# 布局

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title></title>

<style type="text/css">

## /\*流体布局

左右模块各自向左右浮动，并设置中间模块的 margin 值使中间模块宽度自适应。

缺点就是主要内容无法最先加载，当页面内容较多时会影响用户体验。

\*/

/\*.left{

float: left;

width: 200px;

height: 200px;

background-color: red;

}

.right{

float: right;

width: 200px;

height: 200px;

background-color: green;

}

.main{

margin:0 300px;

height: 300px;

background: orange;

}\*/

## /\*BFC 三栏布局

BFC 规则有这样的描述：BFC 区域，不会与浮动元素重叠。因此我们可以利用这一点来实现 3 列布局。

缺点跟方法一类似，主要内容模块无法最先加载，当页面中内容较多时会影响用户体验。因此为了解决这个问题，有了下面要介绍的布局方案双飞翼布局。

\*/

/\*.left{

width: 200px;

height: 300px;

float: left;

margin-right: 120px;

background-color: red;

}

.right{

width: 200px;

height: 300px;

float: right;

margin-left: 200px;

background-color: green;

}

.main{

height: 300px;

background-color: orange;

overflow: hidden;

}\*/

## /\*双飞翼布局

利用的是浮动元素 margin 负值的应用，感兴趣的同学可以上网搜搜原理。

主体内容可以优先加载，HTML 代码结构稍微复杂点。

\*/

/\*.content{

width: 100%;

float: left;

}

.main{

height: 200px;

margin-left: 210px;

margin-right: 220px;

background-color: red;

}

.left{

float: left;

width: 200px;

height: 200px;

margin-left: -100%;

background-color: green;

}

.right{

float: right;

width: 200px;

height: 200px;

margin-left: -200px;

background-color: orange;

}\*/

## /\*圣杯布局

跟双飞翼布局很像，有一些细节上的区别，相对于双飞翼布局来说，HTML 结构相对简单，但是样式定义就稍微复杂，也是优先加载内容主体。

\*/

/\*.container{

margin-left: 120px;

margin-right: 200px;

}

.main{

float: left;

width: 100%;

height: 300px;

background-color: red;

}

.left{

float: left;

margin-left: -100%;

left: -120px;

position: relative;

height: 300px;

background-color: yellow;

width: 120px;

}

.right{

float: left;

width: 200px;

margin-left: -200px;

right: -200px;

height: 300px;

position: relative;

background-color: orange;

}\*/

## /\*Flex 布局

简单实用，未来的趋势，需要考虑浏览器的兼容性。

\*/

/\*.container {

display: flex;

}

.main {

flex-grow: 1;

height: 300px;

background-color: red;

}

.left {

order: -1;

flex: 0 1 200px;

margin-right: 20px;

height: 300px;

background-color: blue;

}

.right {

flex: 0 1 100px;

margin-left: 20px;

height: 300px;

background-color: green;

}\*/

## /\*Table 布局

缺点：无法设置栏间距\*/

/\*.container{

display: table;

width: 100%;

}

.left,.main,.right{

display: table-cell;

}

.main{

height: 300px;

background: red;

}

.left{

width: 200px;

height: 300px;

background: orange;

}

.right{

width: 300px;

height: 300px;

background: green;

}\*/

## /\*绝对定位布局

简单实用，并且主要内容可以优先加载。\*/

.container{

position: relative;

}

.main{

height: 300px;

margin: 0 200px;

background-color: red;

}

.left{

position: absolute;

left: 0;

top: 0;

width: 150px;

height: 300px;

background-color: orange;

}

.right{

position: absolute;

right: 0;

top: 0;

width: 180px;

height: 300px;

background-color: yellow;

}

</style>

</head>

## HTML代码

<body>

<!-- 流式布局、BFC布局 -->

<!-- <div class="left"></div>

<div class="right"></div>

<div class="main">

main

</div> -->

<!-- 双飞翼布局 -->

<!-- <div class="content">

<div class="main"></div>

</div>

<div class="left"></div>

<div class="right"></div> -->

<!-- 圣杯布局 Flex布局 绝对定位布局-->

<div class="container">

<div class="main">123</div>

<div class="left">123</div>

<div class="right">123</div>

</div>

<!-- Table 布局 -->

<!-- <div class="container">

<div class="left">123</div>

<div class="main">123</div>

<div class="right">123</div>

</div> -->

</body>

</html>

# 负margin值

第一种情况：元素没有设置浮动且没有设置定位或者 position 为 static

如果元素没有设置浮动并且没有设置定位或者 position 属性为 static 的情况下，对元素的 margin 设置负值会有以下的效果：

**设置的 margin 的方向为 top 或者 left**

当设置负值的 margin 的方向为 top 或者 left 的时候，元素会按照设置的方向移动相应的距离。

比如，设置 margin-left: -100px;。 那么，元素会往左移动 100px。对于设置 margin-top 也是一样的道理。

**设置的 margin 的方向为 bottom 或者 right**

当设置负值的 margin 的方向为 bottom 或者 right 的时候，元素本身并不会移动，元素后面的其他元素会往该元素的方向移动相应的距离，并且覆盖在该元素上面。

