chokcoco / iCSS

谈谈一些有趣的CSS题目（1~5） #1

New issue

Open chokcoco opened this issue on 19 Sep 2016 · 18 comments

chokcoco commented on 19 Sep 2016 • edited Owner

开本 issues ，讨论一些有趣的 CSS 题目，抛开实用性而言，一些题目为了拓宽一下解决问题的思路，此外，涉及一些容易忽视的 CSS 细节。

解题不考虑兼容性，题目天马行空，想到什么说什么，如果解题中有你感觉到生僻的 CSS 属性，赶紧去补习一下吧。

不断更新，不断更新，不断更新，重要的事情说三遍。

1、下面这个图形，只使用一个标签，可以有多少种实现方式：

假设我们的单标签是一个 div：

```
<div></div>
```

定义如下通用 css ：

```
div{
    position:relative;
    width:200px;
    height:60px;
    background:#ddd;
}
```

法一：border

这个应该是最最最容易想到的了

```
div{
    border-left:5px solid deeppink;
}
```

法二：使用伪元素

一个标签，算上 before 与 after 伪元素，其实算是有三个标签，这也是很多单标签作图的基础，本题中，使用伪元素可以轻易完成。

```
div::after{
    content:"";
    width:5px;
    height:60px;
    position:absolute;
    top:0;
    left:0;
```

Assignees

No one assigned

Labels

None yet

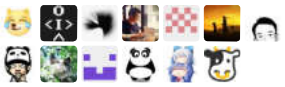
Projects

None yet

Milestone

No milestone

13 participants



```
    background:deeppink;
}
```

法三：外 box-shadow

盒阴影 `box-shadow` 大部分人都只是用了生成阴影，其实阴影可以有多重阴影，阴影不可以不虚化，这就需要去了解一下 `box-shaodw` 的每一个参数具体作用。使用 `box-shaodw` 解题：

```
div{
    box-shadow:-5px 0px 0 0 deeppink;
}
```

法四：内 box-shadow

盒阴影还有一个参数 `inset` ，用于设置内阴影，也可以完成：

```
div{
    box-shadow:inset 5px 0px 0 0 deeppink;
}
```

法五：drop-shadow

`drop-shadow` 是 CSS3 新增滤镜 `filter` 中的其中一个滤镜，也可以生成阴影，不过它的数值参数个数只有 3 个，比之 box-shadow 少一个。

```
div{
    filter:drop-shadow(-5px 0 0 deeppink);
}
```

法六：渐变 linearGradient

灵活使用 CSS3 的渐变可以完成大量想不到的图形，CSS3 的渐变分为线性渐变和径向渐变，本题使用线性渐变，可以轻易解题：

```
div{
    background-image:linear-gradient(90deg, deeppink 0px, deeppink 5px, transparent 5px);
}
```

法七：轮廓 outline

这个用的比较少，outline（轮廓）是绘制于元素周围的一条线，位于边框边缘的外围，可起到突出元素的作用。这个方法算是下下之选。

```
div{
    height:50px;
    outline:5px solid deeppink;
}
div{
    position:absolute;
    content:"";
    top:-5px;
    bottom:-5px;
    right:-5px;
    left:0;
    background:#ddd;
}
```

法八: 滚动条

这个方法由 小火柴的蓝色理想 提供，通过改变滚动条样式实现：

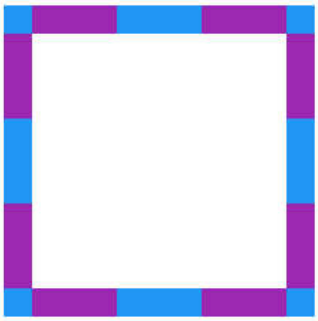
```
div{
    width:205px;
    background:deeppink;
    overflow-y:scroll;
}
div::-webkit-scrollbar{
    width: 200px;
    background-color:#ddd;
}
```

