网易微专业之《前端开发工程师》

学习笔记

开始时间：2015.12.28

《JavaScript 程序设计》

基础篇（二）

六、JS 语句

JS 流程控制：

JavaScript 对程序流程的控制跟其他编程语言是一样的，主要有 3 种：

- **（1）顺序结构**
- **（2）选择结构**
- **（3）循环结构**

（1）顺序结构

顺序结构是 JavaScript 中最基本的结构，说白了就是按照从上到下、从左到右的顺序执行。

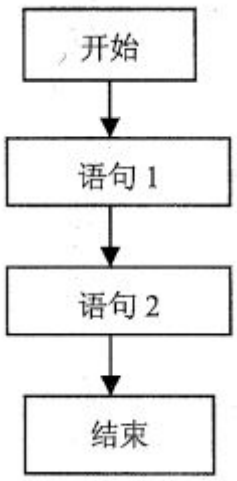


图 1 顺序结构

（2）选择结构

选择结构是按照给定的逻辑条件来决定执行的顺序，有单向选择、双向选择和多向选择之分，但是程序在执行过程中都只是执行其中的一条分支。

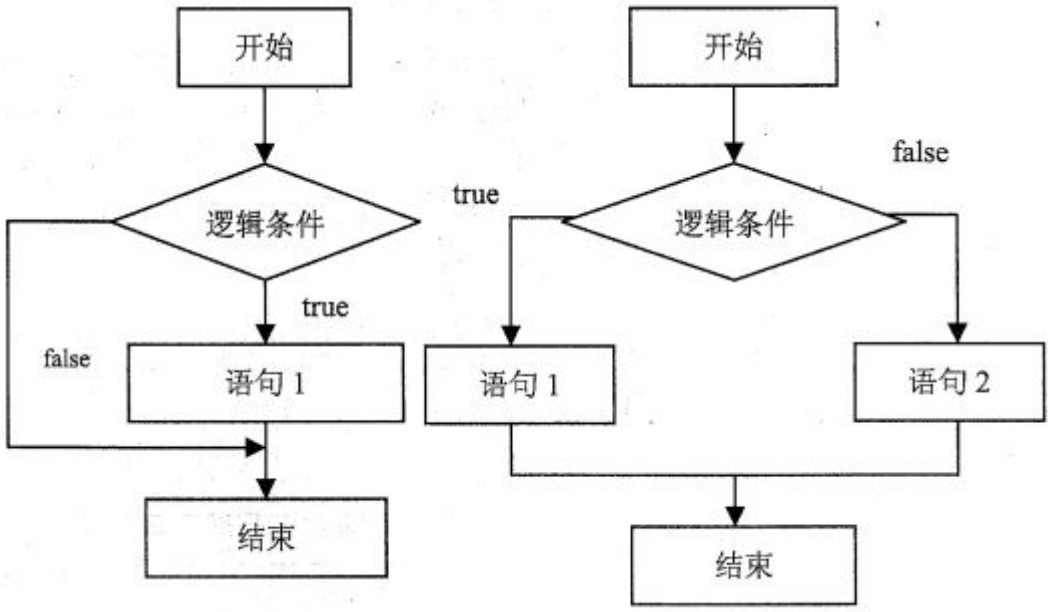


图 2 选择结构

对于选择结构，我们至少要掌握一下几种方法：

- **（1）if 语句;**
- **（2）if……else 语句;**
- **（3）if……else if……语句;**
- **（4）if 语句的嵌套;**
- **（5）switch 语句;**

（3）循环结构

循环结构即根据代码的逻辑条件来判断是否重复执行某一段程序。若逻辑条件为 true，则进入循环重复执行；若逻辑条件为 false，则退出循环。

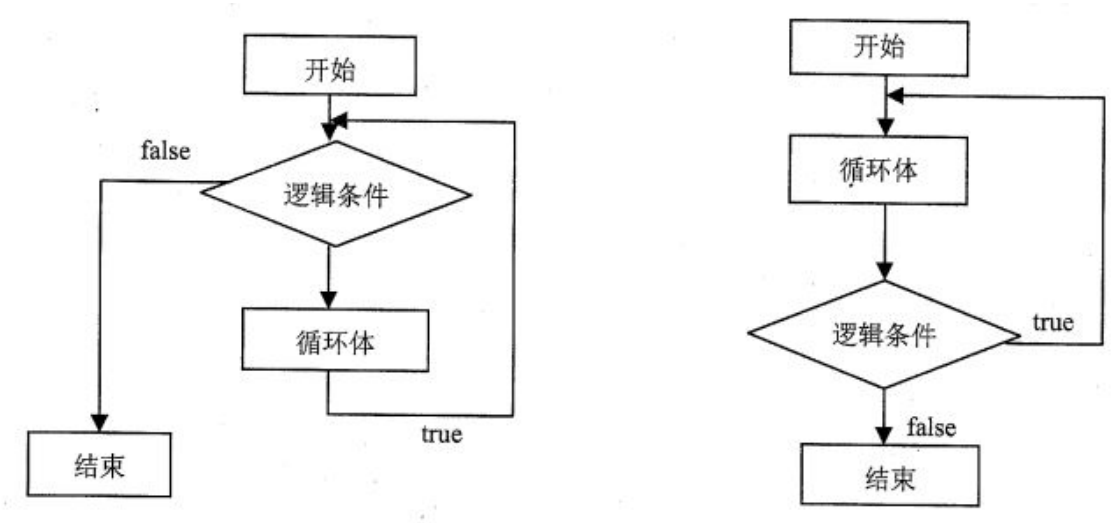


图 3 循环结构

循环结构语句主要包括 3 种：

- **（1）while 语句；**
- **（2）do……while 语句；**
- **（3）for 语句；**

JS 语句：

条件语句

if 语句

if 语句是使用最为普遍的条件选择语句，每一种编程语言的 if 语句都差不多。if 语句类型共有 3 种：

- **（1）if 语句（单向选择）；**
- **（2）if……else 语句（双向选择）；**
- **（3）if……else if 语句（多向选择）；**

1.if 语句

单一的 if 语句是“单分支选择结构语句”。

语法：

```
if(条件)
{ 条件成立时执行代码}
```

说明：

其中“条件”可以是任何一种逻辑表达式，如果“条件语句”的返回结果为 true，则程序先执行大括号“{}”中的“执行语句”，然后接着执行 if 后面的其他语句。

如果“条件语句”的返回结果为 false，则程序跳过“{}”的“执行语句”，直接执行程序后面的其他语句。

2.if……else 语句

“if……else”语句是“双向分支选择结构语句”，通常用于需要用两个程序分支来执行的情况（双向选择），也就是在 if 语句基础上多了一个分支。

语法：

```
if(条件)
{ 条件成立时执行的代码}
else
{条件不成立时执行的代码}
```

代码表示如下:

```
<script type="text/javascript">
  var mycarrer = "HTML"; //mycarrer 变量存储技能
  if (mycarrer == "HTML")
    { document.write("你面试成功，欢迎加入公司。"); }
  else  //否则，技能不是 HTML
    { document.write("你面试不成功，不能加入公司。");}
</script>
```

