你需要知道的所有 Flexbox 排列方式

<https://blog.csdn.net/weixin_34384915/article/details/87943067>

**属性**

许多人告诉我他们在使用 flexbox 的时候很难区别是应该使用以 align- 还是 justify- 开头的属性。所以你需要知道：

justify- 实现主轴上的排列方式。即排列与你的 flex-direction 相同的方向。

align- 实现交叉轴上的排列方式。即排列与你的 flex-direction 相垂直的方向。

在下文中，根据主轴和交叉轴而不是水平和垂直的方向来思考会更容易理解。（主轴和交叉轴）和物理方位一点关系都没有。

**用 justify-content 来排列主轴**

我们将会从主轴排列来开始讨论。在主轴上，我们通过 justify-content 属性来实现排列。这个属性的作用对象是我们的所有 flexbox 子元素所组成的组。同时也控制着组内所有元素的间距。

默认的 justify-content 值是 flex-start。这也就是为什么你声明 display: flex 之后你的所有 flexbox 子元素朝着你的 flex 盒子的开始排成一行。如果你有一个值为 row 的 flex-direction 属性同时页面是从左到右读的语言（例如英语）的话，这些字元素将会从左边开始排列。

**用 align-content 来排列交叉轴**

如果你给你的 flex 容器添加了 flex-wrap: wrap 同时也有好几条 flex 排列行，那你可以用 align-content 属性来在交叉轴上排列你的 flex 排列行。不过，这将会需要交叉轴上有额外的空间。在下面这个 demo 中，我的交叉轴作为竖直的列在运行，同时我设置了这个 flex 容器的高度为 60vh。由于这个高度比我展示 flex 子元素所需的高度大，所以我的容器有了交叉轴方向上的空余空间。

我可以使用下面所有的 align-content 属性值：

**简写方法 place-content**

在盒子元素排列规范中，我们发现了一个简写方法 place-content。使用这个属性意味着你可以一次性设置 justify-content 和 align-content。它的第一个值是 align-content，第二个值是 justify-content 。如果你仅仅设置了一个值 A，那么这两个值都将设置成 A，因此：

.container {

place-content: space-between stretch;

}

和下面一样：

.container {

align-content: space-between;

justify-content: stretch;

}

如果我们使用：

.container {

place-content: space-between;

}

那将和下面一样：

.container {

align-content: space-between;

justify-content: space-between;

}

**用 align-items 来排列交叉轴**

我们现在知道，我们可以对（所有 flex 子元素所组成的）子元素组进行关于 flex 元素和 flex 排列行的操作。不过，我们希望有其它方式即通过声明元素与元素之间在数轴上的关系来操作我们的元素。你的 flex 容器有一个高度，这个高度可能是由容器内最高的子元素所决定的，就如下图一样。

容器的高度被第三个子元素所定义（放大预览）

flex 容器的高度可以通过给 flex 容器添加一个 height 属性所代替：

容器的高度通过该容器的大小属性所定义（放大预览）

flex 子元素看起来都被拉伸到最高的子元素的高度的原因是 align-items 的初始值是 stretch。子元素们在交叉轴上被拉伸成 flex 容器在那个方向上的尺寸了。

请记住哪里出现 align-items 会导致困惑：如果你有一个具有多个 flex 排列行的 flex 容器，那么每一个 flex 排列行都会像一个新的 flex 容器一样，（该行的）最高的 flex 子元素将会决定哪一行的所有 flex 子元素高度。

除了设置拉伸的初始值之外，你也可以给 align-items 属性设置一个值 flex-start，在这种情况下 flex 子元素将会在容器的开始之处排列同时也不会拉伸。

设置值 flex-end 将会把它们（flex 子元素）移到交叉轴的结束之处。

如果你使用值 center ，那 flex 子元素将会排列在交叉轴中央：

我们也可以设置依据文字基准线排列。这将会确保（flex 子元素）以文字的基准线排列，而不是盒子的边框。

**使用 align-self 来设置单个元素的排列**

align-items 意味着你可以一次设置所有的 flex 子元素。这个操作的真正原理是对所有的 flex 子元素一一设置其 align-self 值。当然你也可以任意单一的 flex 子元素设置 align-self 值来使其与同一个 flex 容器的其它 flex 子元素不一样。

在下面的例子中，我使用了 align-items 属性来设置 flex 子元素组的排列方式是 center，但是同时也给第一个和最后一个设置了 align-self 属性来改变他们的排列方式。

**为什么没有 justify-self？**

一个很常见的问题是为什么不能在主轴上排列单个元素或元素组？ 为什么在主轴上没有 -self 排列属性？如果你认为 justify-content 和 align-content 的作用是关于空余空间分布的，那么它们没有单独的排列方法的理由就显而易见了。我们将flex 子元素作为一整个组进行处理，并以某种方式分配可用空间——在组的开头及结尾或 flex 子元素之间。

想想 justify-content 和 align-content 在 CSS Gird 布局中如何起作用也是很有帮助的。这两个属性用于描述在 gird 容器和 gird 块之间的空余空间如何分配。再次地，我们将 gird 块当作一个组，然后这些属性决定他们之间的所有额外空间。正如我们在 Gird 和 Flexbox 中展示的作用那样，我们不能指定某一个元素去做一些不一样的事情。不过，有一个方法可以实现你想要的在主轴上类似 self 属性的布局，那就是使用自动外边距。

**在主轴上使用自动外边距**

在 Flexbox 中使用自动外边距来排列主轴上的单个元素或者一组元素的效果非常好。在下面的例子中，我们实现了一个共同的设计模式。我有一个使用 Flexbox 的导航栏，其子元素以行的形式排列同时使用了默认值 justify-content: start。我想让最后的那个子元素和其它子元素分开并展示在 flex 排列行的最后面——假设该行有足够的空间。

我定位到了那个元素并且把它的 margin-left 属性设置成了 auto。这意味着它的外边距将会尽可能地占用它左边的空间，这意味着那个子元素被推到了最右边。

<div class="container" id="container">

<div class="item">One</div>

<div class="item">Two Two</div>

<div class="item push">Three Three Three</div>

</div>复制代码

.container {

display: flex;

