

22-浏览器DOM：你知道HTML的节点有哪几种吗？

你好，我是winter。

今天我们进入浏览器API的学习, 这一节课，我们来学习一下DOM API。

DOM API是最早被设计出来的一批API，也是用途最广的API，所以早年的技术社区，常常用DOM来泛指浏览器中所有的API。不过今天这里我们要介绍的DOM，指的就是狭义的文档对象模型。

DOM API介绍

首先我们先来讲一讲什么叫做文档对象模型。

顾名思义，文档对象模型是用来描述文档，这里的文档，是特指HTML文档（也用于XML文档，但是本课不讨论XML）。同时它又是一个“对象模型”，这意味着它使用的是对象这样的概念来描述HTML文档。

说起HTML文档，这是大家最熟悉的东西了，我们都知道，HTML文档是一个由标签嵌套而成的树形结构，因此，DOM也是使用树形的对象模型来描述一个HTML文档。

DOM API大致会包含4个部分。

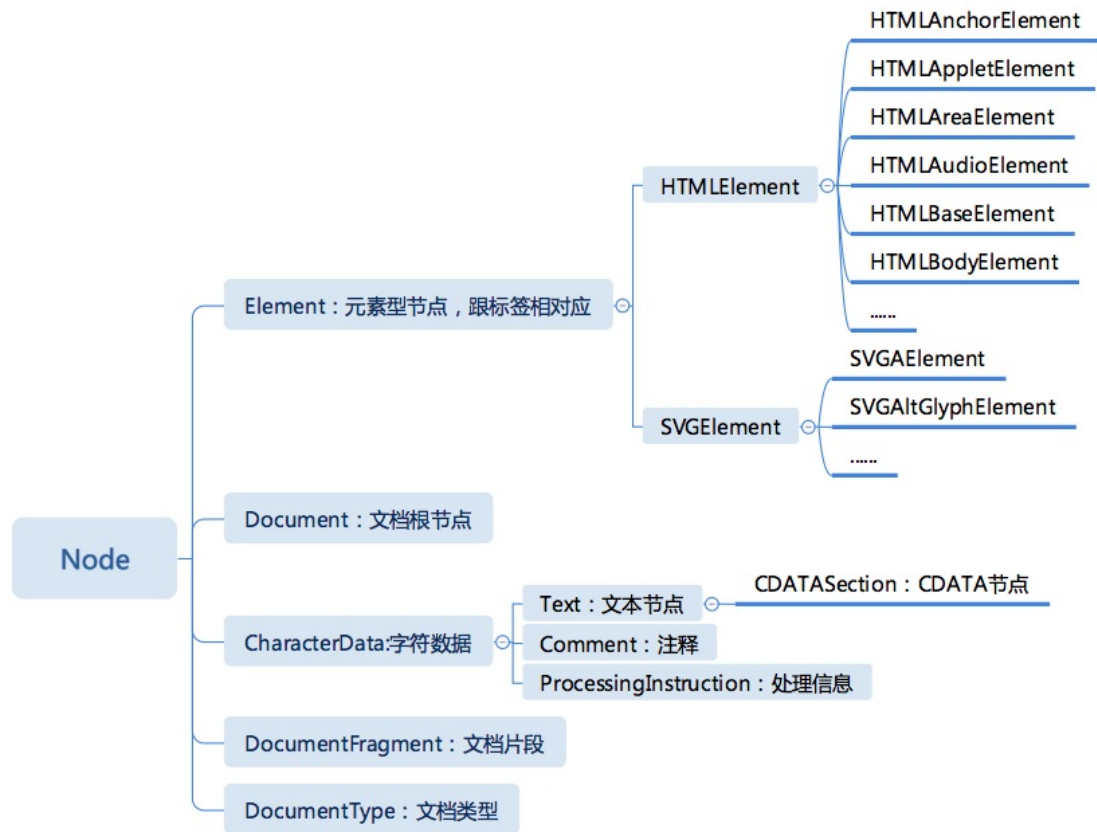
- 节点：DOM树形结构中的节点相关API。
- 事件：触发和监听事件相关API。
- Range：操作文字范围相关API。
- 遍历：遍历DOM需要的API。

