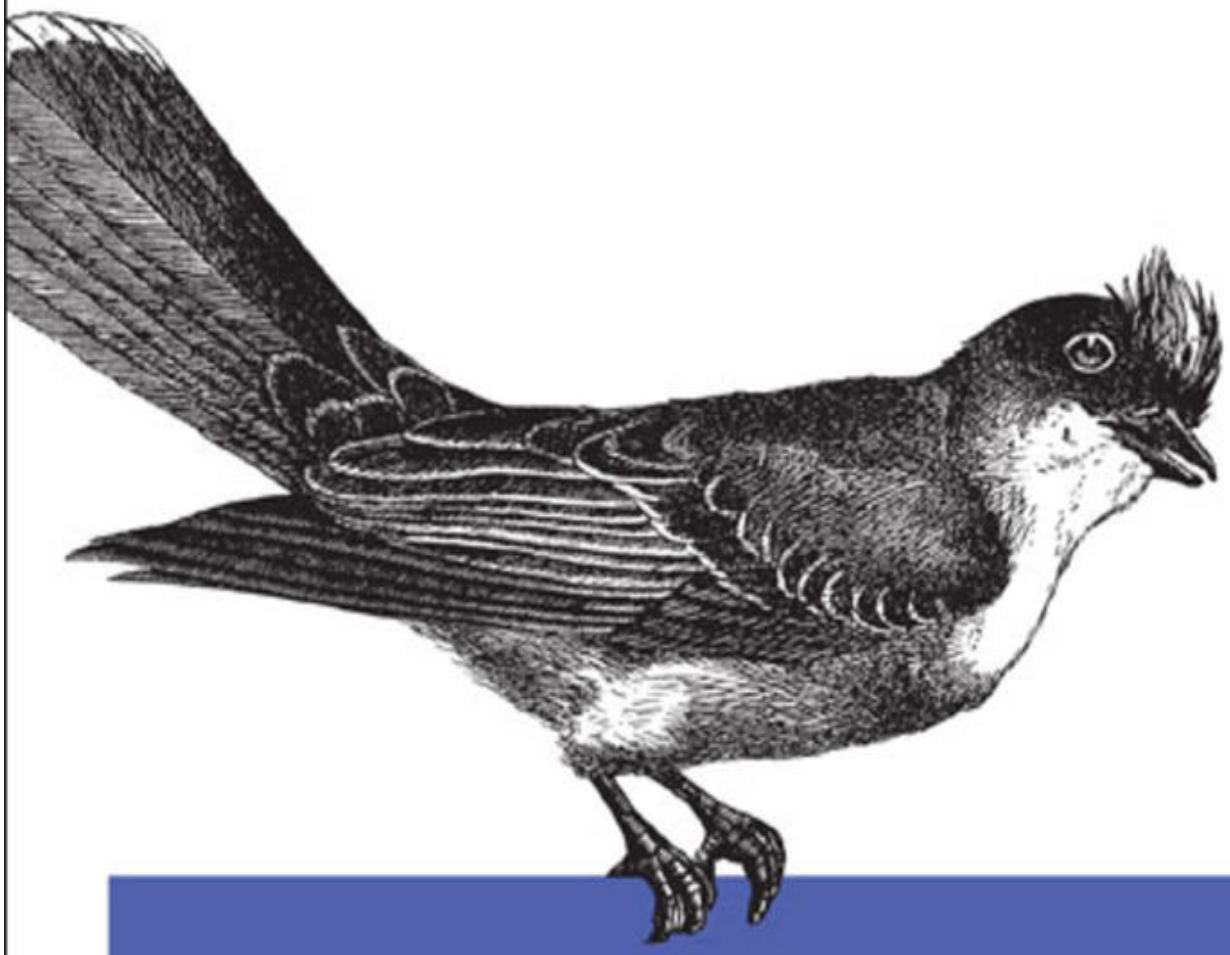


Web 前端干货铺

JayaZhao 著

doyoe

margin系列

移动开发系列

其他

margin系列之keyword auto

margin系列之keyword auto

> 原作者：doyoe
> 原文地址：http://blog.doyoe.com/2013/11/29/css/margin%E7%B3%BB%E5%88%97%E4%B9%8Bkeyword%20auto/

margin的重要性：

有个不容置疑的事，前端开发人员没有人能够忽视CSS margin的重要性。CSS coding时，margin的使用

频率就如同呼吸般频繁，如果我可以说得夸张点的话。

margin作为CSS盒模型基本组成要素之一，是非常Basis的一个技术手段，所以我想对于它的一些基本情况应该不用太介绍了？

margin经常被用来做什么？

- 让块元素水平居中；
- 让元素之间留有舒适的留白；
- 处理特殊的first或last，大家懂的？
- 一些布局；

需要注意的地方：

- margin折叠；
- margin的百分比值；
- margin的auto值；
- margin和相对偏移top, right, bottom, left的异同；
- IE6浮动双margin Bug；
- IE6浮动相邻元素3px Bug；

看起来似乎有不少的知识点？恩，这些都是我们需要了解的，包括一些没有被列举出来的点。

今天要讲的其实只是其中很少的一部分，恩，标题里有：keyword auto

keyword auto

auto是margin的可选值之一。相信大家平时使用auto值时，最多的用法大概是 `margin: 0 auto;` 和 margin: auto;，恩，是的，块元素水平居中。让我们来看看代码实现：

CSS:

```
#demo{
    width: 500px;
    margin: auto; /* 或者 margin: 0 auto; */
}
```

HTML:

```
<div id="demo">
    <p>恩，我就是那个需要水平居中的家伙。</p>
</div>
```

为了更明显点，我们来看个例子：margin实现块元素水平居中。Cool，这么简单就实现了水平居中。

不过你可能也发现了不论是 `margin: auto;` 还是 `margin: 0 auto;` 效果都是一样的，都是让 #demo 水平居中了，但纵向并没有任何变化。

大家都知道 margin 是复合属性，也就是说 `margin: auto;` 其实相当于 `margin: auto auto auto auto;` ，`margin: 0 auto;` 相当于 `margin: 0 auto 0 auto;` ，四个值分别对应上右下左。至于CSS中的上、右、下、左顺序就不做赘述了。

根据规范，`margin-top: auto;` 和 `margin-bottom: auto;` ，其计算值为0。这也就解释了为什么 `margin: auto;` 等同于 `margin: 0 auto;` 。但仅此而已吗？让我们来看看规范描述：

> 原文：On the A edge and C edge, the used value of ‘auto’ is 0.
>
> 翻译：如果场景是A和C，那么其 auto 计算值为 0。

‘writing-mode’ of containing block	‘direction’ of containing block	Meaning of… “A”	“B”	“C”	“D”
‘horizontal-tb’	‘ltr’	top	right	bottom	left
	‘rtl’	top	left	bottom	right
‘vertical-rl’	‘ltr’	right	bottom	left	top
	‘rtl’	right	top	left	bottom
‘vertical-lr’	‘rtl’	left	bottom	right	top
	‘ltr’	left	top	right	bottom

> 更详细请参阅：margin properties

由此可见，它们还与书写模式 `writing-mode` 和 文档流方向 `direction` 有关。所以我们说 `margin: auto;` 等同于 `margin: 0 auto;` 是不太精准的，因为还有前置条件。
了解这些很重要，这有助于理解 `margin` 属性的设计意图。
OK，聊了这么多，我们回到默认的 `writing-mode: horizontal-tb;` 和 `direction: ltr;` 的情况继续往下，后面的话题都基于这个前提。

为什么auto能实现水平居中？

这是因为水平方向的 `auto` ，其计算值取决于可用空间（剩余空间）。

> 原文：On the B edge and D edge, the used value depends on the available space.
> 翻译：如果场景是B和D，那么其 `auto` 计算值取决于可用空间。

想象这样一个场景，一个宽100px的p被包含在一个宽500px的div内，此时设置 p 的 margin-left 值为 auto，大家觉得结果会怎样？

CSS:

```
#demo{
    width: 500px;
}
#demo p{
    width: 100px;
    margin-left: auto;
}
```

HTML:

```
<div id="demo">
    <p>恩，我就是那个p。</p>
</div>
```

结果你猜到了吗？没猜到也不怕，用事实说话：margin-left关键字auto结果猜想。

好了，结果得到了，p相对于包含块右对齐了，这与规范描述一致。 `margin-left:auto;` 自动占据了包含块的可用空间，即 500 - 100px = 400px。也就是说auto最后的计算值为400px，即 `margin-left:400px;` 。所以 `margin-right:auto;` 的结果会相当于左对齐。
到这里，相信大家都知道为什么 `margin: auto;` 和 `margin: 0 auto;` 能实现水平居中了。因为左右方向的auto值均分了可用空间，使得块元素得以在包含块内居中显示。
至于垂直方向上为什么无法居中，还有更深层的原因吗？大家可以思考一下。

可参考：

- http://www.w3.org/TR/css3-box/#margins
- http://dev.w3.org/csswg/css-box/#the-margin-properties
- http://dev.w3.org/csswg/css-box/#Calculating

margin系列之百分比

margin系列之百分比

> 原作者：doyoe
> 原文链接：http://blog.doyoe.com/2013/11/30/css/margin%E7%B3%BB%E5%88%97%E4%B9%8B%E7%99%BE%E5%88%86%E6%AF%94/

你可能从没注意过它

在 margin系列之keyword auto 中，说过了 `margin` 值为 `auto` 的情况，这次要聊的是值为百分比的情形。

我必须承认这是一个非常基础的知识点，但有一段时间我发现很多人对此有错误的认知。偶尔在面试或者分享的时候，我会问到这个问题，有人会脱口而出的告诉我他对此的感性理解。

或许现在大多数人对此不屑一顾，但我仍想说一说，这样以后就可以不再问类似的问题了。

假设有这样一个场景

仍然以一个问题来开始，这是我之前在 微博 发过的，原文是这样的：

假设一个块级包含容器，宽1000px，高600px，块级子元素定义 `margin:10% 5%;` 大家说说 `margin` 的 `top, right, bottom, left` 计算值最终是多少？

我记得得到不少 `100px 30px 100px 30px` 的反馈，我们来还原代码：

CSS:

```
#demo{
    width: 1000px;
    height: 600px;
}
#demo p{
    margin: 10% 5%;
}
```

HTML:

```
<div id="demo">
    <p>恩，注意看我所在的位置。</p>
</div>
```

事实告诉我们结果是 `100px 50px 100px 50px` ，不论结果是否符合你的预期，我们先来看demo验证一下：margin百分比结果猜想，当然，你也根据上面还原的代码自己创建一个例子。

为什么会这样？

诧异吗？不用怀疑浏览器出了问题，因为这是正确的实现。

规范中注明 `margin` 的百分比值参照其包含块的宽度进行计算。

当然，它不会这么简单，和上篇文章 keyword auto 一样，这只发生在默认的

`writing-mode: horizontal-tb;` 和 `direction: ltr;` 的情况下。

当书写模式变成纵向的时候，其参照将会变成包含块的高度。我们改变一下上面例子中的CSS书写模式：

CSS:

```
#demo{
    -webkit-writing-mode: vertical-rl; /* for browsers of webkit engine */
    writing-mode: tb-rl; /* for ie */
}
```

在 #demo 中添加这2句，其它code不变。也有个例子供观：书写模式影响margin百分比的参照对象。

恩，这回的结果是 `60px 30px 60px 30px` ，因为其参照对象变成了包含块的高度。

顺带再说说

你是否觉得这不符合常规的感性认知？感性认知更多感觉应该横向参照包含块宽度，纵向参照包含块高度。

其实这是为了要横向和纵向2个方向都创建相同的margin，如果它们的参照物不一致，那就无法得到两个方向相同的留白。

你可能会问那为什么要选择宽度做参照而不是高度呢？

这其实更多的要从CSS设计意图上去想，因为CSS的基础需求是排版，而通常我们所见的横排文字，其水平宽度一定（仔细回想一下，如果没有显式的定义宽度或者强制一行显示，都会遇到边界换行，而不是水平延展），垂直方向可以无限延展。但当书写模式为纵向时，其参照就变成了高度而不再是宽度了。

还记得我们在 margin系列之keyword auto 留了个问题：为什么 `margin: auto;` 无法再纵向上垂直居中？其实原因也是上面所说的，因为纵向是可以无限延展的，所以没有一个一定的值可以被参照被用来计算。

受书写模式影响的其它特性：

- margin折叠
- margin的keyword auto value
- padding的百分比值

可参考：

- http://dev.w3.org/csswg/css-box/#the-margin-properties
- http://dev.w3.org/csswg/css-box/#ltpercentagegt
- [http://dev.w3.org/csswg/css-box/#Calculating](http://dev.w3.org/csswg/css-box/#Calculating

margin系列之与相对偏移的异同

margin系列之与相对偏移的异同

> 原作者：doyoe
> 原文链接：http://blog.doyoe.com/2013/12/02/css/margin%E7%B3%BB%E5%88%97%E4%B9%8B%E4%B8%8E%E7%9B%B8%E5%AF%B9%E5%81%8F%E7%A7%BB%E7%9A%84%E5%BC%82%E5%90%8C/

也许我们是一样的

可能大家都用会 `margin` 或者相对偏移的 `top, right, bottom, left` 来做一些类似元素偏移的事，其实我也会。这回我们只聊 `relative` 下的 `top, right, bottom, left` 。

比如说我们想让一个 div 向下偏移 50 个像素，通常会这样：

Case 1:

```
#demo .margin-top{
    margin-top: 50px;
}
```

Case 2:

```
#demo .relative-top{
    position:relative;
    top: 50px;
}
```

HTML:

```
<div id="demo">
    <div class="margin-top">我是margin-top:50px</div>
    <div class="relative-top">我是relative top:50px</div>
</div>
```

上述2种方式，我们都可以完成 div 向下偏移 50 个像素的需求。来看看 `DEMO1`：margin-top vs. relative top

其实它们真的有相似的地方

从上面的例子，可以发现不论是 `margin-top` 还是 `relative top` 都是以自身作为参照物进行偏移的。

顺带说一下 `absolute` 偏移相对的是包含块，并且其偏移值是从包含块的 `padding` 区域开始计算。

但它们真的不一样，我们来看看规范怎么说：

margin:

> 原文：Margins in CSS serve to add both horizontal and vertical space between boxes.
>
> 翻译：CSS中的margin用来添加盒子之间的水平和垂直间隙。

top, right, bottom, left:

> 原文：An element is said to be positioned if its ‘position’ property has a value other than ‘static’. Positioned elements generate positioned boxes, and may be laid out according to the following four physical properties: top, right, bottom, left.
>
> 翻译：一个元素的position属性值如果不为static则发生定位。定位元素会产生定位盒，并且会根据top, right, bottom, left 这4个物理属性进行排版布局。

