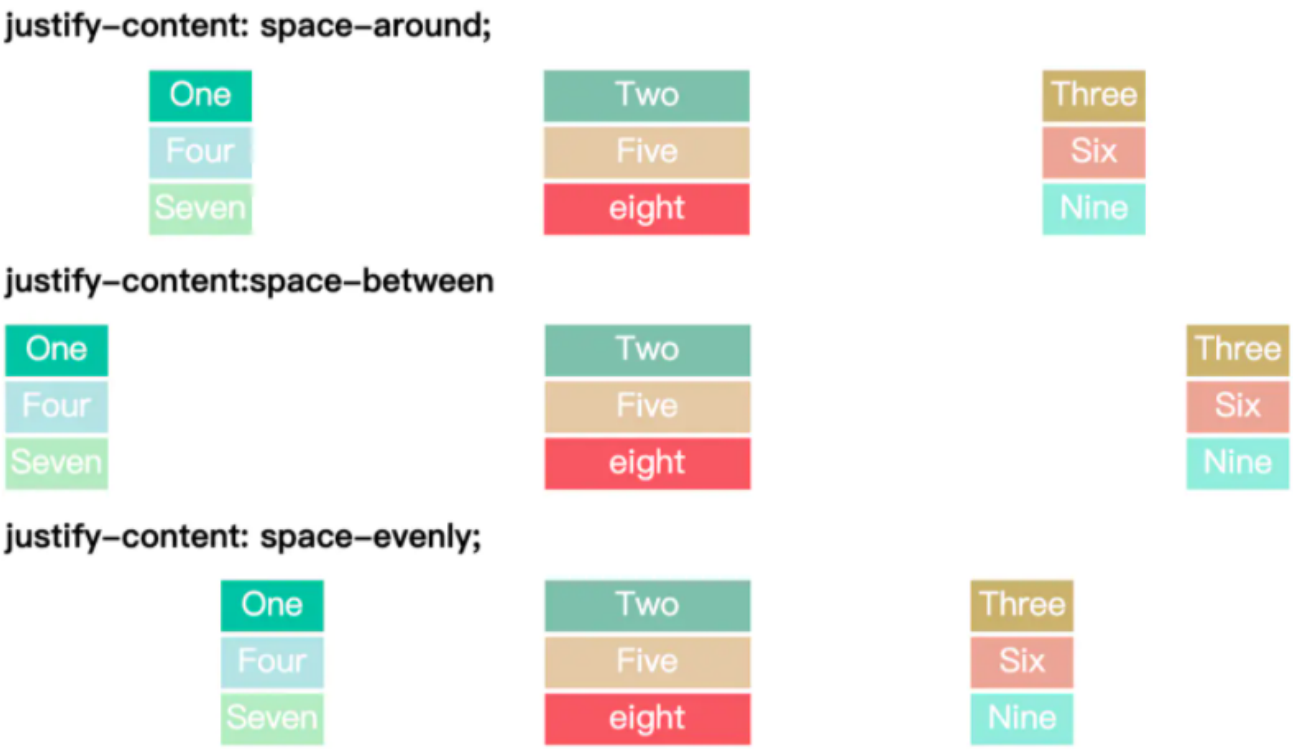
- space-around – 每个项目两侧的间隔相等。所以，项目之间的间隔比项目与容器边框的间隔大一倍
- space-between – 项目与项目的间隔相等，项目与容器边框之间没有间隔
- space-evenly – 项目与项目的间隔相等，项目与容器边框之间也是同样长度的间隔
- stretch – 项目大小没有指定时，拉伸占据整个网格容器



13.2.8. grid-auto-columns 属性和 grid-auto-rows 属性

有时候，一些项目的指定位置，在现有网格的外部，就会产生显示网格和隐式网格

比如网格只有3列，但是某一个项目指定在第5行。这时，浏览器会自动生成多余的网格，以便放置项目。超出的部分就是隐式网格

而 `grid-auto-rows` 与 `grid-auto-columns` 就是专门用于指定隐式网格的宽高

关于项目属性，有如下：

13.2.9. grid-column-start 属性、grid-column-end 属性、grid-row-start 属性以及grid-row-end 属性

指定网格项目所在的四个边框，分别定位在哪根网格线，从而指定项目的位置

- grid-column-start 属性：左边框所在的垂直网格线
- grid-column-end 属性：右边框所在的垂直网格线
- grid-row-start 属性：上边框所在的水平网格线
- grid-row-end 属性：下边框所在的水平网格线

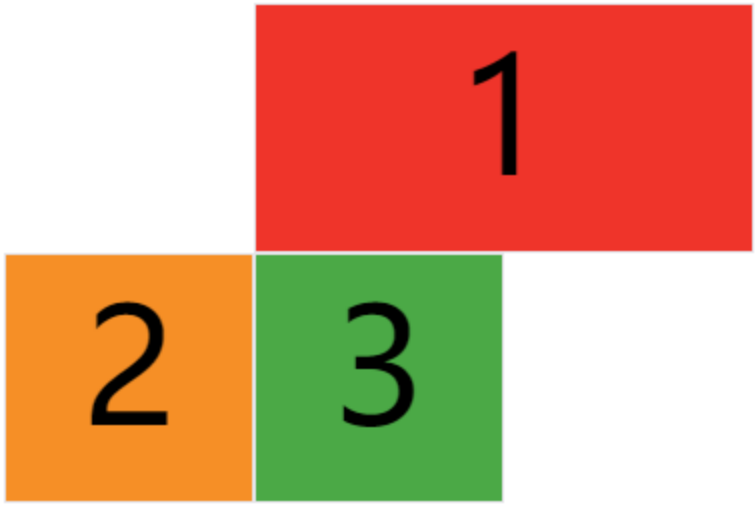
举个例子：

HTML | 复制代码

```html
<style>
    #container{
        display: grid;
        grid-template-columns: 100px 100px 100px;
        grid-template-rows: 100px 100px 100px;
    }
    .item-1 {
        grid-column-start: 2;
        grid-column-end: 4;
    }
</style>

<div id="container">
    <div class="item item-1">1</div>
    <div class="item item-2">2</div>
    <div class="item item-3">3</div>
</div>
```

