DOM

> 本文将详细介绍DOM相关的知识点，包括但不限于Document文档结构、Node节点、Node节点的类型、Node节点的关系以及DOM的基本操作(节点的获取 、 节点的创建 、 节点的插入 、 节点的克隆 和 删除 等)等内容，在文章的最后还以附录的形式列出了DOM相关的所有属性和方法，需要指出的是本文不包含任何浏览器历史、内核以及页面渲染的内容，如有需要请参考 浏览器、内核和引擎 和 HTML页面渲染的基本过程 这两篇文章。

1.0 关于DOM

基本情况

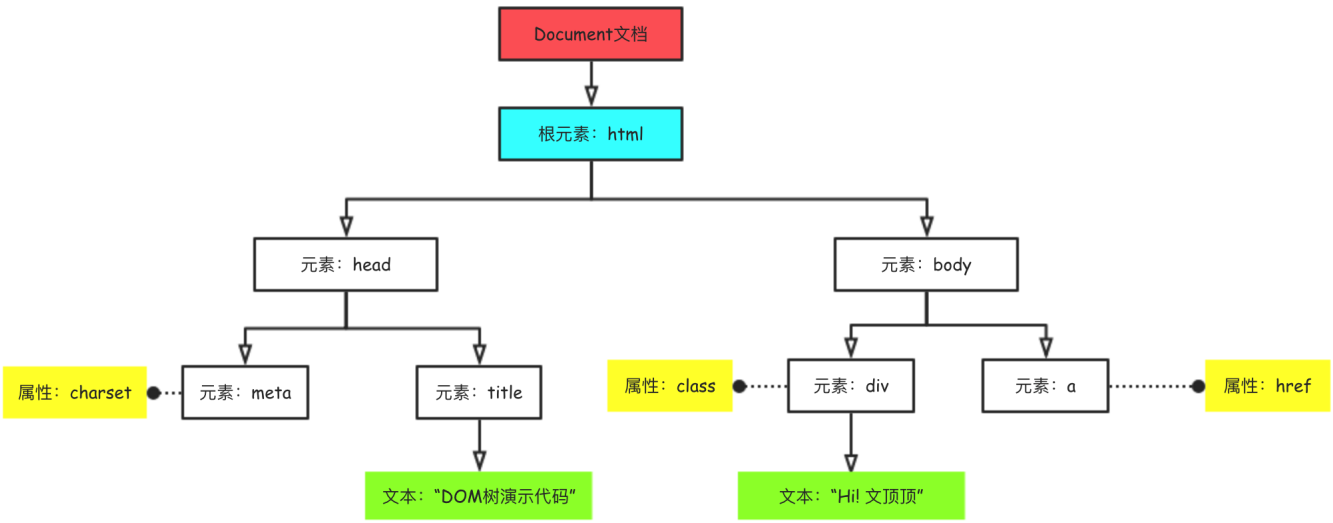
DOM（全称为Document Object Model）即 文档对象模型 ，是用于表示和操作HTML或XML文档内容的一套基础API(Application Programming Interface)。DOM把整个页面映射为一个多层节点结构，HTML 或 XML页面中的每个组成部分都是某种类型的节点，这些节点又包含着不同类型的数据。当网页被加载时，浏览器会内部的引擎会根据DOM模型，将结构化文档（比如HTML和XML）解析成一系列的节点，再由这些节点构建出一种树状结构（DOM Tree）。

> ✧ 有时候我们可能会看到DHTML这个专业术语：DHTML是动画HTML的简称，其并不是一项新的技术，而是描述HTML CSS JavaScript技术组合的术语。它曾被认为是HTML/XHTML CSS和JavaScript相结合的产物，像今天的HTML5，但真正凝聚它们的是DOM。

下面给出一段简单的HTML示例代码和对应的DOM树结构图。图示中的的方框代表着文档中的一个个节点，每个方框（节点）暨一个Node对象，所有这些节点组成了DOM树。

```HTML
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>DOM树演示代码</title>
</head>
<body>
<div class="className">Hi！ 文顶顶</div>
<a href="http://www.wendingding.com"></a>
</body>
</html>
```

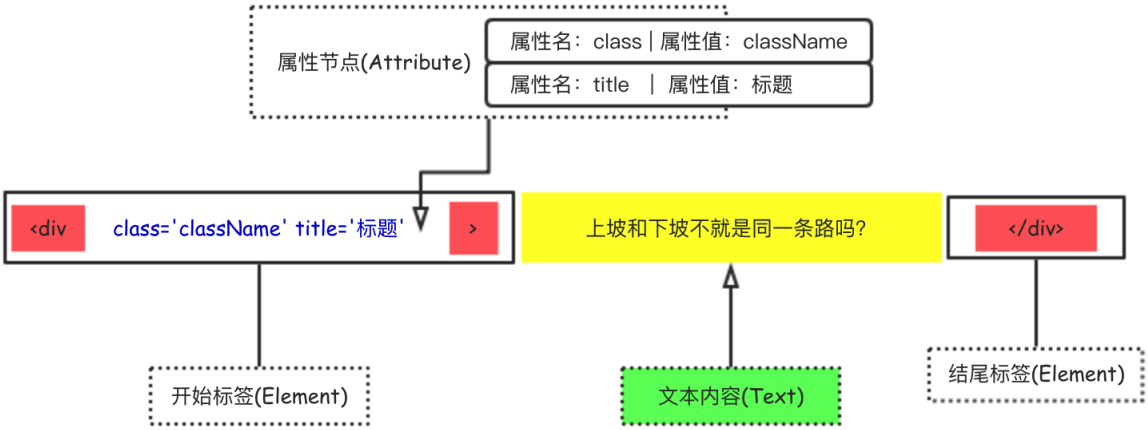


节点的类型

HTML页面中拥有众多类型的节点，不同类型的节点其表示和操作的方式有很大的差异，下面分别列出：

> ❏ Text：标签之间或标签包含的文本内容
> ❏ Comment：HTML或XML中的注释
> ❏ Element：网页的各种HTML标签，如a标签 div标签等
> ❏ Document：整个DOM树的根，代表整个文档
> ❏ Attribute：网页元素的属性节点
> ❏ DocumentType：文档类型（doctype）标签
> ❏ DocumentFragment：文档的片段，如果感觉费解请移步MDN-DocumentFragment

