

CSS排版：从毕升开始，我们就开始用正常流了

winter 2019-03-14



你好，我是 winter。今天我们来聊聊 CSS 的正常流。

我想，在 CSS 中，大家最讨厌的大概就是排版部分了。因为早年的 CSS 设计上不能够很好地支持软件排版需求，导致大家需要使用很多黑科技，让很多新人望而却步。

现在 CSS 提供了很多种排版方式，我们有很多选项可以选择自己适合的那一种，然而，正常流却是我们绕不开的一种排版。

我们能够在网上看到关于正常流的各种资料，比如：块级格式化上下文、margin 折叠等等……这一系列的概念光是听起来就令人非常头痛。

所以我相信很多同学一定会奇怪：正常流到底正常在哪里。事实上，我认为正常流本身是简单和符合直觉的东西。

我们之所以会觉得它奇怪，是因为如果我们从严苛的 CSS 标准角度去理解正常流，规定排版的算法，就需要引入上述那些复杂的概念。但是，如果我们单纯地从感性认知的层面去理

解正常流，它其实是简单的。

下面，就让我们先抛弃掉所有的已知概念，从感性认知的角度出发，一起去理解一下正常流。

正常流的行为

首先，我们先从词源来讲一讲排版这件事。

在毕昇发明活字印刷之前，排版这项工作是不存在的，相应的操作叫做“雕版”。人们要想印刷书籍，就需要依靠雕版工人去手工雕刻印版。

活字印刷的出现，将排版这个词引入进来，排版是活字印刷的 15 道工序之一，不论是古代的木质活字印刷，还是近代的铅质活字印刷，排版的过程是由排版工人一个字一个字从字架检出，再排入版框中。实际上，这个过程就是一个流式处理的过程。

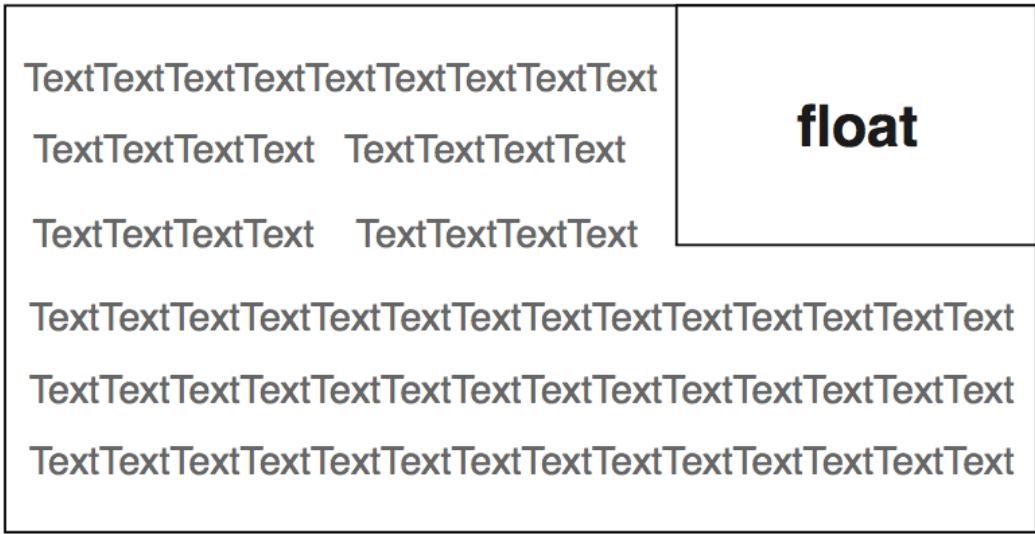
从古代活字印刷开始，到现代的出版行业，再到今天的 Web，排版过程其实并没有什么本质的变化，只不过，今天在我们的 CSS 中，排版需要处理的内容，不再是简单的大小相同的木字或者铅字，而是有着不同字体和字号的富文本，以及插入在富文本中大小不等的盒。

