# JavaScript基础入门

第6天课堂笔记

# 目录

[JavaScript基础入门 1](#_Toc24732)

[目录 2](#_Toc4266)

[一、上次字符串的作业 3](#_Toc25761)

[二、aruguments 5](#_Toc10822)

[三、IIFE 7](#_Toc2799)

[四、结合数组观察闭包 9](#_Toc12696)

[五、冒泡排序法 10](#_Toc2907)

[六、DOM 11](#_Toc24011)

[6.1 整体感知 11](#_Toc11397)

[6.2 得到元素getElementById 12](#_Toc32427)

[6.3 更改HTML属性 14](#_Toc23532)

[6.4 操作元素样式 15](#_Toc23805)

[6.5 事件监听 16](#_Toc4991)

# 一、上次字符串的作业

设计一个函数，repeat(char,n); 接收两个参数，一个字符串char，一个数数值n。返回重复n次的char。

比如repeat(“★”,10)返回★★★★★★★★★★。

方法1：

|  |
| --- |
| 1. function repeat(char,n){ 2. var str = ""; 3. for(var i = 1 ; i <= n ; i++){ 4. str += char; 5. } 6. return str; 7. } |

字符串如果多，效率问题严重，比如拼接64个星星，应该循环64次。

方法2：

二分法，我们已经有了★★，出现4个五角星，就把★★翻倍就行了。

所以就是把任何数字都要1、2、4、8、16、32……组成的。二进制的事儿。

二进制，只有0、1两个数字，位权的概念：

|  |
| --- |
| 1. 10101 |

1个1 + 0个2 + 1个4 + 0个8 + 1个16 = 21

那么反过来，我们十进制34等于二进制多少呢？32 + 2

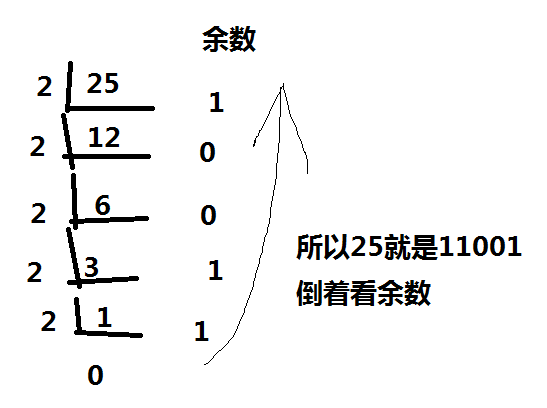
|  |
| --- |
| 1. 100010 |

再比如15等于多少？ 8+4+2+1

|  |
| --- |
| 1. 1111 |

我们现在有没有系统的方法论，把数字25拆分成为1、2、4、8、16、32、64……的和？有，短除法：

短除法为什么能成功？其实很简单，25除以2，余数是1，所以这个25必然蕴含一个1。



实际上短除法就蕴含了一个事儿“奇变偶不变”。

现在要凑25个星星，实际上就是用16个星星+8个星星+1个星星。

就是用计算机模拟短除。

找两个字符串中的最大相同子串：

比如：

str1 = “66666**javascript**888888”;

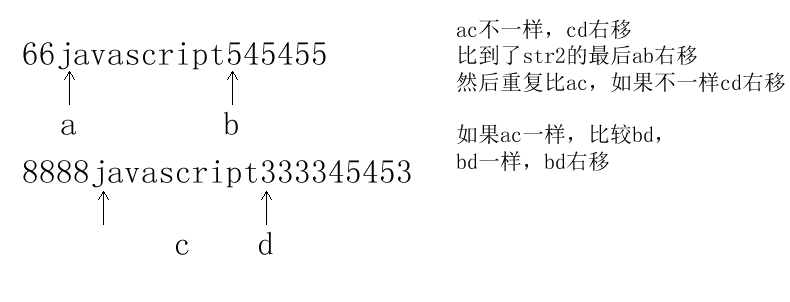
str2=”444**javascript**99”;

里面的最大相同子串javascript

两种思路：

思路1比较简单，就是我们现在遍历str2的所有子串，然后检查是否是str1的一部分，如果是，看看长度，把最长的挑出来。

思路2：四指针法



# 二、aruguments

我们之前学习函数的时候，遗漏这个知识点。

复习一下函数，我们定义函数的时候，参数要罗列在小括号里面，这个参数，叫做形参：

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

调用函数的时候，传参数，这个参数叫做实参。JS不要求形参数量和实参数量一样多：

|  |
| --- |
| 1. fn(1,2); //合法的，虽然少了参数，但是不报错 2. fn(1,2,3,4,5,6,7,8) //合法，多了，也不报错。 |