抛开实用性，仅仅是模拟出这个样式的话，这个方法真的让人眼前一亮。

上述就是想到的 8 种方法，不排除有没想到的，希望有其他的方法可以在评论中提出，具体 8 种实现可以戳这里看看：

codepen-单标签左边竖条的实现方式

2、类似下面这个图形，只使用一个标签，可以有多少种实现方式：



假设我们的单标签为 `div`：

```
<div></div>
```

定义如下通用 CSS:

```
div{
    position:relative;
    width: 180px;
    height: 180px;
}
```

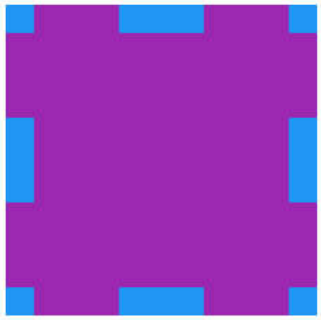
这一题主要考查的是盒子模型 `Box Model` 与 背景 `background` 的关系，以及使用 `background-clip` 改变背景的填充方式。

`background` 在 `Box Model` 中，他是布满整个元素的盒子区域的，并不是从 `padding` 内部开始（也就是说从 border 就开始啦），只不过实线边框（solid）部分遮住了部分 `background` ，所以我们使用虚线边框（dashed）就可以看到背景色是从 `border` 内部开始的。

我们给 `div` 添加如下样式：

```
div{
    background:#9c27b0;
    border:20px dashed #2196f3;
}
```

结果如下：



但有一点需要注意，`background-color` 是从元素的边框左上角起到右下角止，而 `background-image` 却不一样，他是从 `padding` 边缘的左上角起而到 `border` 的右下角边缘止。

background image 的绘制中有两个因素决定了绘图区域：

1. background positioning area。 `background-origin` 属性决定了这个**相对定位位置**，默认为 `padding-box` 。所以默认的背景图片绘制是从 padding box 的左上顶点开始的。

2. background painting area。 `background-clip` 属性决定了**绘制区间**，默认为 `border-box` 。所以在 `background-repeat: repeat` 的情况下：

> The image is repeated in this direction as often as needed to cover the background painting area.

嗯，什么意思呢，你可以戳进这个 demo 看看，正常情况下的背景图填充如下：

当然，这个填充规则是可以通过 `background-clip` 改变的。

> `background-clip` 设置元素的背景（背景图片或颜色）是否延伸到边框下面。

语法：

```
background-clip: border-box;  // 背景延伸到边框外沿（但是在边框之下）
background-clip: padding-box; // 边框下面没有背景，即背景延伸到内边距外沿。
background-clip: content-box; // 背景裁剪到内容区 (content-box) 外沿。
```

继续说回本题，接下来，只需要将中间部分填充为白色即可，这个用伪元素可以轻松完成，所以，其中一个方法如下：

```
div{
    background:#9c27b0;
    border:20px dashed #2196f3;
}
div::after{
    content:"";
    position:absolute;
    top:0;
    left:0;
    bottom:0;
    right:0;
    background:#fff;
}
```

法二：

上面的方法，我们使用了 `div` 的背景色默认情况下从 `border` 开始填充，及伪元素设置白色背景色填充 `div` 的中间的 `padding-box` 区域完成图形。

也可以反过来，使用伪元素背景色从 `border-box` 开始填充，使用 `div` 的背景色填充中间 `padding-box` 区域。

```
div{
    background:#fff;
    background-clip:padding-box;
    border:20px dashed #cccc99;
}
div::before{
    content:"";
    position:absolute;
```

```
    top:-20px;
    left:-20px;
    bottom:-20px;
    right:-20px;
    background:#996699;
    z-index:-1;
}
```

具体的 Demo 戳这里 。

上面 法二 除了用到了 background-clip 改变背景的填充区域，还用到了 z-index 触发元素生成了堆叠上下文（stacking context），改变了元素的层叠顺序（stacking levle），让 伪元素背景色 叠到了 div 背景色 之下，这两个概念下题会提及。

法....