}

.item {

padding: 10px;

}

.push {

margin-left: auto;

}

如果你在主轴上使用了自动外边距，那 justify-content 将不会有任何作用，因为自动外边距将会占据所有之前用在 justify-content 上的空间。

**回退排列**

每一个排列方法详细来说都会有一个回退排列，它会说明在你请求的排列方式无法实现时会发生什么。举个例子，如果在你的 flex 容器中只有一个子元素但你声明了 justify-content: space-between，会发生什么？答案是（该属性的）回退排列 flex-start 会让你唯一的那个子元素排列在 flex 容器的开始之处。对于 justify-content: space-around，回退排列 center 将会被使用。

在现在的规范中你不能改变回退排列的值，所以如果你希望 space-between 的回退值是 center 而不是 flex-start 的话，并没有方法能实现。这是 一份规范笔记，它描述了未来版本可能会支持这种方式。

**安全和非安全的排列**

最新的一个添加到盒子元素排列规范的是使用

safe 和 unsafe

关键词的关于安全和非安全的排列的概念。

看下面的代码，最后一个元素相较于容器太宽了同时是 unsafe 排列并且 flex 容器是在页面左边的，当子元素溢出界面之外时，其被裁减了。

.container {

display: flex;

flex-direction: column;

width: 100px;

align-items: unsafe center;

}

.item:last-child {

width: 200px;

}

不安全的排列将会按照你定义的排列但可能导致界面数据丢失（放大预览）

安全的排列将会保护界面数据免于丢失，方式是通过重新移动溢出区间到其他地方：

.container {

display: flex;

flex-direction: column;

width: 100px;

align-items: safe center;

}

.item:last-child {

width: 200px;

}

安全的排列会尝试避免数据丢失（放大预览）

这些关键词现在还很少有浏览器支持（译者注：自测 Chrome 69 已支持），不过，它展示了在盒子元素排列规范中带给了 Flexbox 额外的控制方式。

HTML:

<div class="container" id="container">

<div class="item">One</div>

<div class="item">Two Two</div>

<div class="item">Three Three Three</div>

</div>

.container {

padding: 10px;

display: flex;

flex-direction: column;

width: 100px;

align-items: safe center;

}

.item {

padding: 10px;

background-color: rgba(111,41,97,.3);

border: 2px solid rgba(111,41,97,.5);

}

.item:last-child {

width: 200px;

}

**总结**

Flexbox 的排列属性最初以列表的方式出现，但是现在它们有了自己的规范同时也适用于其它的布局环境。这里是一些小知识可能帮助你如何在 Flexbox 中使用它们：

justify- 适用于主轴，align- 适用于交叉轴；

使用 align-content 和 justify-content 时你需要空余空间；

align-content 和 justify-content 属性面向的是子元素组、作用是分享空间。因此，你不能指定一个特定的子元素同时它们也没有对应 -self 排列属性；

如果你想去排列一个子元素，或者在主轴上分离出一个组，请用自动外边距实现；

align-items 属性设置了整个子元素组的所有 align-self 值。可以通过设置 align-self 属性来设置一个特定的子元素。

一个完整的Flexbox指南

<https://www.w3cplus.com/css3/a-guide-to-flexbox.html>

Flexbox布局（Flexible Box)模块旨在提供一个更加有效的方式制定、调整和分布一个容器里的项目布局，即使他们的大小是未知或者是动态的。（这里我们称为Flex）。

Flex布局主要思想是让容器有能力让其子项目能够改变其宽度、高度(甚至顺序)，以最佳方式填充可用空间（主要是为了适应所有类型的显示设备和屏幕大小）。Flex容器会使子项目（伸缩项目）扩展来填满可用空间，或缩小他们以防止溢出容器。

最重要的是，Flexbox布局方向不可预知，他不像常规的布局（块就是从上到下，内联就从左到右）。而那些常规的适合页面布局，但对于支持大型或者杂的应用程序（特别是当他涉及到取向改变、缩放、拉伸和收缩等）就缺乏灵活性。

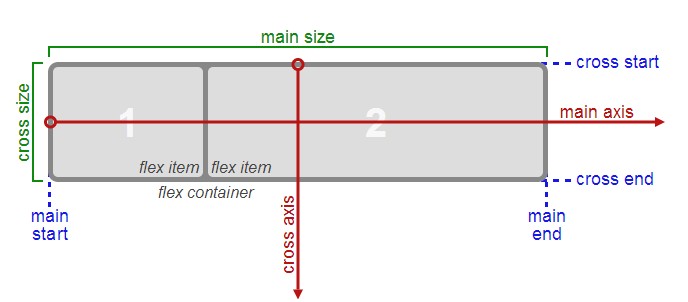
注：Flexbox布局最适合应用程序的组件和小规模的布局，而网格布局更适合那些更大规模的布局。

**一、基本要素**

因为Flexbox是整个模块，而不是一个属性，它涉及很多东西，包括其整个组属性。他们当中一部分是容器

（父元素，称为“伸缩容器”），另一部分是子元素（称为“伸缩项目”）。

常规布局是基于块和内联流方向，而Flex布局是基于flex-flow流。请看看来自w3c规范中的这张图，解释了flex布局的主要思想。



基本上，伸缩项目是沿着主轴（main axis），从主轴起点（main-start）到主轴终点（main-end）或者沿着侧轴（cross axis），从侧轴起点（cross-start）到侧轴终点（cross-end）排列。

**主轴（main axis）**：伸缩容器的主轴，伸缩项目主要沿着这条轴进行排列布局。小心，它不一定是水平的；这主要取决于“justify-content”属性（详细见下文）。

**主轴起点（main-start）和主轴终点（main-end）**：伸缩项目放置在伸缩容器内从主轴起点（main-start）向主轴终点（main-start）方向。

**主轴尺寸（main size）**：伸缩项目在主轴方向的宽度或高度就是主轴的尺寸。伸缩项目主要的大小属性要么是宽度，要么是高度属性，由哪一个对着主轴方向决定。

**侧轴（cross axis）**：垂直于主轴称为侧轴。它的方向主要取决于主轴方向。

**侧轴起点（cross-start）和侧轴终点（cross-end）**：伸缩行的配置从容器的侧轴起点边开始，往侧轴终点边结束。

**侧轴尺寸（cross size）：**伸缩项目的在侧轴方向的宽度或高度就是项目的侧轴长度，伸缩项目的侧轴长度属性是「width」或「height」属性，由哪一个对着侧轴方向决定。

**二、父级（弹性容器）属性**

**display:flex | inline-flex**;(适用于伸缩容器，也就是伸缩项目的父元素)

这个是用来定义伸缩容器，是内联还是块取决于设置的值。这个时候，他的所有子元素将变成flex文档流，称为伸缩项目。

display: other values | flex | inline-flex;

