深入理解css中position属性及z-index属性

在网页设计中，position属性的使用是非常重要的。有时如果不能认识清楚这个属性，将会给我们带来很多意想不到的困难。

position属性共有四种不同的定位方法，分别是static、fixed、relative、absolute,sticky。最后将会介绍和position属性密切相关的z-index属性。

第一部分：position: static

static定位是HTML元素的默认值，即没有定位，元素出现在正常的流中，因此，这种定位就不会收到top，bottom,left,right的影响。

如html代码如下：

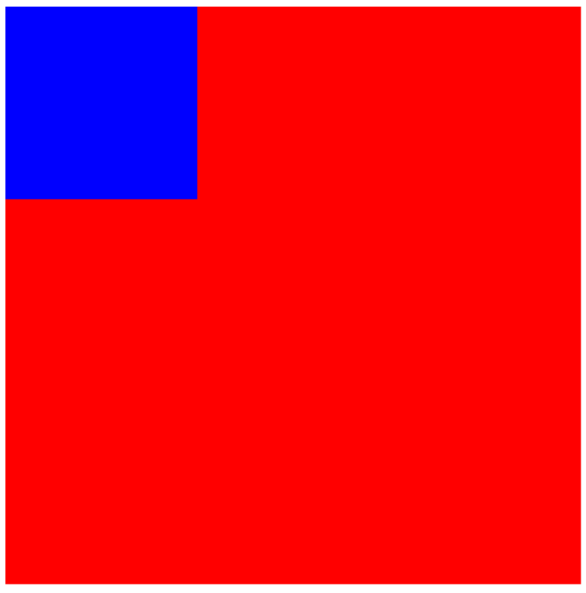
```
<div class="wrap">
    <div class="content"></div>
</div>
```

css代码如下：

```
.wrap{width: 300px;height: 300px; background: red;}

.content{position: static; top:100px; width: 100px;height: 100px; background: blue;}
```

效果图如下：



我们发现，虽然设置了static以及top，但是元素仍然出现在正常的流中。

第二部分：fixed定位

fixed定位是指元素的位置相对于浏览器窗口是固定位置，即使窗口是滚动的它也不会滚动，且fixed定位使元素的位置与文档流无关，因此不占据空间，且它会和其他元素发生重叠。

html代码如下：

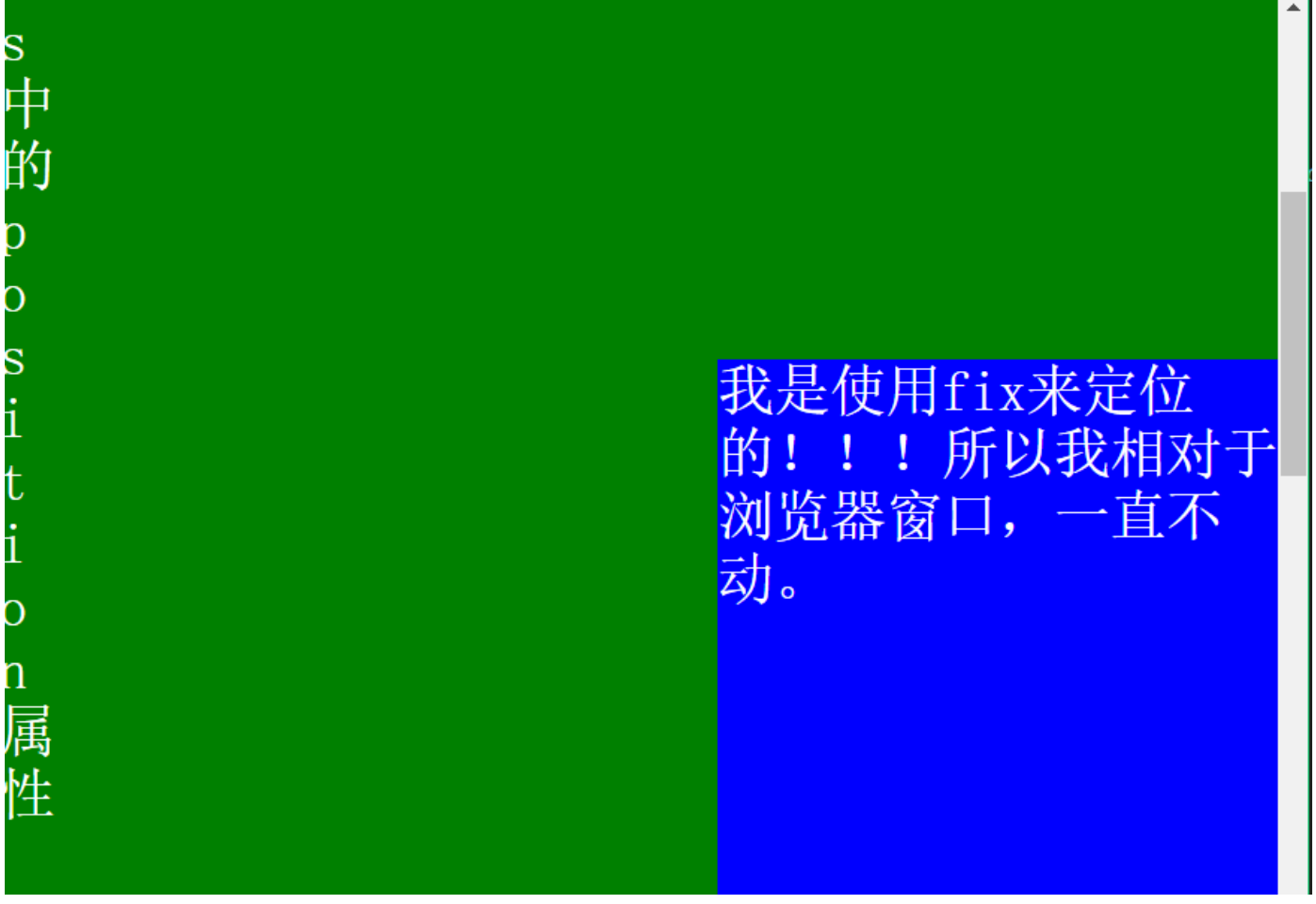
```
<div class="content">我是使用fix来定位的！！！所以我相对于浏览器窗口，一直不动。</div>
```

css代码如下：

```
body{height:1500px; background: green; font-size: 30px; color:white;}

.content{ position: fixed; right:0;bottom: 0; width: 300px;height: 300px; background: blue;}
```

效果图如下：



即右下角的div永远不会动，就像经常弹出来的广告！！！

值得注意的是：fixed定位在IE7和IE8下需要描述！DOCTYPE才能支持。

第三部分：relative定位

相对定位元素的定位是相对它自己的正常位置的定位。

关键：如何理解其自身的坐标呢?

