12个flex布局属性

容器属性

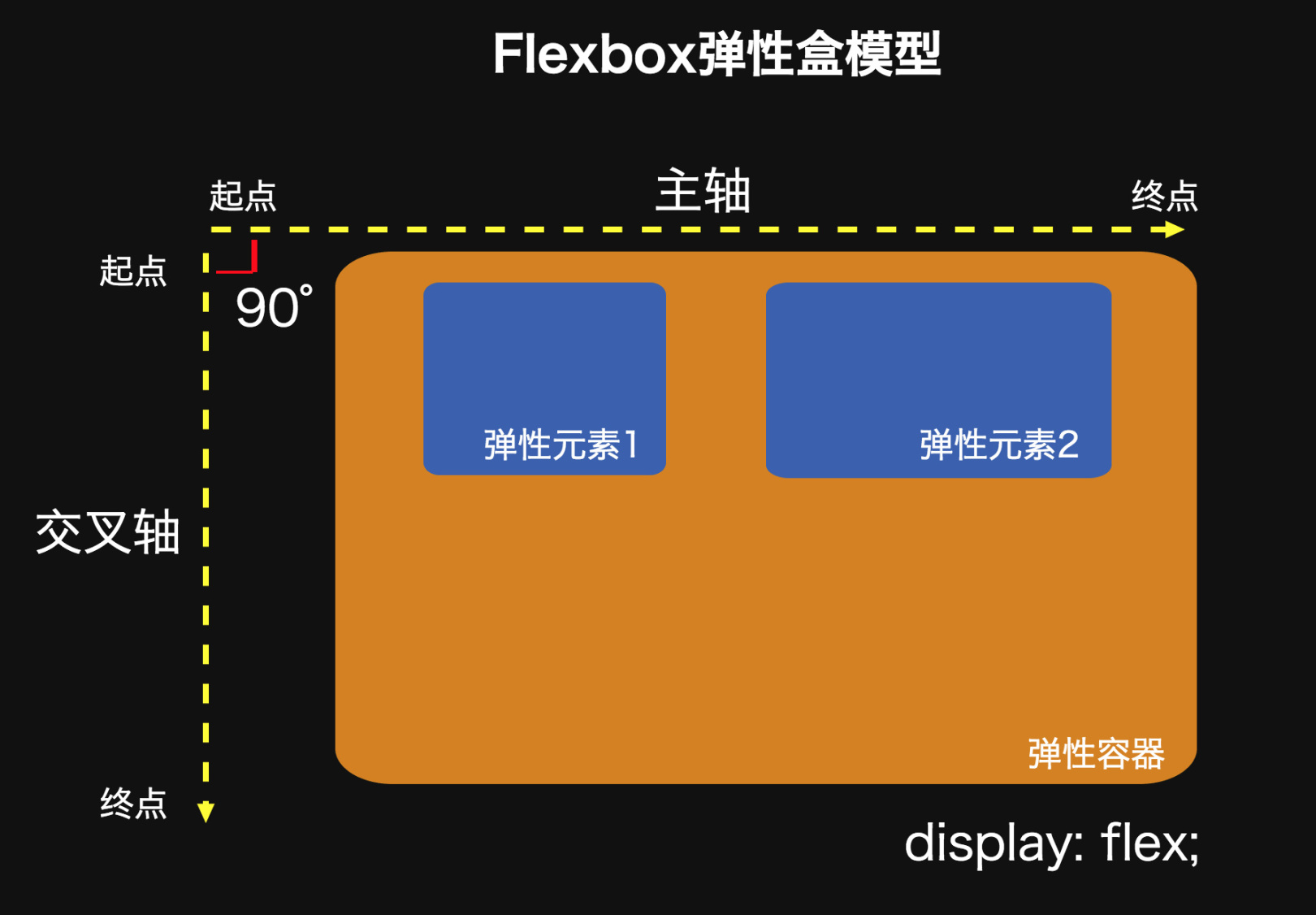
* flex-flow
* flex-direction
* flex-wrap
* justify-content
* align-items
* align-content

元素属性

* order
* flex-grow
* flex-shrink
* flex-basis
* flex
* align-self

就连老外也都在twitter吐槽不好理解，可见还是有一定的学习成本。

## **一、flex弹性盒模型**

对于某个元素只要声明了display: flex;，那么这个元素就成为了弹性容器

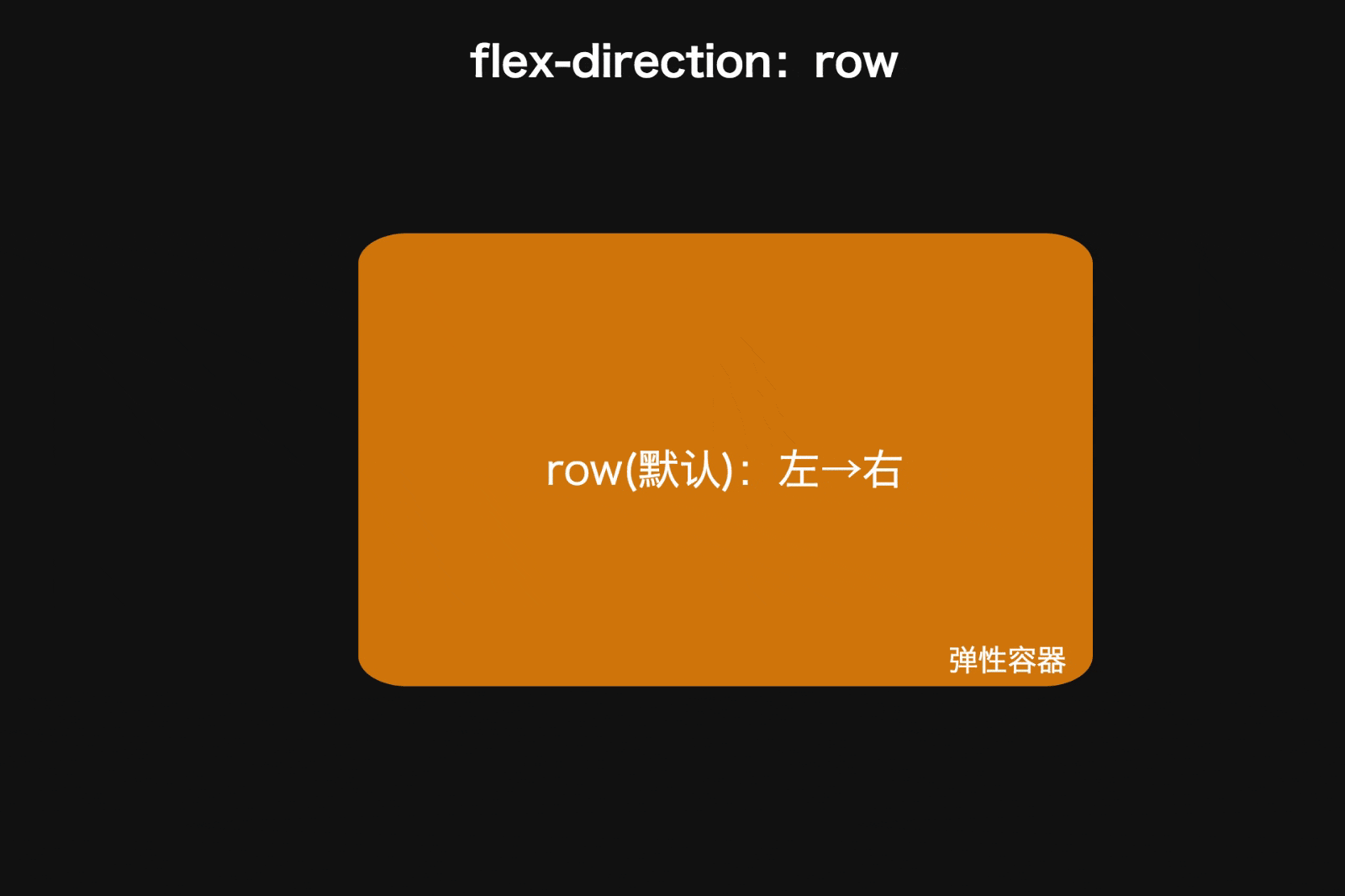
1. 每个弹性容器都有两根轴：**主轴和交叉轴**，两轴之间成90度关系。注意：**水平的不一定就是主轴。**
2. 每根轴都有**起点和终点**，这对于元素的对齐非常重要。
3. 弹性容器中的所有子元素称为<弹性元素>，**弹性元素永远沿主轴排列**。
4. 弹性元素也可以通过display:flex设置为另一个弹性容器，形成嵌套关系。因此**一个元素既可以是弹性容器也可以是弹性元素**。

