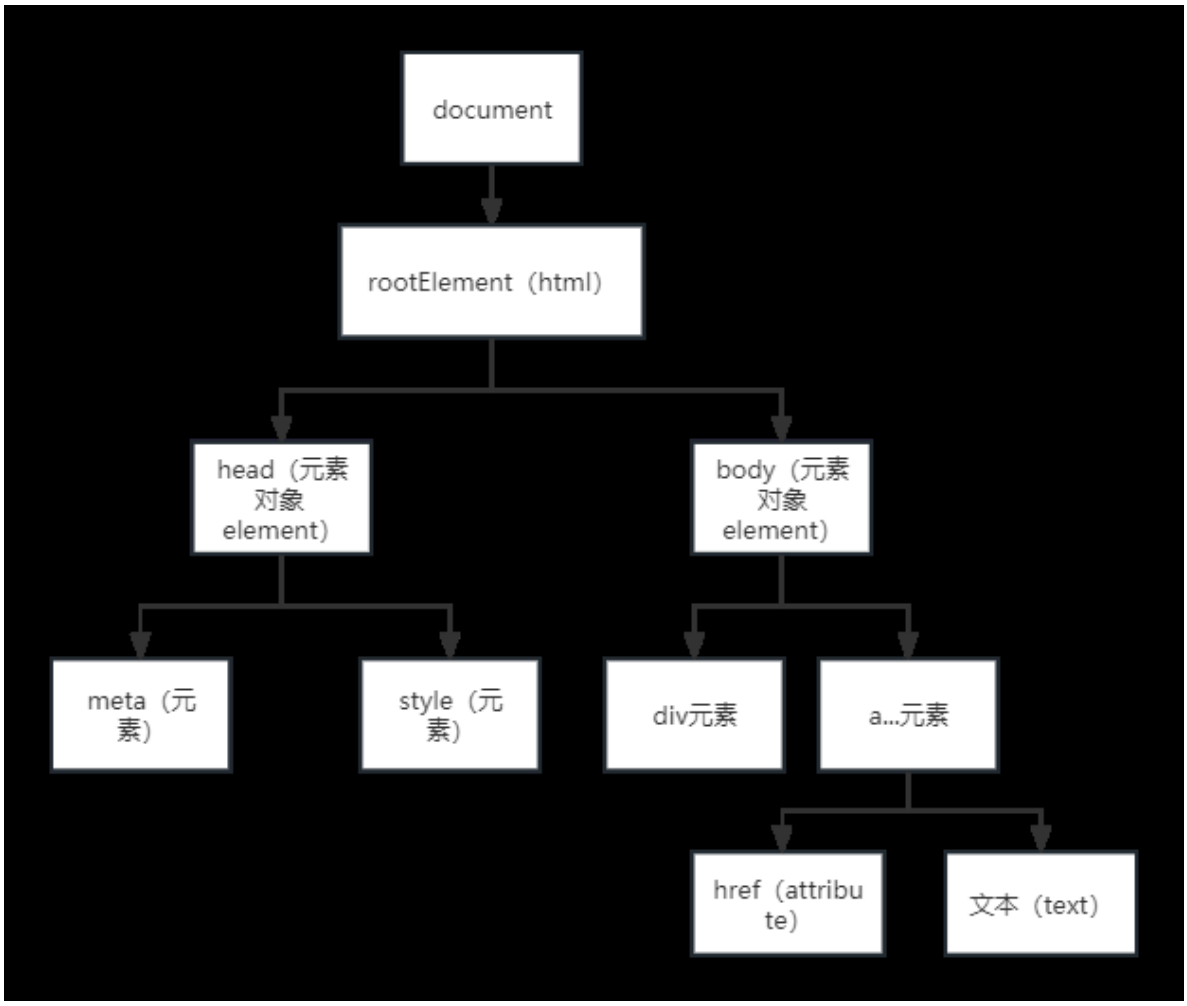
day09 DOM

DOM概述

DOM 文档对象模型(document object model) 主要是用于操作html文档及相关内容（css）。对于文档的操作会造成浏览器的重新渲染（重绘（改变的一个元素的显示内容及部分样式） 重排（改变一个元素的位置及大小尺寸）（回流）重绘不一定重排 重排必定发生重绘）

DOM具备的内容

- **document** （属于BOM）文档对象
- rootElement （根元素）
- **element** （所有的标签都是element）
- **attribute** （所有标签的属性都是attribute）
- **text** （所有标签内容的文本都是text）



