BOM 和 DOM

`JavaScript基础`

什么是 BOM？

1.浏览器对象模型 Browser Object Model
2.BOM的核心对象是window，同时window也是JavaScript的全局对象
3.BOM主要负责来对浏览器窗口进行操作和窗口与窗口之间的交互

主要包括：
navigator 导航器对象、history 历史对象、screen、显示器对象、location 对象、对话框、定时器、document

navigator

> - appCodeName 返回浏览器的代码名
> - appName 返回浏览器厂商名称
> - appVersion 返回浏览器的平台和版本信息
> - cookieEnabled 返回浏览器是否开启cookie的布尔值
> - platform 返回浏览器的操作系统平台
> - userAgent 返回客户机发送到服务器的 user-agent头部信息

> userAgent 可以用来检测当前浏览器型号和版本

---

history

> - back() 返回前一个url

- forward() 返回下一个url
- go(index) 返回具体的某个页面，正数是前面的，负数是后面的，超出返回 undefined

---

screen

- availHeight 返回显示器的可用高度
- availWidth 返回显示器的可用宽度
- Height 返回屏幕的像素高度
- Width 返回屏幕的像素宽度
- colorDepth 返回屏幕的颜色位数

```
1.  window.screen.width/height //在移动端可以获取到设备的屏幕宽高
```

注意这些属性都是只读的。

---

location

- 属性
  - hash 设置或者返回从#开始的URL
  - host 返回主机和当前URL的端口号
  - hostname 返回不带端口号的服务器名称
  - href 设置或者返回完整的URL，location对象的toString()方法也可以返回这个值
  - pathname 设置或者返回URL的路径部分
  - search 返回从?到#号之间包括?的URL查询字符串
  - port 返回URL中指定的端口号，如果不存在返回 ""
- 方法
  - assign(url) 打开新的URL并在浏览器历史记录里生成一条记录，如果用 location.href 或者 window.location 也是调用这个方法
  - reload() 重新加载当前页面 (刷新)
  - replace() 用新的文档替换当前文档，（替换历史记录）

定时器

setInterval(fn,delay) ===> clearInterval(index)

setTimeout(fn,delay) ===> clearTimeout(index)

弹窗、对话框

> - alert()
> - confirm() 确定返回true 取消返回false
> - prompt() 确定返回输入的文本 取消返回null

焦点事件

> - focus 窗口 聚焦
> - blur 窗口失焦

```
// 应用：定时器的开关
window.onfocus = function(){};
window.onblur = function(){};
```

窗口尺寸和滚动条

> - window.innerWidth/window.innerHeight 浏览器文档的可视宽高
> - window.outerWidth/window.outerHeight 浏览器窗口的可视宽高
> - window.pageXOffset/window.pageYOffset 浏览器滚动条的滚动距离

- 如果需要兼容按照下面的做法:(以纵向滚动条为例)

```
var sTop = window.pageYOffset || document.body.scrollTop || document.documentElement.scrollTop
```

- scrollTo(x,y) 方法可以将滚动条移动到指定位置

- 此方法只在文档加载的时候可以自动触发，文档加载完成，页面刷新是无法触发的，解决办法：放到定时器里。

```
(function (win){
    var timer = setInterval(function(){
        if(win.pageYOffset > 0){
            win.scrollTo(0,0);
        }else{
            clearInterval(timer);
        }
    },20)
})(window)
```

DOM

1.文档对象模型 Document Object Model
2.一套HTML和XML提供的API（应用程序编程接口）
3.DOM描述了一个层次化的节点，允许添加、移除、修改页面中的某个部分。

