## 背景

### 给背景设置和去除边框

默认情况下背景会延伸到边框所在区域的下层，比如给一个有背景的元素应用一道老土的虚线边框，就可以看出来。即使你使用的是不透明的实色边框，这个事实也不会有任何改变。在css2.1中，这就是背景的工作原理。图如下：



从css3开始可以采用background-clip属性来调整默认行为带来的不便，即可以设置边框。此属性有3个值：

1. border-box：背景会被元素边框外沿框裁切掉
2. content-box：背景会被裁切到内容框
3. Padding-box：背景会被裁切到内边距的外沿

例子：

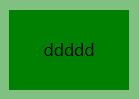




将background-clip的值设置为padding-box或者content-box，产生如下效果(边距还是实际存在)：

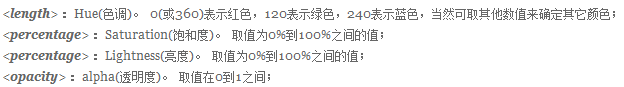


设置为border-box：



例子中hasl函数介绍：

语法：



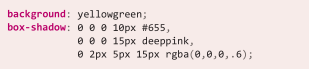
### 设置多重边框

可以通过box-shadow属性来设置多重边框和投影，其语法格式如下：

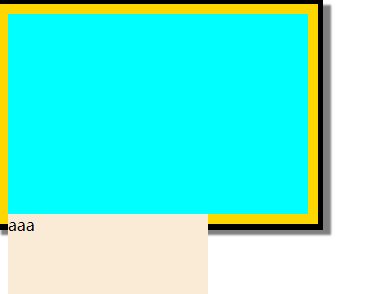


其中第四个参数可以通过指定正值或负值，可以让投影面积加大或者减小。

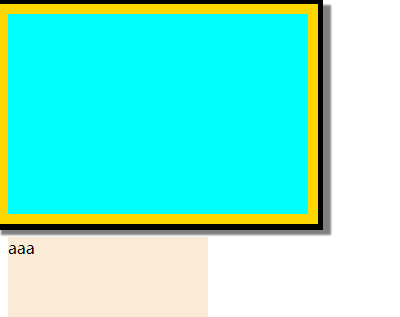
box-shadow 是层层叠加的，第一层投影位于最顶层，依次类推。因此，你需要按此规律调整扩张半径。比如说，在前面的代码中，我们想在外圈再加一道 5px 的外框，那就需要指定扩张半径的值为15px （ 10px+5px ）。如果你愿意，甚至还可以在这些“边框”的底下再加一层常规的投影：



通过此属性设置的边框或者投影，它不会影响到布局，也不会受到box-sizing属性的影响不过，如下所示设置了两个边距和一个投影，一共三个边距，但最终是没有影响到布局的：

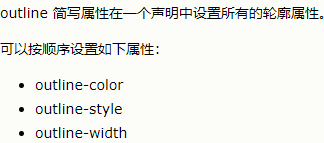


你还是可以通过内边距或外边距（这取决于投影是内嵌和还是外扩的）来额外模拟出边框所需要占据的空间，如下为设置了具体的边距：



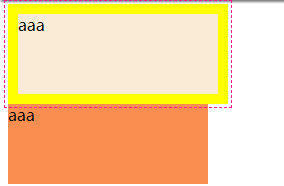
上述方法所创建出的假“边框”出现在元素的外圈。它们并不会响应鼠标事件，比如悬停或点击。如果这一点非常重要，你可以给box-shadow 属性加上 inset 关键字，来使投影绘制在元素的内圈。请注意，此时你需要增加额外的内边距来腾出足够的空隙。

在某些情况下，你可能只需要两层边框，那就可以先设置一层常规边框，再加上outline（描边）属性来产生外层的边框。这种方法的一大优点在于边框样式十分灵活，不像上面的 box-shadow 方案只能模拟实线边框（假设我们需要产生虚线边框效果， box-shadow 就没辙了）。outline属性的语法如下：