比如，设置 margin-right: -100px;。那么，元素本身并不会移动，后面的元素会向左移动 100px 到该元素上。对于设置 margin-bottom 也是同样的道理。

同时，在元素不指定宽度的情况下，如果设置 margin-left 或者 margin-right 为负值的话，会在元素对应的方向上增加其宽度。效果就和设置 padding-left 或者 padding-right 一样。

第二种情况：元素没有设置浮动且 position 为 relative

如果元素没有设置浮动，但是设置了相对定位，设置 margin 为负值的时候，表现如下：

**设置的 margin 的方向为 top 或者 left**

当设置负值的 margin 的方向为 top 或者 left 的时候，元素也会按照设置的方向移动相应的距离。

**设置的 margin 的方向为 bottom 或者 right**

当设置 margin-bottom/left 的时候，元素本身也不会移动，元素后面的其他元素也会往该元素的方向移动相应的距离，但是，该元素会覆盖在后面的元素上面 (当然，此处说的情况肯定是后面的元素没有设置定位以及 z-index 的情况)。

第三种情况：元素没有设置浮动且 position 为 absolute

如果元素没有设置浮动，但是设置了绝对定位，设置 margin 为负值的时候，表现如下：

**设置的 margin 的方向为 top 或者 left**

当设置负值的 margin 的方向为 top 或者 left 的时候，元素也会按照设置的方向移动相应的距离。

**设置的 margin 的方向为 bottom 或者 right**

由于设置绝对定位的元素已经脱离了标准文档流，所以，设置 margin-right/bottom 对后面的元素并没有影响。

第四种情况：元素设置了浮动

肯定没有既设置了浮动又设置绝对定位的情况，那样太荒唐了。  
设置了浮动的元素，再设置 postion: relative; 的话，元素的行为和单独设置 float 是一样的。

对于设置了浮动的元素，设置 margin 为负值的时候，表现如下：

如果设置的 margin 的方向与浮动的方向相同，那么，元素会往对应的方向移动对应的距离。

比如：

.elem {

float: right;

margin-right: -100px;

}

该元素则会向右移动 100px。

如果设置 margin 的方向与浮动的方向相反，则元素本身不动，元素之前或者之后的元素会向钙元素的方向移动相应的距离。

比如：

.elem {

float: right;

margin-left: -100px;

}

位于该元素左边的元素则会向右移动 100px，同时覆盖在该元素上。

如果后面的元素也设置了浮动的话，我们以一个具体的例子来说明。

<div class="container">

<div class="left"></div>

<div class="right"></div>

</div>

.container {

min-height: 300px;

margin: 30px auto;

overflow: hidden;

border: 1px solid #000000;

.left {

float: left;

width: 400px;

height: 200px;

margin-right: -300px;

background: purple;

}

.right {

float: left;

width: 300px;

height: 200px;

background: #cccccc;

}

}

.left 和 .right 都设置了浮动，在 .left 上设置了 margin-right: -300px;，那么，.right 会向左移动 300px，从而覆盖在 .left 上。这种行为与没有既没有设置浮动也没有设置定位的表现类似。

到此，我们把设置负 margin 的各种情况以及在各种情况下的表现都大概了解了一遍。那么，我们真正运用到实际中会是什么样子呢。

半遮挡的标题

原谅我措辞不好，大概就是下图的效果：



按照一般的思想，肯定是直接给 title 设置绝对定位，然后再将其调整过去。

但是，按照我们现在所说的，其实很简单就能实现这个效果。

这里只写了主要部分的代码。

<div class="title">Hey This is title!</div>

<div class="content">Hah! This is content.</div>

.title {

position: relative;

width: 200px;

height: 60px;

margin-bottom: -30px;

margin-left: -20px;

background: #000000;

}

.content {

max-width: 800px;

height: 400px;

padding: 0 50px;

background: yellow;

