Flutter(四)之Flutter的布局Widget

原创 coderwhy coderwhy

前言一：接下来一段时间我会陆续更新一些列Flutter文字教程

更新进度： 每周至少两篇；

更新地点： 首发于公众号，第二天更新于掘金、思否、开发者头条等地方；

更多交流： 可以添加我的微信 372623326，关注我的微博：coderwhy

希望大家可以 **帮忙转发，点击在看** ，给我更多的创作动力。

前言二：为了实现界面内组件的各种排布方式，我们需要进行布局，和其他端不同的是，Flutter中因为万物皆Widget，所以布局也是使用Widget来完成的。

Flutter中的布局组件非常多，有31个用于布局的组件， **Flutter布局组件** [1] ；

在学习的过程中，我们没必要一个个全部掌握，掌握最常用的，一些特殊的组件用到时去查文档即可；

Flutter将布局组件分成了 `单子布局组件` （Single-child layout widgets） 和 `多子布局组件` （Multi-child layout widgets）

一. 单子布局组件

单子布局组件的含义是其只有一个子组件，可以通过设置一些属性设置该子组件所在的位置信息等。

比较常用的单子布局组件有：Align、Center、Padding、Container。

1.1. Align组件

1.1.1. Align介绍

看到 `Align` 这个词，我们就知道它有我们的对齐方式有关。

在其他端的开发中（iOS、Android、前端）Align通常只是一个属性而已，但是Flutter中Align也是一个组件。

我们可以通过源码来看一下Align有哪些属性：

```
const Align({
  Key key,
  this.alignment: Alignment.center, // 对齐方式，默认居中对
齐
  this.widthFactor, // 宽度因子，不设置的情况，会尽可能大
  this.heightFactor, // 高度因子，不设置的情况，会尽可能大
  Widget child // 要布局的子Widget
})
```

这里我们特别解释一下 `widthFactor` 和 `heightFactor` 作用：

- 因为子组件在父组件中的对齐方式必须有一个前提，就是父组件得知道自己的范围（宽度和高度）；
- 如果 `widthFactor` 和 `heightFactor` 不设置，那么默认Align会尽可能的大（尽可能占据自己所在的父组件）；
- 我们也可以对他们进行设置，比如widthFactor设置为3，那么相对于Align的宽度是子组件跨度的3倍；

1.1.2. Align演练

我们简单演练一下Align

```
class MyHomeBody extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Align(
      child: Icon(Icons.pets, size: 36, color: Colors.red),
      alignment: Alignment.bottomRight,
      widthFactor: 3,
      heightFactor: 3,
    );
  }
}
```

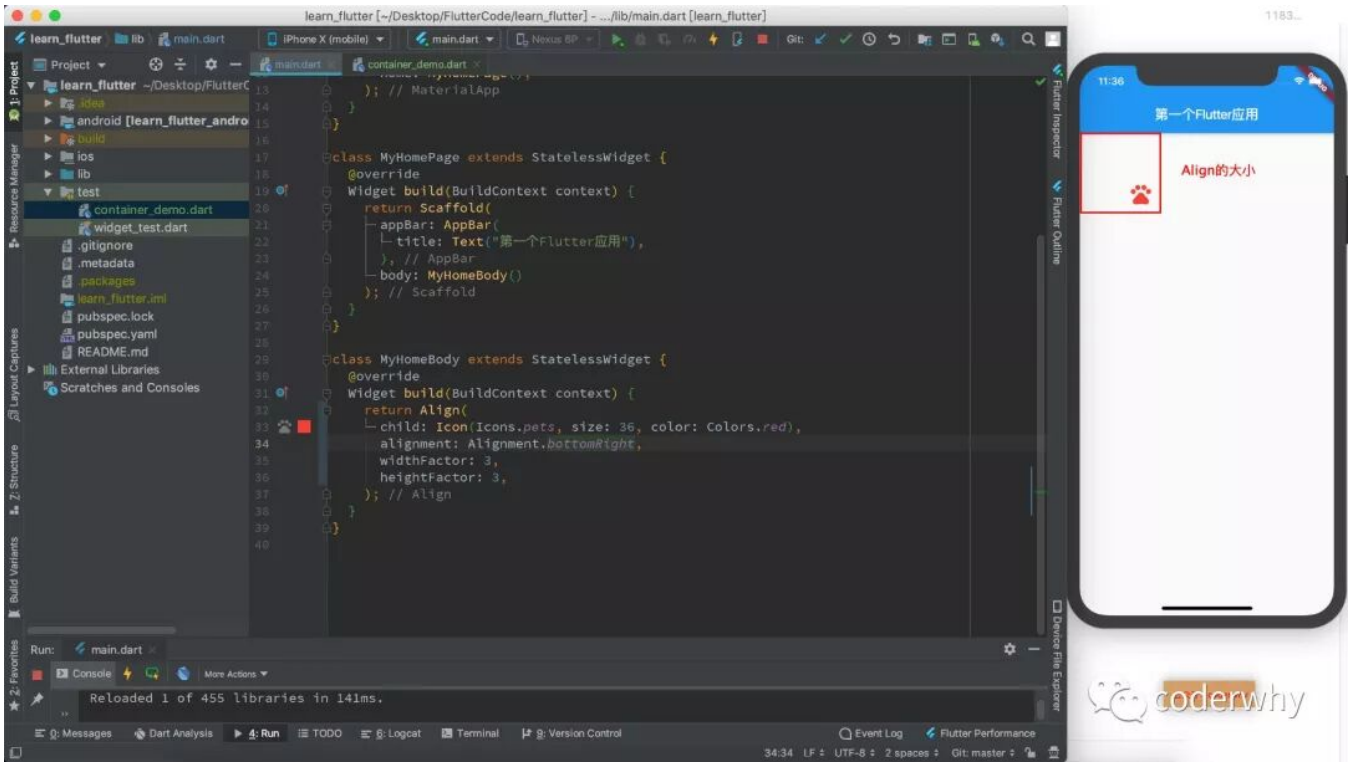


image-20190903113659914

1.2. Center组件

1.2.1. Center介绍

Center组件我们在前面已经用过很多次了。

事实上Center组件继承自Align，只是将alignment设置为Alignment.center。

源码分析：

```
class Center extends Align {
  const Center({
    Key key,
    double widthFactor,
    double heightFactor,
    Widget child
  }) : super(key: key, widthFactor: widthFactor, heightFactor: heightFactor, child: child);
}
```

1.2.2. Center演练

我们将上面的代码Align换成Center

```
class MyHomeBody extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Center(
      child: Icon(Icons.pets, size: 36, color: Colors.red),
      widthFactor: 3,
      heightFactor: 3,
    );
  }
}
```

