

# 内置对象

- JavaScript 中的对象分为3种：自定义对象、内置对象、浏览器对象
- 内置对象就是指 JS 语言自带的一些对象，这些对象供开发者使用，并提供了一些常用的或是最基本而必要的功能
- JavaScript 提供了多个内置对象：Math、Date、Array、String等

## 查阅文档

学习一个内置对象的使用，只要学会其常用成员的使用即可，我们可以通过查文档学习，可以通过MDN/W3C来查询

MDN: <https://developer.mozilla.org/zh-CN/>

1. 查阅该方法的功能
2. 查看里面参数的意义和类型
3. 查看返回值的意义和类型
4. 通过 demo 进行测试

## Math对象

Math 对象不是构造函数，它具有数学常数和函数的属性和方法。跟数学相关的运算（求绝对值，取整、最大值等）可以使用 Math 中的成员。

通过 MDN 上的 Math 对象 里面的 `max` 方法我们可以通过练习来熟悉

```
1      <script>
2          // Math数学对象，不是一个构造函数，所以我们不需要new 来调用，而是直接使用里面的
          属性和方法即可
3
4          //Math.PI    圆周率    是静态属性，使用中不带括号
          console.log(Math.PI);
5
6
7          var re = Math.max(1,5,23,123,0);
8          //Math.max()    函数返回一组数中的最大值    是方法，需要带括号。
          console.log(re);
9
10     </script>
```

```
1      <script>
2          function getMaxOfArray(numArray) {
3              return Math.max.apply(null, numArray);
4          }
5          var a = getMaxOfArray([1,0,23,123,1231234,5454545]);
6          console.log(a);
7      </script>
```

## 封装自己的数学对象

通过函数的封装，我们也可以构建自己的数学对象。

```
1  <script>
2      var myMath = {
3          PI: 3.141592653,
4          max: function() {
5              var max = arguments[0];
6              for (var i = 1; i < arguments.length; i++) {
7                  if (arguments[i] > max) {
8                      max = arguments[i];
9                  }
10             }
11             return max;
12         },
13         min: function() {
14             var min = arguments[0];
15             for (var i = 1; i < arguments.length; i++) {
16                 if (arguments[i] < min) {
17                     min = arguments[i];
18                 }
19             }
20             return min;
21         }
22     }
23     console.log(myMath.PI);
24     console.log(myMath.max(1, 5, 9));
25     console.log(myMath.min(1, 5, 9));
26 </script>
```

## Math的绝对值和三个取整方法

**Math.floor() 向下取整**

**Math.ceil() 向上取整**

**Math.round() 四舍五入，就近取整**

**Math.abs() 绝对值**

```
1
2  <script>
3      //绝对值
4      console.log(Math.abs(1));
5      console.log(Math.abs(-1));
6      console.log(Math.abs('-1')); //隐式转换，会把字符串 -1 转换为数字型
7      console.log(Math.abs('string')); //NaN
8
9      // round(), 四舍五入，但是.5特殊，会往大了取。
10     console.log(Math.round(1.1)); //1
11     console.log(Math.round(1.5)); //2
12     console.log(Math.round(1.9)); //2
13     console.log(Math.round(-1.1)); //-1
```

```

14     console.log(Math.round(-1.5));      //-1
15
16     //向下取整
17     console.log(Math.floor(1.1));        //1
18     console.log(Math.floor(1.9));        //1
19
20     //向上取整
21     console.log(Math.ceil(1.1));         //2
22     console.log(Math.ceil(1.9));         //2
23 </script>

```

## 随机数方法 random()

函数返回一个浮点数，伪随机数在范围从**0到小于1**，也就是说，从0（包括0）往上，但是不包括1（排除1）。 $0 \leq x < 1$

```

1 <script>
2     console.log(Math.random());
3 </script>

```

例子:

```

1 <script>
2     function getRandomInt(min, max) {
3         min = Math.ceil(min);
4         max = Math.floor(max);
5         return Math.floor(Math.random() * (max - min)) + min; //不含最大
    值，含最小值
6     }
7     var g = getRandomInt(1,10);
8     console.log(g);
9 </script>

```

## 案例:随机点名

```

1 <script>
2     function getRandomInt(min, max) {
3         min = Math.ceil(min);
4         max = Math.floor(max);
5         return Math.floor(Math.random() * (max - min)) + min; //不含最大
    值，含最小值
6     }
7     var arr = ['张三','李四','王二麻子'];
8     var arrR = getRandomInt(0,2);    //通过索引值，进行判定
9     console.log(arr[arrR]);
10    // 优化方法
11    var arr1 = ['张三','李四','王二麻子','甲','乙','丙','丁'];
12    var arrR1 = getRandomInt(0,arr1.length-1);
13    console.log(arr1[arrR1]);
14 </script>