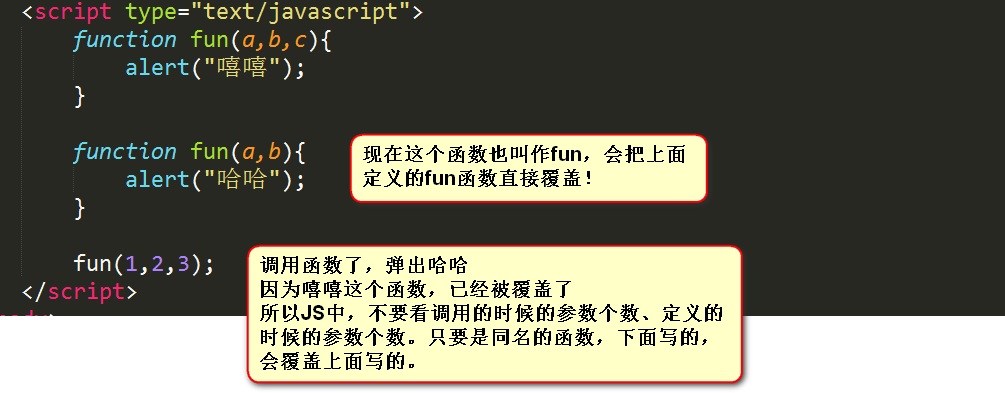
实际上，这个东西叫做方法的重载。在Java中，同一个函数名，但是参数个数不一样，视为是两个函数。

也就是说，Java中能够定义两个：

|  |
| --- |
| 1. function sum(a,b){ 2. } 3. function sum(a,b,c){ 4. } |

同名的两个function，都是sum函数，但是java中是允许这么做的，因为参数个数不一样，这种现象叫做重载。

JavaScript没有重载的概念：



JS有一个非常强大的东西，就是每一个函数内部，都可以使用一个arguments这个类数组对象。

这个arguments对象，就涵盖了所有实参。

调用函数的时候，比如：

|  |
| --- |
| 1. fun(45,436,457,34,23,12); |

此时函数内部，arguments就有一个下标，就依次等于上面调用的数：

|  |
| --- |
| 1. arguments[0] // 45 2. arguments[1] // 436 3. …… 4. arguments[5] //12 |

如果函数里面有形式参数列表，那么是和arguments同步的：

|  |
| --- |
| 1. function fun(a,b){ 2. arguments[0] = 8; //改变了第一个参数的值 3. alert(a); //8 ，弹出改变后的值 4. } 5. fun(45,436,457,34,23,12); |

arguments的功能，是模拟函数的重载，使得同一个函数，根据参数个数的不同，有不同的作用。

比如现在我们想设计一个sum函数，如果传进来一个参数，就得到这个数字的加1；如果是2个参数，那么返回两个数字的和。比如

|  |
| --- |
| 1. sum(10); //1 2. sum(3,4); //7 |

就要通过arguments.length 实际参数的个数，来进行判断：

|  |
| --- |
| 1. function sum(a,b){ 2. //如果实际参数的长度是1，说白了，你只传进来一个参数 3. switch(arguments.length){ 4. case 1: 5. return ++a; 6. break; 7. case 2: 8. return a + b; 9. break; 10. } 11. } |

