JS基础

JS三部分

ECMAScript：规定了JS中的一些基础核心的知识
DOM：提供了一些操作页面中元素的属性和方法
BOM：提供了一些操作浏览器的属性和方法

JS中的输出方式

1. `console.log()`：控制台输出，不会转换数据类型，想输出什么格式的数据都可以
2. `console.dir(box)`：比log输出的更加详细一些
3. `document.write()`：页面中打印输出内容
4. `alert()`：弹出一个确定按钮的提示框，点击确定提示框消失；`默认调用toString()方法`转换成字符串；阻塞代码执行
5. `conform()`：在alert的基础上，提供确定和取消两个按钮（点击确定，返回true，点击取消，返回false）
6. `prompt()`：在conform的基础上，增加输入的效果，点击取消按钮，返回`null`；点击确定按钮，会获取到输入的内容（如果没有输入任何内容，获取到的是`''`）

JS命名规范

- 遵循`驼峰命名法`
- 可以使用数字、字母、下划线和$进行命名，数字不能放在开头，$xxx一般都是jQuery获取到的，_xxx一般是全局或者公有的变量
- JS中`很多词都有特殊含义，叫做“关键字”`，现在不是关键字，`以后可能会作为关键字的叫做“保留字”`，关键字和保留字都不能随便用来命名。

浏览器内核

浏览器中识别代码、绘制页面的东西叫做浏览器内核（渲染引擎）
前端开发常用的浏览器

- 谷歌浏览器(Chrome)：Webkit内核(V8引擎)，使用该引擎的还有：safari、大部分国产浏览器（百度、搜狗、猎豹、QQ、uc、360……）、安卓和IOS大部分手机浏览器
- 火狐浏览器(Firefox): Gecko内核
- 欧朋浏览器(Opera浏览器)：Presto内核
- IE浏览器：Trident内核

JS数据类型转换

对象数据类型： 普通的对象、数组 、正则 、Date的实例、Math…
parseInt: `把字符串内容从左到右依次识别，直到遇到一个非有效数字为止`，把找到的数字返回；
如果第一个字符是非有效数字，那么直接返回NaN；

```
console.log(Number(undefined));// NaN
console.log(Number(null)) //0

typeof 10 + true + [] + null +
undefined+{};
// "numbertruenullundefined[object Object]"
```

`toFixed()`：保留小数点位数的方法，返回值是一个`字符串`

```
(2).toFixed(2)//"2.00"
```

转换为number类型

- isNaN、Number、parseInt、parseFloat和进行加减乘除数学运算时会自动转换为number类型

加法的特殊性：在遇到字符串的时候是字符串拼接，其他时候是加法运算。
字符串拼接：先调用`toString()`方法把其他类型的值转换为字符串然后再拼接
其他数据类型的toString是直接的把值用单（双）引号包起来即可，只有对象的有特殊性，`对象.toString() == '[object Object]'`

转换为布尔类型

- Boolean()、!!和条件判断的时候，会自动转换为bool类型

只有`0、""、NaN 、null 、undefined`这五个值转布尔值是false；`其他的都是ture`

```
Boolean([])     //ture
Boolean({})     //true
Boolean("3px")  //true
```

- `在进行==比较的时候，如果两边数据类型不相同，浏览器会把等号两边默认转换为数字类型，然后再比较`，===不会这样

对象和对象：比较的是空间地址

```
[]==[];     //false
```

对象和数字：把对象转换为数字再比较

```
[]==0       //ture
({})==NaN   //false NaN和自己不相等,和其他任何值都不相等
```

对象和字符串：把两边都转换为数字再比较
对象和布尔：把两边都转换为数字再比较
字符串和数字：字符串转数字

字符串和布尔：把两边都转换为数字再比较
布尔和数字：布尔转数字
总结：`用 == 比较的时候，左右两边数据类型不一样，浏览器会把两边的类型都转换为数字然后再比较，null和undefined除外`

```
null == undefined            //ture
null === undefined           //false
undefined == 'undefined'     //false
null =='null'                //false
//undefined和null与其他任何值都不相等
```