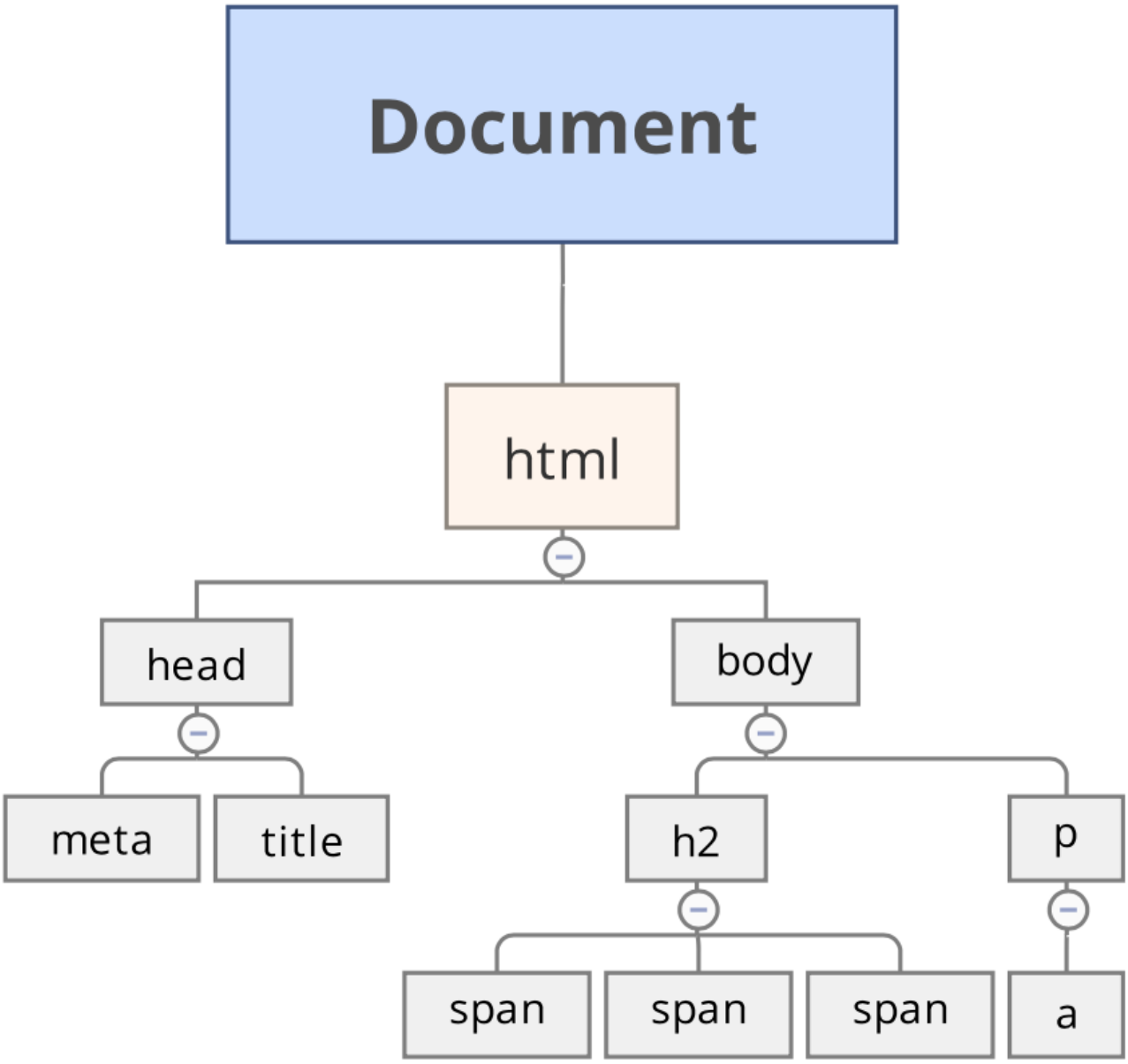
节点层次

DOM可以将任何HTML或者XML文档描绘成一个由多层节点构成的结构。节点分为不同的类型，这里咱们主要学习HTML中的节点类型。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
    <h2>
        <span></span>
        <span></span>
    </h2>
    <p>
        <a href=""></a>
    </p>
```

