DOM(文档对象模型) 是从浏览器的视角来观察页面中的元素以及每个元素的属性，由此得出这些元素的一个家族树。通过DOM，可以确定元素之间的相互关系。在CSS 中引用DOM 中特定的位置，就可以选中相应的HTML 元素，并修改其样式属性。

通过下面这个例子来理解DOM

// HTML 代码正确缩进，以表明标签的层次关系

<body>

<section>

<h1>The Document Object Model</h1>

<p>The page's HTML markup structure defines the DOM.</p>

</section>

</body>

与标记对应的简单DOM 结构图



直观的表现出了元素之间的关系，父元素、子元素、兄弟元素一目了然。

**DOM模型结构**

最常见的节点类型：

**元素**：元素是XML，HTML的基本构建。

元素可以有其他元素、文本节点或两者兼有来作为其子节点。

元素节点还是可以有属性的唯一类型的节点。

属性：属性节点包含关于元素节点的信息，但实际上，不认为它是元素的子节点。**文本**：确切来讲，文本节点是文本。它可以包含许多信息或仅仅是空白。

**文档（根节点）** ：文档节点是整个文档中所有其他节点的父节点。（根节点不等于根元素节点）

较不常见的节点类型：CDATA、注释、处理指令。

**DOM 节点**：

根据 W3C 的 HTML DOM 标准，HTML 文档中的所有内容都是节点：

* + - 整个文档是一个文档节点
    - 每个 HTML 元素是元素节点
    - HTML 元素内的文本是文本节点
    - 每个 HTML 属性是属性节点
    - 注释是注释节点

DOM的四个基本接口在DOM接口规范中，有四个基本的接口，Document，Node，NodeList，NamedNodeMap。

**Document**

Document接口是对文档进行操作的入口，它是从Node接口继承过来的。

**Node**

Node接口是其他大多数接口的父类。

在DOM树中，Node接口代表了树中的一个节点。

**NodeList**

NodeList接口是一个节点的集合，它包含了某个节点中的所有子节点。

它提供了对节点集合的抽象定义，并不包含如何实现这个节点集的定义。　　NodeList用于表示有顺序关系的一组节点，比如某个节点的子节点序列。

在DOM中，NodeList的对象是live的，对文档的改变，会直接反映到相关的NodeList对象中。

**NamedNodeMap**

NamedNodeMap接口也是一个节点的集合，通过该接口，可以建立节点名和节点之间的一一映射关系，从而利用节点名可以直接访问特定的节点，这个接口主要用在属性节点的表示上。

尽管NamedNodeMap所包含的节点可以通过索引来进行访问，但是这只是提供了一种枚举方法，NamedNodeMap所包含的节点集中节点是无序的。

与NodeList相同，在DOM中，NamedNodeMap对象也是live的。

通过Javascript可以对DOM节点做任意操作，是得js能够改变HTMl。



**DOM常见操作**



## DOM事件

DOM同时两种事件模型：冒泡型事件和捕获型事件

**冒泡型事件**：事件按照从最特定的事件目标到最不特定的事件目标的顺序触发

<body onclick="handleClick()">

<div onclick="handleClick()">Click Me</div>

</body>

触发的顺序是：div、body、html(IE 6.0和Mozilla 1.0)、document、window(Mozilla 1.0)

**捕获型事件**：与冒泡事件相反的过程，事件从最不精确的对象开始触发，然后到最精确

上面例子触发的顺序是：document、div

DOM事件模型最独特的性质是，文本节点也触发事件(在IE中不会)。

1）**事件处理函数/监听函数**

在JavaScript中：

var oDiv = document.getElementById("div1");

oDiv.onclick = function(){ //onclick只能用小写

alert("Clicked!");

}

或者

var elem =document.getElementById(“id”)

elem.onmouseover=handleMouseOver //handleMouseOver 是函数名

Function handleMouseOve(e){...}

在HTML中：

<div onclick="javascript: alert("Clicked!")"></div> //onclick大小写任意

扩展:

IE事件处理程序attachEvent()和detachEvent()

在IE中，每个元素和window对象都有两个方法：attachEvent()和detachEvent()，这两个方法接受两个相同的参数，事件处理程序名称和事件处理程序函数，如：

[object].attachEvent("name\_of\_event\_handler","function\_to\_attach")

[object].detachEvent("name\_of\_event\_handler","function\_to\_remove")

var fnClick = function(){

alert("Clicked!");

}

oDiv.attachEvent("onclick", fnClick); //添加事件处理函数

oDiv.attachEvent("onclick", fnClickAnother); // 可以添加多个事件处理函数

oDiv.detachEvent("onclick", fnClick); //移除事件处理函数

在使用attachEvent()方法的情况下，事件处理程序会在全局作用域中运行，因此this等于window。

2）**跨浏览器的事件处理程序**

addHandler()和removeHandler()

addHandler()方法属于一个叫EventUntil()的对象，这两个方法均接受三个相同的参数，要操作的元素，事件名称和事件处理程序函数。

3）**事件类型**

鼠标事件：click、dbclick、mousedown、mouseup、mouseover、mouseout、mousemove

键盘事件：keydown、keypress、keyup

HTML事件：load、unload、abort、error、select、change、submit、reset、resize、scroll、focus、blur

4）**事件处理器**

执行JavaScript 代码的程序在事件发生时会对事件做出响应。为了响应一个特定事件

而被执行的代码称为事件处理器。

在HTML标签中使用事件处理器的语法是：

<HTML标签 事件处理器="JavaScript代码''>

5）**事件处理程序**

事件就是用户或浏览器自身执行的某种动作。比如click,mouseup,keydown,mouseover等都是事件的名字。而响应某个事件的函数就叫事件处理程序（事件监听器），事件处理程序以on开头，因此click的事件处理程序就是onclick

6）**DOM 0级事件处理程序**

DOM 0级事件处理程序：把一个函数赋值给一个事件的处理程序属性

<input type="button" value="按钮2" id="ben2"/>

var btn2=document.getElementById('btn2');获得btn2按钮对象

btn2.onclick //给btn2添加onclick属性，属性又触发了一个事件处理程序

btn2.onclick=function(){

}

//添加匿名函数

btn2.onclick=null //删除onclick属性

7）**DOM 2级事件处理程序**

DOM 2级事件定义了两个方法，用于指定和删除事件处理程序的操作。addEventListener()和removeEventListener()

addEventListener()和removeEventListener()

在DOM中，addEventListener()和removeEventListener()用来分配和移除事件处理函数，与IE不同的是，这些方法需要三个参数：事件名称，要分配的函数和处理函数是用于冒泡阶段(false)还是捕获阶段(true)，默认为冒泡阶段false

[object].addEventListener("name\_of\_event",fnhander,bcapture)

[object].removeEventListener("name\_of\_event",fnhander,bcapture)

var fnClick = function(){

alert("Clicked!");

}

oDiv.addEventListener("onclick", fnClick, false); //添加事件处理函数

oDiv.addEventListener("onclick", fnClickAnother, false); // 与IE一样，可以添加多个事件处理函数

oDiv.removeEventListener("onclick", fnClick, false); //移除事件处理函数

如果使用addEventListener()将事件处理函数加入到捕获阶段，则必须在removeEventListener()中指明是捕获阶段，才能正确地将这个事件处理函数删除

oDiv.onclick = fnClick;

oDiv.onclick = fnClickAnother; //使用直接赋值，后续的事件处理函数会覆盖前面的处理函数

oDiv.onclick = fnClick;

oDiv.addEventListener("onclick", fnClickAnother, false); //会按顺序进行调用，不会覆盖