JavaScript

主讲：崔译

一、简介

- JavaScript 简称 JS
- 是一种运行在浏览器中的 客户端 脚本语言
- 可以编写在HTML文件中或者`.js`文件中
- 由浏览器解释和运行
- 和`Java`没有任何关系
- 作用
  - 数据校验（用户名不能为空，密码至少6位....）
  - 获取远端（服务器中的）数据，动态的（异步的）生成网页
  - 操作DOM对象

二、HelloWorld

1、在HTML文件中编写

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <!-- JS代码编写在script标签中，script标签一般位于head中 -->
    <!-- <script type="text/javascript"></script> -->
    <script>
        alert("HelloWorld");
    </script>
</head>
<body>
</body>
</html>
```

2、在`js`文件中编写

2-1 编写js文件

```
alert("HelloWorld-2");
```

2-2 在HTML中引入js文件

```
<!-- 使用src属性指定js文件路径 -->
<script src="../js/02.js"></script>
```

2-3 注意点

如果`script`标签具有src属性，那么`<script></script>`正反标签中间 **不能写js代码**

```
<script src="../js/02.js">
  // JS 中的注释
  // 此处代码无效
  // src的值是一个网络地址
  //   理解上来看的
  // script会将地址返回的内容作为script中的内容
  // 原有内容被覆盖
  alert(1);
</script>
```

三、JS的数据类型

- JS 是弱类型语言
- JS 的变量 在 运行时 确定类型

  ```
  // java 代码
  String s = "aaa";
  ```

- JS使用`var`或者 `let` 定义变量
- 数据类型
  - 基本数据类型
    - number 所有的数值（包括整数和小数）
    - string 字符串
    - boolean 布尔
    - null
    - undefined
  - 复合数据类型
    - Array 数组
    - Date 日期
    - Function 函数（方法）(**js中可以将函数赋值给变量**)
    - Object 对象

```
var s = "abc";
console.log(s);
var b = false;
console.log(b)
s = 123;
console.log(s);
// 可以使用typeof方法查看变量的数据类型
```

四、JS中的运算符

1、基本运算符

> +、-、*、/
>
> +=、-=、*=、/=
>
> ++、--
>
> %
>
> && || & |
>
> 三目运算符

2、JS特有运算符

2-1、`===`

```
var a  = 99;
var b = "99";
// 比较字面值
console.log(a == b);
// 比较值和类型
console.log(a === b);
```

2-2、`**`

次方运算`a**b`表示`Math.pow(a,b)`，即：a 的 b 次方

3、注意点

JS 会选择合适的类型进行运算

```
var a = "2";
var b = 2;

console.log("a==b的结果是"+(a==b));
console.log(a + b);
console.log(a - b);
// 减法运算结果一定是一个数字
// 下面表达式的结果一定不是具体数字
```

```
console.log("a" -  2);//NaN  not a number
//NaN  not a number 是一个特殊的数值
//NaN 的特性 和 mysql 中的null 一致
console.log(typeof(NaN));
console.log(NaN == NaN)//false
console.log(isNaN(NaN));
```

五、流程控制语句

```
var a = 20;

if(a>20){
}else if (a > 30){
}else{
}

for(var i = 0 ; i < a ;i++){}

while(a < 40){
  a++;
}

switch (a) {
  case 2:
    break;
  default:
    break;
}
```

六、JS中的函数

1、特点

- 使用关键字`function`定义函数
- 没有返回值类型（**有返回值**）
- 没有参数类型
- 使用`return`关键字返回函数执行结果，**JS的函数永远有返回值（没有return返回undefined）**
- JS 的函数 **是对象，可以赋值给一个变量**

2、定义方式

2-1 方式1

```
function f1(a,b)
{
// return a + b;
}

var result = f1(2,54);
console.log(result);
```

2-2 方式2

```
// 匿名函数
var f2 = function(a,b){
  return a + b;
}
console.log(f2(2,3));
//var f4 = f2; f4是方法
//var f5 = f2();f5是方法返回值

// var f3 = 3;
// f3();//f3 is not a function 只要f3是一个function，就可以使用()调用
```

2-3 方式3

