第 3 章　页面布局

学习目标：

- 了解盒子模型的基本原理
- 掌握浮动与定位
- 熟练掌握 Flex 布局方式

3.1　盒子模型

在页面设计中，为了控制各个模块在页面中的位置，只要掌握了盒子模型以及盒子模型的各个属性和应用方法，就能轻松地控制页面中的各个元素。

所谓盒子模型，就是我们在页面设计中经常用到的一种思维模型，它和我们生活中看到的盒子相当，也就是一个用来盛装内容的容器。在 CSS 中，一个独立的盒子模型由内容（content）、内边距（padding）、边框（border）和外边距（margin）4 个部分组成，如图 3-1 所示。

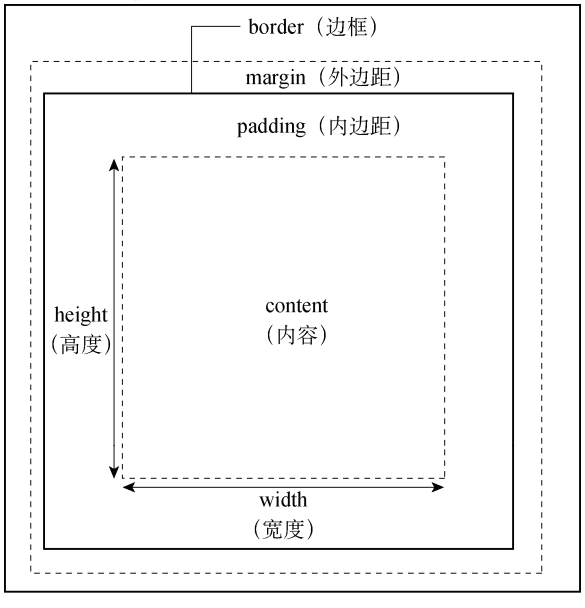


图 3-1　盒子模型结构

此外，对 padding、border 和 margin 可以进一步细化为上下左右 4 个部分，在 CSS 中可以分别进行设置，如图 3-2 所示。

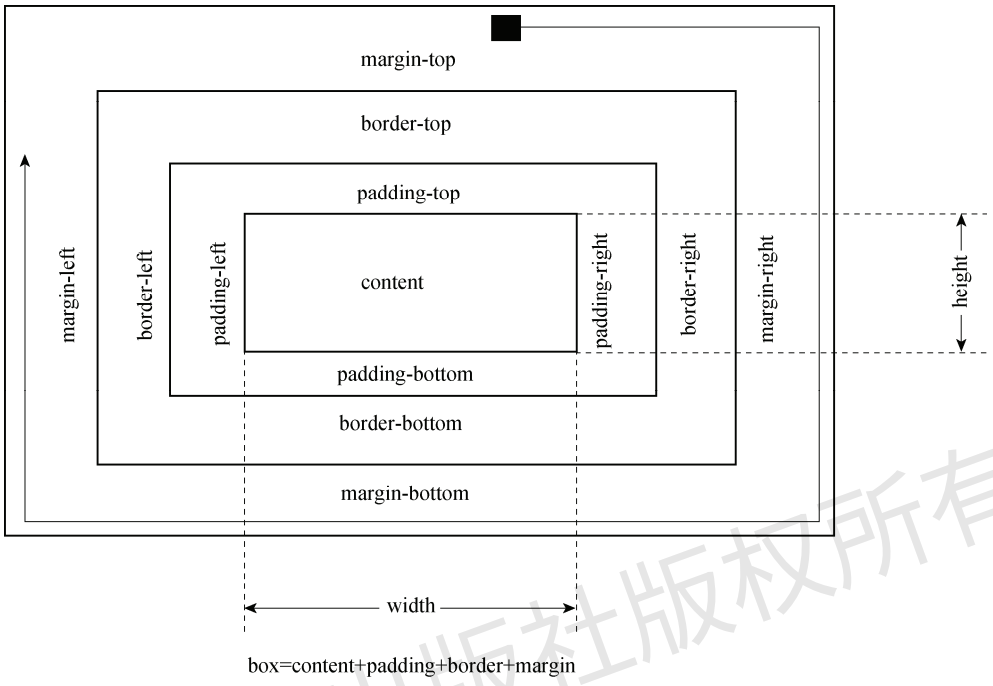


图 3-2　盒子模型元素

图中各属性的含义如下：

- width 和 height 表示内容的宽度和高度。
- padding-top、padding-right、padding-bottom 和 padding-left 分别表示上内边距、右内边距、底内边距和左内边距。
- border-top、border-right、border-bottom 和 border-left 分别表示上边框、右边框、底边框和左边框。
- margin-top、margin-right、margin-bottom 和 margin-left 分别表示上外边距、右外边距、底外边距和左外边距。

因此，一个盒子实际所占有的宽度（高度）应该由“内容”+“内边距”+“边框”+“外边距”组成。例如：

```
.box{
   width:70px;
   padding:5px;
   margin:10px;
}
```

