JavaScript基础

标签　　空

新增属性

JavaScript（简称“JS”） 是一种具有函数优先的轻量级，解释型或即时编译型的编程语言。虽然它是作为开发Web页面的脚本语言而出名，但是它也被用到了很多非浏览器环境中，JavaScript 基于原型编程、多范式的动态脚本语言，并且支持面向对象、命令式、声明式、函数式编程范式。

html 中使用 js

原生html页面支持对js代码的嵌套。

标签内部：

```HTML
<button onclick="alert('弹出信息!!!');">确认</button>
```

script标签：在head中调用时执行，在body中页面加载后执行

```HTML
<!-- 直接写在标签内部会在刷新页面后直接执行，如果内部操作h5标签 -->
<script type="text/javascript">
    alert('弹出信息!!!')
</script>
```

直接写在标签内部会在刷新页面后直接执行，因此通常是标签内部与script标签搭配使用：

```HTML
<script type="text/javascript">
    // 定义函数,可在js中被调用（注意，不能先调用后定义，而是从上往下执行）
    function info(){
        alert('弹出信息!!!')
    }
</script>

<button onclick="info()">确认</button>
```

引入js文件：

```HTML
<head>
  <script type="text/javascript"  src="js/test.js"></script>
</head>
```

注意事项

使用 js 获取 dom 元素时，要确保 dom 节点已被渲染，因此在 head 标签中获取节点会为 null，正确方式如下：

```HTML
<body >
    <div id="imgDiv"></div>
    <script>
      var imgDiv =  document.getElementById('imgDiv');
    </script>
</body>
```

html 节点操作

document 是整个树对象，element 是具体的节点，即树枝。不过 document 也代表根节点，因此很多 element 的方法也可使用。

Window 对象

window 在 js 表示浏览器对象，js 可以通过它与浏览器交互。

获取 Dom 树：

```JavaScript
window.document.getElementById("header");
```

获取浏览器窗口尺寸：

```JavaScript
window.innerHeight
window.innerWidth
```

获取屏幕信息：window.screen

```JavaScript
screen.availWidth
screen.availHeight
```

获取地址栏信息：window.location

```JavaScript
localhost.protocol // 协议
localhost.hostname // 域名
localhost.port     // 端口
localhost.pathname // url路径
localhost.href     // url链接

```

Document 树

html 文档是由众多父子标签组成的 xml 格式文件，document 则代表这个 xml 文件的对象，使用 dom 树可以在 js 获取或设置标签的信息。

节点 element 操作

获取节点

```JavaScript
let element = document.getElementById("test");          //根据id属性值获取
let element = document.getElementsByTagName("div");     //根据标签名获取
let element = document.getElementsByClassName("test"); //根据class属性值获取

let element = document.querySelector(".test > div");   // 使用css选择器查找一个
let element = document.querySelectorAll(".test > div");// 使用css选择器查找全部

let element = element1.parentNode;                     //获取父节点
let elements = element1.childNodes;                    //获取子节点数组
let element = element1.firstChild;                     //获取第一个子节点
let elements = element1.attributes;                    //获取节点属性数组

let element = document.head;                           // 获取head节点
let element = documnet.body;                           // 获取body节点
```

添加节点

```JavaScript
element.appendChild(element1);                          //插入子节点
element.insertBefore(newElement,oldElement);              //在子节点前插入节点

let element = document.createElement("div");          //创建节点
let element = element1.cloneNode(true);                                 //深克隆，复制节点（含子节点）
let element = element1.cloneNode(false);                                //浅克隆，复制节点（不含子节点）
```

删除节点

```JavaScript
element.removeChild(element1);        //删除节点
```

修改节点

```JavaScript
element.replaceChild(newElement,oldElement);   //替换节点
```

比较节点

```JavaScript
document.isEqualNode(document1);        //比较两个节点是否相等，相等返回true
document.contains(document1);           //判断节点是否是自身子节点
```

值操作

获取

```JavaScript
var body = document.innerHTML;          //获取节点标签HTML内容
var title = document.title;             // 获取文档标题
var value = document.nodeValue;         //元素节点值为空、文本节点为文本，属性节点为属性值
var html = document.documentElement;    //获取全部文档内容
var body = document.body;               //获取文档主体
```

设置

```JavaScript
element.innerHTML="<div>hello</div>";  //修改节点内容
element.createTextNode("hello");       //创建文本节点，实际上就加了节点内容
element.write("hello");               // 写内容，可以是任何内容
```

修改

```JavaScript
element.style.样式名="样式值";          //修改css样式
```

文本转节点：通过设置节点值可将文本变成节点。

```JavaScript
let element =document.createElement("div");
element.innerHTML = htmlCodeTxt
```

属性操作

```JavaScript
let attr = element.createAttribute("class");        //创建属性
attr.value = "test";                                //属性赋值

let body = element.getAttribute("class");           //获取属性值
body.setAttribute("class","test");                      //设置属性值
```

