**CSS3边框 圆角效果 border-radius**

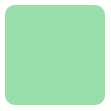
**border-radius**是向元素添加圆角边框。

**使用方法：**

**border-radius:10px;** /\* 所有角都使用半径为10px的圆角 \*/

[](http://img.mukewang.com/52e216d2000195ef01110111.jpg)

**border-radius: 5px 4px 3px 2px;** /\* 四个半径值分别是左上角、右上角、右下角和左下角，顺时针 \*/

[](http://img.mukewang.com/52e216f9000131a201110111.jpg)

不要以为border-radius的值只能用**px**单位，你还可以用**百分比**或者**em**，但兼容性目前还不太好。

**实心上半圆：**

方法：把高度(height)设为宽度（width）的一半，并且只设置左上角和右上角的半径与元素的高度一致（大于也是可以的）。

div{

height:50px;/\*是width的一半\*/

width:100px;

background:#9da;

border-radius:50px 50px 0 0;/\*半径至少设置为height的值\*/

}

**实心圆：**  
方法：把宽度（width）与高度(height)值设置为一致（也就是正方形），并且四个圆角值都设置为它们值的一半。如下代码：

div{

height:100px;/\*与width设置一致\*/

width:100px;

background:#9da;

border-radius:50px;/\*四个圆角值都设置为宽度或高度值的一半\*/

}

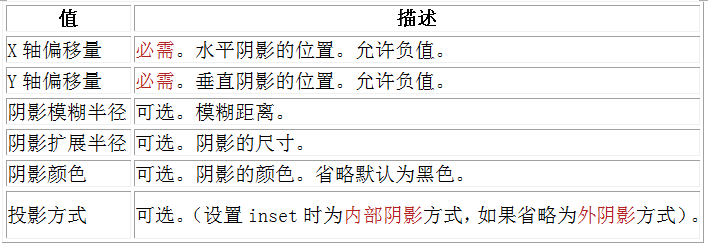
## CSS3边框 阴影 box-shadow（一）

box-shadow是向盒子添加阴影。支持添加一个或者多个。

很简单的一段代码，就实现了投影效果，酷毙了。我们来看下语法：

box-shadow: X轴偏移量 Y轴偏移量 [阴影模糊半径] [阴影扩展半径] [阴影颜色] [投影方式];

参数介绍：

[](http://img.mukewang.com/54292d620001ffb107080250.jpg)

注意：inset 可以写在参数的第一个或最后一个，其它位置是无效的。

**为元素设置外阴影：**

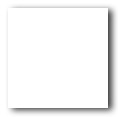
示例代码：

.box\_shadow{

box-shadow:4px 2px 6px #333333;

}

效果：

[](http://img.mukewang.com/52e21c9000013c8f01180117.jpg)

**为元素设置内阴影：**

示例代码：

.box\_shadow{

box-shadow:4px 2px 6px #333333 inset;

}

效果：

[](http://img.mukewang.com/54292b990001ebea01290115.jpg)

**添加多个阴影：**

以上的语法的介绍，就这么简单，如果添加多个阴影，只需用逗号隔开即可。如：

.box\_shadow{

box-shadow:4px 2px 6px #f00, -4px -2px 6px #000, 0px 0px 12px 5px #33CC00 inset;

}

效果：

[](http://img.mukewang.com/52e21d210001356d01170116.jpg)

## CSS3边框 阴影 box-shadow（二）

**1、阴影模糊半径与阴影扩展半径的区别**

阴影模糊半径：此参数可选，其值只能是为正值，如果其值为0时，表示阴影不具有模糊效果，其值越大阴影的边缘就越模糊；

阴影扩展半径：此参数可选，其值可以是正负值，如果值为正，则整个阴影都延展扩大，反之值为负值时，则缩小；

**2、X轴偏移量和Y轴偏移量值可以设置为负数**

box-shadow: X轴偏移量 Y轴偏移量 [阴影模糊半径] [阴影扩展半径] [阴影颜色] [投影方式];

X轴偏移量为负数：

.boxshadow-outset{

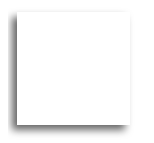
width:100px;

height:100px;

box-shadow:-4px 4px 6px #666;

}

效果图：

[](http://img.mukewang.com/548fd91c000140a901540143.jpg)

Y轴偏移量为负数：

.boxshadow-outset{

width:100px;

height:100px;

box-shadow:4px -4px 6px #666;

}

效果图：

[](http://img.mukewang.com/548fd93e00011dd101570142.jpg)

## CSS3边框 为边框应用图片 border-image

顾名思义就是为边框应用背景图片，它和我们常用的background属性比较相似。例如：

background:url(xx.jpg) 10px 20px no-repeat;

但是又比背景图片复杂一些。

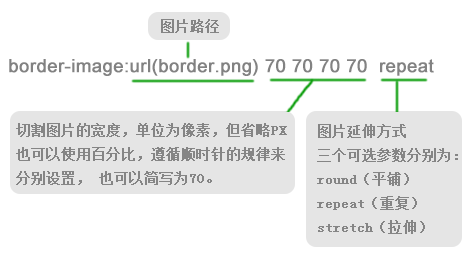
想象一下：一个矩形，有四个边框。如果应用了边框图片，图片该怎么分布呢？ 图片会自动被切割分成四等分。用于四个边框。

可以理解为它是一个切片工具，会自动把用做边框的图片切割。怎么切割呢？为了方便理解，做了一张特殊的图片，由9个矩形（70\*70像素）拼成的一张图（210\*210像素），并标注好序号，是不是像传说中的九宫图，如下：

[](http://img.mukewang.com/52e21fea000127e802100210.jpg)

我们把上图当作边框图片 来应用一下， 看一看是什么效果

根据border-image的语法：

[](http://img.mukewang.com/52e2201a0001b1e004720260.jpg)

#border-image{

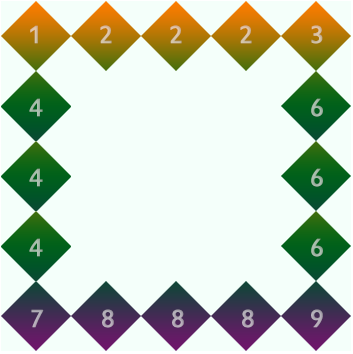
background:#F4FFFA;

width:210px; height:210px; border:70px solid #ddd;

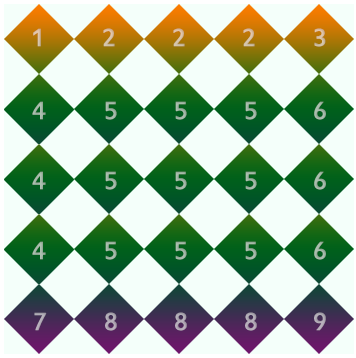
**border-image:url(borderimg.png) 70 repeat**

}

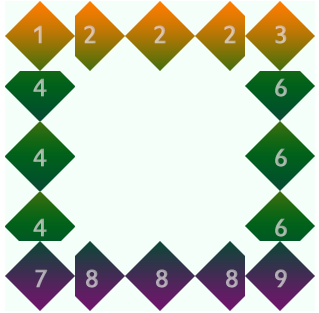
效果：

[](http://img.mukewang.com/52e220780001091a03530354.jpg)

从序号可以看出div的四个角分别对应了背景图片的四个角。而**2,4,6,8** 被重复。**5在哪？**因为是从四周向中心切割图片的所以，5显示不出来。而在chrome浏览器中5是存在的，下图的样子：

[](http://img.mukewang.com/52e220be0001cef603560358.jpg)

**repeat**的意思就是重复，目前因为是刚好被整除，效果看不出来。如果改下DIV的宽高，再来看重复的效果：

[](http://img.mukewang.com/52e2210700016f5603190313.jpg)

边角部分为裁掉了，可见**repeat**就是一直重复，然后超出部分剪裁掉，而且是居中开始重复。

**Round 参数**：Round可以理解为圆满的铺满。为了实现圆满所以会压缩（或拉伸）；

#border-image {

width:170px;

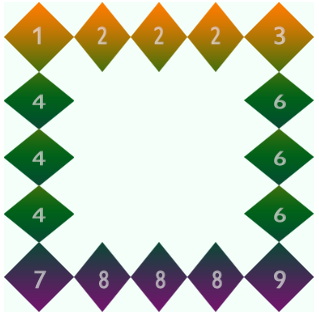
height:170px;

border:70px solid;

border-image:url(borderimg.png) 70 round;

}

效果：

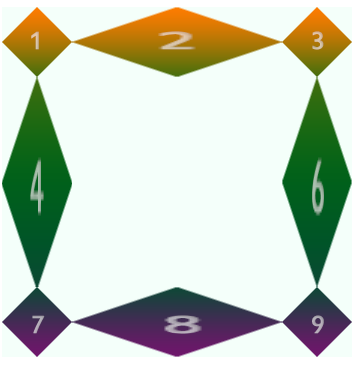
[](http://img.mukewang.com/52e2216700014c4103190312.jpg)

可见图片被压扁了。

Stretch 很好理解就是拉伸，有多长拉多长。有多远“滚”多远。

border-image:url(borderimg.png) 70 stretch

看一下效果：

[](http://img.mukewang.com/52e2218d0001d45403530366.jpg)

2,4,6,8分别被拉伸显示。

注意：Chrome下，中间部分也会被拉伸，webkit浏览器对于round属性和repeat属性似乎没有区分，显示效果是一样的。

Firefox 26.0 下是可以准确区分的。

## CSS3颜色 颜色之RGBA

RGB是一种色彩标准，是由红(R)、绿(G)、蓝(B)的变化以及相互叠加来得到各式各样的颜色。RGBA是在RGB的基础上增加了控制**alpha**透明度的参数。

语法：

color：rgba(R,G,B,A)

以上R、G、B三个参数，正整数值的取值范围为：0 - 255。百分数值的取值范围为：0.0% - 100.0%。超出范围的数值将被截至其最接近的取值极限。并非所有浏览器都支持使用百分数值。A为透明度参数，取值在0~1之间，不可为负值。

代码示例：

background-color:rgba(100,120,60,0.5);

## CSS3颜色 渐变色彩

**CSS3 Gradient**分为**线性渐变(linear)**和**径向渐变(radial)**。由于不同的渲染引擎实现渐变的语法不同，这里我们只针对线性渐变的 W3C 标准语法来分析其用法，其余大家可以查阅相关资料。W3C 语法已经得到了 IE10+、Firefox19.0+、Chrome26.0+ 和 Opera12.1+等浏览器的支持。

这一小节我们来说一下**线性渐变：**

[](http://img.mukewang.com/54b72b2e0001500103790158.jpg)

**参数：**

第一个参数:**指定渐变方向**，可以用“**角度**”的关键词或**“英文”**来表示：

[](http://img.mukewang.com/542a25da00017e9406980223.jpg)

（单击图片可放大）

第一个参数省略时，默认为“180deg”，等同于“to bottom”。

第二个和第三个参数，**表示颜色的起始点和结束点**，可以有多个颜色值。

background-image:linear-gradient(to left, red, orange,yellow,green,blue,indigo,violet);

效果图：

[](http://img.mukewang.com/54b72c080001611c04230192.jpg)

## CSS3文字与字体 text-overflow 与 word-wrap

**text-overflow**用来设置是否使用一个**省略标记**（...）标示对象内文本的溢出。

语法：

[](http://img.mukewang.com/53070cc00001a5bc06000200.jpg)

但是**text-overflow**只是用来说明文字溢出时用什么方式显示，要实现溢出时产生**省略号**的效果，还须定义**强制文本在一行内显示**（white-space:nowrap）及**溢出内容为隐藏**（overflow:hidden），只有这样才能实现**溢出文本显示省略号**的效果，代码如下：

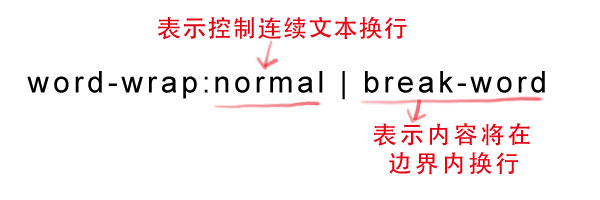
text-overflow:ellipsis;

overflow:hidden;

white-space:nowrap;

同时，**word-wrap**也可以用来设置**文本行为**，当前行超过指定容器的边界时是否断开转行。

语法：

[](http://img.mukewang.com/53070cf700018a2b06000200.jpg)

**normal**为浏览器默认值，**break-word**设置在**长单词**或 **URL地址**内部进行换行，此属性不常用，用浏览器默认值即可。

## CSS3文字与字体 嵌入字体@font-face

**@font-face**能够加载服务器端的字体文件，让浏览器端可以显示用户电脑里没有安装的字体。

语法：

@font-face {

font-family : 字体名称;

src : 字体文件在服务器上的相对或绝对路径;

}

这样设置之后，就可以像使用普通字体一样在**（font-\*）**中设置字体样式。

比如：

p {

font-size :12px;

font-family : "My Font";

/\*必须项，设置@font-face中font-family同样的值\*/

}

## CSS3文字与字体 文本阴影text-shadow

text-shadow可以用来设置文本的阴影效果。

**语法：**

text-shadow: X-Offset Y-Offset blur color;

X-Offset：表示阴影的水平偏移距离，其值为正值时阴影向右偏移，反之向左偏移；

Y-Offset：是指阴影的垂直偏移距离，如果其值是正值时，阴影向下偏移，反之向上偏移；

Blur：是指阴影的模糊程度，其值不能是负值，如果值越大，阴影越模糊，反之阴影越清晰，如果不需要阴影模糊可以将Blur值设置为0；

Color：是指阴影的颜色，其可以使用rgba色。

比如，我们可以用下面代码实现设置阴影效果。

text-shadow: 0 1px 1px #fff

## CSS3背景 background-origin

设置元素背景图片的**原始起始位置**。

语法：

background-origin ： border-box | padding-box | content-box;

参数分别表示背景图片是从**边框**，还是**内边距（默认值）**，或者是**内容区域**开始显示。

效果如下：



**需要注意的是**，如果背景不是**no-repeat**，这个属性无效，它会从边框开始显示。

## CSS3背景 background-clip

用来将背景图片做适当的**裁剪**以适应实际需要。

语法：

background-clip ： border-box | padding-box | content-box | no-clip

参数分别表示从**边框、**或**内填充**，或者**内容区域**向外裁剪背景。**no-clip**表示不裁切，和**参数border-box**显示同样的效果。backgroud-clip默认值为**border-box**。

效果如下图所示：



## CSS3背景 background-size

设置背景图片的大小，以**长度值**或**百分比**显示，还可以通过**cover**和**contain**来对图片进行伸缩。

语法：

background-size: auto | <长度值> | <百分比> | cover | contain

取值说明：

**1、auto**：默认值，不改变背景图片的原始高度和宽度；

**2、<长度值>**：成对出现如200px 50px，将背景图片宽高依次设置为前面两个值，当设置一个值时，将其作为图片宽度值来**等比缩放**；

**3、<百分比>**：0％~100％之间的任何值，将背景图片宽高依次设置为所在元素宽高乘以前面百分比得出的数值，当设置一个值时同上；

**4、cover**：顾名思义为**覆盖**，即将背景图片等比缩放以**填满整个容器**；

**5、contain**：容纳，即将背景图片等比缩放至**某一边紧贴容器边缘为止**。

**提示：**大家可以在右边的编辑窗口输入自己的代码尝试不同取值的效果。

## CSS3背景 multiple backgrounds

多重背景，也就是CSS2里**background**的属性外加**origin**、**clip**和**size**组成的新background的多次叠加，缩写时为用**逗号**隔开的每组值；用分解写法时，如果有多个背景图片，而其他属性只有一个（例如background-repeat只有一个），表明所有背景图片应用该属性值。

语法缩写如下：

background ： [background-color] | [background-image] | [background-position][/background-size] | [background-repeat] | [background-attachment] | [background-clip] | [background-origin],...