```
// lambda 表达式 是匿名函数的一种写法
// 在jdk9+ 也存在该写法

// 1. 小括号中的是参数，没有参数也要写（）
// 2. => 后的是方法体，
//      2-1 如果方法体只有一行，并且是return语句 ，那么{}和return可以省略
//      2-2 否则{} 和return 不能省略
var f3 = (a,b) => a + b;
```

3、函数参数的意义

- js中的函数的参数列表 是为调用函数时 传入的参数 **起别名**
- 调用JS函数时，传入的参数个数 可以 和 声明函数时定义的参数个数 **不一致**
- js中 **不存在**方法的重载，后定义的覆盖先定义的（基于方式1定义的函数）

```
let f1 = (a,b) => {
  console.log(a,b);
}

f1(1,2,3);
f1(1);

function f(a,b){
  console.log(1)
}
```

```
function f(a,b,c)
{
  console.log(2);
}

f(1,2);
```

4、参数的默认值

```
// 如果调用方法时，传入的参数个数小于定义方法时的参数个数
// 那么多出的参数值为undefined
// 为undefined的参数赋默认值
function f(a=1,b=2,c=3)
{
    console.log(a,b,c);
}

f(10);// 10 2 3

function f1(a = 9,b = (a)=>a**2)
{
  var s = b(a);
  console.log(s);
}

f1(3,(a)=>a+1);//  4
f1(3);//  9
f1();//   81
```

七、`let` or `var`

1、`let` 特点

let 关键字的行为和特点 与java中定义的变量一致，定义的是顶层对象和方法

2、`var`特点

- 允许重复定义变量
- 定义的是window对象的属性和方法
- 存在 变量提升

> 变量提升指的是：将变量的声明提升到当前作用域的第一行
>
> var a = 3;
>
> var a ;
>
> a = 3;

```

//1
```

```
// for(var i = 0;i<10;i++)
// {

// }
// alert(i);//10

//2
// var i = 20;
// function test()
// {
//  alert(i);
//  var i = 30;
// }

// var i = 20;
// function test()
// {
//  var i;
//  alert(i);
//  i = 30;
// }
// test();

// 3
// var i = 20;
// function test()
// {
//  alert(i);
//  i = 30;
//  alert(i);
// }
// test();

//4
// var arr = [];
// for(var i = 0; i < 10 ; i++)
// {
//  arr[i] = function(){
//      alert(i);
//  }
// }
// var f = arr[4];
// f();

// var a  = 20;

// // 立即执行函数
// var f = function(b){
//  return b;
// }(a);

// a = 30;
```

```
// console.log(f);

var arr = [];
for(var i = 0; i < 10 ; i++)
{
  // JS的闭包
  arr[i] = function(x){
    return function(){
      alert(x);
    }
  }(i);
}
var f = arr[4];
f();
```

八、常用类型

1、null 和 undefined

```
let a;// undefined 值未定义
let b = null;// b 有值 值为null

// console.log(a,b)

// c is not defined 变量未定义 NullPointerException
// console.log(c);

console.log( a == b);//true


let c = "";//空字符串
let d = 0;// 数值0
let e = false; // 布尔类型false

console.log(c == d);
console.log(c == e);
console.log(d == e);

// 在JavaScript中, 空字符串、0、false、null、undefined 都表示"false"
if(a || b || c || d || e){
  alert(1);
}else{
  alert(2);
}
// 在JavaScript中, 除了上面五种情况，其他任意的变量 都表示"true"
let f = 2;
// 在js的if中，可以放任意类型变量或者表达式
if(f){

  alert(1111);
```

```
}
```

2、数值类型

2-1 定义方式

```
// 十进制
let a = 123;
let b = 22.2;

// 十六进制
let c = 0x12;
console.log(c);

//八进制
let d = 013;
console.log(d);

// 二进制
let e = 0b1010;
console.log(e);

// IE6 --》 0
// Chrome.... --> 80
let f = 080;
console.log(f);
```

2-2 和字符串的转换

```
// number --> string
let numStr = num + "";

// string ---> number
let str = "234";

let n1 = str - 0;
console.log(n1,typeof(n1));

let n2 = parseInt(str);
console.log(n2,typeof(n2));

let n3 = parseFloat(str);
console.log(n3,typeof(n3));

let n4 = Number.parseInt(str);

let n5 = Number.parseFloat(str);

