# JavaScript基础入门

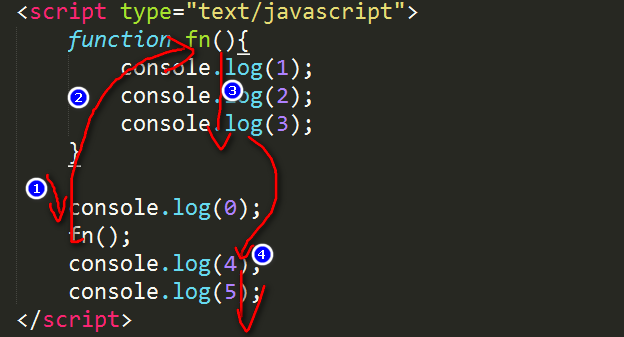
第5天课堂笔记

# 一、上次课知识复习

## 复习

函数：function语句的集合，就是将多个语句封装到一起。

函数的执行，要会自己遍历，遇见fn()执行语句，就要移交控制权。函数执行完毕之后，控制权又移交回来了。



函数的参数要罗列在function定义圆括号中，用逗号隔开，叫做形式参数。调用的时候，圆括号里面是实际参数。

参数在JS中是不用指定类型的，调用的时候参数个数也可以和定义的时候不一样。

|  |
| --- |
| 1. //函数的参数和返回值 2. function sum(a,b){ 3. return a + b; 4. alert("哈哈，你永远看不见我，我在return后面"); 5. } 6. //矮化为表达式了 7. console.log(sum(1,"2")); |

用函数简化程序的书写，简化实际上你要深刻理解：算法效率可能更低了，但是程序可读性强了，程序更好写了。

函数声明头的提升，程序一开始就会有一个预解析的过程，程序会通看全部代码，把所有的函数名字都放到开头预习一下，程序自己知道了，页面上有这个函数定义。但是，函数表达式是不能预解析的。

|  |
| --- |
| 1. //函数声明头的提升 2. fn(); //正常会弹窗，把调用写在前面，因为函数声明头会预解析。 3. fn2(); //报错。函数表达式是不能预解析的 4. function fn(){ 5. alert("我是函数"); 6. } 7. **//函数表达式是不能预解析的** 8. var fn2 = function(){ 9. alert("我是函数2"); 10. } |

函数优先 ： 如果同一个标识符，在程序中又是变量的名字，又是函数的名字，解析器会把标识符给函数。

|  |
| --- |
| 1. a(); 2. var a = 1; 3. function a(){ 4. alert("我是函数"); 5. } |



在a();之前，函数function a()就已经预解析了，a就是函数了，虽然变量a也有一个变量声明头的提升，但是干不过函数声明头的提升，所以此时a()就会执行函数。

|  |
| --- |
| 1. var a = 1; 2. function a(){ 3. alert("我是函数"); 4. } 6. a(); |



在执行var a = 1之前，函数已经把function a()预解析了，程序就已经知道页面上有一个函数叫做a。但是开始执行程序之后，定义了一个变量a，所以标识符a，就又变成变量了。遇见function定义，程序会无视，因为已经预解析了。直到a()运行的时候，a就是变量，无法运行，报错。

|  |
| --- |
| 1. a(); 2. var a = 1; 3. var a = function(){ 4. alert("我是函数"); 5. } |



函数表达式是不会预解析的，所以预解析的就是变量a的定义，就是undefined，undefined是无法执行的。

## 作业题目

**第1题**： 1! + 2! + 3! + …… + 10!

小的单元就是阶乘，大的事情就是把所有阶乘相加。

如果有一个函数，叫做jiecheng(a)，返回的就是a的阶乘，那么：

|  |
| --- |
| 1. var sum = 0; 2. for(var i = 1 ; i <= 10 ; i++){ 3. sum += jiecheng(i); 4. } |

函数需要我们自己编写

**第2题**：找一个三位数，各个数位的阶乘和，等于自己

这个题目又用了一次阶乘函数。一次定义多次使用，底层的函数（轮子）是一样的，高层的业务是不同的。

|  |
| --- |
| 1. for(var i = 100 ; i <= 999 ; i++){ 2. var gewei = i % 10; 3. var shiwei = parseInt(i / 10) % 10; 4. var baiwei = parseInt(i / 100); 5. if(multiple(gewei) + multiple(shiwei) + multiple(baiwei) == i){ 6. console.log(i); 7. } 8. } |

