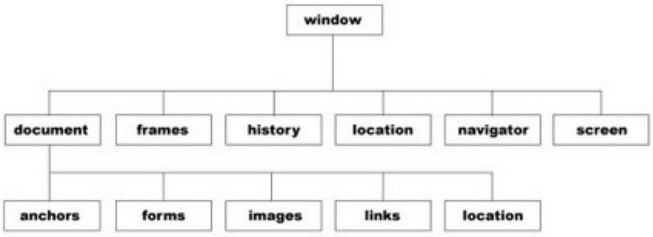
1 BOM (Browser Object Model)

1.1 介绍

IE4.0和Netscape Navigator4.0提供了一种可能操作浏览器的BOM，可以对浏览器窗口进行访问和操作。BOM由多个对象组成，其中代表浏览器窗口的window对象是BOM的顶层对象，其他对象都是该对象的子对象。

关系如图所示：



BOM的主要功能包括：

1. 弹出新浏览器窗口的能力。
2. 移动、关闭和更改浏览器窗口大小的能力。
3. 提供web浏览器详细信息的导航对象。
4. 提供浏览器载入的页面的详细信息的本地对象。
5. 提供供用户屏幕分辨率详细信息的屏幕对象。
6. 支持cookie。
7. IE对BOM进行扩展以包括ActiveX对象类，可以通过JavaScript来实现ActiveX对象。

由于BOM没有相关标准，每个浏览器都有它自己对BOM的实现方式。BOM有窗口对象、导航对象等一些实际上已经默认的标准，但对于这些对象和其他一些对象，每个浏览器都定义了自己的属性和方式。

注：由于moveBy、moveTo、resizeBy、resizeTo(对浏览器的移动和改变大小的能力)只在IE浏览器下支持，故暂不作讨论，下面只讨论非常常用的对象、方法和属性。

1.2 定时器(window对象)

1.2.1 setTimeout(定时执行)

window对象的setTimeout方法用来实现一个函数能够在指定的毫秒数之后运行。

语法如下：

```
setTimeout(executedFunction, millisecond);
```

或：

```
setTimeout("code", millisecond);
```

其中，executedFunction放入一个函数，millisecond则是设置一个以毫秒为单位的时间，并在此时间间隔之后执行executedFunction函数。

例如：

```
function execute() {
    console.info("执行函数! ");
}
//放入命名函数
setTimeout(execute, 1000);
```

又例如：

```
//放入匿名函数
setTimeout(function() {
    console.info("执行函数！");
}, 1000);
```

还可以直接放入执行代码：

```
setTimeout("console.info('执行函数！')", 1000);
```

另外，setTimeout()可以返回一个值，这个值可以传递给clearTimeout()用于销毁这个timeout定时器。

如：

```
var timer = setTimeout(function() {
    console.info(1);
}, 1000);
clearTimeout(timer);
```

1.2.2 setInterval(间隔执行)

window对象的setInterval方法与setTimeout方法很类似，只是这个方法会在指定毫秒数的间隔里重复调用。

如：

```
//setInterval也可执行放入执行代码或者命名函数的引用
setInterval(function() {
    console.info("执行函数！");
}, 1000);
```

使用clearInterval可销毁这个interval定时器。

如：

```
var timer = setInterval(function() {
    console.info("执行函数！");
}, 1000);
clearInterval(timer);
```

1.3 弹出对话框(window对象)

1.3.1 alert

alert()向用户显示一条消息并等待用户关闭对话框。

例如：

```
alert("夜深了，早点休息！");
```

但是需要注意的是，alert方法会产生阻塞，也就是说在用户关掉他们所显示的对话框之前，alert后面的代码不会执行。这就意味着在弹出一个对话框之前，代码就会停止运行。

1.3.2 confirm

confirm()也显示一条消息，要求用户单击“确定”或“取消”按钮，并返回一个布尔值。

例如：

```
var bool = confirm("May I have a date with you?");
console.info(bool);
```

需要注意的是，confirm方法也会产生阻塞。

1.3.3 prompt

prompt()同样也显示一条消息，等待用户输入字符串，并返回那个字符串。

prompt()有两种写法,分别是：

```
var inputVal = prompt("请输入一个值");
console.info(typeof inputVal);
```