从上的包含关系可以看出大的内容可以获取或操作包含的内容也就是

- document可以获取rootElement
- document可以获取element
- element可以获取里面的element
- element可以获取里面attribute以及text

获取元素相关的操作 document和element都有的方法

- getElementById 通过id来获取元素 （元素 element）
- getElementsByClassName 通过classname获取所有的符合的元素 （伪数组）
- getElementsByTagName 通过标签名获取所有符合的元素 （伪数组）
- getElementsByName 通过name属性来获取符合的元素 （伪数组）
- querySelector 通过传入选择器来获取符合的第一个元素 （元素 element）
- querySelectorAll 通过传入选择器获取符合的所有元素 （伪数组）
- 所有跟上面一致后面带NS表示命名空间获取查找

```
// 获取元素的方式
// 通过id获取div
console.log(document.getElementById('box'))//传入一个id名返回一个元素（htmlElement）
console.log(document.getElementsByClassName('context'))//传入一个class名返回
HTMLCollection 伪数组
console.log(document.getElementsByTagName('div'))//传入一个标签名返回HTMLCollection
console.log(document.getElementsByName('hello'))//传入一个name属性名返回NodeList 伪
数组
console.log(document.querySelector('.context')) //根据选择器获取匹配的第一个元素
console.log(document.querySelectorAll('#box')) //根据选择器获取所有匹配的 返回
NodeList 伪数组
//通过id为box的div获取里面的a标签
var box = document.getElementById('box')
console.log(box.querySelector('a'))
//document获取根元素
console.log( document.getRootNode())
```

伪数组具备下标的特性及length属性 但是不具备数组的方法

element操作attribute

getAttribute 获取属性值的方法 setAttribute 设置属性的方法

```
var element = document.getElementById('box')
//使用getAttribute函数来获取属性值
var value = element.getAttribute('href') //根据属性名获取属性值
console.log(value)
//getAttribute相匹配的setAttribute
element.setAttribute('number','10')//设置属性number值为10
element.setAttribute('number','20')//修改属性number为20
```

getAttributeNode 获取属性对象 setAttributeNode 设置属性对象

```
//获取单个的属性节点 及 设置对应的属性节点
var obj = element.getAttributeNode('class') //返回的是一个属性对象
console.log(obj.value)
//传入一个Attr对象 进行设置
// element.setAttributeNode()
```

获取所有的属性名 getAttributeNames

```
var attrNames = element.getAttributeNames() //返回所有的属性名
console.log(attrNames)
```

attributes 属性 （属于element的属性）

```
//使用attributes来获取所有的属性对象 伪数组 NameNodeMap
console.log(element.attributes)
// console.log(element.attributes)
//访问对应的class属性对象
//利用下标访问
console.log(element.attributes[2])//Attribute对象
//利用key来访问
console.log(element.attributes['class'].value)
//Attribute对象里面的属性值的访问使用value属性方法
console.log(element.attributes[2].value)//class的属性值
//NameNodeMap里面对应的属性的增删改查
var attrs = element.attributes
//查询 getNamedItem item
console.log(attrs.getNamedItem('class').value)
console.log(attrs.item(2).value)
//修改 添加是一样的方法
attrs[0].value = 'http://www.baidu.com'
// attrs.setNamedItem() 传递一个Attr对象来进行修改或者添加
//删除方法
attrs.removeNamedItem('hello')
```

element都具备的属性 （直接赋值或者获取）

- id
- className
- tagName
- titile
- style
- innerHTML 显示的html代码 和 innerText 显示的文本

```
console.log(box.innerHTML) //直接获取里面所有的html代码 赋值的时候会自动识别html代码(xss攻击)
box.innerHTML = '<font color="red">hahahah</font>'
// console.log(box.innerText) //只会获取文本 不会识别html代码
box.innerText = '<font color="red">hahahah</font>'
```

- 对于本身自带的属性可以直接点的形式来进行获取或者赋值

```
//对于element元素本身自带的属性 可以直接设置 a标签本身自带href input标签本身自带value属性
//通过直接点出来的形式进行设置和获取
// id class style 标签名  title
var box = document.querySelector('div')
console.log(box.id)
console.log(box.className)
console.log(box.style) //返回一个样式对象
console.log(box.tagName) //标签名
console.log(box.title) //标签名
//对于标签本身不自带的属性 出现的值为undefined
console.log(box.hi)
box.id = 'box'
box.className = 'context'
//直接设置a标签的href属性
element.href = 'http://www.bilibili.com'
```