还可以使用outline-offset属性来控制外层边框和元素之间的距离，此属性还可以设置负值。

outline和outline-offset的例子：

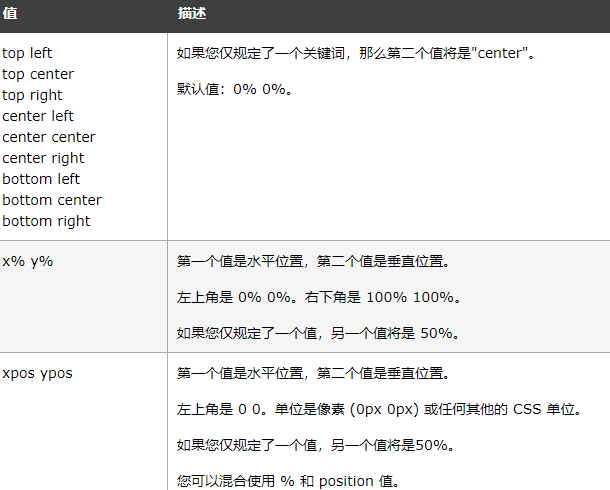


#### 圆角边框

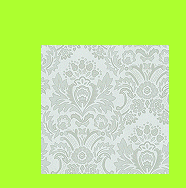
border-radius

### 背景定位

#### background-position

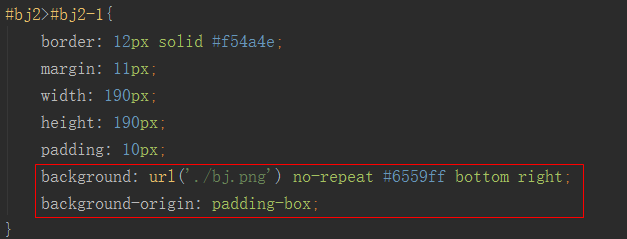
语法格式🡪



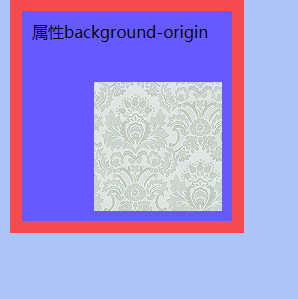


#### background-origin

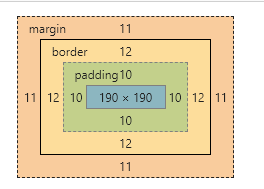
默认情况下， background-position 是以 padding box 为准的，这样边框才不会遮住背景图片。因此， top left 默认指的是 padding box 的左上角。不过，在背景与边框（第三版）（http://w3.org/TR/css3-background）中，我们得到了一个新的属性 background-origin ，可以用它来改变这种行为。在默认情况下，它的值是padding-box 。如果把它的值改成 content-box，我们在 background-position 属性中使用的边角关键字将会以内容区的边缘作为基准（也就是说，此时背景图片距离边角的偏移量就跟内边距保持一致了）。如：



效果如下

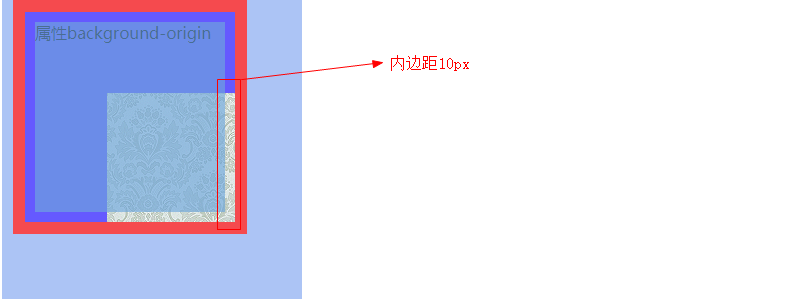


上面例子中的box模型图，如下：



background-origin属性由个值：padding-box、border-box、content-box

上面这个例子设置为border-box和padding-box都是下面这种效果：



注意上面这个例子中最终bj2-1的高度是234（190+12\*2+10\*2=234）

#### calc()函数

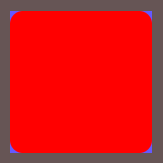
背景图片定位到距离底边 10px 且距离右边 20px 的位置。如果我们仍然以左上角偏移的思路来考虑，其实就是希望它有一个 100% - 20px 的水平偏移量，以及 100% - 10px 的垂直偏移量。谢天谢地， calc() 函数允许我们执行此类运算，它可以完美地在background-position 属性中使用：