通过设置 `grid-column` 属性，指定1号项目的左边框是第二根垂直网格线，右边框是第四根垂直网格线



13.2.10. grid-area 属性

`grid-area` 属性指定项目放在哪一个区域

```css
.item-1 {
  grid-area: e;
}
```

意思为将1号项目位于 `e` 区域

与上述讲到的 `grid-template-areas` 搭配使用

13.2.11. justify-self 属性、align-self 属性以及 place-self 属性

`justify-self` 属性设置单元格内容的水平位置（左中右），跟 `justify-items` 属性的用法完全一致，但只作用于单个项目。

`align-self` 属性设置单元格内容的垂直位置（上中下），跟 `align-items` 属性的用法完全一致，也是只作用于单个项目

```css
.item {
  justify-self: start | end | center | stretch;
  align-self: start | end | center | stretch;
}
```

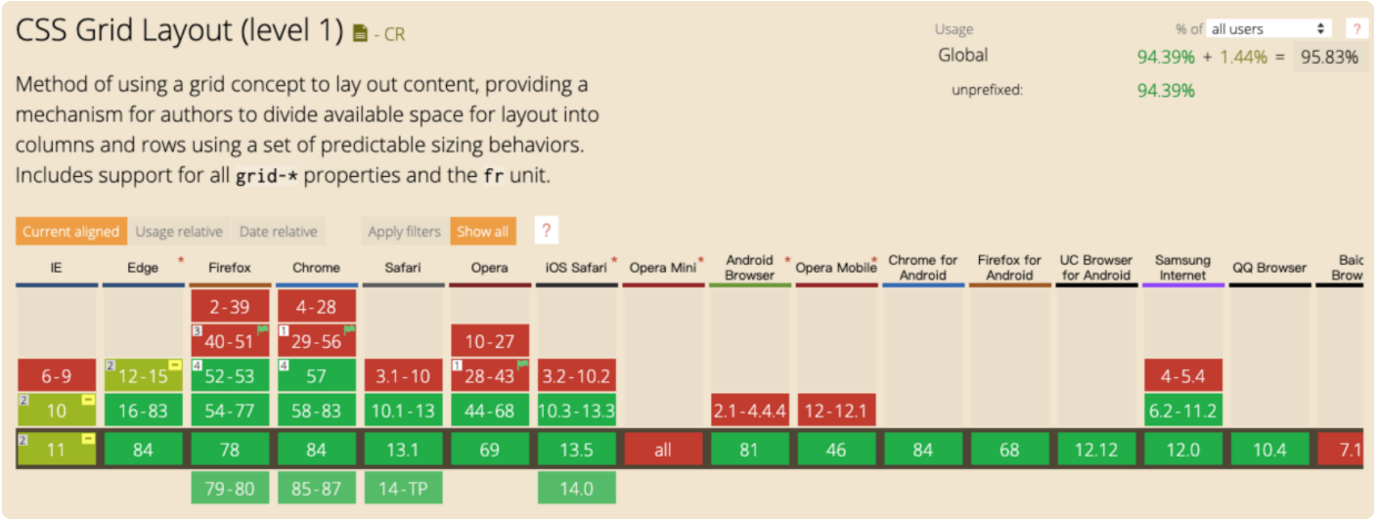
这两个属性都可以取下面四个值。

- start：对齐单元格的起始边缘。
- end：对齐单元格的结束边缘。
- center：单元格内部居中。
- stretch：拉伸，占满单元格的整个宽度（默认值）

13.3. 应用场景

文章开头就讲到，`Grid` 是一个强大的布局，如一些常见的 CSS 布局，如居中，两列布局，三列布局等等是很容易实现的，在以前的文章中，也有使用 `Grid` 布局完成对应的功能

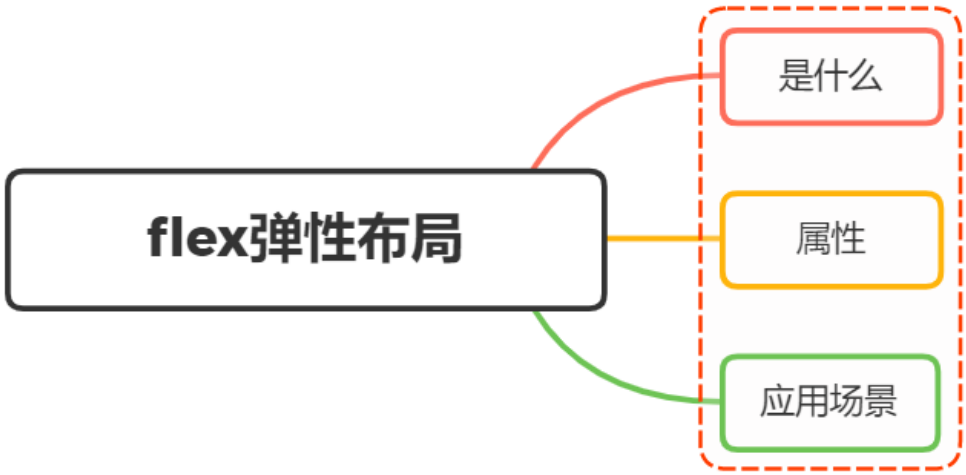
关于兼容性问题，结果如下：



总体兼容性还不错，但在 IE 10 以下不支持

目前，`Grid` 布局在手机端支持还不算太友好

14. 说说flexbox（弹性盒布局模型）,以及适用场景?