`数字 + 对象`，先把对象转换为字符串，然后进行`字符串的拼接`

```
1 + {}   //'1[object Object]'
1 + [1]  // '11'
{} + 1   // 1，以花括号开头的表达式，解释运行的时候会将{}看做一个空代码块，所以真正运行的是 +1
```

`+ val`：和`Number(val)`效果一样，是将val转成number类型

```
+{}  //NaN
+[]  //0
```

数组的删除

```
var arr = ['a', 'b', 'c', 'd'];
delete arr[0]; //true
console.log(arr);
arr[0]  //undefined
//[empty, 'b', 'c', 'd']，通过delete删除，数组的length是不变的
```

undefined的几种情况

1. 变量只声明，未定义，默认存储值是undefined
2. 变量提升阶段，带var的默认为undefined
3. 获取对象的一个不存在的属性名，属性值默认undefined
4. 数组中某一项没有值，获取的结果为undefined
5. 函数有形参未传值，形参默认是undefined
6. 函数没有**return**，或**return**后面什么都没有，那么函数默认的返回undefined
7. JS严格模式下，没有明确的主体，**this**默认指向undefined
8. 数组的find方法，没有找到的情况下是undefined

null的几种情况

1. 手动给变量或者对象的某一个属性值设置为null(相当于初始化，一般会在后面的代码中进行赋值)
2. DOM元素获取中，如果没有获取到指定的元素，结果一般为null
3. 正则捕获中，没有捕获到结果，默认为null
4. Object.prototype.__proto__的值为null
5. arguments.callee.caller:当前函数在全局作用域下执行，结果是null;
6. 元素节点和文本节点的nodeValue

对象几点注意

```
1. var obj={
    1:2
    2a:11 //语法错误：属性名不能以数字开头；但是如果只有数字，则可以作为属性名
};

2. 如果属性名是一个变量,只能用中括号的方式添加
3. 如果属性名是一个数字,只能用中括号的方式添加，obj[1]或者obj['1']//引号可以不写
4. 当存储的属性名不是字符串也不是数字的时候，浏览器会把这个值转换为字符串（toStrin
g），然后再进行存储

obj[{}]=300;  //先把({}).toString()后的结果作为对象的属性名存储进来 obj['[objec
t Object]']=300，获取的时候也是先把对象转换为字符串'[object Object]'
```

数据类型检测

```
1. typeof
2. instanceof:检测当前实例是否属于这个类
3. constructor：获取当前实例的构造器
4. Object.prototype.toString.call()：获取当前实例的所属信息
```

for...in 与 hasOwnProperty( )

在使用**for···in**对对象进行遍历时，大部分浏览器都是`先把对象中的键值对进行排序`（把数字属性名放在前面，升序；其次把非数字属性名按照之前编写的顺序排列；小数算作字符不算数字），循环的时候按照重新排列的顺序依次遍历。**for...in**循环在循环对象的属性时也会遍历'原型链'：

```
for (var key in foo) {
    if (foo.hasOwnProperty(key)) {
        console.log(key);
    }
}
```

```
&& 和 ||
var a = val1 || val2;   //如果val1为真就将val1赋值给a否则就将val2赋值给a
var b = val1 && val2    //如果val1为真就将val2赋值给b否则就将val1赋值给b
```

函数

函数的形参本质是一个变量；实参的本质是一个值

```
常用'三元运算符'实现简单的if···else逻辑，即：
typeof num2 === 'undefined' ? num2 = 0:null;
还有一种更简单的写法：mum2 = num2 || 0
```

arguments: 实参集合

```
arguments.callee:存储的是当前函数本身
arguments.caller:存储的是当前函数的执行主体
arguments.callee.caller:存储的是当前函数的宿主函数（在哪里执行的），在全局作用域下执行的，结果是null；
//在JS严格模式下不允许使用
```

形参跟随：在非严格模式下下，改变形参arguments也会对应改变，改变arguments形参也会对应改变

```
var arr = [1,2];
function fn4(a){
    a[0] = 2;
    arguments[0] = [];
    a[0] = 3;
    console.log(a);
}
fn4(arr);//[3]
console.log(arr);//[2,2]
```

