Javascript预习

javascript书写方式

嵌入式: 写在标签中

外链式: 写在js文件中,通过

行内式:

div的内容

五个常用的输出语句

document.write() 在页面上输出内容,如果输出的内容是标签,一样会解析标签

console.log() 在控制台输出内容

window.alert() 弹窗

window.prompt() 弹出一个输入框,供用户输入,得到的值是String类型

window.confirm() 确认框,点击确定,返回true,点击取消,返回false

window.可以省略,也就是alert(),prompt(),confirm()

javascript的注释

```
单行注释：// 注释内容

多行注释：/*注释内容*/
```

变量

什么是变量 会变化的量

如何使用变量

```
var 变量名 = 变量值;

var 变量1=值1,变量2=值2;

var 变量名;
变量名 = 值;

var 变量1,变量2;
变量1 = 值1;
变量2 = 值2;
```

变量的命名规则和规范 区分大小写 不能使用数字开头

变量的数据类型分为： 两大类 、 七小种

- 基本数据类型 只能存储一个值

```
字符串string            被引号包裹的内容就是字符串
数值number             正数,负数,0,NaN
布尔boolean            true false
空型   null
未定型undefined         定义一个变量没有赋值得到undefined   或者   定义一个变量的值就是undefined
```

- 复合数据类型 至少存储一个值，可以存储多个值

```
数组array    对象object
```

如何获取变量的数据类型

```
typeof(变量名)
或者
typeof 变量名
```

运算符

算术运算符:常见的有+,-,*,/等

++ 自增运算符 前++跟后++ 共同点:最后的结果都会加1 不同的是,有其他代码的时候

前++,会自加一,再赋值 后++,会赋值,再自加一

-- 自减运算符 前-- 跟后 -- 共同点:最后的结果都会减1 不同的是,有其他代码的时候

前--,会自减一,再赋值 后--,会赋值,再自减一

% 取余(求模)

10%2 = 0

11%2 = 1

+号运算符

> 如果两个运算数都是数值,他就是加法　算术运算符
>
> 如果两个运算数有一个是字符串,那个+号就是就是字符连接符

赋值运算符:

+=

var str = 'a';

```
str+='123';     //  str=str+'123'= 'a'+'123'='a123'
```

逻辑运算符

&& 逻辑与 两个条件都是true,才为true

|| 逻辑或 两个条件,只要有一个是true,最后结果就为true

! 逻辑非(取反) !true => false !false => true

分支结构

①单分支

```
if(条件表达式){
    成立执行的代码块;
}
```

②双分支

```
if(条件表达式){
   成立执行的代码块;
}else{
   不成立执行的代码块;
}
```

③多分支

```
if(条件表达式1){
   条件1成立执行的代码块;
}else if(条件表达式2){
   条件1不成立,但是条件2成立       执行的代码块;
}else if(条件表达式3){
   条件1不成立,条件2也不成立,但是条件3成立       成立执行的代码块;
}else{
   上述条件都不成立执行的代码块;
}
```

④switch语句

```
switch(值){
    case 值1:
        当值全等于值1的时候,执行的代码块
        break;
    case 值2:
        当值全等于值2的时候,执行的代码块
        break;
    case 值3:
        当值全等于值3的时候,执行的代码块
        break;
    default:
        找不到对应case,执行的代码块
}
```

循环结构

①for循环

```
for(初始化;判断条件;改变变量的值){
    //循环体
}

//输出1~10
for(var i=1;i<=10;i++){
    //i的变化 1 2 3 4 5 6 7 8 9 10
    document.write(i);
}
```

②while循环

```
初始化;
while(判断条件){
    //循环体
    改变变量的值
}

//输出1~10
var i=1;
while(i<=10){
    document.write(i);
    i++;
}
```

③do...while循环

```
初始化;
do{
    //循环体
    改变变量的值
}while(判断条件)

//输出1~10
var i=1;
do{
    document.write(i);
    i++;
}while(i<=10)
```

④循环中断关键字

```
break    跳出整个循环
continue 跳出本次循环,继续下一次循环
```

数组

定义数组

```
var arr = [10,20,30];
var arr = new Array(5);// 生成数组长度为5的空数组
var arr = new Array(10,20,30);
```

