Flex 盒模型详解

之前我们说过经典盒模型和 IE 盒模型，*今天我们来探究一下 CSS3 中引入的弹性盒模型，也就是 Flex 盒模型*。弹性盒模型和我们之前的说的盒模型有什么关系，是新的一类么，我们之前说过对于盒模型可以通过 Box-sizing 进行设置，border-box，以及 content-box; 但是没有讲第三个值么，还是说最近刚刚更新的(扯淡，早就有了)，都不是，这一类准确名称被称为，弹性盒子。之前所说的是对于盒子大小限定的类型，而我们弹性盒子注重的功能性的地方， 所以他们不冲突，是相辅相成的，接下来我们要探究一下 **Flex 盒子的“弹性”**

Flex 盒子定义

Flex 是 Flexible Box 的缩写，翻译成中文就是“弹性盒子”，用来为盒状模型提供最大的灵活性。任何一个容器都可以指定为 Flex 布局。

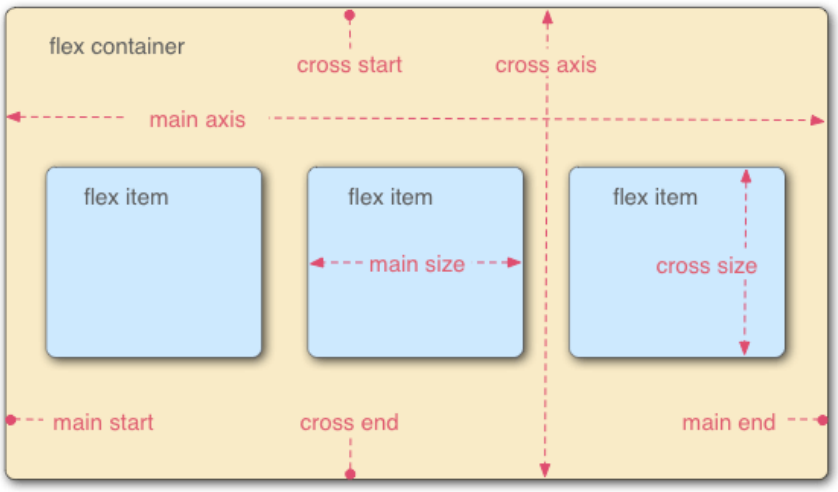
基本概念

在探究盒模型之前，我们需要了解一些基本概念。

采用 Flex 布局的元素，被称为 Flex 容器(flex container)，简称“容器”。其所有子元素自动成为容器成员，称为 Flex 项目(Flex item)，简称“项目”。

容器默认存在两根轴：水平方向主轴(main axis)和垂直方向交叉轴(cross axis)，默认项目按主轴排列。（所以 Flex 布局默认 direction 为 row）

- main start/main end：主轴开始位置/结束位置；
- cross start/cross end：交叉轴开始位置/结束位置；
- main size/cross size：单个项目占据主轴/交叉轴的空间；



介绍了基本的概念之后，接下来我们来看看弹性盒子有哪些属性以及里面的位置是如何计算的？

属性

display: flex | inline-flex;(元素将升级为弹性盒子). 前者容器升级为块级盒子，后者容器将升级为行内盒子. 元素采用 flex 布局以后，子元素的 float, clear, vertical-align 属性都将失效.

容器上的属性	flex-direction	Flex-wrap	Flex-flow	Justify-content	Align-items	Align-centent
项目上的属性	order	Flex-grow	flex	Flex-shrink	Flex-basis	Align-self

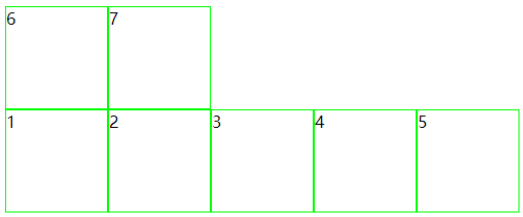
1. flex-direction: 指定主轴方向。

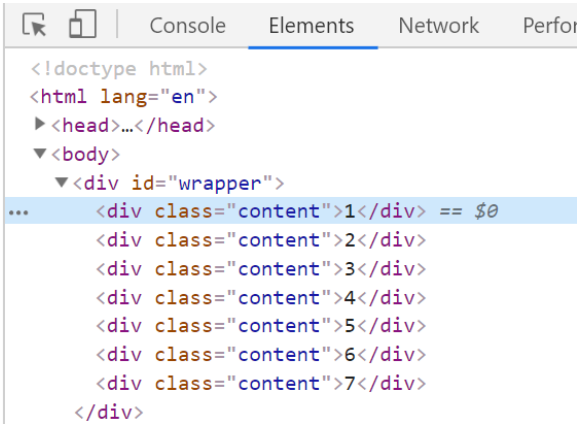
flex-direction 值	描述
row(默认)	指定主轴水平，子项目从左至右排列➔
row-reverse	指定主轴水平，子项目从右至左排列←
column	指定主轴垂直，子项目从上至下排列⬇
Column-reverse	指定主轴垂直，子项目从下至上排列↑