匿名函数

1. 函数表达式

```
函数表达式：把一个函数作为值赋值给一个变量或者元素的某个事件。
```

```
var f = function () {
        console.log(1)
    }
f();
```

```
var fn = function sum() {
    console.log(1, sum);
};
fn();//=>1 ƒ sum() {console.log(1, sum);//=>匿名函数起的名字只能在函数里面使用
sum();//=>Uncaught ReferenceError: sum is not defined
```

2. 自执行函数：函数的创建和执行放在一起了，创建之后立马执行。

```
;(function (m,n) {
    console.log(1)
    console.log(m);
    console.log(n);
})();
符号只是控制语法规则，前面的符号还可以是+ - ~ ！,也可以不写
```

内置类原型扩展实现:(3).plus(2).minus(1)

```
~function (pro) {
    pro.plus = function plus(val) {
        return this + Number(val);
    };
    pro.minus = function minus(val) {
        return this - Number(val);
    };
}(Number.prototype);
(3).plus(2).minus(1);//4
```

数组常用方法

17个

```
push pop unshift shift
slice splice indexOf lastIndexOf
sort reverse concat join
map forEach includes filter
toString
```

1. 方法作用
2. 方法的参数
3. 方法返回值
4. 原有的数组是否发生改变;

1. push

```
1)：向数组末尾新增一项；
2)：参数是新增的那一项;可以传多个；
3)：新数组的数组成员的个数；
4)：原有数组发生改变；
```

2. pop

```
1）：删除数组的最后一项
2）：不需要传参数
3）：被删除的那一项
4）：原有数组发生改变；
```

3. unshift

```
1)：向数组开头新增一项；
2)：需要传参数
3)：新数组的数组成员个数
4)：原有数组发生改变；
```

4. shift

```
1)：删除数组的第一项
2)：不需要传参
3)：被删除的那一项
4)：原有数组的发生改变；
```

5. slice

```
1）：数组的截取
2）：slice(m,n)：从数组索引m开始，截取到索引n，但是不包含n
    slice(m) ：从索引m开始，截取到末尾；
    // 索引负数：  让当前length+负数；
3）：返回值是截取的数组
4）：原有数组不发生改变；
//slice() / slice(0) 数组克隆
```

6. splice

```
1）：删除数组中的某几项
2）：splice(m,n)：从索引开始，删除n个
     splice(m) ：从索引m开始删除到末尾；
     splice(m,x,n);替换,从索引m开始，删除x个，用n替换；
3）：返回值；删除那几项，并且以数组返回
```

```
4) : 原有数组发生改变;
//splice(0) 清空数组, 返回一个和原来一样的数组
//splice()  返回一个新的空数组
```

7. sort

```
1) : 数组排序
2) : 参数
   1.sort() : 只能排数组各项是相同位数的数组
   2.sort(function(a,b){return a-b}) 升序
   3.sort(function(a,b){return b-a}) 降序
3) : 排序之后的数组
4) : 原有数组发生改变
```

8. reverse

```
1): 将数组颠倒过来
2): 不需要传参数
3): 数组成员顺序倒过来之后的数组
4): 原有数组发生改变;
```

9. concat

```
1) : 数组的拼接
2) :
   1.不传参数:  concat() //什么都没有拼接, 克隆一个新数组
   2.传参数, (数组/每一项) ; 把传入的实参拼接d到新的数组中;
3) : 拼接之后的新数组
4) : 原有数组不发生改变;
```

10. join

```
1): 把数组成员按照特定的字符连接成一个字符串;
2):
   1.不传参数,会默认按照逗号分开//[1,2,3].join() => '1,2,3', 和toString()方法效果相同
   2.传参数, (特定的字符)
3): 拼接之后的字符串
4):原有数组不发生改变
```

11. indexOf

```
1): 检测数组成员在数组中第一次出现的索引位置; 判断当前项是否在数组中存在; 如果不存在, 返回-1;
2): 需要参数
```

3)：返回在数组中第一次出现的索引；
4)：原有数组不发生改变；

12. lastIndexOf

1)：检测数组成员在数组中最后一次出现的索引位置；判断当前项是否在数组中存在；如果不存在，返回-1；
2)：需要参数
3)：返回在数组中最后一次出现的索引；
4)：原有数组不发生改变；

