

35-CSSFlex排版：为什么垂直居中这么难？

你好，我是winter。今天我们来谈谈Flex排版。

我们在前面多次讲过，正常流排版的设计来源于数百年来出版行业的排版经验，而HTML诞生之初，也确实是一种“超文本”存在的。

但是，自上世纪90年代以来，Web标准和各种Web应用蓬勃发展，网页的功能逐渐从“文本信息”向着“软件功能”过渡，这个思路的变化导致了：CSS的正常流逐渐不满足人民群众的需求了。

这是因为文字排版的思路是“改变文字和盒的相对位置，把它放进特定的版面中”，而软件界面的思路则是“改变盒的大小，使得它们的结构保持固定”。

因此，在早年的CSS中，“使盒按照外部尺寸变化”的能力非常弱。在我入行前端的时间（大约2006年），CSS三大经典问题：垂直居中问题，两列等高问题，自适应宽问题。这是在其它UI系统中最为基本的问题，而到了CSS中，却变成了困扰工程师的三座大山。

机智的前端开发者们，曾经创造了各种黑科技来解决问题，包括著名的table布局、负margin、float与clear等等。在这种情况下，Flex布局被随着CSS3一起提出（最初叫box布局），可以说是解决了大问题。

React Native则更为大胆地使用了纯粹的Flex排版，不再支持正常流，最终也很好地支持了大量的应用界面布局，这一点也证明了Flex排版的潜力。

今天，我们就从设计、原理和应用三个方面来学习一下Flex布局，我们先从设计开始。

Flex的设计

Flex在英文中是可伸缩的意思，一些翻译会把它译作弹性，我觉得有点不太准确，但是确实中文中没有更好的词。

Flex排版的核心是display:flex和flex属性，它们配合使用。具有display:flex的元素我们称为flex容器，它的子元素或者盒被称作flex项。

flex项如果有flex属性，会根据flex方向代替宽/高属性，形成“填补剩余尺寸”的特性，这是一种典型的“根据外部容器决定内部尺寸”的思路，也是我们最常用的Windows和Apple窗口系统的设计思路。

Flex的原理

说完了设计，我们再来看看原理，Flex的实现并不复杂，我曾经写过一个基本实现提交给spritejs项目，代码可以[参考这里](#)。

下面我们就来讲解一下，如何实现一个Flex布局。

首先，Flex布局支持横向和纵向，这样我们就需要做一个抽象，我们把Flex延伸的方向称为“主轴”，把跟它垂直的方向称为“交叉轴”。这样，flex项中的width和height就会称为交叉轴尺寸或者主轴尺寸。

而Flex又支持反向排布，这样，我们又需要抽象出交叉轴起点、交叉轴终点、主轴起点、主轴终点，它们可

能是top、left、bottom、right。

Flex布局中有一种特殊的情况，那就是flex容器没有被指定主轴尺寸，这个时候，实际上Flex属性完全没有用了，所有Flex尺寸都可以被当做0来处理，Flex容器的主轴尺寸等于其它所有flex项主轴尺寸之和。

接下来我们开始做Flex排版。

第一步是把flex项分行，有Flex属性的flex项可以暂且认为主轴尺寸为0，所以，它可以一定放进当前行。

接下来我们把flex项逐个放入行，不允许换行的话，我们就“无脑地”把flex项放进同一行。允许换行的话，我们就先设定主轴剩余空间为Flex容器主轴尺寸，每放入一个就把主轴剩余空间减掉它的主轴尺寸，直到某个flex项放不进去为止，换下一行，重复前面动作。

分行过程中，我们会顺便对每一行计算两个属性：交叉轴尺寸和主轴剩余空间，交叉轴尺寸是本行所有交叉轴尺寸的最大值，而主轴剩余空间前面已经说过。

第二步我们来计算每个flex项主轴尺寸和位置。

如果Flex容器是不允许换行的，并且最后主轴尺寸超出了Flex容器，就要做等比缩放。

如果Flex容器有多行，那么根据我们前面的分行算法，必然有主轴剩余空间，这时候，我们要找出本行所有的带Flex属性的flex项，把剩余空间按Flex比例分给他们即可。

做好之后，我们就可以根据主轴排布方向，确定每个flex项的主轴位置坐标了。

如果本行完全没有带flex属性的flex项，justify-content机制就要生效了，它的几个不同的值会影响剩余空白如何分配，作为实现者，我们只要在计算Flex项坐标的时候，加上一个数值即可。

例如，如果是flex-start就要加到第一个flex项身上，如果是center就给第一个flex项加一半的尺寸，如果是space-between，就要给除了第一个以外的每个flex项加上“flex项数减一分之一”。

第三步我们来计算flex项的交叉轴尺寸和位置。

交叉轴的计算首先是根据align-content计算每一行的位置，这部分跟justify-content非常类似。

再根据alignItems和flex项的alignSelf来确定每个元素在行内的位置。

计算完主轴和交叉轴，每个flex项的坐标、尺寸就都确定了，这样我们就完成了整个的flex布局。

Flex的应用

接下来我们来尝试用flex排版来解决一下当年的CSS三大经典问题（简直易如反掌）。

垂直居中：

```
<div id="parent">
```

```
<div id="child">
  </div>
</div>
```

```
#parent {
  display: flex;
  width: 300px;
  height: 300px;
  outline: solid 1px;
  justify-content: center;
  align-content: center;
  align-items: center;
}
#child {
  width: 100px;
  height: 100px;
  outline: solid 1px;
}
```

