js常用API：

数据类型：

- **Number**
- **Object**
- **String**
- **Null**
- **Undefined**
- **Boolean**

  Ps：转换规则是除了`undefined null false 0 NaN '' 或 " "`，其他值都视为`true`

操作方法：

- `Boolean()`布尔值转换，返回`true 或 false`
- `typeof`返回一个值的数据类型，检查一个没有声明的变量返回`undefined`
- `instanceof`运算符返回一个布尔值，表示对象是否为某个构造函数的实例
- `xx.indexOf()`返回某个指定的字符串值在字符串中首次出现的位置，对于大小写敏感，没有找到就返回-1
- `in`运算符 检查某个键名是否存在的运算符`in`，适用于对象，也适用于数组 xx in xx
- 遍历数组

```
//for ... in
for(let i in a){
  console.log(a[i]);
}
// forEach
```

对象操作方法 Object:

- Object.defineProperty() 通过描述对象，定义某个属性
- Object.keys 和 Object.getOwnPropertyNames() 用于遍历对象的属性
- Object.create() 该方法可以指定原型对象和属性，返回一个新的对象
- Object.getPrototypeOf() 获取对象的Prototype对象
- obj.toString() 返回一个对象的字符串形式，默认情况下返回类型字符串，通过自定义toString方法，可以让对象在自动类型转换时，得到想要的字符串形式 通过函数的call方法，可以在任意值上调用这个方法，帮助我们判断这个值的类型

```
Object.prototype.toString.call(value)

Object.prototype.toString.call(2) // "[object Number]"
Object.prototype.toString.call('') // "[object String]"
Object.prototype.toString.call(true) // "[object Boolean]"
Object.prototype.toString.call(undefined) // "[object Undefined]"
Object.prototype.toString.call(null) // "[object Null]"
Object.prototype.toString.call(Math) // "[object Math]"
Object.prototype.toString.call({}) // "[object Object]"
Object.prototype.toString.call([]) // "[object Array]"
```

- obj.hasOwnProperty('') 检查该实例对象自身是否具有该属性

数学对象操作方法 Math:

- Math.abs()：绝对值

```
Math.abs(1) // 1
Math.abs(-1) // 1
```

- Math.ceil()：向上取整
- Math.floor()：向下取整

```
Math.floor(3.2) // 3
Math.floor(-3.2) // -4
Math.ceil(3.2) // 4
Math.ceil(-3.2) // -3
```

- Math.max()：最大值
- Math.min()：最小值

```
Math.max(2, -1, 5) // 5
Math.min(2, -1, 5) // -1
Math.min() // Infinity
Math.max() // -Infinity
```

- Math.pow()：指数运算,第一个参数为底数，第二个参数为幂的指数值

```
Math.pow(2, 2) // 4
```

- Math.sqrt()：平方根，参数为负数返回NaN

```
Math.sqrt(4) // 2
Math.sqrt(-4) // NaN
```

- Math.log()：自然对数，返回以e为底的自然对数值

```
Math.log(Math.E) // 1
Math.log(10) // 2.302585092994046
```

- Math.exp()：e的指数，返回常数e的参数次方

```
Math.exp(1) // 2.718281828459045
Math.exp(3) // 20.085536923187668
```

- Math.round()：四舍五入

```
Math.round(0.1) // 0
Math.round(0.5) // 1
```

- Math.random()：随机数，返回0到1之间的一个随机数，可能等于0，但一定小于1

```
Math.random() // 0.7151307314634323
```

- **三角函数**
- Math.sin()：返回参数的正弦（参数为弧度值）
- Math.cos()：返回参数的余弦（参数为弧度值）
- Math.tan()：返回参数的正切（参数为弧度值）
- Math.asin()：返回参数的反正弦（返回值为弧度值）
- Math.acos()：返回参数的反余弦（返回值为弧度值）
- Math.atan()：返回参数的反正切（返回值为弧度值）

```
Math.sin(0) // 0
Math.cos(0) // 1
Math.tan(0) // 0

Math.sin(Math.PI / 2) // 1

Math.asin(1) // 1.5707963267948966
Math.acos(1) // 0
Math.atan(1) // 0.7853981633974483
```

数值操作方法:

- parseInt() 方法用于将字符串转为整数
- parseFloat() 方法用于将一个字符串转为浮点数
- isNaN() 方法可以用来判断一个值是否为NaN
- isFinite() 方法返回一个布尔值，表示某个值是否为正常的数值

数组对象方法 Array:

- indexOf方法返回给定元素在数组中第一次出现的位置，如果没有出现则返回-1
- Array.isArray() 返回一个布尔值，表示参数是否是数组
- xx.valueOf() 返回数组对象的原始值
- toString() 把数组转换为字符串，并返回结果
- concat() 拼接，多个数组合并，原数组不变
- join(',') 把数组所有元素放到一个字符串中，可以通过指定分隔符分隔
- pop() 删除并返回数组的最后一个元素，会改变原数组
- push() 向数组的末尾添加一个或更多元素，并返回新的长度，会改变原数组
- reverse() 颠倒数组中元素的顺序，该方法将改变原数组
- shift() 删除并返回数组的第一个元素
- unshift() 方法可向数组的开头添加一个或更多元素，并返回新的长度

- slice(start,end) 截取目标数组的一部分，返回一个新数组，原数组不变

常用于将类数组对象转为真正的数组

```
Array.prototype.slice.call({ 0: 'a', 1: 'b', length: 2 })
// ['a', 'b']

Array.prototype.slice.call(document.querySelectorAll("div"));
Array.prototype.slice.call(arguments);
```

- splice(start, count, addElement1, addElement2, ...) 删除原数组一部分成员，并可以在删除部分添加新成员，返回的是被删除元素，会改变原数组

参数1：删除的起始位置；参数2：被删除的个数；参数3以后：添加的新元素；起始位置如果是负数，就表示从倒数位置开始删除

```
var a = ['a', 'b', 'c', 'd', 'e', 'f'];
a.splice(4, 2) // ["e", "f"]
a // ["a", "b", "c", "d"]
```

- sort() 排序 默认按字典排序

```
//按大小排序
arr.sort(function(a,b){
  return a - b;
})
```

- map() 将所有成员传入参数函数，然后把每一次的执行结果组成一个新数组返回
- forEach(element,index,array) 遍历数组，无法中断执行 当前值 当前位置 整个数组
- filter() 过滤成员，满足条件的组成数组返回
- reduce()和reduceRight() 依次处理数组的每一个成员，最终累计为一个值

```
let arr = [6,7];
let a=arr.reduce(function(a,b){
   return a*b
});
```

日期对象操作方法 Date:

get方法

- getTime()：返回实例距离1970年1月1日00:00:00的毫秒数，等同于valueOf方法。
- getDate()：返回实例对象对应每个月的几号（从1开始）。
- getDay()：返回星期几，星期日为0，星期一为1，以此类推。
- getYear()：返回距离1900的年数。
- getFullYear()：返回四位的年份。
- getMonth()：返回月份（0表示1月，11表示12月）。
- getHours()：返回小时（0-23）。
- getMilliseconds()：返回毫秒（0-999）。
- getMinutes()：返回分钟（0-59）。

- getSeconds()：返回秒（0-59）。
- getTimezoneOffset()：返回当前时间与 UTC 的时区差异，以分钟表示，返回结果考虑到了夏令时因素。

set方法:

- setDate(date)：设置实例对象对应的每个月的几号（1-31），返回改变后毫秒时间戳。
- setYear(year): 设置距离1900年的年数。
- setFullYear(year [, month, date])：设置四位年份。
- setHours(hour [, min, sec, ms])：设置小时（0-23）。
- setMilliseconds()：设置毫秒（0-999）。
- setMinutes(min [, sec, ms])：设置分钟（0-59）。
- setMonth(month [, date])：设置月份（0-11）。
- setSeconds(sec [, ms])：设置秒（0-59）。
- setTime(milliseconds)：设置毫秒时间戳。

需要注意的是，凡是涉及到设置月份，都是从0开始算的，即0是1月，11是12月

JSON对象操作方法 JavaScript Object Notation:

- 在线检测JSON格式 https://jsonlint.com/
- JSON.stringify() 用于将一个值转换成JSON字符串,对于原始类型的字符串，转换结果会带双引号

```
JSON.stringify('abc') // ""abc""
JSON.stringify(1) // "1"
JSON.stringify(false) // "false"
JSON.stringify([]) // "[]"
JSON.stringify({}) // "{}"

JSON.stringify([1, "false", false])
// '[1,"false",false]'

JSON.stringify({ name: "张三" })
// '{"name":"张三"}'
```