```
15.   </body>
16.   </html>
```

可以将上面这个简单的HTML文档表示为一个层次结构：



DOM1级定义了一套Node接口，这套接口将由DOM的所有的节点来实现。在JS中这套接口是由Node接口来实现的（不懂得话，可以无视这句话）。

> 每个节点都有一个nodeType属性，用来判断这个节点是个什么类型

主要要掌握的节点类型：

> - 元素节点 对应数字 1 #Element null
> - 属性节点 对应数字 2 #attr value
> - 文本节点 对应数字 3 #text value
> - 注释节点 对应数字 8 #comment value
> - 文档节点 对应数字 9 #document null

获取元素的子节点

> 每个节点都有一个childNodes方法，用来获取当前元素的所有子节点(可以用item()也可以用[])

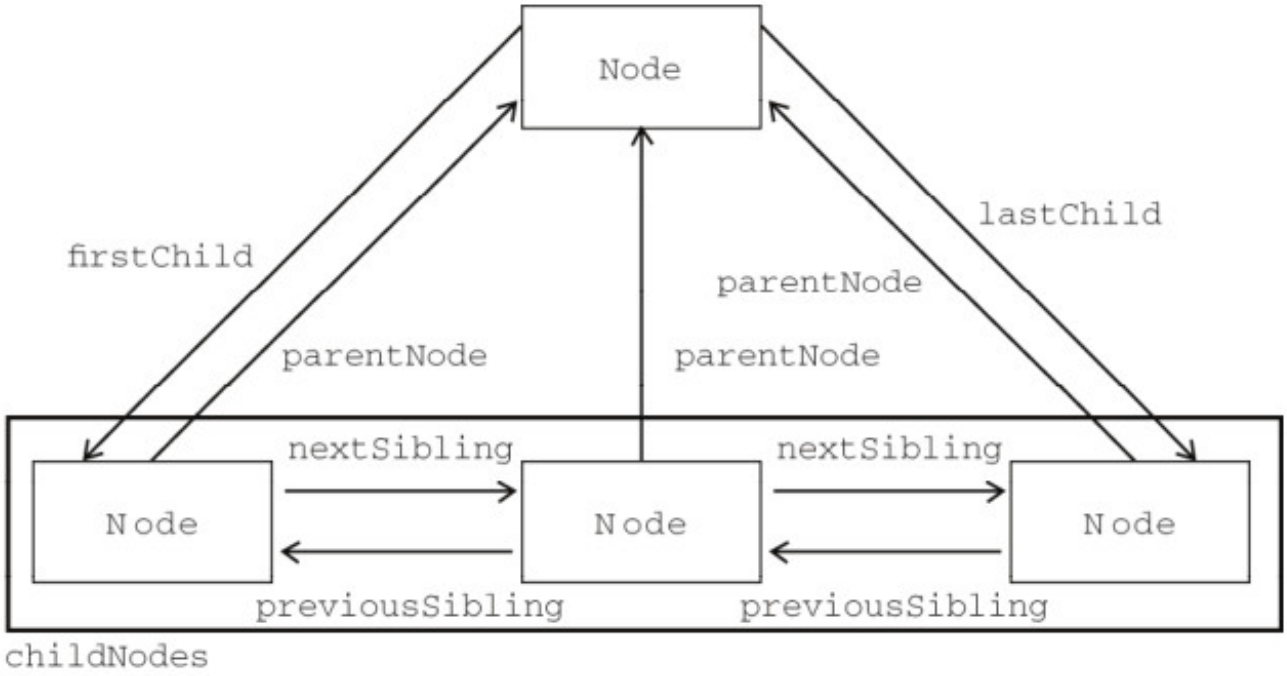
```
<body>
    <div id="box">
        <!-- 注释 -->
        <span>1</span>
        <span>2</span>
        <span>3</span>
    </div>
    <script>
        var box = document.getElementById('box');
        var childs = box.childNodes;

