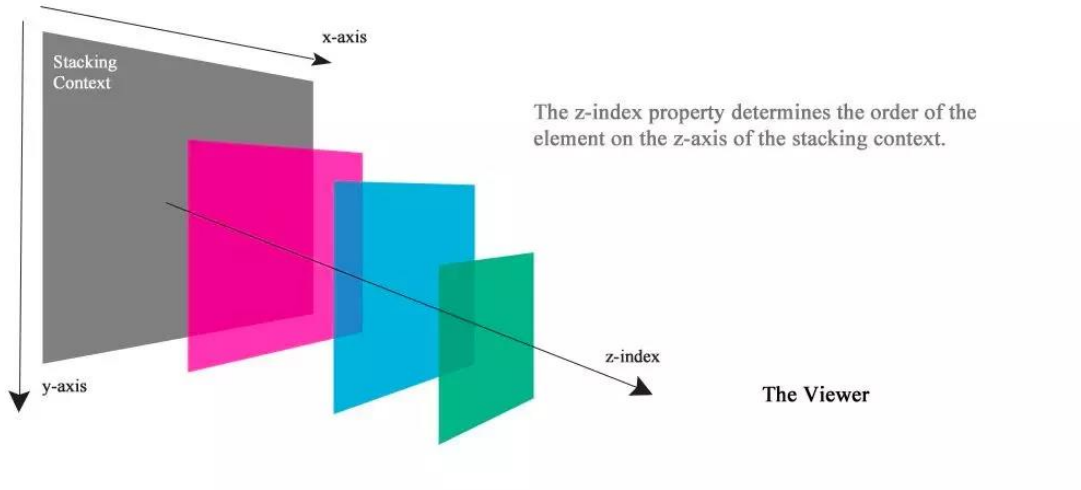
CSS 是如何影响浏览器元素在文档中的排列

之前在项目的过程中遇到了一个问题，某个 `div` 希望始终显示在最上面，而在之后的元素都显示在它之下，当时设置了 `z-index` 也没有效果，不知道什么原因，因此找了一下 `CSS` 相关资料，解决了这个问题的同时，也学习了很多知识，特此和大家分享一下。

屏幕是一个二维平面，然而 HTML 元素却是排列在三维坐标系中， `x` 为水平方向， `y` 为垂直方向， `z` 为屏幕由内向外方向，我们在看屏幕的时候是沿着 `z` 轴方向从外向内的。由此，元素在用户视角就形成了层叠的关系，某个元素可能覆盖了其他元素也可能被其他元素覆盖；



这里有几个重要的概念：**层叠上下文**（堆叠上下文，Stacking Context）、**层叠等级**（层叠水平，Stacking Level）、**层叠顺序**（层叠次序，堆叠顺序，Stacking Order）、**z-index**、**BFC**（块级格式化上下文，Block Formatting Context），这些概念共同决定了你看到元素的位置，下面我们就围绕着这几个概念来一起学习一下。

声明：

1. 以下定位元素指的是 `position:absolute|fixed|relative|sticky`
2. 以下非定位元素指的是 `position:initial|static`

1. 层叠上下文 (Stacking Context)