13. map

1）：遍历数组和映射
2）：需要参数
3）：映射之后的数组
4）：原有数组可以发生改变（**return** xxx;）

14. forEach

1）：遍历数组；
2）：需要参数
3）：返回值是**undefined**
4）：原有数组不发生改变；

15. toString

1）：转字符串
2）：不需要参数
3）：返回一个去了中括号之后的字符串
4）：原有数组不变；

16. includes

```
arr.includes(searchElement, fromIndex)
//从索引fromIndex开始往后查找，有没有searchElement
//如果有返回true，如果没有返回false
```

数组复制：

```
1. ary.slice()
2. ary.slice(0)
3. ary.concat()
```

字符串常用方法

字符串有索引，从0开始，有一个`length属性`存储了字符串的长度。

```
charAt charCodeAt substr substring
slice split indexOf lastIndexOf
toUpperCase toLowerCase concat trim
replace match search valueOf
localCompare
```

1.charAt: 通过索引获取对应的字符

```
charAt和str[索引]的区别?
当指定的索引不存在时:
str[索引]获取的结果是undefined
charAt获取到的结果是空字符串''
```

2.charCodeAt : 通过索引获取对应字符的Unicode编码（`对应ASCII码表`）

```
0-9 : 48-57;
A-Z : 65-90;
a-z : 97--122;
```

3.substr :

```
substr(m,n): 从索引m开始，截取n个;
如果第一个参数是负数，表示倒数计算的字符位置。如果第二个参数是负数，将被自动转为0，因此会返回空字符串。
'JavaScript'.substr(-6) // "Script"
'JavaScript'.substr(4, -1) // ""
```

4.substring:

```
从索引m开始，截取到索引n;不包含n;它与slice作用相同，但有一些奇怪的规则，因此不建议使用这个方法，优先使用slice
//如果第二个参数大于第一个参数，substring方法会自动更换两个参数的位置
'JavaScript'.substring(10, 4) // "Script"
// 等同于
'JavaScript'.substring(4, 10) // "Script"
//如果参数是负数，substring方法会自动将负数转为0。
'Javascript'.substring(-3) // "JavaScript"
'JavaScript'.substring(4, -3) // "Java"
```

5.slice :

slice(m,n): 从原字符串取出子字符串并返回，不改变原字符串

```
'JavaScript'.slice(2, 1) // "" 如果第一个参数大于第二个参数，slice方法返回一个空字符串
'JavaScript'.slice(-6) // "Script"
'JavaScript'.slice(0, -6) // "Java"
```

上述三个实现字符串截取的方法中，如果一个参数都不传递，相当于字符串的克隆

6.split :

将字符串按照特定的字符分隔成数组中的每一项，和数组的join方法是对应的

```
var ary = [a,b,c,d];

str = 'a+b+c+d'
str.split('') // ["a", "+", "b", "+", "c", "+", "d"],如果分割规则为空字符串，则返回数组的成员是原字符串的每一个字符。
str.split() // ['a+b+c+d'], 省略参数，此时返回数组的唯一成员就是原字符串
'a||c'.split('|') // ['a', '', 'c'], 如果满足分割规则的两个部分紧邻着（即中间没有其他字符），则返回数组之中会有一个空字符串。
'a||||c'.split('|') //["a", "", "", "", "c"]
'|b|c'.split('|') // ["", "b", "c"]
'a|b|'.split('|') // ["a", "b", ""]
```

```
7.toUpperCase: 将小写字母全部转大写字母
8.toLowerCase: 将大写字母全部转换成小写字母
//它们都返回一个新字符串，不改变原字符串

//这个方法也可以将布尔值或数组转为大写字符串，但是需要通过call方法使用。
String.prototype.toUpperCase.call(true)
// 'TRUE'
String.prototype.toUpperCase.call(['a', 'b', 'c'])
// 'A,B,C'
String.prototype.toUpperCase.call([a, b, c])
//Uncaught ReferenceError: a is not defined
```

```
9.indexOf ： 检测当前字符在字符串中第一次出现的索引位置；如果不存在，返回-1；
10.lastIndexOf ： 检测的是最后一次出现的索引位置；如果不存在，返回-1；
//它们还可以接受第二个参数，对于indexOf方法，第二个参数表示从该位置开始向后匹配；对于lastIndexOf，第二个参数表示从该位置起向前匹配。
'hello world'.indexOf('o', 6) // 7
'hello world'.lastIndexOf('o', 6) // 4
```

```
11.concat: 用于拼接两个字符串，返回一个新字符串，不改变原字符串,该方法可以接受多个参数，如果参数不是字符串，concat方法会将其先转为字符串，然后再拼接。
var s1 = 'abc';
var s2 = 'def';
```

```
s1.concat(s2) // "abcdef"
s1 // "abc"
'a'.concat('b', 'c') // "abc"

var one = 1;
var two = 2;
''.concat(one, two) // "12"
```

