

**提示**  
使用中括号操作符来 [访问对象的变量属性](https://www.w3cschool.cn/codecamp/accessing-objects-properties-with-variables.html)。

例子：// Setup

var collection = {

2548: {

album: "Slippery When Wet",

artist: "Bon Jovi",

tracks: [

"Let It Rock",

"You Give Love a Bad Name"

]

},

2468: {

album: "1999",

artist: "Prince",

tracks: [

"1999",

"Little Red Corvette"

]

},

1245: {

artist: "Robert Palmer",

tracks: [ ]

},

5439: {

album: "ABBA Gold"

}

};

// Keep a copy of the collection for tests

var collectionCopy = JSON.parse(JSON.stringify(collection));

// Only change code below this line

function update(id, prop, value) {

if (prop == "tracks" && value != "" ) {

collection[id][prop].push(value); /\*update(1245, "tracks", "Addicted to Love") 之后, tracks 的最后一个元素应该是 "Addicted to Love" \*/

} else if (value!= "" ) {

collection[id][prop] = value; /\*update(5439, "artist", "ABBA")  之后, artist 应该为 "ABBA"\*/

} else if(value == ""){

delete collection[id][prop];

}/\*update(2548, "tracks", "") 之后, tracks 应该不存在 \*/

return collection;

}

update(2548, "artist", "");

# [JSON.parse()与JSON.stringify()的区别](https://www.cnblogs.com/goatling/p/6293692.html)

**JSON.parse()【从一个字符串中解析出json对象】**

例子：

//定义一个字符串

var data='{"name":"goatling"}'

//解析对象​

​JSON.parse(data)

结果是：

​name:"goatling"

**JSON.stringify()【从一个对象中解析出字符串】**

var data={name:'goatling'}

JSON.stringify(data)

结果是：

'{"name":"goatling"}'

# JavaScript 使用for语句循环迭代

一个条件语句只能执行一次代码，而通过一个循环语句你可以多次执行代码。

JavaScript 中最常见的循环就是“for循环”因为它可以允许特定次数的循环。

for循环中的三个表达式用分号隔开：

for ([初始化]; [条件判断]; [计数器])

初始化语句只会在执行循环开始之前执行一次。它通常用于定义和设置你的循环变量。

条件判断语句会在每一轮循环的开始执行，只要条件判断为 true 就会继续执行循环。当条件为 false的时候，循环将停止执行。这意味着，如果条件在一开始就为 false，这个循环将不会执行。

计数器是在每一轮循环结束时执行，通常用于递增或递减。

在下面的例子中，先初始化i = 0，条件 i < 5 为真，进入第一次循环，执行大括号里的代码，第一次循环结束。递增i的值，条件判断，就这样依次执行下去，直到条件判断为假，整个循环结束。

var ourArray = [];  
for (var i = 0; i < 5; i++) {  
ourArray.push(i);  
}

最终 ourArray 的值为 [0,1,2,3,4].

# JavaScript 使用for语句循环按奇数顺序迭代

for循环不必按照顺序来迭代，通过更改我们的 表达式，我们可以按照偶数顺序来迭代。

初始化 i = 0，当 i < 10 的时候继续循环。

i += 2 让 i 每次循环之后增加2。

var ourArray = [];  
for (var i = 0; i < 10; i += 2) {  
ourArray.push(i);  
}

循环结束后，ourArray 的值为 [0,2,4,6,8]。

让我们改变 表达式，这样我们可以用奇数来数。

JavaScript 使用for循环迭代输出数组

JavaScript中的常见任务是迭代数组的内容，一种方法是使用for循环。

下面的代码将输出数组arr 的每个元素到控制台：

var arr = [10,9,8,7,6];  
for (var i=0; i < arr.length; i++) {  
 console.log(arr[i]);  
}

记住数组的索引从零开始的，这意味着数组的最后一个元素的下标是：数组的长度 - 1。我们这个循环的 *条件* 是 i < arr.length，当 i 的值为 长度-1 的时候循环就停止了。

JavaScript 循环语句综合实战

如果你有一个二维数组，可以使用嵌套循环的逻辑，先遍历外面的数组，再遍历里面的子数组。下面是一个例子：

var arr = [  
[1,2], [3,4], [5,6]  
];  
for (var i=0; i < arr.length; i++) {  
for (var j=0; j < arr[i].length; j++) {  
console.log(arr[i][j]);  
}  
}

这样就可以一次输出 arr 中的每个子元素。请注意，对于内部循环，我们可以通过 arr[i] 的 .length 来获得子数组的长度，因为 arr[i] 的本身就是一个数组。

# JavaScript 使用while语句循环迭代

您可以使用循环多次运行相同的代码；

另一种类型的 JavaScript 循环被称为while循环，因为它运行“while”一个指定的条件为真，一旦该条件不再为真，就停止

var ourArray = [];   
var i = 0;   
while(i < 5) {   
ourArray.push(i);   
i++;   
}return i；

