概述

关于 web 开发中在各种浏览器中有完全一样的表现是很多人的目标，然而这是一个永远无法真正实现的目标，但在很多情形，一种近似的兼容还是相对容易实现。本文将跨浏览器开发和测试测试有关的各种问题与知识。

浏览器家族

一般来说，跨浏览器是指针对不同的渲染引擎： Trident，Gecko，Webkit，Presto 等。

Trident (Internet Explorer), Gecko (Firefox), Webkit (Chrome and Safari) and Presto (Opera)

Html 标记

为页面添加 DOCTYPE

由于不同浏览器对标签,样式表的解释不尽相同,所以需要为 html 文件定义一个标准的文档类型,使不同浏览器尽量按照一个统一的 html 标准来解析渲染页面。

!DOCTYPE 声明指定文档遵从的 DTD，如：

<TML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

标准 HTML 标签的正确使用

尽量使用 div+css 布局,不用 table 做布局。

使用 table 做布局容易造成代码冗余，相对 编写来说，代码繁多。并且，table 需要将元素都下载后进行显示，相应的网页打开速度也较慢。

应该使用标准化的页面结构：DIV+CSS。这种布局方式代码简洁，页面浏览速度较快，页面布局灵活，改版时只需改 CSS 样式即可实现页面重新布局，而不用改动程序，从而降低了网站改版的成本。

注意标签的闭合关系,尤其是在 form 标签中嵌套 div 等其他标签时。

有的时候页面上会出现多余的空白，即使重新设置了 margin 也无法避免，这个时候有可能是页面元素标签闭合出现了不配对的情况，如：

```
<div class="outer">
    <form name="testForm">
        <div class="inner">
            <input name="title" type="text" />
        </form>
    </div>
</div>
```

定义 table 时使用 tbody 元素，以保证包括 IE 在内的所有浏览器可正确使用

即使 table 没有显示定义 tbody 元素,浏览器也会认为 tr 节点的父节点是一个自动添加的默认 tbody 节点.为了避免使用 javascript 操纵 tr 节点时可能产生的误会， 还是手动添加一个比较好，如：

```
<table id="myTable">
<tbody id="myTableBody">
  <tr>
    <td>
</td>
</tr>
</tbody>
</table>
```

注意标签及属性的大小写

有的时候，有些绑定在元素上的事件在 IE 浏览器下响应，在 safari 或其他浏览器下却不响应。这时候需要检查事件绑定方式的正确性，高级事件的绑定需要区别 IE 和其他浏览器写两套 javascript，而简单事件模型需要注意一下绑定事件名的大小写。如：

```
<input type="text" name="keywordSearch" onFocus="clearValue()" >
```

这里应该用小写的 onfocus，并且显示的添加标签闭合符号才是规范的写法。

```
<input type="text" name="keywordSearch" onfocus="clearValue()" />
```

注意标签的属性值设置

<script>标签的 language 和 type 属性

<script>标签的 language 属性是用来定义脚本语言版本的，正确的赋值应该形如<script>用来告诉浏览器（主要是 IE）使用 1.2 版本的 javascript 语法来解释；而 type 属性才是用来定义脚本类型的，是 w3c 的标准属性，并且使用小写属性才是符合标准的做法。如果不是特别情况下需要告诉浏览器按照较低版本的 javascrip 语言进行解释的话（目前大多数浏览器支持的 javascript 版本是 1.5），一般不需要定义 language 属性，但是 type 属性是需要定义的。所以应该把代码中的

<SCRIPT Language="JavaScript">改为<script>

<a>标签的 alt 和 title 属性

虽然 alt 和 title 这两个属性的值在 IE 下都会以 tool tip 的方式在鼠标悬停时显示，但是二者还是有区别的。alt 是图片没有显示出来的时候的替代显示，而 title 才是鼠标放到上面时的提示。

<input>标签的 checked、readonly 属性

在早期版本的 HTML 中，并没有强制规定所有的属性都应该赋值，在表示一个选中的复选框时，<input checked > 这样的写法是被认可的。但是根据 XHTML 的标准，这样的文法并不是一个严格的 XML 格式，应该注意对属性的赋值和标签的闭合，以顺应 HTML 标准发展的趋势，写成这样：

<input checked="checked" />

<input readonly="readonly" />

<option>标签的 select ed 属性

与上一条相同的理由，<select>选项中<option>标签的 selected 属性也应该赋值：

```
< option selected="selected" />
```

<img>标签的 align="absmiddle" 属性

根据 XHTML 的标准，HTML 标签应该专注于内容的表示，而不是样式的控制，样式应该交给 CSS 调整。所以废弃了一些旧的标签和属性，比如标签和<i>标签都会让标签内容中文字以斜体显示，但是<i>标签这种单纯以样式命名的标签已经属于废弃的标准了，取而代之的是表示 emphasis（强调）含义的标签。同理，<img>标签的和 align="absmiddle" 属性表示该图片和相邻文字垂直居中对齐，这也是表示样式的属性，应该使用 CSS 而不是这个属性来控制图片的对齐样式，避免两处样式控制的相互影响。

<iframe>标签的 frameborder 属性

在使用 iframe 时，IE 中或许只要设置 border="0"就可以不显示 iframe 的边框了，但是标准的控制 frame 窗口边框的属性是 frameborder，应该设置 frameborder="0"才能在 IE 之外的其他浏览器中同样隐藏 frame 的边框：

```
<iframe frameborder="0" />
```

<table>标签的 cellpadding 属性

这个属性同<img>标签的 align 属性一样，也是一个僭越了 HTML 自身表示内容的职责而控制样式的一个属性，它规定的是单元之间的空间。从实用角度出发，最好不要规定 cell padding，而是使用 CSS 来控制单元格的内边距。

<td>标签的 nowrap 属性

nowrap 是表示内容不自动换行的属性,但是同上面的属性一样,这是一个控制样式的属性。

在 HTML 4.01 中，<td>标签的 "bgcolor"、"height"、"width" 以及 "nowrap" 是不被赞成使用的。在 XHTML 1.0 Strict DTD 中，<td>标签的 "bgcolor"、"height"、"width" 以及 "nowrap" 是不被支持的。

CSS 层叠样式

CSS 盒子模型

如果你想实现不需要很多复杂的跨浏览器兼容的 CSS 代码,透彻地理解 CSS 盒子模型是首要事情，CSS 盒子模型并不难，且基本支持所有浏览器，除了某些特定条件下的 IE 浏览器。

CSS 盒子模型负责处理以下事情:

一个 blcok (区块)级对象占据多大的空间

该对象的边界，留白

盒子的尺寸

盒子与页面其它元素的相对位置

CSS 盒子模型有以下准则:

Block (区块)对象都是矩形 (事实上所有对象都如此)

其尺寸由 width, height, padding, borders, 以及 margins 决定

如果不设置高度，该盒子的高度将自动适应其包含的内容，加上留白等(除非使用了 float)