可以把上面的缩写拆解成以下形式：

background-image:url1,url2,...,urlN;

background-repeat : repeat1,repeat2,...,repeatN;

backround-position : position1,position2,...,positionN;

background-size : size1,size2,...,sizeN;

background-attachment : attachment1,attachment2,...,attachmentN;

background-clip : clip1,clip2,...,clipN;

background-origin : origin1,origin2,...,originN;

background-color : color;

**注意：**

1. 用逗号隔开每组 background 的缩写值；
2. 如果有 size 值，需要紧跟 position 并且用 "/" 隔开；
3. 如果有多个背景图片，而其他属性只有一个（例如 background-repeat 只有一个），表明所有背景图片应用该属性值。
4. background-color 只能设置一个。

举例：

有三张单独的图片：







使用多背景技术实现：



<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>多重背景</title>

<style type="text/css">

.demo{

width: 300px;

height: 140px;

border: 1px solid #999;

background-image: url(http://img.mukewang.com/54cf2365000140e600740095.jpg),

url(http://img.mukewang.com/54cf238a0001728d00740095.jpg),

url(http://img.mukewang.com/54cf23b60001fd9700740096.jpg);

background-position: left top, 100px 0, 200px 0;

background-repeat: no-repeat, no-repeat, no-repeat;

margin:0 0 20px 0;

}

.task {

width: 300px;

height: 140px;

border: 1px solid #999;

background:url(http://static.mukewang.com/static/img/logo\_index.png) no-repeat,

url(http://static.mukewang.com/static/img/logo\_index.png) no-repeat;

background-position: left top, 100px 0;

background-repeat: no-repeat, no-repeat;

}

</style>

</head>

<body>

<div class="demo"></div>

<div class="task"></div>

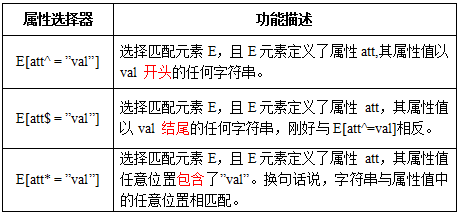
</body>

</html>

## CSS3选择器 属性选择器

   在HTML中，通过各种各样的属性可以给元素增加很多附加的信息。例如，通过id属性可以将不同div元素进行区分。

   在CSS2中引入了一些属性选择器，而CSS3在CSS2的基础上对属性选择器进行了扩展，新增了3个属性选择器，使得属性选择器有了**通配符**的概念，这三个属性选择器与CSS2的属性选择器共同构成了CSS功能强大的属性选择器。如下表所示：

[](http://img.mukewang.com/56653eba0001b07004610215.jpg)

(单击可放大)

**实例展示：**

**html代码**：

<a href="xxx.pdf">我链接的是PDF文件</a>

<a href="#" class="icon">我类名是icon</a>

<a href="#" title="我的title是more">我的title是more</a>

**css代码**

a[class^=icon]{

background: green;

color:#fff;

}

a[href$=pdf]{

background: orange;

color: #fff;

}

a[title\*=more]{

background: blue;

color: #fff;

}

 结果显示：

[](http://img.mukewang.com/53202e050001c07e03820068.jpg)

## CSS3 结构性伪类选择器—root

:root选择器，从字面上我们就可以很清楚的理解是根选择器，他的意思就是匹配元素E所在文档的根元素。在HTML文档中，根元素始终是<html>。

**示例演示：**

通过“:root”选择器设置背景颜色

**HTML代码：**

<div>:root选择器的演示</div>

**CSS代码：**

:root {

background:orange;

}

**演示结果：**

http://img.mukewang.com/531eb1d90001735d01560050.jpg

“:root”选择器等同于<html>元素，简单点说：

:root{background:orange}

html {background:orange;}

得到的效果等同。

建议使用:root方法。

另外在IE９以下还可以借助“:root”实现hack功能。

## CSS3 结构性伪类选择器—not

:not选择器称为**否定选择器**，和jQuery中的:not选择器一模一样，**可以选择除某个元素之外的所有元素**。就拿form元素来说，比如说你想给表单中**除submit按钮之外**的input元素添加红色边框，**CSS代码**可以写成：

form {

width: 200px;

margin: 20px auto;

}

div {

margin-bottom: 20px;

}

**input:not([type="submit"]){**

**border:1px solid red;**

**}**

**相关HTML代码：**

<form action="#">

<div>

<label for="name">Text Input:</label>

<input type="text" name="name" id="name" placeholder="John Smith" />

</div>

<div>

<label for="name">Password Input:</label>

<input type="text" name="name" id="name" placeholder="John Smith" />

</div>

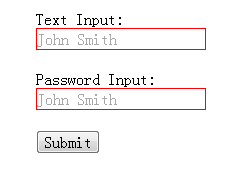
<div>

<input type="submit" value="Submit" />

</div>

</form> ​

**演示结果：**

****

## CSS3 结构性伪类选择器—empty

:empty选择器表示的就是空。用来选择没有任何内容的元素，这里没有内容指的是一点内容都没有，**哪怕是一个空格**。

示例显示：

比如说，你的文档中有三个段落p元素，你想把没有任何内容的P元素隐藏起来。我们就可以使用“:empty”选择器来控制。

**HTML代码：**

<p>我是一个段落</p>

<p> </p>

<p></p>​

**CSS代码：**

p{

background: orange;

min-height: 30px;

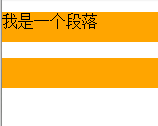
}

p:empty {

display: none;

}​

**演示结果：**

****

## CSS3 结构性伪类选择器—target

:target选择器称为目标选择器，用来匹配文档(页面)的**url的某个标志符的目标元素**。我们先来上个例子，然后再做分析。

**示例展示**

点击链接显示隐藏的段落。

**HTML代码：**

<h2><a href="**#brand**">Brand</a></h2>

<div class="menuSection" **id="brand"**>

content for Brand

</div>

**CSS代码：**

.menuSection{

display: none;

}

:target{/\*这里的:target就是指id="brand"的div对象\*/

display:block;

}

**演示结果：**

[](http://img.mukewang.com/53217ea30001103002230101.jpg)

**分析：**

1、具体来说，触发元素的URL中的标志符通常会包含一个**#号**，后面带有一个**标志符名称**，上面代码中是：#brand

2、：target就是用来匹配id为“brand”的元素（id="brand"的元素）,上面代码中是那个div元素。

**多个url（多个target）处理：**

就像上面的例子，#brand与后面的id="brand"是对应的，当同一个页面上有很多的url的时候你可以取不同的名字，只要#号后对的名称与id=""中的名称对应就可以了。  
如下面例子：  
html代码：

<h2><a href="#brand">Brand</a></h2>

<div class="menuSection" id="brand">

content for Brand

</div>

<h2><a href="#jake">Brand</a></h2>

<div class="menuSection" id="jake">

content for jake

</div>

<h2><a href="#aron">Brand</a></h2>

<div class="menuSection" id="aron">

content for aron

</div>

css代码：

#brand:target {

background: orange;

color: #fff;

}

#jake:target {

background: blue;

color: #fff;

}

#aron:target {

background: red;

color: #fff;

}

上面的代码可以对不同的target对象分别设置不的样式。

## CSS3 结构性伪类选择器—first-child

“:first-child”选择器表示的是**选择父元素的第一个子元素的元素E**。简单点理解就是选择元素中的第一个子元素，记住是**子元素**，而不是后代元素。

**示例演示**

通过“:first-child”选择器定位列表中的第一个列表项，并将序列号颜色变为红色。

**HTML代码：**

<ol>

<li><a href="##">Link1</a></li>

<li><a href="##">Link2</a></li>

<li><a href="##">link3</a></li>

</ol>

**CSS代码：**

ol > li{

font-size:20px;

font-weight: bold;

margin-bottom: 10px;

}

ol a {

font-size: 16px;

font-weight: normal;

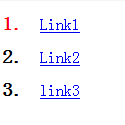
}

ol > li:first-child{

color: red;

}

**演示结果：**



## CSS3 结构性伪类选择器—last-child

“:last-child”选择器与“:first-child”选择器作用类似，不同的是“:last-child”选择器选择的是元素的最后一个子元素。例如，需要改变的是列表中的最后一个“li”的背景色，就可以使用这个选择器，

ul>li:last-child{background:blue;}

**示例演示**

在博客的排版中，每个段落都有15px的margin-bottom，假设不想让博客“post”中最后一个段落不需要底部的margin值，可以使用“:last-child”选择器。

**HTML代码：**

<div class="post">

<p>第一段落</p>

<p>第二段落</p>

<p>第三段落</p>

<p>第四段落</p>

<p>第五段落</p>

</div>​

**CSS代码：**

.post {

padding: 10px;

border: 1px solid #ccc;

width: 200px;

margin: 20px auto;

}

.post p {

margin:0 0 15px 0;

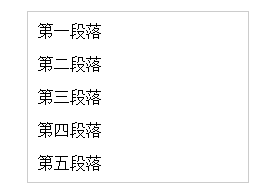
}

.post p:last-child {

margin-bottom:0;

}

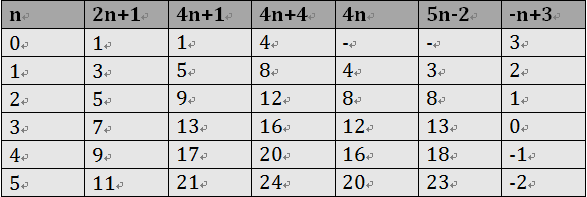
**演示结果：**

****

## CSS3 结构性伪类选择器—nth-child(n)

“:nth-child(n)”选择器用来定位某个**父元素**的**一个或多个特定的子元素**。其中“n”是其参数，而且可以是整数值(1,2,3,4)，也可以是表达式(2n+1、-n+5)和关键词(odd、even)，但参数n的起始值始终是1，而不是0。也就是说，参数n的值为0时，选择器将选择不到任何匹配的元素。

**经验与技巧:**当“:nth-child(n)”选择器中的n为一个表达式时，其中n是从0开始计算，当表达式的值为0或小于0的时候，不选择任何匹配的元素。如下表所示：



**案例演示**

  通过“:nth-child(n)”选择器，并且参数使用表达式“2n”，将偶数行列表背景色设置为橙色。

**HTML代码：**

<ol>

<li>item1</li>

<li>item2</li>

<li>item3</li>

<li>item4</li>

<li>item5</li>

<li>item6</li>

<li>item7</li>

<li>item8</li>

<li>item9</li>

<li>item10</li>

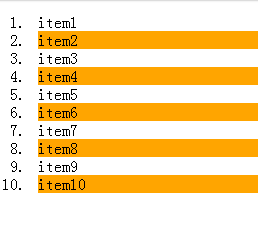
</ol>​

**CSS代码：**

ol > li:nth-child(2n){

background: orange;

}

**演示结果：  
**

## CSS3 结构性伪类选择器—nth-last-child(n)

“:nth-last-child(n)”选择器和前面的“:nth-child(n)”选择器非常的相似，只是这里多了一个“last”，所起的作用和“:nth-child(n)”选择器有所区别，从某父元素的最后一个子元素开始计算，来选择特定的元素。

**案例演示**

选择列表中倒数第五个列表项，将其背景设置为橙色。

**HTML代码：**

<ol>

<li>item1</li>

<li>item2</li>

<li>item3</li>

<li>item4</li>

<li>item5</li>

<li>item6</li>

<li>item7</li>

<li>item8</li>

<li>item9</li>

<li>item10</li>

<li>item11</li>

<li>item12</li>

<li>item13</li>

<li>item14</li>

<li>item15</li>

</ol>​

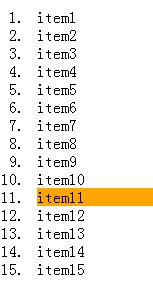
**CSS代码：**

ol > li:nth-last-child(5){

background: orange;

}

**演示结果：**



## CSS3 first-of-type选择器

“:first-of-type”选择器类似于“:first-child”选择器，不同之处就是**指定了元素的类型**,其主要用来定位一个父元素下的某个类型的第一个子元素。

**示例演示：**

通过“:first-of-type”选择器，定位div容器中的第一个p元素（p不一定是容器中的第一个子元素），并设置其背景色为橙色。

**HTML代码：**

<div class="wrapper">

<div>我是一个块元素，我是.wrapper的第一个子元素</div>

<p>我是一个段落元素，我是不是.wrapper的第一个子元素，但是他的第一个段落元素</p>

<p>我是一个段落元素</p>

<div>我是一个块元素</div>

</div>

**CSS代码：**

.wrapper {

width: 500px;

margin: 20px auto;

padding: 10px;

border: 1px solid #ccc;

color: #fff;

}

.wrapper > div {

background: green;

}

.wrapper > p {

background: blue;

}

**/\*我要改变第一个段落的背景为橙色\*/**

.wrapper > p:first-of-type {

background: orange;

}

演示结果：



## CSS3 nth-of-type(n)选择器

“:nth-of-type(n)”选择器和“:nth-child(n)”选择器非常类似，不同的是它只计算父元素中指定的某种类型的子元素。当某个元素中的子元素不单单是同一种类型的子元素时，使用“:nth-of-type(n)”选择器来定位于父元素中某种类型的子元素是非常方便和有用的。在“:nth-of-type(n)”选择器中的“n”和“:nth-child(n)”选择器中的“n”参数也一样，可以是具体的**整数**，也可以是**表达式**，还可以是**关键词**。

**示例演示**

通过“:nth-of-type(2n)”选择器，将容器“div.wrapper”中偶数段数的背景设置为橙色。

**HTML代码：**

<div class="wrapper">

<div>我是一个Div元素</div>

<p>我是一个段落元素</p>

<div>我是一个Div元素</div>

<p>我是一个段落</p>

<div>我是一个Div元素</div>

<p>我是一个段落</p>

<div>我是一个Div元素</div>

<p>我是一个段落</p>

<div>我是一个Div元素</div>

<p>我是一个段落</p>

<div>我是一个Div元素</div>

<p>我是一个段落</p>

<div>我是一个Div元素</div>

<p>我是一个段落</p>

<div>我是一个Div元素</div>

<p>我是一个段落</p>

</div>

**CSS代码：**

.wrapper > p:nth-of-type(2n){

background: orange;

}

**演示结果：**



## CSS3 last-of-type选择器

“:last-of-type”选择器和“:first-of-type”选择器功能是一样的，不同的是他选择是父元素下的某个类型的最后一个子元素。

**示例演示**

通过“:last-of-type”选择器，将容器“div.wrapper”中最后一个段落元素背景设置为橙色

（**提示：**这个段落不是“div.wrapper”容器的最后一个子元素）。

**HTML代码：**

<div class="wrapper">

<p>我是第一个段落</p>

<p>我是第二个段落</p>

<p>我是第三个段落</p>

<div>我是第一个Div元素</div>

<div>我是第二个Div元素</div>

<div>我是第三个Div元素</div>

</div>

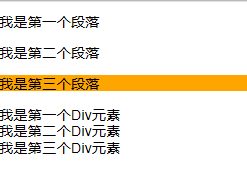
**CSS代码：**

.wrapper > p:last-of-type{

  background: orange;

}

演示结果：



## CSS3 nth-last-of-type(n)选择器

“:nth-last-of-type(n)”选择器和“:nth-of-type(n)”选择器是一样的，选择父元素中指定的某种子元素类型，但它的起始方向是从最后一个子元素开始，而且它的使用方法类似于上节中介绍的“:nth-last-child(n)”选择器一样。