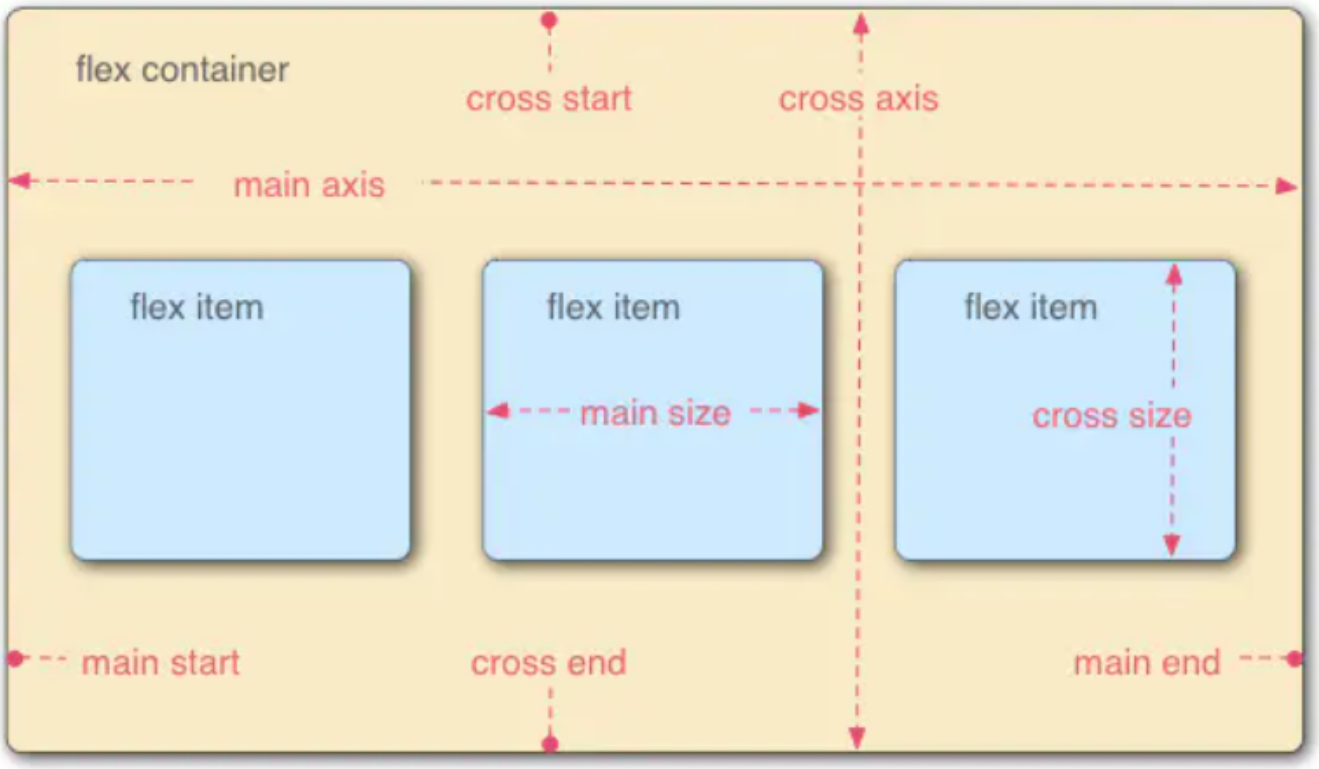


14.1. 是什么

`Flexible Box` 简称 `flex`，意为”弹性布局”，可以简便、完整、响应式地实现各种页面布局

采用Flex布局的元素，称为 `flex` 容器 `container`

它的所有子元素自动成为容器成员，称为 `flex` 项目 `item`



容器中默认存在两条轴，主轴和交叉轴，呈90度关系。项目默认沿主轴排列，通过 `flex-direction` 来决定主轴的方向

每根轴都有起点和终点，这对于元素的对齐非常重要

14.2. 属性

关于 `flex` 常用的属性，我们可以划分为容器属性和容器成员属性

容器属性有：

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

14.2.1. flex-direction

决定主轴的方向(即项目的排列方向)

```css
.container {
    flex-direction: row | row-reverse | column | column-reverse;
}
```

属性对应如下：

- row（默认值）：主轴为水平方向，起点在左端
- row-reverse：主轴为水平方向，起点在右端
- column：主轴为垂直方向，起点在上沿。
- column-reverse：主轴为垂直方向，起点在下沿

如下图所示：

14.2.2. flex-wrap

弹性元素永远沿主轴排列，那么如果主轴排不下，通过 `flex-wrap` 决定容器内项目是否可换行

```css
.container {
    flex-wrap: nowrap | wrap | wrap-reverse;
}
```

属性对应如下：

- nowrap（默认值）：不换行
- wrap：换行，第一行在下方
- wrap-reverse：换行，第一行在上方

默认情况是不换行，但这里也不会任由元素直接溢出容器，会涉及到元素的弹性伸缩

14.2.3. flex-flow

是 `flex-direction` 属性和 `flex-wrap` 属性的简写形式，默认值为 `row nowrap`

```css
.box {
  flex-flow: <flex-direction> || <flex-wrap>;
}
```