尽管在HTML页面中存在着如此众多类型的节点，但我们真正需要关注的主要还是： **元素节点** 、 **属性节点** 和 **文本节点** 。在(HTML|XHTML)文档中，文本节点总是被包含在元素节点的内部，属性节点用来对元素做出更具体的描述。

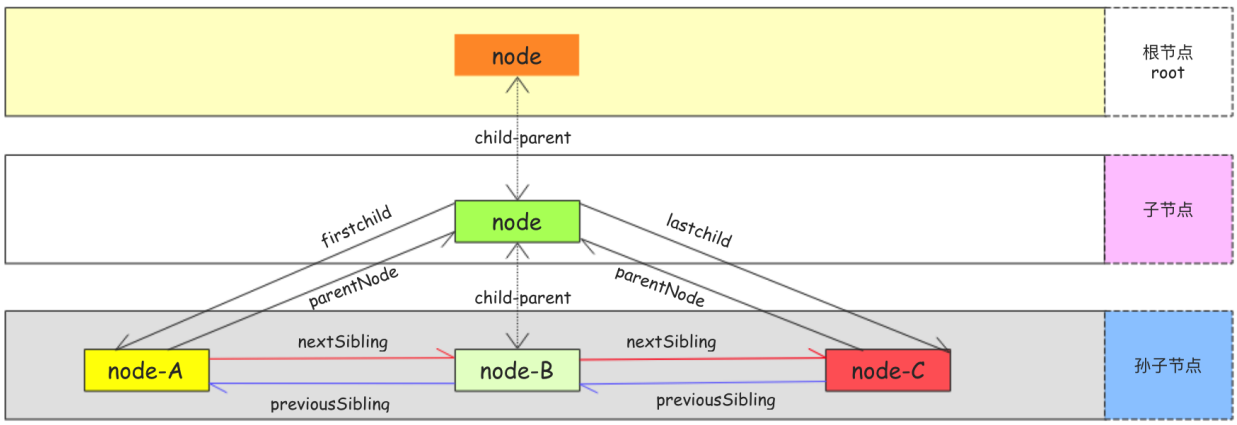


HTML

在上面的图示中，我们提供了一个div标签，该标签拥有“上坡和下坡不就是同一条路吗”文本内容和两个属性节点。
① 个div标签由开始标签和结尾标签组成，本身是Element类型的。
② “上坡和下坡不就是同一条路吗”作为div标签的文本内容，本身是Text类型的。
③ div标签中的class和title是属性节点(key-value)，本身是Attribute类型的。

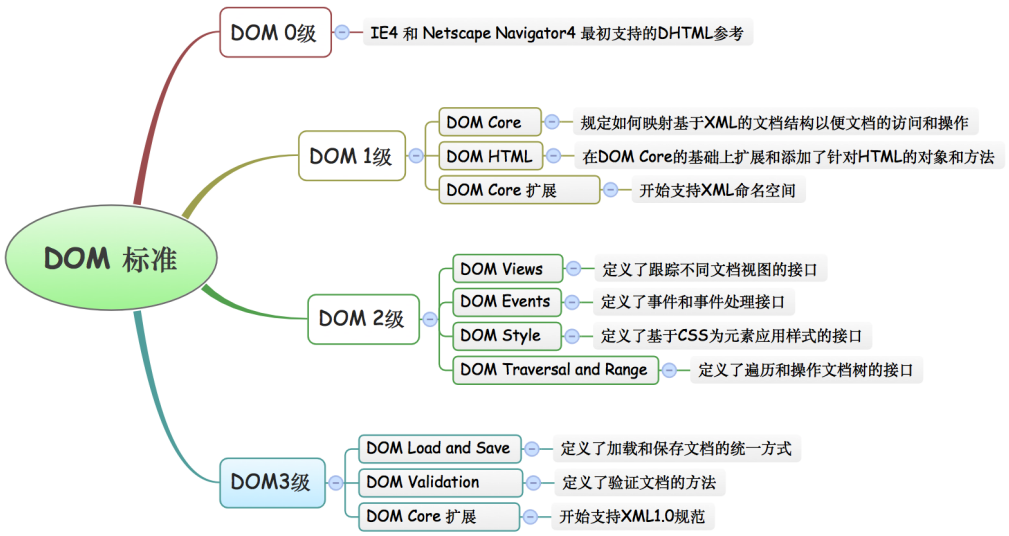
节点关系

DOM中节点的关系主要有 子节点 父节点 后代节点(子孙节点) 祖先节点 兄弟节点 这几种情况，我们可以通过下面的示意图先对DOM中的节点关系有一个简单的认识。



DOM标准

从 IE4 和 Netscape Navigator4 开始，它们开始分别支持不同形式的DHTML(Dynamic HTML)为Web技术的发展带来了很大的便利，但是也因为微软和 Netscape 在DHTML的技术发展方面各持己见，导致编写的网页如果要同时运行在它们的浏览器上面需要做大量的适配工作，它们甚至互不兼容。因此，负责制定Web通信标准的 W3C(World Wide Web Consortium)开始着手规划DOM规范，DOM规范有DOM0、DOM1、DOM2和DOM3等4个级别，具体范围可以参考下图。



规范说明

目前DOM规范的0级、1级和2级基本上已经被主流浏览器全部支持，DOM3级被部分支持(IE9+已经全部支持DOM1、2、3)。此外需说明的是除DOM规范外还存在一些扩展性的规范，比如 HTML5 、 Selectors API 和 Element Traversal 以及 SVG (Scalable Vector Graphic) 规范等，在支持(兼容)这些规范的浏览器中都可以使用它们提供的标准API。下面给出相关规范的文档地址，详情请自行查阅。

DOM1 规范
DOM 2规范
DOM 3 规范
Selectors API规范文档
HTML5 规范文档
Element Traversal 规范文档
Scalable Vector Graphic规范
ECMAScript-262/2018/6