意思很明白，`margin` 是用来增加自身与它人之间的空白，而 `top, right, bottom, left` 是用来对自身进行排版，作用完全不同。
也就是说 `margin` 是互动的，因为它要影响它人；而 `top, right, bottom, left` 是孤独的，它只是自己一个人玩，不影响它人。

回到之前那个例子

在 `DEMO1` 中，我们看到2个方法都可以做到向下偏移50px，不过它们的意义不太一样。
margin的case: 让该div的顶部与其相邻的元素（这里即其包含块）留有50px的空白。

top的case: 让该div距离其包含块顶部边缘50px，因为是 relative ，所以这里是距离div自己的顶部边缘。

我们大胆假设一下

如果我们设置 `margin-bottom` 和 `bottom` 的值也为50px，它们的表现将完全不一样，你觉得呢？
恩，试试：

Case 1:

```
#demo .margin-bottom{
    margin-bottom: 50px;
}
```

Case 2:

```
#demo .relative-bottom{
    position: relative;
    bottom: 50px;
}
```

HTML:

```
<div id="demo">
    <p class="margin-bottom">我是margin-bottom:50px</p>
    <p class="relative-bottom">我是relative bottom:50px</p>
</div>
```

验证猜想的时刻到了，来看看 DEMO2 : 对margin-bottom和bottom的表现猜想

结果有出乎你的意料吗？好吧，不论怎么，解释下为什么会这样？

前面我们说过 margin 是用来增加自身与它人之间的间隙，所以它距包含块底部有50px，这应该能理解？那为什么 bottom 会跑到上面去？相信仔细看了之前的描述，你应该知道。因为它要相对自己的底部边缘有50px，恩，不是-50px，所以它等于是向上偏移了50px，很简单，不是吗？
还有一个细节你注意到了吗？ bottom 没有撑开它的包含块，仔细看看它的包含块的背景色区域。这正好也验证了之前说的 top, right, bottom, left 是孤独的，它只是自己一个人玩，不影响它人。
孤独患者

我们将 DEMO1 稍改改，为其加上一点上下文，再看看结果：

Case 1:

```
#demo .margin-top p{
    margin-top: 50px;
}
```

Case 2:

```
#demo .relative-top p{
    position:relative;
    top: 50px;
}
```

HTML:

```
<div id="demo">
   <div class="margin-top">
      <p>我是margin-top:50px</p>
      我是一段随便什么上下文
   </div>
   <div class="relative-top">
      <p>我是relative top:50px</p>
      我是一段随便什么上下文
   </div>
</div>
```

迫不及待的要看看实际例子了，不是么？ DEMO3：再次验证一下top, right, bottom, left是孤独患者

至此可以再次说明 top, right, bottom, left 真的和其上下文一毛钱关系都没有，绝对的孤单患者。

所以 margin 和 top, right, bottom, left 分别要在什么场景使用，应该可以有考量的依据了，不是么？enjoy it.

似乎还漏了点啥

差点就这么结篇了，想起还有点遗漏的地方。

当position为relative时，如果top和bottom都是auto，则它们的计算值是0，right和left亦然；如果top和bottom其中一个为auto，则auto相当于另一个的负值，即top = -bottom，right和left亦然；如果top和bottom的值都不为auto，则忽略bottom，如果right和left的值都不为auto，则忽略right。

好吧，不得不再写个例子： DEMO4：top, right, bottom, left详述

至于margin，就留给大家思考一下也不错 ^_^
enjoy it again.

可参考：

- http://dev.w3.org/csswg/css-box/#the-margin-properties
- http://dev.w3.org/csswg/css-position/#box-offsets-trbl

margin系列之外边距折叠

margin系列之外边距折叠

> 原作者：doyoe

> 原文链接：http://blog.doyoe.com/2013/12/04/css/margin%E7%B3%BB%E5%88%97%E4%B9%8B%E5%A4%96%E8%BE%B9%E8%B7%9D%E6%8A%98%E5%8F%A0/

不怀疑你也在工作中遇见过

几乎可以不用怀疑，前端工作中的你一定遇见过 margin 折叠。

不确定？别着急，你可能写过这样的代码：

CSS:

```
p{
   margin: 50px;
}
```

HTML:

```
<div id="demo">
   <p>我是一个华丽的段落，别看我文字少</p>
   <p>我是另一个华丽的段落</p>
</div>
```

大家觉得这 2p 之间会发生点什么？是会合体呢？还是分开？来看看 `DEMO1` margin折叠

好吧，它们真的合体了。按照常规思路，这 `2p` 之间的空白应该是第一个 `p` 的 `margin-bottom 50px` 加上第二 `p` 的 `margin-top 50px`，即 `50 + 50px = 100px` ，但结果总是出乎意料不是么？它们之间只剩下了 `50px`，这就是 `margin折叠`。所以任何人遇见过我都不会觉得意外，因为这样的Code看起来没有任何问题。

它们之间到底发生了些什么？

这 `2p` 内部到底发生了什么，才会有这样的表现？

早在CSS1中就有对 `margin` 折叠的说明，我们来看看：

> 原文：The width of the margin on non-floating block-level elements specifies the minimum distance to the edges of surrounding boxes. Two or more adjoining vertical margins (i.e., with no border, padding or content between them) are collapsed to use the maximum of the margin values. In most cases, after collapsing the vertical margins the result is visually more pleasing and closer to what the designer expects.
>
> 翻译：外边距用来指定非浮动元素与其周围盒子边缘的最小距离。两个或两个以上的相邻的垂直外边距会被折叠并使用它们之间最大的那个外边距值。多数情况下，折叠垂直外边距可以在视觉上显得更美

> 观，也更贴近设计师的预期。

从这段话中，我们能获得一些有用的信息：

- 发生折叠需要是相邻的非浮动元素；
- 折叠发生在垂直外边距上，即margin-top/margin-bottom；
- 折叠后取其中最大的那个margin值作为最终值；

所以 `DEMO1` 中的 `2p` 符合折叠的条件，相邻且都不是浮动元素，于是它们就自然合体了。至于取最大的那个值作为折叠后的最终值，因为都是50px，所以无所谓谁谁谁了。

为什么会有margin折叠这样的设计？

从上文中，应该能知道个大概？在前面的文章中，我们说过，CSS的基本模型是排版。只是前端工程师现在做得更多的是 `布局` 而非 `文字排版`，但CSS并未界定这两者的区别。而 `margin` 折叠是为实现文本排版的段落间距而提供的特性。所以很多时候 `margin` 折叠的特性会带给我们诸多疑惑。

再回到 `DEMO1` 仔细看看，你会惊讶的发现，其实你只要设置每个 p 有相同的垂直外边距，由于发生会 margin 折叠，所有的 p 之间，及包含块与 p 之间的间隙都是相同的，非常美妙且实现简单，不是么？这正印证了我们引用 CSS1 中的那短话：多数情况下，折叠垂直外边距可以在视觉上显得更美观，也更贴近设计师的预期。

浮动元素真的不会发生margin折叠吗？

质疑精神一直都是进步最重要的驱动力之一，我们为什么不能呢？改下代码看看：

CSS:

```
p{
    float: left;
    margin: 50px;
}
```

只改CSS代码，HTML不变。迫不及待的想看到验证情况，来吧，还等什么。 `DEMO2` 验证浮动元素是否会发生margin折叠。

结果告诉我们，真的没有折叠， `2p` 间的间隙足足有 `100px`。

剩下的2点大家就自行验证吧，相信你能得到满意的额测试结果。

仅此而已？

回想一下我们在 margin系列之百分比 文中提到过受书写模式影响的一些特性，非常不幸， `margin` 折叠正好是其中之一。

是的，在CSS2及后续的规范中，将margin 折叠描述得更为详尽了。

在水平书写模式下，发生 `margin` 折叠的是垂直方向，即 `margin-top/margin-bottom` ，在垂直书写模式下， `margin` 折叠发生在水平方向上，即 `margin-right/margin-left` 。

现在我们来解释一下到底什么是margin折叠？

在CSS中，两个或以上的块元素（可能是兄弟，也可能不是）之间的相邻外边距可以被合并成一个单独的外边距。通过此方式合并的外边距被称为折叠，且产生的已合并的外边距被称为折叠外边距。

处于同一个块级上下文中的块元素，没有行框、没有间隙、没有内边距和边框隔开它们，这样的元素垂直边缘毗邻，则称之为相邻。

什么是垂直边缘毗邻？

- 元素的上外边距和其属于常规流中的第一个孩子的上外边距。
- 元素的下外边距和其属于常规流中的下一个兄弟的上外边距。
- 属于常规流中的最后一个孩子的下外边距和其父亲的下外边距，如果其父亲的高度计算值为 `auto` 。
- 元素的上、下外边距，如果该元素没有建立新的块级格式上下文，且 `min-height` 的计算值为零、 `height` 的计算值为零或 `auto` 、且没有属于常规流中的孩子。

说得很清楚了，我想是的。你可能需要注意的是发生 `margin` 折叠的元素不一定是兄弟关系，也能是父子或祖先的关系。

如何避免margin折叠？

我想肯定有人要问，那我不想有 `margin` 折叠的情况发生，该怎么办？其实从上面的规则中，我们已经可以抽出避免 `margin` 折叠的条件来。

- `margin` 折叠元素只发生在块元素上；
- 浮动元素不与其他元素 `margin` 折叠；
- 定义了属性overflow且值不为visible（即创建了新的块级格式化上下文）的块元素，不与它的子元素发生 `margin` 折叠；
- 绝对定位元素的 `margin` 不与任何 `margin` 发生折叠。
- 特殊：根元素的 `margin` 不与其它任何 `margin` 发生折叠；
- 如果常规流中的一个块元素没有 `border-top` 、 `padding-top` ，且其第一个浮动的块级子元素没有间隙，则该元素的上外边距会与其常规流中的第一个块级子元素的上外边距折叠。
  可能有些绕，我们验证一下 `DEMO3` ，在其第一个浮动子元素加个全角空格做间隙，来个反证 `DEMO4`
- 如果一个元素的 `min-height` 属性为0，且没有上或下边框以及上或下内边距，且 `height` 为0或者 `auto` ，且不包含行框，且其属于常规流的所有孩子的外边距都折叠了，则折叠其外边距。验证一下 `DEMO5`

这样干掉margin折叠

如果不想发生 margin 折叠，那么你可以根据上面的规则得到方法，不是么？我把它改成非块元素，让它浮动，让它绝对定位，让它 overflow:hidden DEMO6，用边框隔开它们 DEMO7...随你怎样，信手拈来。

今天状态不太好，有些地方写得欠妥，之后可能会修改一下。

BTW: 这篇文章里可能有不少之前文章中没出现过的名词，比如：块级上下文、行框、常规流，如果你对此不甚了解，可以先找资料看看，我以后会讲到。

enjoy it.

可参考：

- http://www.w3.org/TR/css3-box/#margins
- http://www.w3.org/TR/css3-box/#collapsing-margins
- http://dev.w3.org/csswg/css-box/#collapsing-margins
- http://www.w3.org/TR/CSS1/#vertical-formatting

margin系列之内秀篇

margin系列之内秀篇

> 原作者：doyoe
> 原文链接：http://blog.doyoe.com/2013/12/06/css/margin%E7%B3%BB%E5%88%97%E4%B9%8B%E5%86%85%E7%A7%80%E7%AF%87/

最Cool的利器

一样东西在不同的场景，不同的人手里，所能做的事会有很大不同。我深切的以为 margin 绝对是 CSS 中最有能力的利器之一，不知大家以为然否？

前面几篇文章大概的讲了一些关于 margin 的特性，所以本篇会聊聊 margin 的实际应用场景，也算让自己休息一下，不用再讲知识点。

有个很典型的需求

相信接下来这个需求，你十有八九实现过，甚至实现过多次，来看 图一：

我们看到这个图中，有个列表，每个列表项下面都有一条线，但最后一项没有。我们预期的代码是：

```
<div id="demo">
   <h3>标题列表</h3>
   <ul>
      <li>&raquo; 有点累想歇一下好长一个标题</li>
      <li>&raquo; 有点累想歇一下好长一个标题</li>
      <li>&raquo; 有点累想歇一下好长一个标题</li>
      <li>&raquo; 有点累想歇一下好长一个标题</li>
      <li>&raquo; 有点累想歇一下好长一个标题</li>
   </ul>
</div>
```

如果每项都有条底线，我们可以很简单的做到，如下：

```
#demo li{
   border-bottom: 1px solid #ccc;
}
```

然而为了处理最后一项，事情就变得有点纠结了。我知道肯定有人要说，我们有 `:first-child`，`:nth-last-child(n)`，`:nth-last-of-type(n)` 之类的CSS3选择符，要处理这个，太easy了。恩，我也不得不承认，CSS3确认让事情变得简单多了。但我们可能需要面对一些国情，因为需要照顾一些弱小者，比如IE6-8，它们离CSS3的世界太远。

传说中的first/last解决方案

所以我们需要找别的方法，于是这样的代码，相信你见过无数遍了：

```
<div id="demo">
    <h3>标题列表</h3>
    <ul>
        <li>&raquo; 有点累想歇一下好长一个标题</li>
        <li>&raquo; 有点累想歇一下好长一个标题</li>
        <li>&raquo; 有点累想歇一下好长一个标题</li>
        <li>&raquo; 有点累想歇一下好长一个标题</li>
        <li class="last">&raquo; 有点累想歇一下好长一个标题</li>
    </ul>
</div>
```

我没乱说，你肯定见到类似的代码千百遍了？是的，它确实能够解决我们的问题，请看 DEMO1 传说中的first/last解决方案，代码如下：

```
#demo .last{
    border-bottom: 0 none;
}
```

使用特殊的class来单独处理这项，但我不是很喜欢这样的code，原因大致有：

- 需单独定义一个差异化的class；
- 不利于数据循环输出，因为还得判断是否最后一项；

margin的神来之笔

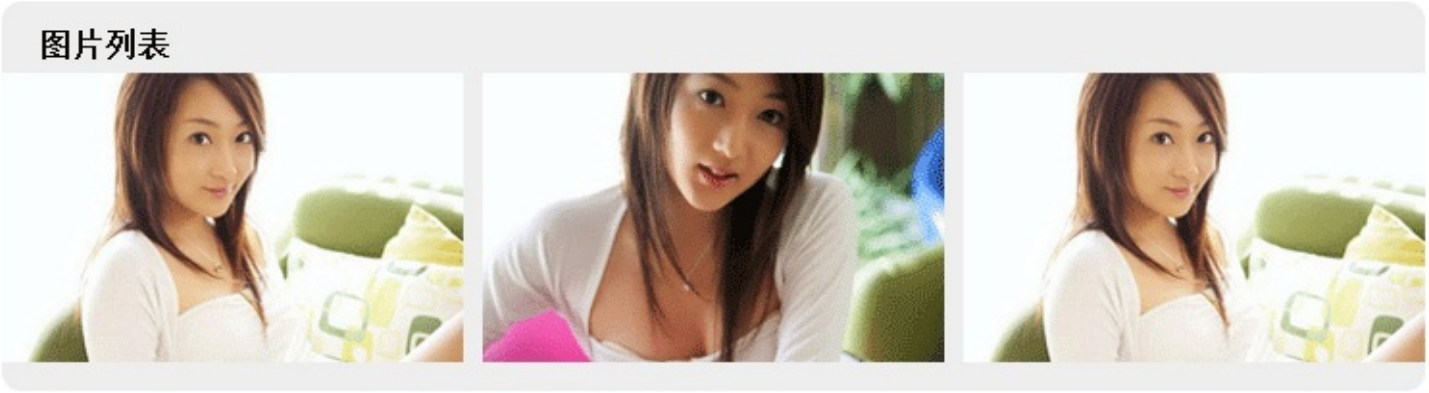
基于以上的原因，肯定会有其它的解决方案出现，这时margin无疑是非常不错的选择，来看代码：

```
#demo{
    overflow:hidden;
}
#demo ul{
    margin-bottom: -1px;
}
```