14.2.4. justify-content

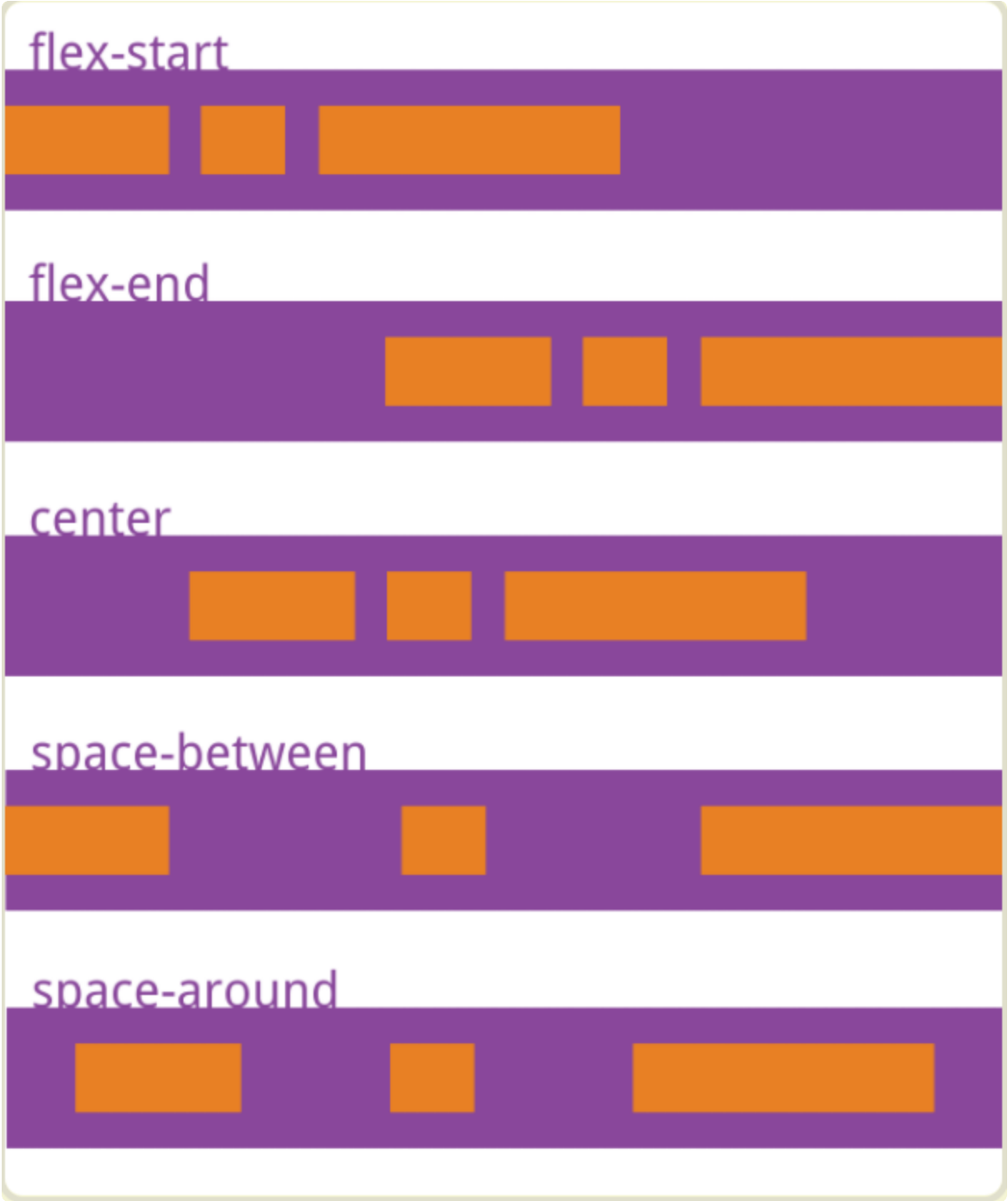
定义了项目在主轴上的对齐方式

```css
.box {
    justify-content: flex-start | flex-end | center | space-between | space-around;
}
```

属性对应如下：

- flex-start（默认值）：左对齐
- flex-end：右对齐
- center：居中
- space-between：两端对齐，项目之间的间隔都相等
- space-around：两个项目两侧间隔相等

效果图如下：



14.2.5. align-items

定义项目在交叉轴上如何对齐

CSS | 复制代码

```css
.box {
  align-items: flex-start | flex-end | center | baseline | stretch;
}
```

属性对应如下：

- flex-start：交叉轴的起点对齐
- flex-end：交叉轴的终点对齐
- center：交叉轴的中点对齐
- baseline: 项目的第一行文字的基线对齐
- stretch（默认值）：如果项目未设置高度或设为auto，将占满整个容器的高度

14.2.6. align-content

定义了多根轴线的对齐方式。如果项目只有一根轴线，该属性不起作用

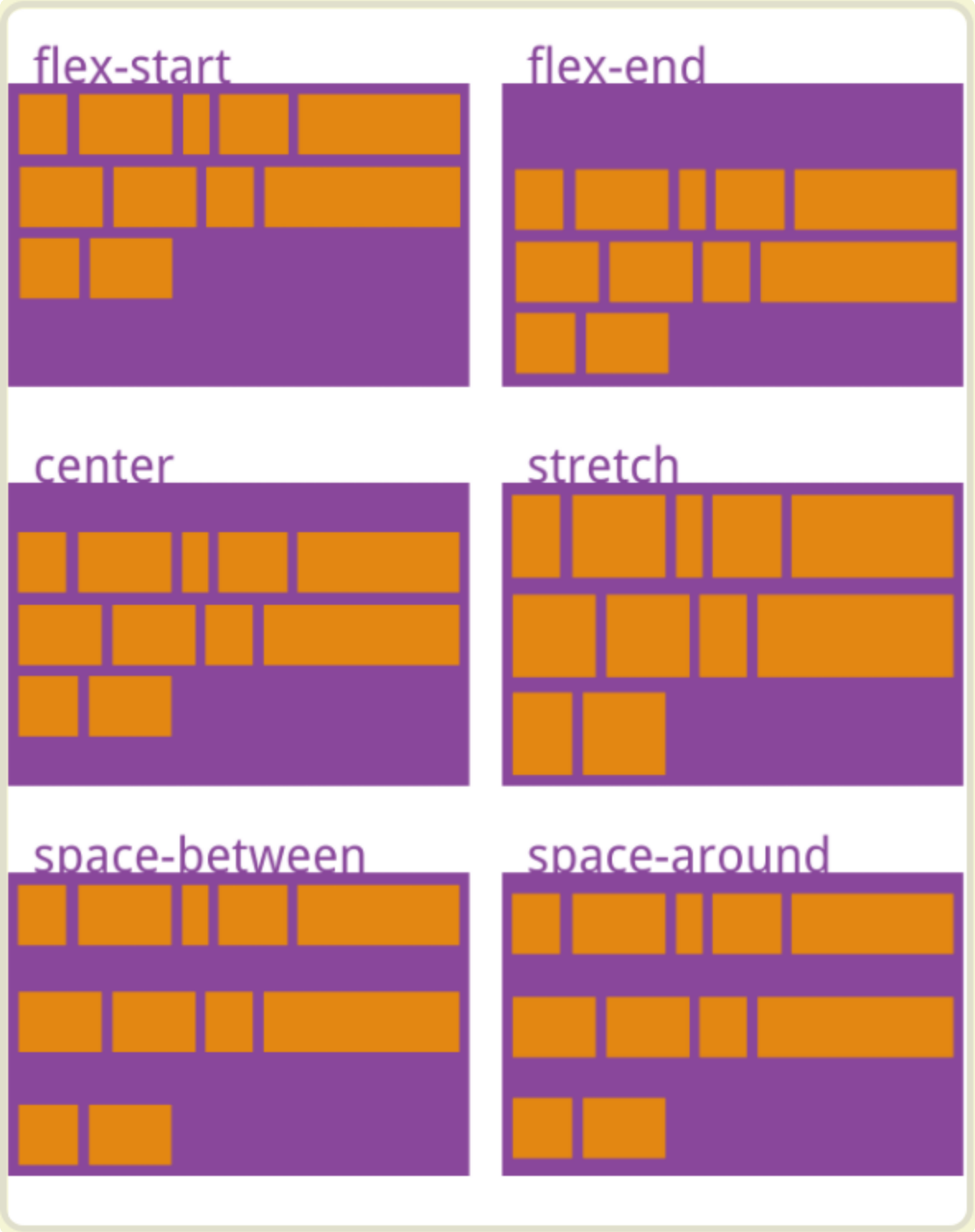
CSS | 复制代码

```css
.box {
    align-content: flex-start | flex-end | center | space-between | space-around | stretch;
}
```