让我们看这样一个例子，hmtl如下：

```
1 <h2>这是位于正常位置的标题</h2>
2 <h2 class="pos_bottom">这个标题相对于其正常位置向下移动</h2>
3 <h2 class="pos_right">这个标题相对于其正常位置向右移动</h2>
```

css代码如下：

```
1 .pos_bottom{position:relative; bottom:-20px;}
2 .pos_right{position:relative;left:50px;}
```

效果图如下：

这是位于正常位置的标题

这个标题相对于其正常位置向左移动

这个标题相对于其正常位置向右移动

即bottom:-20px;； 向下移动。 left:50px;向右移动。

即可以理解为：移动后是移动前的负的位置。

比如上例中，移动后是移动前负的bottom:-20px;即移动后是移动前bottom:20px;也就是说，移动后是移动前的向下20px；

又如： left:50px;移动后是移动前左边的-50px;那么也就是说移动后是移动前的右边的50px。

即：移动后对于移动前：如果值为负数，则直接换成整数；如果值为整数，则直接改变相对方向。

弄清楚了relative是如何移动的，下面我们看一看移动之后是否会产生其他的影响。

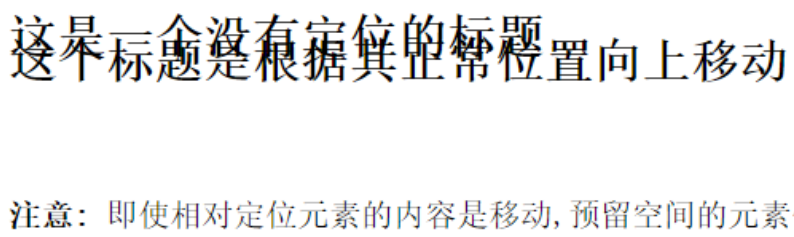
html代码如下：

```
1  <h2>这是一个没有定位的标题</h2>
2  <h2 class="pos_top">这个标题是根据其正常位置向上移动</h2>
3  <p><b>注意:</b> 即使相对定位元素的内容是移动,预留空间的元素仍保存在正常流动。</p>
```

css代码如下：

```
1  h2.pos_top{position:relative;top:-35px;}
```

效果图如下：



根据之前的说法，top:-35px；值是负数，则直接换成正数，即移动后相对与移动前向上偏移了35px；我们发现于上，移动后和上面的元素发生了重叠；于下，即使相对元素的内容移动了，但是预留空间的元素仍然保存在正常流动，也就是说相对移动之后，不会对下面的其他元素造成影响。

注意：top：20px；是指子元素的margin外侧和包裹元素的border内侧之间的距离是20px。

第四部分：absolute定位

绝对定位的元素相对于最近的已定位父元素，如果元素没有已定位的父元素，那么它的位置相对于<html>。

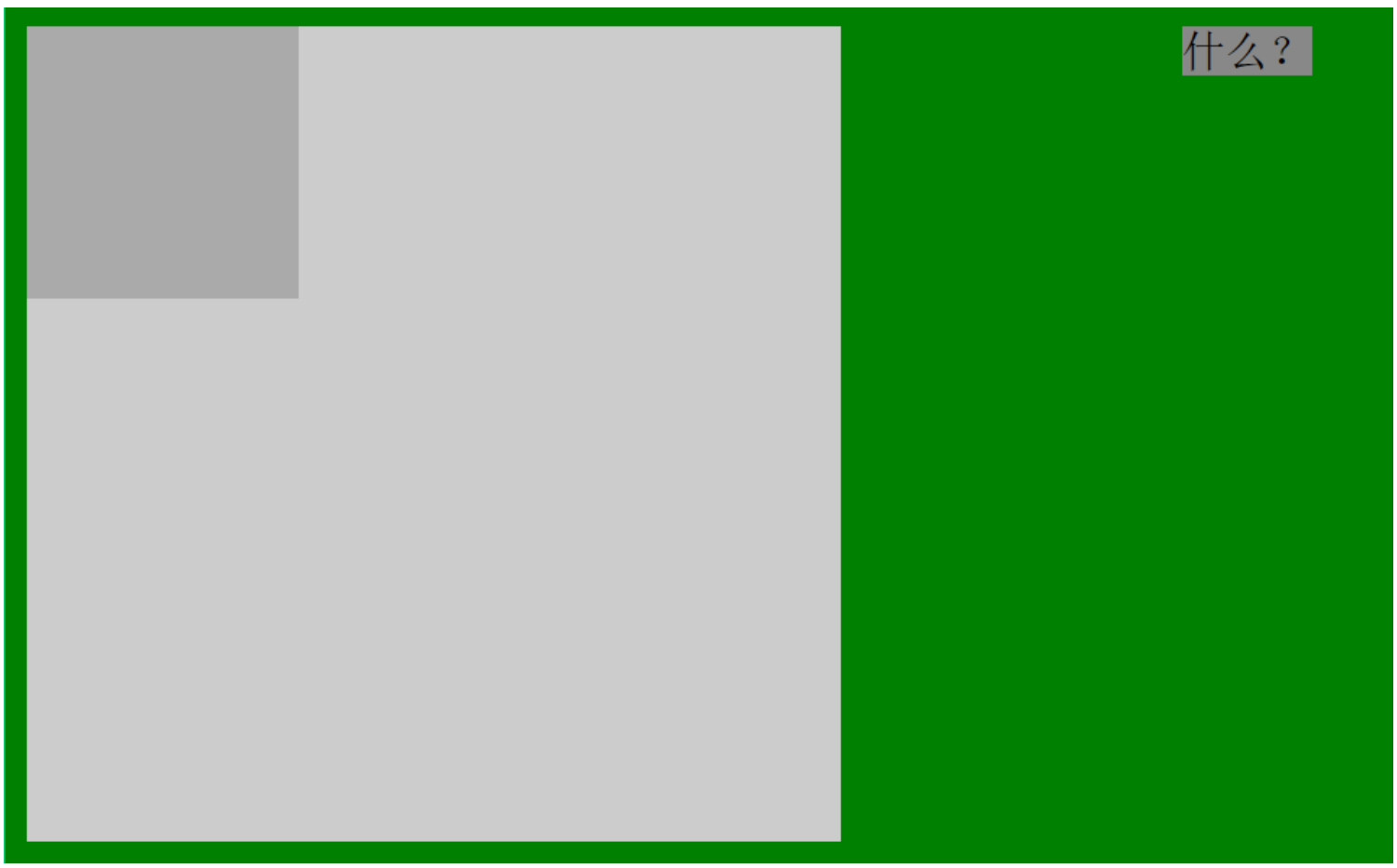
下面举几个例子：

例子1：

```
1
2  <!DOCTYPE html>
3  <html lang="en">
4  <head>
5      <meta charset="UTF-8">
6      <title>绝对定位</title>
7      <style>                    body{background:green;}
```

```
        .parent{ width: 500px;height: 500px;background: #ccc;}
        .son{ width: 300px;height: 300px;background: #aaa;}
        span{position: absolute; right: 30px; background: #888;}
    </style>
</head>
<body>
    <div class="parent">
        <div class="son">
            <span>什么？</span>
        </div>
    </div>
</body>
</html>
```