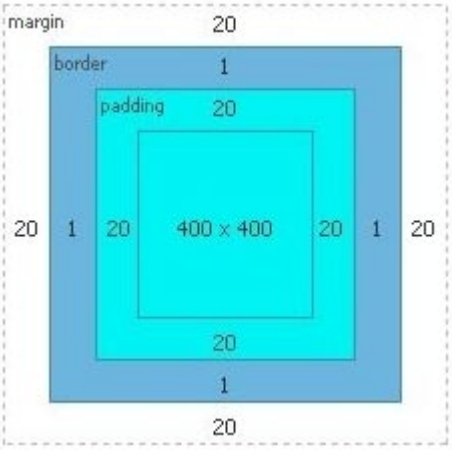
如果不设置宽度,一个非 float 型盒子水平上将充满其父容器(扣除父容器的留白)

处理 block 级对象时，必须注意以下事项:

如果一个盒子的宽度设置为 100%，它就不能再设置 margins, padding, 和 borders，否则会撑破其父容器。

垂直毗邻的 margin 会引起复杂的坍塌问题，导致布局问题(比如两个垂直毗邻的 Block 对象，上面的对象的 bottom-margin 为 40，下面的对象的 top-margin 为 20，则两个对象的间距将是 40，而不是 60。)

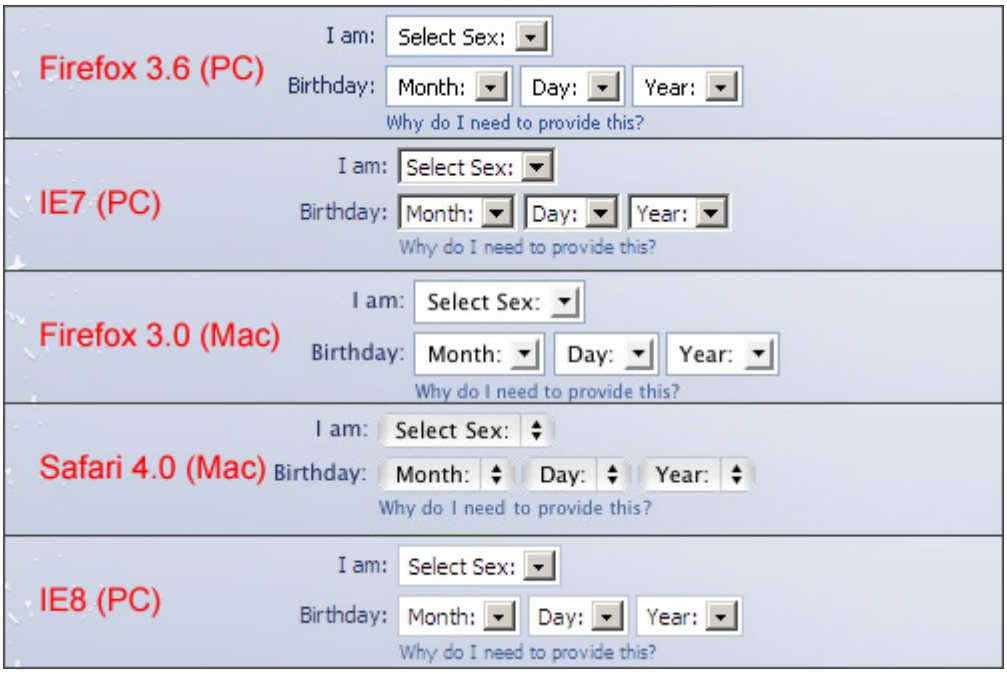
拥有相对位置和绝对位置的对象，拥有不同的行为



在 Firefox 的 Firebug 中显示的盒子模型

Form 控件在不同浏览器显示总是不同

以下是 Facebook 首页中的 select 控件，在 5 种不同浏览器的显示差异(基于 Adobe’s Browserlab 截图)

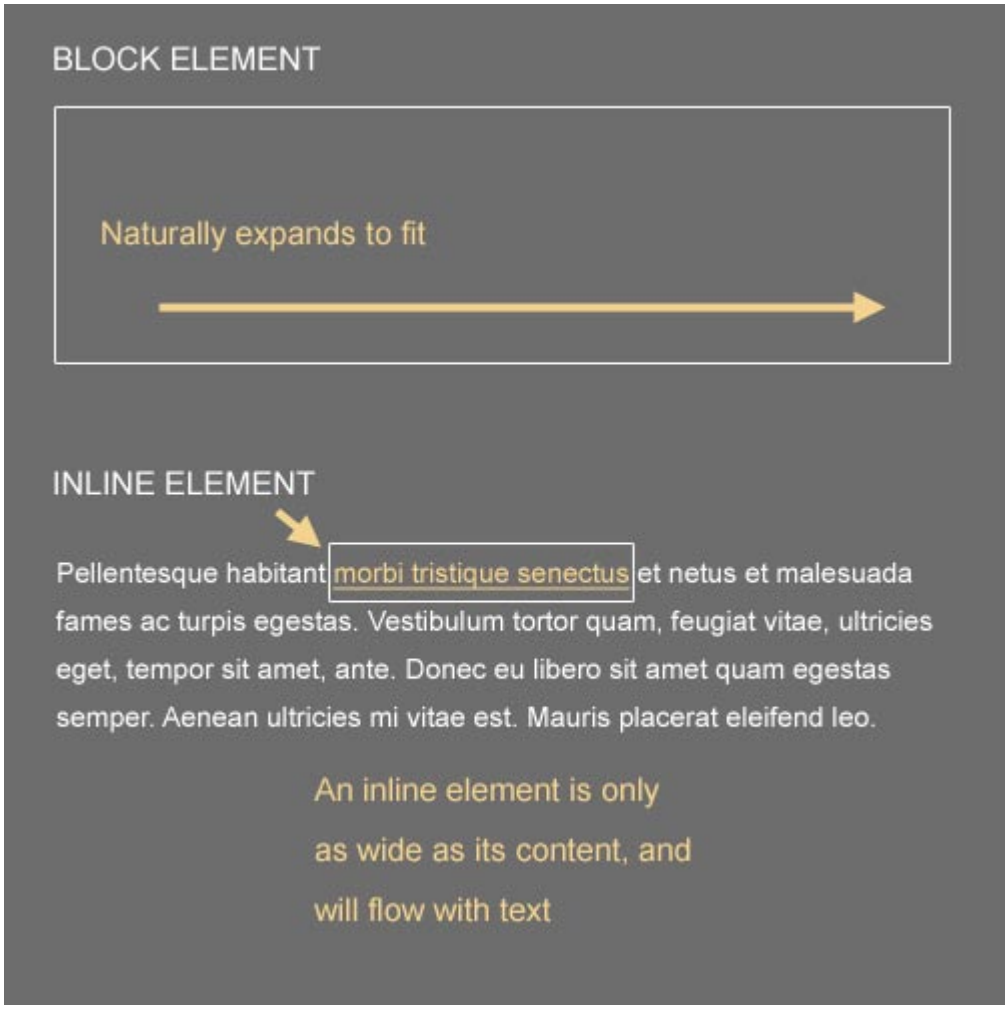


某些 Form 控件，如果要求必须跨浏览器一致，可以找到变通办法，如，可以使用图片 替代 submit 按钮,但有一些控件,比如 radio,select， textarea，文件选择框，是永远都不可能一模一样的。

