JS操作

2020年9月18日　　9:33

BOM与DOM操作

```
# 截至目前为止 我们虽然已经学会了js语法 但是你会发现跟浏览器和html文件还是一点关系没有
"""
BOM
  浏览器对象模型  Browser Object Model
    js代码操作浏览器
DOM
  文档对象模型    Document Object Model
    js代码操作标签
"""
```

BOM操作

```
# window对象
window对象指代的就是浏览器窗口

window.innerHeight  浏览器窗口的高度
900
window.innerWidth   浏览器窗口的宽度
1680

window.open('https://www.mzitu.com/','','height=400px,width=400px,top=400px,left=400px')
# 新建窗口打开页面 第二个参数写空即可 第三个参数写新建的窗口的大小和位置
# 扩展父子页面通信window.opener()  了解

window.close()  关闭当前页面
```

window子对象

```
window.navigator.appName
"Netscape"
window.navigator.appVersion
"5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/81.0.4044.138 Safari/537.36"

window.navigator.userAgent    掌握  # 用来表示当前是否是一个浏览器
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/81.0.4044.138 Safari/537.36"

"""
扩展:仿爬措施
  1.最简单最常用的一个就是校验当前请求的发起者是否是一个浏览器
    userAgent
    user-agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.138
Safari/537.36
  如何破解该措施
    在你的代码中加上上面的user-agent配置即可
"""

window.navigator.platform
"MacIntel"

# 如果是window的子对象 那么window可以省略不写
```

history对象

```
window.history.back()  回退到上一页
window.history.forward()  前进到下一页
# 对应的就是你浏览器左上方的两个的箭头
```

location对象(掌握)

```
window.location.href  # 获取当前页面的url
window.location.href = url  # 跳转到指定的url
window.location.reload()  # 刷新页面  浏览器左上方的小圆圈
```

弹出框

- 警告框
- 确认框
- 提示框

```
alert('你不要过来啊！！！')
undefined

confirm('你确定真的要这么做吗?能不能有其他方式能够满足你...')
false
confirm('你确定真的要这么做吗?能不能有其他方式能够满足你...')
true
prompt('手牌号给我看一下','22号消费888')
"来宾三位"
```

计时器相关

- 过一段时间之后触发(一次)
- 每隔一段时间触发一次(循环)

```
<script>
    function func1() {
        alert(123)
    }
    let t = setTimeout(func1,3000);  // 毫秒为单位 3秒之后自动执行func1函数

    clearTimeout(t)       // 取消定时任务  如果你想要清除定时任务 需要日前用变量指代定时任务


    function func2() {
        alert(123)
    }
    function show(){
        let t = setInterval(func2,3000);  // 每隔3秒执行一次
        function inner(){
            clearInterval(t)  // 清除定时器
        }
        setTimeout(inner,9000)  // 9秒中之后触发
    }
    show()
</script>
```

DOM操作

```
"""
DOM树的概念

所有的标签都可以称之为是节点

JavaScript 可以通过DOM创建动态的 HTML:

JavaScript 能够改变页面中的所有 HTML 元素
JavaScript 能够改变页面中的所有 HTML 属性
JavaScript 能够改变页面中的所有 CSS 样式
JavaScript 能够对页面中的所有事件做出反应


DOM操作操作的是标签 而一个html页面上的标签有很多
  1.先学如何查找标签
  2.再学DOM操作标签
"""
```

```
DOM操作需要用关键字document起手
"""
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <div id="d1">div
    <div>div>div
    </div>
        <p class="c1">div>p
        <span>
            div>p>span
        </span>
        </p>
        <p>div>p</p>
    </div>

    <div>
        div+div
    </div>
</body>
</html>
```

查找标签

- 直接查找(必须要掌握)

```
"""
id查找
类查找
标签查找
"""
# 注意三个方法的返回值是不一样的
document.getElementById('d1')
<div id="d1">...</div>

document.getElementsByClassName('c1')
HTMLCollection [p.c1]0: p.c1length: 1__proto__: HTMLCollection

document.getElementsByTagName('div')
HTMLCollection(3) [div#d1, div, div, d1: div#d1]
```

```
HTMLCollection(3) [div#d1, div, div, d1: div#d1]
```

```
let divEle = document.getElementsByTagName('div')[1]
divEle
<div>•div>div•</div>•
"""
当你用变量名指代标签对象的时候 一般情况下都推荐你书写成
xxxEle
  divEle
  aEle
  pEle

"""
```