// 非常规
let s = "123.4";
let s1 = parseInt(s);
```

```
console.log(s1);//123

s = "123abc";
s1 = parseInt(s);
console.log(s1);

s = "d123abc";
s1 = parseInt(s);//NaN
console.log(s1);
```

3、字符串类型

3-1、定义方式

```
let name = "老王";

let address = '隔壁';

let introduce = `
  这是模板字符串，
  该字符串可以换行，
  还可以获取变量值，
  比如：简介：${address}${name},
  ${3+4},${'aa'+'bb'}
`;

console.log(introduce);
```

3-2、常用方法

> charAt / substring / indexOf / lastIndexOf / toLowerCase / toUpperCase /
>
> replace / split / trim / startsWith / endsWith / length(属性)

JS特有方法

- repeat(n) 将字符串重复n次返回
- includes("") 判断一个字符串是否包含另一个字符串
- padStart(n,a) 在原字符串前 补充a 至 n 位
- padEnd(n,a) 在原字符串后 补充a 至n位

3-3 正则表达式

1. 定义方式

```
let regExp = /正则表达式/模式;
let regExp = new RegExp("表达式","模式");
```

2. 相关方法
    1. replace
    2. split

3. search(正则表达式) 检索与正则表达式匹配的值，返回第一个匹配的下标
4. match（正则表达式）找出匹配的子串，返回数组

3. 模式
    1. `i` 忽略大小写
    2. `g` 进行全局搜索

4. 表达式语法

```
// 能够写出简单正则
\d \w \D  \W {m} {m,n} {m,} * + ?  [a-z]  ^ $
// 其他正则 认识
```

4、日期类型

```
// 定义方式
let d1 = new Date(); //  当前系统时间
let d2 = new Date("2012");
let d3 = new Date(0);// 传入毫秒数

// 常用方法
let d = new Date();
// 距离1970-1-1毫秒数
let itme = d.getTime();
let year = d.getFullYear();
let mon = d.getMonth()+1;//注意月份0到11 要+1
let date = d.getDate();
let hour = d.getHours();
let min = d.getMinutes();
let sec = d.getSeconds();
// 毫秒， 不足一秒的时间
let minSec = d.getMilliseconds();
// 星期几
let day = d.getDay();

// 每个get 对应一个set
console.log(`
当前时间是${year}-${mon}-${date} ${hour}:${min}:${sec}.${minSec}
今天是星期${day}
`);


// 常见考题
let d = new Date("2012","4","31");// 5-31
console.log(d.getMonth());// -1 --> 4

d = new Date("2012","12","32");// 2013-1-32 --> 2013-2-1
console.log(d.toLocaleString());

d = new Date();
d.setFullYear(2012,1,28);
d.setDate(31);
```

```
console.log(d.getMonth());
console.log(d.getDate());

// string 和 Date 转换
//Date ---> String
let str = new Date().toLocaleString();
console.log(str);
// string ---> date
str = "2012/12/12 22:55:23";
// 将字符串转成 毫秒数
let time = Date.parse(str);
let d = new Date(time);
console.log(d);
```

5、数组类型

1、定义方式

```
// 定义一个空数组
let arr1 = [];
let arr2 = new Array();

// 定义一个有元素的数组
// js数组中的元素类型可以不一致
let arr3 = [1,2,"abc",false,0.5,new Date(),()=>{}];
let arr4 = new Array(1,2,"abc",false,0.5,new Date(),()=>{});
let arr5 = new Array(9,2);

// 定义一个长度为9的数组
let arr5 = new Array(9);

// 二维数组
let arr6 = [
  [0,2,3],
  [4,3,8],
  [43,56,0]
];
```

2、数组元素的访问方式

```
let arr = [1,2,"abc",false,0.5,new Date(),()=>{}];
//  通过下标访问 从0开始
let item2 = arr[2];
console.log(item2);

arr[2] = "deddd";
console.log(arr);

// JS数组为动态数组，不存在下标越界

let arr1 = [1,2];
```

```
console.log(arr1[4]);//undefined

let arr2 = new Array(1,2);
arr2[9] = 23;
console.log(arr2.length,arr2,arr2[4]);


// 遍历
let arr4 = [5,4,3,2,1];


console.log('方式 1');
for(let i = 0 ; i < arr4.length ; i++)
{
  console.log(i,arr4[i]);
}


console.log('方式2');
for(let i in arr4)
{
  console.log(i,arr4[i]);
}

console.log('方式3');
for(let item of arr4)
{
  console.log(item);
}
```

3、常用方法

3-1 添加

```
// 向数组末尾添加元素
arr.push(99);
arr.push(1,2,3);
console.log(arr);

// 向数组头部添加元素
arr.unshift('x');
arr.unshift('y','z');
console.log(arr);
```

3-2 删除