属性对应如下：

- flex-start：与交叉轴的起点对齐
- flex-end：与交叉轴的终点对齐
- center：与交叉轴的中点对齐
- space-between：与交叉轴两端对齐，轴线之间的间隔平均分布
- space-around：每根轴线两侧的间隔都相等。所以，轴线之间的间隔比轴线与边框的间隔大一倍
- stretch（默认值）：轴线占满整个交叉轴

效果图如下：



容器成员属性如下：

- `order`
- `flex-grow`
- `flex-shrink`
- `flex-basis`
- `flex`

- `align-self`

14.2.7. order

定义项目的排列顺序。数值越小，排列越靠前，默认为0

```css
.item {
    order: <integer>;
}
```

14.2.8. flex-grow

上面讲到当容器设为 `flex-wrap: nowrap;` 不换行的时候，容器宽度有不够分的情况，弹性元素会根据 `flex-grow` 来决定

定义项目的放大比例（容器宽度>元素总宽度时如何伸展）

默认为 `0`，即如果存在剩余空间，也不放大

```css
.item {
    flex-grow: <number>;
}
```

如果所有项目的 `flex-grow` 属性都为1，则它们将等分剩余空间（如果有的话）

如果一个项目的 `flex-grow` 属性为2，其他项目都为1，则前者占据的剩余空间将比其他项多一倍

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

14.2.9. flex-shrink

定义了项目的缩小比例（容器宽度<元素总宽度时如何收缩），默认为1，即如果空间不足，该项目将缩小

CSS | 复制代码

```css
.item {
    flex-shrink: <number>; /* default 1 */
}
```

如果所有项目的 `flex-shrink` 属性都为1，当空间不足时，都将等比例缩小

如果一个项目的 `flex-shrink` 属性为0，其他项目都为1，则空间不足时，前者不缩小

在容器宽度有剩余时， `flex-shrink` 也是不会生效的

14.2.10. 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

浏览器根据这个属性，计算主轴是否有多余空间，默认值为 `auto` ，即项目的本来大小，如设置了 `width` 则元素尺寸由 `width/height` 决定（主轴方向），没有设置则由内容决定

```css
.item {
    flex-basis: <length> | auto; /* default auto */
}
```

当设置为0的是，会根据内容撑开

它可以设为跟 `width` 或 `height` 属性一样的值（比如350px），则项目将占据固定空间

14.2.11. flex

`flex` 属性是 `flex-grow`，`flex-shrink` 和 `flex-basis` 的简写，默认值为 `0 1 auto`，也是比较难懂的一个复合属性

```css
.item {
  flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]
}
```

一些属性有：

- flex: 1 = flex: 1 1 0%
- flex: 2 = flex: 2 1 0%
- flex: auto = flex: 1 1 auto
- flex: none = flex: 0 0 auto，常用于固定尺寸不伸缩