## **二、主轴**

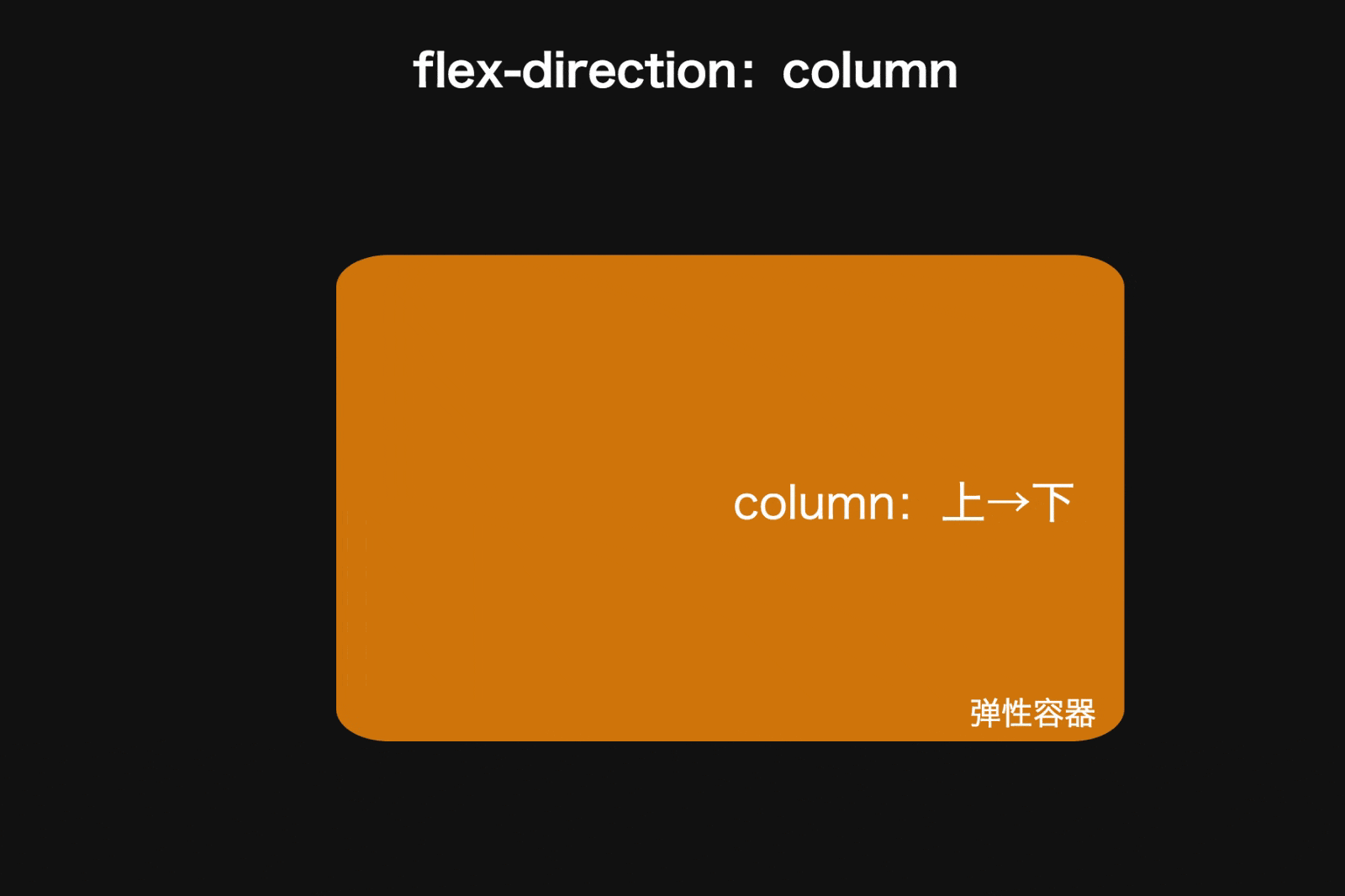
**flex布局大部分的属性都是作用于主轴的，在交叉轴上很多时候只能被动地变化**。