效果如下：

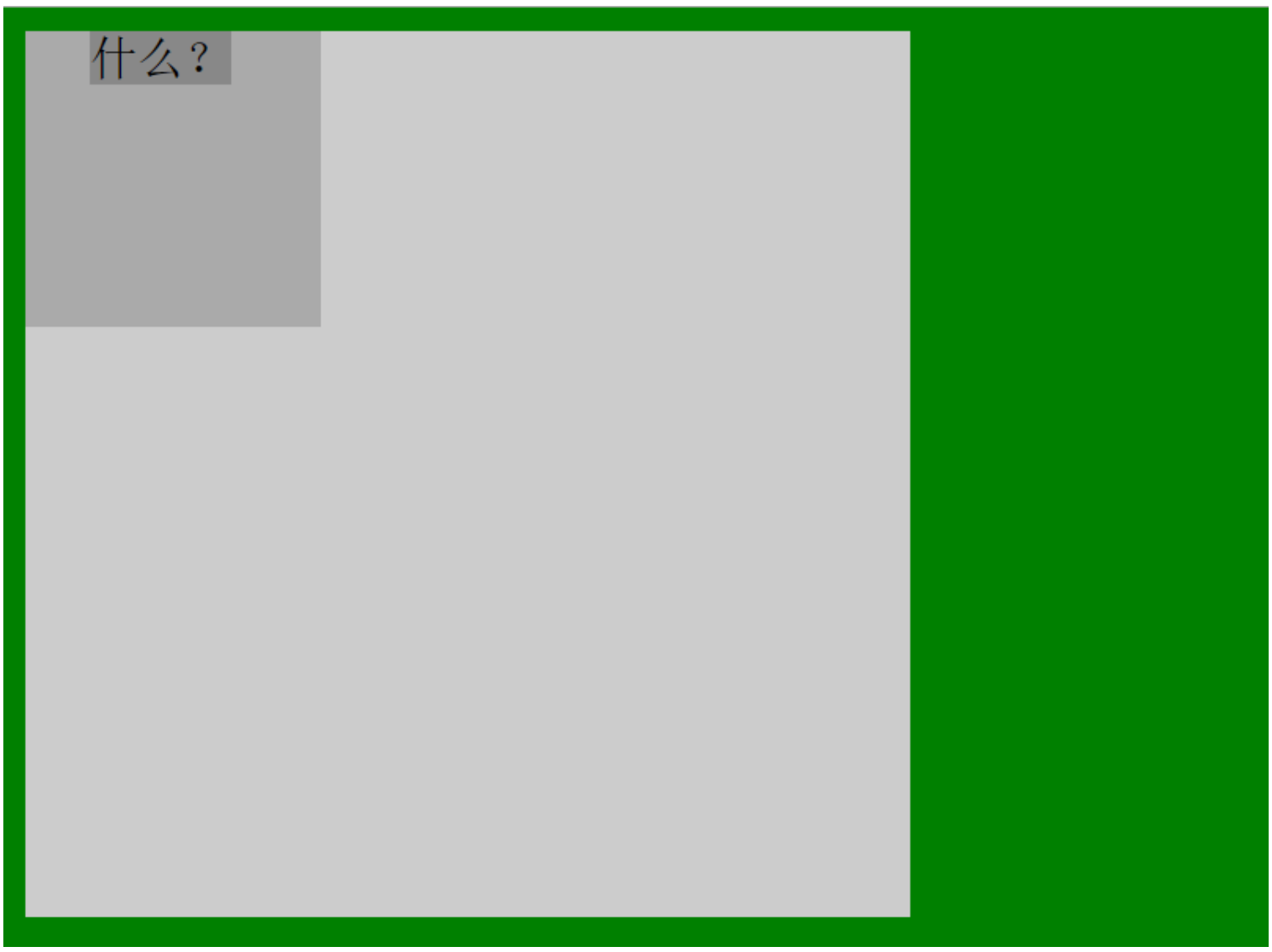


即我只在span中设置了position:absolute；而在其父元素中都没有，于是它的位置是相对于html的。

例2:

```
.son{position: relative; width: 100px;height: 100px;background: #aaa;
}
```

