# 前端三件套Html Css Js

# Js语法

## 数组的方法有哪些(18个)

push 末尾添加

pop 末尾删除

shift 首部删除

unshift 首部添加

concat 数组合并

join 数组元素 通过连接符 连接 数组.join(“-”)

reverse 数组反转

sort 数组排序

map 原数组的每一项执行函数后，返回一个新的数组。原数组不变。（注意该方法和forEach的区别）。

forEach 针对每一个元素执行提供的函数，对数据的操作会改变原数组。（注意该方法和map的区别，若直接打印Array.forEach，结果为undefined）。

filter 过滤数组中，符合条件的元素并返回一个新的数组。

indexOf 检测当前值在数组中第一次出现的位置索引，未找到返回-1

includes 判断一个数组是否包含一个指定的值

slice 从start开始，end之前结束，不到end；如果不给end值，从start开始到数组结束。start可以给负值，-1表示数组最后位置，-2表示倒数第二个，以此类推，顾前不顾后。

在数组中读取元素：

var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"]; var citrus = fruits.slice(1,3);

citrus 结果输出:

Orange,Lemon

splice 删除元素并添加元素，从index位置开始删除howmany个元素，并将arr1、arr2...数据从index位置依次插入。howmany为0时，则不删除元素。原数组改变。

对数组进行增删改

增加：ary.splice(n,0,m)从索引n开始删除0项，把m或者更多的内容插入到索引n的前面

返回空数组

修改：ary.splice(n,x,m)从索引n开始删除x个，m替换删除的部分

把原有内容删除掉，然后用新内容替换掉

删除：ary.splice(n,m) 从索引n开始删除m个内容

（如果第二个参数省略，则从n删除到末尾）

返回删除的新数组，原有数组改变

reduce() 方法接收一个函数作为累加器，数组中的每个值（从左到右）开始缩减，最终计算为一个值。

some 对数组中的每一项进行判断，若都不符合则返回false，否则返回true。

every 对数组中的每一项进行判断，若都符合则返回true，否则返回false。

## 常用JS——string字符串方法

### slice substr substring

一、关于字符串分割  
1、[slice](https://so.csdn.net/so/search?q=slice&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/where_slr/article/details/_blank)（start,end）;  
关于这个方法，一定要搞懂四个关键点：

（1）截取[字符串](https://so.csdn.net/so/search?q=%E5%AD%97%E7%AC%A6%E4%B8%B2&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/where_slr/article/details/_blank)时**不包括下标为end**的元素。

（2）**end是可选[参数](https://so.csdn.net/so/search?q=%E5%8F%82%E6%95%B0&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/where_slr/article/details/_blank)**，没有时，默认从start到结束的所有字符串。

**（3）[String](https://so.csdn.net/so/search?q=String&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/where_slr/article/details/_blank).slice与Array.slice区别。**

（4）参数为负数时，是如何处理的。

其中第3点其实就是在JavaScript中字符串和[数组](https://so.csdn.net/so/search?q=%E6%95%B0%E7%BB%84&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/where_slr/article/details/_blank)都具有这个方法，它们的返回结果形式不一样，一个是一段字符串，一个是一段**浅复制后的数组**。另外三个点接着往下看你就会明白。

2、substr（start，length）；

关于这个方法，也需要搞懂几个关键点：

（1）第二个参数是子串中的字符数，**必须是数值**。可以没有。

（2）参数为负数时如何进行处理。

（3）最重要一点来自官网说明，如下所示：

ECMAscript 没有对该方法进行标准化，因此反对使用它。

由于substr() 的参数指定的是子串的开始位置和长度，因此它可以替代 [substring](https://so.csdn.net/so/search?q=substring&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/where_slr/article/details/_blank)() 和 slice() 来使用。

3、substring（start，end）；

和slice类似，关于这个方法，同样需要搞懂几个关键点：

（1）返回的字符串中不包括end处的**字符**。

（2）参数为负数时如何进行处理。

（3）如果参数 start 与 end 相等，那么该方法返回的就是**一个空串**（即长度为 0 的字符串）。

（4）如果 start 比 end 大，那么该方法在提取子串之前会先交换这两个参数。

这个方法的作用同样可以使用slice方法来替代。

区别和使用：

（一）：都接收两个参数，slice和substring接收的是**起始位置和结束位置**(注意：不包括结束位置)，而substr接收的则是起始位置和所要返回的字符串长度。

  
看一下控制台运行出来的结果，其中slice/substring都是从0开始截取3到6位置但不包括6位置的字符串"lo “，对比之下，substr截取3位置之后的**6个字符**串"lo wor”。

（二）：需要注意substring是**以两个参数中较小一个作为起始位置，较大的参数作为结束位置。**IMG_264

上面这个（6，3）会默认变成（3，6）将小的当起始位置来处理。

（三）：参数为负数时如何解析。

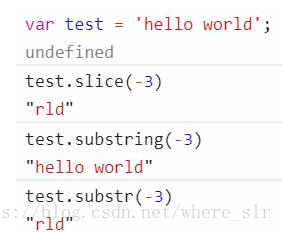
（我认为这种负数的情况就应该在标准中禁止掉，参数中有负数时直接报错就行，可现在竟然能为负数，那我们只有两条出路：1，避免在code的时候出现它；2，就是今天我们要做的，彻底搞懂它。一次性消灭掉）

slice：将字符串的长度与对应的负数相加，结果作为参数。

substr：只会将第一个参数与字符串长度相加后的结果作为第一个参数（第一个参数为负数）。

**substring：直接将负参数直接转成0。**

第一，先看一个参数的情况：



解释一下上面的结果：字符串长度是11，（11-3=8），所以slice（-3）和substr（-3）从下标为8的字母开始。得到的结果就是"rld"。而substring直接将负数据转为0，所以输出结果就是"hello world".

第二，看一下两个参数的情况：

先回忆一下上面的定义：

slice：**长度与负数相加作为参数**。

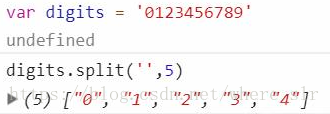
substring：**负数直接转为0**。

substr：**仅将第一个参数与长度相加作为第一个参数。**  


解释一下输出的结果：slice（3，11-4）就是从下标3截取到下标7的字符串，这里就是"lo w"。substring（3，0），其中-4直接转成0，所以由定义从下标0截取到3，这里表示"hel"。最后一个substr第一个参数不是负数，第二个表示长度的参数为负数时，输出只能是空字符串。

### Split

4、[split](https://so.csdn.net/so/search?q=split&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/where_slr/article/details/_blank)（字符串/正则,length）；



来阐述一下。split方法把一个字符串**string分割成片段创建一个字符串数组**，可选参数howmany可以限制被分割的片段的数量。separator参数可以是一个字符串或一个正则表达式。

分割每个字符，包括空格:

var str="How are you doing today?";  
var n=str.split("");

n 输出数组值得结果:

H,o,w, ,a,r,e, ,y,o,u, ,d,o,i,n,g, ,t,o,d,a,y,?

var str="How are you doing today?";  
var n=str.split();

n 输出数组值得结果:

How are you doing today?

var str="How are you doing today?";  
var n=str.split(" ",3);

n 将输出3个数组的值:

How,are,you

使用一个字符作为分隔符:

var str="How are you doing today?";  
var n=str.split("o");

n 输出数组值得结果:

H,w are y,u d,ing t,day?

二、其它字符串处理方法

### 1、charAt（pos）

返回指定位置（如上pos）的字符。如果pos小于0或者大于等于字符串的长度string.length，它会返回空字符串。

var name = 'hello world';

name.chartAt(0);

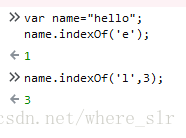
结果"h"

2、concat（string…）  
用于连接两个或者多个字符串。相较于数组[Array](https://so.csdn.net/so/search?q=Array&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/where_slr/article/details/_blank).concat()。其实字符串到是用的不多，主要是之前使用加号（+）会更方便一些，新语法时一步优化字符串拼接的操作。

IMG_268

### 3、indexOf（searchString，position）

在string内查找另一个字符串searchString。如果它被找到，就返回第1个匹配字符的位置，否则返回-1。

需要记住一点是：可选参数**position可设置从string的某个指定的位置开始查找**。  


### lastIndexOf（searchString，position） 与indexOf方法类似，只不过它是从该字符串的末尾开始查找而不是从开头。

### localeCompare（target） **用来比较两个字符串，返回比较结果数字。**如果stringObject 小于 target，则 localeCompare() 返回小于 0 的数。如果 stringObject 大于 target，则该方法返回大于 0 的数。如果两个字符串相等，或根据本地排序规则没有区别，该方法返回 **0**。 IMG_270

### 6、match（regexp）

match方法让字符串和一个正则表达式进行匹配。它依据g标识来决定如何进行匹配。如果没有g标识，那么调用string.match(regexp)的结果与调用regexp.exec(string)的结果相同。**带g标识返回的是一个结果数组**，表示不区分大小写