本题主要是想讨论一下 CSS 的盒子模型 Box Model 与 背景 background 的关系，其实本题就是在于一个 dashed 边框，内部使用颜色填充即可，与上面第一题异曲同工，使用阴影、渐变都可以完成，感兴趣可以自己尝试一下其他解法。

3、层叠顺序（stacking level）与堆栈上下文（stacking context）知多少？

z-index 看上去其实很简单，根据 z-index 的高低决定层叠的优先级，实则深入进去，会发现内有乾坤。

看看下面这题，定义两个 div A 和 B，被包括在同一个父 div 标签下。HTML结构如下：

```
<div class="container">
    <div class="inline-block">#divA display:inline-block</div>
    <div class="float"> #divB float:left</div>
</div>
```

它们的 CSS 定义如下：

```
.container{
    position:relative;
    background:#ddd;
}
.container > div{
    width:200px;
    height:200px;
}
.float{
    float:left;
    background-color:deeppink;
}
.inline-block{
    display:inline-block;
    background-color:yellowgreen;
    margin-left:-100px;
}
```

大概描述起来，意思就是拥有共同父容器的两个 DIV 重叠在一起，是 display:inline-block 叠在上面，还是 float:left 叠在上面？

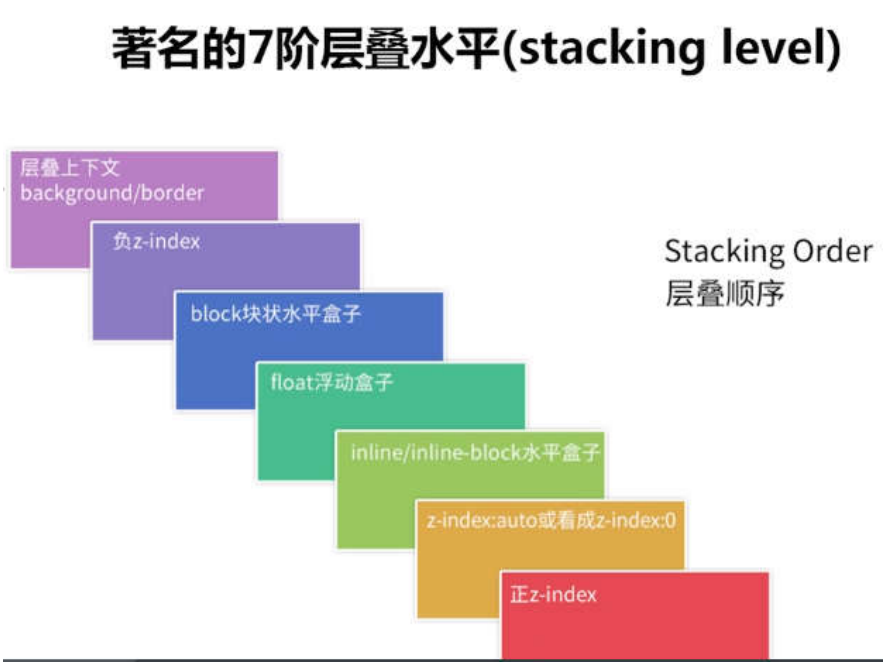
注意这里 DOM 的顺序，是先生成 display:inline-block ，再生成 float:left 。当然也可以把两个的 DOM 顺序调转如下：

```
<div class="container">
    <div class="float"> #divB float:left</div>
    <div class="inline-block">#divA display:inline-block</div>
</div>
```

会发现，无论顺序如何，始终是 display:inline-block 的 div 叠在上方。

Demo戳我。

这里其实是涉及了所谓的层叠水平（stacking level），有一张图可以很好的诠释：



运用上图的逻辑，上面的题目就迎刃而解， `inline-blcok` 的 `stacking level` 比之 `float` 要高，所以无论 DOM 的先后顺序都堆叠在上面。

不过上面图示的说法有一些不准确，按照 W3官方 的说法，准确的 7 层为：

1、the background and borders of the element forming the stacking context.

