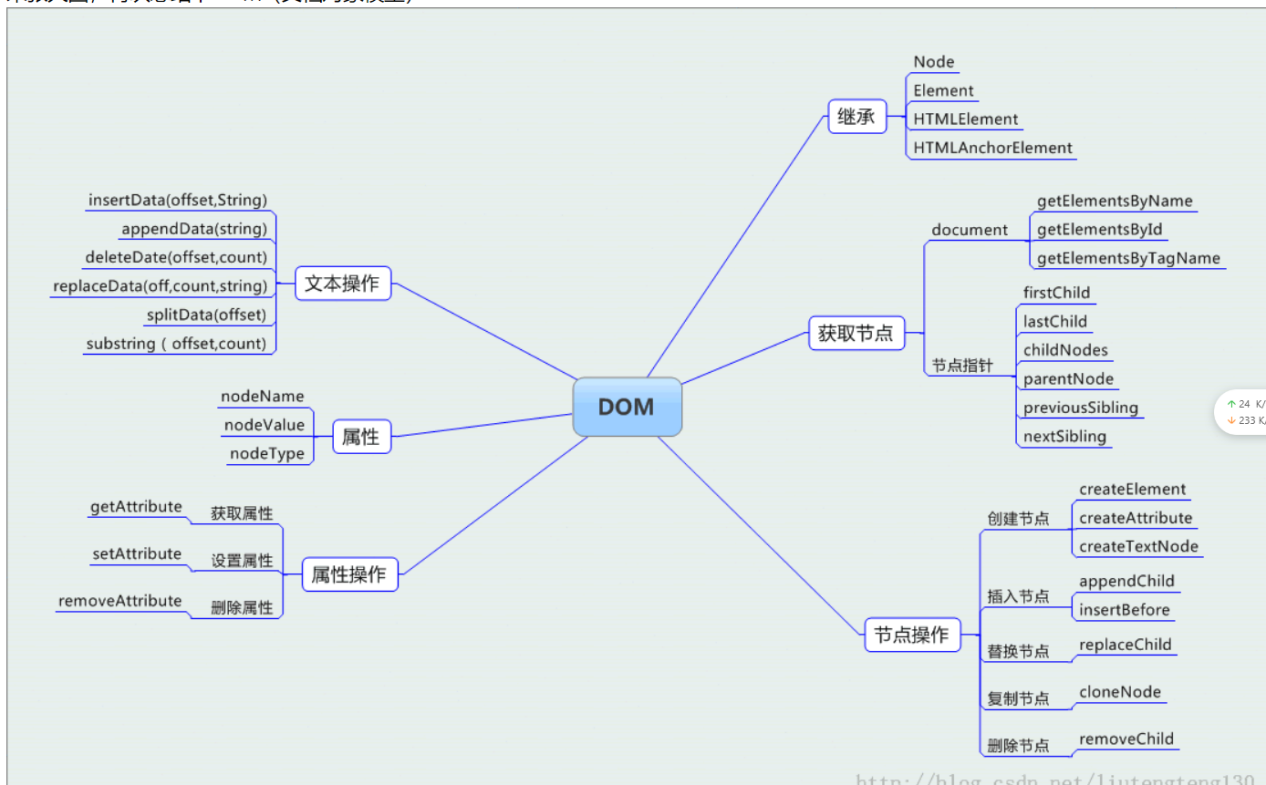
Document

文档操作



/\*

JavaScript 事件参考



\*/

1. j1

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>hello world!!!</title>

<script type="text/javascript">

// alert("1hello world!!!");//弹窗

// document.write("2hello world!!");//页面

// console.log("3hello world!!!");//控制台

//int a=10,b=20;

var a=10,b=20;

//int temp=a+b;

var temp=a+b;

// System.out.print();

alert(temp);

</script>

</head>

<body>

</body>

</html>

Sj2

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>引用js代码，数据类型，正则表达式</title>

<script type="text/javascript" src="js/s1.js"></script>

</head>

<body>

</body>

</html>

Sj3

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>全局函数，自定义函数,自定义对象</title>

<script type="text/javascript" src="js/s2.js"></script>

</head>

<body>

</body>

</html>

Sj4

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>BOM对象 定时器 location</title>

<script type="text/javascript" src="js/s3.js"></script>

</head>

<body>

</body>

</html>

Sj5

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>hello包含</title>

</head>

<body>

<!--可以将js代码编写到标签的onclick属性中

当我们点击按钮时，js代码才会执行

虽然可以写在标签的属性中，但是他们属于结构与行为耦合，不方便维护，不推荐使用

-->

<button onclick="alert('你好')">hi</button>

<a href="javascript:alert('hhhh')">ohhh</a>

<!--可以将js代码写在超链接的href属性中，这样当点击超链接时，会执行js代码-->

<a href="javascript:">ohhh</a>

</body>

</html>

Js1

// alert("hello world!!!");

// var str="hello world";输出

// alert(str);输出

// document.write(str);输出

/\*document.writeln(typeof(str));

str=10;

document.writeln(typeof(str));

str=true;

document.writeln(typeof(str));

str=12.03;

document.writeln(typeof(str));

str=null;

document.writeln(typeof(str));\*/

//typeof()显示数据类型

/\*var str=10;

var str2="10";

document.writeln(str==str2);

document.writeln(str===str2);

===全等，必须要类型和值都相同\*/

/\* var reg=/^\s\*$/;//0到多个空格

document.write(reg.test(" a "));

必须全部字符符合正则返回true

有一个字符不符合正则就会返回false\*/

/\*var reg=/^\s+$/;//直接方式

document.write(reg.test(" we"));

var reg1=/\s+/;//普通方式

document.write(reg1.test(" qeq"));\*/

/\*/^表达式$/

只要有无法成立正则的字符存在，即为false。。

全部符合为true.

(检查严格，眼睛不揉沙子)。

适用于表单校验。

/表达式/

只要有成立正则的字符存在，即为true。。

全部不符合为false

(检查不严格，懒人思想)。

适用于字符串查找、替换。\*/

/\*var arr= [1,2,3,"a",true];//常用的JS数组。长度5。

var arr= new Array();创建一一个 数组对象，数组长度默认为0。\*/

/\*var ar=[1,2,33,4,"qeq"];

document.write(ar.length);

ar[1]="qqq";

ar[ar.length]="qqqwq";//可以给ar直接增加空间，添加元素

document.writeln(ar);

ar[10]=10;

document.writeln(ar);

document.writeln(ar.length);\*/

/\*var arr=['a','b','cc'];

document.writeln(arr.join('@'));//链接数组

document.writeln(arr.reverse());//反转数组

document.writeln(arr.pop());//删除最后一个元素并且返回数组长度

document.writeln(arr.push('qwq','mmm')); //向数组的末尾添加一个或更多元素，并返回新的长度。

document.writeln(arr.reverse())//颠倒数组中元素的顺序\*/

Js2

/\*

var msg="https://www.bilibili.com/video/av73840600?p=60小飞";

var temp=encodeURI(msg);//16进制编码转换

document.writeln(temp);

document.writeln(decodeURI(temp));//解码

\*/

/\*

var str="10.123";

var str2="10.123a";

var num1=parseInt(str);

var num2=parseFloat(str);

var num3=parseInt(str2);

var num4=parseFloat(str2);

document.writeln(typeof (num1)+num1);

document.writeln(typeof (num2)+num2);

document.writeln(typeof (num3)+num3);

document.writeln(typeof (num4)+num4);

/!\*如果字符串的某个字符从字面值上无法转为数字,那么从该字符开始停止转换,仅返回

前面正确的转换值。(例如: 11.5a55， parselnt 结果11，parseFloat 结果11.5)。

+

如果字符串的第-一个字符就无法以字面值上转为数字，那么停止转换，返回NaN.

NaN (Not A Number, -一个数字类型的标识，表示不是一个正确的数字)。

\*!/\*/

/\*

function getSum(a,b){

return a+b;

}

function getSum(a,b,c){

return a+b+c;

}

document.writeln(getSum(1,2));

document.writeln(getSum(1,2,3));

/!\*

JavaScript函数定义必须用小写的function;

JavaScript函数无需定义返回值类型，直接在function后面书写方法名;。

参数的定义无需使用var关键字，否则报错;

JavaScript函数体中,return可以不写，也可以return具体值,或者仅仅写return;。

JavaScript函数调用执行完毕一定有返回值，值及类型根据return决定，如果未return

具体值，返回值为undefined;

JavaScript函数若同名，则不存在方法重载，只有方法相互覆盖，最后定义的函数覆盖

之前的定义;。

因为JavaScript不存在函数重载,所以JavaScript仅根据方法名来调用函数,即使实参与

函数的形参不匹配，也不会影响正常调用;

\*!/\*/

/\*

function Peron(){

this.name="小飞";

this.age=20;

}

var a =new Peron();

document.writeln(a.name);

document.writeln(a.age);

\*/

/\*var per={name:"小飞",age:"20"};

document.writeln(per.age+per.name);\*/

Js3

/\*

确认窗口，有返回值

var a;

a=confirm("hello");

document.writeln(a);\*/

/\*var id;

function run1(){

alert("hello");

clearInterval(id);/!\*取消定时器\*!/

}

setInterval("run1()",2000);/!\*定时器，时间单位是毫秒\*!/

id=setInterval("run1()",2000);\*/

/\*

1,尽可能在setInterval中自己清除（停止）。（设置判断）

2、如果将setInterval赋给一个全局变量，

在其他地方ClearInterval进行清除有可能达不到效果。

(原因：那边刚清除，这边可能又重新赋值了，惨痛的经历！）

var timesRun = 0;

var interval = setInterval(function(){

timesRun += 1;

if(timesRun === 2){

clearInterval(interval);

}

document.writeln("hello");

//do whatever here..

}, 2000);\*/

/\*function run(){

document.writeln("hello");

}

setTimeout("run()",2000);/!\*一次性定时器 取消和循环定时器一样\*!/

\*/

/\*var a=location.href;

document.writeln(a);/!\*输出当前地址\*!/

location.href="study24.html";/!\*跳转对应地址,设置href属性，浏览器就会跳转到对应的路径。\*!/

\*/

/\*function a{

location.href="https://www.bilibili.com/";

}

setTimeout(a,2000);/!\*延迟跳转\*!/\*/

//开始

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>hello world!!!</title>

<script type="text/javascript">

// alert("1hello world!!!");//弹窗

// document.write("2hello world!!");//页面

// console.log("3hello world!!!");//控制台

//int a=10,b=20;

var a=10,b=20;

//int temp=a+b;

var temp=a+b;

// System.out.print();

alert(temp);

</script>

</head>

<body>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Title</title>

</head>

<body>

<script>

function Person(name,age){

this.name=name;

this.age=age;

this.get=function (){

document.writeln(this.name);

};

}

var a=new Person("zzp",20);

a.get();

</script>

</body>

</html>

<!--

/\*

\* 创建一个构造函数，专门用来创建Person对象的

\* 构造函数就是一个普通的函数，创建方式和普通函数没有区别,

\* 不同的是构造函数习惯上首字母大写

\*

\* 构造函数和普通函数的区别就是调用方式的不同

\* 普通函数是直接调用，而构造函数需要使用new关键字来调用

\*

\* 构造函数的执行流程：

\* 1.立刻创建一个新的对象

\* 2.将新建的对象设置为函数中this,在构造函数中可以使用this来引用新建的对象

\* 3.逐行执行函数中的代码

\* 4.将新建的对象作为返回值返回

\*

\* 使用同一个构造函数创建的对象，我们称为一类对象，也将一个构造函数称为一个类。

\* 我们将通过一个构造函数创建的对象，称为是该类的实例

\*

\* this的情况：

\* 1.当以函数的形式调用时，this是window

\* 2.当以方法的形式调用时，谁调用方法this就是谁

\* 3.当以构造函数的形式调用时，this就是新创建的那个对象

\*

\*/

-->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>原型</title>

</head>

<body>

<script>

function Person(name,age){

this.name=name;

this.age=age;

this.pget=get;