```

## Date()对象

- Date 对象和 Math 对象不一样，他是一个构造函数，所以我们需要实例化后才能使用
- Date 实例用来处理日期和时间

### 获取当前时间必须实例化

使用 `new`；来实例化对象

```
1 <script>
2   var now = new Date();
3   console.log(now);
4 </script>
```

### Date()构造函数的参数

如果括号里面有时间，就返回参数里面的时间。例如日期格式 字符串 为 '2019-5-1'，可以写成 `new Date('2019-5-1')` 或者 `new Date('2019/5/1')`

1. 如果Date()不写参数，就返回当前时间

```
1 <script>
2   var now = new Date();
3   console.log(now);
4 </script>
```

2. 如果Date()里面写参数，就返回括号里面输入的时间

```
1. 1 <script>
2   // 2.参数常用的写法 数字型 2019,10,1 字符串型 '2019-10-1
   8:8:8' 时分秒
3   // 如果Date()里面写参数，就返回括号里面输入的时间
4   var data = new Date(2019,10,1);
5   console.log(data); // Fri Nov 01 2019 00:00:00
   GMT+0800 (中国标准时间) , 返回的是11月不是10月
6
7   var data2 = new Date('2019-10-1 8:8:8');
8   console.log(data2); // Tue Oct 01 2019 08:08:08
   GMT+0800 (中国标准时间) ,有时间值的
9 </script>
```

## 日期格式化

我们想要 2019-8-8 8:8:8 格式的日期，要怎么办？

需要获取日期指定的部分，所以我们要手动的得到这种格式

方法名	说明	代码
getFullYear()	获取年份	dObj.getFullYear()
getMonth()	获取当月(0-11)	dObj.getMonth()
getDate()	获取当天日期	dObj.getDate()
getDay()	获取星期几(周日0到周六6)	dObj.getDay()
getHours()	获取当前小时	dObj.getHours()
getMinutes()	获取当前分钟	dObj.getMinutes()
getSeconds()	获取当前秒钟	dObj.getSeconds()

```
1      <script>
2          var now = new Date();
3          console.log(now.getFullYear()); //获取当前年份
4          console.log(now.getMonth()+1);    //获取月份（0-11）,返回的月份小一个月
      记得月份 +1
5          console.log(now.getDate()); //返回的是几号
6          console.log(now.getDay()); //周一返回1 周六返回六 周日返回0
7          console.log(now.getHours()); //获取当前小时
8          console.log(now.getMinutes()); //获取当前分钟数
9          console.log(now.getSeconds()); //获取当前秒钟数
10     </script>
```

**例子：使用Date()对象，写一个当前时间。**

```
1      <script>
2          var now = new Date();
3          var year = now.getFullYear();
4          var month = now.getMonth() + 1;
5          var day = now.getDate();
6          var hours = now.getHours();
7          var minutes = now.getMinutes();
8          var seconds = now.getSeconds();
9          console.log('今天
      是'+year+'年'+month+'月'+day+'日'+hours+'点'+minutes+'分'+seconds+'秒');
10     </script>
```

**案例：封装一个函数返回当前的时分秒，格式 08:08:08**

```
1      <script>
2          // 封装一个函数返回当前的时分秒    格式 08:08:08
3          function getTime() {
4              var time = new Date();
```

```

5         var h = time.getHours();
6         h = h < 10 ? '0'+h : h;
7         var m = time.getMinutes();
8         m = m < 10 ? '0'+m : m;
9         var s = time.getSeconds();
10        s = s < 10 ? '0'+s : s;
11        return h+':'+m+':'+s;
12    }
13    console.log(getTime());
14 </script>

```

## 获取日期的总的毫秒形式

Date对象是基于1970年1月1日（世界标准时间）起的毫秒数，获得Date总的毫秒数(时间戳)。

注意：不是当前时间的毫秒数，而是距离1970年1月1号过了多少毫秒数

### 1、通过valueOf()、getTime() 两种方法来获取

```

1 // 使用途径，可以用来生成订单数(少量订单)，因为不同时间的毫秒数是不一样的。
2 var date = new Date();
3 console.log(date.valueOf()); //是现在时间距离1970.1.1 总的毫秒数
4 console.log(date.getTime());