此盒子所占的宽度如图 3-3 所示。

CSS 中的布局都是基于盒子模型的，不同类型元素对盒子模型的处理不同，比如块级元素和行内元素、浮动元素和定位元素的处理方式也不同。

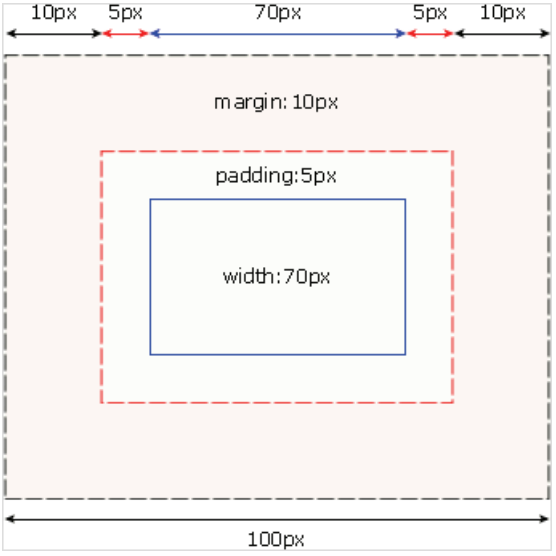


图 3-3　盒子所占的宽度

3.2　块级元素与行内元素

元素按显示方式分为块级元素、行内元素及行内块元素，它们的显示方式由 display 属性控制。

3.2.1　块级元素

块级元素默认占一行高度，一般一行内只有一个块级元素（浮动后除外），添加新的块级元素时，会自动换行，块级元素一般作为容器出现。块级元素的特点如下：

- 一个块级元素占一行。
- 块级元素的高度默认由内容决定，除非自定义高度。
- 块级元素的宽度默认是父级元素的内容区宽度，除非自定义宽度。
- 块级元素的宽度、高度、外边距及内边距都可自定义。
- 块级元素可容纳块级元素和行内元素。

<view/>组件默认为块级元素，使用<view/>组件演示盒子模型及块级元素的例子如下：

```
<!--每个块级元素占一行-->
<view style="border:1px solid #f00">块级元素 1</view>
<view style="border:1px solid #00f;width:200px;height:80px">块级元素 3</view>
<view style="border:1px solid #0f0;margin:15px;padding:20px">块级元素 2</view>
<!--块级元素的宽度、高度自定义设置-->
<!--块级元素的高度随内容决定，内容为块级元素-->
<view style="border:1px solid #ccc;">
    <view style="height:60px">块级元素 4</view>
</view>
<!--块级元素的高度随内容决定，内容为文本元素，块级元素的宽度为 100px-->
```

```
<view style="borde:1px solid #f00;width:100px;background-color:#ccc">父级元素高度随内容决定,内容为文本</view>
```

块级元素显示效果如图 3-4 所示。

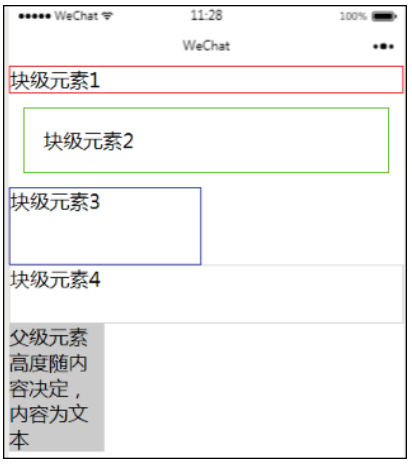


图 3-4　块级元素显示效果

3.2.2　行内元素

行内元素，不必从新的一行开始，它通常会和它前后的其他行内元素显示在同一行中，它们不占有独立的区域，仅仅靠自身内容支撑结构，一般不可以设置大小，常用于控制页面中文本的样式。通过设置 display 属性为 inline 将一个元素设置为行内元素。行内元素的特点如下：

- 行内元素不能设置高度和宽度，高度和宽度由内容决定。
- 行内元素内不能放置块级元素，只能容纳文本或其他行内元素。
- 同一块内，行内元素和其他行内元素在一行显示。

<text/>组件默认为行内元素，使用<view/>及<text/>组件演示盒子模型及行内元素的例子如下：

```
<view style="padding:20px">
      <text style="border:1px solid #f00">文本 1</text>
      <text style="border:1px solid #0f0;margin:10px; padding:5px">文本 2</text>
      <view style="border:1px solid #00f;display:inline">块级元素设置为行内元素
</view>一行显示不下,自动换行显示
</view>
```

行内元素显示效果如图 3-5 所示。

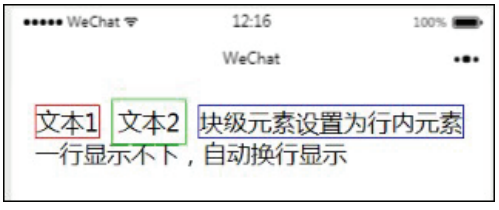


图 3-5　行内元素显示效果

3.2.3　行内块元素

当元素的 display 属性设置为 inline-block 时，元素被设置为一个行内块元素，行内块元素可设置高、宽、内外边距。示例代码如下：

```
    <view>
    元素的显示方式<view style="display:inline-block;border:1pxsolid #f00;margin:
10px;padding:10px;width:200px;">块级元素、行内元素和行内块元素</view>三种类型。
</view>
```

行内块元素运行效果如图 3-6 所示。

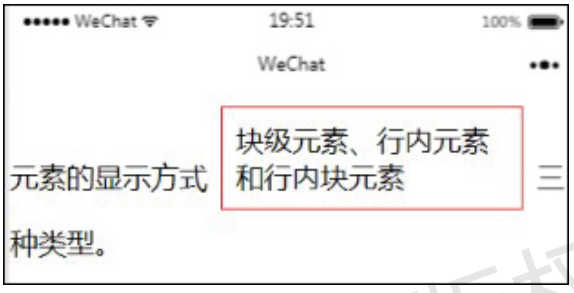


图 3-6　行内块元素运行效果

3.3　浮动与定位

3.3.1　元素浮动

元素浮动就是指设置了浮动属性的元素会脱离标准文档流的控制，移到其父元素中指定位置的过程。在 CSS 中，通过 float 属性来定义浮动，其基本格式如下：

```
{float:none|left|right;}
```