请注意：

CSS的columns在伸缩容器上没有效果。

float、clear和vertical-align在伸缩项目上没有效果。

flex-direction（适用于伸缩容器，也就是伸缩项目的父元素）

这个主要用来创建主轴，从而定义了伸缩项目放置在伸缩容器的方向。

**flex-direction**: row | row-reverse | column | column-reverse

row(默认值)：在“ltr”排版方式下从左向右排列；在“rtl”排版方式下从右向左排列。

row-reverse：与row排列方向相反，在“ltr”排版方式下从右向左排列；在“rtl”排版方式下从左向右排列。

column：类似 于row，不过是从上到下排列

column-reverse：类似于row-reverse，不过是从下到上排列。

主轴起点与主轴终点方向分别等同于当前书写模式的始与结方向。其中“ltr”所指文本书写方式是“left-to-right”也就是从左向右书写；而“rtl”所指的刚好与“ltr”方式相反，其书写方式是“right-to-left”，也就是从右向左书写。

**flex-wrap**(适用于伸缩容器，也就是伸缩项目的父元素)

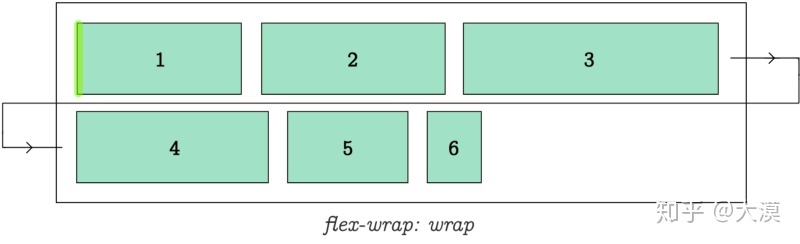
flex-wrap用来决定Flex项目的宽度和超过Flex容器的宽度时，Flex项目换不换行排列。前面也看到了，默认值为nowrap是不换行，如果你把值换成wrap，Flex容器容不下的时候就会自动换行。