```

### 2、+new Date() 较为简单的写法

```

1 var date1 = +new Date(); // +new Date() 返回的也是总毫秒数
2 console.log(date1);

```

### 3、HTML5新增的方法，Date.now() 获取总毫秒数。

```

1 console.log(Date.now()); //H5新增的方法，也是获取距离1970年1月1号的总毫秒数。

```

## 案例：倒计时效果

思路：

- 1、截止的时间 减去 现在的时间 = 剩余的时间（倒计时）
- 2、使用时间戳实现以下效果，用户输入时间的总毫秒数减去现在的总毫秒数，得到剩余时间的毫秒数。
- 3、把剩余的时间毫秒数转换为 天、时、分、秒 (需要将时间戳转换为时分秒)

公式如下：

- d = parseInt(总秒数/60/60/24); //计算天数
- h = parseInt(总秒数/60/60%24); //计算小时
- m = parseInt(总秒数/60%60); //计算分数
- s = parseInt(总秒数%60); //计算当前秒数

```

1 <script>

```

```

2      function demo(time) {
3          var nowTime = +new Date();          //获取现在距离1970年的总毫秒数
4          var input = +new Date(time);        //获取活动截止日期距离1970年的总
毫秒数
5          var times = (input - nowTime) / 1000;    //活动截止日期 - 现在时间 =
距离活动截止的倒计时
6          var d = parseInt(times/60/60/24);    //计算天数
7          d = d < 10 ? '0' + d : d;
8          var h = parseInt(times/60/60%24);    //计算小时
9          h = h < 10 ? '0' + h : h;
10         var m = parseInt(times/60%60);        //计算分数
11         m = m < 10 ? '0' + m : m;
12         var s = parseInt(times%60);           //计算当前秒数
13         s = s < 10 ? '0' + s : s;
14         return d + '天' + h + '时' + m + '分' + s + '秒';
15     }
16     console.log(demo('2022-3-18 18:00:00'));
17     var date = new Date();
18     console.log(date);
19 </script>

```

## 数组对象

### 数组对象的创建

创建数组对象的两种方式

- 字面量方式
- new Array()

#### 字面量方式

```

1 <script>
2     var arr = [1,2,3,4];
3 </script>

```

#### 利用 new Array() 创建数组

```

1 <script>
2     var arr1 = new Array(); //创建了一个空数组
3     var arr2 = new Array(2); //一个2,代表了数组长度为2,但是里面的数组元素为空
4     var arr3 = new Array(2,3) //等价于 [2,3] ,里面有两个数组元素 2,3
5     console.log(arr1);
6     console.log(arr2);
7     console.log(arr3);
8 </script>

```

## 检测是否为数组

- `instanceof` 运算符，可以判断一个对象是否属于某种类型
- `Array.isArray()` 用于判断一个对象是否为数组，`isArray()` 是 HTML5 中提供的方法

```
1      <script>
2          var arr = [1, 23];
3          var obj = {};
4          // instanceof 运算符，可以判断一个对象是否属于某种类型
5          console.log(arr instanceof Array); // true
6          console.log(obj instanceof Array); // false
7          // Array.isArray() 用于判断一个对象是否为数组，isArray() 是 HTML5 中提供
      的方法
8          console.log(Array.isArray(arr));    // true
9          console.log(Array.isArray(obj));    // false
10     </script>
```

**例子：通过检测函数输入的值是否为数组，使用 `instanceof` 或者 `Array.isArray()` 来判定。**

原代码：

```
1      <script>
2          // 通过翻转数组来举例，如果函数输入的不是一个数组，就不会实现我们想要的数组翻转的
      效果。
3          function reverse(arr) {
4              var newArr = [];
5              for (var i = arr.length - 1; i >= 0; i--) {
6                  newArr[newArr.length] = arr[i];
7              }
8              return newArr;
9          }
10         var arr1 = reverse([1, 3, 4, 6, 9]);
11         console.log(arr1);
12         // 通过此例子可以看出，我们需要修改代码，来判定输入的值，是否为一个数组
13         var arr2 = reverse('qwe', 'asd', 'zxc');
14         console.log(arr2);
15
16     </script>
```

添加数组判定 后的代码：

```
1      <script>
2          function reverse(arr) {
3              if(arr instanceof Array){
4                  var newArr = [];
5                  for (var i = arr.length - 1; i >= 0; i--) {
6                      newArr[newArr.length] = arr[i];
7                  }
8                  return newArr;
9              }else{
10                 return '函数参数要求必须为数组格式[1,2,3,4]';
11             }
12         }
```



```

12     }
13     console.log(reverse(1,2,3,4,5));
14 </script>

```

## 添加删除数组元素

方法名	说明	返回值
push(参数1...)	末尾添加一个或多个元素，注意修改原数组	并返回新的长度
pop()	删除数组最后一个元素，一次只能删除一个元素	返回它删除的元素的值
unshift(参数1...)	向数组的开头添加一个或更多元素，注意修改原数组	并返回新的长度
shift()	删除数组的第一个元素，数组长度减1，无参数，修改原数组	并返回第一个元素

