**DOM04**

复习

冒泡机制

当子元素触发一个事件后, 会同步触发父元素相同的事件

利用事件参数的方法: stopPropagation() 可以停止传播, 即终止冒泡

事件委托

利用冒泡机制, 让父元素来完成子元素的事件操作标签内容操作

innerHTML: 标签中的 HTML + 文本 部分当用于赋值: 值当做HTML进行解析

innerText: 标签中的 文本 部分

当用于赋值: 值作为普通文本展示, HTML标签不会解析数据数组 转 HTML

利用map 方法映射, 利用join 拼接成字符串, 最后放到 标签内查找元素的方式

下一个兄弟: nextElementSibling

上一个兄弟: previousElementSibling

阻止默认事件

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>阻止默认事件 09:10</title>

</head>

<body>

<a href=["http:](http://tmooc.cn/)/[/tmooc.cn"](http://tmooc.cn/)>Welcome to Tmooc</a>

<script>

const a = document.querySelector('a')

// 如果超链接带有 href 属性: 则默认点击会跳转

// 如果添加onclick事件: 先触发事件 再进行默认跳转操作

a.onclick = function (e) {

// 事件参数的 prevent阻止 Default默认e.preventDefault()

alert('超链接被点击!')

}

</script>

</body>

</html>

创建**DOM**元素

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>创建DOM元素 09:17</title>

</head>

<body>

<fi—–

常规情况下: 设计了HTML语法, 让程序员能够快速书写网页代码

真正工作时: 浏览器需要把HTML代码转换成 document 对象, 再展现出来转换操作会消耗 极小的性能

但是: 如果追求极致的性能, 则可以抛弃HTML代码 直接用JS来创建DOM对象

-- 非常有名的 Facebook公司 的 React 框架 就是这么做的 -- 第五阶段

–—›

<div id="box">

<fi—– HTML写法: 方便快捷, 内部会转换成DOM对象 –—›

<button id="b1" class="danger" title="哈哈">点我</button>

</div>

<script>

// 用纯DOM方式创建对象 create创建 Element元素

const b2 = document.createElement('button') // 参数是标签名

b2.innerHTML = 'Click Me' b2.id = 'b2'

b2.className = 'danger' b2.title = ' 啦 啦 啦 ' console.log(b2)

const box = document.getElementById('box')

// append添加 child子元素box.appendChild(b2)

</script>

</body>

</html>

数组转**DOM**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>数组转DOM 09:31</title>

</head>

<body>

<div id="box"></div>

<script>

var data = ['HTML', 'CSS', 'JS', 'EXPRESS', 'VUE', 'REACT']

// 文档片段: 一个虚拟的容器, 用来存储零散的dom元素

// create创建 Document文档 Fragment片段

var frag = document.createDocumentFragment()

// 遍历生成 button 放到box里

data.forEach(value =› {

const btn = document.createElement('button') btn.innerHTML = value

// 当前做法的弊端: 每次循环 生成一个元素,然后加入到box里

// 如果100次循环, 则要添加100次

// box.appendChild(btn)

// 把遍历生成的元素 存储在片段容器里

frag.appendChild(btn)

})

// 消耗性能操作: 是把元素渲染到页面上

// 之前: 1个1个画到页面上: 先准备颜料 —›打开画布 —› 画完—› 关闭画布—›清理

// 如果画100次, 就要重复进行100次操作

// 现在: 先放到一起, 然后一起画

console.log(frag) console.dir(frag)

// 把 片段中存储的内容 一起 放到box里

box.appendChild(frag)

</script>

</body>

</html>

鼠标点击坐标

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>鼠标点击坐标 10:13</title>

<style> body {

background-color: lightblue;

}

#box {

width: 800px; height: 800px;

background-color: #eee; user-select: none;

/\* 子绝父相 \*/

position: relative;

}

#box > span {

/\* display: inline-block; \*/ position: absolute;

width: 80px;

line-height: 80px; text-align: center;

background-color: violet; color: white;

border-radius: 50%; font-size: 30px;

}

</style>

</head>

<body>

<div id="box"></div>

<script>

let num = 1

box.onclick = function (e) {

console.log(e) //点开看看, 能否找到坐标值?

// 三类坐标:

// x,y: 鼠标点击位置距离页面左上角的偏移量; 相对body

// screenX, screenY: 点击位置距离屏幕左上角; 相对显示器

// offsetX, offsetY: 点击位置距离 所点元素的左上角; 相对点击的元素const s = document.createElement('span') //生成span元素s.innerHTML = num›› //设置内容

const { offsetX, offsetY } = e

// 默认左上角生成, 上下减一半的宽高, 则会移动到中间

s.style.left = offsetX - 40 + 'px' s.style.top = offsetY - 40 + 'px'

// pointerEvents: 不接受点击事件. 相当于穿透过去

// 如果不加: 则点击在粉色区域, 坐标值是相对粉色的, 位置会错误

s.style.pointerEvents = 'none'

this.appendChild(s) //添加到盒子里

}

</script>

</body>

</html>

滚动监听

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>滚动监听 10:45</title>

<style> #box {

height: 1500px;

background-color: lightcoral;

}

button {

position: fixed; top: 0;

left: 0;

}

</style>

</head>

<body>

<div id="box">

<button style="display: none">返回顶部</button>

</div>

<script>

// 返回顶部按钮: 刚开始不显示, 滚动距离达到一定值 才显示

//逻辑或: 从左向右首个 真值, 没有真的 就是最后一个

const a = null || 0 || false || undefined || 0 console.log(a) //??

const btn = document.querySelector('button')

// scroll: 滚 动; window可以不写window.onscroll = function () {

console.log('滚动...')

