chokcoco / iCSS

谈谈一些有趣的CSS题目（6~10） #2

New issue

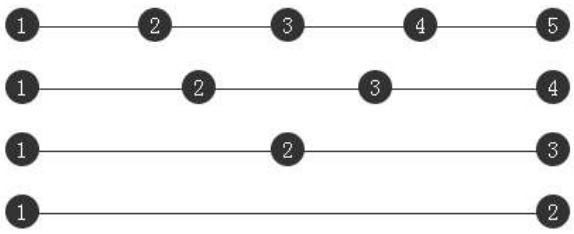
Open chokcoco opened this issue on 23 Sep 2016 · 18 comments

chokcoco commented on 23 Sep 2016 • edited Owner

> **（本页图片可能会404，刷一下就出来了）**

6、全兼容的多列均匀布局问题

如何实现下列这种多列均匀布局：



法一： display:flex

CSS3 弹性盒子(Flexible Box 或 Flexbox)，是一种布局方式，当页面需要适应不同的屏幕大小以及设备类型时，它依然能确保元素拥有更恰当的排布行为。

当然 flex 布局应用于移动端不错，PC 端需要全兼容的话，兼容性不够，此处略过不谈。

法二：借助伪元素及 text-align:justify

定义如下 HTML 样式：

```
<div class="container">
    <div class="justify">
        <i>1</i>
        <i>2</i>
        <i>3</i>
        <i>4</i>
        <i>5</i>
    </div>
</div>
```

我们知道，有个 text-align:justify 可以实现两端对齐文本效果。

> text-align CSS属性定义行内内容（例如文字）如何相对它的块父元素对齐。text-align 并不控制块元素自己的对齐，只控制它的行内内容的对齐。
>
> text-align:justify 表示文字向两侧对齐。

一开始我猜测使用它可以实现，采用如下 CSS ：

```
.justify{
  text-align: justify;
}
```

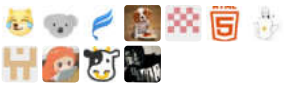
Assignees
No one assigned

Labels
None yet

Projects
None yet

Milestone
No milestone

11 participants



```
.justify i{
  width:24px;
  line-height:24px;
  display:inline-block;
  text-align:center;
  border-radius:50%;
}
```

结果如下：

Demo戳我

没有得到意料之中的结果，并没有实现所谓的两端对齐，查找原因，在 W3C 找到这样一段解释:

最后一个水平对齐属性是 `justify` ，它会带来自己的一些问题。CSS 中没有说明如何处理连字符，因为不同的语言有不同的连字符规则。规范没有尝试去调和这样一些很可能不完备的规则，而是干脆不提这个问题。

额，我看完上面一大段解释还是没明白上面意思，再继续查证，才找到原因：

虽然 `text-align:justify` 属性是全兼容的，但是要使用它实现两端对齐，需要注意在模块之间添加**[空格/换行符/制表符]**才能起作用。

也就是说每一个 *1* 间隙，至少需要有一个空格或者换行或者制表符才行。

好的，我们尝试一下更新一下 `HTML` 结构，采用同样的 CSS：

```
<div class="container">
    <div class="justify">
        <i>1</i>

        <i>2</i>

        <i>3</i>

        <i>4</i>

        <i>5</i>

    </div>
</div>
```

尝试给每一块中间添加一个换行符，结果如下：

Demo戳我

啊哦，还是不行啊。

再寻找原因，原来是出在最后一个元素上面，然后我找到了 `text-align-last` 这个属性， `text-align-last` 属性规定如何对齐文本的最后一行，并且 `text-align-last` 属性只有在 `text-align` 属性设置为 `justify` 时才起作用。

尝试给容器添加 `text-align-last:justify` ：

```
.justify{
  text-align: justify;
  text-align-last: justify; // 新增这一行
}

.justify i{
  width:24px;
  line-height:24px;
  display:inline-block;
  text-align:center;
  border-radius:50%;
}
```

发现终于可以了，实现了多列均匀布局：

Demo戳我

结束了？没有，查看一下 `text-align-last` 的兼容性：

但是一看兼容性，惨不忍睹，只有 IE8+ 和 最新的 chrome 支持 `text-align-last` 属性，也就是说，如果你不是在使用 IE8+ 或者 最新版的 chrome 观看本文，上面 Demo 里的打开的 codePen 例子还是没有均匀分布。