理解 block 级和 inline 级对象的区别

这个看似简单的问题事如果能透彻地理解，会受益匪浅。

下图讲解了 block 级对象和 inline 级对象的区别:



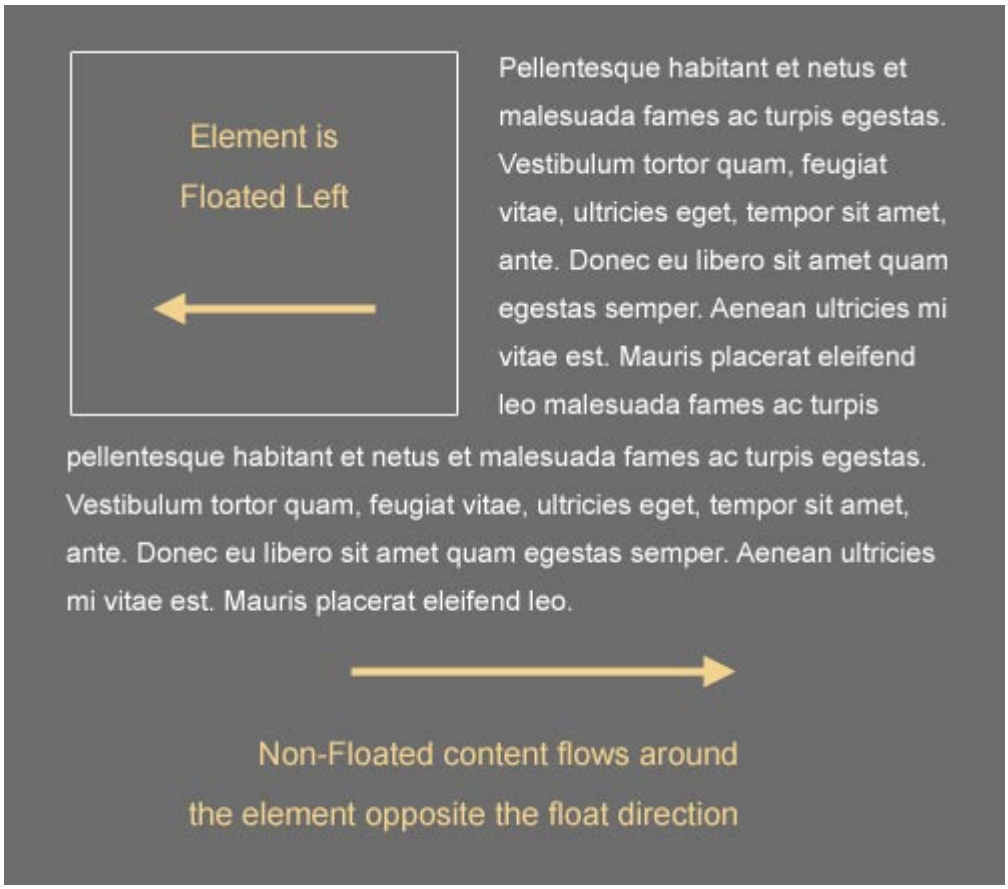
下面是 block 级对象和 inline 级对象的基本区别:

- Block 级对象会自然地水平充满其父容器，因此没有必要为之设置 100% 宽度属性
- Block 级对象的起始摆放位置是其父容器的左上边界，并顺排在其前面的兄弟 Block 对象的下方（除非设置 float 或绝对位置）
- Inline 级对象会忽略其宽度和高度设置
- Inline 级对象会随着文字排版，并受排版属性的影响（如 white-space, font-size, letter-spacing）
- Inline 级对象可以使用 vertical-align 属性控制其垂直对齐，block 级对象不可以
- Inline 级对象的下方会保留一些自然的空间，以适应字母 g 一类的会向下探出的笔画

- 一个设置为 float 的 inline 对象将变成 block 对象

理解 Floating 和 Clearing 属性

实现多栏排版的最好方法是使用 float 属性，float 也是一个将使你受益匪浅的属性。一个 float 对象可以居左或居右，一个设置为 float 的对象，将根据设置的方向，左移或右移到其父容器的边界，或其前面的 float 对象的边界，而紧随其后的非 float 对象或内容，则包围在其相反的方向。



以下是使用 float 和 clear 属性的一些重要准则:

- 一个 float 对象，将从其置身的 block 级非 float 内容流中跳出，换句话说，如果你要将一个 box 向左边 float，它后面的 block 级非 float 对象会显示到下方，inline 级内容会在旁边包围
- 要让一段内容从一侧包围一个 float 对象，这段内容必须要么是 inline 级的，要么也设置为相同方向的 float

- 一个 float 对象，如果没有设置宽度，则会自动缩成其包含的内容的宽度，因此最好为 float 对象明确设置宽度
- 如果一个 block 对象包含 float 子对象，会出现本文中阐述的问题。
- 一个设置了 clear 属性的对象，将不会包围其前面的 float 对象
- 一个既设置了 clear 又设置了 float 属性的对象，只有 clear:left 属性生效，clear:right 不起作用

css hack 的使用

CSS 类级别的 hack

仅 IE7 识别 *+html {…}

IE6 及 IE6 以下识别 * html {…}

opera、safari、chrome 识别：

@media all and (min-width: 0px){…} //Firefox3.0.6 不识别，但 Firefox3.6 也识别该规则，如果 Firefox 版本有严格要求，请使用下一条规则

@media screen and (-webkit-min-device-pixel-ratio:0){…} //IE、Firefox 不识别该规则

仅 opera 识别：

@media screen and (-webkit-min-device-pixel-ratio:10000), not all and (-webkit-min-device-pixel-ratio:0{…}

CSS 属性级别的 hack

仅 IE 识别 ： *margin-left:10px\9;*

仅 IE8 识别 ： *margin-left:10px\0;*

IE6/IE7 识别 ： **margin-left:10px;*

仅 IE6 识别 ： *_margin-left:10px;*

CSS Hack 综合示例：

```
/**add 'margin-top: -10px;' for IE7/Firefox/Opera/Safari/Chrome , 'margin-top: 5px;' fo
r IE8 **/

.news_list01 h2 span{float: right; margin-top: 5px; *margin-top: -10px; display: inlin
e}

@media all and (min-width: 0px){

    .news_list01 h2 span{float: right; height: 19px; margin: 0 0 0 0; padding-top: 16p
x; padding-bottom: 0; display: inline}

}
```

