

# 网易微专业之《前端开发工程师》

## 学习笔记

开始时间：2016.01.28

## 《DOM 编程艺术》

### 基础篇（一）

#### 一、文档树

#### DOM 对象

DOM，全称“**D**ocument **O**bject **M**odel（文档对象模型）”，它是由 W3C 组织定义的一个标准。

在前端开发时，我们往往需要在页面某个地方添加一个元素或者删除元素，这种添加元素、删除元素的操作就是通过 DOM 来实现的。

说白了，**DOM 就是一个接口（API），我们可以通过 DOM 来操作页面中各种元素，例如添加元素、删除元素、替换元素等。**

记住，DOM 就是文档对象模型，文档对象模型就是 DOM.

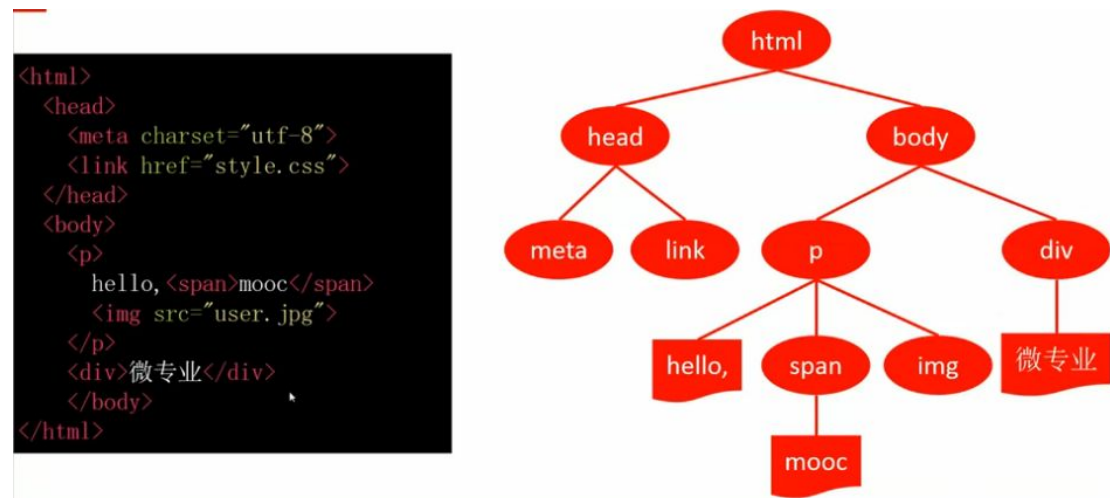
#### DOM 包含内容：

- DOM Core
- DOM HTML
- DOM Style
- DOM Event

#### DOM 结构

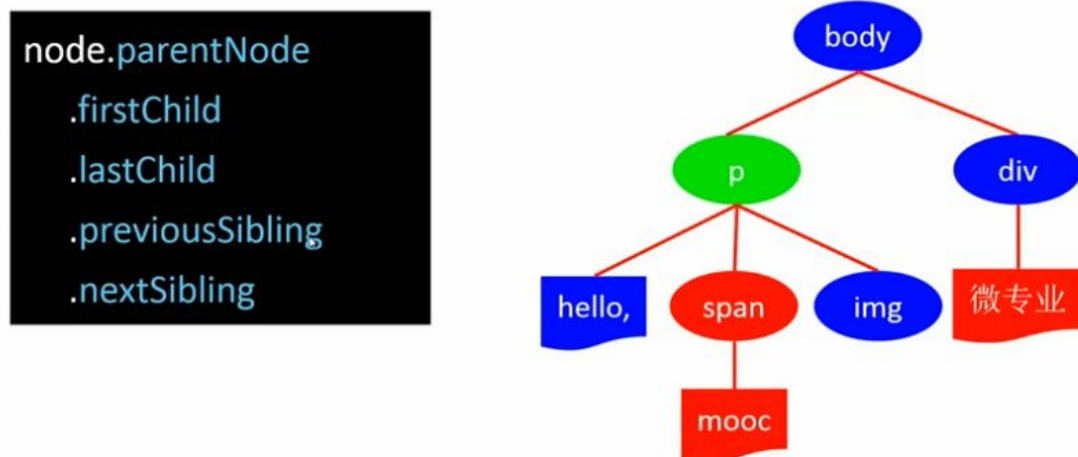
DOM 采用树形结构作为分层结构，以树节点形式表示页面中各种元素或内容。下

图左边的 HTML 文档用 DOM 树形结构表示如下：



在 DOM 中，每一个元素看成一个节点，而每一个节点就是一个“对象”。也就是我们在操作元素时，把每一个元素节点看成一个对象，然后使用这个对象的属性和方法进行相关操作。

节点遍历：

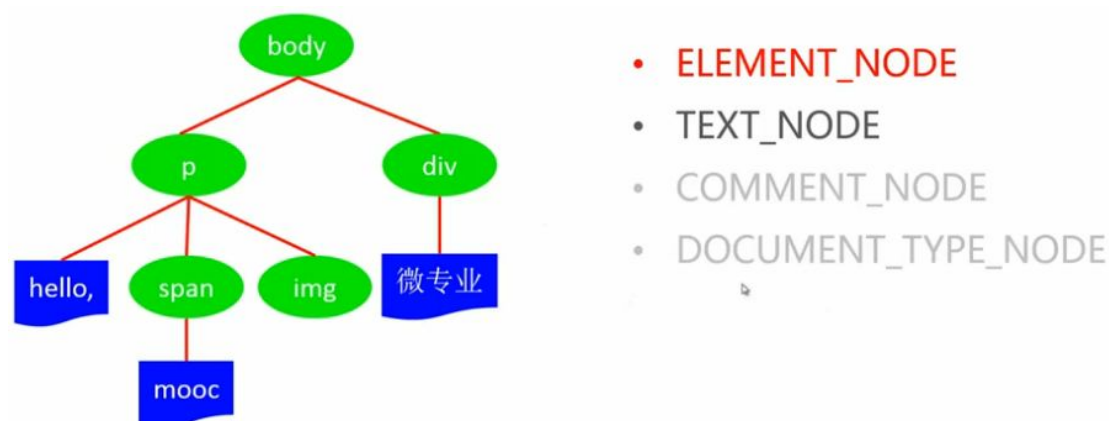


在 DOM 中，遍历 HTML 文档树，我们可以通过使用 parentNode、firstChild、lastChild、previousSibling 和 nextSibling 等属性来实现。

方法↕	说明↕
<code>childNodes</code> ↕	返回一个数组，这个数组由给定元素节点的子节点构成↕
<code>firstChild</code> ↕	返回第一个子节点↕
<code>lastChild</code> ↕	返回最后一个子节点↕
<code>parentNode</code> ↕	返回一个给定节点的父节点↕
<code>nextSibling</code> ↕	返回给定节点的下一个子节点↕
<code>previousSibling</code> ↕	返回给定节点的上一个子节点↕

其实除了上表列举的属性之外，还有 `nodeName`、`nodeValue` 和 `nodeType` 等几个比较重要的属性。

节点类型：



- 1. 元素节点：**上图中`<body>`、`<p>`、`<div>`等都是元素节点，即标签。
- 2. 文本节点：**向用户展示的内容，如`<p>...</p>`中的 `hello/mooc` 等文本。
- 3. 属性节点：**元素属性，如`<a>`标签的链接属性 `href="http://www.163.com"`。