2. flex-wrap 指定如何换行.

flex-wrap 值	描述
nowrap(默认)	默认不换行
wrap	正常换行
Wrap-reverse	换行，且前面的行在底部

Wrap-reverse 效果如下：





3. flex-flow 它是 flex-direction 和 flex-wrap 的简写形式，默认值为 row nowrap。这个属性就不过多介绍了。

4. justify-content 指定主轴上子项目的对齐方式.(通常为水平方向对齐方式)

Justify-content 值	描述（主轴方向）
flex-start	子项目起始位置与 main start 位置对齐
flex-end	子项目末尾位置与 main end 位置对齐
center	在主轴方向居中于容器
space-between	与交叉轴两端对齐，子项目之间的间隔全部相等
space-around	子项目两侧的距离相等，它们之间的距离两倍于它们与主轴起始或末尾位置的距离.

这里要注意，**当我们使用 flex-start,flex-end, center,强调在主轴方向的开始，结尾，或者中间，所以 justify-content 受到 flex-direction 的影响。**

5. align-items 指定交叉轴上子项目的对齐方式.(通常为垂直方向对齐方式);

align-items 的值	描述(子项目—交叉轴方向)
flex-start	子项目起始位置与 cross start 位置对齐
flex-end	子项目末尾位置与 cross end 位置对齐
center	在交叉轴方向居中于容器
baseline	第一行文字的基线对齐
stretch	高度未定(或 auto)时，将占满容器的高度

同样在这里面也受 flex-direction 影响

6. align-content 指定多根主轴的对齐方式. 若只有一根主轴，则无效.

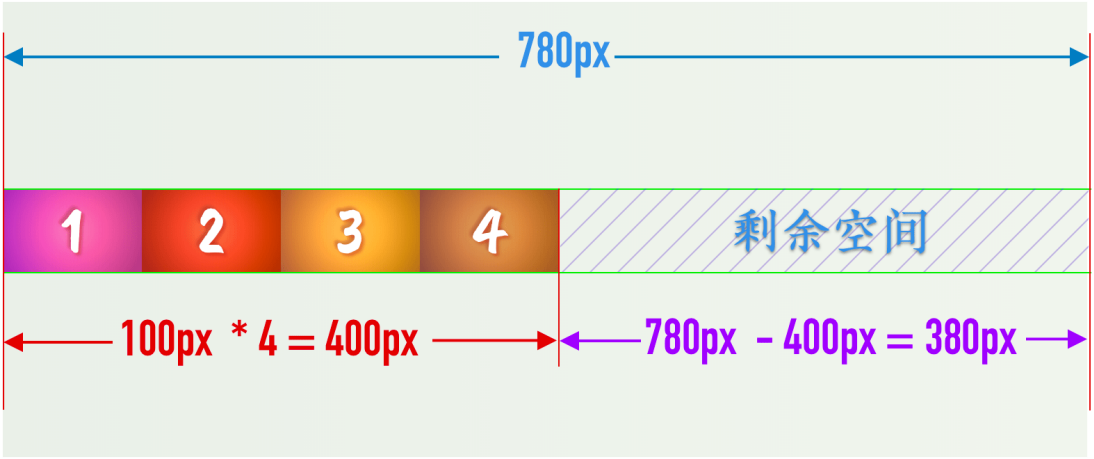
align-content 的值	
flex-start	顶部与 cross start 位置对齐
flex-end	底部与 cross end 位置对齐
center	在交叉轴方向居中于容器
space-between	与交叉轴两端对齐，间隔全部相等
space-around	子项目两侧的距离相等，它们之间的距离两倍于它们与主轴起始或末尾位置的距离.
stretch(默认)	多根主轴上的子项目充满交叉轴