数组下标

```
下标是从0开始的递增的数字,最大下标为数组长度-1
```

访问数组中某个元素

```
数组变量名[下标]
```

数组长度

```
数组变量名.length
```

遍历数组

```
for(var i=0;i<arr.length;i++){
    console.log(arr[i]);
}
```

二维数组的定义与二维数组的遍历

```
什么是二维数组?  数组中的元素又是数组

var students = [
    ["张三",23,"男"],
    ["李四",24,"女"],
    ["王五",25,"男"]
];
```

```
得到张三 =>  students[0][0]
得到24  =>   students[1][1]


for(var i=0;i<students.length;i++){
    for(var j=0;j<students[i].length;j++){
        console.log(students[i][j])
    }
}
```

函数

定义函数

```
function 函数名([形参列表]){
    函数体;
}

定义函数的时候,函数体不会执行
```

调用函数

```
调用函数以后函数体才会执行,调用几次函数,就会执行几次函数体

函数名([实参列表])
```

return关键字

```
return向函数的调用者返回指定内容
return终止函数体的执行
return没有输出的功能
```

匿名函数

没有名字的函数,一般用于事件驱动程序中

```
btn.onclick=function(){
    console.log(111)
}
```

变量的作用域 分为：全局作用域 和 局部作用域

全局作用域 在函数外定义的变量或者函数　　在任何地方都可以使用

局部作用域 在函数内定义的变量或者函数　　在定义的函数中才可以使用

对象

对象的分类 `自定义对象` 和 `内置对象`

创建自定义对象

①使用{}

```
// 定义一个对象
var girlFriend = {
    height:180,
    weight:"75kg",
    shopping:function(){
        console.log("花你的钱");
    }
}

// 访问对象属性
console.log( girlFriend.height );
// 访问对象方法
girlFriend.shopping();
```

②new Object

```
// 定义一个对象
var girlFriend = new Object();
girlFriend.height=180;
girlFriend.weight="75kg";
girlFriend.shopping=function(){
    console.log("花你的钱");
}

// 访问对象属性
console.log( girlFriend.height );
// 访问对象方法
girlFriend.shopping();
```

③构造函数创建自定义对象

```
// 定义一个类
function Person(name,age,sex){
    this.name = name;
    this.age = age;
    this.sex = sex;
    this.say = function(){
        console.log("hello,my name is "+this.name);
    }
}

// 需要通过new实例化对象
var p1 = new Person("张三",23,"男");
p1.say();

var p2 = new Person("李四",24,"女");
p2.say();
```

JSON数据格式

JSON格式的定义,定义JSON对象

```
var json = {
    "属性名1":属性值1,
    "属性名2":属性值2,
}

json中只能有属性,不能有方法
```

访问 JSON对象的数据

```
var json = {
    "code":200,
    "msg":"获取数据成功"
}
console.log(json.code);
console.log(json["msg"]);
```

遍历JSON对象的属性

```
var json = {
    "code":200,
    "msg":"获取数据成功",
    "status":"success",
        "date":new Date().toString()
}
for(attr in json){
    console.log(json[attr]);
}
```

JSON数组 就是 把多个json对象放在一个数组中

```
var jsonArr = [
    {"name":"张三","age":23,"sex":"男"},
    {"name":"李四","age":24,"sex":"女"},
    {"name":"王五","age":25,"sex":"男"},
];

// console.log( jsonArr[1]["name"] );

for(var i=0;i<jsonArr.length;i++){
        for(attr in jsonArr[i]){
        console.log(jsonArr[i][attr]);
        }
}
```

内置对象

内置String对象

```
如何创建String对象?
var str = "abc";
```

```
var str = new String("abc");

如何获取字符串的长度
str.length

字符串[下标]    得到字符串中指定下标的某个字符串

字符串.substr(start[,length])    截取字符串
        var str = "hello world";
        console.log( str.substr(2,3));
        console.log( str.substr(2) );

字符串.indexOf("要查找的字符或字符串")    查找子字符串是否存在,存在返回所在位置的下标   不存在返回-1
        var str = "hello world";
        console.log( str.indexOf("w") );
        console.log( str.indexOf("a") );

字符串.split(分割符)  根据分隔符把字符串分割成一个数组
```

内置Math对象

```
Math.floor(数值)                向下取整
Math.ceil(数值)          向上取整
Math.random()              随机数 0~1之间所有小数   0可以取,1取不到

Math.floor(Math.random()*(最大值-最小值)+最小值)  得到最小值到最大值的之间的整数,包括最小,不包括最大值
Math.floor(Math.random()*(最大值-最小值+1)+最小值) 得到最小值到最大值的之间的整数,包括最小,最大值
```

内置Date对象

```
如何创建Date对象?
var date = new Date();
var date = new Date("2020/2/2 5:5:12");
var date = new Date("2020-2-2 5:5:12");
```

> 如果我们想要 2008-08-08 08:08:08 格式怎么办?