**第3题**：编写一个函数rev(a)，接收一个参数整数a，能够返回a的逆序。

rev(33657) 返回75633

并且位数不限，rev(12)返回21。

方法至少有5种，现在只用数学方法来制作。

|  |
| --- |
| 1. function rev(a){ 2. var b = 0; //每次调用这个函数的时候，b要重置为0 3. while(a){ 4. b = b \* 10 + a % 10; //b自己乘以10然后加上a的末位数字 5. a = parseInt(a / 10); //a自己变短一位数，去掉末位 6. } 7. return b; 8. } |

用数学方法制作一个新的题目：

封装一个函数rev2(a)，接收一个整数a，能够：

rev(1234)，返回12344321

**第4题**：找“亲密数”A-B如果是亲密数，那么A的所有约数和等于B，B的所有约数和等于A。注意所有的约束和不包括自己。

有一个新的算法思想，叫做“定甲试乙”

比如，现在认为A是8，那么B是8的所有约数和，就是1+2+4=7。

验算7的约束和是否是8， 7的约数和是1，

所以8和7就不是亲密数了，8和任何数字都不是亲密数。

# 二、作用域

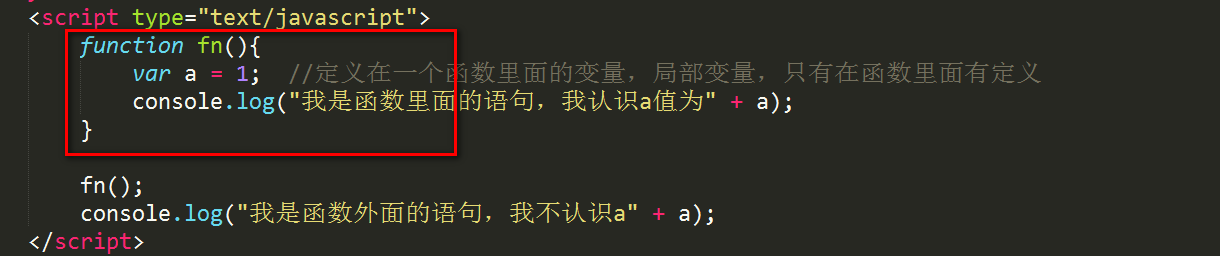
## 2.1 函数能封闭住定义域

一个变量如果定义在了一个function里面，那么这个变量就是一个局部变量，只在这个function里面有定义。出了这个function，就如同没有定义过一样。

|  |
| --- |
| 1. <script type="text/javascript"> 2. function fn(){ 3. var a = 1; //定义在一个函数里面的变量，局部变量，只有在函数里面有定义 4. console.log(**"我是函数里面的语句，我认识a值为" + a**); 5. } 6. fn(); 7. console.log(**"我是函数外面的语句，我不认识a" + a**); 8. </script> |



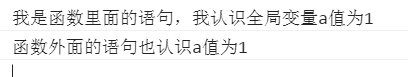
a被var在了function里面，所以现在这个a变量只在红框范围内有定义：



JavaScript变量作用域非常的简单，没有块级作用域，管理住作用域的只有一个东西：函数。

如果一个变量，没有定义在任何的function中，那么它将在全部程序范围内都有定义：

|  |
| --- |
| 1. **var a = 1;** //定义在全局范围内的一个变量，全局变量，在程序任何一个角落都有定义 2. function fn(){ 3. console.log("我是函数里面的语句，我认识全局变量a值为" + a); 4. } 5. fn(); 6. console.log("函数外面的语句也认识a值为" + a) |



总结一下：

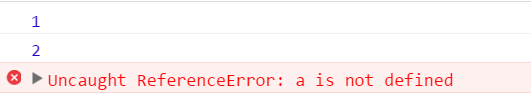
● 定义在function里面的变脸，叫做局部变量，只在function里面有定义，出了function没有定义的。