3.if……else if 嵌套语句

“if……else if”语句是“多分支选择结构语句”，用于选择多个代码块之一来执行。

语法：

```
if(条件 1)
{ 条件 1 成立时执行的代码}else  if(条件 2)
{ 条件 2 成立时执行的代码}
...else  if(条件 n)
{ 条件 n 成立时执行的代码}else
{ 条件 1、2 至 n 不成立时执行的代码}
```

```
<script type="text/javascript">
  var myscore = 86;
  if(myscore < 60){
      document.write("成绩不及格，加油了！");
  }else if(myscore < 75){
      document.write("成绩良好，不错啊");
  }else if(myscore < 85){
      document.write("成绩很好，很棒");
  }else{
      document.write("成绩优秀，超级棒");
  }
</script>
```

switch 语句

JavaScript 中，switch 语句也是选择结构中很常用的语句。switch 语句用于将一个表达式同多个值进行比较，并根据比较结果选择执行语句。

语法：

```
switch(表达式)
{case 值 1:
  执行代码块 1  break;
case 值 2:
  执行代码块 2
  break;
...
case 值 n:
  执行代码块 n
  break;
```

```
default:
  与 case 值 1 、 case 值 2...case 值 n 不同时执行的代码
}
```

语法说明:

Switch 必须赋初始值，值与每个 case 值匹配。满足执行该 case 后的所有语句，并用 break 语句来阻止运行下一个 case。如所有 case 值都不匹配，执行 **default** 后的语句。

default 语句是可选的，当其他所有的 case 语句定义的值都不满足时，就执行 default 后面的语句块。

对于 if 语句和 switch 语句，最核心的一点就是：对于判断条件较少的可以使用 if 语句，但是在实现一些多条件判断中，就应该使用 switch 语句。

示例代码如下:

```
<script type="text/javascript">
var myscore = 6; //myscore变量存储分数,假设为6
switch (myscore) //switch实现判断，case 6匹配
{
 case 0:
 case 1:
 case 2:
 case 3:
 case 4:
 case 5:
   degree = "继续努力！";
   document.write("评语: "+degree+"<br>");
   break;
 case 6:
   degree = "及格，加油!";
   document.write("评语:"+degree+"<br>");
   break;
 case 7:
   degree = "凑合，奋进!";
   document.write("评语: "+degree+"<br>");
   break;
 case 8:
   degree = "很棒，很棒！";
   document.write("评语:"+degree+"<br>");
   break;
 case 9:
 case 10:
   degree = "高手，大牛！";
   document.write("评语: "+degree);
}
</script>
```

循环语句

while 语句

while 语句是条件判断语句，也是循环语句。

语法：

```
while(条件表达式语句)
{
    执行语句块;
}
```

说明：

当“条件表达式语句”的返回值为 true 时，就会执行大括号“{}”中的语句块，当执行完大括号“{}”的语句块后，再次检测条件表达式的返回值，如果返回值还为 true，则重复执行大括号“{}”中的语句块，直到返回值为 false 时，才结束整个循环过程，接着往下执行 while 代码段后面的程序代码。

使用 while 语句要注意以下几点：

（1）应该使用大括号“{}”包含多条语句，即使是一条语句也最好使用大括号；

（2）在循环体中应该包含使得循环可以退出的语句，比如上面的“i++”。对于循环体，要是没有条件，循环就会无休止地运作下去，变成一个“死循环”，从而可能导致浏览器崩溃；

do...while 语句

“do...while 语句”跟 while 语句是非常类似的，唯一的区别在于：while 语句先判断是否符合条件，然后再执行循环体语句；do...while 语句先执行循环体语句一次，然后再判断是否符合条件。

语法：

```
do
{
    执行语句块;
}
while(条件表达式语句);
```

说明：

do...while 语句是先无条件执行循环体一次再判断是否符合条件的，如果符合条件，则重复执行循环体，如果不符合条件，则退出循环。

两者之间的区别：

（1）do...while 语句和 while 语句是可以相互转换的；

（2）do...while 语句将先执行一遍循环体中的语句，然后才判断条件表达式的真假。这是它与 while 语句的本质区别；

for 循环

for 语句通常由 2 部分组成：一是“条件控制部分”，二是“循环体”。

语法：

```
for(初始化表达式;循环条件表达式;循环后的操作表达式)
{
    执行语句块;
}
```

说明：

在使用 for 循环之前要先设定一个计数器变量，可以在 for 循环之前定义，也可以在使用时直接进行定义。

上面的语法中，“初始化表达式”表示计数器变量的初始值；“循环条件表达式”是一个计数器变量的表达式，决定了计数器的最大值；

For 循环两种写法

```
var arr=[12, 5, 7];
```

```
for(var i=0;i<arr.length;i++)
{
    alert('第'+i+'个东西：'+arr[i]);
}
/*第一种写法*/
/*
for(var i in arr)
{
    alert('第'+i+'个东西：'+arr[i]);
}
第二种写法*/
```

For-in 循环

语法格式：

for(属性名 in 对象) {语句}

跳转语句

JavaScript 支持的跳转语句主要有 2 种：

- **（1）break 语句；**
- **（2）continue 语句；**

break 语句与 continue 语句的主要区别是：break 是彻底结束循环，而 continue 是结束本次循环。

break 语句

break 语句用于退出包含在最内层的循环或者退出一个 switch 语句。break 语句通常用于 while、do...while、switch 或 for 语句中。