事件操作

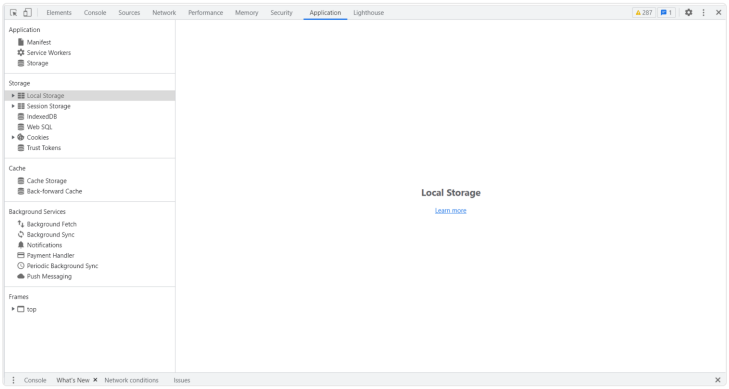
```JavaScript
// 进入页面
element.onload = function(){ }
// 离开页面
element.onunload = function(){ }
```

```
// 鼠标按下
element.onmousedown = function (){ }
// 鼠标松开
element.onmouseup =function(){ }
// 点击事件
element.onclick = function(){ }
// 鼠标停留
element.onmouseover = function(){ }
// 鼠标离开
element.onmouseout = function(){ }

// 输入事件
element.onchange = function(){ }
```

事件大全：https://www.w3school.com.cn/jsref/dom_obj_event.asp

前端应用



localStorage 与 sessionStorage

- localStorage 用于长久保存整个网站的数据，保存的数据没有过期时间，直到手动去除。
- sessionStorage 用于临时保存同一窗口(或标签页)的数据，在关闭窗口或标签页之后将会删除这些数据。

操作：localStorage 与 sessionStorage 操作一致，下面以 localStorage 举例。

JavaScript

```
// 添加、修改键
window.localStorage.setItem("username","wenge")

// 查询键
let username = window.localStorage.getItem("username")

// 移除键
window.localStorage.removeItem("username")

// 清空键
window.localStorage.clear("username")
```

Cookies

Cookie 是客户端存储的一些键-值对数据，存储于在电脑本地的文本文件中。

JavaScript

```
// 设置cookie
document.cookie="id=1;username=wenge;"

// 获取cookie
let cookies = document.cookie
cookies.split(";").forEach(cookie =>{
  let kv = cookie.replace(" ","").split("=")
  console.log('key=',kv[0],'value=',kv[1])
});
```

webSQL

打开数据库

```JavaScript
const openDB = () =>{
  return window.openDatabase("managerDB","1.0","存储",2 * 1024 * 1024,()=>{})
}
```

创建表

```JavaScript
const createTable = (db,sql) => {
  db.transaction((ctx,res) =>{
    ctx.executeSql(sql,[],()=>{},(ctx,error)=>{console.log(sql.split(" ")[0] + 'error:',error)})
  })
}
```

增删改：（和创建表一样）

```JavaScript
export function updateData(db,sql) {
  db.transaction((ctx,res) =>{
    ctx.executeSql(sql,[],()=>{},(ctx,error)=>{console.log(sql.split(" ")[0] + 'error:',error)})
  })
}
```

查询：使用同步方法，防止异步获取不到数据

```JavaScript
export const  selectData = async (db,sql) => {
  let data = [];
  await db.transaction(async (ctx,res) =>{
    await ctx.executeSql(sql,[],(ctx,res)=>{
      for (let i in res.rows) {
        data.push(res.rows[i])
      }
    },(ctx,error)=>{console.log(sql.split(" ")[0] + 'error:',error)})
  })
  return data
}
```

调用查询：

```JavaScript
async loadClassData(){
    let listData =  await dbs.selectData(this.db,`select * from ${this.classTable}`)
    this.classData = listData
},
```

async 的返回值会被使用 Promise 包含，因此获取其返回值的方法必须也为 async。

静态资源调用

由于浏览器同源策略，本地 js 无法直接获取静态文件，因此可以通过一些手段绕过此限制。

jsonp

json 内容存放到 test.js

```JavaScript
let p = {
  "id":1,
  "name":"张三"
}
```

使用<script>引入：

```HTML
<script src = "./test.js"></script>
<script>
  function run(){
    console.log(p)

```

```
6     }
    </script>
```

按键事件

- onkeydown：按键按下时；
- onkeypress：按键按下后；
- onkeyup：按键松开时；

事件处理方法：

```
function keyDown(e){

    // 组合此处键可以添加多个条件
    if(e.keyCode == 13){
      alert("按下enter");
    }

}
```

监听事件：

```
// 全局监听
document.onkeydown =  keyDown;

// 局部监听
<div onkeydown="keyDown()"/>
```

键值：

键盘按键键值

字母和数字键的键码值(keyCode)							
按键	键码	按键	键码	按键	键码	按键	键码
A	65	J	74	S	83	1	49
B	66	K	75	T	84	2	50
C	67	L	76	U	85	3	51
D	68	M	77	V	86	4	52
E	69	N	78	W	87	5	53
F	70	O	79	X	88	6	54
G	71	P	80	Y	89	7	55
H	72	Q	81	Z	90	8	56
I	73	R	82	0	48	9	57