**示例演示**

通过“:nth-last-of-type(n)”选择器将容器“div.wrapper”中的倒数第三个段落背景设置为橙色。

**HTML代码：**

<div class="wrapper">

<p>我是第一个段落</p>

<p>我是第二个段落</p>

<p>我是第三个段落</p>

<p>我是第四个段落</p>

<p>我是第五个段落</p>

<div>我是一个Div元素</div>

<p>我是第六个段落</p>

<p>我是第七个段落</p>

</div>

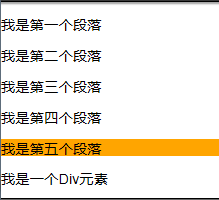
**CSS代码：**

.wrapper > p:nth-last-of-type(3){

background: orange;

}

**演示结果：**



## CSS3 only-child选择器

“:only-child”选择器选择的是父元素中只有一个子元素，而且只有唯一的一个子元素。也就是说，匹配的元素的父元素中仅有一个子元素，而且是一个**唯一的子元素**。

**示例演示**

通过“:only-child”选择器，来控制仅有一个子元素的背景样式，为了更好的理解，我们这个示例通过对比的方式来向大家演示。

**HTML代码：**

<div class="post">

<p>我是一个段落</p>

<p>我是一个段落</p>

</div>

<div class="post">

<p>我是一个段落</p>

</div>

**CSS代码：**

.post p {

background: green;

color: #fff;

padding: 10px;

}

.post p:only-child {

background: orange;

}

**演示结果：**



## CSS3 only-of-type选择器

“:only-of-type”选择器用来选择一个元素是它的父元素的唯一一个相同类型的子元素。这样说或许不太好理解，换一种说法。“:only-of-type”是表示一个元素他有很多个子元素，而其中只有一种类型的子元素是唯一的，使用“:only-of-type”选择器就可以选中这个元素中的唯一一个类型子元素。

**示例演示**

通过“:only-of-type”选择器来修改容器中仅有一个div元素的背景色为橙色。

**HTML代码：**

<div class="wrapper">

<p>我是一个段落</p>

<p>我是一个段落</p>

<p>我是一个段落</p>

<div>我是一个Div元素</div>

</div>

<div class="wrapper">

<div>我是一个Div</div>

<ul>

<li>我是一个列表项</li>

</ul>

<p>我是一个段落</p>

</div>

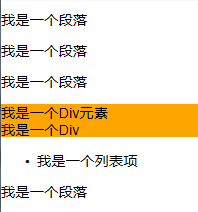
**CSS代码：**

.wrapper > div:only-of-type {

background: orange;

}

**演示结果：**



## CSS3选择器 :enabled选择器

在Web的表单中，有些表单元素有可用（**“:enabled”**）和不可用（“**:disabled**”）状态，比如输入框，密码框，复选框等。在默认情况之下，这些表单元素都处在可用状态。那么我们可以通过伪选择器“**:enabled**”对这些表单元素设置样式。

**示例演示**

通过“:enabled”选择器，修改文本输入框的边框为2像素的红色边框，并设置它的背景为**灰色**。

**HTML代码:**

<form action="#">

  <div>

    <label for="name">Text Input:</label>

    <input type="text" name="name" id="name" placeholder="可用输入框"  />

  </div>

   <div>

    <label for="name">Text Input:</label>

    <input type="text" name="name" id="name" placeholder="禁用输入框"  disabled="disabled" />

  </div>

</form>

**CSS代码：**

div{

  margin: 20px;

}

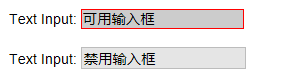
input[type="text"]:enabled {

  background: #ccc;

  border: 2px solid red;

}

**结果演示**



## CSS3选择器 :disabled选择器

“**:disabled**”选择器刚好与“**:enabled**”选择器相反，用来选择不可用表单元素。要正常使用“:disabled”选择器，需要在表单元素的HTML中设置“disabled”属性。

**示例演示**

通过“**:disabled**”选择器，给不可用输入框设置明显的样式。

**HTML代码：**

<form action="#">

<div>

<input type="text" name="name" id="name" placeholder="我是可用输入框" />

</div>

<div>

<input type="text" name="name" id="name" placeholder="我是不可用输入框" disabled />

</div>

</form>

**CSS代码**

form {

margin: 50px;

}

div {

margin-bottom: 20px;

}

input {

background: #fff;

padding: 10px;

border: 1px solid orange;

border-radius: 3px;

}

input[type="text"]:disabled {

background: rgba(0,0,0,.15);

border: 1px solid rgba(0,0,0,.15);

color: rgba(0,0,0,.15);

}

**结果演示：**



## CSS3选择器 :checked选择器

在表单元素中，单选按钮和复选按钮都具有**选中**和**未选中**状态。（大家都知道，要覆写这两个按钮默认样式比较困难）。在CSS3中，我们可以通过状态选择器“:checked”配合其他标签实现**自定义样式**。而“**:checked**”表示的是选中状态。

**示例演示：**

通过“:checked”状态来自定义复选框效果。

**HTML代码**

<form action="#">

<div class="wrapper">

<div class="box">

<input type="checkbox" checked="checked" id="usename" /><span>√</span>

</div>

<lable for="usename">我是选中状态</lable>

</div>

<div class="wrapper">

<div class="box">

<input type="checkbox" id="usepwd" /><span>√</span>

</div>

<label for="usepwd">我是未选中状态</label>

</div>

</form>

**CSS代码：**

form {

border: 1px solid #ccc;

padding: 20px;

width: 300px;

margin: 30px auto;

}

.wrapper {

margin-bottom: 10px;

}

.box {

display: inline-block;

width: 20px;

height: 20px;

margin-right: 10px;

position: relative;

border: 2px solid orange;

vertical-align: middle;

}

.box input {

opacity: 0;

position: absolute;

top:0;

left:0;

}

.box span {

position: absolute;

top: -10px;

right: 3px;

font-size: 30px;

font-weight: bold;

font-family: Arial;

-webkit-transform: rotate(30deg);

transform: rotate(30deg);

color: orange;

}

input[type="checkbox"] + span {

opacity: 0;

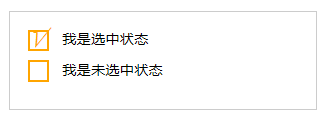
}

input[type="checkbox"]:checked + span {

opacity: 1;

}

结果演示



## CSS3选择器 ::selection选择器

“**::selection**”伪元素是用来匹配**突出显示**的文本(用鼠标选择文本时的文本)。浏览器默认情况下，用鼠标选择网页文本是以“深蓝的背景，白色的字体”显示的，效果如下图所示：

[](http://img.mukewang.com/533e50910001bff808670219.jpg)

从上图中可以看出，用鼠标选中“专注IT、互联网技术”、“纯干货、学以致用”、“没错、这是免费的”这三行文本中，默认显示样式为：蓝色背景、白色文本。

有的时候设计要求,不使用上图那种浏览器默认的突出文本效果，需要一个与众不同的效果，此时“**::selection**”伪元素就非常的实用。不过在**Firefox浏览器**还需要添加前缀。

**示例演示:**

通过“::selection”选择器，将Web中选中的文本背景变成红色，文本变成绿色。

**HTML代码：**

<p>“::selection”伪元素是用来匹配突出显示的文本。浏览器默认情况下，选择网站文本是深蓝的背景，白色的字体，</p>

**CSS代码：**

::-moz-selection {

background: red;

color: green;

}

::selection {

background: red;

color: green;

}

结果演示：

[http://img.mukewang.com/533e52b80001973007260033.jpg](http://img.mukewang.com/533e52b80001973007260033.jpg)

注意：

1、IE9+、Opera、Google Chrome 以及 Safari 中支持 ::selection 选择器。

2、Firefox 支持替代的 ::-moz-selection。

## CSS3选择器 :read-only选择器

“**:read-only**”伪类选择器用来指定处于只读状态元素的样式。简单点理解就是，元素中设置了“**readonly=’readonly’**”

**示例演示**

通过“:read-only”选择器来设置地址文本框的样式。

**HTML代码：**

<form action="#">

<div>

<label for="name">姓名:</label>

<input type="text" name="name" id="name" placeholder="大漠" />

</div>

<div>

<label for="address">地址:</label>

<input type="text" name="address" id="address" placeholder="中国上海" **readonly="readonly"** />

</div>

</form>

**CSS代码：**

form {

width: 300px;

padding: 10px;

border: 1px solid #ccc;

margin: 50px auto;

}

form > div {

margin-bottom: 10px;

}

input[type="text"]{

border: 1px solid orange;

padding: 5px;

background: #fff;

border-radius: 5px;

}

input[type="text"]:-moz-read-only{

border-color: #ccc;

}

input[type="text"]:read-only{

border-color: #ccc;

}

**结果演示**

[](http://img.mukewang.com/5445d7c30001e3fb03870139.jpg)

## CSS3选择器 :read-write选择器

“**:read-write**”选择器刚好与“**:read-only**”选择器**相反**，主要用来指定当元素处于**非只读状态**时的样式。

**示例演示**

使用“**:read-write**”选择器来设置不是只读控件的文本框样式。

**HTML代码：**

<form action="#">

<div>

<label for="name">姓名:</label>

<input type="text" name="name" id="name" placeholder="大漠" />

</div>

<div>

<label for="address">地址:</label>

<input type="text" name="address" id="address" placeholder="中国上海" readonly="readonly" />

</div>

</form>

**CSS代码：**

form {

width: 300px;

padding: 10px;

border: 1px solid #ccc;

margin: 50px auto;

}

form > div {

margin-bottom: 10px;

}

input[type="text"]{

border: 1px solid orange;

padding: 5px;

background: #fff;

border-radius: 5px;

}

input[type="text"]:-moz-read-only{

border-color: #ccc;

}

input[type="text"]:read-only{

border-color: #ccc;

}

input[type="text"]:-moz-read-write{

border-color: #f36;

}

input[type="text"]:read-write{

border-color: #f36;

}

**结果演示：**

[](http://img.mukewang.com/5445d97b0001665203980142.jpg)

## CSS3选择器 ::before和::after

::before和::after这两个主要用来给元素的前面或后面插入内容，这两个常和"content"配合使用，使用的场景最多的就是清除浮动。

.clearfix::before,

.clearfix::after {

content: ".";

display: block;

height: 0;

visibility: hidden;

}

.clearfix:after {clear: both;}

.clearfix {zoom: 1;}

当然可以利用他们制作出其他更好的效果，比如右侧中的阴影效果，也是通过这个来实现的。

关键代码分析：

.effect::before, .effect::after{

    content:"";

    position:absolute;

    z-index:-1;

    -webkit-box-shadow:0 0 20px rgba(0,0,0,0.8);

    -moz-box-shadow:0 0 20px rgba(0,0,0,0.8);

    box-shadow:0 0 20px rgba(0,0,0,0.8);

    top:50%;

    bottom:0;

    left:10px;

    right:10px;

    -moz-border-radius:100px / 10px;

    border-radius:100px / 10px;

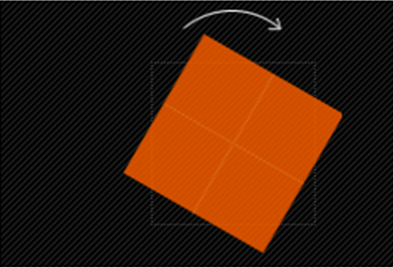
}

上面代码作用在class名叫.effect上的div的前（before）后(after)都添加一个空元素，然后为这两个空元素添加阴影特效。

想看这个知识点深入讲解的小伙伴请观看[《css3实现图片阴影效果》](http://www.imooc.com/learn/240)中的[第1-6小节](http://www.imooc.com/video/5865)。

## CSS3变形--旋转 rotate()

**旋转rotate()函数**通过指定的角度参数使元素相对原点进行旋转。它主要在二维空间内进行操作，设置一个角度值，用来指定旋转的幅度。如果这个值为**正值**，元素相对原点中心**顺时针**旋转；如果这个值为**负值**，元素相对原点中心**逆时针**旋转。如下图所示：



**HTML代码：**

<div class="wrapper">

<div></div>

</div>

**CSS代码：**

.wrapper {

width: 200px;

height: 200px;

border: 1px dotted red;

margin: 100px auto;

}

.wrapper div {

width: 200px;

height: 200px;

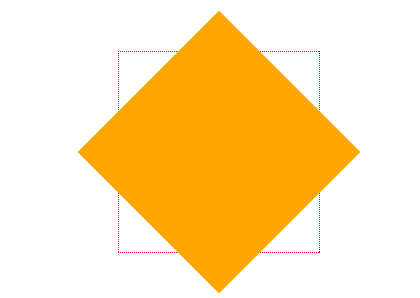
background: orange;

-webkit-transform: rotate(45deg);

transform: rotate(45deg);

}

**演示结果**

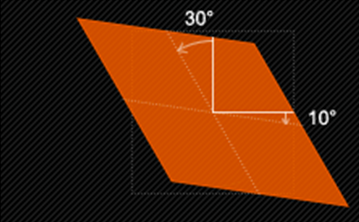
****

## CSS3中的变形--扭曲 skew()

扭曲skew()函数能够让元素**倾斜显示**。它可以将一个对象以其中心位置围绕着**X轴**和**Y轴**按照一定的角度倾斜。这与rotate()函数的旋转不同，rotate()函数只是旋转，而不会改变元素的形状。skew()函数不会旋转，而只会改变元素的形状。

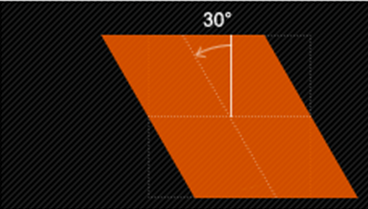
**Skew()具有三种情况：**

**1、skew(x,y)使元素在水平和垂直方向同时扭曲**（X轴和Y轴同时按一定的角度值进行扭曲变形）；

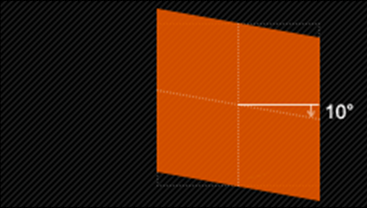


第一个参数对应X轴，第二个参数对应Y轴。如果第二个参数未提供，则值为0，也就是Y轴方向上无斜切。

**2、skewX(x)仅使元素在水平方向扭曲变形**（X轴扭曲变形）；



**3、skewY(y)仅使元素在垂直方向扭曲变形**（Y轴扭曲变形）



**示例演示：**

通过skew（）函数将长方形变成平行四边形。

**HTML代码：**

<div class="wrapper">

<div>我变成平形四边形</div>

</div>

**CSS代码：**

.wrapper {

width: 300px;

height: 100px;

border: 2px dotted red;

margin: 30px auto;

}

.wrapper div {

width: 300px;

height: 100px;

line-height: 100px;

text-align: center;

color: #fff;

background: orange;

-webkit-transform: skew(45deg);

-moz-transform:skew(45deg)

transform:skew(45deg);

}

**演示结果**

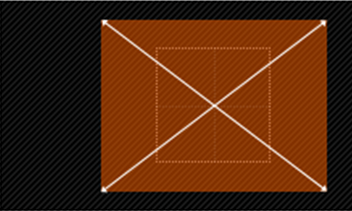
****

## CSS3中的变形--缩放 scale()

**缩放 scale()函数**让元素根据中心原点对对象进行缩放。

缩放 scale 具有三种情况：

**1、 scale(X,Y)使元素水平方向和垂直方向同时缩放**（也就是X轴和Y轴同时缩放）



例如：

div:hover {

-webkit-transform: scale(1.5,0.5);