在字符串中查找 "ain":

var str="The rain in SPAIN stays mainly in the plain"; var n=str.match(/ain/g);

n 输出数组结果值:

ain,ain,ain

### 7、replace（searchValue，replaceValue） 作用：replace方法对string进行查找和替换操作，并返回一个新的字符串。

取值：而参数searchValue可以是一个字符串或者一个正则表达式对象。

第一种情况：如果searchValue是一个字符串，那么searchValue只会在第1次出现 的地方被替换

IMG_271

## 函数的多种写法

1.常规写法 具名函数

　　// 函数写法

function funName () {

alert('常规写法');

}

// 调用

funName();

2.将方法作为一个对象 对象函数

　　// 作为对象方法

var obj = {

funName:function(){

alert('这个必须放在一个对象内部，放在外边会出错！');

}

}

// 调用方法

obj.funName();

3.匿名函数

// 匿名函数

var myfun = function(){

alert('这是一种声明函数的方式，左边是一个变量，右边是一个函数的表达式，意思就是把一个匿名的函数表达式复制给了变量myfun，只是声明了一个变量指向了一个函数对象。');

}

// 调用

myfun();

4.构造函数中给对象添加方法，通常在构造函数中用到。

javascript中的每个对象都有prototype属性，Javascript中对象的prototype属性的解释是：返回对象类型原型的引用。

　　// 给对象添加方法

var funName = function(){}

funName.prototype.myfun = function(){

alert('这是在funName函数上的原始对象上加了一个myfun方法，构造函数中用到');

}

// 调用

var funname = new funName();// 创建对象

funname.myfun();

在给对象添加方法时可以用一下方式添加多个方法：

　　// 给对象添加多个方法

var funName = function(){}

funName.prototype = {

fun1:function(){

alert('fun1');

}

,fun2:function(){

alert('fun2');

}

}

// 调用

var funname = new funName();// 创建对象

funname.fun1();

funname.fun2();

## Js输出

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。

innerHTML与inner'Text

innerHTML指的是从对象的起始位置到终止位置的全部内容,包括Html标签。  
innerText   指的是从起始位置到终止位置的内容,但它去除Html标签。

## 字面量（js的数据类型）

**Java八种基本类型**

整型:int long short byte逻辑:boolean 字符char 浮点:float double

**Js中的九种数据类型**

值类型(基本类型)：

Number

String

Boolean

Null

Undefined

引用数据类型：

Date

Array

Object

Function

Synbol

变量必须以字母开头

变量也能以 $ 和 \_ 符号开头（不过我们不推荐这么做）

变量名称对大小写敏感（y 和 Y 是不同的变量）

未使用值来声明的变量，其值实际上是 undefined

NaN 的数据类型是 number

数组(Array)的数据类型是 object

日期(Date)的数据类型为 object

null 的数据类型是 object

未定义变量的数据类型为 undefined

typeof [1,2,3,4]              // 返回 object

typeof {name:'John', age:34}  // 返回 object  
typeof new Date()             // 返回 object  
typeof function () {}         // 返回 function

constructor 属性返回所有 JavaScript 变量的**构造函数**

[1,2,3,4].constructor              // 返回函数 Array()   { [native code] }  
{name:'John', age:34}.constructor  // 返回函数 Object()  { [native code] }  
new Date().constructor             // 返回函数 Date()    { [native code] }  
function () {}.constructor         // 返回函数 Function(){ [native code] }

## Undefined 和 Null

Undefined 这个值表示变量不含有值。

可以通过将变量的值设置为 null 来清空变量。

### null

在 JavaScript 中 null 表示 "什么都没有"。

null是一个只有一个值的特殊类型。表示一个空对象引用。

|  |  |
| --- | --- |
| IMG_256 | 用 typeof 检测 null 返回是**object。** |

你可以设置为 null 来清空对象:

### undefined

在 JavaScript 中, undefined 是一个没有设置值的变量。

typeof 一个没有值的变量会返回 **undefined**。

null 和 undefined 的**值相等**，但**类型**不等：

1、定义

 （1）undefined：是所有没有赋值变量的默认值，**自动赋值**。

 （2）null：主动**释放一个变量引用的对象**，表示一个变量**不再指向**任何对象地址。

2、何时使用null?

当使用完一个比较大的对象时，需要对其进行**释放内存**时，设置为 null。

3、null 与 undefined 的异同点是什么呢？

共同点：都是**原始类型**，保存在**栈中变量**本地。

不同点：

（1）undefined——表示变量声明过但并未赋过值。

它是所有未赋值变量**默认值**，例如：

var a; // a 自动被赋值为 undefined

（2）null——表示一个变量将来可能指向一个对象。

一般用于主动释放指向对象的引用，例如：

var emps = ['ss','nn'];

emps = null; // **释放指向数组的引用**

## 对象

真实生活中，一辆汽车是一个对象。

对象有它的属性，如重量和颜色等，方法有启动停止等:

那么实例就是驾驶员

有时，我们会希望函数将值返回调用它的地方。

通过使用 return 语句就可以实现。

在使用 return 语句时，函数会停止执行，并返回指定的值。

JavaScript 变量的生命期从它们被声明的时间开始。

局部变量会在函数运行以后被删除。

全局变量会在页面关闭后被删除。

如果变量在函数内没有声明（没有使用 var 关键字），该变量为全局变量。

## JavaScript作用和方法

|  |  |
| --- | --- |
| onchange | HTML 元素改变 |
| onclick | 用户点击 HTML 元素 |
| onmouseover | 鼠标指针移动到指定的元素上时发生 |
| onmouseout | 用户从一个 HTML 元素上移开鼠标时发生 |
| onkeydown | 用户按下键盘按键 |
| onload | 浏览器已完成页面的加载 |

JavaScript作用：事件可以用于处理表单验证，用户输入，用户行为及浏览器动作:

测试字符串长度时候，反斜杠是一个转义字符。 转义字符将特殊字符转换为字符串字符：

转义字符 (\) 可以用于转义撇号，换行，引号，等其他特殊字符。长度不包括空格

使用 new 关键字将字符串定义为一个对象： var firstName = new String("John")

=== 为绝对相等，即数据类型与值都必须相等。

## 字符串的方法

|  |  |
| --- | --- |
| charAt() | 返回指定索引位置的字符 |
| charCodeAt() | 返回指定索引位置字符的 Unicode 值 |
| concat() | 连接两个或多个字符串，返回**连接**后的字符串 |
| fromCharCode() | 将 Unicode 转换为字符串 |
| indexOf() | 返回字符串中检索指定字符第一次出现的位置 |
| lastIndexOf() | 返回字符串中检索指定字符最后一次出现的位置 |
| localeCompare() | 用本地特定的顺序来比较两个字符串 |
| match() | 找到一个或多个正则表达式的匹配 |
| replace() | 替换与**正则表达式**匹配的子串 |
| search() | 检索与**正则表达式**相匹配的值 |
| slice() | 提取字符串的片断，并在**新的字符串**中返回被提取的部分 |
| split() | 把字符串分割为子字符串数组 |
| substr() | 从**起始索引号**提取字符串中指定**数目**的字符 |
| substring() | 提取字符串中两个指定的**索引号**之间的字符 |
| toLocaleLowerCase() | 根据主机的语言环境把字符串转换为小写，只有几种语言（如土耳其语）具有地方特有的大小写映射 |
| toLocaleUpperCase() | 根据主机的语言环境把字符串转换为大写，只有几种语言（如土耳其语）具有地方特有的大小写映射 |
| toLowerCase() | 把字符串转换为小写 |
| toString() | 返回字符串**对象**值 |
| toUpperCase() | 把字符串转换为大写 |
| trim() | 移除字符串**首尾**空白 |
| valueOf() | 返回某个字符串对象的原始值 |

## 运算符

算术运算符 赋值运算符 比较运算符 逻辑运算符 条件运算符

“.”操作符的优先级要高于赋值

break 语句用于跳出循环。

continue 用于跳过循环中的一个迭代。

break 的作用是跳出代码块, 所以 break 可以使用于循环和 **switch** 等

continue 的作用是进入下一个迭代, 所以 continue 只能用于**循环**的代码块。

## 将数字转换为字符串

全局方法**String()** 可以将数字转换为字符串。

该方法可用于任何类型的数字，字母，变量，表达式：

String(123)       // 将数字 123 转换为字符串并返回

Number 方法 toString() 也是有同样的效果。

.toString()

## Date

在 [Date 方法](https://www.runoob.com/jsref/jsref-obj-date.html) 章节中，你可以查看更多关于日期转换为字符串的函数：

|  |  |
| --- | --- |
| **方法** | **描述** |
| **getDate()** | 从 Date 对象返回一个月中的某一天 (1 ~ 31)。 |
| **getDay()** | 从 Date 对象返回一周中的某一天 (0 ~ 6)。 |
| getFullYear() | 从 Date 对象以四位数字返回年份。 |
| getHours() | 返回 Date 对象的小时 (0 ~ 23)。 |
| getMilliseconds() | 返回 Date 对象的毫秒(0 ~ 999)。 |
| getMinutes() | 返回 Date 对象的分钟 (0 ~ 59)。 |
| getMonth() | 从 Date 对象返回月份 (0 ~ 11)。 |
| getSeconds() | 返回 Date 对象的秒数 (0 ~ 59)。 |
| getTime() | 返回 1970 年 1 月 1 日至今的毫秒数。 |

## 将字符串转换为数字

全局方法 Number() 可以将字符串转换为数字。

字符串包含数字(如 "3.14") 转换为数字 (如 3.14).