并且，在这些过程中，都会有一个正常流的存在。那么，正常流是什么样的呢？

我们可以用一句话来描述正常流的排版行为，那就是：依次排列，排不下了换行。这个操作很简单吧，我想，任何一个不懂排版的人都会将其作为排版时的第一反应。

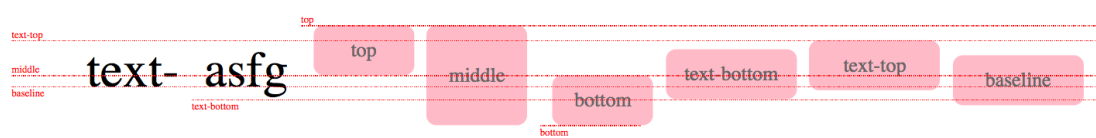
理解了正常流的基本概念，剩下的功能只需要在它的基础上延伸一下就好。

在正常流基础上，我们有 float 相关规则，使得一些盒占据了正常流需要的空间，我们可以把 float 理解为“文字环绕”。



我们还有 vertical-align 相关规则规定了如何在垂直方向对齐盒。vertical-align 相关规则看起来复杂，但是实际上，基线、文字顶 / 底、行顶 / 底都是我们正常书写文字时需要用到的概念，只是我们平时不一定会总结它们。

下图展示了在不同的 vertical-align 设置时，盒与文字是如何混合排版的。为了方便你理解，我们用代码给大家标注了基线、文字顶 / 底、行顶 / 底等概念。



([点击大图查看](#))

除此之外，margin 折叠是很多人非常不理解的一种设计，但是实际上我们可以把 margin 理解为“一个元素规定了自身周围至少需要的空间”，这样，我们就非常容易理解为什么 margin 需要折叠了。

正常流的原理

我们前面描述了正常流的行为，接下来我们要切换一下模式，用比较严谨的姿势来理解一下正常流。

在 CSS 标准中，规定了如何排布每一个文字或者盒的算法，这个算法依赖一个排版的“当前状态”，CSS 把这个当前状态称为“格式化上下文（formatting context）”。

我们可以认为排版过程是这样的：

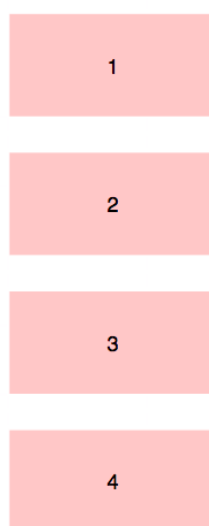
格式化上下文 + 盒 / 文字 = 位置

formatting context + boxes/charater = positions

我们需要排版的盒，是分为块级盒和行内级盒的，所以排版需要分别为它们规定了块级格式化上下文和行内级格式化上下文。

与正常流一样，如果我们单纯地看格式化上下文，规则其实是非常简单的。

块级格式化上下文顺次排列元素：



行内级格式化上下文顺次排列元素：



注意，块级和行内级元素的排版，受文字书写方向的影响，这里我们讲上下左右只是为了方便你直观理解。

当我们要把正常流中的一个盒或者文字排版，需要分成三种情况处理。

当遇到块级盒：排入块级格式化上下文。

当遇到行内级盒或者文字：首先尝试排入行内级格式化上下文，如果排不下，那么创建一个行盒，先将行盒排版（行盒是块级，所以到第一种情况），行盒会创建一个行内级格式化上下文。

遇到 float 盒：把盒的顶部跟当前行内级上下文上边缘对齐，然后根据 float 的方向把盒的对应边缘对到块级格式化上下文的边缘，之后重排当前行盒。

我们以上讲的都是一个块级格式化上下文中的排版规则，实际上，页面中的布局没有那么简单，一些元素会在其内部创建新的块级格式化上下文，这些元素有：

1. 浮动元素；
2. 绝对定位元素；
3. 非块级但仍能包含块级元素的容器（如 inline-blocks, table-cells, table-captions）；
4. 块级的能包含块级元素的容器，且属性 overflow 不为 visible。

这里的最后一条比较绕，实际上，我个人喜欢用另一种思路去理解它：

自身为块级，且 overflow 为 visible 的块级元素容器，它的块级格式化上下文和外部的块级格式化上下文发生了融合，也就是说，如果不考虑盒模型相关的属性，这样的元素从排版的角度就好像根本不存在。

好了，到这里我们已经讲完了正常流的排版详细规则，但是理解规则仅仅是基础，我们还需要掌握一些技巧。

正常流的使用技巧

现在，我们就一起来动手用实际的例子来研究一下。我们今天来看看等分布局 and 自适应宽，从这两种经典布局问题入手，一起来探索一下正常流的使用技巧。

等分布局问题

横向等分布局是一个很常见的需求，按照一般的思路，我们可以使用百分比宽度来解决，我们参考以下代码：

[复制代码](#)

```
1 <div class="outer">
2   <div class="inner"></div>
3   <div class="inner"></div>
4   <div class="inner"></div>
5 </div>
6 .inner {
7   width:33.33%;
8   height:300px;
9   display:inline-block;
10  outline:solid 1px blue;
11 }
```