- 间接查找(熟悉)

```
let pEle = document.getElementsByClassName('c1')[0]  # 注意是否需要索引取值

pEle.parentElement  # 拿父节点
<div id="d1">"div
    "<div>div>div</div><p class="c1">…
</p><p>div>p</p></div>
pEle.parentElement.parentElement
<body>…</body>
pEle.parentElement.parentElement.parentElement
<html lang="en"><head>…</head><body>…</body></html>
pEle.parentElement.parentElement.parentElement.parentElement
null

let divEle = document.getElementById('d1')
divEle.children  # 获取所有的子标签
divEle.children[0]
<div>div>div</div>

divEle.firstElementChild
<div>div>div</div>

divEle.lastElementChild
<p>div>p</p>

divEle.nextElementSibling  # 同级别下面第一个
<div>div下面div</div>

divEle.previousElementSibling  # 同级别上面第一个
<div>div上面的div</div>
```

节点操作

```
"""
通过DOM操作动态的创建img标签
并且给标签加属性
最后将标签添加到文本中
"""
let imgEle = document.createElement('img')  # 创建标签

imgEle.src = '111.png'  # 给标签设置默认的属性
"111.png"
imgEle

imgEle.username = 'jason'  # 自定义的属性没办法点的方式直接设置
"jason"
imgEle
<img src="111.png">
imgEle.setAttribute('username','jason')  # 既可以设置自定义的属性也可以设置默认的书写
undefined
imgEle
<img src="111.png" username="jason">
imgEle.setAttribute('title','一张图片')
imgEle
<img src="111.png" username="jason" title="一张图片">

let divEle = document.getElementById('d1')
undefined
divEle.appendChild(imgEle)  # 标签内部添加元素(尾部追加)
<img src="111.png" username="jason" title="一张图片">
"""
创建a标签
设置属性
设置文本
添加到标签内部
  添加到指定的标签的上面
"""
let aEle = document.createElement('a')

aEle
<a></a>
aEle.href = 'https://www.mzitu.com/'
"https://www.mzitu.com/"
aEle
<a href="https://www.mzitu.com/"></a>

aEle.innerText = '点我有你好看!'  # 给标签设置文本内容
"点我有你好看!"
```

```
aEle.innerText = '点我有你好看!'  # 给标签设置文本内容
"点我有你好看!"
aEle
<a href="https://www.mzitu.com/">点我有你好看!</a>
let divEle = document.getElementById('d1')
undefined
let pEle = document.getElementById('d2')
undefined
divEle.insertBefore(aEle,pEle)  # 添加标签内容指定位置添加
<a href="https://www.mzitu.com/">点我有你好看!</a>
```

```
"""
额外补充
  appendChild()
    removeChild()
    replaceChild()


  setAttribute()  设置属性
    getAttribute()  获取属性
    removeAttribute()  移除属性
"""
# innerText与innerHTML
divEle.innerText  # 获取标签内部所有的文本
"div 点我有你好看!
div>p
div>span"

divEle.innerHTML  # 内部文本和标签都拿到
"div
        <a href="https://www.mzitu.com/">点我有你好看!</a><p
id="d2">div&gt;p</p>
        <span>div&gt;span</span>
    "

divEle.innerText = 'heiheihei'
"heiheihei"
divEle.innerHTML = 'hahahaha'
"hahahaha"

divEle.innerText = '<h1>heiheihei</h1>'  # 不识别html标签
"<h1>heiheihei</h1>"
divEle.innerHTML = '<h1>hahahaha</h1>'  # 识别html标签
"<h1>hahahaha</h1>"
```

获取值操作

```
# 获取用户数据标签内部的数据
let seEle = document.getElementById('d2')
seEle.value
"111"
seEle.value
"333"

let inputEle = document.getElementById('d1')
inputEle.value
""
```

```
# 如何获取用户上传的文件数据
let fileEle = document.getElementById('d3')
fileEle.value  # 无法获取到文件数据
"C:\fakepath\02_测试路由.png"

fileEle.files
FileList {0: File, length: 1}0: File {name: "02_测试路由.png",
lastModified: 1557043082000, lastModifiedDate: Sun May 05 2019
15:58:02 GMT+0800 (中国标准时间), webkitRelativePath: "", size: 29580,
…}length: 1__proto__: FileList

fileEle.files[0]  # 获取文件数据
File {name: "02_测试路由.png", lastModified: 1557043082000,
lastModifiedDate: Sun May 05 2019 15:58:02 GMT+0800 (中国标准时间),
webkitRelativePath: "", size: 29580, …}
```