我们需要手动的得到这种格式

方法名	功能
getFullYear()	获取4位数的年份
getMonth()	获取月份 返回值 **0~11**　　0表示1月 11表示12月
getDate()	返回一个月中的某一天 返回值：1~31
getHours()	小时 返回值0~23
getMinutes()	获取分钟 返回值：0~59
getSeconds()	获取秒数 返回值：0~59
getMilliseconds()	获取毫秒 返回值：0~999
getDay()	获取一周中的某一天 就是星期几 返回值：**0~6** 0代表星期天 1代表星期一
getTime()	获取时间戳 返回从1970年1月1日 一直到现在的**毫秒数**!
toLocaleString()	根据本地时间把 Date 对象转换为字符串，并返回结果。

> 注意 月份 和 星期 取值范围是从 0开始的。

getDay 方法所返回的值是一个处于 0 到 6 之间的整数，它代表了一周中的某一天，返回值与一周中日期的对应关系如下：

值	星期
0	星期天
1	星期一
2	星期二
3	星期三
4	星期四
5	星期五
6	星期六

内置Array对象

如何创建Array对象,如何创建数组

```
var arr = [];
var arr = [10,20,30];
var arr = new Array();
var arr = new Array(10);
var arr = new Array(10,20,30);
```

检查是否为数组

```
Array.isArray(变量名)

var arr = [10,20,30];
console.log( Array.isArray(arr) );
```

> 数组添加和删除元素方法

方法名	说明	返回值
push(参数1..)	修改原数组，末尾添加一个或多个数组元素	并返回新的长度
pop()	删除 数组的最后一个元素，把数组长度减 1 无参数	返回它删除的元素的值
unshift(参数1...)	向数组的开头添加一个或更多数组元素	并返回新的长度
shift()	把数组的第一个元素从其中删除，把数组长度减 1 无参数	返回它删除的元素的值

> 数组的所有元素合并到一个字符串中

方法名	说明	返回值
toString()	把数组转成字符串,逗号分隔每一项	返回一个字符串
join('分隔符')	用于把数组中的所有元素以分隔符连接,合并为一个字符串; 如果这个参数分隔符没有写 则默认使用英文状态下的**逗号**进行连接	返回一个字符串

> 数组截取slice、数组删除splice

方法名	说明	返回值
slice(begin下标,end下标)	数组截取	返回截取出来的新数组 包括 `begin` ，不包括 `end` ,这个**不会**影响原数组
splice(从第几个开始,要删除几个)	数组删除	返回被删除元素组成的数组 ,这个**会**影响原数组

DOM对象

> 获取对象的多种方法

描述	方法
根据ID获取元素	document.getElementById(id)
根据标签名获取元素	document.getElementsByTagName('标签名')
根据标签名获取元素	父元素对象.getElementsByTagName('标签名')
根据类名获取元素	document.getElementsByClassName("类名")
根据CSS选择器获取元素	**document.querySelector("css选择器")**
根据CSS选择器获取元素	**document.querySelectorAll("css选择器")**
获取body元素	document.body
获取html元素	document.documentElement

> DOM改变元素内容

```
innerHTML  innerText
```

> DOM操作元素的属性

```
操作自有属性  比如img有src属性   a标签href属性
```

> DOM操作元素的样式

```
元素.style.css属性名 = css属性值;          css属性名如果有短横杠要去掉,驼峰法 比如font-size=>fontSize

元素.className = "类名";
```

> 自定义属性操作

- 获取属性值

```
元素.getAttribute("属性名");

元素.getAttribute("class");
元素.getAttribute("src");
元素.getAttribute("index");
```

- 设置属性值

```
元素.setAttribute("属性名","属性值");

元素.setAttribute("class","current");
元素.setAttribute("src","images/1.gif");
元素.setAttribute("index",1);
```

- 移除属性

```
元素.removeAttribute("属性名");
```

> 节点操作

- 父级节点　　子节点.parentNode
- 子节点　　父节点.children
- 创建节点　　document.createElement("标签名")
- 添加节点　　父节点.appendChild(子节点)
- 删除节点　　父节点.removeChild(子节点)