### **1. 主轴的方向**

我们可以在弹性容器上通过flex-direction修改主轴的方向。



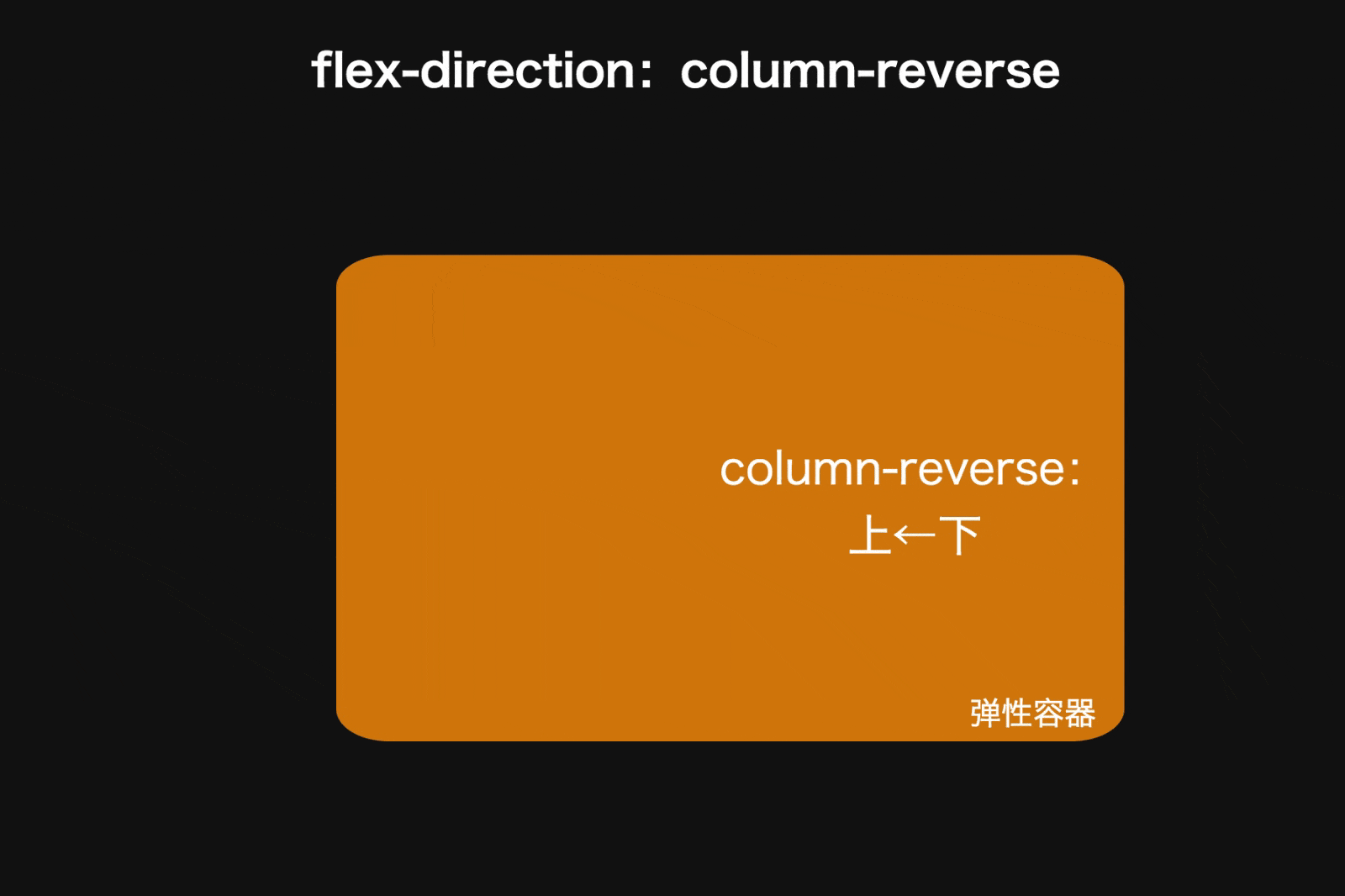
flex-direction:row



flex-direction:column



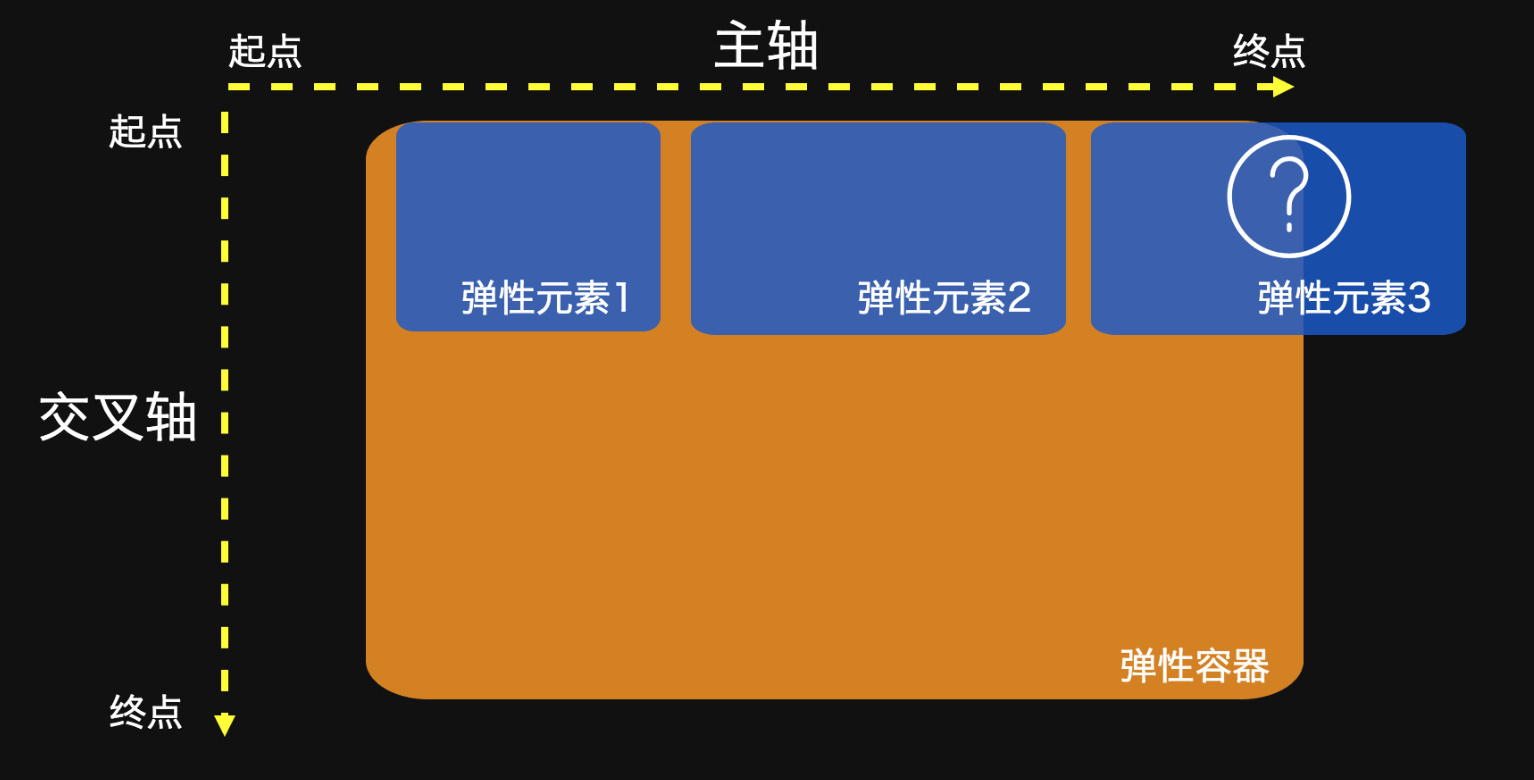
flex-direction:row-reverse



flex-direction:column-reverse

### **2. 沿主轴的排列处理**

弹性元素永远沿主轴排列，那么如果主轴排不下，该如何处理？

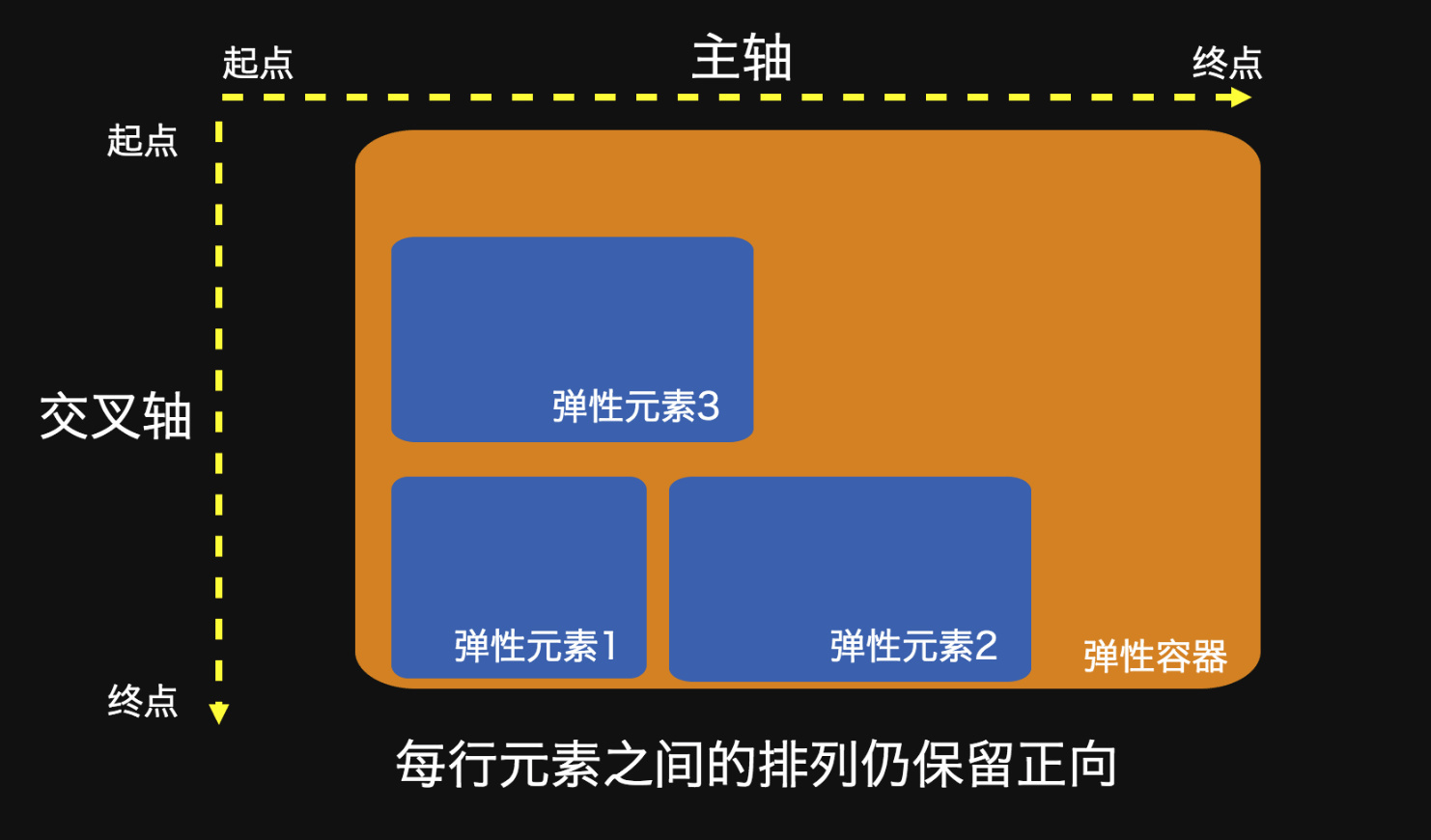


通过设置flex-wrap: nowrap | wrap | wrap-reverse可使得主轴上的元素不折行、折行、反向折行。

nowrap不折行，难道任由元素直接溢出容器吗？当然不会，那么这里就涉及到元素的弹性伸缩应对，下面会讲到。