元素内样式的操作

获取样式

style 返回一个样式对象 通过样式对象.样式名来获取

```
//获取样式
var div = document.querySelector('div')
//第一种方式 使用style属性进行获取
console.log(div.style) //返回的是一个对象 样式对象 里面包含的对应的样式
console.log(div.style[0]) //获取style里面包含的样式 fontsize样式名 (如果不是内嵌的样式那么使用style属性无法获取)
console.log(div.style.fontSize)//获取style属性中包含的fontSize的样式值
//如果要获取style属性中包含样式（内嵌的样式）那么使用element.style.样式名（使用驼峰命名）
//但是这种获取方式无法获取外联样式及内联样式 只能获取内嵌
console.log(div.style.color)
```

getComputedStyle 返回一个样式对象 通过样式对象.样式名来获取

```
//第二种获取方式 通过对应的方法来进行获取 (兼容问题)
//第一个参数需要被获取的元素 这个返回的style里面是有内置的默认值的 如果你设置的样式他就会覆盖这个默认值
console.log(getComputedStyle(div))//属于window的方法 返回的是样式对象 他可以获取所有的样式
console.log(getComputedStyle(div).position)
console.log(getComputedStyle(div).color)
console.log(getComputedStyle(div).fontSize)
```

存在兼容问题

```
//兼容问题解决
//兼容问题 ie的
// element.currentStyle
function getStyle(ele,name){
    return getComputedStyle?getComputedStyle(ele)
[name]:element.currentStyle['name']
}
```

设置样式

style来进行设置

优点

- 可以单独对于某一个样式进行精确控制

缺点

- 每次只能设置一个样式
- 他会发生多次重绘和重排 （效率低）

```
//设置样式
// 第一种方式 (每次设置只能设置一个)
div.style.fontWeight = 700
div.style.fontSize = '40px'
```

通过指定class名字来设置

```
//第二种设置通过class名字来设置(一次性可以设置多个样式)
div.className = 'font'
```

优点

- 可以一次性设置多个样式
- 只会发生一次重绘和重排 （效率高）

缺点

- 没有办法做到随意更改

创建对应的内容

- 创建元素 createElement
- 创建属性 createAttribute
- 创建文本 createTextNode
- 创建片段 createDocumentFragment

```
//创建元素
var ele = document.createElement('a') //传入标签
console.log(ele)
//创建属性
var attr = document.createAttribute('href')//传入属性名 通过value来进行赋值
attr.value = 'http://www.baidu.com'
console.log(attr)
//将attr和ele联系在一块 建议使用setAttribute
ele.setAttributeNode(attr)
//创建文本节点 传入对应的文本内容
var text = document.createTextNode('你好')
console.log(text)
//appenChild 建议使用innerText
ele.appendChild(text) //将文本添加到元素中
console.log(ele)
//创建片段
// document.createDocumentFragment
```

节点相关操作

节点主要分为三类 分别为

- 元素节点 （element）
- 属性节点 （attribute）
- 文本节点 （textNode）

关于节点的属性

- nodeType 节点类型
- nodeName 节点名
- nodeValue 节点值

节点名	nodeType	nodeName	nodeValue
元素节点	1	标签名	null
属性节点	2	属性名	属性值
文本节点	3	#text	文本值

```
var ele = document.createElement('a') //传入标签
//创建属性
var attr = document.createAttribute('href')//传入属性名 通过value来进行赋值
attr.value = 'http://www.baidu.com'
//创建文本节点 传入对应的文本内容
var text = document.createTextNode('你好')
console.log(ele.nodeType)//1
console.log(attr.nodeType)//2
console.log(text.nodeType)//3
console.log(ele.nodeName)//标签名 大写
console.log(ele.tagName)//标签名 大写
console.log(attr.nodeName)//属性名
console.log(text.nodeName)//#text
console.log(ele.nodeValue)//null
console.log(attr.nodeValue)//属性值
console.log(text.nodeValue)//文本值
```