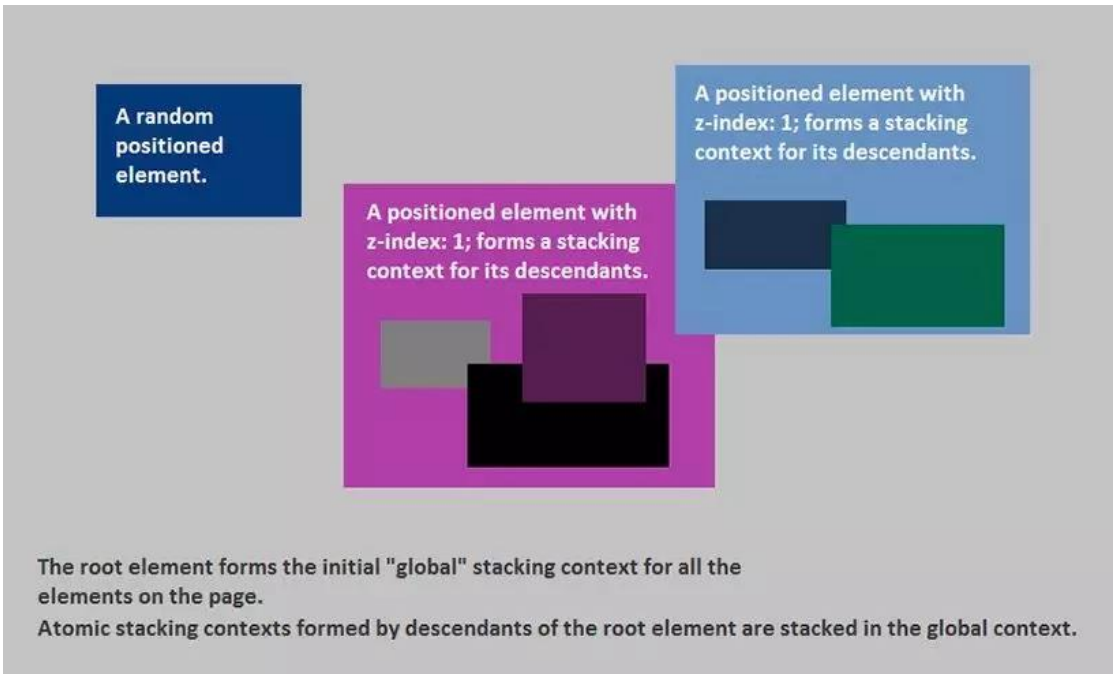
层叠上下文(堆叠上下文，Stacking Context)，是 HTML 中一个三维的概念。在 CSS2.1 规范中，每个元素的位置是三维的，当元素发生层叠，这时它可能覆盖了其他元素或者被其他元素覆盖；排在 `z` 轴越靠上的位置，距离屏幕观察者越近。

文章 <关于 z-index 那些你不知道的事> 有一个很好的比喻，这里引用一下；

可以想象一张桌子，上面有一堆物品，这张桌子就代表着一个层叠上下文。如果在第一张桌子旁还有第二张桌子，那第二张桌子就代表着另一个层叠上下文。现在想象在第一张桌子上有四个小方块，他们都直接放在桌子上。在这四个小方块之上有一片玻璃，而在玻璃片上有一盘水果。这些方块、玻璃片、水果盘，各自都代表着层叠上下文中一个不同的层叠层，而这个层叠上下文就是桌子。

每一个网页都像一个房间，这个房间就是 `<html></html>`，其他层叠上下文就像这个房间里的桌子，HTML 标签中的一切都被置于这个房间中。

当给一个元素的 `position` 值赋为 `fixed` 或 `sticky` 值时，你就创建了一个新的层叠上下文，其中有着独立于页面上其他层叠上下文和层叠层的层叠层，这就相当于你把另一张桌子带到了房间里。



层叠上下文 1 (Stacking Context 1)是由文档根元素形成的， 层叠上下文 2 和 3 (Stacking Context 2, 3) 都是层叠上下文 1 (Stacking Context 1) 上的层叠层。他们各自也都形成了新的层叠上下文，其中包含着新的层叠上下文。

在层叠上下文中，其子元素按照上面解释的规则进行层叠。形成层叠上下文的方法有：

- 根元素 `<html></html>`;
- `position` 值为 `absolute|relative`，且 `z-index` 值不为 `auto`;
- `position` 值为 `fixed|sticky`;
- `z-index` 值不为 `auto` 的 flex 元素，即父元素属性 `display:flex|inline-flex`;
- `opacity` 属性值小于 `1` 的元素;
- `transform` 属性值不为 `none` 的元素;
- `mix-blend-mode` 属性值不为 `normal` 的元素;