2、the child stacking contexts with negative stack levels (most negative first).

3、the in-flow, non-inline-level, non-positioned descendants.

4、the non-positioned floats.

5、the in-flow, inline-level, non-positioned descendants, including inline tables and inline blocks.

6、the child stacking contexts with stack level 0 and the positioned descendants with stack level 0.

7、the child stacking contexts with positive stack levels (least positive first).

稍微翻译一下：

1、形成堆叠上下文环境的元素的背景与边框

2、拥有负 `z-index` 的子堆叠上下文元素（负的越高越堆叠层级越低）

3、正常流式布局，非 `inline-block` ，无 `position` 定位（static除外）的子元素

4、无 `position` 定位（static除外）的 float 浮动元素

5、正常流式布局， `inline-block` 元素，无 `position` 定位（static除外）的子元素（包括 display:table 和 display:inline ）

6、拥有 `z-index:0` 的子堆叠上下文元素

7、拥有正 `z-index:` 的子堆叠上下文元素（正的越低越堆叠层级越低）

所以我们的两个 `div` 的比较是基于上面所列出来的 4 和 5 。5 的 `stacking level` 更高，所以叠得更高。

不过！不过！不过！重点来了，请注意，上面的比较是基于两个 `div` 都没有形成 `堆叠上下文` 这个为基础的。下面我们修改一下题目，给两个 `div` ，增加一个 `opacity`：

```
.container{
    position:relative;
    background:#ddd;
}
.container > div{
    width:200px;
    height:200px;
    opacity:0.9; // 注意这里，增加一个 opacity
}
```

```
.float{
    float:left;
    background-color:deeppink;
}
.inline-block{
    display:inline-block;
    background-color:yellowgreen;
    margin-left:-100px;
}
```

Demo戳我。

会看到， `inline-block` 的 `div` 不再一定叠在 `float` 的 `div` 之上，而是和 HTML 代码中 DOM 的堆放顺序有关，后添加的 div 会 叠在先添加的 div 之上。

这里的关键点在于，添加的 `opacity:0.9` 这个让两个 div 都生成了 `stacking context（堆叠上下文）` 的概念。此时，要对两者进行层叠排列，就需要 z-index ，z-index 越高的层叠层级越高。

堆叠上下文是HTML元素的三维概念，这些HTML元素在一条假想的相对于面向（电脑屏幕的）视窗或者网页的用户的 z 轴上延伸，HTML 元素依据其自身属性按照优先级顺序占用层叠上下文的空间。

那么，如何触发一个元素形成 `堆叠上下文` ？方法如下，摘自 MDN：

- 根元素 (HTML),
- z-index 值不为 "auto"的 绝对/相对定位，
- 一个 z-index 值不为 "auto"的 flex 项目 (flex item)，即：父元素 display: flex|inline-flex，
- opacity 属性值小于 1 的元素（参考 the specification for opacity），
- transform 属性值不为 "none"的元素，
- mix-blend-mode 属性值不为 "normal"的元素，
- filter值不为“none”的元素，
- perspective值不为“none”的元素，
- isolation 属性被设置为 "isolate"的元素，
- position: fixed
- 在 will-change 中指定了任意 CSS 属性，即便你没有直接指定这些属性的值
- -webkit-overflow-scrolling 属性被设置 "touch"的元素

所以，上面我们给两个 `div` 添加 `opacity` 属性的目的就是为了形成 `stacking context` 。也就是说添加 opacity 替换成上面列出来这些属性都是可以达到同样的效果。

