JavaScript 之葵花宝典

学习基础之前，总结一下个人学习新技术的方法：

A> 这技术是什么？
B> 技术有什么特点？
C> 这技术怎么用?
D> 写 demo 测试

notes:测试环境为浏览器 IE9 谷歌 53 火狐 49 IDE 编译器为 sublime text3

js 学习之基础部分

一、JavaScript 是什么？

JavaScript 是一门成熟的动态编程语言，应用于 HTML 文档是能够在网站上提供动态交互能力。
JavaScript 本身是非常简洁却非常灵活的。一些常见的功能包括：

- 应用编程接口 (APIs) 。内置于浏览器中提供像动态编写 HTML 和 CSS ，抓取操控用户摄像头的数据流和操作 3D 图像和音频样品。
- 第三方 APIs 允许开发者将其他公司网站如 Twitter 或 Facebook 与自己的网站合并功能。
- 第三方框架和库，你可以使用你的 HTML 来快速编写网站和应用。

ps:学习方法：因为 js 的大量使用，可以借助别人写好的拿来修改，所以“站在别人肩膀上”的学习方法很有效。

二、JavaScript 的特点

JavaScript 跟 java 很类似，但是又没有 java 那么复杂，所以它有 java 的一些编程思想，但是又不完全是。总结起来有以下的特点：

- 简单
  JavaScript 是一种脚本式语言，才有小段程序块的方式来实现，同时 JavaScript 也是一种解释性语言，它的基本结构与 c、c++、java 等类似，但是它不需要像其他语言一样需要先编译，而是在运行的过程中，逐行的解释
- 动态
  JavaScript 是动态的，它可以直接对浏览器端的输入做出响应。因为它对浏览器的请求采取的方式是事件驱动，这种方式就是在比如主页中执行了某种操作所产生的动作，这就是事件，鼠标点击、拖动窗口等，事件产生后就会引起事件响应。
- 对象

JavaScript 是基于对象的，也就是说它可以运用自己创建的对象。

- 安全

  JavaScript 是安全的，主要是因为它不允许访问本地磁盘，并且不能将数据存储到服务器上，不允许对网络文档进行修改和删除，只能通过浏览器实现信息交互，从而防止数据丢失。

- 跨平台

  JavaScript 依赖于浏览器，与操作系统环境无关，只要是能运行浏览器的环境都能执行 JavaScript。

- 节省 CGI 的交互时间

  JavaScript 是基于浏览器来实现的，所以很多用户操作，比如用户的表单验证都可以在浏览器端就完成而不需要直接调用 CGI 程序，通常只需要把最后的校验结果交给 CGI 即可。

  Ps:CGI，公共网关接口，外部程序与 web 服务器之间交互的接口，是在 CGI 程序和 Web 服务器之间传递信息的规程。CGI 规范允许 Web 服务器执行外部程序，并将它们的输出发送给 Web 浏览器，CGI 将 Web 的一组简单的静态超媒体文档变成一个完整的新的交互式媒体。CGI 程序使网页具有交互功能。

第一个测试代码：

使用说明： testjs.js js01.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xh
-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
    <title>test</title>
    <!--引入外部js文件-->
    <script type="text/javascript" src="testjs.js"></script>
    <!--当前文档内的js代码块-->
    <script type="text/javascript">
        function changebtn(){
            i = document.getElementById("t_id");
            i.innerHTML="改变后的文本";
        }
    </script>
</head>
<body>
    <p id="t_id">原本的内容</p>
    <button type="button" onclick="changebtn()">click</button>
</body>
</html>
```

三、基础语法

i. 变量

变量是存储数据的容器。声明一个变量：

var a =1;

let a = 1; //let 声明了一个块级域的局部变量，并且可以给它一个初始化值

JavaScript 中的数据类型：

变量	解释	示例
String	字符串，一段文本。要指示变量是字符串，你应该将它们用引号包裹起来。	`var myVariable = 'Bob';`
Number	数字，一个数字。不用引号包围。	`var myVariable = 10;`
Boolean	布尔型，一个 True/False（真 / 假）值。`true/false` 是 JS 里的特殊关键字，不需要引号。	`var myVariable = true;`
Array	数组，一种允许你存储多个值在一个引用里的结构。	`var myVariable = [1,'Bob','Steve',10];` 调用数组的元素只需: `myVariable[0]`, `myVariable[1]`, 等等.
Object	对象，基本上 JavaScript 里的任何东西都是对象，而且都可以被储存在变量里。将这个记在脑子里。	`var myVariable = document.querySelector('h1');` 上面所有示例都是对象。

当然还有一个 undefined 类型 ，表示未定义类型。
ps:JavaScript 是区分大小写的，所以变量的声明必须大小写分明。

- **let 和 var**

let 的语法为：let var1 [= value1] [, var2 [= value2]] [, ..., varN [= valueN]];
let 允许把变量的作用域限制在块级域中。与 var 不同处是：var 申明变量要么是全局的，要么是函数级的，而无法是块级的。

```
<script type="text/javascript">
    function varTest(){
        var x = 20;
        if(true){
            var x =30;
            alert(x);// 30
        }
        alert(x);//30
    }
    function letTest(){
        let x  = 20 ;
        if(true){
            let x =30;
            alert(x); //30
        }
        alert(x);//20
    }
</script>
```

用 let 定义的变量的作用域是定义它们的块内，以及包含在这个块中的子块 ，这一点有点象 var,只是 var 定义的变量的作用域是定义它们的函数内。

```
<script type="text/javascript">
    function test(){
        let x = 10;
        let x = 10;
        let x = 10;
    }
</script>
```

如上，当重复定义 let 一个变量的时候回出现 SyntaxError 错误，但是 var 不会。这种想象被称为 let 的暂存死区。

```
<script type="text/javascript">
    function test(){
        alert(x);
        let x = 10;
    }
</script>
```

同样的在变量提升中，let 依旧存在暂存死区，所以上面的代码会报错 ReferenceError。

```
<script type="text/javascript">
    function test(){
        var x= 0 ;
        for(let x =x ; x<10;x++){
            alert(x);
        }
    }
</script>
```

如上代码，let 在循环体中，x 会变成 undefined。
所以要在循环体中使用 let 就要如下：

```
function test(){
    for(let x =0 ; x<10;x++){
        alert(x);
    }
}
```

再来对比一下 var 和 let

```
<script type="text/javascript">
    var a = 1;
    var b = 2;
    if(a ==1){
        let a = 4;
        var b = 3;
        alert(a);//4
        alert(b);//3
    }
    alert(a);//1
    alert(b);//3
</script>
```

所以很清楚的知道，var 的作用域是函数，而 let 的作用域是块。

ii. **常量(es6 新增)**

const 声明创建一个只读的常量。这不意味着常量指向的值不可变，而是变量标识符的值只能赋值一次。

比如 ： const a = 10 ;

由于是 es6 新增的内容，所以浏览器的兼容不是很好。

兼容性如下表

Feature	Chrome	Edge	Firefox (Gecko)	Internet Explorer	Opera	Safari
Basic support	21	(Yes)	36 (36)	11	12	5.1
Reassignment fails	20	(Yes)	13 (13)	11	?	?

所以为了使用常量，并且保证兼容性，可以使用闭包来实现。(闭包后面再讲)

测试代码如下：

```
<script type="text/javascript">
    var cc = (function(){
        var const_a = 100;
        var method={};
        method.getConstA=function(){
            return const_a;
        }
        return method;
    })();
    var c= cc.getConstA();
    alert(c);
   /* const b = 200;
    alert(b);*/
</script>
```

iii. **运算符**

运算符是一个根据两个值（或变量）做出结果的代数符号。

常用的运算符如下：

运算符	解释	符号	示例
加	用来相加两个数字，或者连接两个字符串。	+	`6 + 9;` `"Hello " + "world!";`

减、乘、除	这些运算符操作将与你期望它们在基础数学中所做的一样。	-, *, /	`9 - 3;`
			`8 * 2; // JS中的乘是一个"*"号;`
			`9 / 3;`

赋值号	你之前已经见过这个符号了：它将一个值赋给一个变量	=	`var myVariable = 'Bob';`
相等	它将测试两个值是否相等，而且会返回一个 true/false（布尔型）值。	===	`var myVariable = 3;` `myVariable === 4;`

非，不等	经常与相等运算一起使用，非运算符在JS中表示逻辑非——它也返回一个布尔值。	!, !==	原本的值是 true，但是返回了 false 因为之后我们做了非运算：
			
			`var myVariable = 3;`
			`!myVariable === 3;`
			
			这里我们测试了"我的 myVariable 是否等于 3". 这里返回了 false, 因为它等于 3.
			
			`var myVariable = 3;`
			`myVariable !== 3;`

iv. 表达式

表达式，expression，JavaScript 中的一个短语，js 解释器会对这个 exp 进行解释，然后产生一个值。

比如最简单的一个常量或者变量名：