wrap折行，顾名思义就是另起一行，那么折行之后行与行之间的间距（对齐）怎样调整？这里又涉及到交叉轴上的多行对齐。

wrap-reverse反向折行，是从容器底部开始的折行，但每行元素之间的排列仍保留正向。



### **3. 一个复合属性**

flex-flow = flex-drection + flex-wrap

## **三、元素如何弹性伸缩应对**

当flex-wrap: nowrap;不折行时，容器宽度有剩余/不够分

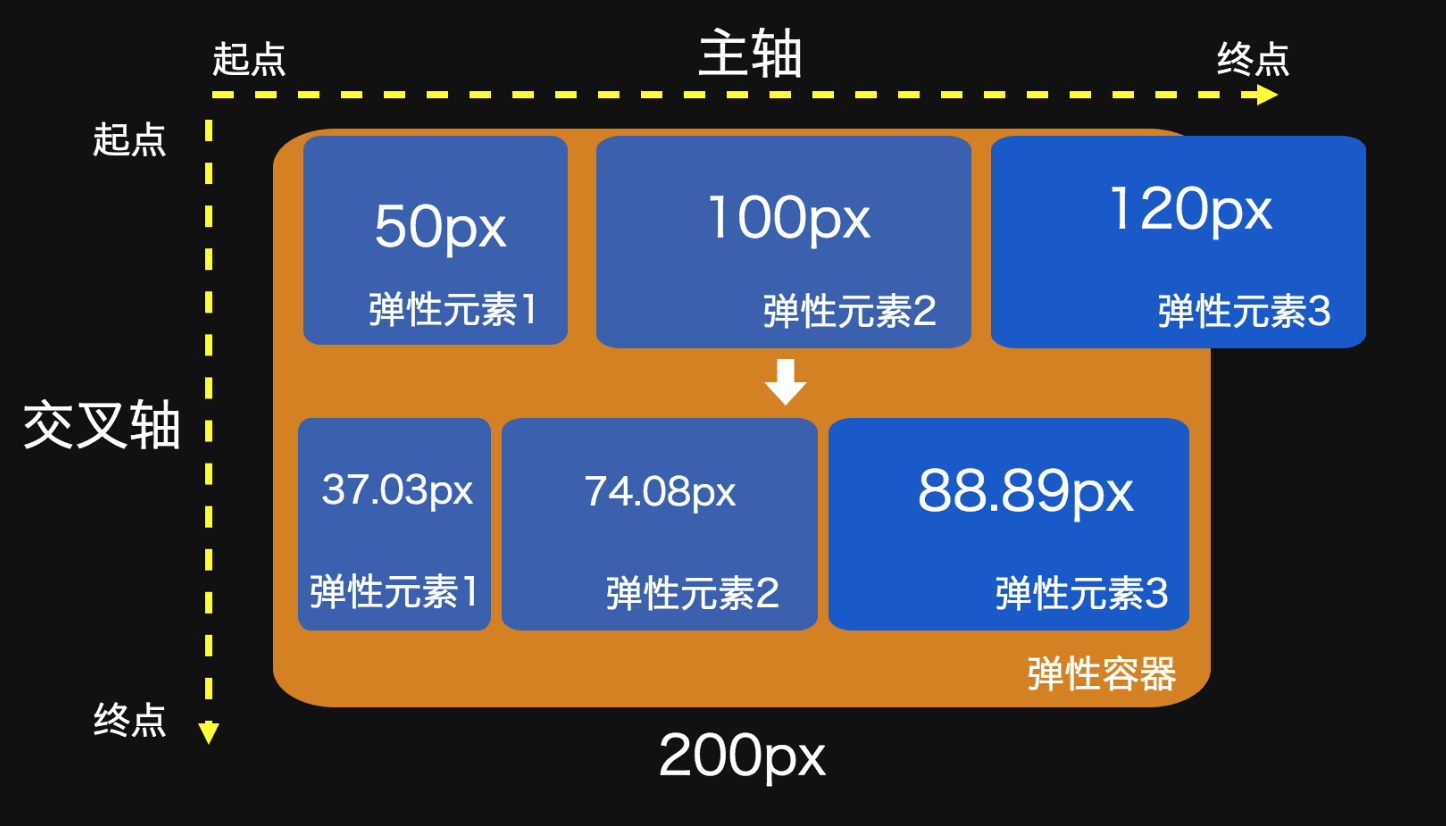
这里针对上面两种场景，引入两个属性(需应用在弹性元素上)

flex-shrink：缩小比例（容器宽度<元素总宽度时如何收缩）

flex-grow：放大比例（容器宽度>元素总宽度时如何伸展）

### **1. flex-shrink: 缩小比例**

flex-shrink默认为1，也就是当容器宽度不够分配时，元素都将等比例缩小，占满整个宽度，如下图。



#container {

display: flex;

flex-wrap: nowrap;