相较于上一个例子，我只修改了class为son的元素的css，设置为position：relative；效果图如下：

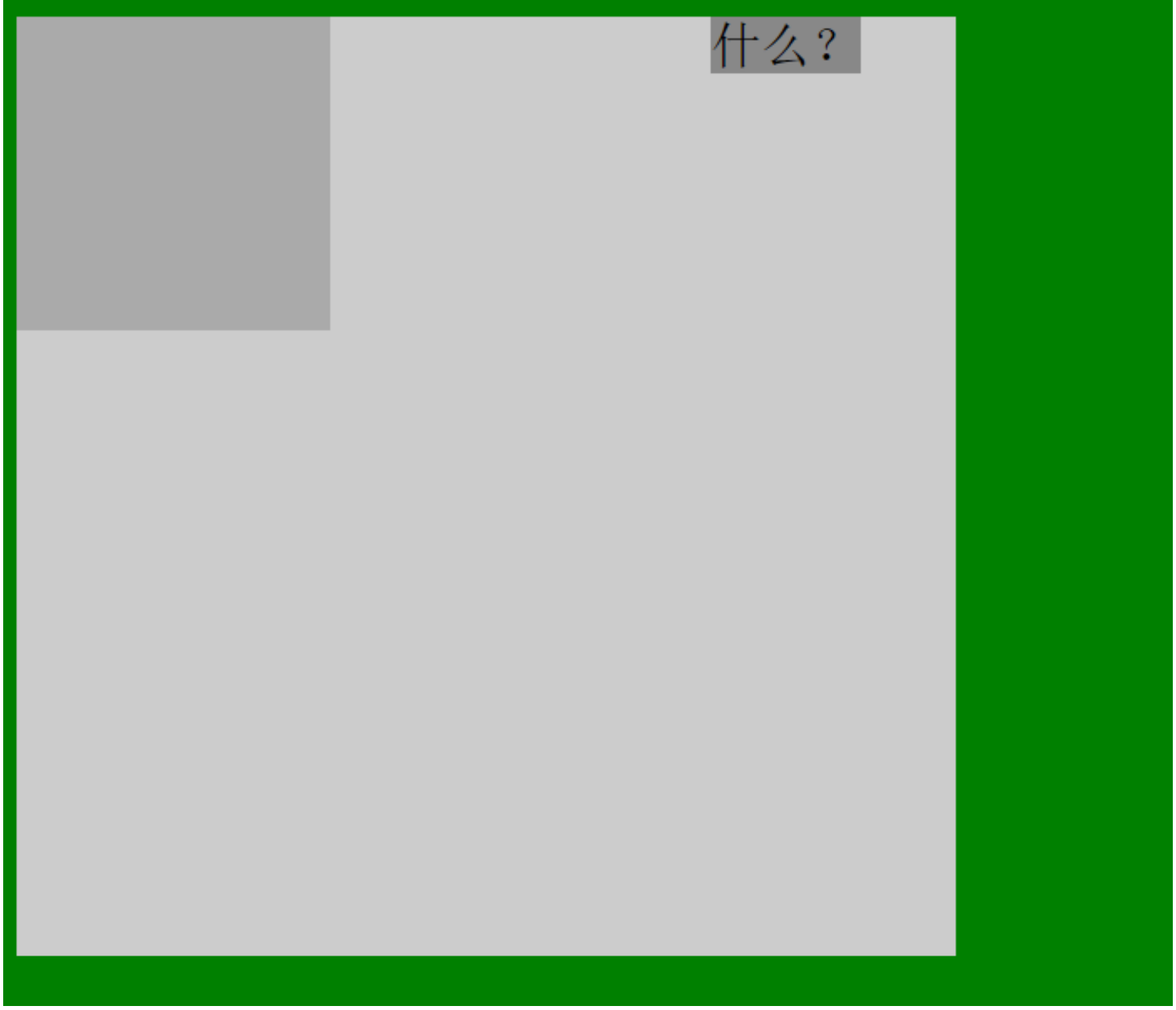


于是，我们发现现在span的位置是相对于设有position属性的class为son的父元素的。

例3:

```
.parent{position: absolute; width: 300px;height: 300px;background:
#ccc;}
```

这个例子我只是修改了第一个例子中的css--设置了position:absolute；效果如下：



于是我们发现，现在span的定位是相对于具有position:absolute的属性的class为parent的父元素。

例4：

```
1  .parent{position:fixed; width: 300px;height: 300px;background: #ccc;}
```

相对于例1，我添加了fixed的position属性，发现结果和例3是一模一样的。

例5：

```
1  .parent{position:static; width: 300px;height: 300px;background:
   #ccc;}
```

相对于例1，我添加了static的position属性（即html的默认属性），结果和例1是一样的。

综上所述，当某个absolute定位元素的父元素具有position:relative/absolute/fixed时，定位元素都会依据父元素而定位，而父元

素没有设置position属性或者设置了默认属性，那么定位属性会依据html元素来定位。

第五部分：重叠的元素--z-index属性

首先声明：z-index只能在position属性值为relative或absolute或fixed的元素上有效。

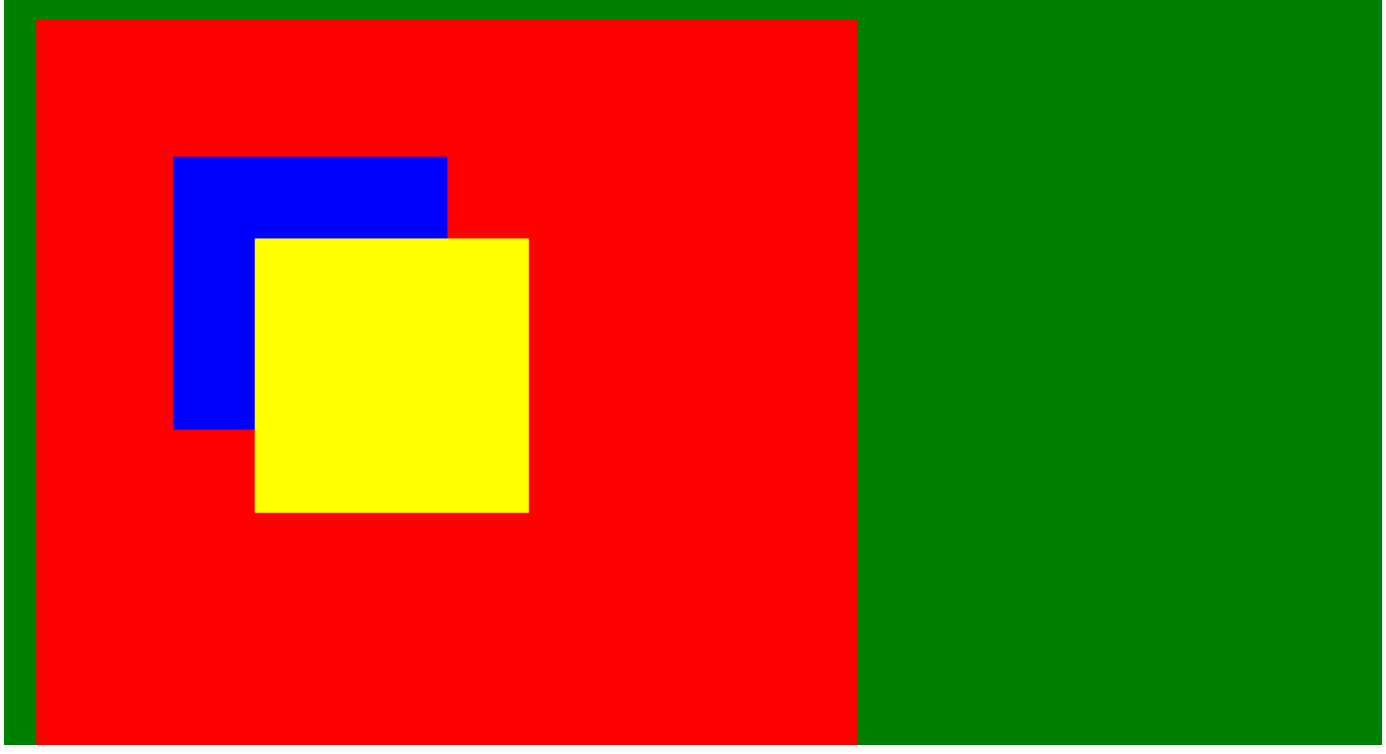
基本原理是：z-index的值可以控制定位元素在垂直于显示屏幕方向（z轴）上的堆叠顺序(stack order),值大的元素发生重叠时会在值小的元素上面。

