**js之DOM学习笔记**

**javascript --DOM笔记**

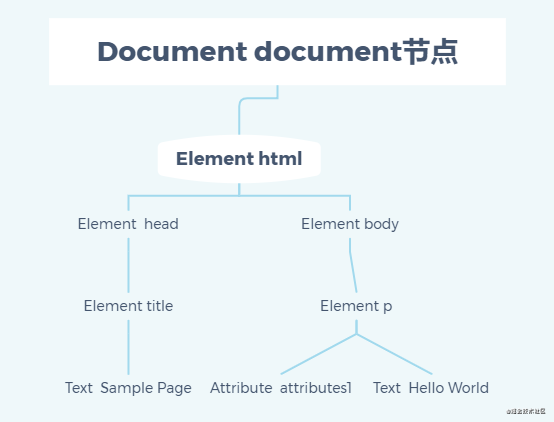
DOM文档对象模型是HTML和XML文档的编程接口

下面是一个标准的html页面

<!DOCTYPE html><html lang="en"><head> <title>Sample Page</title></head><body> <p class='attributes1'>Hello World</p></body></html>

复制代码

它的层级结构如下:



document节点表示每个文档的根节点。

根节点的唯一子节点是元素,称之为文档元素 。它是文档最外层的元素，每个文档只能有一个。

在JavaScript中，所有节点都继承Node类型， 因此所有类型都共享相同的基本属性和方法。

**1.节点**

**1.1 node description**

nodeType nodeName和nodeValue保存着有关节点的信息

nodeType属性表示该节点的类型

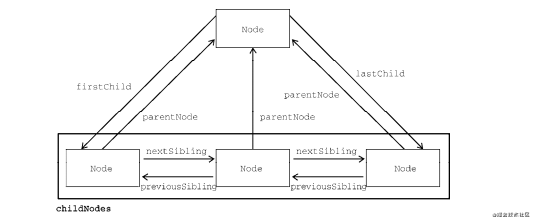
nodeName为元素的标签名称

nodeValue为元素标签的值（通常指的是改标签的内容，但里面的文本不属于这个值 文本属于文本结点 文本结点的nodeValue是文本， 如上面的p元素的子节点 Text的nodeValue是Hello World）（元素的nodeValue通常是null 如p元素）

DOM中的12种节点类型

* Node.ELEMENT\_NODE(1)
* Node.ATTRBUTE\_NODE(2)
* Node.TEXT\_NODE(3)
* Node.CDATA\_SECTION\_NODE(4)
* Node.ENTITY\_REFERENCE\_NODE(5)
* Node.ENTITY\_NODE(6)
* Node.PROCESSING\_INSTRUCTION\_NODE(7)
* Node.COMMENT\_NODE(8)
* Node.DOCUMENT\_NODE(9)
* Node.DOCUMENT\_TYPE\_NODE(10)
* Node.DOCUMENT\_FRAGMENT\_NODE(11)
* Node.NOTATION\_NODE(12)

**1.2 node relationship**



关系类型(节点的属性)：

childNodes 包含一个NodeList的实例（类数组对象）（可通过索引值访问，也可以通过item()方法访问, 具有length属性)

* firstChild 父节点的第一个子节点
* lastChild 父节点的最后一个子节点

如果父节点只有一个子节点 则firstChild和lastChild指向同一个节点; 如果父节点没有子节点, 则firstChild和lastChild都为null

parentNode 父节点

previousSibling 第一个节点的previousSibling属性是null

nextSibling 最后一个节点的nextSibling属性是null

hasChildNodes()方法 用于检测某节点是否存在子节点(返回true, 则说明该节点有一个或多个子节点)

ownerDocument属性 是一个指向代表整个文档的文档节点指针



### 1.3 操纵节点(增替删)

1. 增加节点

appendChild()方法

作用: 用于在childNodes列表末尾添加节点

调用者: 想要添加子节点的节点

参数: 要添加的新的节点的名称

(如果传入文档中已经存在的节点, 则这个节点会从之前的位置被转移到新位置)