或

```
var inputVal = prompt("请输入一个值", "默认值");
```

需要注意的是，prompt方法也会产生阻塞。

1.4 打开新窗口和关闭窗口(window对象)

1.4.1 open函数

使用window对象的open函数可以打开一个新的浏览器窗口（或标签页，这通常和浏览器的配置选项有关）。window.open()载入指定的url到新的或已存在的窗口中，并返回代表那个窗口的window对象。它有4个可选的参数。

1. 第一个参数是要在新窗口中显示的文档的url。如果这个参数省略了（可以是空串），那么会使用空白页面的url about:blank。
2. 第二个参数是新打开的窗口的名字（即window.name），如果使用 `_blank`、`_self`、`_parent` 或 `_top` 参数值则是指定引用的文档将要显示在新的空白窗口、自身窗口、父窗口或顶层窗口中，有点类似于a标签的target。
3. 第三个参数是非标准的，HTML5规范也主张浏览器应该忽略它，它有很多键值对可以用来设置新打开的窗口的大小位置等信息。
4. 第四个参数只在第二个参数命名的是一个已存在的窗口时才有用。它是一个布尔值，声明了由第一个参数指定的url是应用替换掉窗口浏览器历史的当前记录（true），还是应该在窗口浏览历史中创建一个新的记录（false），后者是默认的设置。

第三个参数的键值对如下：

属性名	属性值	说明
left	Number	新创建窗口的左坐标，不能为负数
top	Number	新创建窗口的上坐标，不能为负数
height	Number	设置新创建的窗口的高度，该数字不能小于100
width	Number	设置新窗口的窗口宽度，该数字不能小于100
resizable	yes,no	判断新窗口是否能通过拖动边线调整大小，默认值是no
scrollable	yes,no	判断新窗口的视口容不下要显示的内容时是否允许滚动，默认值是no
toolbar	yes,no	判断新窗口是否显示工具栏，默认值是no
status	yes,no	判断新窗口是否显示状态栏，默认值是no
location	yes,no	判断新窗口是否显示地址栏，默认值是no

例如：

```
var newWindow = open("local.html", "mycall", "");
```

或者：

```
var newWindow = open("https://www.baidu.com", "_self", "");
```

window.open()是广告商用来在你浏览网页时采用的“页面之前弹出”或“页面之后弹出”窗口的一种方法。由于对于这种烦人的弹出窗口的滥用，因此大部分浏览器都增加了弹出窗口过滤系统。因此这种open()函数可以被Firefox等一些浏览器禁用，大量的弹出窗口时对浏览者耐心的挑战，因此尽量少用弹出式窗口。

1.4.2 window.close函数

就像open()函数打开一个新窗口一样，window.close()函数将关闭一个窗口。

例如：

```
//会关闭掉窗口，firefox是清空文档
window.close();
```

有例如：

```
var newWindow = open("https://www.baidu.com", "_blank", "");
//也可以关闭已打开的窗口
setTimeout("newWindow.close();", 1000);
```

1.5 访问指定URL(window.location对象)

window对象的location对象属性有一个href属性，用来指定需要载入的页面的url。

例如：

```
location.href = "https://www.baidu.com";
```

1.6 访问历史(window.history对象)

1.6.1 history.back函数

该函数用来使页面退回到上一个浏览页面，如果该页面时第一个打开的，则该方法没有任何效果。

如：

```
history.back();//返回上一个页面
```

1.6.1 history.foward函数

该函数用来使页面前进到下一个浏览页面，前提是之前使用了back或者go方法。

如：

```
history.foward();//进入下一个页面
```

1.6.1 history.go()函数

go(num)函数可指定前进或后退多少个页面，其中的num控制前进、后退的页面数，若num为正数则为前进（如果为1则相当于foward函数），如果num为负数则为后退（如果为-1则相当于back函数）。

如：

```
history.go(-1);//返回上一个页面
```

1.7 获取客户端屏幕信息(window.screen对象)

1. window.screen.height屏幕高度，以像素记。
2. window.screen.width屏幕宽度，以像素记。
3. window.screen.availHeight可以使用的屏幕高度，不包含工具栏等，以像素记。

