1. flex弹性 盒模型

- 2009年提出，现在所有浏览器都支持
- `flex`属性

作用在flex容器上	作用在flex子项上
`flex-direction`	`order`
`flex-wrap`	`flex-grow`
`flex-flow`	`flex-shrink`
`justify-content`	`flex-basis`
`align-items`	`flex`
`align-content`	`align-self`

- **作用在父容器上的属性**
- `flex-direction`： 用来控制子项整体布局方向，是从左往右还是从右往左，是从上往下还是从下往上。
  - `row`：默认值，显示为行。方向为当前文档水平流方向，默认情况下是从左往右
  - `row-reverse`：显示为行。但方向和`row`属性值是反的
  - `column`：显示为列
  - `column-reverse`：显示为列。但方向和`column`属性值是反的。
- `flex-wrap`：用来控制子项整体单行显示还是换行显示。
  - `nowrap`：默认值，表示单行显示，不换行。
    - `nowrap`不会换行。如果子元素宽度大于容器，会压缩子元素，不让子元素溢出；如果压缩的只剩文字和内容，则会溢出。
  - `wrap`：宽度不足换行显示
  - `wrap-reverse`：子元素从最后一列开始排列，宽度不足向上换行。
- `flex-flow: direction wrap`
  - `flex-flow`属性是`flex-direction`和`flex-wrap`的缩写，表示`flex`布局的`flow`流动性。**第一个值表示方向，第二个值表示换行**，中间用空格隔开。

- `justify-content`：决定每行**主轴 方向上子项的对齐和分布方式。主轴上有多余空间起作用**
  - 主轴：根据`flex-direction`决定。

代码	对齐方式
`flex-start`	默认值，表现为起始位置对齐。
`flex-end`	表现为结束位置对齐。

代码	对齐方式
`center`	表现为居中对齐。
`space-between`	两端对齐，多余的空白间距在元素中间分配(`between`的意思是在...之间)
`space-around`	`around`是环绕的意思， 每个`flex`子项两侧都环绕互不相干扰的等宽空白间距 最终视觉上 边缘两侧的空白只有中间空白宽度的一半。
`space-evenly`	`evenly`是匀称、平等的意思。 视觉上每个`flex`子项两侧的空白间距完全相等。

- `align-items` ： 控制**副轴上子项在每一行的对齐方式,需要行内拥有多余空间**
  - `align-items`中的`items`指的就是`flex`子项们，因此`align-items`指的就是`flex`子项们相对于`flex`容器在**行内侧轴方向上的对齐方式。**

代码	对齐方式
`strtch`	默认值,`flex`子项拉伸,如果子项没有固定高度，则高度和此行高度相等。
`flex-start`	表现为容器顶部对齐，如果没有固定高，高度根据内容决定。
`flex-end`	表现为容器底部对齐,如果没有固定高，高度根据内容决定。
`center`	表现为 垂直居中对齐,如果没有固定高，高度根据内容决定。
`baseline`	基于项目的行内文字的基线对齐

- `align-content` 决定每行**副轴 方向上子项的对齐和分布方式。副轴上有多余空间起作用**
  - `align-content`可以看成和`justify-content`相似且对立的属性，如果所有`flex`子项只有一行，则`align-content`属性是没有任何效果的。（`align-content`是针对侧轴的）

代码	对齐方式
`stretch`	默认值。每一行子元素都拉伸，如果共有两行flex元素， 则每一行拉伸高度是50%。
`flex-start`	表现为起始位置对齐。
`flex-end`	表现为结束位置对齐。
`center`	表现为居中对齐。

代码	对齐方式
`space-between`	表现为两端对齐。
`space-around`	每一行上下都享有独立不重叠的空间。
`space-evnely`	每一个元素都完全上下等分。

- **作用在子项上的属性**
- `order`：可以通过设置`order`改变某一个`flex`子项的排序位置。所有`flex`子的项默认`order`属性值为`0`，可以小于`0`,**值越小排序越靠前。**

- `flex-grow`：属性中`grow`是扩展的意思，扩展的就是`flex`子项所占据的宽度，扩展所**侵占**的空间就是除去元素外的的**剩余空白间隙。**默认值为`0`,最小值为`0`，整个空白空间整合是`1`。