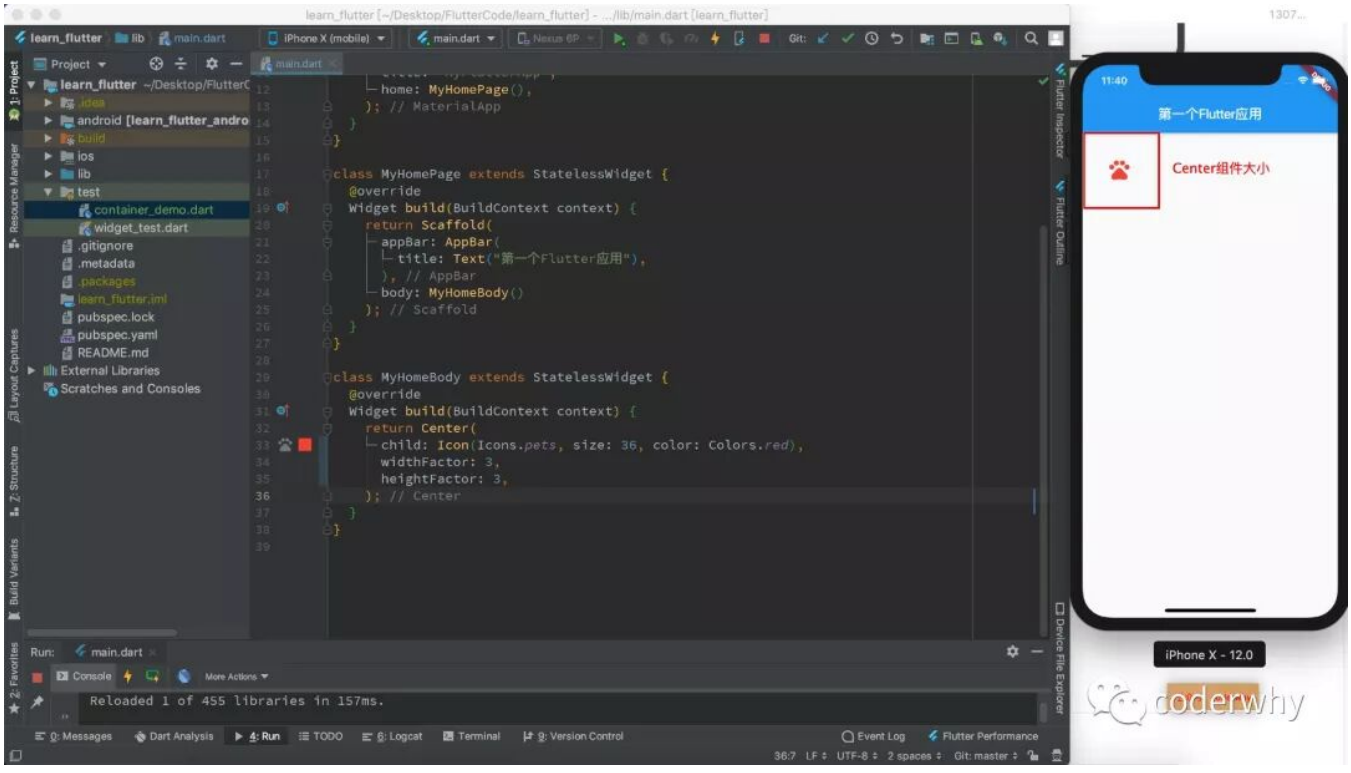


image-20190903114101386

1.3. Padding组件

1.3.1. Padding介绍

> Padding组件在其他端也是一个属性而已，但是在Flutter中是一个Widget，但是Flutter中没有Margin这样一个Widget，这是因为外边距也可以通过Padding来完成。
>
> Padding通常用于设置子Widget到父Widget的边距（你可以称之为是父组件的内边距或子Widget的外边距）。

源码分析：

```
const Padding({
  Key key,
  @required this.padding, // EdgeInsetsGeometry类型（抽象类），使用EdgeInsets
  Widget child,
})
```

1.3.2. Padding演练

代码演练：

```
class MyHomeBody extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: EdgeInsets.all(20),
      child: Text(
        "莫听穿林打叶声，何妨吟啸且徐行。竹杖芒鞋轻胜马，谁怕？一蓑烟雨任平生。"
,
        style: TextStyle(
          color: Colors.redAccent,
          fontSize: 18
        ),
      ),
    );
  }
}
```

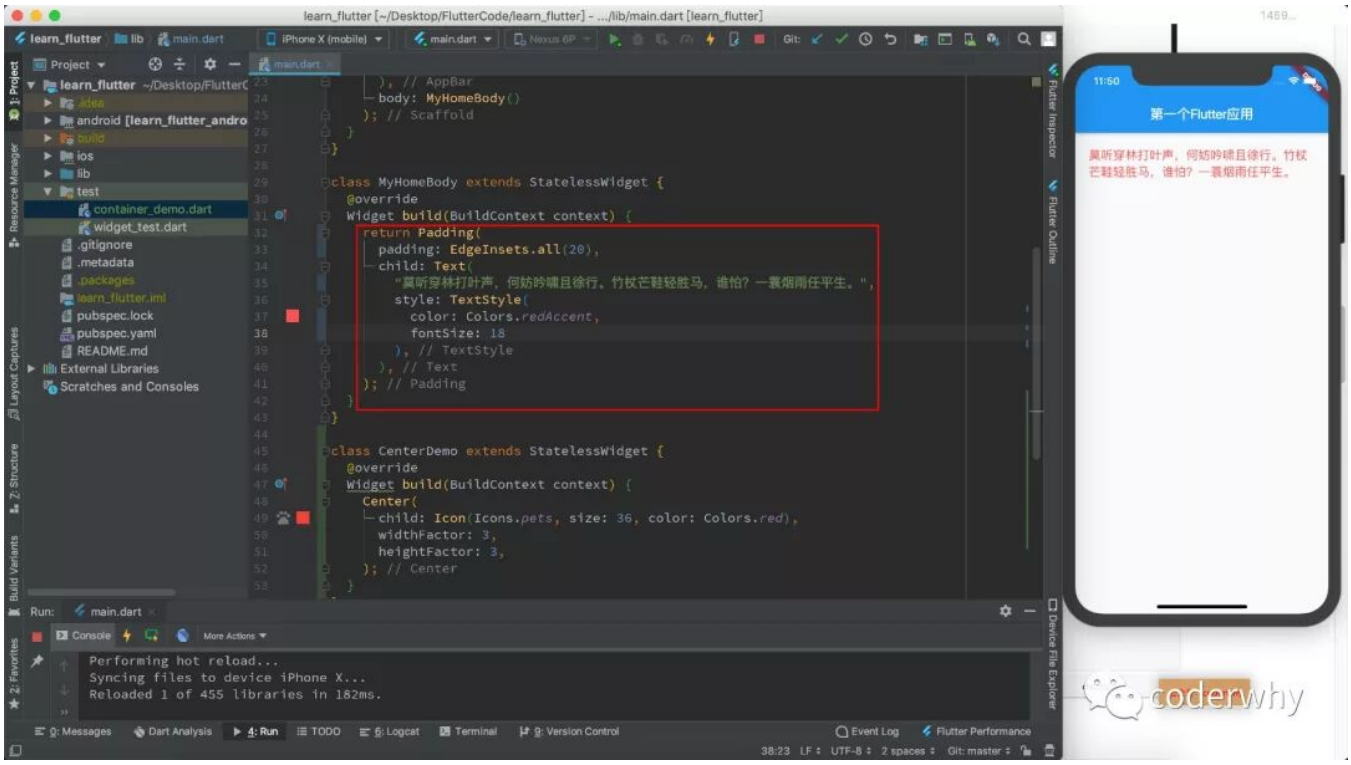


image-20190903115027193

1.4. Container组件

> Container组件类似于其他Android中的View，iOS中的UIView。
>
> 如果你需要一个视图，有一个背景颜色、图像、有固定的尺寸、需要一个边框、圆角等效果，那么就可以使用Container组件。