CSS代码如上，HTML代码当时使用开篇时的那段，结果请看：DEMO2 margin解决方案

是不是很Cool，完全避免了上述的问题，并且代码量很小。至于为什么可以这样实现，前几篇文章里有说过，margin是互动的，能影响其上下文的布局。本例中，当 ul `margin-bottom:-1px` ，其本身的高并不会被改变，但其相邻的元素则会往上 `1px` ，这时相邻的元素即其包含块 #demo，所以给 #demo `overflow:hidden` ，就直接将那 1px 的边框给裁剪掉了。
再来个相似的需求

看看下述的 图二 ，这应该也是一种非常常见的图片列表需求：



只关注图片之间的间隙，我们发现3张图片，却只有2个间隙。不论你是用 margin-left 或者说是 margin-right ，都无法直接达成这个需求。
当然，可以像 DEMO1 那样给第一个或者最后一个添加一个特殊类 first/last 来解决。但这种方式刚被说不喜欢，所以想想用 margin 方式吧，思路应该说是和 DEMO2 毫无二致。来看代码：

CSS

```
#demo{
   overflow:hidden;
}
#demo ul{
   margin-right:-10px;
}
```

HTML

```
<div id="demo">
   <h3>图片列表</h3>
   <ul>
      <li><img src="images/1.jpg" alt="" /></li>
      <li><img src="images/2.jpg" alt="" /></li>
      <li><img src="images/1.jpg" alt="" /></li>
   </ul>
</div>
```

恩，就这么简单，很美妙。效果可移步 DEMO3 margin处理图片列表间隙解决方案

我知道不少人还会使用给图片列表容器加宽度的方式来进行处理，当然，它很OK，不过不够灵活，因为在不同场景下，宽度可能不一样，这样的code无法被提取为公用样式，复用性不强。

而 margin 的方式完全不care几乎任何场景，都可以使用，因为在大多数情况，我们这样一个图片模块都是自适应宽度的，因为它会处于某个layout下，宽度完全取决于layout，所以其实在真实场景下 #demo 的 overflow 并不是必须的，也就是说 margin-right 的负值理论上可以预设成一个很大的值。

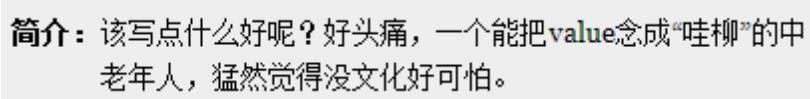
CSS

```
#demo ul{
   margin-right:-100px; /* 这个可以设置得比li的间隙更大，所以理论上可以写一次而适用于真实场景的任何情况 */
}
```

看我们简单还原的真实场景使用方式：`DEMO4` 模拟真实场景：margin处理图片列表间隙解决方案。恩，就这样，灵活性和可扩展性爆棚，不是么？

缩进实例

依然先贴个图，以下是 `图三` ：



貌似是个好常见的需求场景，当然，要实现这样的效果，对于大家来说都不过是信手拈来，再容易不过。

HTML

```
<div>
   <strong>简介：</strong>
   <p>该写点什么好呢？好头痛，一个能把value念成“哇柳”的中老年人，猛然觉得没文化好可怕。</p>
</div>
```

你可能随手就会写下 `float + margin/padding` ， `float + bfc` ， `absolute + margin/padding` ， `flex` 等方案中的随意一个，恩，都很Cool，我也常这么干。

只是有的时候在一个小场景下，希望能比较轻量的出来这样的缩进，可能不想有浮动，绝对定位，清除浮动之类的，怎么破？

margin依然是你很好的选择

你想到了吗？是的，用margin。

HTML

```
<p><strong>简介：</strong>该写点什么好呢？好头痛，一个能把value念成“哇柳”的中老年人，猛然觉得没文化好可怕。</p>
```

CSS

```
p{
    padding-left:45px;
}
strong{
    margin-left:-45px;
}
```

看起来很简单，没有浮动，没有绝对定位，没有其它重布局，很清凉有木有？

甚至 HTML 也可以更简单，因为无需对后面那长段做任何处理，所以不需要再加包裹。来看看具体例子吧。 DEMO5 margin缩进实例。我想这样的轻量方式，在一定时候还是有使用价值的，不是么？

视觉欺骗伪等高

等高布局在一段时间内好似挺火，方案也涌现过不少，如 图四 ：

该图要求，不论是主栏还是侧栏，总是以最高的那列为基准高度。核心代码：

CSS

```
#doc{
    overflow:hidden;
}
#main,#aside{
    margin-bottom:-999px;
    padding-bottom:999px;
}
```

HTML

```
<div id="doc">
    <div id="main">主内容栏<br />占位内容</div>
    <div id="aside">侧边栏</div>
</div>
```

先看看结果： DEMO6 margin伪等高布局

效果和我们的要求一致，达到了等高布局。需要提醒的是，这其实只是视觉欺骗，做到的了伪高等高。主栏和侧栏的实际高度其实并不相等，之所以可以达成这样的效果，其原因在于负 margin 值。我们前文中有提到过， margin 会影响其上下文布局，当我们将 margin-bottom 设置为负值时，其相邻的包含块元素，底部会自动上去其负值的高度，直到最高的那列底部边缘为止，然后裁剪。但该列本身的高度并不会发生变化，同时因为有 padding-bottom 向下扩展，颜色被填充满padding区域，于是达到视觉上的等高。

描述的貌似有点复杂，没文化好可怕。差不多就这样，不能接着往下写了，要不收不住。

作为 `CSS` 的重要属性 `margin` 有很多可被挖掘的潜力，需要更多的是想法。enjoy it.

margin系列之bug巡演

margin系列之bug巡演

> 原作者：doyoe
> 原文链接：http://blog.doyoe.com/2013/12/10/css/margin%E7%B3%BB%E5%88%97%E4%B9%8Bbug%E5%B7%A1%E6%BC%94/

我所知道的浏览器margin bug

- IE6浮动双倍margin bug；
- IE6浮动相邻元素3px bug；
- E6/7 clear引发的margin-top bug；
- 待补充的有一堆

为bug生为bug死为bug欲仙欲死的日子

各浏览器的实现差异或者由此而引入的错误，一直都是前端开发人员的梦魇。相信大多数的前端都为此而精疲力尽过，浏览器bug你所知有几？

IE6浮动双倍margin bug

这当是IE6最为经典的bug之一。高大上的前端，你肯定从未与其失之交臂过。

触发方式

- 元素被设置浮动
- 元素在与浮动一致的方向上设置margin值

来看看详细的代码吧：

HTML

```
<div id="demo">
   <p>IE6下浮动方向上的margin值将会双倍于其指定值</p>
</div>
```

CSS

```
#demo p{
   float:left;
   margin-left:10px;
}
```

效果对比

IE6下浮动方向上的margin值将会双倍于其指定值 （图一）

图一 是非IE6下的效果

IE6下浮动方向上的margin值将会双倍于其指定值 （图二）

图二 是IE6下的效果

从图一和图二的对比，我们肉眼就可以发现区别。是的，IE6下左边的外边距变成了 `margin-left` 指定值的2倍，而其它浏览器下正常，这就是经典的IE6浮动元素双倍边距bug。来看看具体的例子：`DEMO1` IE6浮动元素双倍margin bug重现

很开心告诉你，问题要比这还更复杂一些，接着往下看。

同个浮动方向的元素只有第一个元素会double margin

`double margin` 并不会发生在所有的浮动元素上，同个包含块内，在相同的浮动方向上，它只发生在第一个浮动元素上。

用代码说话：

HTML

```
<div id="demo">
   <p>第一个float:left</p>
   <p>第二个float:left</p>
   <p>第三个float:left</p>
</div>
```

CSS Code不变，加多2个浮动元素，再来看具体情况，有图有真相：

第一个float:left 第二个float:left 第三个float:left （图三）

看到图三结果一目了然，三个 float:left 的元素只有第一个元素才 double margin 了。用个例子来终结它： DEMO2 同个浮动方向的元素只有第一个元素会double margin

double margin只发生在float:left时？

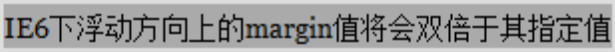
你觉得呢？结果当然不会是这样。在之前，我们只说过在同个浮动方向的第一个浮动元素会double margin，并没有说只有 float:left 才触发。

我们将 DEMO1 的CSS简单改改，HTML不变

CSS

```
#demo p{
    float:right;
    margin-right:10px;
}
```

结果会是怎样呢？看 图四 ：



（图四）

在图四中，我们看到右侧的外边距明显比指定值 margin-right:10px 要大，恩，确实，它是20px，也double了。瞧瞧： DEMO3 IE6 double margin也会发生在float:right时

既有左浮动又有右浮动的情况将会是怎样呢？

我们先来将代码呈上：

HTML

```
<div id="demo">
    <p class="a">1 float:left</p>
    <p class="b">2 float:left</p>
    <p class="c">3 float:right</p>
    <p class="d">4 float:right</p>
</div>
```

CSS

```
#demo .a,#demo .b{
    float:left;
    margin-left:10px;
}
#demo .c,#demo .d{
    float:right;
    margin-right:10px;
}
```

是的，你可能想到了，第一个左浮动元素和第一个右浮动元素都将会出现 double margin 。来看 图五：

（图五）

左右都 double margin 了，这看似挺复杂，其实为什么会这样，前面都讲得比较明白了，所以应该能理解？本例也奉上： DEMO4 复杂的double margin

double margin 不仅仅出现在margin-left/right

和大多数其它 margin 特性一样， double margin 也受书写模式 writing-mode 影响。我们在开篇所说的触发条件之一 元素在与浮动一致的方向设置margin值 ，其实并不完全精确。当 writing-mode 为纵向时，会发生 double margin 的方向也相应变成了纵向。
当书写模式 writing-mode 纵向时，设置 float:right 时，会发生什么？来看代码：

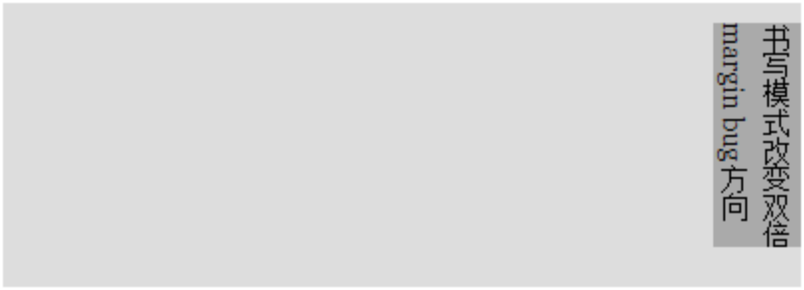
HTML

```
<div id="demo">
    <p>书写模式改变双倍margin bug方向</p>
</div>
```

CSS

```
#demo{
    -webkit-writing-mode:vertical-rl;
    writing-mode:tb-rl;
}
#demo p{
    float:right;
    margin:10px 0;
}
```

CSS Code中，我们同时设置了 margin-top/bottom 的值都为 10px。你预期会 double 的方向是 top or bottom ？不太确定？看到 图六 你就知道了：



（图六）

图六清晰的验证了 writing-mode 会影响 double margin 的方向；并且当设置了 float:right 时，只有 margin-bottom 会 double 。看看示例吧： DEMO5 书写模式改变IE6浮动双倍margin bug方向

float:left 时， double margin 的将会是 top or bottom ?

大家再猜猜，在书写模式为纵向时，设置了 float:left，结果又将会如何？
我们只简单的将 DEMO5 中的CSS改成 float:left 其余不变，于是得到 图七 如下：

你会惊讶的发现， margin-top/botto m 两个方向都出现了 double，这真是一件好神奇的事，事实胜于雄辩： DEMO6 书写模式纵向时margin-top/bottom都将double

写到这，关于IE6浮动双倍margin bug就说的差不多了，包括触发方式，各种情景下的变化，还有解决方案。哦，解决方案貌似还没写...

fix IE6浮动双倍margin bug

我们以 DEMO1 作为需要fix的case
给IE6在会 double margin 的方向上设置小一倍的margin值，如下：

CSS

```
#demo p{
    float:left;
    margin-left:10px;
    _margin-left:5px;
}
```

恩，IE6的hack，就不再赘述了。不过这种处理方式有一个明显的缺陷，那就是不够灵活，无法通用。因为

当标准 margin 值改变时，这个值就得变化。所以不推荐使用这种方式。

display:inline

CSS

```
#demo p{
    _display:inline;
    float:left;
    margin-left:10px;
}
```

恩，仍然是only ie6的hack，不过这个方案更Cool，它不需要care margin值到底是什么，足够灵活。看具体的例子吧： DEMO7 修复IE6浮动双倍margin bug。至于为什么会有这种解法，我想只能问问微软的童鞋了。

完全没想到，单一个双边距bug就写了这么长的篇幅，本打算一篇文章涵盖一堆bug，看来得分篇了。

margin系列之内秀篇（二）

margin系列之内秀篇（二）

> 原作者：doyoe
> 原文链接：http://blog.doyoe.com/2013/12/14/css/margin%E7%B3%BB%E5%88%97%E4%B9%8B%E5%86%85%E7%A7%80%E7%AF%87%EF%BC%88%E4%BA%8C%EF%BC%89/

可挖掘性

之前已经写过一篇关于 margin 应用场景的文章：margin系列之内秀篇，当然，它的应用场景会远大于文中所述，无法一一列举。

所以本篇权当是对此的补遗好了，各位客官如有比较Cool的想法都可以留言给我，我会视情况补丁进来。

1像素圆角

这有什么好聊的吗？ border-radius 瞬间可将之秒杀。恩，有的时候你不得不承认CSS3真是一把大杀器。不过当年我们是怎么做的？（会暴露年龄么？）

先看看我们要做什么, 图一 ：

（图一）

如上图一，我们会这样写：

HTML

```
<div id="demo">
    <a href="?"><span>1px圆角</span></a>
    <a href="?"><span>确定</span></a>
    <a href="?"><span>取消</span></a>
</div>
```