上面说了要使用 `text-align:justify` 实现多列布局，要配合 `text-align-last` ，但是它的兼容性又不好，真的没办法了么，其实还是有的，使用伪元素配合，不需要 `text-align-last` 属性。

我们给 `class="justify"` 的 `div` 添加一个伪元素：

```
.justify{
  text-align: justify;
}

.justify i{
  width:24px;
  line-height:24px;
  display:inline-block;
  text-align:center;
  border-radius:50%;
}

.justify:after {
  content: "";
  display: inline-block;
  position: relative;
  width: 100%;
}
```

去掉了 `text-align-last: justify` 了，增加一个伪元素，效果如下：

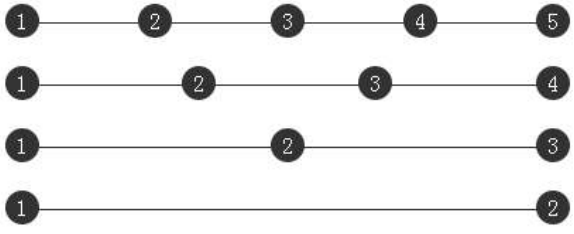
Demo戳我，任意列数均匀布局

通过给伪元素 `:after` 设置 `inline-block` 设置宽度 `100%` ，配合容器的 `text-align: justify` 就可以轻松实现多列均匀布局了。再多配合几句 hack 代码，可以实现兼容到 IE6+ ，最重要的是代码不长，很好理解。

那么为什么使用了 :after 伪元素之后就可以实现对齐了呢？

原因在于 justify 只有在存在第二行的情况下，第一行才两端对齐，所以在这里，我们需要制造一个假的第二行，而 :after 伪元素正好再适合不过。

最终实现题目初始所示：



Demo戳我，任意列数均匀布局

7、消失的边界线问题

看看下图，经常会在一些导航栏中见到，要求每行中最后一列的右边框消失，如何在所有浏览器中最便捷优雅的实现？

如果是不需要兼容 `IE8-` ，那么使用 CSS3 新增的选择器无疑是一种好方法。

```
// 使用伪类选择器，选择第 3n 个元素去掉边框
li:nth-child(3n){
  border-right:none;
}
```

当然，如果个数确定也不多，给需要去掉右边框的元素直接添加一个特定的 class 也就完事。不过这样不够优雅。

这里有个小技巧，就是通过添加反向添加边框并且增加一个负的 `margin` 来实现。

首先，假定我们的 `ul` 结构如下：

```
<div class="ul-container">
    <ul>
        <li>测试</li>
        <li>消失</li>
        <li>边界线</li>
        <li>右侧</li>
        <li>边界线</li>
        <li>消失</li>
        <li>测试</li>
    </ul>
</div>
```

如图中所示，假定每行排列 3 个 `li` ，每个 `li` 宽 `100px` ，我们的 ul 和 ul-container 宽度都设为 `300px` 。

最重要的是，每个 `li` 设置一个左边框而不是右边框：

```
.ul-container,
ul{
  width:300px;
}

li{
  float:left;
  width:99px;
  border-left:1px solid #999;
}
```

我们会得到如下这样的结果：

接下来，我们将容器 `ul-container` 设置为 `overflow:hidden` ，并且将 `ul` 左移一个像素 `margin-left:-1px` 。

这样 `ul` 中第一列的所有边框都因为左移了一像素并且被 `overflow:hidden` 而消失了，造成了下一个 `li` 的右边框看着像左边框一样，其实只是个障眼法：

```
.ul-container{
  overflow:hidden;
}
ul{
  margin-left:-1px;
}
```

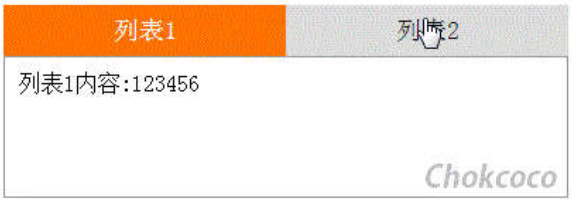
效果图就如一开始图示所示：

Demo戳我

这种做法可以适应不同 `li` 个数的所有情况，因为每个新添加的 `li` ，都会生成一个左边框与上一个 `li` 元素分开，只是在视觉上看上去像是上一个 `li` 元素的右边框。

