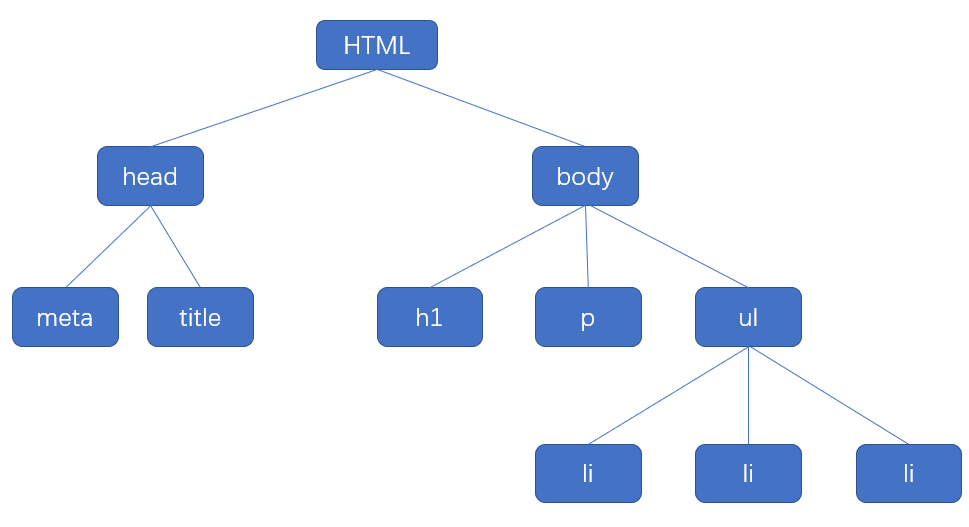
### 标题

#### 1.DOM（Document Object Model）

DOM文档对象模型，是针对HTML和XML文档的一个API，DOM代表加载到浏览器窗口的当前网页，DOM会把这份文档表示成一颗树。和树结构是一样的，根节点是HTML，具有父节点、兄弟节点和子节点。



当我们操作dom的时候就像在操作数据一样，轻松的对他们进行改变。

元素解析之后才能操作它。

1.1节点

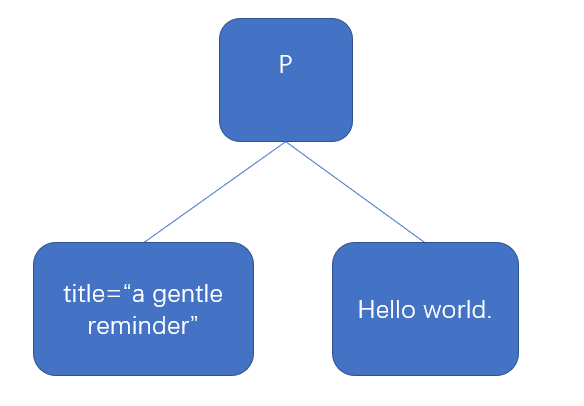
文档是由节点构成的集合，只不过此时的节点是文档树上的树枝和树叶，是由document开始，往下派出一层一层的节点。

元素节点：形成了文档的结构；

文本节点：为文档提供内容；

属性节点：对元素做出更具体的描述。

<p title="a gentle reminder">Hello world.</p>



1.2获取元素

有三种DOM方法可获取元素节点，分别是通过元素ID、通过标签名字和通过类名字来获取。

getElementById()：返回一个对象。

getElementsByTagName()：返回一个对象数组。

getElementsByClassName()：返回一个对象数组。

1.3获取和设置属性

通过获取元素方法得到元素以后，我们就可以获取相应的属性并进行操作。

getAttribute：不属于document对象，不能通过document对象调用，只能通过元素节点对象调用。

 <div title="getAttribute" id="p">hello world</div>

    var p = document.getElementById('p')

    alert(p.getAttribute('title'))



setAttribute：也是只能用于元素节点

  <div title="getAttribute" id="p">hello world</div>

  <script>

    var p = document.getElementById('p')

    alert(p.getAttribute('title'))

    p.setAttribute('title', 'setAttibute')

    alert(p.getAttribute('title'))