```
true    //一个布尔常量
```

v. 流程控制

条件语句包含 if else 和 switch case

问题 1：判断 a 是否小于 10；

```
<!--当前文档内的js代码块-->
<script type="text/javascript">
    var a  = 1;
    if(a<10){
        alert("a:"+a+"小于10");
    }
</script>
```

问题 2：判断某分数所在等级，90 以上为优，80-90 为良，70-80 为中，60-70 为差，60 以下为不及格。<当然可以使用 if else 但是在多种条件判断的情况下，switch 比 if else 效率更高>

ps：switch case 语句只计算一次开始处的表达式，而 if else 会计算每一个 if 字句，当 if 的表达式字句都相同的时候可以使用 switch 替换 if。

```
<script type="text/javascript">
    //判断某分数所在的等级
    var score ,flag;
    score  =  85 ; //默认值
    flag = Math.floor(score/10);
    switch(flag){
        case 9:
        alert("优秀");
        break;
        case 8:
        alert("良");
        break;
        case 7:
        alert("中");
        break;
        case 6:
        alert("差");
        break;
        default:
        alert("不及格");

    }
</script>
```

switch 语句注意 break 结束当前的循环，不然就会死循环。

vi. 循环控制

循环结构是为了处理重复执行的代码，有

while 语句、do...while 语句、for 循环、for in 循环。

```
<!--当前文档内的js代码块-->
<!--循环打印出h1-h7号字体-->
<script type="text/javascript">
   var a =1;
   while(a<7){
    document.write("<h"+a+">h"+a+"号字体"+"</h"+a+">");
    a++;
   }
</script>
```

结果自己把代码在浏览器运行一下。

同样的代码，在 do while 结构中如下实现：

do while 语句，花括号中的语句至少执行一次才会判断。

```
<!--当前文档内的js代码块-->
<!--循环打印出h1-h7号字体-->
<script type="text/javascript">
    var a =1;
    do{
      document.write("<h"+a+">h"+a+"号字体"+"</h"+a+">");
     a++;
    }while(a<8);
</script>
```

同样的使用 for 循环来实现

```
<!--当前文档内的js代码块-->
<!--循环打印出h1-h7号字体-->
<script type="text/javascript">
   for(a=1;a<8;a++){
     document.write("<h"+a+">h"+a+"号字体"+"</h"+a+">");
   }
</script>
```

同样的使用 for in 来实现

```
<!--当前文档内的js代码块-->
<!--循环打印出h1-h7号字体-->
<script type="text/javascript">
    var as=[1,2,3,4,5,6,7,8];
    var a ;
   for(a in as){
     document.write("<h"+a+">h"+a+"号字体"+"</h"+a+">");
   }
</script>
```

for in 用来遍历数组或者对象的属性。

vii. 异常处理

异常处理很重要，至少通过异常处理，你可以快速的知道代码块运行所产生的异常，方便快速定位和修改 bug.

```
try {
//语句块;
}catch{
//异常块
}finally{
//语句块
}
```

在 try catch 结构中，可以没有 finally，但是一定要有 catch，不然异常处理语句没有意义。

```
<head>
    <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
    <title>test</title>
    <!--当前文档内的js代码块-->
    <!--捕获js异常-->
    <script type="text/javascript">
      var txt = "";
      function showmsg(){
        try{
            alert("try 代码块");
            add("sssssss");//该方法不存在，所以会进入catch
        }catch(err){
            txt="此页面有个错误  \n\n";
            txt+="错误描述: "+err.description+"\n\n";
            alert(txt);
        }
      }
    </script>
</head>
<body>
  <button type="button" onclick="showmsg()"> show error</button>
</body>
```

\n 是换行符 等同于 html 中的</br>标签

viii. console.log()和 alert()的区别

- alert()

  alert 有阻塞作用，不点击确定，后面的代码是没法继续执行的。

  alert 只能输出 string 字符串，如果 alert 输出的是对象，那么会自动调用 toString()方法。

  alert 不支持多个参数的写法，只能输出第一个参数

```
alert(1,2,3,4);
```

  输出的只会是 1

- console.log()

  console 调用的是浏览器方法，会在浏览器的控制台输出

  console.log 可以任何类型的数据

  console.log 支持多个参数的写法

```
console.log(1,2,3,4);
```

  输出 1 2 3 4

  看看下面的一个例子

  对一个数组的数字进行排序：

```
<!--当前文档内的js代码块-->
<!--数组排序-->
<script type="text/javascript">
  var score =[1,2,3,4];
  var sortscore =[];
  console.log(score);//排序之前的数组
  sortscore= score.sort(orderfnc);
  console.log(sortscore);//排序之后的数组
  function orderfnc(a,b){
    return b - a;
  }
</script>
```

结果为：

```
[ 1, 2, 3, 4 ]
[ 4, 3, 2, 1 ]
```

```
<!--当前文档内的js代码块-->
<!--数组排序-->
<script type="text/javascript">
  var score =[1,2,3,4];
  var sortscore =[];
  alert(score);//排序之前的数组
  sortscore= score.sort(orderfnc);
  alert(sortscore);//排序之后的数组
  function orderfnc(a,b){
    return b - a;
  }
</script>
```

输出 1,2,3,4　　4,3,2,1

但是　这两段代码 其中第一段代码在谷歌浏览器中的结果 却是 4,3,2,1 4,3,2,1
这是因为谷歌浏览器对于 console.log 执行的是延迟执行，也就是“惰性求值”，对于这种数字的计算，对象的引用就会出问题。所以要好好思考一下 console.log 的使用。

js 学习之对象

四、对象是什么？

JavaScript 是基于对象的语言，但是不是面向对象，这样说是因为 JavaScript 没有为对象提供抽象、继承、重载等面向对象特性的功能。JavaScript 对象是由属性 properties 和方法 method 组成的。

属性是为了存储数据，方法是为了操作数据。

介绍对象之前，先了解一下对象的操作语句

- ✓ for in 语句

```
for(变量 in 对象){
    //语句块
}
```

可以对对象的所有属性进行循环操作。

- ✓ with 语句

```
with(对象){
    //语句块
}
```

在语句块内，任何变量的引用都是这个对象的属性。

- ✓ this 关键字

this 是对当前的引用。为了避免一个对象引入多个对象混乱的现象而产生了 this。

- ✓ new 运算符

new 可以创建一个新对象。

对象属性的引用

- ✓ . 点运算符。 比如 object.name =" 张三" ;
- ✓ 对象下标。 比如 objects[0] =" 张三" ;
- ✓ 字符串 。比如 object ["name"] =" 张三" ;

对象方法的引用

对象.方法() 比如 object.add();

五、内置对象

- **Array**

array 是个数组对象。用来存在共同特效的元素。用下标或者 key 来标识元素。

Array 有四个属性 index , input , length,prototype

创建数组

获取数组长度

遍历数组

```
var citys=new Array();
citys[0] = "武汉市";
citys[1] = "宜昌市";
citys[2] = "黄冈市";
citys[3] = "鄂州市";
//获取数组长度
 console.log(citys.length);  //4
//通过索引访问元素
 console.log(citys[0]); //武汉市
 //添加一个元素到数组末尾
 citys.push("北京市");
 //访问数组最后一个元素
 console.log(citys[citys.length-1]);
 //删除数组末尾元素
 citys.pop();
 //删除数组头部元素
 citys.shift();
 //添加数组头部元素
 citys.unshift("上海市");
 //找到某个元素的索引
 console.log(citys.indexOf("黄冈市"));
 //复制数组
 var cpCitys=citys.slice();
 console.log(cpCitys.length);
 //通过forEach遍历数组
citys.forEach(function(item,index,array){
    console.log(item,index);
})
```

- ✓ concat() 合并两个元素的方法