14.1. Container介绍

Container在开发中被使用的频率是非常高的，特别是我们经常会将其作为容器组件。

下面我们来看一下Container有哪些属性：

```
Container({
  this.alignment,
  this.padding, //容器内补白，属于decoration的装饰范围
  Color color, // 背景色
  Decoration decoration, // 背景装饰
  Decoration foregroundDecoration, //前景装饰
  double width,//容器的宽度
  double height, //容器的高度
  BoxConstraints constraints, //容器大小的限制条件
  this.margin,//容器外补白，不属于decoration的装饰范围
  this.transform, //变换
  this.child,
})
```

大多数属性在介绍其它容器时都已经介绍过了，不再赘述，但有两点需要说明：

- 容器的大小可以通过 `width` 、 `height` 属性来指定，也可以通过 `constraints` 来指定，如果同时存在时， `width` 、 `height` 优先。实际上Container内部会根据 `width` 、 `height` 来生成一个 `constraints` ；
- `color` 和 `decoration` 是互斥的，实际上，当指定color时，Container内会自动创建一个decoration；
- `decoration` 属性稍后我们详细学习；

1.4.2. Container演练

简单进行一个演示：

```
class MyHomeBody extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Center(
      child: Container(
        color: Color.fromRGBO(3, 3, 255, .5),
        width: 100,
        height: 100,
        child: Icon(Icons.pets, size: 32, color: Colors.white),
      ),
    );
  }
}
```

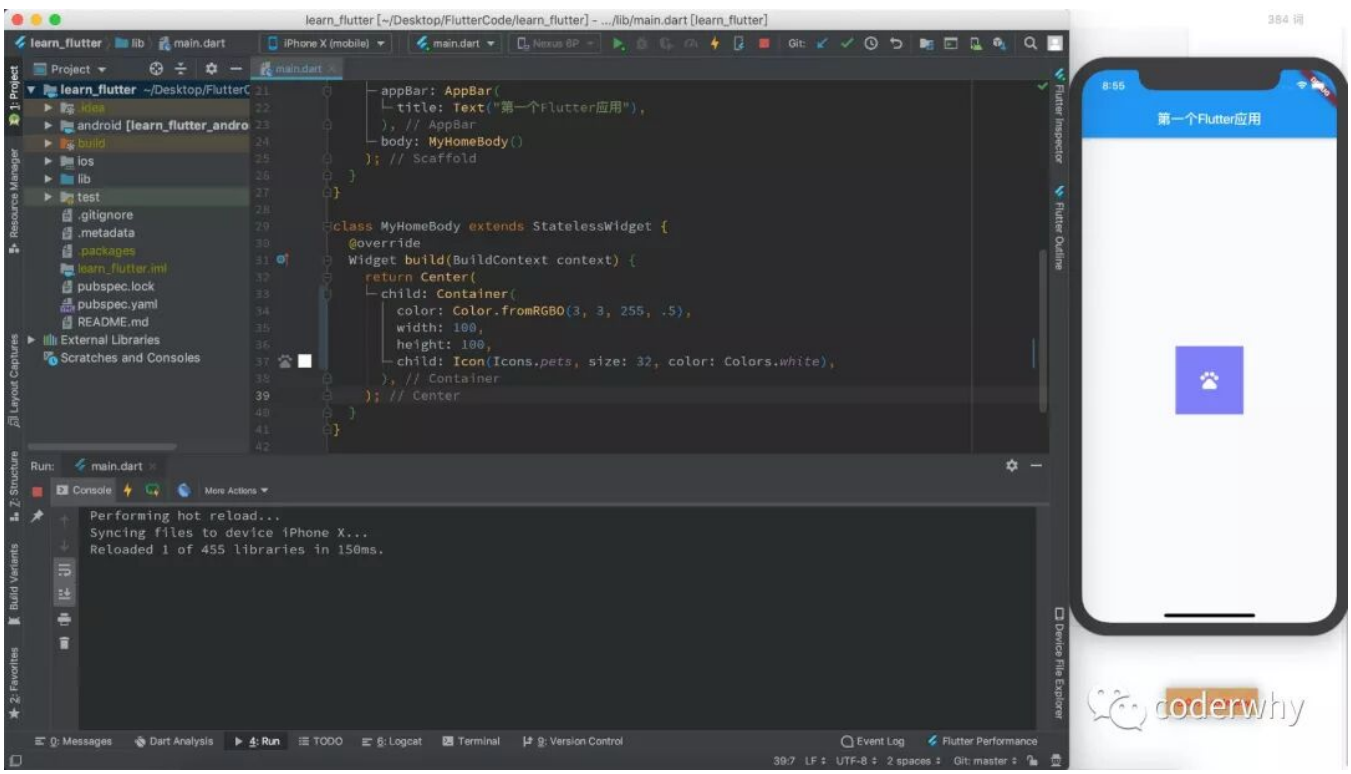


image-20190903085602884

1.4.3. BoxDecoration

Container有一个非常重要的属性 `decoration` ：

- 他对应的类型是Decoration类型，但是它是一个抽象类。

- 在开发中，我们经常使用它的实现类BoxDecoration来进行实例化。

BoxDecoration常见属性：

```
  const BoxDecoration({
    this.color, // 颜色，会和Container中的color属性冲突
    this.image, // 背景图片
    this.border, // 边框，对应类型是Border类型，里面每一个边框使用BorderSide
    this.borderRadius, // 圆角效果
    this.boxShadow, // 阴影效果
    this.gradient, // 渐变效果
    this.backgroundBlendMode, // 背景混合
    this.shape = BoxShape.rectangle, // 形变
  })
```

部分效果演示：

```
class MyHomeBody extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Center(
      child: Container(
//        color: Color.fromRGBO(3, 3, 255, .5),
        width: 150,
        height: 150,
        child: Icon(Icons.pets, size: 32, color: Colors.white),
        decoration: BoxDecoration(
          color: Colors.amber, // 背景颜色
          border: Border.all(
            color: Colors.redAccent,
            width: 3,
            style: BorderStyle.solid
          ), // 这里也可以使用Border.all统一设置
//            top: BorderSide(
//              color: Colors.redAccent,
//              width: 3,
//              style: BorderStyle.solid
//            ),
          borderRadius: BorderRadius.circular(20), // 这里也可以使用.only分别设
置
          boxShadow: [
            BoxShadow(
              offset: Offset(5, 5),
              color: Colors.purple,
              blurRadius: 5
            )
          ],
//          shape: BoxShape.circle, // 会和borderRadius冲突
          gradient: LinearGradient(
            colors: [
              Colors.green,
              Colors.red
            ]
          )
        ),
      ),
    );
  }
}
```