让我们尝试通过将值推送到数组来获得一个while循环

JavaScript 使用循环语句查找通讯录

我们有一个联系人列表，里面存储着不同联系人的数组对象。

函数 lookUpProfile 有两个预定义参数：firstName值和prop属性 。

该函数应检查firstName是实际联系人的firstName，给定的属性（prop）是该联系人的属性。

如果它们都存在，函数返回prop属性对应的值。

如果firstName 值不存在，返回 "No such contact"。

如果prop 属性不存在，返回 "No such property"

# JavaScript 使用random()生成随机小数

随机数对于创建随机行为很有用,计算机的行为只有两种：确定性和随机性。

而随机数最适合用来创建这种随机行为。

Math.random()用来生成一个在0(包括0)到1(不包括1)之间的随机小数，因此Math.random()可能返回0但绝不会返回1。

**提示**  
随后的函数都会在return执行前调用，所以我们可以直接返回Math.random()的值。

JavaScript 使用random()生成随机数

很好，我们可以生成随机的小数，但如果我们使用它来生成随机的整数，这将是更有用的机整数。

1. 用 Math.random() 生成一个随机小数。
2. 把这个随机小数乘以 20。
3. 用 Math.floor() 向下取整 获得它最近的整数。

记住 Math.random() 永远不会返回 1。同时因为我们是在用 Math.floor() 向下取整，所以最终我们获得的结果不可能有 20。这确保了我们获得了一个在0到19之间的整数。

把操作连缀起来，代码类似于下面：

Math.floor(Math.random() \* 20);

我们先调用 Math.random()，把它的结果乘以20，然后把上一步的结果传给 Math.floor()，最终通过向下取整获得最近的整数。

JavaScript 使用random()在一个范围内生成随机数

我们之前生成的随机数是在0到某个数之间，现在我们可以生成一个落在两个特定数字范围内的随机数间。

我们需要定义一个最小值和一个最大值。

下面是我们将要使用的方法，仔细看看并尝试理解这行代码正在干什么：

Math.floor(Math.random() \* (max - min + 1)) + min

（返回的数在Min(包括myMin)和Max(包括Max)之间的随机数。）

JavaScript 使用正则表达式操作字符串

Regular expressions 正则表达式被用来根据某种匹配模式来寻找strings中的某些单词。

例如：如果我们想要找到字符串The dog chased the cat中单词 the，我们可以使用下面的正则表达式: /the/gi

我们可以把这个正则表达式分成几段：

/ 是这个正则表达式的头部

the 是我们想要匹配的模式

/ 是这个正则表达式的尾部

g 代表着 global(全局)，意味着返回所有的匹配而不仅仅是第一个。

i 代表着忽略大小写，意思是当我们寻找匹配的字符串的时候忽略掉字母的大小写。

JavaScript 使用正则表达式选取数值

我们可以在正则表达式中使用特殊选择器来选取特殊类型的值。

特殊选择器中的一种就是数字选择器\d，意思是被用来获取一个字符串的数字。

在JavaScript中, 数字选择器类似于: /\d/g。

在选择器后面添加一个加号标记(+)，例如：/\d+/g，它允许这个正则表达式匹配一个或更多数字。

尾部的g是'global'的简写，意思是允许这个正则表达式 找到所有的匹配而不是仅仅找到第一个匹配。

JavaScript 使用正则表达式选取空白字符

我们也可以使用正则表达式选择器 \s 来查找一个字符串中的空白。

空白字符有 " " (空格符)、\r (回车符)、\n(换行符)、\t (制表符) 和 \f (换页符)。

空白正则表达式类似于：

/\s+/g

JavaScript 使用正则表达式反转匹配

你可以用正则表达式选择器的大写版本来反转任何匹配（相反意思）。

例如：\s 匹配任何空白字符，\S 匹配任何非空白字符。

JavaScript 让小游戏项目运作起来

让我们用 jQuery 选择器 $(".slot") 获得所有老虎机。

一旦获取到所有老虎机，我们可以通过中括号操作符获取到每一个老虎机：

$($(".slot")[0]).html(slotOne);

jQuery将会获取到第一个老虎机，并更新它的HTML为正确的数字。

JavaScript 为小游戏项目添加图片

现在给我们的老虎机加点图片。

我们已经为你准备好了图片images，我们可以通过不同的索引来获取每个图片。

现在让我们设置第一个老虎机根据随机数来显示一张图片：

$($('.slot')[0]).html('<img src = "' + images[slotOne-1] + '">');

# W3Cschool

JavaScript：写入 HTML 输出

## Write方法

实例

document.write("<h1>This is a heading</h1>");

document.write("<p>This is a paragraph</p>");