```
// 删除并返回末尾的元素
let item = arr.pop();
console.log(arr);
console.log(item);

// 删除并返回头部元素
item = arr.shift();
console.log(arr);
console.log(item);
```

3-3 删除并添加

```
// 从第几个开始  删除几个   后面的items 是添加的元素
// 返回删除的数组
//arr.splice(start: int, deleteCount: int, items...: any)
let deleteArr = arr.splice(2, 6, 'x','y','z');
console.log(arr);
console.log(deleteArr);

// 向某个位置index后插入一个元素item
arr.splice(index, 0, item);
```

3-4 拼接

```
// 将数组转换成字符串,默认以逗号拼接
let str = arr.join();
console.log(str);
// 使用参数修改拼接符
str = arr.join("$");
console.log(str);
```

3-5 切片

```
// 从一个数组中截取子数组，substring [start,end)
// arr.slice(start: int, end: int)
let newArr = arr.slice(2, 6);
console.log(newArr.length,newArr,arr);
```

3-6 反转数组

```
console.log(arr);
arr.reverse();
console.log(arr);
// 将HelloWorld变成dlroWolleH输出
let str = "HelloWorld";
let s = str.split('').reverse().join('');
console.log(s);
```

3-7 排序

冒泡排序

```
let someArr = [4,1,5,2,8,12];
// 使用自然顺序  对于数值，按位比较
// someArr.sort();
// 对于数值类型数组
someArr.sort( (a,b)=> b - a );
console.log(someArr);
```

6、对象类型(重点)

1、基本规则

- 对象类型（可以理解为java中的引用类型）包括符合数据类型（日期，数组...）和自定义数据类型（可以理解为java中的面向对象）
- 对象类型的 数据类型 都是 `object`

  ```
  console.log(typeof(new Date()) , new Date().constructor.name );
  console.log(typeof([]) , [].constructor.name);
  ```

- 类的方法可以进行扩展（自己为类添加成员方法）

  ```
  // 为Array类 添加一个成员方法，方法名叫做some，
  Array.prototype.some = function(){
    //this代表 当前类对象，方法调用者
    console.log(this)
  }
  var arr = [1,2,3];
  arr.some();
  ```

- 对象的属性可以进行自由扩展

  ```
  var date = new Date();

  // 为对象扩展属性
  date.itany = "网博优壹";
  console.log(date.itany);

  var newDate = new Date();
  console.log(newDate.itany);
  ```

- 可以使用`for..in`遍历任意一个对象

```
for(let i in window)
{
  console.log(i,newDate[i]);
}
```

2、创建对象

2-1 使用`Object`创建对象

```
function createUser(name,age)
{
  let user = new Object();
  user.name = name;
  user.age = age;
  user.sayHello = function(){
    console.log(`Hello,I am ${user.name},I am ${user.age} years old`);
  }
  return user;
}

var u1 = createUser("张三",35);
u1.sayHello();
```

2-2 使用构造方法创建对象

```
function User(name,age)
{
  this.name = name;
  this.age = age;
  this.sayHello = function(){
    console.log(`Hello,I am ${this.name},I am ${this.age} years old`);
  }
}
var u1 = new User("张4",44);
u1.sayHello();
```

2-3 创建JSON对象(重点)

```
// JSON对象，是一种表示对象的特殊语法
    //  1、一对大括号
    //  2、大括号中定义对象的属性和函数
    //  3、属性名和属性值，函数名和函数 之间  使用 冒号 分隔
    //  4、属性和属性、属性和函数、函数和函数之间使用 逗号分隔
    //  5、一般情况下，一个属性或者一个函数写在一行
    //  6、属性名可以不加双引号，但是要知道，加了双引号的叫做 标准JSON

// let user = {}; 空对象
// let user = {
//  "name":"张三",
//  "age":22,
```

```
//  "sayHello":function(){
//      console.log("Hello")
//  }
// };


// 定义方式1
let user = {
  name:"张三",
  age:22,
  sayHello:function(){
    console.log("Hello")
  }
};

console.log(user.name);
console.log(user.age);
user.sayHello();


// 方式2
let username = "老何";
let age = 30;
let color = "black";
let someUser = {
  "name":username,
  "age":age,
  "skinColor":color
}
console.log(someUser);

// 方式3   对象的简洁表示法
let name = "老何";
let addr = "SGY";
let skinColor = "red";