- `filter`、`perspective`、`clip-path`、`mask`、`mask-image`、`mask-border`、`motion-path`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；
- `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`属性设置为`touch`的元素；

总结:

1. 层叠上下文可以包含在其他层叠上下文中，并且一起组建了一个有层级的层叠上下文；
2. 每个层叠上下文完全独立于它的兄弟元素，当处理层叠时只考虑子元素，类似于 BFC；
3. 每个层叠上下文是自包含的：当元素的内容发生层叠后，整个该元素将会**在父级叠上下文**中按顺序进行层叠；

2. 层叠等级（Stacking Level）

层叠等级（层叠水平，Stacking Level）决定了在同一个层叠上下文中，元素在`z`轴上的显示的顺序；

1. 普通元素的层叠等级优先由其所在的层叠上下文决定；
2. 层叠等级的比较，只有在同一个层叠上下文元素中才有意义；
3. 在同一个层叠上下文中，层叠等级描述定义的是该层叠上下文中的元素在`z`轴上的上下顺序；

对于普通元素的层叠水平探讨只局限于在当前层叠上下文中：

> 层叠上下文本身是一个强力的「层叠结界」，普通的元素水平是无法突破这个结界和结界外的元素去较量层叠水平的。
>
> — CSS 世界

另外，层叠等级并不一定由 `z-index` 决定，只有定位元素的层叠等级才由 `z-index` 决定，其他类型元素的层叠等级由层叠顺序、他们在 HTML 中出现的顺序、他们的祖先元素的层叠等级一同决定，详细的规则见下面层叠顺序的介绍。

3. z-index

> 在 CSS 2.1 中，所有的盒模型元素都处于三维坐标系中。除了我们常用的横坐标和纵坐标， 盒模型元素还可以沿着「z 轴」层叠摆放，当他们相互覆盖时，z 轴顺序就变得十分重要。
>
> -- CSS 2.1 Section 9.9.1 - Layered presentation

`z-index` 只适用于定位的元素，对非定位元素无效，它可以被设置为正整数、负整数、 `0`、 `auto`，如果一个定位元素没有设置 `z-index`，那么默认为 `auto`;

元素的 `z-index` 值只在同一个层叠上下文中有意义。如果父级层叠上下文的层叠等级低于另一个层叠上下文的，那么它 `z-index` 设的再高也没用。所以如果你遇到 `z-index` 值设了很大，但是不起作用的话，就去看看它的父级层叠上下文是否被其他层叠上下文盖住了。

4. 层叠顺序 (Stacking Order)

层叠顺序（层叠次序，堆叠顺序，Stacking Order）描述的是元素在同一个层叠上下文中的顺序**规则**（之前的层叠上下文和层叠等级是概念），从层叠的底部开始，共有七种层叠顺序：

1. **背景和边框**：形成层叠上下文的元素的背景和边框。
2. **负 z-index 值**：层叠上下文内有着负 `z-index` 值的定位子元素，负的越大层叠等级越低；
3. **块级盒**：文档流中的块级、非定位子元素；
4. **浮动盒**：非定位浮动元素；
5. **行内盒**：文档流中行内、非定位子元素；
6. **z-index: 0**：`z-index` 为 `0` 或 `auto` 的定位元素， 这些元素形成了新的层叠上下文；
7. **正 z-index 值**： `z-index` 为正的定位元素，正的越大层叠等级越高；

第 7 级顺序的元素会显示在之前顺序元素的上方，也就是看起来覆盖了更低级的元素：

除层叠顺序优先级规则之外，还有一条**后来居上**规则：同一个层叠顺序的元素按照在 HTML 里出现的顺序依次层叠。这两个规则共同决定浏览器元素在文档中是如何层叠的。

5. 文档流 (Document Flow)

5.1 常规流 (Normal flow)