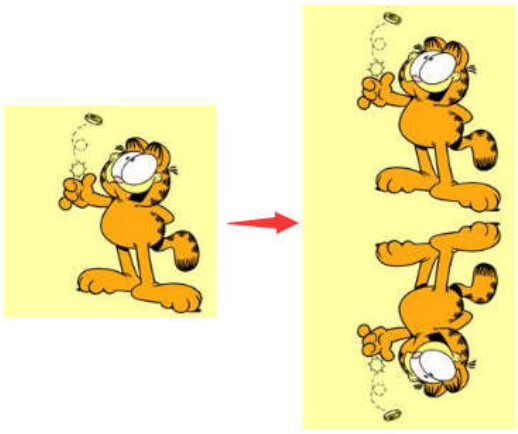
在层叠上下文中，其子元素同样也按照上面解释的规则进行层叠。 特别值得一提的是，其子元素的 z-index 值只在父级层叠上下文中有意义。意思就是父元素的 `z-index` 低于父元素另一个同级元素，子元素 `z-index` 再高也没用。

理解上面的 `stacking-level` 与 `stacking-context` 是理解 CSS 的层叠顺序的关键。

4、从倒影说起，谈谈 CSS 继承 inherit

给定一张有如下背景图的 div：

制作如下的倒影效果：

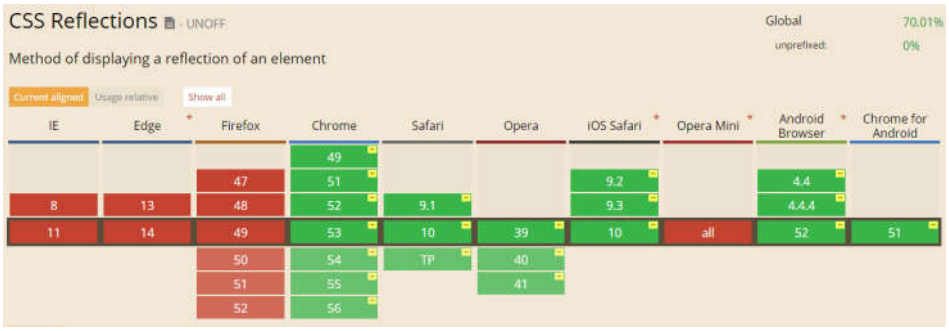


方法很多，但是我们当然要寻找最快最便捷的方法，至少得是无论图片怎么变化， `div` 大小怎么变化，我们都不用去改我们的代码。

法一：-webkit-box-reflect

这是一个十分新的 CSS 属性，使用起来十分简单，可以从各个方向反射我们内容。不过兼容性过于惨淡：

> 基本上是只有 -webkit- 内核的浏览器才支持。



不过使用起来真的是方便，解题如下：

```
div{
    -webkit-box-reflect: below;
}
```

-webkit- 内核下查看Demo

`box-reflect` 有四个方向可以选， `below` | `above` | `left` | `right` 代表下上左右。

法二：inherit，使用继承

本题主要还是为了介绍这种方法，兼容性好。

`inherit` 是啥，每个 CSS 属性定义的概述都指出了这个属性是默认继承的 ("Inherited: Yes") 还是默认不继承的 ("Inherited: no")。这决定了当你没有为元素的属性指定值时该如何计算值。

灵活使用 `inherit` 继承父值，可以解决许多看似复杂的问题。对于本题，我们对图片容器添加一个伪元素，使用 `background-image:inherit` 继承父值的背景图值，就可以做到无论图片如何变，我们的 CSS 代码都无需改动：

```
div::after {
    content: "";
    position: absolute;
    top: 100%;
    left: 0;
    right: 0;
    bottom: -100%;
    background-image: inherit;
    transform: rotateX(180deg);;
}
```

Demo戳我。

我们使用伪元素 `background-image: inherit;` 继承父元素的背景图，再使用 transform 旋转容器达到反射的效果。

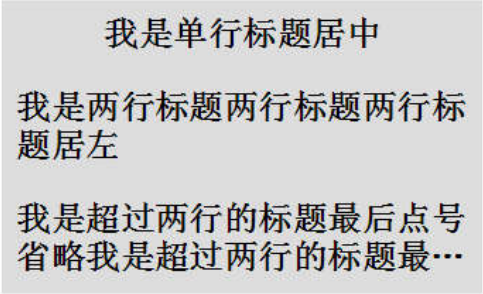
说到底，CSS 属性的取值就是由默认值（initial），继承（inherit）与加权系统构成的（其实还有 `unset(未设置)`、 `revert(还原)`），厘清它们的关系及使用方法对熟练使用 CSS 大有裨益。