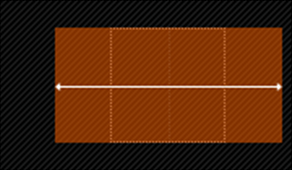
  -moz-transform:scale(1.5,0.5)

transform: scale(1.5,0.5);

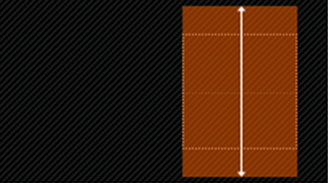
}

注意：Y是一个可选参数，如果没有设置Y值，则表示X，Y两个方向的缩放倍数是一样的。

**2、scaleX(x)元素仅水平方向缩放**（X轴缩放）



**3、scaleY(y)元素仅垂直方向缩放**（Y轴缩放）



**HTML代码：**

<div class="wrapper">

<div>我将放大1.5倍</div>

</div>

**CSS代码：**

.wrapper {

width: 200px;

height: 200px;

border:2px dashed red;

margin: 100px auto;

}

.wrapper div {

width: 200px;

height: 200px;

line-height: 200px;

background: orange;

text-align: center;

color: #fff;

}

.wrapper div:hover {

opacity: .5;

-webkit-transform: scale(1.5);

  -moz-transform:scale(1.5)

transform: scale(1.5);

}

**演示结果**



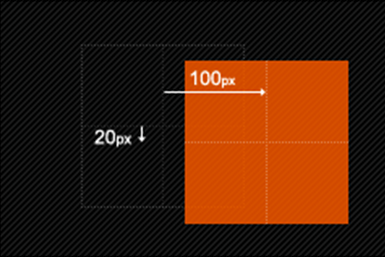
**注意：** **scale()**的取值默认的值为1，当值设置为**0.01**到**0.99**之间的任何值，作用使一个元素缩小；而任何大于或等于**1.01**的值，作用是让元素放大。

## CSS3中的变形--位移 translate()

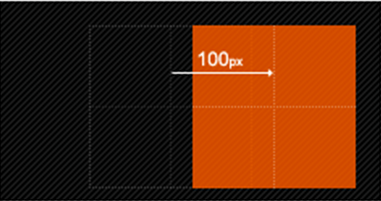
**translate()函数**可以将元素向指定的方向移动，类似于position中的**relative**。或以简单的理解为，使用translate()函数，可以把元素从原来的位置移动，而不影响在X、Y轴上的任何Web组件。

**translate我们分为三种情况：**

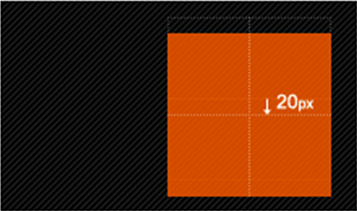
**1、translate(x,y)水平方向和垂直方向同时移动**（也就是X轴和Y轴同时移动）



**2、translateX(x)仅水平方向移动**（X轴移动）



**3、translateY(Y)仅垂直方向移动**（Y轴移动）



**实例演示：**通过translate()函数将元素向Y轴下方移动50px,X轴右方移动100px。

**HTML代码：**

<div class="wrapper">

<div>我向右向下移动</div>

</div>

**CSS代码：**

.wrapper {

width: 200px;

height: 200px;

border: 2px dotted red;

margin: 20px auto;

}

.wrapper div {

width: 200px;

height: 200px;

line-height: 200px;

text-align: center;

background: orange;

color: #fff;

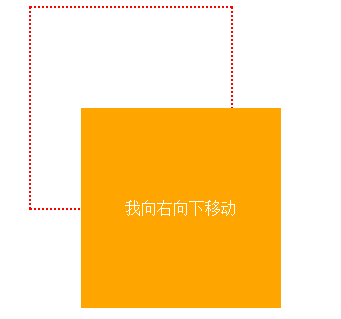
-webkit-transform: translate(50px,100px);

  -moz-transform:translate(50px,100px);

transform: translate(50px,100px);

}

**演示结果**

****

## CSS3中的变形--矩阵 matrix()

**matrix()**是一个含六个值的(a,b,c,d,e,f)变换矩阵，用来指定一个2D变换，相当于直接应用一个[a b c d e f]变换矩阵。就是基于水平方向（X轴）和垂直方向（Y轴）重新定位元素,此属性值使用涉及到数学中的矩阵，我在这里只是简单的说一下CSS3中的transform有这么一个属性值，如果需要深入了解，需要对数学矩阵有一定的知识。

**示例演示：**通过matrix()函数来模拟transform中translate()位移的效果。  
**HTML代码：**

<div class="wrapper">

<div></div>

</div>

**CSS代码：**

.wrapper {

width: 300px;

height: 200px;

border: 2px dotted red;

margin: 40px auto;

}

.wrapper div {

width:300px;

height: 200px;

background: orange;

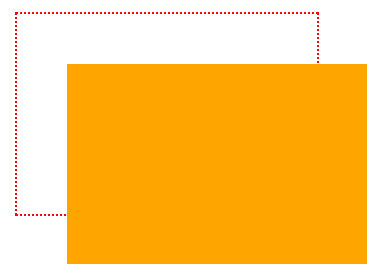
-webkit-transform: matrix(1,0,0,1,50,50);

-moz-transform:matrix(1,0,0,1,50,50);

transform: matrix(1,0,0,1,50,50);

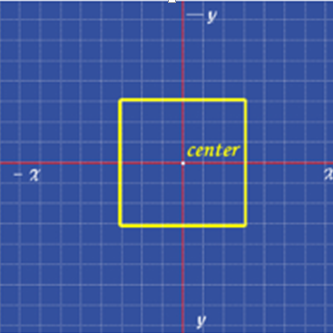
}

**演示结果：**

****

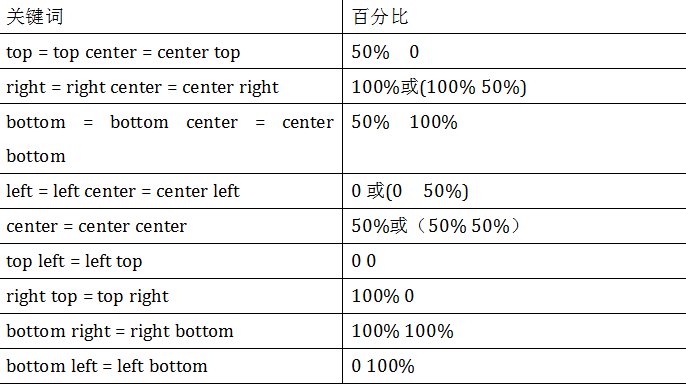
## CSS3中的变形--原点 transform-origin

任何一个元素都有一个中心点，默认情况之下，其中心点是居于元素X轴和Y轴的50%处。如下图所示：



在没有重置transform-origin改变元素原点位置的情况下，CSS变形进行的旋转、位移、缩放，扭曲等操作都是**以元素自己中心位置进行变形**。但很多时候，我们可以通过transform-origin来对元素进行原点位置改变，使元素原点不在元素的中心位置，以达到需要的原点位置。

transform-origin取值和元素设置背景中的background-position取值类似，如下表所示：



**示例展示：**

通过transform-origin改变元素原点到左上角，然后进行顺时旋转45度。

**HTML代码：**

<div class="wrapper">

<div>原点在默认位置处</div>

</div>

<div class="wrapper transform-origin">

<div>原点重置到左上角</div>

</div>

**CSS代码：**

.wrapper {

width: 300px;

height: 300px;

float: left;

margin: 100px;

border: 2px dotted red;

line-height: 300px;

text-align: center;

}

.wrapper div {

background: orange;

-webkit-transform: rotate(45deg);

transform: rotate(45deg);

}

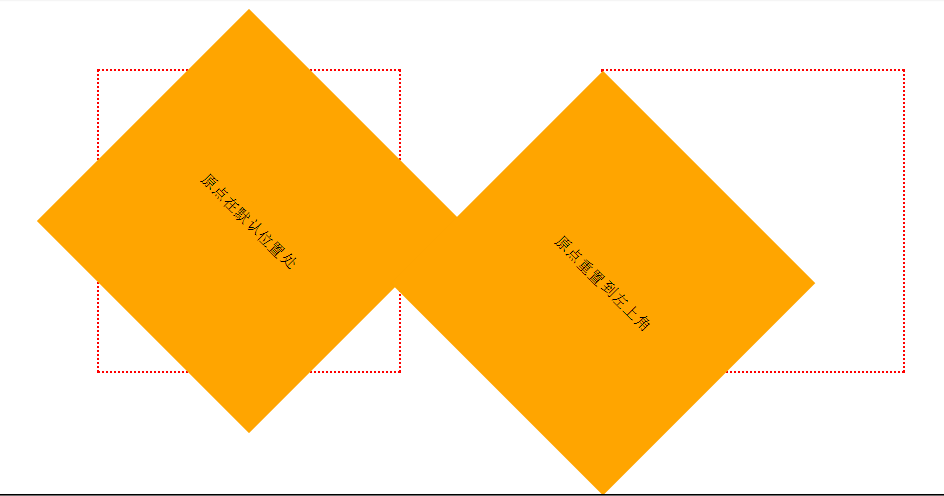
.transform-origin div {

-webkit-transform-origin: left top;

transform-origin: left top;

}

**演示结果：**

****

## CSS3中的动画--过渡属性 transition-property

早期在Web中要实现动画效果，都是依赖于JavaScript或Flash来完成。但在CSS3中新增加了一个新的模块transition，它可以通过一些简单的CSS事件来触发元素的外观变化，让效果显得更加细腻。简单点说，**就是通过鼠标的单击、获得焦点，被点击或对元素任何改变中触发，并平滑地以动画效果改变CSS的属性值。**

在CSS中创建简单的过渡效果可以从以下几个步骤来实现：

第一，在默认样式中声明元素的初始状态样式；

第二，声明过渡元素最终状态样式，比如悬浮状态；

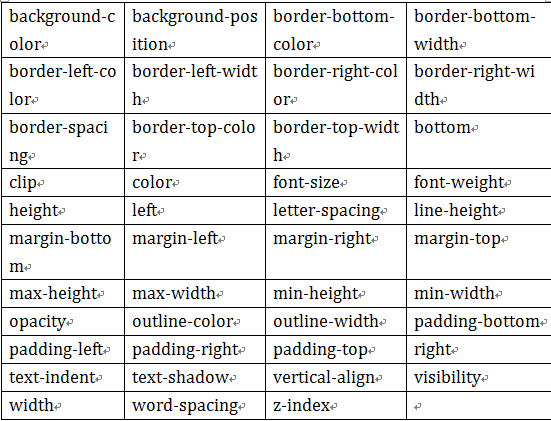
第三，在默认样式中通过添加过渡函数，添加一些不同的样式。

CSS3的过度transition属性是一个复合属性，主要包括以下几个子属性：

* transition-property:指定过渡或动态模拟的CSS属性
* transition-duration:指定完成过渡所需的时间
* transition-timing-function:指定过渡函数
* transition-delay:指定开始出现的延迟时间

先来看**transition-property属性**

**transition-property**用来指定**过渡动画**的CSS属性名称，而这个过渡属性只有具备一个**中点值的属性**（需要产生动画的属性）才能具备过渡效果，其对应具有过渡的CSS属性主要有：

[](http://img.mukewang.com/5344eca300010a8005510421.jpg)

**HTML:**

<div></div>

**CSS:**

div {

width: 200px;

height: 200px;

background-color:red;

margin: 20px auto;

-webkit-transition: background-color .5s ease .1s;

transition: background-color .5s ease .1s;

}

div:hover {

background-color: orange;

}

**演示结果:**

**鼠标移入**

****

**鼠标移出**

****

**特别注意**：当“transition-property”属性设置为**all**时，表示的是所有中点值的属性。

用一个简单的例子来说明这个问题：

假设你的初始状态设置了样式“width”,“height”,“background”,当你在终始状态都改变了这三个属性，那么**all**代表的就是“width”、“height”和“background”。如果你的终始状态只改变了“width”和“height”时，那么**all**代表的就是“width”和“height”。

## CSS3中的动画--过渡所需时间 transition-duration

**transition-duration**属性主要用来设置一个属性过渡到另一个属性所需的时间，也就是从旧属性过渡到新属性花费的时间长度，俗称**持续时间**。

**案例演示：**

在鼠标悬停（hover）状态下，让容器从直角慢慢过渡到圆角，并让整个动画持续0.5s。

HTML:

<div></div>

CSS:

div {

width: 300px;

height: 200px;

background-color: orange;

margin: 20px auto;

-webkit-transition-property: -webkit-border-radius;

transition-property: border-radius;

-webkit-transition-duration: .5s;

transition-duration: .5s;

-webkit-transition-timing-function: ease-out;

transition-timing-function: ease-out;

-webkit-transition-delay: .2s;

transition-delay: .2s;

}

div:hover {

border-radius: 20px;

}

演示结果：

**鼠标移入**

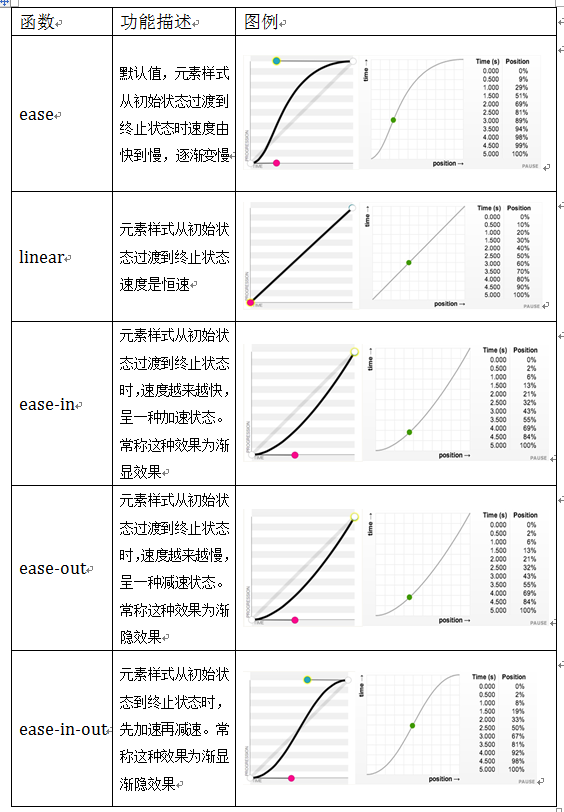


**鼠标移出**



## CSS3中的动画--过渡函数 transition-timing-function

transition-timing-function属性指的是过渡的“缓动函数”。主要用来指定浏览器的过渡速度，以及过渡期间的操作进展情况，其中要包括以下几种函数：

[](http://img.mukewang.com/5344f1320001481905640812.jpg)

(单击图片可放大)

**案例展示：**

在hover状态下，让容器从一个正方形慢慢过渡到一个圆形，而整个过渡是先加速再减速，也就是运用ease-in-out函数。

**HTML代码:**

<div></div>

**CSS代码:**

div {

width: 200px;

height: 200px;

background: red;

margin: 20px auto;

-webkit-transition-property: -webkit-border-radius;

transition-property: border-radius;

-webkit-transition-duration: .5s;

transition-duration: .5s;

-webkit-transition-timing-function: ease-in-out;

transition-timing-function: ease-in-out;

-webkit-transition-delay: .2s;

transition-delay: .2s;

}

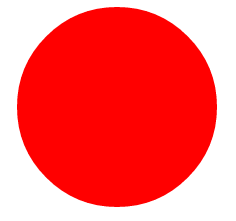
div:hover {

border-radius: 100%;