- 在常规流中，盒一个接着一个排列;
- 在块级格式化上下文里面， 它们竖着排列;
- 在行内格式化上下文里面， 它们横着排列;
- 当 `position` 为 `static` 或 `relative`，并且 `float` 为 `none` 时会触发常规流;
- 对于静态定位(static positioning)， `position:static`，盒的位置是常规流布局里的位置;
- 对于相对定位(relative positioning)， `position:relative`，盒偏移位置由 `top`、`bottom`、`left`、`right` 属性定义。即使有偏移，仍然保留原有的位置，其它常规流不能占用这个位置。

5.2 浮动 (Floats)

1. 左浮动元素尽量靠左、靠上，右浮动同理;
2. 这导致常规流环绕在它的周边，除非设置 `clear` 属性;
3. 浮动元素不会影响块级元素的布局;
4. 但浮动元素会影响行内元素的布局，让其围绕在自己周围，撑大父级元素，从而间接影响块级元素布局;
5. 最高点不会超过当前行的最高点、它前面的浮动元素的最高点;

6. 不超过它的包含块，除非元素本身已经比包含块更宽；
7. 行内元素出现在左浮动元素的右边和右浮动元素的左边，左浮动元素的左边和右浮动元素的右边是不会摆放浮动元素的；

5.3 绝对定位 (Absolute positioning)

1. 绝对定位方案，盒从常规流中被移除，不影响常规流的布局；
2. 它的定位相对于它的包含块，相关 CSS 属性：`top`、`bottom`、`left`、`right`；
3. 如果元素的属性 `position` 为 `absolute` 或 `fixed`，它是绝对定位元素；
4. 对于 `position:absolute`，元素定位将相对于上级元素中最近的一个 `relative`、`fixed`、`absolute`，如果没有则相对于 body；

6. BFC （Block Formatting Context）

6.1 什么是 BFC

BFC (Block Formatting Context) 块级格式化上下文，是用于布局块级盒子的一块渲染区域，相对应的还有 IFC（Inline Formatting Context）内联格式化上下文，不是本文重点，读者可以自行查阅相关知识。

> BFC 是 Web 页面 CSS 视觉渲染的一部分，用于决定块盒子的布局及浮动相互影响范围的一个区域。
>
> — MDN - 块格式化上下文

一个 BFC 的范围包含创建该上下文元素的所有子元素，但**不包括**创建了新 BFC 的子元素的内部元素。这从另一方角度说明，一个元素不能同时存在于两个 BFC 中。因为如果一个元素能够同时处于两个 BFC 中，那么就意味着这个元素能与两个 BFC 中的元素发生作用，就违反了 BFC 的隔离作用。

触发 BFC 的方式有：

1. 根元素，即 HTML 标签;
2. 浮动元素，即 `float` 值为 `left`、 `right`;
3. `overflow` 值不为 `visible`，即值为 `auto`、 `scroll`、 `hidden`;
4. `display` 值为 `inline-block`、`table-cell`、`table-caption`、`table`、`inline-table`、 `flex`、 `inline-flex`、 `grid`、 `inline-grid`;
5. 定位元素： `position` 值为 `absolute`、 `fixed`;
6. `contain` 为 `layout`、 `content`、 `paint` 的元素;

注意： `display:table` 也可以生成 BFC 的原因在于 Table 会默认生成一个匿名的 `table-cell`，是这个匿名的 `table-cell` 生成了 BFC。

6.2 用法

1. 阻止相邻元素的 margin 合并

属于同一个 BFC 的两个相邻块级子元素的上下 `margin` 会发生重叠，(设置 `writing-mode:tb-rl` 时，水平 `margin` 会发生重叠)。所以当两个相邻块级子元素分属于不同的 BFC 时可以阻止 `margin` 重叠。可以给任一个相邻块级盒子的外面包一个 div，通过改变此 div 的属性使两个原盒子分属于两个不同的 BFC，以此来阻止 `margin` 重叠。