```
var arr = new Array();
arr[0]="ff第一元素";
arr[1]="90";
arr[2]="*********";
var arr2 = new Array();
arr2[0]="ddd";
console.log(arr.concat(arr2));
```

- ✓ join();可以用指定的分隔符分割元素。

```
var arr = new Array();
arr[0]="ff第一元素";
arr[1]="90";
arr[2]="**********";
console.log(arr.join("."));
```

***经典题目，冒泡排序

```
function sortAB(a,b){
    return b - a ;
}
var arr = new Array(8);
arr[0]=90;
arr[1]=5;
arr[2]=13;
arr[3]=35;
arr[4]=9;
arr[5]=86;
arr[6]=1;
arr[7]=0;
document.write("排序前数组为:   "+arr+"</br>");
document.write("排序后数组为:   "+arr.sort(sortAB));
```

String

string 是字符串对象，使用频率很高；**从 es6 开始，字符串字面量也被称为模板字符串。

字符串有两个属性 length 和 prototype

```
var str = new String();
str = "test \n test";
console.log(str);
```

字符串中 可以加入一些常用的转义字符，比如 \0 \n \\ \t \f

charAt()访问某个字符串

```
var str = new String();
str = "test \0 \n cano";
console.log(str.charAt(2));
```

**在 es5 中提供了一种新的思路，就是把字符串当做数组来处理。每个字符对应一个数值索引。

```
var str = new String();
str = "test \0 \n cano";
console.log(str[2]);
```

字符串的比大小

先看例子

```
var a = "100";
var b ="200";
console.log(a>b,a<b); // false true
```

再看

```
var a = "100a";
var b ="1000b";
console.log(a>b,a<b); //  true false
```

字符串的比较步骤如下：

```
var a = "100a";
var b ="1000b";
console.log(a>b,a<b); //  true false
console.log("1".charCodeAt(0));//49
console.log("1".charCodeAt(0));//49
console.log("a".charCodeAt(0)); //97
console.log("0".charCodeAt(0)); //48
a>b --> a[0]="1",b[0]="1" 两个的charCode是相等的 49 -->
同样的比较一直比出谁的charCode大就谁大 a的code=97 >0的code 48
```

然后送大家一句话

```
console.log("选择">"努力"); //true
```

indexOf(searchVal,[fromIndex]) 返回第一个字符所在位置

```
var str = "ever thing";
console.log(str.indexOf("e"));
```

substring(indexA,indexB) 获取 a 到 b 的字符串

toLowerCase() toUpperCase() 大小写转换

replace() 替换

```
var str = "ever thing  pk";
console.log(str.replace("ing","kio")); //ever thkio pk
 //创建正则表达式 来替换 ver 字符
var reg = new RegExp("ver","g");
console.log(str.replace(reg,"900"));//e900 thing  pk
```

Math

Math 是一个内置对象， 为数学常量和数学函数提供了属性和方法，而不是一个函数对象。

与其它全局对象不同的是, Math 不是一个构造器. Math 的所有属性和方法都是静态的

提供的属性有

Math.E (欧拉常数)

Math.LN2(2 的自然对数, 约等于 0.693)

Math.LN10(10 的自然对数, 约等于 2.303)

Math.LOG2E(以 2 为底 E 的对数，约等于 1.443)

Math.PI(圆周率，一个圆的周长和直径之比，约等于 3.14159)

```
var m = -1;
//m的绝对值
console.log(Math.abs(m)); //1
```

✧ max()和 min() 比大小

```
var m = -1;
var n = 0;
console.log(Math.max(m,n)); //0
```

✧ random()随机数 产生 0-1 之间的数字

```
    <script type="text/javascript">
       function createRandom(){
        var str = Math.floor(Math.random()*1000)+1;
        alert(str);
       }

    </script>
</head>
<body>
  <button type="button" onclick="createRandom()">click</button>
</body>
```

Date

创建一个 date 实例，Date 基于 1970 年 1 月 1 日。

常用的构造器如下：

```
var today = new Date();
var today = new Date(1453094034000);// by timestamp(accurate to the millimeter)
var birthday = new Date("December 17, 1995 03:24:00");
var birthday = new Date("1995-12-17T03:24:00");
var birthday = new Date(1995,11,17);
var birthday = new Date(1995,11,17,3,24,0);
```

Date 有两个属性 length 和 prototype

问题：写一个倒计时，用来倒计时 2017 年除夕(2017-01-27)剩余时间，精确到时分秒

```
    <!--当前文档内的js代码块-->
    <!--2017除夕倒计时-->
    <script type="text/javascript">
        function showtime(){
            var endtime= new Date("2017/01/27 00:00:00"); //截止时间
            var nowtime = new Date();
            var t =endtime.getTime() - nowtime.getTime();
            var d=Math.floor(t/1000/60/60/24);//获取天数
            var h=Math.floor(t/1000/60/60%24);//获取小时
            var m=Math.floor(t/1000/60%60);//获取分钟
            var s=Math.floor(t/1000%60);//获取秒数
            document.getElementById("t_d").innerHTML = d + "天";
            document.getElementById("t_h").innerHTML = h + "时";
            document.getElementById("t_m").innerHTML = m + "分";
            document.getElementById("t_s").innerHTML = s + "秒";
        }
        setInterval(showtime, 1000);//定时1秒执行一次
    </script>
</head>
<body>
    <div id="msg">
        倒计时 2017年除夕 还有
        <span id="t_d">00天</span>
        <span id="t_h">00时</span>
        <span id="t_m">00分</span>
        <span id="t_s">00秒</span>
    </div>
</body>
```

自定义对象

有时候，会发现 js 提供的内置对象没法满足实际需求，这个时候需要自定义。

```
function myobj(name,sex){
    this.name=name;
    this.sex=sex;
}
var test= new myobj("张三","男");
console.log(test.name,test.sex);//张三 男
```

使用原型 prototype 来创建对象。

javascript 中的每个对象都有 prototype 属性，Javascript 中对象的 prototype 属性的解释是：返回对象类型原型的引用。

```
function myobj(time){
    this.time=new Date();
}
myobj.prototype.name="张三";
myobj.prototype.sex="男";
myobj.prototype.showyear=function(){
    var year=this.time.getYear()-1;
    return year;
}
var test = new myobj();
console.log(test.name);//张三
console.log(test.showyear());//115
```

其中 showyear 就是原型方法。

prototype 返回对象的引用。
A.prototype = new B();

理解 prototype 不应把它和继承混淆。A 的 prototype 为 B 的一个实例，可以理解 A 将 B 中的方法和属性全部克隆了一遍。A 能使用 B 的方法和属性。这里强调的是克隆而不是继承。可以出现这种情况：A 的 prototype 是 B 的实例，同时 B 的 prototype 也是 A 的实例。

```
function baseObj(){
    this.showMsg=function(){
        alert("hello baseobj");
    }
}
function extendObj(){

}
extendObj.prototype=new baseObj();
var test = new extendObj();
test.showMsg();//调用的就是baseobj的showMsg方法
```

继续看

问题，如果 extendObj 中本身包含有一个与 baseObj 的方法同名的方法会怎么样？

那么调用的会谁的方法？

```
function baseObj(){
    this.showMsg=function(){
        alert("hello baseobj");
    }
}
function extendObj(){
    this.showMsg=function(){
        alert("hello extendobj");
    }
}
extendObj.prototype=new baseObj();
var test = new extendObj();
test.showMsg();//调用的就是extendobj的showMsg方法
```

这说明：函数运行时会先去本体的函数中去找，如果找到则运行，找不到则去 prototype 中寻找函数。或者可以理解为 prototype 不会克隆同名函数。

继续思考，如果我还想用原来的对象的方法怎么办？？？

```
function baseObj(){
    this.showMsg=function(){
        alert("hello baseobj");
    }
}
function extendObj(){
    this.showMsg=function(){
        alert("hello extendobj");
    }
}
extendObj.prototype=new baseObj();
var testextend = new extendObj();
var testbase= new baseObj();
testbase.showMsg.call(testextend);//调用的就是baseobj的showMsg方法
```

使用 call
因为想调用 base 的 showMsg 方法，所以是 call(extend);

在 es6 中，api 新增了一个 Object.setPrototypeOf(obj, prototype)
将一个指定的对象的原型设置为另一个对象或者 null(既对象的[[Prototype]]内部属性).
但是该方法兼容性还不行，有兴趣自己查看 api.

六、窗口对象

- **Document**

每个 window 都有一个 document 属性 ,该属性表示在窗口显示的 html 文件的 Document 对象。
前面已经使用了很多次的 write()方法，用来输出 html 内容。
document 的属性和方法比较多，只介绍几个常用的。

api 介绍 https://developer.mozilla.org/zh-CN/docs/Web/API/Document

✓ location 属性

Document.location 是一个只读属性，返回一个 Location 对象，包含有文档的 URL 相关的信息，并提供了改变该 URL 和加载其他 URL 的方法.