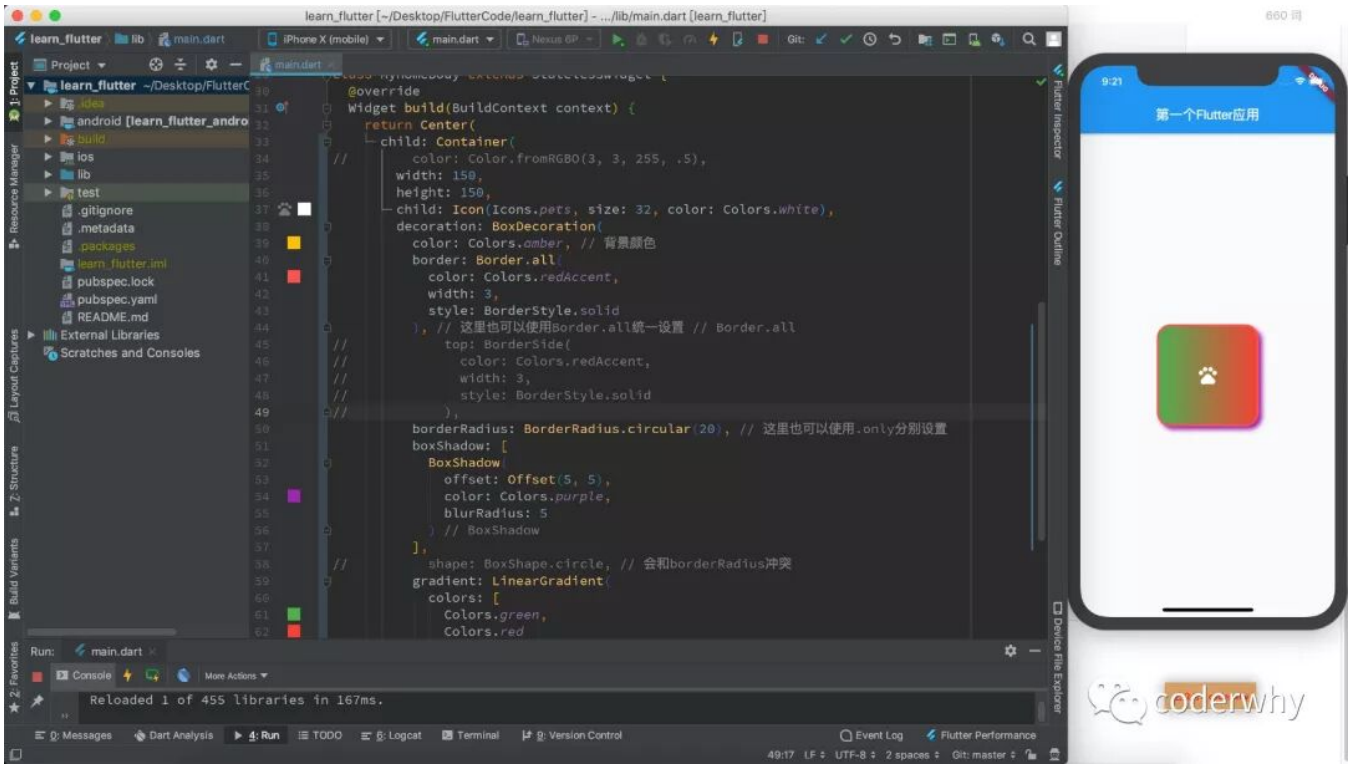


image-20190903092137469

1.4.4. 实现圆角图像

上一个章节我们提到可以通过 `Container+BoxDecoration` 来实现圆角图像。

实现代码如下：

```
class HomeContent extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Center(
      child: Container(
        width: 200,
        height: 200,
        decoration: BoxDecoration(
          borderRadius: BorderRadius.circular(20),
          image: DecorationImage(
            image: NetworkImage("https://tva1.sinaimg.cn/large/006y8mN6gy1g7aa03bmfpj3069069mx8.jpg")
,
          )
        ),
      ),
    );
  }
}
```