2.0 Node && Element && nodeType

Node（节点）和 Element（元素节点）是严格区分的。也就是说Node和Element不能简单的混为一谈，因为很多人都搞不清楚它们的关系，所以这里单独拿出来讨论。

Node 节点，表示构成DOM树的最小组成部分。换句话来说，在页面中不论是元素节点、文本节点还是注释或者别的东西本质上都是Node节点。

Element 元素节点，是Node节点中的一种类型。

通俗的来讲，node节点就像人一样，是一种基本的类型。（大哲学家柏拉图对人的定义是：人是两腿无毛会直立行走的动物：） 而人这种基本类型中，又存在着小孩、中年人、老年人、学生、教师、司机、男人、女人等种种具体的类型。

对应到这里的关系，那么Element其实是node的一种更具体的类型。不止Element，像Text、Comment以及Attribute等等这些其实都是特殊的Node，它们拥有自己的类型常量（TEXT_NODE、COMMENT_NODE以及

ATTRIBUTE_NODE）用于区分彼此。文档中所有的node节点都拥有nodeType属性，我们可以通过该属性的值来确定节点的具体类型，下面列出对应关系。

类型常量	数值	节点类型	类型常量	数值	
ELEMENT_NODE	1	元素节点（Element）	PROCESSING_INSTRUCTION_NODE	7	XML文档声明
ATTRIBUTE_NODE	2	属性节点（Attribute）	COMMENT_NODE	8	注释节点（Comment）
TEXT_NODE	3	文本节点(Text)	DOCUMENT_NODE	9	Document节点，代表整个文档
CDATA_SECTION_NODE	4	CDATASection节点	DOCUMENT_TYPE_NODE	10	文档声明
ENTITY_REFERENCE_NODE	5	XML的实体参考节点	DOCUMENT_FRAGMENT_NODE	11	文档片段
ENTITY_NODE	6	XML <!ENTITY ...>节点	NOTATION_NODE	12	XML <!NOTATION ...>节点

HTML

```
# 可以直接在开发者工具的控制台中像下面这样检测和验证节点的类型
document.body.nodeType                                  //输出结果为1
document.body.ELEMENT_NODE                               //输出结果为1
document.body.ELEMENT_NODE == document.body.nodeType //输出结果为true

# 需要注意的是ELEMENT_NODE是常量
```

NodeList 和 HTMLCollection类型

相信很多开发者都有这样的经验，**“我们通过节点的childNodes属性获取的结果和children属性获取的结果是不一样的”**。下面我们通过一段简短的代码来说明它们的不同。

HTML

```
...
<div id="demoID">
    我是测试的文字---A！！
    <div>div1</div>
    <div>div2</div>
    <div class="className">div3</div>
    <!--注释的内容:后面跟span标签-->
    <span>我是span</span>
    我是测试的文字---B！！
</div>
<script>
    var oDiv = document.getElementById("demoID");
    console.log("元素节点的children属性 == HTMLCollection类型");
    console.log(oDiv.children);
    console.log(oDiv.children.length);
    console.log("元素节点的childNodes属性 ==  NodeList类型");
    console.log(oDiv.childNodes);
    console.log(oDiv.childNodes.length);
</script>
...
```

```
元素节点的children属性 == HTMLCollection类型
▶ HTMLCollection(4) [div, div, div.className, span]
4
元素节点的childNodes属性 ==  NodeList类型
▶ NodeList(11) [text, div, text, div, text, div.className, text, comment, text, span, text]
11
```

通过代码的执行情况可以发现，元素节点(这里为id为demoID的div元素)的children属性得到的是HTMLCollection类型的伪数组，而childNodes属性得到的是NodeList型的伪数组。[注意：它们是伪数组的结构，可以遍历但非真正意义上的数组]。

NodeList是Node集合，而HTMLCollection可以认为是Element的集合。NodeList、NameNodeMap和HTMLCollection它们的结构类似，保存的都是基于DOM结构动态查询的结果，而非快照，故而在执行遍历操作的时候需要注意无限循环的问题。

> ! 通常来说Document和HTMLDocument以及Element类型与和HTMLElement类型是严格区分的。Document类型代表一个HTML或XML文档，Element类型代表该文档中的一个元素。而HTMLDocument和HTMLElement通常只针对HTML文档和其元素。

3.0 DOM操作基础

节点基本操作

节点的获取

HTML

```
<div id="box">
  <li>测试1</li>
  <li>测试2</li>
</div>
<div class="box1" name="test">box1-1</div>
<div class="box1">box1-2</div>
<form action="" name="formTest"></form>
<script>

  /*获取元素节点的方法
  * [1] document.getElementById(id)      通过id来获取标签(1)
  * [2] getElementsByTagName(tagname)    通过标签名称来获取标签(n)

  * [3] getElementsByClassName()         通过class来获取标签(n)
  * [4] document.getElementsByName()     通过name属性节点来获取标签(n)
  * */

  var oDiv  = document.getElementById("box");           //div#box
  var oDivs = document.getElementsByClassName("box1");  //[box1,box1]

  /*提示：直接在已有标签内部查找，速度更快*/
  var oLis  = oDiv.getElementsByTagName("li");          //[li,li]
  var oDivN = document.getElementsByName("test")        //[div.box1]

</script>
```

说明 上面代码中展示了获取节点的常用方式，此外 Selectors API 标准还为我们提供了两个非常强大的基于CSS选择器查询DOM节点的方法。它们分别是 `querySelector()` 和 `querySelectorAll()` 方法，均接收一个CSS选择器作为参数，目前 `IE8+\Firefox 3.5+\ Safari 3.1+\Chrome和Opera 10+` 均支持它们。

节点的属性

`nodeType` 获取节点的类型，元素为1，属性为2，文本为3
`nodeName` 获取节点的名称，元素节点返回大写标签名，属性节点返回属性名，文本节点返回#text
`nodeValue` 获取节点的值，元素节点固定返回 `null` ，属性节点返回属性值，文本节点直接返回内容