其中，none 表示元素不浮动，默认值；left 表示元素向左浮动；right 表示元素向右浮动。

下面分别对 box1，box2，box3 元素左浮动，代码如下：

```
<view>box1，box2，box3 没浮动</view>
<view style="border: 1px solid #f00; padding: 5px">
    <view style="border: 1px solid #0f0">box1</view>
    <view style="border: 1px solid #0f0">box2</view>
    <view style="border: 1px solid #0f0">box3</view>
</view>
<view>box1 左浮动</view>
<view style="border: 1px solid #f00; padding: 5px">
    <view style="float: left; border: 1px solid #0f0">box1</view>
    <view style="border: 1px solid #0f0">box2</view>
    <view style="border: 1px solid #0f0">box3</view>
</view>
<view>box1 box2 左浮动</view>
    <view style="border: 1px solid #f00; padding: 5px">
```

```
    <view style="float: left; border: 1px solid #0f0">box1</view>
    <view style="float:left;border:1px solid #0f0">box2</view>
    <view style="border:1px solid #0f0">box3</view>
</view>
<view>box1 box2 box3 左浮动</view>
    <view style="border:1pxsolid#f00;padding:5px">
    <view style="float:left;border:1px solid #0f0">box1</view>
    <view style="float:left;border:1px solid #0f0">box2</view>
    <view style="float:left;border:1px solid #0f0">box3</view>
</view>
```

元素浮动运行效果如图 3-7 所示。

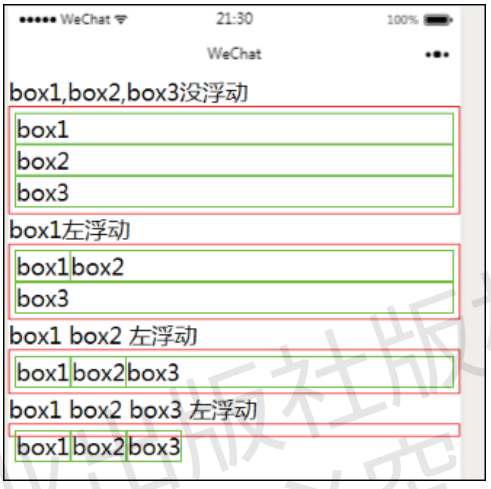


图 3-7　元素浮动运行效果

通过案例我们发现，当 box3 左浮动后，父元素的边框不能包裹 box3 元素，这时我们可以通过清除浮动来解决，代码如下：

```
<view>box1 box2 左浮动 box3 清除左浮动</view>
<view style="border:1pxsolid#f00;padding:5px">
<view style="float:left;border:1px solid #0f0">box1</view>
<view style="float:left;border:1px solid #0f0">box2</view>
<view style="clear:left;border:1px solid #0f0">box3</view>
</view>
```

清除浮动运行效果如图 3-8 所示。

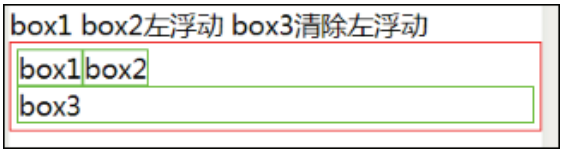


图 3-8　清除浮动运行效果

另一种方式是在父元素外添加一空元素，实现父元素包裹浮动元素，代码如下：

```
//wxml
<view>box1 box2 box3 左浮动在父元素后添加一空元素</view>
<view style="border:1pxsolid#f00;padding:5px"class="clearfloat">
<view style="float:left;border:1px solid #0f0">box1</view>
```

```
<view style="float:left;border:1px solid #0f0">box2</view>
<view style="float:left;border:1px solid #0f0">box3</view>
</view>
//wxss
.clearfloat::after{display:block;clear:both;height:0;content:""}
```

添加一空元素运行效果如图 3-9 所示。

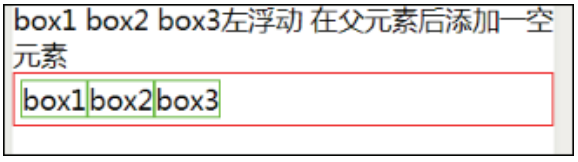


图 3-9　添加一空元素运行效果

3.3.2　元素定位

浮动布局虽然灵活，但却无法对元素的位置进行精确的控制。在 CSS 中，通过 position 属性可以实现对页面元素的精确定位。其基本格式如下：

```
{position:static|relative|absolute|fixed}
```

各参数含义介绍如下。

static：默认值，该元素按照标准流进行布局。

relative：相对定位，相对于它在原文档流的位置进行定位，它后面的盒子仍以标准流方式对待它。

absolute：绝对定位，相对于其上一个已经定位的父元素进行定位，绝对定位的盒子从标准流中脱离，它对其后的兄弟盒子的定位没有影响。

fixed：固定定位，相对于浏览器窗口进行定位。

下列示例中分别对 box1，box2，box3 元素定位，代码如下：

```
<!--三个元素匀未定位 static-->
<view style="border:1px solid #0f0;width:100px;height:100px">box1</view>
<view style="border:1px solid #0f0;width:100px;height:100px">box2</view>
<view style="border:1px solid #0f0;width:100px;height:100px">box3</view>
```

效果如图 3-10（a）所示（静态定位）。

```
<!--box2 元素相对定位 relative top:30px left:30px-->
<view style="border:1px solid #0f0;width:100px;height:100px">box1</view>
<view style="border:1px solid #0f0;width:100px;height:100px;position:relative;
left:30px;top:30px">box2</view>
<view style="border:1px solid #0f0;width:100px;height:100px">box3</view>
```