**提示：**您只能在 HTML 输出中使用 document.write。如果您在文档加载后使用该方法，会覆盖整个文档。

## JavaScript：对事件作出反应

实例

<button type="button" onclick="alert('Welcome!')">点击这里</button>

alert() 函数在 JavaScript 中并不常用，但它对于代码测试非常方便。

onclick 事件只是您即将在本教程中学到的众多事件之一。

## innerText属性

innerText：页面获取内容的时候，会把标签过滤掉，会对标签进行转义。

## Value属性

## innerHTML属性

设置或返回表格行的开始和结束标签之间的 HTML，会对标签进解析。

<html>

<head>

<script type="text/javascript">

function getInnerHTML()

{

alert(document.getElementById("tr1").innerHTML);

}

</script>

</head>

<body>

<table border="1">

<tr id="tr1">

<th>Firstname</th>

<th>Lastname</th>

</tr>

<tr id="tr2">

<td>Peter</td>

<td>Griffin</td>

</tr>

</table>

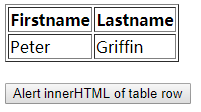
<br />

<input type="button" onclick="getInnerHTML()"

value="Alert innerHTML of table row" />

</body>

</html>

显示如下：

点击后

## Jquery中使用text方法

设置或者获取所选元素的文本内容；

## Jquery中使用html方法

html();设置或者获取所选元素的内容（包括html标记）；

## Jquery中使用val方法

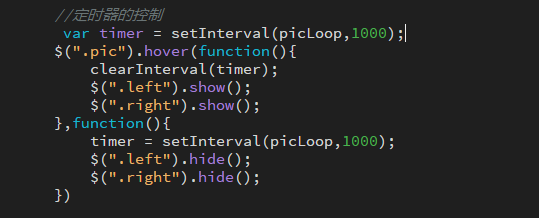
val();设置或者获取表单字段的值（前提是表单设置了value属性）；

## 计时器setInterval()clearInterval()方法

setInterval() 方法可按照指定的周期（以毫秒计）来调用函数或计算表达式。

setInterval() 方法会不停地调用函数，直到 clearInterval() 被调用或窗口被关闭。

由 setInterval() 返回的 ID 值可用作 clearInterval() 方法的参数。



## Match方法

match() 方法可在字符串内检索指定的值，或找到一个或多个正则表达式的匹配。

该方法类似 indexOf() 和 lastIndexOf()，但是它返回指定的值，而不是字符串的位置。

stringObject.match(searchvalue)

stringObject.match(regexp)

|  |  |
| --- | --- |
| **参数** | **描述** |
| searchvalue | 必需。规定要检索的字符串值。 |
| regexp | 必需。规定要匹配的模式的 RegExp 对象。如果该参数不是 RegExp 对象，则需要首先把它传递给 RegExp 构造函数，将其转换为 RegExp 对象。 |

## isNaN函数

isNaN() 函数用于检查其参数是否是非数字值。

isNaN(x)

|  |  |
| --- | --- |
| **参数** | **描述** |
| x | 必需。要检测的值。 |

如果 x 是特殊的非数字值 NaN（或者能被转换为这样的值），返回的值就是 true。如果 x 是其他值,则返回 false。

# Document对象

Document 对象使我们可以从脚本中对 HTML 页面中的所有元素进行访问。

## Document 对象集合

|  |  |
| --- | --- |
| **集合** | **描述** |
| [all[]](http://www.w3school.com.cn/jsref/coll_doc_all.asp) | 提供对文档中所有 HTML 元素的访问。 |
| [anchors[]](http://www.w3school.com.cn/jsref/coll_doc_anchors.asp) | 返回对文档中所有 Anchor 对象的引用。1[返回文档中锚的数目](http://www.w3school.com.cn/tiy/t.asp?f=hdom_document_anchors)  2[返回文档中第一个锚（a标签）的 innerHTML](http://www.w3school.com.cn/tiy/t.asp?f=hdom_document_anchors2) |
| applets | 返回对文档中所有 Applet 对象的引用。 |
| [forms[]](http://www.w3school.com.cn/jsref/coll_doc_forms.asp) | 返回对文档中所有 Form 对象引用。1[计算文档中表单的数目](http://www.w3school.com.cn/tiy/t.asp?f=hdom_collection_length)  2[访问集合中的项目](http://www.w3school.com.cn/tiy/t.asp?f=hdom_collection_item) |
| [images[]](http://www.w3school.com.cn/jsref/coll_doc_images.asp) | 返回对文档中所有 Image 对象引用。[计算文档中的图像数目](http://www.w3school.com.cn/tiy/t.asp?f=hdom_collection_imglength) |
| [links[]](http://www.w3school.com.cn/jsref/coll_doc_links.asp) | 返回对文档中所有 Area 和 Link 对象引用。 （1[返回文档中链接的数目](http://www.w3school.com.cn/tiy/t.asp?f=hdom_document_links)  2[返回文档中第一个链接的 id](http://www.w3school.com.cn/tiy/t.asp?f=hdom_document_links2)） |