空字符串转换为 0。

其他的字符串会转换为 NaN (不是个数字)。

|  |  |
| --- | --- |
| **方法** | **描述** |
| parseFloat() | 解析一个字符串，并返回一个浮点数。 |
| parseInt() | 解析一个字符串，并返回一个整数。 |

 + 可用于将变量转换为数字：

全局方法 Number() 可将布尔值转换为数字。

Number(false)     // 返回 0  
Number(true)      // 返回 1

## 自动转换类型

5 + null    // 返回 5         null 转换为 0  
"5" + null  // 返回"5null"   null 转换为 "null"  
"5" + 1     // 返回 "51"      1 转换为 "1"    
"5" - 1     // 返回 4         "5" 转换为 5

## 自动转换为字符串

当你尝试输出一个对象或一个变量时 JavaScript 会自动调用变量的 toString() 方法：

document.getElementById("demo").innerHTML = myVar;  
  
myVar = {name:"Fjohn"}  // toString 转换为 "[object Object]"  
myVar = [1,2,3,4]       // toString 转换为 "1,2,3,4"  
myVar = new Date()      // toString 转换为 "Fri Jul 18 2014 09:08:55 GMT+0200"

数字和布尔值也经常相互转换：

myVar = 123             // toString 转换为 "123"  
myVar = true            // toString 转换为 "true"  
myVar = false           // toString 转换为 "false"

## 正则表达式

/runoob/i  是一个正则表达式。

runoob  是一个正则表达式主体 (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()。

**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串。

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：

实例

检索字符串中 "Runoob" 的子串：

var str = "Visit Runoob!";

var n = str.search("Runoob"); 6 包括空格

var txt = str.replace(/microsoft/i,"Runoob"); Visit Runoob!

var txt = str.replace("Microsoft","Runoob");请访问 Runoob!

|  |  |
| --- | --- |
| **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。 |
| **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。 |

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。

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

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。e

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。

try { ... //异常的抛出 } catch(e) { ... //异常的捕获与处理 } finally { ... //结束处理 }

## JavaScript 声明提升

JavaScript 中，**函数及变量的声明**都将被提升到函数的最**顶部**。

JavaScript 中，变量可以在使用后声明，也就是变量可以先使用再声明。

JavaScript **初始化**不会提升

JavaScript 只有声明的变量会提升，初始化的不会。

 y 输出了 undefined，这是因为变量声明 (var y) 提升了，但是初始化(y = 7) 并不会提升，所以 y 变量是一个未定义的变量。

## 为什么使用严格模式:

消除Javascript语法的一些**不合理、不严谨**之处，减少一些怪异行为;

消除代码运行的一些**不安全**之处，保证代码运行的安全；

**提高编译器效率，增加运行速度；**

为未来新版本的Javascript做好铺垫。

"严格模式"体现了Javascript更合理、更安全、更严谨的发展方向，包括IE 10在内的主流浏览器，都已经支持它，许多大项目已经开始全面拥抱它。

比较运算中，=== 为恒等计算符，同时检查表达式的值与类型

for和if是没有作用域的 都是全局变量

## 表单验证

下面的函数用来检查用户是否已填写表单中的必填（或必选）项目。假如必填或必选项为空，那么警告框会弹出，并且函数的返回值为 false，否则函数的返回值则为 true（意味着数据没有问题）：

function validateForm() {

var x=document.forms["myForm"]["fname"].value;

if (x==null || x=="") {

alert("姓必须填写");

return false; } }

以上函数在 form 表单提交时被调用:

**实例**

<form name="myForm" action="demo-form.php" onsubmit="return validateForm()" method="post">

姓: <input type="text" name="fname">

<input type="submit" value="提交">

</form>

## this的指向

在方法中，this 表示该对象方法所属的**对象**。

如果单独使用，this 表示全局对象。全局对象为 [object Window]:

在函数中，this 表示全局对象。

在函数中，在严格模式下，**this 是未定义的(undefined)**。

在事件中，this 表示接收事件的元素。

类似 call() 和 apply() 方法可以将 this 引用到任何对象。

## let和var

let 声明的变量只在 let 命令所在的代码块内有效。

const 声明一个只读的常量，一旦声明，常量的值就不能改变。

在 ES6 之前，JavaScript 只有两种作用域： 全局变量 与 函数内的局部变量。

在第一个实例中，使用了 var 关键字，它声明的变量是全局的，包括循环体内与循环体外。

在第二个实例中，使用 let 关键字， 它声明的变量作用域只在循环体内，循环体外的变量不受影响。

在相同的作用域或块级作用域中，不能使用 let 关键字来重置 var 关键字声明的变量:

在相同的作用域或块级作用域中，不能使用 let 关键字来重置 let 关键字声明的变量:

使用 var 关键字声明的变量在任何地方都可以修改：

在相同的作用域或块级作用域中，不能使用 var 关键字来重置 let 关键字声明的变量:

## 变量提升

JavaScript 中，var 关键字定义的变量可以在使用后声明，也就是变量可以先使用再声明

carName = "Volvo";

var carName;可以 不管有没有初始化(赋值 )都可以访问(赋值undefined) 变量提升直接赋值undefined

let 关键字定义的变量则不可以在使用后声明，也就是变量需要**先声明再使用**。

carName = "Volvo";

let carName; 不可以 没有初始化(赋值)之前不可以被访问(赋值undefined) 变量提升不会自动赋值undefined

const 用于声明一个或多个常量，**声明时必须进行初始化**，且初始化后值不可再修改：

const定义常量与使用let 定义的变量**相似两点**：

二者都是**块级作用域**

都不能和它所在作用域内的其他变量或函数拥有**相同的名称**

两者还有以下**两点区别**：

const声明的常量必须**初始化**，而let声明的变量不用

const 定义常量的值不能通过再赋值**修改**，也不能再次声明。而 let 定义的变量值可以修改。

使用 const 定义的对象或者数组，其实是可变的 可以修改**对象属性 数组元素**

## Js中获取dom元素的方法

document.getElementById('元素的ID')

document.getElementsByClassName(“元素的Class”)

　document.getElementsByTagName('元素的标签名')

　document.querySelector(“”)获取一个#id .class  
　document.querySelectorAll(“”)获取多个#id .class

## 什么是 JSON?

JSON 英文全称 JavaScript Object Notation

JSON 是**一种轻量级的数据交换格式**。

JSON是独立的语言 \*

JSON 易于理解。

JSON 是用于**存储和传输数据**的格式。

JSON 通常用于**服务端向网页传递数据** 。

JSON 数据格式为 **键/值 对**，就像 JavaScript 对象属性。

键/值对包括字段名称（在双引号中），后面一个冒号，然后是值：

JSON 对象保存在大括号内。

就像在 JavaScript 中, 对象可以保存多个 键/值 对：

JSON 数组保存在中括号内。 数组对象

使用 JavaScript 内置函数 **JSON.parse() 将字符串转换为 JavaScript 对象**:

|  |  |
| --- | --- |
| **函数** | **描述** |
| [JSON.parse()](https://www.runoob.com/js/javascript-json-parse.html" \t "https://www.runoob.com/js/_blank) | 用于将一个 JSON 字符串转换为 JavaScript 对象。 |
| [JSON.stringify()](https://www.runoob.com/js/javascript-json-stringify.html" \t "https://www.runoob.com/js/_blank) | 用于将 JavaScript 值转换为 JSON 字符串。 |

## href="#"与href="javascript:void(0)"的区别

# 包含了一个位置信息，默认的**锚是#top 也就是网页的上端**。

而javascript:void(0), 仅仅表示一个**死链接**。

在页面很长的时候会使用**# 来定位页面的具体位置**，格式为：**# + id。**

如果你要定义一个死链接请使用 javascript:void(0) 。

异步（Asynchronous, async）是与同步（Synchronous, sync）

## Promise

JavaScript 中的**异步操作函数往往通过回调函数**来实现异步任务的结果处理

Promise 构造函数只有一个参数，是一个函数，这个函数在**构造之后会直接被异步运行**，所以我们称之为**起始函数**。起始函数包含两个参数 resolve 和 reject。

当 Promise 被构造时，起始函数会被异步执行：

Promise 类有 .then() .catch() 和 .finally() 三个方法，这三个方法的参数都是一个函数，.then() 可以将参数中的函数添加到当前 Promise 的**正常执行**序列，.catch() 则是设定 Promise 的**异常处理**序列，.finally() 是在 Promise 执行的最后一定会执行的序列。 .then() 传入的函数会按顺序依次执行，有任何异常都会直接跳到 catch 序列：

**resolve 和 reject 的作用域只有起始函数**，不包括 then 以及其他序列；

resolve 和 reject 并不能够使起始函数**停止运行**，别忘了 **return**。

异步函数 async function 中可以使用 await 指令**，await 指令后必须跟着一个 Promise**，异步函数会在这个 Promise 运行中**暂停**，直到其运行结束再继续运行。

如果 Promise 有一个正常的返回值，await 语句也会返回它：

promise构造函数是同步执行的，then方法是异步执行的

Promise是同步的立即执行函数

## 代码规范

通常运算符 ( = + - \* / ) 前后需要添加空格:

变量名推荐使用驼峰法来命名(camelCase):

通常使用 4 个空格符号来缩进代码块：

**复杂语句**的通用规则:

将左花括号放在第一行的结尾。

左花括号前添加一空格。

将右花括号独立放在一行。

不要以分号结束一个复杂的声明

**对象**定义的规则:

将左花括号与类名放在同一行。

冒号与属性值间有个空格。

字符串使用双引号，数字不需要。

最后一个属性-值对后面不要添加逗号。

将右花括号独立放在一行，并以分号作为结束符号。

**命名规则**

一般很多代码语言的命名规则都是类似的，例如:

变量和函数为小驼峰法标识, 即除第一个单词之外，其他单词首字母大写（ lowerCamelCase）

全局变量为大写 (UPPERCASE )

常量 (如 PI) 为大写 (UPPERCASE )

变量命名你是否使用这几种规则： hyp-hens, camelCase, 或 under\_scores ?