格式如下：

```
for(初始条件;判断条件;循环后条件值更新)
{
  if(特殊情况)
  {break;}
  循环代码
}
```

continue 语句

continue 语句跟 break 语句类似。不同之处在于，continue 语句用于退出本次循环，并开始下一次循环。而 break 语句是退出所有循环！

语句结构格式如下：

```
for(初始条件;判断条件;循环后条件值更新)
{
  if(特殊情况)
  { continue; }
 循环代码
}
```

上面的循环中，当特殊情况发生的时候，本次循环将被跳过，而后续的循环则不会受到影响。

跟 break 语句一样，continue 语句也只能用在 white、do...while、for 和 switch 等循环语句中。

With 语句

格式：**with(表达式){语句}**

对同一个对象进行多次操作时，使用 with 语句来简化代码。

异常捕获语句

try 语句允许我们定义在执行时进行错误测试的代码块。

catch 语句允许我们定义当 try 代码块发生错误时，所执行的代码块。

JavaScript 语句 **try** 和 **catch** 是成对出现的。

格式：

try{

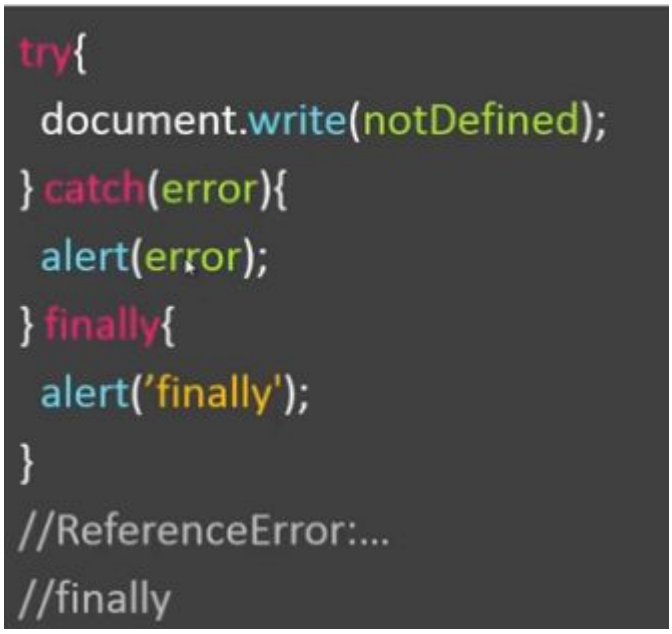
语句

}catch(exception){

语句

} finally{

语句}　　//不管上面的语句是否出错，都执行。


try{
document.write(notDefined);
} catch(error){
alert(error);
} finally{
alert('finally');
}
//ReferenceError:...
//finally

七、JS 数值

在 JavaScript 中，数值对象共有 2 种：

- **（1）Number 对象；**

  （2）Math 对象；

在 JavaScript 中，Math 对象是无需使用 new 关键词创建的，因此我们可以直接调用 Math 对象的属性和方法，来对数据进行计算。

使用 Math 的属性和方法，代码如下：

```
<script type="text/javascript">
  var mypi=Math.PI;
  var myabs=Math.abs(-15);
  document.write(mypi);
  document.write(myabs);
</script>
```

运行结果:

```
3.141592653589793
15
```

注意：Math 对象是一个固有的对象，无需创建它，直接把 Math 作为对象使用就可以调用其所有属性和方法。这是它与 Date,String 对象的区别。

Math 对象属性

属性	说明
E	返回算术常量 e，即自然对数的底数（约等于 2.718）。
LN2	返回 2 的自然对数（约等于 0.693）。
LN10	返回 10 的自然对数（约等于 2.302）。
LOG2E	返回以 2 为底的 e 的对数（约等于 1.442）。
LOG10E	返回以 10 为底的 e 的对数（约等于 0.434）。
PI	返回圆周率（约等于 3.14159）。
SQRT1_2	返回返回 2 的平方根的倒数（约等于 0.707）。
SQRT2	返回 2 的平方根（约等于 1.414）。

Math 对象方法

方法	描述
abs(x)	返回数的绝对值。
acos(x)	返回数的反余弦值。
asin(x)	返回数的反正弦值。
atan(x)	返回数字的反正切值
atan2(y,x)	返回由 x 轴到点(x,y)的角度(以弧度为单位)
ceil(x)	对数进行上舍入。
cos(x)	返回数的余弦。
exp(x)	返回 e 的指数。
floor(x)	对数进行下舍入。
log(x)	返回数的自然对数（底为e）。
max(x,y)	返回 x 和 y 中的最高值。
min(x,y)	返回 x 和 y 中的最低值。
pow(x,y)	返回 x 的 y 次幂。
random()	返回 0～1 之间的随机数。
round(x)	把数四舍五入为最接近的整数。
sin(x)	返回数的正弦。
sqrt(x)	返回数的平方根。
tan(x)	返回角的正切。
toSource()	返回该对象的源代码。
valueOf()	返回 Math 对象的原始值。

max()方法和 min()方法---返回多个数的最大值和最小值

在 JavaScript 中，我们可以使用 Math 对象的 max()方法返回多个数中的最大值，也可以使用 Math 对象的 min()方法返回多个数中的最小值。

语法：

```
Math.max(数 1,数 2,…,数 n)
Math.min(数 1,数 2,…,数 n)
```

sqrt()方法---求平方根

在 JavaScript 中，我们可以使用 Math 对象的 sqrt()方法返回一个数的平方根。

语法：

```
Math.sqrt(x)
```

说明：

参数 x 为必选项，且必须是大于等于 0 的数。计算结果的返回值是参数 x 的平方根。

如果 x 小于 0，则返回 NaN。

pow()方法---求数的幂

在 JavaScript 中，可以使用 Math 对象的 pow()方法求一个数的多次幂。

语法：

```
Math.pow(x,y)
```

说明：

x 是底数，且必须是数字。y 是幂数，且必须是数字。如果结果是虚数或负数，则该方法将返回 NaN。如果由于指数过大而引起浮点溢出，则该方法将返回 Infinity（即“无限”的意思）。