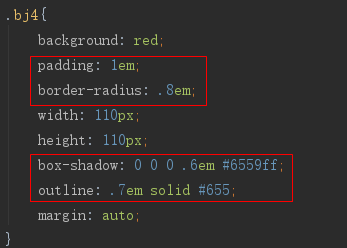
使用calc()函数可以完美的达到background-origin属性的效果

### 边框内圆角

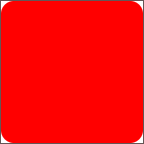
实现如图所示的效果：



关键css代码：



上面得效果主要使用了圆角和描边，圆角是由border-radius来控制的，描边是由box-shadow和outline来控制的。如果不加box-shadow，产生的效果会像下面一样：



通过找个图可以得知，产生那种描边的效果是通过设置box-shadow来填充达到的。主要是设置合适的描边宽度，真好填满图中的空白。关于如何设置，这里有一个定理：

扩展半径需要比描边的宽度值小，但同时又要比(这里的r表示border-radius）。这意味着如果描边的宽度比，那就不可能使用此方式达成描边填充效果的。

### 条纹背景（线性渐变）

#### 横向背景

产生横向的条纹背景：background: linear-gradient(颜色,..)

如：background:linear-gradient(blue,yellow,red,black,green);

具体查看：《css揭秘》---第二章 背景与边框的垂直条纹

#### 竖向背景

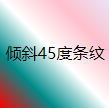


效果如下：



#### 指定倾斜角度的背景





linear-gradient() 和 radial-gradient() 还各有一个循环式的加强版： repeating-linear-gradient() 和 repeating-radial-gradient() 。它们的工作方式跟前两者类似，只有一点不同：色标是无限循环重复的，直到填满整个背景，使用它们可以随心所欲的改变渐变的角度，如下所示：









如果需要产生垂直或竖向条纹，最好是使用上面的linear-gradient

#### 灵活的同色系条纹

根据 CSS 图像（第四版）（http://w3.org/TR/css4-images）计划新增的一个简化语法来看，很快我们就可以在同一个色标上指定两个位置值了。这个简写语法的含义相当于两个连续的色标具有相同的颜色和不同的位置，这个特性在创建渐变图案时是十分有用的。



### 复杂的背景图案

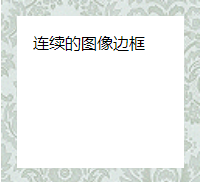
查看《css解密》第2章的背景与边框 6复杂的背景图案

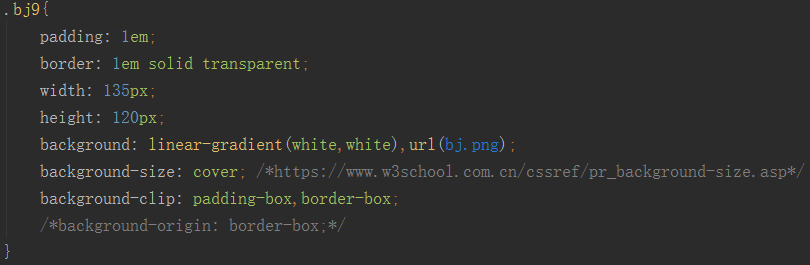
实现复杂的网格图案建议使用svg语法：

<https://philiprogers.com/svgpatterns/#microbialmat>

### 连续的图像边框

实现如下所示的效果：





实现这个效果的思路是，在背景图片之上，再添加一层纯白的实色背景。为了让下层的图片透过边框区域显示出来，我们需要给两层背景指定不同的background-clip。只能在最底层设置背景色，因此需要用一道从白色过渡到白色的 CSS 渐变来模拟出纯白实色背景的效果。