flex-wrap: nowrap | wrap | wrap-reverse

nowrap (默认值) : 所有的 flex 项都会在同一行上排布，也就是我们常说的单行，或不换行。

wrap: flex 项将从上到下根据实际情况排布再多行上，也就是我们常说的多行，或会换行。

wrap-reverse: flex 项将 从下到上 根据实际情况排布再多行上折行。

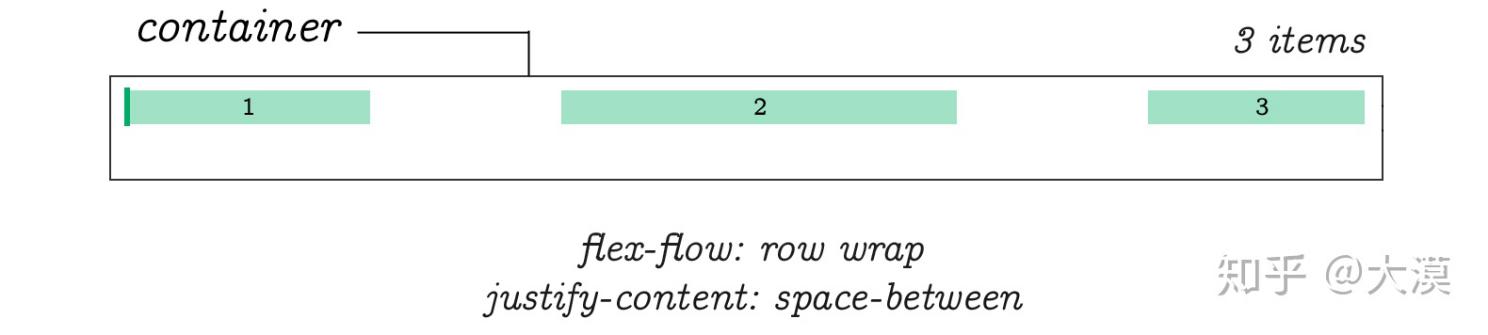
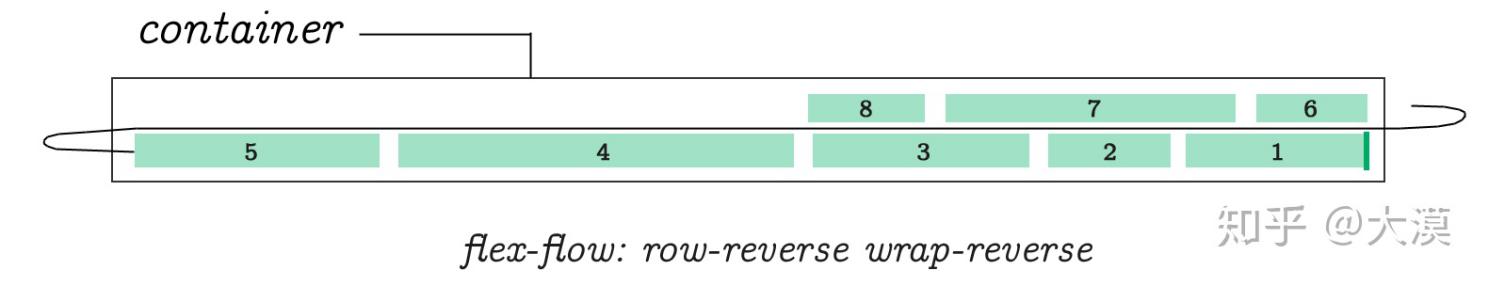
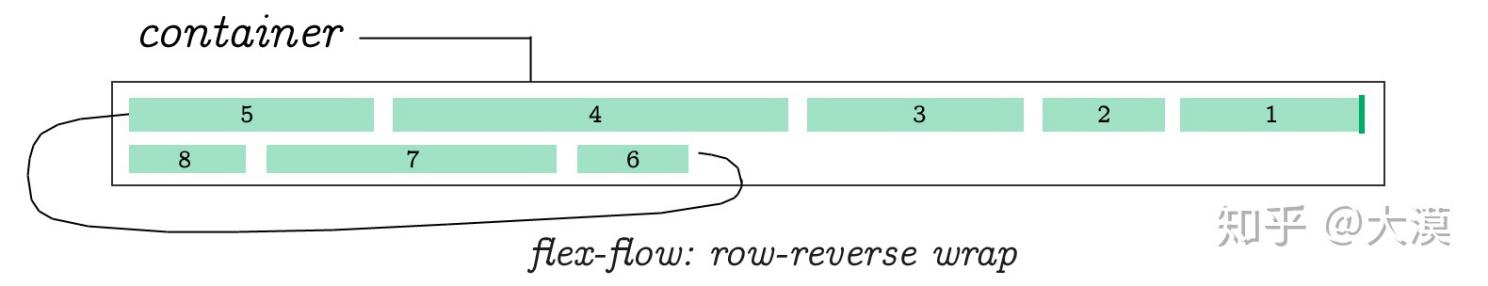
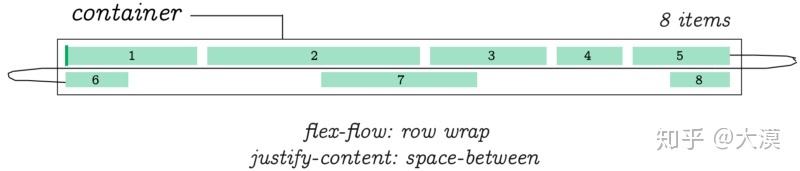
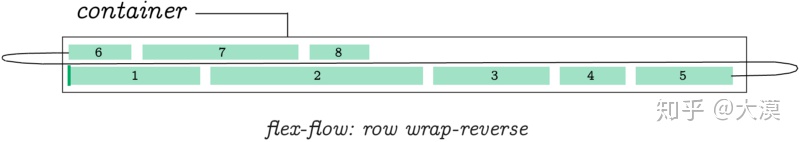
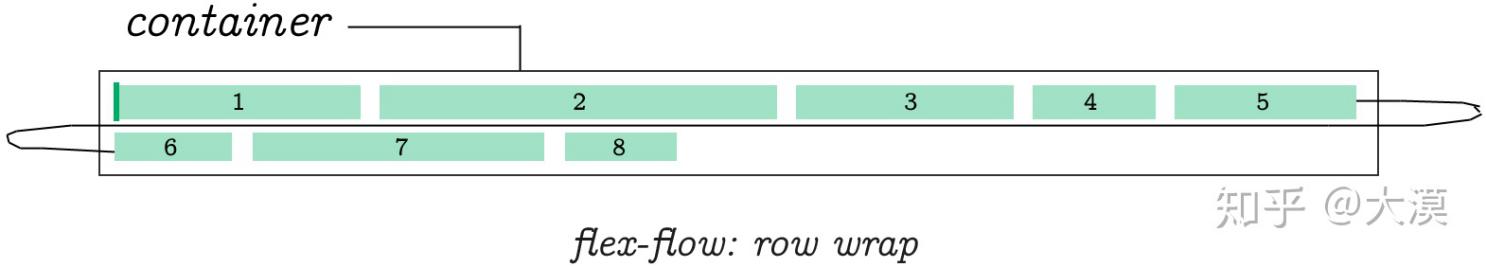
****

**flex-flow**（适用于伸缩容器，也就是伸缩项目的父元素）

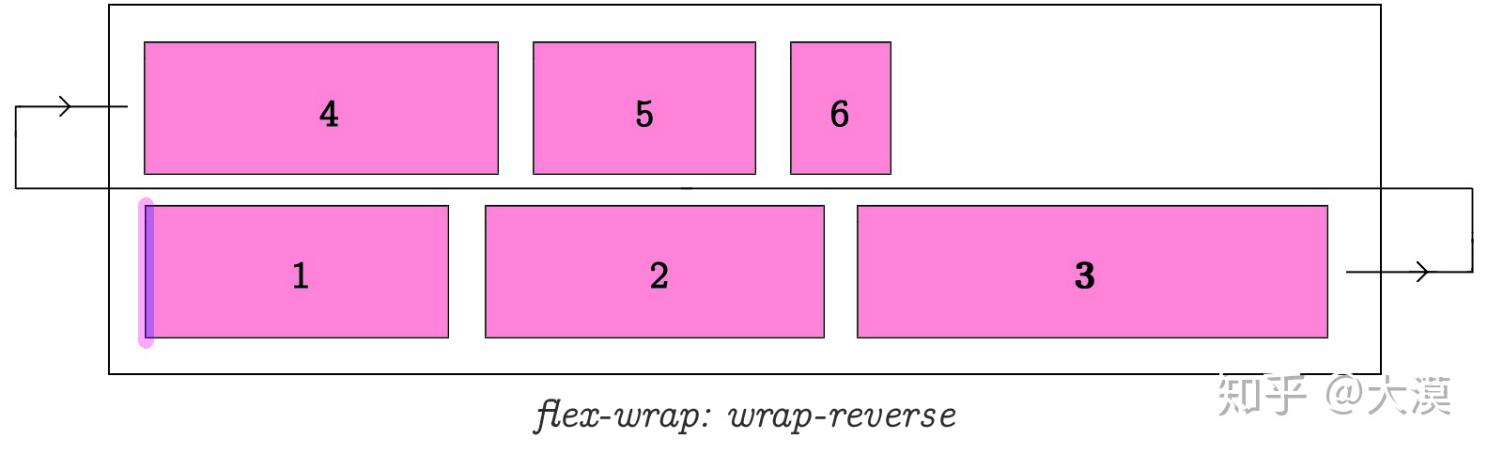
这个是“flex-direction”和“flex-wrap”属性的缩写版本。同时定义了伸缩容器的主轴和侧轴。其默认值为“row nowrap”。