`flex:1` 和 `flex:auto` 的区别，可以归结于 `flex-basis:0` 和 `flex-basis:auto` 的区别

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

当设置为 `auto` 时（相对弹性元素），此时则需要在伸缩时将元素尺寸纳入考虑

注意：建议优先使用这个属性，而不是单独写三个分离的属性，因为浏览器会推算相关值

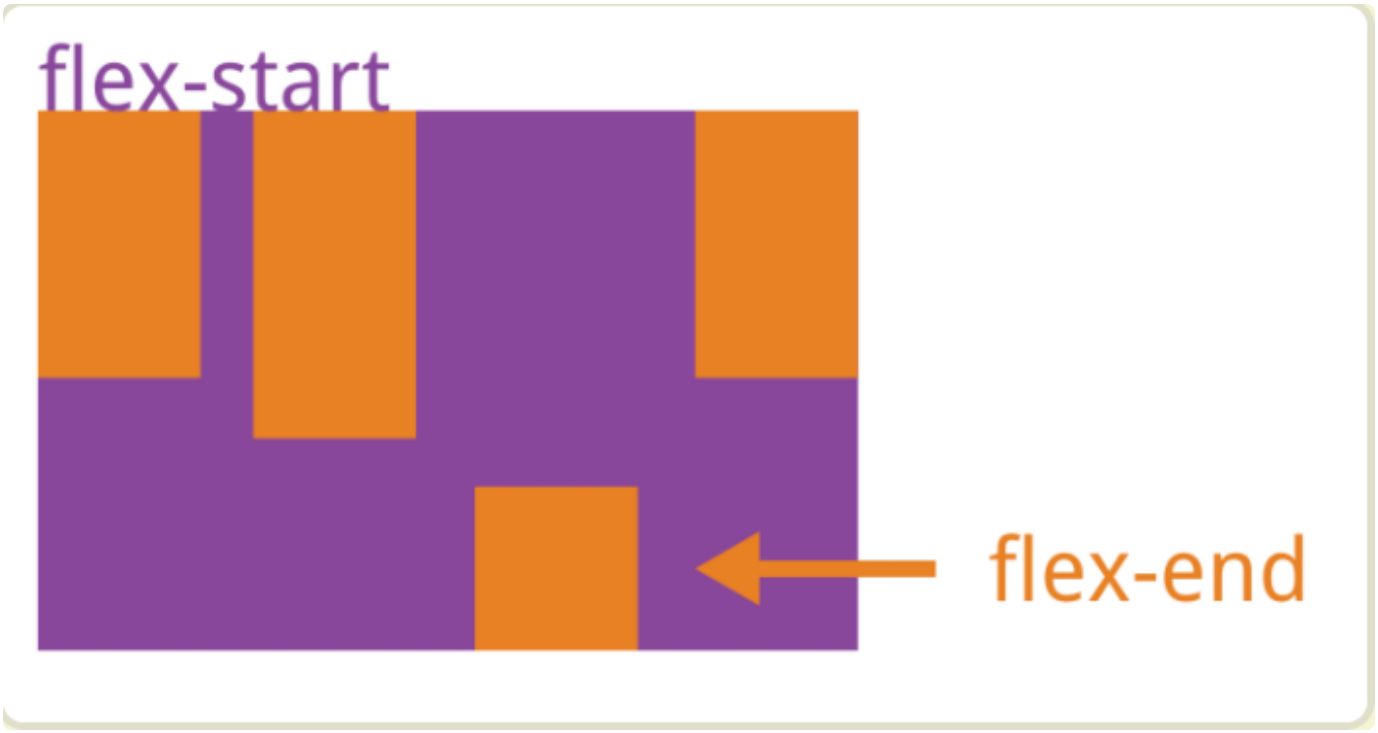
14.2.12. align-self

允许单个项目有与其他项目不一样的对齐方式，可覆盖 `align-items` 属性

默认值为 `auto`，表示继承父元素的 `align-items` 属性，如果没有父元素，则等同于 `stretch`

```css
.item {
    align-self: auto | flex-start | flex-end | center | baseline | stretch;
}
```

效果图如下：



14.3. 应用场景

在以前的文章中，我们能够通过 `flex` 简单粗暴的实现元素水平垂直方向的居中，以及在两栏三栏自适应布局中通过 `flex` 完成，这里就不再展开代码的演示

包括现在在移动端、小程序这边的开发，都建议使用 `flex` 进行布局

15. 说说设备像素、css像素、设备独立像素、dpr、ppi 之间的区别?

15.1. 背景

在 `css` 中我们通常使用px作为单位，在PC浏览器中 `css` 的1个像素都是对应着电脑屏幕的1个物理像素

这会造成一种错觉，我们会认为 `css` 中的像素就是设备的物理像素

但实际情况却并非如此， `css` 中的像素只是一个抽象的单位，在不同的设备或不同的环境中， `css` 中的1px所代表的设备物理像素是不同的

当我们做移动端开发时，同为1px的设置，在不同分辨率的移动设备上显示效果却有很大差异

这背后就涉及了css像素、设备像素、设备独立像素、dpr、ppi的概念

15.2. 介绍

15.2.1. CSS像素

CSS像素（css pixel, px）：适用于web编程，在 CSS 中以 px 为后缀，是一个长度单位

在 CSS 规范中，长度单位可以分为两类，绝对单位以及相对单位

px是一个相对单位，相对的是设备像素（device pixel）

一般情况，页面缩放比为1，1个CSS像素等于1个设备独立像素

`CSS` 像素又具有两个方面的相对性：

- 在同一个设备上，每1个 CSS 像素所代表的设备像素是可以变化的（比如调整屏幕的分辨率）
- 在不同的设备之间，每1个 CSS 像素所代表的设备像素是可以变化的（比如两个不同型号的手机）

在页面进行缩放操作也会 引起 `css` 中 `px` 的变化，假设页面放大一倍，原来的 1px 的东西变成 2px，在实际宽度不变的情况下1px 变得跟原来的 2px 的长度（长宽）一样了（元素会占据更多的设备像素）