HTML 和 CSS 的横杠(-)字符:

HTML5 属性可以以 data- (如：data-quantity, data-price) 作为前缀。

CSS 使用 - 来连接**属性名** (font-size)。

**函数**

匿名函数

在函数表达式存储在变量后，变量也可作为一个函数使用：

var x = function () {}

以上函数实际上是一个 匿名函数 (函数没有名称)。

函数存储在变量中，不需要函数名称，通常通过变量名来调用。

具名函数

function x(){}

对象函数

const obj={

x:function(){}

}

构造函数

var myFunction = new Function("a", "b", "return a \* b");  
  
var x = myFunction(4, 3);

也可以写成匿名函数

var myFunction = function (a, b) {return a \* b};  
  
var x = myFunction(4, 3);

## 函数提升（Hoisting）

在之前的教程中我们已经了解了 "hoisting(提升)"。

提升（Hoisting）是 JavaScript 默认将**当前作用域提升到前面去的行为**。

提升（Hoisting）应用在**变量的声明与函数的声明**。

因此，函数可以在声明之前调用：

**let和箭头函数必须先声明再调用 var和普通函数函数可以先调用再声明**

自调用函数

函数表达式可以 "自调用"。

自调用表达式会自动调用。

**如果表达式后面紧跟 () ，则会自动调用。**

不能自调用声明的函数。

通过添加括号，来说明它是一个函数表达式：

实例

(function () {  
    var x = "Hello!!";      // 我将调用自己  
})();

**匿名自我调用的函数 (没有函数名)。**

函数可作为一个值使用

函数是对象 有方法和属性

当我们使用箭头函数的时候，箭头函数会默认帮我们绑定外层 this 的值，所以在箭头函数中 **this 的值和外层的 this 是一样**的。

箭头函数是**不能提升**的，所以需要在使用之前定义。

使用**const 比使用 var 更安全，因为函数表达式始终是一个常量**。

如果函数部分只是一个语句，则可以省略 return 关键字和大括号 {}

## 函数显式参数(Parameters)形参与隐式参数(Arguments)实参

函数显式参数在函数定义时列出。

函数隐式参数在函数调用时传递给函数真正的值。

JavaScript 函数定义显式参数时没有指定**数据类型**。

JavaScript 函数对隐式参数没有进行**类型检测**。

JavaScript 函数对隐式参数的**个数**没有进行检测。

ES5 中如果函数在调用时未提供隐式参数，参数会默认设置为： **undefined**

形参初始化是默认参数会被实参所**替换**

JavaScript 函数有个内置的对象 **arguments 对象**。

argument 对象包含了函数调用的**参数数组**。

通过这种方式你可以很方便的找到最大的一个参数的值：

x = findMax(1, 123, 500, 115, 44, 88);

function findMax() {

var i, max = arguments[0];

if(arguments.length < 2)

return max;

for (i = 0; i < arguments.length; i++) {

if (arguments[i] > max) {

max = arguments[i]; } }

return max; }

或者创建一个函数用来统计所有数值的和：

x = sumAll(1, 123, 500, 115, 44, 88);

function sumAll() {

var i, sum = 0;

for (i = 0; i < arguments.length; i++) {

sum += arguments[i]; }

return sum; }

## 函数调用

fullName 方法是一个函数。函数属于对象。 myObject 是函数的所有者。

this对象，拥有 JavaScript 代码。**实例中 this 的值为 myObject 对象**。

构造函数的调用会**创建一个新的对象**。新对象会继承构造函数的**属性和方法。**

**x就是那个新对象**

function myFunction(arg1, arg2) {

this.firstName = arg1;

this.lastName = arg2; }

//This creates a new object var x = new myFunction("John","Doe");

x.firstName;

call() 和 apply() 是预定义的函数方法。 两个方法可用于调用函数，两个方法的第**一个参数必须是对象本身**。

function myFunction(a, b) {

return a \* b; }

**myObject** = myFunction.call(**myObject**, 10, 2);

两个方法都使用了对象本身作为第一个参数。 两者的**区别在于第二个参数**： apply传入的是一个**参数数组**，也就是将多个参数组合成为一个数组传入，而call则作为call的参数传入（从**第二个参数**开始）。

function myFunction(a, b) {

return a \* b; }

myArray = [10, 2];

myObject = myFunction.apply(myObject, myArray);

## 闭包函数

全局变量的作用域是全局性的，即在整个JavaScript程序中，全局变量处处都在。

而在函数内部声明的变量，只在函数内部起作用。这些变量是局部变量，作用域是局部性的；函数的参数也是局部性的，只在函数内部起作用。

闭包处理

var add = (function () {

var counter = 0;

return function () {

return counter += 1;} })();

add(); add(); add(); // 计数器为 3

非常棒的部分是它可以访问**函数上一层作用域**的计数器。

这个叫作 JavaScript 闭包。它使得函数拥有**私有变量**变成可能

私有变量：类似let的作用

**闭包是一种保护私有变量的机制**，在函数执行时形成私有的作用域，保护里面的私有变量不受外界干扰。

直观的说就是形成一个**不销毁的栈**环境。

**创建对象时会自动调用构造函数方法 constructor()**。

class Runoob {

constructor(name, url) {

this.name = name;

this.url = url; } }

let site = new Runoob("菜鸟教程", "https://www.runoob.com");

构造方法是一种特殊的方法：

构造**方法**名为 constructor()。

函数有function 类->构造方法没有 构造函数有

构造方法在创建新对象时会**自动执行**。

构造方法用于**初始化对象属性**。

如果不定义构造方法，JavaScript 会自动添加一个空的构造方法。

### 继承

JavaScript 类继承使用 **extends** 关键字。

继承允许我们依据另一个类来定义一个类，这使得创建和维护一个应用程序变得更容易。

super() 方法用于调用父类的构造函数。

class Site { constructor(name) { this.sitename = name; } }

class Runoob extends Site { constructor(name, age) { super(name); this.age = age; }}

super() 方法引用父类的构造方法。

通过在构造方法中调用 super() 方法，我们调用了父类的构造方法，这样就可以访问父类的属性和方法。

继承对于代码可复用性很有用。

class Runoob {

constructor(name) {

this.sitename = name; }

get s\_name() {

return this.sitename; }

set s\_name(x) {

this.sitename = x; } }

let noob = new Runoob("菜鸟教程");

document.getElementById("demo").innerHTML = noob.s\_name;

函数声明和类声明之间的一个重要区别在于, 函数声明会提升，类声明不会。

getter 是一种**获得属性值**的方法，setter是一种**设置属性值**的方法

getter负责查询值，它**不带任何参数**，setter则负责设置键值，**值是以参数的形式传递**，在他的函数体中，**一切的return都是无效的**

静态方法是使用 **static** 关键字修饰的方法，又叫**类方法，属于类的**，但不属于对象，在实例化对象之前可以通过 **类名.方法名** 调用**静态方法。**

静态方法不能在**对象**上调用，只能在类中调用。

实例等于创建了新的对象

可以在类中调用 'hello()' 方法

不能通过实例化后的对象调用静态方法

在对象 noob 中使用静态方法，可以作为一个参数传递给它：可以传递实例

class Runoob {

constructor(name) {

this.name = name; }

static hello(x) {

return "Hello " + x.name; } }

let noob = new Runoob("菜鸟教程"); document.getElementById("demo").innerHTML = Runoob.hello(noob);

## DOM事件

### Dom事件

onload 和 onunload 事件会在用户进入或离开页面时被触发。

onload 事件可用于检测访问者的浏览器类型和浏览器版本，并基于这些信息来加载网页的正确版本。

onload 和 onunload 事件可用于处理 cookie。

onchange 事件

onchange 事件常结合对输入字段的验证来使用。

下面是一个如何使用 onchange 的例子。当用户改变输入字段的内容时，会调用 upperCase() 函数。

<input type="text" id="fname" onchange="upperCase()">

输入小写字母后 离开输入框会将小写字母大写

onmouseover 和 onmouseout 事件

onmousedown、onmouseup 以及 onclick 事件

**addEventListener() 方法**

第一个参数是事件的类型 (如 "click" 或 "mousedown").