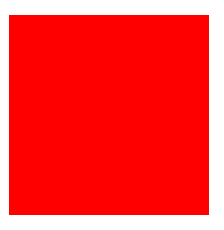
}

演示结果

**鼠标移入：**



**鼠标移出：**



## CSS3中的动画--过渡延迟时间 transition-delay

**transition-delay属性**和**transition-duration属性**极其类似，不同的是transition-duration是用来设置过渡动画的持续时间，而transition-delay主要用来指定**一个动画开始执行的时间**，也就是说当改变元素属性值后多长时间开始执行。

有时我们想改变两个或者多个css属性的**transition效果**时，只要把几个transition的声明串在一起，用**逗号**（“，”）隔开，然后各自可以有各自不同的**延续时间**和其**时间的速率变换方式**。但需要值得注意的一点：第一个时间的值为 transition-duration，第二个为transition-delay。

例如：**a{ transition: background 0.8s ease-in 0.3,color 0.6s ease-out 0.3;}**

**示例演示：**

通过transition属性将一个200px \*200px的橙色容器，在鼠标悬浮状态时，过渡到一个300px \* 300px的红色容器。而且整个过渡0.1s后触发，并且整个过渡持续0.28s。

**HTML代码:**

<div class="wrapper">

<div>鼠标放到我的身上来</div>

</div>

**CSS代码:**

.wrapper {

width: 400px;

height: 400px;

margin: 20px auto;

border: 2px dotted red;

}

.wrapper div {

width: 200px;

height: 200px;

background-color: orange;

-webkit-transition: all .28s ease-in .1s;

transition: all .28s ease-in .1s;

}

.wrapper div:hover {

width: 300px;

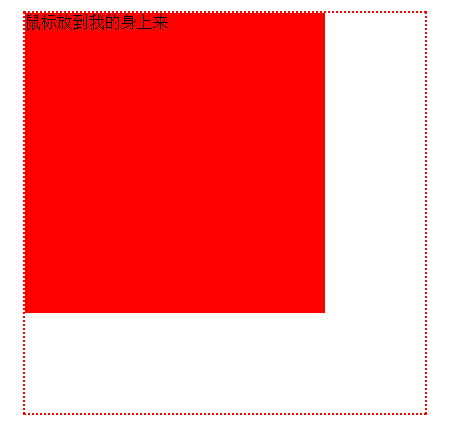
height: 300px;

background-color: red;

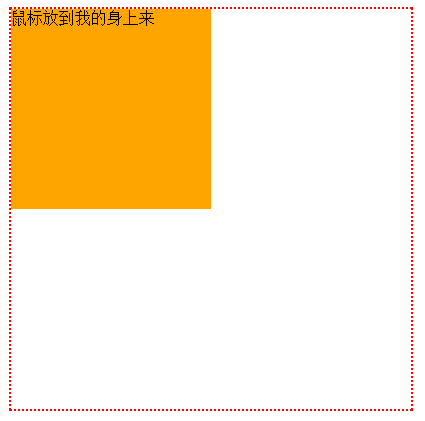
}

演示结果

**鼠标移入：**



**鼠标移出：**



## CSS3 Keyframes介绍

Keyframes被称为**关键帧**，其类似于Flash中的关键帧。在CSS3中其主要以“@keyframes”开头，后面紧跟着是动画名称加上一对花括号“{…}”，括号中就是一些不同时间段样式规则。

@keyframes changecolor{

0%{

background: red;

}

100%{

background: green;

}

}

在一个“@keyframes”中的样式规则可以由多个百分比构成的，如在“0%”到“100%”之间创建更多个百分比，分别给每个百分比中给需要有动画效果的元素加上不同的样式，从而达到一种在不断变化的效果。

**经验与技巧：**在@keyframes中定义动画名称时，其中0%和100%还可以使用关键词**from**和**to**来代表，其中0%对应的是from，100%对应的是to。

浏览器的支持情况：

[](http://img.mukewang.com/534fa4450001513c05120121.jpg)**Chrome** 和 **Safari** 需要前缀 **-webkit-**；**Foxfire** 需要前缀 **-moz-**。

**案例演示**

通过“@keyframes”声明一个名叫“wobble”的动画，从“0%”开始到“100%”结束，同时还经历了一个“40%”和“60%”两个过程。“wobble”动画在“0%”时元素定位到left为100px，背景色为green，然后在“40%”时元素过渡到left为150px,背景色为orange,接着在“60%”时元素过渡到left为75px，背景色为blue，最后“100%”时结束动画，元素又回到起点left为100px处，背景色变为red。

HTML:

<div>鼠标放到我身上</div>

CSS:

@keyframes wobble {

0% {

margin-left: 100px;

background:green;

}

40% {

margin-left:150px;

background:orange;

}

60% {

margin-left: 75px;

background: blue;

}

100% {

margin-left: 100px;

background: red;

}

}

div {

width: 100px;

height: 100px;

background:red;

color: #fff;

}

div:hover{

animation: wobble 5s ease .1s;

}

## CSS3中调用动画

**animation-name属性**主要是用来调用**@keyframes** 定义好的动画。需要特别注意: animation-name 调用的**动画名**需要和“@keyframes”**定义的动画名称**完全一致（**区分大小写**），如果不一致将不具有任何动画效果。

语法：

animation-name: none | IDENT[,none|DENT]\*;

1、IDENT是由 **@keyframes** 创建的动画名，上面已经讲过了（animation-name 调用的动画名需要和“@keyframes”定义的动画名称完全一致）；

2、**none**为默认值，当值为 none 时，将没有任何动画效果,这可以用于覆盖任何动画。

注意：需要在 Chrome 和 Safari 上面的基础上加上-webkit-前缀，Firefox加上-moz-。

## CSS3中设置动画播放时间

animation-duration主要用来设置CSS3动画播放时间，其使用方法和transition-duration类似，是用来指定元素播放动画所持续的时间长，也就是**完成从0%到100%一次动画所需时间**。单位：S秒

**语法规则**

animation-duration: <time>[,<time>]\*

取值<time>为数值，单位为秒，其默认值为“**0**”，这意味着动画周期为“0”，也就是没有动画效果（如果值为负值会被视为“0”）。

**案例演示：**

制作一个矩形变成圆形的动画，整个动画播放时间持续了10s钟。

HTML:

<div>Hover Me</div>

CSS:

@keyframes toradius{

from {

border-radius: 0;

}

to {

border-radius: 100%;

}

}

div {

width: 200px;

height: 200px;

line-height: 200px;

text-align: center;

color: #fff;

background: green;

margin: 20px auto;

}

div:hover {

animation-name: toradius;

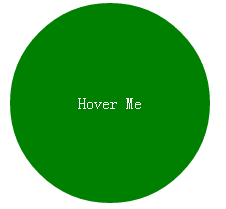
animation-duration: 10s;

animation-timing-function: ease-in-out;

animation-delay: .1s;

}

鼠标移入



鼠标移出



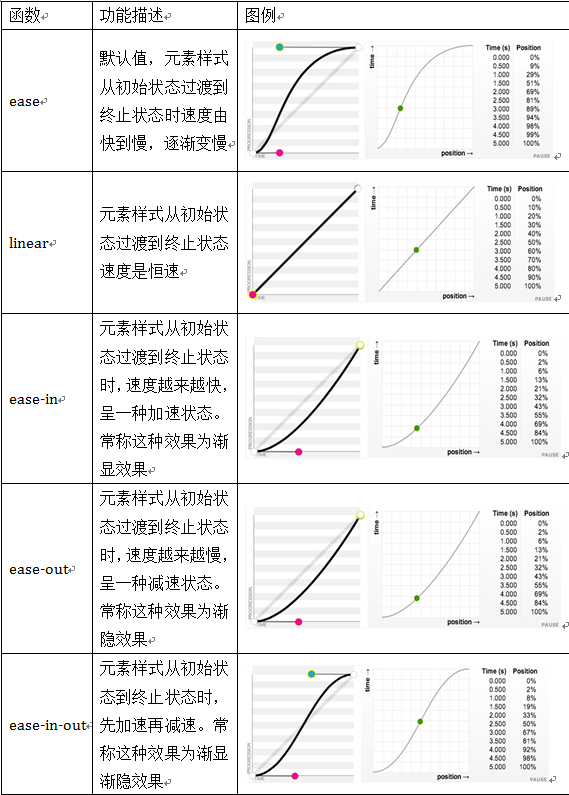
## CSS3中设置动画播放方式

animation-timing-function属性主要用来设**置动画播放方式**。主要让元素根据时间的推进来改变属性值的变换速率，简单点说就是动画的播放方式。

**语法规则：**

animation-timing-function:ease | linear | ease-in | ease-out | ease-in-out | cubic-bezier(<number>, <number>, <number>, <number>) [, ease | linear | ease-in | ease-out | ease-in-out | cubic-bezier(<number>, <number>, <number>, <number>)]\*

它和transition中的transition-timing-function一样，具有以下几种变换方式：ease,ease-in,ease-in-out,ease-out,linear和cubic-bezier。对应功如下：

[](http://img.mukewang.com/534b8bb100016e9f05690795.jpg)

在调用move动画播放中，让元素样式从初始状态到终止状态时，先加速再减速，也就是渐显渐隐效果。

## CSS3中设置动画开始播放的时间

animation-delay属性用来定义动画开始播放的时间，用来触发动画播放的时间点。和transition-delay属性一样，用于定义在浏览器开始执行动画之前等待的时间。

**语法规则：**

animation-delay:<time>[,<time>]\*

**案例演示：**

浏览器渲染样式之后2S之后触发move动画。

**HTML:**

<div><span></span></div>

**CSS:**

@keyframes move {

0%{

transform: translate(0);

}

15%{

transform: translate(100px,180px);

}

30%{

transform: translate(150px,0);

}

45%{

transform: translate(250px,180px);

}

60%{

transform:translate(300px,0);

}

75%{

transform: translate(450px,180px);

}

100%{

transfrom: translate(480px,0);

}

}

div {

width: 500px;

height: 200px;

border: 1px solid red;

margin: 20px auto;

}

div span {

display: inline-block;

width: 20px;

height: 20px;

background: green;

border-radius: 100%;

animation-name:move;

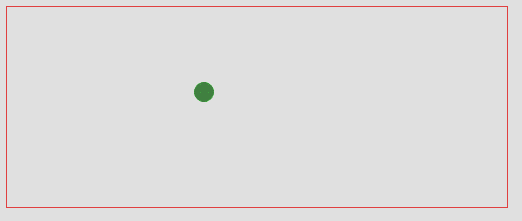
animation-duration: 10s;

animation-timing-function:ease;

**animation-delay:2s;**

animation-iteration-count:infinite;

}

结果展示：[](http://img.mukewang.com/5360bd2c00014b8505220221.jpg)

## CSS3中设置动画播放次数

animation-iteration-count属性主要用来定义动画的**播放次数。**

**语法规则：**

animation-iteration-count: infinite | <number> [, infinite | <number>]\*

1、其值通常为**整数**，但也可以使用带有小数的数字，其默认值为1，这意味着动画将从开始到结束只播放一次。

2、如果取值为**infinite**，动画将会无限次的播放。

**举例：**

通过animation-iteration-count属性让动画move只播放5次，代码设置为：

**animation-iteration-count:5;**

**注意：Chrome或Safari浏览器，需要加入-webkit-前缀！**

## CSS3中设置动画播放方向

animation-direction属性主要用来设置**动画播放方向，**其语法规则如下：

animation-direction:normal | alternate [, normal | alternate]\*

其主要有两个值：**normal**、**alternate**

1、**normal**是默认值，如果设置为normal时，动画的每次循环都是向前播放；

2、另一个值是**alternate**，他的作用是，动画播放在**第偶数次**向前播放，**第奇数次**向反方向播放。

例如：通过animation-direction属性，将move动画播放动画方向设置为alternate，代码为：

**animation-direction:alternate;**

**注意：Chrome或Safari浏览器，需要加入-webkit-前缀！**

## CSS3中设置动画的播放状态

animation-play-state属性主要用来控制元素动画的**播放状态**。

**参数：**

其主要有两个值：**running**和**paused**。

其中running是其默认值，主要作用就是类似于音乐播放器一样，可以通过paused将正在播放的动画停下来，也可以通过running将暂停的动画重新播放，这里的重新播放不一定是从元素动画的开始播放，而是从暂停的那个位置开始播放。另外如果暂停了动画的播放，元素的样式将回到最原始设置状态。

例如，页面加载时，动画不播放。代码如下：

animation-play-state:paused;

**CSS3中设置动画时间外属性**

animation-fill-mode属性定义在动画开始之前和结束之后发生的操作。主要具有四个属性值：**none、forwards、backwords**和**both**。其四个属性值对应效果如下：

|  |  |
| --- | --- |
| **属性值** | **效果** |
| none | 默认值，表示动画将按预期进行和结束，在动画完成其最后一帧时，动画会反转到初始帧处 |
| forwards | 表示动画在结束后继续应用最后的关键帧的位置 |
| backwards | 会在向元素应用动画样式时迅速应用动画的初始帧 |
| both | 元素动画同时具有forwards和backwards效果 |

在默认情况之下，动画不会影响它的关键帧之外的属性，使用animation-fill-mode属性可以修改动画的默认行为。简单的说就是告诉动画在第一关键帧上等待动画开始，或者在动画结束时停在最后一个关键帧上而不回到动画的第一帧上。或者同时具有这两个效果。

例如：让动画停在最一帧处。代码如下：

animation-fill-mode:forwards;

**CSS3 多列布局——Columns**

为了能在Web页面中方便实现类似报纸、杂志那种多列排版的布局，W3C特意给CSS3增加了一个**多列布局模块**（CSS Multi Column Layout Module）。它主要应用在文本的**多列布局**方面，这种布局在报纸和杂志上都使用了几十年了，但要在Web页面上实现这样的效果还是有相当大的难度，庆幸的是，CSS3的多列布局可以轻松实现。接下来咱们一起学习多列布局相关的知识。

**语法：**

columns：<column-width> || <column-count>

多列布局columns属性参数主要就两个属性参数：列宽和列数。

|  |  |
| --- | --- |
| **参数** | **参数说明** |
| <column-width> | 主要用来定义多列中每列的宽度 |
| <column-count> | 主要用来定义多列中的列数 |

举例：要显示2栏显示，每栏宽度为200px，代码为：

**columns: 200px 2;**

到目前为止大部分主流浏览器都对其支持：

[](http://img.mukewang.com/5360c43c00010efd05080132.jpg)

**CSS3 多列布局——column-width**

column-width的使用和CSS中的width属性一样，不过不同的是，column-width属性在定义元素列宽的时候，既可以单独使用，也可以和多列属性中其他属性配合使用。其基本语法如下所示 ；

column-width: auto | <length>

取值说明

|  |  |
| --- | --- |
| **属性值** | **说明** |
| auto | 如果column-width设置值为auto或者没有显式的设置值时，元素多列的列宽将由其他属性来决定，比如前面的示例就是由列数column-count来决定。 |
| <length> | 使用固定值来设置元素列的宽度，其主要是由数值和长度单位组成，不过其值只能是正值，不能为负值。 |

**CSS3 多列布局——column-count**

column-count属性主要用来给元素指定想要的列数和允许的最大列数。其语法规则：

column-count：auto | <integer>

**取值说明：**

|  |  |
| --- | --- |
| **属性值** | **属性值说明** |
| auto | 此值为column-count的默认值，表示元素只有一列，其主要依靠浏览器计算自动设置。 |
| <integer> | 此值为正整数值，主要用来定义元素的列数，取值为大于0的整数，负值无效。 |

 例如：将列分成四列显示，代码如下：

column-count:4;