下面我们通过几个例子继续来理解这个属性。

例1：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>绝对定位</title>
    <style>
        body{background: green;}
        .parent{position: relative; right: -5px; width: 300px;height: 300px;background: red;}
        .son1{position: absolute; left: 50px;top:50px; width: 100px;height: 100px;background: blue; }
        .son2{position: absolute; left: 80px;top: 80px; width: 100px;height: 100px;background: yellow; }
    </style>
</head>
<body>
    <div class="parent">
        <div class="son1"></div>
        <div class="son2"></div>
    </div>
</body>
</html>
```

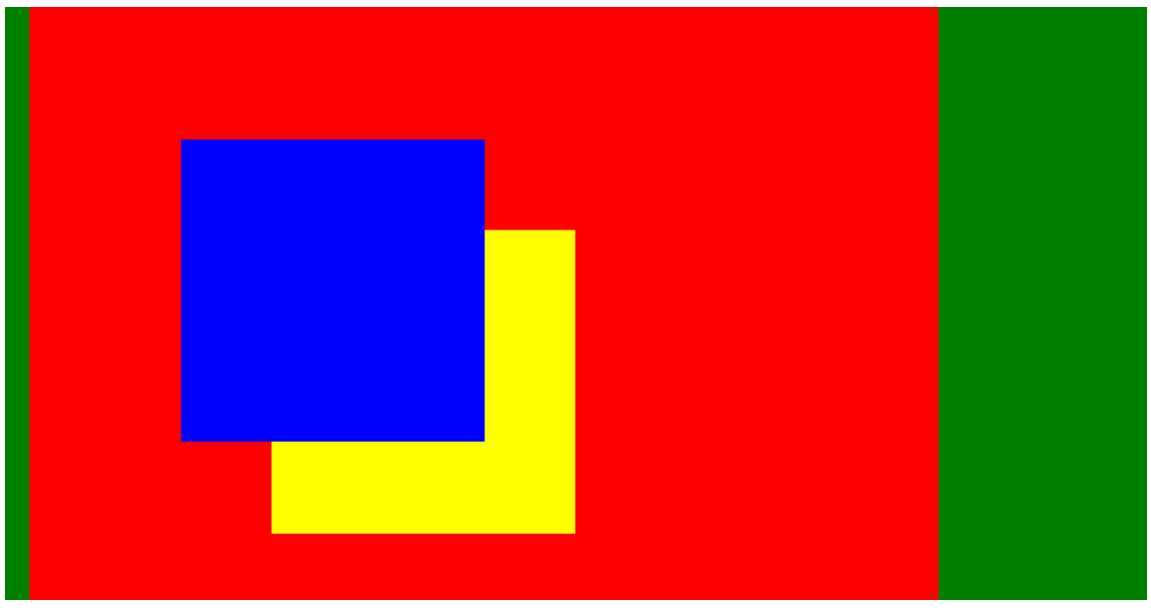
即son1和son2是parent的两个子元素，效果图如下：



这是没有使用z-index，我们发现son2在son1之上，这是因为son2在html中排在了son1之后，所以后来者将前者覆盖，如果我们颠倒以下两者的顺序，就会发现蓝色(son1)在上了。

例2：

在son1中加入z-index：1；可以发现效果如下：

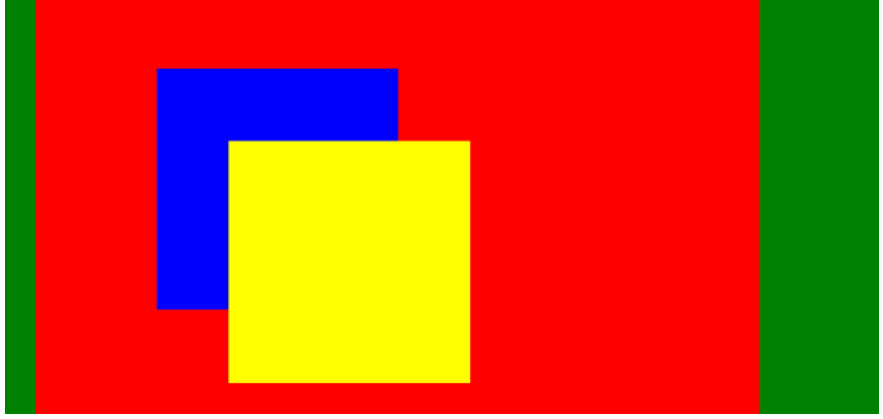


也就是说son2的index值是小于1的。

如果我们给son2也加上z-index:1;呢？结果得到黄色(son2)就在上面了。（因为一旦z-index值相等，情况就和都不设置index值一样了）

例3：

在son2中加入z-index:5;可以发现效果如下：



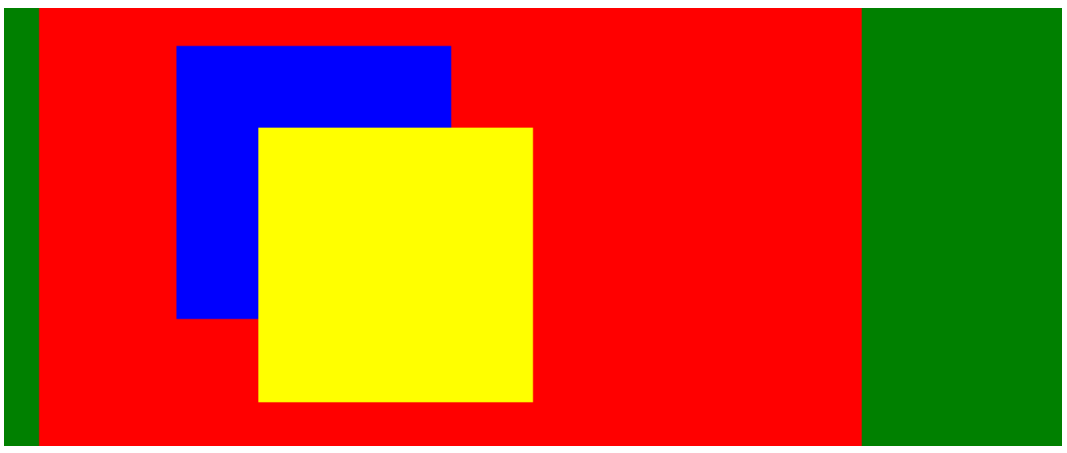
即黄色(son2)又在上面了，这个很简单，不作过多讨论。

例4：

在父元素添加z-index:10;

在son1和son2添加z-index:5; 这样理论上父元素就会在上面（黄色覆盖蓝色和黄色）；

结果如下：



结果没有变!!!!! 这就说明父元素和子元素不能做z-index的比较！！！但真的是这样吗？看下一个例子：