// 属性名 在 字面上 和 属性值对应的变量名一致
// let otherUser = {
//  "name":name,
//  "addr":addr,
//  skinColor:skinColor
// }

// 在这种情况下，属性名和冒号可以省略不写
let otherUser = {name,addr,skinColor}
console.log(otherUser);
```

3、对象属性的访问

```
let user = {
  name : 'abc',
  age :22,
  addr:'asd',
  f:function(data){
    console.log(data);
  }
}
```

3-1 使用`对象.属性名`访问

```
user.name
```

3-2 使用`对象[属性名]`访问

```
console.log(user['name']);
let age = "addr";
console.log(user[age],user['age'],user.age);
```

3-3 属性的遍历

```
for(let property in user)
{
  console.log(property,user[property]);
}
```

4、对象的解构赋值

分解对象结构，进行赋值操作

- 等号左右类型要一致
- 将等号右边的值 赋值到 等号左边的 "对应" 位置

4-1 数组结构赋值

```
// 数组结构赋值
let arr = [1,2,3];
// 数组使用下标进行对应
let [a,b,c] = arr;
console.log(a,b,c);
// 案例
let x = 22;
let y = 33;
[y,x] = [x,y]
console.log('x',x);
console.log('y',y);
```

4-2 对象结构赋值

```
// 对象结构赋值
let user = {
  "name":"abc",
  "age":22
}
// 对象使用属性名进行对应
let {age,name} = user;
console.log('age',age);
console.log('name',name);
//案例
let people = {
  name:'zs',
  age:22,
  addr:{
    province:'JS',
    city:'NJ',
    address:{
      street:'LPZL',
      no : 458
    }
  }
}
// 请解构赋值取出 age city 和 no
let {
  age,
  addr:{
    city,
    address:{
      no
    }
  }
} = people;
console.log(age,city,no);
```

4-3 字符串的结构赋值

```
let str = "gjf";
let [a,b,c] = str;
console.log(c,b,a);
console.log(str[2]);
```

4-4 JSON属性名和参数名不相同

```
let user = {
  name:"abc",
  pwd :"123"
}

let {name:username,pwd:password} = user;
console.log(username,password);
```

7、综合案例

```
/*
            1、用户输入3个人的信息
                姓名,语文成绩，数学成绩  格式如下：张三,80,93
            2、将每条用户信息变成一个JSON对象(name,chinese,math)
            3、将对象放到一个数组中（先输入的用户在数组开头，末尾追加）
            4、按照语文成绩从小到大排序  输出在控制台
            5、按照平均成绩从大到小排序  输出在控制台
        */
let users = [];
for(let i = 0 ; i<3;i++)
{
  let str = prompt("请输入");
  let [name,chinese,math] = str.split(",");
  users.push({name,chinese,math});
}

// users.sort( (a,b) => a.chinese - b.chinese );
// console.log(users);


users.sort( (a,b) => {
  var avg_a = ( parseInt(a.chinese) + parseInt(a.math) ) / 2;
  var avg_b = ( parseInt(b.chinese) + parseInt(b.math) ) / 2;
  return avg_b - avg_a;
} );
console.log(users);

// a,80,60
// b,90,56
// c,70,72
```

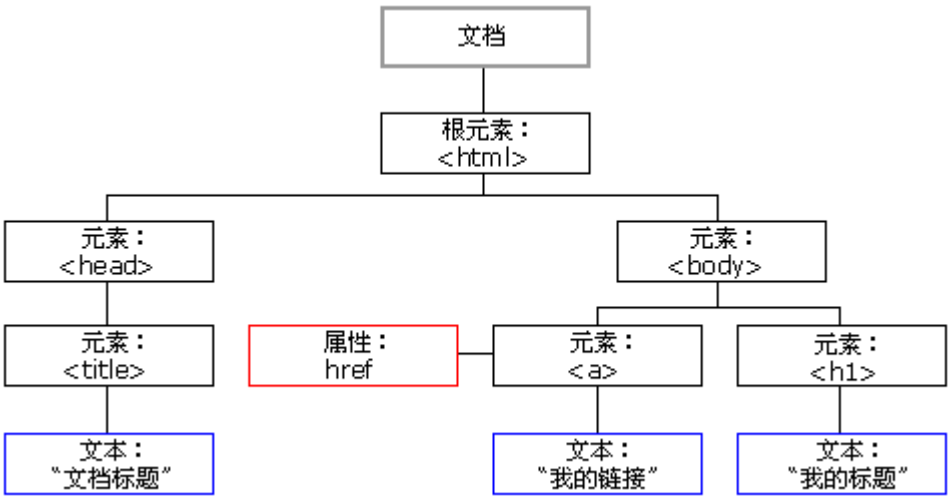
九、JS操作DOM

1、DOM简介

> HTML DOM 定义了访问和操作 HTML 文档的标准方法。
>
> DOM 将 HTML 文档表达为树结构

浏览器在加载HTML代码的时候，会根据W3C规范，将HTML代码解析成内置`document`对象

- HTML代码和DOM对象有 对应关系
- 操作的是`document`（DOM）对象，不是HTML代码
- DOM对象属性发生改变时，浏览器会自动地重新渲染页面



2、访问节点

2-0 HTML文档