4. window.screen.availWidth可以使用的屏幕高度，以像素记。

2 DOM(Document Object Model)

2.1 DHTML(Dynamic HTML)

在讲解DOM之前，我们首先要来了解下DHTML。DHTML指动态HTML（Dynamic HTML）。另外，DHTML 不是由万维网联盟（W3C）规定的标准。DHTML 是一个营销术语 - 被网景公司（Netscape）和微软公司用来描述 4.x 代浏览器应当支持的新技术。

DHTML 是一种用来创建动态站点的技术组合物。对大多数人来说，DHTML 意味着 HTML 4.0、样式表以及 JavaScript 的结合物。

所以，DHTML技术是由如下几个技术组成的：

1. HTML4.0

通过 HTML 4.0，所有的格式化（信息）可移出HTML文档，并写入一个独立的样式表中。因为 HTML 4.0 可以把文档的表现从其结构中分离，我们可以在不搞乱文档内容的情况下完全地控制表现层。

2. 层叠样式表(CSS)

通过 CSS，我们得到了一种用于 HTML 文档的样式和布局模型。 由于 CSS 使开发者有能力同时控制多个网页的样式和布局，CSS 可以称作 Web 设计领域的一个突破。作为开发者，您可以为每个 HTML 元素定义样式，并把它应用到您希望的任意多的页面上。如果需要做一个全局的改变，只需简单地改变样式，Web 中所有的元素都会被自动地更新。

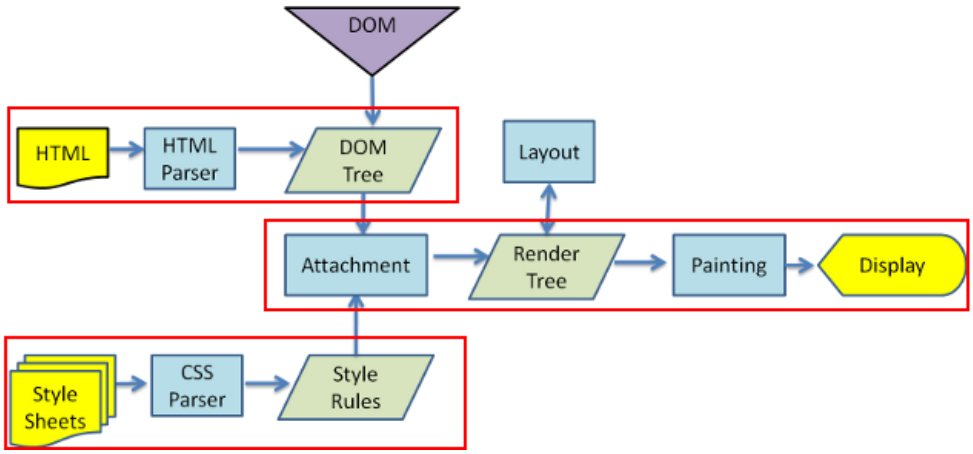
3. 文档对象模型(DOM)

DOM 指文档对象模型。 HTML DOM 是针对 HTML 的文档对象模型。 HTML DOM 定义了针对 HTML 的一套标准的对象，以及访问和处理 HTML 对象的标准方法。 “W3C 文档对象模型（DOM）是一个中立于语言和平台的接口，它允许程序和脚本动态地访问和更新文档的内容、结构以及样式”。

4. JavaScript

使您有能力编写可控制所有 HTML 元素的代码。

2.2 浏览器工作原理



一句话，浏览器将载入的html变为dom树，但是此时没有任何显示样式。所以显示的样式，都是css定义的，浏览器只会通过css来渲染视图样式。那么浏览器会在分别加载好dom树和计算形成css最终样式之后，在一起渲染成一个完整的页面呈现出来。