● 定义在全局范围内的，没写在任何function里面的，叫做全局变量，都认识。

## 2.2 作用域链

当遇见一个变量时，JS引擎会从其所在的作用域依次向外层查找，查找会在找到第一个匹配的标识符的时候停止。

|  |
| --- |
| 1. function outer(){ 2. var a = 1; //a的作用域就是outer 3. inner(); 4. function inner(){ 5. var b = 2; //b的作用域就是inner 6. console.log(a); //能够正常输出1，**a在本层没有定义，就是找上层** 7. console.log(b); //能够正常输出2 8. } 9. } 10. outer(); 11. console.log(a); //报错，因为a的作用域outer |



多层嵌套，如果有同名的变量，那么就会发生“遮蔽效应”：

|  |
| --- |
| 1. **var a = 1;**  //全局变量 2. function fn(){ 3. **var a = 5; //就把外层的a给遮蔽了，这函数内部看不见外层的a了。** 4. console.log(a); //输出5，变量在当前作用域寻找，找到了a的定义值为5 5. } 6. fn(); 7. console.log(a); //输出1,变量在当前作用域寻找，找到了a的定义值为1 |



作用域链：一个变量在使用的时候得几呢？就会在当前层去寻找它的定义，找不到，找上一层function，直到找到全局变量，如果全局也没有，就报错。

|  |
| --- |
| 1. var a = 1; //全局变量 2. var b = 2; //全局变量 3. function outer(){ 4. var a = 3; **//遮蔽了外层的a，a局部变量** 5. function inner(){ 6. var b = 4; **//遮蔽了外层的b，b局部变量** 7. console.log(a); //① 输出3,a现在在当前层找不到定义的，所以就上一层寻找 8. console.log(b); //② 输出4 9. } 10. inner(); //调用函数 11. console.log(a); //③ 输出3 12. console.log(b); //④ 输出2 b现在在当前层找不到定义的，所以就上一层寻找 13. } 14. outer(); //执行函数，控制权交给了outer 15. console.log(a); // ⑤ 输出1 16. console.log(b); // ⑥ 输出2 |

## 2.3 不写var就自动成全局变量了

|  |
| --- |
| 1. function fn(){ 2. a = 1; **//这个a第一次赋值的时候，并没有var过，** 3. **//所以就自动的在全局的范围帮你var了一次** 4. } 5. fn(); 6. console.log(a); |

这是JS的一个机理，如果遇见了一个标识符，从来没有var过，并且还赋值了：

|  |
| --- |
| 1. absdf = 123; |

那么就会自动帮你在全局范围内定义var absdf;

告诉我们一个道理，变量要老老实实写var。

## 2.4 函数的参数，会默认定义为这个函数的局部变量

|  |
| --- |
| 1. function fn(a,b,c,d){ 2. } |

a,b,c,d就是一个fn内部的局部变量，出了fn就没有定义。

## 2.5 全局变量的作用

全局变量挺有用的，有两个功能：

功能1：通信，共同操作同一个变量

两个函数同时操作同一个变量，一个增加，一个减少，函数和函数通信。

|  |
| --- |
| 1. var num = 0; 2. function add(){ 3. num++; 4. } 5. function remove(){ 6. num--; 7. } |

功能2：累加，重复调用函数的时候，不会重置

|  |
| --- |
| 1. var num = 0; 2. function baoshu(){ 3. num++; 4. console.log(num); 5. } 6. baoshu(); //1 7. baoshu(); //2 8. baoshu(); //3 |

如果num定义在baoshu里面，每次执行函数就会把num重置为0：

|  |
| --- |
| 1. function baoshu(){ 2. **var num = 0;** 3. num++; 4. console.log(num); 5. } 6. baoshu(); **//1** 7. baoshu(); **//1** 8. baoshu(); **//1** |

## 2.6 函数的定义也有作用域

|  |
| --- |
| 1. //这个函数返回a的平方加b的平方 2. function pingfanghe(a,b){ 3. return pingfang(a) + pingfang(b); 4. //返回m的平方 5. **function pingfang(m){** 6. **return Math.pow(m,2)** 7. **}** 8. } 9. // 现在相求4的平方，想输出16 10. **pingfang(4);** //报错，因为全局作用域下，没有一个函数叫做pingfang |

