是一个API

Chrome渲染页面DOM使用Webkit的WebCore，但JS引擎是V8

就像两座岛屿，中间是收费的桥梁，JS（ECMAScript）每一次操作DOM就要经过这座桥梁

**node类型**

比如Node.ELEMENT\_NODE，指元素

**nodeType**

只读

有些被废弃，以[MDN](https://developer.mozilla.org/en-US/docs/Web/API/Node/nodeType)为准

元素(element)=1，文本(text)=3，注释(comment)=8，DocumentFragment=11

<div class="wrapper">

<p class="content">hello, world</p>

</div>

document.querySelector(".wrapper").nodeType // 1, 等于Node.ELEMENT\_NODE

**nodeName**

nodeType等于1的时候，nodeName是元素的标签名

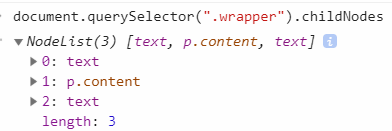


**父子关系/只读**

**childNodes**

每个node都有的属性。返回一个NodeList，有length属性，可以方括号访问里面的项，但他不是Array的实例

并且随着元素的变化会实时更新（比如遍历地append一个个节点，不先存NodeList的长度，在循环量直接NodeList.length，会进死循环）



**parentNode**

父节点

document.querySelector(".content").parentNode

**nextSibling/previousSibling**

兄弟节点，不存在则为null

**firstChild/lastChild**



**hasChildNodes()**

是否有子节点

**操作方法**

节点都是唯一的，一个文档内不可能存在多个完全一样的节点

**appendChild/removeChild(目标节点)**

append放到父节点的最后。如果本身就有节点，那么相当于移动，将原有节点删除，放到最后

remove删除目标节点

var newEle = document.createElement("div");

newEle.innerHTML = "hello";

document.querySelector(".wrapper").append(newEle)

**insertBefore(新节点，目标节点)**

放在目标节点之前

<div class="wrapper">

<p class="content">hello</p>

<p id="world" class="content">world</p>

</div>

var newEle = document.createElement("div"),

target = document.querySelector("#world");

newEle.innerHTML = "Neverland";

document.querySelector(".wrapper").insertBefore(newEle, target)

**replaceChild（新节点，目标节点）**

代替目标节点

用法和insertBefore一样

**cloneNode(flag)**

flag为true，会复制该节点的所有子节点

<div class="wrapper" onclick="handleClick()">

123

<p class="content">hello</p>

</div>

function handleClick() {

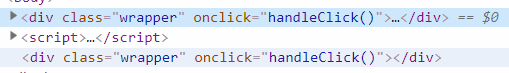
console.log("hello")

}

const doc = document;

const clones = doc.querySelector(".wrapper").cloneNode();

doc.body.appendChild(clones);



cloneNode(true)的时候



**Document类型**

document.nodeName为"#document", nodeType是9

document.documentElement指HTML，document.body指body

document.title获取<title>的值，可以对其修改

document.URL：地址栏完整的值，修改无效

**元素相关的方法**

**getElementById**

返回第一个匹配，只有document有这个方法

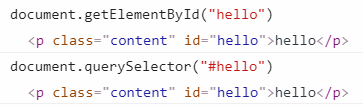
<div class="wrapper">

123

<p class="content" id="hello">hello</p>

<p class="content">word</p>

</div>



**getElementsByTagName**

与NodeList类似，也不是Array实例，并随DOM更新而更新

Element类型也可以使用



**createElement(标签名)**

创建元素，要添加到DOM里面，还需要调用[操作方法](#_操作方法)

**createTextNode(内容)**

const doc = document;

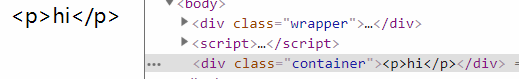
var newEl = doc.createElement("div");

newEl.className="container";

var newTxt = doc.createTextNode("<p>hi</p>");

newEl.appendChild(newTxt);

doc.body.appendChild(newEl)



**Element类型**

nodeType是1,

nodeName是标签名，和tagName一样。值始终大写



**常用属性**

id, title, className

<p class="content custom" id="hello">hello</p>



**常用方法**

**特性：getAttribute/setAttribute/removeAttribute**

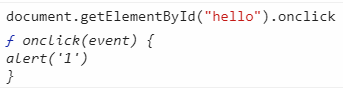
写在标签上的都可以get，哪怕是自定义的比如data-options

获取到的是字符串，比如onclick, style。

所以一般都用来获取自定义的属性。其他都通过 .属性名 直接拿

<p class="content custom" id="hello" onclick="alert('1')">hello</p>





**property 和 attribute**

* property 是DOM中的属性，是JavaScript里的对象；一般是固有属性

比如nodeName, nodeType。

比如checkbox的checked

* attribute 是HTML标签上的特性，它的值只能够是字符串。比如在标签上自定义属性名

也就是boolean型的不行

**DocumentFragment类型**

用createDocumentFragment()创建，通常当做仓库使用。

最常见的就是遍历循环，先把循环生成的节点放在fragment，在fragment被append到文档之前不会引起浏览器渲染。

一旦fragment被append，这个fragment里面的所有子节点就会从fragment内删除

const doc = document;

let newFragment = doc.createDocumentFragment(),

target = doc.querySelector(".wrapper");

for(let i=0;i<2;i++) {

var el = doc.createElement("p");

var elTxt = doc.createTextNode("hi"+i);

el.appendChild(elTxt);

newFragment.appendChild(el);