}

function get(){

document.writeln(this.name);

};

/\* 创建一个Person构造函数

\* - 在Person构造函数中，为每一个对象都添加了一个sayName方法，

\* 目前我们的方法是在构造函数内部创建的，

\* 也就是构造函数每执行一次就会创建一个新的sayName方法

\* 也是所有实例的sayName都是唯一的。

\* 这样就导致了构造函数执行一次就会创建一个新的方法，

\* 执行10000次就会创建10000个新的方法，而10000个方法都是一摸一样的

\* 这是完全没有必要，完全可以使所有的对象共享同一个方法

\*/

var a=new Person("zzp",20);

a.pget();

</script>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Title</title>

</head>

<body>

<script>

function Person(name,age){

this.name=name;

this.age=age;

// this.pget=get;

}

// function get(){

// document.writeln(this.name,this.age);

// };

Person.prototype.get=function (){

document.writeln(this.name,this.age);

}

var a=new Person("zzp",20);

a.get();

document.writeln(a.prototype);

document.writeln(a.\_\_proto\_\_);

document.writeln("get" in a);/\*判断get在不在a中，在原型也相当于在函数中\*/

document.writeln(a.hasOwnProperty("get"))/\*判断get在不在a中,原型中的不算\*/

</script>

</body>

</html>

<!--

/\* 原型 prototype

\*

\* 我们所创建的每一个函数，解析器都会向函数中添加一个属性prototype

\* 这个属性对应着一个对象，这个对象就是我们所谓的原型对象

\* 如果函数作为普通函数调用prototype没有任何作用

\* 当函数以构造函数的形式调用时，它所创建的对象中都会有一个隐含的属性，

\* 指向该构造函数的原型对象，我们可以通过\_\_proto\_\_来访问该属性

\*

\* 原型对象就相当于一个公共的区域，所有同一个类的实例都可以访问到这个原型对象，

\* 我们可以将对象中共有的内容，统一设置到原型对象中。

\*

\* 当我们访问对象的一个属性或方法时，它会先在对象自身中寻找，如果有则直接使用，

\* 如果没有则会去原型对象中寻找，如果找到则直接使用

\*

\* 以后我们创建构造函数时，可以将这些对象共有的属性和方法，统一添加到构造函数的原型对象中，

\* 这样不用分别为每一个对象添加，也不会影响到全局作用域，就可以使每个对象都具有这些属性和方法了

\*/-->

<!DOCTYPE html>

<html lang="zh">

<head>

<meta charset="UTF-8">

<title>数组</title>

</head>

<body>

<script>

var arr=[];

arr[0]=12;

arr[1]=20;

document.writeln(arr[0]+"@");

var arr1=[12,55,4,2];

document.writeln(arr1+"@");

arr1.push("zzp");/\*向末尾添加，并且会返回添加后的长度\*/

document.writeln(arr1+"@");

arr1.pop();/\*删除最后一个元素,并且返回删除的值\*/

document.writeln(arr1+"@");

arr1.unshift("szr");/\*向前面添加元素，并且返回添加后的长度\*/

document.writeln(arr1+"@");

</script>

</body>

</html>

<!--

push() 向末尾添加，并且会返回添加后的长度

pop() 删除最后一个元素,并且返回删除的值

unshift() 向前面添加元素，并且返回添加后的长度

shift 删除前面第一个元素,并且返回删除的值

-->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>数组，原型</title>

</head>

<body>

<script type="text/javascript">

function Person(name , age){

this.name = name;

this.age = age;

}

//修改Person原型的toString

Person.prototype.toString = function(){

return "Person[name="+this.name+",age="+this.age+"]";

};

//创建一个Person对象

var per = new Person("孙悟空",18);

var per2 = new Person("猪八戒",28);

var per3 = new Person("红孩儿",8);

var per4 = new Person("蜘蛛精",16);

var per5 = new Person("二郎神",38);

/\*

\* 将这些person对象放入到一个数组中

\*/

var perArr = [per,per2,per3,per4,per5];

function getAdult(perArr){

var nArr=[];

for (var i=0;i<perArr.length;i++){

if (perArr[i].age>18) nArr.push(perArr[i])

}

return nArr;

}/\*筛选年龄大于18岁的人\*/

var a=getAdult(perArr);

document.writeln(perArr);

document.writeln(a);

</script>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>forEach 遍历 slice 选择</title>

</head>

<body>

<script>

var arr=["zzp","szr","dhf","ll"];

arr.forEach(function (a,b,c){

document.writeln(a) ;/\*返回正在遍历的值\*/

document.writeln(b) ;/\*返回正在遍历的编号\*/

/\*document.writeln(c) ;/!\*返回全部\*!/\*/

}

)

document.writeln("@@@")

document.writeln(arr.slice(0,2));

document.writeln("@@@@")

arr.unshift("hh","uu","ee");

document.writeln(arr.splice(0,2,"wu","qwq"));

</script>

</body>

</html>

<!--

forEach( )方法需要一个函数作为参数

-

像这种函数，由我们创建但是不由我们调用的，我们称为回调函数

-

数组中有几个元素函数就会执行几次，每次执行时，浏览器会将遍历到的元素

以实参的形式传递进来，我们可以来定义形参，来读取这些内容

浏览器会在回调函数中传递三个参数:

-->

<!--

slice(start,end )

可以用来从数组提取指定元素

不会影响原数组，是调用一个新赋值的数组

参数:

1.截取开始的位置的索引,包含开始索引

2.截取结束的位置的索引，不包含结束索引

第二个参数可以省略不写,此时会截取从开始索引往后的所有元素

-索引可以传递一个负值，如果传递一一个负值，则从后往前计算

-1倒数第一个

-->

<!--

/\*

\* splice()

\* - 可以用于删除数组中的指定元素

\* - 使用splice()会影响到原数组，会将指定元素从原数组中删除

\* 并将被删除的元素作为返回值返回

\* - 参数：

\* 第一个，表示开始位置的索引

\* 第二个，表示删除的数量

\* 第三个及以后。。

\* 可以传递一些新的元素，这些元素将会自动插入到开始位置索引前边

\*

\*/

-->

<!DOCTYPE html>

<html lang="zh">

<head>

<meta charset="UTF-8">

<title>删除数组相同多余元素</title>

</head>

<body>

<script>

var arr=[1,1,2,1,2,3,4,5,4,4,5,1,1,2,1,2,3,4,5,4,4,5]

//方法一

function comArr(arr, a){

for (var i=0;i<arr.length;i++){

if (arr[i]===a) return true;

}

}

function chaArr(arr){

var arr1=[arr[0]];

var arr2=[];

for (var i=1,j=1,q=0;i<arr.length;i++)

{

if(comArr(arr2,arr[i])) ;

else if (comArr(arr1,arr[i])) arr2[q++]=arr[i];

else arr1[j++]=arr[i];

}

return arr1;

}//不会改变原数组

document.writeln(chaArr(arr));

// document.writeln(chaArr(arr).length);

//方法二

function chaArr2(arr){

for (var i=0;i<arr.length;i++){

for (var j=i+1;j<arr.length;j++){

if(arr[i]===arr[j]) {

arr.splice(j,1);

j--;

}

}

}

}//会改变原数组

chaArr2(arr);

document.writeln(arr);

</script>

</body>

</html>

<!DOCTYPE html>

<html lang="zh">

<head>

<meta charset="UTF-8">

<title>数组剩余算法</title>

</head>

<body>

<script type="text/javascript">

var arr=["zzp","szr","dhf","ll"];

var arr1=[1,2,3,4,5];

document.writeln(arr.concat(arr1));//连接

document.writeln(arr.join("@"));//转换成字符串

document.writeln(arr.reverse());//反转

var arr2=[21,15,548,545,1,11,5];

var arr3=["wq","eqw","wwsa","qqw"];

document.writeln(arr2.sort());//排序

document.writeln(arr3.sort());//排序

</script>

</body>

</html>

<!--

concat()可以连接两个或多个数组，并将新的数组返回(也可以传元素)

\* - 该方法不会对原数组产生影响

-->

<!-- \* join()

\* - 该方法可以将数组转换为一个字符串

\* - 该方法不会对原数组产生影响，而是将转换后的字符串作为结果返回

\* - 在join()中可以指定一个字符串作为参数，这个字符串将会成为数组中元素的连接符

\* 如果不指定连接符，则默认使用,作为连接符

-->

<!--

reverse()

\* - 该方法用来反转数组（前边的去后边，后边的去前边）

\* - 该方法会直接修改原数组

-->

<!--

/\*

\* sort()

\* - 可以用来对数组中的元素进行排序

\* - 也会影响原数组，默认会按照Unicode编码进行排序

\* 即使对于纯数字的数组，使用sort()排序时，也会按照Unicode编码来排序，

\* 所以对数字进排序时，可能会得到错误的结果。

\*

\* 我们可以自己来指定排序的规则

\* 我们可以在sort()添加一个回调函数，来指定排序规则，

\* 回调函数中需要定义两个形参,

\* 浏览器将会分别使用数组中的元素作为实参去调用回调函数

\* 使用哪个元素调用不确定，但是肯定的是在数组中a一定在b前边

\* - 浏览器会根据回调函数的返回值来决定元素的顺序，

\* 如果返回一个大于0的值，则元素会交换位置

\* 如果返回一个小于0的值，则元素位置不变

\* 如果返回一个0，则认为两个元素相等，也不交换位置

\*

\* - 如果需要升序排列，则返回 a-b

\* 如果需要降序排列，则返回b-a

\*/

-->

<!DOCTYPE html>

<html lang="zh">

<head>

<meta charset="UTF-8">

<title>Title</title>

</head>

<body>

<script>

function fun(){

document.writeln(arguments.length);

document.writeln(arguments[0]);

document.writeln(arguments.callee===fun);

}

fun("hello",true);

</script>

</body>

</html>

<!--

在调用函数时，浏览器每次都会传递进两个隐含的参数：

\* 1.函数的上下文对象 this

\* 2.封装实参的对象 arguments

\* - arguments是一个类数组对象,它也可以通过索引来操作数据，也可以获取长度

\* - 在调用函数时，我们所传递的实参都会在arguments中保存

\* - arguments.length可以用来获取实参的长度

\* - 我们即使不定义形参，也可以通过arguments来使用实参，

\* 只不过比较麻烦

\* arguments[0] 表示第一个实参

\* arguments[1] 表示第二个实参 。。。

\* - 它里边有一个属性叫做callee，

\* 这个属性对应一个函数对象，就是当前正在指向的函数的对象

\*

-->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Date Math</title>

</head>

<body>

<script>

let d1=new Date("11/7/2000 0:0:0");

document.writeln(d1.getTime());

document.writeln("@")

let d2=Date.now();

document.writeln(d2);

document.writeln(Math.PI);

</script>

</body>

</html>

<!--

Date对象

\* - 在JS中使用Date对象来表示一个时间

\*/

//创建一个Date对象

//如果直接使用构造函数创建一个Date对象，则会封装为当前代码执行的时间

var d = new Date();

//创建一个指定的时间对象

//需要在构造函数中传递一个表示时间的字符串作为参数

//日期的格式 月份/日/年 时:分:秒

-->

<!--

\* Math

\* - Math和其他的对象不同，它不是一个构造函数，

\* 它属于一个工具类不用创建对象，它里边封装了数学运算相关的属性和方法

\* - 比如

\* Math.PI 表示的圆周率

\*/

//console.log(Math.PI);