公式：

|  |
| --- |
| 1. function 大{ 2. function 小{ 3. } 4. 小(); //可以运行 5. } 6. 小(); //不能运行，因为小函数定义在了大函数里面，离开大函数没有作用域。 |

# 三、闭包

## 3.1 闭包

任何的培训机构，任何的书，讲闭包，一定是下面的案例：

|  |
| --- |
| 1. function outer(){ 2. var a = 333; 3. function inner(){ 4. console.log(a); 5. } 6. return inner; 7. } 8. var inn = outer(); 9. inn(); //弹出333 |

推导过程：

我们之前已经学习过，inner()这个函数不能在outer外面调用，因为outer外面没有inner的定义：

|  |
| --- |
| 1. function outer(){ 2. var a = 888; 3. function inner(){ 4. console.log(a); 5. } 6. } 7. **//在全局调用inner但是全局没有inner的定义，所以报错** 8. inner(); |

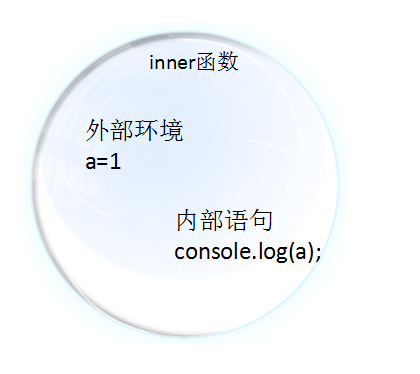
但是我们现在就想在全局作用域下，运行outer内部的inner,此时我们必须想一些奇奇怪怪的方法。

有一个简单可行的办法，就是让outer自己return掉inner：

|  |
| --- |
| 1. function outer(){ 2. var a = 333; 3. function inner(){ 4. console.log(a); 5. } 6. **return inner;** //outer返回了inner的引用 7. } 8. var **inn** = outer(); //inn就是inner函数了 9. inn(); //执行inn，全局作用域下没有a的定义 10. //但是函数闭包，能够把定义函数的时候的作用域一起记忆住 11. //能够输出333 |

这就说明了，inner函数能够持久保存自己定义时的所处环境，并且即使自己在其他的环境被调用的时候，依然可以访问自己定义时所处环境的值。

一个函数可以把它自己内部的语句，和自己声明时所处的作用域一起封装成了一个密闭环境，我们称为“闭包” （Closures）。



**每个函数都是闭包，每个函数天生都能够记忆自己定义时所处的作用域环境**。但是，我们必须将这个函数，挪到别的作用域，才能更好的观察闭包。这样才能实验它有没有把作用域给“记住”。

**我们发现，把一个函数从它定义的那个作用域，挪走，运行。嘿，这个函数居然能够记忆住定义时的那个作用域**。**不管函数走到哪里，定义时的作用域就带到了哪里。这就是闭包。**

闭包在工作中是一个用来防止产生隐患的事情，而不是加以利用的性质。

因为我们总喜欢在函数定义的环境中运行函数。从来不会把函数往外挪。那为啥学习闭包，防止一些隐患，面试绝对考。

## 3.2 闭包的性质

每次重新引用函数的时候，闭包是全新的。

|  |
| --- |
| 1. function outer(){ 2. var count = 0; 3. function inner(){ 4. count++; 5. console.log(count); 6. } 7. return inner; 8. } 9. var inn1 = outer(); 10. var inn2 = outer(); 11. inn1(); //1 12. inn1(); //2 13. inn1(); //3 14. inn1(); //4 15. inn2(); //1 16. inn2(); //2 17. inn1(); //5 |

无论它在何处被调用，它总是能访问它定义时所处作用域中的全部变量

# 四、数组

## 4.1 数组概念

数组(array)是一个有序的数据集合。说白了，数组就是一组数。

|  |
| --- |
| 1. var arr = [16,33,23,12,53]; |

**数组的字面量就是方括号**，这是定义数组的最简单的方式。至于更多创建数组的方式，我们等到高级JS再学。

里面用逗号隔开了很多值，最后一项没有逗号。

变量a就是一个数组变量，里面存储的不是一个数字，而是一组数。可以使用下标，或称为索引值index来精确访问数组中的某一个项，下标从0开始。

|  |
| --- |
| 1. console.log(**arr[0]**); //输出16 2. console.log(**arr[1]**); //输出33 3. console.log(**arr[4]**); //输出53 |