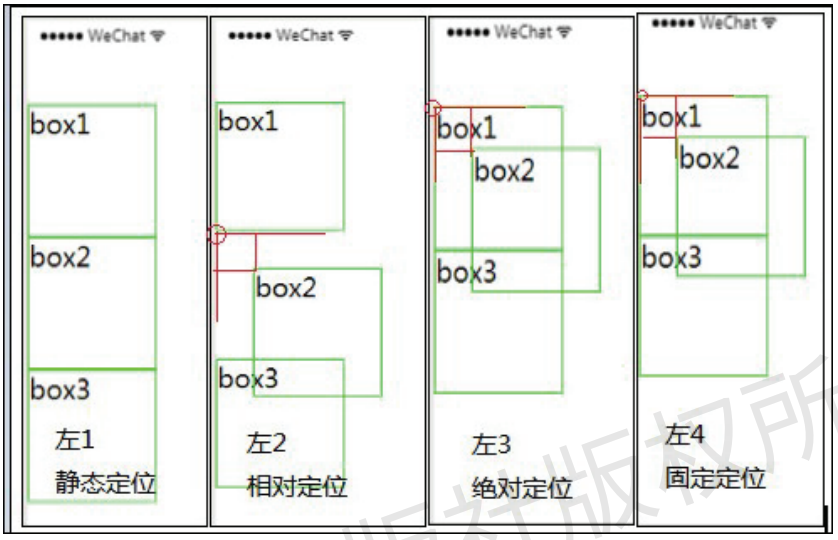
效果如图 3-10（b）所示（相对定位）。

```
<!--box2 元素绝对定位 absolute top:30px left:30px-->
<view style="border:1px solid #0f0;width:100px;height:100px">box1</view>
<view style="border:1px solid #0f0;width:100px;height:100px;position:absolute;
left:30px;top:30px">box2</view>
<view style="border:1px solid #0f0;width:100px;height:100px">box3</view>
```

效果如图 3-10（c）所示（绝对定位）。

```
<!--box2 元素固定定位 fixed top:30px left:30px-->
<view style="border:1px solid #0f0;width:100px;height:100px">box1</view>
<view  style="border:1px  solid  #0f0;width:100px;height:100px;position:fixed;
left:30px;top:30px">box2</view>
<view style="border:1px solid #0f0;width:100px;height:100px">box3</view>
```

元素定位运行效果如图 3-10（d）所示（固定定位）。



（a）　（b）　（c）　（d）

图 3-10　元素定位运行效果

通过案例我们发现，图 3-10（c）（绝对定位）和图 3-10（d）（固定定位）效果相同。这是因为它们的父元素是 page，没有定位。如果我们把它们的父元素设置为相对定位，其运行效果如图 3-11 所示。

box1，box2，box3 的父元素采用相对定位，box2 采用绝对定位，代码如下：

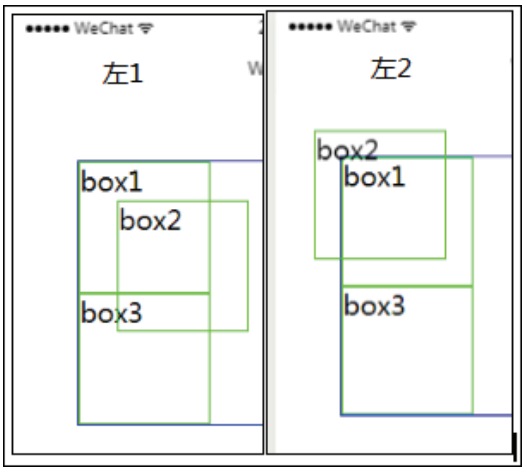
```
<view style="position:relative;top:50px;left:50px;border:1pxsolid#00f">
   <view style="border:1px solid #0f0;width:100px;height:100px">box1</view>
   <view style="border:1px solid #0f0;width:100px;height:100px;position:
absolute;left:30px;top:30px">box2</view>
   <view style="border:1px solid #0f0;width:100px;height:100px">box3</view>
</view>
```

其运行效果如图 3-11（a）所示。

box1，box2，box3 的父元素采用相对定位，box2 固定定位，代码如下：

```
<view style="position:relative;top:50px;left:50px;border:1pxsolid#00f">
   <view style="border:1px solid #0f0;width:100px;height:100px">box1</view>
   <view style="border:1px solid #0f0;width:100px;height:100px;position:fixed;
left:30px;top:30px">box2</view>
<view style="border:1px solid #0f0;width:100px;height:100px">box3</view>
</view>
```

其运行效果如图 3-11（b）所示。



（a）　　（b）

图 3-11　父元素相对定位，box2 绝对定位、固定定位

3.4　Flex 布局

Flex 布局是 W3C 组织在 2009 年提出的一种新布局方案，该布局可以简单快速地完成各种伸缩性的设计，很好地支持响应式布局。Flex 是 Flexible Box 的缩写，意为弹性盒子模型，可以简便、完整、响应式地实现各种页面布局。

Flex 布局主要由容器和项目组成。采用 Flex 布局的元素，称为 Flex 容器（flex contianer），它的所有直接子元素自动成为容器的成员，称为 Flex 项目（flex item）。

容器默认存在两根轴：水平的主轴（mainaxis）和垂直的交叉轴（crossaxis）。主轴的开始位置（与边框的交叉点）叫做 main start，结束位置叫做 main end；交叉轴的开始位置叫做 cross start，结束位置叫做 cross end。