5、单行居中显示文字，多行居左显示，最多两行超过用省略号结尾

这题就厉害了我的哥。

题目就是如上要求，使用纯 CSS，完成单行文本居中显示文字，多行居左显示，最多两行超过用省略号结尾，效果如下：

不愿看长篇大论的可以先看看效果：-webkit- 内核下 Demo 戳我



接下来就一步一步来实现这个效果。

首先是单行居中，多行居左

居中需要用到 `text-align:center` ，居左是默认值也就是 `text-align:left` 。如合让两者结合起来达到单行居中，多行居左呢？这就需要多一个标签，假设一开始我们定义如下：

```
<h2>单行居中，多行居左</h2>
```

现在，我们在 `h2` 中间，嵌套多一层标签 `p` ：

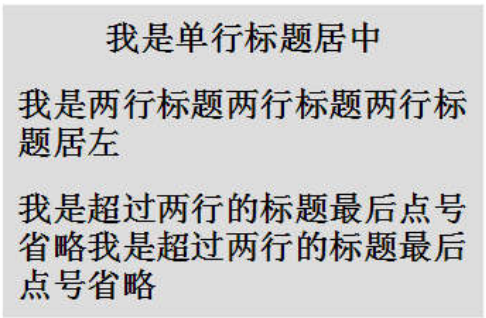
```
<h2><p>单行居中，多行居左</p></h2>
```

我们让内层 `p` 居左 `text-align:left` ，外层 `h2` 居中 `text-align:center` ，并且将 `p` 设置为 `display:inline-block` ，利用 `inline-block` 元素可以被父级 `text-align:center` 居中的特性，这样就可以实现单行居中，多行居左，CSS 如下：

```
p {
    display: inline-block;
    text-align: left;
}

h2{
    text-align: center;
}
```

得到的效果如下：



超出两行省略

完成了第一步，接下来要实现的是超出两行显示省略符号。

多行省略是有专门的新 CSS 属性可以实现的，但是有些兼容性不大好。主要用到如下几个：

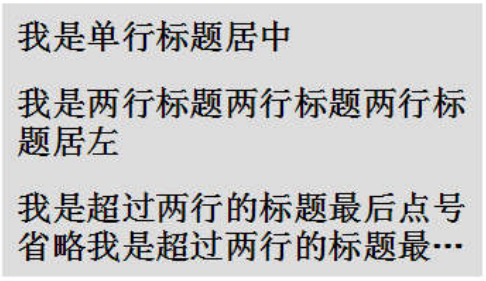
- display: -webkit-box; // 设置display，将对象作为弹性伸缩盒子模型显示
- -webkit-line-clamp: 2; // 限制在一个块元素显示的文本的行数
- -webkit-box-orient: vertical; // 规定框的子元素应该被水平或垂直排列

上述 3 条样式配合 `overflow : hidden` 和 `text-overflow: ellipsis` 即可实现 `webkit` 内核下的多行省略。好，我们将上述说的一共 5 条样式添加给 `p` 元素

```
p {
    display: inline-block;
    text-align: left;
    overflow : hidden;
    text-overflow: ellipsis;
    display: -webkit-box;
    -webkit-line-clamp: 2;
    -webkit-box-orient: vertical;
}

h2{
    text-align: center;
}
```

看看效果如下：



（在 -webkit- 内核浏览器下）发现，虽然超出两行的是被省略了，但是第一行也变回了居左，而没有居中。

看回上面的 CSS 中的 `p` 元素，原因在于我们第一个设置的 `display: inline-block` ，被接下来设置的 `display: -webkit-box` 给覆盖掉了，所以不再是 `inline-block` 特性的内部 `p` 元素占据了一整行，也就自然而然的不再居中，而变成了正常的居左展示。