## Document 对象属性

|  |  |
| --- | --- |
| **属性** | **描述** |
| body | 提供对 <body> 元素的直接访问。  对于定义了框架集的文档，该属性引用最外层的 <frameset>。 |
| [cookie](http://www.w3school.com.cn/jsref/prop_doc_cookie.asp) | 设置或返回与当前文档有关的所有 cookie。[返回与当前文档相关的 cookie](http://www.w3school.com.cn/tiy/t.asp?f=hdom_document_cookie) |
| [domain](http://www.w3school.com.cn/jsref/prop_doc_domain.asp) | 返回当前文档的域名。[返回下载当前文档的服务器域名](http://www.w3school.com.cn/tiy/t.asp?f=hdom_document_domain) |
| [lastModified](http://www.w3school.com.cn/jsref/prop_doc_lastmodified.asp) | 返回文档被最后修改的日期和时间。 |
| [referrer](http://www.w3school.com.cn/jsref/prop_doc_referrer.asp) | 返回载入当前文档的文档的 URL。 |
| [title](http://www.w3school.com.cn/jsref/prop_doc_title.asp) | 返回当前文档的标题。 |
| [URL](http://www.w3school.com.cn/jsref/prop_doc_url.asp) | 返回当前文档的 URL。 |

## Document 对象方法

|  |  |
| --- | --- |
| **方法** | **描述** |
| [close()](http://www.w3school.com.cn/jsref/met_doc_close.asp) | 关闭用 document.open() 方法打开的输出流，并显示选定的数据。 |
| [getElementById()](http://www.w3school.com.cn/jsref/met_doc_getelementbyid.asp) | 返回对拥有指定 id 的第一个对象的引用。 |
| [getElementsByName()](http://www.w3school.com.cn/jsref/met_doc_getelementsbyname.asp) | 返回带有指定名称的对象集合。 |
| [getElementsByTagName()](http://www.w3school.com.cn/jsref/met_doc_getelementsbytagname.asp) | 返回带有指定标签名的对象集合。 （a、p、input等） |
| [open()](http://www.w3school.com.cn/jsref/met_doc_open.asp) | 打开一个流，以收集来自任何 document.write() 或 document.writeln() 方法的输出。 |
| [write()](http://www.w3school.com.cn/jsref/met_doc_write.asp) | 向文档写 HTML 表达式 或 JavaScript 代码。 |
| [writeln()](http://www.w3school.com.cn/jsref/met_doc_writeln.asp) | 等同于 write() 方法，不同的是在每个表达式之后写一个换行符。 |

代码：document.write("Hello World! ","Hello You! ","<p style='color:blue;'>Hello World!</p>")

Hello World! Hello You!

Hello World!

注意：

请使用 document.write() 仅仅向文档输出写内容。

如果在文档已完成加载后执行 document.write，整个 HTML 页面将被覆盖：

<!DOCTYPE html>

<html>

<body>

<h1>我的第一张网页</h1>

<p>我的第一个段落。</p>

<button onclick="myFunction()">点击这里</button>

<script>

function myFunction()

{

document.write("糟糕！文档消失了。");

}

</script>

</body>

</html>

结果：



点击后：



## Open方法

open() 方法可打开一个新文档，并擦除当前文档的内容。

document.open(mimetype,replace)

|  |  |
| --- | --- |
| **参数** | **描述** |
| mimetype | 可选。规定正在写的文档的类型。默认值是 "text/html"。 |
| replace | 可选。当此参数设置后，可引起新文档从父文档继承历史条目。 |

该方法将擦除当前 HTML 文档的内容，开始一个新的文档，新文档用 write() 方法或 writeln() 方法编写。

**重要事项：**调用 open() 方法打开一个新文档并且用 write() 方法设置文档内容后，必须记住用 close 方法关闭文档，并迫使其内容显示出来。

**注释：**属于被覆盖的文档的一部分的脚本或事件句柄不能调用该方法，因为脚本或事件句柄自身也会被覆盖。

# JavaScript 输出

JavaScript 没有任何打印或者输出的函数。

JavaScript 可以通过不同的方式来输出数据：

* 使用 **window.alert()** 弹出警告框。
* 使用 **document.write()** 方法将内容写到 HTML 文档中。
* 使用 **innerHTML** 写入到 HTML 元素。
* 使用 **console.log()** 写入到浏览器的控制台。

|  |  |
| --- | --- |
| **语句** | **描述** |
| break | 用于跳出循环。 |
| catch | 语句块，在 try 语句块执行出错时执行 catch 语句块。 |
| continue | 跳过循环中的一个迭代。 |
| do ... while | 执行一个语句块，在条件语句为 true 时继续执行该语句块。 |
| for | 在条件语句为 true 时，可以将代码块执行指定的次数。 |
| for ... in | 用于遍历数组或者对象的属性（对数组或者对象的属性进行循环操作）。 |
| function | 定义一个函数 |
| if ... else | 用于基于不同的条件来执行不同的动作。 |
| return | 退出函数 |
| switch | 用于基于不同的条件来执行不同的动作。 |
| throw | 抛出（生成）错误 。 |
| try | 实现错误处理，与 catch 一同使用。 |
| var | 声明一个变量。 |
| while | 当条件语句为 true 时，执行语句块。 |