/\*

\* abs()可以用来计算一个数的绝对值

\*/

//console.log(Math.abs(-1));

/\*

\* Math.ceil()

\* - 可以对一个数进行向上取整，小数位只有有值就自动进1

\* Math.floor()

\* - 可以对一个数进行向下取整，小数部分会被舍掉

\* Math.round()

\* - 可以对一个数进行四舍五入取整

\*/

//console.log(Math.ceil(1.1));

//console.log(Math.floor(1.99));

//console.log(Math.round(1.4));

/\*

\* Math.random()

\* - 可以用来生成一个0-1之间的随机数

\* - 生成一个0-10的随机数

\* - 生成一个0-x之间的随机数

\* Math.round(Math.random()\*x)

\*

\* - 生成一个1-10

\* - 生成一个x-y之间的随机数

\* Math.round(Math.random()\*(y-x)+x)

\*/

/\*for(var i=0 ; i<100 ; i++){

//console.log(Math.round(Math.random()\*10));

//console.log(Math.round(Math.random()\*20));

//console.log(Math.round(Math.random()\*9)+1);

//console.log(Math.round(Math.random()\*8)+2);

//生成1-6之间的随机数

console.log(Math.round(Math.random()\*5+1));

}\*/

/\*

\* max() 可以获取多个数中的最大值

\* min() 可以获取多个数中的最小值

\*/

-->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>引用js代码，数据类型，正则表达式</title>

<script type="text/javascript" src="js/s1.js"></script>

</head>

<body>

</body>

</html>

<!DOCTYPE html>

<html lang="zh">

<head>

<meta charset="UTF-8">

<title>字符串</title>

</head>

<body>

<script>

let str="hello world!";

let str1="lbw";

console.log(str.charAt(2));//返回索引的字符

console.log(str.charCodeAt(2));//根据索引查找编码（Unicode编码）

console.log(String.fromCharCode(20013));//根据编码输出字符（Unicode编码）

console.log(str.concat(str1));//连接

console.log(str.indexOf("h"));//输出参数位置

console.log(str.lastIndexOf("h"));//输出按后面算参数位置

console.log(str.slice(0,5));//截取字符

console.log(str.slice(0,1));//截取字符

console.log(str.split("o"));//以参数为分隔，分割字符为数组

console.log(str.toUpperCase());//大写

console.log(str.toLowerCase());//小写

</script>

</body>

</html>

<!--

/\*

\* 在底层字符串是以字符数组的形式保存的

\* ["H","e","l"]

\*/

/\*

\* length属性

\* - 可以用来获取字符串的长度

\*/

//console.log(str.length);

//console.log(str[5]);

/\*

\* charAt()

\* - 可以返回字符串中指定位置的字符

\* - 根据索引获取指定的字符

\*/

\*

\* charCodeAt()

\* - 获取指定位置字符的字符编码（Unicode编码）

\*/

/\*

\* String.formCharCode()

\* - 可以根据字符编码去获取字符

\*/

/\*

\* concat()

\* - 可以用来连接两个或多个字符串

\* - 作用和+一样

\*/

-->

<!--

/\*

\* indexOf()

\* - 该方法可以检索一个字符串中是否含有指定内容

\* - 如果字符串中含有该内容，则会返回其第一次出现的索引

\* 如果没有找到指定的内容，则返回-1

\* - 可以指定一个第二个参数，指定开始查找的位置

\*

\* lastIndexOf();

\* - 该方法的用法和indexOf()一样，

\* 不同的是indexOf是从前往后找，

\* 而lastIndexOf是从后往前找

\* - 也可以指定开始查找的位置

\*/

\* slice()

\* - 可以从字符串中截取指定的内容

\* - 不会影响原字符串，而是将截取到内容返回

\* - 参数：

\* 第一个，开始位置的索引（包括开始位置）

\* 第二个，结束位置的索引（不包括结束位置）

\* - 如果省略第二个参数，则会截取到后边所有的

\* - 也可以传递一个负数作为参数，负数的话将会从后边计算

substring()

\* - 可以用来截取一个字符串，可以slice()类似

\* - 参数：

\* - 第一个：开始截取位置的索引（包括开始位置）

\* - 第二个：结束位置的索引（不包括结束位置）

\* - 不同的是这个方法不能接受负值作为参数，

\* 如果传递了一个负值，则默认使用0

\* - 而且他还自动调整参数的位置，如果第二个参数小于第一个，则自动交换

\*/

substr()

\* - 用来截取字符串

\* - 参数：

\* 1.截取开始位置的索引

\*

\* split()

\* - 可以将一个字符串拆分为一个数组

\* - 参数：

\* -需要一个字符串作为参数，将会根据该字符串去拆分数组

\*/

/\*

\* toUpperCase()

\* - 将一个字符串转换为大写并返回

\*/

\* toLowerCase()

\* -将一个字符串转换为小写并返回

\*/

-->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>正则表达式</title>

</head>

<body>

<script>

let reg=new RegExp("a","i");

let reg1=/l/i;/\*简便方法\*/

let str="a";

console.log(reg.test(str));

console.log(reg.test("All"));

console.log(reg1.test("All"));

let reg2=/[ab]/i;

let reg3=/[a-z]/;

console.log(reg2.test("adc"))

console.log(reg3.test("abcdef"));

let reg4=/a[b-z]c/;

console.log(reg4.test("aecaa"));

let reg5=/[^ab]/;

console.log(reg5.test("ab"));

</script>

</body>

</html>

<!--

\* 正则表达式

\* - admin@atguigu.com

\* - admin@.com adminatguigu.com

\* - 邮件的规则：

\* 1.前边可以是xxxx乱七八糟

\* 2.跟着一个@

\* 3.后边可以是xxxx乱七八糟

\* 4..com获取其他的乱七八糟

\*

\* - 正则表达式用于定义一些字符串的规则，

\* 计算机可以根据正则表达式，来检查一个字符串是否符合规则，

\* 获取将字符串中符合规则的内容提取出来

\*/

//创建正则表达式的对象

/\*

\* 语法：

\* var 变量 = new RegExp("正则表达式","匹配模式");

\* 使用typeof检查正则对象，会返回object

\* var reg = new RegExp("a"); 这个正则表达式可以来检查一个字符串中是否含有a

\* 在构造函数中可以传递一个匹配模式作为第二个参数，

\* 可以是

\* i 忽略大小写

\* g 全局匹配模式-->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>正则表达式</title>

</head>

<body>

<script>

let str="d1a2d1ada12d1s5d45c45x48a4";

console.log(str.split(/[A-z]/));

let str1="abc avc acc aaa";

console.log(str1.search(/a[bde]c/));

console.log(str.match(/[A-z]/g));

console.log(str.replace(/a/g,"@"));

console.log(str.replace(/[a-z]/g,""));

</script>

</body>

</html>

<!--

split()

\* - 可以将一个字符串拆分为一个数组

\* - 方法中可以传递一个正则表达式作为参数，这样方法将会根据正则表达式去拆分字符串

\* - 这个方法即使不指定全局匹配，也会全都插分

\*/

/\*

\* search()

\* - 可以搜索字符串中是否含有指定内容

\* - 如果搜索到指定内容，则会返回第一次出现的索引，如果没有搜索到返回-1

\* - 它可以接受一个正则表达式作为参数，然后会根据正则表达式去检索字符串

\* - search()只会查找第一个，即使设置全局匹配也没用

\*/

\*

\* match()

\* - 可以根据正则表达式，从一个字符串中将符合条件的内容提取出来

\* - 默认情况下我们的match只会找到第一个符合要求的内容，找到以后就停止检索

\* 我们可以设置正则表达式为全局匹配模式，这样就会匹配到所有的内容

\* 可以为一个正则表达式设置多个匹配模式，且顺序无所谓

\* - match()会将匹配到的内容封装到一个数组中返回，即使只查询到一个结果

\*

\* replace()

\* - 可以将字符串中指定内容替换为新的内容

\* - 参数：

\* 1.被替换的内容，可以接受一个正则表达式作为参数

\* 2.新的内容

\* - 默认只会替换第一个

-->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>正则表达式 手机号正则表达式 电子邮件正则表达式</title>

</head>

<body>

<script>

let re1=/^\w+(\.(jpg|bmp|png))$/;

console.log(re1.test("www.jpg"));

let reg=/a{3}/;

let reg1=/ab{1,3}c/;

console.log(reg.test("aaaaasdas5"));

console.log(reg1.test("abbc"));

let reg2=/ab+c/;

console.log(reg2.test("adacabcc"));

let reg3=/ab\*c/;

let reg4=/ab?(cc)/;

console.log(reg3.test("abccc"));

console.log(reg4.test("abc"));

let reg5=/^a/;

console.log(reg5.test("dqasd"));

let reg6=/a$/;

console.log(reg6.test("aaaf"));

let reg7=/^a$/;

console.log(reg7.test("aaa"));

console.log(reg7.test("a"));

let num="13243687254";

let reg8=/^1[3-9][0-9]{9}$/;

console.log(reg8.test(num));

let reg9=/\./;

console.log(reg9.test("1628590697@qq.com"));

let srt=" l bw ";

console.log(srt.replace(/\s/g,""));

console.log(srt.replace(/^\s\*|\s\*$/g,""));

let qq="13243687254@168.com";

let reg10=/^\w+@\w+(\.com)$/;

console.log(reg10.test(qq));

</script>

</body>

</html>

<!--

/\*

\* 量词

\* - 通过量词可以设置一个内容出现的次数

\* - 量词只对它前边的一个内容起作用

\* - {n} 正好出现n次

\* - {m,n} 出现m-n次

\* - {m,} m次以上

\* - + 至少一个，相当于{1,}

\* - \* 0个或多个，相当于{0,}

\* - ? 0个或1个，相当于{0,1}

\*/

-->

<!--

检查一个字符串中是否以a开头

\* ^ 表示开头

\* $ 表示结尾

-->

<!--

\* \w

\* - 任意字母、数字、\_ [A-z0-9\_]

\* \W

\* - 除了字母、数字、\_ [^A-z0-9\_]

\* \d

\* - 任意的数字 [0-9]

\* \D

\* - 除了数字 [^0-9]

\* \s

\* - 空格

\* \S

\* - 除了空格

\* \b

\* - 单词边界

\* \B

\* - 除了单词边界

\*/

-->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>DOM</title>

<script>

/\*window.onload 当网页加载完js才执行\*/

window.onload=function (){

let a=document.getElementById("bt");

console.log(a);

a.innerHTML="提交";

a.onclick=function (){

alert("雅美蝶");

};

}

</script>

</head>

<body>

<button id="bt">我是按钮</button>

</body>

</html>

<!--

/\*

\* 浏览器在加载一个页面时，是按照自上向下的顺序加载的，

\* 读取到一行就运行一行,如果将script标签写到页面的上边，

\* 在代码执行时，页面还没有加载，页面没有加载DOM对象也没有加载

\* 会导致无法获取到DOM对象

\*/

/\*

\* onload事件会在整个页面加载完成之后才触发

\* 为window绑定一个onload事件

\* 该事件对应的响应函数将会在页面加载完成之后执行，

\* 这样可以确保我们的代码执行时所有的DOM对象已经加载完毕了

\*

\*/

-->

<!--

\*

\* innerHTML用于获取元素内部的HTML代码的

\* 对于自结束标签，这个属性没有意义

\*/

//alert(inputs[i].innerHTML);