项目默认沿主轴排列。单个项目占据的主轴空间叫做 main size，占据的交叉轴空间叫做 cross size，如图 3-12 所示。

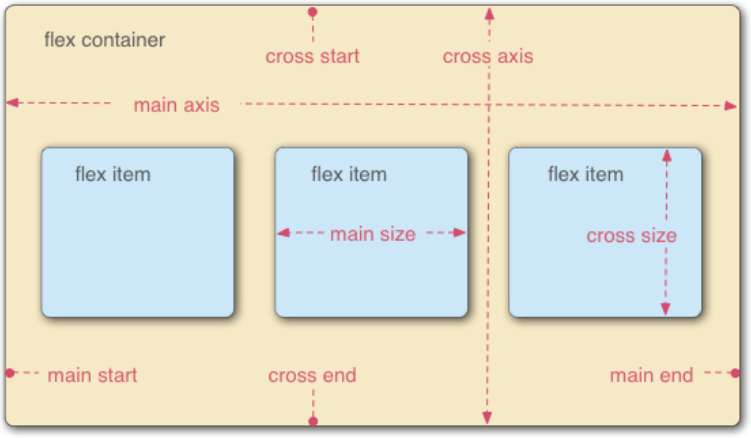


图 3-12　Flex 布局模型

通过设置 display 属性将一个元素指定为 Flex 布局，通过设置 flex-direction 属性来指定主轴方向，主轴既可以是水平方向，也可以是垂直方向。

3.4.1 容器属性

Flex 容器支持的属性有 7 种，如表 3-1 所示。

表 3-1 Flex 容器支持的属性

属 性 名	功 能
display	指定元素是否为 Flex 布局
flex-direction	指定主轴方向，决定项目的排列方向
flex-wrap	定义项目如何换行（超过一行时）
flex-flow	flex-direction 和 flex-wrap 的简写形式
justify-content	定义项目主轴上的对齐方式
align-items	定义项目在交叉轴的对齐方式
align-content	定义多根轴线的对齐方式

1．display

display 用来指定元素是否为 Flex 布局，语法格式为：

```
.box{display:flex|inline-flex;}
```

- flex：块级 Flex 布局，该元素变为弹性盒子。
- inline-flex：行内 Flex 布局，行内容器符合行内元素的特征，同时在容器内又符合 Flex 布局规范。

设置了 Flex 布局之后，子元素的 float、clear 和 vertical-align 属性将失效。

2．flex-direction

flex-direction 用于设置主轴的方向，即项目排列的方向，语法格式为：

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

- row：主轴为水平方向，起点在左端，当元素设置为 Flex 布局时，主轴默认为 row。
- row-reverse：主轴为水平方向，起点在右端。
- column：主轴为垂直方向，起点在顶端。
- column-reverse：主轴为垂直方向，起点在底端。

图 3-13 分别表示了不同主轴方向元素的显示效果。

3．flex-wrap

flex-wrap 用来指定项目如果在一条轴线排不下时，是否换行，其语法格式如下：

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

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

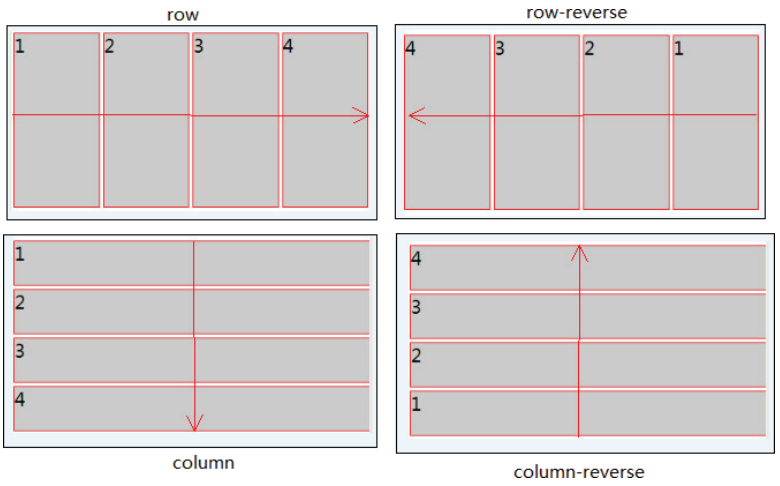


图 3-13　不同主轴方向元素的显示效果

当设置换行时，还需要设置 align-item 属性配合自动换行，但 align-item 的值不能为“stretch”。

图 3-14 表示了不同 flex-wrap 的显示效果。

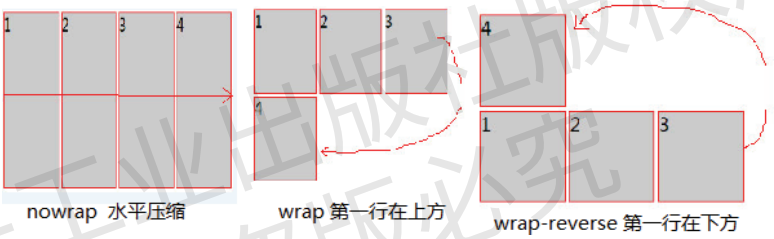


图 3-14　不同 flex-wrap 的显示效果

4．flex-flow

flex-flow 是 flex-direction 和 flex-wrap 的简写形式，默认值为 row nowrap。语法格式如下：

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

示例代码如下：

```
.box{flex-flow:row nowrap;}//水平方向不换行
.box{flex-flow:row-reverse wrap;}//水平逆方向换行
.box{flex-flow:column wrap-reverse;}垂直方向逆方向换行
```