其中tansparent表示边框是透明色彩，这样才能保证底层的背景能够覆盖上来。

background-clip中的一个属性padding-box表示设置的白色渐变从内边距开始裁减掉其它部分（白色边框部分被裁掉），这样就可以让边框部分显示出底层背景；border-box表示设置的底层背景从边框开始裁减掉其他部分（内容和内边距部分被裁掉），这样就可以让内容部分显示出白色背景，因为白色背景上面是文字，所以最终就形成了上面的效果。

关于background-clip属性介绍：

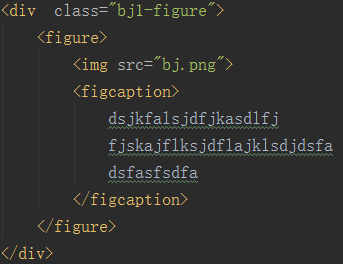
1. border-box：默认值。意味着从边框开始向内裁掉其余部分，最终只保留了边框部分
2. padding-box：从内边距开始裁减掉其余部分，边框会被裁掉
3. content-box：意味着将内容以外的部分都会裁减掉，只留下边距、边框

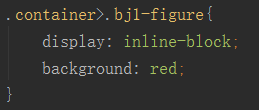
注意：上面3个值前提是有边框和边距

详细用法和background-image的用法参考：查看《css解密》第2章的背景与边框 8连续的图像边框

## 结构和布局

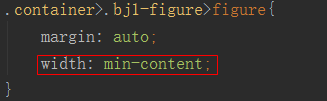
众所周知，如果不给元素指定一个具体的 height ，它就会自动适应其内容的高度。假如我们希望 width 也具有类似的行为，该怎么做呢？举个例子，假设我们用 HTML5 来标记图片元素，结构代码可能是这样的：