}

我们为 title 设置了两个 margin 的负值，分别是 margin-bottom: -30px;，以及 margin-left: -20px;。

设置 margin-bottom 是为了让 content 向上移动，设置 margin-left 是为了让 title 向左移动一小段距离。

还有个需要注意的地方就是，需要给 title 设置 position: relative;，根据我们的第二种情况所说的，这样，才能保证 title 覆盖在 content 之上。

简单的一列定宽的两列流式布局

根据我们的最后一种情况，通过设置 margin 为负值，我们可以很容易的实现一列定宽的两列流式布局。

<div class="container">

<div class="left"></div>

<div class="right"></div>

</div>

.left {

float: left;

width: 100%;

height: 200px;

margin-right: -300px;

background: purple;

}

.right {

float: left;

width: 300px;

height: 200px;

background: #cccccc;

}

唯一需要注意的地方就是设置了 100% 宽度的元素上的 margin 负值的绝对值一定要和定宽的元素的宽度相同。

两边固定，中间自适应的三列布局

这是一个很老的话题了，以前也有各种实现的方式，比如双飞翼布局，或者圣杯布局。

我们此处就以双飞翼布局来作示例。

先设置页面结构：

<div class="container">

<div class="center"></div>

<div class="left"></div>

<div class="right"></div>

</div>

此处我们没有把 center 放在中间，具体原因后面会解释。

然后，我们设置这三列都浮动：

.left,

.right,

.center {

float: left;

height: 500px;

}

同时为他们指定宽度：

.left {

width: 300px;

background: #000000;

}

.right {

width: 400px;

background: #00FFFF;

}

.center {

width: 100%;

background: #93c759;

}

现在我们要让 left 在左边，相当于就是让它覆盖在 center 的上面，所以，只需要这样一句：

margin-left: -100%;

同时，要让 right 在右边，同理，这样设置：

margin-left: -400px;

注意，此处的 margin 值的绝对值与 right 的宽度值相同。

其实，这样设置，我们的三列布局就基本完成了。

那么，我们为什么要把 center 放在 left 和 right 之前呢？

这个其实涉及到元素的堆叠顺序的知识 (这里就不详细讲解了，后面有时间的话专门拿一篇文章来讲解吧)，此处简单说明一下。

由于我们的三列都设置了浮动，所以，从某种意义上说，它们三个是在同一个平面的 (相当于 z-index 相同)，那么，这里就不能根据 CSS 来判断堆叠顺序了。所以，此处的 HTML 结构就决定了它们的堆叠顺序：所谓后来居上。

我们要让 left 在 center 之上，所以，肯定需要让 left 元素放在 center 之前。

所以，三列布局完整的 SCSS 代码如下：

.container {

overflow: hidden;

.left,

.right,

.center {

float: left;

height: 500px;

}

.left {

width: 300px;

margin-left: -100%;

background: #000000;

}

.right {

width: 400px;

margin-left: -400px;

background: #00FFFF;

}

.center {

width: 100%;

background: #93c759;

}

}

# BFC

**一、BFC是什么？**

　　在解释 BFC 是什么之前，需要先介绍 Box、Formatting Context的概念。

**Box: CSS布局的基本单位**

　　Box 是 CSS 布局的对象和基本单位， 直观点来说，就是一个页面是由很多个 Box 组成的。元素的类型和 display 属性，决定了这个 Box 的类型。 不同类型的 Box， 会参与不同的 Formatting Context（一个决定如何渲染文档的容器），因此Box内的元素会以不同的方式渲染。让我们看看有哪些盒子：

* block-level box:display 属性为 block, list-item, table 的元素，会生成 block-level box。并且参与 block fomatting context；
* inline-level box:display 属性为 inline, inline-block, inline-table 的元素，会生成 inline-level box。并且参与 inline formatting context；
* run-in box: css3 中才有， 这儿先不讲了。

**Formatting context**

　　Formatting context 是 W3C CSS2.1 规范中的一个概念。它是页面中的一块渲染区域，并且有一套渲染规则，它决定了其子元素将如何定位，以及和其他元素的关系和相互作用。最常见的 Formatting context 有 Block fomatting context (简称BFC)和 Inline formatting context (简称IFC)。

　　CSS2.1 中只有 BFC 和 IFC, [**CSS3**](http://www.cnblogs.com/lhb25/category/146075.html)中还增加了 GFC 和 FFC。

**BFC 定义**

　　BFC(Block formatting context)直译为"块级格式化上下文"。它是一个独立的渲染区域，只有Block-level box参与， 它规定了内部的Block-level Box如何布局，并且与这个区域外部毫不相干。