abs()方法---求一个数的绝对值

在 JavaScript 中，我们可以使用 Math 对象的 abs()方法来求一个数的绝对值。

语法：

```
Math.abs(x)
```

说明：

abs，也就是 absolute 的缩写，这样你也很容易记住这个方法。

random()方法---生成 0-1 之间的随机数

在 JavaScript 中，我们可以使用 Math 对象的 random()方法返回 0~1 之间的一个随机数。

语法：

```
Math.random()
```

说明：

random()方法是没有参数的，直接调用即可。random()方法返回值是 0~1 之间的一个伪随机数。

ceil()方法---返回大于等于指定数的最小整数

在 JavaScript 中，我们可以使用 Math 对象的 ceil()方法对一个数进行上舍入。所谓的“上舍入”，也就是返回大于或等于指定数的最小整数。

语法：

```
Math.ceil(x)
```

说明：

参数 x 必须是一个数值。Math.ceil(x)返回大于等于 x 的最小整数。

floor()方法---返回小于等于指定数的最小整数

在 JavaScript 中，我们可以使用 Math 对象的 floor()方法对一个数进行下舍入。所谓的“下舍入”，也就是返回小于或等于指定数的最小整数。

语法：

```
Math.floor(x)
```

说明：

参数 x 必须是一个数值。Math.floor (x)返回小于等于 x 的最小整数。

ceil()方法和 floor()方法命名很有意思，ceil 是“天花板”的意思，而 floor 是“地板”的意思，大家根据其含义很形象地理解这两个函数。

round()方法---取整运算

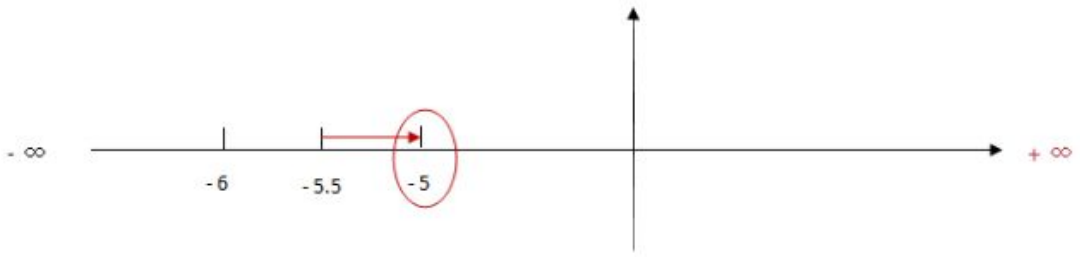
在 JavaScript 中 ,我们可以使用 Math 对象的 round()方法把一个浮点数四舍五入取整。

语法：

```
Math.round(x)
```

注意：

1. 返回与 x 最接近的整数。
2. 对于 0.5，该方法将进行上舍入。(5.5 将舍入为 6)
3. 如果 x 与两侧整数同等接近，则结果接近 +∞方向的数字值 。(如 -5.5 将舍入为 -5; -5.52 将舍入为 -6),如下图:



Math.round(x),Math.ceil(x),Math.floor(x)三者比较示例：

	Math.round(x)	Math.ceil(x)	Math.floor(x)
1.1	1	2	1
1.9	2	2	1

与数值相关的一些方法/对象/函数：

parseInt(string,radix)与 parseFloat(string)

在 JavaScript 中，将字符串型数据转换为数值型数据有 parseInt()和 parseFloat()这 2 种方法。其中，parseInt()可以将字符串转换为整型数据；parseFloat()可以将字符串转换为浮点型数据。

语法：

```
parseInt()  //将字符串型转换为整型
parseFloat()  //将字符串型转换为浮点型
```

说明：将字符串型转换为整型，前提是字符串一定要是数值字符串。

Number(value)

代表数值数据类型和提供数值常数的对象。参数 *value* 是要创建的 Number 对象的数值，或是要转换成数字的值。

.toFixed(digits)---将数字转换为指定小数位数的字符串。

应用于 Number 对象，返回一个字符串。将一个数字转换为指定小数位数的字符串。不传入参数，就是没小数位。返回值为四舍五入。Digits 表示小数点后的数字位数。其值必须在 0 - 20 之间，包括 0 和 20。

案例：(100.123).toFixed(2) // "100.12"
(100.123).toFixed(0) // "100"

parseInt/parseFloat/Number 比较：

	parseInt(string)	parseFloat(string)	Number(value)
“100.1”	100	100.1	100.1
“12.4b5”	12	12.4	NaN
“www”	NaN	NaN	NaN

toString() ---将数字转换为字符串。

使用指定的进制，将一个数字转换为字符串。不传入参数，默认为十进制。

参数：

①value {Number} ：表示进制数，取值范围：2 到 36

返回值：

{String} 转换后进制的字符串

示例：

```
(10).toString(); // => 10 : 默认为十进制
(10).toString(2); // => 1010 : 二进制
(10).toString(10); // => 10 : 十进制
(10).toString(16); // => a : 十六进制
```

八、JS 字符串

length 属性简介

在 JavaScript 中，对于字符串来说，要掌握的属性就只有一个，那就是 length 属性。我们可以通过 length 属性来获取字符串的长度。

语法：

```
字符串名.length
```

案例："micromajor".length // 10

match()方法简介

在 JavaScript 中，使用 match()方法可以从字符串内索引指定的值，或者找到一个或多个正则表达式的匹配。

语法：

```
stringObject.match(字符串)     //匹配字符串;
stringObject.match(正则表达式)  //匹配正则表达式
```

说明：

stringObject 指的是字符串对象。match()方法类似于 indexOf()方法，但是它返回的是指定的值，而不是字符串的位置。

match()方法就是用来检索一个字符串是否存在。如果存在的话，返回要检索的字符串；如果不存在的话，返回 null。

案例：

```
"micromajor163".match(/[0-9]/) // ["1"]
"micromajor163".match(/[0-9]/g)// ["1","6","3"]
"micromajor163".match(/[A-Z]/) // null
```

search()方法简介

在 JavaScript 中，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。

语法：

```
stringObject.search(字符串)        //检索字符串;
stringObject.search(正则表达式)   //检索正则表达式
```

说明：

stringObject 指的是字符串对象。search()方法返回的是子字符串的起始位置，如果没有找到任何匹配的子串，则返回-1。

案例：

```
"micromajor163".search(/[0-9]/) // 10
"micromajor163".search(/[A-Z]/) // -1
```

indexOf()方法简介

在 JavaScript 中，可以使用 indexOf() 方法可返回某个指定的字符串值在字符串中首次出现的位置。

语法：

```
stringObject.indexOf(substring, startpos)
```

参数	描述
substring	必需。规定需检索的字符串值。
startpos	可选的整数参数。规定在字符串中开始检索的位置。它的合法取值是 0 到 stringObject.length - 1。如省略该参数，则将从字符串的首字符开始检索。

stringObject 表示字符串对象。indexOf()方法跟 search()方法差不多，跟 match()方法类似，不同的是 indexOf()方法返回的是字符串的位置，而 match()方法返回的是指定的字符串。