# 正则表达式

是由一个字符序列形成的搜索模式。

当你在文本中搜索数据时，你可以用搜索模式来描述你要查询的内容。

正则表达式可以是一个简单的字符，或一个更复杂的模式。

正则表达式可用于所有文本搜索和文本替换的操作。

/pattern/modifiers;

var patt = /w3cschool/i

实例解析：

**/w3cschool/i**  是一个正则表达式。

**w3cschool**  是一个模式 (用于检索)。

**i**  是一个修饰符 (搜索不区分大小写)。

在 JavaScript 中，正则表达式通常用于两个字符串方法 : search() 和 replace()。

## 正则表达式修饰符

**修饰符** 可以在全局搜索中不区分大小写:

|  |  |
| --- | --- |
| **修饰符** | **描述** |
| i | 执行对大小写不敏感的匹配。 |
| g | 执行全局匹配（查找所有匹配而非在找到第一个匹配后停止）。 |
| m | 执行多行匹配。 |

## 正则表达式模式

方括号用于查找某个范围内的字符：

|  |  |
| --- | --- |
| **表达式** | **描述** |
| [abc] | 查找方括号之间的任何字符。 |
| [0-9] | 查找任何从 0 至 9 的数字。 |
| (x|y) | 查找任何以 | 分隔的选项。 |

元字符是拥有特殊含义的字符：

|  |  |
| --- | --- |
| **元字符** | **描述** |
| \d | 查找数字。 |
| \s | 查找空白字符。 |
| \b | 匹配单词边界。 |
| \uxxxx | 查找以十六进制数 xxxx 规定的 Unicode 字符。 |

量词:

|  |  |
| --- | --- |
| **量词** | **描述** |
| n+ | 匹配任何包含至少一个 n 的字符串。 |
| n\* | 匹配任何包含零个或多个 n 的字符串。 |
| n? | 匹配任何包含零个或一个 n 的字符串。 |

## search() 方法

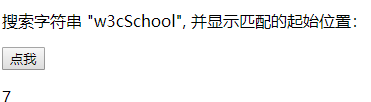
用于检索字符串中指定的子字符串，或检索与正则表达式相匹配的子字符串，并返回子字符串的起始位置。

## replace() 方法

用于在字符串中用一些字符替换另一些字符，或替换一个与正则表达式匹配的子字符串。

例如：



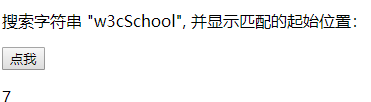


## search() 方法使用字符串

search 方法可使用字符串作为参数。字符串参数会转换为正则表达式：

检索字符串中 "w3cschool" 的子字符串：

var str = "Visit w3cschool!";   
var n = str.search("w3cschool");



## replace() 方法使用正则表达式

使用正则表达式且不区分大小写将字符串中的 Microsoft 替换为 w3cschool :

var str = "Visit Microsoft!";   
var res = str.replace(/microsoft/i, "w3cschool");

结果输出为:

Visit w3cschool!

## replace() 方法使用字符串

replace() 方法将接收字符串作为参数：

var str = "Visit Microsoft!";   
var res = str.replace("Microsoft", "w3cschool");

## 使用 test()

test() 方法是一个正则表达式方法。

test() 方法用于检测一个字符串是否匹配某个模式，如果字符串中含有匹配的文本，则返回 true，否则返回 false。

以下实例用于搜索字符串中的字符 "e"：

var patt = /e/;  注意//必须要有  
patt.test("The best things in life are free!");

字符串中含有 "e"，所以该实例输出为：true

你可以不用设置正则表达式的变量，以上两行代码可以合并为一行：

/e/.test("The best things in life are free!")

## 使用 exec()

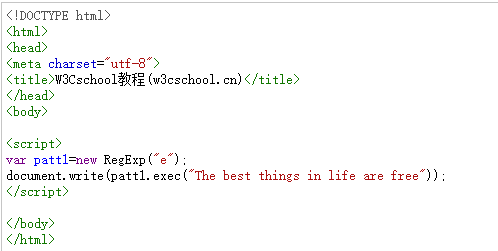
exec() 方法是一个正则表达式方法。

exec() 方法用于检索字符串中的正则表达式的匹配。

该函数返回一个数组，其中存放匹配的结果。如果未找到匹配，则返回值为 null。

以下实例用于搜索字符串中的字母 "e":

/e/.exec("The best things in life are free!");



字符串中含有 "e"，所以该实例输出为: e

## 使用 compile()

compile() 方法用于改变 RegExp。

compile() 既可以改变检索模式，也可以添加或删除第二个参数。

var patt1=new RegExp("e"); document.write(patt1.test("The best things in life are free")); patt1.compile("d"); document.write(patt1.test("The best things in life are free"));