HTML 代码片断级别的 hack（仅 IE 识别）

① *<!--[if !IE]> 除 IE 外都可识别的代码片断<![endif]-->*

② *<!--[if IE]> 所有的 IE 可识别的代码片断 <![endif]-->*

③ *<!--[if IE 7]> 仅 IE7 可识别的代码片断 <![endif]-->*

④ *<!--[if lt IE 7]> IE7 以及 IE7 以下版本可识别的代码片断<![endif]-->*

⑤ *<!--[if gte IE 7]> IE7 以及 IE7 以上版本可识别的代码片断 <![endif]-->*

用脚本设置 CSS 属性

设置元素的 style 样式

```
Var spanElement = document.getElementById("mySpan");

//下面写法保证出 IE 外，所有浏览器可用

spanElement.setAttribute("style","font-weight:bold;color:red;");

//下面的写法保证 IE 可用

spanElement.style.cssText="font-weight:bold;color:red;";
```

设置元素的 class 属性

```
Var element = document.getElementById("myElement");

//下面的写法保证除 IE 外，所有浏览器可用

Element.setAttribute("class","styleClass");

//下面写法保证 IE 可用

Element.setAttribute("className","styleClass");
```

具体 CSS 效果的实现

按钮悬停时鼠标呈现手的形状

cursor:hand 和 *cursor:pointer* 效果是一样的，当鼠标移动至该元素时呈现手的形状。但是应该尽量使用 *cursor:pointer* 而非 *cursor:hand*，因为 *cursor:hand* 只有 IE 识别，而 *cursor:pointer* 才是 CSS2.0 的标准属性，IE 之外的浏览器也支持。

窗口滚动条显示问题

在使用弹出窗口或者框窗口架的时候，有时会有多余的滚动条出现，这时需要从多个方面进行确认：

1. 弹出窗口时 *window.open* 方法参数中设置的窗样式是否定义了 scroll=yes
2. 框架标签的属性中是否设置了 scrolling="yes"
3. 窗口或框架内页面的 CSS 中，是否对 html 或 body 的 overflow 进行了样式定义，如果没有请参考如下代码。

```
html {
    margin: 0;
    padding: 0;
    overflow-x:hidden;
    overflow-y:hidden;
}
```

line-height 属性

line-height 行高指的是文本行的基线间的距离，即字体最底端与字体内部顶端之间的距离。如下图所示：

文本之间的空白距离（行距）不仅仅是行高决定的，同时也受字号的影响。有时侯同一行内的不同元素底边没有对齐，有可能就是行高不统一造成的，这时关调整高度和对齐方式是不够的，还需要调整 *line-height* 属性。

display:inline-block

display 属性有三个值：*block*，*inline*，*inline-block*。其中 *display:block* 就是将元素显示为块级元素；*display:inline* 就是将元素显示为行内元素；*display:inline-block* 将对象呈递为内联对象，但是对象的内容作为块对象呈递。旁边的内联对象会被呈递在同一行内。

l *block* 元素的特点是：总是在新行上开始；高度，行高以及顶和底边距都可控制；宽度缺省是它的容器的 100%，除非设定一个宽度。* *，*<p>*，*<h1>*，*<form>*，*<ul>* 和 *<li>*是块元素的例子， *display* 属性默认值为 *block*。

l *inline* 元素的特点是： 和其他元素都在一行上；高，行高及顶和底边距不可改变；宽度就是它的文字或图片的宽度，不可改变。**，*<a>*，*<label>*，*<input>*，*<img>*，** 和**是 *inline* 元素的例子， *display* 属性默认值为 *inline*。

l *inline-block* 的元素特点：呈现为内联对象，四周元素保持在同一行，但可以设置宽度和高度地块元素的属性，目前 IE8、Firefox3、Opera、Safari 都可以支持该属性了。

div 中的文字自动换行问题

目前控制换行是使用以下 CSS：

```
div. content {
    word-wrap:break-word;
```

```
    overflow:hidden;
}
```

在 IE 、Firefox、Safari、Chrome 下没有任何问题，但是在 Opera 下，长串英文会被遮住超出的内容。如果想要让长串英文字符也自动换行，还需要设置 *word-break:break-all;*（但是，此方式会导致普通的英文语句中的单词会被断开，ie 下也是）。英文单词在排版规则上不应该被断开，长串英文字符其实就是一个比较长的单词，自然也不需要断开换行显示了，所以一般不需要额外设置 *word-break:break-all;* 。

textarea 中的文字自动换行问题

在 textarea 中设置输入内容的自动换行，也是在 CSS 中设置 *word-wrap:break-word;*属性。需要额外注意的是 textarea 元素本身有一个 warp 属性，其取值含义如下：

- *off*：不自动换行；
- *hard*：自动硬回车换行，换行标记一同被传送到服务器端（Opera、Chrome 下不传）；
- *soft*：自动软回车换行，换行标记不会传送到服务器端。

避免对 IE 的依赖

DHTML

DHTML 是个好东西，大大方面了前端的交互实现，使得获取页面元素以及动态修改页面元素变的简单无比。但是所有的浏览器都认识这些语法吗？

document.all

document.all 不是所有浏览器都能识别，要写出更通用的代码，最好还是通过 id 来得到，使用 *document.getElementById(...)*

element.outerText, element.innerText, element.outerHTML, element.innerHTML

element.outerText, *element.innerText*, *element.outerHTML* 是属于 IE 特有的, 而 *element.innerHTML* 是通用的。如果要在其他浏览器下使用不通用的属性，可以参考以下代码实现：

```
if(!isIE()){
    HTMLElement.prototype.__defineGetter__("innerText",
    function(){
        var anyString = "";
        var childS = this.childNodes;
        for(var i=0; i<childS.length; i++){

            if(childS[i].nodeType==1)
                anyString += childS[i].innerText;
            else if(childS[i].nodeType==3)
                anyString += ahildS[i].nodeValue;
        }
        return anyString;
    }
    );
```

```
    HTMLElement.prototype.__defineSetter__("innerText",
    function(sText){
        this.textContent=sText;


    }
    );
}
```

document.forms.actionForm.inputName.value

之前用 *document.all.title.value* 来获取名为 *actionForm* 的标单中名为 *title* 的 *input* 域值得地方，应该改为 *document.forms.actionForm.input.value*，要这么使用，首先要保证 HTML 中 form 标签与其他标签结构上有完整的闭合关系。