/\*

\* 如果需要读取元素节点属性，

\* 直接使用 元素.属性名

\* 例子：元素.id 元素.name 元素.value

\* 注意：class属性不能采用这种方式，

\* 读取class属性时需要使用 元素.className

\*/

-->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>DOM练习 图片切换</title>

<style>

div{

width: 200px;

height: 200px;

margin: 50px auto;

text-align: center;

}

img{

height: 200px;

}

button{

background: #ef475d;

}

</style>

<script>

window.onload=function (){

let prev=document.getElementById("prev");

let next=document.getElementById("next");

let arr=["img/1.jpg","img/2.jpg","img/3.jpg","img/4.jpg","img/5.jpg"];

let img=document.getElementsByTagName("img")[0];

let span=document.getElementsByTagName("span")[0];

let v=0;

prev.onclick=function (){

v--;

if(v<0){

img.src=arr[arr.length-1];

v=arr.length-1;

}

else {

img.src=arr[v];

}

span.innerHTML=("第"+(v+1)+"张");

console.log(img.src);

}

next.onclick=function (){

v++;

if(v>arr.length-1){

img.src=arr[0];

v=0;

}

else {

img.src=arr[v];

}

span.innerHTML=("第"+(v+1)+"张");

console.log(img.src);

}

}

</script>

</head>

<body>

<div id="img">

<span>第1张</span>

<img src="img/1.jpg" alt="1">

<button id="prev">上一张</button>

<button id="next">下一张</button>

</div>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>DOM汇总练习</title>

<style>

body {

width: 800px;

margin-left: auto;

margin-right: auto;

}

button {

width: 300px;

margin-bottom: 10px;

}

#btnList {

float:left;

}

#total{

width: 450px;

float:left;

}

ul{

list-style-type: none;

margin: 0px;

padding: 0px;

}

.inner li{

border-style: solid;

border-width: 1px;

padding: 5px;

margin: 5px;

background-color: #99ff99;

float:left;

}

.inner{

width:400px;

border-style: solid;

border-width: 1px;

margin-bottom: 10px;

padding: 10px;

float: left;

}

</style>

</head>

<script type="text/javascript">

/\*

\* 定义一个函数，专门用来为指定元素绑定单击响应函数

\* 参数：

\* idStr 要绑定单击响应函数的对象的id属性值

\* fun 事件的回调函数，当单击元素时，该函数将会被触发

\*/

function myClick(idStr , fun){

var btn = document.getElementById(idStr);

btn.onclick = fun;

}

window.onload = function(){

//为id为btn01的按钮绑定一个单击响应函数

var btn01 = document.getElementById("btn01");

btn01.onclick = function(){

//查找#bj节点

var bj = document.getElementById("bj");

//打印bj

//innerHTML 通过这个属性可以获取到元素内部的html代码

alert(bj.innerHTML);

};

//为id为btn02的按钮绑定一个单击响应函数

var btn02 = document.getElementById("btn02");

btn02.onclick = function(){

//查找所有li节点

//getElementsByTagName()可以根据标签名来获取一组元素节点对象

//这个方法会给我们返回一个类数组对象，所有查询到的元素都会封装到对象中

//即使查询到的元素只有一个，也会封装到数组中返回

var lis = document.getElementsByTagName("li");

//打印lis

//alert(lis.length);

//变量lis

for(var i=0 ; i<lis.length ; i++){

alert(lis[i].innerHTML);

}

};

//为id为btn03的按钮绑定一个单击响应函数

var btn03 = document.getElementById("btn03");

btn03.onclick = function(){

//查找name=gender的所有节点

var inputs = document.getElementsByName("gender");

//alert(inputs.length);

for(var i=0 ; i<inputs.length ; i++){

/\*

\* innerHTML用于获取元素内部的HTML代码的

\* 对于自结束标签，这个属性没有意义

\*/

//alert(inputs[i].innerHTML);

/\*

\* 如果需要读取元素节点属性，

\* 直接使用 元素.属性名

\* 例子：元素.id 元素.name 元素.value

\* 注意：class属性不能采用这种方式，

\* 读取class属性时需要使用 元素.className

\*/

alert(inputs[i].className);

}

};

//为id为btn04的按钮绑定一个单击响应函数

var btn04 = document.getElementById("btn04");

btn04.onclick = function(){

//获取id为city的元素

var city = document.getElementById("city");

//查找#city下所有li节点

var lis = city.getElementsByTagName("li");

for(var i=0 ; i<lis.length ; i++){

alert(lis[i].innerHTML);

}

};

//为id为btn05的按钮绑定一个单击响应函数

var btn05 = document.getElementById("btn05");

btn05.onclick = function(){

//获取id为city的节点

var city = document.getElementById("city");

//返回#city的所有子节点

/\*

\* childNodes属性会获取包括文本节点在呢的所有节点

\* 根据DOM标签标签间空白也会当成文本节点

\* 注意：在IE8及以下的浏览器中，不会将空白文本当成子节点，

\* 所以该属性在IE8中会返回4个子元素而其他浏览器是9个

\*/

var cns = city.childNodes;

//alert(cns.length);

/\*for(var i=0 ; i<cns.length ; i++){

alert(cns[i]);

}\*/

/\*

\* children属性可以获取当前元素的所有子元素

\*/

var cns2 = city.children;

alert(cns2.length);

};

//为id为btn06的按钮绑定一个单击响应函数

var btn06 = document.getElementById("btn06");

btn06.onclick = function(){

//获取id为phone的元素

var phone = document.getElementById("phone");

//返回#phone的第一个子节点

//phone.childNodes[0];

//firstChild可以获取到当前元素的第一个子节点（包括空白文本节点）

var fir = phone.firstChild;

//firstElementChild获取当前元素的第一个子元素

/\*

\* firstElementChild不支持IE8及以下的浏览器，

\* 如果需要兼容他们尽量不要使用

\*/

//fir = phone.firstElementChild;

alert(fir);

};

//为id为btn07的按钮绑定一个单击响应函数

myClick("btn07",function(){

//获取id为bj的节点

var bj = document.getElementById("bj");

//返回#bj的父节点

var pn = bj.parentNode;

alert(pn.innerHTML);

/\*

\* innerText

\* - 该属性可以获取到元素内部的文本内容

\* - 它和innerHTML类似，不同的是它会自动将html去除

\*/

//alert(pn.innerText);

});

//为id为btn08的按钮绑定一个单击响应函数

myClick("btn08",function(){

//获取id为android的元素

var and = document.getElementById("android");

//返回#android的前一个兄弟节点（也可能获取到空白的文本）

var ps = and.previousSibling;

//previousElementSibling获取前一个兄弟元素，IE8及以下不支持

//var pe = and.previousElementSibling;

alert(ps);

});

//读取#username的value属性值

myClick("btn09",function(){

//获取id为username的元素

var um = document.getElementById("username");

//读取um的value属性值

//文本框的value属性值，就是文本框中填写的内容

alert(um.value);

});

//设置#username的value属性值

myClick("btn10",function(){

//获取id为username的元素

var um = document.getElementById("username");

um.value = "今天天气真不错~~~";

});

//返回#bj的文本值

myClick("btn11",function(){

//获取id为bj的元素

var bj = document.getElementById("bj");

//alert(bj.innerHTML);

//alert(bj.innerText);

//获取bj中的文本节点

/\*var fc = bj.firstChild;

alert(fc.nodeValue);\*/

alert(bj.firstChild.nodeValue);

});

};

</script>

</head>

<body>

<div id="total">

<div class="inner">

<p>

你喜欢哪个城市?

</p>

<ul id="city">

<li id="bj">北京</li>

<li>上海</li>

<li>东京</li>

<li>首尔</li>

</ul>

<br>

<br>

<p>

你喜欢哪款单机游戏?

</p>

<ul id="game">

<li id="rl">红警</li>

<li>实况</li>

<li>极品飞车</li>

<li>魔兽</li>

</ul>

<br />

<br />

<p>

你手机的操作系统是?

</p>

<ul id="phone"><li>IOS</li> <li id="android">Android</li><li>Windows Phone</li></ul>

</div>

<div class="inner">

gender:

<input class="hello" type="radio" name="gender" value="male"/>

Male

<input class="hello" type="radio" name="gender" value="female"/>

Female

<br>

<br>

name:

<input type="text" name="name" id="username" value="abcde"/>

</div>

</div>

<div id="btnList">

<div><button id="btn01">查找#bj节点</button></div>

<div><button id="btn02">查找所有li节点</button></div>

<div><button id="btn03">查找name=gender的所有节点</button></div>

<div><button id="btn04">查找#city下所有li节点</button></div>

<div><button id="btn05">返回#city的所有子节点</button></div>

<div><button id="btn06">返回#phone的第一个子节点</button></div>

<div><button id="btn07">返回#bj的父节点</button></div>

<div><button id="btn08">返回#android的前一个兄弟节点</button></div>

<div><button id="btn09">返回#username的value属性值</button></div>

<div><button id="btn10">设置#username的value属性值</button></div>

<div><button id="btn11">返回#bj的文本值</button></div>

</div>

</body>

</html>

<!DOCTYPE html>

<html lang="ch">

<head>

<meta charset="UTF-8">

<title>选择练习</title>

<script>

window.onload=function (){

let checkAll=document.getElementById("checkedAllBtn");

let intrms=document.getElementsByName("items");

let checkNo=document.getElementById("checkedNoBtn");

let checkR=document.getElementById("checkedRevBtn");

let send=document.getElementById("sendBtn");

/\*全选，取消全选\*/

let check=document.getElementById("checkedAllBox");

let spanC=document.getElementById("check");

check.onclick=function (){

if (check.checked) {

for (let i = 0; i < intrms.length; i++) {

intrms[i].checked=true;

spanC.innerText="取消全选";

}

}

else {

for (let i = 0; i < intrms.length; i++) {

intrms[i].checked=false;

spanC.innerText="全选";

}

}

}

/\*上面的函数封装\*/

function che(){

for (let j = 0; j < intrms.length; j++) {

if (!intrms[j].checked){

check.checked=false;

spanC.innerText="全选";

break;

}

else {

check.checked=true;

spanC.innerText="取消全选";

}

}

}

for (let i = 0; i < intrms.length; i++) {

intrms[i].onclick=function (){

for (let j = 0; j < intrms.length; j++) {

if (!intrms[j].checked){

check.checked=false;

spanC.innerText="全选";

break;

}

else {

check.checked=true;

spanC.innerText="取消全选";

}

}

}

}

/\*全选按钮\*/

checkAll.onclick=function (){

for (let i = 0; i < intrms.length; i++) {

intrms[i].checked=true;

}

che();

};

/\*全不选\*/

checkNo.onclick=function (){

for (let i = 0; i < intrms.length; i++) {

intrms[i].checked=false;

}

che();

};

/\*反选\*/

checkR.onclick=function (){

for (let i = 0; i < intrms.length; i++) {

intrms[i].checked = !intrms[i].checked;

}

che();

};

/\*提交，弹出\*/

send.onclick=function (){

for (let i = 0; i < intrms.length; i++) {

if (intrms[i].checked) alert(intrms[i].value);

}

}

}

</script>

</head>

<body>

<form method="post" action="">

你爱好的运动是？<input type="checkbox" id="checkedAllBox" />

<span id="check">全选</span>

<br />

<input type="checkbox" name="items" value="足球" />足球

<input type="checkbox" name="items" value="篮球" />篮球

<input type="checkbox" name="items" value="羽毛球" />羽毛球

<input type="checkbox" name="items" value="乒乓球" />乒乓球

<br />

<input type="button" id="checkedAllBtn" value="全　选" />

<input type="button" id="checkedNoBtn" value="全不选" />

<input type="button" id="checkedRevBtn" value="反　选" />

<input type="button" id="sendBtn" value="提　交" />

</form>

</body>

</html>

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>DOM其他用法</title>

<script type="text/javascript">

window.onload = function(){

//获取body标签

//var body = document.getElementsByTagName("body")[0];

