1. 定时器(属于BOM部分)

1>基本定时器

(1)格式 --> setInterval( function () {}, time );

(2)扩展：

##每隔time时间执行一次function(执行无限次), 且time只识别一次

##定时器会返回一个唯一标识

##可用一个变量接收定时器唯一标识, 用于清除定时器(clearInterval) -->

var timer = setInterval(function () {}, 1000) clearInterval(timer)

2>一次性定时器

(1)格式 --> steTimeout(function() {}, time)

(2)扩展：

##每隔time时间执行一次function(只执行一次)

##清除定时器clearTimeout()

1. 脚本化css
2. 读写CSS属性

(1)domEle.style.prop

##可读写行间样式,没有兼容性问题

(2)注意格式

##float这样的关键字属性应在前面加css -->float --> cssFloat

##复合属性必须拆解，组合单词采用小驼峰式写法

##写入值必须是字符串格式

1. 查询计算样式

(1)window.getComputedStyle(ele,null)[prop]

(2)注意事项：

##计算样式只读

##返回的计算样式的值都是绝对值，没有相对单位

##IE8及其以下的版本不兼容

##()里第二个参数可以获取伪元素的样式

(3)IE计算样式

##ele.currentStyle[prop]

++计算样式只读

++返回的计算样式值不是经过转换的绝对值

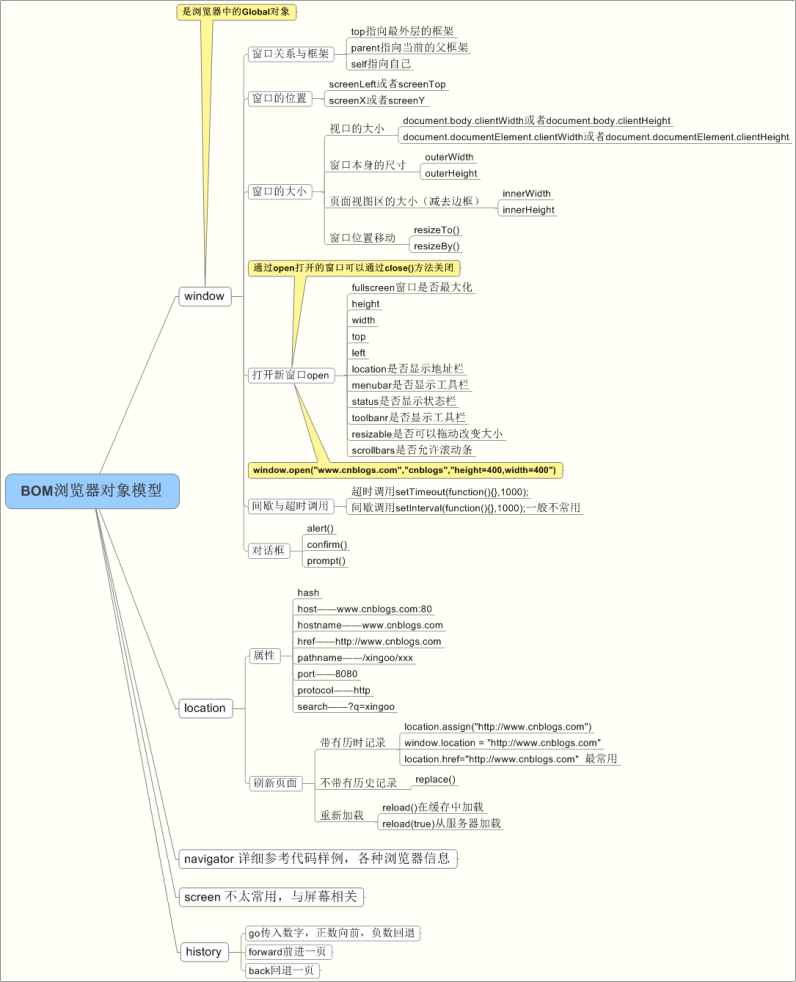
++IE独有的属性

1. BOM部分

1>基本概念

是浏览器对象模型, 可以与浏览器窗口进行互动的对象结构, BOM由多个对象结构构成, 其中代表浏览器窗口的window对象是BOM的顶层对象, 其他对象是该对象的子对象