数组中并不规定保存相同类型的项，**但实际应用中，我们一般还是将相同类型的项保存在其中**。

下面的数组中，存储的内容类型都是不一样的是合法的。

|  |
| --- |
| 1. var arr = [1,"哈哈",true,undefined,"么么哒"]; |

数组有一个属性，就是length，英语是长度的意思，表示这个数组的项的个数。

先说说什么是“属性”，数组是对象（放心里，后面高级JS课我们就会知道这个事情），对象有属性，属性就是描述这个对象的特点、特性、特征。用点来表示一个对象的属性：

|  |
| --- |
| 1. arr.length |

|  |
| --- |
| 1. var arr = [34,563,4576,334,46,433]; 2. alert(arr.length); **//6，数组里面一共有6项** |

数组中有几项，那么就会弹出几。

现在我们发现，数组最后一项的下标，是length-1。

|  |
| --- |
| 1. arr[arr.length - 1] //表示数组的最后一项 |

数组的最大下标是arr.length - 1，尝试输出一个大于这个数字的下标：

|  |
| --- |
| 1. arr[100]; //undefined; |

尝试比length-1还大的下标，就是undefined。

好玩儿的是，我们可以跳跃着指定数组：

|  |
| --- |
| 1. arr[66] = 88; |

那么此时其他没有指定的项目就是undefined。数组的

|  |
| --- |
| 1. arr.length; //67了，我们设置了下标为66的项，强制“抻长”了数组 |

写一个小于数组元素数量的值会缩短数组，写0会彻底清空数组。数组我们现在arr.length = 2;

那么它就会有两项a[0]、a[1]其他项就丢失了。

## 4.2 数组的遍历

数组里面存放的是一组数，那么我们经常需要对这些数字都进行一些操作。

就需要用for循环语句来遍历它。这是一个经典的for壳子：

|  |
| --- |
| 1. for(var i = 0 ; i <= arr.length - 1 ; i++){ 2. 就可以对arr[i]来进行操作、判断…… 3. } |

## 4.3 数组是一个引用类型

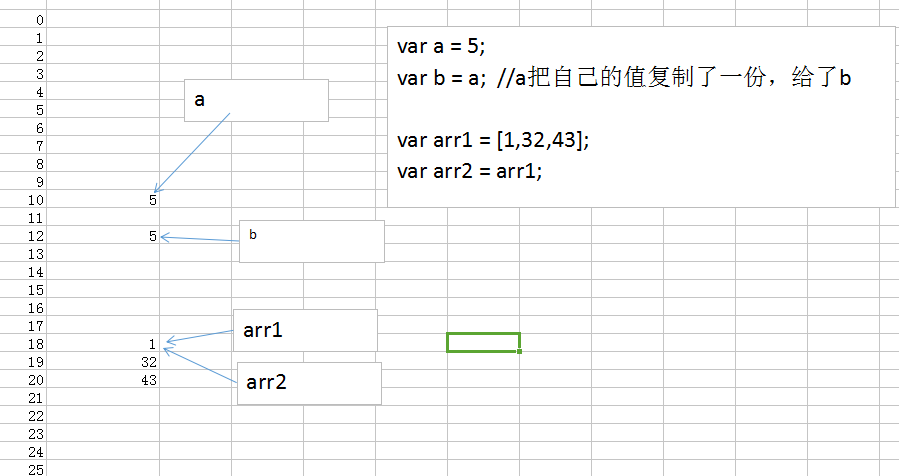
|  |
| --- |
| 1. var arr = [1,2,3,4]; 2. console.log(typeof arr); //object |

用typeof arr来检测，你会发现数组输出object。数组是对象。

那怎么检测数组是数组呢？高级JS课程再说。

保存数组的变量，实际上保存是数组内存地址：

|  |
| --- |
| 1. var a = [1,2,3,4]; 2. var b = a; 3. b[0] = 88; //修改的是数组b下标为0的项 4. console.log(a); //数组a的下标为0的项也改为88 |