- `flex-shrink`：属性中的`shrink`是收缩的意思，`flex-shrink`主要处理**当`flex`容器空间不足的时候，单个元素的收缩比例**，默认值为`1`，`0`表示不收缩，最小值为`0`。
  若想不让元素在空间不足时收缩，把flex-shrink设置为0。

- `flex-basis`：`flex-basis`定义了在**分配剩余空间之前的默认元素大小。**
  - `flex-basis`优先级大于宽。设置为`auto`后，如果有固定宽，则宽度就是固定宽，如果没有固定宽，则根据内容决定宽度。

- `flex`：`flex`属性是`flex-grow`、`flex-shrink`、`flex-basis`的缩写。
  - 属性值仅有一个数字时，该值为`flex-grow`的值，默认`flex-shrink`为`1`，`flex-basis`为`0%`;

- `align-self`：**控制某一个子项行内垂直对齐方式。**
  - 和`align-items`一样，只不过`align-self`是作用在单一子项上
  - `align-self`：属性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。
- `flex`特性：
  1. 当内容宽度大于容器宽度时，内容不会换行，而是自己调节内容（多余）宽度来适应容器。
  2. 子容器高度默认和父容器相等。
  3. 子容器如果不设置宽度，则宽度根据内容分配。

2.Grid网格布局

- `Grid`网格布局是一个二维的布局方法，纵横两个方向总是同时存在。

作用在grid容器上	作用在grid子项上
`grid-template-columns`	`grid-column-start`
`grid-template-rows`	`grid-column-end`
`grid-template-areas`	`grid-row-start`
`grid-template`	`grid-row-end`
`grid-column-gap`	`grid-column`
`grid-row-gap`	`grid-row`
`grid-gap`	`grid-area`
`justify-items`	`justify-self`
`align-items`	`align-self`
`place-items`	`place-self`
`justify-content`	
`align-content`	
`place-content`	

- **作用在`grid`容器上**
- `grid-template-columns`和`grid-template-rows`
  - 对网格进行纵横划分，形成二维布局。单位可以是像素,百分比,自适应以及`fr`单位(网格空间比例单位)。
    - `fr`单位，最小值为`0`，宽和高默认值为`1`，可以设置大于`1`，情况和`flex-grow`分配剩余空间一样。

```
grid-template-columns:1fr 2fr 1fr;
/* 表示容器列数为3，容器的宽分为4fr（4份） 中间的列占两份 */

grid-template-rows:.1fr .1fr .2fr;
/* 表示分为3行，总行高占容器高的40%(.4fr)。*/
```

  - 有时候，我们网格划分是很规律的，如果要添加多个纵横网格时，可以利用`repeat()`语法进行简化操作。

```
grid-template-rows:repeat( 3 , 1fr )
/* grid-template-rows:repeat( 行数 , 每一行的大小，可以是百分比，像素 ) */
```

- `grid-template-areas`和`grid-template`
  - `area`是区域的意思，`grid-template-areas`就是给网格划分区域的。此时`grid`子项只要使用`grid-area`属性指定其隶属于哪个区域。
    - 用法：在父容器进行划分区域，子容器指定区域名。(子元素添加:`grid-area:区域名;`)
    - `grid-template-areas`**不允许形成特殊图形，只能形成矩形。**
    - `grid-template-areas`的**命名不允许数字开头。**

```
.grid{
  display: grid;
  width: 300px;
  height: 300px;
  grid-template-rows: repeat(3,1fr);
  grid-template-columns: repeat(3,1fr);
  grid-template-areas:
  "name1 name2 name3"
  "name4 name4 name5"
  "name6 name7 name7";
}
.grid>div:nth-child(1){
  grid-area: name1;  /* 使用时不用加"" */
  background: red;
}
```

  - `grid-template`是`grid-template-rows`,`grid-template-columns`,`grid-template-areas`属性的缩写。

```
grid-template:
"name1 name2 name3" 1fr
"name4 name4 name5" 1fr
"name6 name7 name7" 2fr
/1fr     1fr    1fr;
```

  - `grid-template-columns`和`grid-template-rows`、`grid-template-areas`、`grid-template`只是指定网格区域，`grid-area: name1;`指定当前元素入驻哪个网格，**在此网格内的对齐方式由`justify-items`和`align-items`指定。**