事件相关API和事件模型，我们会用单独的课程讲解，所以我们本篇文章重点会为你介绍节点和遍历相关API。

DOM API 数量很多，我希望给你提供一个理解DOM API设计的思路，避免单靠机械的方式去死记硬背。

节点

DOM的树形结构所有的节点有统一的接口Node，我们按照继承关系，给你介绍一下节点的类型。



在这些节点中，除了Document和DocumentFrangment，都有与之对应的HTML写法，我们可以看一下。

```
Element: <tagname>...</tagname>
Text: text
Comment: <!-- comments -->
DocumentType: <!Doctype html>
ProcessingInstruction: <?a 1?>
```

我们在编写HTML代码并且运行后，就会在内存中得到这样一棵DOM树，HTML的写法会被转化成对应的文档模型，而我们则可以通过JavaScript等语言去访问这个文档模型。

这里我们每天都需要用到，要重点掌握的是：Document、Element、Text节点。

DocumentFragment也非常有用，它常常被用来高性能地批量添加节点。因为Comment、DocumentType和ProcessingInstruction很少需要运行时去修改和操作，所以有所了解即可。

Node

Node是DOM树继承关系的根节点，它定义了DOM节点在DOM树上的操作，首先，Node提供了一组属性，来表示它在DOM树中的关系，它们是：

- parentNode
- childNodes
- firstChild
- lastChild

- nextSibling
- previousSibling

从命名上，我们可以很清晰地看出，这一组属性提供了前、后、父、子关系，有了这几个属性，我们可以很方便地根据相对位置获取元素。当然，Node中也提供了操作DOM树的API，主要有下面几种。

- appendChild
- insertBefore
- removeChild
- replaceChild

这个命名跟上面一样，我们基本可以知道API的作用。这几个API的设计可以说是饱受诟病。其中最主要的批评是它不对称——只有before，没有after，而jQuery等框架都对其做了补充。

实际上，appendChild和insertBefore的这个设计，是一个“最小原则”的设计，这两个API是满足插入任意位置的必要API，而insertAfter，则可以由这两个API实现出来。

我个人其实不太喜欢这个设计，对我而言，insertAt(pos) 更符合审美一些。当然，不论喜不喜欢，这个标准已经确定，我们还是必须要掌握它。

这里从设计的角度还想要谈一点，那就是，所有这几个修改型的API，全都是在父元素上操作的，比如我们要想实现“删除一个元素的上一个元素”，必须要先用parentNode获取其父元素。

这样的设计是符合面向对象的基本原则的。还记得我们在JavaScript对象部分讲的对象基本特征吗？“拥有哪些子元素”是父元素的一种状态，所以修改状态，应该是父元素的行为。这个设计我认为是DOM API中好的部分。

到此为止，Node提供的API已经可以很方便（大概吧）地对树进行增、删、遍历等操作了。

除此之外，Node还提供了一些高级API，我们来认识一下它们。

- compareDocumentPosition 是一个用于比较两个节点中关系的函数。
- contains 检查一个节点是否包含另一个节点的函数。
- isEqualNode 检查两个节点是否完全相同。
- isSameNode 检查两个节点是否是同一个节点，实际上在JavaScript中可以用“===”。
- cloneNode 复制一个节点，如果传入参数true，则会连同子元素做深拷贝。

DOM标准规定了节点必须从文档的create方法创建出来，不能够使用原生的JavaScript的new运算。于是document对象有这些方法。

- createElement
- createTextNode
- createCDATASection

- createComment
- createProcessingInstruction
- createDocumentFragment
- createDocumentType