class、css操作

```
let divEle = document.getElementById('d1')
undefined
divEle.classList  # 获取标签所有的类属性
DOMTokenList(3) ["c1", "bg_red", "bg_green", value: "c1 bg_red
bg_green"]


divEle.classList.remove('bg_red')  # 移除某个类属性
undefined

divEle.classList.add('bg_red')  # 添加类属性
undefined
divEle.classList.contains('c1')  # 验证是否包含某个类属性
true
divEle.classList.contains('c2')
false

divEle.classList.toggle('bg_red')  # 有则删除无则添加
false
divEle.classList.toggle('bg_red')
true
```

```
# DOM操作操作标签样式 统一先用style起手
let pEle = document.getElementsByTagName('p')[0]
undefined
pEle.style.color = 'red'
"red"
pEle.style.fontSize = '28px'
"28px"
pEle.style.backgroundColor = 'yellow'
"yellow"
pEle.style.border = '3px solid red'
"3px solid red"
```

事件

```
"""
达到某个事先设定的条件 自动触发的动作
"""
# 绑定事件的两种方式
<button onclick="func1()">点我</button>
<button id="d1">点我</button>

<script>
    // 第一种绑定事件的方式
    function func1() {
        alert(111)
    }
    // 第二种
    let btnEle = document.getElementById('d1');
    btnEle.onclick = function () {
        alert(222)
    }
</script>
```

原生js事件绑定

我们直接写几个案例，看懂即可

- 开关灯案例

```
<div id="d1" class="c1 bg_red bg_green"></div>
    <button id="d2">变色</button>

    <script>
        let btnEle = document.getElementById('d2')
        let divEle = document.getElementById('d1')
        btnEle.onclick = function () {  // 绑定点击事件
            // 动态的修改div标签的类属性
            divEle.classList.toggle('bg_red')
        }
    </script>
```

- input框获取焦点失去焦点案例

```
<input type="text" value="老板 去吗?" id="d1">

<script>
    let iEle = document.getElementById('d1')
    // 获取焦点事件
    iEle.onfocus = function () {
        // 将input框内部值去除
        iEle.value = ''
        //  点value就是获取   等号赋值就是设置
    }
    // 失去焦点事件
    iEle.onblur = function () {
        // 给input标签重写赋值
        iEle.value = '没钱 不去!'
    }
</script>
```

- 实时展示当前时间

```
<input type="text" id="d1" style="display: block;height: 50px;width: 200px">
<button id="d2">开始</button>
<button id="d3">结束</button>

<script>
    // 先定义一个全局存储定时器的变量
    let t = null
    let inputEle = document.getElementById('d1')
    let startBtnEle = document.getElementById('d2')
    let endBtnEle = document.getElementById('d3')
    // 1 访问页面之后 将访问的时间展示到input框中
    // 2 动态展示当前时间
    // 3 页面上加两个按钮 一个开始 一个结束
    function showTime() {
        let currentTime = new Date();
        inputEle.value = currentTime.toLocaleString()
    }

    startBtnEle.onclick = function () {
        // 限制定时器只能开一个
        if(!t){
            t = setInterval(showTime,1000)  // 每点击一次就会开设一个定时器 而t只指代最后一个
        }
    }
    endBtnEle.onclick = function () {
        clearInterval(t)
        // 还应该将t重置为空
        t = null
    }
</script>
```

·省市联动

```
<body>
<select name="" id="d1">
    <option value="--CHOOSE--" disabled selected>--请选择--</option>
</select>
<select name="" id="d2"></select>
<script>
    let proEle = document.getElementById('d1')
    let cityEle = document.getElementById('d2')
    date = {
        "河北":["廊坊","邯郸"],
        "北京":["朝阳区","四合院"],
        "上海":["红灯区","新区"]

    };
    for (let Province in date) {
        let opEle = document.createElement('option')
        opEle.innerText = Province
        opEle.value = Province
        proEle.appendChild(opEle)
    }
    proEle.onchange = function () {
        let currentPro = proEle.value
        let currentCityList = date[currentPro]
        cityEle.innerHTML = ''
        for (let i=0;i<currentCityList.length;i++){
            let currentCity = currentCityList[i]
            let opEle = document.createElement('option')
            opEle.innerText = currentCity
            cityEle.appendChild(opEle)
        }
    }

</script>

</body>
```