返回值: 返回新添加的节点

insertBefore()方法

作用: 向某节点之前插入节点

调用者: 想要添加子节点的节点

参数: 要插入的节点(新的节点)和参照节点

(如果参照节点是null 则insertBefore和appendChild方法的效果一样即插入最后一个结点)

返回值: 返回新插入的节点

1. 替换结点

replaceChild()方法

作用: 替换某个节点

调用者: 上同

参数: 要插入的节点(新的节点)和要被替换的节点

返回值: 被替换的节点

1. 删除节点

removeChild()方法

作用: 移除节点

调用者: 上同

参数: 要被移除的节点

返回值: 被移除的节点

上述四个方法都是用于操纵调用者的子节点 并且被替换和删除的节点原则上来说还是属于该文档所拥有 但文档中已经没有了它们的位置

**1.4 其他方法**

* cloneNode()方法

作用: 复制节点

调用者: 某节点

参数: 布尔参数(表示是否深复制)

传入参数为true 会进行深复制 即复制节点及整个子DOM树

传入参数为false 只会复制调用该方法的节点

返回值: 返回与调用它的节点一模一样的节点

复制返回的节点属于文档所有, 但尚未指定父节点 所以可称为孤儿节点(orphan)

该方法不会复制添加到DOM节点的JavaScript属性, 比如事件处理程序。这个方法只会复制HTML属性以及可选地复制子节点, 除此之外则一概不会复制。

* normalize()方法

作用: 处理文档子树中的文本节点

调用者: 具有文本子节点的节点

参数:无

该方法会检测这个节点的所有后代, 从中搜索上述两种情形(不包含文本的文本节点或文本节点之间互为同胞关系).如果发现空文本节点, 则将其删除; 如果有两个同胞节点是相邻的, 则将其合并为一个文本节点。

**2. Node类型**

**2.1 Document类型**

Document 类型是JavaScript中表示文档节点的类型。 在浏览器中， 文档对象document是HTMLDocument的实例， 表示整个HTML页面。 document是window对象的属性， 因此是一个全局对象。

Document类型的节点特征

* nodeType等于9
* nodeName值为‘#document’
* nodeValue 值为null
* ownerDocument值为null
* 子节点(可以是DocumentType(最多一个)/Element(最多一个)/ProcessingInstruction/comment类型)

上述的层级结构图中 document是Document类型, 其表示整个HTML页面或者其他的XML文档

document对象可用于获取关于页面的信息以及操纵其外观和底层结构

**2.1.1 文档子节点**

子节点(可以是DocumentType(最多一个)/Element(最多一个)/ProcessingInstruction/comment类型)

* documentElement属性

该属性始终指向HTML页面中的元素(也可以通过document.childNodes[0]访问/firstChild/lastChild)

* body属性(document.body)

该属性始终指向HTML页面的 元素(document.body)

* doctype属性

该属性指向文档声明(document.doctype)

**2.1.2 文档信息**

* title属性

指向head元素中的title属性,包含元素中的文本, 通常显示在浏览器窗口或标签页的标题栏

通过该属性可以读写页面的标题

以下三个属性暴露出的信息都可以从请求头中获取

* URL属性

包含当前页面的完整URL(地址栏中的URL)

* domain属性

包含页面的域名(只有domain可以设置;不能给这个属性设置URL不包含的值/属性一旦放松就不能收紧)

解决跨域问题

因为跨源通信存在安全隐患, 所以不同子域的页面间无法通过JavaScript通信.