上面的这些方法都是用于创建对应的节点类型。你可以自己尝试一下。

Element 与 Attribute

Node提供了树形结构上节点相关的操作。而大部分时候，我们比较关注的是元素。Element表示元素，它是Node的子类。

元素对应了HTML中的标签，它既有子节点，又有属性。所以Element子类中，有一系列操作属性的方法。

我们需要注意，对DOM而言，Attribute和Property是完全不同的含义，只有特性场景下，两者才会互相关联（这里在后面我会详细讲解，今天的文章里我就不展开了）。

首先，我们可以把元素的Attribute当作字符串来看待，这样就有以下的API：

- getAttribute
- setAttribute
- removeAttribute
- hasAttribute

如果你追求极致的性能，还可以把Attribute当作节点：

- getAttributeNode
- setAttributeNode

此外，如果你喜欢property一样的访问attribute，还可以使用 attributes 对象，比如 document.body.attributes.class = “a” 等效于 document.body.setAttribute(“class” , “a”)。

查找元素

document节点提供了查找元素的能力。比如有下面的几种。

- querySelector
- querySelectorAll
- getElementById
- getElementsByName
- getElementsByTagName
- getElementsByClassName

我们需要注意，`getElementById`、`getElementsByName`、`getElementsByTagName`、`getElementsByClassName`，这几个API的性能高于`querySelector`。

而 `getElementsByName`、`getElementsByTagName`、`getElementsByClassName` 获取的集合并非数组，而是一个能够动态更新的集合。

我们看一个例子：

```
var collection = document.getElementsByClassName('winter');
console.log(collection.length);
var winter = document.createElement('div');
winter.setAttribute('class', 'winter')
document.documentElement.appendChild(winter)
console.log(collection.length);
```

在这段代码中，我们先获取了页面的className为winter的元素集合，不出意外的话，应该是空。

我们通过`console.log`可以看到集合的大小为0。之后我们添加了一个class为winter的div，这时候我们再看集合，可以发现，集合中出现了新添加的元素。

这说明浏览器内部是有高速的索引机制，来动态更新这样的集合的。所以，尽管`querySelector`系列的API非常强大，我们还是应该尽量使用`getElement`系列的API。

遍历

前面已经提到过，通过Node的相关属性，我们可以用JavaScript遍历整个树。实际上，DOM API中还提供了 `NodeIterator` 和 `TreeWalker` 来遍历树。

比起直接用属性来遍历，`NodeIterator` 和 `TreeWalker` 提供了过滤功能，还可以把属性节点也包含在遍历之内。

`NodeIterator`的基本用法示例如下：

```
var iterator = document.createNodeIterator(document.body, NodeFilter.SHOW_TEXT | NodeFilter.SHOW_COMMENT, n
var node;
while(node = iterator.nextNode())
{
    console.log(node);
}
```

这个API的设计非常老派，这么讲的原因主要有两点，一是循环并没有类似“`hasNext`”这样的方法，而是直接以`nextNode`返回`null`来标志结束，二是第二个参数是掩码，这两个设计都是传统C语言里比较常见的用法。

放到今天看，这个迭代器无法匹配JavaScript的迭代器语法，而且JavaScript位运算并不高效，掩码的设计就

徒增复杂性了。

这里请你注意一下这个例子中的处理方法，通常掩码型参数，我们都是用按位或运算来叠加。而针对这种返回null表示结束的迭代器，我使用了在while循环条件中赋值，来保证循环次数和调用next次数严格一致（但这样写可能违反了某些编码规范）。

我们再来看一下TreeWalker的用法。

```
var walker = document.createTreeWalker(document.body, NodeFilter.SHOW_ELEMENT, null, false)
var node;
while(node = walker.nextNode())
{
    if(node.tagName === "p")
        node.nextSibling();
    console.log(node);
}
```

比起NodeIterator，TreeWalker多了在DOM树上自由移动当前节点的能力，一般来说，这种API用于“跳过”某些节点，或者重复遍历某些节点。

总的来说，我个人不太喜欢TreeWalker和NodeIterator这两个API，建议需要遍历DOM的时候，直接使用递归和Node的属性。

Range

Range API 是一个比较专业的领域，如果不做富文本编辑类的业务，不需要太深入。这里我们就仅介绍概念和给出基本用法的示例，你只要掌握即可。

Range API 表示一个HTML上的范围，这个范围是以文字为最小单位的，所以Range不一定包含完整的节点，它可能是Text节点中的一段，也可以是头尾两个Text的一部分加上中间的元素。