```
document.location="url";(只读)
document.location.reload("url";);
window.location="url";
location="url";
document.href="url"
document.location.href="url"
document.location.replace="url"
document.action="url";
document.submit();

document.location.href
document.location.replace

都可以实现从A页面切换到B页面，但他们的区别是：
用document.location.href切换后，可以退回到原页面。而用document.location.
    replace切换后，不可以通过“后退”退回到原页面。

关于document.location.href或其他可回退的切换方式
document.location 相当于 document.URL 声明了装载文档的URL，
除非发生了服务器重定向，否则该属性的值与Window.location.href的值是一样的.

** history.go(-1);//返回上一页
document.iframe.location.href='url';//改变框架内容
```

✓ open/write/close 方法

close 一般搭配 write 方法，用来关闭流的操作。

```
document.open();//开启文件流的写入
document.write("document write start");//写入文件
document.close();//关闭写入
```

ps:如果忘记关闭文档，浏览器就不能控制要显示的文档是否完整，而且浏览器写入的 html 会缓存起来，只有在调用 close 后才会显示出来。这样会造成严重的 io 异常，浏览器崩溃。

✓ getElementById 、getElementByClassName 、getElementsByTagName

```
document.getElementById(id); //通过id获取匹配的元素
document.getElementsByName(name);//通过name获取匹配的元素
document.getElementsByClassName(className);//通过classname来获取匹配的元素
document.getElementsByTagName(name);//通过tagname获取匹配的元素
document.getElementsByTagNameNS(namespace, localname);
```

Form

一个 form 对象代表了一个 html 表单。通常 form 存在于 from[]集合中 ,通过 document.form[0] 来调用。

html dom 的表单是通过 submit 按钮来提交，reset 来重置的。类似的 js 的 form 对象也有 submit()、reset()、handleEvent()方法。

```
<div >
    <form name="form_a">
        文本框 a:<input type="text" onchange="document.form_b.elements[0].value=this.value">
    </form>
    <form name="form_b">
        文本框 b: <input type="text" onchange="document.forms[0].elements[0].value=this.value">
    </form>
    form_b引用了当前文档中的第一个对象forms[0]=form_a
    from_a的onchange事件触发后会根据form_b的值来设置form_a的text的值
</div>
```

form 对象 最常用的就是做表单验证。

```
    <!--验证姓名和邮箱格式-->
    <script type="text/javascript">
        function formcheck(){
            if(document.forms[0].u_name.value==""){
                alert("姓名不能为空");
                document.forms[0].u_name.focus();//光标移到此处
                return false;
            }
            if(document.forms[0].u_mail.value.indexOf("@")==-1||document.forms[0].u_mail.value==""){
                alert("邮箱地址无效！");
                document.forms[0].u_mail.focus();//光标移到此处
                return false;
            }
        }
    </script>
</head>
<body>
    <div >
        <form method="post" action="#" onsubmit="formcheck()">
            姓名: <input type="text" name="u_name"><br/>
            邮箱: <input type="text" name="u_mail"><br/>
            <input type="submit" value="提交">
            <input type="reset" value="重置">
        </form>
    </div>
```

Anchor/Link

anchor 锚点对象，表示 html 的超链接的 name，通常和 link 对象一起出现。在 html 文档中 a 标签出现一次就产生一个 anchor，其中 html 的 a 的 name 就是 anchor ，href 就是 link。同样的是 document.anchors[] 和 document.links[]集合。

document.anchors[].name ==name 文本

document.anchors[].text ==text 文本值 文本

document.links[].href==href 文本

Image

image 对象表示的是 html 对应的<img>标记产生的元素集合 ,存在于 document.images[]中。

document.images[].src ==src

七、DOM 对象

dom 是文档对象模型的缩写，是独立于语言和平台的一套接口。可以用来读取、删除等操作 xml 和 html 文档的内容。

html Dom

dom 的本质就是 html 文档的一个结构化的视图。理解就是，html 是一颗节点树，每一个节点代表着可以和它进行交互的对象。所以 html 的节点就是一个个对象。

dom 提供的接口很多，不累赘的描述，实例来说明

常用的读取操作 html 元素如下

```
    <script type="text/javascript">
        //获取根节点
        document.documentElement;//<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
        //获取指定的元素节点
        document.getElementsByTagName("body")[0];  //FF <body> Chrome  <body>...</body>
        //获取整个页面某元素节点
        document.getElementsByTagName("a"); //[a,a]
        //创建元素节点
        document.createElement("div");//<div>
        //创建文本节点
        document.createTextNode("test");// <TextNode textContent="test">
        //根据标签名获得元素节点
        document.getElementById("div2");  //<div id="div2">
        //根据id获取指定属性的值
        var unametxt=document.getElementById("u_name");
        //设置指定属性的值
        unametxt.setAttribute("value", "张三");
        //获取属性的值
        var unameval= unametxt.getAttribute("value");
        //搜索是否包含指定名称的属性，如果有返回true
        var val =unametxt.hasAttribute("value");
        //删除元素中指定的属性
        unametxt.removeAttribute("value");
    </script>
</head>
<body>
    <div id="div2">
       <a></a>
       <a></a>
       <div id="div">
            测试文本<input type="text" id="u_name">
       </div>
    </div>
</body>
```

需要知道和了解的知识如下：

DOM对象树不同节点的名值对比

节点	nodeName（节点名）	nodeValue(节点值)	nodeType值
Element元素节点	对应标签名的大写形式如：HTML	Null	1
Attr属性节点	文档中定义的属性名 如：type	文档中定义的属性值 如：button	2
Text文本节点	#text	文本内容 如：133	3
Comment注释节点	#comment	注释内容 如：comment	8
Document根节点	#document	Null	9

根节点的属性和方法：

属性	描述
documentElement	表示文档的根元素节点 在HTML文档中，它表示<html>这个标签代表的元素节点
方法	描述
getElementById()	返回文档中具有制定id属性的Element节点
	方法参数为节点的id的属性值
getElementByTagName（）	以数组方式返回文档中具有制定标签的Element节点，其书序为在文档中出现的顺序 标签名指的是像body，table这样的HTML标签
	方法参数为标签名称。
createElement()	用指定的标记名创建新的Element节点对象
	方法参数为节点标签的名字
createTextNode()	用指定的文本创建新的文本节点对象
	方法参数为文本信息
createAttibute()	用指定名字创建新的Attr节点对象
	方法参数为属性的名字
createComment()	用指定的字符串创建新的Comment节点对象
	方法参数为注释信息

元素节点的属性和方法

属性	描述
tagName	元素节点对应的标签的大写名字，例<table>元素的标签名字为Table
方法	描述
getElementsByTabName()	以数组方式返回当前与元素节点的子孙节点中具有指定标签名的所有元素节点，其准许为在文档中出现的顺序
	方法参数为节点的标签名
getAttribute()	以字符串形式返回指定属性的值
	方法参数为属性名称
getAttributeNode()	以属性节点对象的形式返回指定属性的值
	方法参数为属性名称
setAttribute()	设置指定的属性的值，如果该属性不存在则添加新属性
	方法的第一个参数为属性的名称 方法的第二个参数为属性的值
setAttributeNode()	把指定的属性节点添加到该元素的属性列表中
	方法的参数为属性节点对象
has Attribute()	如果该元素具有制定名字的属性，则返回true
removeAttribute()	从元素节点中删除指定的属性
	方法参数为属性的名称
removeAttributeNode()	从元素节点的属性列表中删除指定的Attr节点
	方法参数为属性的名称

在知道了上面的一些知识后，写个例子，遍历 html