12.trim : trim方法用于去除字符串两端的空格，返回一个新字符串，不改变原字符串
trimLeft ： 去掉字符串中左边的空格
trimRight ： 去掉字符串右边的空格

```
//该方法去除的不仅是空格，还包括制表符（\t、\v）、换行符（\n）和回车符（\r）。
'\r\nabc \t'.trim() // 'abc'
```

13.replace :实现字符的替换，"原有字符串不发生变化"

```
replace(oldStr,newStr);
replace(regExp,newStr);
'aaa'.replace('a', 'b') // "baa"
//执行一次replace只能替换一次，如果有好几个都需要替换，在不使用正则的情况下，我们需要执行很多次replace，真实项目中replace一般都和正则搭配使用。
```

14. match: 用于检测原字符串是否匹配某个子字符串，返回一个数组，成员为匹配的第一个字符串。如果没有找到匹配，则返回**null**

```
'cat, bat, sat, fat'.match('at')
//返回一个数组：["at", index: 1, input: "cat, bat, sat, fat", groups: undefined]
```

返回数组的index属性和input属性，分别表示匹配字符串开始的索引和原始字符串。

15. localeCompare() 用于比较两个字符串，返回一个整数

```
strA.localCompare(strB)
//strA < strB, 返回-1
//strA = strB, 返回0
//strA > strB, 返回1
```

该方法的最大特点，就是会"考虑自然语言的顺序"

```
'1'.localeCompare('a') //-1
'1'.localeCompare('2') //-1
'a'.localeCompare('b') //-1
'B'.localeCompare('a') // 1
'B'.localeCompare('b') // 1
'B'.localeCompare('C') // -1
//规律：数字、字母的顺序：0-9a-zA-Z
```

16. search():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

```
//search方法的用法等同于match，但是返回值为匹配的第一个位置。如果没有找到匹配，则返
```

```
回-1
'cat, bat, sat, fat'.search('bat') //5
'cat, bat, sat, fat'.search('ta') // -1
search方法还可以使用正则表达式作为参数

17. valueOf(): 返回某个字符串对象的原始值

String.fromCharCode(十进制的unicode值):把值按照ACSII码表，转换为原来的信息，和charCodeAt正好对应。
//String.fromCharCode(104, 101, 108, 108, 111)
// "hello"
```

DOM获取元素

```
1.document.getElementById
//上下文只能是document,ID重复了获取第一个
//IE6~7中会把表单元素的name当做id使用

2.document.getElementsByTagName //获取到的结果是一个类数组集合

3.document.getElementsByClassName    //类数组集合，IE6-8不兼容

4.document.getElementsByName     //上下文也只能是document
//在IE浏览器中只对表单元素的name起作用，所以这个方法一般都是用来操作表单元素的

5.document.documentElement 获取当前的HTML标签
6.document.body  获取页面的body标签
7.document.head 获取head标签

8.document.querySelector() //通过css选择器获取"一个"DOM对象，如果有多个符合条件的元素只获取第一个
document.querySelector("p");     //获取文档中第一个 <p> 元素
9.document.querySelectorAll();  //通过css选择器获取`所有`符合条件的DOM元素
//这两个方法IE6-8不兼容，多用于移动端的开发,没有DOM映射
```