**CSS3 列间距column-gap**

column-gap主要用来设置列与列之间的**间距**，其语法规则如下：

column-gap: normal || <length>

**取值说明**

|  |  |
| --- | --- |
| **属性值** | **属性值说明** |
| normal | 默认值，默值为1em（如果你的字号是px，其默认值为你的font-size值）。 |
| <length> | 此值用来设置列与列之间的距离，其可以使用px,em单位的任何整数值，但不能是负值。 |

例如：将内容分三列显列，列与列之间的间距为2em，实现代码为：

column-count: 3;

column-gap: 2em;

**CSS3 列表边框column-rule**

column-rule主要是用来定义列与列之间的**边框宽度、边框样式**和**边框颜色**。简单点说，就有点类似于常用的border属性。但column-rule是不占用任何空间位置的，在列与列之间改变其宽度不会改变任何列的位置。

**语法规则：**

column-rule:<column-rule-width>|<column-rule-style>|<column-rule-color>

**取值说明：**

|  |  |
| --- | --- |
| **属性值** | **属性值说明** |
| column-rule-width | 类似于border-width属性，主要用来定义列边框的宽度，其默认值为“medium”，column-rule-width属性接受任意浮点数，但不接收负值。但也像border-width属性一样，可以使用关键词：medium、thick和thin。 |
| column-rule-style | 类似于border-style属性，主要用来定义列边框样式，其默认值为“none”。column-rule-style属性值与border-style属值相同，包括none、hidden、dotted、dashed、solid、double、groove、ridge、inset、outset。 |
| column-rule-color | 类似于border-color属性，主要用来定义列边框颜色，其默认值为前景色color的值，使用时相当于border-color。column-rule-color接受所有的颜色。如果不希望显示颜色，也可以将其设置为transparent(透明色) |

例如：为了能有效区分栏目列之间的关系，可以为其设置一个列边框，代码为：

column-rule: 2px dotted green;

**CSS3 跨列设置column-span**

column-span主要用来定义一个分列元素中的子元素能跨列多少。column-width、column-count等属性能让一元素分成多列，不管里面元素如何排放顺序，他们都是从左向右的放置内容，但有时我们需要基中一段内容或一个标题不进行分列，也就是横跨所有列，此时column-span就可以轻松实现，此属性的语法如下。

column-span: none | all

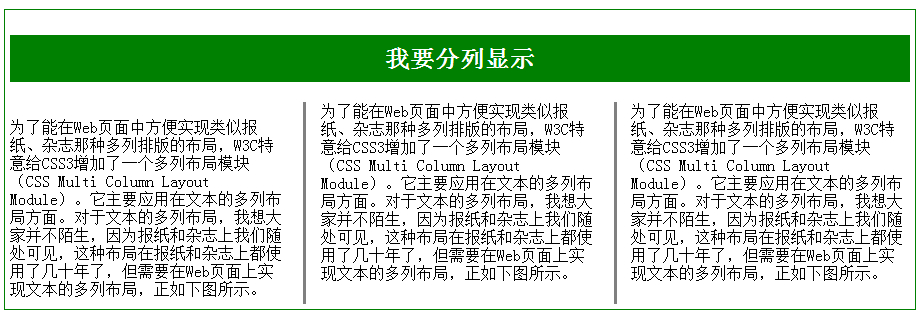
**取值说明**

|  |  |
| --- | --- |
| **属性值** | **属性值说明** |
| none | 此值为column-span的默认值，表示不跨越任何列。 |
| all | 这个值跟none值刚好相反，表示的是元素跨越所有列，并定位在列的Ｚ轴之上。 |

例如：将第一个标题跨越所有列，代码：

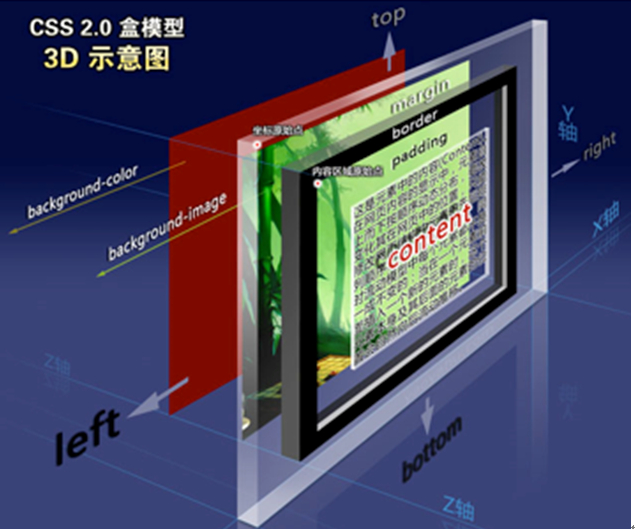
column-span:all;

效果如下：

[](http://img.mukewang.com/5365d594000190af09240316.jpg)

## CSS3 盒子模型

CSS中有一种基础设计模式叫盒模型，盒模型定义了Web页面中的元素中如何来解析。CSS中每一个元素都是一个盒模型，包括html和body标签元素。在盒模型中主要包括width、height、border、background、padding和margin这些属性，而且他们之间的层次关系可以相互影响，来看一张盒模型的3D展示图：

[](http://img.mukewang.com/5365d7b10001e8d506350529.jpg)

（单击可放大）

从图中可以看出padding属性和content属性层叠background-image属性，层叠background-color属性，这个是存在的，它们四者之间构成了Ｚ轴（垂直屏幕的坐标）多重层叠关系。但是border属性与margin属性、padding属性三者之间应该是平面上的并级关系，并不能构成Ｚ轴的层叠关系。

**box-sizing：**

在CSS中盒模型被分为两种，第一种是w3c的标准模型，另一种是IE的传统模型，它们相同之处都是对元素计算尺寸的模型，具体说不是对元素的width、height、padding和border以及元素实际尺寸的计算关系，它们不同之处是两者的计算方法不一致，原则上来说盒模型是分得很细的，这里所看到的主要是外盒模型和内盒模型，如下面计算公式所示：

1. **W3C标准盒模型**

外盒尺寸计算（元素空间尺寸）

element空间高度＝内容高度＋内距＋边框＋外距

element空间宽度＝内容宽度＋内距＋边框＋外距

内盒尺寸计算（元素大小）

element高度＝内容高度＋内距＋边框（height为内容高度）

element宽度＝内容宽度＋内距＋边框（width为内容宽度）

**2.IE传统下盒模型（IE6以下，不包含IE6版本或”QuirksMode下IE5.5+”）**

外盒尺寸计算（元素空间尺寸）

element空间高度＝内容高度＋外距（height包含了元素内容宽度、边框、内距）

element宽间宽度＝内容宽度＋外距（width包含了元素内容宽度、边框、内距）

内盒尺寸计算（元素大小）

element高度＝内容高度（height包含了元素内容宽度、边框、内距）

element宽度＝内容宽度（width包含了元素内容宽度、边框、内距）

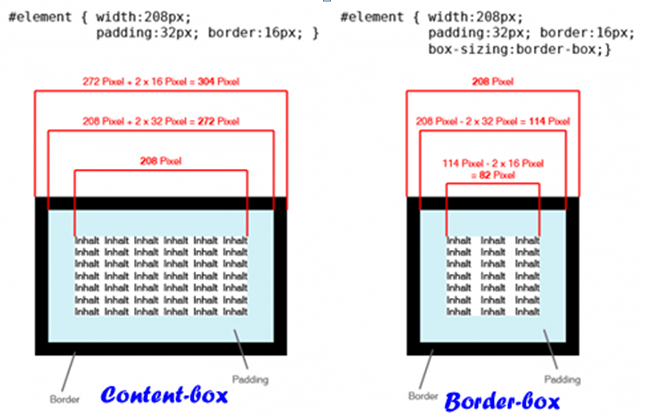
在CSS3中新增加了**box-sizing**属性，能够事先定义盒模型的尺寸解析方式，其语法规则如下：

box-sizing: content-box | border-box | inherit

**取值说明**

|  |  |
| --- | --- |
| **属性值** | **属性值说明** |
| content-box | 默认值，其让元素维持W3C的标准盒模型，也就是说元素的宽度和高度（width/height）等于元素边框宽度（border）加上元素内距（padding）加上元素内容宽度或高度（content width/ height），也就是element width/height = border + padding + content width / height |
| border-box | 重新定义CSS2.1中盒模型组成的模式，让元素维持IE传统的盒模型（IE6以下版本和IE6-7怪异模式），也就是说元素的宽度或高度等于元素内容的宽度或高度。从上面盒模型介绍可知，这里的内容宽度或高度包含了元素的border、padding、内容的宽度或高度（此处的内容宽度或高度＝盒子的宽度或高度—边框—内距）。 |
| inherit | 使元素继承父元素的盒模型模式 |

其中最为关键的是box-sizing中content-box和border-box两者的区别，他们之间的区别可以通过下图来展示，其对盒模型的不同解析：

[](http://img.mukewang.com/5365d98000018fa606460416.jpg)

（单击可放大）

## CSS3 伸缩布局（一）

CSS3引入了一种新的布局模式——**Flexbox布局**，即**伸缩布局盒模型**（Flexible Box），用来提供一个更加有效的方式制定、调整和分布一个容器里项目布局，即使它们的大小是未知或者动态的，这里简称为**Flex**。

Flexbox布局常用于设计比较复杂的页面，可以轻松的实现屏幕和浏览器窗口大小发生变化时保持元素的相对位置和大小不变，同时减少了依赖于浮动布局实现元素位置的定义以及重置元素的大小。

Flexbox布局在定义伸缩项目大小时伸缩容器会预留一些可用空间，让你可以调节伸缩项目的相对大小和位置。例如，你可以确保伸缩容器中的多余空间平均分配多个伸缩项目，当然，如果你的伸缩容器没有足够大的空间放置伸缩项目时，浏览器会根据一定的比例减少伸缩项目的大小，使其不溢出伸缩容器。综合而言，Flexbox布局功能主要具有以下几点：

第一，屏幕和浏览器窗口大小发生改变也可以灵活调整布局；

第二，可以指定伸缩项目沿着主轴或侧轴按比例分配额外空间（伸缩容器额外空间），从而调整伸缩项目的大小；

第三，可以指定伸缩项目沿着主轴或侧轴将伸缩容器额外空间，分配到伸缩项目之前、之后或之间；

第四，可以指定如何将垂直于元素布局轴的额外空间分布到该元素的周围；

第五，可以控制元素在页面上的布局方向；

第六，可以按照不同于文档对象模型（DOM）所指定排序方式对屏幕上的元素重新排序。也就是说可以在浏览器渲染中不按照文档流先后顺序重排伸缩项目顺序。

Flexbox规范版本众多，浏览器对此语法支持度也各有不同，接下来的内容以最新语法版本为例向大家展示：

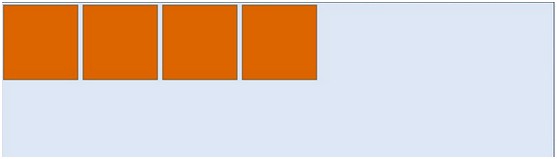
**1.创建一个flex容器**

任何一个flexbox布局的第一步是需要创建一个flex容器。为此给元素设置display属性的值为flex。在Safari浏览器中，你依然需要添加前缀-webkit，

.flexcontainer{ display: -webkit-flex; display: flex; }

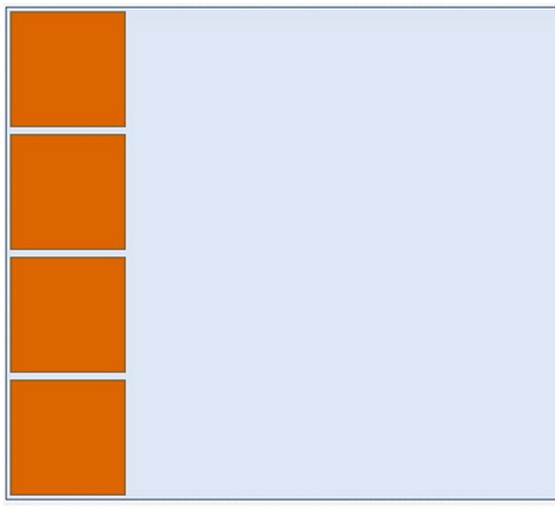
**2.Flex项目显示**

Flex项目是Flex容器的子元素。他们沿着主要轴和横轴定位。默认的是沿着水平轴排列一行。你可以通过flex-direction来改变主轴方向修改为column，其默认值是row。

[](http://img.mukewang.com/5365e24300018b8005570159.jpg)

**3.Flex项目列显示**

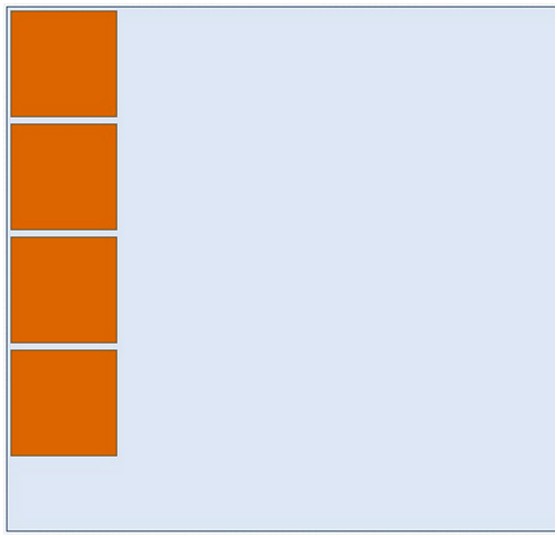
.flexcontainer{ display: -webkit-flex; display: flex; -webkit-flex-direction: column; flex-direction: column; }

[](http://img.mukewang.com/5365e2b300016b2405570509.jpg)

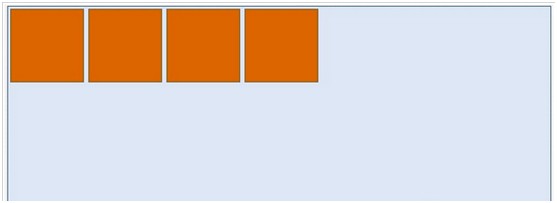
**4.Flex项目移动到顶部**

如何将flex项目移动到顶部，取决于主轴的方向。如果它是垂直的方向通过**align-items**设置；如果它是水平的方向通过**justify-content**设置。

.flexcontainer{ -webkit-flex-direction: column; flex-direction: column; -webkit-justify-content: flex-start; justify-content: flex-start; }

[](http://img.mukewang.com/5365e694000180fd05570538.jpg)

.flexcontainer{ display: -webkit-flex; display: flex; -webkit-flex-direction: row; flex-direction: row; -webkit-align-items: flex-start; align-items: flex-start; }

[](http://img.mukewang.com/5365e7160001b9e405570203.jpg)

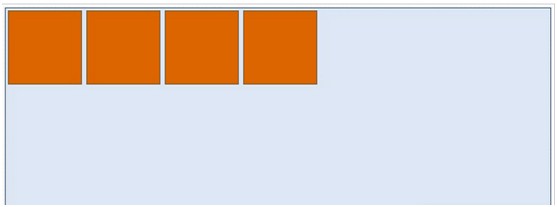
## CSS3 伸缩布局（二）

Flexbox规范版本众多，浏览器对此语法支持度也各有不同，接下来的内容以最新语法版本为例向大家展示：**（接上一节）**

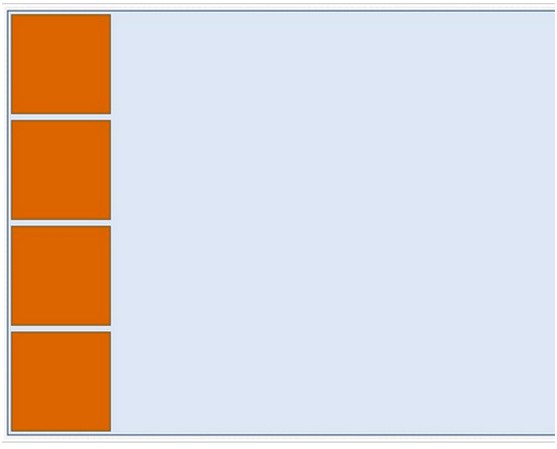
**5.Flex项目移到左边**

flex项目称动到左边或右边也取决于主轴的方向。如果flex-direction设置为row，设置justify-content控制方向；如果设置为column，设置align-items控制方向。