```
    <title>test</title>
    <!--当前文档内的js代码块-->
    <!--遍历dom树-->
    <script type="text/javascript">
       function countTag(n){
        //初始化计数器
        var defcount= 0 ;
        //判断n的节点是不是element对象
        if(n.nodeType==1){
            defaultValue++;//计算+1
        }
        //n的子节点
        var child= n.childNodes;
        //遍历子节点 递归
        for(var i =0 ; i<child.length;i++){
            defcount+=countTag(child[i]);
        }
        //返回计数的结果
        return defcount;
       }
    </script>
</head>
<body onload="alert(countTag(document.body))">
        <div>
            <a href="#"></a>
            <span></span>
        </div>
</body>
```

****IE 与 FireFox 的 DOM 对象树差异如下：

- IE 会把一些没有在文档中定义的属性也加入 DOM 树。
- IE 不会把 title 中的文本内容加入 DOM 树。
- IE 会把换行缩进这样的信息过滤掉，FireFox 则会认为是有用的文本内容，并作为文本节点的一部分加入 DOM 树。
- IE 不会把 script 标签中的内容加入 DOM 树，FireFox 将里面的内容加入 DOM 树。

xml Dom

xml dom 学习 可以参照 w3cschool 上的。

http://www.w3school.com.cn/xmldom/index.asp

八、其他对象

window 对象

前面使用过 document 对象，其实 document 对象的父对象还是 window 对象。它是客户端默认的全局对象，是一个根对象。

```
    <!--当前文档内的js代码块-->
    <!--window对象-->
    <script type="text/javascript">
       function checkWin () {
          if(window.closed){
            alert("窗口已经关闭");
          }else{
            alert("窗口没有关闭");
          }
       }
       //创建一个新窗口
       var win= window.open("","","width=400,height=300,menubar=yes,status=yes");
       win.document.write("<head><title>new title</title></head>");
       win.document.write("<div><h3>new win h3</h3></div>");
       //设置浏览器状态栏，实际上最新的浏览器对于状态栏都有不同程度的显示优化
       function setWinStatus(){
            window.status="设置后的status";
       }
    </script>
</head>
<body>
    <button type="button" onclick="checkWin()">检查窗口是否关闭</button>
    <button type="button" onclick="setWinStatus()">设置</button>
</body>
```

screen 对象

screen 是 js 解释器在运行的时候创建的，包含了浏览器显示屏幕的信息。每一个 window 对象的 screen 属性都引用了一个 screen 对象。比如可以修改显示屏的高度宽度，显示像素，屏幕字体等。

```
 with(document){
      write("屏幕的高宽："+screen.width+"*"+screen.height+"<br/>");
      write("屏幕的颜色："+screen.colorDepth+"<br/>");
      write("屏幕的颜色分辨率："+screen.pixelDepth+"<br/>");
 }
```

location/navigator 对象

location 是 js 解释器在运行的时候创建的，包含了浏览器当前 url 的信息。同样的 location 对象存储在 window 对象的 location 属性中。

navigator 也是包含在 window 对象之中。IE 中这个对象等价于 clientInformation 对象，浏览器运行的各种信息。

```
with(document){
      write("浏览器代码："+navigator.appCodeName+"<br/>");
      write("浏览器名称："+navigator.appName+"<br/>");
      write("是否启动cookie："+navigator.cookieEnabled+"<br/>");
      write("user-agent 头部值："+navigator.userAgent+"<br/>");
      write("浏览器系统默认语言："+navigator.userLanguage+"<br/>");
     /* write("浏览器版本："+navigator.appVersion"<br/>");*/
      write("浏览器系统平台："+navigator.platform+"<br/>");
}
```

```
with(document){
    write("当前网址的主机名和端口： "+location.host+"<br/>");
    write("当前网址的主机名： "+location.hostname+"<br/>");
    write("当前网址的锚点： "+location.hash+"<br/>");
    write("当前网址的URL： "+location.href+"<br/>");
    write("当前网址的URL端口： "+location.port+"<br/>");
    write("当前网址的URL协议： "+location.protocol+"<br/>");
}
```

location 中常用的两种方法：

```
//加载新的URL
function replaceURL(){
 window.location.replace("https://www.baidu.com");
}
//重新加载当前文档
function reloadURL(){
 window.location.reload();
}
```

history 对象

history 同样是包含在 window 对象的 history 属性中，包含了浏览器访问后的 URL 数组。一般来说我们能访问只能是 history 对象的 length 属性。

```
history.length;
```

```
//加载history列表的前一个url
history.back();
//加载history列表的下一个url
history.forward();
//加载history列表的某一个页面
history.go(n);
```

以下例子，在浏览器地址栏先输入几个网址，然后打开 demo