### push(参数1...) 末尾添加一个或多个元素，注意修改原数组,并返回新的长度

```

1     <script>
2         // 1.push() 在我们数组的末尾，添加一个或者多个数组元素 push 推
3         var arr = [1, 2, 3];
4         arr.push('哈哈', '秦晓');
5         console.log(arr.push('哈哈', '秦晓'));
6         console.log(arr);
7         console.log(arr.push(4, '秦晓'));
8         console.log(arr);
9         // push 完毕之后，返回结果是新数组的长度
10    </script>

```

- 1、push() 是可以给数组追加新的元素
- 2、push() 参数直接 写数组元素
- 3、push() 完毕之后，返回的结果是 新数组的长度
- 4、原数组也会发生变化

### unshift(参数1...) 向数组的开头添加一个或更多元素，注意修改原数组,并返回新的长度

```

1     <script>
2         // unshift 在我们数组的开头 添加一个或者多个数组元素
3         var arr = [1, 2, 3];
4         arr.unshift('blue');
5         console.log(arr.unshift('red'));
6         console.log(arr);
7     </script>

```

- 1、unshift() 是可以给数组前面追加新的元素
- 2、unshift() 参数直接写 数组元素
- 3、unshift() 完毕之后，返回的结果是 新数组的长度
- 4、原数组也会发生变化

### pop() 删除数组最后一个元素，一次只能删除一个元素 返回它删除的元素的值

```
1      <script>
2          //pop() 删除数组最后一个元素，一次只能删除一个元素 返回它删除的元素的值 ,不加
    参数
3          var arr = [1,2,3,4,5];
4          console.log(arr.pop());
5      </script>
```

- 1、pop() 是可以删除数组的最后一个元素，一次只能删除一个元素
- 2、pop() 没有参数
- 3、pop() 完毕之后，返回的结果是 删除的那个元素
- 4、原数组也会发生变化

### shift() 删除数组的第一个元素，数组长度减1，无参数，修改原数组，并返回第一个元素

```
1      <script>
2          // shift() 它删除数组的第一个元素，一次只能删除一个元素
3          var arr = [1,2,3,4,5];
4          arr.shift(); //不加参数
5          console.log(arr);
6      </script>
```

- 1、shift() 是可以删除数组的第一个元素，记住一次只能删除一个元素
- 2、shift()没有参数
- 3、shift()完毕之后，返回的结果是，删除的那个元素
- 4、原数组会发生变化

### 案例：筛选数组

有一个包含工资的数组[1500,1200,2000,2100,1800],要求把数组中工资超过2000的删除，剩余的放到新数组里面

```

1      <script>
2          var arr = [1500, 1200, 2000, 2100, 1800];
3          var newArr = [];
4          for (var i = 0; i < arr.length; i++) {
5              if (arr[i] < 2000) {
6                  newArr[newArr.length] = arr[i];
7              }
8          }
9          console.log(newArr);
10     </script>

```

使用push() ,添加新数组

```

1      <script>
2          var arr = [1500, 1200, 2000, 2100, 1800];
3          var newArr = [];
4          for (var i = 0; i < arr.length; i++) {
5              if (arr[i] < 2000) {
6                  newArr.push(arr[i]);
7              }
8          }
9          console.log(newArr);
10     </script>

```

## 数组的排序

方法名	说明	是否修改原数组
reverse()	颠倒数组中元素的顺序，无参数	该方法会改变原来的数组，返回新数组
sort()	对数组的元素进行排序	该方法会改变原来的数组，返回新数组

### reverse()

```

1      // 1、翻转数组
2          var arr = ['red', 'blue', 'green'];
3          arr.reverse();
4          console.log(arr);    // ['green', 'blue', 'red']

```

### sort()

sort() 对于个位数的使用

```

1      // 2、数组排序(冒泡排序)
2          var arr1 = [3,4,1,2,9];
3          arr1.sort();
4          console.log(arr1);  //[1, 2, 3, 4, 9]

```

sort()对于双位数的排序，因为sort()，进行排序的判定是从 **第一个数字** 进行判定的，所以在进行多位数判定时会出错。所以需要用到方法。**学会使用**

出错案例：

```
1 var arr2 = [2,31,23,41,9,0];
2 arr2.sort();
3 console.log(arr2);
```

解决方法

```
1 var arr2 = [2,31,23,41,9,0];
2 arr2.sort(function(a,b){
3     return a - b; //升序的顺序排列
4     return b - a; //降序的排列
5 });
6 console.log(arr2);
```