        console.log(childs.length);  //9
    </script>
</body>
```

用 childNodes 获取到的是当前一层的所有子节点，包括文本和注释节点。

> 每个节点有一个childNodes属性，其中保存着一个NodeList对象,NodeList是一种类数组对象，用于保存一组有序的节点，可以通过为止来访问这些节点。请注意，虽然可以通过方括号[]的语法来访问NodeList的值，而且这个对象也有length属性，但它并不是Array的实例。NodeList对象的独特之处在于，它实际上是基于DOM结构动态执行查询的结果，因此DOM结构的变化能够自动反应在NodeList对象中。我们常说，NodeList是由生命，有呼吸的的对象，而不是我们第一次访问它们的某个瞬间拍摄下来的一张快

> 照。
>
> 每个节点都有nodeName和nodeValue属性，分别用来获取节点的名字和节点的值，对于文档节点和元素节点，获取到的nodeValue永远都是null，对于注释节点文本节点获取到的就是它们本身。

一张图看懂各个节点之间的关系：



操作元素节点

> - 创建元素节点
>   - document.createElement(tag)
> - 创建文本节点
>   - document.createTextNode(text)
> - 全部都是 父级.方法 剪切操作 返回添加的节点 剪切后的事件依然存在
>   - appendChild(node) 向父节点最后添加节点
>   - insertBefore(new,old) 向某个元素前面添加节点，如果第二个参数是null那么它的功能就是appendChild。
>   - replaceChild(new,old) 替换节点
>   - removeChild(node) 删除节点

- 克隆节点
  - cloneNode(boolean)不写参数默认false只克隆当前节点，写true的话会进行深度复制。但是不会复制事件。

操作文本节点

- 全部都是 文本节点.方法 返回添加的节点
  - appendData(text) 将text添加到节点的末尾
  - deleteData(offset,count) 从offset指定的位置开始删除count个字符
  - insertData(offset,text) 在offset指定的位置插入text
  - replaceData(offset,count,text) 用text替换从offset指定的位置开始到 offset + count为止处的文本。

document 的一些属性和方法

- document.documentElement ===> html
- document.body ===> body
- document.image ===> 文档中所有图片的集合
- document.from ===> 文档中所有表单元素的集合
- document.links ===> 文档中所有链接a元素的集合

元素的属性操作

- ele.getAttribute('attr') 获取元素的某个属性
- ele.setAttribute('attr',value) 给元素设置属性
- ele.removeAttribute('attr') 删除元素身上的某个属性

以上这些方法操作的都是元素标签身上的属性，用 ele.attr 的是无法获取和设置的，对应设置节点属性，如果使用大写的字符，会自动转换成小写。

- ele.style获取的是style对象，getAttribute('style') 获取到的是后面的字符串。
- ele.src 获取绝对路径 getAttribute('src') 获取到的src后面的字符串

获取元素的属性集合: ele.attributes (可以用item()也可以用[],然后使用nodeName获取对应

的属性名，使用nodeValue获取元素的属性值)

HTML5新增的自定义属性 使用 data-name="value" 注意 如果是这种格式的：data-hello-world 会转换为 helloWorld ,在JS中使用ele.dataset可以获取到元素自定义属性的一个对象，这个对象不是类数组。

获取元素节点

> - node.children 获取节点下的所有元素节点
> - node.firstElementChild 获取节点下的第一个元素节点
> - node.lastElementChild 获取节点下的最后一个元素节点
> - node.previousElementSibling 获取元素的上一个兄弟节点
> - node.nextElementSibling 获取元素的下一个兄弟节点

两个动态获取元素的方法:

> - document||content.getElementsByTagName(tag)
> - document||content.getElementsByClassName(class1 class2 ...)

classList 对象

通过 元素.classList 获取到当前元素的class列表 （ 类数组 ）

- 在这个对象下有4个方法
  - add(class) 将指定的字符串添加到class列表中
  - contains(class) 判断列表中是否有某个class
  - remove(class) 移除某个class
  - toggle(class) 如果列表中已经存在就删除返回false，否则添加返回true

对表格的操作

```
<table id="table">
    <tr>
        <td></td>
        <td></td>
```

```
            <td></td>
        </tr>
        <tr>
            <td></td>
            <td></td>
            <td></td>
        </tr>
    </table>
    <script>
        var tab = document.getElementById('table');
        tab.tBodies[0].rows[0].cells[0]
    </script>
```

- 以下全部为 #tableElement.方法
  - createTHead() 创建thead元素，并自动插入
  - createTFoot() 创建tfoot元素，并自动插入
  - createCaption() 创建caption，并自动插入
  - deleteTHead() 删除thead元素
  - deleteTFoot() 删除tfoot元素
  - deleteCaption() 删除caption元素
  - insertRow(pos) 向rows合集中插入一行
- rows.insertCell(pos) 向cells合集中插入一个单元格

获取元素的宽高和位置