2>思维导图



3>BOM对象

(1)window对象, 是JS的最顶层对象, 其他的BOM对象都是window对象的属性

##详细说名

window对象双重角色

++是javascript访问浏览器窗口的一个接口,

++又是ECMscript规定的Global对象

##常见用法

++弹框类方法, 前面省略window

--alert(“提示信息”)

--confirm(“确认信息")

--prompt(“弹出输入框”)

--open(“url地址”, “打开方式(可以是-self或-black)”, “新窗口的大小”)

注意：如果url为空，则默认打开一个空白页面，如果打开方式为空，则默认为新窗口方式打开页面。

返回值为：返回新打开窗口的window对象

--close()关闭当前页面

注意：存在兼容问题 谷歌(默认直接关闭)IE(询问用户)

++其他方法

--window.moveTo() 移动当前窗口

--window.resizeTo() 调整当前窗口尺寸

++定时器方法(更详细内容参考 一. 定时器)

--setTimeout(函数, 时间)只执行一次

--setInterval(函数, 时间)执行无数次

(2)document对象, 文档对象

(3)location对象, 浏览器当前URL信息

##详细说明

window.location对象：用于获得当前页面的地址 (URL)，并把浏览器重定向到新的页面。在编写时可不使用 window 这个前缀。

##方法

--location.href = “url地址”

--hash 返回#号后面的字符串，不包含散列，则返回空字符串

--host 返回服务器名称和端口号

--pathname 返回目录和文件名。 /project/test.html

--search 返回？号后面的所有值

--port 返回URL中的指定的端口号，如URL中不包含端口号返回空字符串

--portocol 返回页面使用的协议。 http:或https:

(4)navigator对象, 浏览器本身信息

##详细说明

window.navigator 对象包含有关访问者浏览器的信息。在编写时可不使用 window 这个前缀。

##方法

--navigator.platform：操作系统类型；

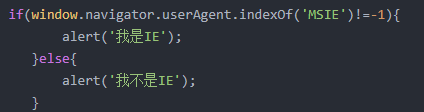
--navigator.userAgent：浏览器设定的User-Agent字符串。