(COMMENT\_NODE/DOCUMENT\_TYPE\_NODE 节点不经常使用。)

## 节点属性

在文档对象模型 (DOM) 中，每个节点都是一个对象。DOM 节点有三个重要的属性：

1. `nodeName`：节点的名称
2. `nodeValue`：节点的值
3. `nodeType`：节点的类型

**一、`nodeName` 属性：**节点的名称，是只读的。

1. 元素节点的 nodeName 与标签名相同
2. 属性节点的 nodeName 是属性的名称
3. 文本节点的 nodeName 永远是 #text
4. 文档节点的 nodeName 永远是 #document

## 二、nodeValue 属性：节点的值

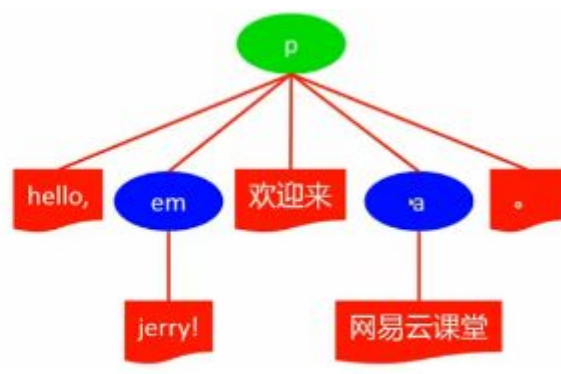
1. 元素节点的 nodeValue 是 undefined 或 null
2. 文本节点的 nodeValue 是文本自身
3. 属性节点的 nodeValue 是属性的值

## 三、nodeType 属性：节点的类型，是只读的。以下常用的几种结点类型：

### 元素类型    节点类型

元素	1
属性	2
文本	3
注释	8
文档	9

## 元素遍历



## 如何实现浏览器兼容版的 element.children

在 Javascript 中，可以通过 children 来获取所有子节点。

语法：nodeObject.children

其中，nodeObject 为节点对象（元素节点），返回值为所有子节点的集合（数组）。

例如，获取 id="demo" 的节点的所有子节点：

```
document.getElementById("demo").children;
```

children 只返回 HTML 节点，甚至不返回文本节点，虽然不是标准的 DOM 属性，但是得到了几乎所有浏览器的支持。

注意：在 IE 中，children 包含注释节点。

一般情况下，我们是希望获取元素节点，可以通过 nodeType 属性来进行筛选，nodeType==1 的节点为元素节点。

## 如何实现浏览器兼容版的 `element.children`

```
function getElementChild(element){
    if(!element.children){
        var elementArr=[];

        var nodelist = element.childNodes;

        for(var i = 0; i < nodelist.length; i++) {

            if(nodelist[i].nodeType == 1) {

                elementArr.push(nodelist[i]);
            };
        };

        return elementArr;
    }else{
        return element.children;
    }
}
```

## 二、节点操作

### 获取节点

- `getElementById`
- `getElementsByClassName`
- `getElementsByTagName`
- `querySelector`

### `getElementById`

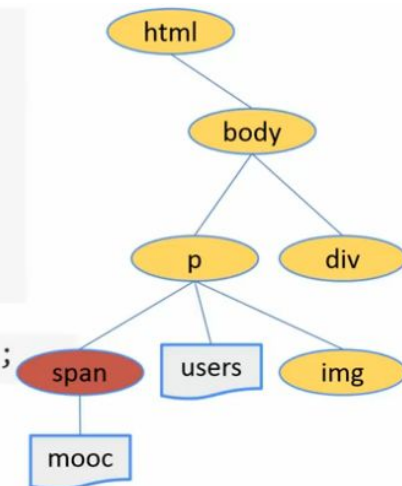
```
var elm = document.getElementById(IDString);
```

参数名称	类型	是否必选	描述
IDString	String	是	id字符串
elm	Node		被标记id为IDString的节点

```
function $(id){
    return document.getElementById(id);
}
```

示例:

```
<html>
  <head>
    <meta charset="utf-8"/>
    <link rel="stylesheet" href="css/style.css"/>
  </head>
  <body>
    <p id="p"> hello,<span id="type">mooc</span> users </p>
    <div>
      
    </div>
  </body>
</html>
```



```
var type = document.getElementById('type');
```

```
var p = $('p')
```

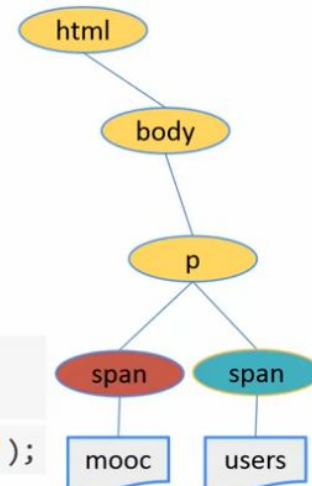
### getElementsByName

```
var collection=[elm|document].getElementsByName(classes);
```

参数名称	类型	是否必选	描述
classes	String	是	一个或多个样式（由空格分隔）
collection	HTMLCollection		live html collection

示例: 获取 span 元素

```
<html>
  <head>
    <meta charset="utf-8"/>
    <link href="/css/style.css"/>
  </head>
  <body>
    <p> hello,<span class="flag">mooc</span>
      <span class="flag z-flag">users</span> </p>
  </body>
</html>
```



```
var list = p.getElementsByClassName('flag');
```

```
var type = list[0],user = list[1];
```

```
var list= p.getElementsByClassName('flag z-flag');
```

### getElementsByTagName



`var collection = [elm|document].getElementsByTagName(tag)`

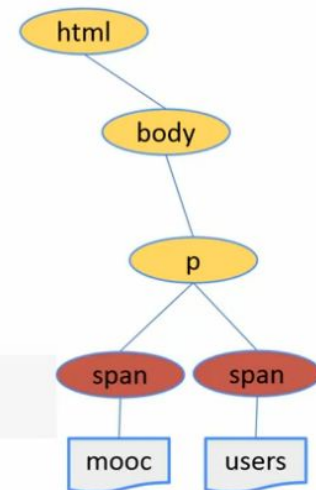
参数名称	类型	是否必选	描述
tag	String	是	标签名或*
collection	HTMLCollection		live html collection

示例：获取 span 标签

```
<html>
<head>
  <meta charset="utf-8"/>
  <link href="/css/style.css"/>
</head>
<body>
  <p> hello,<span>mooc</span>
    <span>users</span> </p>
  
</p>
<div>
</div>
</body>
</html>
```

```
var list = p.getElementsByTagName('span');
var type = list[0], user = list[1];
```

```
var list= p.getElementsByTagName('*');
```



querySelector

`var nodeList = [elm|document].querySelector[All] (selector)`

参数名称	类型	是否必选	描述
selector	String	是	css 选择器
nodeList	HTMLCollection		not live collection