flex-flow: <‘flex-direction’> || <‘flex-wrap’>

在Flexbox布局中除了flex-direction和flex-wrap来控制之外，还可以使用它们的的简写属性flex-flow。具体的情形主要有：



当flex-direction取值为column时，flex-flow属性的行为与前面的示例完全相同。



**justify-content**（适用于伸缩容器，也就是伸缩项目的父元素）

这个是用来定义伸缩项目沿着主轴线的对齐方式。当一行上的所有伸缩项目都不能伸缩或可伸缩但是已经达到其最大长度时，这一属性才会对多余的空间进行分配。当项目溢出某一行时，这一属性也会在项目的对齐上施加一些控制。

justify-content: flex-start | flex-end | center | space-between | space-around

\*\*flex-start(默认值)：伸缩项目向一行的起始位置靠齐。

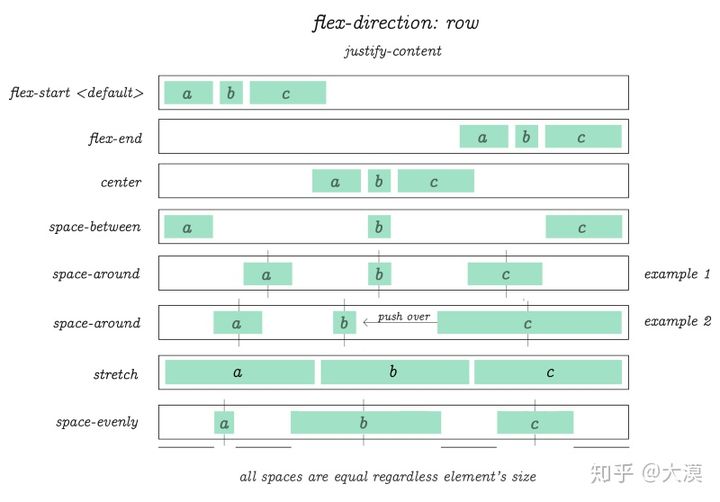
\*\*flex-end：伸缩项目向一行的结束位置靠齐。

\*\*center：伸缩项目向一行的中间位置靠齐。

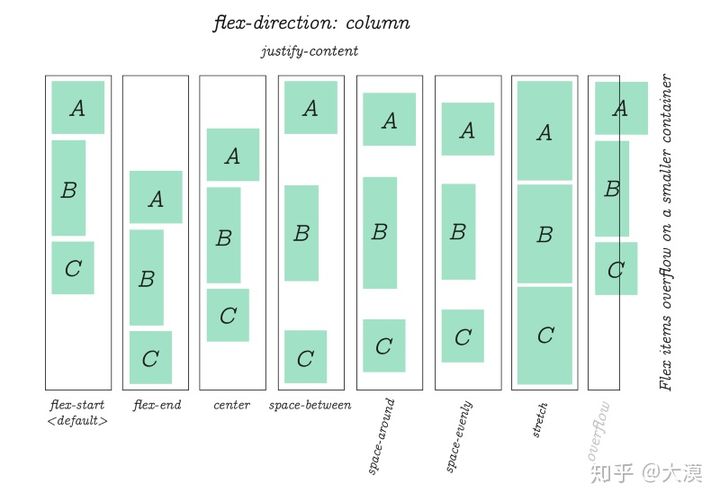
\*\*space-between：伸缩项目会平均地分布在行里。第一个伸缩项目一行中的最开始位置，最后一个伸缩项目在一行中最终点位置。

\*\*space-around：伸缩项目会平均地分布在行里，两端保留一半的空间。

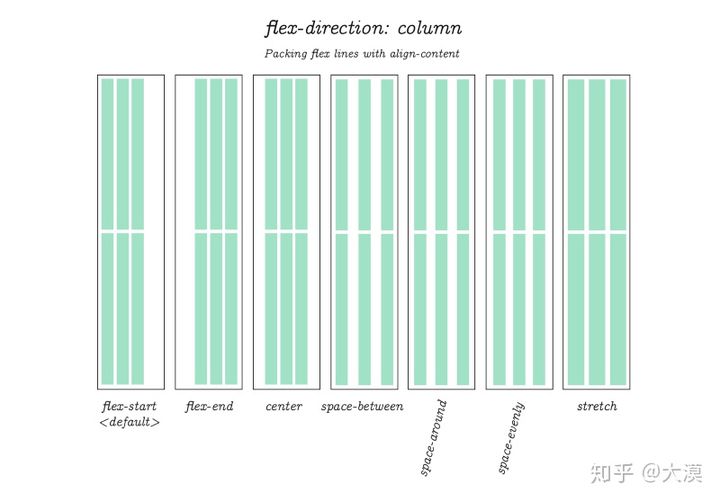
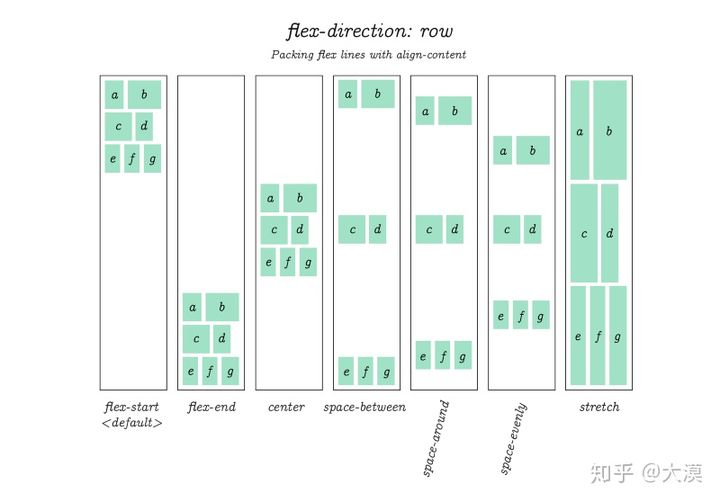
\*\*space-evenly: 任何两个 flex 项之间的间距（以及到 flex 容器边缘的空间）相等。（注：该属性以前很少看到，原因是以前浏览器不支持，chrome 也是 60 版本之后才支持。延伸一下，align-content: space-evenly 也是这个逻辑，大家可以查看下面的demo。 ）