setAttribute设置一个节点的属性，这个属性无则创建，有则覆盖。



这时候去看网页源代码，仍然是改变前的属性值，也就是说setAttribute做出的修改不会反应在文档本身的源代码里。

这是因为DOM的工作模式，先加载文档的静态内容，再动态刷新，动态刷新不影响文档的静态内容（对页面内容进行刷新却不需要在浏览器里刷新页面）。

attribute：修改对象属性，不会体现到html结构中；

property：修改html属性，会改变html结构。

尽量用property

1.4操作节点

appendChild()：向子节点列表末尾添加一个节点。

insertBefore()：把要插入的节点放到某个特定的位置。

replaceChild()：把要插入的节点替换掉现在的节点。

cloneNode()：对节点进行复制，接受一个布尔值参数，true为深拷贝，false为浅拷贝。

案例：

  <div id="div1" class="container">

    <p id="p1">一段文字 1</p>

    <p id="p2">一段文字 2</p>

    <p>一段文字 3</p>

  </div>

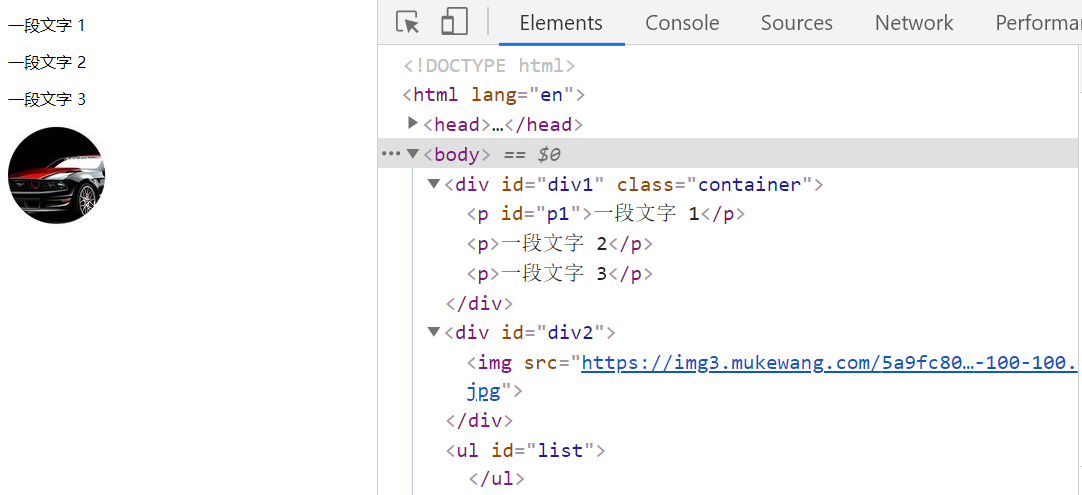
  <div id="div2">

      <img src="https://img3.mukewang.com/5a9fc8070001a82402060220-100-100.jpg"/>

  </div>

      <ul id="list">

  </ul>



  <script>

    const div1 = document.getElementById('div1')

    const p1 = document.createElement('p')

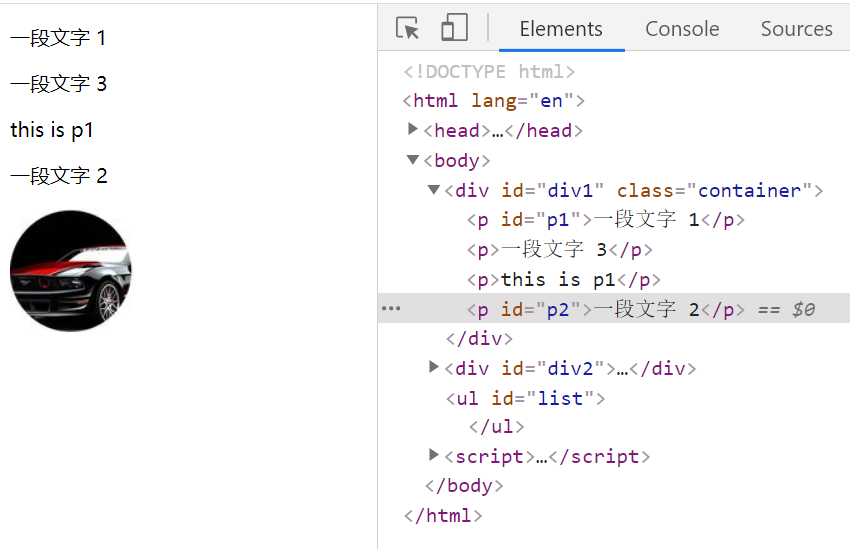
    p1.innerHTML = 'this is p1'

    div1.appendChild(p1)

    const p2 = document.getElementById('p2')

    div1.appendChild(p2)

  </script>



从以上代码案例可以看出：当添加没有的新节点时，直接添加到节点的列表；当添加已有几点的时候，把已有节点移动到最后。

1.6快速插入DOM节点

为了快速插入DOM节点，要使用DOM片段。因为DOM的操作非常昂贵，占用CPU比较多，频繁操作可能会造成卡顿，所以我们要将频繁操作改成一次性操作。



const list = document.getElementById('list')

for(let i = 0; i < 10; i++){

    const li = document.createElement('li')

    li.innerHTML = `List item ${i}`

    list.appendChild(li)