**BFC布局规则：**

1. 内部的Box会在垂直方向，一个接一个地放置。
2. Box垂直方向的距离由margin决定。属于同一个BFC的两个相邻Box的margin会发生重叠
3. 每个元素的margin box的左边， 与包含块border box的左边相接触(对于从左往右的格式化，否则相反)。即使存在浮动也是如此。
4. BFC的区域不会与float box重叠。
5. BFC就是页面上的一个隔离的独立容器，容器里面的子元素不会影响到外面的元素。反之也如此。
6. 计算BFC的高度时，浮动元素也参与计算

**二、哪些元素会生成BFC?**

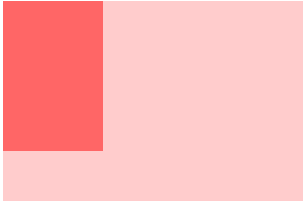
1. 根元素
2. float属性不为none
3. position为absolute或fixed
4. display为inline-block, table-cell, table-caption, flex, inline-flex
5. overflow不为visible

**三、BFC的作用及原理**

**1. 自适应两栏布局**

　　代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | <style>      body {          width: 300px;          position: relative;      }        .aside {          width: 100px;          height: 150px;          float: left;          background: #f66;      }        .main {          height: 200px;          background: #fcc;      }  </style>  <body>      <div class="aside"></div>      <div class="main"></div>  </body> |

　　页面：  
　　

　　根据BFC布局规则第3条：

每个元素的margin box的左边， 与包含块border box的左边相接触(对于从左往右的格式化，否则相反)。即使存在浮动也是如此。

　　因此，虽然存在浮动的元素aslide，但main的左边依然会与包含块的左边相接触。

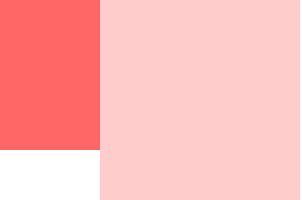
　　根据BFC布局规则第四条：

BFC的区域不会与float box重叠。

　　我们可以通过通过触发main生成BFC， 来实现自适应两栏布局。

|  |  |
| --- | --- |
| 1  2  3 | .main {      overflow: hidden;  } |

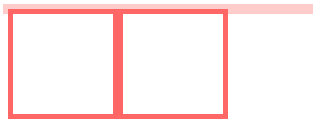
　　当触发main生成BFC后，这个新的BFC不会与浮动的aside重叠。因此会根据包含块的宽度，和aside的宽度，自动变窄。效果如下：

**

**2. 清除内部浮动**

　　代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <style>      .par {          border: 5px solid #fcc;          width: 300px;      }        .child {          border: 5px solid #f66;          width:100px;          height: 100px;          float: left;      }  </style>  <body>      <div class="par">          <div class="child"></div>          <div class="child"></div>      </div>  </body> |

　　页面：  
　　

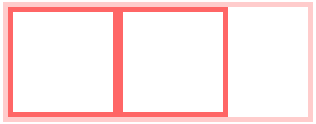
　　根据BFC布局规则第六条：

计算BFC的高度时，浮动元素也参与计算

　　为达到清除内部浮动，我们可以触发par生成BFC，那么par在计算高度时，par内部的浮动元素child也会参与计算。

　　代码：

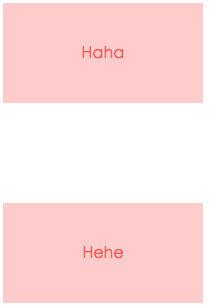
|  |  |
| --- | --- |
| 1  2  3 | .par {      overflow: hidden;  } |

　　效果如下：  
　　 ￼

**3. 防止垂直 margin 重叠**

　　代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <style>      p {          color: #f55;          background: #fcc;          width: 200px;          line-height: 100px;          text-align:center;          margin: 100px;      }  </style>  <body>      <p>Haha</p>      <p>Hehe</p>  </body> |

　　页面：  
　　

　　两个p之间的距离为100px，发送了margin重叠。  
　　根据BFC布局规则第二条：

Box垂直方向的距离由margin决定。属于同一个BFC的两个相邻Box的margin会发生重叠

　　我们可以在p外面包裹一层容器，并触发该容器生成一个BFC。那么两个P便不属于同一个BFC，就不会发生margin重叠了。  
　　代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <style>      .wrap {          overflow: hidden;      }      p {          color: #f55;          background: #fcc;          width: 200px;          line-height: 100px;          text-align:center;          margin: 100px;      }  </style>  <body>      <p>Haha</p>      <div class="wrap">          <p>Hehe</p>      </div>  </body> |

　　效果如下:  
　　