8、纯CSS的导航栏Tab切换方案

不用 `Javascript` ，使用纯 `CSS` 方案，实现类似下图的导航栏切换：



CSS 的强大之处有的时候超乎我们的想象，Tab 切换，常规而言确实需要用到一定的脚本才能实现。下面看看如何使用 CSS 完成同样的事情。

实现 Tab 切换的难点在于如何使用 CSS 接收到用户的点击事情并对相关的节点进行操作。即是：

1. 如何接收点击事件
2. 如何操作相关DOM

下面看看如何使用两种不同的方法实现需求：

法一： `:target` 伪类选择器

首先，我们要解决的问题是 `如何接收点击事件` ，这里第一种方法我们采用 `:target` 伪类接收。

> `:target` 是 CSS3 新增的一个伪类，可用于选取当前活动的目标元素。当然 URL 末尾带有锚名称 #，就可以指向文档内某个具体的元素。这个被链接的元素就是目标元素(target element)。它需要一个 id 去匹配文档中的 target 。

解释很难理解，看看实际的使用情况，假设我们的 `HTML` 代码如下：

```
<ul class='nav'>
    <li>列表1</li>
    <li>列表2</li>
</ul>
<div>列表1内容:123456</div>
<div>列表2内容:abcdefgkijkl</div>
```

由于要使用 `:target` ，需要 HTML 锚点，以及锚点对应的 HTML 片段。所以上面的结构要变成：

```
<ul class='nav'>
    <li><a href="#content1">列表1</a></li>
    <li><a href="#content2">列表2</a></li>
</ul>
<div id="content1">列表1内容:123456</div>
<div id="content2">列表2内容:abcdefgkijkl</div>
```

这样，上面 `<a href="#content1">` 中的锚点 `#content1` 就对应了列表1 `<div id="content1">` 。锚点2与之相同对应列表2。

接下来，我们就可以使用 `:target` 接受到点击事件，并操作对应的 DOM 了：

```
#content1,
#content2{
    display:none;
}

#content1:target,
#content2:target{
    display:block;
}
```

上面的 CSS 代码，一开始页面中的 `#content1` 与 `#content2` 都是隐藏的，当点击列表1触发 `href="#content1"` 时，页面的 URL 会发生变化：

1. 由 `www.example.com` 变成 `www.example.com#content1`
2. 接下来会触发 `#content1:target{ }` 这条 CSS 规则， `#content1` 元素由 `display:none` 变成 `display:block` ，点击列表2亦是如此。

如此即达到了 Tab 切换。当然除了 `content1 content2` 的切换，我们的 `li` 元素样式也要不断变化，这个时候，就需要我们在 DOM 结构布局的时候多留心，在 `#content1:target` 触发的时候可以同时去修改 `li` 的样式。

在上面 `HTML` 的基础上，再修改一下，变成如下结构：

```
<div id="content1">列表1内容:123456</div>
<div id="content2">列表2内容:abcdefgkijkl</div>
<ul class='nav'>
    <li><a href="#content1">列表1</a></li>
    <li><a href="#content2">列表2</a></li>
</ul>
```

仔细对比一下与上面结构的异同，这里我只是将 `ul` 从两个 `content` 上面挪到了下面，为什么要这样做呢？

因为这里需要使用兄弟选择符 ~ 。

> ~~EF{ cssRules } ，CSS3 兄弟选择符(EF)~~，选择 E 元素后面的所有兄弟元素 F。
>
> 注意这里，最重要的一句话是 E~F 只能选择 E 元素 **之后** 的 F 元素，所以顺序就显得很重要了。

在这样调换位置之后，通过兄弟选择符 ~ 可以操作整个 `.nav` 的样式。

```
#content1:target ~ .nav li{
    // 改变li元素的背景色和字体颜色
    &:first-child{
        background:#ff7300;
        color:#fff;
    }
}

#content2:target ~ .nav li{
    // 改变li元素的背景色和字体颜色
    &:last-child{
        background:#ff7300;
        color:#fff;
    }
}
```

上面的 CSS 规则中，我们使用 ~ 选择符，在 `#content1:target` 和 `#content2:target` 触发的时候分别去控制两个导航 `li` 元素的样式。

至此两个问题， `1. 如何接收点击事件` 与 `2. 如何操作相关DOM` 都已经解决，剩下的是一些小样式的修补工作。

Demo戳我：纯CSS导航切换(:target伪类实现)

法二： `<input type="radio"> && <label for="">`

上面的方法通过添加 `<a>` 标签添加页面锚点的方式接收点击事件。

这里还有一种方式能够接收到点击事件，就是拥有 `checked` 属性的表单元素， `<input type="radio">` 或者 `<input type="checkbox">` 。