接下来说说子元素的属性，这里面既有比较复杂的点了，耐心的学下去。

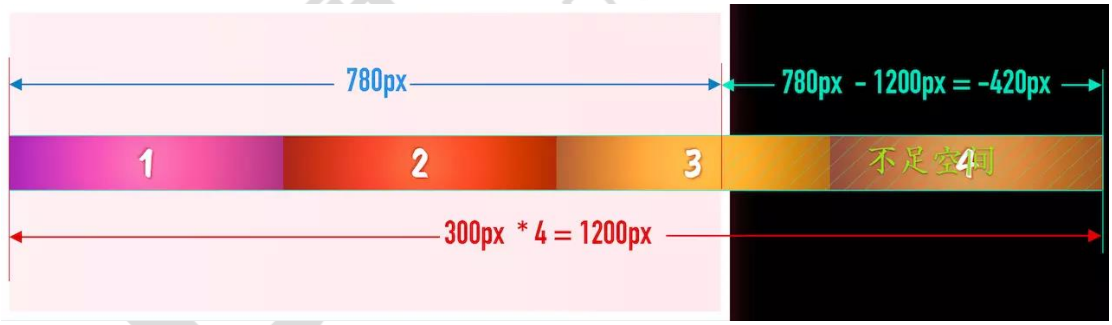
1. flex
   flex 属性是 flex-grow, flex-shrink 和 flex-basis 的简写，默认值为 0 1 auto。后两个属性可选。

相关概念：

剩余空间：当所有 Flex 项目尺寸大小之和小于 Flex 容器时，Flex 容器就会有多余的空间没有被填充，那么这个空间就被称为 Flex 容器的剩余空间（Positive Free Space）



不足空间：当所有 Flex 项目尺寸大小之和大于 Flex 容器时，Flex 容器就没有足够的空间容纳所有 Flex 项目，那么多出来的这个空间就被称为负空间（Negative Free Space）



lex-basis 属性相关的运用

flex-basis： 默认值为 auto
如果 Flex 项目 显式的 设置了 width 值，同时 flex-basis 为 auto 时，则 Flex 项目的宽度为按 width 来计算，如果未显式设置 width，则按 Flex 项目的内容宽度来计算
如果 Flex 项目显式的设置了 width 值，同时显式设置了 flex-basis 的具体值，则 Flex 项目会忽略 width 值，会按 flex-basis 来计算 Flex 项目，当 Flex 容器剩余空间不足时，Flex 项目的实际宽度并不会按 flex-basis 来计算，会根据 flex-grow 和 flex-shrink 设置的值给 Flex 项目分配相应的空间
如果 Flex 项目显式的设置了 min-width 或 max-width 值时，当 flex-basis 计算出来的值小于 min-width 则按 min-width 值设置 Flex 项目宽度，反之，计算出来的值大于 max-width 值时，则按 max-width 的值设置 Flex 项目宽度

flex-grow
当 Flex 容器有一定的剩余空间时，flex-grow 可以让 Flex 项目分配 Flex 容器剩余的空间，每个 Flex 项目将根据 flex-grow 因子扩展，从而让 Flex 项目布满整个 Flex 容器（有效利用 Flex 容器的剩余空间）。当所有的 Flex 项目具有一个相同的 flex-

grow 值时，那么 Flex 项目将会平均分配 Flex 容器剩余的空间。

如何 grow 的？
假设：一个 宽度 1000px 的 Flex 容器 下有 2 个 Flex 项目，如图：

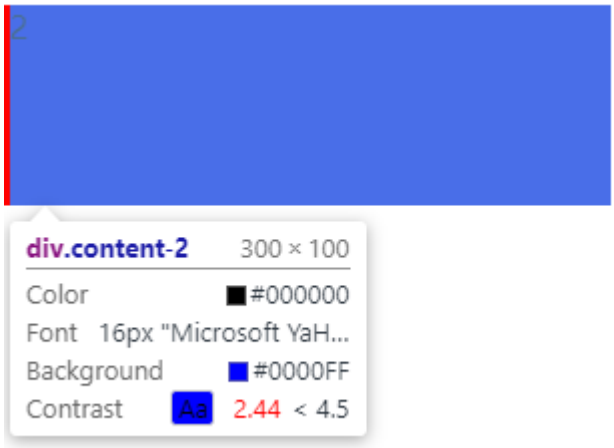
CSS：

```
#wrapper {
            display: flex;
            width: 1000px;
            height: 200px;
        }

        .content-1 {
            height: 100px;
            background-color: red;
            flex-grow: 5;
            flex-shrink: 5;
            flex-basis: 200px;
        }

        .content-2 {
            height: 100px;
            background-color: blue;
            flex-grow: 1;
            flex-shrink: 1;
            flex-basis: 200px;
        }
```