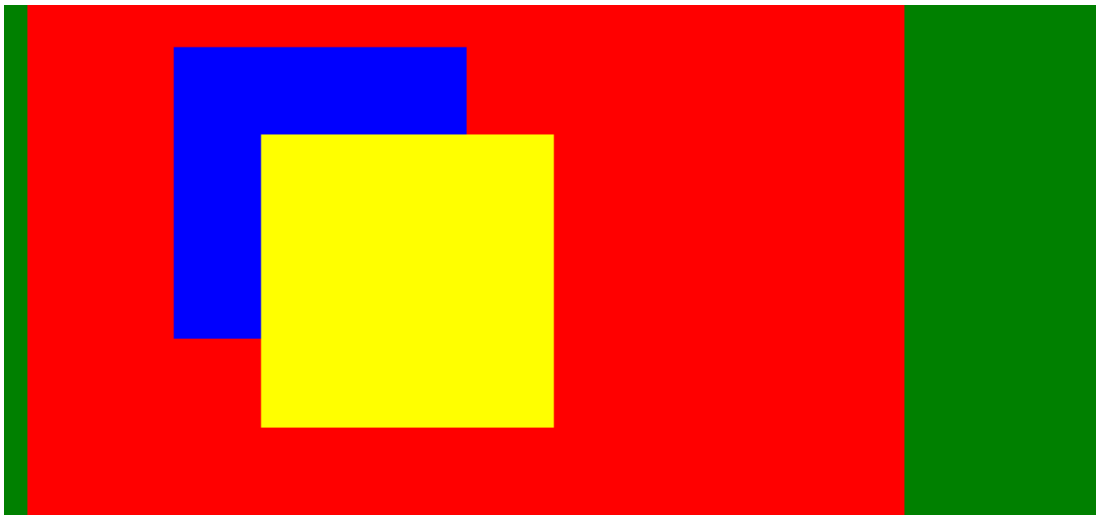
例5：

把两个子元素的z-index值同时设置为-5；父元素不设置z-index属性。结果如下：

成功！！说明在父元素和子元素之间还是可以作比较的！！！只是需要我们把子元素的z-index值设为负数。

例6：

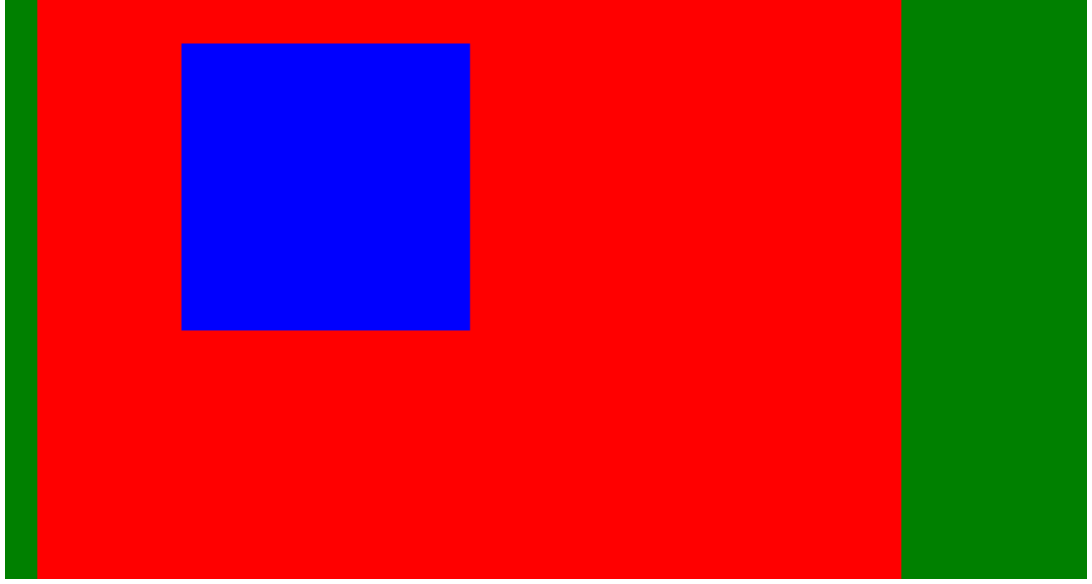
我们在例5的基础上再给父元素添加一个z-index:10，讲道理~应该也可以得到和例5相同的结果吧！！



然而.... 看来我们不能设置父元素的z-index值，否则就不能出现我们想要的效果。下面再看一个有趣的例子!

例7：

我们根据例6的经验不设置父元素的值，现在设置son1(蓝色)的z-index为5，son2的z-index为-5，看下面的结果：



即son1在最上面，父元素在中间，son2在最下面。

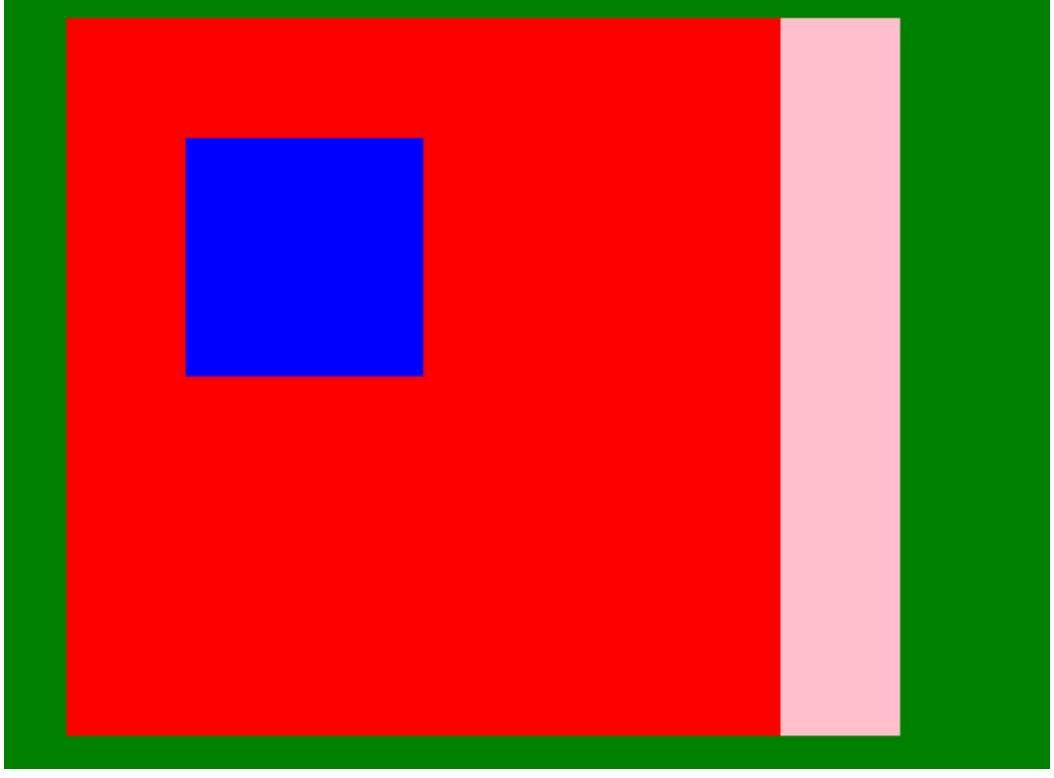
对于z-index的探索就这样结束了吗？？当然没有，看下面几个更为有趣的例子吧。

例8：

代码如下：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>绝对定位</title>
    <style>
        body{background: green;}
        .parent1{position: absolute; left:  50px; width: 300px;height: 300px;background: red;z-index: 15;}
        .parent2{ position: absolute;left:  100px;width: 300px;height: 300px;background: pink; z-index: 10;}
        .son1{position: absolute; left: 50px;top:50px; width: 100px;height: 100px;background: blue;  }
        .son2{position: absolute; left: 50px;top: 50px; width: 100px;height: 100px;background: yellow;  }
    </style>
</head>
<body>
    <div class="parent1">
        <div class="son1"></div>
    </div>
    <div class="parent2">
        <div class="son2"></div>
    </div>
</body>
</html>
```