效果：

实际上，我们想要这个figure元素和它所包含的图片一样宽（即实现图片和内容一样宽），那么可以采用如下的方式改写css：

最终效果如下所示：



figure标签：

<https://baijiahao.baidu.com/s?id=1631416429989070749&wfr=spider&for=pc>

关于css的4个自适应关键字：

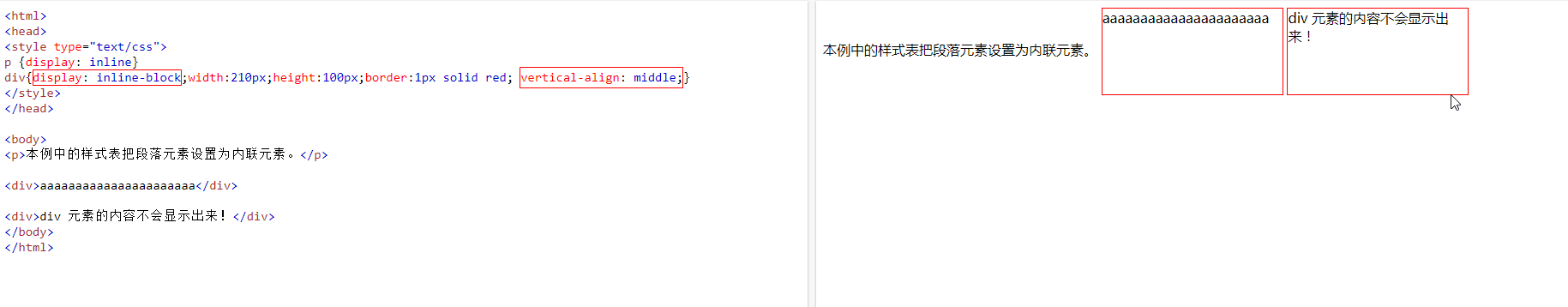
1、（学习时建议阅读）

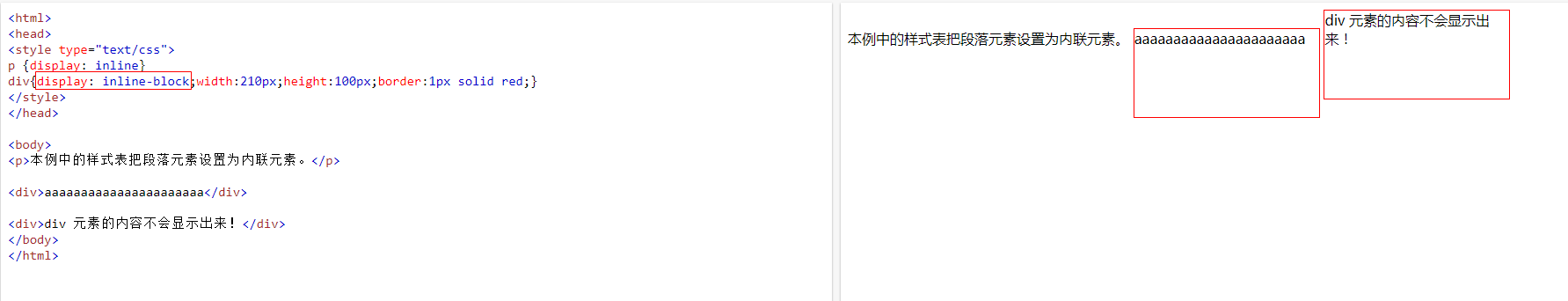
<https://www.zhangxinxu.com/wordpress/2016/05/css3-width-max-contnet-min-content-fit-content/>

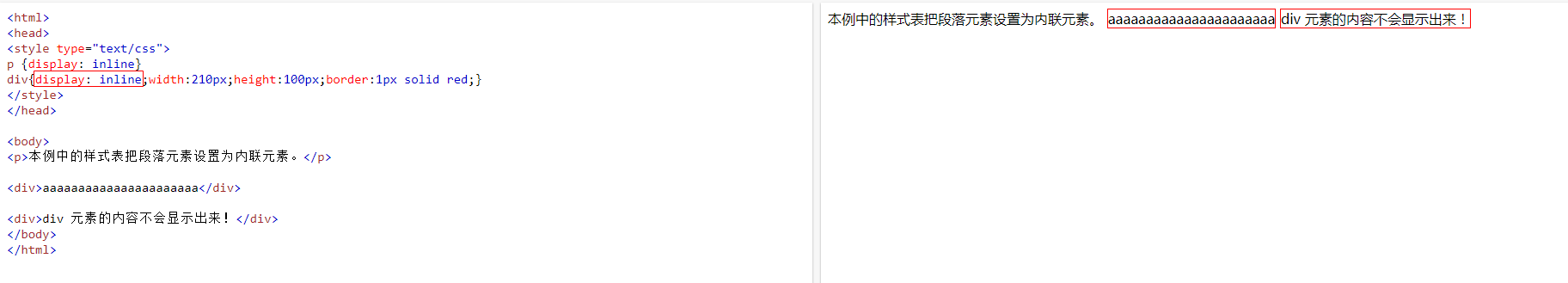
2、<https://www.cnblogs.com/mj878/p/5947541.html>