5．justify-content

justify-content 属性用于定义项目在主轴上的对齐方式，语法格式如下：

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

justify-content 属性与主轴方向有关，默认主轴从左到右水平对齐。

- flex-start：左对齐，默认值。
- flex-end：右对齐。
- center：居中。

- space-between：两端对齐，项目之间的间隔都相等。
- sace-around：每个项目两侧的间隔相等。

图 3-15 展示了不同 justify-content 的显示效果。

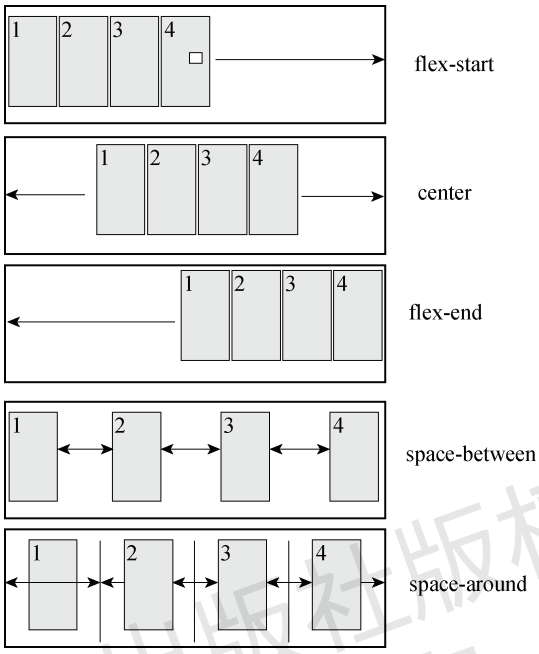


图 3-15　不同 justify-content 的显示效果

6．align-items

align-items 用于指定项目在交叉轴上的对齐方式，语法格式如下：

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

align-tiems 与交叉轴方向有关，默认由上到下的顺序交叉。

- flex-start：交叉轴起点对齐。
- flex-end：交叉轴终点对齐。
- center：交叉轴中线对齐。
- baseline：项目根据它们第一行文字的基线对齐。
- stretch：如果项目未设置高度或设置为 auto，项目将在交叉轴方向拉伸填充容器，默认值。

图 3-16 展示了不同对齐方式的效果，其代码如下：

```
//.wxml
<view class="cont1">
    <view class="item">1</view>
    <view class="itemitem2">2</view>
    <view class="itemitem3">3</view>
    <view class="itemitem4">4</view>
</view>
//wxss
.cont1{ display:
flex;
```

```
flex-direction:row;
align-items:baseline;
}
.item{
background-color:#ccc;
border:1px solid #f00;
height:100px;
width:50px;
margin:2px;
}
.item2{ height:80px;
}
.item3{ display:
flex;
height:50px;
align-items:flex-end;
}
.item4{ height:
120px;
}
```

7．align-content

align-content 用来定义项目有多根轴线（出现换行后）时在交叉轴上的对齐方式，如果只有一根轴线，该属性不起作用。其语法格式如下：

```
.box{align-content:flex-start|flex-end|end|basline|stretch;}
```

各属性值的含义等同 align-items 属性，图 3-17 展示了不同 align-content 的显示效果。

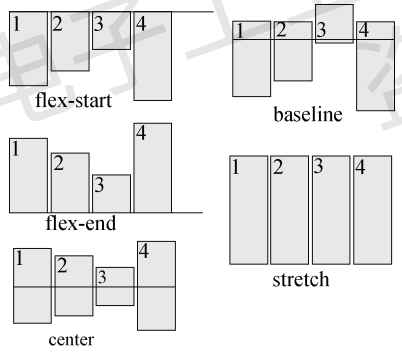


图 3-16　不同对齐方式的效果

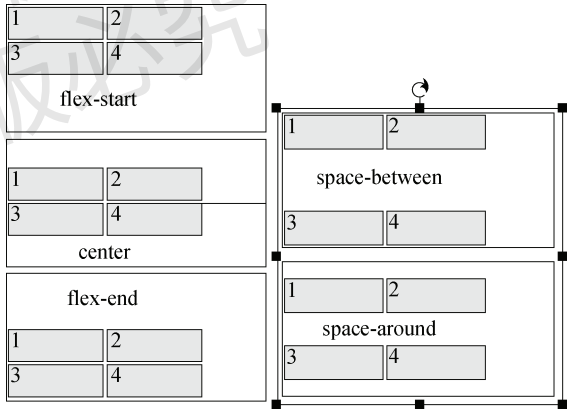


图 3-17　不同 align-content 的显示效果

3.4.2　项目属性

容器内的项目支持 6 个属性，如表 3-2 所示。

表 3-2　项目属性及描述

属性名	功能
order	定义项目的排列顺序
flex-grow	定义项目的放大比例（当有多余空间时）

（续表）

属性名	功能
flex-shrink	定义项目的缩小比例（当空间不足时）
flex-basis	定义在分配多余空间之前，项目占据的主轴空间
flex	flex-grow、flex-shrink、flex-basis 的简写
align-self	用来设置单独的伸缩项目在交叉轴上的对齐方式

1．order

order 属性用于定义项目的排列顺序。其数值越小，排列越靠前，默认为 0。其语法格式如下：

```
.item{order:<number>;}
```

示例代码如下：

```
<view class="cont1">
    <view class="item" >1</view>
    <view class="item ">2</view>
    <view class="item ">3</view>
    <view class="item ">4</view>
</view>
<view class="cont1">
    <view class="item" style="order:1" >1</view>
    <view class="item" style="order:3">2</view>
    <view class="item" style="order:2">3</view>
    <view class="item">4</view>
</view>
```

