# DOM操作笔记1,看这里: [DOM学习笔记1](jsdom学习笔记1.docx)

# 这里是笔记2

# 12.DOM元素的属性(attribute)p142

## 属性是由内置属性和自定义属性之分,他们的获取方式不一样

## 1>所有的属性(标准和自定义的)都支持的操作方式

|  |
| --- |
|  |

### 注意:

### 1.setAttribute既可以修改属性,也可以添加新属性

### 2.对应内置属性,其实直接使用elem.属性名的方法更好,因为上面的方法获取到的值总是字符串,如果需要其他类型还需要转换,不太方便

# 13.元素的属性(property)操作 p143

|  |
| --- |
|  |

## 注意区分:对象里面的属性叫做property,元素的属性叫做attribute,我们用js获取到的dom元素其实就是一个对象, 意味着我们使用.操作符获取到的就叫做property.也可以这么理解:每一个HTML标签的内置属性或者说标准属性attribute都会对应一个DOM对象的属性property,所以对应标准属性,你可以有2种方法来获取,对应自定义的属性attribute,他们是没有对应的property的,因为DOM的设计者根本不可能知道你将会添加一个什么样的属性attribute.

## 课堂案例

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>元素的属性:property</title>  </head>  <body>      <div class="box" id="myBox" info="hello">          div的文本      </div>      姓名:<input type="text" class="in" value="">      <script>          //property就是对象里面的属性,我们用Dom获取到的就是一个个的对象,此时我们操作的就是property          //每一个HTML标签的内置属性或者说标准属性attribute都会对应一个DOM对象的属性property,          // 所以对于标准属性,你可以有2种方法来获取,对应自定义的属性attribute,他们是没有对应的property的.          //自定义属性不能用.操作符来获取          //1.获取元素的dom对象          let divBox = document.querySelector(".box")          //class he id是标准属性,可以有2种方法获取          console.log(divBox.getAttribute("id")); //myBox          console.log(divBox.id);                 //myBox          //自定义属性无法使用.操作符获取,只能使用getAttribute方法来获取          console.log(divBox.info);                   //undefined          console.log(divBox.getAttribute("info"));   // hello          //不过用js代码设置的属性是可以用.操作符来获取,用getAttribute获取不到,查看HTML源码,发现是不会添加到源码中的          //用setAttribute方法是可以往标签里面添加属性的.用DOM操作是不能往标签里面添加属性的.          //虽然DOM对象和指定的标签关联,但是DOM对象添加property不会反映在标签上面.          divBox.msg = "Welcome"          console.log(divBox.msg);          console.log(divBox.getAttribute("msg")); //null            let input = document.querySelector(".in")          // input.value = "Jordon" //这个的优先级比较高,用它设置后,下面的方法无法覆盖          input.setAttribute("value","Mike") //用这种方法设置属性可能不会在界面上实时更新      </script>  </body>  </html> |

# 14.元素的class和style--p144

## js中动态修改元素的样式,

|  |
| --- |
|  |

## 课堂案例

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>元素的class和style属性</title>      <style>          .yellow{              background-color: yellow;              color:white;              font-size: 17px;              border: 1px dashed #ccc;              width: 250px;              height: 250px;          }          .deeppink{              background-color: deeppink;              color:gray;              font-size: 18px;              border: 1px solid #999;              text-align: center;              width: 200px;              height: 200px;          }          .lime{              background-color: lime;              color:skyblue;              font-size: 17px;              width: 220px;              height: 220px;          }          .purple{              background-color: purple;              color:white;              font-size: 17px;              border: 1px dashed deeppink;              line-height: 300px;              width: 250px;              height: 300px;          }          .brown{              background-color: brown;              color:green;              font-size: 19px;              width: 250px;              height: 280px;          }      </style>  </head>  <body>      <div class="box" style="text-align: center;">          类名叫做box的div元素      </div>      <p id="p1">          P标签的内容      </p>      <script>          //我们这里实现一个功能点击div元素把它的背景颜色修改为随机颜色,并且把柱体变大          let colors = ["red","yellow","green","blue","deeppink","pink","lime","purple","brown","sky blue"]          let index = Math.floor(Math.random()\*colors.length)          console.log(index);          let boxDiv = document.querySelector(".box")          boxDiv.onclick = function (e) {              boxDiv.style.backgroundColor = colors[index]              boxDiv.style.color = "white"              boxDiv.style.fontSize = "24px"              index = Math.floor(Math.random()\*colors.length)          }      </script>      <script>          //我们更加推荐使用添加类名的方法来改变样式          let p = document.querySelector("#p1")          let clses = ["yellow","deeppink","lime","purple","brown"]          let idx = Math.floor(Math.random()\*clses.length)          console.log(idx);            p.onclick = function(e) {              p.className = clses[idx]              idx = Math.floor(Math.random()\*clses.length)          }      </script>  </body>  </html> |