节点的创建和插入

```
/*节点创建的相关方法
 * [1] document.createElement(type)           创建元素节点
 * [2] document.createTextNode(text)          创建文本节点
 * [3] document.createAttribute()             创建属性节点
 *
 * 节点插入的相关方法
 * [1]  ele.appendChild(node)              把node插入到ele子节点的末尾
 * [2]  ele.insertBefore(newNode,node)     在ele的子节点node前插入新的子节点newNode
 * [3]  ele.setAttributeNode(attrNode)     在指定元素中插入属性节点
 * */

/*01-创建div标签并插入到页面中*/
var oDiv = document.createElement("div");

/*02-给div创建文本节点*/
var oText = document.createTextNode("滴答滴答~");

/*03-把文本节点插入到div中*/
oDiv.appendChild(oText);

/*04-创建test属性节点*/
var oAttr = document.createAttribute("test");

/*05-设置属性节点的值*/
oAttr.nodeValue = "test的value";

/*06-把属性节点设置到div标签中*/
oDiv.setAttributeNode(oAttr);

/*07-创建span元素节点*/
var oSpan = document.createElement("span");

/*08-把span标签插入到oDiv里文本节点的前面*/
oDiv.insertBefore(oSpan,oText);

/*09-把div标签插入到body子节点的末尾*/
document.body.appendChild(oDiv);
console.log(oDiv);

/*备注：测试各种类型node的nodeType、nodeName和nodeValue属性*/
/*备注：nodeName和tagName是等价的，均为全部大写的标签名(类型)*/
console.log("元素节点-oSpan.nodeType = " +oSpan.nodeType);  //oSpan.nodeType = 1
console.log("属性节点-oAttr.nodeType = " +oAttr.nodeType);  //oAttr.nodeType = 2
console.log("文本节点-oText.nodeType = " +oText.nodeType);  //oText.nodeType = 3

console.log("元素节点-oSpan.nodeName = " +oSpan.nodeName);  //oSpan.nodeName = SPAN
console.log("属性节点-oAttr.nodeType = " +oAttr.nodeName);  //oAttr.nodeType = test
console.log("文本节点-oText.nodeName = " +oText.nodeName);  //oText.nodeName = #text
```

```
console.log("元素节点-oSpan.nodeValue =" +oSpan.nodeValue);//oSpan.nodeValue = null
console.log("属性节点-oAttr.nodeValue =" +oAttr.nodeValue);//oAttr.nodeValue = test的value
console.log("文本节点-oText.nodeValue =" +oText.nodeValue);//oText.nodeValue = 滴答滴答~
```

```
▼<div test="test的value">
  <span></span>
  "滴答滴答~"
</div>
```

说明 上面的代码中我分别演示了创建元素节点(`createElement`)、文本节点(`createTextNode`)以及属性节点(`createAttribute`)的相关方法，在创建节点后对它们执行了插入操作，主要用到了 `appendChild` 和 `insertBefore` 方法。这里需要说明的是，上面的代码仅供方法演示用，在实际的开发中用上面的这种方式来处理未免太过麻烦和复杂。通常我们会有更简单的方式来操作 `文本节点` 和 `属性节点` ，下面以渐进的方式给出两种更简单的实现方案。

```
/*01-创建div标签并插入到页面中*/
var oDiv = document.createElement("div");

/*02-创建文本节点并插入*/
oDiv.innerText = "滴滴答答~";

/*03-创建属性及节点并设置后插入*/
oDiv.setAttribute("test","test的value值");

/*04-创建span元素节点*/
var oSpen = document.createElement("span");

/*05-把span标签插入到div中文本节点前*/
oDiv.insertBefore(oSpen,oDiv.childNodes[0]);

/*06-把div标签插入到body子节点的末尾*/
document.body.appendChild(oDiv);
console.log(oDiv);
```

说明 上面的代码中我们通过 `innerText` 属性来简化了创建文本节点并插入的操作，且使用了更方便直接的 `setAttribute` 方法来简化了 创建属性节点 - 设置 - 插入到标签 的过程。

```
/*01-创建div标签并插入到页面中*/
var oDiv = document.createElement("div");

/*02-设置标签的节点和属性内容*/
oDiv.innerHTML = "<span></span>滴滴答答~";

/*03-设置div标签的属性节点*/
oDiv.setAttribute("test","test的value值");

/*04-把div标签插入到body子节点的末尾*/
document.body.appendChild(oDiv);
console.log(oDiv);
```

说明 上面的代码中我们使用 innerHTML 属性来完成了设置div标签子节点工作，该属性和 innerText 很相像，区别在于 innerHTML 在解析的时候会解析元素节点而 innerText 则解析为纯文本形态。

基于节点关系的操作

HTML

```
<body>
<div id="box">
  <span>我是spanA</span>
  <span class="span-class">我是spanB</span>
  我是div标签的文本
  <span>我是spanC</span>
</div>
<script>

 /* 父节点: parentNode
  * 父元素: parentElement
  *
  * 子节点: childNodes firstChild          lastChild
  * 子元素: children   firstElementChild   lastElementChild
  *
  * 兄弟节点: previousSibling           nextSibling
  * 兄弟元素: previousElementSibling    nextElementSibling
  */

  /*01-获取外层的div标签*/
  var oDiv  = document.getElementById("box");

  /*02-对比子节点和子元素*/
 console.log(oDiv.childNodes);          //NodeList(7)
 console.log(oDiv.children);            //HTMLCollection(3)

 console.log(oDiv.firstChild);          //#text
 console.log(oDiv.firstElementChild);   //<span>我是spanA</span>

 console.log(oDiv.lastChild);           //#text
 console.log(oDiv.lastElementChild);    //<span>我是spanC</span>

 /*03-对比兄弟节点和兄弟元素*/
  var oSpan = document.getElementsByClassName("span-class")[0];
 console.log(oSpan.previousSibling);          //#text
 console.log(oSpan.previousElementSibling);   //<span>我是spanA</span>

 console.log(oSpan.nextSibling);              //我是div标签的文本
 console.log(oSpan.nextElementSibling);       //<span>我是spanC</span>
</script>
</body>
```