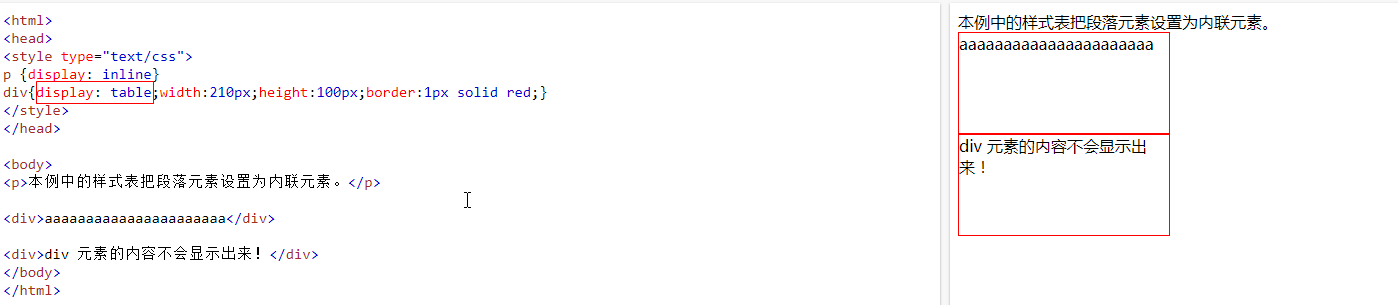
### 关于display的行内、块状、行内快元素

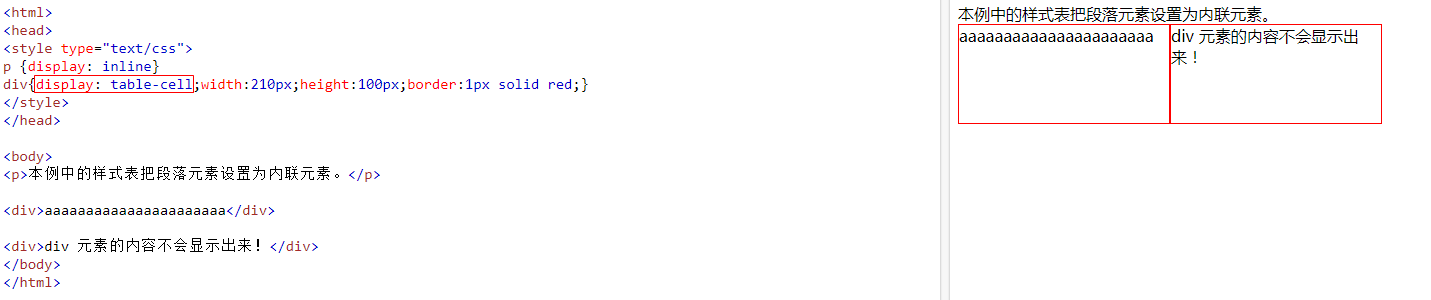
1. 行内（inline）：不能设置元素的宽高，默认宽度是父元素宽度，高度是内容高度
2. 块状（block）：可以自定义设置元素的宽高。多个block元素是从上往下排列
3. 行内块（inline-block）：行内块元素，可以设置宽高，多个元素可以横向排列。它综合了行内和块状的优点