案例："micro-major".indexOf("-")　　// 5

"micro-major-web".indexOf("-") // 5
"micro-major".indexOf("major") // 6
"micromajor".indexOf("-")　　// -1

replace()方法简介

在 JavaScript 中，replace()方法常常用于在字符串中用一些字符替换另一些字符，或者替换一个与正则表达式匹配的子串。

语法：

```
stringObject.replace(原字符,替换字符)
stringObject.replace(正则表达式,替换字符)　//匹配正则表达式
```

案例："micromajor163".replace("163","###")　// "micromajor###"

"micromajor163".replace(/[0-9]/,"#")　// "micromajor#63"
"micromajor163".replace(/[0-9]/g,"#") // "micromajor###"
"micromajor163".replace(/[0-9]/g,"")　// "micromajor"

charAt()方法简介

在 JavaScript 中，可以使用 charAt()方法来获取字符串中的某一个字符。这个方法我们在之前的教程中已经多次接触了。这个方法非常好用，在实际开发中也经常用到。

语法：

```
stringObject.charAt(n)
```

说明：

string.Object 表示字符串对象。n 是数字，表示字符串中第几个字符。注意，字符串中第一个字符的下标是 0，第二个字符的下标是 1，以此类推。

案例："micromajor".charAt(0)　// "m"

```
"micromajor".charAt(100) // ""
```

字符串英文大小写转化

在 JavaScript 中，使用 toLowerCase()和 toUpperCase()这两种方法来转化字符串的大小写。其中，toLowerCase()方法将大写字符串转换为小写字符串；toUpperCase()将小写字符串转换为大写字符串。

语法：

```
字符串名. toLowerCase()     //将大写字符串转换为小写字符串
字符串名. toUpperCase()     //将小写字符串转换为大写字符串
```

说明：

此外，还有 2 种大小写转化方法：toLocaleLowerCase()和 toLocaleUpperCase()。这两个方法我们有可能一辈子都用不到，大家要是别的书籍中看到，可以直接忽略。

案例："MicroMajor".toLowerCase() // "micromajor"

"MicroMajor".toUpperCase() // "MICROMAJOR"

连接字符串-concat()

在 JavaScript 中，可以使用 concat()方法来连接 2 个或多个字符串。

语法：

```
字符串 1.concat(字符串 2,字符串 3,…,字符串 n);
```

说明：

concat()方法将“字符串 2,字符串 3,…,字符串 n”按照顺序连接到字符串 1 的尾部，并返回连接后的字符串。

连接字符串可以有 2 种方式，一种是使用 concat()方法，另外一种更加简单，使用“+”

运算符就可以了。

案例：

```
"0571" + "-" + "88888888"  // "0571-88888888"
```

“0571”.concat(“-”,”88888888”); //”0571-88888888”

比较字符串

在 javascript 中 ,可以使用 localeCompare()方法用本地特定的顺序来比较两个字符串。

语法：

```
1字符串 1.localeCompare(字符串 2)
```

说明：

比较完成后，返回值是一个数字。

（1）如果字符串 1 小于字符串 2，则返回小于 0 的数字；

（2）如果字符串 1 大于字符串 2，则返回数字 1；

（3）如果字符串 1 等于字符串 2，则返回数字 0；

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <script type="text/javascript">
        var str1= "JavaScript";
        var str2 = "javascript";
        var str3 = str1.localeCompare(str2);
        document.write(str3);
    </script>
</head>
<body>
</body>
</html>
```

在浏览器预览效果为：1

（备注：这里的排序是按照什么规则来排序的？

经测试，不是按照 ASCII 码来排序的，ASCII 码中 9<A<a，用在此处是错误的。

查询，有人说的是根据本地系统的语言排序规则来的，XP 下是排序是发音。 ）

split()方法

在 javascript 中，可以使用 split()方法把一个字符串分割成字符串数组。

语法：

```
stringObject.split(separator,limit)
```

参数	描述
separator	必需。从该参数指定的地方分割 stringObject。
limit	可选参数，分割的次数，如设置该参数，返回的子串不会多于这个参数指定的数组，如果无此参数为不限制次数

说明：

separator 分割符可以是一个字符、多个字符或一个正则表达式。分割符并不作为返回数组元素的一部分。

```
案例："micro major".split(" ")       // ["micro","major"]

"micro major".split(" ",1)    // ["micro"]
"micro2major".split(/[0-9]/) // ["micro","major"]
```

从字符串提取字符串-substring()

在 JavaScript 中，可以使用 substring()方法来提取字符串中的某一部分字符串。

语法：

```
1字符串.substring(start 开始位置,stop 结束位置)
```

说明：

1.开始位置是一个非负的整数，表示从哪个位置开始截取。结束位置也是一个非负的整数，表示在哪里结束截取。

2.返回的内容是从 开始位置(包含 start 位置的字符)到 结束位置-1 处的所有字符 ,其长度为 stop 减 start。

3.如果参数 start 与 stop 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 stop 大，那么该方法在提取子串之前会先交换这两个参数。

```
案例："micromajor".substring(5,7)  // "ma"
"micromajor".substring(5)     // "major"
```

提取指定数目的字符 substr()

substr() 方法从字符串中提取从 startPos 位置开始的指定数目的字符串。

语法:

```
stringObject.substr(startPos,length)
```

参数说明:

参数	描述
startPos	必需。要提取的子串的起始位置。必须是数值。
length	可选。提取字符串的长度。如果省略，返回从 stringObject 的开始位置 *startPos* 到 stringObject 的结尾的字符。

注意：如果参数 startPos 是负数，从字符串的尾部开始算起的位置。也就是说，-1 指字符串中最后一个字符，-2 指倒数第二个字符，以此类推。

如果 startPos 为负数且绝对值大于字符串长度，startPos 为 0。

```
案例："micromajor".substr(5,2)  // "ma"
"micromajor".substr(5)     // "major"
```

截取字符串-str.slice()

语法：*stringObj*.**slice(***start*, [*end*]**)**

slice 方法一直复制到 end 所指定的元素，但是不包括该元素。如果 start 为负，将它作为 length + start 处理，此处 length 为数组的长度。如果 end 为负，就将它作为 length

+end 处理，此处 length 为数组的长度。如果省略 end ，那么 slice 方法将一直复制到 arrayObj 的结尾。如果 end 出现在 start 之前，不复制任何元素到新数组中。

案例：