CSS

```
#demo a,#demo span{
    display:inline-block;
    vertical-align:top;
}
#demo span{
    margin:1px -1px; /* 关键规则 */
}
```

一条CSS规则我们就可以实现1px圆角，你信吗？来看 DEMO1 ：margin实现1px圆角

看到DEMO1的结果后，你会发现我们确实做到了1px圆角，很简单，有木有？在没有 border-radius 的年代，我们也很欢乐。

看到Code后，我想应该不用太解释为什么可以实现？
BTW，多像素圆角也可以参考这种方式来实现，如果你实在不想用图片的话。

已知宽高元素水平垂直居中

必须说，这是一个非常典型的 margin 应用，虽然如今看起来貌似使用场景不是太大，但还是好多人喜欢在面试时对人问起，我偶尔也会，但不多。

假设一个宽300px，高300px的盒子要在整个页面中水平垂直居中，我们可以这样做：

HTML

```
<div id="demo">这是一个水平垂直居中的容器</div>
```

CSS

```
#demo{
    position:absolute;
    top:50%;
    left:50%;
    width:300px;
    height:300px;
    margin-top:-150px;
    margin-left:-150px;
}
```

恩，是的，借助绝对定位。我们先来看看 DEMO2 ：margin实现已知宽高元素水平垂直居中

先通过 top/left 将 #demo 的绝对定位流起始位置设置为当前屏的中心点，此时 #demo 的左上角这个点其实已经是相对于页面水平垂直居中了，由于它的宽度和高度都是300px，所以从起始位置向右下各延伸300px后才是整个 #demo 的真正位置。此时整个 #demo 其实并不是水平垂直居中的，除非我们将 #demo 的中心点放在当前屏的中心点上。怎么做？

这时我们就需要使用 margin 了，在 margin系列之与相对偏移的异同 这篇文章里，我们就说过 margin 是以自身作为参照物进行位置偏移的。所以我们只需要将 #demo 相对自身向上偏移150px，向左偏移150px，就能够实现将自身的中心点放在当前屏的中心点上，也就实现了自身在当前屏的水平垂直居中。

为什么代码里是 -150px ？好吧，如果用 margin-top 来实现向上偏移，必须是负值，不是么？如果是正值的话，就是向下偏移了，其实也相当于是 margin-bottom 的正值， margin-left 亦然，我们在 margin系列之与相对偏移的异同 文章最后同样说过这个。基础知识很重要，有木有？

tabstrip底边线重合

先上个需求，如 图二 ：

（图二）

看到 图二 ，我想大家可能知道可能知道要做什么了。

对，我们要做的就是 tab 项与底边线重合，这应该是我们常见的场景了， margin 仍然是最佳选择。先来看代码：

HTML

```
<div id="demo">
    <a href="?">分类一</a>
    <a href="?" class="on">分类二</a>
    <a href="?">分类三</a>
    <a href="?">分类四</a>
</div>
```

CSS

```
#demo{
    border-bottom:1px solid #aaa;
}
#demo a{
    display:inline-block;
    margin-bottom:-1px;
    border:1px solid #aaa;
}
#demo .on{
    border-bottom-color:#fff;
}
```

要实现 `tab` 中各项与包含块的底边线重合，重点在于将包含块的底边线向上偏移1px，这样就与 `tab` 各项的底部重合在一起。

怎样可以做到让包含块底边线向上偏移1px？恩，`margin` 是那么的顺其自然。我们只需要将 `tab` 各项的 `margin-bottom` 设置为 `-1px` 即可，其本身高度不变，但包含块底部会向上1px。

来看看具体实现的例子 `DEMO3` ：tabstrip底边线重合

双重边线

实际场景可能比这会稍复杂一些，我们看个大概即可，主要是拓宽一下思路，来看 `图三` ：

（图三）

从图三中，我们可以看到每行都会有一个双色的边线，这就是我们要做的，HTML代码大约是这样：

HTML

```
<div id="demo">
    <ul>
        <li>这是一个双重边线的示例</li>
        <li>这是一个双重边线的示例</li>
        <li>这是一个双重边线的示例</li>
        <li>这是一个双重边线的示例</li>
    </ul>
</div>
```

怎么做？恩，我们可以用常规的方式来解决，比如完全使用 `border` ：

CSS Case1

```
#demo li{
    border-top:1px solid #fff;
    border-bottom:1px solid #ccc;
}
```

结果出来后，我们会发现最顶部多出了一条线，同时最底部又少了一条线。当然，这都可以被解决，我们可以让 `ul` 来辅助完成，例如让其 负margin-top + border-bottom，不过如果 `ul` 或者其父元素有垂直方向 `padding` 的话，处理起来可能会稍显蛋疼。
还有其他解？当然，会有的，来看：

CSS Case2

```
#demo ul{
    overflow:hidden;
    background:#fff;
}
#demo li{
    margin-bottom:1px;
    border-bottom:1px solid #ccc;
    background:#eee;
}
```

是的，选择 `margin` 作为实现手段。以 `ul` 的底色配合 `margin` 模拟出线条的外观，这其实也挺讨人喜欢的，不是么？看具体实现 `DEMO4` ：双重边线

`margin` 模拟边线还可以做什么？比如做个表格神马的，看看 `DEMO5` ：margin模拟表格边线

margin系列之bug巡演（二）

margin系列之bug巡演（二）

> 原作者：doyoe
> 原文链接：http://blog.doyoe.com/2013/12/17/css/margin%E7%B3%BB%E5%88%97%E4%B9%8Bbug%E5%B7%A1%E6%BC%94%EF%BC%88%E4%BA%8C%EF%BC%89/

IE6/7 clear引发的margin-top bug

我知道，这是一个被谈及较少的bug，但我几乎可以肯定你在遇见过的同时并没有把它当成是一个bug。

w3c关于 clear 特性的描述

设置了 `clear` 为非 `none` 值的元素，其顶部 `border` 边界在垂直方向不允许出现在之前的浮动元素底部 `margin` 之上。

什么意思呢？用段代码来阐述：

HTML

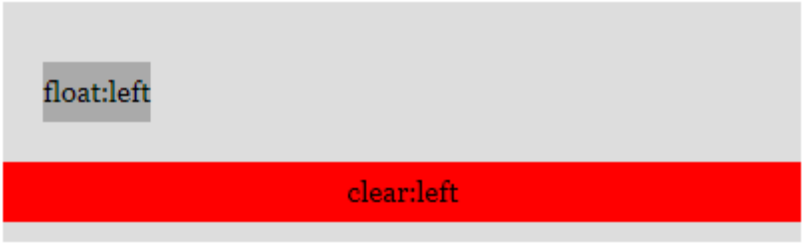
```
<div class="a">float:left</div>
<div class="b">clear:left</div>
```

CSS

```
.a{
    float:left;
    height:30px;
    margin:20px;
}
.b{
    clear:left;
    height:30px;
    margin-top:-30px;
}
```

如上代码，你认为 `.b` 是否会相对自身向上偏移 `30px` 呢？然后盖住 `.a` 底部 10px？如果你真这么猜想，那就错了。

来看上述代码，我们会得到什么样的结果，如 `图一`：



（图一）

恩，你觉得这可能会是落后浏览器才这样，比如IE6/7。很高兴的告诉你，其实高级浏览器才这样，IE6/7的表现会是之前你猜想的那样，如下 `图二`：

（图二）

不论你相信与否，看个例子你就明白了 DEMO1 ：clear margin 猜想，你会发现就算将 margin-top 去掉， .b 的位置也丝毫不会改变。

为什么会这样？

我们已经说过设置了 clear 为非 none 值的元素其顶部 border 边界不允许出现在之前浮动元素的底部margin边界之上。也就是说必须在垂直方向上递次堆叠却不能重合。

所以说高级浏览器是遵循w3c规范的，而IE6/7的实现明显有悖该规则。

`
虽然拥有 clear 特性的元素其 border`
顶部边界不允许超越之前浮动元素的底部margin边界之上，但是其margin可以和之前浮动元素的任何区域重合。我们稍微改下之前代码：

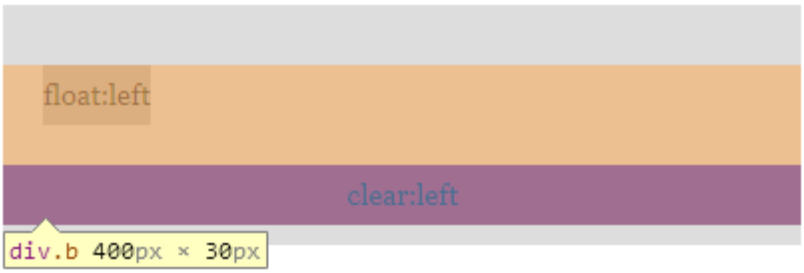
HTML

```
<div class="a">float:left</div>
<div class="b">clear:left</div>
```

CSS

```
.a{
    float:left;
    height:30px;
    margin:20px;
}
.b{
    clear:left;
    height:30px;
    margin-top:50px;
}
```

我们将 .b 的 margin-top 修改为一个正值，能得到如下 图三 ：



（图三）

图中的黄色区域是 .b 的 margin-top ，你会发现，它可以和 .a 的任何区域重合。同时，你也可以发现 图三 和 图一 居然是一样的效果， .b 的实际位置都没变化过。来看例子 DEMO2 ：clear margin 验证

这是否说明拥有 clear 特性的元素对其之前的浮动元素没有任何影响？因为不论是正值还是负值，其位置

都不会发生变化。

如果你这样想，那你就又错了。

拥有clear特性的元素其margin紧邻之前的浮动元素依然有效

是的，在某个临界值，这一切会发生改变，并非全然无效。
这个临界值是什么？
临界值是包含块内浮动元素的实际高度，即浮动元素的 margin + border + padding + height，拿我们的 DEMO2 来详述：
DEMO2 中的浮动元素 .a 的实际高度为 30 + 20*2 = 70px，也就说当 .b 的 margin-top 大于 70px 时，超过的部分将会使得 .b 发生偏移。

CSS

```
.b{
    clear:left;
    height:30px;
    margin-top:100px;
}
```

我们将 DEMO2 中的 margin-top 改成 100px，再看看具体情况 DEMO3：clear margin 验证2，你可以手动的修改其 margin-top 值，看看临界值是否如前所述。

解决方案

IE6/7下由 clear 特性引发的 margin-top bug，并没有像 double margin 那样的万精油 display:inline 解决方案，所以需要寻求的是让IE6/7和其它浏览器绕过此问题来进行解决。

例如：

- 尽量避免为设置了 clear 为非 none 值的元素定义margin-top；
- 如果必须，可以将拥有 clear 特性的元素作为容器，在其子元素上设置margin-top；
- 视情况换成padding-top；

要注意的问题

.a 和 .b 需要在处在同一个块级上下文内，或者其包含块拥有 padding-top/border-top，否则临界值情况将失效，不过任何IE目前都不需要此前置条件。用IE和非IE查看 DEMO4：clear margin 验证3

margin系列之bug巡演（三）

margin系列之bug巡演（三）

> 原作者：doyoe
> 原文链接：
> http://blog.doyoe.com/2013/12/20/css/margin%E7%B3%BB%E5%88%97%E4%B9%8Bbug%E5%B7%A1%E6%BC%94%EF%BC%88%E4%B8%89%EF%BC%89/

IE8按钮margin auto居中失效Bug

你会猛然觉得，这是正解啊，因为 `button` 或者 `input type button类型` 的元素是 `inline-level` 的。

不对啊，`button` 应该是 `inline` 的吧？哦，可能是 `inline-block` ？

在这之前，我们似乎要先明确一些基础知识。

什么是 inline-level 元素？

要知道 `inline-level` 元素并不等于 `inline` 元素，也就是说 `行内级元素` 与 `行内元素` 是两个不同的概念。

`inline-level` 元素包含 `display` 值为：

- inline
- inline-block
- inline-table
- inline-flex
- other inline-*

以上情况时，元素可被称之为 `inline-level` 元素，但不都是 `inline` 元素。

什么是 block-level 元素？

`block-level` 指的是 `display` 值为 `block` 的元素吗？我知道不少人一直有这样的认知，不过这不完全准确。

`block-level` 元素包含 `display` 值为：

- block
- list-item
- table
- table-*
- flex

- 如果position既不是static也不是relative、float不是none或者元素是根元素，当display:inline-table时，display的计算值为table；当display值为 inline | inline-block | run-in | table-* 时，display的计算值为block

有如上情况时的元素均被称之为 `block-level` 元素。同时 `block-level` 和 `block` 也不是同一个概念，所以如果你认为 `display` 值为 `list-item` 的 `li` 不是 `块级元素`，那就错了。

看到这里，你对 `块级元素` ， `块元素` ， `行内级元素` ， `行内元素`
这个4个概念，应该已经有了比较清晰的了解？

margin keyword auto只能应用在常规流中的 block-level 元素上

- 当一个块级元素定义了 `position` 值为非 `static` 和 `relative` 之外的值时，margin-right/left auto的计算值为0；
- 当一个块级元素定义了 `float` 值为非 `none` 之外的值时，margin-right/left auto 的计算值为0；
- 非块级元素的margin-right/left auto 的计算值为0；

计算值为0，即说明其应用使用值的意图失败。所以在有如上情形的场景中，都无法使用 `auto` 来实现水平居中。同时也说明了，只有 `normal flow` 的 `block-level` 才能应用 margin keyword auto。

margin可以应用于所有元素吗？

这显然不行。准确的说：margin可以应用在除某些table-*元素和某些行内元素之外的所有元素上。

和margin亲近的table-*系元素

- table
- inline-table
- table-caption

除了 `display` 值为以上3种之外的 table系 元素，都不能应用 `margin` ，比如：th, td。

和margin亲近的 inline-level 元素

我之前面试的时候常会问候选人，行内元素不能设置宽高对吗？大部分人会告诉我说是；然后我又会问，那为什么 `img` 元素可以设置宽高呢？有人会告诉我，因为 `img` 是个特殊的元素？接着我又会问题，`img` 是如何特殊的？然后，然后就没然后了，因为没声音了。

恩， `img` 确实是个特殊的元素。它特殊在哪里？它的特殊就在于它是一个行内置换元素。

所有的置换元素都可以设置 `margin` 属性，并且可以设置宽高，这就是为什么 `img` 是行内元素却可以设置 `width` 和 `height` 。

什么是置换元素（Replaced elements）？

一个元素拥有内在的二维属性，其宽高属性受外部资源影响，默认拥有CSS格式，这样的元素被称为置换元素。

意思就是说置换元素的宽高不完全由CSS决定，还受其自身内容和外部资源所影响。

举个例子来说，仍然说 `img` 元素吧，你会发现，如果你 `src` 进来不同尺寸的资源，那么在 `viewport` 上显示的图片宽高也是不同的，也就是说 `img` 元素的宽高会受外部资源影响。