**总结**

　　其实以上的几个例子都体现了BFC布局规则第五条：

BFC就是页面上的一个隔离的独立容器，容器里面的子元素不会影响到外面的元素。反之也如此。

　　因为BFC内部的元素和外部的元素绝对不会互相影响，因此， 当BFC外部存在浮动时，它不应该影响BFC内部Box的布局，BFC会通过变窄，而不与浮动有重叠。同样的，当BFC内部有浮动时，为了不影响外部元素的布局，BFC计算高度时会包括浮动的高度。避免margin重叠也是这样的一个道理。

# 水平垂直居中

|  |
| --- |
| **关于flex**  关于flex：设为Flex布局以后，子元素的float、clear和vertical-align属性将失效 flex 是2012年的语法，也将是以后标准的语法，大部分浏览器已经实现了无前缀版本。dispaly:flex也是实际的版本  display: box;/\*最早09年草案\*/  display: -webkit-box; /\* Safari \*/  display: -webkit-flex; /\* Chrome, WebKit \*/  display: flex;/\*12年第五次草案 Chrome 29+, Firefox 22+, IE 11+, Opera 12.1/17/18, Android 4.4+ \*/ flex水平居中和垂直居中 flex + justify-content + align-items 只需设置父节点属性，无需设置子元素  <div class="parent">  <div class="child">Demo</div>  </div>  <style>  .parent {  display: flex;  justify-content: center; /\* 水平居中 \*/  align-items: center; /\*垂直居中\*/  }  </style>  只设置 justify-content: center未设置align-items: center子元素高度将被拉伸至与父元素高度一致  缺点：有兼容性问题 table水平垂直居中 inline-block + text-align + table-cell + vertical-align  <div class="parent">  <div class="child">Demo</div>  </div>  <style>  .parent {  text-align: center;//水平居中  display: table-cell;  vertical-align: middle;//垂直居中  }  .child {  display: inline-block;//防止块级元素宽度独占一行，内联元素可不设置  }  </style>  vertical-align的百分比值不是相对于字体大小或者其他什么属性计算的，而是相对于line-height计算的。 absolute+transform 水平垂直居中 <div class="parent">  <div class="child">Demo</div>  </div>  <style>  .parent {  position: relative;  }  .child {  position: absolute;  left: 50%;  top: 50%;  transform: translate(-50%, -50%);  }  </style> relative+absolute + negative margin .parent{  position:relative;  }  .child{  width: 100px;  height: 100px;  position: absolute;  top: 50%;  left: 50%;  margin: -50px 0 0 -50px;  }  缺点：需设置子元素宽高  **绝对定位方式+四个方向置0**  子元素可以是块级元素也可以是行内元素，没有影响  .parent{  position:relative  }  .child{  margin:auto;  height: 100px;  width: 100px;  position: absolute;  top: 0; left: 0; bottom: 0; right: 0;  }  需设置子元素宽高，否则子元素将与父元素宽高一致 line-height+text-align文本水平垂直居中 <div class="parent">  <div class="child">Demo</div>  </div>  .child{  text-align: center;  width: 100px;  height: 100px;  line-height: 100px;  /\*display: inline-block;\*/内联元素设置  } 子元素未知宽高的水平垂直居中 通过使用 box-align and box-pack 属性，居中 div 框的子元素：  .parent{  position: relative;  top: 0; left: 0; right: 0; bottom: 0;  display: -webkit-box;  -webkit-box-align: center;  -webkit-box-pack: center;  }  .child{  -webkit-box-flex: 0;  }  缺点：IE 不支持 box-pack 和 box-align 属性。  或者也可以使用上面提到的 flex水平居中和垂直居中  **水平居中**   * 对于行内元素(inline)：text-align: center; * 对于块级元素(block)：设置宽度且 marigin-left 和 margin-right 是设成 auto * 对于多个块级元素：对父元素设置 text-align: center;，对子元素设置 display: inline-block;；或者使用 flex 布局   **垂直居中**  对于行内元素(inline)   * 单行：设置上下 pandding 相等；或者设置 line-height 和 height 相等 * 多行：设置上下 pandding 相等；或者设置 display: table-cell; 和 vertical-align: middle;；或者使用 flex 布局；或者使用伪元素   对于块级元素(block)：下面前两种方案，父元素需使用相对布局   * 已知高度：子元素使用绝对布局 top: 50%;，再用负的 margin-top 把子元素往上拉一半的高度 * 未知高度：子元素使用绝对布局 position: absolute; top: 50%; transform: translateY(-50%); * 使用 Flexbox：选择方向，justify-content: center; |