第二个参数是事件触发后调用的函数。

第三个参数是个布尔值用于描述事件是冒泡还是捕获。该参数是可选的。

addEventListener() 方法允许向同一个元素添加多个事件，且不会覆盖已存在的事件：

在 **冒泡** 中，内部元素的事件会先被触发，然后再触发外部元素，即： <p> 元素的点击事件先触发，然后会触发 <div> 元素的点击事件。

在**捕获** 中，外部元素的事件会先被触发，然后才会触发内部元素的事件，即： <div> 元素的点击事件先触发 ，然后再触发 <p> 元素的点击事件。

**默认值为 false, 即冒泡传递**，当值为 true 时, 事件使用捕获传递。

### 三种点击事件方式

<!-- 第三种方式-->

<button id="btn" onclick="threeFn()">点我</button>

<script type="text/javascript">

var btn = document.getElementById("btn");

// 第一种 通过点击事件 btn.onclick = function(){

alert("这是第一种点击方式");

}

// 第二种 监听点击事件 btn.addEventListener('click', function(){

alert("这是第二中点击方式");

})

// 第三种 通过方法响应点击事件

function threeFn(){

alert("这是第三种点击方式");

}

</script>

## 创建节点

创建新的 HTML 元素 (节点) - **appendChild()**

<script>

var para = document.createElement("p");

var node = document.createTextNode("这是一个新的段落。"); para.appendChild(node);

var element = document.getElementById("div1");

element.appendChild(para); </script>

创建新的 HTML 元素 (节点) - **insertBefore()**

以上的实例我们使用了 appendChild() 方法，它用于**添加新元素到尾部**。

如果我们需要将新**元素添加到开始位置**，可以使用 insertBefore() 方法:

var element = document.getElementById("div1");

var child = document.getElementById("p1");

var para = document.createElement("p");

element.insertBefore(para, child);

para插入到child前面

**移除已存在的元素**

要移除一个元素，你需要知道该元素的父元素。

替换 HTML 元素 - **replaceChild()**

我们可以使用 replaceChild() 方法来替换 HTML DOM 中的元素。

前面的替换到后面的

以下代码选取了文档中所有的 <p> 节点：

var myNodeList = document.querySelectorAll("p");

## Object.prototype

在 JavaScript 中，几乎所有的**对象都是 Object 类型的实例**，它们都会从 **Object.prototype** 继承属性和方法。

Object 构造函数创建一个对象包装器。

Object 构造函数，会根据给定的参数创建对象，具体有以下情况：

如果给定值是 **null 或 undefined**，将会创建并返回一个空对象。

如果传进去的是**一个基本类型的值**，则会构造其包装类型的对象。

如果传进去的是**引用类型的值**，仍然会返回这个值，经他们复制的变量保有和源对象相同的引用地址。

**当以非构造函数形式被调用时，Object 的行为等同于 new Object()。**

**构造对象**

function person(firstname,lastname,age,eyecolor) {

this.firstname=firstname;

this.lastname=lastname;

this.age=age;

this.eyecolor=eyecolor; }

把方法添加到 JavaScript 对象

方法只不过是附加在对象上的函数。

在构造器函数内部定义对象的方法：

所有的 JavaScript **对象都会从一个 prototype（原型对象）**中继承属性和方法。

**prototype 继承**

所有的 JavaScript 对象都会从一个 prototype（原型对象）中继承属性和方法：

Date 对象从 Date.prototype 继承。

Array 对象从 Array.prototype 继承。

Person 对象从 Person.prototype 继承。

所有 JavaScript 中的对象都是位于原型链顶端的 Object 的实例。

JavaScript 对象有一个指向一个原型对象的链。当试图访问一个对象的属性时，它不仅仅在该对象上搜寻，还会搜寻该对象的原型，以及该**对象的原型的原型**，依次层层向上搜索，直到找到一个名字匹配的属性或到达原型链的末尾。

Date 对象, Array 对象, 以及 Person 对象从 Object.prototype 继承。

添加属性和方法

有的时候我们想要在所有已经存在的对象添加新的属性或方法。

另外，有时候我们想要在对象的**构造函数**中**添加属性或方法**。

使用 **prototype 属性**就可以给对象的构造函数添加新的属性：

function Person(first, last, age, eyecolor) {

this.firstName = first;

this.lastName = last;

this.age = age;

this.eyeColor = eyecolor; }

Person.prototype.nationality = "English";

当然我们也可以使用 prototype 属性就可以给对象的构造函数添加新的方法：

## Number对象

默认情况下，JavaScript 数字为十进制显示。

但是你可以使用 toString() 方法 输出16进制、8进制、2进制。

无穷大（Infinity）

NaN 属性是代表**非数字值的特殊值**。该属性用于指示某个值不是数字。可以把 Number 对象设置为该值，来指示其不是数字值。

**数字可以是数字或者对象**

## 字符串（String） 对象

字符串使用 indexOf() 来定位字符串中某一个指定的字符首次出现的位置：

如果没找到对应的字符函数返回-1

lastIndexOf() 方法在字符串末尾开始查找字符串出现的**位置**。

match()函数用来查找字符串中特定的字符，并且如果找到的话，则返回这个**字符**。

replace() 方法在字符串中用某些字符替换另一些**字符**。包括正则表达式字符

字符串大小写转换使用函数 **toUpperCase() / toLowerCase()**:

字符串使用split()函数转为数组:

Javascript 中可以使用反斜线（\）插入特殊符号，如：撇号,引号等其他特殊符号

字符串属性和方法

属性:

length

prototype

constructor

方法:

charAt()

charCodeAt()

concat()

fromCharCode()

indexOf()

lastIndexOf()

match()

replace()

search()

slice()

split()

substr()

substring()

toLowerCase()

toUpperCase()

valueOf()

0

-0

null

""

false

undefined

NaN

那么对象的值为 false。否则，其值为 true（即使当变量值为字符串 "false" 时）！

## 浏览器对象模型 (BOM)

JavaScript 交互性方面的相同方法和属性，因此常被认为是 BOM 的方法和属性。

Window 对象

所有浏览器都支持 window 对象。它表示浏览器窗口。

所有 **JavaScript 全局对象、函数以及变量均**自动成为 window 对象的成员。

全局变量是 window 对象的属性。

全局函数是 window 对象的方法。

**甚至 HTML DOM 的 document 也是 window 对象的属性之一：**

**Bom方法**

window.open() - 打开新窗口

window.close() - 关闭当前窗口

window.moveTo() - 移动当前窗口

window.resizeTo() - 调整当前窗口的尺寸

### Window Location

window.location 对象在编写时可不使用 window 这个前缀。 一些例子：

一些实例:

location.hostname 返回 web 主机的域名

location.pathname 返回当前页面的路径和文件名

location.port 返回 web 主机的端口 （80 或 443）

location.protocol 返回所使用的 web 协议（http: 或 https:）

Window Location Href

location.href 属性返回当前页面的 URL。

Window Location Assign

**location.assign() 方法加载新的文档。**

### Window History

history.back() - 与在浏览器点击后退按钮相同

history.forward() - 与在浏览器中点击向前按钮相同

除此之外可以用 history.go() 这个方法来实现向前，后退的能。

window.navigator 对象包含有关访问者浏览器的信息。

### Window Navigator(导航)

window.navigator 对象在编写时可不使用 window 这个前缀。

alter警告框

confirm确认框 true false

prompt提示框 输入值或者null

\n换行

setInterval() - 间隔指定的毫秒数不停地执行指定的代码。

setTimeout() - 在指定的毫秒数后执行指定代码。

注意: setInterval() 和 setTimeout() 是 HTML DOM Window对象的两个方法

clearInterval() 方法用于停止 setInterval() 方法执行的函数代码。

clearTimeout() 方法用于停止执行setTimeout()方法的函数代码。

## JavaScript 保留关键字

Javascript 的保留关键字不可以用作变量、标签或者函数名。有些保留关键字是作为 Javascript 以后扩展使用。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| abstract | arguments | boolean | break | byte |
| case | catch | char | class\* | const |
| continue | debugger | default | delete | do |
| double | else | enum\* | eval | export\* |
| extends\* | false | final | finally | float |
| for | function | goto | if | implements |
| import\* | in | instanceof | int | interface |
| let | long | native | new | null |
| package | private | protected | public | return |
| short | static | super\* | switch | synchronized |
| this | throw | throws | transient | true |
| try | typeof | var | void | volatile |
| while | with | yield |  |  |

## 什么是 Cookie？

Cookie 用于存储 **web 页面的用户信息**。

Cookie 是一些数据, 存储于你电脑上的文本文件中。

当 web 服务器向浏览器发送 web 页面时，在连接关闭后，服务端不会记录用户的信息。

Cookie 的作用就是用于解决 "如何记录客户端的用户信息":