/\*

\* 在document中有一个属性body，它保存的是body的引用

\*/

var body = document.body;

/\*

\* document.documentElement保存的是html根标签

\*/

var html = document.documentElement;

//console.log(html);

/\*

\* document.all代表页面中所有的元素

\*/

var all = document.all;

//console.log(all.length);

/\*for(var i=0 ; i<all.length ; i++){

console.log(all[i]);

}\*/

//all = document.getElementsByTagName("\*");

//console.log(all.length);

/\*

\* 根据元素的class属性值查询一组元素节点对象

\* getElementsByClassName()可以根据class属性值获取一组元素节点对象，

\* 但是该方法不支持IE8及以下的浏览器

\*/

//var box1 = document.getElementsByClassName("box1");

//console.log(box1.length);

//获取页面中的所有的div

//var divs = document.getElementsByTagName("div");

//获取class为box1中的所有的div

//.box1 div

/\*

\* document.querySelector()

\* - 需要一个选择器的字符串作为参数，可以根据一个CSS选择器来查询一个元素节点对象

\* - 虽然IE8中没有getElementsByClassName()但是可以使用querySelector()代替

\* - 使用该方法总会返回唯一的一个元素，如果满足条件的元素有多个，那么它只会返回第一个

\*/

var div = document.querySelector(".box1 div");

var box1 = document.querySelector(".box1");

//console.log(div.innerHTML);

//console.log(box1.innerHTML);

/\*

\* document.querySelectorAll()

\* - 该方法和querySelector()用法类似，不同的是它会将符合条件的元素封装到一个数组中返回

\* - 即使符合条件的元素只有一个，它也会返回数组

\*/

box1 = document.querySelectorAll(".box1");

box1 = document.querySelectorAll("#box2");

console.log(box1);

};

</script>

</head>

<body>

<div id="box2"></div>

<div class="box1">

我是第一个box1

<div>我是box1中的div</div>

</div>

<div class="box1">

<div>我是box1中的div</div>

</div>

<div class="box1">

<div>我是box1中的div</div>

</div>

<div class="box1">

<div>我是box1中的div</div>

</div>

<div></div>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>增删</title>

<script>

window.onload = function() {

//创建一个"广州"节点,添加到#city下

myClick("btn01",function(){

//创建广州节点 <li>广州</li>

//创建li元素节点

/\*

\* document.createElement()

\* 可以用于创建一个元素节点对象，

\* 它需要一个标签名作为参数，将会根据该标签名创建元素节点对象，

\* 并将创建好的对象作为返回值返回

\*/

var li = document.createElement("li");

//创建广州文本节点

/\*

\* document.createTextNode()

\* 可以用来创建一个文本节点对象

\* 需要一个文本内容作为参数，将会根据该内容创建文本节点，并将新的节点返回

\*/

var gzText = document.createTextNode("广州");

//将gzText设置li的子节点

/\*

\* appendChild()

\* - 向一个父节点中添加一个新的子节点

\* - 用法：父节点.appendChild(子节点);

\*/

li.appendChild(gzText);

//获取id为city的节点

var city = document.getElementById("city");

//将广州添加到city下

city.appendChild(li);

});

//将"广州"节点插入到#bj前面

myClick("btn02",function(){

//创建一个广州

var li = document.createElement("li");

var gzText = document.createTextNode("广州");

li.appendChild(gzText);

//获取id为bj的节点

var bj = document.getElementById("bj");

//获取city

var city = document.getElementById("city");

/\*

\* insertBefore()

\* - 可以在指定的子节点前插入新的子节点

\* - 语法：

\* 父节点.insertBefore(新节点,旧节点);

\*/

city.insertBefore(li , bj);

});

//使用"广州"节点替换#bj节点

myClick("btn03",function(){

//创建一个广州

var li = document.createElement("li");

var gzText = document.createTextNode("广州");

li.appendChild(gzText);

//获取id为bj的节点

var bj = document.getElementById("bj");

//获取city

var city = document.getElementById("city");

/\*

\* replaceChild()

\* - 可以使用指定的子节点替换已有的子节点

\* - 语法：父节点.replaceChild(新节点,旧节点);

\*/

city.replaceChild(li , bj);

});

//删除#bj节点

myClick("btn04",function(){

//获取id为bj的节点

var bj = document.getElementById("bj");

//获取city

var city = document.getElementById("city");

/\*

\* removeChild()

\* - 可以删除一个子节点

\* - 语法：父节点.removeChild(子节点);

\*

\* 子节点.parentNode.removeChild(子节点);

\*/

//city.removeChild(bj);

bj.parentNode.removeChild(bj);

});

//读取#city内的HTML代码

myClick("btn05",function(){

//获取city

var city = document.getElementById("city");

alert(city.innerHTML);

});

//设置#bj内的HTML代码

myClick("btn06" , function(){

//获取bj

var bj = document.getElementById("bj");

bj.innerHTML = "昌平";

});

myClick("btn07",function(){

//向city中添加广州

var city = document.getElementById("city");

/\*

\* 使用innerHTML也可以完成DOM的增删改的相关操作

\* 一般我们会两种方式结合使用

\*/

//city.innerHTML += "<li>广州</li>";

//创建一个li

var li = document.createElement("li");

//向li中设置文本

li.innerHTML = "广州";

//将li添加到city中

city.appendChild(li);

});

};

function myClick(idStr, fun) {

var btn = document.getElementById(idStr);

btn.onclick = fun;

}

</script>

<body>

<div id="total">

<div class="inner">

<p>

你喜欢哪个城市?

</p>

<ul id="city">

<li id="bj">北京</li>

<li>上海</li>

<li>东京</li>

<li>首尔</li>

</ul>

</div>

</div>

<div id="btnList">

<div><button id="btn01">创建一个"广州"节点,添加到#city下</button></div>

<div><button id="btn02">将"广州"节点插入到#bj前面</button></div>

<div><button id="btn03">使用"广州"节点替换#bj节点</button></div>

<div><button id="btn04">删除#bj节点</button></div>

<div><button id="btn05">读取#city内的HTML代码</button></div>

<div><button id="btn06">设置#bj内的HTML代码</button></div>

<div><button id="btn07">创建一个"广州"节点,添加到#city下</button></div>

</div>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>全局函数，自定义函数,自定义对象</title>

<script type="text/javascript" src="js/s2.js"></script>

</head>

<body>

</body>

</html>

<!DOCTYPE html>

<html lang="zh">

<head>

<meta charset="UTF-8">

<title>练习增删改</title>

<style>

#total {

width: 450px;

margin-left: auto;

margin-right: auto;

}

ul {

list-style-type: none;

}

li {

border-style: solid;

border-width: 1px;

padding: 5px;

margin: 5px;

background-color: #99ff99;

float: left;

}

.inner {

width: 400px;

border-style: solid;

border-width: 1px;

margin: 10px;

padding: 10px;

float: left;

}

#employeeTable {

border-spacing: 1px;

background-color: black;

margin: 80px auto 10px auto;

}

th,td {

background-color: white;

}

#formDiv {

width: 250px;

border-style: solid;

border-width: 1px;

margin: 50px auto 10px auto;

padding: 10px;

}

#formDiv input {

width: 100%;

}

.word {

width: 40px;

}

.inp {

width: 200px;

}

</style>

<script>

window.onload=function (){

function delA(){

let tr=this.parentNode.parentNode

if (window.confirm("确认"+tr.children[0].innerHTML+"删除?"))

tr.parentNode.removeChild(tr);

//return false;/\*取消超链接的跳转行为,不常用，可能404\*/

}

let aAll=document.getElementsByTagName("a");

for (let i = 0; i < aAll.length; i++) {

aAll[i].onclick=delA;

}

let sub=document.getElementById("addEmpButton");

sub.onclick=function (){

let emp=document.getElementById("empName").value;

let em=document.getElementById("email").value;

let sm=document.getElementById("salary").value;

let trS=document.createElement("tr");

trS.innerHTML="<td>"+emp+"</td>"+

"<td>"+em+"</td>"+

"<td>"+sm+"</td>"+

"<td><a href=\'javascript:;\'>Delete</a></td>";

let trSA=trS.getElementsByTagName("a")[0];

trSA.onclick=delA;

let empl=document.getElementById("employeeTable");

empl.appendChild(trS);

}

}

</script>

</head>

<body>

<table id="employeeTable">

<tr>

<th>Name</th>

<th>Email</th>

<th>Salary</th>

<th>&nbsp;</th>

</tr>

<tr>

<td>Tom</td>

<td>tom@tom.com</td>

<td>5000</td>

<td><a href="javascript:;">Delete</a></td>

</tr>

<tr>

<td>Jerry</td>

<td>jerry@sohu.com</td>

<td>8000</td>

<td><a href="javascript:;">Delete</a></td>

</tr>

<tr>

<td>Bob</td>

<td>bob@tom.com</td>

<td>10000</td>

<td><a href="javascript:;">Delete</a></td>

</tr>

</table>

<div id="formDiv">

<h4>添加新员工</h4>

<table>

<tr>

<td class="word">name: </td>

<td class="inp">

<input type="text" name="empName" id="empName" />

</td>

</tr>

<tr>

<td class="word">email: </td>

<td class="inp">

<input type="text" name="email" id="email" />

</td>

</tr>

<tr>

<td class="word">salary: </td>

<td class="inp">

<input type="text" name="salary" id="salary" />

</td>

</tr>

<tr>

<td colspan="2" align="center">

<button id="addEmpButton" value="abc">

Submit

</button>

</td>

</tr>

</table>

</div>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>DOM 修改样式</title>

<style>

#box1{

width: 200px;

height: 200px;

background: #ef475d;

}

</style>

</head>

<body>

<div id="box1"></div>

<button id="but">sub</button>

<script>

let box1=document.getElementById("box1");

let but=document.getElementById("but");

but.onclick=function (){

box1.style.width="400px";

box1.style.height="400px";

box1.style.background="red";

}

</script>

</body>

</html>

<!--

通过JS修改元素的样式:

语法:元素.style.样式名=样式值

\* 注意：如果CSS的样式名中含有-，

\* 这种名称在JS中是不合法的比如background-color

\* 需要将这种样式名修改为驼峰命名法，

\* 去掉-，然后将-后的字母大写

\*

\* 我们通过style属性设置的样式都是内联样式，

\* 而内联样式有较高的优先级，所以通过JS修改的样式往往会立即显示

\*

\* 但是如果在样式中写了!important，则此时样式会有最高的优先级，

\* 即使通过JS也不能覆盖该样式，此时将会导致JS修改样式失效

\* 所以尽量不要为样式添加!important

/读取box1的样式

/\*

\* 语法：元素.style.样式名

\*

\* 通过style属性设置和读取的都是内联样式

\* 无法读取样式表中的样式

-->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>读取样式表</title>

<style>

#box1{

width: 200px;

height: 200px;

background: #ef475d;

}

</style>

</head>

<body>

<div id="box1"></div>

<script>

let box1=document.getElementById("box1");

box1.style.width="100px";

// alert(getComputedStyle(box1,null).width);