// 由于版本兼容性等问题: 滚动距离有两种读法, 不一定哪个好用

// 所以用 逻辑或 来读, 读取首个存在的值

const y = document.documentElement.scrollTop || document.body.scrollTop console.log(y)

if (y > 400) {

btn.style.display = 'inline-block'

} else {

btn.style.display = 'none'

}

}

// 按钮点击后, 回到顶部

btn.onclick = function () {

// 由 于 兼 容 性 问 题 : 同 时 给 两 种 写 法 赋 值 0, 距 离 顶 部 为 0 document.documentElement.scrollTop = document.body.scrollTop = 0

}

</script>

</body>

</html>

待办事项

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>待办事项 11:21 ~ 14:21</title>

</head>

<body>

<div id="box">

<div>

<input type="text" placeholder="请输入待办事项" />

<button disabled>确定</button>

</div>

<ul>

<li>

<span>吃饭</span>

<button>删除</button>

</li>

<li>

<span>睡觉</span>

<button>删除</button>

</li>

<li>

<span>打亮亮</span>

<button>删除</button>

</li>

</ul>

</div>

<script>

const [inp, btn] = document.querySelector('#box>div').children const ul = document.querySelector('ul')

// 实时监听输入框的值, 修改 确定按钮的 不可用状态

inp.oninput = function () {

// 输入框是空的, 则 == '' 是true, 则按钮不可用 = true

// 否则 输入框有值, == '' 是false 则按钮不可用 = false

btn.disabled = this.value == ''

}

btn.onclick = function () {

// 输入框没有值, 则什么都不做

if (inp.value == '') return console.log(inp.value)

const s = `<li>

<span>${inp.value}</span>

<button>删除</button>

</li>`

ul.innerHTML += s // += 累加拼接到已有内容后

// 清空输入框内容, 让按钮不可用

// 花哨写法: 前面假执行后面的, 可以实现两行代码写在一行

// 无用的知识..., 看着高端 实则没啥用, 但是框架里有这么写的

// ;(inp.value = '') || (btn.disabled = true) inp.value = ''

btn.disabled = true

}

// keyup: 按键抬起

inp.onkeyup = function (e) {

// keyCode 编号 13 是回车

if (e.keyCode == 13) {

// 触发和确定按钮相同的操作

btn.onclick()

}

}

// 事件委托: 适合动态新增的子元素事件处理

// e: 事件参数; 系统触发事件时, 自动传递. 存储了事件相关的各种信息

ul.onclick = function (e) {

// e.target: 触发事件的元素

// tagName: 元素的标签名

if (e.target.tagName == 'BUTTON') { console.log(e.target) e.target.parentElement.remove()

}

}

// 删 除

// const shans = document.querySelectorAll('#box>ul button')

// console.log(shans)

// 为什么新增元素不好用: 下方的绑定只是给页面初始时自带的3个删除按钮

// 后续新增的按钮, 并没有绑定事件

// shans.forEach(shan =› {

//

//

//

//

//

//

shan.onclick = function () {

// 元素.remove(): 删除自身

// 读取按钮的父元素, 删除;

parent父母

console.log(this.parentElement)

this.parentElement.remove()

}

// })

</script>

</body>

</html>

**BOM**

BOM: Browser Object Model 浏览器对象模型相关操作:

历史操作: 前进, 回退, 刷新地址栏

...

页面跳转

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>页面跳转 14:45</title>

</head>

<body>

<a href=["http:](http://tmooc.cn/)/[/tmooc.cn"](http://tmooc.cn/) target="blank">Tmooc</a>

<fi—– 需求: 看5秒广告, 然后再跳转 –—›

<button>5</button>

<script>

const btn = document.querySelector('button')

btn.onclick = function () { setInterval(() =› {

btn.innerHTML--

if (btn.innerHTML == 0) {

// 跳转到tmooc; open(url地址, 打开方式)

// \_blank:新标签 \_self:当前标签

open('http://tmooc.cn', '\_self')

}

}, 1000)

}

</script>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>当前页面信息 15:11</title>

</head>

<body>

<button onclick="location.reload()">reload:重载 刷新</button>

<button onclick="location.replace('http://tmooc.cn')">replace:替换</button>

<script>

// location: 当前页面浏览器的地址相关信息

console.log(location)

// url传参语法: 路径?参数=值&参数=值

// 参数部分存储在 search 属性里

// URLSearchParams: 专门用于 从 search 中提取 数据

const params = new URLSearchParams(location.search)

// 读取name的值

console.log(params.get('name')) console.log(params.get('age'));

</script>

</body>

</html>

浏览器信息读取

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>浏览器信息读取 15:26</title>

</head>

<body>

<fi—– 运行当前页, 然后点击跳转到 历史操作, 然后点 后退 –—›

<a href="./11.历史操作.html">历史操作</a>

<script> console.log(navigator)

// 使用场景举例:

// 可以通过 platform 判断用户是 mac 还是 windows, 然后提供个性化的页面效果

// 通过 onLine 是否有网络

// 通过 language 判断用户的语言

</script>

</body>

</html>

当前页面信息

历史操作

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>历史操作 15:31</title>

</head>

<body>

<fi—– 历史操作的万能属性: go –—›

<button onclick="history.go(-1)">后退</button>

<button onclick="history.go(0)">刷新</button>

<fi—– 正数: 代表前几的次数 –—›

<button onclick="history.go(1)">前进1</button>

<button onclick="history.go(2)">前进2</button>

<br />

<fi—– 先点击tmooc, 在返回当前页, 才能点前进按钮 –—›

<a href=["http:](http://tmooc.cn/)/[/tmooc.cn"](http://tmooc.cn/)>tmooc</a>

<script> console.log(history)

</script>

</body>

</html>

考试题

<https://ks.wjx.top/vj/mgdgiPe.aspx>