更牛逼的，我们可以无限参数，设计一个函数sum，能够接受无限参数

|  |
| --- |
| 1. sum(3,4,6,5,8,2) //28 |

# 三、IIFE

IIFE就是immediately-invoked function expression，即时调用函数表达式

如果一个函数，在定义的时候，我们就想直接调用它，就是一个IIFE。

我们试图在定义函数的后面，直接写圆括号：

|  |
| --- |
| 1. function fun(){ 2. alert("哈哈") 3. }(); |

控制台报错，这是因为函数是一个函数体，并不是表达式，只有表达式能够用()来执行。

所以就要把function fun(){}“降级”， 从函数体降级为表达式。方法有很多：

|  |
| --- |
| 1. **+**function fun(){ 2. alert("哈哈") 3. }(); |

|  |
| --- |
| 1. **-**function fun(){ 2. alert("哈哈") 3. }(); |

更通常的：

|  |
| --- |
| 1. **(**function fun(){ 2. alert("哈哈") 3. }**)**(); |

用这种方法定义的函数，名字是无效的，其他的地方想调用这个函数

|  |
| --- |
| 1. fun("嘻嘻"); |

就会报错。

所以IIFE里面的函数，都是匿名函数：

|  |
| --- |
| 1. (function(){ 2. alert("哈哈"); 3. })(); |

上面就是一个标准的IIFE。

设计一个函数，这个函数接收三个参数，比如sum(4,7,9);返回的是前两个数字大的那个数字，与第三个数字的和。

|  |
| --- |
| 1. sum(4,2,3); //7 2. sum(2,4,3); //7 3. sum(5,4,3); //8 |

|  |
| --- |
| 1. function sum(a,b,c){ 2. return (function(a,b){ 3. return a >= b ? a : b; 4. })(a,b) + c; 5. } |

红色部分是一个IIFE，本质上是一个表达式，表达式计算之后，就是值，什么值呢？a、b中大的那个数字。

# 四、结合数组观察闭包

我们说过，数组中，什么都能放。能放string、能放数字。更能放函数。

下面的例子就是经典的数组与闭包的结合：

|  |
| --- |
| 1. var arr = []; //空数组 2. //用循环语句去填充数组里面的每个项 3. for(var i = 0 ; i <= 10 ; i++){ 4. arr[i] = function(){ 5. alert(i); //我们在这里，试图每个函数弹出自己的序号 6. } 7. } 8. //循环语句执行完毕，arr里面有11个函数了，都是弹出么么哒 9. arr[6](); //11 10. arr[9](); //11 11. arr[10](); //11 |

弹出的都是11，而不是预想的6、9、10。原因就是每个函数定义的时候，都产生闭包，函数就认识i了，而不是说把i这个值赋值一份记忆住，而是动态的、有呼吸、有生命的认识这个i，i在调用的时候得几了，这个函数认为i就是几。

所以在调用函数的时候，i已经变为了11，所以每个函数都弹出11了。

解决方法：

|  |
| --- |
| 1. for(var i = 0 ; i <= 10 ; i++){ 2. **(function(m){** 3. arr[**m**] = function(){ 4. alert(**m**); 5. } 6. **})(i);** 7. } |

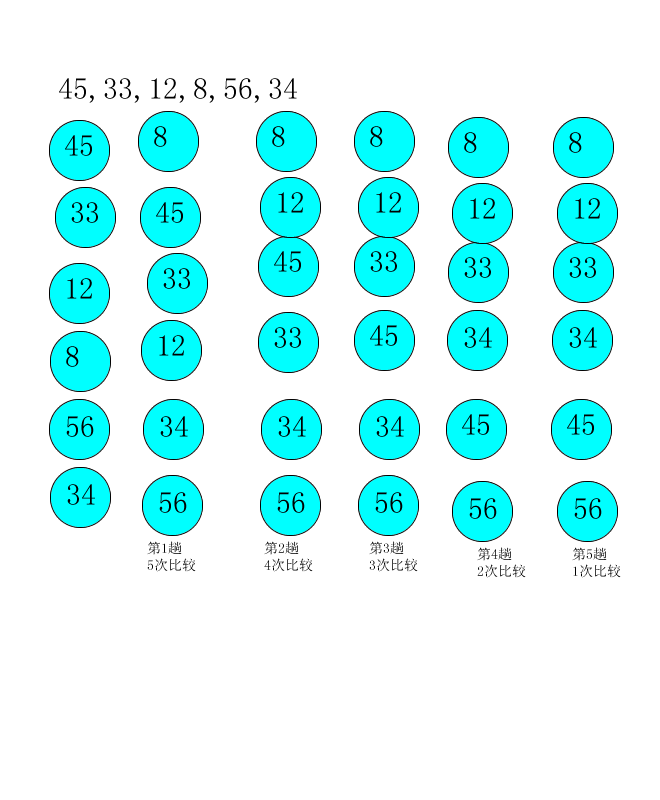
每个arr在赋值的时候，都是在一个IIFE里面，IIFE里面是作用域隔离的，所以即使变量名都是m，但是视为“不同国家的m”，每个arr里面的小函数，都只能看见自己的m值。m值纵然改变，那也察觉不到。