2. 阻止元素被浮动元素覆盖

一个正常文档流的块级元素可能被一个 `float` 元素覆盖，挤占正常文档流，因此可以设置一个元素的 `float`、 `display`、 `position` 值等方式触发 BFC，以阻止被浮动盒子覆盖。

3. 包含浮动元素

通过改变包含浮动子元素的父盒子的属性值，触发 BFC，以此来包含子元素的浮动盒子。

7. 实战

下面一起来看几个例子实战一下，帮助理解。

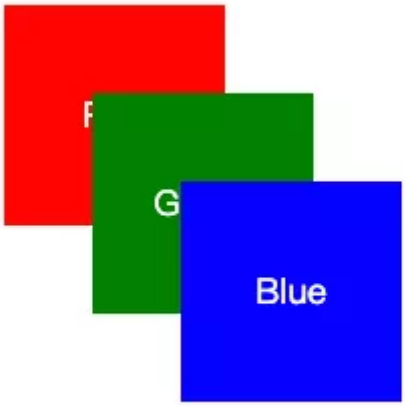
7.1 普通情况

三个 `relative` 定位的 `div` 块中各有 `absolute` 的不同颜色的 `span.red`、`span.green`、 `span.blue`，它们都设置了 `position:absolute`；

那么当没有元素包含 `z-index` 属性时，这个例子中的元素按照如下顺序层叠（从底到顶顺序）：

1. 根元素的背景和边界；
2. 块级非定位元素按 HTML 中的出现顺序层叠；
3. 行内非定位元素按 HTML 中的出现顺序层叠；
4. 定位元素按 HTML 中的出现顺序层叠；

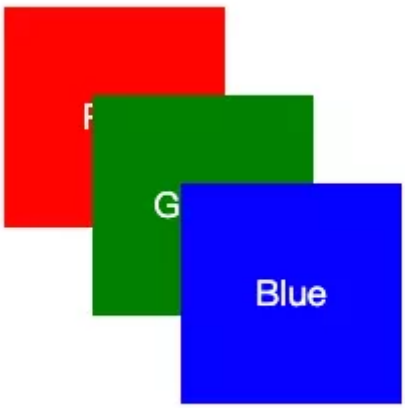
红绿蓝都属于 `z-index` 为 `auto` 的定位元素，因此按照 7 层层叠顺序规则来说同属于层叠顺序第 6 级，所以按 HTML 中的出现顺序层叠：`红->绿->蓝`



7.2 在相同层叠上下文的父元素内的情况

红绿位于一个 `div.first-box` 下，蓝位于 `div.second-box` 下，红绿蓝都设置了 `position:absolute`， `first-box` 与 `second-box` 都设置了 `position:relative`;

这个例子中，红蓝绿元素的父元素 `first-box` 与 `second-box` 都没有生成新的层叠上下文，都属于根层叠上下文中的元素，且都是层叠顺序第 6 级，所以按 HTML 中的出现顺序层叠：`红->绿->蓝`



7.3 给子元素增加 z-index

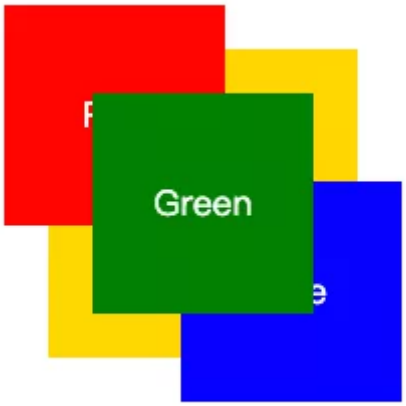
红绿位于一个 `div.first-box` 下，蓝黄位于 `div.second-box` 下，红绿蓝都设置了 `position:absolute`，如果这时给绿加一个属性 `z-index:1`，那么此时 `.green` 位于最上面；