事件

事件三要素

```
事件源
事件类型
事件驱动程序
```

事件的绑定方式 行内绑定方式 和 动态绑定方式

```
//行内绑定方式
function fn(){
    alert("1111");
}

<button onclick="fn()">按钮</button>
```

```
//动态绑定方式

<button>按钮</button>
var btn = document.querySelector("button");
btn.onclick = function(){
    alert("11111");
}
```

常见的鼠标事件

鼠标事件	触发条件
onclick	鼠标点击左键触发
onmouseover	鼠标经过触发
onmouseout	鼠标离开触发
onfocus	获得鼠标焦点触发
onblur	失去鼠标焦点触发
onmousemove	鼠标移动触发
onmouseup	鼠标弹起触发
onmousedown	鼠标按下触发

> 事件高级-注册事件（2种方式）

给元素添加事件,称为注册事件或者绑定事件

注册事件有两种方式:传统方式和监听注册方式

传统注册方式

- 利用 on 开头的事件 onclick
- <button onclick= "alert('hi~')" ></button>
- btn.onclick = function() {}
- 特点： 注册事件的唯一性
- 同一个元素同一个事件只能设置一个处理函数，最后注册的处理函数将会覆盖前面注册的处理函数

监听注册方式

- w3c 标准 推荐方式
- addEventListener() 它是一个方法
- IE9 之前的 IE 不支持此方法，可使用 attachEvent() 代替
- 特点：同一个元素同一个事件可以注册多个监听器
- 按注册顺序依次执行

> 删除事件（解绑事件）

1. **传统注册方式**

```
eventTarget.on事件名称 = null;

比如:eventTarget.onclick = null;
```

2. **方法监听注册方式**

```
① eventTarget.removeEventListener(type,listener[,useCapture]);
② eventTarget.detachEvent(eventNameWithOn,callback);
```

> 事件对象

事件发生后，跟事件相关的一系列信息数据的集合都放到这个对象里面，这个对象就是事件对象。

比如:

1. 谁绑定了这个事件。
2. 鼠标触发事件的话，会得到鼠标的相关信息，如鼠标位置。
3. 键盘触发事件的话，会得到键盘的相关信息，如按了哪个键。

> 事件对象的使用

事件触发发生时就会**产生事件对象**，并且**系统会以实参的形式**传给事件处理函数。

所以，**在事件处理函数中声明1个形参用来接收事件对象**。

```
eventTarget.onclick = function(event){
    // 这个event 就是事件对象,我们还喜欢写成e或者evt
};

eventTarget.addEventListener("click",function(event){
    // 这个event 就是事件对象,我们还喜欢写成e或者evt
});

eventTarget.attachEvent("onclick",function(event){
    // 这个event 就是事件对象,我们还喜欢写成e或者evt
});
```

> 事件对象的属性和方法

事件对象属性方法	说明
e.target	返回触发事件的对象 标准
e.stopPropagation()	该方法阻止冒泡 标准
e.preventDefault()	该方法阻止默认事件(默认行为) 标准 比如不让链接跳转
return false	利用return false 也能阻止默认行为 没有兼容问题 但是return后面的代码不执行

> 鼠标事件对象

event 事件对象是事件相关的一系列信息的集合.

现阶段我们主要用鼠标事件对象**MouseEvent** 和 键盘事件对象**KeyBoardEvent**

鼠标事件对象	说明
e.pageX	返回鼠标相对于**文档页面**的X坐标 IE9以后支持
e.pageY	返回鼠标相对于**文档页面**的Y坐标 IE9以后支持

> 键盘事件

键盘时间	触发条件
onkeyup	某个键盘按键被松开时触发
onkeydown	某个键盘按键被按下时触发
onkeypress	某个键盘按键被按下时触发 但是它不识别功能键 比如 ctrl shift 箭头等

> 键盘事件对象

需要传event事件对象参数才可以使用

键盘事件对象属性	说明
keyCode	返回该键的ASCII值

注意:

1. **onkeydown和onkeyup不区分字母大小写,onkeypress区分字母大小写**
2. 在我们实际开发中,我们更多的使用 `keydown` 和 `keyup` , 它能识别所有的键(包括功能键)
3. **keypress不识别功能键,但是keyCode属性能够区分大小写**,返回不同的ASCII值