假设原来需要 320px 才能填满的宽度现在只需要 160px

px会受到下面的因素的影响而变化：

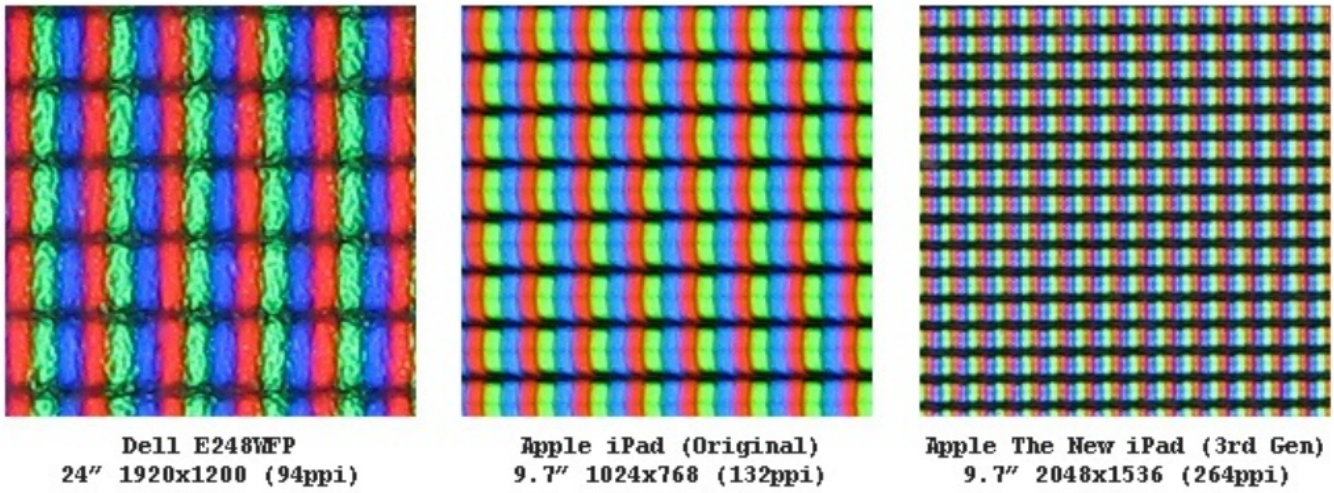
- 每英寸像素（PPI）
- 设备像素比（DPR）

15.2.2. 设备像素

设备像素（device pixels），又称为物理像素

指设备能控制显示的最小物理单位，不一定是一个小正方形区块，也没有标准的宽高，只是用于显示丰富色彩的一个“点”而已

可以参考公园里的景观变色彩灯，一个彩灯(物理像素)由红、蓝、绿小灯组成，三盏小灯不同的亮度混合出各种色彩



从屏幕在工厂生产出的那天起，它上面设备像素点就固定不变了，单位为 `pt`

15.2.3. 设备独立像素

设备独立像素（Device Independent Pixel）：与设备无关的逻辑像素，代表可以通过程序控制使用的虚拟像素，是一个总体概念，包括了CSS像素

在 `javaScript` 中可以通过 `window.screen.width/ window.screen.height` 查看

比如我们会说“电脑屏幕在 2560x1600分辨率下不适合玩游戏，我们把它调为 1440x900”，这里的“分辨率”（非严谨说法）指的就是设备独立像素

一个设备独立像素里可能包含1个或者多个物理像素点，包含的越多则屏幕看起来越清晰

至于为什么出现设备独立像素这种虚拟像素单位概念，下面举个例子：

iPhone 3GS 和 iPhone 4/4s 的尺寸都是 3.5 寸，但 iPhone 3GS 的分辨率是 320x480，iPhone 4/4s 的分辨率是 640x960

这意味着，iPhone 3GS 有 320 个物理像素，iPhone 4/4s 有 640 个物理像素

如果我们按照真实的物理像素进行布局，比如说我们按照 320 物理像素进行布局，到了 640 物理像素的手机上就会有一半的空白，为了避免这种问题，就产生了虚拟像素单位

我们统一 iPhone 3GS 和 iPhone 4/4s 都是 320 个虚拟像素，只是在 iPhone 3GS 上，最终 1 个虚拟像素换算成 1 个物理像素，在 iphone 4s 中，1 个虚拟像素最终换算成 2 个物理像素

至于 1 个虚拟像素被换算成几个物理像素，这个数值我们称之为设备像素比，也就是下面介绍的 `dpr`

15.2.4. dpr

dpr（device pixel ratio），设备像素比，代表设备独立像素到设备像素的转换关系，
在 `JavaScript` 中可以通过 `window.devicePixelRatio` 获取