说明 上面代码中简单介绍了基于节点关系来操作标签的代码，我们能看到的是DOM它为我们提供了两套方法，分别是基于 所有node类型 的方法和基于 ElementNode类型 的方法，在开发中具体使用的时候后者居多，且因为我们在操作的时候得到的返回值很多情况下都是 伪数组 结构，因此直接使用 [ ] 语法通过下标(索引)方式引用子元素(节点)会更方便些。

> DOM提供两套节点关系API是因为在 `IE9-的版本` 中childNodes属性不会返回文本标签间的空格(文本节点)和其他的浏览器中则会返回，它们的实现存在较大的差异性。DOM标准为规范元素遍历故而新增了 `firstElementChild` 、 `lastElementChild` 、 `previousElementSibling` 、 `nextElementSibling` 以及 `childElementCount` 这5个API， `children` 的情况类似。

复制 | 删除 | 检查 | 替换

HTML

```
<div class="box">
  <span>我是spanA</span>
  <span>我是spanB</span>
</div>
<script>
  /*
  * cloneNode()               克隆(复制)节点，接收布尔值参数，true表示深复制，默认为false。
  * hasChildNodes()          判断当前节点是否拥有子节点,返回布尔值。
  * removeChild(ele)         删除（并返回）当前节点指定子节点ele
  * replaceChild(new,old)    使用新的节点来替换旧的子节点
  */

  var oDiv  = document.getElementsByClassName("box")[0];
  console.log(oDiv.hasChildNodes());              //true 检查div标签是否存在子节点
  var oSpanA = oDiv.children[0];                  //获取spanA标签
  var oSpanB = oDiv.children[1];                  //获取spanB标签
  oDiv.appendChild(oSpanA.cloneNode(true));       //复制spanA标签并插入到div标签末尾
  console.log(oDiv.removeChild(oSpanB));          //<span>我是spanB</span>
  console.log(oSpanA.hasChildNodes());           //true  检查spanA是否存在子节点
  oSpanA.replaceChild(oSpanB,oSpanA.firstChild); //替换操作
</script>
```

CSS样式相关的操作

如果要通过JavaScript代码来操作标签的样式，可以在获取指定的DOM标签后直接通过style属性来获取和设置。

HTML

```
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <style>
    .box{border: 1px solid #0f0f0f;padding: 10px}
  </style>
</head>
<body>
<div class="box" style="background: #efefef">凉风有信，秋月无边</div>
<script>
  var oDiv = document.getElementsByClassName("box")[0];

  /*A 操作标签的内联样式*/
  /*01-读取标签的样式*/
  console.log(oDiv.style.background);                 //rgb(239, 239, 239)
  console.log(oDiv.style.border);                     //" "

  /*02-设置标签的样式*/
```

```
  oDiv.style.color = "red";                                  //设置标签内文字颜色(前景色)

  /*B 操作标签的样式(包括内联样式)*/
  console.log(window.getComputedStyle(oDiv).border);  //1px solid rgb(15, 15, 15)
  console.log(window.getComputedStyle(oDiv).color);   //rgb(255, 0, 0)
</script>
</body>
```

说明 在上面代码中我们用到了 `getComputedStyle(ele,pseudo)` 方法来获取元素的样式，其中 `ele` 表示要获取样式的元素，而 `pseudo` 参数是一个可选的伪元素样式字符串。此外，该方法在 `IE8-` 中存在兼容性问题，可以使用 `currentStyle` 处理兼容处理。

属性和属性节点

> 实话说 `属性和属性节点` 这是一对很难描述清楚的概念，因此我将在接下来章节花很多心思来重点讲解它们并对对比。需要明确的是它们完全是两个不同的概念，只是大多数人在大多数情况下总是把它们混为一谈，甚至搞不清楚所以然，所以就有了这么一个章节。

属性 我们把变量封装到对象中就成了属性。因此在讨论属性的时候，必须先把对象的问题搞清楚，因为 `所谓属性只能是某个对象的属性` 。那谁是对象呢？其实我们在讨论DOM的时候，DOM树的每个 `node节点` 都是对象。对象的特点是什么？对象是键值对的集合，以 `key-value` 的特定形式展示，对象是有类型的，譬如Object类型的对象、Array类型的数组对象等。如何检测对象的类型？可以通过 `typeof` 关键字查看其返回值是否是 `object` ,也可以直接调用 `Object.prototype.toString()` 方法来认祖归宗，确认其真实类型。

那么，请看DOM节点的对象特征。

HTML
```
<div class="box" title="我是标题" xxx="我是xxx">我是div</div>
<script>

  var oDiv = document.getElementsByClassName("box")[0];
  var obj  = {name:"文顶顶",age:18},arr = [1,2,3];

  /*01-检查对象类型-typeof*/
  console.log(typeof obj,typeof arr, typeof oDiv);  //object object object

  /*02-检查对象类型-toString()   返回值 [对象类型  构造函数]*/
  console.log({}.toString.call(obj));               //[object Object]
  console.log({}.toString.call(arr));               //[object Array]
  console.log({}.toString.call(oDiv));              //[object HTMLDivElement]

  /*03-对象的访问 增删改查操作*/
  console.log(oDiv.wendingding);                    //undefined

  /*03-A 添加*/
  oDiv.wendingding = "文顶顶";                       //增加wendingding属性
  oDiv.name      = "测试的名字";                     //增加logName属性
  /*03-B 查询*/
  console.log(oDiv.wendingding);                    //"文顶顶"
  /*03-C 修改*/
  oDiv.wendingding = "光头强";
  console.log(oDiv.wendingding);                    //光头强
```

```
  /*03-D 删除*/
  console.log(delete oDiv.wendingding);              //true
  console.log(oDiv.wendingding);                     //undefined

  /*03-E 在div对象身上添加并调用方法*/
  oDiv.logName = function () {
    console.log("logName() => " + this.name);
  }
  oDiv.logName();                                    //"logName() =>测试的名字"
</script>
```

看到这儿，我想你已经清楚了事实。
需要说明的是，通常我们在开发中很少会像上面这样来进行操作，而且上面代码中的 `name` 、 `logName` 以及 `wendingding` 均不会出现在div的标签结构中，而是作为对象的属性和方法直接保存。下面简单列出在DOM对象上常用的属性。