# 五、冒泡排序法

面试的时候，会考数组的排序。这时候你就问了，老师数组不是有sort么？为什么还要自己学排序呢？

我也不知道面试官怎么想的。JS是唯一一个天生内置数组排序的语言，还那么的好用，还能自定义。

冒泡排序原理图：



现在有n个数字进行排序，大的趟数是n-1趟，做多的趟要比较n-1次

所以，总比较次数，就是n-1 + n-2 + n-3 …… +1 次

比如，6个数字要排序，要比较5+4+3+2+1 共15次。

# 六、DOM

之前的所有学习，都是在控制台中，就是天天alert()、console.log()、prompt()

之前学习的属于语言，了解JS这个语言的特性；从今天开始我们就要学习JS控制页面上的元素了。

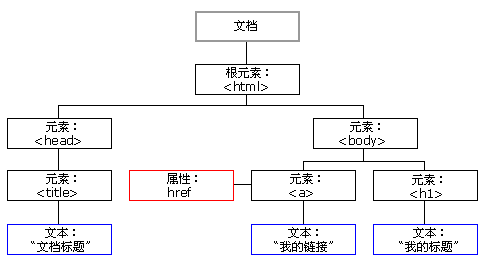
语言的部分没有结束，少了一大部分面向对象，我们在高级JS里面介绍。高级JS结束之后，JS仍然没有结束，后面设计模式、MVC课程都是再学JS的语言。

## 6.1 整体感知

DOM（Document Object Model，文档对象模型）描绘了一个层次化的节点树，允许开发人员添加、移除和修改页面的某一部分。**这使得JavaScript操作HTML，不是在操作字符串，而是在操作节点，极大地降低了编程难度**。编写例子整体感知一下这个事儿。

DOM规范在1998年10月制定，称为“DOM1级规范”。随着ECMAScript的升级，DOM也发展出了2级规范、3级规范。另外，早于1998年的DOM也有事实上的标准，我们称为0级规范。

DOM对很多东西做了抽象，提供了丰富的API：取得元素、css样式、事件、运动、元素尺寸位置、节点操作。每个知识体系都非常庞大，千丝万缕。我们今天的课程，把一些线头都掐出来，日后的课程深入研究每个线头。



## 6.2 得到元素getElementById

我们HTML负责页面的布局、结构。JS要操作HTML标签，第一件事儿就是得到这个标签。

说白了，就是把HTML标签拿到JS里面来。

JS中，最最基本的得到元素的方法有两个：

document.getElementById()  
document.getElementsByTagName()

现在我们发现，document对象的两个方法getElementById()、getElementsByTagName()

JavaScript通过document对象表示文档，它表示整个页面。它有很多属性和方法，包含了绝大多数多页面的特征和操作。学习DOM，说白了就是学习document对象。

getElementById()方法，通过id得到元素

这个单词实际上是由几个小单词组成的：get得到 Element元素 By通过 Id id属性

这几个单词合成一个词，用的是驼峰命名法。像一个头朝左边，低头吃草的骆驼。首字母小写，之后每个单词的首字母都大写，中间没有短横、空格。



那么此时就能够通过id来得到页面的元素了。

需要注意书写位置。

|  |
| --- |
| 1. var oBox = document.getElementById("box"); 2. console.log(typeof oBox); //object |

需要注意的是，getElementById里面就写id就行了，不要多写#

注意代码书写的位置，现在我们要用JS得到box这个盒子，所以JS代码就要写在HTML标签的下面。这样子浏览器先渲染HTML节点，然后再执行JS。

|  |
| --- |
| 1. <div id="box"></div> 2. <script type="text/javascript"> 3. //通过id得到页面中的元素，保存到oBox中，为什么加一个o，习惯 4. //表示这个变量是个对象 5. var oBox = document.getElementById("box"); 6. oBox.style.backgroundColor = "red"; 7. </script> |

如果说页面上没有匹配的id的HTML元素，将返回null， null就是空对象。

页面中不能有多个元素的id相同，如果有相同的元素，那么只返回文档中第一次出现的元素。

id大小写要严格区分，但是在IE6、7、8中，大小写是不区分的。

IE7及较低版本还有一个怪癖，表单元素name特性也会被当做id。为了避免这个问题，所以页面上的name最好也不要和任何id相同。

|  |
| --- |
| 1. <input type="text" **name="haha"** /> 2. <script type="text/javascript"> 3. var oInput = document.**getElementById("haha")**; 4. oInput.style.backgroundColor = "red"; 5. </script> |