```
    <script type="text/javascript">
     //加载history列表的前一个url
      function backURL(){
        document.navbar.url.value="";
        parent.history.back();
      }
      //加载history列表的下一个url
      function forwardURL(){
        document.navbar.url.value="";
        history.forward();
      }
      //加载history列表的某一个页面
      function goURL(){
        parent.location=document.navbar.url.value;
      }
    </script>
</head>
<body>
    <form action="" name="navbar" onsubmit="goURL();return false;">
        URL:<input type="text" name="url" size="100" />
        <button type="button"  onclick="backURL()">后退</button>
        <button type="button"  onclick="forwardURL()">前进</button>
        <button type="button"  onclick="goURL()">跳转</button>
    </form>
</body>
```

- **frame 对象(略去)**

实际上很多浏览器已经删除了 frame 的支持。。。。。。

xml 部分 有兴趣可以学习一下 w3cschool 上的

- **正则表达式**

正则表达式部分，简单的讲述一下怎么读。

例如：[\u4e00-\u9fa5]

其中 \ 在正则表达式中很常见，表示下一个字符为特殊字符、原义字符、反向引用和八进制转义字符

这两个 unicode 值正好是 Unicode 表中的汉字的头和尾，所以这个正则匹配的汉字。

\ba\w*\b 匹配以字母 a 开头的单词——先是某个单词开始处(\b)，然后是字母 a,然后是任意数量的字母或数字(\w*)，最后是单词结束处(\b)

\d+ 匹配 1 个或更多连续的数字。这里的+是和*类似的元字符，不同的是*匹配重复任意次(可能是 0 次)，而+则匹配重复 1 次或更多次。

\b\w{6}\b 匹配刚好 6 个字符的单词。

✓ **元字符**

常用的元字符如下：

代码	说明
.	匹配除换行符以外的任意字符
\w	匹配字母或数字或下划线或汉字
\s	匹配任意的空白符
\d	匹配数字
\b	匹配单词的开始或结束
^	匹配字符串的开始
$	匹配字符串的结束

元字符^（和数字 6 在同一个键位上的符号）和$都匹配一个位置，这和\b 有点类似。^匹配你要用来查找的字符串的开头，$匹配结尾

比如我要匹配 QQ 号必须是 5 位到 12 位数字：

^\d{5,12}$

解释如下：这里的{5,12}和前面介绍过的{2}是类似的，只不过{2}匹配只能不多不少重复 2 次，{5,12}则是重复的次数不能少于 5 次，不能多于 12 次，否则都不匹配。

因为使用了^和$，所以输入的整个字符串都要用来和\d{5,12}来匹配，也就是说整个输入必须是 5 到 12 个数字，因此如果输入的 QQ 号能匹配这个正则表达式的话，那就符合要求了。

和忽略大小写的选项类似，有些正则表达式处理工具还有一个处理多行的选项，

如果选中了这个选项，^和$的意义就变成了匹配行的开始处和结束处。

✓ **限定符**

常用的限定符如下：

代码/语法	说明
*	重复零次或更多次
+	重复一次或更多次
?	重复零次或一次
{n}	重复n次
{n,}	重复n次或更多次
{n,m}	重复n到m次

例子前面已经写过。

思考，如何匹配电话号码？ 比如(027)88881111 或者 027-88881111

分析一下，首先需要一个转义符 也就是最开始说的 \(这个括号可能出现 0 次或者 1 次所以后面就是个？ ，再接着是 0 和两个数字，于是就是 \d[2] ,后面又出现了一个)，或者-,或者没有，也就是?,然后接着的就是 8 个数字 也就是 \d{8} ，结果是什么？ 自己拼起来就是。

✓ **分支条件**

但是上面的例子其实是不完全的，比如 027)88881111 它也能匹配正确

所以，这时候 要告诉大家的是， 遇到这种问题就需要分支条件来判断了。 用 | 分隔符来分割不同的条件

优化后就是下面的：

0\d{2}-\d{8}|0\d{3}-\d{7}这个表达式能匹配两种以连字号分隔的电话号码：一种是三位区号，8 位本地号(如 010-12345678)，一种是 4 位区号，7 位本地号(0271-8888666)

✓ **分组**

还有一种情况，比如我要匹配一个 ip 地址，那么上面说的就感觉做不到了，这时候告诉大家另一个概念，分组

用小括号()来指定子表达式(也叫做分组)，然后你就可以指定这个子表达式的重复次数

(\d{1,3}\.){3}\d{1,3}是一个简单的 IP 地址匹配表达式

同样的，教你分析一下这个表达式

因为用了分组， 那么就一组一组的来读，\d{1,3}匹配 1 到 3 位的数字, (\d{1,3}\.){3}匹配三位数字加上一个英文句号(这个整体也就是这个分组)重复 3 次，最后再加上一个一到三位的数字(\d{1,3})

当然这个表达式其实也不完整，比如 256.256.256.256 这种错误的 ip 也能匹配到，那么怎么优化？ 自己可以思考下：

优化有的 ip 正则是 ((2[0-4]\d|25[0-5]|[01]?\d\d?)\.){3}(2[0-4]\d|25[0-5]|[01]?\d\d?)

✓ **反义**

最后教一下 反义

有时需要查找不属于某个能简单定义的字符类的字符。比如想查找除了数字以外，其它任意字符都行的情况，这时需要用到反义。

比如，要匹配 不包含空白符的字符串 ？ 怎么做？？？

常用的反义代码：

代码/语法	说明
\W	匹配任意不是字母，数字，下划线，汉字的字符
\S	匹配任意不是空白符的字符
\D	匹配任意非数字的字符
\B	匹配不是单词开头或结束的位置
[^x]	匹配除了x以外的任意字符
[^aeiou]	匹配除了aeiou这几个字母以外的任意字符

\S+ 就是用来匹配不包含空白符的字符串

如果你不觉得正则表达式很难读写的话，要么你是一个天才，要么，你不是地球人。正则表达式的语法很令人头疼，即使对经常使用它的人来说也是如此。

给大家介绍一个在线的正则表达式测试网站：http://tool.oschina.net/regex/

闭包

闭包是一个很抽象的概念，书上也说的很模糊，个人理解，要搞懂闭包，就要知道 js 的作用域链、垃圾回收机制、函数嵌套。

简单的来说，每个 function 其实就是一个闭包。

```
function a(){
    var i =1;
    function b (){
        alert(i+3);
    }
    return b;
}
var c =a();
c();
```

上面的代码有什么特点？

- ✧ b 函数嵌套在 a 函数里
- ✧ a 函数返回值为 b 函数
- ✧ c 函数指向 b 函数

✓ 作用域链

简单来说，作用域链就是函数在定义的时候创建的，用于寻找使用到的变量的值的一个索引。内部的规则是，把函数自身的本地变量放在最前面，把自身的父级函数中的变量放在其次，把再高一级的函数中的变量再放在后面，依次类推，一直到全局变量位置

```
自身变量--> 父级函数变量-->高一级函数变量--->...-->全局变量
```

于是函数查询一个变量的值的时候，会从最前面的变量查找，一直查找到全局变量。如果全局变量也没有值，那么久返回 undefined。

再来分析开始的代码：

1.先搜索自身的活动对象，如果存在则返回，如果不存在将继续搜索函数 a 的活动对象，依次查找，直到找到为止。

2.如果函数 b 存在 prototype 原型对象 则在查找完自身的活动对象后先查找自身的原型对象，再继续查找。这就是 Javascript 中的变量查找机制。

3.如果整个作用域链上都无法找到，则返回 undefined

可以做个测试，如下代码：

```
function a(){
    var i =1;
    function b(){
        alert(i+3);
    }
    return b;
}
var c= a(); //?
alert(c);
c();
```

还是这段代码， 如果我直接打印 c 会是什么？

自己 试一试， 就能很好的理解我说的引用了。

然后就能理解 ()(); 这种写法了。

✓ 垃圾回收机制

关于垃圾回收，一般来说，一个函数在创建，也就是执行的时候会给其中定义的变量划分内存空间来保持，以保证后面的使用，等函数执行完毕，返回(return)，对应的内存空间才会被回收，下次再执行的时候就是重新分配空间。

但是如果这个函数内部又嵌套了另一个函数,而这个函数是有可能在外部被调用到的.并且这个内部函数又使用了外部函数的某些变量的话.这种内存回收机制就会出现问题.如果在外部函数返回后,又直接调用了内部函数,那么内部函数就无法读取到他所需要的外部函数中变量的值了.所以 js 解释器在遇到函数定义的时候,会自动把函数和他可能使用的变量(包括本地变量和父级和祖先级函数的变量(自由变量))一起保存起来.这就是闭包了。

这些变量不会被内存回收掉，只有当内部的函数不可被调用或者删除了，才会销毁这个闭包，而没有任何一个闭包引用的变量才会被下一次的内存回收器给回收。

换句话说，有了闭包，嵌套函数的结构才可以正常执行。

此时再来看最开始的代码：

分析：a 返回函数 b 的引用给 c，又函数 b 的作用域链包含了对函数 a 的活动对象的引用，也就是说 b 可以访问到 a 中定义的所有变量和函数。函数 b 被 c 引用，函数 b 又依赖函数 a，因此函数 a 在返回后不会被 GC 回收。

看问题：为了让 foo 函数内部的函数能够适应循环的变量 i，那么该怎么实现？

常规的，想到的是如下的代码：

```
var res=[];
function foo(){
    for(var i =0;i<3;i++){
        res[i]=function(){
            alert(i);
        }
    }
};
foo();
res[0]();
res[1]();
res[2]();
```

但是，结果是什么？

根据前面说的，此时内部函数获取的变量实际上是 i 的索引值，也就是最后运行后的值 3，

也就是说闭包 function 引用的变量 i 实际上是个自由变量，也就是对 i 的引用，而不是 i 的值，当 i 改变后，闭包 function 里的值也会跟着改变。

那么，要达到目标效果怎么办？

思路： 在内部函数创建的时候捕获变量 i 的值并且保存，如下：

```
var res=[];
function foo(){
    for(var i =0;i<3;i++){
        res[i]=(function(j){
            alert(j);
        })(i);
    }
};
foo();
res[0]();
res[1]();
res[2]();
```

闭包的应用场景：

上面讲述完闭包，那么现在就说一下闭包的使用目的： 保护函数内的变量安全。

1) 比如开始的例子里，a 函数的 i 只有 b 函数能访问，而 c 函数是没法访问的，这样就保护了 i 的安全。
2) 在内存中维持一个变量。依然如前例，由于闭包，函数 a 中 i 的一直存在于内存中，因此每次执行 c()，都会给 i 自加 1。
3) 通过保护变量的安全实现 JS 私有属性和私有方法（不能被外部访问）

私有属性和方法在 Constructor 构造器外是无法被访问的