计算公式如下：

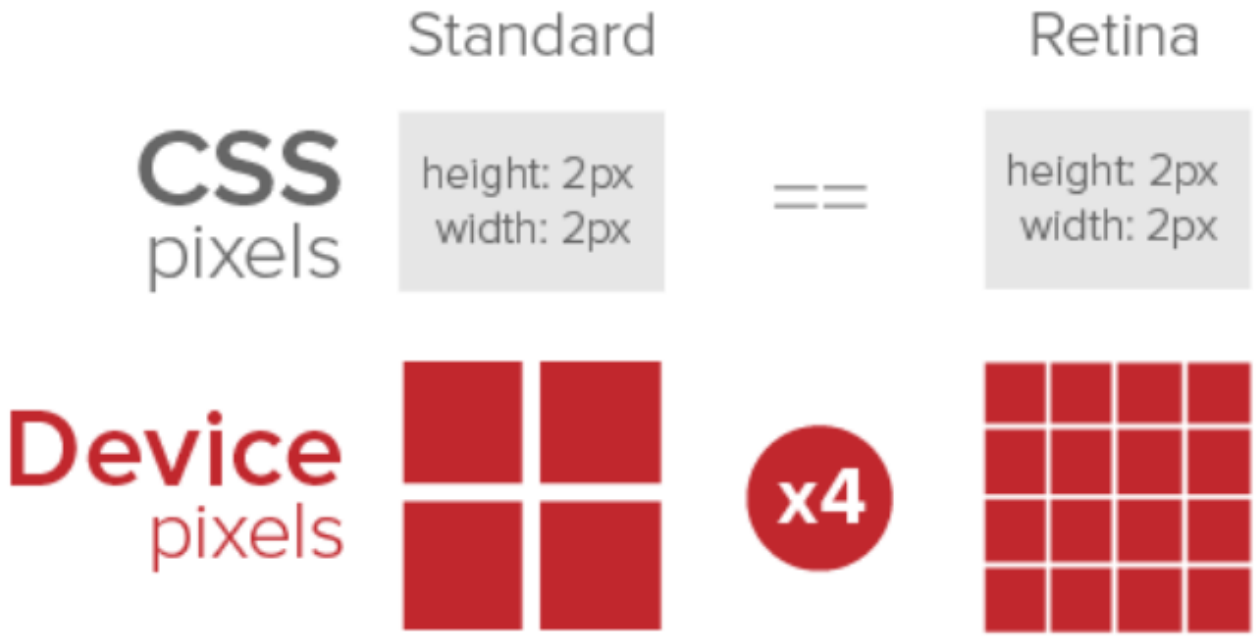
DPR = 设备像素 / 设备独立像素

当设备像素比为1:1时，使用1（1×1）个设备像素显示1个CSS像素

当设备像素比为2:1时，使用4（2×2）个设备像素显示1个CSS像素

当设备像素比为3:1时，使用9（3×3）个设备像素显示1个CSS像素

如下图所示：

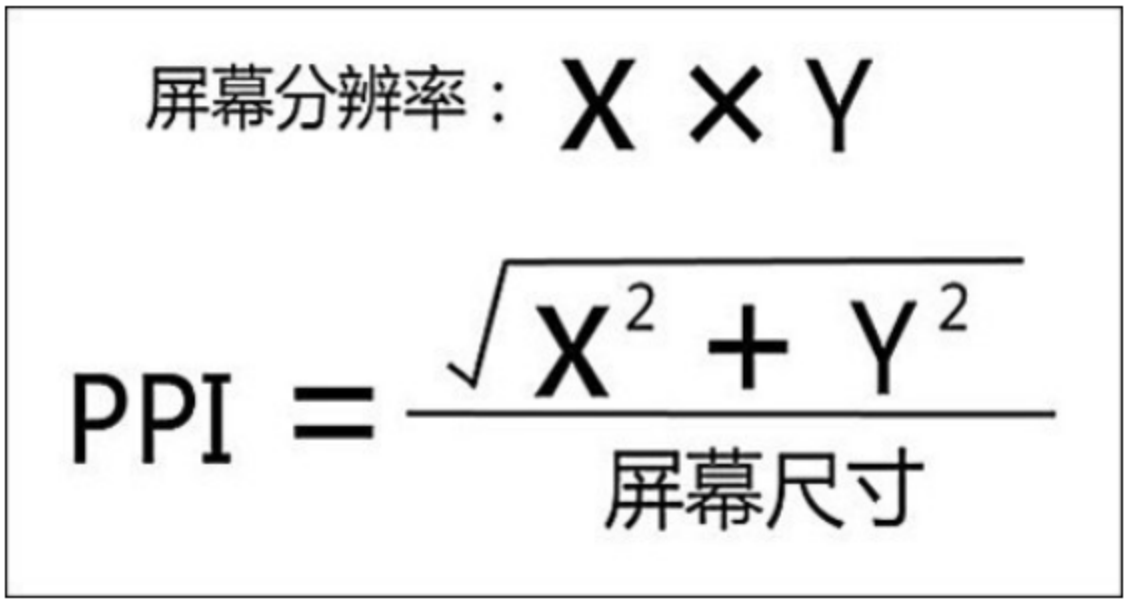


当 `dpr` 为3，那么 `1px` 的 `CSS` 像素宽度对应 `3px` 的物理像素的宽度，1px的 `CSS` 像素高度对应 `3px` 的物理像素高度

15.2.5. ppi

ppi （pixel per inch），每英寸像素，表示每英寸所包含的像素点数目，更确切的说法应该是像素密度。数值越高，说明屏幕能以更高密度显示图像

计算公式如下：



15.3. 总结

无缩放情况下，1个CSS像素等于1个设备独立像素

设备像素由屏幕生产之后就不发生改变，而设备独立像素是一个虚拟单位会发生改变

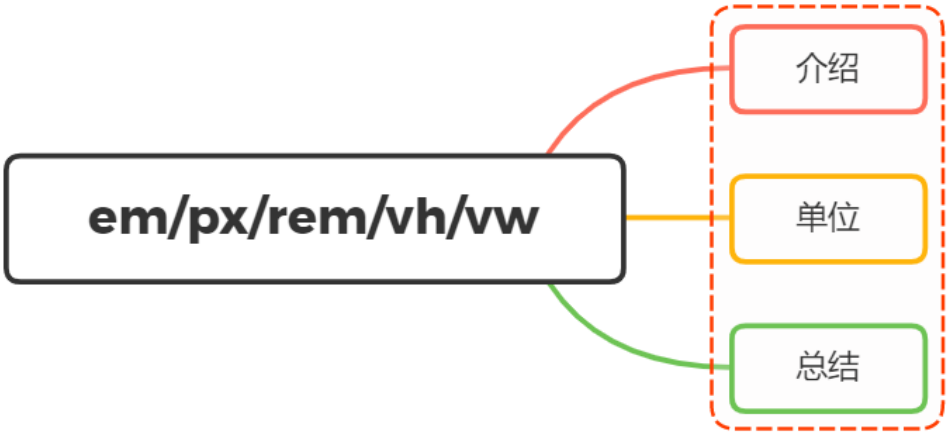
PC端中，1个设备独立像素 = 1个设备像素 （在100%，未缩放的情况下）

在移动端中，标准屏幕（160ppi）下 1个设备独立像素 = 1个设备像素

设备像素比（dpr） = 设备像素 / 设备独立像素

每英寸像素（ppi），值越大，图像越清晰

16. 说说em/px/rem/vh/vw区别?



16.1. 介绍

传统的项目开发中，我们只会用到 `px` 、 `%` 、 `em` 这几个单位，它可以适用于大部分的项目开发，且拥有比较良好的兼容性

从 `CSS3` 开始，浏览器对计量单位的支持又提升到了另外一个境界，新增了 `rem` 、 `vh` 、 `vw` 、 `vm` 等一些新的计量单位

利用这些新的单位开发出比较良好的响应式页面，适应多种不同分辨率的终端，包括移动设备等

16.2. 单位

在 `css` 单位中，可以分为长度单位、绝对单位，如下表所指示