思路是创建一个只有一行的flexbox，然后用align-items:center;和align-content:center;来保证行位于容器中，元素位于行中。

两列等高：

```
<div class="parent">
  <div class="child" style="height:300px;">
  </div>
  <div class="child">
  </div>
</div>
<br/>
<div class="parent">
  <div class="child" >
  </div>
  <div class="child" style="height:300px;">
  </div>
</div>
```

```
.parent {
  display: flex;
  width: 300px;
  justify-content: center;
  align-content: center;
  align-items: stretch;
}
.child {
  width: 100px;
  outline: solid 1px;
}
```

思路是创建一个只有一行的flexbox，然后用stretch属性让每个元素高度都等于行高。

自适应宽：

```
<div class="parent">
  <div class="child1">
  </div>
  <div class="child2">
  </div>
</div>
```

```
.parent {
  display: flex;
  width: 300px;
  height: 200px;
  background-color: pink;
}
.child1 {
  width: 100px;
  background-color: lightblue;
}
.child2 {
  width: 100px;
  flex: 1;
  outline: solid 1px;
}
```

这个就是Flex设计的基本能力了，给要自适应的元素添加flex属性即可。

总结

今天我们从Flex的设计、原理和应用三个方面一起学习了Flex排版。

我们先从感性的角度，介绍了flex的设计，flex的设计是一种不同于流布局的，自外而内的设计思路。

接下来我们讲解了flex的实现原理，也就是具体的排版算法。要想理解flex排版的原理，主轴和交叉轴是非常重要的抽象，flex排版三个步骤：分行、计算主轴、计算交叉轴。

最后我们给出了几个例子，解决了旧时代的CSS三大经典问题。

最后，给你留一个小问题，请根据我的代码和文字，编写一段使用“position: absolute”来模拟flex布局的js。大家可以根据自己的水平，简化需求，比如可以实现一个仅仅支持横向的、单行的、所有flex项必须指定高度的flex布局。

重学前端

每天 10 分钟，重构你的前端知识体系

winter 程劭非

前手机淘宝前端负责人



新版升级：点击「👤请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

精选留言：

- 许童童 2019-04-13 11:39:10
flex好用，gird更好用。会讲一下吗？[2赞]
- 阿成 2019-04-15 11:31:08
<https://gist.github.com/aimergenge/e26193440fa38ebbb9a54847540c29c7> [1赞]
- favorlm 2019-04-14 20:37:30
自动填充剩余宽度
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<meta http-equiv="X-UA-Compatible" content="ie=edge" />
<title>自适应宽</title>
<style>
.box {
position: absolute;
width: 300px;
height: 100px;
background-color: cyan;
}
.childbox1Attr {
position: absolute;
background-color: bisque;
width: 35px;
height: 35px;
}
.childbox2Attr {
position: absolute;

```
background-color: beige;
```

```
width: 35px;
```

```
height: 35px;
```

```
}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<div class="box">
```

```
<div class="childbox1Attr"></div>
```

```
<div class="childbox2Attr"></div>
```

```
</div>
```

```
<script>
```

```
const childbox1Attr = {
```

```
width: '100'
```

```
};
```

```
const childbox2Attr = {
```

```
width: '100',
```

```
flex: '1'
```

```
};
```

```
const box = document.getElementsByClassName('box')[0];
```

```
const childbox1Ele = document.getElementsByClassName('childbox1Attr')[0];
```

```
const childbox2Ele = document.getElementsByClassName('childbox2Attr')[0];
```

```
const boxComputedStyle = window.getComputedStyle(box, null);
```

```
const child1ComputedStyle = window.getComputedStyle(childbox1Ele, null);
```

```
childbox1Ele.style.width = childbox1Attr.width+'px';
```

```
if(childbox2Attr.flex !== null || childbox2Attr.flex!=""){
```

```
if(childbox2Attr.flex === '1'){
```

```
// 利用一下不安全特性，实在不知道怎么写
```

```
const childbox2EleWidth = boxComputedStyle["width"].substring(0,3) - child1ComputedStyle["width"].s  
ubstring(0,3);
```

```
childbox2Ele.style.width = childbox2EleWidth + 'px';
```

```
childbox2Ele.style.left = child1ComputedStyle["width"]
```

```
}
```

```
}
```

```
</script>
```

```
</body>
```

```
</html>
```

```
[1赞]
```

- 北拉 2019-04-14 18:53:55

试过好多次，找了很多方法flex兼容ie9以下，每次都失败，有什么好的解决办法吗 [1赞]

- cjd 2019-04-15 14:23:48

直接calc(100% - n)

- 彘豪 2019-04-13 14:12:21

gird布局如果后面winter老师没有讲到，推荐你找找大漠老师的文章来看看，另外阮一峰老师也写了一篇，二者可以结合起来看看