在页面上把document.domain设置为相同的值, 这些页面就可以访问对方的 JavaScript对象了。

(如: 一个加载自[www.wrox.com](https://link.juejin.cn/?target=www.wrox.com)的页面中包含一个内嵌窗格, 其中页面加载自p2p.wrox.com.由于这两个页面的document.domain包含不同的字符串, 内部和外部页面相互之间不能访问对方的 JavaScript对象. 如果把每个页面的document.domain设置为wrox.com, 那么这两个页面之间就可以通信了)

* referrer属性

包含链接到当前页面的那个页面的URL.(如果当前页面没有来源, 则referrrer属性包含空字符串)

**2.1.3 定位元素**

* getElementById()方法

参数: 节点的id值(大小写敏感)

返回: 要定位的节点(换句话说具有该id的节点)/null(没有带有该id的元素)

调用者: document对象

如果页面中存在多个具有相同id的元素, 则该方法返回在文档中出现的第一个元素

* getElementsByTagName()方法

参数: 要获取元素的标签名

返回: 返回包含零个或多个元素的NodeList(在HTML文档中, 这个方法返回一个HTMLCollection对象 和NodeList类型 可通过索引和item()方法访问某个具体的元素)

(HTMLCollection对象的一个额外的方法 namedItem()可通过标签的name水性取得某一项的引用/当然也可以通过中括号的方式来获取该元素)

(document.getElementsByTagName('img').namedItem('myImg'))

(document.getElementsByTagName('img')['myImg'])

对于HTMLCollection对象来说中括号既可以接收数字索引, 也可以接收字符串索引. 在后台 数字索引会调用item() 字符串索引会调用namedItem()

调用者: document对象

document.getElementsByTagName(\*)可以取得文档的所有元素

* getElementsByName()

参数: 元素name属性的属性值

返回: 返回具有给定name属性的所有元素的HTMLCollection对象(此处的namedItem()方法只会取得第一项, 因为单选按钮的name属性都一样)

通常用于单选按钮

**2.1.4 特殊集合(一些定位属性)**

这些集合也都是HTMLCollection的实例

* document.anchors 包含文档中所有带name属性的a元素
* document.links 包含文档中所有带href属性的a元素
* document.forms 包含文档中所有的form元素(与 document.getElementsByTagName('form')返回的结果相同)
* document.images 包含文档中所有的img元素(与 document.getElementsByTagName('img')返回的结果相同)

**2.1.5 文档写入(4个方法)**

* write()/writeIn()

参数: 一个字符串 可以将这个字符串写入网页中

调用者: document (document.write('字符串'))

区别: writeIn()还会在字符串末尾追加一个换行符(\n)

这两个方法可以用来在页面加载期间向页面中动态添加内容

这两个方法也经常用于动态包含外部资源 对于标签参数的传入需要添加转义字符即为<\/script>

* open()/close()

这两个方法分别用于打开和关闭网页输出流.

**2.2 Element类型**

Element表示XML或HTML元素 对外暴露出访问元素标签名、子节点和属性的能力。

上面层级结构中 html p head body 等都属于Element类型

该类型的特征如下：

* nodeType等于1
* nodeName值为元素的标签名
* nodeValue值为null
* parentNode值为Document或Element对象
* 子节点可以是Element Text Comment ProcessingInstruction CDATASection EntityReference类型

可以通过nodeName或tagName属性来获取元素的标签名(它们返回的值相同)。

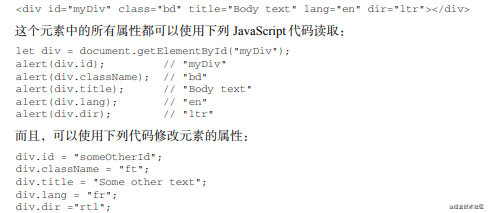
tagName属性返回的元素标签名始终以全大写表示

**2.2.1 HTML元素**

所有HTML元素都通过HTMLElement类型表示, 包括直接实例和间接实例.而HTMLElement直接继承Element并增加了一些属性.(以下为HTMLElement对象的属性用于获取HTML元素属性的信息)

* id 元素在文档中的唯一标识符;(HTML元素的id属性)
* title 包含元素的额外信息, 通常以提示条形式展示;(HTML元素的title属性)
* lang 元素内容的语言代码(一般来说只是标签的属性)
* className 相当于class属性, 用于指定元素的 CSS类(因为class是ES的关键字, 所以不能直接使用这个名字)

以上所有这些都可以用来获取对应的属性值, 也可以用来修改相应的值.