--navigator.appName：浏览器名称

--navigator.appVersion：浏览器版本

--navigator.language：浏览器设置的语言；

--userAgent是最常用的属性，用来完成浏览器判断(配合ondexOf) -->



(4)screen对象, 客户端屏幕信息

##属性

--window.screen 对象包含有关用户屏幕的信息。

--screen.availWidth 属性

返回访问者屏幕的宽度，以像素计，减去界面特性，比如窗口任务栏。

-- screen.availHeight 属性

返回访问者屏幕的高度，以像素计，减去界面特性，比如窗口任务栏。

(5)history对象, 浏览器访问历史信息

##方法

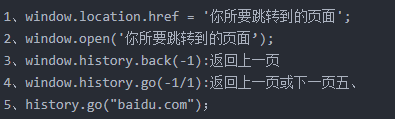
--window.history 对象包含浏览器的历史。

为了保护用户隐私，对 JavaScript 访问该对象的方法做出了限制。

--history.back() - 加载历史列表中的前一个 URL。返回上一页。

--history.forward() - 加载历史列表中的下一个 URL。返回下一页。

1. 综合页面跳转方法 -->



4>window对象常用方法补充

(1)window.innerHeight(innerWidth)：

表示可见区域的高度(宽度)如果底部(右侧)有滚动条, 会把底部(右侧)滚动条的高度(宽度)计算进去，此方法适用于标准浏览器及IE9+

(2)document.documentElement.clientHeight(clientWidth)：

表示可见区域的高度(宽度)不会把滚动条的高度宽度计算进去, 此方法适用于标准浏览器及低版本IE标准模式。

(3)document.documentElement.offsetHeight(offsetWidth)：

整个文档的高度(宽度)offsetWidth获得的值与clientWidth一样，而offsetHeight获得的是整个文档的高度。

(4)document.documentElement.scrollHeight(scrollWidth)：

获得网页正文全文高(宽), 包含边线宽度。

(5)document.body.clientHeight(clientWidth)：

整个文档的高度(宽度)clientWidth获得的值等于

document.documentElement.offsetWidth剪去body边线宽度，而clientHeight获得的是整个body的高度减去边线宽度。

(6)document.body.offsetHeight(offsetWidth)：

获得的值等于document.body.clientHeight（clientWidth）加上body的边线(border)宽度，即这两个值取到到的包含边线宽度。

(7)document.body.scrollHeight(scrollWidth)：

获得网页正文全文高和宽, 包含边线宽度。

(8)document.body.scrollTop（scrollLeft）：

浏览器窗口顶部与文档顶部之间的距离(浏览器窗口与文档左侧之间的距离)，也就是滚动条滚动的距离。

(9)window.pageYOffset(pageXOffset)：

浏览器窗口顶部与文档顶部之间的距离(浏览器窗口与文档左侧之间的距离), 也就是滚动条滚动的距离。

(10)window.screen.height（width）：

屏幕分辨率的高(宽)

(11)window.screen.availHeight（availWidth）：

屏幕可用工作区高(宽)。

(12)以上方法区别

##offsetTop、clientTop、scrollTop分别代表什么

--offsetTop:距离上方或上层控件的位置, 整型, 单位像素。

--clientTop:元素的内容区距离元素边缘的距离, 即元素的边线宽度。

--scrollTop:元素窗口顶部与元素顶部之间的距离, 即元素内部滚动条滚动的距离

四．DOM部分

1>选中dom

(1).getElementsById();通过ID名选择出元素 不区分id的大小写

(2).getElementsByTagName();通过标签名选择出元素（表单元素）

(3).getElementsByName();通过name选择出元素

(4).gtElementsByClassName();ie9一下不能用 通过class民选择出元素

下面的是静态的是副本，上面的是动态的，实时的。

(5).querySelector("里面css怎么选这里怎么选");

(6).querySelectorAll("里面css怎么选这里怎么选");选择出一组

2>遍历节点树(注意是节点)

(1)parentNode 父节点(最顶端是#document)；

(2)childNodes[第几个节点，可不写]--子节点们（所有的子节点）

(3)firstChild-第一个子节点

(4)lastChild-最后一个子节点

(5)nextSibling-后一个兄弟节点

(6)previousSibing 前一个兄弟节点

3>节点类型

(1)元素节点 -----1

(2)属性节点-----2

(3)文本节点-----3

(4)注释节点----8

(5)document节点-------9

(6)DocumentFragment-----11

4>基于元素点的遍历(除了chidren其他的都是IE9一下不兼容的)

(1)parentElement（顶端是html）返回单前元素父元素节点

(2)chidern 返回当前元素的子元素节点

(3)firstElementChild 返回第一个元素节点

(4)lastElementChild 返回最后一个元素节点

(5)nextElementSibling后一个元素节点

(6)previousElementSibling前一个元素节点

5>节点的四个属性

(1)nodeName

##元素的标签名称, 以大写的形式表示，只读

##属性节点显示属性名称, 文本节点显示#text, 文档节点显示#document

(2)nodeValue 只有文本节点和注释节点属性节点，有(text,comment)可读可写

(3)nodeType 返回节点的类型(>3), 只可读

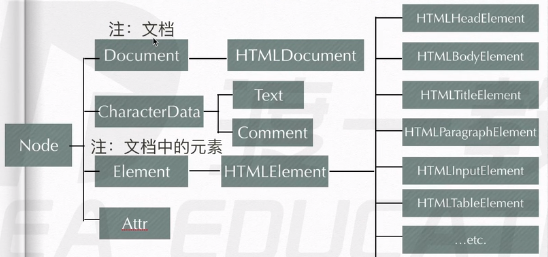
(4)attributes属性节点的集合

6>检测方法

(1)hasChildNodes();方法 检测有没有子节点有turn没有false。

7>DOM结构树(继承关系)