不同 order 属性的运行效果如图 3-18 所示

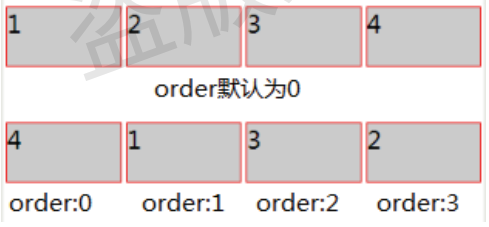


图 3-18　不同 order 属性的运行效果

2．flex-grow

flex-grow 用于定义项目的放大比例，默认为 0，即不放大。其语法格式如下：

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

示例代码如下：

```
<view class="cont1">
    <view class="item" >1</view>
    <view class="item ">2</view>
    <view class="item ">3</view>
    <view class="item ">4</view>
</view>
<view class="cont1">
```

```
    <view class="item" >1</view>
    <view class="item" style="flex-grow:1">2</view>
    <view class="item" style="flex-grow:2">3</view>
    <view class="item ">4</view>
</view>
```

flex-grow 示例运行效果如图 3-19 所示。示例中把剩余空间分为 3 份（flow-grow:1+flow-grow:2），其中元素 2 占 1 份，元素 3 占 2 份。

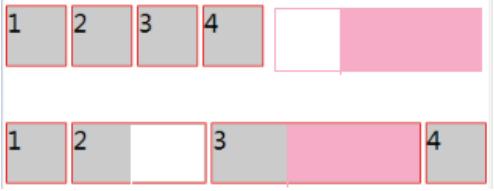


图 3-19　flex-grow 示例运行效果

3．flex-shrink

flex-shrink 用来定义项目的缩小比例，默认为 1，如果空间不足，该项目将缩小，语法格式如下：

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

示例代码如下：

```
<view class="cont1">
    <view class="item" >1</view>
    <view class="item ">2</view>
    <view class="item ">3</view>
    <view class="item ">4</view>
</view>
<view class="cont1">
    <view class="item" >1</view>
    <view class="item" style="flex-shrink:2">2</view>
    <view class="item" style="flex-shrink:1">3</view>
    <view class="item" style="flex-shrink:4">4</view>
</view>
```

flex-shrink 示例运行效果如图 3-20 所示。

假定容器宽度为 800px，4 个元素的宽度分别为 240px，元素宽度比容器多 160px（即 240×4−800）。当 flex-shrink 属性为 1 时，由于空间不足，4 个项目等比例缩小，每个元素变为 200px，如图 3-20 中的上半部分；当 4 个元素的缩小比例为 1、2、1、4 时，它们的宽度分别减小 20px、40px、20px、80px［计算公式为 160×缩小比例/(1+2+1+4)］，则它们的实际宽度变为 220px、200px、220px、160px。

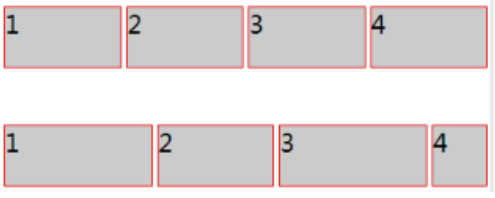


图 3-20　flex-shrink 示例运行效果

4．flex-basis

flex-basis 属性用来定义伸缩项目的基准值，剩余的空间将按比例进行缩放。它的默认值为 auto，即项目的本来大小，语法格式如下：

```
.item{flex-basis:<number>|auto;}
```

示例代码如下：

```
<view class="cont1">
    <view class="item">1</view>
    <view class="item ">2</view>
    <view class="item ">3</view>
    <view class="item ">4</view>
</view>
<view class="cont1">
    <view class="item">1</view>
    <view class="item" style="flex-basis:100px" >2</view>
    <view class="item" style="flex-basis:200px" >3</view>
    <view class="item ">4</view>
</view>
```

flex-basis 示例运行效果如图 3-21 所示。

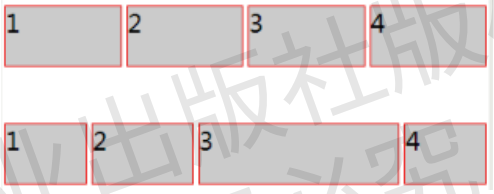


图 3-21　flex-basis 示例运行效果

5．flex

flex 属性是 flex-grow、flex-shrink 和 flex-basis 的简写，其默认值为 0，1，auto。语法格式如下：

```
.item{flex:<flex-grow>|<flex-shrink>|<flex-basis>;}
```

示例代码如下：

```
.item{flex:auto;}//等价于.item{flex:1 1 auto;}
.item{flex:none;}//等价于.item{flex:0 0 auto;}
```

6．align-self

align-self 属性用来指定单独的伸缩项目在交叉轴上的对齐方式。该属性会重写默认的对齐方式，语法格式如下：

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

该属性值中除了 auto，其余的和容器 align-items 属性值完全一致。

auto 表示继承容器 align-items 属性，如果没有父元素，则等于 stretch，默认值。

本章小结

本章首先讲解页面布局中最基本的盒子模型，其次讲解浮动和定位，最后重点讲解 Flex 布局的基本原理、容器和项目的相关属性。学好这些内容可为小程序项目的布局打下良好的基础。页面布局知识体系如图 3-22 所示。