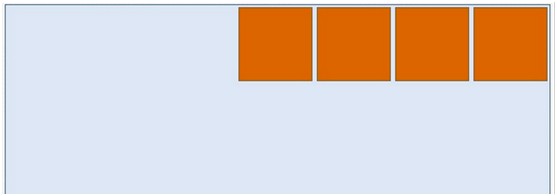
.flexcontainer{ display: -webkit-flex; display: flex; -webkit-flex-direction: row; flex-direction: row; -webkit-justify-content: flex-start; justify-content: flex-start; }

[](http://img.mukewang.com/5365e8f700012f8a05570207.jpg)

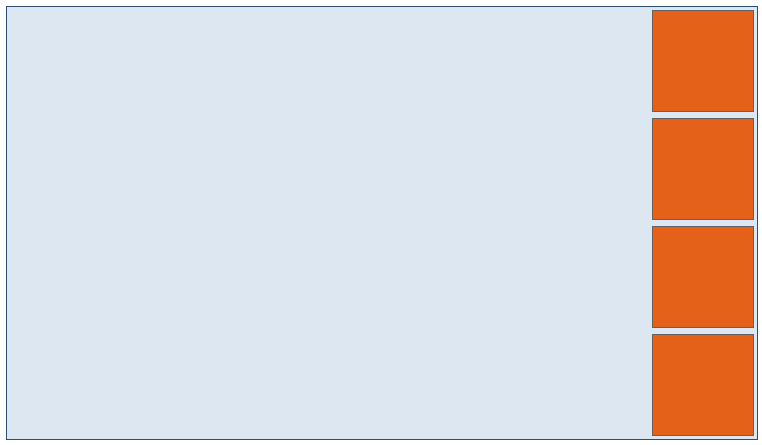
.flexcontainer{ display: -webkit-flex; display: flex; -webkit-flex-direction: column; flex-direction: column; -webkit-align-items: flex-start; align-items: flex-start; }

[](http://img.mukewang.com/5365e9e00001fd9305570449.jpg)

**6.Flex项目移动右边**

.flexcontainer{ display: -webkit-flex; display: flex; -webkit-flex-direction: row; flex-direction: row; -webkit-justify-content: flex-end; justify-content: flex-end; }[](http://img.mukewang.com/5365ea440001835305570196.jpg)

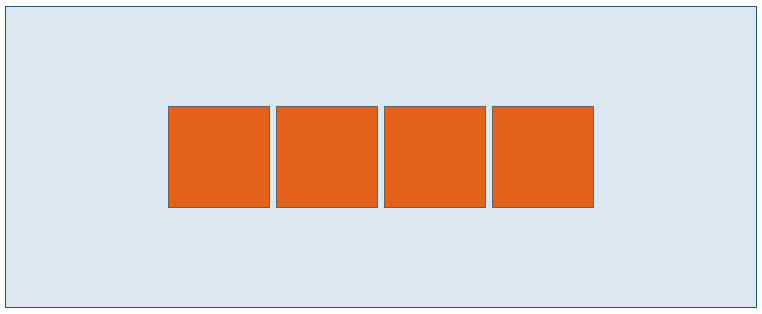
.flexcontainer{ display: -webkit-flex; display: flex; -webkit-flex-direction: column; flex-direction: column; -webkit-align-items: flex-end; align-items: flex-end; }

[](http://img.mukewang.com/5365ebe9000199b107620445.jpg)

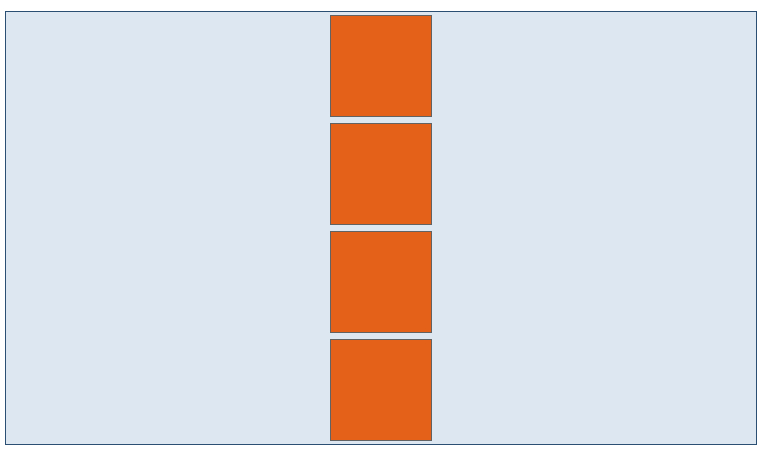
**7.水平垂直居中**

在Flexbox容器中制作水平垂直居中是微不足道的。设置justify-content或者align-items为**center**。另外根据主轴的方向设置flex-direction为row或column。

.flexcontainer{ display: -webkit-flex; display: flex; -webkit-flex-direction: row; flex-direction: row; -webkit-align-items: center; align-items: center; -webkit-justify-content: center; justify-content: center; }

[](http://img.mukewang.com/5365ec680001868c07620314.jpg)

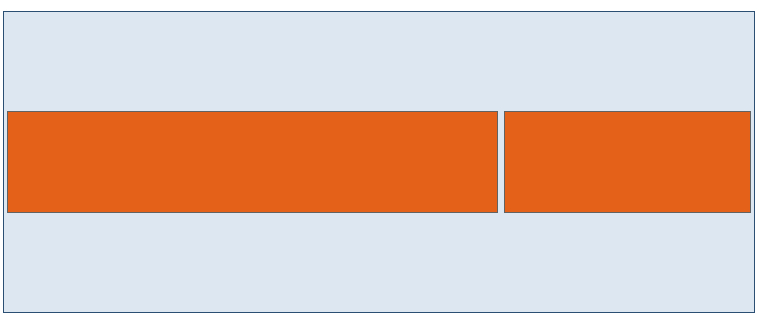
.flexcontainer{ display: -webkit-flex; display: flex; -webkit-flex-direction: column; flex-direction: column; -webkit-align-items: center; align-items: center; -webkit-justify-content: center; justify-content: center; }

[](http://img.mukewang.com/5365ecee00016b0107630450.jpg)

**8.Flex项目实现自动伸缩**

您可以定义一个flex项目，如何相对于flex容器实现自动的伸缩。需要给每个flex项目设置flex属性设置需要伸缩的值。

.bigitem{ -webkit-flex:200; flex:200; }  .smallitem{ -webkit-flex:100; flex:100; }

[](http://img.mukewang.com/5365ed860001a8a907620321.jpg)

**Media Queries——媒体类型（一）**

随着科学技术不断的向前发展，网页的浏览终端越来越多样化，用户可以通过：宽屏电视、台式电脑、笔记本电脑、平板电脑和智能手机来访问你的网站。尽管你无法保证一个网站在不同屏幕尺寸和不同设备上看起来完全一模一样，但至少要让你的Web页面能适配用户的终端，让他更好的呈现在你的用户面前。在本节中，将会学到如何使用CSS3中的Media Queries模块来让一个页面适应不同的终端（或屏幕尺寸），从而让你的页面让用户有一个更好的体验。

**Media Queries**

Media Queries是CSS3新增加的一个模块功能，其最大的特色就是通过CSS3来查询媒体，然后调用对应的样式。其实这个功能在CSS2.1中就有出现过，只不过那个时候此功能并不强大，我们最常见的就是给打印设备添加打印样式。随着时代的变化，这个模块功能越来越强大。

在彻底的了解Media Queries我们需要了解其中的两个重要部分，第一个是媒体类型，第二个是媒体特性。下面的内容我们简单的来了解这两个部分：

**一、媒体类型**

媒体类型（Media Type）在CSS2中是一个常见的属性，也是一个非常有用的属性，可以通过媒体类型对不同的设备指定不同的样式。

在CSS2中常碰到的就是all(全部)、screen(屏幕)、print(页面打印或打印预览模式)，其实媒体类型远不止这三种，W3C总共列出了10种媒体类型。如下表所示：

|  |  |
| --- | --- |
| **值** | **设备类型** |
| All | 所有设备 |
| Braille | 盲人用点字法触觉回馈设备 |
| Embossed | 盲文打印机 |
| Handheld | 便携设备 |
| Print | 打印用纸或打印预览视图 |
| Projection | 各种投影设备 |
| Screen | 电脑显示器 |
| Speech | 语音或音频合成器 |
| Tv | 电视机类型设备 |
| Tty | 使用固定密度字母栅格的媒介，比如电传打字机和终端 |

其中**Screen**、**All**和**Print**为最常见的三种媒体类型。

**media queries——媒体类型（二）**

在实际中媒体类型有近十种之多，实际之中常用的也就那么几种，不过媒体类型的引用方法也有多种，常见的有：**link标签**、**@import**和CSS3新增的**@media**几种：

**link方法**

link方法引入媒体类型其实就是在<link>标签引用样式的时候，通过link标签中的media属性来指定不同的媒体类型。如下所示。

<link rel="stylesheet" type="text/css" href="style.css" media="screen" />

<link rel="stylesheet" type="text/css" href="print.css" media="print" />

**@import方法**

@import可以引用样式文件，同样也可以用来引用媒体类型。@import引入媒体类型主要有两种方式，一种是在样式中通过@import调用另一个样式文件；另一种方法是在<head></head>标签中的<style></style>中引入，但这种使用方法在IE6~7都不被支持，如样式文件中调用另一个样式文件时，就可以指定对应的媒体类型。

@importurl(reset.css) screen;

@importurl(print.css) print;

在<head>中的<style>标签中引入媒体类型方法。

<head>

<style type="text/css">

@importurl(style.css) all;

</style>

</head>

**@media方法**

@media是CSS3中新引进的一个特性，被称为媒体查询。在页面中也可以通过这个属性来引入媒体类型。@media引入媒体类型和@import有点类似也具有两方式。

（1）在样式文件中引用媒体类型：

@media screen {

选择器{/\*你的样式代码写在这里…\*/}

}

（2）使用@media引入媒体类型的方式是在<head>标签中的<style>中引用。

<head>

<style type="text/css">

@media screen{

选择器{/\*你的样式代码写在这里…\*/}

}

</style>

</head>

**Media Queries使用方法**

了解完这些概念性的东西，同学们最想知道的是Media Queries要如何使用？下面我们一起来探讨其使用方法：

Media Queries能在不同的条件下使用不同的样式，使页面在不同在终端设备下达到不同的渲染效果。前面简单的介绍了Media Queries如何引用到项目中，但Media Queries有其自己的使用规则。具体来说,Media Queries的使用方法如下。

@media 媒体类型and （媒体特性）{你的样式}

注意：使用Media Queries必须要使用“@media”开头，然后指定媒体类型（也可以称为设备类型），随后是指定媒体特性（也可以称之为设备特性）。媒体特性的书写方式和样式的书写方式非常相似，主要分为两个部分，第一个部分指的是媒体特性，第二部分为媒体特性所指定的值，而且这两个部分之间使用冒号分隔。例如：

(max-width: 480px)

从前面表中可以得知，主要有十种媒体类型和13种媒体特性，将其组合就类似于不同的CSS集合。但与CSS属性不同的是，媒体特性是通过min/max来表示大于等于或小于做为逻辑判断，而不是使用小于（<）和大于（>）这样的符号来判断。接下来一起来看看Media Queries在实际项目中常用的方式。

**1. 最大宽度max-width**

“max-width”是媒体特性中最常用的一个特性，其意思是指媒体类型小于或等于指定的宽度时，样式生效。如：

@media screen and (max-width:480px){

.ads {

display:none;

}

}

上面表示的是：当屏幕小于或等于480px时,页面中的广告区块（.ads）都将被隐藏。

**2.最小宽度min-width**

“min-width”与“max-width”相反，指的是媒体类型大于或等于指定宽度时，样式生效。

@media screen and (min-width:900px){

.wrapper{width: 980px;}

}

上面表示的是：当屏幕大于或等于900px时，容器“.wrapper”的宽度为980px。

**3.多个媒体特性使用**

Media Queries可以使用关键词**"and**"将多个媒体特性结合在一起。也就是说，一个Media Query中可以包含0到多个表达式，表达式又可以包含0到多个关键字，以及一种媒体类型。

当屏幕在600px~900px之间时，body的背景色渲染为“#f5f5f5”，如下所示。

@media screen and (min-width:600px) and (max-width:900px){

body {background-color:#f5f5f5;}

}

**4.设备屏幕的输出宽度Device Width**

在智能设备上，例如iPhone、iPad等，还可以根据屏幕设备的尺寸来设置相应的样式（或者调用相应的样式文件）。同样的，对于屏幕设备同样可以使用“min/max”对应参数，如“min-device-width”或者“max-device-width”。

<link rel="stylesheet" media="screen and (max-device-width:480px)" href="iphone.css" />

上面的代码指的是“iphone.css”样式适用于最大设备宽度为480px，比如说iPhone上的显示，这里的“max-device-width”所指的是设备的实际分辨率，也就是指可视面积分辨率。

**5. not关键词**

使用关键词“not”是用来排除某种制定的媒体类型，也就是用来排除符合表达式的设备。换句话说，not关键词表示对后面的表达式执行取反操作，如：

@media not print and (max-width: 1200px){样式代码}

上面代码表示的是：样式代码将被使用在除打印设备和设备宽度小于1200px下所有设备中。

**6.only关键词**

only用来指定某种特定的媒体类型，可以用来排除不支持媒体查询的浏览器。其实only很多时候是用来对那些不支持Media Query但却支持Media Type的设备隐藏样式表的。其主要有：支持媒体特性的设备，正常调用样式，此时就当only不存在；表示不支持媒体特性但又支持媒体类型的设备，这样就会不读样式，因为其先会读取only而不是screen；另外不支持Media Queries的浏览器，不论是否支持only，样式都不会被采用。如

<linkrel="stylesheet" media="only screen and (max-device-width:240px)" href="android240.css" />

在Media Query中如果没有明确指定Media Type，那么其默认为all，如：

<linkrel="stylesheet" media="(min-width:701px) and (max-width:900px)" href="mediu.css" />

另外在样式中，还可以使用多条语句来将同一个样式应用于不同的媒体类型和媒体特性中，指定方式如下所示。

<linkrel="stylesheet" type="text/css" href="style.css" media="handheld and (max-width:480px), screen and (min-width:960px)" />

上面代码中style.css样式被用在宽度小于或等于480px的手持设备上，或者被用于屏幕宽度大于或等于960px的设备上。

到目前为止，CSS3 Media Queries得到了众多浏览器支持，除了**IE6-8**浏览器不支持之外，在所有现代浏览器中都可以完美支持。还有一个与众不同的时，**Media Queries在其他浏览器中不要像其他CSS3属性一样在不同的浏览器中添加前缀**。

## Responsive设计（一）

什么是响应式设计呢？维基百科是这样对响应式作的描述：“Responsive设计简单的称为RWD，是精心提供各种设备都能浏览网页的一种设计方法，RWD能让你的网页在不同的设备中展现不同的设计风格。”从这一点描述来说，RWD不是流体布局，也不是网格布局，而是一种独特的网页设计方法。