## 数组对象

方法名	说明	返回值
indexOf()	数组中查找该元素的第一个索引	如果存在返回索引号，如果不存在，则返回-1
lastIndexOf()	在数组中查找最后一个索引，从后向前索引	如果存在返回索引号，如果不存在，则返回-1

1、返回数组元素索引号方法 indexOf(数组元素) 作用就是返回该数组元素的索引号,只返回第一个满足条件的索引号。

2、它只发返回第一个满足条件的索引号。

3、如果找不到元素，则返回-1。

```
1 <script>
2     //返回数组元素索引号方法 indexOf(数组元素) 作用就是返回该数组元素的索引号,只返回
    第一个满足条件的索引号，从前面开始查找
3     //它只发返回第一个满足条件的索引号
4     //如果找不到元素，则返回-1
5     var arr = ['red','green','blue','pink','blue'];
6     console.log(arr.indexOf('blue')); // 2
7     // 返回数组元素索引号方法 indexOf(数组元素) 作用就是返回该数组元素的索引号,只返回
    第一个满足条件的索引号，从后面开始查找
8     console.log(arr.lastIndexOf('blue')); // 4
9 </script>
```

### 案例：数组去重

将数组中['c', 'a', 'z', 'a', 'x', 'a', 'x', 'c', 'b'], 中相同的数组元素去掉。

思路：

①、将旧数组中不重复的元素放到新数组中，重复的元素只保留一个。

②、遍历数组，用旧数组去查询新数组，如果该元素在新数组里面没有出现过，我们添加进去，否则就不添加。

旧数组: ['c', 'a', 'z', 'a', 'x', 'a', 'x', 'c', 'b']

新数组: []

③、我们怎么知道该元素没有存在？利用 `新数组.indexOf(数组元素)` 如果返回是 `-1` 就说明 新数组里面没有该元素

```
1      <script>
2          // 封装一个去重的函数 unique 独一无二的
3          function unique(arr) {
4              var newArr = [];
5              for (var i = 0; i < arr.length; i++) {
6                  if (newArr.indexOf(arr[i]) === -1) {
7                      newArr.push(arr[i]);
8                  }
9              }
10             return newArr;
11         }
12         var demo = unique(['c', 'a', 'z', 'a', 'x', 'a', 'x', 'c', 'b']);
13         var demo1 = unique(['blue', 'green', 'blue']);
14         console.log(demo);
15         console.log(demo1);
16     </script>
```

## 数组转换为字符串

方法名	说明	返回值
toString()	把数组转换成字符串，逗号分隔每一项	返回一个字符串
join('分隔符')	方法用于把数组中的所有元素转换为一个字符串	返回一个字符串

**toString()** 将我们的数组转换为字符串

```
1      <script>
2          // 数组转换为字符串
3          // 1、toString() 将数组转换为字符串
4          var arr = [1,2,3];
5          console.log(arr.toString());
6      </script>
```

**join('分隔符')**

```
1      <script>
2          var arr = [1,2,3];
3          console.log(arr.join());    //括号里面不写值，默认使用逗号分割
4          console.log(arr.join('.')); //写值后，可以自定义分割符号
5          console.log(arr.join('&'));
6      </script>
```

# 字符串对象

## 基本包装类型了解

```
1      <script>
2          var str = 'abc';
3          console.log(str.length);
4          // 对象 才有属性和方法 复杂数据类型才有 属性和方法
5          // 简单数据类型为什么会有 length 属性呢？
6          //按道理基本数据类型是没有属性和方法的，而对象才有属性和方法，但上面代码却可以执
行，这是因为 js 会把基本数据类型包装为复杂数据类型，其执行过程如下：
7          // js中的基本包装类型：会把简单数据类型 包装成为了 复杂数据类型
8          // 1、把简单数据类型包装为复杂数据类型
9          var temp = new String('abc'); //使用 new String 创建的字符串为 复杂数
据类型
10         // 2、通过临时变量的值 给 str
11         str = temp;
12         // 3、销毁这个临时变量
13         temp = null;
14     </script>
```

为了方便操作基本数据类型，JavaScript 还提供了三个特殊的引用类型：`String`、`Number`和 `Boolean`。

**基本包装类型**就是把简单数据类型包装成为复杂数据类型，这样基本数据类型就有了属性和方法。

## 字符串的不可变性

```
1      <script>
2          // 字符串的不可变性
3          var str = 'demo';
4          console.log(str);
5          str = 'new demo';
6          console.log(str); //当字符串重新赋值后，会占用一块内存资源，当赋值的字
字符串过多后，内存占用过大，会造成电脑卡顿。
7          // 所以根据字符串的不可变性，不建议大量拼接字符串，大量拼接字符串会造成电脑卡顿。
8          var str = '';
9          for(var i = 1;i <= 1000000; i++){
10              str += i;
11          }
12          console.log(str);
13     </script>
```