```
function Constructor(…) {
var that = this;
var name = value;
function name(…) {…}
}
```

cookie

再说说 cookie，由于 js 是运行在客户端的，所以不能直接设置 session，但是它可以操作浏览器的 cookie。

cookie 是一些键值对形式的数据(key=value)，当浏览器从服务器上请求 web 页面时， 属于该页面的 cookies 会被添加到该请求中。服务端通过这种方式来获取用户的信息。

在 js 中可以使用 document.cookie 属性来创建 、读取、及删除 cookies。

```
with(document){
    //创建cookie
    cookie="username=miss A;pwd=123456";
    //读取cookie
    console.log(cookie.split(";"[0].split("=")[1]));
    //删除cookie
    cookie="username=zhangsan;pwd=";
    console.log(cookie.split(";"[0].split("=")[1]));
}
```

```
//设置初始化cookie
function initCookie(){
    var cname="username";
    var cvalue= document.getElementById("username").value;
    var exdays= 30;
    var d = new Date();
    d.setTime(d.getTime()+(exdays*24*60*60*1000));
    var expires = "expires="+d.toGMTString();
    document.cookie = cname+"="+cvalue+"; "+expires;
    alert("当前cookie为: "+cname+"="+cvalue+"; ");
}
//设置cookie
function setCookie(cname,cvalue,exdays){
var d = new Date();
d.setTime(d.getTime()+(exdays*24*60*60*1000));
var expires = "expires="+d.toGMTString();
document.cookie = cname+"="+cvalue+"; "+expires;
}
//获取cookie
function getCookie(cname){
    var name = cname + "=";
    var ca = document.cookie.split(';');
    for(var i=0; i<ca.length; i++) {
        var c = ca[i].trim();
        if (c.indexOf(name)==0) return c.substring(name.length,c.length);
    }
    return "";
}
function checkCookie(){
    var user=getCookie("username");
    if (user!=""){
        alert("hello " + user);
    }
    else {
        alert("请输入你的姓名");
    }
}
```

```
//删除cookie
function removeCookie(){
    var user=getCookie("username");
    if(user!=""){
        user="";
        setCookie("username",user,30);
        alert("cookie已被删除");
    }
}
```

```
<form action=" " method="get" onsubmit="initCookie()">
    请输入姓名：<input id="username" type="text"  />
    <input type="submit" value="提交" />
</form>
<button type="button" onclick="checkCookie()">查看cookie</button>
<button type="button" onclick="removeCookie()">删除cookie</button>
```

完整的操作 cookie。

ajax

ajax：一种请求数据的方式，不需要刷新整个页面；

ajax 的技术核心是 XMLHttpRequest 对象；

ajax 请求过程：创建 XMLHttpRequest 对象、连接服务器、发送请求、接收响应数据；

常见的格式如下：

```
//封装的ajax
ajax({
    url: "test.do",                 //请求地址
    type: "POST",                       //请求方式
    data: { name: "c3gen", pwd: "123456" },        //请求参数
    dataType: "json",
    success: function (response) {
        // 此处放成功后执行的代码
        // 此处略去
    },
    fail: function (status) {
        // 此处放失败后执行的代码
    }
});
```

```
function ajax(options) {
    options = options || {};
    options.type = (options.type || "GET").toUpperCase();
    options.dataType = options.dataType || "json";
    //格式化参数
    var params = formatParams(options.data);

    //创建 - 非IE6 - 第一步
    if (window.XMLHttpRequest) {
        var xhr = new XMLHttpRequest();
    } else { //IE6及其以下版本浏览器
        var xhr = new ActiveXObject('Microsoft.XMLHTTP');
    }

    //接收 - 第三步
    xhr.onreadystatechange = function () {
        if (xhr.readyState == 4) {
            var status = xhr.status;
            if (status >= 200 && status < 300) {
                options.success && options.success(xhr.responseText, xhr.responseXML);
            } else {
                options.fail && options.fail(status);
            }
        }
    }
    //连接 和 发送 - 第二步
    if (options.type == "GET") {
        xhr.open("GET", options.url + "?" + params, true);
        xhr.send(null);
    } else if (options.type == "POST") {
        xhr.open("POST", options.url, true);
        //设置表单提交时的内容类型
        xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
        xhr.send(params);
    }
}
```

```
//格式化参数
function formatParams(data) {
    var arr = [];
    for (var name in data) {
        arr.push(encodeURIComponent(name) + "=" + encodeURIComponent(data[name]));
    }
    arr.push(("v=" + Math.random()).replace("."));
    return arr.join("&");
}
```

1) ajax 关键的 就是创建 XMLHttpRequest 对象，创建后就可以调用 xhr.open 方法
2) open()接收三个参数请求方式、请求地址、是否异步请求。
3) 然后调用 send()方法发生请求。
4) 最后就是接收返回的响应，收到响应后，响应的数据会自动填充 XHR 对象，相关属性如下

responseText：响应返回的主体内容，为字符串类型；

responseXML :如果响应的内容类型是 "text/xml" 或 "application/xml" ,这个属性中将保存着相应的 xml 数据，是 XML 对应的 document 类型；

status：响应的 HTTP 状态码；

statusText：HTTP 状态的说明；

**重点说一下 readyState 属性：

XHR 对象的 readyState 属性表示请求/响应过程的当前活动阶段，这个属性的值如下

0-未初始化，尚未调用 open()方法；

1-启动，调用了 open()方法，未调用 send()方法；

2-发送，已经调用了 send()方法，未接收到响应；

3-接收，已经接收到部分响应数据；

4-完成，已经接收到全部响应数据；

只要 readyState 的值变化，就会调用 readystatechange 事件，(其实为了逻辑上通顺，可以把 readystatechange 放到 send 之后，因为 send 时请求服务器，会进行网络通信，需要时间，在 send 之后指定 readystatechange 事件处理程序也是可以的，我一般都是这样用，但为了规范和跨浏览器兼容性，还是在 open 之前进行指定吧)。

5) readystatechange 事件。

在 readystatechange 事件中，先判断响应是否接收完成，然后判断服务器是否成功处理请求，xhr.status 是状态码，状态码以 2 开头的都是成功，304 表示从缓存中获取，上面的代码在每次请求的时候都加入了随机数，所以不会从缓存中取值，故该状态不需判断。

6) ajax 不可跨域！！！

jsonp

JSONP(JSON with Padding) 是一种跨域请求方式。主要原理是利用了 script 标签可以跨域请求的特点，由其 src 属性发送请求到服务器，服务器返回 js 代码，网页端接受响应，然后就直接执行了，这和通过 script 标签引用外部文件的原理是一样的。

JSONP 由两部分组成：回调函数和数据，回调函数一般是由网页端控制，作为参数发往服务器端，服务器端把该函数和数据拼成字符串返回。

```
<srcipt type="text/javascript" src="url?callback=jsonpCallback"/>
<script type="text/javascript">
    function jsonpCallback(result) {
        //alert(result);
    }
    var script = document.createElement("script");
    script.setAttribute("type", "text/javascript");
    script.src = src;
    document.body.appendChild(script);
</script>
```

为什么要跨域？

跨域的产生是因为浏览器的同源策略，也就是一个页面的 ajax 只能获取这个页面相同源或者相同域的数据。

什么叫同源或者同域？

也就是协议、域名、端口号都一样的 URL。

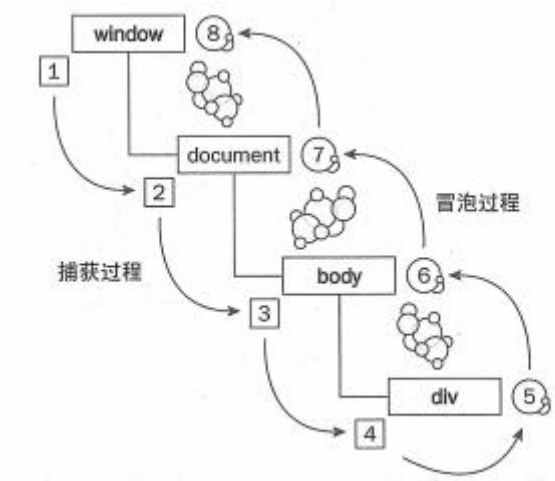
```
http://www.baidu.com  和  https://www.baidu.com 不同，因为协议不同；

http://localhost:8080  和  http://localhost:1000 不同，因为端口不同；

http://localhost:8080  和  https://www.baidu.com 不同，协议、域名、端口号都不同，根本不是一家的。
```

事件处理

事件，提供了与当前窗口交互的可能性。这里简单的介绍一下事件处理机制。

浏览器采用的是异步事件驱动的方式来协调文档、浏览器、元素以及对象的事件(event)的。

如下，一张经典的图。



当一个事件发生时，它会先从浏览器顶级对象 Window 一路向下传递，一直传递到触发这个事件的那个元素，这也就是事件捕获过程。然而，一切并没有结束，事件又从这个元素一路向上传递到 Window 对象，这也就是事件冒泡过程（但是在 IE8 以及其之前版本中，事件模型并未定义捕获过程，只有冒泡过程）。

问题来了：事件是在捕获过程处理的，还是在冒泡过程处理的？