都是数组[1,2,3]内容、长度、项的位置完全一样，但是不==。

|  |
| --- |
| 1. var a = [1,2,3]; 2. var b = [1,2,3]; 3. console.log(a == b); |

**那时因为引用类型比较的是地址，变量a和变量b指向的位置不一样，不能判相等**。

如果a里面存储的是基本类型，那么语句b=a就是把a的值赋值一份给b。如果a里面存储的是引用类型，那么b将指向a现在指向的位置，a的值不会赋值一份，a、b指向同一个地方。

# 五、数组的常见方法

数组是对象，现在你要知道对象都有属性和方法。

属性已经介绍了，数组有length属性。属性就是描述对象的特点的，比如“性别”、“姓名”、“身高”

方法就是对象能够执行的事情。比如“吃饭”、“睡觉”、“打dota”

我们现在就要来学数组能执行什么方法。



## 5.1 数组的头尾操作pop()、push()、shift()、unshift()

push()方法，在数组的末尾添加项目，可以添加1个，也可以添加多个。

|  |
| --- |
| 1. var arr = ["东","西","南","北"]; 2. arr.push("中","发","白"); //在数组最后添加项 3. console.log(arr); |

push就是推的意思，推入。



pop()方法，删除数组的最后一项，只能删除最后一项，无法删除多项。能够返回被删除的元素。

push 尾插

pop 尾删

unshift 头插

shift 头删

## 5.2 数组的合并与拆分 concat()、slice()

concat()方法就是合并两个数组

|  |
| --- |
| 1. var arr1 = ["东","西","南","北"]; 2. var arr2 = ["一条","二条"]; 3. arr1.concat(arr2); //这里有一个超级大坑，concat是把arr1和arr2合并为一个新数组返回 4. console.log(arr1); //不变 |

所以必须：

|  |
| --- |
| 1. **arr1 =** arr1.concat(arr2); |

concat()的参数非常灵活，可以是数组变量、数组字面量、散的值也行：

|  |
| --- |
| 1. var arr1 = ["东","西","南","北"]; 2. var arr2 = ["一条","二条"]; 3. arr1 = arr1.concat(arr2,["一筒","八条"],"幺鸡"); 4. console.log(arr1); |



slice()方法可从已有的数组中返回选定的元素。

|  |
| --- |
| 1. var arr = ["东","西","南","北","中","发","白"]; 2. var arr2 = arr.slice(1,4); //截取下标为1、2、3的为一个新数组返回 3. console.log(arr2); //["西", "南", "北"] |

arr.slice(start,end) 返回一个新的数组，包含从 start 到 end （不包括该元素）的元素。

只有开头：

|  |
| --- |
| 1. var arr = ["东","西","南","北","中","发","白"]; 2. var arr2 = arr.slice(3); // 从下标为3的项目开始截取后面全部了 3. console.log(arr2); // ["北", "中", "发", "白"] |

slice(a,b)取出了b-a项

## 5.3 多功能splice()插入、删除、替换

我们先确认一个事情，

|  |
| --- |
| 1. arr.splice(3,2,"斑马","骆驼"); |

一旦应用，arr立即改变。并不需要重新复制，换句话说，这个函数不返回新的数组。

|  |
| --- |
| 1. var arr = ["A","B","C",**"D","E"**,"F","G"]; 2. arr.splice(3,2,"X","Y","Z","思密达"); //从数组下标为3开始这项，连数2项，改为…… 3. console.log(arr); |



|  |
| --- |
| 1. // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*插入一些项 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2. var arr = ["A","B","C","D","E","F","G"]; 3. arr.splice(2,**0**,"嘻嘻","哈哈"); /**/插入到下标为2的项前，不删除项目** 4. console.log(arr); |



splice依据参数的多少，和参数是什么，有多功能。现在你要能反应过来。

删除数组的最后8项。

|  |
| --- |
| 1. arr.pop(); 2. arr.pop(); 3. arr.pop(); 4. arr.pop(); 5. arr.pop(); 6. arr.pop(); 7. arr.pop(); 8. arr.pop(); |