## 根据字符返回位置

字符串所有的方法，都不会修改字符串本身(字符串是不可变的)，操作完成会返回一个新的字符串

方法名	说明
indexOf('要查找的字符', 开始的位置)	返回指定内容在原字符串中的位置，如果找不到就返回-1，开始的位置是index索引号
lastIndexOf()	从后往前找，只找第一个匹配的

```
1      <script>
2          // 字符串对象 根据字符返回位置 str.indexOf('要查找的字符', [起始的位置])
3          var str = '改革春风吹满地，春天来了';
4          console.log(str.indexOf('春')); //默认从0开始查找，结果为2
5          console.log(str.indexOf('春', 3)); // 从索引号是 3的位置开始往后查找，结果是
8
6      </script>
```

### 案例：查找字符串“abcoefoxyozzopp”中所有o出现的位置以及次数

- 核心算法：先查找第一个o出现的位置
- 然后 只要 indexOf返回的结果不是 -1 就继续往后查找
- 因为 indexOf 只能查找到第一个，所以后面的查找，一定是当前索引加1，从而继续查找

```
1      <script>
2          // 查找字符串“abcoefoxyozzopp”中所有o出现的位置以及次数
3          // 核心算法：先查找第一个o出现的位置
4          // 然后 只要 indexOf返回的结果不是 -1 就继续往后查找
5          // 因为 indexOf 只能查找到第一个，所以后面的查找，一定是当前索引加1，从而继续查
找
6          var str = "abcoefoxyozzopp";
7          var index = str.indexOf('o'); //3
8          var num = 0;
9          // console.log(index);
10         while (index !== -1) {
11             console.log(index);
12             num++;
13             index = str.indexOf('o', index + 1);
14         }
15         console.log('o出现的次数是：' + num);
16     </script>
```

## 根据位置返回字符

方法名	说明	使用
charAt(index)	返回指定位置的字符(index字符串的索引号)	str.charAt(0)



方法名	说明	使用
charCodeAt(index)	获取指定位置处字符的ASCII码(index索引号)	str.charCodeAt(0)
str[index]	获取指定位置处字符	HTML,IE8+支持和charAt()等效

### charAt(index) 返回指定位置的字符(index字符串的索引号)

1	<script>
2	var arr = 'andy';
3	console.log(arr.charAt(3)); //
4	// 遍历所有字符
5	for(var i = 0;i < arr.length;i++){
6	console.log(arr .charAt(i));
7	}
8	</script>

### charCodeAt(index) 获取指定位置处字符的ASCII码(index索引号)

**ASCII码**：在计算机中，所有的数据在存储和运算时都要使用二进制数表示（因为计算机用高电平和低电平分别表示1和0），例如，像a、b、c、d这样的52个字母（包括大写）以及0、1等数字还有一些常用的符号（例如\*、#、@等）在计算机中存储时也要使用二进制数来表示，而具体用哪些二进制数字表示哪个符号，当然每个人都可以约定自己的一套（这就叫编码），而大家如果要想互相通信而不造成混乱，那么大家就必须使用相同的编码规则，于是美国有关的标准化组织就出台了ASCII编码，统一规定了上述常用符号用哪些二进制数来表示。