由于字符串中存在 "e"，而没有 "d"，以上代码的输出是：

truefalse

## 对代码行进行折行

您可以在文本字符串中使用反斜杠对代码行进行换行。下面的例子会正确地显示：

document.write("Hello \

World!");

## Write的注意

不能直接输出对象，不然会显示Object

<!DOCTYPE html>

<html>

<body>

<script>

var carname1="Bill Gates";

var carname2='Bill Gates';

var answer1="Nice to meet you!";

var answer2="He is called 'Bill'";

var answer3='He is called "Bill"';

document.write(carname1 + carname2+"<br>"+"carname2")

</script>

</body>

</html>



## toUpperCase() 方法来把字符串中字母转换为大写

var message="Hello world!";

var x=message.toUpperCase();

在以上代码执行后，x 的值是：

HELLO WORLD!

## toLowerCase（） 方法来把字符串中字母转换为大写

## onload 事件

onload 事件会在页面或图像加载完成后立即发生。

onload="SomeJavaScriptCode"

## jQuery里的ready（）事件

允许使用以下三种语法：

1. $(document).ready(*function*)
2. $().ready(*function*)
3. $(*function*)

|  |  |
| --- | --- |
| **参数** | **描述** |
| function | 必需。规定当文档加载后要运行的函数。 |

**提示：**ready() 函数不应与 <body onload=""> 一起使用。

## jQuery attr方法

attr() 方法设置或返回被选元素的属性值。

根据该方法不同的参数，其工作方式也有所差异。

返回被选元素的属性值。

$(*selector*).attr(*attribute*)

|  |  |
| --- | --- |
| **参数** | **描述** |
| *attribute* | 规定要获取其值的属性。 |

设置被选元素的属性和值。

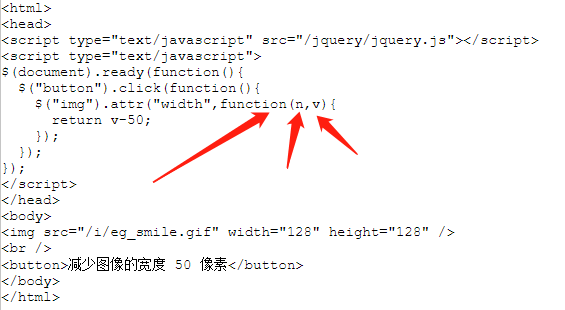
$(*selector*).attr(*attribute*,*value*)

|  |  |
| --- | --- |
| **参数** | **描述** |
| *attribute* | 规定属性的名称。 |
| *value* | 规定属性的值。 |

使用函数来设置被选元素的属性和值。

$(*selector*).attr(*attribute*,*function(index,oldvalue)*)

|  |  |
| --- | --- |
| **参数** | **描述** |
| *attribute* | 规定属性的名称。 |
| *function(index,oldvalue)* | 规定返回属性值的函数。  该函数可接收并使用选择器的 index 值和当前属性值。 |





为被选元素设置一个以上的属性和值。

$(*selector*).attr({*attribute:value*, *attribute:value* ...})

|  |  |
| --- | --- |
| **参数** | **描述** |
| *attribute:value* | 规定一个或多个属性/值对。 |

## jquery页面显示文本text

$(".num").text(index31); // 数量跟着显示

# 阻止冒泡、起泡

点击div里面的盒子，也默认点击div，触发两个事件。

e.stopPropagation();

注意：方法里的参数需要加e

例如

function 方法名称(e) {

e.stopPropagation();

}

# React

## 打包后的路径

打包后的路径是按照打包后的html的位置，写路径

标签里的属性写=号，js属性里写：。

## JavaScript打印显示

JavaScript 能够以不同方式“显示”数据：

* 使用 window.alert() 写入警告框
* 使用 document.write() 写入 HTML 输出

**注意：**在 HTML 文档完全加载后使用 **document.write()** 将**删除所有已有的 HTML**

**提示：document.write()** 方法仅用于测试。

* 使用 innerHTML 写入 HTML 元素
* 使用 console.log() 写入浏览器控制台

## JavaScript 关键词

JavaScript 语句常常通过某个关键词来标识需要执行的 JavaScript 动作。

下面的表格列出了一部分将在教程中学到的关键词：

|  |  |
| --- | --- |
| **关键词** | **描述** |
| break | 终止 switch 或循环。 |
| continue | 跳出循环并在顶端开始。 |
| debugger | 停止执行 JavaScript，并调用调试函数（如果可用）。 |
| do ... while | 执行语句块，并在条件为真时重复代码块。 |
| for | 标记需被执行的语句块，只要条件为真。 |
| function | 声明函数。 |
| if ... else | 标记需被执行的语句块，根据某个条件。 |
| return | 退出函数。 |
| switch | 标记需被执行的语句块，根据不同的情况。 |
| try ... catch | 对语句块实现错误处理。 |
| var | 声明变量。 |