</script>

</body>

</html>

<!--

\* 获取元素的当前显示的样式

\* 语法：元素.currentStyle.样式名

\* 它可以用来读取当前元素正在显示的样式

\* 如果当前元素没有设置该样式，则获取它的默认值

\*

\* currentStyle只有IE浏览器支持，其他的浏览器都不支持

\* 在其他浏览器中可以使用

\* getComputedStyle()这个方法来获取元素当前的样式

\* 这个方法是window的方法，可以直接使用

\* 需要两个参数

\* 第一个：要获取样式的元素

\* 第二个：可以传递一个伪元素，一般都传null

\*

\* 该方法会返回一个对象，对象中封装了当前元素对应的样式

\* 可以通过对象.样式名来读取样式

\* 如果获取的样式没有设置，则会获取到真实的值，而不是默认值

\* 比如：没有设置width，它不会获取到auto，而是一个长度

\*

\* 但是该方法不支持IE8及以下的浏览器

\*

\* 通过currentStyle和getComputedStyle()读取到的样式都是只读的，

\* 不能修改，如果要修改必须通过style属性

-->

<!DOCTYPE html>

<html lang="zh">

<head>

<meta charset="UTF-8">

<title>滚动条</title>

<style type="text/css">

#info{

width: 300px;

height: 500px;

background-color: #bfa;

overflow: auto;

}

</style>

<script type="text/javascript">

window.onload = function(){

/\*

\* 当垂直滚动条滚动到底时使表单项可用

\* onscroll

\* - 该事件会在元素的滚动条滚动时触发

\*/

//获取id为info的p元素

var info = document.getElementById("info");

//获取两个表单项

var inputs = document.getElementsByTagName("input");

//为info绑定一个滚动条滚动的事件

info.onscroll = function(){

//检查垂直滚动条是否滚动到底

if((info.clientHeight + info.scrollTop)==info.scrollHeight)

//判断语句有问题，未解决

{

//滚动条滚动到底，使表单项可用

/\*

\* disabled属性可以设置一个元素是否禁用，

\* 如果设置为true，则元素禁用

\* 如果设置为false，则元素可用

\*/

inputs[0].disabled = false;

inputs[1].disabled = false;

}

};

};

</script>

</head>

<body>

<h3>欢迎亲爱的用户注册</h3>

<p id="info">

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

亲爱的用户，请仔细阅读以下协议，如果你不仔细阅读你就别注册

</p>

<!-- 如果为表单项添加disabled="disabled" 则表单项将变成不可用的状态 -->

<input type="checkbox" disabled="disabled" />我已仔细阅读协议，一定遵守

<input type="submit" value="注册" disabled="disabled" />

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>鼠标位置，获得鼠标键盘信息</title>

<style>

#get{

width: 400px;

height: 400px;

border: 1px solid black;

}

#show{

width: 200px;

height: 50px;

border: 1px solid black;

}

</style>

</head>

<body>

<div id="get"></div>

<div id="show"></div>

<script>

let get=document.getElementById("get");

let show=document.getElementById("show");

get.onmousemove=function (a){

a=a||window.a;//解决事件对象的兼容性问题

let x=a.clientX;

let y=a.clientY;

show.innerHTML="x="+x+",y="+y;

}

</script>

</body>

</html>

<!--

/\*

\* onmousemove

\* - 该事件将会在鼠标在元素中移动时被触发

\*

\* 事件对象

\* - 当事件的响应函数被触发时，浏览器每次都会将一个事件对象作为实参传递进响应函数,

\* 在事件对象中封装了当前事件相关的一切信息，比如：鼠标的坐标 键盘哪个按键被按下 鼠标滚轮滚动的方向。。。

\*/

在IE8中，响应函数被处罚时，浏览器不会传递事件对象，

\* 在IE8及以下的浏览器中，是将事件对象作为window对象的属性保存的

-->

<!DOCTYPE html>

<html lang="zh">

<head>

<meta charset="UTF-8">

<title>div跟随鼠标移动</title>

<style type="text/css">

#box1{

width: 100px;

height: 100px;

background-color: red;

/\*

\* 开启box1的绝对定位

\*/

position: absolute;

}

</style>

<script type="text/javascript">

window.onload = function(){

/\*

\* 使div可以跟随鼠标移动

\*/

//获取box1

var box1 = document.getElementById("box1");

//绑定鼠标移动事件

document.onmousemove = function(event){

//解决兼容问题

event = event || window.event;

//获取滚动条滚动的距离

/\*

\* chrome认为浏览器的滚动条是body的，可以通过body.scrollTop来获取

\* 火狐等浏览器认为浏览器的滚动条是html的，

\*/

var st = document.body.scrollTop || document.documentElement.scrollTop;

var sl = document.body.scrollLeft || document.documentElement.scrollLeft;

//var st = document.documentElement.scrollTop;

//获取到鼠标的坐标

/\*

\* clientX和clientY

\* 用于获取鼠标在当前的可见窗口的坐标

\* div的偏移量，是相对于整个页面的

\*

\* pageX和pageY可以获取鼠标相对于当前页面的坐标

\* 但是这个两个属性在IE8中不支持，所以如果需要兼容IE8，则不要使用

\*/

var left = event.clientX;

var　top = event.clientY;

//设置div的偏移量

box1.style.left = left + sl + "px";

box1.style.top = top + st + "px";

};

};

</script>

</head>

<body style="height: 1000px;width: 2000px;">

<div id="box1"></div>

</body>

</html>

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>冒泡响应,事件委派</title>

<style type="text/css">

#box1{

width: 200px;

height: 200px;

background-color: yellowgreen;

}

#s1{

background-color: yellow;

}

</style>

<script type="text/javascript">

window.onload = function(){

/\*

\* 事件的冒泡（Bubble）

\* - 所谓的冒泡指的就是事件的向上传导，当后代元素上的事件被触发时，其祖先元素的相同事件也会被触发

\* - 在开发中大部分情况冒泡都是有用的,如果不希望发生事件冒泡可以通过事件对象来取消冒泡

\*

\*/

//为s1绑定一个单击响应函数

var s1 = document.getElementById("s1");

s1.onclick = function(event){

event = event || window.event;

alert("我是span的单击响应函数");

//取消冒泡

//可以将事件对象的cancelBubble设置为true，即可取消冒泡

event.cancelBubble = true;

};

//为box1绑定一个单击响应函数

var box1 = document.getElementById("box1");

box1.onclick = function(event){

event = event || window.event;

alert("我是div的单击响应函数");

event.cancelBubble = true;

};

//为body绑定一个单击响应函数

document.body.onclick = function(){

alert("我是body的单击响应函数");

};

};

</script>

</head>

<body>

<div id="box1">

我是box1

<span id="s1">我是span</span>

</div>

</body>

</html>

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>事件的绑定，事件的传播</title>

<script type="text/javascript">

window.onload = function(){

/\*

\* 点击按钮以后弹出一个内容

\*/

//获取按钮对象

var btn01 = document.getElementById("btn01");

/\*

\* 使用 对象.事件 = 函数 的形式绑定响应函数，

\* 它只能同时为一个元素的一个事件绑定一个响应函数，

\* 不能绑定多个，如果绑定了多个，则后边会覆盖掉前边的

\*/

//为btn01绑定一个单击响应函数

/\*btn01.onclick = function(){

alert(1);

};\*/

//为btn01绑定第二个响应函数

/\*btn01.onclick = function(){

alert(2);

};\*/

/\*

\* addEventListener()

\* - 通过这个方法也可以为元素绑定响应函数

\* - 参数：

\* 1.事件的字符串，不要on

\* 2.回调函数，当事件触发时该函数会被调用

\* 3.是否在捕获阶段触发事件，需要一个布尔值，一般都传false

\*

\* 使用addEventListener()可以同时为一个元素的相同事件同时绑定多个响应函数，

\* 这样当事件被触发时，响应函数将会按照函数的绑定顺序执行

\*

\* 这个方法不支持IE8及以下的浏览器

\*/

/\*btn01.addEventListener("click",function(){

alert(1);

},false);

btn01.addEventListener("click",function(){

alert(2);

},false);

btn01.addEventListener("click",function(){

alert(3);

},false);\*/

/\*

\* attachEvent()

\* - 在IE8中可以使用attachEvent()来绑定事件

\* - 参数：

\* 1.事件的字符串，要on

\* 2.回调函数

\*

\* - 这个方法也可以同时为一个事件绑定多个处理函数，

\* 不同的是它是后绑定先执行，执行顺序和addEventListener()相反

\*/

/\*btn01.attachEvent("onclick",function(){

alert(1);

});

btn01.attachEvent("onclick",function(){

alert(2);

});

btn01.attachEvent("onclick",function(){

alert(3);

});\*/

/\*btn01.addEventListener("click",function(){

alert(this);

},false);\*/

/\*btn01.attachEvent("onclick",function(){

alert(this);

});\*/

bind(btn01 , "click" , function(){

alert(this);

});

};

//定义一个函数，用来为指定元素绑定响应函数

/\*

\* addEventListener()中的this，是绑定事件的对象

\* attachEvent()中的this，是window

\* 需要统一两个方法this

\*/

/\*

\* 参数：

\* obj 要绑定事件的对象

\* eventStr 事件的字符串(不要on)

\* callback 回调函数

\*/

function bind(obj , eventStr , callback){

if(obj.addEventListener){

//大部分浏览器兼容的方式

obj.addEventListener(eventStr , callback , false);

}else{

/\*

\* this是谁由调用方式决定

\* callback.call(obj)

\*/

//IE8及以下

obj.attachEvent("on"+eventStr , function(){

//在匿名函数中调用回调函数

callback.call(obj);

});

}

}

</script>

</head>

<body>

<button id="btn01">点我一下</button>

</body>

</html>

<!--

事件的传播

\* - 关于事件的传播网景公司和微软公司有不同的理解

\* - 微软公司认为事件应该是由内向外传播，也就是当事件触发时，应该先触发当前元素上的事件，

\* 然后再向当前元素的祖先元素上传播，也就说事件应该在冒泡阶段执行。

\* - 网景公司认为事件应该是由外向内传播的，也就是当前事件触发时，应该先触发当前元素的最外层的祖先元素的事件，

\* 然后在向内传播给后代元素

\* - W3C综合了两个公司的方案，将事件传播分成了三个阶段

\* 1.捕获阶段

\* - 在捕获阶段时从最外层的祖先元素，向目标元素进行事件的捕获，但是默认此时不会触发事件

\* 2.目标阶段

\* - 事件捕获到目标元素，捕获结束开始在目标元素上触发事件

\* 3.冒泡阶段

\* - 事件从目标元素向他的祖先元素传递，依次触发祖先元素上的事件

\*

\* - 如果希望在捕获阶段就触发事件，可以将addEventListener()的第三个参数设置为true

\* 一般情况下我们不会希望在捕获阶段触发事件，所以这个参数一般都是false

\*

\* - IE8及以下的浏览器中没有捕获阶段

-->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>鼠标拖拽</title>

<style>

#box1{

width: 200px;

height: 200px;

background: #ef475d;

position: absolute;

}

</style>

</head>

<body>

<div id="box1">hello</div>

<script>

let box1=document.getElementById("box1");

box1.onmousedown=function (mo){

let xl=mo.clientX-box1.offsetLeft;

let yl=mo.clientY-box1.offsetTop;//解决鼠标在左上角，原理是减去元素自带的距离

//给document可以让其不管在哪都可以停下

document.onmousemove=function (mo){

box1.style.left=mo.clientX-xl+"px";

box1.style.top=mo.clientY-yl+"px";

}

document.onmouseup=function (){

document.onmousemove=null;

document.onmouseup=null;