```
<div id="d" >xxxx</div>
<span>bbbb</span>
<span>cccc</span>
<ul>
  <li>1</li>
  <li>2</li>
  <li>3</li>
  <li>4</li>
</ul>
<input type="checkbox" name="hobby">
<input type="checkbox" name="hobby">
<input type="checkbox" name="abc">
```

2-1 根据id查询

```
// 页面加载完成后执行
// 是一个方法，该方法默认为null，
// 可以通过代码覆盖该属性值
// 这个方法会在页面加载完成后自动调用
window.onload = function(){
  // 根据id获取标签  html中id值可以重复  该方法只会返回第一个
  var div = document.getElementById('d');
  console.log(div);
}
```

2-2 根据标签名查询

```
// 根据标签名获取标签，返回的是数组
var spans = document.getElementsByTagName("span");
console.log(spans);
```

2-3 根据name属性查询

```
//根据name属性获取标签
var cbx = document.getElementsByName("hobby");
console.log(cbx);
```

2-3 使用CSS选择器

```
// 根据选择器获取一个标签 选择器和css选择器完全一致
var div = document.querySelector("#d");
console.log(div);
// 根据选择器获取多个标签 选择器和css选择器完全一致
var cbx  = document.querySelectorAll("[name=hobby]");
console.log(cbx);
```

3、访问属性

3-1 HTML属性 和 DOM 属性

- HTML属性 是指 直观的 写在HTML代码中的所有属性（包括自定义属性）
- DOM属性 是 虚拟属性，是浏览器解析HTML代码转换成DOM对象时，对象中的属性
- **一般情况下，HTML属性都存在一个同名的DOM属性**
- **特例**

HTML	DOM
class	className
所有的自定义属性	-
等等.....	

3-2 访问DOM属性

`dom对象.属性名` 或者 `dom对象['属性名']`

上面的访问方式 放在 等号 右边 取值操作，放在等号左边 赋值操作

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script>
        function doChange()
        {
            // 点击按钮，将div在矩形 和 圆形之间 来回切换
            let div = document.getElementById("d");
            div.className = div.className ? "" : "circle";
```

```
        }
    </script>
    <style>
        div{
            width: 100px;
            height: 100px;
            border:1px solid black;
        }
        .circle{
            border-radius:50%;
        }
    </style>
</head>
<body>
    <div id="d" class="" ></div>
    <button onclick="doChange()">按钮</button>
</body>
</html>
```

3-3 访问HTML属性

- `getAttribute("属性名")` 获取HTML属性值
- `setAttribute("属性名","属性值")` 设置HTML属性值

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script>

        window.onload = function(){
            let div = document.querySelector(".d");
            let attrValue = div.getAttribute("a");
            console.log(attrValue);
            div.setAttribute("a", "haha");
            div.setAttribute("b","abc");
            // div.className = "c";
            div.setAttribute("class","c");
        }

    </script>
    <style>
        .c{
            background-color:green;
        }
    </style>
</head>
<body>
```

```
    <div class="d" a="b">aaa</div>
</body>
</html>
```

4、访问CSS样式

4-1 访问 `class`

详见3 访问属性

4-2 访问style

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script>

        window.onload = function(){
            //style 是div 的属性
            let div = document.querySelector(".d");
            // stylesheet 对象
            let style = div.style;
            console.log(style);

            // 修改style属性中height的值为400px
            div.style.height = "400px";
            // 修改style属性为height:600px
            div.style = "height:600px";

            // -----------------注意：如果CSS属性为aaa-bbb形式--------
            // 修改背景色为绿色
            // 方式1 使用[]访问属性
            div.style['background-color'] = "green";
            // 方式2 改写成aaaBbb形式
            div.style.backgroundColor = "yellow";


            // 修改左边框为红色
            // border-left-color:red;
            div.style.borderLeftColor = "red";

            //---注意：只能访问style属性，不能访问class样式-------
            let wid = div.style.width;
            console.log("width:",wid);

        }
    </script>
    <style>
        .d{

            width: 100px;
```

```
            border:1px solid black;
        }
    </style>
</head>
<body>

    <div class="d" style="height: 200px;background-color: red"></div>

</body>
</html>
```

