█索引

addEventListener的第三个参数 （事件绑定的参数以及阻止冒泡）

JS存储及序列化（sessionStorage，localStorage, 序列化）

用||来引用对象中不确定的元素

三元表达式

sort排序以及给对象中的数组按照某个元素的值进行排序

█addEventListener的第三个参数

●一句话理解，如果给嵌套的元素绑定同一个事件，那么第三个参数决定是捕获还是冒泡的执行（也就是决定了事件的执行顺序）

不写第三个参数则默认为false，冒泡执行，如下例子：

<!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>

</head>

<body>

    <h1>测试</h1>

    <div id="d">

        <ul id="u">

            <li id="l">点点点</li>

        </ul>

    </div>

    <script>

        function dian (event) {

            var nn = event.currentTarget.tagName

            console.log(nn)

        }

        var d = document.getElementById('d')

        var u = document.getElementById('u')

        var l = document.getElementById('l')

        // 但三个参数不写，默认为false，即冒泡执行

        d.addEventListener('click', dian)

        u.addEventListener('click', dian)

        l.addEventListener('click', dian)

        // 输出：

// LI

// UL

// DIV

    </script>

</body>

</html>

第三个参数为true时，则捕获执行，如下例：

<!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>

</head>

<body>

    <h1>测试</h1>

    <div id="d">

        <ul id="u">

            <li id="l">点点点</li>

        </ul>

    </div>

    <script>

        function dian (event) {

            var nn = event.currentTarget.tagName

            console.log(nn)

        }

        var d = document.getElementById('d')

        var u = document.getElementById('u')

        var l = document.getElementById('l')

        // 第三个参数为true时，则捕获执行

        d.addEventListener('click', dian, true)

        u.addEventListener('click', dian, true)

        l.addEventListener('click', dian, true)

        // 输出：

        // DIV

        // UL

        // LI

    </script>

</body>

</html>

参考网址：

<https://my.oschina.net/u/1454562/blog/205010>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="target-densitydpi=device-dpi, width=device-width, initial-scale=1, user-scalable=no, minimum-scale=1.0, maximum-scale=1.0">

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

<title>Document</title>

</head>

<body>

<div id='d'>

<p id='p'>

<span id='s'>Click Me!</span>

</p>

</div>

<script>

var div = document.getElementById('d');

var p = document.getElementById('p');

var span = document.getElementById('s');

function onClickFn (event) {

var tagName = event.currentTarget.tagName;

var phase = event.eventPhase;

console.log(tagName, phase);

}

div.addEventListener('click', onClickFn, false);

p.addEventListener('click', onClickFn, false);

// 如果是false

// 点击Click Me!后会输出

// P 3

// DIV 3

// 如果是true

// 点击Click Me!后会输出

// DIV 1

// P 1

// 综上，一个事件的传递过程包含三个阶段，分别称为：

// 1 捕获阶段，2 目标阶段，3 冒泡阶段

// 由此，addEventListener的第三个参数设置为true和false的区别已经非常清晰了：

// true表示该元素在事件的“捕获阶段”（由外往内传递时）响应事件；

// false表示该元素在事件的“冒泡阶段”（由内向外传递时）响应时间。

</script>

</body>

</html>

●阻止冒泡

event.stopPropagation()

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="target-densitydpi=device-dpi, width=device-width, initial-scale=1, user-scalable=no, minimum-scale=1.0, maximum-scale=1.0">

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

<title>Document</title>

</head>

<body>

<div id='d'>

<p id='p'>

<span id='s'>Click Me!</span>

</p>

</div>

<script>

var div = document.getElementById('d');

var p = document.getElementById('p');

var span = document.getElementById('s');

function onClickFn (event) {

var tagName = event.currentTarget.tagName;

var phase = event.eventPhase;

console.log(tagName, phase);

}

function onClickFn2 (event) {

var tagName = event.currentTarget.tagName;

var phase = event.eventPhase;

console.log(tagName, phase);

event.stopPropagation(); //阻止冒泡

}

div.addEventListener('click', onClickFn, false);

p.addEventListener('click', onClickFn2, false);

// 点击Click Me!后

// 只会输出

// P 3

// 而div的绑定事件被阻止了

</script>

</body>

</html>

█JS存储及序列化

●存储基础

<!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>

</head>

<body>

    <h1>测试</h1>

    <script>

        // JS的两种存储方法

        // sessionStorage是人如其名,只针对当前session(会话)有效,关闭标签页即失效;

        // localStorage则不然,即使关闭了标签页甚至浏览器,依然存在,下次打开页面时,依然可以直接使用,

        // 直到清除浏览器缓存后,localStorage的内容才会被清理掉;

        //根据key获取对应的值;

        // window.sessionStorage.getItem(key);

        // window.localStorage.getItem(key);

        //新增key-value,若key已存在,则更新value;

        // window.sessionStorage.setItem(key,value);

        // window.localStorage.setItem(key,value);

        //根据key移除对应的值

        // window.sessionStorage.removeItem(key);

        // window.localStorage.removeItem(key);

        //移除全部key-value

        // window.sessionStorage.clear();

        // window.localStorage.clear();

        //根据索引获取对应key

        // window.sessionStorage.key(index);