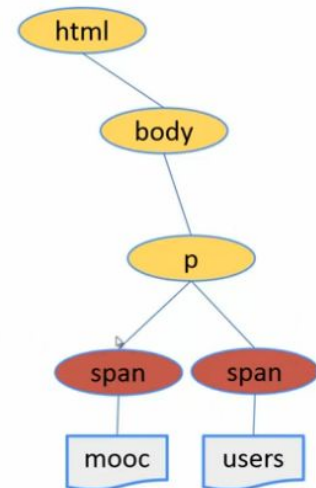
```
function $(selector){
    return document.querySelectorAll(selector)
}
```

```
var dom = document.getElementById('dom');
var liLiveList = dom.getElementsByTagName('li');
var liNotLiveList = dom.querySelectorAll('li');
```

示例：获取 span 标签

```
<html>
<head>
  <meta charset="utf-8"/>
  <link href="/css/style.css"/>
</head>
<body>
  <p> hello,<span>mooc</span>
      <span>users</span>
      
  </p>
  <div>
  </div>
</body>
</html>
```

```
var list = p.querySelectorAll('p span');
var type = list[0], user = list[1];
```



getElementsByName ---兼容 IE6 方案:

```
function getElementsByName(elm,clazz){
  if(elm.getElementsByClassName){
    return elm.getElementsByClassName(clazz);
  } else {
    var list = elm.getElementsByTagName('*'),result=[];
    for(var i=0,l=list.length;i<l;i++){
      if((' '+list[i].className+' ').indexOf(' '+clazz+' ')!==-1){
        result.push(list[i]);
      }
    }
    return result;
  }
}
```

```
elm.className = 'm-top';
getElementsByName(elm, 'top');
```

## 创建节点

- createElement
- innerHTML

### createElement

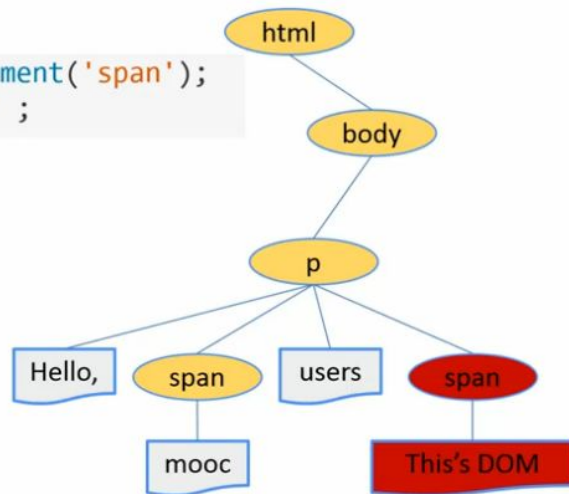


```
var element = document.createElement(tag)
```

参数名称	类型	是否必选	描述
tag	String	是	标签名
element	Node		

```
var element = document.createElement('ul');  
var element = document.createElement('li');  
var element = document.createElement('name');  
var element = document.createElement('span');  
var element = document.createElement('image');
```

```
var element = document.createElement('span');  
element.textContent='This's DOM' ;
```



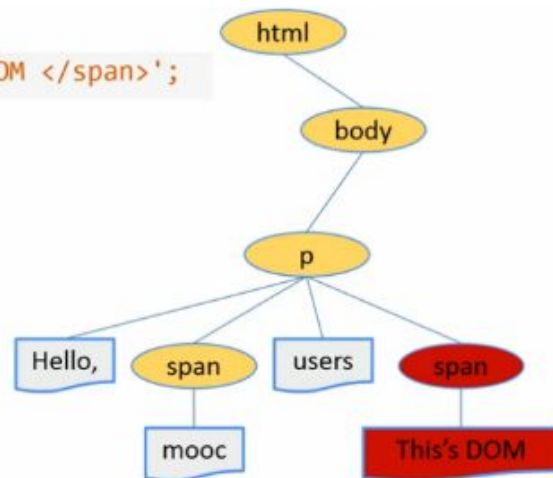
innerHTML

`element.innerHTML = HTMLString`

参数名称	类型	是否必选	描述
HTMLString	String		
element			同步解析HTMLString为element dom树

```
elm.innerHTML = '<ul>\n    <li class="item">1</li>\n    <li class="item">2</li>\n</ul>';
```

```
p.innerHTML += '<span> this\'s DOM </span>';
```



## 创建文本节点 createTextNode

`createTextNode()` 方法创建新的文本节点，返回新创建的 `Text` 节点。

**语法：**

```
document.createTextNode(data)
```

`createElement` & `innerHTML` 应用：

- 节点个数
- 事件处理
- 结合使用

以下代码是 `createElement` & `innerHTML` 两者结合使用的例子：

```
var tpl = '<div class="item">\n    <img class=" j-flag"/>\n    <div class="j-flag"></div>\n</div>';\nvar box = document.createElement('div');\nbox.innerHTML = tpl;\nvar list = box.getElementsByClassName('j-flag');\nlist[0].src = '/imgurl.jpg';\nlist[1].innerText = 'my name is netease';
```

## 插入节点

- appendChild
- insertBefore
- insertAdjacentElement
- insertAdjacentHTML

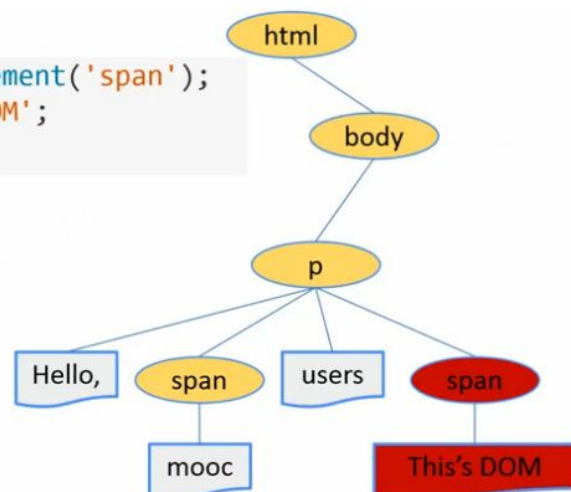
### appendChild

- `var child = parent.appendChild(child);`

参数名称	类型	是否必选	描述
child	Node	是	将节点插入到父节点最后子节点之后
parent	Node		父节点

```
<p id="p">\nhello,\n<span id="type">mooc</span>\nusers\n<!-- appendChild-->\n</p>
```

```
var element = document.createElement('span');\nelement.textContent = 'This's DOM';\np.appendChild(element);
```



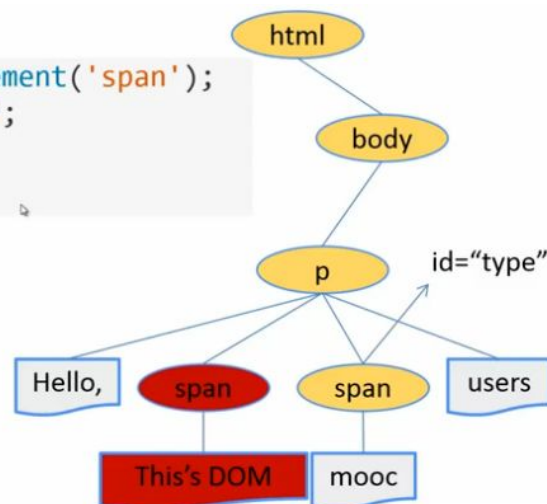
## insertBefore

```
var newElm = parent.insertBefore(newElm, rElm);
```