在这段 HTML 代码中，我们放了三个 div，用 CSS 给它们指定了百分比宽度，并且指定为 inline-block。

但是这段代码执行之后，效果跟我们预期不同，我们可以发现，每个 div 并非紧挨，中间有空白，这是因为我们为了代码格式加入的换行和空格被 HTML 当作空格文本，跟 inline 盒混排了的缘故。

解决方案是修改 HTML 代码，去掉空格和换行：

[复制代码](#)


```
1 <div class="outer"><div class="inner"></div><div class="inner"></div><div clas:
```

但是这样做影响了源代码的可读性，一个变通的方案是，改变 outer 中的字号为 0。

[复制代码](#)


```
1 .inner {
2   width:33.33%;
3   height:300px;
4   display:inline-block;
5   outline:solid 1px blue;
6   font-size:30px;
7 }
8 .outer {
9   font-size:0;
10 }
```

在某些浏览器中，因为像素计算精度问题，还是会出现换行，我们给 outer 添加一个特定宽度：

 复制代码

```
1 .inner {
2     width:33.33%;
3     height:300px;
4     display:inline-block;
5     outline:solid 1px blue;
6 }
7 .outer {
8     width:101px
9 }
```

这个代码在某些旧版本浏览器中会出现换行。为了保险起见，我们给最后一个 div 加上一个负的右 margin：

 复制代码

```
1 .outer {
2     width:101px
3 }
4
5 .inner {
6     width:33.33%;
7     height:300px;
8     display:inline-block;
9     outline:solid 1px blue;
10 }
11
12 .inner:last-child {
13     margin-right:-5px;
14 }
```

这样就可以解决旧版本浏览器的问题了。


除了使用 inline-block，float 也可以实现类似的效果，但是 float 元素只能做顶对齐，不如 inline-block 灵活。

自适应宽

我们再来说说自适应宽。在 IE6 统治浏览器市场的旧时代，自适应宽（一个元素固定宽度，另一个元素填满父容器剩余宽度）是个经典的布局问题，我们现在就看一下如何使用正

常流来解决。


我们首先来看一下问题。

 复制代码

```
1 <div class="outer">
2   <div class="fixed"></div>
3   <div class="auto"></div>
4 </div>
5 .fixed {
6   width:200px;
7 }
8 .fixed, .auto {
9   height:300px;
10  outline:solid 1px blue;
11 }
```


这里 fixed 这个 div 宽度已经被指定好，我们需要添加 css 代码尝试让 auto 填满剩余宽度。

使用正常流解决这个问题的思路是，利用负 margin：

 复制代码

```
1 .fixed {
2   display:inline-block;
3   vertical-align:top;
4 }
5 .auto {
6   margin-left:-200px;
7   width:100%;
8   display:inline-block;
9   vertical-align:top;
10 }
```

但是，这样做会导致 auto 中的内容位置不对，所以我们还需要使用 padding 把内容挤出来，最终完整代码如下：

 复制代码

```
1 .fixed {
2   display:inline-block;
3   vertical-align:top;
4 }
```



```
5  .auto {  
6      margin-left:-200px;  
7      padding-left:200px;  
8      box-sizing:border-box;  
9      width:100%;  
10     display:inline-block;  
11     vertical-align:top;  
12 }
```

这样就给 auto 添加了 padding-left 和 box-sizing 两个属性。

总的来说，正常流布局主要是使用 inline-block 来作为内容的容器，利用块级格式化上下文的纵向排布和行内级格式化上下文的横向排布来完成布局的，我们需要根据需求的横向和纵向排布要求，来选择元素的 display 属性。