如果再在 `.second-box` 下 `.green` 后加一个绝对定位的 `span.gold`，设置 `z-index:-1`，那么它将位于红绿蓝的下面；

这个例子中，红蓝绿黄元素的父元素中都没有生成新的层叠上下文，都属于根层叠上下文中的元素

1. 红蓝都没有设置 z-index，同属于层叠顺序中的第 6 级，按 HTML 中的出现顺序层叠；
2. 绿设置了正的 z-index，属于第 7 级；
3. 黄设置了负的 z-index，属于第 2 级；

所以这个例子中的从底到高显示的顺序就是：`黄->红->蓝->绿`



7.4 在不同层叠上下文的父元素内的情况

红绿位于一个 `div.first-box` 下，蓝位于 `div.second-box` 下，红绿蓝都设置了 `position:absolute`，如果 `first-box` 的 z-index 设置的比 `second-b`

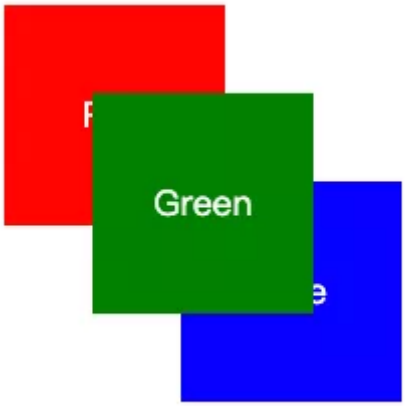
ox 的大，那么此时无论蓝的 z-index 设置的多大 z-index:999，蓝都位于红绿的下面；如果我们只更改红绿的 z-index 值，由于这两个元素都在父元素 first-box 产生的层叠上下文中，此时谁的 z-index 值大，谁在上面；

这个例子中，红绿蓝都属于设置了 z-index 的定位元素，不过他们的父元素创建了新的层叠上下文；

1. 红绿的父元素 first-box 是设置了正 z-index 的定位元素，因此创建了一个层叠上下文，属于层叠顺序中的第 7 级；
2. 蓝的父元素 second-box 也同样创建了一个层叠上下文，属于层叠顺序中的第 6 级；
3. 按照层叠顺序， first-box 中所有元素都排在 second-box 上；
4. 红绿都属于层叠上下文 first-box 中且设置了不同的正 z-index，都属于层叠顺序中第 7 级；
5. 蓝属于层叠上下文 second-box，且设置了一个很大的正 z-index，属于层叠元素中第 7 级；
6. 虽然蓝的 z-index 很大，但是因为 second-box 的层叠等级比 first-box 小，因此位于红绿之下；

   所以这个例子中从低到到显示的顺序：蓝->红->绿

   (我遇到的的情况就属于这个例子类似情形)



7.5 给子元素设置 opacity

红绿位于 `div.first-box` 下，蓝位于 `div.second-box` 下，红绿蓝都设置了 `position:absolute`，绿设置了 `z-index:1`，那么此时绿位于红蓝的最上面；

如果此时给 `first-box` 设置 `opacity:.99`，这时无论红绿的 z-index 设置的多大 `z-index:999`，蓝都位于红绿的上面；

如果再在 `.second-box` 下 `.green` 后加一个 `span.gold`，设置 `z-index:-1`，那么它将位于红绿蓝的下面；

之前已经介绍了，设置 `opacity` 也可以形成层叠上下文，因此：

1. `first-box` 设置了 `opacity`， `first-box` 成为了一个新的层叠上下文；
2. `second-box` 没有形成新的层叠上下文，因此其中的元素都属于根层叠上下文；
3. 黄属于层叠顺序中第 2 级，红绿属于第 7 级， `first-box` 属于第 6 级，蓝属于层叠顺序中第 6 级且按 HTML 出现顺序位于 `first-box` 之上；

   所以这个例子中从低到到显示的顺序：`黄->红->绿->蓝`