2.3 DOM介绍

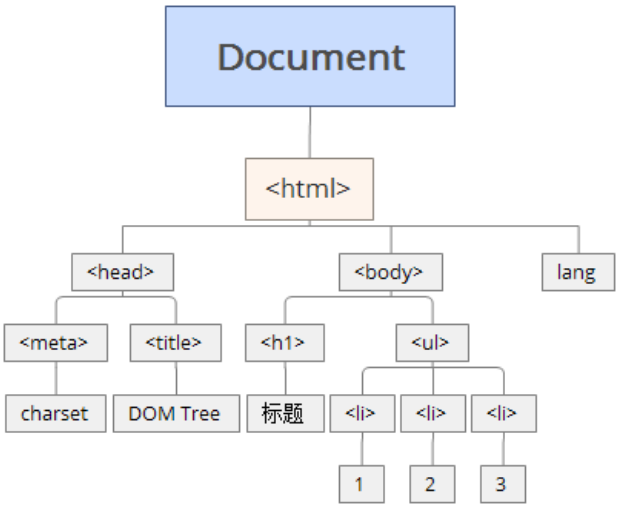
2.3.1 DOM树

为了使我们能够通过编程的方式来控制网页，W3C组织提出了文档对象模型DOM（Document Object Model）。需要知道的是，DOM可以以一种独立于平台和语言的方式访问和修改一个文档的内容和结构。换句话来说，这是表示和处理一个HTML或XML文档的常用方法。另外，DOM的设计是以对象管理组织的规约为基础的，因此可以用于任何编程语言。

例如，HTML有代码如下：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>DOM Tree</title>
</head>
<body>
    <h1>标题</h1>
    <ul>
        <li>1</li>
        <li>2</li>
        <li>3</li>
    </ul>
</body>
</html>
```

那么次文档结构被解析成的DOM树如下：

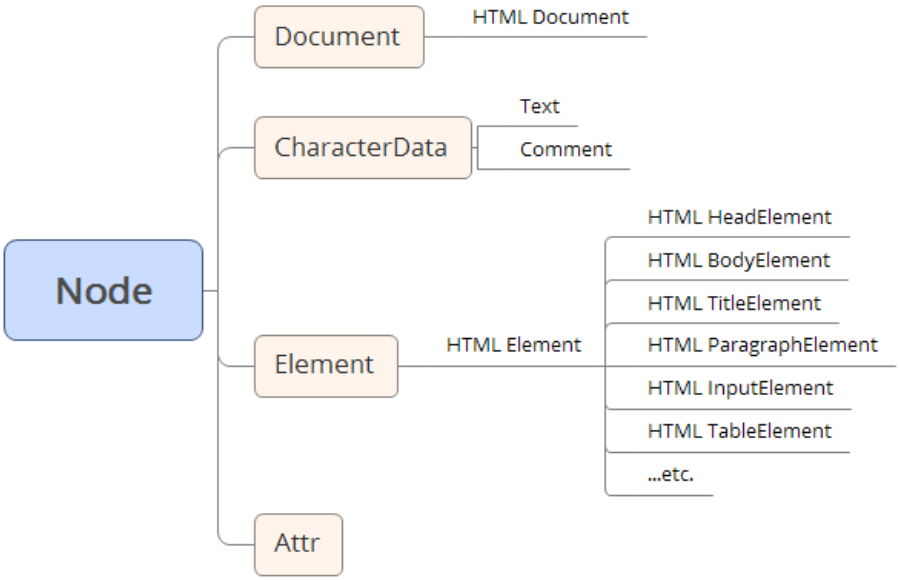


浏览器将网页加载到内存时，首先在内存中为整个文档创建一个Document对象，然后从文档中出现的第一个标记开始，按照XHTML的层次结构，一个一个的加载。每加载一个标签、注释、文本或属性，就将其当做节点(node)。当网页被装载到内存后，实际上在内存中按照XHTML层次结构，形成了一个倒挂的节点对象树结构。这个树状结构，在DOM属于中被称为DOM树。

2.3.2 DOM节点类型

- 文档节点 Document - 根节点。
- 文档类型节点 DocumentType - DTD引用即<!DOCTYPE>。
- 元素节点 Element - 标签。
- 文本节点 Text - 标签中的文本或CDataSection内包含的普通文本。
- 属性节点 Attr - 元素的属性。
- CDataSection - 通常是文本节点和注释节点的父类。
- 注释节点 Comment - 注释。

如图：



2.3.3 节点的属性和方法

属性/方法	类型/返回类型	说明
nodeName	String	节点的名字
nodeValue	String	节点的值
nodeType	Number	节点的类型常量之一
ownerDocument	Document	所属文档
firstChild	Node	childNodes列表中的第一个节点
lastChild	Node	childNodes列表中的最后一个节点
childNodes	NodeList	节点的列表
previousSibling	Node	前一个兄弟节点
nextSibling	Node	后一个兄弟节点
hasChildNodes()	Boolean	是否含有子节点
attributes	NamedNodeMap	属性集合
appendChild(node)	Node	将node添加到childNodes的末尾
insertBefore(newnode, refnode)	Node	在childNodes中的refnode之前插入newnode
removeChild(node)	Node	从childNodes中删除node
replaceChild(newnode, oldnode)	Node	将childNodes中的oldnode替换成newnode

2.3.4 DOM节点类型的常量和值

节点类型	节点类型常量	节点类型值
元素节点	Node.ELEMENT_NODE	1
属性节点	Node.ATTRBUTE_NODE	2

文本节点	Node.TEXT_NODE	3
CDATA节点	Node.CDATA_SECTION_NODE	4
注释节点	Node.COMMENT_NODE	8
文档节点	Node.DOCUMENT_NODE	9
文档类型节点	Node.DOCUMENT_TYPE_NODE	10

例如：