        // window.localStorage.key(index);

        window.sessionStorage.setItem('ws1', '这是sessionStorage储存的信息ws1')

        console.log(window.sessionStorage.getItem('ws1'))

        window.localStorage.setItem('ws2', '这是localStorage储存的信息ws2')

        console.log(window.localStorage.getItem('ws2'))

    </script>

</body>

</html>

●序列化基础

<!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>

</head>

<body>

    <h1>测试</h1>

    <script>

        // 序列化（将对象转化为字符串）

        var a = {

            '姓名': 'Tom',

            '年龄': '22'

        }

        var b = JSON.stringify(a)

        console.log(a)

        console.log(typeof a)

        console.log(b)

        console.log(typeof b)

        // 输出如下：

        // {姓名: "Tom", 年龄: "22"}

        // object

        // {"姓名":"Tom","年龄":"22"}

        // string

        // 反序列化（将字符串转化为对象）

        var c = JSON.parse(b)

        console.log(c)

        console.log(typeof c)

        // 输出如下：

        // {姓名: "Tom", 年龄: "22"}

        // object

    </script>

</body>

</html>

●在使用js储存时，只能储存字符串。因此如果要储存对象，需要先进行序列化

<!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>

</head>

<body>

    <h1>测试</h1>

    <script>

        var a = {

            '姓名': 'Tom',

            '年龄': '22'

        }

        console.log(a.姓名) //Tom

        // 使用JS储存只能储存字符串，所以要储存对象，需要先序列化成字符串再储存

        // 在使用的时候需要先反序列化成对象，再使用

        var b = JSON.stringify(a) //把a序列化成字符串b

        window.sessionStorage.setItem('tom', b) //把b存到sessionStorage

        var c = window.sessionStorage.getItem('tom') //从sessionStorage里拿到储存的字符串

        var d = JSON.parse(c) //把字符串反序列化成对象

        console.log(d.年龄) //22

    </script>

</body>

</html>

█用||来引用对象中不确定的元素

var p = {

np: {

nnp: 'haha'

}

}

// 当不确定p里面是否有某个属性时，可以用||

// 从左至右，只要有一个成立，就返回这个

var r = p.np.nnp || p.np || p

console.log(r) //haha

var r2 = p.np.nnnp || p.np || p

console.log(r2) //{ nnp: 'haha' }

█三元表达式

var flag = true

// 如果flag为true则执行冒号前面的代码。否则执行冒号后面的代码

flag ? console.log(111) : console.log(222)

// 输出：111

function say1 () {

console.log('这是1')

}

function say2 () {

console.log('这是2')

}

var flag2 = false

flag2 ? say1() : say2()

// 输出：这是2

█sort排序以及给对象中的数组按照某个元素的值进行排序

●Sort默认是对字符串排序，根据数字，字母，汉字来排序

<script>

        var a = [100, 'ab', 3, 0, 2, 'bc', '我', '啊', '祖国', 'c', 1000]

        // sort默认是字符串排序，即数字到字母到汉字拼音顺序

        console.log(a.sort()) //[0, 100, 1000, 2, 3, "ab", "bc", "c", "啊", "我", "祖国"]

    </script>

如果要给数字安装大小排序，则需要给sort传一个参数

<script>

        // 如果需要从数学意义上对数字的大小进行排序，就要给sort传参，参数是以下函数

        // 以下两种函数写法都可以

        // function compare (num1, num2) {

        //  return num1 - num2

        // }

        function compare(value1, value2) {

            if (value1 < value2) {

                return -1;

            } else if (value1 > value2) {

                return 1;

            } else {

                return 0;

            }

        }

        var a = [100, 3, 0, -1, 2, -88, 1000]

        console.log(a.sort(compare)) //[-88, -1, 0, 2, 3, 100, 1000]

    </script>

●给对象中的数组按照某个元素的值进行排序

参考网址：

<https://www.jb51.net/article/152582.htm>

<!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>

</head>

<body>

    <h1>测试</h1>

    <div id="d">

        <ul id="u">

            <li id="l">点点点</li>

        </ul>

    </div>

    <script>

        var students = [

            {

                name: 'tom',

                age: '22'

            },

            {

                name: 'jerry',

                age: 18

            },

            {

                name: 'torry',

                age: '21'

            }

        ]

        // 定义比较函数，将要排序的自动作为参数prop传入

        // 字段如果不是数字（一般接口返回的是字符串），则自动转成数字

        var compare = function (prop) {

            return function (obj1, obj2) {

                var val1 = obj1[prop];

                var val2 = obj2[prop];

                if (!isNaN(Number(val1)) && !isNaN(Number(val2))) {

                val1 = Number(val1);

                val2 = Number(val2);

            }

            if (val1 < val2) {

                return -1;

                } else if (val1 > val2) {

                return 1;

                } else {

                return 0;

                }

            }

        }

        // 使用sort排序，将定义好的compare函数作为参数

        students.sort(compare('age'))

        console.log(JSON.stringify(students))

        // 输出：

        // [{"name":"jerry","age":"18"},{"name":"torry","age":"21"},{"name":"tom","age":"22"}]

    </script>

</body>

</html>