参数名称	类型	是否必选	描述
newElm	Node	是	将插入的节点
rElm	Node	是	newElm将插入的之前节点
parent	Node	是	父节点

```
<p id="p">
hello,
<!-- insertBefore-->
<span id="type">mooc</span>
users
</p>
```

```
var element = document.createElement('span');
element.innerHTML = 'This's DOM';
var span = $('type');
p.insertBefore(element, span);
```



## insertAdjacentElement

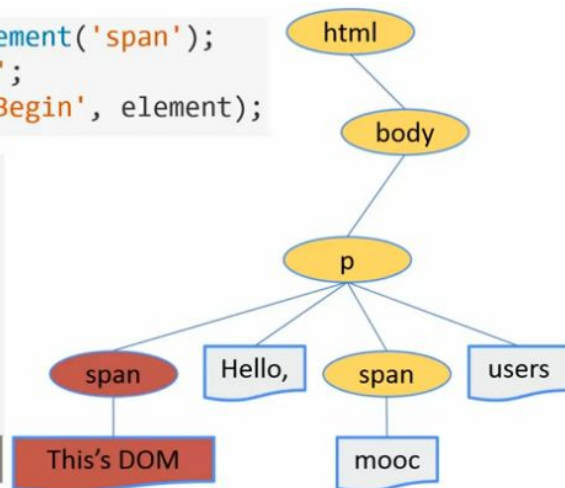
```
var oElm = elm.insertAdjacentElement(sWhere, oElm);
```

参数名称	类型	是否必选	描述
sWhere	String	是	位置描述 (beforeBegin,afterBegin,beforeEnd,afterEnd)
oElm	Node	是	插入的节点
elm	Node	是	参照节点

```
var element = document.createElement('span');
element.innerHTML = 'This's DOM';
p. insertAdjacentElement('afterBegin', element);
```

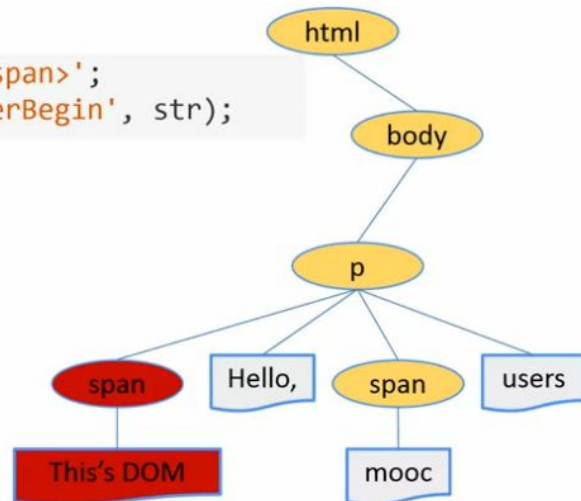
```
<!-- beforeBegin -->
<p id="p">
<!-- afterBegin -->
hello,
<span id="type">mooc</span>
users
<!-- beforeEnd -->
</p>
<!-- afterEnd -->
```

HTML



## insertAdjacentHTML

```
var str = '<span>This's DOM</span>';
p. insertAdjacentElement('afterBegin', str);
```



应用-insertAdjacentElement-兼容性方法:

```
if(typeof HTMLElement!="undefined" && !HTMLElement.prototype.insertAdjacentElement) {
    HTMLElement.prototype.insertAdjacentElement = function(whence,parsedNode) {
        switch (whence) {
            case 'beforeBegin':
                this.parentNode.insertBefore(parsedNode,this); break;
            case 'afterBegin':
                this.insertBefore(parsedNode,this.firstChild); break;
            case 'beforeEnd':
                this.appendChild(parsedNode); break;
            case 'afterEnd':
                if (this.nextSibling)
                    this.parentNode.insertBefore(parsedNode,this.nextSibling);
                else
                    this.parentNode.appendChild(parsedNode);
                break;
        }
    }
}
```

JS

## 修改节点

- innerHTML



- `textContent(innerText)`

## innerHTML

万能的 innerHTML

```
elm.innerHTML = '';  
elm.innerHTML = '';  
elm.innerHTML = '<ul><li>1</li><li>2</li></ul>';  
elm.innerHTML += '<a href="http://www.163.com">网易</a>';
```


### textContent(innerText):

`textContent` 是 W3C 定义的标准，而 `innerText` 不是一个标准，但是一个流行使用的属性。

FF 兼容 `innerText` 解决方法:

```
HTMLElement.prototype.__defineGetter__("innerText", function () {  
    return this.textContent;  
})  
HTMLElement.prototype.__defineSetter__("innerText", function(s) {  
    this.textContent = s;  
})
```

### innerHTML VS textContent(innerText)

`elm1.innerHTML = '';`  

`elm1.textContent = '';`  ``

`elm1.innerHTML = '<script src="http://x.com/inject.js">';`

在 JavaScript 中,我们可以使用 `innerHTML` 和 `innerText` 这 2 个属性很方便地获取和设置某一个元素内部子元素或文本。

`innerHTML` 属性被多数浏览器所支持,而 `innerText` 只能被 IE、chrome 等支持而不被 Firefox 支持。

`innerHTML` 属性声明了元素含有的 HTML 文本,不包括元素本身的开始标记和结束标记。设置该属性可以用于为指定的 HTML 文本替换元素的内容。



`innerText` 属性与 `innerHTML` 属性的功能类似，只是该属性只能声明元素包含的文本内容。即使指定的是 HTML 文本，它也会认为是普通文本而原样输出。

## 删除节点

- `removeChild`
- `replaceChild`
- `innerHTML`

### `removeChild`

```
var child = parent.removeChild(child);
```

参数名称	类型	是否必选	描述
child	Node	是	需要删除的节点
parent	Node	是	child的父节点
child	Node	是	被删除的节点

```
function remove(elm){  
    elm.parentNode.removeChild(elm);  
}
```

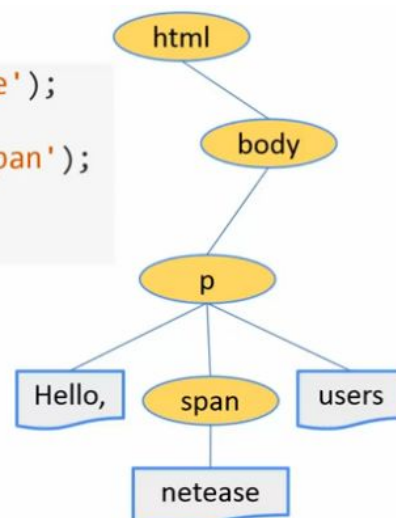
```
remove($('#type'))
```

### `replaceChild`

```
var oElm = parent.replaceChild(newChild,oElm);
```

参数名称	类型	是否必选	描述
newChild	Node	是	需要插入的节点
oElm	Node	是	需要删除的节点
parent	Node	是	oElm的父节点
oElm	Node		删除的节点

```
var type = document.getElementById('type');  
var p = document.getElementById('p');  
var netease = document.createElement('span');  
netease.innerText = 'netease';  
p.replaceChild(netease,type);
```