```
//body显然为元素节点
var nodeType = document.body.nodeType;
console.info(nodeType);
```

结果：

```
1
```

3 DOM操作HTML

3.1 所有类型节点的通用操作

3.1.1 获取节点

1.利用节点关系获取节点

- childNodes - 获取所有子节点。
- firstChild - 获取第一个子节点。
- lastChild - 获取最后一个子节点。
- previousSibling - 获取上一个兄弟节点。
- nextSibling - 获取下一个兄弟节点。
- parentNode - 获取父节点。

2.直接获取节点

- document.getElementById("id") - 根据id获取节点。
- document.getElementsByTagName("div") - 根据标签名称获取节点。
- document.getElementsByClassName("class") - 根据类名获取节点。
- document.getElementsByName("name") - 根据名称获取节点。

3.1.2 创建的节点

1.创建新的节点

- createElement("div") - 创建div的元素节点。
- createTextNode("text") - 创建包含“text”的文本节点。

例如：

```
document.createElement("p");
document.createTextNode("哈哈");
```

这个时候你在页面上是看不到p标签的，因为它还没有跟dom树关联在一起，创建了一个节点一定要跟dom树发生关联。

2.克隆节点

- cloneNode(bool) - 克隆节点，bool为false时只克隆该元素节点，而bool为true时会克隆该节点和该节点的所有子节点。

3.1.3 添加节点

- appendChild(node) - 在所有子节点之后添加一个node。
- insertBefore(node, refnode) - 在refnode之前添加一个node，记住此次添加是发生在某元素的子节点上的，即在某元素的refnode子节点前面加一个node。

3.1.4 删除节点

- removeChild(node) - 删除node子节点。

3.1.5 替换节点

- raplaceChild(newnode, oldnode) - 把oldnode替换为newnode，记住此次替换是发生在某元素的子节点上的，即是把某元素的oldnode子节点替换为newnode。

3.1.6 通用操作（增删改）

- innerHTML - 某元素节点所包含的所有HTML，包括所有**元素节点** 、 **文本节点** 以及**属性节点**。

3.3 表格操作

3.3.1 新增

- tableElement.insertRow(position) - 新增一行，返回值为一个tr节点即一个rowElement，position位置从0开始计数。
- rowElement.insertCell(position) - 新增一个单元格，返回值为一个td节点即一个cellElement，position位置从0开始计数。

3.3.2 删除

- tableElement.deleteRow(position) - 删除某行，position从0开始计数。
- rowElement.deleteCell(position) - 删除某行，position从0开始计数。

3.3.3 访问

- tableElement.rows - 获取表格所包含的的所有tr元素节点所组成的集合数组。
- rowElement.cells - 获取行元素节点所包含的所有td元素节点所组成的集合数组。

3.4 表单控件操作

3.4.1 表单控件通用属性

通用属性包括value和text，即对于所有的input节点或select节点以及select下的option节点都是通用的。

- value - 值。
- text - 文本内容。

3.4.2 select下拉框

1.访问option节点

- selectElement.options - 返回该select节点内子节点option节点所组成的数组。

2.删除option节点

- selectElement.remove(position) - 基于select节点的remove方法，删除指定位置的option节点，从0开始计数。

3.select节点直接取值

- value - 使用通用属性value来取值，来获取选中的值。
- selectedIndex - 获取被选中的option节点的位置。

4.option节点取值

- value - 使用通用属性来获取值。
- text - 使用通用属性来获取文本。
- selected - 是否被选中。
- defaultSelected - 是否默认被选中。

3.4.3 radio单选框和checkbox复选框

- value - 使用通用属性获取值。
- checked - 是否被选中。

3.4.4 input输入框

- value - 使用通用的属性来取值。
- text - 使用通用属性来获取文本。

3.5 操作元素属性

- node.setAttribute("attrName", "val") - 修改或新增某属性"attrName"的值为"val"。
- node.getAttribute("attrName") - 获取某属性"attrName"的值。
- node.removeAttribute("attrName") - 删除某属性。

4 DOM操作CSS

4.1 内联样式的操作

从IE4.0开始，为页面上的每个元素节点引入了一个stlye对象属性，用这个对象来管理元素的CSS**内联样式**。DOM采用了这个方法，并将其作为访问一个元素的样式信息的标准做法。

DOM操作某样式属性的写法规范：

CSS样式属性	DOM写法
background-color	node.style.backgroundColor
color	node.style.color
font	node.style.font
font-family	node.style.fontFamily

例如：