假设有这样的结构：

```
<input class="nav1" type="radio">

<ul class='nav'>
    <li>列表1</li>
</ul>
```

对于上面的结构，当我们点击 `<input class="nav1" type="radio">` 单选框表单元素的时候，使用 `:checked` 是可以捕获到点击事件的。

```
.nav1:checked ~ .nav li {
  // 进行样式操作
}
```

> 同样用到了兄弟选择符 ~

这样，当接收到表单元素的点击事件时，可以通过兄弟选择符 ~ 操作它的兄弟元素的样式。

但是，这里有个问题 **我们的 Tab 切换，要点击的是 `<li>` 元素，而不是表单元素**，所以这里很重要的一点是，使用 `<label for="">` 绑定表单元素。看看如下结构：

```
<input class="nav1" id="li1" type="radio">

<ul class='nav'>
    <li><label for="li1">列表1</label></li>
</ul>
```

通过使用 `<label>` 包裹一个 `<li>` 元素，而 `<label>` 有一个属性 `for` 可以绑定一个表单元素。

上面的 `<li>` 中，有一层 `<label for="li">` ，里面的 **`for="li1"`** 意思是绑定 id 为li1 的表单元素。

> label 标签中的 for 定义：for 属性规定 label 与哪个表单元素绑定。

这样改造之后，当我们点击 `<li>` 元素的时候，相当于点击了 `<input class="nav1" id="li1" type="radio">` 这个单选框表单元素，而这个表单元素被点击选中的时候，又可以被 `:checked` 伪类捕获到。

这个时候，我们就可以将页面上的表单元素隐藏，做到点击 `<li>` 相当于点击表单：

```
input{
    display:none;
}
```

这样，应用到本题目，我们应该建立如下 DOM 结构：

```
<div class="container">
    <input class="nav1" id="li1" type="radio" name="nav">
    <input class="nav2" id="li2" type="radio" name="nav">
    <ul class='nav'>
        <li class='active'><label for="li1">列表1</label></li>
        <li><label for="li2">列表2</label></li>
    </ul>
    <div class="content">
        <div class="content1">列表1内容:123456</div>
        <div class="content1">列表2内容:abcdefgkijkl</div>
    </div>
</div>
```

使用两个单选框，分别对应两个导航选项，运用上面介绍的 label 绑定表单， `:checked` 接收点击事件，可以得到第二解法。

看看最后的结果：

Demo戳我：纯CSS导航切换(label 绑定 input:radio && ~)

9、巧妙的多列等高布局

规定下面的布局，实现多列等高布局。

```
<div id="container">
    <div class="left">多列等高布局左</div>
    <div class="right">多列等高布局右</div>
</div>
```

多列等高布局，算是比较常见的一种布局，要求两列布局看上去高度一致（**就是通常是两列背景色一致**）。

如果只是两列背景颜色高度一致，有很多方法可以实现。

法一、使用 `display:flex` 的方式实现

Demo戳我: `display:flex`

法二、使用正负 margin 与 padding 相冲的方式实现

Demo戳我:正负 margin 与 padding 相冲

法三、父容器设置背景色实现

Demo戳我:父容器设置背景色

法四、父容器多重背景色--线性渐变

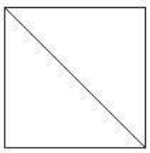
Demo戳我:线性渐变

法五、 `display:table-cell` 实现

Demo戳我:display:table-cell

10、CSS 斜线的实现

如何使用单个标签，实现下图所示的斜线效果。



这种类似于表格的斜线效果，细细研究一下，还是有一些挺有趣的方法可以实现之。

我们假定我们的 `HTML` 结构如下：

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

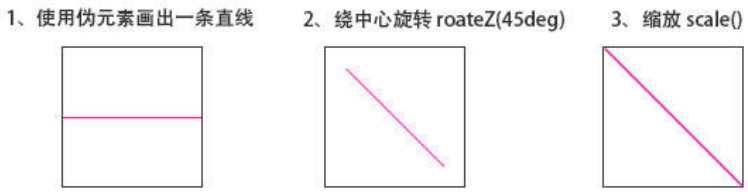
假定高宽各为 100px，在单个标签局限内，看看能有多少种方法实现。

法一、CSS3 旋转缩放

这个应该属于看到需求第一眼就可以想到的方法了。

这里我们使用 **伪元素** 画出一条直线，然后绕 div 中心旋转 45deg ，再缩放一下就可以得到。

简单的一张流程图：



Demo戳我:CSS3旋转缩放斜线

法二、线性渐变实现

这种方法使用了背景的线性渐变实现，渐变背景很重要的一点是，**虽然名字唤作渐变，但是也是可以画出实色而非渐变色。**

我们选定线性渐变的方向为 45deg，依次将渐变色值设为： `transparent -> deeppink -> deeppink -> transparent` 。