上图的示例中，每行只有三个Flex项目。如果Flex项目很多的时候，这些属性同样适用。只不过看你是否配合前面的flex-flow来做其他相关的排列控制。



上图看到的效果都是diplay: flex时的场景。但display还可以取值为inline-flex。此时你看到的情景如下：



**align-item**（适用于伸缩容器，也就是伸缩项目的父元素）

这个主要用来定义伸缩项目可以在伸缩容器的当前行的侧轴上对齐方式。可以把他想像成侧轴（垂直于主轴）的“justify-content”。

align-items: flex-start | flex-end | center | baseline | stretch

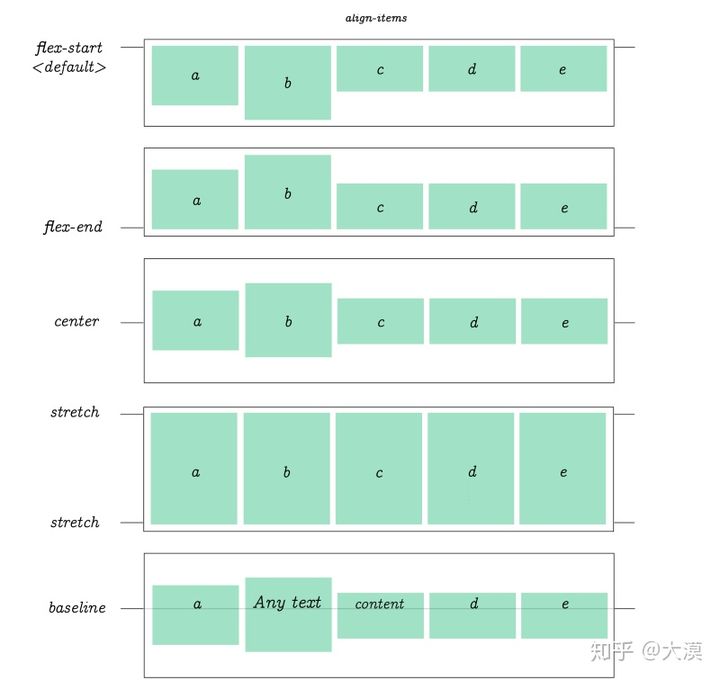
flex-start：伸缩项目在侧轴起点边的外边距紧靠住该行在侧轴起始的边。

flex-end：伸缩项目在侧轴终点边的外边距靠住该行在侧轴终点的边 。

center：伸缩项目的外边距盒在该行的侧轴上居中放置。

baseline：伸缩项目根据他们的基线对齐。

stretch（默认值）：伸缩项目拉伸填充整个伸缩容器。此值会使项目的外边距盒的尺寸在遵照「min/max-width/height」属性的限制下尽可能接近所在行的尺寸。



**align-content**

如果你给你的 flex 容器添加了 flex-wrap: wrap 同时也有好几条 flex 排列行，那你可以用 align-content 属性来在交叉轴上排列你的 flex 排列行。不过，这将会需要交叉轴上有额外的空间。在下面这个 demo 中，我的交叉轴作为竖直的列在运行，同时我设置了这个 flex 容器的高度为 60vh。由于这个高度比我展示 flex 子元素所需的高度大，所以我的容器有了交叉轴方向上的空余空间。

***注意：当只有一行 flex 项时，此属性不起作用。***

CSS 代码:

.container {

align-content: flex-start | flex-end | center | space-between | space-around | stretch;