两块长度分别为：



一个长 700 ,另一个长 300， 是怎么计算的呢？
计算过程如下：
容器宽度 1000px - flex-basis[1] - flex-basis[2] > 0 即 有剩余空间
剩余空间 = 容器宽度 837px - flex-basis[1] - flex-basis[2] = 1000px - 200px -200px = 600px
flex-grow 发挥作用如下：
flex-grow [1] * x + flex-grow[2] * x = 600px
5x + 1x = 600
x= 100px
所以
Flex 项目 1 的总宽度 = flex-basis[1] + flex-grow[1] * x = 200px + 100px * 5 = 700px
Flex 项目 2 的总宽度 = flex-basis[2] + flex-grow[2] * x = 200px + 100px * 1 = 300px

flex-shrink

flex-shrink 是用来控制 Flex 项目缩放因子。当所有 Flex 项目宽度之和大于 Flex 容器时，将会溢出容器（flex-wrap 为 nowrap 时），flex-shrink 就可以根据 Flex 项目设置的数值比例来分配 Flex 容器的不足空间，也就是按比例因子缩小自身的宽度，以免溢出 Flex 容器。

Shrink 计算：
CSS:

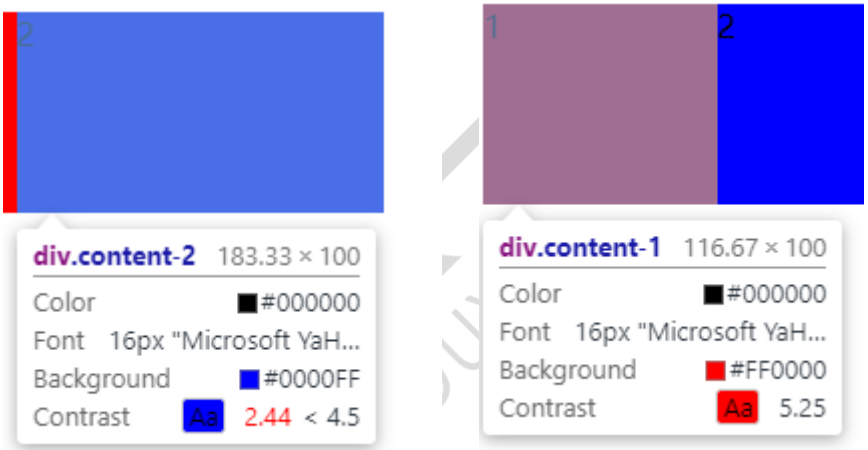
```
#wrapper {
            display: flex;
            width: 300px;
            height: 200px;
        }

        .content-1 {
            height: 100px;
            background-color: red;
```

```
        flex-grow: 5;
        flex-shrink: 5;
        flex-basis: 200px;
    }

    .content-2 {
        height: 100px;
        background-color: blue;
        flex-grow: 1;
        flex-shrink: 1;
        flex-basis: 200px;
    }
```

样式：



Shrink 计算过程：

容器 宽度 300px - flex-basis[1](200) - flex-basis[2](200) < 0 即 有不足空间

不足空间 = 容器 宽度 837px - flex-basis[1] - flex-basis[2] = 400px - 200px -200px = -100px

flex-shrink 发挥作用如下：

flex-shrink [1] * x + flex-shrink[2] * x = 763px

5x + 1x = 100px

x = 16.66px

所以

Flex 项目 2 的总宽度 = flex-basis[1] - flex-shrink[1] * x = 200px – 16.66px * 5 = 116.67px

Flex 项目 1 的总宽度 = flex-basis[2] + flex-shrink[2] * x = 200px - 16.66px * 1 = 183.33px

下面借助一些图让大家更容易理解。

$basis_i$: `flex-basis` 定義的預借空間
$freeSpace$: flex container 寬度扣除 `flex-basis` 之後的剩餘空間

$width_{grow_i}$: 是 grow 算出來的元素寬度
$growSize_i$: 是元素伸展量
$grow_i$: 是元素設定的 `flex-grow` 的值

$width_{shrink_i}$: 是 shrink 算出來的元素寬度
$shrinkSize_i$: 是元素伸展量
$shrink_i$: 是元素設定的 `flex-shrink` 的值

基本参数

$$width_{grow_i} = basis_i + growSize_i$$

$$growSize_i = freeSpace \times \frac{grow_i}{\sum_{i=0}^{n}(grow_i)}$$

flex-grow

$$width_{shink_i} = basis_i + shinkSize_i$$

$$shinkSize_i = freeSpace \times \frac{basis_i \times shink_i}{\sum_{i=0}^{n}(basis_i \times shink_i)}$$

flex-shrink & flex-basis

```
flex: 1 1 0;          flex: flex-grow flex-shrink flex-basis
```