DOM节点

node：节点，浏览器认为在一个HTML页面中的所有内容都是节点（包括`文档、元素、属性、文本、注释`节点等）

```
      nodeType nodeName nodeValue
元素节点    1    大写的标签名    null
文本节点    3       #text     文本内容
注释节点    8     #comment    注释内容
document   9     #document     null
//高版本浏览器会把空格和换行都当作 文本节点
```

```
1.childNodes ： 获取当前元素所有的子节点;
//获取到的是一个子节点集合（类数组），不仅仅是元素节点，文本、注释都会包含在内

2.children ： 获取当前元素的子元素节点

3.firstChild ： 获取第一个子节点;
//firstElementChild ： 获取第一个子元素节点； 在IE8及以下，不兼容

4.lastChild :获取最后一个子节点;
//lastElementChild ： 获取最后一个子元素节点；在IE8及以下，不兼容

5.previousSibling ： 获取上一个哥哥节点
//previousElementSibling 获取上一个哥哥元素节点，IE6-8不兼容

6.nextSibling ： 获取下一个弟弟节点
//nextElementSibling ： 获取下一个弟弟元素节点，IE6-8不兼容

7.parentNode: 获取当前元素的父亲节点
```

动态操作DOM

```
1.document.createElement('标签名')
//通过标签名创建一个元素,标签名大小写无所谓
//document.createAttribute()//创建一个属性节点
//document.createTextNode() //创建文本节点

2. appendChild  //在容器的最后添加一个元素

3.insertBefore  //[container].insertBefore(newChild,oldChild)

4.removeChild //[container].removeChild(child)
//返回被删除的节点，如果节点不存在则返回 null

5.replaceChild : [container].replaceChild(newChild,oldChild);

6.cloneNode ： 复制节点，默认浅克隆,参数:
true: //深克隆，会把当前元素以及子孙节点全部获取到
false或不传://浅克隆，只会克隆当前元素节点;

7.setAttribute
  getAttribute
  removeAttribute
//设置/获取/删除 当前元素的某一个自定义属性
var oBox=document.getElementByI d('box');

//=>把当前元素作为一个对象，在对象对应的堆内存中新增一个自定义的属性
oBox.myIndex = 10;//=>设置
console.log(oBox['myIndex']);//获取
delete oBox.myIndex; //=>删除
```

```
//=>基于Attribute等DOM方法完成自定义
oBox.setAttribute('myColor','red'); //=>设置
oBox.getAttribute('myColor'); //=>获取
oBox.removeAttribute('myColor'); //=>删除

/*以上下两种机制属于独立的运作体制，不能互相混淆使用
第一种是基于对象键值对操作方式，修改当前元素对象的堆内存空间来完成
第二种是直接修改页面中HTML标签的结构来完成(此种办法设置的自定义属性可以在结构上呈现出来)*/

8.classList :
classList属性是新集合类型DOMTokenList的实例//它有一个表示自己包含多少元素的length属性，而要取得每个元素可以使用item()方法，也可以使用方括号法
classList属性是只读的，但是可以使用add()和remove()方法修改它

box.classList.add("cup");    //新增class
box.classList.remove("box");     // 删除class
box.classList.replace("box","hghg");     // 替换class;

var header=document.getElementsByClassName('header')[0];
console.log(header.classList);
//返回值：一个DOMTokenList，包含元素的类名列表
//DOMTokenList ["header", value: "header"]
```

Math

```
typeof Math -> 'object'
Math: {} 是window下的一个键值对;
//console.log(window.Math);
```

Math对象常用属性

```
Math.E 自然对数的底数（或称为基数），约等于 2.718
Math.PI
Math.SQRT2
Math.LOG10E 属性表示以 10 为底数，e 的对数
LN2:
LN10:
SQRT1_2:
```