再说说 `input` 元素，随便在页面上扔一个input，你都能发现它拥有一个默认的宽高，这就是它所具有的内在二维宽高属性，并且该类元素会受UA影响，不同UA下所呈现外观会有不同。

常见的置换元素有哪些？

img, object, button, input, textarea, select等

行内非置换元素真的不能应用margin吗？

什么是非置换元素？除了置换元素之外的元素，我想将这样的元素称之为非置换元素是没有大碍的。

那么行内非置换元素真的无法设置 margin 吗？我想在工作中你一定碰到过很多这样的场景，给一个 `a` 或者 `span` 定义间隙。这时我们写：

CSS

```
span{margin:5px 10px;}
```

结果发现 `span` 的水平方向上的 `margin` 定义生效了，但垂直方向上的 `margin` 定义却没被应用。

是的，这就是行内非置换元素使用 `margin` 时的表征，所以对各种特性的理解，在让自己的代码更有效上是大有裨益的。

回归正题

我们本来是想说IE8按钮margin auto居中失效Bug的，扯了不少题外话。
我们知道 `margin keyword auto` 不能应用在处于常规流中的 `block-level` 之外的元素上，所以我有这样的一段代码：

CSS

```
button{display:block;margin:auto;}
```

HTML

```
<div id="demo">
   <button>按钮</button>
</div>
```

恩，我们将 button 显式的转换为了 block ，同时我们知道 button 作为置换元素，本身具备内在宽高，也就是说这时，我只需要加上 margin:auto ，该按钮就应该在其包含块里水平居中。
是的，所有浏览器都和预期是一样，实现了水平居中，但是却出现了奇葩的IE8，完全无效，甚至不如原始社会的IE6。来看看示例 DEMO1 ：IE8按钮margin auto居中失效Bug

通过以上例子，你有没有突然感觉到，如果要让一个置换元素在包含块中水平居中，出乎预料的简单，只需要 display:block;margin:auto; 即可。

注意事项

令人意外的是，只有 button 和 input type 为 button 相关元素的时候，在IE8中才会水平居中失效；如: input type text 或 img 时， margin keyword auto 运作正常。

解决方案

- 给其显示的定义宽度
- 不改变其display值，包含块text-align:center
- 其它水平居中方案，如：absolute + 负margin

margin系列之布局篇

margin系列之布局篇

> 原作者：doyoe
> 原文链接：http://blog.doyoe.com/2013/12/31/css/margin%E7%B3%BB%E5%88%97%E4%B9%8B%E5%B8%83%E5%B1%80%E7%AF%87/

前端工程师对CSS的基本诉求

布局能力或许是Web前端工程师对CSS的最基本的诉求，当开始进入到这个岗位，就避免不了要和CSS打交道，而和CSS交往，布局当然是不可或缺的。

很遗憾的是，CSS2.1之前都没有出现真正意义上的布局属性，直至现如今的CSS3，才开始出现了一些，如：flex, grid 等，不过其兼容性及国内浏览器的使用情况，真令人捉急。

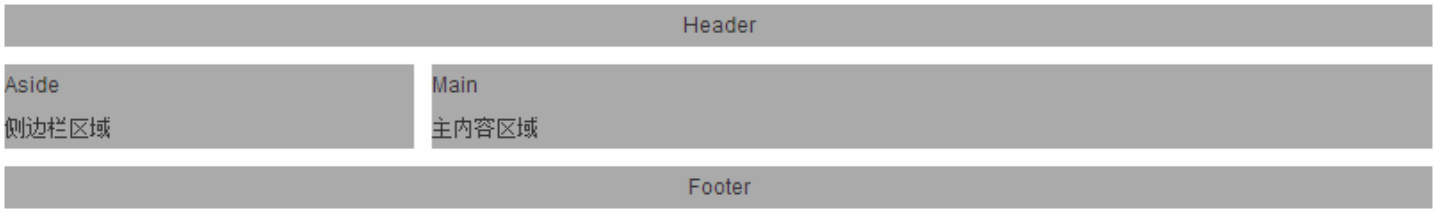
不过，有需求就会有变通，对于达成布局目的，已衍生出各式各样的方法，如：float, inline-block, table, absolute 等等。

margin的布局之道

其实，这个话题有点脱离 `margin` 的能力范围，因为单纯的 `margin` 并无法完成复杂布局，它更多做的是辅助，但却又难以替代。

经典左右结构

两栏结构应该是最常见和经典的网页呈现之一吧？如下 `图一` ：



（图一）

相信对于这样一个网页呈现，你不会陌生。那么你有多少种方案可以达成该布局？我想，4至5种应该是保守估计吧？
这次，我们只看 `margin` 是如何做的。

absolute + margin 方式

HTML

```
<header id="hd">头部</header>
<div id="bd">
    <aside id="aside">侧边栏固定宽度</aside>
    <div id="main">主内容栏自适应宽度</div>
</div>
<footer id="ft">底部</footer>
```

CSS

```
#aside{
    position:absolute;
    top:0;
    left:0;
    width:200px;
}
#main{
    margin-left:210px;
}
```

如上关键代码，我们即可实现 图一 布局，该布局有一个特点就是， #main 可以自适应可用空间。

假定 HTML 是给定的，我们来解读一下 CSS 代码：
我们知道块级元素的特性之一是换新行，也就是说，如果想让 #main 和 #aside 在同行显示，我们要么改变其显示属性为 inline-level（即之前说的inline-block布局方式），要么改变其流方式（absolute, float, flex and etc...）。

如上述代码，我们使用了 absolute，即让 '#aside' 脱离常规流，通过绝对定位到想要的位置。

主内容栏自适应宽度

同时你会发现，我们并有改变 #main 的显示属性或者流方式，也就是说其仍然具备块级元素的特性，所以它会自动适应剩余宽度，即我们常说的自适应宽度。

我们并不希望 #main 区域会包含 #aside 在内，于是利用 margin 给 '#aside' 预留出足够其显示的空间，即可达成我们所要的布局。

可能你会问为什么是 margin-left:210px 而不是 200px，实际确实应该是 200px，多出来的 10px 只是为了创建一个列间隙，与布局实现无关。
来看看具体的实现 DEMO1：margin+absolute布局：左栏固定主内容自适应
就这样，是不是很简单？其实它还有亮点，那就是：

任意调整列顺序

在不修改 HTML 的情况下，只需简单的修改 CSS，我们即可让左右两栏的顺序调换，来看代码：

CSS

```
#aside{
    position:absolute;
    top:0;
    right:0;
    width:200px;
}
#main{
    margin-right:210px;
}
```

其实现原理没变，同样看看 DEMO2：margin+absolute布局：右栏固定主内容自适应

主内容优先显示

可以更Cool一点，你觉得呢？很多时候，你也许会考虑到，不论在何种情况下，总想保证主要的内容优先于次要的内容呈现给用户，那么，怎么做？

很简单，只需要将主要内容的HTML排在次要内容的HTML之前即可，因为它是顺序加载渲染的。我们可以

这样：

HTML

```
<header id="hd">头部</header>
<div id="bd">
   <div id="main">主内容栏自适应宽度</div>
   <aside id="aside">侧边栏固定宽度</aside>
</div>
<footer id="ft">底部</footer>
```

是的，我们只需要将 `#main` 的HTML挪到 `#aside` 的HTML前面，令人兴奋的是，改变HTML之后，CSS不需要做任何改变。我们来看 `DEMO3`：margin+absolute布局：左栏固定主内容自适应，主内容有限显示

当然，调正列顺序的 `DEMO4`：margin+absolute布局：右栏固定主内容自适应，主内容有限显示 也同样简单，我们只需要写HTML时注意一下即可。

致命缺陷

列举了 `absolute+margin` 布局的很多优点，但只说一个问题，就足以让你在是否选用这种方式时深思熟虑，是什么呢？

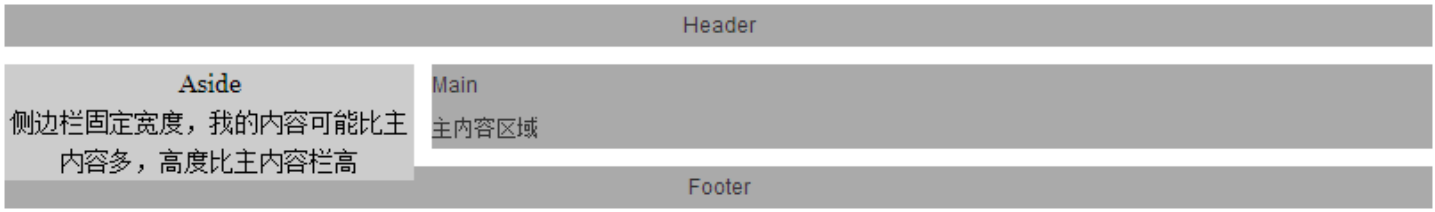
我们知道 `absolute` 是定位流，脱离正常排版，也就是说绝对定位元素不影响其上下文的排版方式，你意识到我想说什么了么？

OK，用代码来演示：

HTML

```
<header id="hd">头部</header>
<div id="bd">
   <div id="main">主内容栏自适应宽度</div>
   <aside id="aside">侧边栏固定宽度，我的内容可能比主内容多，高度比主内容栏高</aside>
</div>
<footer id="ft">底部</footer>
```

看完代码，估计你猜到了。是的，`#aside` 无法撑开父元素的高度，它将会溢出父元素区域，结果如下图：



（图二）

来看看这缺陷所导致的情况 DEMO5：margin+absolute布局的致命缺陷

此时假设你设置父元素 overflow:hidden 那么溢出部分将会被裁减，同样不符合布局意图，无法可破。所以在内容量不可控的场景，不推荐使用这种方式。

float + margin 方式

和 absolute + margin 方式一样， float + margin 方式一样是经典的利用来布局的方案，并且被更广泛使用。我们仍然以 图一 为例，来看代码：

HTML

```
<header id="hd">头部</header>
<div id="bd">
    <aside id="aside">侧边栏固定宽度</aside>
    <div id="main">主内容栏自适应宽度</div>
</div>
<footer id="ft">底部</footer>
```

CSS

```
#aside{
    float:left;
    width:200px;
}
#main{
    margin-left:210px;
}
```

如上述代码，我们使用了 float ，即从图文环绕形态演变而来。当 #aside 定义了 float ，那么紧随其后的元素将会环绕在其周围。不过环绕并不是我们想要的结果，我们想要的是 ‘#main’ 也自成封闭矩形，所以利用 margin 留出足够 #aside 显示的空间，中断环绕即可。

当然，此时 #main 也是自适应宽度的，来看具体实例 DEMO6：margin+float布局：左栏固定主内容自适应

它是否也具备可任意调整列顺序的特点？何不一试？

CSS

```
#aside{
    float:right;
    width:200px;
}
#main{
    margin-right:210px;
}
```

看过 DEMO7：margin+float布局：右栏固定主内容自适应，你会发现，是的，这种方式也支持任意调整列顺序，很棒。

从这种趋势看来，貌似 float + margin 的方式会成为黑马，不过遗憾的告诉你，这种方式无法支持主内容优先显示。但我们有更Cool的解决方案。

float + 负margin 方式

接下来我要说的大家可能都猜到了，对，经典的圣杯布局。至于圣杯的名字由来，大家可以自行Google，这里不做赘述。

恩，HTML当然是使用主内容优先显示的那种：

HTML

```
<header id="hd">头部</header>
<div id="bd">
    <div id="main">主内容栏自适应宽度</div>
    <aside id="aside">侧边栏固定宽度</aside>
</div>
<footer id="ft">底部</footer>
```

CSS

```
#bd{
    padding-left:210px;
}
#aside{
    float:left;
    position:relative;
    left:-210px;
    width:200px;
    margin-left:-100%;
}
#main{
    float:left;
    width:100%;
}
```

如上代码，既是圣杯布局的核心Code，如果你看懂了，你会发现，这其实很简单，不是么？

简单解释一下上面的CSS Code，首先我们是在做一个左侧固定宽度，右侧自适应宽度的布局。我们说过要让块级元素在同行显示的条件：改变显示方式，改变流方式，这里我们选择了使用 `float` 来将 `#main` 和 `#aside` 变成浮动流。

OK，这时我们具备 `#main` 和 `#aside` 能在同行显示的前置条件。我们知道，浮动元素其宽度如果没有显式定义，则由其内容决定。正好， `#aside` 是定宽的，所以显示给它定义 `width:200px` ，但此时 `#main` 该怎么办？不设置 `width` 不对，因为宽度将被内容左右，设置 `width:100%` 也不对，因为这样的话，就没有 `#aside` 的立足之地了，正确的应该是 `width: calc(100% - 200px)` ，不是么？可惜，这是新特性，只好作罢。

变通？是的，有的时候稍微换个思路，你会觉得豁然开朗。

`#main` 不是要自适应吗？那就给它个 `100%` ，怎么做？我们在包含块 `#bd` 中就将 `#aside` 的宽度刨除，宽度全部都给 `#main` 。恩，我们只需要这样 `#bd{padding-left:210px;}` （10px仍然是用来做间隙的），这时 `#main` 就可以设置 `width:100%` 了，由于 `#bd` 设置了 `padding` ，所以已在左边预留出了一块宽 `210px` 的区域。此时的问题在于如果将 `#aside` 挪到这个地方，你想对了，我们是在聊 负margin 布局，自然需要利用上。

`#aside{margin-left:-100%;}` 这样可以了吗？很明显，这样还不行，此时 `#aside` 和 `#main` 的起始位置将会重合，因为 `#aside` 的 `margin-left` 计算值是相对包含块来计算的，而此时包含块的宽度等于 `#main` 的宽度。

如何让 `#aside` 再向左偏移 `210px` ？显然 `margin` 是不行了，因为我们已经用掉它了。如果你看过之前的文章的话，你可能还记得，有一篇文章讲 margin系列之与相对偏移的异同。恩，是的，这时我们可以借助相对偏移。

向左偏移 `210px` 是件很简单的事： `#aside{position:relative;left:-210px;}` 。

至此，你的布局OK了，这就是圣杯的实现方式。来看已实现好的示例 `DEMO8` ：圣杯：左栏固定主内容自适应

当然，圣杯布局必须可以任意调整列顺序，要不，怎么能说是更Cool些的方案呢？

CSS

```
#bd{
    padding-right:210px;
}
#aside{
    float:left;
    position:relative;
    right:-210px;
    width:200px;
    margin-left:-200px;
}
#main{
    float:left;
    width:100%;
}
```