Table 操作

moveRow (iSource , iTarget)方法

oRow = tableObject.moveRow (iSource , iTarget)，这个方法可以很方便实现 table 中 tr 的动态顺序调整。但是这个方法是 IE 内核自己实现的，不是 DOM 标准方法，所以别的浏览器没有。在使用到了这些 IE 独有的方法的地方，要么换用标准的 DOM 节点操纵方式——*insertBefore(currobj, beforeObj.nextSibling)*，要么先在 HTMLDocument 类的 prototype 上自己实现一个 moveRow 方法：

```
function getTRArray(){
    ......
```

```
    //将需要操纵的tr 都放入数组作为该方法的返回值
}


function getTRByIndex(sourceELIndex){
    var trArray = getTRArray();
    var result = trArray[sourceELIndex];
    return result;
}


if( !isIE && HTMLElement.moveRow == null )
{
    //入参说明:
    //sourceELIndex ： 需要移动的tr 在tbody 中的第几行（>=1）
    //targetELIndex ： 需要移动到tbody 中的第几行（>=1，<=行数）
    HTMLElement.prototype.moveRow = function(sourceELIndex,targetELIndex)
    {
        var tbObject = document.getElementById("tbodyEL");
        var resultEL;
```

```
        if(sourceELIndex>=targetELIndex)
        {//move up
            var s = sourceELIndex-1;
            var t = targetELIndex-1;
        }else{
            var s = sourceELIndex-1;
            var t = targetELIndex;
        }
        var sourceEL = getTRByIndex(s);
        var targetEL = getTRByIndex(t);
         //alert("begin"+sourceELIndex+targetELIndex);
        //alert("begin"+s+t);
        tbObject.insertBefore(sourceEL,targetEL);
        resultEL = sourceEL;
        return resultEL;
    }
}
```

剪贴板操作

IE、 Firefox 可以支持 JavaScript 往剪贴板写入内容

IE 可以很方便的支持剪贴板内容写入命令，可以使用 *execCommand()*，也可以利用 *window.clipboardData*。

使用 *execCommand*，需要先从页面选中要复制到剪贴板的内容，如以下代码：

```
var doc = obj.createTextRange();
doc.select();
doc.execCommand('copy');
```

使用 *window.clipboardData* 的方法如下，代码中同时实现了 Firefox 下写入剪贴板的功能：

```
if(window.clipboardData) //IE
        {
          window.clipboardData.clearData();
          window.clipboardData.setData("Text", txt);
        }
        else if (window.netscape)
        {
```

```
try { netscape.security.PrivilegeManager.enablePri
vilege("UniversalXPConnect");
}
catch (e)
{
alert("please visit 'about:config' and set signed.applets.codebase_principa
l_support as 'true'");
//提示用户开放浏览器的安全性设置
}

var clip = Components.classes["@mozilla.org/widget/clipboard;1"].createInst
ance(Components.interfaces.nsIClipboard);
if (!clip)
return;
var trans = Components.classes["@mozilla.org/widget/transferable;1"].creat
eInstance(Components.interfaces.nsITransferable);
if (!trans)
return;
trans.addDataFlavor('text/unicode');
var str = new Object();
```

```
            var len = new Object();

            var str = Components.classes["@mozilla.org/supports-string;1"].createInsta
nce(Components.interfaces.nsISupportsString);

            var copytext = txt;

            str.data = copytext;

            trans.setTransferData("text/unicode",str,copytext.length*2);

            var clipid = Components.interfaces.nsIClipboard;

            if (!clip)

            return;

            clip.setData(trans,null,clipid.kGlobalClipboard);

}
```

以上代码可以实现在 IE、Firefox 中往剪贴版写入自定义内容，但是 opera 和 webkit 内核浏览器对安全性要求更高，不支持这种 javascript 直接操作剪贴板内容的方式，只能通过别的脚本语言“曲线救国”了。

Opera 、Safari、Chrome 使用 ActionScript 往剪贴板写入内容

具体的实现可以将原本的动作按钮用 flex 或 flash 实现其外观，替换原来的图片或文字按钮，然后在点击该按钮时，执行以下 ActionScript 脚本：

```
//从浏览器环境中获得需要写入到剪贴板的内容

var s:String = String(ExternalInterface.call("getURL4Clipboard")); //getURL4Clipboard 是
页面上 return 剪贴板内容的 javascript 方法
```

```
//设置剪贴板内容

System.setClipboard(s);

//调用完成设置剪贴板内容后需要继续的JavaScript 函数，比如提示用户信息等

ExternalInterface.call("copyURLCompleted");  //copyURLCompleted 是页面上的javascript 方法，继续执行复制后的工作
```

事件

简单事件模型和高级事件模型

简单事件，就是事件与页面元素直观的绑定在一起的形式，如：

```
<div onclick="alert(this.innerHTML);">

element.onclick = function(){alert(this.innerHTML);}
```

只要不是用了个别浏览器独有的事件，一般的 click，mouseover 事件等在各浏览器中都可以这么使用。

但是当一个事件需要绑定多个监听，或者需要动态注册/移出监听时，简单事件模型就不够用了，需要使用高级事件模型（IE 和其他浏览器在使用高级事件模型时就有区别了）：

```
//注册
 function addEventHandler(element, evtName, callback, useCapture) {
    //DOM 标准
    if (element.addEventListener) {
        element.addEventListener(evtName, callback, useCapture);
    } else {
       //IE 方式,忽略useCapture 参数
       element.attachEvent('on' + evtName, callback);
```

```
    }
  }

  //移除
  function removeEventHandler(element, evtName, callback, useCapture) {
    //DOM 标准
    if (element.removeEventListener) {
        element.removeEventListener(evtName, callback, useCapture);
    } else {
      //IE 方式,忽略 useCapture 参数
      element.dettachEvent('on' + evtName, callback);
    }
  }
```

<a>标签中 onclick 事件与 href 属性的调用顺序关系

在 a 标签响应点击事件时，会先进行 onclick 事件的响应，再执行 href 中的跳转方法。如以下标签点击后会先后提示“button”和“href”：

```
<a href="javascript:alert("href");" onclick="alert(this.innerHTML);"> button </a>
```

但是，最好不要在 href 中定义具体的 javascript 方法，因为这个是定义跳转的链接地址的属性，如果需要先后执行两个 javascript 方法，应该这样写：

```
<a href="#" onclick="alert(this.innerHTML); alert("href");"> button </a>
```

但是上面的写法中如果 onclick 绑定的响应方法中并没有提交请求跳转至别的页面，那么会发现当前网页做了一次刷新，因为以上代码中 href="#"表示跳转到当前页的顶部，但是并没有发出新的 html 请求。有的时候，我们并不希望页面在响应 onclick 事件后又跳回顶部（尤其是页面高度较长，出现滚动条，并且该链接位于页面底部时，跳转至顶部后用户还需要拖动滚动条找回原来位置继续操作时），那么应该在 onclick 后返回 false 值，阻止继续进行 href 定义的动作，如：

```
<a href="#" onclick="alert(this.innerHTML); alert("href");return false;"> button </a>
```