ASCII 字符代码表 一																							
高四位   低四位		ASCII非打印控制字符										ASCII 打印字符											
		0000					0001					0010	0011	0100	0101	0110	0111						
		0					1					2	3	4	5	6	7						
	+进制	字符	ctrl	代码	字符解释	+进制	字符	ctrl	代码	字符解释	+进制	字符	+进制	字符	+进制	字符	+进制	字符	+进制	字符	ctrl		
0000	0	0	BLANK NULL	^@	NUL 空	16	▶	^P	DLE 数据链路转意	32		48	0	64	@	80	P	96	`	112	p		
0001	1	1	☺	^A	SOH 头标开始	17	◀	^Q	DC1 设备控制 1	33	!	49	1	65	A	81	Q	97	a	113	q		
0010	2	2	☺	^B	STX 正文开始	18	↕	^R	DC2 设备控制 2	34	"	50	2	66	B	82	R	98	b	114	r		
0011	3	3	♥	^C	ETX 正文结束	19	!!	^S	DC3 设备控制 3	35	#	51	3	67	C	83	S	99	c	115	s		
0100	4	4	♦	^D	EOT 传输结束	20	¶	^T	DC4 设备控制 4	36	\$	52	4	68	D	84	T	100	d	116	t		
0101	5	5	♣	^E	ENQ 查询	21	§	^U	NAK 反确认	37	%	53	5	69	E	85	U	101	e	117	u		
0110	6	6	♠	^F	ACK 确认	22	■	^V	SYN 同步空闲	38	&	54	6	70	F	86	V	102	f	118	v		
0111	7	7	●	^G	BEL 震铃	23	↑	^W	ETB 传输块结束	39	'	55	7	71	G	87	w	103	g	119	w		
1000	8	8	☐	^H	BS 退格	24	↑	^X	CAN 取消	40	(	56	8	72	H	88	X	104	h	120	x		
1001	9	9	○	^I	TAB 水平制表符	25	↓	^Y	EM 媒体结束	41	)	57	9	73	I	89	Y	105	i	121	y		
1010	A	10	☐	^J	LF 换行/新行	26	→	^Z	SUB 替换	42	*	58	:	74	J	90	Z	106	j	122	z		
1011	B	11	♂	^K	VT 竖直制表符	27	←	^[	ESC 转意	43	+	59	;	75	K	91	[	107	k	123	{		
1100	C	12	♀	^L	FF 换页/新页	28	└	^\	FS 文件分隔符	44	,	60	<	76	L	92	\	108	l	124			
1101	D	13	♪	^M	CR 回车	29	↔	^]	GS 组分隔符	45	-	61	=	77	M	93	]	109	m	125	}		
1110	E	14	🎵	^N	SO 移出	30	▲	^_	RS 记录分隔符	46	.	62	>	78	N	94	^	110	n	126	~		
1111	F	15	☐	^O	SI 移入	31	▼	^-	US 单元分隔符	47	/	63	?	79	O	95	_	111	o	127	Δ		

Back space

注：表中的ASCII字符可以用:ALT + “小键盘上的数字键” 输入

注：表中的ASCII字符可以用:ALT + “小键盘上的数字键” 输入



```

1      <script>
2          // charCodeAt(index)      获取指定位置处字符的ASCII码(index索引号)  目的:
判断用户按下了那个键位
3          var str = 'andy';
4          console.log(str.charCodeAt(0));    //97
5      </script>

```

### str[index] 获取指定位置处字符

```

1      <script>
2          // str 是定义的变量名, 可以自定义
3          var str = 'andy';
4          console.log(str[0]);    //a
5      </script>

```

### 案例：判断一个字符串“abcfoxyozzopp”中出现次数最多的字符，并统计其次数

```

1      <script>
2          // 判断一个字符串 “abcfoxyozzopp” 中出现次数最多的字符，并统计其次数
3          // 核心算法：利用 charAt() 遍历这个字符串
4          // 把每个字符都存储给对象， 如果对象没有该属性，就为1，如果存在了就 +1
5          // 遍历对象，得到最大值和该字符
6          // 有一个对象 来判断是否有该属性 对象['属性名']
7          var o = {
8              age: 18
9          }
10         }
11         if (o['sex']) {
12             console.log('里面有该属性');
13         }
14         } else {
15             console.log('没有该属性');
16         }
17     }
18
19     // 判断一个字符串 'abcfoxyozzopp' 中出现次数最多的字符，并统计其次数。
20     // o.a = 1
21     // o.b = 1
22     // o.c = 1
23     // o.o = 4
24     // 核心算法：利用 charAt() 遍历这个字符串
25     // 把每个字符都存储给对象， 如果对象没有该属性，就为1，如果存在了就 +1
26     // 遍历对象，得到最大值和该字符
27     var str = 'abcfoxyozzopp';
28     var o = {};
29     for (var i = 0; i < str.length; i++) {
30         var chars = str.charAt(i); // chars 是 字符串的每一个字符
31         if (o[chars]) { // o[chars] 得到的是属性值
32             o[chars]++;
33         } else {
34             o[chars] = 1;
35         }
36     }

```

```

36     }
37     console.log(o);
38     // 2. 遍历对象
39     var max = 0;
40     var ch = '';
41     for (var k in o) {
42         // k 得到是 属性名
43         // o[k] 得到的是属性值
44         if (o[k] > max) {
45             max = o[k];
46             ch = k;
47         }
48     }
49     console.log(max);
50     console.log('最多的字符是' + ch);
51 </script>

```

## 字符串操作方法(重点)

方法名	说明
concat(str1,str2,str3...)	concat() 方法用于连接两个或多个字符串。拼接字符串
substr(start,length)	从 start 位置开始(索引号), length 取的个数
slice(start,end)	从 start 位置开始, 截取到 end 位置, end 取不到 (两个都是索引号)
substring(start,end)	从 start 位置开始, 截取到 end 位置, end 取不到 (基本和 slice 相同, 但是不接受负)