这个就直接看示例好了，不再一一解释代码 DEMO9：圣杯：右栏固定主内容自适应

所以圣杯布局具备前两种方式共同的优点，同时没有他们的不足，但圣杯本身也有一些问题，在IE6/7下报废，不过不用慌，因为它可被修复。

你想到方法了吗？

你需要了解的z-index世界

你需要了解的z-index世界

> 原作者：doyoe
> 原文链接：http://blog.doyoe.com/2014/01/21/css/%E4%BD%A0%E9%9C%80%E8%A6%81%E4%BA%86%E8%A7%A3%E7%9A%84z-index%E4%B8%96%E7%95%8C/

z-index的重要性

在我看来，z-index 给了我们日常工作中以极大的帮助，我们用它来定义元素的层叠级别（stack level）。受益于它，你能做Popup, DropDown, Tips, 图文替换等等。

在开始本篇之前，或许我们要先了解一下关于z-index的基本信息。

W3C这样描述

每个元素都具有三维空间位置，除了水平和垂直位置外，还能在 “Z轴” 上层层相叠、排列。元素在 “Z轴” 方向上的呈现顺序，由层叠上下文和层叠级别决定。

在文档中，每个元素仅属于一个层叠上下文。元素的层叠级别为整型，它描述了在相同层叠上下文中元素在 “Z轴” 上的呈现顺序。

同一层叠上下文中，层叠级别大的显示在上，层叠级别小的显示在下，相同层叠级别时，遵循后来居上的原则，即其在HTML文档中的顺序。

不同层叠上下文中，元素呈现顺序以父级层叠上下文的层叠级别来决定呈现的先后顺序，与自身的层叠级别无关。

z-index语法和应用

```
z-index: auto | <integer>
```

`z-index` 接受的属性值为：关键字auto和整数，整数可以是负值（Firefox2.0及之前不支持负值）。

需要注意的是 `z-index` 虽然很给力，却只能应用于定位元素（即设置了 `position` 属性为非 `static` 值），其它情况下， `z-index` 将被忽略。

对于定位元素而言， `z-index` 意味着：

- 确定该元素在当前层叠上下文中的层叠级别。
- 确定该元素是否创建了一个新的局部层叠上下文。

创建层叠上下文

在规范中说明：当某个元素的 `z-index` 未显式定义或者被指定为 `auto` 时，该元素不会产生新的局部层叠上下文。也就是说它可以和兄弟，祖先，后辈元素处在同一个堆叠上下文中，它们被放在一起比较层叠级别，儿子可以盖住祖先，父亲也可以盖住儿子，儿子甚至可以越过祖先，盖住祖先的兄弟，在层叠上下文中，它们是并级的关系。来看这样一个例子 `DEMO1`：z-index与创建层叠上下文

值得高兴的是，大部分浏览器都实现了这个特性；不过在IE6/7下，不论 `z-index` 值是否被显式定义，都将产生新的局部层叠上下文，也就是说子元素不可以越过是定位元素的父亲，子元素都处在新创建的局部层叠上下文中，只能在内部进行层叠级别的比较。

深入浅出

某区域内有个浮层提示或者下拉菜单，于是可能需要遮住该区域之下的区域。

HTML

```
<div class="a">
    ...
    <div class="tips">我是一个简陋的浮层提示</div>
</div>
<div class="b">
    ...
</div>
```

CSS

```
.a{position:relative;}
.tips{position:absolute;z-index:99;}
```

如上HTML/CSS代码，很显然，浮层 tips 将可以覆盖在其父级元素 a 的兄弟元素 b 之上。
于是你的意图得到实现，效果如下 图一 ：

（图一）

这是具体的实现例子 DEMO2：z-index实现元素层叠。

不过很显然，从 DEMO2 来看，你依然无法准确的判断出在各浏览器下， tips 能盖住 b 是因为其父级的定位还是本身的定位。

但是我们可以做这样一个测试，我们让 b 也拥有定位，Code如下：

CSS

```
.a{position:relative;}
.tips{position:absolute;z-index:99;}
.b{position:relative;}
```

这段代码run完之后，就比较纠结了，你能得到的效果将会如下 图二 ：

（图二）

当然要给出具体实现 DEMO3：验证创建局部层叠上下文。

首先，我们来解读一下这个例子：因为 a 和 b 都是 relative 且没有定义 z-index （等同于z-

index:auto），根据后来居上的原则，此时 b 的层叠级别是要高于 a 的，意思就是说 a 是无法遮住 b 的。不过从 DEMO3 中，我们看到 a 的子元素 tips 遮住了 b，这就表示 tips 能越过它，所以可以判断出 a 没有创建新的局部层叠上下文。很明显，这是完全吻合标准对此的定义。

不过这是在非IE6/7之下结果。在IE6/7下，我们看到 tips 并没能遮住 b，也就是说 tips 无法越过父级，因为 a 创建了新的局部层叠上下文，而 a 的层叠级别又比 b 低，所以 tips 无法遮住 b，这也就是在IE6/7下常出现覆盖Bug的根源。

结合 DEMO2 和 DEMO3，你能很肯定的得出以下结论：

- 当定位元素没有显式定义z-index值时，不会创建新的局部层叠上下文
- 子元素有可能和祖先的兄弟或者祖先兄弟的子元素处在同一个层叠上下文中

在实际工作中，有些情况可能是你没注意或者已然存在的。比如你事先可能并不知道 b 也是定位元素，或者由于某些原因，你需要将其设置为定位元素，于是可能出现各种兼容问题。如果你不了解 z-index 是如何创建局部层叠上下文，且又没注意到IE6/7的实现错误，那么处理起这样的问题将会让你深陷泥潭。

所以在实际的场景中，如果是为了相互覆盖而设置为定位，那么显式的定义 z-index 值，将可避免出现创建新局部层叠上下文差异。

如果需要越过祖先和其它区块内部元素进行相互层叠，那么考虑IE6/7的情况，也应该尽量避免给父级元素进定位。

opacity与层叠上下文

我们知道 opacity 属性是用来设置元素不透明度的。但可能知道 opacity 和层叠上下文有关的不多，不过没关系，这里我们简单聊聊这个话题，有两点必须注意：

- 当opacity值小于1时，该元素会创建新的局部层叠上下文，也就是说它可以和定位元素进行层叠层别比较
- 当opacity值小于1时，该元素拥有层叠级别且相当于z-index:0或auto，但不能定义 z-index ，除非本身是定位元素

简单来说，当一个普通的元素定义了 opacity 的值小于1时（比如 opacity:.5 ），那么该元素的层叠级别将会高于普通元素，其效果类同于定位元素没有显式定义 z-index 的情况，唯一的区别是没有显式定义 z-index 的定位元素不会产生局部层叠上下文，而定义了 opacity 值小于1的元素会产生新的局部层叠上下文。

opacity猜想

假定我们有 a，b，c 三个元素，它们相互层层覆盖在一起，如果这时将 a 元素定义为 opacity:.8，你知道结果会怎样吗？

HTML

```
<div class="a">a</div>
<div class="b">b</div>
<div class="c">c</div>
```

CSS

```
.a,.b,.c{width:100px;height:100px;}
.a{opacity:.8;background:#999;}
.b{margin:-70px 0 0 30px;background:#090;}
.c{margin:-70px 0 0 60px;background:#f00;}
```

如果你看明白了我对于 `opacity` 与层叠上下文的描述，相信你可以猜到结果，是的， `a` 元素将会覆盖 `b` 和 `c` 元素，虽然它在HTML文档中出现在 `b` 和 `c` 之前，且不是定位元素。

必须看看具体的示例不是么？ `DEMO4`：opacity与局部层叠上下文猜想。

如果我们将 `b` 和 `c` 设置为定位元素，又将会如何呢？

CSS

```
.a,.b,.c{width:100px;height:100px;}
.a{opacity:.8;background:#999;}
.b{position:relative;margin:-70px 0 0 30px;background:#090;}
.c{position:relative;margin:-70px 0 0 60px;background:#f00;}
```

不急，我们可以接着看示例 `DEMO5`：opacity与局部层叠上下文猜想2。

从 `DEMO4` 和 `DEMO5` 两例，我们可以验证：当一个普通元素定义了 `opacity` 为小于1的值时，该元素将像定位元素一样拥有层叠级别，可以覆盖普通元素，并且其层叠级别与未显式定义 `z-index` 的定位元素一样。

opacity创建局部层叠上下文

与未显式定义 `z-index` 的定位元素唯一不同的是 `opacity` 值小于1的元素会创建局部层叠上下文。

创建局部层叠上下文意味着什么，前文我们已经详述过。所以不再赘述，这里只给一个示例用以验证该特性。先奉上代码：

HTML

```
<div class="a">a
   <div class="d">d</div>
</div>
<div class="b">b</div>
<div class="c">c</div>
```

CSS

```
.a,.b,.c,.d{width:100px;height:100px;}
.a{opacity:.8;background:#999;}
.b{position:relative;margin:-70px 0 0 30px;background:#090;}
.c{position:relative;margin:-70px 0 0 60px;background:#f00;}
.d{position:absolute;z-index:99;height:50px;background:#090;}
```

你可以先看看具体结果 DEMO6：opacity创建新局部层叠上下文。

你会发现虽然 a 的子元素 d 将 z-index 定义为99，但 d 仍然无法遮住 b 和 c 元素，这是因为 a 创建了新的局部层叠上下文， d 元素无法超越父级。

需要注意的是，此时就算 a 元素变成了定位元素，也不能改变其会创建新局部层叠上下文的命运，因为他设置了 opacity:.8 。

按照我们前文所说，如果 a 没有定义 opacity:.8 ，但却像 b 和 c 元素一样设置了 relative ，那么其子元素 d 将可以覆盖 b 和 c ，至于这个例子就不再奉上了，大家随便写个测试一下即可。

图文替换

上述都是理论性的东西，相对枯燥，来个实际点的应用场景。

我们聊聊图文替换的事，相对于使用较广的方案如：缩进正/负值（正/负text-indent）、超小字体、margin溢出、padding溢出、line-height溢出、透明字体、display:none、visibility:hidden等方案而言，使用 z-index 负值的方案，有一些明显的优势：

- 无需考虑是否会有性能问题类同使用上述列举中的前几种方案（比如使用负缩进值-9999px，虽然此时文本被移到屏幕之外或者被裁减，但仍然会绘制一个宽9999px的盒子）；
- 没有像类似超小字体和透明字体一样的方案会需要一些额外的hack；
- 不像display:none方案那样有SEO欺骗嫌疑；
- 当图片加载失败时，可以显示文字；
- and etc...

先来看看一个图文替换的例子 DEMO7：图文替换实例。

在不同的网络环境下，它的表现如下 图三 ：

（图三）

具体的Code很简单：

HTML

```
<a href="#top" title="回到顶部"><span>TOP▲</span></a>
```

CSS

```
a,a span{display:inline-block;width:38px;height:38px;}
a{background:url(images/ico.png) no-repeat;}
a:hover{background-position:0 -39px;color:#fff;}
a span{position:relative;z-index:-1;background-color:#eee;}
a:hover span{background-color:#999;}
```

你会发现我们将 `span` 设置为了 `z-index:-1` ，此时它的层叠级别将比正常的元素还要低，所以它可以被其父元素超链接a盖住，从而在图片正常载入时显示父元素的背景图，在网络环境不好图片载入有问题时，显示自身。

很多时候，要实现一个需求可能有无数种解决方案，能够适应情况越多的方案毫无疑问会脱颖而出，这就要求我们可以去更多的思考，而不是更多的拷贝。

视觉格式化模型中的各种框

视觉格式化模型中的各种框

> 原作者：doyoe
> 原文链接：http://blog.doyoe.com/2015/03/09/css/%E8%A7%86%E8%A7%89%E6%A0%BC%E5%BC%8F%E5%8C%96%E6%A8%A1%E5%9E%8B%E4%B8%AD%E7%9A%84%E5%90%84%E7%A7%8D%E6%A1%86/

在聊这个话题之前，我们可能得先简单说说 `视觉格式化模型` 这个概念。

`视觉格式化模型` 的全称是 `Visual formatting model` ，它被用来描述用户代理（比如浏览器）在图形媒体下如何处理文档树。

在 `视觉格式化模型` 中，每个文档树的元素会根据框模型产生零到多个框（boxes）。这些框的布局取决于框的尺寸，类型，定位方式（正常流，浮动和绝对定位），元素之间的关系和外部信息（例如：视口 ① 大小，置换元素的固有尺寸等等）。

> 举个最简单的例子来讲，假设一个页面上有2个div，那么第2个div的位置会取决第1个div的高度定义；假设更复杂一点，第1个div是浮动的，那么第2个div的位置还要取决于第1个div的宽度。

不同类型的框

在 CSS 中，可能会产生不同类型的框，框的类型取决于 display 属性的设定。某种程度上，框的类型会影响其在视觉格式化模型中的表现。接下来会详细的聊聊这些不同类型的框以及它们在视觉可视化模型中的表现。

在说这个之前，我们先回忆一下，大家常说的一些名词：

- Containing block（包含块）
- Block-level element（块级元素）
- Block element（块元素）
- Block-level box（块级框），Block container box（块容器框），Block box（块框）
- Inline-level element（行内级元素）
- Inline element（行内元素）
- Inline-level box（行内级框），Atomic inline-level box（原子行内级框），Inline box（行内框）
- Anonymous boxes（匿名框）

包含块

首先，来看看什么是 包含块 ？这个说及 CSS 一般的都会提及的基本概念。

一个元素，它的框的尺寸和位置会相对于一个特定的矩形框边缘来计算而得到，这个特定的矩形框称之为该元素的 包含块 。

（为）一个元素生成的框通常会充当其子框的包含块；当我们叫一个 框的包含块 时，其实表达的是 该框所处的包含块 ，而不是其自身产生的包含块。

每个框相对于其包含块（ 该框所处的包含块 ）都会被给予一个位置，不过该框并不局限在包含块内，有可能会溢出，通常这个时候你会借助 overflow 属性来进行处理。

除了说什么是包含块，这里顺带再介绍一下生成包含块的一些特殊场景：

- 由根元素生成的包含块叫做 初始包含块（initial containing block）。
- 对于其它元素，如果元素的 position 值是 relative 或者 static ，其包含块由最近的祖先 块容器框 的内容边界（如果想知道什么是内容边界，可以先看看CSS盒模型）形成。

> 举个例子， td， th 就算有父容器 tr ，但它们的包含块却是由 table 生成，因为 table 是 块容器框 而 tr 不是

- 绝对定位元素的包含块由最近的定位（ position 值非 static ）祖先生成，如果不存在这样的祖先，则采用初始包含块；
- 固定定位元素（ position: fixed ）的包含块一般情况下都由视口 ① 生成；

说了这几个特殊的情景，你会发现并不是所有的包含块都是由父元素所生成。

什么是块级元素？

块级元素是那些视觉上会被格式化成块状的元素，通俗一点来说就是那些会换新行的元素。 display 属性的： block , list-item , table , flex , grid 值都可以将一个元素设置成块级元素。