# 15.DOM对象操作元素的class方法 p145

|  |
| --- |
|  |

## 用js来修改元素的class有2种方法,

## 第一种就是elem.className=类名,这种方法会覆盖原来的类名,有时候不是很好

## 第二种方法使用classList,这种方法允许添加多个类名,不会覆盖原来的类名

## 注意,classList添加元素的方法是add方法说明这个不是数组是一个列表

## 课堂小案例

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>Document</title>      <style>          .cls1{             background-color: #999;             border: 1px solid deeppink;          }          .cls2{             text-align: center;             width: 200px;             height: 300px;             line-height: 300px;          }      </style>  </head>  <body>      <div class="box" style="text-align: center;">          类名叫做box的div元素      </div>      <button class="btn1">单个类名</button>  <button class="btn2">多个类名</button>      <script>          //1.获取元素          let boxDiv = document.querySelector(".box")          let btn1 = document.querySelector(".btn1")          let btn2 = document.querySelector(".btn2")          //用js来修改元素的class有2种方法,第一种就是elem.className=类名,这种方法会覆盖原来的类名,有时候不是很好          btn1.onclick = function (e){             boxDiv.className = "cls1"             console.log(boxDiv.className);            }          //第二种方法使用classList,这种方法允许添加多个类名,不会覆盖原来的类名          btn2.onclick = function (e){              boxDiv.classList.add("cls2") //注意,这里是add方法说明这个不是数组是一个列表              console.log(boxDiv.className);              boxDiv.classList.toggle("box")          }      </script>  </body>  </html> |

# 16.DOM对象操作元素style—p146

## 元素的attribute和dom对象的property在syle中是这样子对应的,attribute的style中background-color对应Dom的backgroundColor.比如我们在标签里面些一下样式

|  |
| --- |
|  |

## 然后在js代码中使用dom来输出这个style对象

|  |
| --- |
|  |
|  |

## 此时如果我们想获取应该属性,我们需要使用驼峰的写法来获取

|  |
| --- |
|  |
|  |

## 可以使用cssText来一次设置多个属性,其实也不推荐,最好是使用添加css的class属性的方法

|  |
| --- |
|  |
|  |

## 需要注意的是:使用dom来读取元素的属性只能够读取到内联样式的属性,在style标签里面些的样式属性是无法读取的.

|  |
| --- |
|  |
|  |

## 那么这种样式应该如何读取?此时需要使用一个函数叫做getComputedStyle来获取

|  |
| --- |
|  |
|  |
|  |

# 17.DOM元素的dataset或者data-\*的用法—p147

## data-\*属性主要是用来解决用DOM无法获取自定义属性的问题,这些属性都会被放置到元素的dataset里面,然后多个单词的属性(也就是data-后面的东西,注意:不会包含data-)就会转化为驼峰的写法.比如我们有这样子一个div,在里面定义了一下data-\*属性