HTML

```
tagName       获取元素元素的标签名
id            设置/获取元素id属性
name          设置/获取元素name属性
style         设置/获取元素的内联样式
className     设置/获取元素的class属性
innerHTML     设置/获取元素的内容（包含html代码）
outerHTML     设置或获取元素及其内容（包含html代码）
innerText     设置或获取位于元素标签内的文本
outerText     设置(包括标签)或获取(不包括标签)元素的文本

offsetTop     当前元素离<定位父级>元素顶部的距离(如果没定位的父级，则相对于根元素html的距离)
offsetLeft    当前元素离<定位父级>元素左边的距离(如果没定位的父级，则相对于根元素html的距离)
offsetWidth   当前元素的宽度（border + padding + content）
offsetHeight  当前元素的高度（border + padding + content）
```

属性节点 的内容和我上文介绍的概念保持一致，我们可以下面的示例图来进行区分。

> **属性节点**区别于**属性**，它们被保存在元素节点(标签)对象的 `attributes属性下` ，该属性对应的是一个对象，里面保存着属性节点的键值对(`key-value`)信息。

HTML

```
<div class="box" title="我是标题" xxx="我是xxx">我是div</div>
<script>

  /*属性节点的相关操作*/
  var oDiv = document.getElementsByClassName("box")[0];

  /*01-获取所有的属性节点*/
  console.log(oDiv.attributes);              //NamedNodeMap["class","title","xxx"]

  /*02-属性节点的访问*/
  /*02-A 查询*/
  console.log(oDiv.getAttribute("class"));  //"box"
  console.log(oDiv.getAttribute("xxx"));    //"我是标题"
  console.log(oDiv.getAttribute("title"));  //"我是xxx"

  /*02-B 添加*/
  oDiv.setAttribute("test","test对应的值");
  console.log(oDiv.getAttribute("test"));   // "test对应的值"

  /*02-C 修改*/
  oDiv.setAttribute("xxx","阿鲁巴");
  console.log(oDiv.getAttribute("xxx"));    //"阿鲁巴"
  console.log(oDiv["xxx"]);                 //undefined 注意：自定义的属性节点不支持这样访问

  /*02-D 删除*/
  oDiv.removeAttribute("xxx")               //删除标签中的"xxx"属性节点
  console.log(oDiv.getAttribute("xxx"));    //null

  /*备注
  * 创建属性节点 var attr = createAttribute(attr-key);
  * 设置属性节点 attr.value = attr-value; ele.setAttributeNode(attr)
  * 获取属性节点
  * [1] ele.attributes[attr-key].value
  * [2] ele.getAttributeNode[attr-key].value
  * [3] ele.getAttribute(attr-key)
  * */
</script>
```

上面代码中展示了操作属性节点的相关方法，当然除了这些方法外，其实我们还可以通过 `attributes` 的方法来对它们进行操作，标签的 `attributes` 属性对应的是一个 `NamesNodeMap` 类型的对象，它本质上同 NodeList 和 HTMLCollection 类型一样，是一个动态的集合。需说明的是，虽然标签的attributes属性提供了 `getNameItem( )` 、 `removeNameItem( )` 、 `setNameItem( )` 以及 `item( )` 等方法也能操作属性节点，但是因为操作不是很方便因此很少使用。

> ! 关于属性和属性节点，在著名的 jQuery 框架中封装了两对方法来分别进行处理，其中 `prop()和removeProp()` 方法专门用于对属性进行操作，而 `attr()和removeAttr()` 方法则用于操作属性节点。此外，对DOM对象而言，有时候我们也可以直接通过属性来操作属性节点(如 `id、value 和 className` 等)，可以认为这是为方便操作而提供的 `天桥` 。虽然在写代码时能快速安全的完成功能即可，但有些时候能够正确的区分对象属性和属性节点是意义重大的。

通常，标签身上的标准属性节点都会有一个与之对应的属性，譬如 id 和 class（对应className）等，但非标准的属性节点则并非如此。另外，HTML5标准建议我们在给标签(ElementNode)添加非标准属性的时候总是添加 data-前缀 ，用于说明这些这些非标准属性提供的是与渲染无关的信息。在使用 data- 前缀的方式给标签添加了自定义属性后，我们可以通过标签的 dataset 属性来访问它们，该属性是一个 DOMStringMap 的实例对象，维护着一个 key-value 的映射。

```
/*<idv id="myDiv" data-info="信息" data-index="2"></idv>*/

/*00-获取页面中指定的div标签*/
var oDiv = document.getElementById("myDiv");

/*01-读取自定义属性节点的值*/
console.log(oDiv.dataset);         //DOMStringMap {index: "2",info: "信息"}
console.log(oDiv.dataset.index);  //2
console.log(oDiv.dataset.info);   //"信息"

/*02-设置|修改自定义属性节点的值*/
oDiv.dataset.index = 10;           //修改
oDiv.dataset.des   = "描述信息";
/*<idv id="myDiv" data-info="信息" data-index="10" data-des='描述信息'></idv>*/

/*03-其他操作属性节点的方式(对比)*/
/*03-A getAttribute()*/
console.log(oDiv.getAttribute("info"));          //null  注意不能生了data-前缀
console.log(oDiv.getAttribute("data-info"));     //"信息"

/*03-B attributes*/
/*说明：使用nodeName和name可以访问属性名，使用nodeValue和value访问属性值*/
console.log(oDiv.attributes.getNamedItem("data-info").nodeName); //"data-info"
console.log(oDiv.attributes.getNamedItem("data-info").nodeValue);//"信息"

/*03-C getAttributeNode()*/
console.log(oDiv.getAttributeNode("data-info"));                 //data-info="信息"
console.log(oDiv.getAttributeNode("data-info").name);            //data-info
console.log(oDiv.getAttributeNode("data-info").value);           //信息

/*04-补充：获取元素所有属性节点名称数组*/
console.log(oDiv.getAttributeNames());  //["id", "data-info", "data-index", "data-des"]
```