BOM对象

什么是BOM

BOM（Browser Object Model）即浏览器对象模型，它提供了独立于内容而与浏览器窗口进行交互的对象，其核心对象是 window。

BOM 由一系列相关的对象构成，并且每个对象都提供了很多方法与属性。

BOM 缺乏标准，JavaScript 语法的标准化组织是 ECMA，DOM 的标准化组织是 W3C，BOM 最初是Netscape 浏览器标准的一部分。

DOM

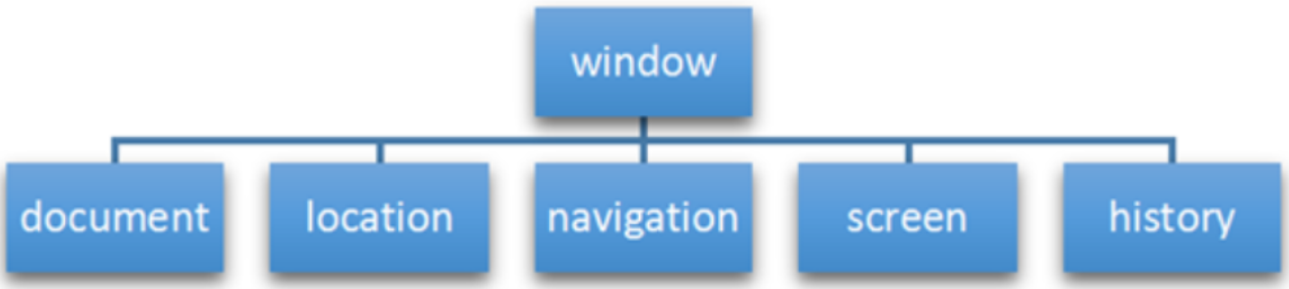
- 文档对象模型
- DOM 就是把「文档」当做一个「对象」来看待
- DOM 的顶级对象是 document
- DOM 主要学习的是操作页面元素
- DOM 是 W3C 标准规范

BOM

- 浏览器对象模型
- 把「浏览器」当做一个「对象」来看待
- BOM 的顶级对象是 window
- BOM 学习的是浏览器窗口交互的一些对象
- BOM 是浏览器厂商在各自浏览器上定义的，兼容性较差

BOM的构成

BOM 比 DOM 更大，它包含 DOM。



window对象

> 页面（窗口）加载事件

```
window.onload = function(){}
或者
window.addEventListener("load",function(){});
```

定时器

```
var timer = window.setInterval(function(){
    // 代码
},1000)

function fn(){
        console.log("hello");
}

var timer = window.setInterval(fn,1000)

var timer = window.setInterval("fn()",1000)
```

清除定时器

```
window.clearInterval(timer)
```

this指向问题

this的指向在函数定义的时候是确定不了的，只有函数执行的时候才能确定this到底指向谁，一般情况下this的最终指向的是那个调用它的对象。

现阶段，我们先了解一下几个this指向

1. 全局作用域或者普通函数中this指向全局对象window（注意延时器跟定时器里面的this指向window）
2. 对象里面的方法,谁调用, this就指向谁
3. 构造函数中this指向构造函数的实例

```
<ul>
        <li>111</li>
        <li>222</li>
        <li>333</li>
</ul>

var lis = document.querySelectorAll("ul li");
for(var i=0;i<lis.length;i++){
    lis[i].onclick=function(){
        console.log(this);// 谁触发代表谁
    }
}
```

```
console.log(this);//window

function fn(){
    console.log(this);// window
}

setInterval(function(){
        console.log(this)//window
},1000)

setTimeout(function(){
        console.log(this)//window
},1000)
```

location 对象的属性

location对象属性	返回值
location.href	**获取或者设置 URL**

```
location.href = "https://www.baidu.com";
```

navigator对象

navigator 对象包含有关浏览器的信息，它有很多属性，我们最常用的是 **userAgent**，该属性可以返回由客户机发送服务器的 user-agent 头部的值。

下面前端代码可以判断用户那个终端打开页面，实现跳转

```
if((navigator.userAgent.match(/(phone|pad|pod|iPhone|iPod|ios|iPad|Android|Mobile|BlackBerry|IEMobile|MQQBrowser|JUC|Fennec|wOSBrowser|BrowserNG|WebOS|Symbian|Windows Phone)/i))) {
    window.location.href = "";     //手机
 } else {
    window.location.href = "";     //电脑
 }
```

history对象

window对象给我们提供了一个 history对象，与浏览器历史记录进行交互。该对象包含用户（在浏览器窗口中）访问过的URL。

history对象方法	作用
history.back()	实现后退1个页面
history.forward()	实现前进1个页面
history.go(n)	参数n为数值,可以实现前进或者后退n个页面；n如果是正值代表前进,n如果是负值代表后退； 比如 history.go(1)代表前进一个页面；history.go(-1)代表后退一个页面；