当用户访问 web 页面时，他的名字可以记录在 cookie 中。

在用户下一次访问该页面时，可以在 cookie 中读取用户访问记录。

Cookie 以名/值对形式存储，如下所示:

username=John Doe

当浏览器从服务器上请求 web 页面时， 属于该页面的 cookie 会被添加到该请求中。服务端通过这种方式来获取用户的信息。

JavaScript 可以使用 **document.cookie** 属性来创建 、读取、及删除 cookie。

您还可以为 cookie **添加一个过期时间**（以 UTC 或 GMT 时间）。默认情况下，cookie 在浏览器关闭时删除：

删除 cookie 非常简单。您只需要设置 expires 参数为以前的时间即可，如下所示，设置为 Thu, 01 Jan 1970 00:00:00 GMT:

# Html

### 基础知识

HyperText Markup Language

<!DOCTYPE>声明有助于浏览器中正确显示网页。

注释： 请始终将正斜杠添加到子文件夹。假如这样书写链接：href="https://www.runoob.com/html"，就会向服务器产生两次 HTTP 请求。这是因为服务器会添加正斜杠到这个地址，然后创建一个新的请求，就像这样：href="https://www.runoob.com/html/"。

META 元素通常用于指定网页的描述，关键词，文件的最后修改时间，作者，和其他元数据。

元数据可以使用于浏览器（如何显示内容或重新加载页面），搜索引擎（关键词），或其他Web服务。

表格的表头使用 <th> 标签进行定义。

大多数浏览器会把表头显示为粗体居中的文本：

自定义列表不仅仅是一列项目，而是项目及其注释的组合。

自定义列表以 <dl> 标签开始。每个自定义列表项以 <dt> 开始。每个自定义列表项的定义以 <dd> 开始。

大多数 HTML 元素被定义为块级元素或内联(行内)元素。

<textarea rows="10" cols="30">

我是一个文本框。

</textarea>

### 表单元素

|  |  |
| --- | --- |
| [<form>](https://www.runoob.com/tags/tag-form.html" \t "https://www.runoob.com/html/_blank) | 定义供用户输入的表单 |
| [<input>](https://www.runoob.com/tags/tag-input.html" \t "https://www.runoob.com/html/_blank) | 定义输入域 |
| [<textarea>](https://www.runoob.com/tags/tag-textarea.html" \t "https://www.runoob.com/html/_blank) | 定义文本域 (一个多行的输入控件) |
| [<label>](https://www.runoob.com/tags/tag-label.html" \t "https://www.runoob.com/html/_blank) | 定义了 <input> 元素的标签，一般为输入标题 |
| [<fieldset>](https://www.runoob.com/tags/tag-fieldset.html" \t "https://www.runoob.com/html/_blank) | 定义了一组相关的表单元素，并使用外框包含起来 |
| [<legend>](https://www.runoob.com/tags/tag-legend.html" \t "https://www.runoob.com/html/_blank) | 定义了 <fieldset> 元素的标题 |
| [<select>](https://www.runoob.com/tags/tag-select.html" \t "https://www.runoob.com/html/_blank) | 定义了下拉选项列表 |
| [<optgroup>](https://www.runoob.com/tags/tag-optgroup.html" \t "https://www.runoob.com/html/_blank) | 定义选项组 |
| [<option>](https://www.runoob.com/tags/tag-option.html" \t "https://www.runoob.com/html/_blank) | 定义下拉列表中的选项 |
| [<button>](https://www.runoob.com/tags/tag-button.html" \t "https://www.runoob.com/html/_blank) | 定义一个点击按钮 |
| [<datalist>](https://www.runoob.com/tags/tag-datalist.html" \t "https://www.runoob.com/html/_blank)**New** | 指定一个预先定义的输入控件选项列表 |
| [<keygen>](https://www.runoob.com/tags/tag-keygen.html" \t "https://www.runoob.com/html/_blank)**New** | 定义了表单的密钥对生成器字段 |
| [<output>](https://www.runoob.com/tags/tag-output.html" \t "https://www.runoob.com/html/_blank)**New** | 定义一个计算结果 |

<select>

<optgroup label="Swedish Cars">

<option value="volvo">Volvo</option>

<option value="saab">Saab</option>

</optgroup>

</select>

**table里面的**

colspan用来指定单元格横向跨越的列数：colspan就是合并列的，colspan=2的话就是合并两列。

rowspan用来指定单元格纵向跨越的行数:rowspan就是用来合并行的，比如rowspan=2就是合并两行

cellpadding：代表单元格边框到内容之间的距离（留白）

cellspacing：用来指定表格各单元格之间的空隙(距离)。此属性的参数值是数字，表示单元格间隙所占的像素点数。

### 颜色的三种表达形式

颜色名

颜色值由十六进制来表示红、绿、蓝（RGB）。

十六进制颜色值:每个颜色的最低值为 0(十六进制为 00)，最高值为 255(十六进制为FF)。

十六进制值的写法为 # 号后跟三个或六个十六进制字符。

三位数表示法为：#RGB，转换为6位数表示为：#RRGGBB。

### <script>和<link>

<script> 元素既可包含**脚本**语句，也可通过 src 属性指向外部脚本文件。定义了客户端脚本

JavaScript 最常用于图片操作、表单验证以及内容动态更新。

<link> 元素定义资源引用地址 href超级引用资源

### url

scheme://host.domain:port/path/filename

Web浏览器通过URL从Web服务器请求页面

scheme - 定义因特网服务的类型。最常见的类型是 http

host - 定义域主机（http 的默认主机是 www）

domain - 定义因特网域名，比如 runoob.com

:port - 定义主机上的端口号（http 的默认端口号是 80）

path - 定义服务器上的路径（如果省略，则文档必须位于网站的根目录中）。

filename - 定义文档/资源的名称

### 标签英文简写

dt definition term 定义术语

dd definition description 定义描述

dl definition list 定义列表

div division 分隔

em emphasized 加重

h1 header1

br break

hr horizontal rule 水平尺

href hypertext reference超级文本引用

i italic

iframe inline frame 内联框架

li list item

p paragraph

pre preformatted

**rel reload 加载**

src source

sub subscripted 下标

sup superscripted 上标

td table data cell

th table hearder cell

tr table row

var variable 变量

### 块级元素和行内样式

块级元素：div，p，form，ul，li，ol，dl，address，hr，menu，table



行内元素：span，strong，em，br，img，input，lable，select，textarea，cite，a，br，i，img，lable，select，sub，sup，strike中划线，u



# Css

## 样式权值

（内联样式）Inline style > （内部样式）Internal style sheet >（外部样式）External style sheet > 浏览器默认样式

（内联样式）Inline style > （内部样式）Internal style sheet >（外部样式）External style sheet > 浏览器默认样式

1. !important

1. 内联样式表的权值最高 1000；

2. ID 选择器的权值为 100

3. Class 类选择器的权值为 10

4. HTML 标签选择器的权值为 1

## border

border-collapse 属性设置表格的边框是否被折叠成一个单一的边框或隔开：

和cellspacing用来指定表格各单元格之间的空隙(距离)为0 效果一样

表格中的文本对齐和垂直对齐属性。

text-align属性设置水平对齐方式，向左，右，或中心

垂直对齐属性设置垂直对齐，比如顶部，底部或中间：

border-style:dotted dashed solid

border:5px solid red;

轮廓（outline）是绘制于元素周围的一条线，位于边框边缘的外围，可起到突出元素的作用。

轮廓（outline）属性指定元素轮廓的样式、颜色和宽度。

## display:none或visibility:hidden optaicy

隐藏一个元素可以通过把display属性设置为"none"，或把visibility属性设置为"hidden"。但是请注意，这两种方法会产生不同的结果。

visibility:hidden可以隐藏某个元素，但隐藏的元素仍需占用与未隐藏之前一样的空间。也就是说，该元素虽然被隐藏了，但仍然会影响布局。

display:none可以隐藏某个元素，且隐藏的元素不会占用任何空间。也就是说，该元素不但被隐藏了，而且该元素原本占用的空间也会从页面布局中消失。

1、使用 display：none隐藏元素，隐藏后该元素不会在页面中占据位置，也不会响应绑定的监听事件

2、使用visibility:hidden隐藏元素，隐藏后该元素会在页面中占据位置，但是不会响应绑定的监听事件

3、使用 optaicy：0隐藏元素，隐藏后该元素会在页面中占据位置，会响应绑定的监听事件

## 定位position

.tooltip .tooltiptext::after {

content: "";

position: absolute;

bottom: 100%;

left: 50%;

margin-left: -5px;

border-width: 5px;

border-style: solid;

border-color: transparent transparent black transparent;

}

向上的小箭头

position 属性的五个值：

[static](https://www.runoob.com/css/css-positioning.html" \l "position-static)

HTML 元素的默认值，即没有定位，遵循正常的文档流对象。

静态定位的元素不会受到 top, bottom, left, right影响。

[relative](https://www.runoob.com/css/css-positioning.html" \l "position-relative)

相对定位元素的定位是相对其正常位置。

移动相对定位元素，但它原本所占的空间不会改变。

相对定位元素经常被用来作为绝对定位元素的容器块。

[fixed](https://www.runoob.com/css/css-positioning.html" \l "position-fixed)

元素的位置相对于浏览器窗口是固定位置。

即使窗口是滚动的它也不会移动：

Fixed定位使元素的位置与文档流无关，因此不占据空间。

Fixed定位的元素和其他元素重叠。

[absolute](https://www.runoob.com/css/css-positioning.html" \l "position-absolute)