简化为：

|  |
| --- |
| 1. for(var i = 1 ; i <= 8 ; i++){ 2. arr.pop(); 3. } |

也可以：

|  |
| --- |
| 1. arr.splice(-8); |

## 5.4 逆序reverse();

reverse()方法就是立即让数组倒置：

|  |
| --- |
| 1. <script type="text/javascript"> 2. var arr = ["A","B","C","D","E","F","G"]; 3. **arr.reverse(); //不需要赋值** 4. console.log(arr); //["G", "F", "E", "D", "C", "B", "A"] 5. </script> |



## 5.5 排序sort()

sort()方法排序

|  |
| --- |
| 1. var arr = ["G","A","C","B","I","H","G","I","B"]; 2. **arr.sort();** 3. console.log(arr); |



|  |
| --- |
| 1. //sort函数默认是按照字符顺序排的，隐式将数字转为string 2. //比字符编码顺序 3. var arr = [23,435,456,23,2,345,2,32,11,324,32,43,65,667,78,43]; 4. arr.sort(); 5. console.log(arr); |



sort()里面有一个参数，这个参数是一个函数。

|  |
| --- |
| 1. arr.sort(function(a,b){ 2. //如果a要放在b前面，那么返回负数 3. //如果a要放在b后面，那么返回正数 4. //如果a和b不区分大小，那么返回0 5. if(a < b){ 6. return -1; 7. }else if(a > b){ 8. return 1; 9. }else if(a == b){ 10. return 0; 11. } 12. }); |

## 5.6 转为字符串

|  |
| --- |
| 1. var arr = [1,2,3,4,5,6,7]; 2. var str = arr.join("★"); 3. console.log(str); |



语法

|  |
| --- |
| 1. var str = arr.join(分隔符); |

如果不写分隔符，那么等价于用逗号分开

|  |
| --- |
| 1. var arr = [1,2,3,4,5,6,7]; 2. var str = arr.join(); 3. console.log(str); |



# 六、字符串的常见属性和方法

属性length就是字符串的长度

|  |
| --- |
| 1. "我爱你365天 yeah".length |

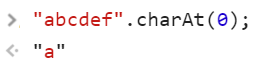
中文、数字、英语字母、空格，都是1个长度

常见方法：

charAt() 返回在指定位置的字符。

|  |
| --- |
| 1. "abcdef".charAt(0); |

和数组下标类似的。



|  |
| --- |
| 1. var str = "abcdefg" 2. for(var i = 0 ; i < str.length ; i++){ 3. console.log(**str.charAt(i)**); 4. } |

concat() 连接字符串。

indexOf() 检索字符串

|  |
| --- |
| 1. "我爱你亲爱的祖国的人民币".indexOf("的"); //5 |

replace() 替换

|  |
| --- |
| 1. "abcdefgaa".replace("a","0"); // "0bcdefgaa" |

把a替换为0 ， 只替换1个。

slice() 提取

|  |
| --- |
| 1. "我爱你亲爱的祖国的人民币".slice(-3,-1) //"人民" |

split() 把字符串→数组， 从什么地方拆分， 就是参数

|  |
| --- |
| 1. "我爱你亲爱**的**祖国**的**人民币".split("的"); |



拆开的是字符串：

|  |
| --- |
| 1. 1嘻嘻2嘻嘻3嘻嘻4嘻嘻5嘻嘻6".split("嘻嘻") |



substr() 截取子串

“字符串”.substr(start,length)

|  |
| --- |
| 1. "abcdefghijklmn".substr(3,5); |

从下标为3的地方开始，取5个字符



substring() 截取子串

“字符串”.substring(start,end); 不包括end

|  |
| --- |
| 1. "abcdefghijklmn".substring(3,5); |



substr和substring不一样：

|  |
| --- |
| 1. "我爱你亲爱的祖国的人民币".substr(6,2) |



|  |
| --- |
| 1. "我爱你亲爱的祖国的人民币".substring(6,8) |



toLowerCase()、toUpperCase();

转换为大小写

|  |
| --- |
| 1. "abcdefghijklmn".toUpperCase(); |

"ABCDEFGHIJKLMN"