元素偏移量offset系列

offset 翻译过来就是偏移量， 我们使用 offset系列相关属性可以动态的得到该元素的位置（偏移）、大小等。

1. 获得元素距离带有定位父元素的位置
2. 获得元素自身的大小（宽度高度）
3. 注意:返回的数值都不带单位

offset系列属性	作用
element.offsetParent	返回作为该元素带有定位的父级元素 如果父辈元素都没有定位则返回body元素
element.offsetTop	返回元素相对带有定位父元素上方的偏移量 如果父辈都没有定位则返回相对body的上方偏移量
element.offsetLeft	返回元素相对带有定位父元素左方的偏移量 如果父辈都没有定位则返回相对body的左方偏移量
element.offsetWidth	返回自身包括**padding、边框、内容区的总宽度**，返回数值不带单位
element.offsetHeight	返回自身包括**padding、边框、内容区的总高度**，返回数值不带单位

元素可视区 client 系列

client 翻译过来就是客户端，我们使用 client 系列的相关属性来获取元素可视区的相关信息。通过 client 系列的相关属性可以动态的得到该元素的边框大小、元素大小等。

client系列属性	作用
element.clientTop	返回上边框的大小
element.clientLeft	返回左边框的大小
element.clientWidth	返回自身包括padding、内容区宽度 ,不含边框 , 返回数值不带单位
element.clientHeight	返回自身包括padding、内容区高度 ,不含边框 , 返回数值不带单位

注意:clientTop可以获取上边框的大小,clientLeft可以获取左边框的大小,没有`clientBottom`跟`clientRight`

小结: client 宽度 和我们offsetWidth 最大的区别就是 **不包含边框**

元素滚动 scroll 系列

scroll 翻译过来就是滚动的，我们使用 scroll 系列的相关属性可以动态的得到该元素的大小、滚动距离等。

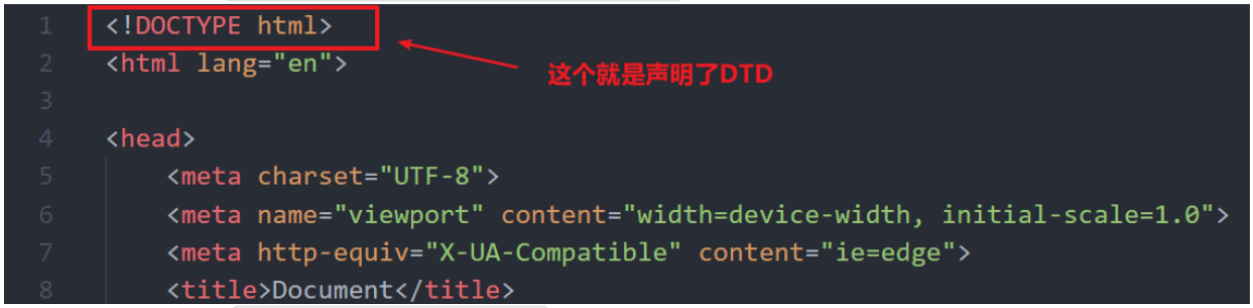
scroll系列属性	作用
element.scrollTop	返回被卷去的上侧距离,返回数值不带单位
element.scrollLeft	返回被卷去的左侧距离,返回数值不带单位
element.scrollWidth	返回自身实际的宽度,**不含边框**,返回数值不带单位
element.scrollHeight	返回自身实际的高度,**不含边框**,返回数值不带单位

注意: scrollHeight跟clientHeight的区别在于,文字内容超出盒子高度的时候,scrollHeight获取的是内容的高度,而clientHeight获取的还是盒子的高度

页面被卷去的头部兼容性解决方案

需要注意的是，**页面被卷去的头部，有兼容性问题**，因此被卷去的头部通常有如下几种写法:

1. 声明了 DTD，使用 `document.documentElement.scrollTop`




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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
```

2. 未声明 DTD，使用 `document.body.scrollTop`
3. 新方法 window.pageYOffset和 window.pageXOffset，**IE9 开始支持**