Math对象常用方法

```
1.Math.abs() :
2.Math.sqrt() :
3.Math.floor() : 向下取整
4.Math.ceil() : 向上取整
```

```
5.Math.max() :
6.Math.min() :
7.Math.random() 随机数，取值范围[0,1)
8.Math.round() : 四舍五入取整
取m-n之间的随机整数 //Math.round(Math.random()*(n-m)+m)
  //Math.round(-12.5) => -12
  //Math.round(-12.51) => -13
  //Math.round(12.5)  => 13
9.Math.pow : //Math.pow(2,3)//8
10.Math.cbrt()//立方根
//Math.cbrt(27)//3
11.Math.exp(x)//e的x次方
12.Math.sign()//返回数字符号，正数返回1、负数返回-1、0返回0
13.Math.trunc()//求小数的 整数部分
14.Math.fround() //将任意的数字转换为离它最近的单精度浮点数形式的数字
```

JS浮点数精度

```
Math.fround() 可以将任意的数字转换为离它最近的单精度浮点数形式的数字
0.1 + 0.2 == 0.3//false
Math.fround(0.1 + 0.2)==Math.fround(0.3)//true
//JavaScript 内部使用64位的双浮点数字，支持很高的精度。但是，有时你需要用32位浮点数字，这时会产生混乱：检查一个64位浮点数和一个32位浮点数是否相等会失败，即使二个数字几乎一模一样
//要解决这个问题，可以使用 Math.fround() 来将64位的浮点数转换为32位浮点数。在内部，JavaScript 继续把这个数字作为64位浮点数看待，仅仅是在尾数部分的第32位执行了“舍入到偶数”的操作，并将后续的尾数位设置为0。如果数字超出32位浮点数的范围，则返回 Infinity 或 -Infinity
Math.fround(2 ** 150); // I nfinity
//如果参数无法转换成数字，或者为NaN，返回NaN
Math.fround() // NaN
Math.fround('abc'); // NaN
Math.fround(NaN); // NaN
```

toFixed奇葩问题:

```
1.35.toFixed(1) // 1.4 正确
1.335.toFixed(2) // 1.33 错误
1.3335.toFixed(3) // 1.333 错误
1.33335.toFixed(4) // 1.3334 正确
1.333335.toFixed(5) // 1.33333 错误
1.3333335.toFixed(6) // 1.333333 错误
toFixed()方法并不会完全按照 四舍五入 的方式把数字转成指定小数位数的数字
//为什么会产生这个问题?
JavaScript中所有数字包括整数和小数都只有一种类型：Number。它使用64位固定长度来表示，也就是标准的 double 双精度浮点数，优点是可以归一化处理整数和小数，节省存储空间
//通用处理方案：把需要计算的数字乘以10的n次幂，换算成计算机能够精确识别的整数，然后再除以10的n次幂
```

Date

Date 对象用`new Date()`创建
ECMAScript 标准要求的Date对象能够代表任何日期和时间，在1/1/1970之前或之后的1亿天内精确到毫秒。所以JavaScript能够表示直到275755年的日期和时间。

> 国际标准时间，简称UTC（Universal Coordinated Time），UTC时间即是GMT格林尼治时间
> 世界标准时间，本初子午线上的地方时，是0时区的区时，与我国的标准时间北京时间（东八区）晚8小时

`Date() 构造函数`有几种不同的形式：生成一个日期对象

```
new Date()  //Thu Dec 13 2018 19:47:55 GMT+0800 (中国标准时间)
new Date(milliseconds)   //new Date(5000) //Thu Jan 01 1970 08:00:05 GMT+0800 (中国标准时间)
new Date(datestring)//参数必须是一个日期形式的字符串
//dateString ： 日期形式字符串的格式主要有两种：
//(1) yyyy/MM/dd HH:mm:ss 推荐！若省略时间，返回的Date对象的时间为 00:00:00。
//(2) yyyy-MM-dd HH:mm:ss 若省略时间，返回的Date对象的时间为 08:00:00(加上本地时区)。若不省略时间，此字符串在IE中会转换失败！
//返回值：一个转换后的Date对象{Date}
//new Date('2018-12-13T11:49:53.999Z')
//输出：Thu Dec 13 2018 19:49:53 GMT+0800 (中国标准时间)

//其他日期格式的字符串：
//'2018-12-13T11:49:53.999Z'
//'Thu Jan 01 1970 08:00:05 GMT+0800 (中国标准时间)'
new Date(year,month,date,hour,minute,second,milliseconds)
//new Date(2018,11,13,1,34,45,12)
//Thu Dec 13 2018 01:34:45 GMT+0800 (中国标准时间)
注意：日期格式中，月份是从0-11，星期从0-6，日期从1-31
```