- `grid-column-gap`（纵向）和`grid-row-gap`（横向）
  - 用来定义网格中网格间隙的尺寸。(注意，间隙在设置对齐方式时，不会被覆盖)
    - `grid-gap`属性是`grid-column-gap`和`grid-row-gap`的缩写。
      - `grid-gap`:横向 纵向;

- `justify-items`和`align-items`
  - `justify-items`指定了**网格内元素的水平呈现方式**，是水平拉伸显示，还是左中右对齐。
  - `align-items`指定了**网格内元素的垂直呈现方式**，是垂直拉伸，还是上中下对齐。

代码	对齐方式
`strtch`	默认值,`grid`子项拉伸,如果子项没有固定高度，则高度和网格相等。
`start`	表现为容器顶部对齐，并且高度根据内容决定。
`end`	表现为容器底部对齐,并且高度根据内容决定。
`center`	表现为 垂直居中对齐,并且高度根据内容决定。
`baseline`	基于项目的第一行文字的基线对齐

  - `place-items`可以让`align-items`和`justify-items`属性写在单个声明中。

```
place-items: align-items  justify-items; /* 先纵向，在横向 */
```

- `justify-content`和`align-content`
  - `justify-content`指定了网格整体的水平分布方式。需要网格外部有空余
  - `align-content`指定了网格的垂直分布方式。需要网格外部有空余

代码	对齐方式
`start`	表现为起始位置对齐。(默认值)
`end`	表现为结束位置对齐。
`center`	表现为居中对齐。
`space-between`	表现为两端对齐。
`space-around`	每一行上下都享有独立不重叠的空间。
`space-evnely`	每一个元素都完全上下等分。

  - `place-content`可以让`align-content`和`justify-content`属性写在单个声明中。

```
place-content: align-content  justify-content;/* 先纵向，在横向 */
```

  - `items`是针对网格中的内容排列，`content`是针对网格的排列

- **作用在子项上的属性**
- `grid-column-strat`：水平方向上占据的起始位置
  - 这里的起始位置是网格的 纵向线 起始线数值为`1`
- `grid-column-end`：水平方向上占据的结束位置（`span`属性)
- `grid-row-start`：垂直方向上占据的起始位置
  - 这里的起始位置是网格的 横向线 起始线数值为`1`
- `grid-row-end`：垂直方向上占据的结束位置（`span`属性)

- 注：`span`属性只在`end`结束位置中拥有

```
grid-column-start:3;  /* 从第三条网格纵线开始 */
grid-column-end: span 2; /* 向后延伸两格距离 */
```

- `grid-column`：`grid-column-start`和`grid-column-end`的缩写
  - `grid-column:start / end (或span属性);`这里的`/`是要写在值中间的
- `grid-row`：`grid-row-start`和`grid-row-end`的缩写

- `grid-area`：表示当前网格所占用的区域，名字和位置两种表示方法。
  - 名字：`grid-area:a1;`
  - 位置：`grid-area: 2 / 3 / 4(或span属性) / 4(或span属性)`
    - `grid-area: y起始 / x起始 / y结束 / x结束`

- `justify-self`：单个网格元素的水平对齐方式。
- `align-self`：单个网格元素的垂直对齐方式。
- `place-self`：`align-self`和`justify-self`的缩写。
  - `place`目前都是先纵向`y`，在横向`x`
- **`content`控制的是网格的位置，`items`控制的是网格内元素的对齐位置，`self`是设置在子项上单独控制这个子项内元素的对其位置**

3.移动端模拟器

- 切换平台之后一定要重新刷新网页。

4.PC和移动端的网页。

- 大一点的网站都是分开开发的，`PC`端一套代码，移动端一套代码。
- https://www.taobao.com -> 后端检测当前设备，`pc`端访问的和手机端访问的网址不同（重定向）。

5. viewport 视口(在移动端才会有)

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

- 在移动端`viewport`视口就是浏览器显示页面内容的屏幕区域。在`viewport`中有两种视口，分别表示为，`visual viewport`(可视窗口)和`layout viewport`(布局视口)。
- `visual viewport`固定大小跟屏幕大小相同，在上面;而`layout viewport`可改变大小，在下面。`layout viewport`默认大小为`980`像素，可通过`document.documentElement.clientWidth`获取。
- 现代网页需要将`layout viewport`设置成跟`visual viewport`同等大小，方便网页制作。
- `viewport`设置