}

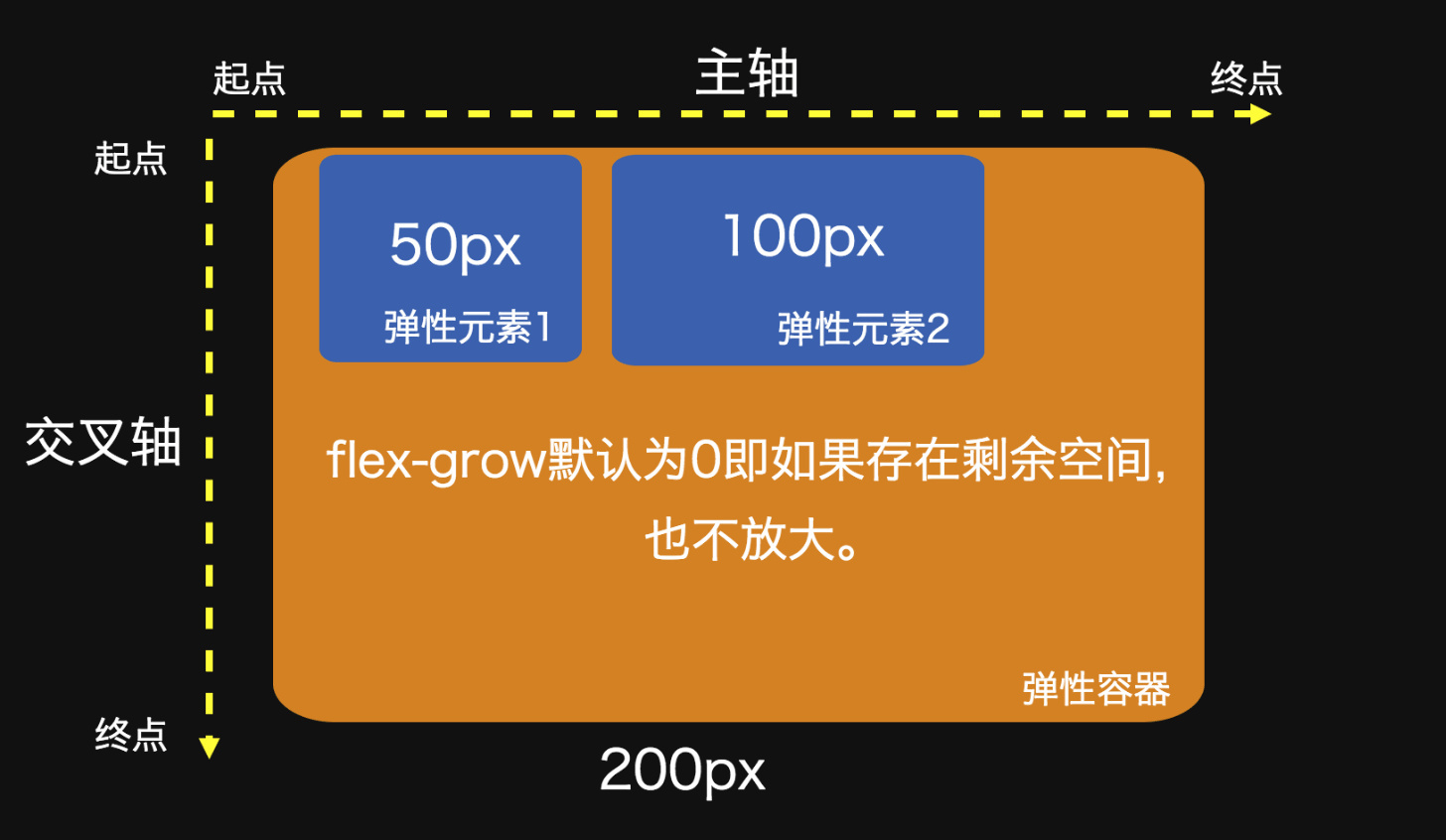
#### **元素收缩的计算方法**

先抛结论：flex-shrink: 1并非严格等比缩小，**它还会考虑弹性元素本身的大小**。

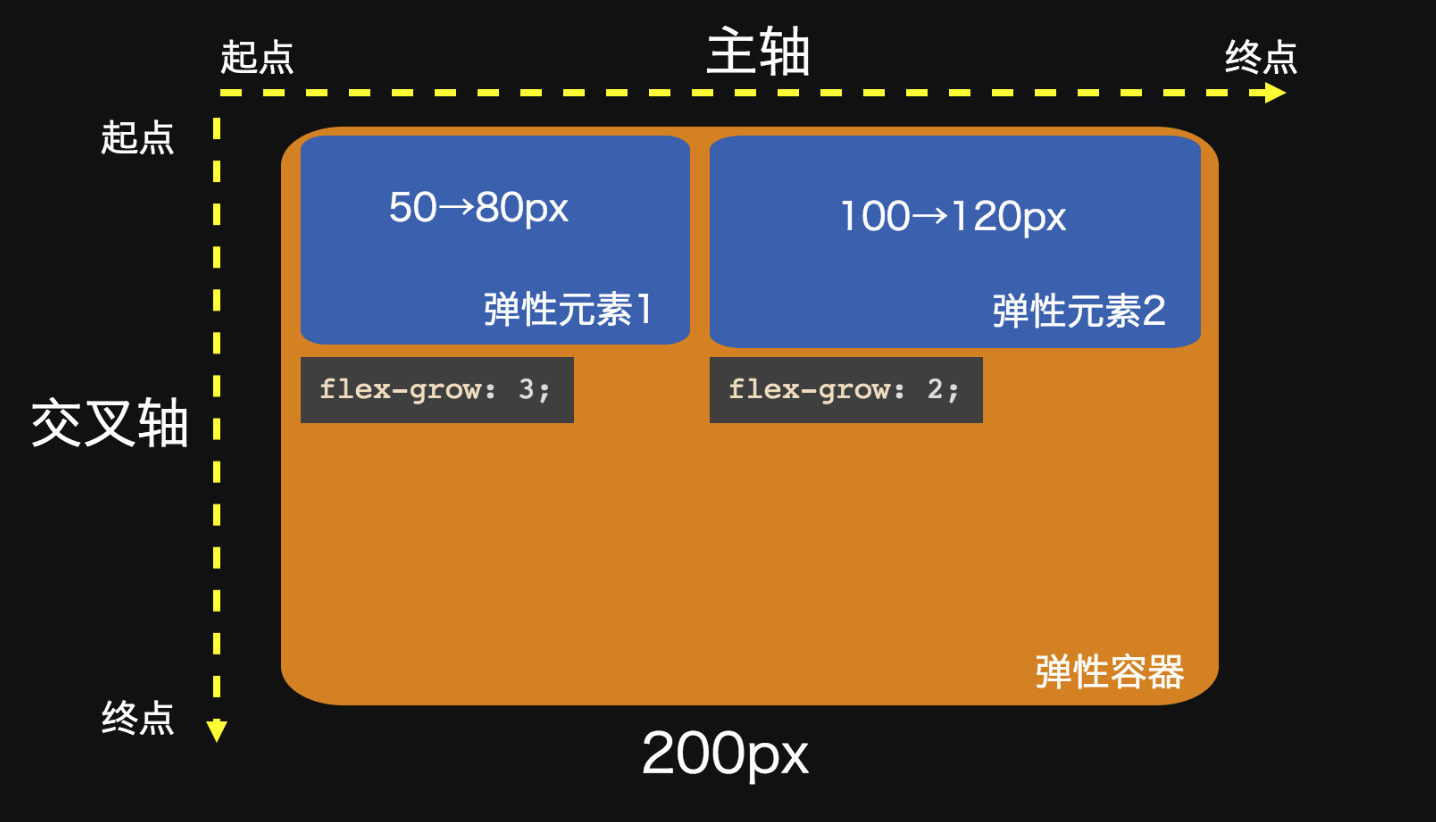
* 容器剩余宽度：-70px
* 缩小因子的分母：1\*50 + 1\*100 + 1\*120 = 270 (1为各元素flex-shrink的值)
* 元素1的缩小因子：1\*50/270
* 元素1的缩小宽度为缩小因子乘于容器剩余宽度：1\*50/270 \* (-70)
* 元素1最后则缩小为：50px + (1\*50/270 \*(-70)) = 37.03px

### **2. flex-grow: 放大比例**

（1）在flex布局中，容器剩余宽度默认是不进行分配的，也就是所有弹性元素的flex-grow都为0。



（2）通过指定flex-grow为大于零的值，实现容器剩余宽度的分配比例设置。



#### **元素放大的计算方法**

* 容器剩余宽度：50px
* 分成每份：50px / (3+2) = 10px
* 元素1放大为：50px + 3 \* 10 = 80px

#### **无多余宽度时，flex-grow无效**

下图中，弹性容器的宽度正好等于元素宽度总和，无多余宽度，此时无论flex-grow是什么值都不会生效。

## **四、弹性处理与刚性尺寸**

其实有些场景我们更希望元素尺寸固定，不需要进行弹性调整。

设置元素尺寸除了width和height以外，flex还提供了一个flex-basis属性。