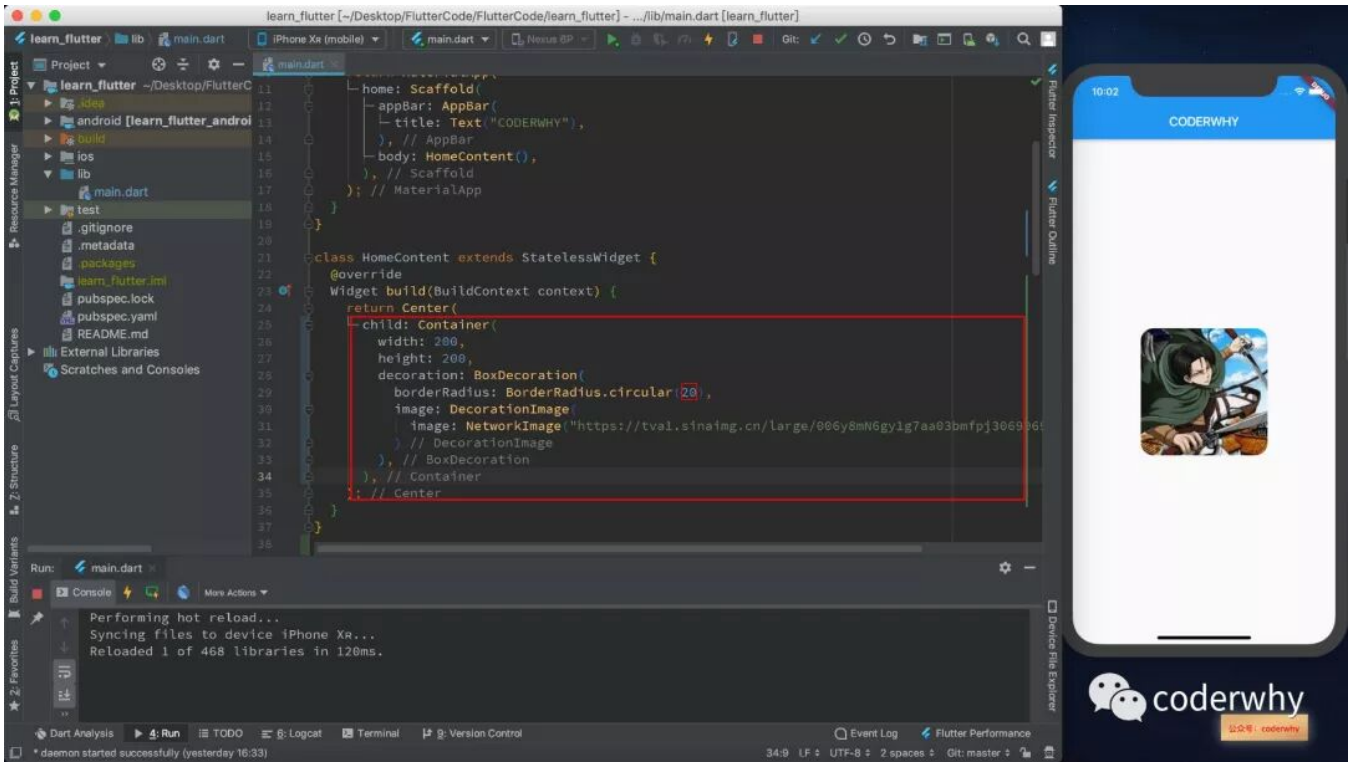


image-20190924100221266

二. 多子布局组件

在开发中，我们经常需要将多个Widget放在一起进行布局，比如水平方向、垂直方向排列，甚至有时候需要他们进行层叠，比如图片上面放一段文字等；

这个时候我们需要使用多子布局组件（Multi-child layout widgets）。

比较常用的多子布局组件是Row、Column、Stack，我们来学习一下他们的使用。

2.1. Flex组件

事实上，我们即将学习的Row组件和Column组件都继承自Flex组件。

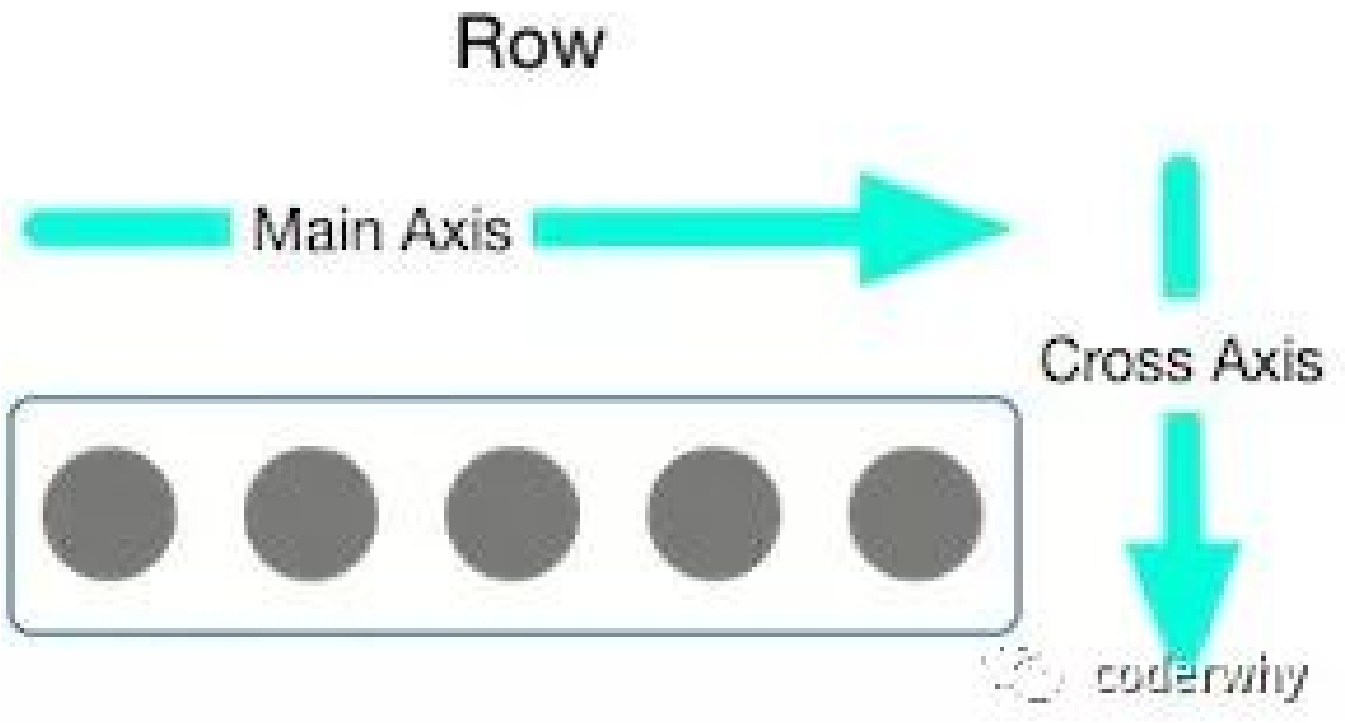
- Flex组件和Row、Column属性主要的区别就是多一个direction。
- 当direction的值为Axis.horizontal的时候，则是Row。
- 当direction的值为Axis.vertical的时候，则是Column。

在学习Row和Column之前，我们先学习 `主轴` 和 `交叉轴` 的概念。

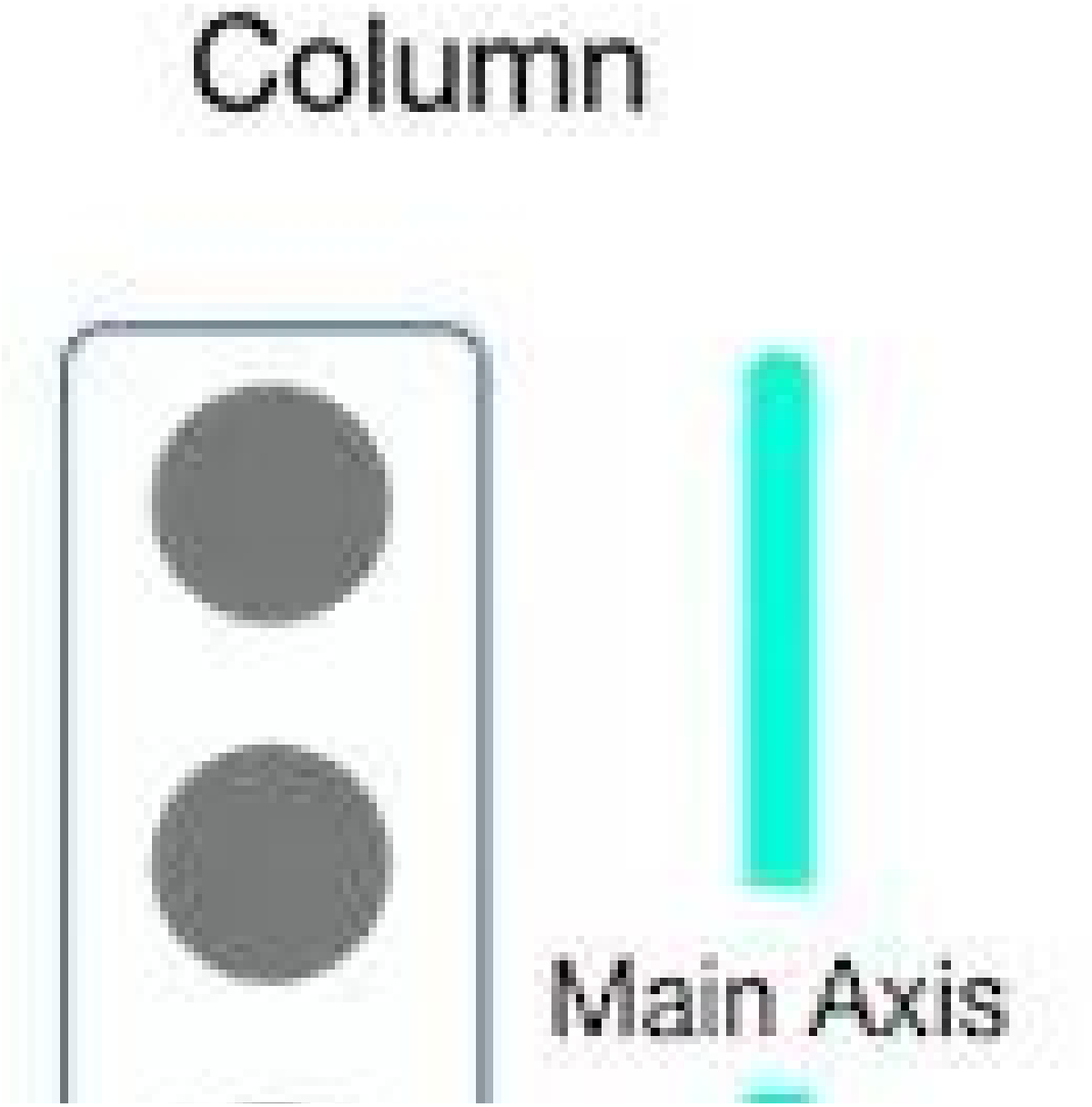
因为Row是一行排布，Column是一列排布，那么它们都存在两个方向，并且两个Widget排列的方向应该是对立的。