}

flex-start：多行在容器的开始位置排布

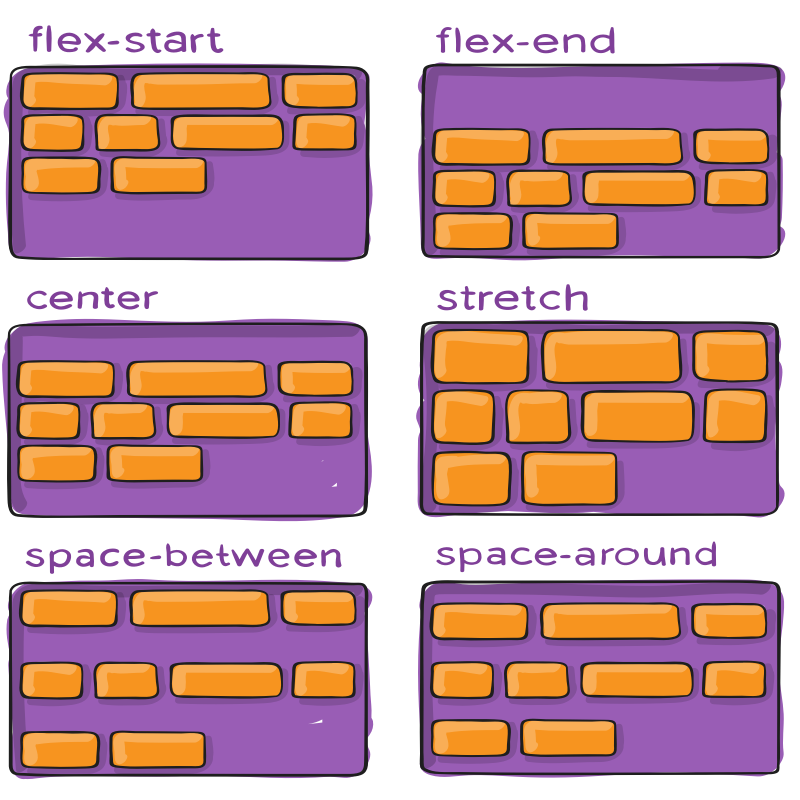
flex-end：多行在容器的结束位置排布

center：多行在容器的总结位置排布

space-between：多行均匀分布；第一行分布在容器的开始位置，最后一行分布在容器的结束位置

space-around: 多行均匀分布，并且每行的间距（包括离容器边缘的间距）相同；

strech (默认值)：多行拉伸以填充满整个剩余空间

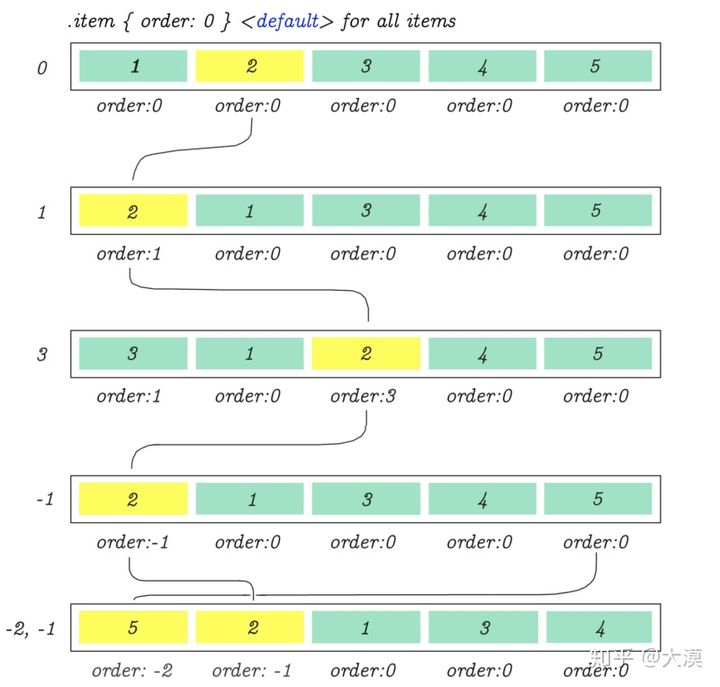


1. **子项（弹性项）的属性**

**order**（适用于伸缩项目，也就是伸缩容器的子元素）

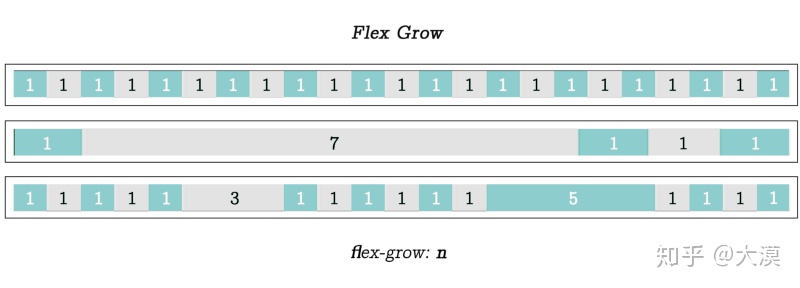
默认情况下，伸缩项目是按照文档流出现先后顺序排列。然而，“order”属性可以控制伸缩项目在他们的伸缩容器出现的顺序。

order: <integer>



**flex-grow**（适用于伸缩项目，也就是伸缩容器的子元素）

flex-grow应用到一个Flex项目时，它将相对于同一行上所有其他Flex项目的大小的总和进行缩放。该值将根据指定的值自动调整。比如上图中的flex-grow分别设置为1、7、3和5的效果。

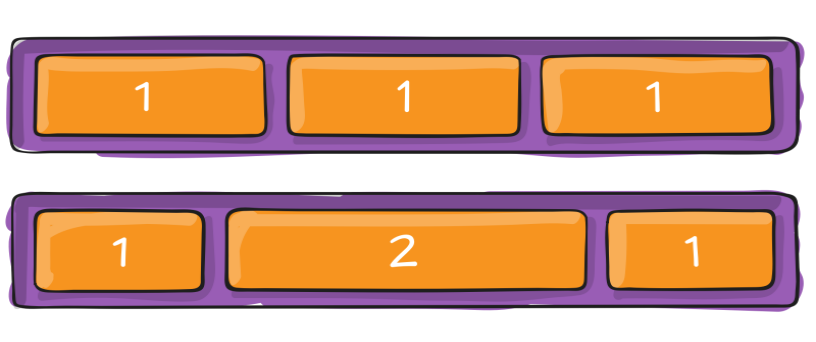


根据需要用来定义伸缩项目的扩展能力。它接受一个不带单位的值做为一个比例。主要用来决定伸缩容器剩余空间按比例应扩展多少空间。

如果所有伸缩项目的“flex-grow”设置了“1”，那么每个伸缩项目将设置为一个大小相等的剩余空间。如果你给其中一个伸缩项目设置了“flex-grow”值为“2”，那么这个伸缩项目所占的剩余空间是其他伸缩项目所占剩余空间的两倍。

flex-grow: <number> (默认值为： 0)

负值同样生效。



*注，举个例子帮助你理解：*

*比如我们得 flex 容器中有 6 个 flex 项，每个 flex 项的 flex-grow 初始值都是 1。如果我们将每个 flex 项的 flex-grow 相加起来，总和为 6。因此 flex 容器的总宽度被平均分成了 6 份。每个 flex 项增长到填充容器可用空间的1/6。*

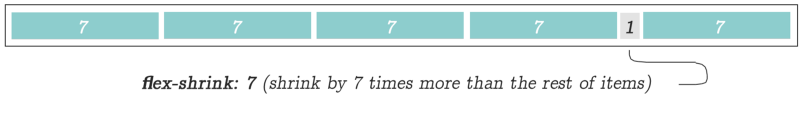
*当我们将第 3 个 flex 项的 flex-grow 设置为 2 时，flex 容器现在被分成了 7 等份，因为所有 flex-grow 属性是：1 + 1 + 2 + 1 + 1 + 1。第 3 个 flex 项占了整个容器空间的 2/7，其他的占了 1/7。*

**flex-shrink**（适用于伸缩项目，也就是伸缩容器的子元素）

根据需要用来定义伸缩项目收缩的能力。flex-shrink和flex-grow刚好相反。

flex-shrink: <number> (默认值为： 1)

负值同样生效。



**flex-basis**(适用于伸缩项目，也就是伸缩容器的子元素)

这个用来设置伸缩基准值，剩余的空间按比率进行伸缩。

flex-basis 定义了在分配剩余空间之前 flex 项默认的大小。可以设置为某个长度值（e.g. 20%, 5rem,等）或者关键字。关键字 auto 意味着 flex 项会按照其本来的大小显示（暂时由 main-size 关键字完成，直到弃用）。关键字 content 意味着根据内容来确定大小——这个关键字到目前没有被很好地支持，所以测试起来比较困难，与content的类似的关键字还有max-content, min-content, fit-content。