```
    <style type="text/css">
        #div1{width: 300px; height: 300px; background: red; overflow:hidden;}
        #div2{margin:50px auto; width: 200px; height: 200px; background: green; overflow:hidden;}
        #div3{margin:50px auto; width: 100px; height: 100px; background: blue;}
    </style>

</head>
<body>
    <div id="div1" onCliCk="console.log('div1');">
            div1
        <div id="div2" oNClick="console.log('div2');">
                div2
            <div id="div3" onclick="console.log('div3');" onclick="console.log('div3333');">
                    div3
            </div>
        </div>
    </div>
</body>
```

如上代码，

①因为 HTML 里面不区分大小写，所以这里事件处理程序属性名大写、小写、大小混写均可，属性值就是相应事件处理程序的 JavaScript 代码；

②若给同一元素写多个 onclick 事件处理属性，浏览器只执行第一个 onclick 里面的代码，后面的会被忽略；

③这种形式是在事件冒泡过程中注册事件处理程序的；

```
<body>
    <div id="div1">
        div1<div id="div2">
            div2<div id="div3">
                div3
            </div>
        </div>
    </div>
    <script type="text/javascript">
        with(document){
            var div1=getElementById("div1");
            var div2=getElementById("div2");
            var div3=getElementById("div3");
            div1.onclick=function(){
                console.log("div1 click");
            }
            div2.onclick=function(){
                console.log("div2 click");
            }
            div3.onclick=function(){
                console.log("div3 click");
            }
            div2.onClick=function(){
                console.log("div1 11");
            }
            div1.onclick=function(){
                console.log("div1 999");
            }
        }
    </script>
</body>
```

如上代码：

(1) js 严格区分大小写，所以 div2.onClick 是不起作用的

(2) 若给同一元素对象写多个 onclick 事件处理属性，后面写的会覆盖前面的（ps：这就是在修改一个对象属性的值，属性的值是唯一确定的）

(3) 这种形式也是在事件冒泡过程中注册事件处理程序的

✧ addEventListener()

包括 Window 对象、Document 对象和所有文档元素等——都定义了一个名叫 addEventListener()的方法，使用这个方法可以为事件目标注册事件处理程序。

addEventListener()接受三个参数：第一个参数是要注册处理程序的事件类型，其值是字符串，但并不包括前缀“on”；第二个参数是指当指定类型的事件发生时应该调用的函数；第三个参数是布尔值，其可以忽略（某些旧的浏览器上不能忽略这个参数），默认值为 false。这种情况是在事件冒泡过程中注册事件处理程序。当其为 true 时，就是在事件捕获过程中注册事件处理程序。

```
<body>
    <div id="div1">
        div1<div id="div2">
            div2<div id="div3">
                div3
            </div>
        </div>
    </div>
    <script type="text/javascript">
        with(document){
            var div1=getElementById("div1");
            var div2=getElementById("div2");
            var div3=getElementById("div3");
            div1.addEventListener('click', function(){ console.log('div1-bubble'); }, false);
            div2.addEventListener('click', function(){ console.log('div2-bubble'); }, false);
            div3.addEventListener('click', function(){ console.log('div3-bubble'); }, false);
            div3.addEventListener('click', function(){ console.log('div3-bubble222'); }, false);
            div1.addEventListener('click', function(){ console.log('div1-capturing'); }, true);
            div2.addEventListener('click', function(){ console.log('div2-capturing'); }, true);
            div3.addEventListener('click', function(){ console.log('div3-capturing'); }, true);
        }
    </script>
</body>
```

(1) addEventListener()第三个参数的作用正如上面所说；

(2) 通过 addEventListener()方法给同一对象注册多个同类型的事件，并不会发生忽略或覆盖，而是会按顺序依次执行；

(3) 相对 addEventListener()的是 removeEventListener()方法，它同样有三个参数，前两个参数自然跟 addEventListener()的意义一样，而第三个参数也只需跟相应的 addEventListener()的第三个参数保持一致即可，同样可以省略，默认值为 false。

**处理顺序的调用

```
    <style type="text/css">
        #div1{width: 300px; height: 300px; background: red; overflow:hidden;}
    </style>

</head>
<body>
    <div id="div1" onclick="console.log('html:'); console.log(this);">div1</div>
</body>
```

思考：this 指向是什么？？

```
<div id="div1" onclick="console.log('html:'); console.log(this);">div1</div>
<script type="text/javascript">
    var div1 = document.getElementById('div1');
    div1.onclick = function(){
     console.log('div1.onclick:');
     console.log(this);
     };
</script>
```

再接着看第二段代码，this 又指什么？

ps：两种方式 this 指向的都是元素本身

第二种，会覆盖第一种方法注册的事件处理。

所以这里事件处理的顺序就清楚了：

(1) 通过 HTML 属性注册的处理程序和通过设置对象属性的处理程序一直优先调用；

(2) 使用 addEventListener()注册的处理程序按照它们的注册顺序依次调用；

**事件取消

事件取消很实用，也比较难搞清楚，我就讲两部分，取消事件的默认操作和取消事件的传播。

A> 取消事件的默认操作

取消事件的浏览器默认操作（比如点击超链接元素会自动发生页面跳转的默认操作）：如果使用前两种方法注册事件处理程序，可以在处理程序中添加返回值 false 来取消事件的浏览器默认操作。在支持 addEventListener()的浏览器中，也可以通过调用事件对象的 preventDefault()方法取消事件的默认操作。至于 IE8 及其之前的浏览器可以通过设置事件对象的 returnValue 属性为 false 来取消事件的默认操作。

```
function cancleHandler(event){
    var event = event || window.event;
    if(event.preventDefault){
        event.preventDefault();
    }
    if(event.returnValue){
        event.returnValue= false;
    }
    return false;
}
```

B> 取消事件的传播

取消事件传播：在支持 addEventListener()的浏览器中，可以调用事件对象的一个 stopPropagation()方法阻止事件的继续传播，它能工作在事件传播期间的任何阶段（捕获期阶段、事件目标本身、冒泡阶段）；但是在 IE8 以及其之前版本的浏览器中并不支持 stopPropagation()方法，而且这些浏览器也不支持事件传播的捕获阶段，相应的，IE 事件对象有一个 cancelBubble 属性，设置这个属性为 true 能阻止事件进一步传播（即阻止其冒泡）

```
with(document){
    var div1=getElementById("div1");
    var div2=getElementById("div2");
    var div3=getElementById("div3");
    div1.onclick= function(){
        console.log("div1");
    }
    div2.onclick=function(){
        console.log("div2");
    }
    //阻止div3 处的点击事件冒泡到div1和div2
    div3.onclick=function(event){
        stopEvent(event);
        console.log("div3");
    }
    //阻止冒泡事件
    function stopEvent(event){
        var event=event || window.event;
        if(event.stopPropagation){
            event.stopPropagation();
        }else{
            event.cancelBubble=true;
        }

    }
}
```

好了，基本上是写完了 js 的部分了，手打的，有些内容有错误请联系我。

参考 api ： https://developer.mozilla.org/zh-CN/docs/Web/JavaScript MDN 是学习的最佳伴侣， 安装一个百度翻译插件或者有道翻译插件，看不懂的直接整页翻译吧。