绝对定位的元素的位置相对于最近的已定位父元素，如果元素没有已定位的父元素，那么它的位置相对于<html>:

absolute 定位使元素的位置与文档流无关，因此不占据空间。

absolute 定位的元素和其他元素重叠。

[sticky](https://www.runoob.com/css/css-positioning.html" \l "position-sticky)

sticky 英文字面意思是粘，粘贴，所以可以把它称之为粘性定位。

position: sticky; 基于用户的滚动位置来定位。

**粘性定位的元素是依赖于用户的滚动，在 position:relative 与 position:fixed 定位之间切换。**

它的行为就像 position:relative; 而当页面滚动超出目标区域时，它的表现就像 position:fixed;，它会固定在目标位置。

元素定位表现为在跨越特定阈值前为相对定位，之后为固定定位。

这个特定阈值指的是 top, right, bottom 或 left 之一，换言之，指定 top, right, bottom 或 left 四个阈值其中之一，才可使粘性定位生效。否则其行为与相对定位相同。

## overflow

CSS overflow 属性可以控制内容溢出元素框时在对应的元素区间内添加滚动条。

overflow属性有以下值：

|  |  |
| --- | --- |
| **值** | **描述** |
| visible | 默认值。内容不会被修剪，会呈现在元素框之外。 |
| hidden | 内容会被修剪，并且其余内容是不可见的。 |
| scroll | 内容会被修剪，但是浏览器会显示滚动条以便查看其余的内容。 |
| auto | 如果内容被修剪，则浏览器会显示滚动条以便查看其余的内容。 |
| inherit | 规定应该从父元素继承 overflow 属性的值。 |

注意:overflow 属性只工作于指定高度的块元素上。

## 居中和左右对齐

**元素居中对齐**

要水平居中对齐一个元素(如 <div>), 可以使用 margin: auto;。

设置到元素的宽度将防止它溢出到容器的边缘。

元素通过指定宽度，并将两边的空外边距平均分配：

**元素垂直居中**

垂直居中 - 使用 position 和 transform

除了使用 padding 和 line-height 属性外,我们还可以使用 transform 属性来设置垂直居中:

.center {

height: 200px;

position: relative;

border: 3px solid green; }

.center p {

margin: 0;

position: absolute;

top: 50%;

left: 50%;

transform: translate(-50%, -50%); }

**元素左右对齐**

左右对齐 - 使用定位方式

我们可以使用 position: absolute; 属性来对齐元素:

左右对齐 - 使用 float 方式

我们也可以使用 float 属性来对齐元素:

**文本垂直水平居中对齐**

垂直居中对齐 - 使用 padding

CSS 中有很多方式可以实现垂直居中对齐。 一个简单的方式就是头部顶部使用 padding:

如果要水平和垂直都居中，可以使用 padding 和 text-align: center:

垂直居中 - 使用 line-height vertical-align:middle

## CSS组合选择符

CSS组合选择符包括各种简单选择符的组合方式。

在 CSS3 中包含了四种组合方式:

后代选择器(以空格     分隔)

子元素选择器(以大于 > 号分隔）

相邻兄弟选择器（以加号 + 分隔）

普通兄弟选择器（以波浪号 ～ 分隔）

## 伪类和伪元素

**anchor伪类**

在支持 CSS 的浏览器中，链接的不同状态都可以以不同的方式显示

实例

a:link {color:#FF0000;} /\* 未访问的链接 \*/

a:visited {color:#00FF00;} /\* 已访问的链接 \*/

a:hover {color:#FF00FF;} /\* 鼠标划过链接 \*/

a:active {color:#0000FF;} /\* 已选中的链接 \*/

注意： 在CSS定义中，a:hover 必须被置于 a:link 和 a:visited 之后，才是有效的。

注意： 在 CSS 定义中，a:active 必须被置于 a:hover 之后，才是有效的。

注意：伪类的名称不区分大小写。

CSS :first-child 伪类

您可以使用 :first-child 伪类来选择父元素的第一个子元素。

**伪元素**

:after :before :nth-child(n) :disabled :checked :first-child :last-child

list-style-type:none - 移除列表前小标志。一个导航栏并不需要列表标记

移除浏览器的默认设置将边距和填充设置为0

text-decoration: none;

h1 {text-decoration:overline;}

h2 {text-decoration:line-through;}

h3 {text-decoration:underline;}

h4 {text-decoration:blink;}

## 行内、行块、块级

行内元素一般是内容的容器，而块级元素一般是其他容器的容器，行内元素适合显示具体内容，而块级元素适合做布局。

块级元素(block)：独占一行，对宽高的属性值生效；如果不给宽度，块级元素就默认为浏览器的宽度，即就是100%宽。

行内元素(inline)：可以多个标签存在一行，对宽高属性值不生效，完全靠内容撑开宽高。

行内块元素(inline-block)：结合的行内和块级的优点，既可以设置长宽，可以让padding和margin生效，又可以和其他行内元素并排。

## 淡入效果过渡

我们可以使用 CSS3 transition 属性及 opacity 属性来实现提示工具的淡入效果:

淡入效果：

.tooltip .tooltiptext {

opacity: 0;

transition: opacity 1s; }

.tooltip:hover .tooltiptext {

opacity: 1; }

## 属性选择器

下面的例子是把包含标题（title）的所有元素变为蓝色：

[title] { color:blue; }

属性和值选择器

下面的实例改变了标题title='runoob'元素的边框样式:

[title=runoob] {

border:5px solid green; }

属性和值的选择器 - 多值

下面是包含指定值的title属性的元素样式的例子，使用（~）分隔属性和值:

[title~=hello] { color:blue; }

下面是包含指定值的lang属性的元素样式的例子，使用（|）分隔属性和值:

input[type="text"]

input[type=text] {

background-color: white;

background-image: url('searchicon.png');

background-position: 10px 10px;

background-repeat: no-repeat;

padding-left: 40px; }

box-sizing: border-box;

transition: width 0.4s ease-in-out;

resize: none;不设置为none textarea文本框右下角可以拖动

border: none;

cursor: pointer;

## CSS 计数器

CSS 计数器使用到以下几个属性：

counter-reset - 创建或者重置计数器

counter-increment - 递增变量

content - 插入生成的内容

counter() 或 counters() 函数 - 将计数器的值添加到元素

body {

counter-reset: section;

}

h2::before {

counter-increment: section;

content: "Section " counter(section) ": ";

}

什么是 !important

CSS 中的 !important 规则用于增加样式的权重。

!important 与优先级无关，但它与最终的结果直接相关，使用一个 !important 规则时，此声明将覆盖任何其他声明。

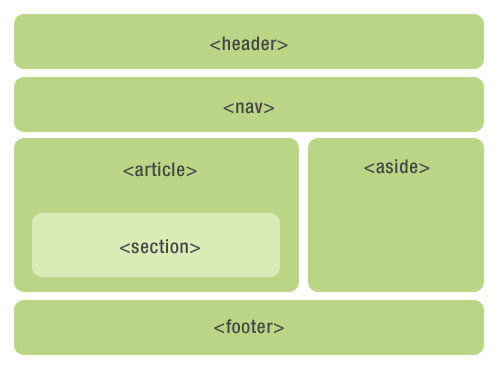
# Html5

## Html5新特性

### 语义标签

header、footer、nav、section、article、aside、details、summary、dialog

、command、mark、main、figure



### 增强型表单属性input类型

color、date、datetime、datetime-local、email、month、number、range、search、tel、time、url、week

placehoder 属性，简短的提示在用户输入值前会显示在输入域上。即我们常见的输入框默认提示，在用户输入后消失。

required  属性，是一个 boolean 属性。要求填写的输入域不能为空

pattern 属性，描述了一个正则表达式用于验证<input> 元素的值。

min 和 max 属性，设置元素最小值与最大值。

step 属性，为输入域规定合法的数字间隔。

height 和 width 属性，用于 image 类型的 <input> 标签的图像高度和宽度。

autofocus 属性，是一个 boolean 属性。规定在页面加载时，域自动地获得焦点。

multiple 属性 ，是一个 boolean 属性。规定<input> 元素中可选择多个值。

### 新表单元素

|  |  |
| --- | --- |
| **标签** | **描述** |
| [<datalist>](https://www.runoob.com/tags/tag-datalist.html) | 定义选项列表。请与 input 元素配合使用该元素，来定义 input 可能的值。 |
| [<keygen>](https://www.runoob.com/tags/tag-keygen.html) | 规定用于表单的密钥对生成器字段。 |
| [<output>](https://www.runoob.com/tags/tag-output.html) | 定义不同类型的输出，比如脚本的输出。 |

### 视频和音频

audio、video controls source

### canvas-图形

Canvas 二维网格

moveTo(x,y) 定义线条开始坐标

lineTo(x,y) 定义线条结束坐标

<canvas></canvas>

var c=document.getElementById("myCanvas");

var ctx=c.getContext("2d");

ctx.moveTo(0,0);

ctx.lineTo(200,100);

ctx.stroke(); //控制空心的文本

arc(x,y,r,start,stop) 画圆

ctx.beginPath(); ctx.arc(95,50,40,0,2\*Math.PI);

**字体图形**

font - 定义字体

fillText(text,x,y) - 在 canvas 上绘制实心的文本

strokeText(text,x,y) - 在 canvas 上绘制空心的文本

### Canvas - 渐变

渐变可以填充在矩形, 圆形, 线条, 文本等等, 各种形状可以自己定义不同的颜色。

以下有两种不同的方式来设置Canvas渐变：

createLinearGradient(x,y,x1,y1) - 创建线条渐变

createRadialGradient(x,y,r,x1,y1,r1) - 创建一个径向/圆渐变

当我们使用渐变对象，必须使用两种或两种以上的停止颜色。

addColorStop()方法指定颜色停止，参数使用坐标来描述，可以是0至1.