|  |
| --- |
|  |

## 然后我们就可以在js里面这么获取

|  |
| --- |
|  |
|  |

## 好处,不用再使用getAttribute函数和setAttribute函数.

## 注意:东西在小程序开发里面经常用到.

## 课堂代码

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>元素的data-\*属性</title>  </head>  <body>      <div class="box" data-info="Hello World" data-msg="You Got Me" data-viewport-width="1000px" data-index-number="1">          div里面的文字      </div>      <script>          //data-\*属性主要是用来解决用DOM无法获取自定义属性的问题,这些属性都会被放置到元素的dataset里面           let box = document.querySelector(".box")           console.log(box.dataset.viewportWidth); //viewport-width==>viewportWidth  1000px           console.log(box.dataset.indexNumber);   //index-number ===>indexNumber    1           console.log(box.getAttribute("data-msg")); //用box.getAttribute来获取的时候,前面的data-不可少           box.dataset.msg="I Got You" //设置值,注意,一定要去掉前面的data-           console.log(box.getAttribute("data-msg"));        </script>  </body>  </html> |

# 18.DOM操作-创建元素 ==p148

|  |
| --- |
|  |

## 添加元素可以使用append和appendChild(老方法),此时会把子元素添加到父元素的最后面

|  |
| --- |
|  |

## 此外还有很多方法

|  |
| --- |
|  |

## 注意:哪些appendChild,insertBefore等等的是老方法,没有我们的新方法好用.

## 课堂代码

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>创建元素</title>  </head>  <body>      <div class="box"></div>      <script>         //dom创建元素         //1.获取div元素         let div = document.querySelector(".box")         let h2 = document.createElement("h2")         h2.textContent="Hello,我是dom创建的h2"         h2.className = "title"         //append      //    div.appendChild(h2)  //这个是老方法      //    div.append(h2)          //这个是新方法,而且它可以一次插入多个元素        let p = document.createElement("p")        p.textContent = "我是p元素"        p.className="content"        div.append(h2,p)   //插入多个元素          let ul = document.createElement("ul")        let li = document.createElement("li")        li.textContent = "second li"        ul.append(li)        li = document.createElement("li")        li.textContent="first li"        //prepend        ul.prepend(li)          document.body.append(ul)        //before and after        //   ul.before(div) //把ul放置div的后面,因为ul前面有div嘛        ul.after(div)  //把ul放置div前面,因为ul后面有div嘛        let h1=document.createElement("h1")        h1.textContent="我是大标题"        document.body.prepend(h1)        //replaceWith        let span = document.createElement("span")        span.textContent="我是span的内容"        p.replaceWith(span) //会替换元素和他的所有子元素      </script>  </body>  </html> |

# 19.dom-移除元素和元素克隆 p149

## 克隆元素使用元素.cloneNode(deep)方法,deep参数表示是否是深拷贝.移除元素使用元素.remove(),注意不要在元素的父元素上面调用remove,会把父元素和兄弟元素统统删除

## 另外,下面还有一下旧方法

|  |
| --- |
|  |

## 老师案例

|  |
| --- |
|  |

## 课堂案例

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>元素移除和元素克隆</title>  </head>  <body>      <button class="clone">克隆box</button><button class="remove">移除原来的div</button>      <div class="box">          <h2>这里是标题</h2>          <p>这里是正文</p>      </div>      <script>          let divBox = document.querySelector(".box")          //这里,如果在克隆的时候传递false,就会创建浅拷贝,我们可以设置为true这样子拷贝才可见          //而且如果你传递false,他就只拷贝最外层的元素,子元素不会拷贝          //cloneNode不仅仅可以克隆元素节点,也可以克隆文本,注释等等的非元素节点          let btnCl = document.querySelector(".clone")          let count = 1          btnCl.onclick = function (){              let divCpy = divBox.cloneNode(true)              divCpy.children[0].textContent = "标题" + count++              document.body.append(divCpy)          }            let btn = document.querySelector(".remove")          btn.onclick = function (){              //这里最好使用元素remove方法,而不要使用父元素的remove方法,因为父元素的remove方法可能会移除所有              //元素移除自己时,会把他的子元素也一起移除              divBox.remove()          }      </script>  </body>  </html> |