---

content属性值	解释
width	设置layout viewport的宽度特定值，device-width表示设备宽。
height	设置layout viewport的高度固定值，一般不进行设置。
initial-scale	设置页面的初始缩放。（设置为1.0）
Minimum-scale	设置页面最小缩放。
maximum-scale	设置页面最大缩放。
user-scalable	设置页面能否缩放。(no表示不允许)

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<!-- 在 content属性内书写 -->
```

6.移动端适配方案

1. 百分比布局，也叫流式布局。代表网站 优酷、百度、天猫、腾讯。
   - 宽度使用%百分比定义，但是高度和文字大小等大都是用px来固定，所以在大屏幕的手机下显示效果会变成有些页面元素**宽度被拉的很长，但是高度、文字大小还是和原来一样**
   - 流式布局中，文字大小是不会发生变化的。
   - 流式布局中，图片会根据设备进行等比缩放。
   - 好处：大屏幕下显示更多内容。
   - 坏处： 宽屏下比例会有些不协调。

2. 等比缩放布局，也叫rem布局。代表网站 网易、爱奇艺、淘宝、美团。
   - em：是一个相对单位，1em等于当前元素或父元素font-size的值。
   - rem：是一个相对单位，1rem等于根元素(html)的font-size的值。
   - vw / wh：把屏幕分为100份，1vw等于1%屏幕的宽，1vh等于1%屏幕的高。(具体看下方单位)
   - 通过动态设置html的font-size和属性的rem可以实现动态布局
   - 动态设置font-size :通过JS或通过vw
   - 要给body重置一下font-size : 16px;

3. 响应式布局 (利用媒介查询和一套网页适配不同的设备，适合中小型的项目)
   - 布局特点：每个屏幕分辨率下面会有一个布局样式，即元素位置和大小都会变。
   - 利用媒体查询，即media queries，可以针对不同的媒体类型定义不同的样式，从而实现响应式布局。
   - @media可以用如下几种方式定义:@media not screen and (max-width: 500px){}

     ```
     <link rel="stylesheet" href="print.css" type="text/css" media="print" />
     ```

```
@import url("print.css") screen;
```

```
@media print { body { font-size: 12px; } }
```

```
<style media="print">body{font-size:12px;}</style>
```

- 媒体类型

名称	描述
`all`	用于所有设备
`print`	用于打印机和打印预览
`screen`	用于电脑屏幕，平板电脑，智能手机等。
`speech`	应用于屏幕阅读器等发声设备

- 运算符：`AND` 与，`NOT` 取反，`OR` 或。
- 常见选项：
  - `min-width`、`max-width`
    `orientation:portrait`(纵向)、`orientation:landscape`(横向)

```
@media screen and (min-width:960px) and (max-width:1200px){
    body{background:yellow;}
    /* 表示浏览器宽度大于等于960px且小于等于1200px时使用样式。*/
}
```

- 响应式代码写到要适配的`CSS`后面。

扩展：html中的单位

单位	描述
`px`	绝对单位，页面按精确像素展示
`em`	相对单位，`1em=父节点字体大小`
`rem`	相对单位，相对根节点`html`的字体大小来计算，`chrome`强制最小字体为12号
`vw`	*viewpoint width*，视窗宽度，`1vw`等于视窗宽度的`1%`。
`vh`	*viewpoint height*，视窗高度，`1vh`等于视窗高度的`1%`。
`vmin`	`vw`和`vh`中较小的那个。
`vmax`	`vw`和`vh`中较大的那个。

单位	描述
`%`	百分比
`in`	寸
`cm`	厘米
`mm`	毫米
`pt`	`point`，大约`1/72`寸
`pc`	`pica`，大约`6pt`， `1/6`寸

- 关于图片问题扩展

```
<picture>
  <source media="(max-width:800px)" srcset="./11.jpg">
  <img src="./1.jpg" alt="">
</picture>
```

  - 当浏览器宽度小于等于`800px`,图片显示会变换。