**2.2.2 属性相关**

* getAttribute()

作用: 获取某元素属性

参数: 要获取元素的属性名(和实际的属性名一样 比如传入class而不是className)(属性名称不区分大小写)

不是HTML语言正式属性的自定义属性也可以获取

调用者: 某HTML元素

返回值: 某元素的属性值

如果给定的属性不存在, 则返回null

使用DOM对象的属性来获取的HTML元素的属性和使用getAttribute()方法获取的属性其中有两个不太一样

* + style属性 使用方法获取的返回值为CSS字符串 而通过属性获取的返回值为CSSStyleDeclaration对象
  + 事件处理程序(比如 onclick) 使用方法获取的返回值为字符串形式的源代码 而通过属性获取的返回值为一个JavaScript函数(未指定该属性则返回null)

综上 getAttribute() 通常用于取得自定义属性的值

* setAttribute()

作用: 给HTML元素设置属性值

参数: 要设置的属性名和属性的值

调用者: 某HTML 元素

返回值:

如果属性已经存在, 则会以指定的值替代原来的值; 如果不存在, 则会以指定的值创建该属性。

直接给DOM对象的属性赋值也可以设置元素属性的值(在DOM对象上添加自定义属性, 不会自动让它变成元素的属性)

* removeAttribute()

删除HTML元素的属性, 不单单清楚属性的值, 而是整个属性都会被清楚

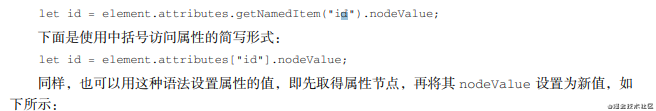
**2.2.3 attributes属性**

Element 类型是唯一使用attributes属性的DOM节点类型。attributes属性包含一个NamedNodeMap实例，类似于NodeList的实时集合。 元素的每个属性都表示为一个Attr节点， 并保存在这个NamedNodeMap对象中。

（就是HTML元素的属性也是一种节点 它对应着DOM对象的attributes属性，其返回值为NamedNodeMap对象；就像HTML元素对应着DOM对象的Element类型一样）

NamedNodeMap对象的方法：

* getNamedItem(name) 返回nodeName属性等于name的节点
* removeNamedItem(name)删除nodeName属性等于name的节点
* setNamedItem(node) 向列表中添加node节点, 以其nodeName为索引
* item(pos) 返回索引位置pos处的节点



一般来说对于HTML元素属性的操作通常使用getAttribute() removeAttribute() 和setAttribute()方法

而attributes属性的应用场景为迭代元素上的所有属性

**2.2.4 创建元素**

document.createElement()方法

参数: 要创建元素的标签名

方法还会把新创建的元素的ownerDocument属性设置为document

元素被创建出来后还需要添加到文档树 才会被浏览器渲染(使用appendChild insertBefore或replaceChild来添加)

**2.2.5 元素后代**

元素的getElementsByTagName()方法

可用来返回当前元素的后代

**2.3 Text类型**

Text节点由Text类型表示, 包含按字面解释的纯文本, 也可能包含转义后的html字符, 但不包含HTML代码

上面层级结构中的文本内容为text节点属于DOM 对象中的Text类型

Text类型的节点的特征:

* nodeType等于3
* nodeName值为'#text'
* nodeName值为节点中包含的文本
* parentNode值为Element对象;
* 不支持子节点

Text节点中包含的文本可以通过nodeValue属性访问, 也可以通过data属性访问, 这两个属性包含相同的值

文本节点暴露的操作文本的方法:

* + appendData(text) 向节点末尾添加文本
  + deleteData(offset, count) 从位置offset开始删除count个字符
  + insertData(offset, text) 在位置offset插入text
  + replaceData(offset, count, text) 用text替换从位置offset到offset+count的文本
  + splitText(offset) 在位置offset将当前文本节点拆分为连个文本节点
  + substringData(offset, count) 提取从位置offset到offset+count的文本

创建文本节点

document.createTextNode()

规范化文本节点

normalize()定义在Node类型上

拆分文本节点

splitText()定义在Text类型上

拆分后 原来的文本节点包含开头到偏移位置前的文本, 新文本节点包含剩下的文本.