我们通过 Range API 可以比节点 API 更精确地操作 DOM 树，凡是节点 API 能做到的，Range API都可以做到，而且可以做到更高性能，但是 Range API 使用起来比较麻烦，所以在实际项目中，并不常用，只有做底层框架和富文本编辑对它有强需求。

创建Range一般是通过设置它的起止来实现，我们可以看一个例子：

```
var range = new Range(),
    firstText = p.childNodes[1],
    secondText = em.firstChild
range.setStart(firstText, 9) // do not forget the leading space
range.setEnd(secondText, 4)
```

此外，通过 Range 也可以从用户选中区域创建，这样的Range用于处理用户选中区域：

```
var range = document.getSelection().getRangeAt(0);
```

更改 Range 选中区段内容的方式主要是取出和插入，分别由extractContents和insertNode来实现。

```
var fragment = range.extractContents()  
range.insertNode(document.createTextNode("aaaa"))
```

最后我们看一个完整的例子。

```
var range = new Range(),  
    firstText = p.childNodes[1],  
    secondText = em.firstChild  
range.setStart(firstText, 9) // do not forget the leading space  
range.setEnd(secondText, 4)  
  
var fragment = range.extractContents()  
range.insertNode(document.createTextNode("aaaa"))
```

这个例子展示了如何使用range来取出元素和在特定位置添加新元素。

总结

在今天的文章中，我们一起了解了DOM API的内容。DOM API大致会包含4个部分。

- 节点：DOM树形结构中的节点相关API。
- 事件：触发和监听事件相关API。
- Range：操作文字范围相关API。
- 遍历：遍历DOM需要的API。

DOM API中还提供了NodeIterator 和 TreeWalker 来遍历树。比起直接用属性来遍历，NodeIterator 和 TreeWalker 提供了过滤功能，还可以把属性节点也包含在遍历之内。

除此之外，我们还谈到了Range的一些基础知识点，这里你掌握即可。

最后，我给你留了一个题目，请你用DOM API来实现遍历整个DOM树，把所有的元素的tagName打印出来。

补充阅读：命名空间

我们本课介绍的所有API，特意忽略了命名空间。

在HTML场景中，需要考虑命名空间的场景不多。最主要的场景是SVG。创建元素和属性相关的API都有带命名空间的版本：

- document
 - createElementNS
 - createAttributeNS
- Element
 - getAttributeNS
 - setAttributeNS
 - getAttributeNodeNS
 - setAttributeNodeNS
 - removeAttributeNS
 - hasAttributeNS
 - attributes.setNamedItemNS
 - attributes.getNamedItemNS
 - attributes.removeNamedItemNS

若要创建Document或者Doctype，也必须要考虑命名空间问题。DOM要求从document.implementation来创建。

- document.implementation.createDocument
- document.implementation.createDocumentType

除此之外，还提供了一个快捷方式，你也可以动手尝试一下。

- document.implementation.createHTMLDocument

猜你喜欢

Vue 开发实战

从 0 开始搭建大型 Vue 项目

戳此试读 



唐金州
一点资讯前端技术专家
Ant Design Vue 作者

精选留言：

- 阿成 2019-03-09 08:47:24
第一段代码中的 DocumentFragment 应该改为 DocumentType...


```
/**
 * @param {Element} el
 * @param {(Element) => void} action
function walk (el, action) {
  if (el) {
    action(el)
    walk(el.firstChild, action)
    walk(el.nextElementSibling, action)
  }
}

walk(document.documentElement, el => console.log(el.nodeName))
```

```
// 如果想要去重...
const set = new Set()
walk(document.documentElement, el => {
  set.add(el.nodeName)
})
for (let n of set)
  console.log(n)
[10赞]
```