或者将#替换成空的 javascript 语句：

```
<a href="javascript:void(0)" onclick="alert(this.innerHTML); alert("href");"> button </a>
```

onload 事件的调用顺序

有的时候在页面初始化时需要调用一些脚本来设置页面元素的初始状态，最标准的做法是用 *<body onload="">*方式或者 *document.onload* 方式调用。*onload* 的事件的触发会在页面元素渲染完毕之后调用，这样就保证了不会出现脚本执行时找不到未渲染的页面元素的情况。如果是在*<head>*区域的*<script>*块中执行脚本，并使用了页面元素时，就很有可能出现找不到元素的错误。如果是在*<body>*区域的*<script>*块中执行脚本，只能使用该*<script>*块之前的页面元素，因为*<body>*区域的元素基本是顺序解析的。

onchange 事件

*<input>*元素和*<select>*的*<option>*元素都支持 *onchangge* 事件，但是经常我们会发现元素的内容值变化了却没有触发 *onchangge* 事件。这是因为 *onchange* 事件的触发还有另一个条件:当前元素处于失去焦点(*onblur*)的状态。所以，*onchange* 事件只能捕获用户操作时的值变化，对于使用 javascript 脚本动态改变元素 value 的情况，它是捕获不到的。

事件截获

因除了 IE 之外的浏览器不能识别 *window.event 所以*要使用以下方式来获取当前事件和事件发生的目标元素：

```
var evt = e?e:(window.event?window.event:null);
```

```
var el = evt.srcElement?evt.srcElement:evt.target;
```

XML 文档处理 和 XMLHttpRequest 对象

加载 xml 文件

IE:

```
xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async="false";
xmlDoc.load("books.xml");
```

其他浏览器:

```
xmlDoc=document.implementation.createDocument("","",null);
xmlDoc.async="false";
xmlDoc.load("books.xml");
```

加载 xml 字符串

IE:

```
xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async="false";
xmlDoc.loadXML(txt);
```

其他浏览器:

```
parser=new DOMParser();
```

```
xmlDoc=parser.parseFromString(txt,"text/xml");
```

selectNodes()、selectSingleNode()方法

这两个方法是 IE 才有的,如果考虑跨浏览器,请尽量使用标准 DOM 操纵方式替代,或者在其他浏览器中先实现以上方法再使用。

xml 属性

IE 可以用 obj.xml 来显示 obj 的节点内容，但是这个属性是其他浏览器没有，所以要用*(new XMLSerializer()).serializeToString(obj)*方法来显示节点内容。

XMLHttpRequest 对象

获得 IE 和其他浏览器中的 XMLHttpRequest 对象

获得 IE 和其他浏览器中的 XMLHttpRequest 对象

```
function getXMLHTTPRequest()
{
   var myRequest = null;
   if(window.XMLHttpRequest)   // IE 以外的浏览器
   {
      myRequest = new XMLHttpRequest();
   }
   else if(typeof ActiveXObject != "undefined")  // IE
```

```
    {
        myRequest = new ActiveXObject("Microsoft.XMLHTTP");
    }
    return myRequest;
}
```

然后我们就可以使用上面的方法来进行请求操作了。

```
    var myRequest = getXMLHTTPRequest();
    xmlhttp.onreadystatechange=stateChange;
    xmlhttp.open("GET",url,true);
    xmlhttp.send(null);
```

open 方法参数说明

xmlhttp.open("GET",url,true)里第一个参数表示请求的方式，“POST”或者“GET”；第二个参数是发送请求的地址；第三个参数是表示是否用异步方式进行请求。如果采用异步方式请求，那么浏览器会在 send 请求后等待请求地址的响应同时，继续执行 send 之后的语句，得到响应后执行 xmlhttp.onreadystatechange 设置的名为 stateChange 的回调方法，所以要把得到响应后的后继操作写在这个回调方法里。

需要注意的是如果采用同步方式发送请求，浏览器会等请求响应后才执行 send 之后的语句，所以最好是把响应后的操作直接写在 send 语句之后，而不是像异步方式请求那样写在 xmlhttp.onreadystatechange 设置的回调方法中，因为各浏览器在同步请求后是否还执行

这个回调方法上还是有些不一致的地方：比如 firefox3.0 在发送同步请求得到响应后就不触发 onreadystatechange 事件，自然就不会执行这个回调方法了。

send 方法参数的默认值设置

不同浏览器的 XMLHttpRequest 虽然接口一样，但在不同浏览器中调用 XMLHttpRequest 的方法和属性却不尽相同。如 send 方法，在 IE 中可以不传参数，如 myXMLHttpRequest.send();仍然可以正常工作，而在 firefox 中，必须为 send 方法传一个参数，也就是说，在 firefox 中 send 方法参数没有默认值，必须为其赋值，哪怕是 null 也可以。

webkit 私有定义

使用在 Safari 和 WebKit 中可用的 CSS 高级特性，你可以为你的网站和网络应用带来一个新的级别的令人兴奋的东西。WebKit 是 Safari 浏览器和 Google Chrome 的渲染引擎。
因为浏览器会简单的无视他们不支持的 CSS 属性，所以在其他浏览器中，这些秘诀中的大部分可能会无效。使用只用 WebKit 支持的属性的网页在基于 WebKit 的浏览器中会有非常出色的视觉和体验，并且在其他浏览器中也会有某些效果。
注意：你在本文中看到的-webkit 前缀是一个浏览器生产商通常使用的一种方式，它暗示该 CSS 属性或规则尚未成为 W3C 标准的一部分。比如，box-shadow 属性还只是开发中的 CSS3 标准的一部分。基于 Mozilla 的浏览器使用-moz 前缀。

简单的阴影

让我们从向你展示为网页中的任意元素添加阴影效果是多么的简单开始吧。下面的代码片段将演示一个轻微旋转并有阴影的图片，这两个效果都是使用 CSS 添加的。

```
<img src="megan.jpg" style="-webkit-transform:rotate(5deg);-webkit-box-shadow:10px 10px 5px #888;" />
```

上面的代码中，transform CSS 属性实现图片旋转，box-shadow 属性为图片添加阴影效果。你可以改变旋转的角度，或者是阴影的参数，仅仅调整那些参数就 OK 了。

滚动与弹出

下一个示例将演示当你把鼠标放到一张图片上事，它会弹出的效果。实现这些只需要使用一个 hover 样式和一个度数改变。该效果的 CSS 如下所示。

```
<style>
 img { -webkit-transform: scale(0.5); }
```

```
img:hover { -webkit-transform: scale(1); }
</style>
<img src="megan.jpg" />
```

鼠标滑过图片，它就会弹出并变大

动画图片翻转

另一个前端工程师常见的现象是，当用户将鼠标放到图片上时，变换为另一张图片。在这个技巧中，让我们看看如何让图片从一个到另一个渐变交换，而不是简单的直接交换两张图片。实现这个效果的 CSS 和 HTML 如下：

```
div.swapper img { -webkit-transition: opacity 1s ease-in-out; }
img.img1, div.swapper:hover img.img2 { opacity: 1.0; }
div.swapper:hover img.img1, img.img2 { opacity: 0; }
<div class="swapper">
<img class="img1" style="position: absolute;" src="megan.jpg">
<img class="img2" src="megan2.jpg">
</div>
```