}

console.log(newFragment.childNodes);

doc.body.appendChild(newFragment);

console.log(newFragment.childNodes);



**选择器相关**

不像getElementById，只存在document

documents, element, DocumentFragment都有

**querySelector/querySelectorAll**

all返回匹配的NodeList。实际上是一组快照，不也就是不会实时更新。也就不会引起死循环

但是性能上还是getElementBy系列更好，见[知乎](https://www.zhihu.com/question/24702250)

因为query系列所有元素都可以使用，自行做取舍

**getElementsByClassName**

可以传多个类名，空格分隔。顺序可以调换

<div class="wrapper">

<p class="content child">hello</p>

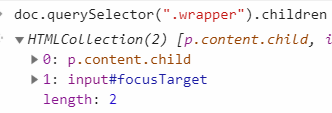
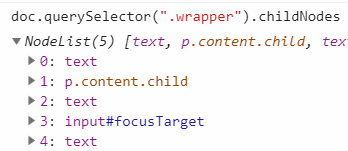
</div>



返回匹配的HTMLCollection ，会引起死循环

**children属性**

子元素，不包括text



**contains方法**

是否后代

let target = doc.querySelector(".wrapper");

let child = target.querySelector(".content");

target.contains(child)//true

**classList属性**

**方法**

* add
* contains
* remove
* toggle

**style对象**

可以获取或者修改内联样式，要加"px"



**focus()**

聚焦元素

**innerHTML/innerText属性**

innerText只会生成一个文本节点

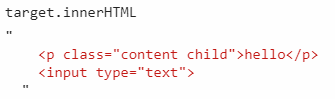
<div class="wrapper">

<p class="content child">hello</p>

<input type="text">

</div>

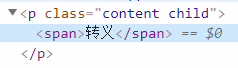
target = document.querySelector(".wrapper");





可以用innerHTML代替createElement进行元素操作





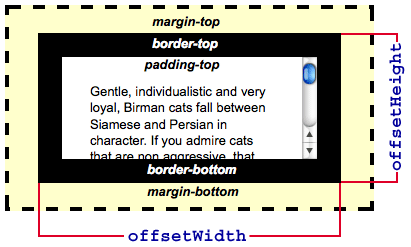
**尺寸**

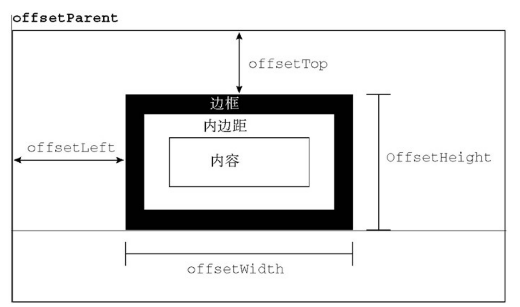
**偏移offset**

只读。

offsetWidth包括滚动条，不包括伪类::before，包括border

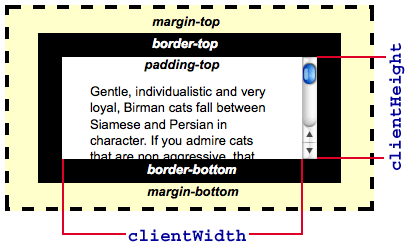
如果display是none, width和height返回0





**client**

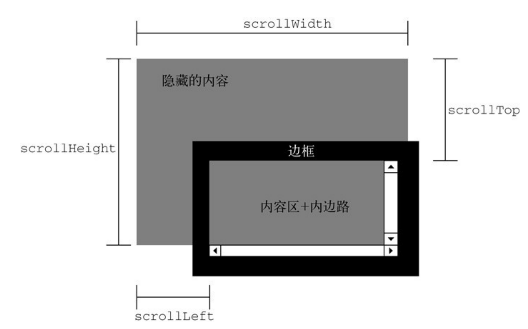
不包括滚动条，不包括border, margin。包括padding



**滚动scroll**

没滚动条的情况下scrollWidth等于clientWidth

有滚动条的情况下，父元素（被撑出滚动条的元素）scrollWidth>clientWidth



scrollHeight也就是被实际内容撑出来的高度

<style>

.wrapper {

height: 200px;

overflow: auto;

margin: 0;

}

.child {

height: 300px;

margin: 0;

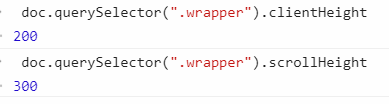
}

</style>

<div class="wrapper">

<p class="content child">hello</p>

</div>



**getBoundingClientRect()**

获取完整的尺寸信息。但是会忽略缩放。比如windows文本放大125%，浏览器内获取到值实际上是100%时候的

**性能**

* 利用DocumentFragment创建DOM，最后放到DOM中
* 或者使用innerHTML代替appendChild