flex-basis设置的是元素在主轴上的初始尺寸，所谓的初始尺寸就是元素在flex-grow和flex-shrink生效前的尺寸。

### **1. 与width/height的区别**

首先以width为例进行比较。看下下面的例子。#container {display:flex;}。

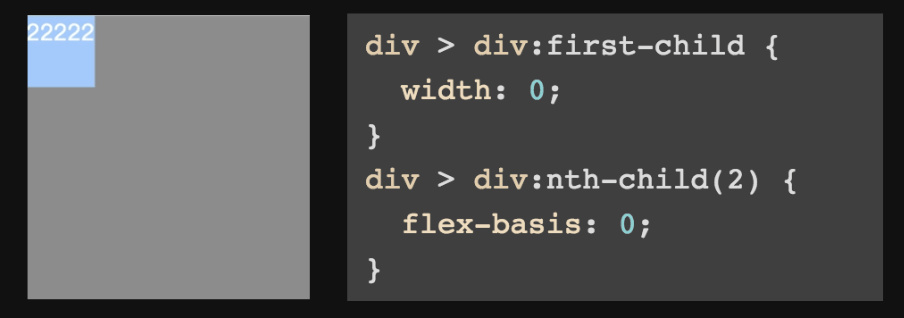
<div id="container">

<div>11111</div>

<div>22222</div>

</div>

#### **(1) 两者都为0**



* width: 0 —— 完全没显示
* flex-basis: 0 —— 根据内容撑开宽度

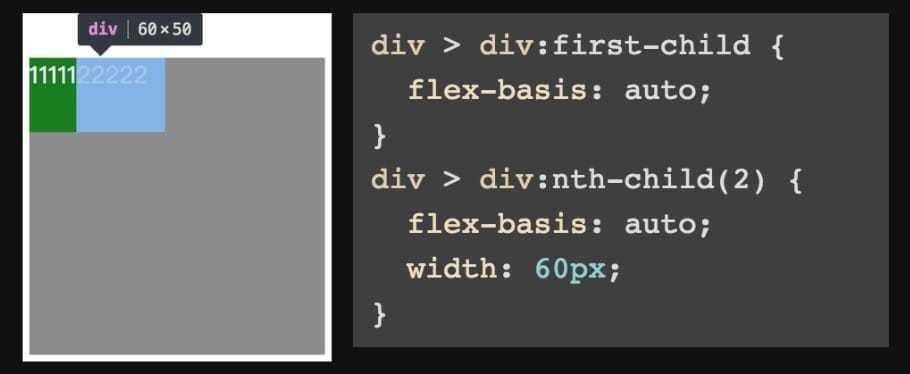
#### **(2) 两者非0**



—— 数值相同时两者等效

—— 同时设置，flex-basis优先级高

#### **(3) flex-basis为auto**



flex-basis为auto时，如设置了width则元素尺寸由width决定；没有设置则由内容决定

#### **(4) flex-basis == 主轴上的尺寸 != width**



* 将主轴方向改为：上→下
* flex-basis == height

### **2. 常用的复合属性 flex**

这个属性应该是最容易迷糊的一个，下面揭开它的真面目。

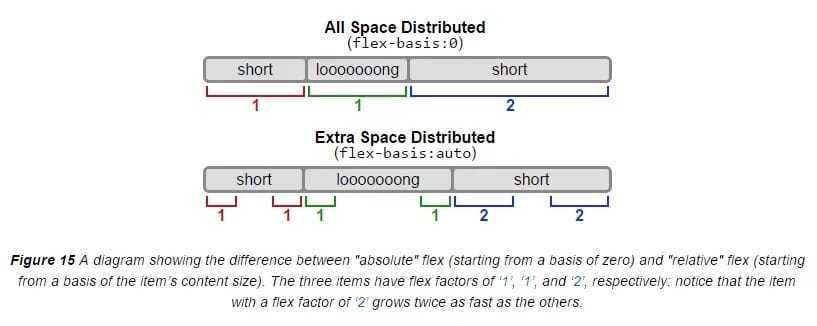
flex = flex-grow + flex-shrink + flex-basis