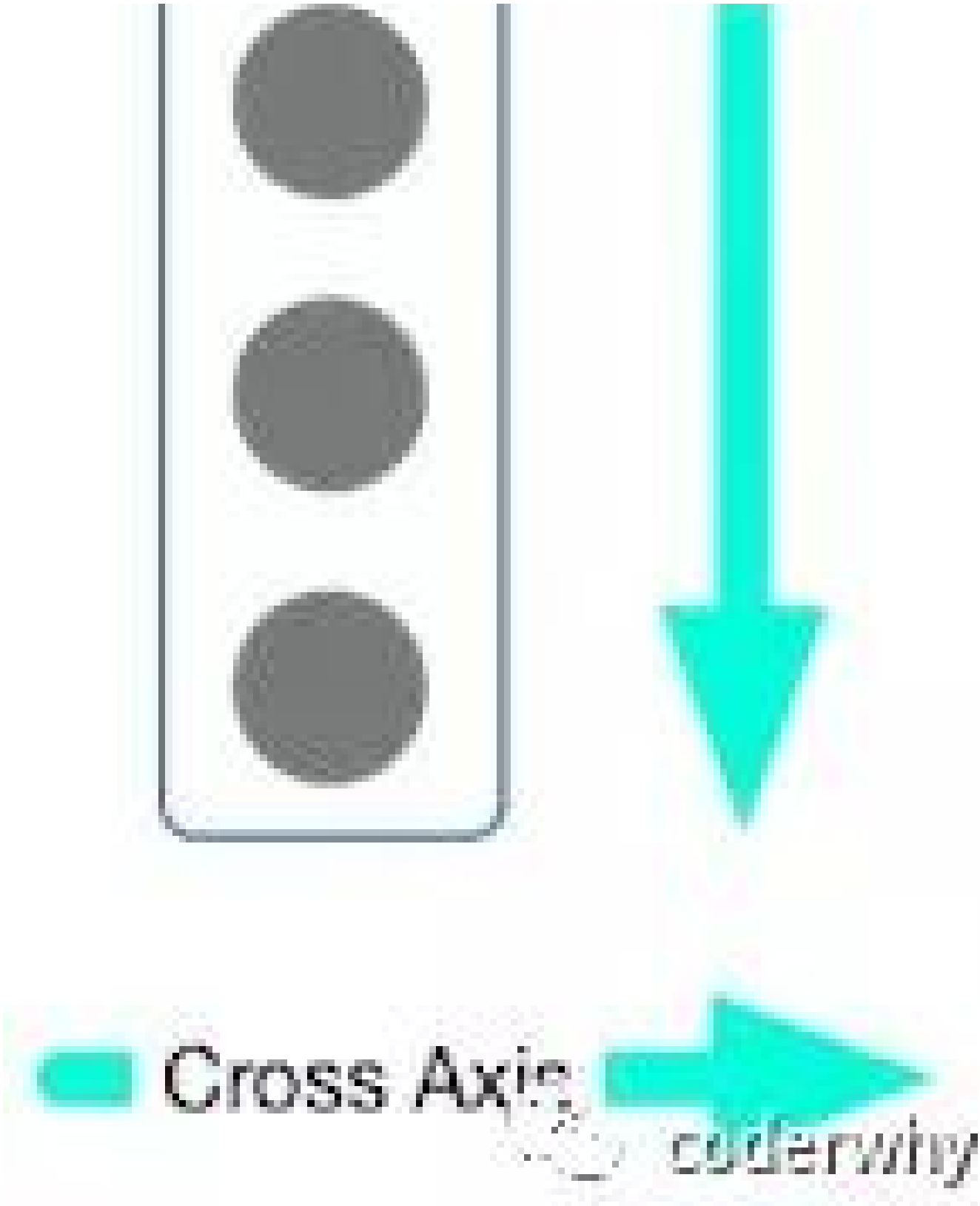
它们之中都有主轴（MainAxis）和交叉轴（CrossAxis）的概念：

- 对于Row来说，主轴（MainAxis）和交叉轴（CrossAxis）分别是下图



- 对于Column来说，主轴（MainAxis）和交叉轴（CrossAxis）分别是下图





2.1. Row组件

2.1.1. Row介绍

> Row组件用于将所有的子Widget排成一行，实际上这种布局应该是借鉴于Web的Flex布局。
>
> 如果熟悉Flex布局，会发现非常简单。

从源码中查看Row的属性：

```
Row({
  Key key,
  MainAxisAlignment mainAxisAlignment = MainAxisAlignment.start, // 主轴对齐方式
  MainAxisSize mainAxisSize = MainAxisSize.max, // 水平方向尽可能大
  CrossAxisAlignment crossAxisAlignment = CrossAxisAlignment.center, // 交叉处对齐方式
  TextDirection textDirection, // 水平方向子widget的布局顺序（默认为系统当前Locale环境的文本方向(如中文、英语都是从左往右，而阿拉伯语是从右往左)）
  VerticalDirection verticalDirection = VerticalDirection.down, // 表示Row纵轴（垂直）的对齐方向
  TextBaseline textBaseline, // 如果上面是baseline对齐方式，那么选择什么模式（有两种可选）
  List<Widget> children = const <Widget>[],
})
```