```
var pEle = document.getElementById("p");
pEle.style.backgroundColor = "red";
```

也可写作：

```
var pEle = document.getElementById("p");
pEle.style["backgroundColor"] = "red";
```

或：

```
var divEle = document.getElementById("div");
divEle.style.setProperty("backgroundColor", "red");
```

对于浏览器兼容性写法如下：

```
node.style.webkitTransform = "";
node.style.MozTransform = "";
node.style.msTransform = "";
node.style.OTransform = "";
node.style.transform = "";
```

要记住的是：setProperty是style对象属性的方法。

4.2 样式表的操作

上一小节我们学了如何操作内联样式，但一般我们写css的时候都会在style标签里面来书写我们的css代码，包括link过来的外部样式表和在html内的style属性里面的内部样式表。我们同样可以使用DOM来操作这些样式表。

获取样式表的代码如下：

```
var cssRules = document.styleSheets[0].cssRules || document.styleSheets[0].rules;
```

注:IE浏览器是rules集合，而遵循DOM规范的浏览器是使用的cssRules。

若有HTML如下:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <style>
        div {
            font-size: 14px;
            background-color: red;
        }
        span {
            text-decoration: underline;
        }
    </style>
    <style>
        p {
            font-weight: bold;
            color: blue;
        }
    </style>
</head>
<body>
    <p>123</p>
</body>
</html>
```

获取css规则集的JavaScript代码如下：

```
var styleSheets = document.styleSheets;
for (var i = 0; i < styleSheets.length; ++i) {
    var cssRules = styleSheets[i].cssRules || styleSheets[i].rules;//获取某个style元素中所有规则集
    for (var j = 0; j < cssRules.length; ++j) {
        var cssRule = cssRules[j];//获取某个规则集
        console.info(cssRule.selectorText + " {");//获取该规则集的选择器
        for (var k = 0; k < cssRule.style.length; ++k) {
            var propertyName = cssRule.style[k];//获取属性名
            var propertyValue = cssRule.style[propertyName];//获取属性值
            console.info(propertyName + ":" + propertyValue);
        }
        console.info("}");
    }
}
```