在这里，”transition”属性使用简化符号来指定（图片）过渡的所有参说。第一个参数将属性指定为动画，第二个参数指定时间，第三个参数为简便指定时间功能。”ease-in-out”只是众多你可以只有支配的时间功能中的一个。你还可以指定一个线形变换、渐入、渐出或高级的立体贝塞尔曲线效果！

CSS 多卷布局

使用纯 CSS 实现多卷，而不用 HTML 的 table 是件相当棘手的事情。由于 CSS3 用于多卷布局的属性在 Safari 和 WebKit 中已经可用，你可以明确的定义卷数，正确实现你所想要的效果。先看一下下面的 CSS 和 HTML 代码：

```
#columns {
-webkit-column-count: 3;
-webkit-column-gap: 25px;
-moz-column-count: 3;
-moz-column-gap: 25px;
column-count: 3;
column-gap: 25px;
}
<div id="columns">
<p>Column A</p>
<p>Column B</p>
<p>Column C</p>
</div>
```

这些代码定义了卷中的 HTML 代码。这些代码定义了这个 DIV 应该被分成 3 卷。每个段落就在他们自己的卷里面。

这些代码同样说明了一种在使用一种尚未成为 W3C 标准的一部分时的可靠机制.

这段代码指定了”column-count”和”column-gap”属性，并带有”-webkit”和”-moz”前缀，以及没有前缀的情况。这意味着这段代码将会像基于 Mozilla 的浏览器一样可以在 Safari 和 WebKit 的浏览器中运行，而且一旦 CSS3 标准被最终确定下来之后，那些前缀就可以去掉了。

CSS 圆角

圆角可能会给网站页面带来一些麻烦，比如，它可能需要为每个角使用一张图片，但是这可能会引起某些表现上的问题（比如不同的浏览器可能表现上会有细微的差别）。在 WebKit 中有效的 CSS3 的”border-radius”属性让实现圆角变得简单，它只需要几行简单的 CSS 代码。如下所示：

```
#boxes div { border: 2px solid black; padding: 10px; margin:5px; width:200px;
  text-align:center; background:#eee; }
<div id="boxes">
<div style="-webkit-border-radius:15px;">
  All sides
</div>
<div style="-webkit-border-bottom-left-radius:15px;-webkit-border-top-right-radius:15px;">
  Opposite corners
</div>
<div style="-webkit-border-top-left-radius:15px;-webkit-border-top-right-radius:15px;">
  Top
</div>
<div style="-webkit-border-top-right-radius:15px;-webkit-border-bottom-right-radius:15px;">
  Side
</div>
</div>
```

这段样式代码定义所有的在 boxes 里面的 DIV 都要有一个比较大的 border（边框）、边距、宽度、背景等。然后每个 div 标签的 border-radius 被设置为不同的方式。

第一个 div 中 border-radius 被设置为所有的角。然后第二个 div，只是左下角和右上角。第三个 div 就像一个 tab，只是上面的角被设置为圆角。最后一个 div 被设置为单边的圆角，只有右边的两个角是圆角。

表单控制

WebKit 同样提供一些新的控制来使用于你的页面。下面的例子演示一个水平滑动条、一些新样式的按钮以及一个很像 Safari 工具栏的搜索框

```
<input type="range" style="-webkit-appearance:slider-horizontal;"><br/><br/>
<button style="-webkit-appearance:button;width:200px;height:30px;">gradient button</button><br/><br/>
<button style="-webkit-appearance:push-button;width:200px;">aqua button</button><br/><br/>
<input type="text" style="-webkit-appearance:searchfield;" value="kitten"></input><br/>
```

html5 表单

输入框占位符

在 HTML5 中，开发人员可以非常容易 的显示一个占位符。什么是占位符？占位符就是出现在输入框的提示文本，当你点击输入栏位，它就自动消失。你可以把用户应该输入的文本样例在文本框提示出 来。一个例子，如果你有一个电话号码输入框，你可以设置占位符（XXX）XXX - XXXX，点击它们时就会消失。
支持的浏览器：FF3.7+、safari4+、chrome4+、opera11+、iphone4+

自动焦点事件

目前 HTML4 要做到自动焦点的方式是使用 JavaScript 把焦点放在一个表单的第一个输入字段。HTML5 只要加载一个网页，网页自动将焦点移到特 定的输入框，和 JavaScript 一样。区别是什么？由于现在只是一个 HTML 标记，用户可以很容易地在他们的浏览器禁用此属性。并非所有浏览器都支持 自动对焦功能，但浏览器不只是简单地忽略该属性。如果你想所有浏览器都行得通，只需添加新的 HTML5 自动对焦属性，然后检测浏览器是否支持自动对焦。如 果可以就不必使用自动对焦的脚本，如果没有的话，就要添加自动对焦的脚本。
支持的浏览器：IE4+、safari4+、chrome3+、opera10+

input 输入类型

电子邮件

email 类型用于应该包含 e-mail 地址的输入域。
在提交表单时，会自动验证 email 域的值。
例如：

```
<input type="email" name="user_email" />
```

网址

url 类型用于应该包含 URL 地址的输入域。
在提交表单时，会自动验证 url 域的值。

例如：

```
Homepage: <input type="url" name="user_url" />
```

数字

number 类型用于应该包含数值的输入域。
您还能够设定对所接受的数字的限定：

```
oints: <input type="number" name="points" min="1" max="10" />
```

属性	值	描述
max	*number*	规定允许的最大值
min	*number*	规定允许的最小值
step	*number*	规定合法的数字间隔（如果 step="3"，则合法的数是 -3,0,3,6 等）
value	*number*	规定默认值

数字滑块

range 类型用于应该包含一定范围内数字值的输入域。
range 类型显示为滑动条。
您还能够设定对所接受的数字的限定：

```
<input type="range" name="points" min="1" max="10" />
```

属性	值	描述
max	*number*	规定允许的最大值
min	*number*	规定允许的最小值
step	*number*	规定合法的数字间隔（如果 step="3"，则合法的数是 -3,0,3,6 等）

value	*number*	规定默认值

日历

HTML5 拥有多个可供选取日期和时间的新输入类型：
date - 选取日、月、年
month - 选取月、年
week - 选取周和年
time - 选取时间（小时和分钟）
datetime - 选取时间、日、月、年（UTC 时间）
datetime-local - 选取时间、日、月、年（本地时间）
下面的例子允许您从日历中选取一个日期：

```
Date: <input type="date" name="user_date" />
```

搜索

search 类型用于搜索域，比如站点搜索或 Google 搜索。
search 域显示为常规的文本域。
例如

```
Date: <input type="search" name="user_date" />
```

表单元素兼容列表

Input type	IE	Firefox	Opera	Chrome	Safari
email	No	4.0	9.0	10.0	No
url	No	4.0	9.0	10.0	No
number	No	No	9.0	7.0	No
range	No	No	9.0	4.0	4.0