5、访问内容

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script>

        window.onload = function(){
            // 取值操作，会有空格
            let div = document.querySelector("#d");
            let content = div.innerHTML;
            console.log(content,content.length,content.trim().length);
            // 可以取出标签
            let newDiv = document.querySelector("#sd");
            content = newDiv.innerHTML.trim();
            console.log(content);
            // 赋值操作，覆盖原有值
            newDiv.innerHTML = "aaaaaaa";
            // 赋值操作，可以解析HTML标签
            newDiv.innerHTML = `
                <h1>HelloWorld</h1>
            `;


        }

        function doAdd()
        {
            let newDiv = document.querySelector("#sd");
            newDiv.innerHTML = newDiv.innerHTML +  `
                <h1>HelloWorld</h1>
            `;
        }

    </script>
</head>
<body>
    <div id="d">

        asda
```

```
    </div>

    <div id="sd">
        <div>asdfladjs</div>
        <a href="#">ads</a>
    </div>

    <button onclick="doAdd()">add</button>
</body>
</html>
```

6、事件绑定

> 为标签绑定事件处理函数
>
> 当某个事件发生的时候，执行的函数

6-1 绑定方式

6-1-1 静态绑定

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script>
        function handler()
        {
            alert("按钮被点击")
        }

    </script>
</head>
<body>
    <!-- 直接在标签上绑定事件 -->
    <button onclick="handler()">btn...</button>

</body>
</html>
```

6-1-2 动态绑定

通过js代码绑定事件

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script>

        window.onload = function(){
```

```
            let btn = document.querySelector("#btn");
            // 通过js 访问按钮的onclick属性
            btn.onclick = function(){
                alert("aaaaaaaa")
            }
        }

    </script>
</head>
<body>
    <button id="btn">aaa</button>
</body>
</html>
```

6-2 常用事件

1. 键盘事件

事件名	作用	备注
onkeydown	键盘按下事件	
onkeyup	键盘弹起事件,可以获取到最后一次敲击的内容	§
onkeypress	键盘敲击事件	

2. 鼠标事件

事件名	作用	备注
onclick	点击事件（作用于所有标签）	§
ondbclick	鼠标双击	
onmousedown	鼠标按下	
onmouseup	鼠标释放	
onmouseover	鼠标悬停	§
onmouseout	鼠标移出	§
onmousemove	鼠标移动	
7个鼠标的拖拽事件	-	-

3. 表单 相关事件

事件名	作用	备注
onblur	输入框失去焦点	§
onfocus	输入框获取到焦点	§
onchange	**一般用于select、input[file]**	§
onselect	输入框中的文字被选中	§
onsubmit	表单提交前，用于form	§

7、练习

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script>
        function doChange(){
            document.querySelector("#target").innerHTML =
                    document.querySelector("#src").value;
        }

        function f1(div,clsName)
        {
            div.className = clsName;
        }

        let arr = ['点不到吧','就是不让你点','着急吧','别试了，点不到的'];
        let index = 0;
        function changePos(btn){
            btn.style.left = Math.random()*948 + "px";
            btn.style.top = Math.random()*175 + "px";
            btn.innerHTML = arr[index++ % arr.length];
        }

        function checkUsername (input) {
            let value = input.value;
            let span = document.querySelector("#msg");
            span.innerHTML= "失败";
            span.style.color = "red";
            if( /^\w{6,}$/.test(value) )
            {
                span.innerHTML= "成功";
                span.style.color = "green";
            }
        }
    </script>
    <style>
```

```
        .target{
            width: 100px;
            height: 100px;
            border:1px solid black;
        }
        .circle{
            background-color:red;
            border-radius:50%;
        }
        .btn{
            position:relative;
            left:20px;
            top:40px;
        }
        .container{
            width: 100%;
            height: 200px;
            border:1px solid black;
        }
    </style>
</head>
<body>
    <input type="text" id="src" onkeypress="doChange()" />
    <h1 id="target">暂无内容</h1>


    <hr>

    <div class="target" onmouseover="f1(this,'target circle')"
onmouseout="f1(this,'target')">

    </div>

    <hr>
    <!--     left: 948px;
        top: 175px; -->
    <div class="container">
        <button class="btn" onmouseover="changePos(this)">永远点不到按钮</button>
    </div>

    <hr>

    <input type="text" id="username" placeholder="请输入用户名"
onblur="checkUsername(this)" />
    <span style="color:#c3c3c3" id="msg">只能是字母数字下划线，至少6位</span>
</body>
</html>
```