}

把插入DOM节点编程一个整体

// 创建一个文档片段，此时还没有插入到 DOM 结构中

const frag = document.createDocumentFragment()

for (let i  = 0; i < 20; i++) {

    const li = document.createElement('li')

    li.innerHTML = `List item ${i}`

    // 先插入文档片段中

    frag.appendChild(li)

}

// 都完成之后，再统一插入到 DOM 结构中

list.appendChild(frag)

console.log(list)

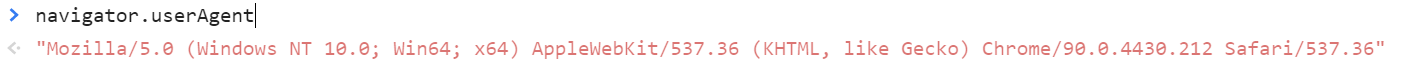
#### 2.BOM（Browser Object Model）浏览器对象模型

BOM是JavaScript的核心，以window对象为基础，代表了浏览器窗口和页面可见的区域，通过JS去操作浏览器。

2.1navigator对象

识别客户端浏览器的事实标准，常用于确定浏览器的类型。

通过userAgent可以拿到浏览器的信息



#### 3.事件

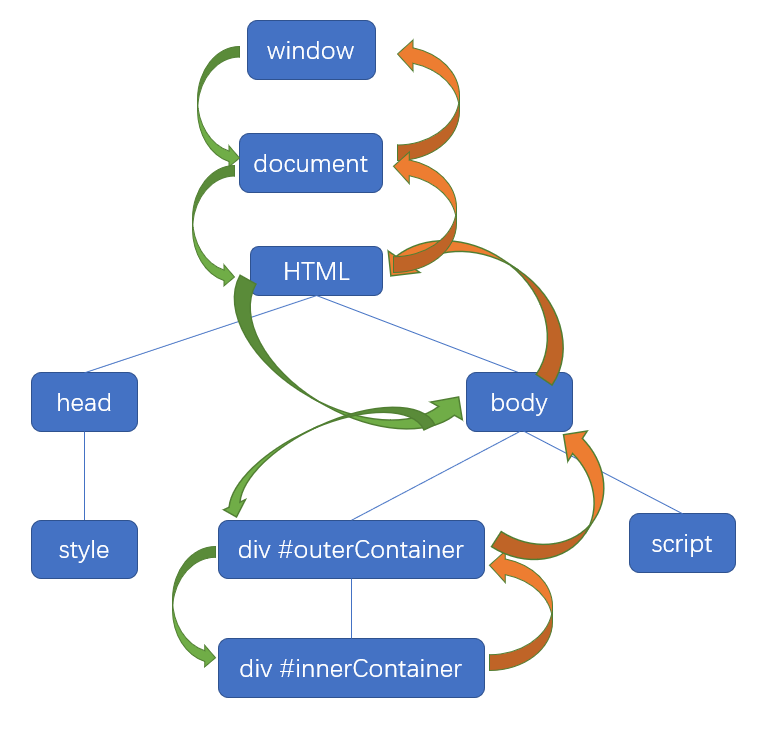
事件就是文档或浏览器窗口发生的一些特定的交互，意味着用户或浏览器执行的某种动作。

事件处理程序：为响应事件而调用的函数，以on开头。

事件捕获：首先被顶部元素捕获，并依次向下传递。

事件冒泡：目标元素捕获之后，事件处理转为冒泡，从目标元素向顶部元素冒泡。

事件代理（委派）：将事件统一绑定给元素的共同祖先元素，这样后代元素上的事件触发时，会冒泡到祖先元素，从而通过祖先元素的相应函数来处理事件，利用冒泡减少事件的绑定。