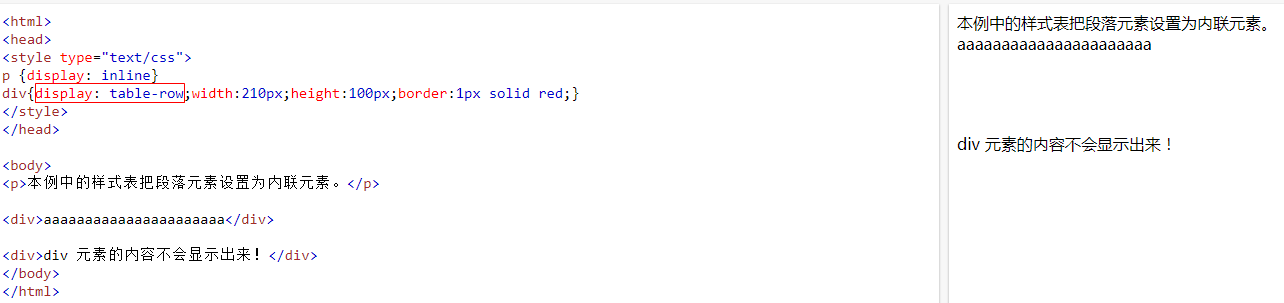












参考：<https://www.cnblogs.com/xinlvtian/p/8822098.html>

display属性详解：<https://segmentfault.com/a/1190000009636727>

### 精确控制表格列宽

Layout-table属性：用来显示表格单元格、行、列的算法规则

<https://www.w3school.com.cn/cssref/pr_tab_table-layout.asp>

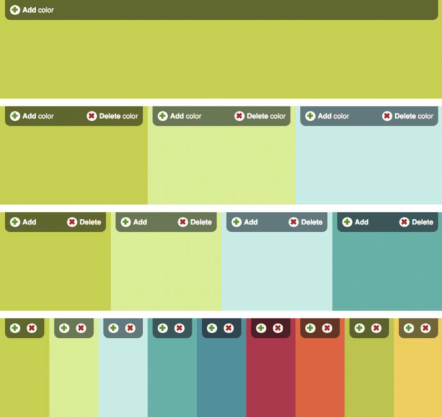
《css揭秘》---第7章 37小节

Overflow属性：规定内容溢出元素后，该如何处理：

<https://www.w3school.com.cn/cssref/pr_pos_overflow.asp>

### 根据兄弟元素的数量来设置样式

问题引出：在某些场景下，我们需要根据兄弟元素的总数来为它们设置样式。如下图所示，当随着色块的数量不断增加该如何设置色块的样式，才能使它们更好的展现在浏览器中。

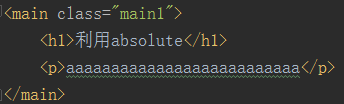


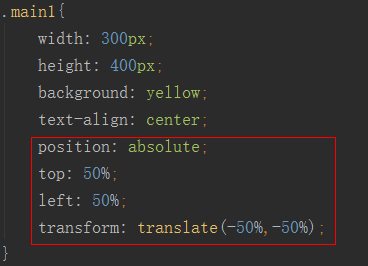
解决方案是使用:nth-child() 和兄弟选择符（ ~ ），具体方案请看-----《css揭秘》第7章 38

### 垂直居中

在 CSS 中对元素进行水平居中是非常简单的：如果它是一个行内元素，就对它的父元素应用 text-align: center ；如果它是一个块级元素，就对它自身应用 margin: auto 。然而如果要对一个元素进行垂直居中，可能光是想想就令人头皮发麻了

#### 基于绝对定位的解决方案

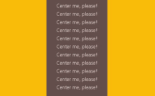




效果如下：



如果需要居中的元素已经在高度上超过了视口，那它的顶部会被视口裁切掉，就可能会像下面这个图示一样：



在某些浏览器中，这个方法可能会导致元素的显示有一些模糊，因为元素可能被放置在半个像素上。这个问题可以用 transform-style: preserve-3d 来修复，不过这个修复手段也可以认为是一个hack，而且很难保证它在未来不会出问题

#### 基于视口单位的解决方案

（不推荐）

#### Flexbox布局

**孙子元素的display为absolute时是相对于祖元素定位吗：**

https://blog.csdn.net/weixin\_41796631/article/details/89604749

flexbox布局语法：

<https://www.runoob.com/w3cnote/flex-grammar.html>

flexbox语法的align-items和align-content的区别：

换行后有多根轴线（轴线的方向和flex-direction是横向还是纵向有关），align-content在只有一根轴线的情况下看不到效果。align-content适用于多根轴线，行与行之间的布局：

https://www.jianshu.com/p/5e7deadd9d9b

https://www.cnblogs.com/yesyes/p/7260462.html

flexbox布局中，元素里面的p标签和div标签位置效果是不一样的，推荐使用div

## css函数

### clamp()函数

作用是把一个值限制在一个上限和下限之间，当这个值超过最小值和最大值的范围时，在最小值和最大值之间选择一个值使用。它接收三个参数：最小值(min)、首选值(middle)、最大值(max)。clamp() 被用在 [<length>](https://developer.mozilla.org/zh-CN/docs/Web/CSS/length)、[<frequency>](https://developer.mozilla.org/zh-CN/docs/Web/CSS/frequency)、[<angle>](https://developer.mozilla.org/zh-CN/docs/Web/CSS/angle)、[<time>](https://developer.mozilla.org/zh-CN/docs/Web/CSS/time)、[<percentage>](https://developer.mozilla.org/zh-CN/docs/Web/CSS/percentage)、[<number>](https://developer.mozilla.org/zh-CN/docs/Web/CSS/number)、[<integer>](https://developer.mozilla.org/zh-CN/docs/Web/CSS/integer) 中都是被允许的。

假如h1的字体大小(用h表示)为：font-size: clamp(16px, 5vw, 34px)，得出16px<5vm/h<34px。依据可视窗口的大小(用t表示)，而决定最终的值。假如 t=300px，这时h=300\*5%=15px，但是事先设置了min=16px，所以h最终为16px；假如t=1400px，这时h=1400\*5%=70px，但是设置了max=34px，所以最终h为34px；假如t=350px，h=350\*5%=17.5，满足公式【16px<5vm/h<34px】，所以h最终为17.5px。