应用：

replaceChild == removeChild&appendChild

注意点：

- remove event
- elm.innerHTML=' '

### 三、属性操作

HTML attribute → DOM property

```
<div>
  <label for="userName">用户名 :</label>
  <input id="userName" type="text" class="u-txt">
</div>
```

HTML

input.

id	"userName"
type	"text"
className	"u-txt"

label.

htmlFor	"userName"
---------	------------

每个html属性对应相应的DOM对象属性

- property accessor
- getAttribute/setAttribute
- dataset

**property accessor:**属性访问器

属性读操作

```
<div>
  <label for="userName">用户名 :</label>
  <input id="userName" type="text" class="u-txt">
</div>
```

```
input.className;    // "u-txt"
input["id"];        // "userName"
```

属性写操作

```
<div>
  <label for="userName">用户名:</label>
  <input id="userName" class="u-txt" value="wwq@163.com">
</div>
```

用户名:

```
input.value = 'wwq@163.com';
input.disabled = true;
```

属性类型:

```
<input class="u-txt"
        maxlength="10"
        disabled
        onclick="showSuggest();">
```

## input.

className	"u-txt"	String
maxLength	10	Number
disabled	true	Boolean
onclick	function onclick(event){...}	Function

转换过的实用对象

通过属性访问符访问的属性，是转换过的实用对象。

属性访问器特点:

- 通用性 ×-名字异常
- 扩展性 ×
- 实用对象 ✓

**getAttribute/setAttribute** 进行属性操作

读

```
var attribute = element.getAttribute(attributeName);
```

```
<div>
    <label for="userName">用户名 : </label>
    <input id="userName" type="text" class="u-txt">
</div>
```

```
input.getAttribute("class");    // "u-txt"
```

写

```
element.setAttribute(name, value);
```

```
<div>
    <label for="userName">用户名 : </label>
    <input id="userName" class="u-txt" value="wwq@163.com">
</div>
```

用户名:

```
input.setAttribute("value", "wwq@163.com");
input.setAttribute("disabled", "");
```

类型

```
<input class="u-txt"
        maxlength="10"
        disabled
        onclick="showSuggest();">
```

## input.getAttribute("

class	"u-txt"	String
maxlength	"10"	String
disabled	""	String
onclick	"showSuggest();"	String

属性字符串

**getAttribute/setAttribute** 进行属性操作特点

- 仅字符串 ×
- 通用性 ✓

**dataset**

- HTMLElement.dataset
- data-\*属性集
- 元素上保存数据

```
<div id="user" data-id="123456" data-account-name="wwq"
      data-name="魏文庆" data-email="wwq123@163.com"
      data-mobile="13524543878">wwq</div>
```

## div.dataset.

id	"123456"
accountName	"wwq"
name	"魏文庆"
email	"wwq123@163.com"
mobile	"13524543878"

在 HTML5 中我们可以使用 `data-` 前缀设置我们需要的自定义属性，来进行一些数据的存放。DOM 对象中有个 `dataset` 属性对象，可以在该对象中存一些与该 DOM 对象相关的数据。如上写法是在生成 dom 对象时对 `dataset` 的一个初始化的动作，`dataset` 中会存在一个 `id` 属性、`accountName/name/email/mobile` 属性。这里需要注意 `data-` 仅仅是一个约定好的前缀，在生成 DOM 对象时所有已 `data-` 开头的属性会将其去掉前缀 `data-` 后存在 `dataset` 中。需要注意的是 `dataset` 中的属性值只能是字符串，非字符串会转换为字符串后存储，所以要存对象类型是务必小心。

#### 案例：

当鼠标移入时，显示出卡片信息，如下图：



实现代码：

#### HTML 部分：

```
<ul>
  <li data-id="123456" data-account-name="wwq"
      data-name="魏文庆" data-email="wwq123@163.com"
      data-mobile="13524543878">wwq</li>
  <li data-id="123457" data-account-name="cjf"
      data-name="蔡剑飞" data-email="cjf123@163.com"
      data-mobile="13968789868">cjf</li>
</ul>
<div id="card" style="display:none">
  <table>
    <caption id="accountName"></caption>
    <tr><th>姓名: </th><td id="name"></td></tr>
    <tr><th>邮箱: </th><td id="email"></td></tr>
    <tr><th>手机: </th><td id="mobile"></td></tr>
  </table>
</div>
```

#### CSS 部分：

```
<style>
```



本笔记由西风潇潇编写，欢迎浏览博客访问更多内容：<http://www.xifengxx.com>

```
li{cursor: default;line-height: 1.8;}
table{border-collapse: collapse;}
th, td, caption{padding: 10px;border: 1px solid #ddd;font-size: 14px;}
th{color: #999;}
caption{font-size: 20px;font-weight: bold;border-bottom: none;}
#card{position: absolute;top: 10px;left: 150px;}
</style>
```

JS 部分:

```
<script>
function $(id) {
    return document.getElementById(id);
}

var lis = document.getElementsByTagName('li');
for(var i = 0, li; li = lis[i]; i++) {
    li.onmouseenter = function(event) {
        event = event || window.event;
        var user = event.target || event.srcElement;
        var data = user.dataset;
        $('accountName').innerText = data.accountName;
        $('name').innerText = data.name;
        $('email').innerText = data.email;
        $('mobile').innerText = data.mobile;
        $('card').style.display = 'block';
    };
    li.onmouseleave = function(event) {
        $('card').style.display = 'none';
    };
}
</script>
```

## 如何实现浏览器兼容版的 element.dataset

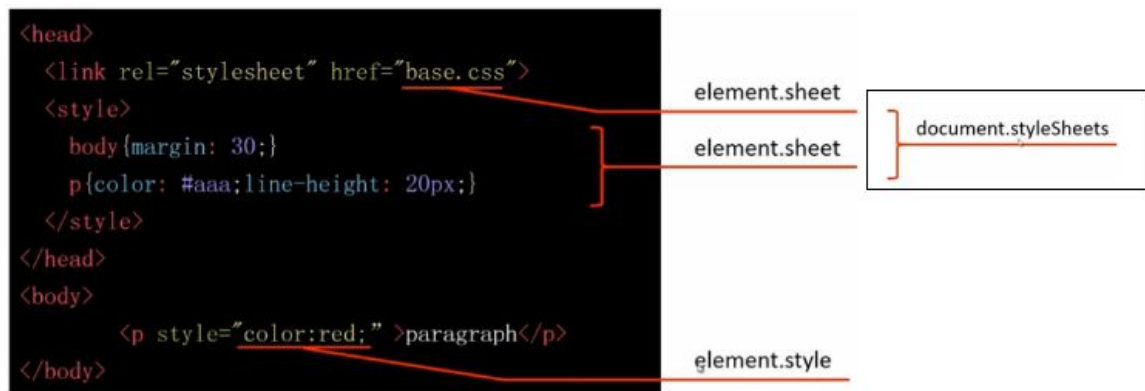
element.dataset 能够获取元素的自定义属性，但是低版本的 ie 不支持（ie11 支持，ie10 及以下不支持），如何在低版本的 ie 上兼容类似的功能。