classList HTML5新增了一种操作元素类名的方式，即给每个标签都提供 classList属性 ，该属性的值是 DOMTokenList 类型的实例，通过该属性来操作标签的类名更简单也更安全。 classList 实例对象拥有 add() 、 contains() 、 remove() 和 toggle() 等方法来操作具体的class，如果要访问某个具体的class当然也可以通过 [ ] 语法或者是 item(index) 。

```
/*<div class="box test active" id="divDemo"></div>*/
/*00-获取页面中指定的元素*/
var oDiv = document.getElementById("divDemo");
console.log(oDiv.classList);            //DOMTokenList(3) ["box", "test", "active"]

/*01-读取元素的class*/
/*[ ]语法来读取指定的class*/
console.log(oDiv.classList[0]);         //"box"
/*item(index)方法来读取指定的class*/
```

```
console.log(oDiv.classList.item(1));    //"test"
/*使用className属性来读取元素的全部class*/
console.log(oDiv.className);            //box test active
console.log(oDiv.className.split(" ")); //["box", "test", "active"]

/*02-添加class*/
oDiv.classList.add("box");              //如class已经存在则不做任何处理
oDiv.classList.add("test2");            //若class尚不存在则添加
/*<div class="box test active test2" id="divDemo"></div>*/

/*03-检查是否存在指定的class,如果存在那么就返回true,否则返回false*/
console.log(oDiv.classList.contains("box"));    //true
console.log(oDiv.classList.contains("haha"));   //false

/*04-删除class*/
oDiv.classList.remove("test");

/*05-切换class 如果存在那么就删除，否则就添加*/
oDiv.classList.toggle("box");                   //删除
oDiv.classList.toggle("box2");                  //添加

/*<div class="active test2 box2" id="divDemo"></div>*/
```

4.0 案例

案例 动态创建表格并设置隔行变色

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="UTF-8">
        <title></title>
        <style>
                table{
                        border:1px solid #ddd;
                        border-collapse: collapse;
                        width:100%;
                        margin-top: 20px;
                }
                td{
                        border:1px solid #ddd;
                        padding:3px 5px;
                }
                .odd{background-color: #199;}
        </style>
</head>
<body>
<label for="row">请输入行数：</label><input type="text" id="row">
<label for="col">请输入列数：</label><input type="text" id="col">
<button class="btn">动态生成表格</button>
<div id="output"></div>
<script>

  /*00-监听页面的加载*/
window.onload = function(){
```

```
    /*01-获取页面中指定的元素*/
    var oRow        = document.getElementById('row');
    var oCol        = document.getElementById('col');
    var oOutput     = document.getElementById('output');
    var oBtn        = output.previousElementSibling;

    /*02-注册按钮的点击事件*/
    oBtn.onclick = function(){
          /*03-获取用户输入的行-列信息*/
          var _row = oRow.value,_col = oCol.value;

          /*04-创建表格标签*/
          var table = document.createElement('table');
          var tbody = document.createElement('tbody');

          /*05-根据行数来动态的创建tr标签并添加到tbody标签中*/
          for(var i=0;i<_row;i++){
            var tr = document.createElement('tr');

            /*设置隔行换色*/
            if(i%2 === 0) tr.className = 'odd';

            tbody.appendChild(tr);

            /*06-根据列数来动态创建rd标签并添加到tr标签中*/
            for(var j=0;j<_col;j++){
                   var td = document.createElement('td');
                   td.innerHTML = '单元格 - ' + i + j;
                   tr.appendChild(td);
                  }
          }

          /*07-把表格插入到页面中显示出来*/
          table.appendChild(tbody);
          oOutput.innerHTML = '';       // 添加前先清空#output
          oOutput.appendChild(table);
    }
}
</script>
</body>
</html>
```

请输入行数： 3 请输入列数： 5 动态生成表格

单元格 – 00	单元格 – 01	单元格 – 02	单元格 – 03	单元格 – 04
单元格 – 10	单元格 – 11	单元格 – 12	单元格 – 13	单元格 – 14
单元格 – 20	单元格 – 21	单元格 – 22	单元格 – 23	单元格 – 24

5.0 附录

Node（节点）的属性和方法

JAVA

```
------------------------
####   Node节点的主要属性
```

```
------------------------

baseURI                         // 当前网页的绝对路径
children                        // 指定节点的所有Element子节点
childNodes                      // 当前节点的所有子节点
childElementCount               // 当前节点所有Element子节点的数量

nodeName                        // 节点名称[只读]

nodeType                        // 节点类型的常数值[只读]
nodeValue                       // Text或Comment节点的文本值[只读]
innerText                       // 节点的文本内容
nextSibling                     // 紧跟在当前节点后面的第一个兄弟节点
isConnected                     // 布尔类型的值，用于检查当前节点与DOM树是否连接[只读]
textContent                     // 当前节点和它的所有后代节点的文本内容
ownerDocument                   // 当前节点所在的顶层文档对象，即Document

previousSibling                 // 当前节点的前一个兄弟节点
parentNode                      // 当前节点的父节点
parentElement                   // 当前节点的父Element节点
firstChild                      // 当前节点的第一个子节点
firstElementChild               // 当前节点的第一个Element子节点
lastChild                       // 当前节点的最后一个子节点
lastElementChild                // 当前节点的最后一个Element子节点

------------------------
####  Node节点的主要方法
------------------------

cloneNode(true);                // 克隆节点，参数传递布尔类型的值（默认为false）
hasChildNodes()                 // 布尔类型的值，表示当前节点是否有子节点
appendChild(node)               // 追加新的节点到当前节点中（插入后作为最后一个节点）
removeChild(node)               // 删除指定的子节点
isEqualNode(noe)                // 布尔类型的值，用于检查两个节点是否（完全）相等
contains(node)                  // 布尔类型的值，用于判断参数节点是否为当前节点的后代节点
normalize()                     // 清理当前节点内的所有Text节点，将去除空的文本节点并合并文本。

insertBefore(newNode,oldNode)   // 在指定子节点之前插入新的子节点
replaceChild(newChild,oldChild) // 替换节点
compareDocumentPosition(node)   // 比较当前节点与指定节点的位置关系，返回不同的掩码。