- 周序猿 2019-03-10 10:14:11

```
// 深度优先
function deepLogTagNames(parentNode){
  console.log(parentNode.tagName)
  const childNodes = parentNode.childNodes
  // 过滤没有 tagName 的节点，遍历输出
  Array.prototype.filter.call(childNodes, item=>item.tagName)
    .forEach(itemNode=>{
      deepLogTagNames(itemNode)
    })
}
deepLogTagNames(document.body)
```

```
// 广度优先
function breadLogTagNames(root){
  const queue = [root]
  while(queue.length) {
    const currentNode = queue.shift()
    const {childNodes, tagName} = currentNode
    tagName && console.log(currentNode.tagName)
    // 过滤没有 tagName 的节点
    Array.prototype.filter.call(childNodes, item=>item.tagName)
      .forEach(itemNode=>{
        queue.push(itemNode)
      })
  }
}
```

breadLogTagNames(document.body) [7赞]

- 我叫张小咩²⁰¹⁹ 2019-03-10 00:44:50

```
var walker = document.createTreeWalker(document.body, NodeFilter.SHOW_ELEMENT, null, false)
var node
while(node = walker.nextNode())
console.log(node.tagName)
----- or recursive -----
```

```
const result = []
function getAllTagName(parent) {
  const childs = Array.from(parent.children)
  result.push(...childs.map(el => el.tagName))
  for (var i = 0; i < childs.length; i++) {
    if (childs[i].children.length) getAllTagName(childs[i])
  }
}
```

```
if (i == 0) return
}
getAllTagName(document)
```

```
console.log(result)
```

[3赞]

- 腾松 2019-03-26 15:37:21

```
function loop(node){
  if(!node){
    return
  }
  if(node.nodeType === document.ELEMENT_NODE)
    console.log(node.nodeName);
  if(node.childNodes){
    node.childNodes.forEach(child => {
      loop(child)
    })
  }
}
loop(document) [1赞]
```

- 宇宙全栈 2019-03-11 10:53:07

第一段代码中的 DocumentFragment 应该改为 DocumentType [1赞]

- 周飞 2019-03-27 22:35:40

```
let tagNameArr = [];
function travaldom(root){
  if(root.tagName && root.tagName !== 'text') tagNameArr.push(root.tagName)
  root.childNodes.forEach(node=>{
    travaldom(node);
  })
}
```

```
});  
}  
travalDOM(document);  
console.log(tagNameArr)
```

- 逐梦无惧 2019-03-27 13:41:12

老师请问这些html的结构化内容有在哪本书进行介绍吗

- 花骨朵 2019-03-22 10:16:00

```
<!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>遍历输出tagName</title>  
</head>  
<body>  
<section>  
<header>This is a header</header>  
<div>  
<h4>This is a content title</h4>  
<p>This is the <em>first</em> paragraph.</p>  
<p>This is the <strong>second</strong> paragraph.</p>  
</div>  
<footer>This is a footer of this page.</footer>  
</section>  
<script>  
const secElement = document.getElementById('sec');  
function getChildTagNames() {  
  const walker = document.createTreeWalker(document.body, NodeFilter.SHOW_ELEMENT, null, false)  
  let node;  
  while(node = walker.nextNode()) {  
    if(node.tagName)  
      console.log(node.tagName);  
  }  
}  
getChildTagNames(secElement);  
</script>  
</body>  
</html>
```

- kino 2019-03-12 11:36:44

insertBefore(newNode,null)和appendChild的区别是啥

- Ppei 2019-03-11 11:33:36

老师：

比如 document.body.attribute.class = “a” 等效于 document.setAttribute(“class” , “a”)。

应该是 document.body.attributes吧？

疑惑：

```
var divDom = document.createElement('div')
```

```
divDom.attributes.id = 'app'
```

```
divDom.setAttribute('class', 'app')
```

为什么在divDom上看不到id属性？append到页面中后，同样用getElementById也选择不到这个DOM，用class就可以。