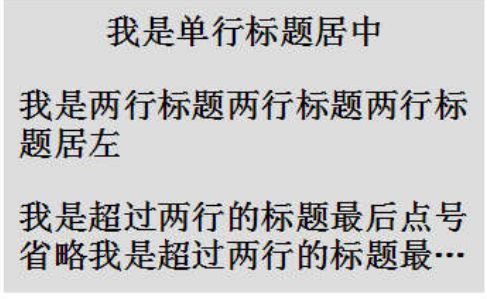
记得上面我们解决**单行居中，多行居左**时的方法吗？上面我们添加多了一层标签解决了问题，这里我们再添加多一层标签，如下：

```
<h2><p><em>单行居中，多行居左<em></p></h2>
```

这里，我们再添加一层 `em` 标签，接下来，

- 设置 `em` 为 `display: -webkit-box`
- 设置 `p` 为 `inline-block`
- 设置 `h2` 为 `text-align: center`

嘿！通过再设置多一层标签，解决 display 的问题，完美解决问题，再看看效果：



-webkit- 内核下 Demo 戳我

法二: 绝对定位障眼法

是的，还有第二种方法……

上面我们为了让第一行居中，使用了三层嵌套标签。

这次我们换一种思路，只使用两层标签，但是我们加多一行。结构如下：

```
<div class="container">
    <h2>
        <p>我是单行标题居中</p>
        <p class="pesudo">我是单行标题居中</p>
    </h2>
</div>
```

这里，新添加了一行 class 为 `pesudo` 的 `p` 标签，标签内容与文本内容一致，但是我们限定死 `class="pesudo"` 的 `p` 标签高度 height 与上面的 `p` 的行高 `line-height` 一致，并设置 `overflow:hidden`，那么这个 `p` 标签**最多只能能展示出一行文本**，接下来使用绝对定位，定位到 `h2` 的顶部，再设置 `text-align:center` 以及背景色与 `h2` 背景色一致。

这样最多显示单行且样式为居中的 `class="pesudo"` p 标签就重叠到了原本的 `p` 标签之上。表现为单行居中，多行时第一行则铺满，解决了我们的问题。多行省略与方法一相同。CSS 如下：

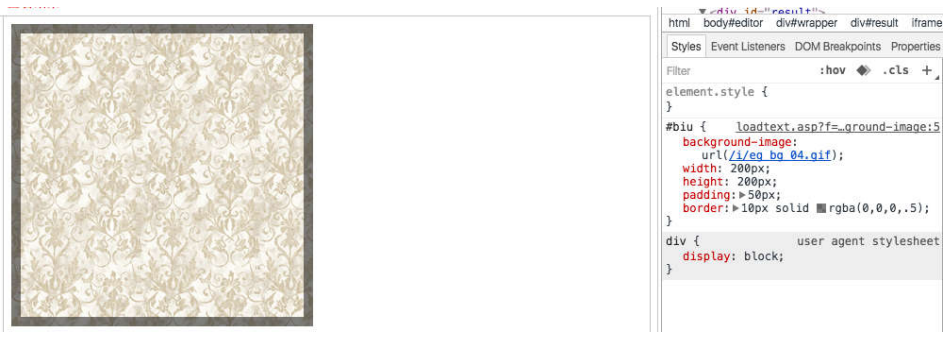
```
h2{
    position:relative;
    line-height:30px;
}
p{
    overflow : hidden;
    text-overflow: ellipsis;
    display: -webkit-box;
    -webkit-line-clamp: 2;
    -webkit-box-orient: vertical;
}
.pesudo{
    position:absolute;
    width:100%;
    height:30px;
    overflow:hidden;
    top:0;
    background:#ddd;
    text-align:center;
}
```

-webkit- 内核下 Demo 戳我

33 1 2 4

idiotWu commented on 19 Sep 2016

`background-image` 默认也是从 `border-box` 延展开的：



wangpengfei15975 commented on 19 Sep 2016

+1

chokcoco commented on 19 Sep 2016 Owner