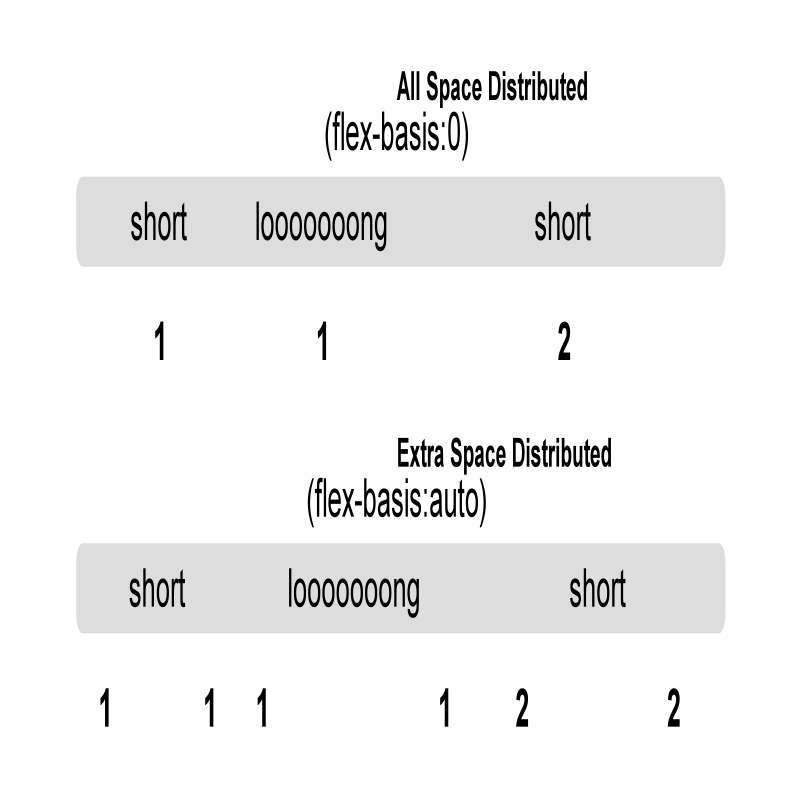
CSS 代码:

.item {

flex-basis: <length> | auto; /\* default auto \*/

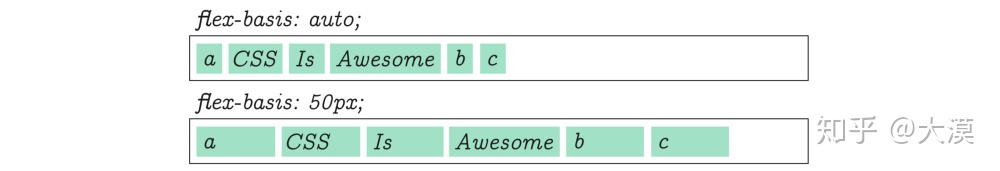
}

如果设置为 0 , 则 flex 项内容周围的空隙不会根据 flex-grow 按比例分配，如果设置为 auto，则 flex 项周围额外的空袭会根据 flex-grow 按照比例分配，如下图：



flex-basis: <length> | auto (默认值为： auto)

负值不合法。



**flex**（适用于伸缩项目，也就是伸缩容器的子元素）

这是“flex-grow”、“flex-shrink”和“flex-basis”三个属性的缩写。其中第二个和第三个参数（flex-shrink、flex-basis）是可选参数。默认值为“0 1 auto”。

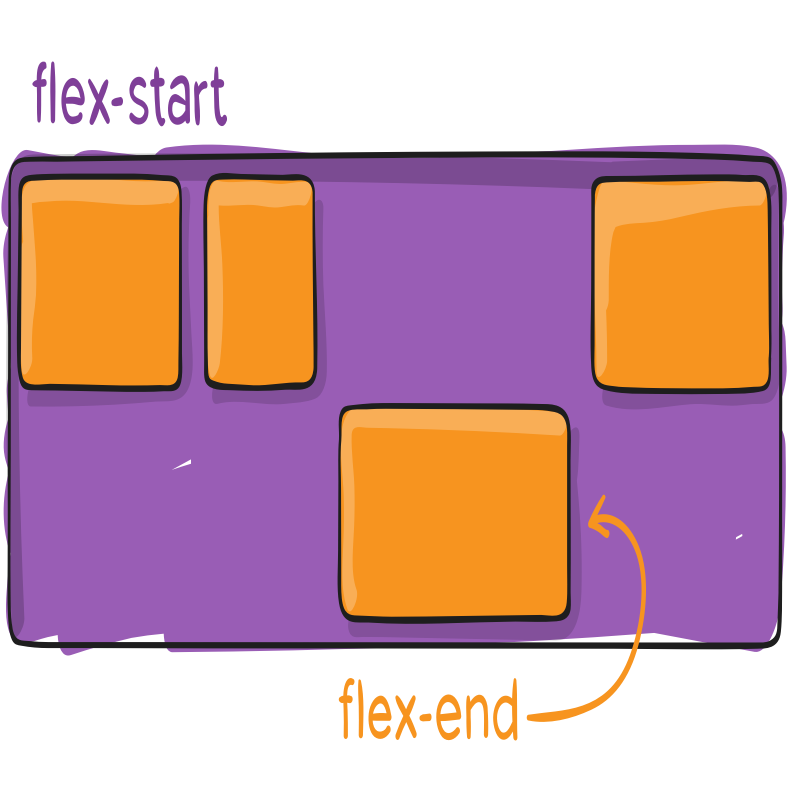
flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]

**align-self**（适用于伸缩项目，也就是伸缩容器的子元素）

用来在单独的伸缩项目上覆写默认的对齐方式。

属性值的介绍请参阅“align-items”的属性值。

align-self: auto | flex-start | flex-end | center | baseline | stretch



**justify-items**

这里要注意，justify-items和前面介绍的justify-content是不一样的。这个是用于Flex项目上，而justify-content是用于Flex容器上。