数字键盘上的键的键码值(keyCode)				功能键键码值(keyCode)			
按键	键码	按键	键码	按键	键码	按键	键码
0	96	8	104	F1	112	F7	118
1	97	9	105	F2	113	F8	119
2	98	*	106	F3	114	F9	120
3	99	+	107	F4	115	F10	121
4	100	Enter	108	F5	116	F11	122
5	101	-	109	F6	117	F12	123
6	102	.	110				
7	103	/	111				

控制键键码值(keyCode)							
按键	键码	按键	键码	按键	键码	按键	键码
BackSpace	8	Esc	27	Right Arrow	39	-_	189
Tab	9	Spacebar	32	Dw Arrow	40	.>	190
Clear	12	Page Up	33	Insert	45	/?	191
Enter	13	Page Down	34	Delete	46	`~	192
Shift	16	End	35	Num Lock	144	[{	219
Control	17	Home	36	;:	186	|	220
Alt	18	Left Arrow	37	=+	187	]}	221
Cape Lock	20	Up Arrow	38	,<	188	'"	222

多媒体键码值(keyCode)							
按键	键码	按键	键码	按键	键码	按键	键码
音量加	175						
音量减	174						
停止	179						
静音	173						
浏览器	172						
邮件	180						
搜索	170						
收藏							

鼠标按键键值

常数名称	十六进制值	十进制值	对应按键
VK_LBUTTON	01	1	鼠标的左键
VK_RBUTTON	02	2	鼠标的右键
VK-CANCEL	03	3	Ctrl+Break (通常不需要处理)
VK_MBUTTON	04	4	鼠标的中键（三按键鼠标）

数据值操作

数组操作

声明数组：

JavaScript

```
// 声明一维数组
let list = []

// 声明二维数组
let list = [[]]
let list = new Array(50).fill()
let list = new Array(50).fill('0') // 默认值0



let list = {1,2,3,4,5,6,7}

// 检查所有元素是否符合条件
let check = list.every(item => item < 10);  // true

// 将符合条件的元素存在新数组
let list2 = list.filter(item => item > 5);  // {6,7}
```

元素操作：

JavaScript

```
list.unshift(100)               // 头部插入
list.push(100)                  // 尾部添加
list.splice(0,2,100,200...)     // 插入：在位置删除2个元素，并插入多个元素

list.pop()                      // 删除末尾

let list = list1..concat(list2) // 合并数组
```

Map 操作

遍历 map:

JavaScript

```
for(let key in map){
    console.log("value",map[key])
}
```

List 转 Map

JavaScript

```
let map = {id: 20,name:'zhangsan'}
```

```
let list = []
for (const key in map) {
  const val = map[key]
  if (val){
    list.push(`${key}=${val}`)
  }
}
```

数值处理

浮点型保留小数位：

```
// 四舍五入保留2位小数

2.446242342.toFixed(2);

// 截取保留2位小数
Math.floor(15.7784514000 * 100) / 100
```

字符串操作

字符串替换：移除所有a、b

```JavaScript
// replaceAll：es2021语法
"xxbxaxxbxa".replaceAll('a','').replaceAll('b','')

// 使用正则：/[ab]/g
"xxbxaxxbxa".replace(/[ab]/g,'')

```

分割字符串

```JavaScript
let list = "aaa,bbb,ccc".split(',');
```

截取字符串

```JavaScript
// 按长度取：start、len
let str = "aaa,bbb,ccc".substr(4,3);    // bbb

// 按位置取: [start,end)
let str = "aaa,bbb.ccc".substring(5,8); // bbb
```

日期值操作

格式化日期：

```JavaScript
// 格式化日期: str->str
formatDate(val) {
    if (val == null) {
      return ''
    }
    const date = new Date(val)
    const year = date.getFullYear()
    let month = date.getMonth() + 1
    if (month < 10) {
      month = '0' + month
    }
    let day = date.getDate()
    if (day < 10) {
      day = '0' + day
    }
    let hours = date.getHours()
    let minutes = date.getMinutes()
    let seconds = date.getSeconds()
    return year + '-' + month + '-' + day + ' ' + hours + ':' + minutes + ':' + seconds
},
```

变量定义

js中的变量值可以是常量，也可以是函数

```JavaScript
// 值变量
let value = 1

// 函数变量
let func = function run(){ }
let func = () => { }
```

ES6 语法

导出 | export、export default

每个页面都是单独的一个模块，其它页面无法直接引用它的属性，可以通过export来导出属性使页面属性公开。

export： 直接导出变量（其它页面导入时，需要使用{ }）

```JavaScript
export var id = 1;

export function text(){ }
```

export default： 导出对象形式，每个页面只能有1个

```JavaScript
export default {

    user: function(){
      return{
        id: 1
      }
    },

    text: function(){ }, // 简写text(){}

}
```

导入 | import、require

导入其它页面中被导出的属性和方法。

import：静态加载，在编译时就进行加载了。

```JavaScript
// 导入具体属性
import {id} from './test.js'


// 导入对象
import test from './test.js'
```

require：动态加载，在运行到此处时才会加载。