节点操作的方法

节点的添加

- append 追加到后面（支持追加多个）
- appendChild （追加到最后）
- insertBefor （插入到某个子元素之前）
- insertAdjacentHTML （插入html）
- insertAdjacentText（插入文本）
- insertAdjacentElement （插入元素）

```
var ele = document.createElement('a') //传入标签
var ele1 = document.createElement('p') //传入标签
//将a标签添加到div中
var box = document.getElementById('box')
//追加子元素 在后面添加
box.appendChild(ele)
//插入到b的前面 第一个参数是需要插入的元素 第二个参数是子元素
//不允许重复插入同一元素 插入同一元素只会改变位置
box.insertBefore(ele,box.querySelector('b'))
//append追加
//  box.append(ele)
//  box.append(ele,ele1) //支持追加多个
box.append('<b>hello</b>')//如果传入的是string类型 他会当作文本节点插入
//插入对应元素的
//传入俩个参数 位置 内容
// 位置字符串 beforeBegin afterBegin 开始 beforeend aterend 结尾
box.insertAdjacentHTML("afterbegin",'<header>hahahah</header>') //插入html
box.insertAdjacentHTML("beforeBegin",'<header>hahahah</header>') //插入html
box.insertAdjacentHTML("beforeend",'<header>hahahah</header>') //插入html
box.insertAdjacentHTML("afterend",'<header>hahahah</header>') //插入html
```

```
// box.insertAdjacentText() //插入文本
box.insertAdjacentText("afterend",'<header>hahahah</header>') //插入文本
// box.insertAdjacentElement() //插入元素
box.insertAdjacentElement("beforeBegin",document.createElement('footer'))
```

删除

- remove 移除本身
- removeChild 传入对应的子节点进行删除

```
//remove移除自身的所有
// box.remove()
// removeChild删除子元素
box.removeChild(box.querySelector('b'))
```

替换

- replaceChild 替换子节点
- replaceChildren 替换所有的子节点

```
//替换子节点 用于替换的节点 子节点
box.replaceChild(document.createElement('div'),box.querySelector('b'))
//替换所有的子节点 replaceChildren
box.replaceChildren(document.createElement('canvas'),document.createTextNode('文本'))
```

克隆节点 （返回新的节点）

- cloneNode 返回一个新的节点传入一个boolean类型的值是否深度克隆

```
//cloneNode 传入一个boolean类型的值 是否深度克隆 默认值为false
console.log( box.cloneNode()) //只会克隆自身 里面的内容不会进行克隆
console.log( box.cloneNode(true)) //会克隆所有的内容
```

判断是否存在子节点

```
//判断是否存在子节点 (不包含属性节点)
console.log(div.hasChildNodes())
```

判断是否存在属性的相关方法

- hasAttribute 判断是否存在指定属性
- hasAttributes 判断是否存在属性

```
<div hello="你好"></div>
<script>
    var div = document.querySelector('div')
    //判断是否存在指定属性（必须声明） 返回值boolean
    console.log(div.hasAttribute('hello'))//true
    console.log(div.hasAttribute('id'))//false
</script>
```

节点的关系 （关系查询） （只读属性）

父子关系

- childNodes 获取的子节点
- children 获取所有的子元素节点
- firstChild 获取第一个子节点
- firstElementChild 获取第一个子元素节点
- lastChild 获取最后一个节点
- lastElementChild 获取最后一个子元素节点
- parentNode 父节点
- parentElement 父元素节点

```
//获取子节点
console.log(div.childNodes)//所有的子节点 NodeList 包含空文本节点
console.log(div.childNodes.length)//所有的子节点的个数
//获取所有的子元素节点
console.log(div.children)//所有的子元素节点 HTMLCollection
console.log(div.children.length)
//获取第一个子节点
console.log(div.firstChild)
//获取第一个元素节点
console.log(div.firstElementChild)
//获取最后一个子节点
console.log(div.lastChild)
//获取最后一个元素节点
console.log(div.lastElementChild)
//获取父节点 parent
console.log(div.parentNode) //父节点
console.log(div.parentElement) //父元素
```

兄弟关系

- previousSibling 前一个节点
- previousElementSibling 前一个元素节点
- nextSibling 后一个节点
- nextElementSibling 后一个元素节点

```
//兄弟关系
console.log(div.previousSibling) //前面兄弟节点
console.log(div.previousElementSibling) //前面的兄弟元素节点
console.log(div.nextSibling) //后面兄弟节点
console.log(div.nextElementSibling) //后面的兄弟元素节点
```

去除空文本节点

```
function removeEmptyTextNode(node) {
    //去除所有的空文本节点  删除
    var childs = node.childNodes //获取所有的子节点
    //遍历 伪数组实现了迭代iterator 可以被for of遍历
    for (var child of childs) {
        //找到所有的空文本节点
        if (child.nodeType == 3 && child.nodeValue.trim() == '') {
            child.remove()
        }
    }
}
```