- JSON.parse() 方法用于将 JSON 字符串转换成对应的值

```
JSON.parse('{}') // {}
JSON.parse('true') // true
JSON.parse('"foo"') // "foo"
JSON.parse('[1, 5, "false"]') // [1, 5, "false"]
JSON.parse('null') // null

var o = JSON.parse('{"name": "张三"}');
o.name // 张三
```

you won't get much work done if you only do it when you feel like it
如果你有心情去做的时候才去完成它，那你基本上不会完成得了多大的事情

A typical man should use this time as an aristorcrat would:to perform rigorous self improvement
一个一般的人，应该像贵族一样花他的时间：用于严格的自我提升。

函数 function:

- **立即执行函数**

```
(function(){ /* code */ }());
// 或者
(function(){ /* code */ })();
(function (){console.log('立即执行函数')}());
(function (){console.log('立即执行函数')})();
```

编程风格

- 使用空格键缩进
- 行尾不使用分号的情况：1.for和while循环 2.分支语句if switch try 3.函数声明语句，注意，函数表达式仍然要使用分号
- 变量声明 最好把变量都放在代码块的头部
- 相等和严格相等 相等运算符会自动转换变量类型，造成很多意想不到的情况.建议不要使用相等运算符，只使用严格相等运算符===

```
0 == ''// true
1 == true // true
2 == true // false
0 == '0' // true
false == 'false' // false
false == '0' // true
' \t\r\n ' == 0 // true
```

- switch...case结构可以用对象结构代替

运算符

赋值运算符

```
// 将 1 赋值给变量 x
var x = 1;

// 将变量 y 的值赋值给变量 x
var x = y;
// 等同于 x = x + y
x += y

// 等同于 x = x - y
x -= y

// 等同于 x = x * y
x *= y

// 等同于 x = x / y
x /= y
```

```
// 等同于 x = x % y
x %= y

// 等同于 x = x ** y
x **= y

// 等同于 x = x >> y
x >>= y

// 等同于 x = x << y
x <<= y

// 等同于 x = x >>> y
x >>>= y

// 等同于 x = x & y
x &= y

// 等同于 x = x | y
x |= y

// 等同于 x = x ^ y
x ^= y
```

布尔运算符

- 取反运算符：！用于将布尔值变为相反值，即true变成false，false变成true
- 且运算符：&& 它的运算规则是：如果第一个运算子的布尔值为true，则返回第二个运算子的值（注意是值，不是布尔值）；如果第一个运算子的布尔值为false，则直接返回第一个运算子的值，且不再对第二个运算子求值。

```
't' && '' // ""
't' && 'f' // "f"
't' && (1 + 2) // 3
'' && 'f' // ""
'' && '' // ""

var x = 1;
(1 - 1) && ( x += 1) // 0
x // 1
```

- 或运算符：|| 或运算符（||）也用于多个表达式的求值。它的运算规则是：如果第一个运算子的布尔值为true，则返回第一个运算子的值，且不再对第二个运算子求值；如果第一个运算子的布尔值为false，则返回第二个运算子的值。

```
't' || '' // "t"
't' || 'f' // "t"
'' || 'f' // "f"
'' || '' // ""
```

- 三元运算符：?: 如果第一个表达式的布尔值为true，则返回第二个表达式的值，否则返回第三个表达式的值。

```
't' ? 'hello' : 'world' // "hello"
0 ? 'hello' : 'world' // "world"
```

绑定this

- func.call(thisValue, arg1, arg2, ...)

call的第一个参数就是this所要指向的那个对象，后面的参数则是函数调用时所需的参数

##History对象

window.history对象包含楼兰器的历史（url）的集合

history方法

- history.back() 后退
- history.forward() 前进
- history.go() 进入历史中的某个页面

##location对象

`window.location`对象用于获取当前页面的地址（url），并把浏览器定向到新的页面

对象属性:

- location.hostname web主机域名
- location.pathname 当前页面的路径和文件名
- location.port web主机端口
- location.protocol web协议(http://或https://)
- location.href 当前页面url
- location.assign() 加载新的文档

screen对象

`window.screen`对象包含有关用户屏幕的信息

属性:

- screen.availWidth 可用的屏幕宽度
- screen.availHeight 可用的屏幕高度
- screen.height 屏幕高度
- screen.width 屏幕宽度

##计时器：(全局API)

- setInterval()　//周期性定时器:　　清除：<clearTimeout（timer）>
- setTimeout()　//一次性定时器：清除：<clearInterval (timer)>
- setImmediate()　//立即执行定时器:

  把回调函数放在事件队列的最前边执行，当主程序执行完就会执行事件 对列中的回调函数

  清除：<clearImmediate(timer)>