#### **flex:1 和 flex:auto 的区别**

其实可以归结于flex-basis:0和flex-basis:auto的区别。

flex-basis是指定初始尺寸，当设置为0时（绝对弹性元素），此时相当于告诉flex-grow和flex-shrink在伸缩的时候不需要考虑我的尺寸；

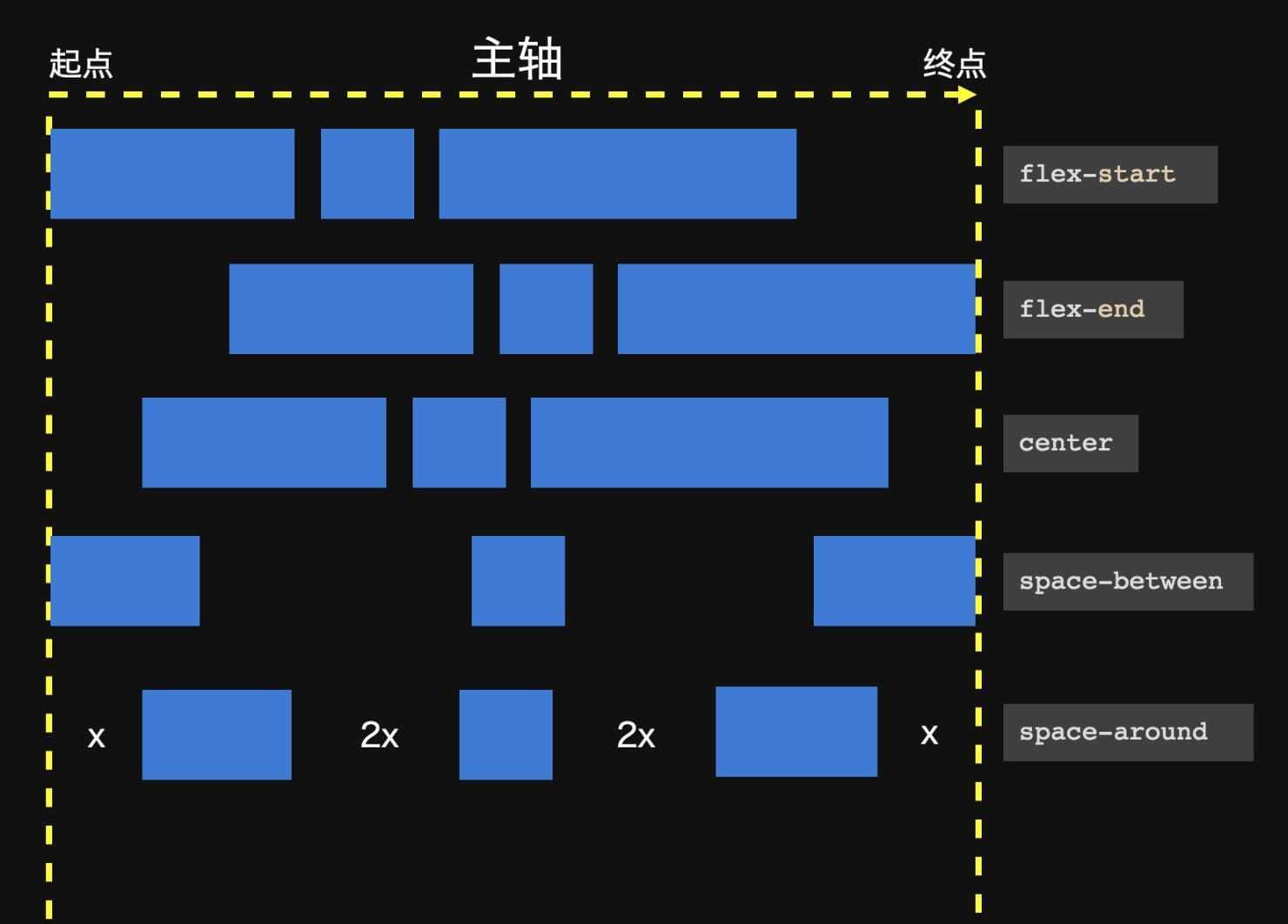
因此从下图（转自W3C）可以看到绝对弹性元素如果flex-grow值是一样的话，那么他们的尺寸一定是一样的。