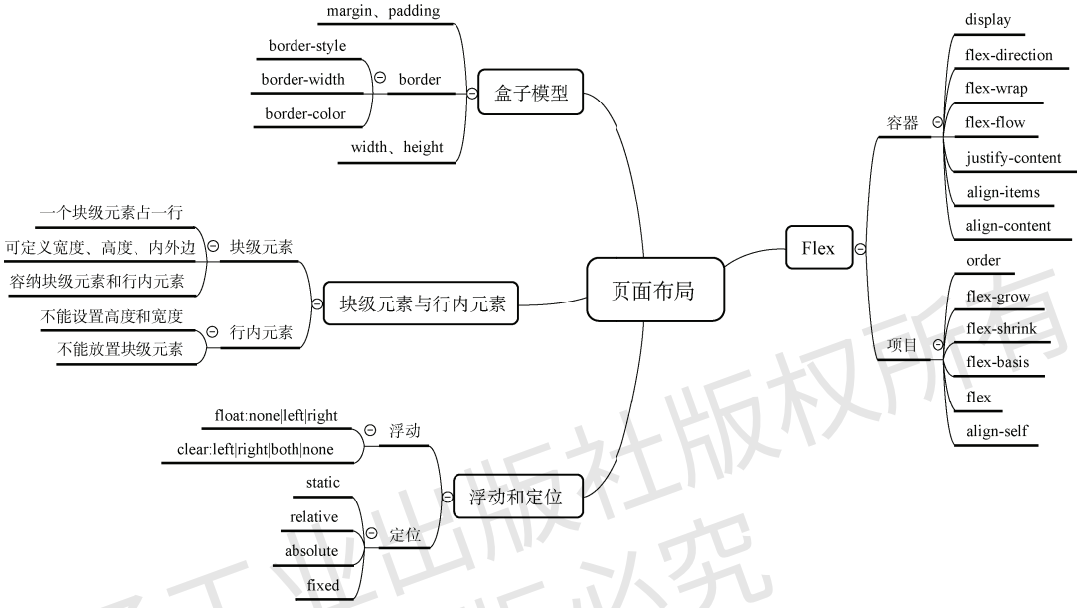


图 3-22　页面布局知识体系

思考练习题

一、选择题

1. 下列选项中，可以改变盒子模型外边距的是（　　）。

A. padding　　B. margin　　C. border　　D. type

2. 下列选项中，可以设置背景图像平铺方式的是（　　）。

A. background-repeat：no-repeat

B. background-attachment：fixed

C. background-attachment：scroll

D. background-repeat：repeat-x

3. 下列不能清除浮动的选项是（　　）。

A. left　　B. right　　C. both　　D. none

4. 在 Flex 布局中，flex-direction 属性有（　　）个选项。

A. 1　　B. 2　　C. 3　　D. 4

5. 在 Flex 布局中，容器内有 4 个项目，容器宽度为 600px，每个项目的宽度为 200px，默认情况下，每个项目的宽度为（　　）。

A. 200px　　B. 150px　　C. 100px　　D. 不确定

二、分析题

分析下列代码，实现图 3-23 所示的页面布局。

```
//cal.wxml
<view class="content">
     <view class="layout-top">
          <view class="screen">168</view>
     </view>
     <view class="layout-bottom">
          <view class="btnGroup">
          <view class="item orange" >C</view>
          <view class="item orange" >←</view>
          <view class="item orange" >#</view>
          <view class="item orange" >+</view>
     </view>
     <view class="btnGroup">
          <view class="item blue" >9</view>
          <view class="item blue" >8</view>
          <view class="item blue" >7</view>
          <view class="item orange" >-</view>
     </view>
     <view class="btnGroup">
          <view class="item blue" >6</view>
          <view class="item blue" >5</view>
          <view class="item blue">4</view>
          <view class="item orange">×</view>
     </view>
     <view class="btnGroup">
          <view class="item blue" >3</view>
          <view class="item blue" >2</view>
          <view class="item blue" >1</view>
          <view class="item orange">÷</view>
     </view>
     <view class="btnGroup">
          <view class="item blue zero" >0</view>
          <view class="item blue" >.</view>
          <view class="item orange" >=</view>
     </view>
   </view>
</view>
//app.wxss
.container {
   height:100%;
   display:flex;
   flex-direction:column;
   align-items:center;
   justify-content:space-between;
   padding:200rpx0;
   box-sizing: border-box;
}
//cal.wxss
.content{
   height:100%;
   display:flex;
   flex-direction:column;
   align-items:center;
   background-color: #ccc;
```

```
    font-family: "Microsoft YaHei";
    overflow-x: hidden;
}
.layout-top{
    width:100%;
    margin-bottom:30rpx;
}
.layout-
    bottom{ width:
    100%;
}
.screen{
    text-align:right;
    width:100%;
    line-height:130rpx;
    padding:0 10rpx;
    font-weight:bold;
    font-size:60px;
    box-sizing:border-box;
    border-top:1px solid #fff;
}
.btnGroup{
    display:flex;
    flex-direction:row;
    flex:1;
    width:100%;
    height:4rem;
    background-color:#fff;
}
.item{
    width:25%;
    display:flex;
    align-items:center;
    flex-direction:column;
    justify-content:center;
    margin-top:1px;
    margin-right:1px;
}
.item:active{
    background-color:#ff0000;
}
.zero{
    width:50%;
}
.orange{
    color:#fef4e9;
    background:#f78d1d;
    font-weight:bold;
}
.blue{
    color:#d9eef7;
    background-color:#0095cd;
}
.iconBtn{
    display:flex;
}
.icon{
    display:flex;
    align-items:center;width:100%;
    justify-content:center;
}
```

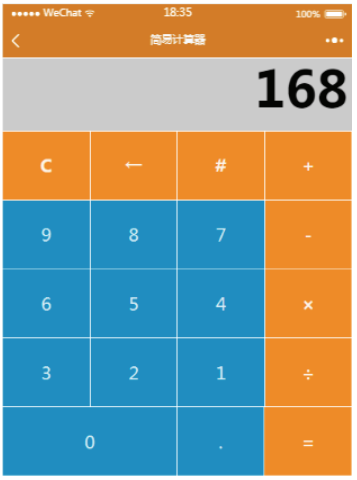


图 3-23　分析题图

三、操作题

分析页面结构，实现如图 3-24 所示的布局效果。

图 3-24　布局效果