(1)基本结构



补充 document继承HTMLDocument和XMLDocument(不常用)继承Document

Node的\_\_proto\_\_(继承)为EventTarget的\_\_proto\_\_(继承)为Object

(2)继承关系实例演示

##getElementsById定义在Document.prototype上 ,

即Element节点上不能使用

##getElementByName定义在了HTMLDocument.prototype上,

即不能在xml document和Element上使用

##getElementByTagName定义在Document.prototype上和Element上,

即都能使用

##HTMLDocument定义了一些常用的属性, body和head分别指其文档内的标签

--document.body表示<body>标签

--document.head表示<head>标签

##Document.prototype上定义了documentElement属性, 指代文档的根标签

--document.documentElement表示<html>标签

9>操作html

(1)增(实在js手里拿着不会显示在页面, 需要显示在页面需先插入appendChild())

##.createElement("标签名");创建元素节点（标签）

##.createTextNode("");创建节点(文本)

##.creatComment("")注释节点

##.createDocumentFragment()；创建文案碎片节点

(2)插

##appendChild(可以理解成push) 页面上已经有的剪切操作 如，

##div.appendChild(span); 原来div和span是兄弟结构的变成父子结构了。

##.insertBefore(a, b); 读insert a Before b (a在b前面) 要选中他的父元素 ##ParentNode.insertBefore(a, b);

(3)删

##.parent.removeChild();选中父节点删除括号里面的子标签 不需要""

##child.remove(); 删掉选中的节点。

(4)改

##parentNode.replaceChild(new(新节点), origin(老节点))new代替origin

10>操作html和文本的方法

(1)ele.innerHTML = (覆盖)

(2)ele.innerHTML += (在内容后面加东西)

(3)ele.innertext = 文本 (覆盖) 小心有span标签

11>操作节点的属性方法

(1)setAttribute("(属性名)class","on(属性值)") -->

class="on" 添加行间属性

(2)getAttribute("标签") 取出行间属性

12>操作节点样式

(1)HTML元素.style.样式属性＝"值"

(2)HTML元素.className="样式名称(定义好的class)"

13>获取元素位置的一些属性

(1)offsetLeft/Top 返回当前元素左边界到它上级元素的左边界的距离，只读属性

(2)offsetHeight/Width 返回元素的高度

(3)offsetParent 返回元素的偏移容器，即对最近的动态定位的包含元素的引用

(4)scrollTop/Left 返回匹配元素的滚动条的垂直位置

(5)clientWidth/Height 返回元素的可见宽度

14>获取坐标的一些属性

(1)offsetX/Y 相对于带有定位的父盒子的x, y坐标

(2)clientX/Y 点击位置距离当前body可视区域的x, y坐标

(3)pageX/Y 对于整个页面来说, 包括了被卷去的body部分的长度

(4)screenX/Y 点击位置距离当前电脑屏幕的x, y坐标

五．事件部分

1>绑定事件方式

(1)ele.onxxx = function(event) {}

##一个元素同一个事件只能绑定一次

##解除事件：ele.onxxx = null/false

##this指向dom元素本身

(2)obj.addEventListener("type", fn, false);

## IE9下不兼容

##可以一个事件绑定多个处理函数，同一个元素执行同一个函数只能执行一次

##ele.removeListener(type, fn, false)里面的全部值要是同一个

++fn需要是一个函数的引用不然找不到上面说说的同一个

##this指向dom元素本身

(3)obj.attachEvent("on+ type", fn)

##IE独有的

##可以一个事件绑定多个处理函数，同一个元素执行同一个函数可以执行多次

##和ele.removeListener(type, fn, false)一样全部要是同一个

##this指向window

2>事件处理模型(一个对象的一个事件类型只能出现一个模型(要么冒泡要么捕获))

(1)事件冒泡

##结构上(非视觉上)嵌套关系的元素，会存在事件冒泡的功能，即同一事件上，自子元素冒泡向父元素。(自底向上)