通过捕获，事件最终传递到目标元素。通过冒泡，事件从目标元素向上冒泡。

3.1鼠标事件

鼠标事件都是在浏览器视口中的某个位置发生的，浏览器视口如下图绿色区域。

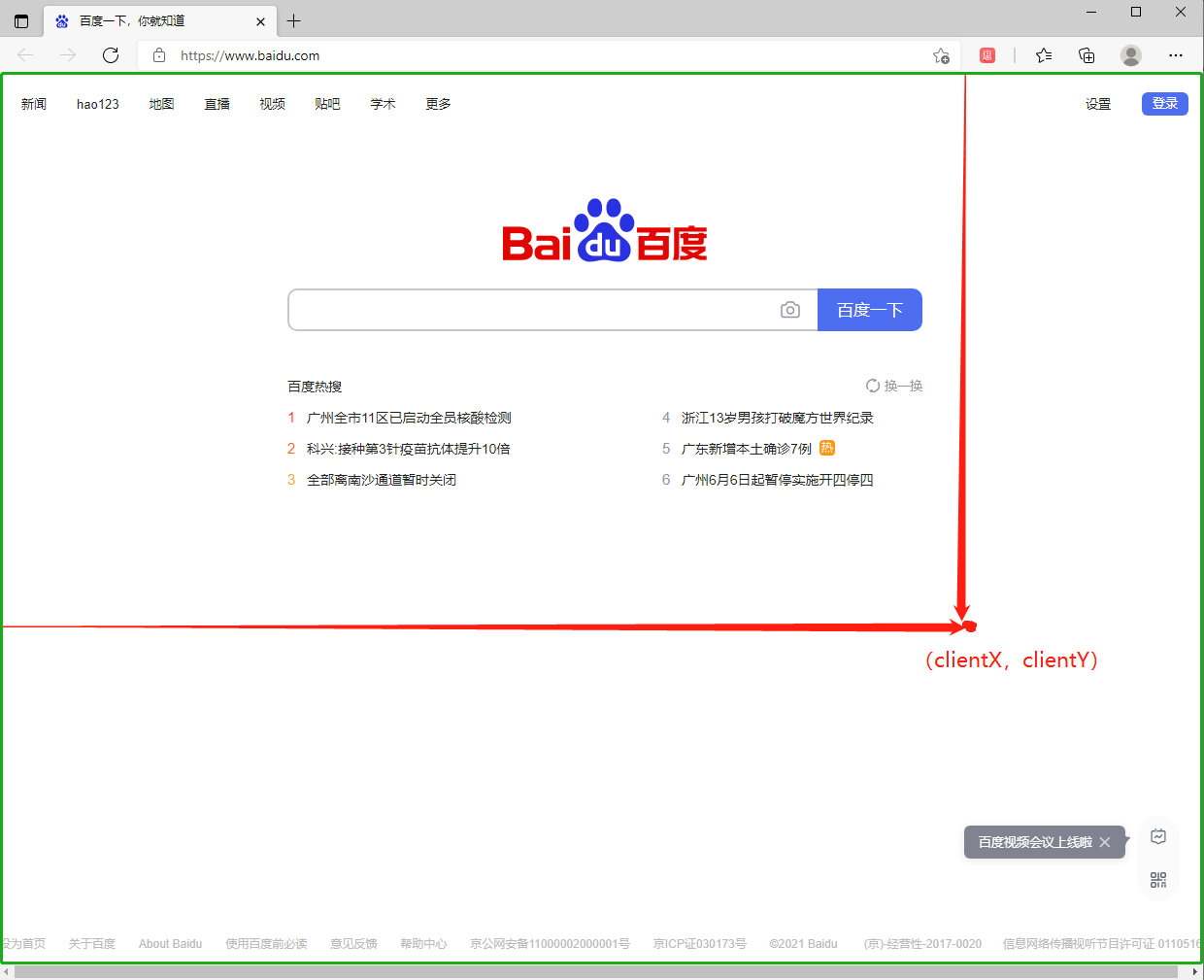
1.客户端坐标是以浏览器视口为基准的。

let div = document.getElementById('myDiv')

div.addEventListener('click',(event) => {

    console.log(`Client coordinates: ${event.clientX},${event.clientY}`);

})



2.页面坐标

页面坐标是事件发生时鼠标光标在页面上的坐标，通过event对象的pageX和pageY可以获取，反应的是光标到 “页面”，注意滚动的页面。

let div = document.getElementById('myDiv')

div.addEventListener('click',(event) => {

    console.log(`Page coordinates: ${event.pageX},${event.pageY}`);

})

IE8以及更早版本没有在event对象上暴露页面坐标，可以通过客户端坐标和滚动信息计算出来。

        let div = document.getElementById('myDiv')

        div.addEventListener('click', (event) => {

            let pageX = event.pageX

            let pageY = event.pageY

            if (pageX === undefined) {

                pageX = event.clientX + (document.body.scrollLeft || document.documentElement.scrollLeft)

            }

            if (pageY === undefined) {

                pageY = event.clientY + (document.body.scrollTop || document.documentElement.scrollTop)

            }

        })

        console.log(`Page coordinates: ${pageX}, ${pageX}`)

3.屏幕坐标

鼠标事件不仅是在浏览器窗口发生，也在整个屏幕上发生，可以通过event对象的screenX和screenY属性获取鼠标光标在屏幕上的坐标。

let div = document.getElementById('myDiv')

div.addEventListener('click',(event) => {

    console.log(`Sceencoordinates: ${event.screenX},${event. screenY}`);

})

#### 二级标题

##### 三级标题