这个方法返回新的文本节点, 具有与原来的文本节点相同的parentNode.

**2.4DocumentFragment类型**

在所有节点类型中, DocumentFragment类型是唯一一个在标记中没有对应表示的类型

文档片段的作用是充当其他要被添加到文档的节点的仓库(类似于一个子树要添加到文档树中)

**2.5 Attr类型**

元素数据在DOM中通过Attr类型表示. Attr类型构造函数和原型在所有浏览器中都可以直接访问.

元素数据对应着上面层级结构中的属性节点, 类似于HTML元素的属性和Element类型的attribute属性类似, 只不过这里单独的将其作为一个DOM类型来处理.

Attr节点特征:

* nodeType等于2
* nodeName值为属性名
* nodeValue值为属性值
* parentNode值为null
* 在HTML中不支持子节点
* 在xml中子节点可以是Text或EntityReference

Attr 对象上有3个属性

* + name 和nodeName一样
  + value 和nodeValue一样
  + specified 为布尔值, 表示属性使用的是默认值还是被指定的值

可以使用createAttribute()方法创建新的Attr节点, 参数为属性名

属性尽管是节点 , 但不被认为是DOM文档树的一部分.

Attr节点很少直接被引用, 通常开发者更喜欢使用 getAttribute() removeAttribute() setAttribute()方法操作属性

**3. DOM编程**

* 动态脚本

注意 IE对

* 动态样式

IE浏览器会对标签施加限制, 不允许访问其子节点 可以通过访问其stylecss属性 的CSSText属性来添加css代码

**4. MutationObserver接口**

MutationObserver可以用来观察整个文档、DOM树的一部分, 或某个元素. 此外还可以观察元素的属性 子节点 文本 或者前三者任意组合的变化. VUEX的mutation就和这里的Mutation有相似之处.

**4.1 基本用法**

MutationObserver的实例要通过调用MutationObserver构造函数并传入一个回调函数来创建

let observer = new MutationObserver(() => console.log('DOM has mutated'))

复制代码

1. observe()方法

上面新创建的MutationObserver实例不会关联DOM的任何部分 也就起不到观测的作用,因而要起到观测的作用, 需要将observer与DOM关联起来, 这时就需要使用observer()方法.

参数: 要观察的DOM节点/MutationObserverInit对象(用于控制观察那些方面的变化)(它是一个键值对形式配置选项的字典.)

let observer = new MutationObserver(() => console.log('DOM has mutated'));observer.observe(document.body {attribute: true})

复制代码

执行上述代码后, body元素上任何属性发生变化都会被这个MutationObserver实例发下, 然后就会异步执行注册的回调函数。body元素后代的修改或其他非属性修改都不会出发回调进入任务队列。

1. 回调与MutationRecord

每次回调都会受到一个MutationRecord实例的数组。这个实例包含的信息包括发生了什么变化， 以及DOM的哪一部分受到了影响。因为回调执行之前可能同时发生多个满足观察条件的事件，所以每次执行回调都会传入一个包含按顺序入队的MutationRecord实例的数组

1. disconnect()方法

默认情况下 只要被观察的元素不被垃圾回收， MutationObserver的回调就会相应DOM变化事件，从而被执行。要提前终止执行回调，可以使用disconnect()方法。

调用disconnect()方法, 会停止观察所有目标.

要想让已经加入任务队列的回调执行，可以使用setTimeout()让已经入列的回调执行完毕在再调用disconnect()(宏任务和微任务的执行顺序)

let observer = new MutationObserver(() => {console.log('body attributes changed')})observer.observe(document.body, {attributes: true})document.body.className = 'foo'setTmeout(() => { observer.disconnect(); document.body.className = 'bar'}, 0);//body attributes changed

复制代码

1. 复用MutationObserver

多次调用observe()方法, 可以复用一个MutationObserver对象观察多个不同的目标节点。此时 MutationRecord的target属性可以标识发生变化事件的目标节点。

1. 重用MutationObserver

调用disconnect()方法并不会结束MutationObserver的生命.