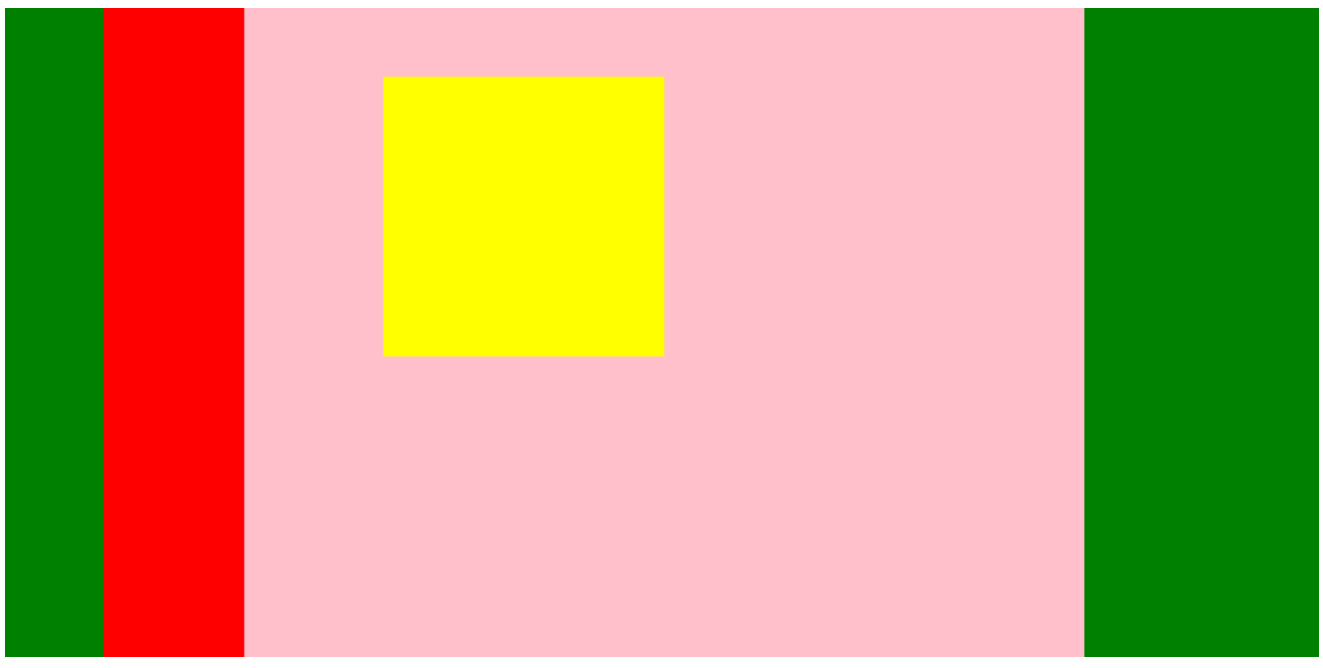
效果如下：



虽然parent1和parent2分别是son1和son2的父元素，按照我们之前的理解，父元素是不可添加z-index值的，否则会导致错误。但是这里parent1和parent2相对于body又是子元素，他俩是同级的，所以就可以进行比较了。且此时parent1的子元素son1（蓝色）在上。

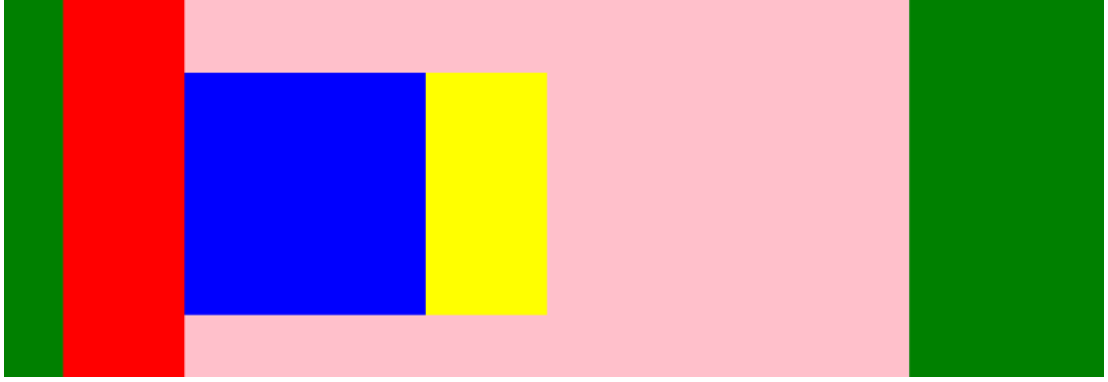
例9：

如果我们在例7的基础上，把parent2的z-index值设为20，就会发现如下效果：



即parent2在上的同时son2也会同时在上。这也就是所谓的“拼爹”了！！

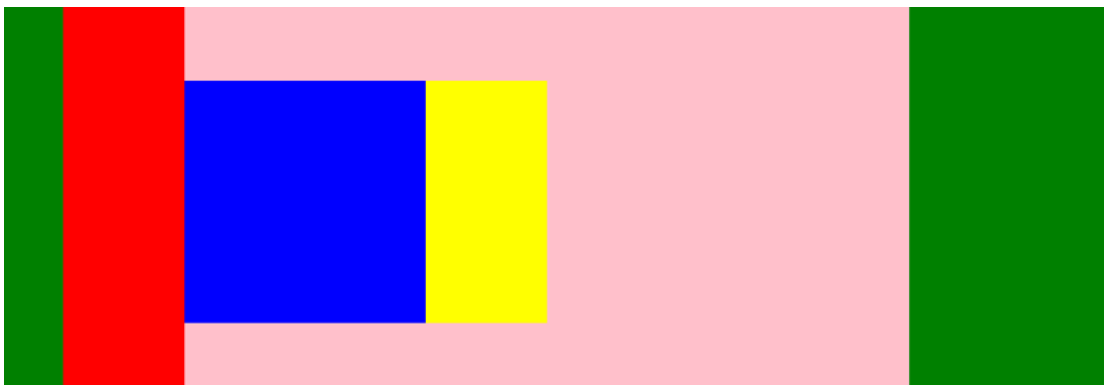
例10.同样在例7的基础上，我们不设置parent1和parent2和son2的index值，而只设置son1的z-index值为10，效果如下：