Date 静态方法:点前面是Date

```
Date.parse(dateString)//把字符串转换为Date对象 ，然后返回此Date对象与'1970/01/01 00:00:00'之间的毫秒值(北京时间的时区为东8区，起点时间实际为：'1970/01/01 08:00:00')
//Date.parse('2018/12/13 20:18:26')
//1544703506000
Date.UTC()  根据世界时返回 1970 年 1 月 1 日 到指定日期的毫秒数
//语法：Date.UTC(year,month,day,hours,minutes,seconds,milliseconds)
//Date.UTC(2018,12,13,20,18,26,999)
//1547410706999
Date.now()//无参数，返回当前日期和时间的Date对象与'1970/01/01 00:00:00'之间的毫秒值(北京时间的时区为东8区，起点时间实际为：'1970/01/01 08:00:00')
```

Date方法：点前面是日期的实例 (2018年12月13日)

```
Date()//返回当日的日期和时间
// "Thu Dec 13 2018 01:41:51 GMT+0800 (中国标准时间)"
//注意：该方法返回的是一个日期字符串，而通过new Date()返回的是一个日期对象
getFullYear()//根据本地时从 Date 对象以四位数字返回年份
getMonth()  //根据本地时从Date对象返回月份 (0 ~ 11)
getDate()//根据本地时从Date对象返回一个月中的某一天 (1 ~ 31)
//var date = new Date()
//date.getDate()//13
//var d = new Date("July 21, 1983 01:15:00");
//var n = d.getDate();//21
getHours()//根据本地时返回 Date 对象的小时 (0 ~ 23)
getMinutes()//根据本地时返回 Date 对象的分钟 (0 ~ 59)
getSeconds()//根据本地时返回 Date 对象的秒数 (0 ~ 59)
getMilliseconds()//根据本地时返回 Date 对象的毫秒(0 ~ 999)
getDay()//根据本地时从Date对象返回一周中的某一天 (0 ~ 6) 星期天为0
getTime()//根据本地时返回 1970 年 1 月 1 日至今的毫秒数

toISOString()//使用ISO标准返回Date对象的字符串格式,格式为: YYYY-MM-DDTHH:mm:ss.sssZ
//var d = new Date(); d.toISOString();
//输出：2018-12-13T09:08:54.121Z(需要在小时的基础上加8，才是真正的时间)
toLocaleString() //根据本地时间把 Date 对象转换为字符串
//var d = new Date(); d.toLocaleString()
//2018/12/13 下午5:18:51
toLocaleDateString()//根据本地时间把Date对象的 日期部分 转换为字符串
//var d = new Date(); d.toLocaleDateString()
//输出：2018/12/13
toLocaleTimeString()//根据本地时间把Date对象的 时间部分 转换为字符串
//var d = new Date(); d.toLocaleTimeString()
//下午5:16:56
toString()  把 Date 对象转换为字符串
//var d=new Date(); d.toString();
//输出：Thu Dec 13 2018 17:23:10 GMT+0800 (CST)
//Date.toString()//"function Date() { [native code] }"
toTimeString()  把 Date 对象的时间部分转换为字符串
//var d = new Date(); d.toTimeString()
//输出：17:27:26 GMT+0800 (CST)
toUTCString() //根据世界时 (UTC) 把 Date 对象转换为字符串
//var d = new Date();d.toUTCString()
//输出：Thu, 13 Dec 2018 09:29:01 GMT
valueOf()   //返回 Date 对象的原始值，原始值返回1970年1月1日午夜以来的毫秒数
//  var d = new Date(); d.valueOf()
//输出：1544693693107
```

定时器

每一种浏览器有自己的最小识别时间；谷歌的时间大概是5~6ms;火狐时间8~10之间； IE 的时间

```
是13~15之间 // 定时器的返回值代表在当前页面中是第几个定时器
```