> `transparent` 为透明色值。

就像这样简单的一句，即可实现斜线效果：

```
background:
    linear-gradient(45deg, transparent 49.5%, deeppink 49.5%, deeppink 50.5%, transparent 50.5%);
```

Demo戳我:CSS斜线（线性渐变实现）

法三、伪元素+三角形

接下来两种方法就有点为了斜线而斜线的感觉。

利用 CSS border ，是可以轻松实现一个类似这样的三角形的：

CSS 代码如下：

```
div{
  border:50px solid transparent;
  border-left:50px solid deeppink;
  border-bottom:50px solid deeppink;
}
```

这里，我们使用 `div` 的两个 `伪元素` 画出两个大小不一的三角形，然后通过叠加在一起的方式，实现一条斜线。

类似这样，配合 div 的白色底色，即可得到一条斜线：

Demo戳我:CSS斜线（伪元素+三角形实现）

法四、clip-path

`clip-path` 是啥？可以算是 CSS3 的新增属性，或者准确来说是 SVG 的 的 CSS 版本。

使用 `clip-path` ，我们可以定义任意想要的剪裁路径。

> 本文不深入探讨 `clip-path` ，可以先移步 MDN 或者其他关于 clip-path 讲解的文章学习一下。

使用 `clip-path` 的多边形规则 `polygen` ，也可以轻松制作一个三角形（本题中，我们依然借助伪元素来使用 `clip-path` ）：

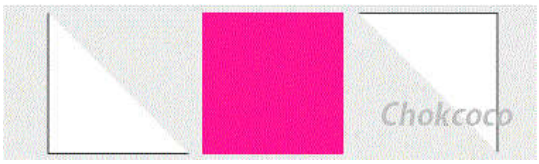
CSS 代码如下：

```
div {
        width: 100px;
        height: 100px;
        -webkit-clip-path: polygon(0 0, 0 100px, 100px 100px, 0 0);
        background: deeppink;
}
```

可以看到 CSS 代码，主要 `polygon(0 0, 0 100px, 100px 100px, 0 0)` 中，其实是一系列路径坐标点，整个图形就是由这些点围起来的区域。

所以使用 `clip-path` 加上两个伪元素我们可以像 `解法三` 一样制作出斜线。

当然，我们也可以换一种方法，殊途同归，解法三也可以这样做，看看下面的效果图：



Demo戳我:CSS斜线（clip-path）

25 6 3 2

AntHp commented on 5 Oct 2016 • edited

7、消失的边界线问题，我在《css设计指南》看到过，有一种巧妙的方法。
li+li{border-left: 1px solid #000;}

chokcoco commented on 8 Oct 2016 • edited Owner

@AntHp
使用 `li+li{}` 好方法，只是这种应该是只适用于单行多列。

fengcms commented on 9 Oct 2016

在项目中有做到过这种两列布局的方法。当时是利用的display:table的方式去做的。

chokcoco commented on 10 Oct 2016 Owner

@fengcms
display:table 的方法适合单行多列，但是应该不适合多行多列吧。

1

champkeh commented on 11 Oct 2016

@AntHp
你说的方式对于单行的li，确实是非常巧妙。
多行的话，感觉还是使用margin:-1px巧妙一点，不过好像得套个div

Serena211 commented on 24 Oct 2016

@chokcoco 多列等高布局也可以用pseudo 来实现, 可以看下我的代码
http://jsfiddle.net/SerenaL/woomsmuf/28/

chokcoco commented on 24 Oct 2016 Owner

@Serena211
是的，好方法。
这题是还没完成的，伪元素的方式我也想过很多，在想如何阐述比较好。
因为有了 `before` 和 `after` 两个伪元素，相当于多了两个标签，这样解法就很丰富了。像你提供的这种解法，如果只用一个伪元素，再在单个伪元素上运用上线性渐变背景色，也可以达到目的。有的时候有了伪元素，反而选择困难了，算得上是一种幸福的烦恼。 😃

gitFuns commented on 11 Nov 2016

@chokcoco https://css.w3ctech.com/ 这个会过去吗