(2)事件捕获(不常规模型)(只有谷歌浏览器实现了)

##结构上(非视觉上)嵌套关系的元素，会存在事件捕获的功能，即同一事件上，自父元素捕获至子元素(事件源元素)。(自顶向下)

(3)注意事项：

##事件触发顺序, 先捕获后冒泡, 注意区分事件绑定的执行顺序

##事件的执行顺序是先绑定先执行(测试捕获和冒泡的先后顺序需注意绑定执行

顺序)

##focus, blur, change, submit, reset, select等事件不冒泡

3>取消事件冒泡

(1)w3c标准event.stopPropagation();IE9以下不支持

(2)event.cancelBubble = true; ie独有的

4>阻止默认事件

(1)return false；(句柄的方式阻止 只能取消句柄绑定的默认事件)

以对象属性的方式注册的事件才生效

(2)event.preventDefault(); W3C标准，IE9一下不兼容

(3)event.returnValue = false; 兼容IE

5>事件对象

event(就是function里面的形参e)非IE存在

IE存在于window.event--->event || window.event

6>事件源对象

(1)event.target 火狐只有这个

(2)event.srcElement IE只有这个

(3)chrome浏览器两个都有

7>鼠标事件

(1)click(敲击事件),

(2)mousedown

(3)mousemove

(4)mouseup

(5)contextmenu(右键出现菜单事件)

(6)mouseover(鼠标放进去出现什么)

(7)mouseout(鼠标出来出现什么)

(8)mouseenter

(7)mouseleave

注意：区分左右键只能用moseup和mousedown事件可以区分 里面的button属性。e.button ==2 右键 e.button == 0 左键

8>键盘事件

(1)keydown,

(2)keyup,

(3)keypress

触发顺序 keydown>keypress>keyup

##keydown和keypress的区别

++keydown能监听所有键盘类事件且charcode值为0(fn不能，fn键是辅助键)

++keypress能监听所有字符类按键charcde为ask码(能区分字母的大小写)

9>文本类事件

(1)input事件(改变就会触发该事件),

(2)focus(聚焦触发),

(3)blur(失去焦点触发),

(4)change(状态位发生改变才触发)(聚焦改变失去焦点才触发)

10>窗体操作类事件(window上的事件)

(1)scroll(滚动条滚动触发该事件),

(2)load(效率最低,等randTree加载绘制完成之后才触发, 常用)

11>总结

(1)事件一般全部都是小写

分割线

1. JSON

1>概念

是一种数据传输格式, 以对象的形式(键值对), 与对象的区别是定义的对象建必须有引号(认为规定) --> { “name”: “zhu”}

2>数据传输常用方法

(1)JSON.stringify(obj) 从一个对象中解析出字符串(传递数据给后端时使用)

-->

使用方法放回结果”{"{"a":"1","b":"2"}"}”

(2)JSON.parse(str) 从一个字符串中解析出JSON对象(前端接收后端数据时使用)

-->

使用方法放回结果Object{a:"1",b:"2"}

1. randerTran绘制页面

1>加载规则

深度优先原则(html --> head -->meta(head里面所有标签) --> body )

2>加载顺序

(1)先解析html(解析完) --> domTree

##解析完毕(DOM节点树的解析完毕)

src里面的资源和href啥的还没有加载出来

##加载完毕

src里面的资源和href啥的加载完毕

##符合加载规则(深度优先原则)

(2)在解析css --> cssTree

--符合加载规则(深度优先原则)

(3)拼接domTree和cssTree形成randerTree, 开始绘制页面

3>randerTree重新绘制(重排reflow)

(1)dom节点的删除, 添加

(2)dom节点的宽高变化, 位置变化, display: none -->block等

(3)offsetWidth, offsetLeft获取宽高位置(重新绘制randerTree获取的数据才是最新的)

4>randerTree不重新绘制, 只重新修改修改的部分(repaint重绘)

修改 如字体颜色, 背景颜色等

1. 异步加载

1>方法

(1)defer异步加载, dom文档解析完毕执行, 代码可以在内部, 只有IE9--可以使 -->



(2)aysnc异步加载, 加载完毕执行, 只能加载外部脚本(js只能是外部引入的模式)