```
"micromajor".slice(5,7)    // "ma"
"micromajor".slice(5)      // "major"
"micromajor".slice(1,-1)   // "icromajo"
"micromajor".slice(-3)     // "jor"
```

转义字符

以反斜杠“\”开头的不可显示的特殊字符通常称为转义字符。通过转义字符可以在字符串中添加不可显示的特殊字符，或者防止引号匹配混乱的问题。

JavaScript 常用的转义字符

转义字符	说明
\b	退格
\n	回车换行
\t	Tab 符号
\f	换页
\'	单引号
\"	双引号
\v	跳格（Tab，水平）
\r	换行
\\	反斜杠
\OOO	八进制整数，范围为 000~777
\xHH	十六进制整数，范围为 00~FF
\uhhhh	十六进制编码的 Unicode 字符

这张表列举了 JavaScript 常用的转义字符，根据个人的开发经验中，只需要记忆\n、\'、\"这 3 个就可以了。

例如，要输入我爱”JavaScript“，

这样写是错误的：document.write(“我爱”JavaScript”“);

用转义字符这样写是正确的：document.write(“我爱\”JavaScript\”“);

```
案例："micro\"major"   // "micro"major"
"micro\\major"   // "micro\major"
"micro\tmajor"   // "micro major"
```

九、JS 对象

JavaScript 中的几乎所有事务都是对象：字符串、数字、数组、日期、函数，等等。你也可以创建自己的对象。JavaScript 提供多个内建对象，比如 String、Date、Array 等等。

每个对象带有**属性**和**方法**。

对象的属性：反映该对象某些特定的性质的，如：字符串的长度、图像的长宽等；

对象的方法：能够在对象上执行的动作。例如，表单的“提交”(Submit)，时间的“获取”(getYear)等；

理解：对象是拥有属性和方法的集合

创建对象的方式（两种）：

```
obj = new Object([value])

Obj = {};
```

*Value 是*可选项，可以是任意一种 JScript 基本数据类型。(Number、Boolean、或 String。)如果 value 为一个对象，返回不作改动的该对象。如果 *value* 为 **null**、**undefined**，或者没有给出，则产生没有内容的对象。

示例：

```
var o = new Object(123);
console.log(o); // => Number 对象
o = new Object(true);
console.log(o); // => Boolean 对象
o = new Object('abc');
console.log(o); // => String 对象
```

对象属性和方法：

通过键值对的方式来创建属性和方法，通过”.”或字符串来获取对象的属性和方法，例如：

```
var car = {
color : "red",
run : function(){alert("run")}
};
car.color;       // "red"
car.run();       // alert("run")
car["color"];   // "red"
car["run"]();   // alert("run")
```

增加属性和方法：

```
var car = {
color : "red",
run : function(){alert("run")}
};
car.type = "suv";
car.stop = function(){alert("stop")};
```

修改属性和方法：

通过重新给属性和方法赋值即可，通过“=”。

```
var car = {
color : "red",
run : function(){alert("run")}
};
car.color = "white";
car.run = function(){alert("run2")};
```

删除属性和方法-delete

```
var car = {
color : "red",
run : function(){alert("run")}
};
delete car.color;
car.color;   // undefined
```

Obj.constructor

constructor 属性始终指向创建当前对象的构造函数。

```
var car = {
color : "red",
run : function(){alert("run")}
};
car.constructor;    // Object
```

```
// 等价于 var foo = new Array(1, 56, 34, 12);
var arr = [1, 56, 34, 12];
console.log(arr.constructor === Array); // true
```

obj.toString()-把对象转成字符串

```
var num = new Number(123);
num.toString();   // "123"
```

obj.valueOf() ---获取对象的原始值

```
var num = new Number(123);
num.valueOf();    // 123
```

obj.hasOwnproperty

该方法返回一个布尔值（true/false），指出一个对象是否具有指定名称的属性。如果 *object* 具有指定名称的属性，那么 **hasOwnProperty** 方法返回 **true**；反之则返回 **false**。此方法无法检查该对象的原型链中是否具有该属性；该属性必须是对象本身的一个成员。

```
var car = {
color : "red",
run : function(){alert("run")}
};
car.hasOwnProperty("color");   // true
car.hasOwnProperty("logo");    // false
```

十、JS 数组

数组对象是一个对象的集合，里边的对象可以是不同类型的。数组的每一个成员对象都有一个“下标”，用来表示它在数组中的位置，是从零开始的

数组定义及使用

数组定义的方法：

1. 定义了一个空数组:

```
var  数组名= new Array(); 或 var 数组名= [];
```

2. 定义时指定有 n 个空元素的数组：

```
var 数组名 =new Array(n);
```

3. 定义指定长度数组并赋值：

```
var myArr = new Array(1,2,3,4);
```

说明：上面定义并创建了 myArr 的数组，包含了 4 个元素：1、2、3、4。其中 myArr[0]=1、myArr[1]=2、myArr[2]=3、myArr[3]=4。

3.定义数组的时候，直接初始化数据：

```
var  数组名 = [<元素 1>, <元素 2>, <元素 3>...];
```

我们定义 myArray 数组，并赋值，**代码如下：**

```
var myArray = [2, 8, 6];
```

说明：定义了一个数组 myArray，里边的元素是：myArray[0] = 2; myArray[1] = 8; myArray[2] = 6。

数组元素使用/修改

```
数组名[下标] = 值;
```

注意: 数组的下标用方括号括起来，从 0 开始。

数组属性

length 用法：<数组对象>.length；返回：数组的长度，即数组里有多少个元素。它等于数组里最后一个元素的下标加一。

数组方法：

方法	描述
concat()	连接两个或更多的数组，并返回结果。
join()	把数组的所有元素放入一个字符串。元素通过指定的分隔符进行分隔。
pop()	删除并返回数组的最后一个元素
push()	向数组的末尾添加一个或更多元素，并返回新的长度。
reverse()	颠倒数组中元素的顺序。
shift()	删除并返回数组的第一个元素
slice()	从某个已有的数组返回选定的元素
sort()	对数组的元素进行排序
splice()	删除元素，并向数组添加新元素。
toSource()	返回该对象的源代码。
toString()	把数组转换为字符串，并返回结果。
toLocaleString()	把数组转换为本地数组，并返回结果。
unshift()	向数组的开头添加一个或更多元素，并返回新的长度。
valueOf()	返回数组对象的原始值

arr.indexOf()

arrObj.**indexOf**(*searchElement*[, *startIndex*])

返回数组对象内数组元素首次出现的位置。

示例：