使用渐变，设置fillStyle或strokeStyle的值为 渐变，然后绘制形状，如矩形，文本，或一条线。

使用 createLinearGradient():

线条渐变：

var c=document.getElementById("myCanvas");

var ctx=c.getContext("2d"); // 创建渐变

var grd=ctx.createLinearGradient(0,0,200,0);

grd.addColorStop(0,"red");

grd.addColorStop(1,"white"); // 填充渐变

ctx.fillStyle=grd;

ctx.fillRect(10,10,150,80);

### svg

拖放API dragstart、drag、dragend

loaclStorage、sessionStorage

Cookie- 有时间限制的数据存储，会过期

localStorage - 没有时间限制的数据存储

sessionStorage - 针对一个 session 的数据存储, 当用户关闭浏览器窗口后，数据会被删除。

### onmessage事件

当通过事件源接收到消息时，会发生 onmessage 事件。

onmessage 事件的 event 对象支持以下属性：

data - 包含实际的消息

origin - 调用事件的文档的 URL

lastEventId - 在事件流中看到的最后一条消息的标识符

### 应用程序缓存

如需启用应用程序缓存，请在文档的<html> 标签中包含 manifest 属性：

<!DOCTYPE HTML>  
<html manifest="demo.appcache">  
...  
</html>

每个指定了 manifest 的页面在用户对其访问时都会被缓存。如果未指定 manifest 属性，则页面不会被缓存（除非在 manifest 文件中直接指定了该页面）。

manifest 文件的建议的文件扩展名是：".appcache"。

IMG_256 请注意，manifest 文件需要配置正确的 MIME-type，即 "text/cache-manifest"。必须在 web 服务器上进行配置。

# Css3

## 97、Css3新特性

### Css3选择器

.class、#id、\*、:link、:active、:hover、:visited、:first-child、

:before、:after、:nth-child、:enabled、:disabled、:checked、:read-only、:required

### 边框

border-image border-radius

box-shadow

box-shadow: x y blur spread color inset;

### 背景

background

background-clip 背景剪裁属性是从指定位置开始绘制。

background-origin 属性指定了背景图像的位置区域。

border-box padding-box content-box

background-size

### 渐变

线性渐变和径向渐变

radial-gradient linear-gradient

background-image: linear-gradient(to right, red , yellow);

### 文本效果

text-outline text-shadow text-break text-justify

**text-wrap**

|  |  |
| --- | --- |
| normal | 只在允许的换行点进行换行。 |
| none | 不换行。元素无法容纳的文本会溢出。 |

white-space:nowrap;

text-overflow

|  |  |  |
| --- | --- | --- |
| **值** | **描述** |  |
| normal | 默认。空白会被浏览器忽略。 |  |
| pre | 空白会被浏览器保留。其行为方式类似 HTML 中的 <pre> 标签。 |  |
| nowrap | 文本不会换行，文本会在在同一行上继续，直到遇到 <br> 标签为止。 |  |
| pre-wrap | 保留空白符序列，但是正常地进行换行。 |  |
| pre-line | 合并空白符序列，但是保留换行符。 |  |
| inherit | 规定应该从父元素继承 white-space 属性的值。 |  |

div 使用 "text-overflow:ellipsis":省略号

div 使用 "text-overflow:clip":剪切

div 使用自定义字符串 "text-overflow: >>"剪切

### 字体

@font-face

### 转换和变形

transform-origin 允许您更改转化元素位置

transform:translate平移 scale缩放 rotate旋转 skew倾斜

transform: translate(50px,100px);

### 过渡

transition

transition-property

transition-duration

transition-delay

transiton-timing-function 默认为ease

|  |  |
| --- | --- |
| **值** | **描述** |
| linear | 规定以相同速度开始至结束的过渡效果（等于 cubic-bezier(0,0,1,1)）。 |
| ease | 规定慢速开始，然后变快，然后慢速结束的过渡效果（cubic-bezier(0.25,0.1,0.25,1)）。 |
| ease-in | 规定以慢速开始的过渡效果（等于 cubic-bezier(0.42,0,1,1)）。 |
| ease-out | 规定以慢速结束的过渡效果（等于 cubic-bezier(0,0,0.58,1)）。 |
| ease-in-out | 规定以慢速开始和结束的过渡效果（等于 cubic-bezier(0.42,0,0.58,1)）。 |

transition: width 1s linear 2s;

### 动画

@keyframes

改变背景色和位置：

@keyframes myfirst {

0% {background: red; left:0px; top:0px;}

25% {background: yellow; left:200px; top:0px;}

50% {background: blue; left:200px; top:200px;}

75% {background: green; left:0px; top:200px;}

100% {background: red; left:0px; top:0px;} }

animation

animation-name

animation-duration

animation-timing-function

animation-delay

animation-iteration-count 播放动画的次数 默认为一

animation-direction

animation-play-state

animation-name: myfirst;

animation-duration: 5s;

animation-timing-function: linear;

animation-delay: 2s;

animation-iteration-count: infinite; 无数次

animation-direction: alternate; 奇数135偶数246播放

animation-play-state: running;

animation: myfirst 5s linear 2s infinite alternate;

### CSS3 盒模型

 在 CSS3 中, 增加了一些新的用户界面特性来调整元素尺寸，框尺寸和外边框，主要包括以下用户界面属性：

resize：none | both | horizontal | vertical | inherit

box-sizing: content-box | border-box | inherit

outline:outline-color outline-style outline-width outine-offset

resize属性指定一个元素是否应该由用户去调整大小。

box-sizing 属性允许您以确切的方式定义适应某个区域的具体内容。

outline-offset 属性对轮廓进行偏移，并在超出边框边缘的位置绘制轮廓。

### CSS3伸缩布局盒模型(弹性盒)

 CSS3 弹性盒（ Flexible Box 或 flexbox），是一种当页面需要适应不同的屏幕大小以及设备类型时确保元素拥有恰当的行为的布局方式。

引入弹性盒布局模型的目的是提供一种更加有效的方式来对一个容器中的子元素进行排列、对齐和分配空白空间。

 下表列出了在弹性盒子中常用到的属性:

|  |  |
| --- | --- |
| [display](http://www.runoob.com/cssref/pr-class-display.html" \t "https://blog.csdn.net/chandoudeyuyi/article/details/_blank) | 指定 HTML 元素盒子类型。 |
| [flex-direction](http://www.runoob.com/cssref/css3-pr-flex-direction.html" \t "https://blog.csdn.net/chandoudeyuyi/article/details/_blank) | 指定了弹性容器中子元素的排列方式 |
| [justify-content](http://www.runoob.com/cssref/css3-pr-justify-content.html" \t "https://blog.csdn.net/chandoudeyuyi/article/details/_blank) | 设置弹性盒子元素在主轴（横轴）方向上的对齐方式。 |
| [align-items](http://www.runoob.com/cssref/css3-pr-align-items.html" \t "https://blog.csdn.net/chandoudeyuyi/article/details/_blank) | 设置弹性盒子元素在侧轴（纵轴）方向上的对齐方式。 |
| [flex-wrap](http://www.runoob.com/cssref/css3-pr-flex-wrap.html" \t "https://blog.csdn.net/chandoudeyuyi/article/details/_blank) | 设置弹性盒子的子元素超出父容器时是否换行。 |
| [align-content](http://www.runoob.com/cssref/css3-pr-align-content.html" \t "https://blog.csdn.net/chandoudeyuyi/article/details/_blank) | 修改 flex-wrap 属性的行为，类似 align-items, 但不是设置子元素对齐，而是设置行对齐 |
| [flex-flow](http://www.runoob.com/cssref/css3-pr-flex-flow.html" \t "https://blog.csdn.net/chandoudeyuyi/article/details/_blank) | flex-direction 和 flex-wrap 的简写 |
| [order](http://www.runoob.com/cssref/css3-pr-order.html" \t "https://blog.csdn.net/chandoudeyuyi/article/details/_blank) | 设置弹性盒子的子元素排列顺序。 |
| [align-self](http://www.runoob.com/cssref/css3-pr-align-self.html" \t "https://blog.csdn.net/chandoudeyuyi/article/details/_blank) | 在弹性子元素上使用。覆盖容器的 align-items 属性。 |
| [flex](http://www.runoob.com/cssref/css3-pr-flex.html" \t "https://blog.csdn.net/chandoudeyuyi/article/details/_blank) | 设置弹性盒子的子元素如何分配空间。 |