响应式设计要考虑元素布局的秩序，而且还需要做到“有求必应”，那就需要满足以下三个条件：网站必须建立灵活的网格基础；引用到网站的图片必须是可伸缩的；不同的显示风格，需要在Media Queries上写不同的样式。

要想灵活的使用Responsive，仅满足这几个条件还是不够的，我们必须对Responsive有一个全面的了解，那么要了解Responsive，就得先了解他的一些术语：

#### ****1.流体网格****

流体网格是一个简单的网格系统，这种网格设计参考了流体设计中的网格系统，将每个网格格子使用百分比单位来控制网格大小。这种网格系统最大的好处是让你的网格大小随时根据屏幕尺寸大小做出相对应的比例缩放。

#### ****2.弹性图片****

弹性图片指的是不给图片设置固定尺寸，而是根据流体网格进行缩放，用于适应各种网格的尺寸。而实现方法是比较简单，一句代码就能搞定的事情。

img {max-width:100%;}

不幸的是，这句代码在IE8浏览器存在一个严重的问题，让你的图片会失踪。当然弹性图片在响应式设计中如何更好的实现，到目前为止都还存在争议，也还在不断的改善之中。

为每一个断点提供不同的图片，这是一个比较头痛的事情，因为Media Queries并不能改变图片“src”的属性值，那有没有办分法可以解决呢？可以参考一下下面的解决方法。使用background-image给元素使用背景图片，显示/隐藏父元素，给父元素使用不同的图片，然后通过Media Queries来控制这些图片显示或隐藏。

来看一个断点解决图片自适应的例子。

<img src="image.jpg" data-src-600px="image-600px.jpg" data-src-800px="image-800px.jpg" alt="" />

对应的CSS代码：

@media (min-device-width:600px){

img[data-src-600px]{

content: attr(data-src-600px,url);

}

}

@media (min-device-width:800px){

img[data-src-800px] {

content:attr(data-src-800px,url);

}

}

#### 请注意：这仅仅是解决响应式图片自适应的一种思路，但这种方案仅仅适配的断点较少。并不太实用，此处仅为扩展同学们的思路。

#### ****3.媒体查询****

媒体查询在CSS3中得到了强大的扩展。使用这个属性可以让你的设计根据用户终端设备适配对应的样式。这也是响应式设计中最为关键的。可以说Responsive设计离开了Medial Queries就失去了他生存的意义。简单的说媒体查询可以根据设备的尺寸，查询出适配的样式。Responsive设计最关注的就是：根据用户的使用设备的当前宽度，你的Web页面将加载一个备用的样式，实现特定的页面风格。

## Responsive设计（二）

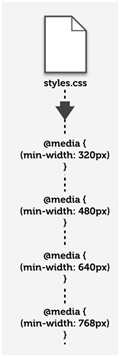
#### ****（接上一节内容）****

#### ****4.屏幕分辨率****

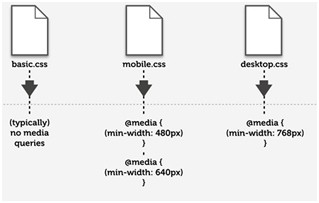
屏幕分辨简单点说就是用户显示器的分辨率，深一点说，屏幕分辨率指的是用户使用的设备浏览您的Web页面时的显示屏幕的分辨率，比如说智能手机浏览器、移动电脑浏览器、平板电脑浏览器和桌面浏览器的分辨率。Responsive设计利用Media Queries属性针对浏览器使用的分辨率来适配对应的CSS样式。因此屏幕分辨率在Responsive设计中是一个很重要的东西，因为只有知道Web页面要在哪种分辨率下显示何种效果，才能调用对应的样式。

#### ****5.主要断点****

主要断点，在Web开发中是一个新词，但对于Responsive设计中是一个很重要的一部分。简单的描述就是，设备宽度的临界点。在Media Queries中，其中媒体特性“min-width”和“max-width”对应的属性值就是响应式设计中的断点值。简单点说，就是使用主要断点和次要断点，创建媒体查询的条件。而每个断点会对应调用一个样式文件（或者样式代码），如下图所示：

[](http://img.mukewang.com/53660bc60001051601200357.jpg)

上图的style.css样式文件运用在Web页面中，但这个样式文件包括了所有风格的样式代码，也就是说所有设备下显示的风格都通过这个样式文件下载下来。当然，在实际中还可以使用另一种方法，也就是在不同的断点加载不同的样式文件，如下图所示。

[](http://img.mukewang.com/53660c230001fb9603190203.jpg)  
上图主要显示的是主要断点，主要断点加载的不同样式文件直接将影响Web的效果。除了主要断点之外，为了满足更多效果时，还可以在这个基础上添次要断点。不过主要断点和次要断点增加之后，需要维护的样式也相应的增加，成本也相对应的增加。因此在实际使用中，需要根据自身产品需求，决定断点。

综合下来，设置断点应把握三个要点：满足主要的断点；有可能的话添加一些别的断点；设置高于1024的桌面断点。

**Responsive布局技巧**

通过上面的介绍，大家对响应式设计有了一定的了解，但在实际制作中还是有一些布局技巧的：

**在Responsive布局中，可以毫无保留的丢弃：**

第一， 尽量少用无关紧要的div；

第二，不要使用内联元素（inline）；

第三，尽量少用JS或flash；

第四，丢弃没用的绝对定位和浮动样式；

第五，摒弃任何冗余结构和不使用100%设置。

有舍才有得，丢弃了一些对Responsive有影响的东东，那么**有哪些东东能帮助Responsive确定更好的布局呢**？

第一，使用HTML5 Doctype和相关指南；

第二，重置好你的样式（reset.css）；

第三，一个简单的有语义的核心布局；

第四，给重要的网页元素使用简单的技巧，比如导航菜单之类元素。

使用这些技巧，无非是为了保持你的HTML简单干净，而且在你的页面布局中的关键部分（元素）不要过分的依赖现代技巧来实现，比如说CSS3特效或者JS脚本。

说了这么多，怎么样的布局或者说HTML结构才是简单干净的呢？这里教大家一个**快速测试**的方法。你首先禁掉你页面中所有的样式（以及与样式相关的信息），在浏览器中打开，如果你的内容排列有序，方便阅读，那么你的这个结构不会差到哪里去。

## Responsive设计——meta标签

1.在web上浏览的时候，响应式布局是好用的，放大缩小页面都可以实现页面变更，但是在手机上调试的时候死活不能用

2.因为css中@media是根据window的宽度来控制css的，所以我尝试输出了一下在window变更的时候window的width，结果发现PC上使用时一切正常，而手机上输出的一直是980。

原因很简单，手机上的浏览器是全屏的，我手机实际宽度是320像素，而我手机分辨率是980宽度，所以手机上打出来的是980而不是320

3.增加

<meta name="viewport" content="width=device-width, initial-scale=1.0">

以后发现手机window的width变为320了，即预期效果  
viewpoint说明

width 控制viewpoint的宽度，可以是固定值，也可以是device-width，即设备的宽度  
height 高度  
initial-scale 控制初始化缩放比例，1.0表示不可以缩放  
maximum-scale 最大缩放比例  
minimum-scale 最小缩放比例

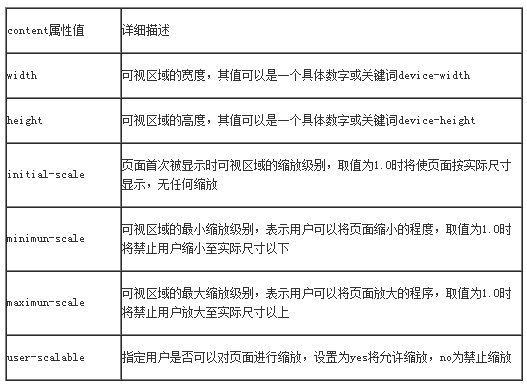
可见如果不定义viewpoint的话，页面宽度以屏幕分辨率为基准，而设置以后可以根据设备宽度来调整页面，达到适配终端大小的效果

最后还有一个不可或缺的东东，那就是meta标签，可以说，在响应式设计中如果没有这个meta标签，你就是蹩脚的，响应式设计就是空谈。

个meta标签被称为可视区域meta标签，其使用方法如下。

<meta name=”viewport” content=”” />

在content属性中主要包括以下属性值，用来处理可视区域。

[](http://img.mukewang.com/53660f2c0001190005270386.jpg)

在实际项目中，为了让Responsive设计在智能设备中能显示正常，也就是浏览Web页面适应屏幕的大小，显示在屏幕上，可以通过这个可视区域的**meta标签**进行重置，告诉他使用设备的宽度为视图的宽度，也就是说禁止其默认的自适应页面的效果，具体设置如下：

<meta name=”viewport” content=”width=device-width,initial-scale=1.0” />

另外一点，由于Responsive设计是结合CSS3的Media Queries属性，才能尽显Responsive设计风格，但大家都清楚，**在IE6-8中完全是不支持CSS3 Media**。下面我们一起来看看CSS3 Meida Queries在标准设备上的运用，大家可以把这些样式加到你的样式文件中，或者单独创建一个名为“responsive.css”文件，并在相应的条件中写上你的样式，让他适合你的设计需求。

脚本下载地址：

media-queries.js(http://code.google.com/p/css3-mediaqueries-js/)

respond.js(https://github.com/scottjehl/Respond)

<!—[if lt IE9]>

<scriptsrc=http://css3-mediaqueries-js.googlecode.com/svn/trunk/css3-mediaqueries.js></script>

​<![endif]>

## Responsive设计——不同设备的分辨率设置

下面我们一起来看看CSS3 Meida Queries在标准设备上的运用，大家可以把这些样式加到你的样式文件中，或者单独创建一个名为“responsive.css”文件，并在相应的条件中写上你的样式，让他适合你的设计需求：

**1.1024px显屏**

@media screen and (max-width : 1024px) {

/\* 样式写在这里 \*/

}

**2.800px显屏**

@media screen and (max-width : 800px) {

/\* 样式写在这里 \*/

}

**3.640px显屏**

@media screen and (max-width : 640px) {

/\* 样式写在这\*/

}

**4.iPad横板显屏**

@media screen and (max-device-width: 1024px) and (orientation: landscape) {

/\* 样式写在这 \*/

}

**5.iPad竖板显屏**

@media screen and (max-device-width: 768px) and (orientation: portrait) {

/\* 样式写在这 \*/

}

**6.iPhone 和 Smartphones**

@media screen and (min-device-width: 320px) and (min-device-width: 480px) {

/\* 样式写在这 \*/

}

现在有关于这方面的运用也是相当的成熟，twitter的[Bootstrap](http://twitter.github.com/bootstrap/scaffolding.html#responsive)第二版本中就加上了这方面的运用。大家可以对比一下：

@media (max-width: 480px) { ... }

@media (max-width: 768px) { ... }

@media (min-width: 768px) and (max-width: 980px) { ... }

 @media (min-width: 1200px) { .. }

**自由缩放属性resize**

为了增强用户体验，CSS3增加了很多新的属性，其中**resize**就是一个重要的属性，它允许用户通过拖动的方式来修改元素的尺寸来改变元素的大小。到目前为止，可以使用overflow属性的任何容器元素。

在此之前，Web设计师为了要实现这样具有拖动效果的UI，使用大量的脚本代码才能实现，这样费时费力，效率极低。

resize属性主要是用来改变元素尺寸大小的，其主要目的是增强用户体验。但使用方法却是极其的简单，先从其语法入手。

resize: none | both | horizontal | vertical | inherit

**取值说明：**

|  |  |
| --- | --- |
| **属性值** | **取值说明** |
| none | 用户不能拖动元素修改尺寸大小。 |
| both | 用户可以拖动元素，同时修改元素的宽度和高度 |
| horizontal | 用户可以拖动元素，仅可以修改元素的宽度，但不能修改元素的高度。 |
| vertical | 用户可以拖动元素，仅可以修改元素的高度，但不能修改元素的宽度。 |
| inherit | 继承父元素的resize属性值。 |

例如：通过resize属性，让文本域可以沿水平方向拖大。代码为：

textarea {

-webkit-resize: horizontal;

-moz-resize: horizontal;

-o-resize: horizontal;

-ms-resize: horizontal;

resize: horizontal;

}

**CSS3外轮廓属性**

外轮廓outline在页面中呈现的效果和边框border呈现的效果极其相似，但和元素边框border完全不同，外轮廓线不占用网页布局空间，不一定是矩形，外轮廓是属于一种动态样式，只有元素获取到焦点或者被激活时呈现。

outline属性早在CSS2中就出现了，主要是用来在元素周围绘制一条轮廓线，可以起到突出元素的作用。但是并未得到各主流浏览器的广泛支持，在CSS3中对outline作了一定的扩展，在以前的基础上增加新特性。outline属性的基本语法如下：

outline: ［outline-color］ || [outline-style] || [outline-width] || [outline-offset] || inherit

从语法中可以看出outline和border边框属性的使用方法极其类似。outline-color相当于border-color、outline-style相当于border-style，而outline-width相当于border-width，只不过CSS3给outline属性增加了一个outline-offset属性，其取值说明如下。

|  |  |
| --- | --- |
| **属性值** | **属性值说明** |
| outline-color | 定义轮廓线的颜色，属性值为CSS中定义的颜色值。在实际应用中，可以将此参数省略，省略时此参数的默认值为黑色。 |
| outline-style | 定义轮廓线的样式，属性为CSS中定义线的样式。在实际应用中，可以将此参数省略，省略时此参数的默认值为none，省略后不对该轮廓线进行任何绘制。 |
| outline-width | 定义轮廓线的宽度，属性值可以为一个宽度值。在实际应用中，可以将此参数省略，省略时此参数的默认值为medium，表示绘制中等宽度的轮廓线。 |
| outline-offset | 定义轮廓边框的偏移位置的数值，此值可以取负数值。当此参数的值为正数值，表示轮廓边框向外偏离多少个像素；当此参数的值为负数值，表示轮廓边框向内偏移多少个像素。 |
| inherit | 元素继承父元素的outline效果。 |

**CSS生成内容**

在Web中插入内容，在CSS2.1时代依靠的是JavaScript来实现。但进入CSS3进代之后我们可以通过CSS3的伪类“:before”，“:after”和CSS3的伪元素“::before”、“::after”来实现，其关键是依靠CSS3中的“content”属性来实现。不过这个属性对于**img**和**input**元素不起作用。

content配合CSS的伪类或者伪元素，一般可以做以下四件事情：

|  |  |
| --- | --- |
| **功能** | **功能说明** |
| none | 不生成任何内容 |
| attr | 插入标签属性值 |
| url | 使用指定的绝对或相对地址插入一个外部资源（图像，声频，视频或浏览器支持的其他任何资源） |
| string | 插入字符串 |

**实例展示：**

在CSS中有一种清除浮动的方法叫“clearfix”。而这个“clearfix”方法就中就使用了“content”，只不过只是在这里插入了一个空格。如下所示：

.clearfix:before,

.clearfix:after {

content:””;

display:table;

}

.clearfix:after {

clear:both;

overflow:hidden;

}

上面这个示例是最常见的，也是最简单的，我们再来看一个插入元素属性值的方法。

假设我们有一个元素：

<a href="##" title="我是一个title属性值，我插在你的后面">我是元素</a>

可以通过”:after”和”content:attr(title)”将元素的”title”值插入到元素内容“我是元素”之后：

a:after {

content:attr(title);

color:#f00;

}

效果如下：

http://img.mukewang.com/5365fae20001649403560028.jpg