Date pickers	No	No	9.0	10.0	No
search	No	4.0	11.0	10.0	No
color	No	No	11.0	No	No

WebKit 和 HTML5

WebKit 是一种浏览器引擎，支撑着 iPhone 内的 Mobile Safari 浏览器以及 Android 内的浏览器背后的技术。WebKit 也在其他的移动环境内有自己的用武之地，但是我们还是将我们的讨论集中于 iPhone 和 Android 平台。

WebKit 是一个开源项目，其起源可追溯到 K Desktop Environment (KDE)。WebKit 项目催生了面向移动设备的现代 Web 应用程序。虽然设备本身的能力和形态因素都相当重要，但移动用户最热衷的仍然是内容。如果移动用户可用的内容只是 Internet 用户可用内容的一个很小的子集，那么用户体验充其量也只能划分为二等。

我们当中的大多数人都更希望生活是连贯的 — 如果我们在家中的笔记本上访问了一个网站，我们同样希望在火车上旅行时仍然访问到同样的内容。内容是最好的应用程序。不管我们身在何处、在做什么，我们都想要访问到我们的数据。WebKit 让 iPhone 和 Android 平台上可以有丰富的内容。

有一点很值得注意，即 WebKit 还应用在了桌面的 Safari 浏览器内，该浏览器是 Mac OS X 平台默认的浏览器。不管我们讨论的是桌面版本还是 iPhone 或 Android 上的浏览器引擎，WebKit 均优先支持 HTML 和 CSS 特性。实际上，WebKit 还支持尚未被其他浏览器采纳的一些 CSS 样式 — 这些特性正在得到 HTML5 规范的考虑。

HTML5 规范是一个技术草案集，涵盖了各种基于浏览器的技术，包括客户端 SQL 存储、转变、转型、转换等。HTML5 的出现已经有些时间了，虽然尚未完成，但是一旦其特性集因主要浏览器平台支持的加入而逐渐稳定后，Web 应用程序的简陋开端将成为永久的记忆。Web 应用程序开发将成为主导 — 并且不只是在传统的桌面浏览器空间，还将在移动领域。移动将一跃成为首要考虑，而不再是后备之选。

移动 Web 应用程序的考虑

为了访问 Web 开发技术，如今，应用程序开发人员有几个选择。第一，应用程序可严格编写为服务器上的 HTML、CSS 和 JavaScript 文件。当然，HTML 内容可以产生自静态 HTML 文件，也可以从任何的服务器端技术（比如 PHP、 ASP.NET、Java Servlets 等）动态生成。所有这些技术追根到底都可简单地用术语 HTML 指代 — 这不是本文讨论的重点所在 — 并且最为重要的是，受 WebKit-支撑的浏览器能够在移动设备上解析和呈现 HTML。

用户通过在移动设备上（即 iPhone 或 Android）打开浏览器应用程序并输入目标服务器对应的 URL：http://yourcompanyname.com/applicationurl 来访问 Web 应用程序。

特定的某个移动 Web 应用程序总是能找到自己的位置：从一般的 Web 站点到高度特定于平台的移动 Web 应用程序。

一般站点的呈现

WebKit 内的呈现引擎，再配以 iPhone 和 Android 平台上的高度直观的 UI，实际上就使得几乎任何一个基于 HTML 的 Web 站点都能呈现在此设备上。Web 页能被正确呈现，不再像原来的移动浏览器体验：内容被包裹起来或是根本不显示。当页面加载后，内容通常被完全缩放以便整个页面都可见，尽管内容会被缩放得非常小，甚至不可读，如图 1 所示。不过，页面是可滚动、放大、缩小的，这就提供了对全部内容的访问。默认地，浏览器使用 980 像素宽的视见区或逻辑尺寸。

要想使 Web 页面从一般的页面变成支持移动设备的页面，Web 应用程序可以在几个方面进行修改。

虽然页面可以在 WebKit 中正确呈现，但是，一个以鼠标为中心的设备（比如笔记本或台式机）与一个以触摸为中心的设备（比如一个 iPhone 或 Android 智能手机）还是有区别的。其中主要的一些差异包括“可单击”区域的物理大小、“悬浮样式”的缺少以及完全不同的事件顺序。如下所列的是在设计一个能被移动用户正常查看的 Web 站点时需要注意的一些事情：

•iPhone/Android 浏览器呈现的屏幕是可读的 — 大大好于传统的移动浏览器 — 所以不要急于草草制作您网站的移动版本。

•手指要大过鼠标指针。在设计可单击的导航时要特别注意这一点 — 不要把链接放得相互太靠近，因为用户不太可能单击了一个链接而不触及相邻的链接。

•悬浮样式将不再奏效，因为用手指不能进行用鼠标指针进行的“悬浮”。

•诸如 mouse-down、mouse-move 等事件在基于触摸的设备上自然大相径庭。这类事件中有一些将被取消，不要指望移动设备上的事件顺序与桌面浏览器上的一样。

让我们来看看要使一个 Web 站点对 iPhone 或 Android 访客具有友好性所面临的最为明显的一个挑战：屏幕大小。我们今天使用的实际移动屏幕尺寸是 320x480。请注意由于用户可能会选择横向查看 Web 内容，所以屏幕大小也可以是 480x320。正如我们在图 1 中看到的，WebKit 将能很好地呈现面向桌面的 Web 页面，但是文本可能会太小以至于若不进行缩放或其他操作就无法有效阅读内容。那么，我们该如何应对这个问题呢？

最为直观也是最不唐突的适合移动用户的方式是通过使用一个特殊的 metatag：viewport。metatag 是一个放入 HTML 文档的 head 元素内的 HTML 标记。如下是一个使用 viewport 标记的简单例子：<meta name="viewport" content="width=device-width" />。当这个 metatag 被添加到一个 HTML 页面后，我们看到此页面被缩放到更为适合这个移动设备的大小，如图 2 所示。如果浏览器不支持此标记，它会简单地忽略此标记。

为了设置特定的值，将 viewport metatag 的 content 属性设为一个显式的值： *<meta name="viewport" content="width=device-width, initial-scale=1.0 user-scalable=yes" />*。通过改变初始值，屏幕就可以按要求被放大或缩小。将值分别设置在 1.0 和 1.3 之间对于 iPhone 和 Android 平台是比较合适的。viewport metatag 还支持最小和最大伸缩，可用来限制用户对呈现页面的控制力。

自具有 320x480 布局的 iPhone 面世以来，其形态系数就一直没有改变过，而随着来自不同制造商、针对不同用户群的更多设备的出现，Android 则有望具备更多样的物理特点。在开发应用程序并以诸如 Android 这类移动设备为目标时，一定要考虑屏幕尺寸、形态系数

以及分辨率方面的潜在多样性。

除了 Android 设备与其他设备之间的这些物理差异之外，经验还表明 Android 的软件还通过设备内置的（on-device）浏览器设置对页面的呈现实施了更多控制。不仅稳定，Android 平台还很灵活。取决于 SDK 等级和制造商，某个设备上的设置很可能不同于您的开发环境。