4.3 获取最终样式

上面两节我们学到了通过DOM来获取元素的样式，需要注意的是，我们所说的样式是指的开发者所定义的样式，即不包括浏览器默认样式和用户自定义样式。但是呢你还是要注意一个问题，因为该元素可能有外部样式表的样式、内部样式表的样式或者是内联样式，这么多样式最终根据层叠算法来判定最终作用在某元素上的样式，那么我们上面的方式可能并不能很方便的就能判定某个元素最终所使用的样式到底是哪个样式，所以下面我们便会学习一下如何取选取某元素的最终样式。

4.3.1 在IE浏览器下获取最终样式

IE浏览器为多有的DOM对象提供一个currentStyle对象，该对象包含了从元素背景色到任何相关CSS规则的style对象的所有属性，通过它我们可以获取某个元素的最终样式。

```
var divEle = document.getElementById("divId");
var backgroundColor = divEle.currentStyle.backgroundColor;
```

其实很简单，相当于style的获取最终样式版。需要注意的是，这个属性是只读的，即不能修改只能读出来。

4.3.2 在遵守DOM规范的浏览器下获取最终样式

DOM对于所有的DOM对象提供了一个方法getComputedStyle()来创建一个类似于Style的对象。该方法接收两个参数，一个是要获取样式的元素，一个是伪元素，例如:hover或:first-letter（如果不需要伪元素，则让第二个参数为null）。并且，我们总可以从document.defaultView对象访问该方法，该对象用于代表当前的表现视图。

```
var divEle = document.getElementById("divId");
var backgroundColor = document.defaultView.getComputedStyle(divEle, null).backgroundColor;
//当然也可以这样写
//window.getComputedStyle(divEle, null);//显然这样更方便，因为window可以不写
```

4.3.3 最终样式兼容性处理

```
function getFinalStyle(node, propertyVal) {
    if (!node.currentStyle) {//dom规范浏览器
        return getComputedStyle(node, null)[propertyVal];
    } else {//ie8及其以下
        return node.currentStyle[propertyVal];
    }
}
var fontSizeVal = getFinalStyle(document.body, "fontSize");
console.info(fontSizeVal);
```

5 DOM事件

在我们平时所写的html代码中有很多是静态标签，它们在页面上呈现出效果是完全静态的，例如： 、<img>或<p>等等。而反观另外的一些html标签，例如：<a>、<input type="submit">、<input type="reset">等等，它们在我们点击之后会发生页面跳转、提交表单或清空表单字段，这些标签相对于静态标签会具有动态性的功能，使用它们可以与用户发生互动，我们可以使用一套代码来构成这种能与用户发生互动的模型，我们称之为事件模型。

我们的DOM事件模型是由事件源（EventTarget）、事件监听器（EventLisener）和事件类型共同组成。

5.1 默认事件

5.1.1 a标签点击事件

如有代码如下：

```
<a href="https://www.baidu.com">百度</a>
```

点击该a标签，该页面可以跳转至百度首页，在整个过程中，通过鼠标点击a标签，来触发a标签上默认的点击事件，从而发生跳转动作。

5.1.2 表单提交事件

如有代码如下：

```
<form action="https://www.baidu.com">
    <input type="submit" value="submit" />
</form>
```

点击submit按钮，该页面就也会跳转至百度首页，在整个过程中，通过鼠标点击submit按钮，来触发form表单内submit按钮的默认事件，从而发生跳转动作。

5.2 注册事件监听器一（DOM Level 0）

使用**HTML标签属性**或**DOM属性** 来注册事件监听器也叫做**DOM Level 0**事件监听器注册方式或**传统事件**处理程序指派方式。

其中，DOM Level 0是W3C文档中的官方说法。

5.2.1 通过HTML标签属性注册

有html代码如下：

```
<button onclick="execute()">click</button>
```

js代码如下：

```
function execute() {
    console.info("执行js代码");
}
```

显然，如上的方式是通过在html标签的onclick属性来注册的事件监听器。

5.2.2 通过DOM属性注册

有html代码如下：

```
<button id="btn">click</button>
```

js代码如下：

```
var btnEle = document.getElementById("btn");
btnEle.onclick = function() {
    console.info("执行js代码");
}
```

如上的方式是通过dom的onclick属性来注册事件监听器，

5.3 事件流

5.3.1 事件冒泡

5.3.2 事件捕获

5.3.3 DOM事件流

5.3.4 IE事件流

5.4 注册事件监听器二（IE浏览器）

DOM Level 0

The term "DOM Level 0" refers to a mix (not formally specified) of HTML document functionalities offered by Netscape Navigator version 3.0 and Microsoft Internet Explorer version 3.0. In some cases, attributes or methods have been included for reasons of backward compatibility with "DOM Level 0".