### concat(str1,str2,str3...) 用于连接两个或多个字符串。拼接字符串

```

1 <script>
2     // concat('字符串1','字符串2','字符串3'...)
3     var arr = 'string';
4     console.log(arr.concat('red'));           //stringred
5 </script>

```

### substr(start,length) 从 start 位置开始(索引号), length 取的个数

```

1 <script>
2     // substr('截取的起始位置','截取几个字符')
3     var str = '改革春风吹满地';
4     console.log(str.substr(2,2));           //春风 , 第一个2是索引号的2, 从第几个开始, 第
        二个2, 是取几个字符
5 </script>

```

**slice(start,end)** 从 start 位置开始，截取到 end 位置，end 取不到 (两个都是索引号)

```
1 <script>
2     // slice('截取的起始位置','截取的截止位置，截止位置不取')
3     // 从 start 位置开始，截取到 end 位置，end 取不到 (两个都是索引号)
4     var str = '改革春风吹满地';
5     console.log(str.slice(0,3));    //改革春,0和3都是索引号
6 </script>
```

**substring(start,end)** 从 start 位置开始，截取到 end 位置，end 取不到 (基本和 slice 相同，但是不接受负)

```
1 <script>
2     // substring('截取的起始位置','截取的截止位置，截止位置不取')
3     // 从 start 位置开始，截取到 end 位置，end 取不到 (基本和 slice 相同，但是不
接受负)
4     var str = '改革春风吹满地';
5     console.log(str.substring(0,2));    //改革,0和2都是索引号
6 </script>
```

**replace()**方法 用于在字符串中用一些字符替换另一些字符

其使用格式: `replace(被替换的字符,要替换为的字符串)`

```
1 <script>
2     // 1. 替换字符 replace('被替换的字符', '替换为的字符') 它只会替换第一个字符
3     var str = 'andyandy';
4     console.log(str.replace('a', 'b'));    //bndyandy
5     // 有一个字符串 'abcfoxyozzopp' 要求把里面所有的 o 替换为 *
6     var str1 = 'abcfoxyozzopp';
7     while (str1.indexOf('o') !== -1) {
8         str1 = str1.replace('o', '*');
9     }
10    console.log(str1);    //abc*ef*xy*zz*pp
11 </script>
```

**split()**方法 用于切分字符串，它可以将字符串切分为数组。在切分完毕之后，返回的是一个新数组。

```
1 <script>
2     var str = 'a,b,c,d';
3     console.log(str.split(','));
4     // 返回的是一个数组 ['a', 'b', 'c', 'd']
5 </script>
```

```

1      <script>
2          // 2. 字符转换为数组 split('分隔符')    前面我们学过 join 把数组转换为字符串
3          var str2 = 'red, pink, blue';
4          console.log(str2.split(','));           //[ 'red', ' pink', ' blue' ]
5          var str3 = 'red&pink&blue';
6          console.log(str3.split('&'));           //[ 'red', 'pink', 'blue' ]
7      </script>

```

```

1      <script>
2          var str = 'a,b,c,d';
3          console.log(str.split('*'));           ///除了逗号都是一个数组
4          console.log(str.split(','));           //逗号就是一个数组
5          console.log(str.split());              //不带符号还是一个数组
6      </script>

```

## 大小写转换

### toUpperCase() 转换大写

```

1      <script>
2          var arr = 'abc';
3          console.log(arr.toUpperCase(arr));    //ABC
4      </script>

```

### toLowerCase() 转换小写

```

1      <script>
2          var arr = 'ABC';
3          console.log(arr.toLowerCase(arr));    //abc
4      </script>

```

## 简单类型于复杂类型 **了解**

简单类型又叫做基本数据类型或者 **值类型**，复杂类型又叫做 **引用类型**。

- 值类型：简单数据类型/基本数据类型，在存储时变量中存储的是值本身，因此叫做值类型
  - string, number, boolean, undefined, null

- ```
1 <script>
2   // 简单数据类型 null 返回的是一个空的对象 object
3   var timer = null;
4   console.log(typeof timer);
5   // 如果有个变量我们以后打算存储为对象，暂时没想好放啥， 这个时候就给
   null
6   // 1. 简单数据类型 是存放在栈里面 里面直接开辟一个空间存放的是值
7   // 2. 复杂数据类型 首先在栈里面存放地址 十六进制表示 然后这个地址
   指向堆里面的数据
8 </script>
```

- 引用类型：复杂数据类型，在存储时变量中存储的仅仅是地址（引用），因此叫做引用数据类型
  - 通过 new 关键字创建的对象（系统对象、自定义对象），如 Object、Array、Date 等