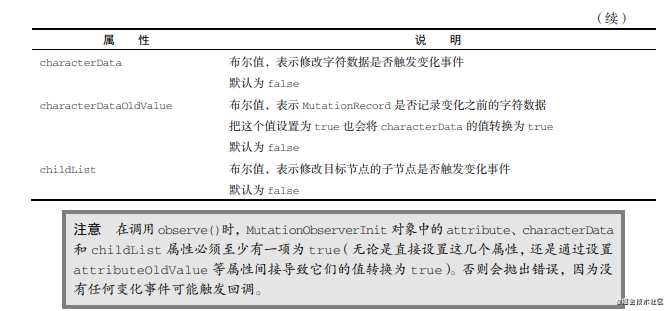
还可以通过重新使用这个观察者, 再将它关联到新的目标节点.



**4.2 MutationObserverInit与观察者范围**

通过observe()方法可以 关联DOM节点和MUtationObserver实例 但这个方法里的第二个参数决定了可以观察的内容和范围 以下为可观察的内容或范围/换句话说 MutationObserverInit对象的属性





注意点:

* 对于子节点的重新排序会报告两次变化事件(尽管调用一个方法即可实现), 因为从技术上会涉及先移除和再添加
* //清空主体document.body.innerHTML = '';let observer = new MutationObserver((mutationRecords) => {console.log(mutationRecords)});document.body.appendChild(document.createElement('div'));document.body.appendChild(document.createElement('span'));observer.observe(document.body, {childList: true});document.body.insertBefore(document.body.lastChild, document.body.firstChild)​

复制代码



* 被观察子树中的节点被移出子树之后仍然能够触发变化事件. 这意味着在子树中的节点离开该子树后, 即使严格来讲该节点已经脱离了原来的子树, 但它仍然会触发变化事件

// 清空主体document.body.innerHTML = '';let observer = new MutationObserver((mutationRecords) => {console.log(mutationRecords)});//创建节点let childRoot = document.createElement('div');let childNode = document.createElement('span');document.body.appendChild(childRoot);childRoot.appendchild(childNode);observer.observe(document.body, {attributes: true, subtree: true});document.body.inserBefore(childNode, childRoot);childNode.setAttribute('foo', 'bar');//移出的节点仍然触发变化事件//[mutationRecords]​

* MutationObserver实例的takeRecords()方法可以清空记录队列, 取出并返回其中的所有MutationRecord实例

该方法在希望断开与观察值目标的联系, 但又希望处理由于调用disconnect()而被抛弃的记录队列中的MutationRecord实例时比较有用

**4.3 性能**

* MutationObserver的引用

MutationObserver实例与目标接单之间的引用关系是非对称的.

MutationObserver 拥有对要观察的目标节点的弱引用.因为是弱引用，所以不会妨碍垃圾回收程序回收目标节点.

然而, 目标节点却拥有对 MutationObserver 的强引用.如果目标节点从DOM中被移除，随后被垃圾回收，则关联的MutationObserver也会被垃圾回收.

* MutationRecord的引用

记录队列中的每个MutationRecord 实例至少包含对已有DOM节点的一个引用。如果变化是 childList 类型，则会包含多个节点的引用。

记录队列和回调处理的默认行为是耗尽这个队列，处理每个 MutationRecord，然后让它们超出作用域并被垃圾回收。

有时候可能需要保存某个观察者的完整变化记录。保存这些 MutationRecord 实例，也就会保存它们引用的节点，因而会妨碍这些节点被回收。

如果需要尽快地释放内存，建议从每个 MutationRecord 中抽取出最有用的信息，然后保存到一个新对象中，最后抛弃 MutationRecord。

文章分类