部分属性详细解析：（不过文字是真的难描述，后续推出视频学习较差）

mainAxisSize：

- 表示Row在主轴(水平)方向占用的空间，默认是 `MainAxisSize.max` ，表示尽可能多的占用水平方向的空间，此时无论子widgets实际占用多少水平空间，Row的宽度始终等于水平方向的最大宽度
- 而 `MainAxisSize.min` 表示尽可能少的占用水平空间，当子widgets没有占满水平剩余空间，则Row的实际宽度等于所有子widgets占用的的水平空间；

mainAxisAlignment：表示子Widgets在Row所占用的水平空间内对齐方式

- 如果mainAxisSize值为 `MainAxisSize.min` ，则此属性无意义，因为子widgets的宽度等于Row的宽度
- 只有当mainAxisSize的值为 `MainAxisSize.max` 时，此属性才有意义
- `MainAxisAlignment.start` 表示沿textDirection的初始方向对齐，
- 如textDirection取值为 `TextDirection.ltr` 时，则 `MainAxisAlignment.start` 表示左对齐，textDirection取值为 `TextDirection.rtl` 时表示从右对齐。
- 而 `MainAxisAlignment.end` 和 `MainAxisAlignment.start` 正好相反；
- `MainAxisAlignment.center` 表示居中对齐。

crossAxisAlignment：表示子Widgets在纵轴方向的对齐方式

- Row的高度等于子Widgets中最高的子元素高度
- 它的取值和MainAxisAlignment一样(包含 `start` 、 `end` 、 `center` 三个值)
- 不同的是crossAxisAlignment的参考系是verticalDirection，即verticalDirection值为 `VerticalDirection.down` 时 `crossAxisAlignment.start` 指顶部对齐，verticalDirection值为 `VerticalDirection.up` 时， `crossAxisAlignment.start` 指底部对齐；而 `crossAxisAlignment.end` 和 `crossAxisAlignment.start` 正好相反；

2.1.2. Row演练

我们来对部分属性进行简单的代码演练，其他一些属性大家自己学习一下

```
class MyHomeBody extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Row(
      mainAxisAlignment: MainAxisAlignment.spaceEvenly,
      crossAxisAlignment: CrossAxisAlignment.end,
      mainAxisSize: MainAxisSize.max,
      children: <Widget>[
        Container(color: Colors.red, width: 60, height: 60),
        Container(color: Colors.blue, width: 80, height: 80),
        Container(color: Colors.green, width: 70, height: 70),
        Container(color: Colors.orange, width: 100, height: 100)
,
      ],
    );
  }
}
```