> 举个例子来说， li 是一个 块级元素，但也有人会说它是一个 块元素 。嗯， li 确实是一个块级元素，但并不是一个块元素，为什么？

什么是块元素？

块元素是 display 属性值为 block 的元素，它应该是 块级元素 的一个子集，而不是等同的，一个 块元素 是一个 块级元素 ，但一个 块级元素 不一定是一个 块元素 ，所以不要混淆。

块级框，块容器框，块框

什么是块级框？

块级元素生成块级框，这些框会参与某些 BFC 。每个块级元素都会生成一个主要的块级框来包含其子框和生成的内容，同时任何定位方式都会与这个主要的块级框有关。

某些块级元素还会在主要的块级框之外产生额外的框：例如 list-item 元素，它需要生成一个额外的框用于包含 list-style-type 。这些额外的框会相对于主要的块级框来进行排版。

什么是块容器框？

一个 块容器框 要么只包含 块级框 ，要么创建一个 IFC 而只包含 行内级框 ，但不能同时包含 块级框 和 行内级框 。

除了 table框 和 置换元素 ，一个 块级框 同时也是一个 块容器框 。 非置换的行内块 和 单元格 是 块容器 但不是 块级框 。

并不是所有的 块级框 都是 块容器框 ，也并不是所有的 块容器框 都是 块级框 。

块级框 和 块容器框 的另外一个重要的区别是： 块级框 需要能够包含其生成的内容，但 块容器框 并不需要。这是什么意思呢？简单的解释一下：

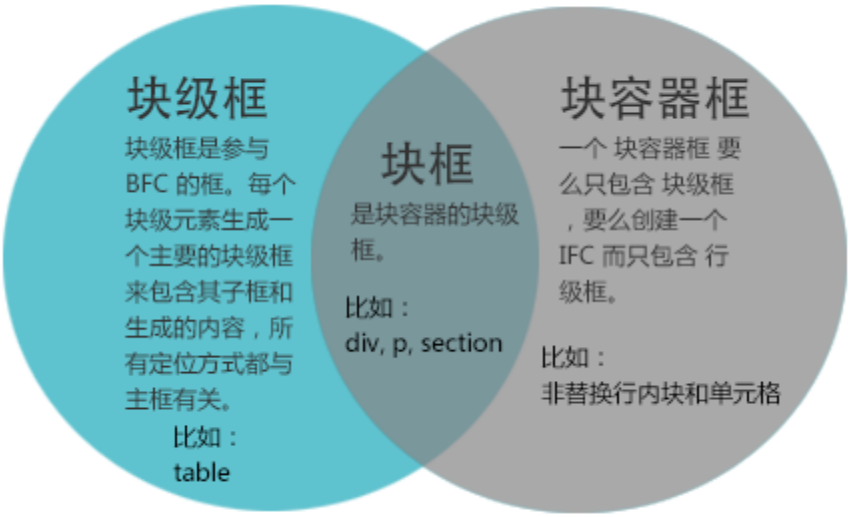
> 比如一个 iframe 其内容由 src 属性所决定，这可以当成是生成的内容，所以 iframe 是一个 块级框 但却不是 块容器框

什么是块框？

简要的来说，是 块容器 的 块级框 称之为 块框 。

可以通过下面这个图来快速的梳理清楚这3者之间的联系：

（图一）



说完了块级框，接下来说说行内级框

什么是行内级元素？

行内级元素是那些不会为自身内容形成新的块，而让内容分布在多行中的元素。 display 属性的： inline ， inline-table ， inline-block ， inline-flex ， inline-grid 值都可以将一个元素设置成行内级元素。

什么是行内元素？

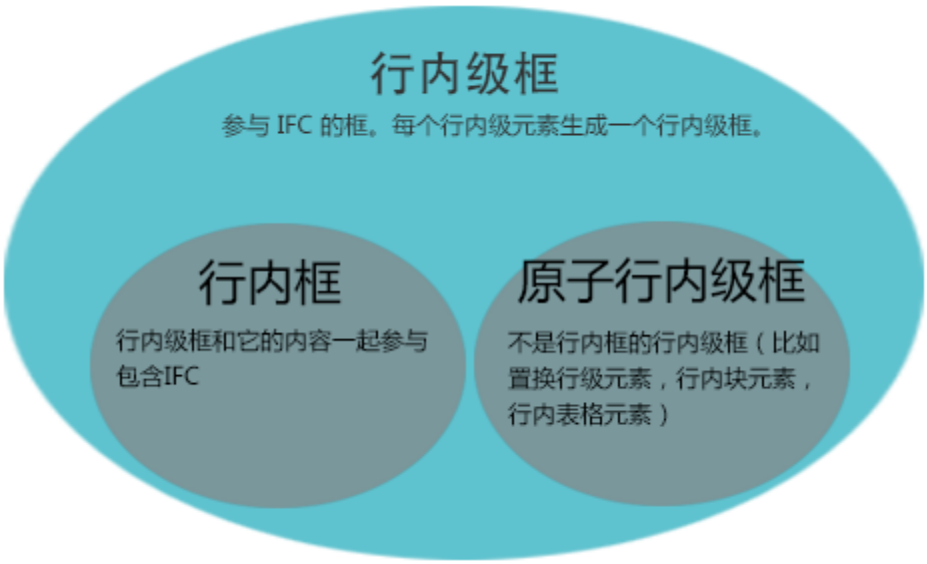
如同块元素之于块级元素的关系，行内元素仅仅是 display 属性值为 inline 的元素，这里不再赘述。

行内级框，原子行内级框，行内框

行内级元素生成行内级框，而这些框会参与某个 IFC 。一个 行内框 是 行内级框 ，且其内容参与了包含它的 IFC 。一个 display 值是 inline 的非置换元素会生成一个行内框。那些不是行内框的行内级框（例如行内级置换元素、行内块元素、行内表格元素）被称为 原子行内级框 ，因为它们是以单一不透明框的形式来参与其 IFC 的。

细心的你会发现并没有一个 行内容器框 与 块容器框 相对应，但却多了一个 原子行内级框 。并且有趣的是 行内块 （包括置换和非置换元素）是 原子行内级框 ，而 非置换行内块 却同时还是 块容器框 。

（图二）



匿名框

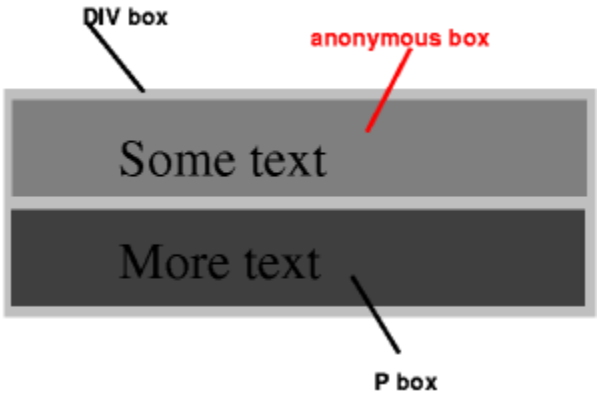
匿名框 包括 匿名块框 和 匿名行内框 。

假设一个 div 中包含有一个 p 和一段纯文本 xxx 。由于 块容器框 只允许要么包含 块级框 ，要么包含 行内级框 ，所以，为了符合这种情况， div 会生成一个匿名的块级框用于包裹 xxx ，这个匿名框就叫做 匿名块框 。

我们拿 W3C 上的一个例子来加深对匿名块框的印象：

```
<div>
 Some text
 <p>More text</p>
</div>
```

（图三）

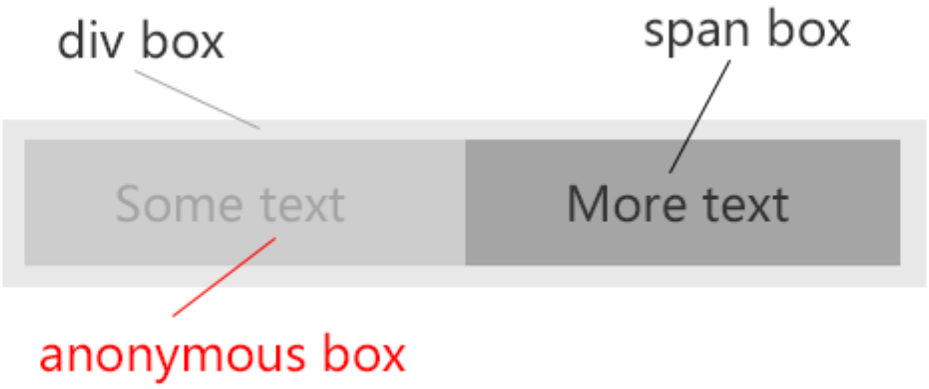


与此同时，我们将上面的代码稍微改一下，将 p 变成 span ：

```
<div>
 Some text
 <span>More text</span>
</div>
```

`div` 生成一个块框，`More text` 由 `span` 生成一个行内框，由于 `Some text` 没有与之相关的行级元素，将由 div 为其生成一个行内框用以包裹，这个框称为匿名行内框。如图四：

（图四）



假设一个匿名框的类型可根据上下文来清晰界定，则 `匿名行内框` 和 `匿名块框` 都可被简称为 `匿名框`。

匿名框的继承属性会从包含它的非匿名框那里继承，非继承属性取其初始值。

附注：

- ① 用户代理一般会向用户提供一个载体（屏幕上的一个窗口或其它可视区域）用以访问文档，这个载体就叫做 `视口`。用户代理可以在视口大小被调整时改变文档的布局。如果视口小于渲染文档的画布区域，用户代理应当提供一个滚动机制。每个画布只能拥有一个视口，但用户代理可以把文档渲染至多个画布上（即为相同文档提供不同的视图）。

说明：

- 最近的文章都是断断续续写的，如读者朋友发现存在描述错误地方请及时提醒。

置换和非置换元素

置换和非置换元素

> 原作者：doyoe
> 原文链接：http://blog.doyoe.com/2015/03/15/css/%E7%BD%AE%E6%8D%A2%E5%92%8C%E9%9D%9E%E7%BD%AE%E6%8D%A2%E5%85%83%E7%B4%A0/

先进一个题外话

在面试一个 `重构`（各大公司的叫法可能不太一样）时，我喜欢从一个点开始问，然后一直延展下去成为一条线，甚至是一个面，直到问到不会的地方，然后又换另外一个点。

> 例如：我可能会说，能简单聊聊 `行内级元素` 和 `块级元素` 的区别吗。

一般这时，候选人都会说到 `行内级元素` 不会换新行，而 `块级元素` 会格式化为块状，即换行。但也有些遗憾的方面（如：混淆了块元素和块级元素，行内元素和行内级元素），当然这看起来似乎不是特别重要。

> 这时我会继续问，`行内元素` 能够定义宽度和高度吗？

不少候选人会说：**不能**

> 我会继续问，说几个你熟悉的 `行内元素` 吧

于是 `span`，`strong`，`em`，`ins` ... 答案我还是比较满意的。

> 我仍然会继续，`img` 是行内元素么？

候选人这时通常会迟疑一下，可能意识到我接下来想问啥了，但还是会回答：**是**

> 于是我会说，那 `img` 能定义宽度和高度么？

有的候选人这时会犹豫，因为如果回答是，就会推翻他之前说的 `行内元素不能定义宽高` ，如果回答不是，似乎又和他所熟知的经验不一致。但通常最后还是会回答：**能**

> 那我就又得问，你之前不是说 `行内元素不能定义宽高` 吗？为什么 `img` 可以？

到这里，候选人基本上不知道要怎么回答好了，最后可能会告诉我，**因为 `img` 是特殊元素**

当然，虽然这么回答也不能说是错误的，但基本上也能知道候选人对这条线的基础的掌握程度了。

但我希望听到的答案是通过解释置换元素相关的概念从而给出答案。

什么是置换元素？

一个 `内容` 不受CSS视觉格式化模型控制，CSS渲染模型并不考虑对此内容的渲染，且元素本身一般拥有固有尺寸（宽度，高度，宽高比）的元素，被称之为置换元素。

什么是非置换元素？

w3c并没有给出明确的非置换元素的解释，但能确定的是除置换元素之外，所有的元素都是非置换元素。

行内级置换和非置换元素的宽度定义

对于行内级非置换元素，宽度设置是不适用的。

对于行内级置换元素来说，其宽度的设置需遵循以下几点：

- 若宽高的计算值都为 `auto` 且元素有固有宽度，则 `width` 的使用值为该固有宽度；

> 典型的例子是：拥有默认宽高的 `input` 当宽度的计算值为auto时，则宽度使用值为其默认的固有宽度

- 若宽度的计算值为 `auto` 且元素有固有宽度，则 `width` 的使用值为该固有宽度；

> 例子同上

- 若宽度的计算值为 `auto` 且高度有 `非auto` 的计算值，并且元素有固有宽高比，则 `width` 的使用值为 `高度使用值 * 固有宽高比` ；

> 典型的例子：`img` 当只定义了其高度值时，其宽度将会根据固有宽高比进行等比设置

- 除此之外，当 `width` 的计算值为 `auto` 时，则宽度的使用值为 `300px`

> 典型的例子：比如iframe, canvas

其它类型的置换元素，其宽度的定义都参照行内置换元素的定义。

行内级置换和非置换元素的高度定义

对于行内级非置换元素，高度设置是不适用的。

对于行内级置换元素来说，其高度的设置需遵循以下几点：

- 若宽高的计算值都为 `auto` 且元素有固有高度，则 `height` 的使用值为该固有高度；
- 若高度的计算值为 `auto` 且元素有固有高度，则 `height` 的使用值为该固有高度；
- 若高度的计算值为 `auto` 且宽度有 `非auto` 的计算值，并且元素有固有宽高比，则 `height` 的使用值为： `宽度使用值 / 固有宽高比` ；
- 若高度的计算值为 `auto` 且上述条件完全不符，则 `height` 的使用值 `不能大于150px` ，且宽度不能大于长方形高度的2倍。

其它类型的置换元素，其高度的定义都参照行内置换元素的定义。

移动前端第一弹：viewport详解

移动前端第一弹：viewport详解

> 原作者：doyoe
> 原文链
> 接：http://blog.doyoe.com/2015/10/13/mobile/%E7%A7%BB%E5%8A%A8%E5%89%8D%E7%