即原本在下面的蓝色son1被提上来了，而没有把父元素（parent1）提上了，诶，不孝顺啊！！

例11.显然，在例10的基础上，如果我们把son2的index值设置的比son1的大，如20，那么son2就会覆盖son1了，并且都在两个父元素只上了！！

效果如下图：



例12.当然，如果我们把两个son的z-index都设置位负数如-5，那么两者就都会被父元素所覆盖：

第六部分：脱离文档流导致的问题

我们知道如果使用position:absolute和position:fixed都会导致元素脱离文档流，由此就会产生相应的问题。举例如下：

```
<!DOCTYPE html>
<html>
<head>
    <title>position</title>
    <style>
        .div1{
            background-color: red;
            padding:20px;
        }
        .div2{
            width: 200px;
            height: 200px;
            background-color: blue;
        }
    </style>
</head>
<body>
    <div class="div1">
        <div class="div2"></div>
    </div>
</body>
</html>
```

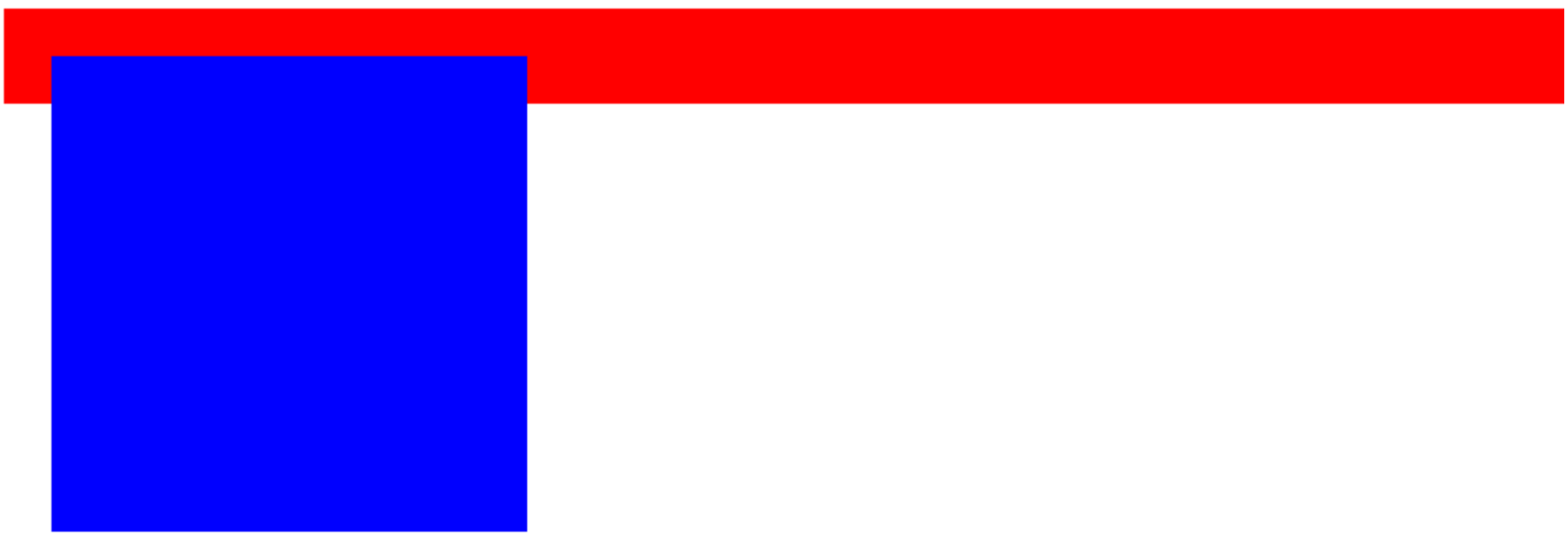
这时效果如下：

即子元素将父元素撑了起来。

但是一旦子元素的position为fixed或者是absolute，那么它就会脱离文档流，这样的后果是父元素无法被撑开，如下所示：

```
<!DOCTYPE html>
<html>
<head>
    <title>position</title>
    <style>
        .div1{
            background-color: red;
            padding:20px;
            position: relative;
        }
        .div2{
            position: absolute; // 添加position:absolute使其脱离文档流
            width: 200px;
            height: 200px;
            background-color: blue;
        }
    </style>
</head>
<body>
    <div class="div1">
        <div class="div2"></div>
    </div>
</body>
</html>
```

最终效果如下所示：



解决方法1：在js中设置父元素高度等于子元素的高度。

解决方法2：给父元素强行设置高度。（对于宽度导致的类似问题就强行设置宽度）

第七部分：position: sticky;

这一部分内容是2017-3-13补充的， 之前没有了解到这个属性，几天才知道这个属性就是我经常使用js使用的方法，就是在一个内容中，我们可以固定一个部分，然后到了另一个内容，又会固定另外一个部分。

同样也可以设置top值， 这个值是border上边缘和包裹元素的下边缘之间的距离，但是一旦滚动起来，就是和浏览器顶部的距离了，话不多说，直接上demo，一看便懂。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>alink</title>
    <style>
        body {
            margin: 0;
            padding: 0;
        }
        .wrap {
            border: 20px solid blue;
        }
        .header {
            position: sticky;
            top: 20px;
            border: 20px solid red;
            margin-top: 20px;
        }
    </style>
</head>
<body>
    <div class="wrap">
        <div class="header">
            这是头部
        </div>
        <div class="content">
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
```

这是内容部分
这是内容部分
这是内容部分

```
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
            这是内容部分<br>
        </div>
    </div>
    <div class="wrap">
        <div class="header">
            这是另一个头部
        </div>
        <div class="content">
            这是另一个内容<br>
            这是另一个内容<br>
            这是另一个内容<br>
            这是另一个内容<br>
            这是另一个内容<br>
            这是另一个内容<br>
            这是另一个内容<br>
            这是另一个内容<br>
            这是另一个内容<br>
            这是另一个内容<br>
            这是另一个内容<br>
```

这是另一个内容
这是另一个内容
这是另一个内容

```
                这是另一个内容<br>
                这是另一个内容<br>
                这是另一个内容<br>
                这是另一个内容<br>
                这是另一个内容<br>
                这是另一个内容<br>
                这是另一个内容<br>
            </div>
        </div>
</body>
</html>
```

第八部分：总结

这一部分知识还是非常有意思的，希望大家可以继续探索，当然如果通过这篇博文给予大家一点点的帮助那就再好不过了！