## **五、容器内如何对齐**

### **1. 主轴上的对齐方式**

**justify-content**



### **2. 交叉轴上的对齐方式**

交叉轴上存在单行和多行两种情况。

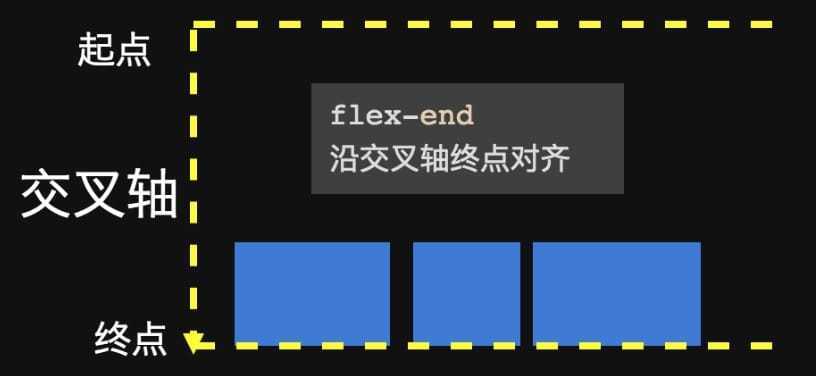
#### **交叉轴上的单行对齐**

**align-items**

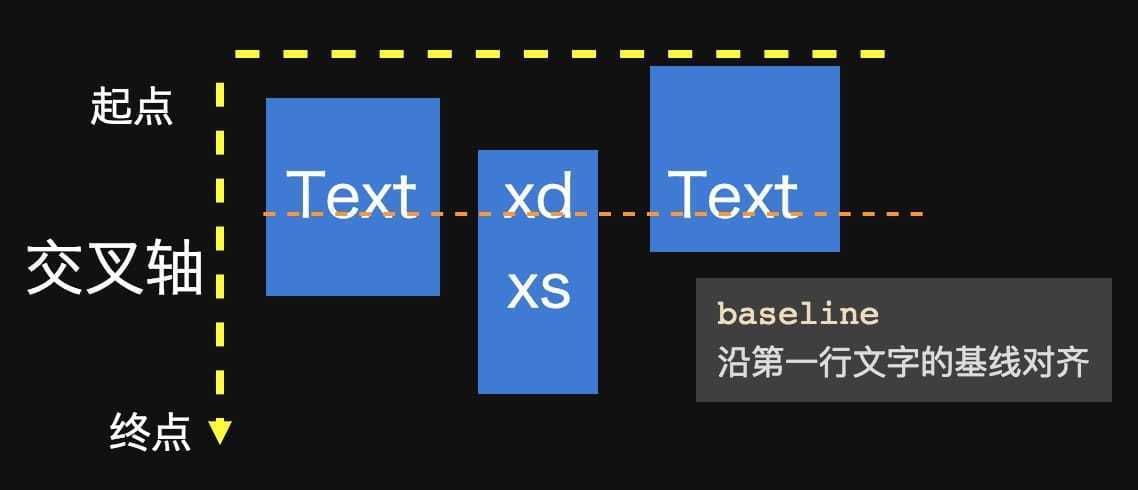
默认值是stretch，当元素没有设置具体尺寸时会将容器在交叉轴方向撑满。











注意，交叉轴不一定是从上往下，这点再次强调也不为过。

#### **交叉轴上的多行对齐**

多行情况下，flex布局提供了align-content属性设置对齐。

align-content只对多行元素有效，**会以多行作为整体进行对齐**，容器必须开启换行。

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

align-content比align-items多了两个值：space-between和space-around。

#### **align-content与align-items异同对比**

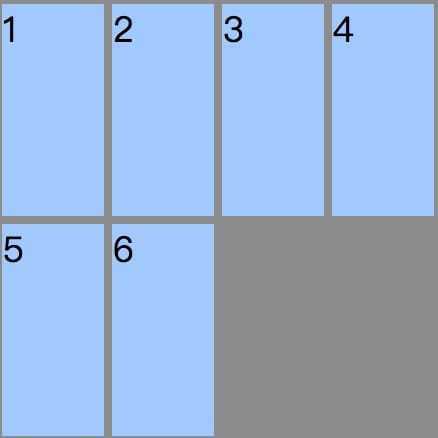
与align-items一样，align-content:默认值也是stretch。两者同时都为stretch时，毫无悬念所有元素都是撑满交叉轴。

#container {

align-items: stretch;

align-content: stretch;

}



当我们将align-items改为flex-start或者给弹性元素设置一个具体高度，此时效果是行与行之间形成了间距。

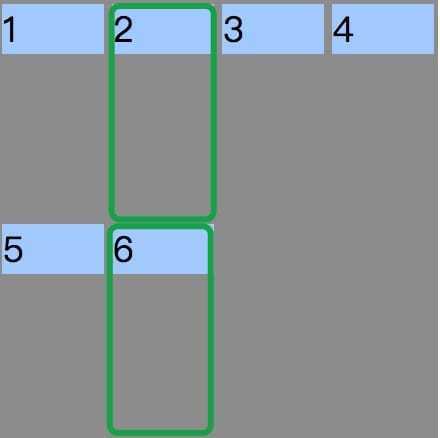
#container {

align-items: flex-start;

align-content: stretch;

}

align-content会以整行为单位，此时会将整行进行拉伸占满交叉轴；



align-content设置为顶对齐，此时以行为单位，整体高度通过内容撑开。

align-items仅仅管一行，在第一个元素设置了高度的情况下，第一行的其他元素遵循align-items: stretch也被拉伸到了50px。而第二行则保持高度不变。

#container {

align-items: stretch;

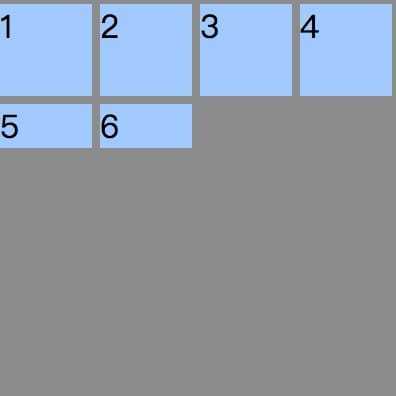
align-content: flex-start;

}

#container > div:first-child {

height: 50px;

}



两者差异总结：

* align-content管全局(所有行视为整体)
* align-items管单行

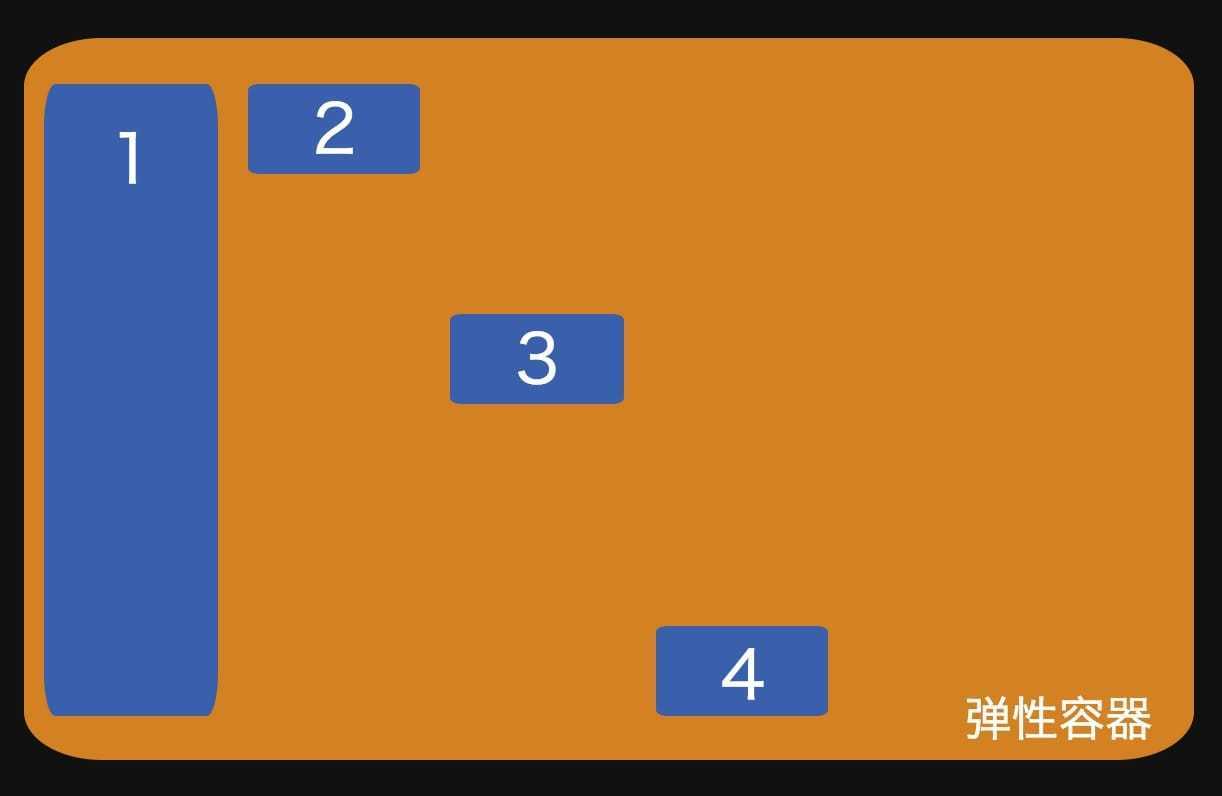
#### **能否更灵活地设置交叉轴对齐**

除了在容器上设置交叉轴对齐，还可以通过align-self单独对某个元素设置交叉轴对齐方式。

值与align-items相同

可覆盖容器的align-items属性

默认值为auto，表示继承父元素的align-items属性



## **六、其他**

### **order：更优雅地调整元素顺序**



#container > div:first-child {

order: 2;

}

#container > div:nth-child(2) {

order: 4;

}

#container > div:nth-child(3) {

order: 1;

}

#container > div:nth-child(4) {

order: 3;

}

order：可设置元素之间的排列顺序

数值越小，越靠前，默认为0

值相同时，以dom中元素排列为准

## **七、总结**