> AB%AF%E7%AC%AC%E4%B8%80%E5%BC%B9%EF%BC%9Aviewport%E8%AF%A6%E8%A7%A3/

前言

这次想聊聊移动开发相关的事。是的，你没有看错，一句话就可以开始你的移动前端开发。

你心里一定在想，什么话这么酷，能够瞬间带入到移动前端开发的世界。

但其实它一点也不新奇，不复杂。

viewport简介

没错，就是 viewport 特性，一个移动专属的 Meta 值，用于定义视口的各种行为。

该特性最先由 Apple 引入，用于解决移动端的页面展示问题，后续被越来越多的厂商跟进。

举个简单的例子来讲为什么会需要它：
我们知道用户大规模使用手机等移动设备来进行网页浏览器，其实得益于智能手持设备的兴起，也就是近几年的事。（还记得不久前的几年，满大街都还是诺基亚的天下么？）

这时有一个很现实的问题摆在了厂商面前，用户并不能很好地通过手机等设备访问网页，因为屏幕太小。

layout viewport

Apple 也发现了这个问题，并且适时的出现，它提出了一个方案用来解决这个问题。在iOS Safari中定义了一个 viewport meta 标签，用来创建一个 虚拟的布局视口（layout viewport），而这个视口的分辨率接近于PC显示器，Apple 将其定义为 980px（其他厂商各有不同①）。

这就很好的解决了早期的页面在手机上显示的问题，由于两者之间的宽度趋近，CSS只需要像在PC上那样渲染页面就行，原有的页面结构不会被破坏。

> ①的描述大致如下，数值不一定持续准确，厂商可能更改，但这个绝对值其实并非特别重要：
> iOS, Android基本都是: 980px
> BlackBerry: 1024px

visual viewport

有了 layout viewport，我们还需要一个视口用来承载它，这个视口可以简单的认为是手持设备物理屏幕的可视区域，我们称之为（视觉视口）visual viewport。这是一个比较直观的概念，因为你能看得见你的手机屏幕。

对于 visual viewport，开发者一般只需要知道它的存在和概念就行，因为无法对它进行任何设置或者修改。很明显，visual viewport 的尺寸不会是一个固定的值，甚至每款设备都可能不同。大致列几种常见设备的 visual viewport 尺寸：

- iPhone4~iPhone5S: 320*480px
- iPhone6~iPhone6S: 375*627px
- iPhone6 Plus~iPhone6S Plus: 414*736px

以 iPhone4S 为例，会在其320px②的 visual viewport 上，创建一个宽 980px 的 layout viewport ，于是用户可以在 visual viewport 中拖动或者缩放网页，来获得良好的浏览效果；布局视口用来配合CSS渲染布局，当我们定义一个容器的宽度为 100% 时，这个容器的实际宽度是 980px 而不是 320px ，通过这种方式大部分网页就能以缩放的形式正常显示在手机屏幕上了。

> ②的描述大致如下：
> 早期移动前端开发工程师常能见到宽640px的设计稿，原因就是UI工程师以物理屏幕宽度为320px的 iPhone4-iPhone5S 作为参照设计；
> 当然，现在你还可能会见到750px和1242px尺寸的设计稿，原因当然是iPhone6的出现

当然，为了更好的适配移动端或者只为移动端设计的应用，单有布局视口和视觉视口还是不够的。

ideal viewport

我们还需要一个视口，它类似于布局视口，但宽度和视觉视口相同，这就是完美视口（ideal viewport）。

有了完美视口，用户不用缩放和拖动网页就能够很好的进行网页浏览。而完美视口也是通过 viewport meta 的某种设置来达到。

说了这么一大堆的东西，貌似都和 viewport 有关联，那么 viewport 到底怎么搞，请继续往下。

> 关于3个视口，PPK已经做了非常棒的阐释，你也可以在 StackOverflow 上找到一些对此描述的相互补充，例如：[1], [2]，有兴趣的童鞋也可以看看

viewport特性

通常情况下， viewport 有以下6种设置。当然厂商可能会增加一些特定的设置，比如iOS Safari7.1增加了一种在网页加载时隐藏地址栏与导航栏的设置： minimal-ui ，不过随后又将之移除了。

Name	Value	Description
width	正整数或 device-width	定义视口的宽度，单位为像素
height	正整数或 device-height	定义视口的高度，单位为像素
initial-scale	[0.0-10.0]	定义初始缩放值

Name	Value	Description
minimum-scale	[0.0-10.0]	定义缩小最小比例，它必须小于或等于maximum-scale设置
maximum-scale	[0.0-10.0]	定义放大最大比例，它必须大于或等于minimum-scale设置
user-scalable	yes/no	定义是否允许用户手动缩放页面，默认值yes

width

width 被用来定义layout viewport 的宽度，如果不指定该属性（或者移除 viewport meta 标签），则 layout viewport 宽度为厂商默认值。如：iPhone为 980px ；

举个例子：

```
<meta name="viewport" content="width=device-width" />
```

此时的 layout viewport 将成为 ideal viewport ，因为 layout viewport 宽度与设备视觉视口宽度一致了。

除了 width 之外，还有一个属性定义也能实现 ideal viewport ，那就是 initial-scale 。

height

与 width 类似，但实际上却不常用，因为没有太多的use case。

initial-scale

如果想页面默认以某个比例放大或者缩小然后呈现给用户，那么可以通过定义 initial-scale 来完成。

initial-scale 用于指定页面的初始缩放比例，假定你这样定义：

```
<meta name="viewport" content="initial-scale=2" />
```

那么用户将会看到2倍大小的页面内容。

在说 width 的时候，我们说到 initial-scale 也能实现 ideal viewport ，是的，你只需要这样做，也可以得到完美视口：

```
<meta name="viewport" content="initial-scale=1" />
```

maximum-scale

在移动端，你可能会考虑用户浏览不便，然后给予用户放大页面的权利，但同时又希望是在一定范围内的

放大，这时就可以使用 maximum-scale 来进行约束。

maximum-scale 用于指定用户能够放大的比例。

举个例子来讲：

```
<meta name="viewport" content="initial-scale=1,maximum-scale=5" />
```

假设页面的默认缩放值 initial-scale 是 1 ，那么用户最终能够将页面放大到这个初始页面大小的5倍。

minimum-scale

类似 maximum-scale 的描述，不过 minimum-scale 是用来指定页面缩小比例的。

通常情况下，为了有更好地体验，不会定义该属性的值比1更小，因为那样页面将变得难以阅读。

user-scalable

如果你不想页面被放大或者缩小，通过定义 user-scalable 来约束用户是否可以通过手势对页面进行缩放即可。
该属性的默认值为 yes ，即可被缩放（如果使用默认值，该属性可以不定义）；当然，如果你的应用不打算让用户拥有缩放权限，可以将该值设置为 no 。

使用方法如下：

```
<meta name="viewport" content="user-scalable=no" />
```

结语

正如开篇所说，这既不高深也不新奇，它而仅仅是一点观念转变。

当你掌握了 viewport ，那么意味着你已经大致了解了移动平台与PC平台的不同，你可以更专注而细致的去解决某些平台差异问题。

melon

BFC神奇背后的原理

BFC神奇背后的原理

> 原作者：melon
> 原文地址：http://melonh.com/css/2014/01/10/the-magic-bfc.html

BFC 已经是一个耳听熟闻的词语了，网上有许多关于 BFC 的文章，介绍了如何触发 BFC ， 以及 BFC 的一些用处（如清浮动，防止margin重叠等）。虽然我知道如何利用BFC解决这些问题，但当别人问我 BFC 是什么，我还是不能很有底气地解释清楚。于是这两天仔细阅读了CSS2.1 spec, 和许多文章，来全面地理解BFC：

1 BFC 是个什么？
2 哪些元素会生成 BFC
3 BFC 的神奇的作用，及背后的原理

一、BFC是什么？

在解释BFC是什么之前，需要先介绍 Box， Formatting context 的概念。

Box: CSS布局的基本单位

Box 是CSS布局的对象和基本单位， 直观点来说，就是一个页面是由很多个 Box 组成的。元素的类型和display属性，决定了这个 Box 的类型。 不同类型的 Box ， 会参与不同的 Formatting context (一个决定如何渲染文档的容器)，因此 Box 内的元素会以不同的方式渲染。让我们看看有哪些盒子：

- block-level box : display属性为block, list-item, table的元素，会生成 block-level box 。并且参与 block fomatting context 。
- inline-level box : display属性为inline, inline-block, inline-table的元素，会生成 inline-level box 。并且参与 inline formatting context 。
- run-in box : css3中才有， 这儿先不讲了

Formatting context

Formatting context 是W3C CSS2.1规范中的一个概念。它是页面中的一块渲染区域，并且有一套渲染规则，它决定了其子元素将如何定位，以及和其他元素的关系和相互作用。

最常见的 Formatting context 有 Block fomatting context (简称 BFC)和 Inline formatting context (简称 IFC)。

CSS2.1 中只有 BFC 和 IFC , CSS3中还增加了 GFC 和 FFC

BFC 定义

BFC (Block formatting context)直译为"块级格式化上下文"。它是一个独立的渲染区域，只有 Block-level box 参与， 它规定了内部的 Block-level Box 如何布局，并且与这个区域外部毫不相干。

BFC布局规则：

1. 内部的 Box 会在垂直方向，一个接一个地放置。
2. Box垂直方向的距离由margin决定。属于同一个 BFC 的两个相邻 Box 的margin会发生重叠
3. 每个元素的margin box的左边， 与包含块border box的左边相接触(对于从左往右的格式化，否则相反)。即使存在浮动也是如此。
4. BFC 的区域不会与 float box 重叠。
5. BFC 就是页面上的一个隔离的独立容器，容器里面的子元素不会影响到外面的元素。反之也如此。
6. 计算 BFC 的高度时，浮动元素也参与计算

二、哪些元素会生成BFC?

1. 根元素
2. float属性不为none
3. position为absolute或fixed
4. display为inline-block, table-cell, table-caption, flex, inline-flex
5. overflow不为visible

三、BFC的作用及原理

1. 自适应两栏布局

代码：

```
<style>
    body {
        width: 300px;
        position: relative;
    }

    .aside {
        width: 100px;
        height: 150px;
        float: left;
        background: #f66;
    }

    .main {
        height: 200px;
        background: #fcc;
    }
</style>
<body>
    <div class="aside"></div>
    <div class="main"></div>
</body>
```

页面：

根据 BFC 布局规则第3条：

> 每个元素的 margin box 的左边， 与包含块 border box 的左边相接触(对于从左往右的格式化，否则相反)。即使存在浮动也是如此。

因此，虽然存在浮动的元素 aslide ，但main的左边依然会与包含块的左边相接触。

根据 BFC 布局规则第四条：

> BFC 的区域不会与 float box 重叠。

我们可以通过通过触发main生成 BFC ， 来实现自适应两栏布局。

```
.main {
    overflow: hidden;
}
```

当触发main生成 BFC 后，这个新的 BFC 不会与浮动的 aside 重叠。因此会根据包含块的宽度，和 aside 的宽度，自动变窄。效果如下：

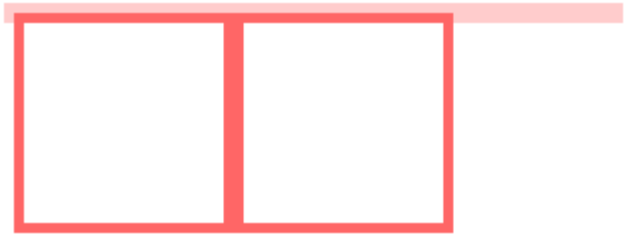
2. 清除内部浮动

代码:

```
<style>
    .par {
        border: 5px solid #fcc;
        width: 300px;
    }

    .child {
        border: 5px solid #f66;
        width:100px;
        height: 100px;
        float: left;
    }
</style>
<body>
    <div class="par">
        <div class="child"></div>
        <div class="child"></div>
    </div>
</body>
```

页面：



根据 BFC 布局规则第六条：

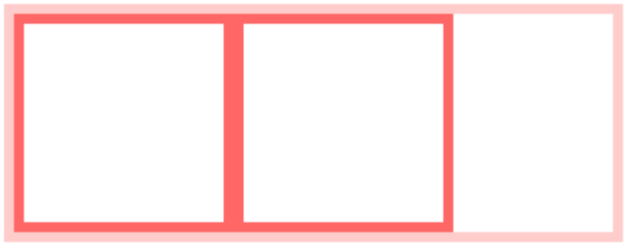
> 计算 BFC 的高度时，浮动元素也参与计算

为达到清除内部浮动，我们可以触发par生成 BFC ，那么par在计算高度时，par内部的浮动元素child也会参与计算。

代码：

```
.par {
    overflow: hidden;
}
```

效果如下：

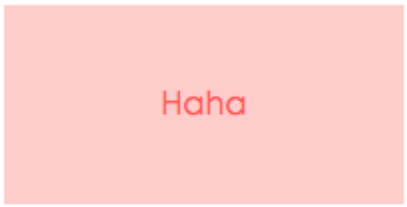


3. 防止垂直margin重叠

代码：

```
<style>
    p {
        color: #f55;
        background: #fcc;
        width: 200px;
        line-height: 100px;
        text-align:center;
        margin: 100px;
    }
</style>
<body>
    <p>Haha</p>
    <p>Hehe</p>
</body>
```

页面：



两个p之间的距离为100px，发送了margin重叠。
根据BFC布局规则第二条：

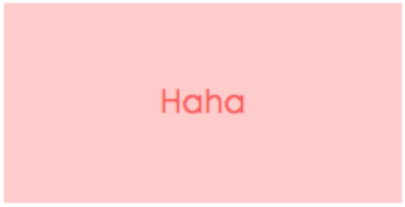
> Box 垂直方向的距离由margin决定。属于同一个 BFC 的两个相邻 Box 的margin会发生重叠

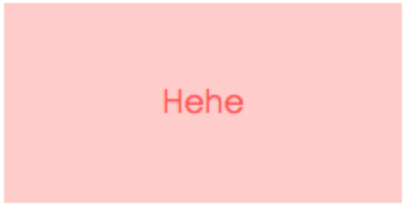
我们可以在 p 外面包裹一层容器，并触发该容器生成一个 BFC 。那么两个P便不属于同一个 BFC ，就不会发生margin重叠了。

代码：

```
<style>
    .wrap {
        overflow: hidden;
    }
    p {
        color: #f55;
        background: #fcc;
        width: 200px;
        line-height: 100px;
        text-align:center;
        margin: 100px;
    }
</style>
<body>
    <p>Haha</p>
    <div class="wrap">
        <p>Hehe</p>
    </div>
</body>
```

效果如下:





总结

其实以上的几个例子都体现了 BFC 布局规则第五条：

> BFC 就是页面上的一个隔离的独立容器，容器里面的子元素不会影响到外面的元素。反之也如此。

因为 BFC 内部的元素和外部的元素绝对不会互相影响，因此， 当 BFC 外部存在浮动时，它不应该影响 BFC 内部 Box 的布局， BFC 会通过变窄，而不与浮动有重叠。同样的，当 BFC 内部有浮动时，为了不影响外部元素的布局， BFC 计算高度时会包括浮动的高度。避免 margin 重叠也是这样的一个道理。