**注释：**JavaScript 关键词指的是保留的单词。保留词无法用作变量名。

## JavaScript 标识符

标识符是名称。

在 JavaScript 中，标识符用于命名变量（以及关键词、函数和标签）

**提示：**数值不可以作为首字符。这样，JavaScript 就能轻松区分标识符和数值。

## JavaScript 与驼峰式大小写

JavaScript 程序员倾向于使用以小写字母开头的驼峰大小写：

firstName, lastName, masterCard, interCity

## JavaScript 标识符

所有 JavaScript **变量**必须以**唯一的名称**的**标识**。

构造变量名称（唯一标识符）的通用规则是：

* 名称可包含字母、数字、下划线和美元符号
* 名称必须以字母开头
* 名称也可以 $ 和 \_ 开头（但是在本教程中我们不会这么做）
* 名称对大小写敏感（y 和 Y 是不同的变量）
* 保留字（比如 JavaScript 的关键词）无法用作变量名称

**提示：**JavaScript 标识符对大小写敏感。

## 变量没有值Value = undefined

在计算机程序中，被声明的变量经常是不带值的。值可以是需被计算的内容，或是之后被提供的数据，比如数据输入

声明之后，变量是没有值的。（技术上，它的值是 undefined。）

赋值运算符向 JavaScript 变量赋值。

|  |  |  |
| --- | --- | --- |
| **运算符** | **例子** | **等同于** |
| = | x = y | x = y |
| += | x += y | x = x + y |
| -= | x -= y | x = x - y |
| \*= | x \*= y | x = x \* y |
| /= | x /= y | x = x / y |
| %= | x %= y | x = x % y |

## JavaScript 比较运算符

|  |  |
| --- | --- |
| **运算符** | **描述** |
| == | 等于 |
| === | 等值等型 |
| != | 不相等 |
| !== | 不等值或不等型 |
| > | 大于 |
| < | 小于 |
| >= | 大于或等于 |
| <= | 小于或等于 |
| ? | 三元运算符 |

## JavaScript 逻辑运算符

|  |  |
| --- | --- |
| **运算符** | **描述** |
| && | 逻辑与 |
| || | 逻辑或 |
| ! | 逻辑非 |

**注释：**[JS 比较](http://www.w3school.com.cn/js/js_comparisons.asp)这一章中完整描述了逻辑运算符。

## JavaScript 类型运算符

|  |  |
| --- | --- |
| **运算符** | **描述** |
| typeof | 返回变量的类型。 |
| instanceof | 返回 true，如果对象是对象类型的实例。 |

**注释：**[JS 类型转换](http://www.w3school.com.cn/js/js_type_conversion.asp)这一章完整描述了类型运算符。

## JavaScript 位运算符

位运算符处理 32 位数。

该运算中的任何数值运算数都会被转换为 32 位的数。结果会被转换回 JavaScript 数。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **运算符** | **描述** | **例子** | **等同于** | **结果** | **十进制** |
| & | 与 | 5 & 1 | 0101 & 0001 | 0001 | 1 |
| | | 或 | 5 | 1 | 0101 | 0001 | 0101 | 5 |
| ~ | 非 | ~ 5 | ~0101 | 1010 | 10 |
| ^ | 异或 | 5 ^ 1 | 0101 ^ 0001 | 0100 | 4 |
| << | 零填充左位移 | 5 << 1 | 0101 << 1 | 1010 | 10 |
| >> | 有符号右位移 | 5 >> 1 | 0101 >> 1 | 0010 | 2 |
| >>> | 零填充右位移 | 5 >>> 1 | 0101 >>> 1 | 0010 | 2 |