------------------------
####  ChildNode相关的方法
------------------------

ChildNode.replace()             // 替换节点
ChildNode.remove()              // 将ChildNode从其父节点的子节点列表中移除
ChildNode.before()              // 在当前标签(节点)的前面插入新的节点
ChildNode.after()               // 在当前标签(节点)的后面插入新的节点
```

Element（元素节点）的属性和方法

> ! 元素节点继承了Node的所有属性和方法，Element本身也作为通用的基类来使用。

下面列出元素节点的主要属性和方法，如果没有特别标注为[读写]的，那么默认为只读。

JAVA

```
------------------------
####  Element主要的属性
------------------------

id                          // 指定元素的id属性[读写]
attributes                  // 指定元素节点相关的所有属性集合(映射)
tagName                     // 指定元素节点的标签名（大写）
innerHTML                   // 指定元素节点包含的节点内容[读写]，区别于innerText
outerHTML                   // 指定元素节点的所有HTML代码，包括它自身和包含的的所有子元素[读写]
className                   // 元素节点的class属性值[读写]
classList                   // 元素节点所有的class属性。
dataset                     // 元素节点中所有的data-*属性。
localName                   // 元素名称本地化的结果
clientTop                   // 元素节点距离它顶部边界的高度
clientLeft                  // 元素节点距离它左边界的宽度
clientHeight                // 元素节点内部(相对于外层元素)的高度
clientWidth                 // 元素节点内部(相对于外层元素)的宽度

style                       // 元素节点的行内样式
scrollHeight                // 元素节点滚动视图的高度
scrollWidth                 // 元素节点滚动视图的宽度
scrollLeft                  // 元素节点横向滚动条距离左端的位移[读写]
scrollTop                   // 元素节点纵向滚动条距离顶部的位移[读写]
offsetHeight                // 元素节点相对于版面或由父坐标 offsetParent 指定的父坐标的高度
offsetWidth                 // 元素节点相对于版面或由父坐标 offsetParent指定的父坐标的宽度
offsetLeft                  // 元素节点相对于版面或由父坐标 offsetParent指定的父坐标的左侧位移
offsetTop                   // 元素节点相对于版面或由父坐标 offsetParent指定的父坐标的顶部位移

firstElementChild           // 获取元素节点的第一个子元素
lastElementChild            // 获取元素节点的最后一个子元素
nextElementSibling          // 获取元素节点的下一个兄弟节点
previousElementSibling      // 获取元素节点的上一个兄弟节点

------------------------
####  Element主要的方法
------------------------

[⊙] 操作属性节点相关的方法
getAttribute()              // 读取指定属性
setAttribute()              // 设置指定属性
hasAttribute()              // 返回布尔类型的值，检查当前元素节点中是否有指定的属性
removeAttribute()           // 移除指定属性

[⊙] 选择器相关方法
querySelector()             // 根据参数获取选中的所有标签中的第一个返回
querySelectorAll()          // 根据参数获取选中的所有标签返回
getElementsByTagName()      // 根据标签名来获取指定的标签返回
getElementsByClassName()    // 根据class的名称来获取指定的标签返回

[⊙] 事件相关方法
addEventListener()          // 添加事件监听的回调函数
removeEventListener()       // 移除事件监听函数
dispatchEvent()             // 触发事件
attachEvent()               // 添加事件监听的回调函数(IE 9-)
detachEvent()               // 移除事件监听函数（IE 9-）
```

```
insertAdjacentHTML()              // 指定的文本解析为HTML或XML，并将结果节点插入到DOM树中的指定位置
```

Document（文档）的属性和方法

```
------------------------
####  Document的主要属性
------------------------

document.body                           // 当前文档的<body>节点
document.head                           // 当前文档的<head>节点
document.defaultView                    // 当前文档对象所在的window对象
document.doctype                        // 当前文档关联的文档类型定义(DTD)
document.documentElement                // 当前文档的根节点
document.activeElement                  // 当前文档中获得焦点的那个元素。

document.links                          // 当前文档的所有连接集合
document.forms                          // 页面中所有表单元素的集合
document.images                         // 页面中所有图片元素的集合
document.embeds                         // 网页中所有嵌入对象的集合
document.scripts                        // 当前文档的所有脚本的集合
document.styleSheets                    // 当前网页的所有样式表的集合

document.URL                            // 当前文档的URL地址
document.cookie                         // 用来操作Cookie[读写]
document.title                          // 当前文档的标题
document.domain                         // 当前文档的域名
document.referrer                       // 当前文档的访问来源
document.location                       // 获取location对象，提供与URL相关的信息
document.readyState                     // 当前文档的状态
document.designMode                     // 控制当前文档是否可编辑[读写]
document.compatMode                     // 返回浏览器处理文档的模式
document.documentURI                    // 当前文档的URI地址
document.lastModified                   // 当前文档最后修改的时间戳
document.characterSet                   // 当前文档的字符集，比如UTF-8等

------------------------
####  Document的主要方法
------------------------

document.open()                         // 用于新建并打开一个文档
document.close()                        // 不安比open方法所新建的文档
document.write()                        // 用于向当前文档写入内容
document.writeIn()                      // 用于向当前文档写入内容，尾部添加换行符

document.createEvent(type)              // 创建事件对象
document.addEventListener()             // 注册事件监听
document.removeEventListener()          // 注销事件
document.dispatchEvent(event)           // 触发事件

document.createElement(tagName)         // 创建元素节点
document.createTextNode(text)           // 创建文本节点
document.createAttribute(name)          // 创建属性节点
document.createDocumentFragment()       // 生成一个DocumentFragment对象

document.getElementById(id)                     // 根据id值获取指定ID的元素节点返回
document.querySelector(selectors)               // 根据选择器参数获取指定的所有标签中的第一个返回
document.querySelectorAll(selectors)            // 根据选择器参数获取指定的所有标签返回
```

```
document.getElementsByTagName(tagName)          // 根据标签名称获取指定的所有标签返回

document.getElementsByClassName(className)      // 根据class名称获取指定的所有标签返回
document.getElementsByName(name)                // 根据name获取拥有指定name属性的元素节点返回
document.elementFromPoint(x,y)                  // 返回位于页面指定位置最上层的Element子节点
```