(3)创建script, 插入DOM中, 加载完毕后callback(回调函数) (常用)

执行时, 不阻塞页面(可实现异步加载, 也能实现按需加载)

##基本形式 ->

var script = document.createElement(“script”);

script.src = “加载的文件(异步下载, 并不会执行)”

document.body.appendChild(script) (执行这行才会执行src里面的代码)

##实例 -->



注意：script.src = “demo.js”应放在if循环的后面, 防止电脑加载资源过快, 先加载完成了资源在触发事件

##封装(完整形式) -->



1. 时间线(加载, 解析顺序)
2. 创建Document对象, 开始解析web页面, 解析HTML元素和他们的文本内容后添

加Element对象和Text节点到文档中。这个阶段Document。readyState = "loading"

2>遇到link外部css，创建线程加载，并继续解析文档。

3>遇到script外部js，并且没有设置async ， defer ，浏览器加载，并阻塞，等待js加载完成并执行该脚本，然后继续解析文档

4>遇到script外部js，并且设置有async，defer 浏览器创建线程加载，并继续解析文档，对于async属性的脚本，脚本加载完成后立即执行(异步禁止使用document.write())。

5>遇到img标签等，先正常解析dom结构，然后浏览器异步加载src，并继续解析文档

6>当文档解析完成，document.readyState = "interactive"；

7>文档解析完成后，所有设置有defer的脚本会按照顺序执行。

8>当文档解析完成之后，document对象触发DOMContentLoaded事件，这也标志着程序执行从同步脚本执行阶段，转化为事件驱动阶段

注意：DOMContentLoaded事件只能绑定在addEventListener绑定

9>当所有saync的脚本加载完成并执行后，img等加载完成后，document.readyState = "complete" window对象触发load事件

10>从此，页面以异步响应方式处理用户输入，网络事件等。

11>注意document.write()问题(覆盖里面全部内容)

(1)异步加载

(2)window.load = function () { document.write() }

1. 总结区分

1>$(document).ready(function () {}) 文档解析完毕在执行

2>window.load = function () {} 文档加载完毕在执行

3>原生js模拟$(function () {})需利用DOMContentLoaded事件

或者监听document.readyState状态为interactive

注意：DOMContentLoaded事件只能绑定在addEventListener绑定

4>解析完毕randerTree绘制完毕, 加载完毕randerTree绘制完毕且外部引入的文件全部加载完毕(src里面的资源和href啥的加载完毕, img等全部加载完毕)

分割线

1. \ 转义字符(常用的)

1>\ 把后面一个字符转化成正常文本显示

2>\n 换行显示

3>\r 行结束符(window操作系统一般回车键都是 /r/n)

4>\t 一个缩进

1. 正则表达式(RegExp)
2. 创建正则表达式对象

(1)var reg = /abc(匹配规则)/i属性 (判断字符串是否含有abc)

(2)var reg = new RegExp(“abc(匹配规则)”, “i(属性)”)

1. 属性(修饰符)

(1)i 忽略大小写

(2)g 全局匹配(全部匹配规则的都返回不管是不是一样的)

(3)m 执行多行匹配(一般配合匹配第一个最后一个)

1. 表达式([], ())

(1)区间匹配

var reg = /[1-9], [A-z], [cd]/

匹配第一位1到9, 第二位A到z, 第三位c或者d的字符串

(2)查找指定选项

var reg = /(abc|bcd)[0-9]/g

匹配abc加上0到9或者bcd加上0到9随便一位数的字符串

4>元字符(表达式的简写)

(1)/\w/ 0到9, A到z, \_ /\W/ ^\w(除了\w的其他全部)

(2)/\d/ 0到9 /\D/ 除了\w的其他全部

(3)/\s/ 空白字符(空格符, 制表符(\t), 回车符(\r), 换行符(\n)等) S同上

(4)/\b(加上边界是什么字符)/ 单词边界

5>量词

(1)n+