```
var telephones = [110,120,114];
telephones.indexOf(120);   // 1
telephones.indexOf(119);   // -1
```

arr.forEach(callback) ---数组循环遍历

语法：array.forEach(callback[, thisObject]);

- callback： 要对每个数组元素执行的回调函数。
- thisObject ：可选， 在执行回调函数时定义的 this 对象。

callback 的参数：

```
[].forEach(function(value, index, array) {
    // ...
});
```

这个语法平时最多还是用来遍历数组，这时候每一个循环得到的是数组的索引(一个整形数字),然后通过数组名[整形索引]获得数组中的对象。

但是这个语法还可以用来遍历对象，拿到的是对象的属性名称(一个字符串).然后通过对象名[属性名称]就可以拿到对象。

forEach 方法中的 function 回调支持 3 个参数，第 1 个是遍历的数组内容；第 2 个是对应的数组索引，第 3 个是数组本身。

示例：

```
var students = [
{id:1,score:80},
{id:2,score:50},
{id:3,score:70}
];
var editScore = function(item,index,array){
item.score += 5;
};
students.forEach(editScore); //使数组中的每个元素的 score 加 5.
```

reverse()方法---数组元素反向排列

在 JavaScript 中，我们可以使用 Array 对象的 reverse()方法将数组中的元素反向排列。注意，**reverse()是一种“排列”方法，而不是“排序”方法。**

说明：reverse()方法会改变原来的数组，而不是创建新的数组。

案例：

```
var students = [
{id:1,score:80},
```

```
{id:2,score:50},
{id:3,score:70}
];
students.reverse();
students[0].score;   // 70
```

sort()方法---数组元素比较排序

在 JavaScript 中 ,我们可以使用 Array 对象的 sort()方法对数组元素进行大小比较排序。

语法：数组对象.sort(函数名)

说明：

其中“**函数名”用来确定元素顺序的函数的名称**，如果这个参数被省略，那么元素将按照 unicode **字符编码顺序**进行升序排序。

1.如果调用该方法时没有使用参数，将按字母顺序对数组中的元素进行排序，说得更精确点，是按照**字符编码的顺序进行排序**。要实现这一点，首先应把数组的元素都转换成字符串（如有必要），以便进行比较。

这时候，字符串的大小是从**最左边第一个字符开始比较，大者为大，小者为小**，若相等，则继续比较后面的字符。

2. 如果想按照其他标准进行排序，就需要提供比较函数，该函数要比较两个值，然后返回一个用于说明这两个值的相对顺序的数字。比较函数应该具有两个参数 a 和 b，其返回值如下：

- 若 a 小于 b，则返回一个小于 0 的值。表示 A 在排序后的序列中出现在 B 之前。
- 若 a 等于 b，则返回 0。则表示 A 和 B 具有相同的排序顺序。
- 若 a 大于 b，则返回一个大于 0 的值，表示 A 在排序后的序列中出现在 B 之后。

比较函数写法：

```
function asc(a,b)
        {
            return a-b;
        }
        //降序比较函数
        function des(a,b)
        {
            return b-a;
        }
arr.sort(asc) //对数组进行升序排列;
```

```
arr.sort(des) //对数组进行降序排列；
```

案例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <meta charset="utf-8">
    <script type="text/javascript">
        //升序比较函数
        function asc(a,b)
        {
            return a-b;
        }
        //降序比较函数
        function des(a,b)
        {
            return b-a;
        }
        //创建数组的同时对元素赋值
        var arr=new Array(3,9,1,12,50,21);
        arr.sort();
        document.write("省略比较函数的数组元素排序"+arr.join(","));
        document.write("<br>");  // 1,12,21,3,50,9
        arr.sort(asc);
        document.write("升序后的数组元素"+arr.join(","));
        document.write("<br/>");//1,3,9,12,21,50
        arr.sort(des);
        document.write("降序后的数组元素"+arr.join(","));
        //50,21,12,9,3,1
    </script>
</head>
<body>
</body>
</html>
```

输出结果为：

```
省略比较函数的数组元素排序 1,12,21,3,50,9
升序后的数组元素 1,3,9,12,21,50
降序后的数组元素 50,21,12,9,3,1
```

unshift()方法---在数组开头添加元素

在 JavaScript 中 ,我们可以使用 Array 对象的 unshift()方法在数组开头添加元素，并返回该数组。

语法：数组对象.unshift(新元素 1,新元素 2,…,新元素 n);

示例：

```
var students = [
{id:1,score:80},
{id:2,score:50},
{id:3,score:70}
];
students.unshift({id:4,score:90});
```

push()方法---在数组末尾添加元素

在 JavaScript 中，我们可以使用 Array 对象的 push()方法向数组的末尾追加一个或多个元素 ,并且返回新的长度(把指定的值添加到数组后的新长度，数组长度改变。)。记住 ,push()方法是在数组的末尾添加元素，而不是在中间插入元素。

语法：数组对象.push(新元素 1,新元素 2,…,新元素 n);

push()方法可以把新的元素按照顺序添加到数组对象中去，它直接修改数组对象，而不是创建一个新的数组。push()方法和 pop()方法使用数组提供的先进后出的功能。

示例：

```
var students = [
{id:1,score:80},
{id:2,score:50},
{id:3,score:70}
];
students.push({id:4,score:90},{id:5,score:60});
```

shift()方法---删除数组中第一个元素

在 JavaScript 中，我们可以使用 Array 对象的 shift()方法来删除数组中第一个元素，并且返回第一个元素的值。

语法：数组对象.shift();

说明：

shift()方法跟 pop()方法类似。其中 unshift()方法用于在数组开头添加元素 ,shift()方法用于删除数组开头第一个元素。

注意，shift()方法不创建新的数组，而是直接修改原来的数组对象。如果数组为空，那么 shift()方法将不会进行任何操作，并且返回 undefined 值。

示例：

```
var students = [
{id:1,score:80},
{id:2,score:50},
{id:3,score:70}
];
students.shift();   // {id:1,score:80}
```

pop()方法---删除数组最后一个元素

在 JavaScript 中 ,我们可以使用 Array 对象的 pop()方法删除并返回数组中的最后一个元素。

语法：数组对象.pop();

说明：

pop()方法将删除数组对象的最后一个元素，并且把数组长度减 1，返回它删除的该元素的值。如果数组已经为空，则 pop()方法不改变数组，并返回 undefined 值。