```
ele.getBoundingClientRect() //返回一个对象，对象里面有元素的绝对 left\top\bottom\right 以及元素的 width 和 height
```

详解 offset client scroll

- 相关参考
- http://blog.csdn.net/lzding/article/details/46371609
- http://www.jb51.net/article/32801.htm
- 代码示例：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>offsetWidth、clientWidth、width、scrollWidth区别及js与jQuery获取的方式</title>
<script type="text/javascript" src="jquery.min.js"></script>

</head>
<body style="border:1px solid #ccc">
        <div id="div" style="width:100px;height:80px;left:10px;top:8px;position:relative;border:1px solid #ccc;margin:20px 10px;padding:7px 6px;">
        </div>

    <script type="text/javascript">
        var divObj = document.getElementById("div");

        var dOffsetWidth = divObj.offsetWidth;//返回元素的宽度（包括元素宽度、内边距和边框，不包括外边距）
        var $dOffsetWidth = $(divObj).outerWidth(false);//参数为true，包括外边距

        var dOffsetHeight = divObj.offsetHeight;//返回元素的高度（包括元素高度、内边距和边框，不包括外边距）
        var $dOffsetHeight = $(divObj).outerHeight(false);//参数为true，包括外边距


        var dClientWidth = divObj.clientWidth;//返回元素的宽度（包括元素宽度、内边距，不包括边框和外边距）
        var $dClientWidth = $(divObj).innerWidth();

        var dClientHeight = divObj.clientHeight;//返回元素的高度（包括元素高度、内边距，不包括边框和外边距）
        var $dClientHeight = $(divObj).innerHeight();

        var dWidth = divObj.style.width;//返回元素的宽度（包括元素宽度，不包括内边距、边框和外边距）
        var $dWidth = $(divObj).width();//width(val)设置宽

        var dHeight = divObj.style.height;//返回元素的高度（包括元素高度，不包括内边距、边框和外边距）
        var $dHeight = $(divObj).height();//height(val)设置高

```

```
        var dscrollWidth = divObj.scrollWidth;//返回元素的宽度（包括元素宽度、内边距和溢出尺寸，不包括边框和外边距），无溢出的情况，与clientWidth相同
        var dscrollHeight = divObj.scrollHeight;//返回元素的高度（包括元素高度、内边距和溢出尺寸，不包括边框和外边距），无溢出的情况，与clientHeight相同


        console.log("dOffsetWidth:"+dOffsetWidth+",dOffsetHeight:"+dOffsetHeight+",dClientWidth:"+dClientWidth+",dClientHeight:"+dClientHeight+",dWidth:"+dWidth+",dHeight:"+dHeight+",dscrollWidth:"+dscrollWidth+",dscrollHeight:"+dscrollHeight);

        console.log("$dOffsetWidth:"+$dOffsetWidth+",$dOffsetHeight:"+$dOffsetHeight+",$dClientWidth:"+$dClientWidth+",$dClientHeight:"+$dClientHeight+",$dWidth:"+$dWidth+",$dHeight:"+$dHeight);

    /*
        注意：offsetWidth(offsetHeight)与style.width(style.height)的区别
        1. style.height 返回的是字符串，如28px，offsetWidth返回的是数值28，如果需要对取得的值进行计算，用offsetWidth比较方便;如果拿到offsetWidth设置style.left的值，需加'px'。
        2. style.width/style.height与scrollWidth/scrollHeight是可读写的属性，clientWidth/clientHeight与offsetWidth/offsetHeight是只读属性
        3. style.height的值需要事先定义，否则取到的值为空。而且必须要定义在html里，如果定义在css里，style.height的值仍然为空，但元素偏移有效；而offsetWidth则仍能取到。
    */

    /*
        总结：
        1、通过style.width(style.height)或者jQuery的$(divObj).width()/$(divObj).height()获取/设置元素的宽高
        2、若要获取元素包含边框的宽度，则可通过divObj.offsetWidth/divObj.offsetHeight或jQuery的$(divObj).outerWidth(false)/$(divObj).outerHeight(false)获取
        3、通过$(divObj).outerWidth(true)/$(divObj).outerHeight(true)获取带外边距的宽度
        4、通过$(divObj).innerWidth()/$(divObj).innerHeight()获取不包含边框、不包含外边距的宽度
    */
    </script>
</body>
</html>
```