**（待补充）**

参考链接：<http://www.gbtags.com/gb/share/5588.htm>

## 四、样式操作

### CSS→DOM



## 内部样式表



`element.sheet.cssRules[1].style.lineHeight`  
`.selectorText`

## 内嵌样式表:



## 更新样式

`element.style`

```
element.style.borderColor = 'red';  
element.style.color = 'red';
```

13564782365

▶ `<input style="border-color: red; color: red;">...</input>`

- 更新一个属性需要一条语句
- 不是我们熟悉的 CSS

### element.style.cssText

```
element.style.cssText = 'border-color:red;color:red;';
```

13564782365

▶ `<input style="border-color: red; color: red;">...</input>`

上面两张方式的缺点：

- 样式混在逻辑中

更新 class

```
.invalid{  
  border-color: red;  
  color: red;  
}
```

CSS

```
element.className += ' invalid' ;
```

13564782365

▶ `<input class="invalid">...</input>`

上述方法特点：一次更新很多元素的样式

### 更换样式表

案例：网页换肤

Script 部分代码：

```
<script src="../util.js"></script>
```

```
<script>
```

```
Util.addListener($('create0'), 'click', createStyleSheet);
```

```
Util.addListener($('create1'), 'click', createStyleSheet1);

function createStyleSheet0(){
    var link = document.createElement('link');
    link.rel = 'stylesheet';
    link.href = 'skin.summer.css';
    document.getElementsByTagName('head')[0].appendChild(link);
}

function createStyleSheet1(){
    var style = document.createElement('style');
    style.innerText = 'body{background-color: #fefaf7;}
        .m-tw .u-img{border-color: #a84c5b;}
        .m-tw p{color: #6d3e48;}
        .m-tw h3{background-color: #a84c5b;}
        .m-tw h3 a, .m-tw h3 a:hover{color: #fff;}';
    document.getElementsByTagName('head')[0].appendChild(style);
}

</script>
```

## 获取样式

### element.style

13564782365

```
<input type="text" style="color: red">
```

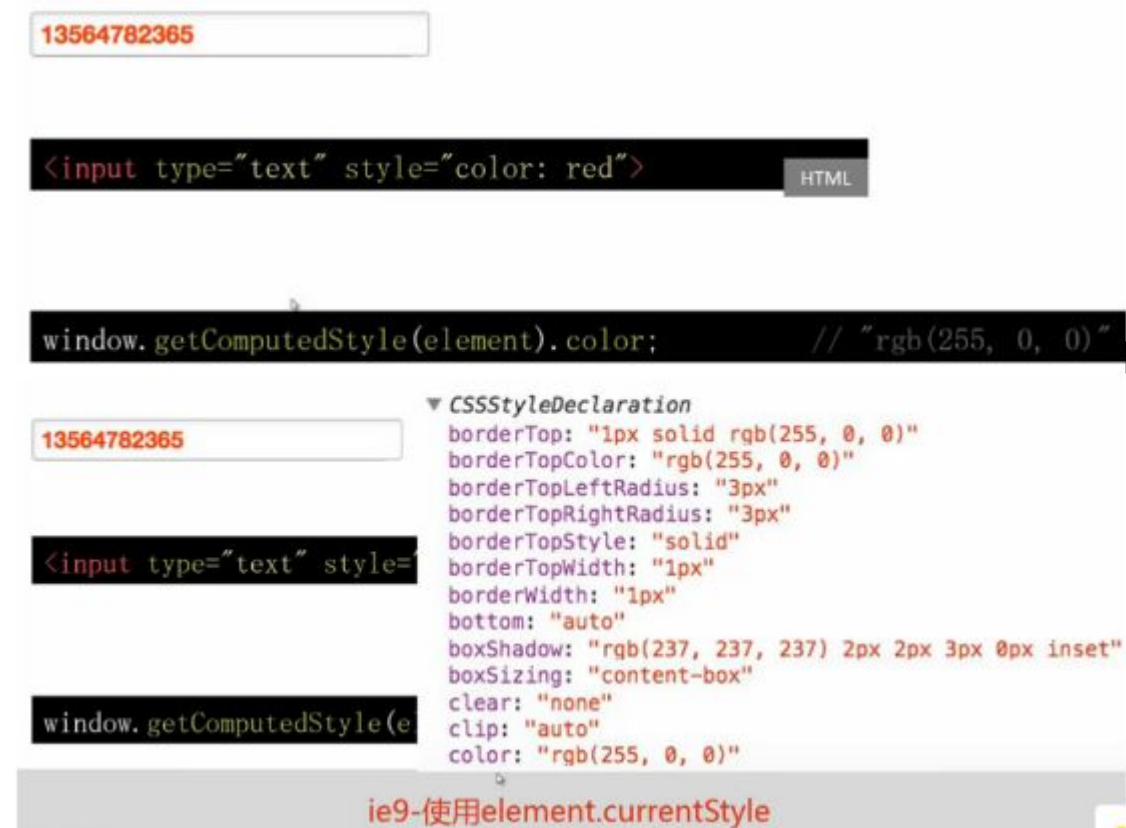
HTML

```
element.style.color; // "red"
```

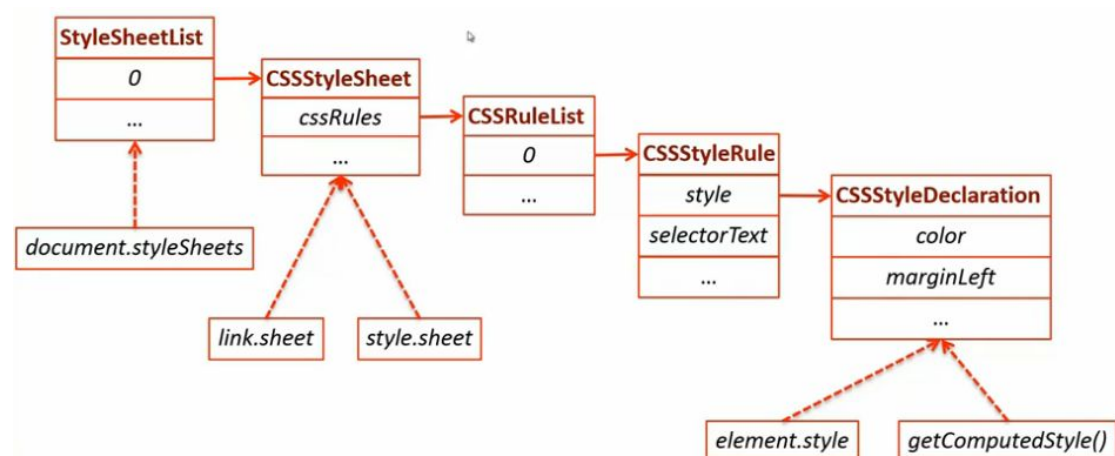
style获取到的不一定是实际样式

**window.getComputedStyle()---获取元素实际样式**

```
var style = window.getComputedStyle(element[,pseudoElt]);
```



## CSS DOM overview



## 如何实现浏览器兼容版的 window.getComputedStyle

window.getComputedStyle 能够获取元素的实际样式，但是低版本的 ie8 及以下不支持，如何在低版本的 ie 上兼容类似的功能。

代码实现：

```
function getStyle(elm){
    if(window.getComputedStyle){
        return window.getComputedStyle(elm);
    }
}
```

```
}else{  
  
    return elm.currentStyle; // ie8 及以下不支持 getComputedStyle 时，shiyongcurrentStyle。  
  
}  
}
```