8、阻止浏览器默认行为

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script>

        function f()
        {
            return false;
        }

        function check () {
            let value = document.querySelector("#d2").value;
            let flag = /\w{6,}/.test(value);
            if(!flag)
            {
                document.querySelector("#msg").innerHTML = "aaaa";
            }
            console.log(flag);
            return flag;
        }

    </script>
</head>
<body>

    <!-- 点击a标签，浏览器会跳转 -->
    <a href="a.html" onclick="return false">aaaa</a>
    <a href="a.html" onclick="return f()">aaaa</a>
    <!-- 点击提交按钮，表单会提交 -->
    <form action="a.html" onsubmit="return false">
        <input type="text" id="d">
        <input type="submit">
    </form>
    <hr>
    <form action="a.html" onsubmit="return check()">
        <input type="text" id="d2">
        <span id="msg"></span>
        <input type="submit">
    </form>

</body>
</html>
```

9、阻止事件冒泡

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <style>
        #d{
```

```
            width: 200px;
            height: 200px;
            background-color:yellow;
        }

        #inner{
            width: 40px;
            height: 40px;
            background-color:red;
            position:relative;
            left: 170px;
            top: 170px;
        }
    </style>
    <script>

        function outerClick (argument) {
            alert(1);
        }

        function innerClick (e) {
            alert(2);
            console.log(e);
            e.cancelBubble = true;
        }

    </script>
</head>
<body>

    <div id="d" onclick="outerClick()">
        <!-- 事件对象 -->
        <div id="inner" onclick="innerClick(event)"></div>
    </div>

</body>
</html>
```

十、`window`对象

1、特性

- `window`对象是js中的一个内置对象
- 我们前面讲的所有方法都是`window`对象中的方法
- 我们前面定义的所有变量其实也是`window`对象属性
- **`window`对象的属性和方法可以不使用`window.`访问**

2、常用属性

2-1 document

2-2 location

> 表示 浏览器地址栏

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script>
        function doGet () {
            let href = location.href;
            console.log("浏览器地址栏中的内容",href);
        }
        function doSet () {
            location.href = "http://bbs.itany.com";
        }
        function doReload()
        {
            location.reload();
        }

    </script>
</head>
<body>
    <button onclick="doGet()">getLocation</button>
    <button onclick="doSet()">setLocation</button>
    <button onclick="doReload()">setReload</button>
</body>
</html>
```

3、常用方法

3-1 onload

3-2 三个提示框

```
// 提示消息
alert("xxx");

// 弹出输入框 点击确定，返回输入的内容，点击取消 返回null
let str = prompt("提示消息", "默认值");
console.log(str);

// 弹出对话框 返回用户选择结果  确定 true 取消 false
let c = confirm("xxxx");
console.log(c);
```

3-3 `setTimeout` / `clearTimeout`

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script>

        // 返回定时器对象，延迟timeout毫秒，执行handler方法
        // let timer = setTimeout(handler,timeout);
        let timer = setTimeout( ()=>{
            console.log('111111111111111111');
        },3000 );

        function stop(){
            // 清除定时器
            clearTimeout(timer);
        }
    </script>
</head>
<body>
    <button onclick="stop()">stop</button>
</body>
</html>
```

3-4 `setInterval` / `clearInterval`

方法的使用方式和timeout是一致的，作用是 每隔一段时间，执行方法

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script>
        // 返回定时器对象，延迟timeout毫秒，执行handler方法
        // let timer = setTimeout(handler,timeout);
        let timer = setInterval( ()=>{
            console.log('111111111111111111');
        },500 );

        function stop(){
            // 清除定时器
            clearInterval(timer);
        }
    </script>
</head>
<body>
    <button onclick="stop()">stop</button>
</body>
</html>
```