The DOM Level 2 Event Model is designed with two main goals. The first goal is the design of a generic event system which allows registration of event handlers, describes event flow through a tree structure, and provides basic contextual information for each event. Additionally, the specification will provide standard modules of events for user interface control and document mutation notifications, including defined contextual information for each of these event modules.

The second goal of the event model is to provide a common subset of the current event systems used in DOM Level 0 browsers. This is intended to foster interoperability of existing scripts and content. It is not expected that this goal will be met with full backwards compatibility. However, the specification attempts to achieve this when possible.

The following sections of the Event Model specification define both the specification for the DOM Event Model and a number of conformant event modules designed for use within the model. The Event Model consists of the two sections on event propagation and event listener registration and the Event interface.

A DOM application may use the hasFeature(feature, version) method of the DOMImplementation interface with parameter values "Events" and "2.0" (respectively) to determine whether or not the event module is supported by the implementation. In order to fully support this module, an implementation must also support the "Core" feature defined in the DOM Level 2 Core specification [DOM Level 2 Core]. Please, refer to additional information about conformance in the DOM Level 2 Core specification [DOM Level 2 Core].

Each event module describes its own feature string in the event module listing.

有html代码如下：

```
<button id="btn">click</button>
```

js代码如下：

```
var btnEle = document.getElementById("btn");
btnEle.attachEvent("onclick", function() {
    console.info("执行js代码");
});
```

当然，js代码还可以写成如下形式：

```
var btnEle = document.getElementById("btn");
btnEle.attachEvent("onclick", execute);
function execute() {
    console.info("执行js代码");
}
```

5.5 注册事件监听器三（DOM Level 2）

有html代码如下：

```
<button id="btn">click</button>
```

js代码如下：

```
var btnEle = document.getElementById("btn");
btnEle.addEventListener("click", function() {
    console.info("执行js代码");
}, false);
```

5.6 注册事件监听器四（兼容性处理）

5.6 事件的类型

5.6.1 鼠标事件

5.6.2 键盘事件

5.6.3 HTML事件

5.7 Event对象

5.8 阻止默认事件

5.9 阻止事件传播

5.10 通过DOM编程触发事件

6 表单验证

6.1 共同属性方法

6.2 阻止默认事件

6.3 文本框取值

6.4 下拉框取值

6.5 单/复选框取值

6.5 输入验证

6.5.1 提交验证

6.5.2 换行验证

6.5.3 输入验证

众所周知，我们可以在一边输入的时候一边就进行验证，看上去就是在输入框里面的文字，每增加或减少一个都会做一次验证，我们就对这些文字进行验证来确保用户输入结果的安全性。

IE对输入框文字改变时便触发的事件是onpropertychange，而遵循DOM事件规范浏览器触发的事件是oninput，故我们欲使用此功能的时候一定要进行兼容性处理。

```
//兼容性封装
function inputPropertyChange(inputEle, execute) {
    if (document.addEventListener) {//dom
        inputEle.addEventListener("input", execute, false);
    } else {//ie
        inputEle.attachEvent("onpropertychange", execute);
    }
}
//调用
var inputEle = document.getElementById("input");
inputPropertyChange(inputEle, function() {
```

```
    //验证处理
});
```

7 表单提交

7.1 传统表单提交

7.2 AJAX提交

7.2.1 同步异步

7 HTML5 音频视频

7.1 音频

7.2 视频

8 HTML5 拖拽

9 HTML5 本地存储

9.1 localStorage

9.2 sessionStorage

10 HTML5 Canvas

10.1 原理概述

10.2 Canvas常用API

10.2.1 描绘

10.2.2 矩形

10.2.3 样式

10.2.4 线段

10.2.5 圆弧

10.2.6 beginPath和closePath

10.2.7 变化

10.2.8 save和restore