## 五、事件

### 什么是 DOM 事件？

Javascript 与 HTML 之间的交互是通过事件来实现的。事件，就是文档或浏览器窗口中发生的一些特定的交互瞬间。比如：

- 点击一个 DOM 元素
- 键盘按下一个键
- 输入框输入内容
- 页面加载完成

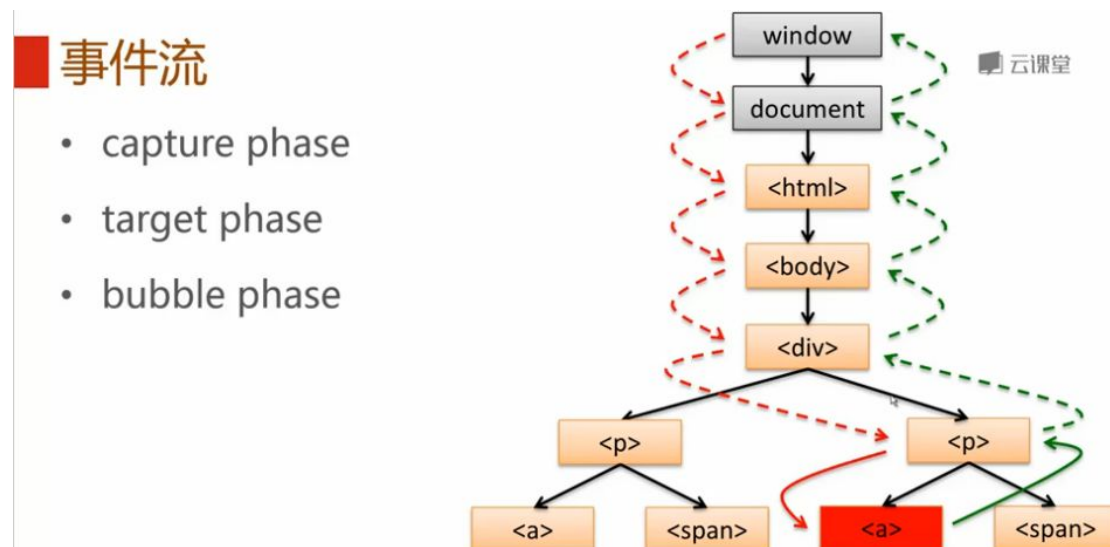
#### 事件流

事件流描述的是从页面中接收事件的顺序。

IE 的事件流是事件冒泡流（event bubbling），即事件开始时由最具体的元素（文档中嵌套层次最深的那个节点）接收，然后逐级向上传播到较为不具体的节点（文档）。

Netscape 的事件流是事件捕获流(event capturing)。事件捕获的思想是不太具体的节点应该更早接收到事件，而最具体的节点应该最后接收到事件。

DOM2 级事件规定的事件流包括三个阶段：事件捕获阶段、处于目标阶段和事件冒泡阶段。





## 事件注册与触发

### 事件注册

- `eventTarget.addEventListener(type,listener[,useCapture])`

```
var elem = document.getElementById('div1');
var clickHandler = function(event){
    //TO DO
}
elem.addEventListener('click', clickHandler, false);
elem.onclick = clickHandler;
```

### 取消事件注册

- `eventTarget.removeEventListener(type,listener[,useCapture])`

```
elem.removeEventListener('click', clickHandler, false);
```

```
elem.onclick = null;
```

### 事件触发

- `eventTarget.dispatchEvent(type)`

```
elem.dispatchEvent('click');
```

### 浏览器兼容型（IE6/7/8）

- 事件注册与取消：`attachEvent()` / `detachEvent()`

这两个方法接受相同的 2 个参数：事件处理程序名称、事件处理程序函数

- 事件触发：`fireEvent(e)`
- no capture: IE8 及更早版本只支持事件冒泡，不支持事件捕获。

### 浏览器兼容型

```
var addEvent = document.addEventListener ?
    function(elem, type, listener, useCapture) {
        elem.addEventListener(type, listener, useCapture);
    } :
    function(elem, type, listener, useCapture) {
        elem.attachEvent('on' + type, listener);
    };
```

```
var delEvent = document.removeEventListener ?
    function(elem, type, listener, useCapture) {
        elem.removeEventListener(type, listener, useCapture);
    } :
    function(elem, type, listener, useCapture) {
        elem.detachEvent('on' + type, listener);
    };
```

## 事件对象

在触发 DOM 上的某个事件时，会产生一个事件对象 `event`，这个对象中包含着所有与事件有关的信息。

```
var elem = document.getElementById('div1');
var clickHandler = function(event){
    //TO DO
}
elem.addEventListener('click', clickHandler, false);
```

```
var elem = document.getElementById('div1');
var clickHandler = function(event) {
    event = event || window.event;
    //TO DO
}
addEvent(elem, 'click', clickHandler, false);
```

事件对象属性:

- type
- target(srcElement)
- currentTarget

事件对象方法:

- stopPropagation
- preventDefault
- stopImmediatePropagation()

## 阻止事件传播

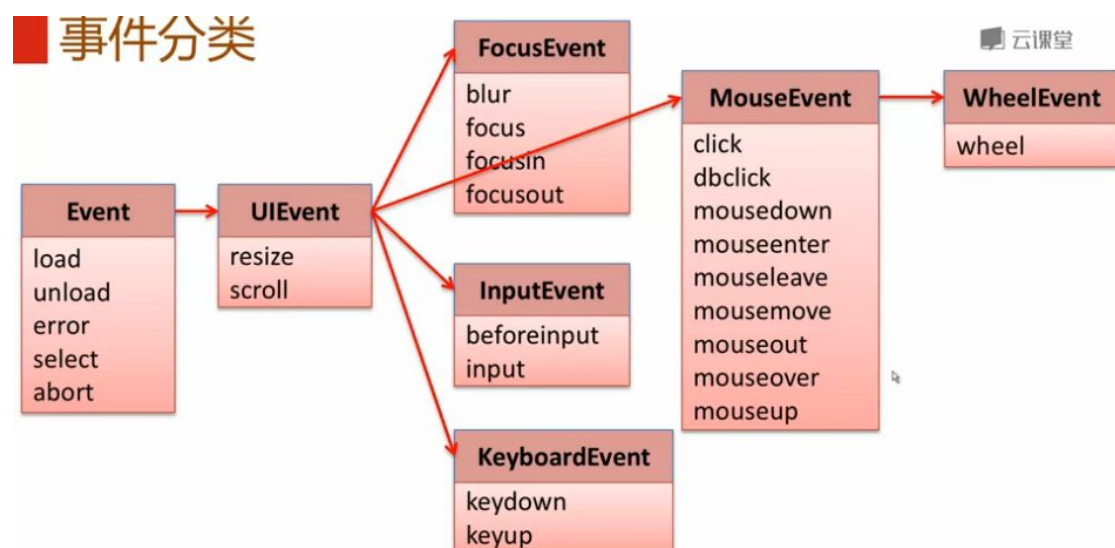
- `event.stopPropagation()` (W3C)
- `event.cancelBubble=true` (IE)
- `event.stopImmediatePropagation()` (W3C)