结语

这次的文章中，我们一起学习了正常流，我们可以用一句话来描述正常流的排版行为，那就是：依次排列，排不下了换行。这也是理解它最简单最源头的方式。

我们将正常流的知识分成了三个部分。

正常流的行为部分，我们从一些感性认知出发，帮助你从思路和源头上理解正常流的行为。

正常流的原理部分，我用更严格的描述方式，给你讲解了 CSS 标准中规定的正常流排版逻辑。

最后的正常流应用部分，我以两个经典布局问题等分布局 and 自适应宽为例，为你讲解了正常流实际使用的一些技巧。

最后，留给你一个思考题：用 JavaScript 写一个仅包含 inline-block 的正常流布局算法。你写好的话，可以留言给我，我们一起讨论。

猜你喜欢

Vue 开发实战

从 0 开始搭建大型 Vue 项目

戳此试读 

唐金州
一点资讯前端技术专家
Ant Design Vue 作者



精选留言(28)



Scorpio

我大flex天下第一!!! 😂

2019-03-14

 2

 71



AICC

试了一下，发现上面第二个例子的代码并不能实现想要的效果

首先，因为html代码的换行使得在inline-block的布局下两个盒子不能被放在一行这个通过父级font-size:0可解决

第二，由于auto在html的上的顺序是比fixed后面的，想像中的层级是高于fixed的，当auto是一个有背景盒子，fixed就被完全遮挡了,可以通过transform: translateZ(0)把它提起来

2019-03-14



 36



William

1. 等宽布局，不用外层font-size:0的方法的话，应该是.inner:not(last-child) { margin-right: -5px;

}吧，前面元素均添加一个负外边距抵消掉空格大小。

2. 因为也是用inline-block，所以自适应宽需要加上

```
.outer {  
  font-size: 0;  
}
```

2019-03-14



 12



阿成

Sir, have a look at this...
<https://github.com/aimergenge/inline-block-layout>

2019-03-16

1

8



七月有风

在 CSS 标准中，规定了如何排布每一个文字或者盒的算法，这个算法依赖一个排版的“当前状态”，CSS 把这个当前状态称为“格式化上下文（formatting context）”。
还是没有理解这句话

2019-03-17

1

5



我要飞

一个元素规定了自身周围至少需要的空间,这个解释深有体会,无可挑剔啊

2019-05-21

3



ycswaves

```
.auto {  
  width: calc(100% - 200px);  
  // ... rest of the necessary styles  
}
```

2019-03-24

3



沉默的话唠

为什么我写后面的完整版的，不会自动排布，宽度总是不够。被撑下去了。

2019-03-20

3



彘豪

grid写大的整体的布局框架，flex写一维的可线性化的布局，这两种布局的兼容性已经更好了，再加上一些模块和脚手架打包的时候能自动为你添加浏览器前缀，布局变得越来越容易了

2019-03-15

2



有铭

为什么三栏平分的那个样式里，给 outer 添加一个特定宽度和给最后一个 div 加上一个负的右 margin，我用chrome试验的结果，是变成了3个宽度很窄的盒子，而且第三个盒子在第二排？

2019-03-14



1



2



王天狗

为什么不用 calc 呢

2020-01-17



1



away

formatting context + boxes/charater = positions 单词charater拼写错误，应是character

2019-05-01



1



1



Sticker

感觉自适应宽还是浮动更爽一点！

2019-04-30



1



孙清海

大师你好！今天再看一本书《数据结构与算法描述JavaScript》 偶然发现了熟悉的名字，程劭非 大师作序！感觉好熟悉，哇这不是!!!我要好好看书了.....

2019-03-15



1



1



C阳

自适应宽例子中，是否应该在.fixed, .auto中加入float:left才能正确显示效果呢？

2019-03-14



1



起而行

js，如果检测到float和每个元素一行的block,就转换成inline-block,前者可以变成固定，后者去自动调整每个元素的间距

2020-03-20



布凡

老师，请教一下，等分的那个例子中，如果<div class="inner">内容</div>中包含内容整个div就会往下掉，这是什么原因导致的呢？另外如果设置.outer { font-size:0;}而.inner中没有设置样式font-size:30px; 这个宽度也不对，能再解释一下吗？

2020-02-27



梧桐

给 outer 添加一个特定宽度， 没有看到什么实际效果啊，下面这段代码还是会换行。

```
.outer {
  width:101px
}

.inner {
  width:33.33%;
  height:300px;
  display:inline-block;
  outline:solid 1px blue;
}

.inner:last-child {
  margin-right:-5px;
}
```

2020-01-16



Geek_fc9b29

三等分、自适应宽度，可以考虑强大的table-cell布局，自带bfc

2019-12-03



芝草晟林

感觉 formatting那一段有点难理解...

作者回复: 确实有点难

2019-07-09