}

}

</script>

</body>

</html>

<!--

\* 当我们拖拽一个网页中的内容时，浏览器会默认去搜索引擎中搜索内容，

\* 此时会导致拖拽功能的异常，这个是浏览器提供的默认行为，

\* 如果不希望发生这个行为，则可以通过return false来取消默认行为

\*

\* 但是这招对IE8不起作用

-->

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>滚轮</title>

<style type="text/css">

#box1{

width: 100px;

height: 100px;

background-color: red;

}

</style>

<script type="text/javascript">

window.onload = function(){

//获取id为box1的div

var box1 = document.getElementById("box1");

//为box1绑定一个鼠标滚轮滚动的事件

/\*

\* onmousewheel鼠标滚轮滚动的事件，会在滚轮滚动时触发，

\* 但是火狐不支持该属性

\*

\* 在火狐中需要使用 DOMMouseScroll 来绑定滚动事件

\* 注意该事件需要通过addEventListener()函数来绑定

\*/

box1.onmousewheel = function(event){

event = event || window.event;

//event.wheelDelta 可以获取鼠标滚轮滚动的方向

//向上滚 120 向下滚 -120

//wheelDelta这个值我们不看大小，只看正负

//alert(event.wheelDelta);

//wheelDelta这个属性火狐中不支持

//在火狐中使用event.detail来获取滚动的方向

//向上滚 -3 向下滚 3

//alert(event.detail);

/\*

\* 当鼠标滚轮向下滚动时，box1变长

\* 当滚轮向上滚动时，box1变短

\*/

//判断鼠标滚轮滚动的方向

if(event.wheelDelta > 0 || event.detail < 0){

//向上滚，box1变短

box1.style.height = box1.clientHeight - 10 + "px";

}else{

//向下滚，box1变长

box1.style.height = box1.clientHeight + 10 + "px";

}

/\*

\* 使用addEventListener()方法绑定响应函数，取消默认行为时不能使用return false

\* 需要使用event来取消默认行为event.preventDefault();

\* 但是IE8不支持event.preventDefault();这个玩意，如果直接调用会报错

\*/

event.preventDefault && event.preventDefault();

/\*

\* 当滚轮滚动时，如果浏览器有滚动条，滚动条会随之滚动，

\* 这是浏览器的默认行为，如果不希望发生，则可以取消默认行为

\*/

return false;

};

//为火狐绑定滚轮事件

bind(box1,"DOMMouseScroll",box1.onmousewheel);

};

function bind(obj , eventStr , callback){

if(obj.addEventListener){

//大部分浏览器兼容的方式

obj.addEventListener(eventStr , callback , false);

}else{

/\*

\* this是谁由调用方式决定

\* callback.call(obj)

\*/

//IE8及以下

obj.attachEvent("on"+eventStr , function(){

//在匿名函数中调用回调函数

callback.call(obj);

});

}

}

</script>

</head>

<body style="height: 2000px;">

<div id="box1"></div>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>BOM对象 定时器 location</title>

<script type="text/javascript" src="js/s3.js"></script>

</head>

<body>

</body>

</html>

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>键盘事件</title>

<script type="text/javascript">

window.onload = function(){

/\*

\* 键盘事件：

\* onkeydown

\* - 按键被按下

\* - 对于onkeydown来说如果一直按着某个按键不松手，则事件会一直触发

\* - 当onkeydown连续触发时，第一次和第二次之间会间隔稍微长一点，其他的会非常的快

\* 这种设计是为了防止误操作的发生。

\* onkeyup

\* - 按键被松开

\*

\* 键盘事件一般都会绑定给一些可以获取到焦点的对象或者是document

\*/

document.onkeydown = function(event){

event = event || window.event;

/\*

\* 可以通过keyCode来获取按键的编码

\* 通过它可以判断哪个按键被按下

\* 除了keyCode，事件对象中还提供了几个属性

\* altKey

\* ctrlKey

\* shiftKey

\* - 这个三个用来判断alt ctrl 和 shift是否被按下

\* 如果按下则返回true，否则返回false

\*/

//console.log(event.keyCode);

//判断一个y是否被按下

//判断y和ctrl是否同时被按下

if(event.keyCode === 89 && event.ctrlKey){

console.log("ctrl和y都被按下了");

}

};

/\*document.onkeyup = function(){

console.log("按键松开了");

};\*/

//获取input

var input = document.getElementsByTagName("input")[0];

input.onkeydown = function(event){

event = event || window.event;

//console.log(event.keyCode);

//数字 48 - 57

//使文本框中不能输入数字

if(event.keyCode >= 48 && event.keyCode <= 57){

//在文本框中输入内容，属于onkeydown的默认行为

//如果在onkeydown中取消了默认行为，则输入的内容，不会出现在文本框中

return false;

}

};

};

</script>

</head>

<body>

<input type="text" />

</body>

</html>

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title></title>

<style type="text/css">

#box1{

width: 100px;

height: 100px;

background-color: red;

position: absolute;

}

</style>

<script type="text/javascript">

//使div可以根据不同的方向键向不同的方向移动

/\*

\* 按左键，div向左移

\* 按右键，div向右移

\* 。。。

\*/

window.onload = function(){

//为document绑定一个按键按下的事件

document.onkeydown = function(event){

event = event || window.event;

//定义一个变量，来表示移动的速度

var speed = 10;

//当用户按了ctrl以后，速度加快

if(event.ctrlKey){

speed = 500;

}

/\*

\* 37 左

\* 38 上

\* 39 右

\* 40 下

\*/

switch(event.keyCode){

case 37:

//alert("向左"); left值减小

box1.style.left = box1.offsetLeft - speed + "px";

break;

case 39:

//alert("向右");

box1.style.left = box1.offsetLeft + speed + "px";

break;

case 38:

//alert("向上");

box1.style.top = box1.offsetTop - speed + "px";

break;

case 40:

//alert("向下");

box1.style.top = box1.offsetTop + speed + "px";

break;

}

};

};

</script>

</head>

<body>

<div id="box1"></div>

</body>

</html>

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title>浏览器识别</title>

<script type="text/javascript">

/\*

\* BOM

\* - 浏览器对象模型

\* - BOM可以使我们通过JS来操作浏览器

\* - 在BOM中为我们提供了一组对象，用来完成对浏览器的操作

\* - BOM对象

\* Window

\* - 代表的是整个浏览器的窗口，同时window也是网页中的全局对象

\* Navigator

\* - 代表的当前浏览器的信息，通过该对象可以来识别不同的浏览器

\* Location

\* - 代表当前浏览器的地址栏信息，通过Location可以获取地址栏信息，或者操作浏览器跳转页面

\* History

\* - 代表浏览器的历史记录，可以通过该对象来操作浏览器的历史记录

\* 由于隐私原因，该对象不能获取到具体的历史记录，只能操作浏览器向前或向后翻页

\* 而且该操作只在当次访问时有效

\* Screen

\* - 代表用户的屏幕的信息，通过该对象可以获取到用户的显示器的相关的信息

\*

\*

\* 这些BOM对象在浏览器中都是作为window对象的属性保存的，

\* 可以通过window对象来使用，也可以直接使用

\*

\*

\*/

//console.log(navigator);

//console.log(location);

//console.log(history);

/\*

\* Navigator

\* - 代表的当前浏览器的信息，通过该对象可以来识别不同的浏览器

\* - 由于历史原因，Navigator对象中的大部分属性都已经不能帮助我们识别浏览器了

\* - 一般我们只会使用userAgent来判断浏览器的信息，

\* userAgent是一个字符串，这个字符串中包含有用来描述浏览器信息的内容，

\* 不同的浏览器会有不同的userAgent

\*

\* 火狐的userAgent

\* Mozilla/5.0 (Windows NT 6.1; WOW64; rv:50.0) Gecko/20100101 Firefox/50.0

\*

\* Chrome的userAgent

\* Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.82 Safari/537.36

\*

\* IE8

\* Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)

\*

\* IE9

\* Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)

\*

\* IE10

\* Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)

\*

\* IE11

\* Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E; rv:11.0) like Gecko

\* - 在IE11中已经将微软和IE相关的标识都已经去除了，所以我们基本已经不能通过UserAgent来识别一个浏览器是否是IE了

\*/

//alert(navigator.appName);

var ua = navigator.userAgent;

console.log(ua);

if(/firefox/i.test(ua)){

alert("你是火狐！！！");

}else if(/chrome/i.test(ua)){

alert("你是Chrome");

}else if(/msie/i.test(ua)){

alert("你是IE浏览器~~~");

}else if("ActiveXObject" in window){

alert("你是IE11，枪毙了你~~~");

}

/\*

\* 如果通过UserAgent不能判断，还可以通过一些浏览器中特有的对象，来判断浏览器的信息

\* 比如：ActiveXObject

\*/

/\*if("ActiveXObject" in window){

alert("你是IE，我已经抓住你了~~~");

}else{

alert("你不是IE~~~");

}\*/

/\*alert("ActiveXObject" in window);\*/

</script>

</head>

<body>

</body>

</html>

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>控制页面回退</title>

<script type="text/javascript">

/\*

\* History

\* - 对象可以用来操作浏览器向前或向后翻页

\*/

window.onload = function(){

//获取按钮对象

var btn = document.getElementById("btn");

btn.onclick = function(){

/\*

\* length

\* - 属性，可以获取到当成访问的链接数量

\*/

//alert(history.length);

/\*

\* back()

\* - 可以用来回退到上一个页面，作用和浏览器的回退按钮一样

\*/

//history.back();

/\*

\* forward()

\* - 可以跳转下一个页面，作用和浏览器的前进按钮一样

\*/

//history.forward();

/\*

\* go()

\* - 可以用来跳转到指定的页面

\* - 它需要一个整数作为参数

\* 1:表示向前跳转一个页面 相当于forward()

\* 2:表示向前跳转两个页面

\* -1:表示向后跳转一个页面

\* -2:表示向后跳转两个页面

\*/

history.go(-2);

};

};

</script>

</head>

<body>

<button id="btn">点我一下</button>

<h1>History</h1>

<a href="01.BOM.html">去BOM</a>

</body>

</html>

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>查看当前地址栏</title>

<script type="text/javascript">

/\*

\* Location

\* - 该对象中封装了浏览器的地址栏的信息

\*/

window.onload = function(){

//获取按钮对象

var btn = document.getElementById("btn");

btn.onclick = function(){

//如果直接打印location，则可以获取到地址栏的信息（当前页面的完整路径）

//alert(location);

/\*

\* 如果直接将location属性修改为一个完整的路径，或相对路径

\* 则我们页面会自动跳转到该路径，并且会生成相应的历史记录

\*/

//location = "http://www.baidu.com";

//location = "01.BOM.html";

/\*

\* assign()

\* - 用来跳转到其他的页面，作用和直接修改location一样

\*/

//location.assign("http://www.baidu.com");

/\*

\* reload()

\* - 用于重新加载当前页面，作用和刷新按钮一样

\* - 如果在方法中传递一个true，作为参数，则会强制清空缓存刷新页面

\*/

//location.reload(true);

/\*

\* replace()

\* - 可以使用一个新的页面替换当前页面，调用完毕也会跳转页面

\* 不会生成历史记录，不能使用回退按钮回退

\*/

location.replace("01.BOM.html");

};

};

</script>

</head>

<body>

<button id="btn">点我一下</button>

<h1>Location</h1>

<input type="text" />

<a href="01.BOM.html">去BOM</a>

</body>

</html>

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title></title>

<script type="text/javascript">

window.onload = function(){

/\*

\* 使图片可以自动切换

\*/

//获取img标签

var img1 = document.getElementById("img1");

//创建一个数组来保存图片的路径

var imgArr = ["img/1.jpg","img/2.jpg","img/3.jpg","img/4.jpg","img/5.jpg"];

//创建一个变量，用来保存当前图片的索引

var index = 0;