IE6、7下，name也会被当做id。

## 6.3 更改HTML属性

HTML标签有很多属性，比如src、href、title等等。

JS可以更改HTML的任何属性，方法是两种：点语法 和 setAttribute()、getAttribute()。

得到一个元素之后，直接打点调用它的属性名，就能对HTML相应的属性进行更改。

“有什么，点什么”

|  |
| --- |
| 1. oTutu.src = "images/2.jpg"; 2. oLianjie.href = "http://www.163.com"; 3. oKK.value = "嘻嘻嘻嘻嘻"; |

注意id不能更改，id是只读的。

class这个属性，要换成className，因为class是JS的保留字。

|  |
| --- |
| 1. oDiv.className = "da"; |

如果想要换一个HTML元素的class属性，那么就要打点调用.className属性。

不仅仅是class属性需要用className避讳一下，还有：

class 要写成.className

for 要写成.htmlFor （label用的）

rowspan 要写成rowSpan

colspan 要写成colSpan

还有一种方法，可以设置HTML元素的属性，就是

|  |
| --- |
| 1. getAttribute(); //得到属性 2. setAttribute(); //设置属性 |

|  |
| --- |
| 1. oImg.setAttribute("src","images/2.jpg"); 2. 等价于 3. oImg.src = “images/2.jpg”; |

setAttribute和点语法有一丢丢不一样：

第一，所有自定义的属性，都不能通过点语法得到

|  |
| --- |
| 1. <div dongcidaci=”38”></div> |

|  |
| --- |
| 1. alert(oBox.dongcidaci); //undefined，自定义的属性，不是w3c的属性，都不能用点语法 2. alert(oBox.getAttribute("dongcidaci")); //38 |

第二，所有的行内样式，点语法.style得到的是一个样式对象。我们可以通过.style.border继续得到小样式。但是getAttribute()得到的是字符串

|  |
| --- |
| 1. var oDiv = document.getElementById("box"); 2. console.log(typeof oDiv.style); //object 3. console.log(typeof oDiv.getAttribute("style")); //string |

第三，getAttribute()不需要避讳，直接

|  |
| --- |
| 1. oDiv.getAttribute(“class”); //就行了 |

点语法的效率远高于getAttribute()、setAttribute()。

所以，如果我们是要读取自定义的属性，必须dongcidaci属性，偶尔用一次getAttribute，除此之外，都用点语法。

## 6.4 操作元素样式

通过点语法.style能够得到所有样式的封装 注意，只能得到行内样式，所有写在css内嵌的、外联的，一律不能得到。需要我们后面学习的知识，得到计算后样式。

我们可以通过语法：oBox.style.css样式名得到某一个样式，需要注意的是，所有css中有连字符的样式，都要转成驼峰：

|  |
| --- |
| 1. console.log(oBox.style.backgroundColor); 2. console.log(oBox.style.borderTopStyle); |

也可以通过=来更改css样式，所有更改的样式，也是行内样式。设置的时候，也是驼峰：

|  |
| --- |
| 1. var oP = document.getElementById("pp"); 2. oP.style.backgroundColor = "skyblue"; 3. oP.style.fontSize = "100px"; |

等号右边，就是css的写法，用引号引起来：

|  |
| --- |
| 1. oP.style.border = "10px solid red"; |

通过点语法.style读、设都在行内。

## 6.5 事件监听

JavaScript制作交互效果，离不开事件。所谓的事件，就是用户的某个行为，能够触发一个函数的执行

今天我们只学习DOM标准中的0级的事件绑定方法：

|  |
| --- |
| 1. // 得到这个box 2. var oDiv = document.getElementById("box"); 3. //事件 4. oDiv.onclick = function(){ 5. alert("你好，点我干嘛，我烦着呢！！"); 6. } |

也可以：

|  |
| --- |
| 1. oDiv.onclick = fun; 2. function fun(){ 3. alert("你好，点我干嘛，我烦着呢！！"); 4. } |

所以现在你就人格升华了，原来我们想要一个函数执行，必须调用这个函数，比如fun();

但是现在你知道了，一个函数可以当做一个事件的处理函数，当这个事件发生的时候，函数也能执行了。

onclick 单击

onmouseover 鼠标进入

onmouseout 鼠标离开

ondblclick 双击

onfocus 得到焦点

onblur 失去焦点

onmousedown 鼠标按下

onmouseup 鼠标按键抬起

onload 当页面完全加载成功

window.onload 表示页面中的所有的代码都已经加载完毕了。