示例：

```
var students = [
{id:1,score:80},
{id:2,score:50},
{id:3,score:70}
];
students.pop();   // {id:3,score:70}
```

slice()方法---获取数组中的某段数组元素

在 JavaScript 中，我们可以使用 Array 对象的 slice()方法来获取数组中的某段数组元素。slice，就是“切片”的意思。

语法：

```
数组对象.slice(start,end)
```

说明：

参数 start 和 end 都是整数。其中，参数 start 是必选项，表示开始元素的位置，是从 0 开始计算的。参数 end 是可选项，表示结束元素的位置，也是从 0 开始计算的。

示例：

```
var students = [
{id:1,score:80},
{id:2,score:50},
{id:3,score:70}
];
var newStudents = students.slice(0,2);
```

splice()方法---从数组中移除一个或多个元素

从一个数组中移除一个或多个元素，如果必要，在所移除元素的位置上插入新元素，返回所移除的元素。

语法：

arrayObj.**splice(***start, deleteCount, [item1[, item2[,...[,itemN]]]]***)**

- Start 必选项。指定从数组中移除元素的开始位置，这个位置是从 0 开始计算的。

- deleteCount 必选项。要移除的元素的个数。
- item1, item2,. . .,itemN 可选项。要在所移除元素的位置上插入的新元素。

splice 方法可以移除从 start 位置开始的指定个数的元素并插入新元素，从而修改 arrayObj。返回值是一个由所移除的元素组成的新 Array 对象。

示例：

```
var students = [
{id:1,score:80},
{id:2,score:50},
{id:3,score:70}
];
students.splice(1,1,{id:4,score:90}); //替换掉第二个数组元素。

var students = [
{id:1,score:80},
{id:2,score:50},
{id:3,score:70}
];
students.splice(1,1);   //删除第二个数组元素。

var students = [
{id:1,score:80},
{id:2,score:50},
{id:3,score:70}
];
students.splice(1,0,{id:4,score:90});   //添加数组元素为第二个数组元素，其他顺延。
```

join()方法---将数组元素连接成字符串

在 JavaScript 中 ,我们可以使用 Array 对象的 join()方法把数组中的**所有元素连接成为一个字符串**。

语法：数组对象.join("分隔符")

说明：

其中分隔符是可选项，用于指定要使用的分隔符。如果省略该参数，则 JavaScript 默认采用**英文逗号**作为分隔符。

示例：

```
var emails = ["wq@163.com","gp@163.com","xl@163.com"];
emails.join(";");  // "wq@163.com;gp@163.com;xl@163.com"
```

concat()方法---多个数组连接为字符串

在 JavaScript 中，我们可以使用 Array 对象的 concat()方法连接两个或多个数组。该方法不会改变现有的数值，而仅仅会返回被连接数组的一个副本。

concat，就是“合并”的意思。

语法：数组 1.concat(数组 2,数组 3,…,数组 n)

（也可以直接用+号来连接多个数组，如：数组 1+数组 2+数组 3+……+数组 n）

示例：

```
var students1 = [
{id:1,score:80},
{id:2,score:50},
{id:3,score:70}
];
var students2 = [
{id:4,score:90},
{id:5,score:60}
];
var students3 = [
{id:6,score:40},
{id:7,score:30}
];
var newStudents = students1.concat(students2,students3);
```

map()方法---映射，原数组被“映射”成对应新数组

语法：`array.map(callback[, thisObject]);`

- callback： 要对每个数组元素执行的回调函数。
- thisObject ： 在执行回调函数时定义的 this 对象。

`callback` 的参数：

```
[].map(function(value, index, array) {
    // ...
});
```

对数组中的每个元素都执行一次指定的函数（callback），并且以每次返回的结果为元素**创建一个新数组**。它只对数组中的非空元素执行指定的函数，没有赋值或者已经删除的元素将被忽略。

回调函数可以有三个参数：当前元素，当前元素的索引和当前的数组对象。**如参数 thisObject 被传递进来，它将被当做回调函数（callback）内部的 this 对象，如果没有传递或者为 null，那么将会使用全局对象。**

map 不会改变原有数组，记住：只有在回调函数执行前传入的数组元素才有效，在回调函数开始执行后才添加的元素将被忽略，而在回调函数开始执行到最后一个元素这一期间，数组元素被删除或者被更改的，将以回调函数访问到该元素的时间为准，被删除的元素将被忽略。

```
示例：var scores = [60,70,80,90];
var addScore = function(item,index,array){
return item+5;
};
scores.map(addScore);   // [65,75,85,95]
```

Reduce 方法---

语法：`array.reduce(callback[, initialValue])`

Callback 函数的参数：

```
[].reduce(function(previous, current, index, array) {
    // ...
});
```

`callback` 函数接受 4 个参数：之前值、当前值、索引值以及数组本身。`initialValue` 参数可选，表示初始值。若指定，则当作最初使用的 `previous` 值；如果缺省，则使用数组的第一个元素作为 `previous` 初始值，同时 `current` 往后排一位，相比有 `initialValue` 值少一次迭代。

```
var sum = [1, 2, 3, 4].reduce(function (previous, current, index, array)
{
  return previous + current;
});
console.log(sum); // 10
```

说明：

1. 因为 `initialValue` 不存在，因此一开始的 `previous` 值等于数组的第一个元素。
2. 从而 `current` 值在第一次调用的时候就是 `2`.
3. 最后两个参数为索引值 `index` 以及数组本身 `array`.

以下为循环执行过程：

```
// 初始设置
previous = initialValue = 1, current = 2
// 第一次迭代
previous = (1 + 2) =  3, current = 3
// 第二次迭代
previous = (3 + 3) =  6, current = 4
// 第三次迭代
previous = (6 + 4) =  10, current = undefined (退出)
```

示例：

```
var students = [
{id:1,score:80},
{id:2,score:50},
{id:3,score:70}
];
var sum = function(previousResult,item,index,array){
return previousResult+item.score;
};
students.reduce(sum,0);   // 200
```

参考资源：

1. Array 方法: indexOf、forEach、map、reduce 等使用实例

---

http://www.zhangxinxu.com/wordpress/2013/04/es5%E6%96%B0%E5%A2%9E%E6%95%B0%E7%BB%84%E6%96%B9%E6%B3%95/

http://ourjs.com/detail/54a9f2ba5695544119000005