//定义一个变量，用来保存定时器的标识

var timer;

//为btn01绑定一个单击响应函数

var btn01 = document.getElementById("btn01");

btn01.onclick = function(){

/\*

\* 目前，我们每点击一次按钮，就会开启一个定时器，

\* 点击多次就会开启多个定时器，这就导致图片的切换速度过快，

\* 并且我们只能关闭最后一次开启的定时器

\*/

//在开启定时器之前，需要将当前元素上的其他定时器关闭

clearInterval(timer);

/\*

\* 开启一个定时器，来自动切换图片

\*/

timer = setInterval(function(){

//使索引自增

index++;

//判断索引是否超过最大索引

/\*if(index >= imgArr.length){

//则将index设置为0

index = 0;

}\*/

index %= imgArr.length;

//修改img1的src属性

img1.src = imgArr[index];

},1000);

};

//为btn02绑定一个单击响应函数

var btn02 = document.getElementById("btn02");

btn02.onclick = function(){

//点击按钮以后，停止图片的自动切换，关闭定时器

/\*

\* clearInterval()可以接收任意参数，

\* 如果参数是一个有效的定时器的标识，则停止对应的定时器

\* 如果参数不是一个有效的标识，则什么也不做

\*/

clearInterval(timer);

};

};

</script>

<style>

img{

display: block;

margin: 10px auto;

width: 600px;

height: 400px;

object-fit: cover;

}

</style>

</head>

<body>

<img id="img1" src="img/1.jpg"/>

<br /><br />

<button id="btn01">开始</button>

<button id="btn02">停止</button>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>轮播图</title>

<link rel="stylesheet" href="css/reset.css">

<style>

.wi{

width: 400px;

height: 400px;

margin: 100px 200px;

/\*overflow: hidden;\*/

border: 2px red solid;

position: absolute;

}

ul{

position: relative;

float: left;

height: 100%;

left: 0;

}

li{

display: inline-block;

width: 400px;

height: 400px;

}

img{

width: 100%;

height: 100%;

/\*object-fit: cover;\*/

}

a{

float: left;

position: relative;

margin-left: 5px;

background: #ef475d;

width: 15px;

height: 15px;

left: 110px;

top: -40px;

border-radius: 50%;

z-index: 1;

}

a:hover{

background: #999999;

}

</style>

<script>

window.onload=function (){

let slid=document.getElementById("slid");

let slidImg=slid.getElementsByTagName("img");

slid.style.width=slidImg.length\*410+"px";

let wi=document.querySelector(".wi");

let wiA=wi.getElementsByTagName("a");

let wiAPick=0;

wiA[wiAPick].style.background="#999999";

}

</script>

</head>

<body>

<div class="wi">

<ul id="slid">

<li><img src="img/a/1.jpg" alt=""></li>

<li><img src="img/a/2.jpg" alt=""></li>

<li><img src="img/a/3.jpg" alt=""></li>

<li><img src="img/a/4.jpg" alt=""></li>

<li><img src="img/a/5.jpg" alt=""></li>

<li><img src="img/a/6.jpg" alt=""></li>

<li><img src="img/a/7.jpg" alt=""></li>

<li><img src="img/a/8.jpg" alt=""></li>

</ul>

<a href="javascript:"></a>

<a href="javascript:"></a>

<a href="javascript:"></a>

<a href="javascript:"></a>

<a href="javascript:"></a>

<a href="javascript:"></a>

<a href="javascript:"></a>

<a href="javascript:"></a>

</div>

</body>

</html>

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>修改class操作</title>

<style type="text/css">

.b1{

width: 100px;

height: 100px;

background-color: red;

}

.b2{

height: 300px;

background-color: yellow;

}

</style>

<script type="text/javascript">

window.onload = function(){

//获取box

var box = document.getElementById("box");

//获取btn01

var btn01 = document.getElementById("btn01");

//为btn01绑定单击响应函数

btn01.onclick = function(){

//修改box的样式

/\*

\* 通过style属性来修改元素的样式，每修改一个样式，浏览器就需要重新渲染一次页面

\* 这样的执行的性能是比较差的，而且这种形式当我们要修改多个样式时，也不太方便

\*/

/\*box.style.width = "200px";

box.style.height = "200px";

box.style.backgroundColor = "yellow";\*/

/\*

\* 我希望一行代码，可以同时修改多个样式

\*/

//修改box的class属性

/\*

\* 我们可以通过修改元素的class属性来间接的修改样式

\* 这样一来，我们只需要修改一次，即可同时修改多个样式，

\* 浏览器只需要重新渲染页面一次，性能比较好，

\* 并且这种方式，可以使表现和行为进一步的分离

\*/

//box.className += " b2";

//addClass(box,"b2");

//alert(hasClass(box,"hello"));

//removeClass(box,"b2");

toggleClass(box,"b2");

};

};

//定义一个函数，用来向一个元素中添加指定的class属性值

/\*

\* 参数:

\* obj 要添加class属性的元素

\* cn 要添加的class值

\*

\*/

function addClass(obj , cn){

//检查obj中是否含有cn

if(!hasClass(obj , cn)){

obj.className += " "+cn;

}

}

/\*

\* 判断一个元素中是否含有指定的class属性值

\* 如果有该class，则返回true，没有则返回false

\*

\*/

function hasClass(obj , cn){

//判断obj中有没有cn class

//创建一个正则表达式

//var reg = /\bb2\b/;

var reg = new RegExp("\\b"+cn+"\\b");

return reg.test(obj.className);

}

/\*

\* 删除一个元素中的指定的class属性

\*/

function removeClass(obj , cn){

//创建一个正则表达式

var reg = new RegExp("\\b"+cn+"\\b");

//删除class

obj.className = obj.className.replace(reg , "");

}

/\*

\* toggleClass可以用来切换一个类

\* 如果元素中具有该类，则删除

\* 如果元素中没有该类，则添加

\*/

function toggleClass(obj , cn){

//判断obj中是否含有cn

if(hasClass(obj , cn)){

//有，则删除

removeClass(obj , cn);

}else{

//没有，则添加

addClass(obj , cn);

}

}

</script>

</head>

<body>

<button id="btn01">点击按钮以后修改box的样式</button>

<br /><br />

<div id="box" class="b1 b2"></div>

</body>

</html>

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>json</title>

<!--

如果需要兼容IE7及以下的JSON操作，则可以通过引入一个外部的js文件来处理

-->

<script type="text/javascript">

/\*

\* JSON

\* - JS中的对象只有JS自己认识，其他的语言都不认识

\* - JSON就是一个特殊格式的字符串，这个字符串可以被任意的语言所识别，

\* 并且可以转换为任意语言中的对象，JSON在开发中主要用来数据的交互

\* - JSON

\* - JavaScript Object Notation JS对象表示法

\* - JSON和JS对象的格式一样，只不过JSON字符串中的属性名必须加双引号

\* 其他的和JS语法一致

\* JSON分类：

\* 1.对象 {}

\* 2.数组 []

\*

\* JSON中允许的值：

\* 1.字符串

\* 2.数值

\* 3.布尔值

\* 4.null

\* 5.对象

\* 6.数组

\*/

//创建一个对象

var arr = '[1,2,3,"hello",true]';

var obj2 = '{"arr":[1,2,3]}';

var arr2 ='[{"name":"孙悟空","age":18,"gender":"男"},{"name":"孙悟空","age":18,"gender":"男"}]';

/\*

\* 将JSON字符串转换为JS中的对象

\* 在JS中，为我们提供了一个工具类，就叫JSON

\* 这个对象可以帮助我们将一个JSON转换为JS对象，也可以将一个JS对象转换为JSON

\*/

var json = '{"name":"孙悟空","age":18,"gender":"男"}';

/\*

\* json --> js对象

\* JSON.parse()

\* - 可以将以JSON字符串转换为js对象

\* - 它需要一个JSON字符串作为参数，会将该字符串转换为JS对象并返回

\*/

var o = JSON.parse(json);

var o2 = JSON.parse(arr);

console.log(o.name);

//console.log(o.gender);

//console.log(o2[1]);

var obj3 = {name:"猪八戒" , age:28 , gender:"男"};

/\*

\* JS对象 ---> JSON

\* JSON.stringify()

\* - 可以将一个JS对象转换

\* 为JSON字符串

\* - 需要一个js对象作为参数，会返回一个JSON字符串

\*/

var str = JSON.stringify(obj3);

//console.log(str);

/\*

\* JSON这个对象在IE7及以下的浏览器中不支持，所以在这些浏览器中调用时会报错

\*/

var str3 = '{"name":"孙悟空","age":18,"gender":"男"}';

JSON.parse(str3);

let obj4={name:"lbw",age:"8"};

let str4=JSON.stringify(obj4);

console.log(str4);

/\*

\* eval()

\* - 这个函数可以用来执行一段字符串形式的JS代码，并将执行结果返回

\* - 如果使用eval()执行的字符串中含有{},它会将{}当成是代码块

\* 如果不希望将其当成代码块解析，则需要在字符串前后各加一个()

\*

\* - eval()这个函数的功能很强大，可以直接执行一个字符串中的js代码，

\* 但是在开发中尽量不要使用，首先它的执行性能比较差，然后它还具有安全隐患

\*/

</script>

</head>

<body>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>hello包含</title>

</head>

<body>

<!--可以将js代码编写到标签的onclick属性中

当我们点击按钮时，js代码才会执行

虽然可以写在标签的属性中，但是他们属于结构与行为耦合，不方便维护，不推荐使用

-->

<button onclick="alert('你好')">hi</button>

<a href="javascript:alert('hhhh')">ohhh</a>

<!--可以将js代码写在超链接的href属性中，这样当点击超链接时，会执行js代码-->

<a href="javascript:">ohhh</a>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>输入 输出</title>

<script type="text/javascript">

var sc=prompt("zzp的成绩");

if (sc>=60){

alert("zzp天下无敌");

}

else alert("zzp is sb");

</script>

</head>

<body>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Title</title>

<script>

</script>

</head>

<body>

<input type="text" id="t1">

<input type="checkbox" name="qwq" checked="wq">

<input type="checkbox" name="qwq" checked="wqw">

<input type="checkbox" name="qwq" checked="qwq">

<ul>

<li>q</li>

<li>w</li>

<li>e</li>

</ul>

<div class="navi">qwq</div>

<div class="navi">qwq</div>

<script>

/\*var t1=document.getElementById("t1");

alert(t1)\*/

/\*let arr=document.getElementsByName("qwq");

alert(arr.length);\*/

/\*let a=document.getElementsByTagName("li");

alert(a.length);\*/

let b=document.getElementsByClassName("navi");

alert(b.length);

</script>

</body>

</html>

<!--

DOM对象的相关方法: document

getElementByld(); - 通过元素ID获取对应元素对象。

getElementsByName(); -- 通过元素的name属性获取符合要求的所有元素。

getElementsByTagName();.--通过元素的元素名属性获取符合要求的所有元素,

getElementsByClassName(); - 通过元素的class属性获取符合要求的所有元素。

-->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Title</title>

<script src="js/s4.js"></script>

</head>

<body>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Title</title>

<script>

var obj={

name:"zzp",

age:20,

get:function (){

document.writeln(this.name,this.age);

},

}

document.writeln(obj.name,obj.age);

function a(name,age){

var obj1={};

obj1.name=name;

obj1.age=age;

obj1.get=function (){

document.writeln(this.name,this.age);

};

return obj1;

}

var b=a("szr",20);

b.get();

document.writeln(b.name)

</script>

</head>

<body>

</body>

</html>

<!--

便捷方法，把函数作为对象调用，新建对象

-->