## 默认行为

`Event.preventDefault()` (W3C)

`Event.returnValue=false` (IE)

## 事件分类



## Event

事件类型	是否冒泡	元素	默认事件	元素例子
load	NO	Window、Document、Element	None	window、image、iframe
unload	NO	Window、Document、Element	None	window
error	NO	Window、Element	None	window、image
select	NO	Element	None	input、textarea
abort	NO	Window、Element	None	window、image

## Window 对象

Window 对象上的 Event 事件包括：

- load
- unload
- error

- abort

## Image 对象

Image 对象上的 Event 事件包括

- load
- error
- abort

```
<image alt="photo" src="http://www.163.com/photo.jpg"
onerror="this.src='http://www.163.com/default.jpg'"/>
```

## UIEvent

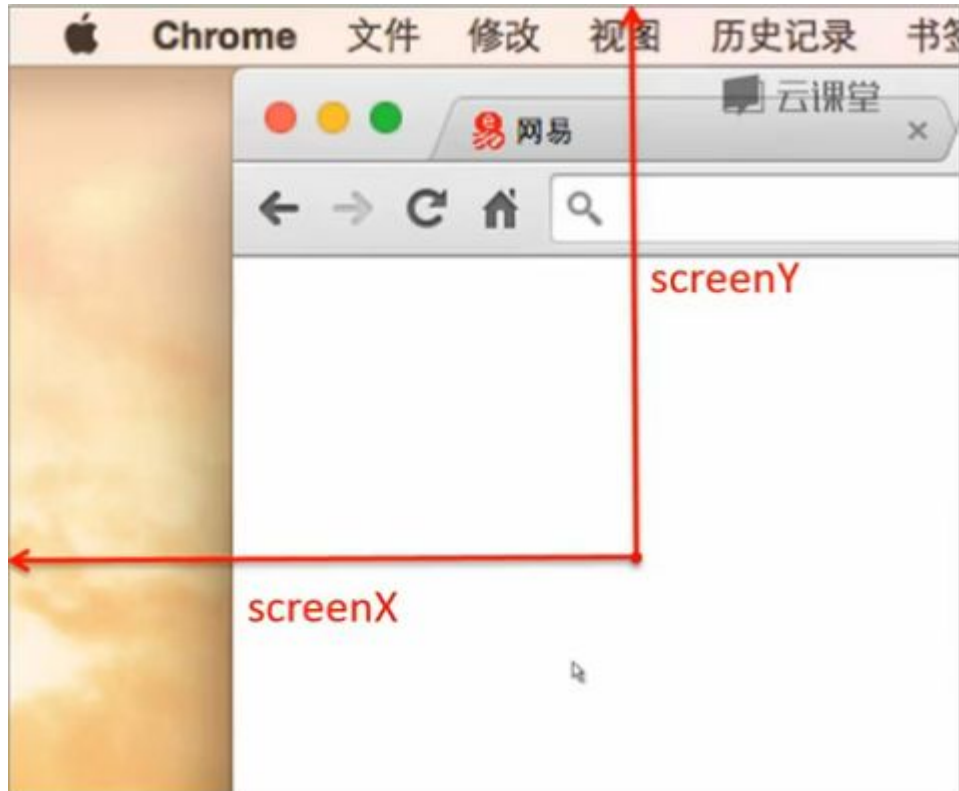
事件类型	是否冒泡	元素	默认事件	元素例子
resize	NO	Window、Element	None	window、iframe
scroll	NO/YES	Document、Element	None	document、div

## MouseEvent

事件类型	是否冒泡	元素	默认事件	元素例子
click	YES	Element	focus/activation	div
dblclick	YES	Element	focus/activation select	div
mousedown	YES	Element	drag/scroll text selection	div
mousemove	YES	Element	None	div
mouseout	YES	Element	None	div
mouseover	YES	Element	None	div
mouseup	YES	Element	context menu	div
mouseenter	NO	Element	None	div
mouseleave	NO	Element	None	div

## MouseEvent 对象属性

- clientX,clientY
- screenX,screenY
- ctrlKey,shiftKey,altKey,metaKey
- button(0,1,2)



### MouseEvent 顺序

从元素 A 上方移过:

-mousemove -> mouseover(A) -> mouseenter(A) -> mousemove(A) -> mouseout(A) -> mouseleave(A)

点击元素

-mousedown -> [mousemove] -> mouseup -> click

例子-拖拽 div

HTML 部分:

```
<div id="div1"></div>
```

CSS 部分:

```
<style type="text/css">
```

```
#div1{  
    position:absolute;top:0;left:0;  
    border:1px solid #000;  
    width:100px;height:100px;  
}
```

```
</style>
```

JS 部分代码:



```
var elem = document.getElementById('div1');
var clientX, clientY, moving;
var mouseDownHandler = function(event) {
    event = event || window.event;
    clientX = event.clientX;
    clientY = event.clientY;
    moving = !0;
}
var mouseMoveHandler = function(event) {
    if (!moving) return;
    event = event || window.event;
    var newClientX = event.clientX,
        newClientY = event.clientY;
    var left = parseInt(elem.style.left) || 0,
        top = parseInt(elem.style.top) || 0;
    elem.style.left = left + (newClientX - clientX) + 'px';
    elem.style.top = top + (newClientY - clientY) + 'px';
    clientX = newClientX;
    clientY = newClientY;
}
var mouseUpHandler = function(event) {
    moving = !1;
}
addEvent(elem, 'mousedown', mouseDownHandler);
addEvent(elem, 'mousemove', mouseMoveHandler);
addEvent(elem, 'mouseup', mouseUpHandler);
```

JavaScript程序设计

## WheelEvent-滚轮事件

事件类型	是否冒泡	元素	默认事件	元素例子
wheel	YES	Element	scroll or zoom document	div

### WheelEvent 对象属性

- deltaMode
- deltaX
- deltaY
- deltaZ

## FocusEvent-焦点事件



事件类型	是否冒泡	元素	默认事件	元素例子
blur	NO	Window、Element	None	window、input
focus	NO	Window、Element	None	window、input
focusin	NO	Window、Element	None	window、input
focusout	NO	Window、Element	None	window、input

### FocusEvent 属性

- relatedTarget

### InputEvent- 输入事件

事件类型	是否冒泡	元素	默认事件	元素例子
beforeinput	YES	Element	update DOM Element	input
input	YES	Element	None	input

onpropertychange(IE)

### KeyboardEvent-键盘事件

事件类型	是否冒泡	元素	默认事件	元素例子
keydown	YES	Element	beforeinput/input focus/blur activation	div,input
keyup	YES	Element	None	div,input

### KeyboardEvent 对象属性

- key
- code
- ctrlKey/shiftKey/altKey/metaKey
- repeat
- keyCode
- charCode
- which

## 事件代理

- 将事件注册到元素的父节点上。

事件代理也被称为“事件委托”。事件委托利用了事件冒泡，只指定一个事件处理程序，就可以管理某一类型的所有事件。

（详见《JavaScript 高级程序设计(第 3 版)》，第 13 章，P402 页）