## JavaScript 运算符优先级值

|  |  |  |  |
| --- | --- | --- | --- |
| **值** | **运算符** | **描述** | **实例** |
| 20 | ( ) | 表达式分组 | (3 + 4) |
|  |  |  |  |
| 19 | . | 成员 | person.name |
| 19 | [] | 成员 | person["name"] |
| 19 | () | 函数调用 | myFunction() |
| 19 | new | 创建 | new Date() |
|  |  |  |  |
| 17 | ++ | 后缀递增 | i++ |
| 17 | -- | 后缀递减 | i-- |
|  |  |  |  |
| 16 | ++ | 前缀递增 | ++i |
| 16 | -- | 前缀递减 | --i |
| 16 | ! | 逻辑否 | !(x==y) |
| 16 | typeof | 类型 | typeof x |
|  |  |  |  |
| 15 | \*\* | 求幂 (ES7) | 10 \*\* 2 |
|  |  |  |  |
| 14 | \* | 乘 | 10 \* 5 |
| 14 | / | 除 | 10 / 5 |
| 14 | % | 模数除法 | 10 % 5 |
|  |  |  |  |
| 13 | + | 加 | 10 + 5 |
| 13 | - | 减 | 10 - 5 |
|  |  |  |  |
| 12 | << | 左位移 | x << 2 |
| 12 | >> | 右位移 | x >> 2 |
| 12 | >>> | 右位移（无符号） | x >>> 2 |
|  |  |  |  |
| 11 | < | 小于 | x < y |
| 11 | <= | 小于或等于 | x <= y |
| 11 | > | 大于 | x > y |
| 11 | >= | 大于或等于 | x >= y |
| 11 | in | 对象中的属性 | "PI" in Math |
| 11 | instanceof | 对象的实例 | instanceof Array |
|  |  |  |  |
| 10 | == | 相等 | x == y |
| 10 | === | 严格相等 | x === y |
| 10 | != | 不相等 | x != y |
| 10 | !== | 严格不相等 | x !== y |
|  |  |  |  |
| 9 | & | 按位与 | x & y |
| 8 | ^ | 按位 XOR | x ^ y |
| 7 | | | 按位或 | x | y |
| 6 | && | 逻辑与 | x && y |
| 5 | || | 逻辑否 | x || y |
| 4 | ? : | 条件 | ? "Yes" : "No" |
|  |  |  |  |
| 3 | = | 赋值 | x = y |
| 3 | += | 赋值 | x += y |
| 3 | -= | 赋值 | x -= y |
| 3 | \*= | 赋值 | x \*= y |
| 3 | %= | 赋值 | x %= y |
| 3 | <<= | 赋值 | x <<= y |
| 3 | >>= | 赋值 | x >>= y |
| 3 | >>>= | 赋值 | x >>>= y |
| 3 | &= | 赋值 | x &= y |
| 3 | ^= | 赋值 | x ^= y |
| 3 | |= | 赋值 | x |= y |
|  |  |  |  |
| 2 | yield | 暂停函数 | yield x |
| 1 | , | 逗号 | 7 , 8 |

## Null

在 JavaScript 中，null 是 "nothing"。它被看做不存在的事物。

不幸的是，在 JavaScript 中，null 的数据类型是对象。

您可以把 null 在 JavaScript 中是对象理解为一个 bug。它本应是 null。

您可以通过设置值为 null 清空对象：

实例

var person = null; // 值是 null，但是类型仍然是对象

[亲自试一试](http://www.w3school.com.cn/tiy/t.asp?f=js_datatypes_null)

您也可以通过设置值为 undefined 清空对象：

实例

var person = undefined; // 值是 undefined，类型是 undefined

[亲自试一试](http://www.w3school.com.cn/tiy/t.asp?f=js_datatypes_undefined_3)

## Undefined 与 Null 的区别

Undefined 与 null 的值相等，但类型不相等：

typeof undefined // undefined

typeof null // object

null === undefined // false

null == undefined // true

[亲自试一试](http://www.w3school.com.cn/tiy/t.asp?f=js_datatypes_undefined_4)

## 原始数据

原始数据值是一种没有额外属性和方法的单一简单数据值。

typeof 运算符可返回以下原始类型之一：

* string
* number
* boolean
* undefined

实例

typeof "Bill" // 返回 "string"

typeof 3.14 // 返回 "number"

typeof true // 返回 "boolean"

typeof false // 返回 "boolean"

typeof x // 返回 "undefined" (假如 x 没有值)

[亲自试一试](http://www.w3school.com.cn/tiy/t.asp?f=js_datatypes_typeof_primitive)

## 复杂数据

typeof 运算符可返回以下两种类型之一：

* function
* object

typeof 运算符把对象、数组或 null 返回 object。

typeof 运算符不会把函数返回 object。

实例

typeof {name:'Bill', age:62} // 返回 "object"

typeof [1,2,3,4] // 返回 "object" (并非 "array"，参见下面的注释)

typeof null // 返回 "object"

typeof function myFunc(){} // 返回 "function"

[亲自试一试](http://www.w3school.com.cn/tiy/t.asp?f=js_datatypes_typeof_complexdata)

typeof 运算符把数组返回为 "object"，因为在 JavaScript 中数组即对象。

# 常见的 HTML 事件

下面是一些常见的 HTML 事件：

|  |  |
| --- | --- |
| **事件** | **描述** |
| onchange | HTML 元素已被改变 |
| onclick | 用户点击了 HTML 元素 |
| onmouseover | 用户把鼠标移动到 HTML 元素上 |
| onmouseout | 用户把鼠标移开 HTML 元素 |
| onkeydown | 用户按下键盘按键 |
| onload | 浏览器已经完成页面加载 |

更完整的列表：[W3School JavaScript 参考手册 HTML DOM 事件](http://www.w3school.com.cn/jsref/dom_obj_event.asp)。

# Js手机自适应

<script src="http://res.wx.qq.com/open/js/jweixin-1.0.0.js"></script>

<script src="http://203.195.235.76/jssdk/js/zepto.min.js"></script>