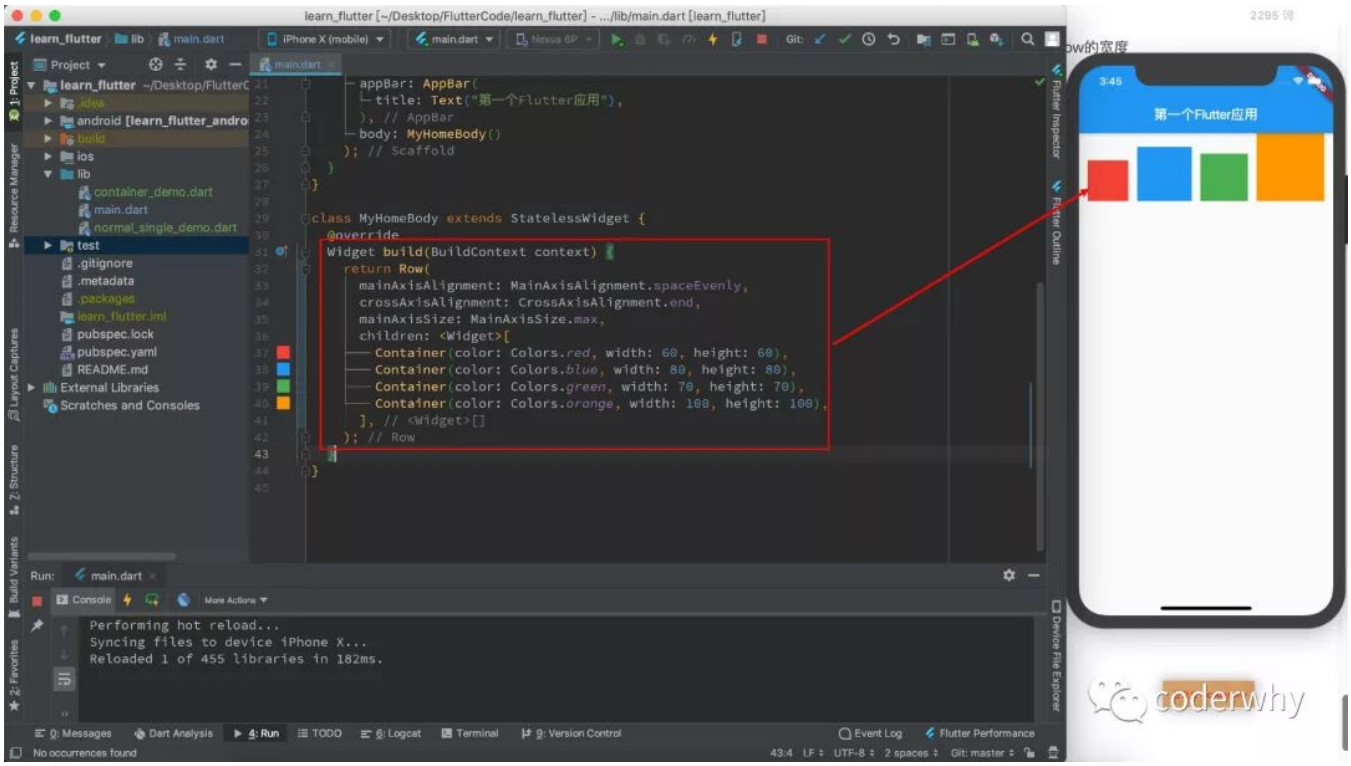


image-20190903154614672

2.1.3. mainAxisSize

默认情况下，Row会尽可能占据多的宽度，让子Widget在其中进行排布，这是因为mainAxisSize属性默认值是 `MainAxisSize.max` 。

我们来看一下，如果这个值被修改为 `MainAxisSize.max` 会什么变化：

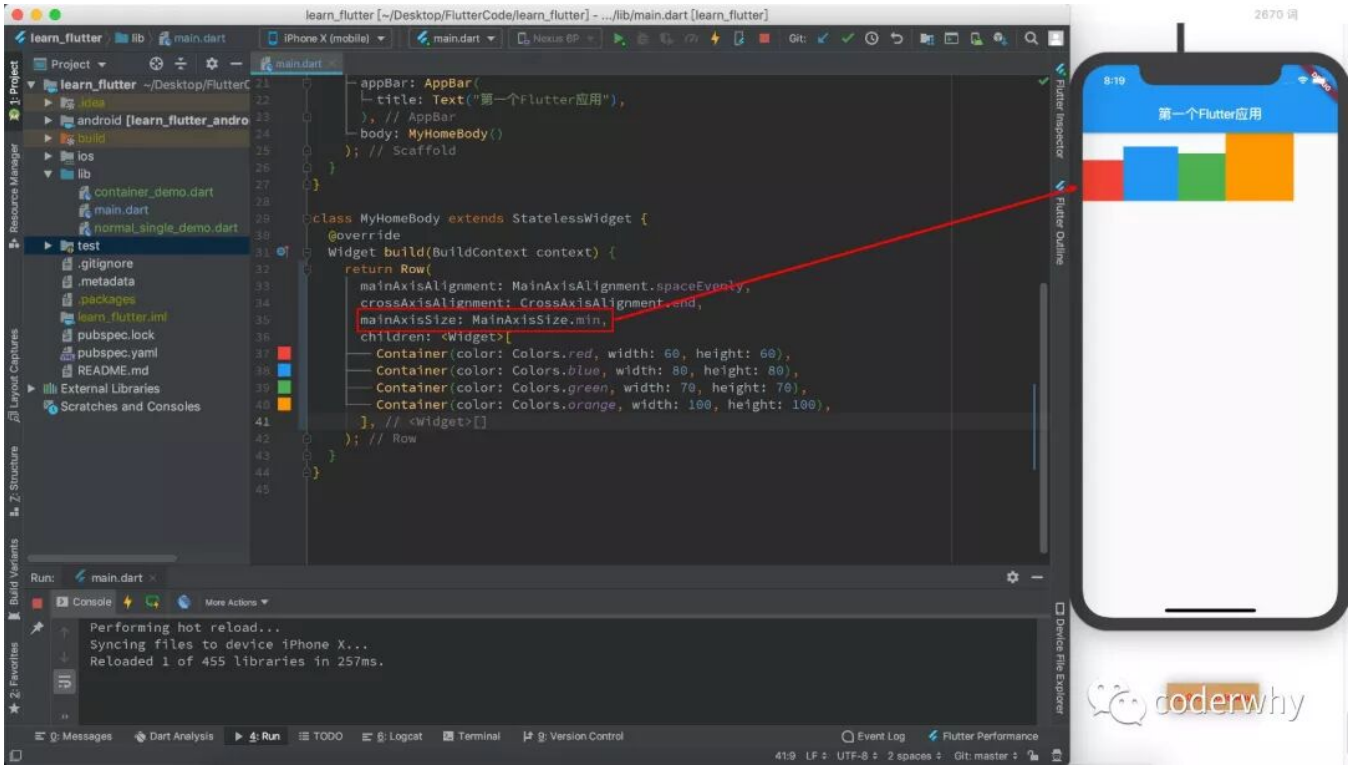


image-20190903201957558

2.1.4. TextBaseline

关于TextBaseline的取值解析

ideographic alphabetic

2.1.5. Expanded

如果我们希望红色和黄色的Container Widget不要设置固定的宽度，而是占据剩余的部分，这个时候应该如何处理呢?

这个时候我们可以使用 `Expanded` 来包裹 Container Widget，并且将它的宽度不设置值；

- flex属性，弹性系数，Row会根据两个Expanded的弹性系数来决定它们占据剩下空间的比例

```
class MyHomeBody extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Row(
      mainAxisAlignment: MainAxisAlignment.spaceEvenly,
      crossAxisAlignment: CrossAxisAlignment.end,
      mainAxisSize: MainAxisSize.min,
      children: <Widget>[
        Expanded(
          flex: 1,
          child: Container(color: Colors.red, height: 60),
        ),
        Container(color: Colors.blue, width: 80, height: 80),
        Container(color: Colors.green, width: 70, height: 70)
,
        Expanded(
          flex: 1,
          child: Container(color: Colors.orange, height: 100)
,
        )
      ],
    );
  }
}
```

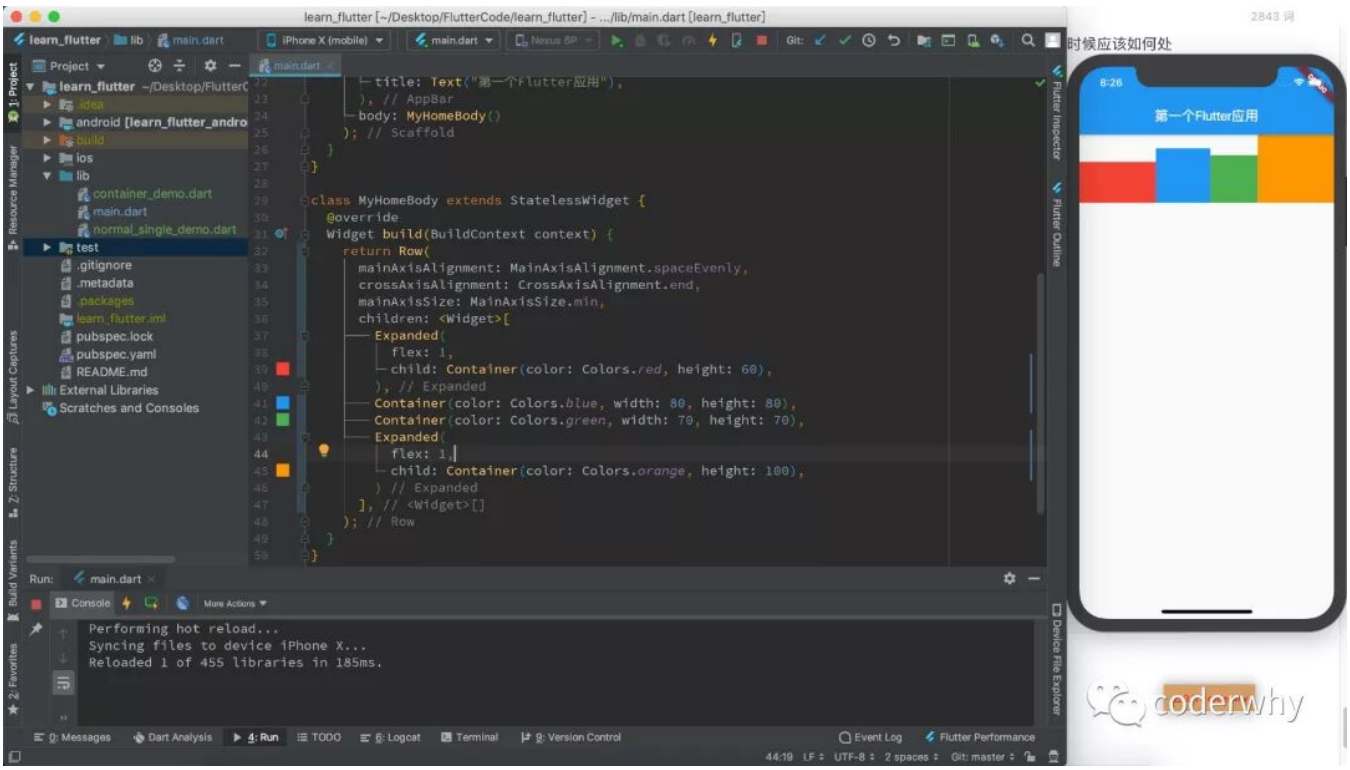


image-20190903202611133

2.2. Column组件

> Column组件用于将所有的子Widget排成一列，学会了前面的Row后，Column只是和row的方向不同而已。

2.2.1. Column介绍

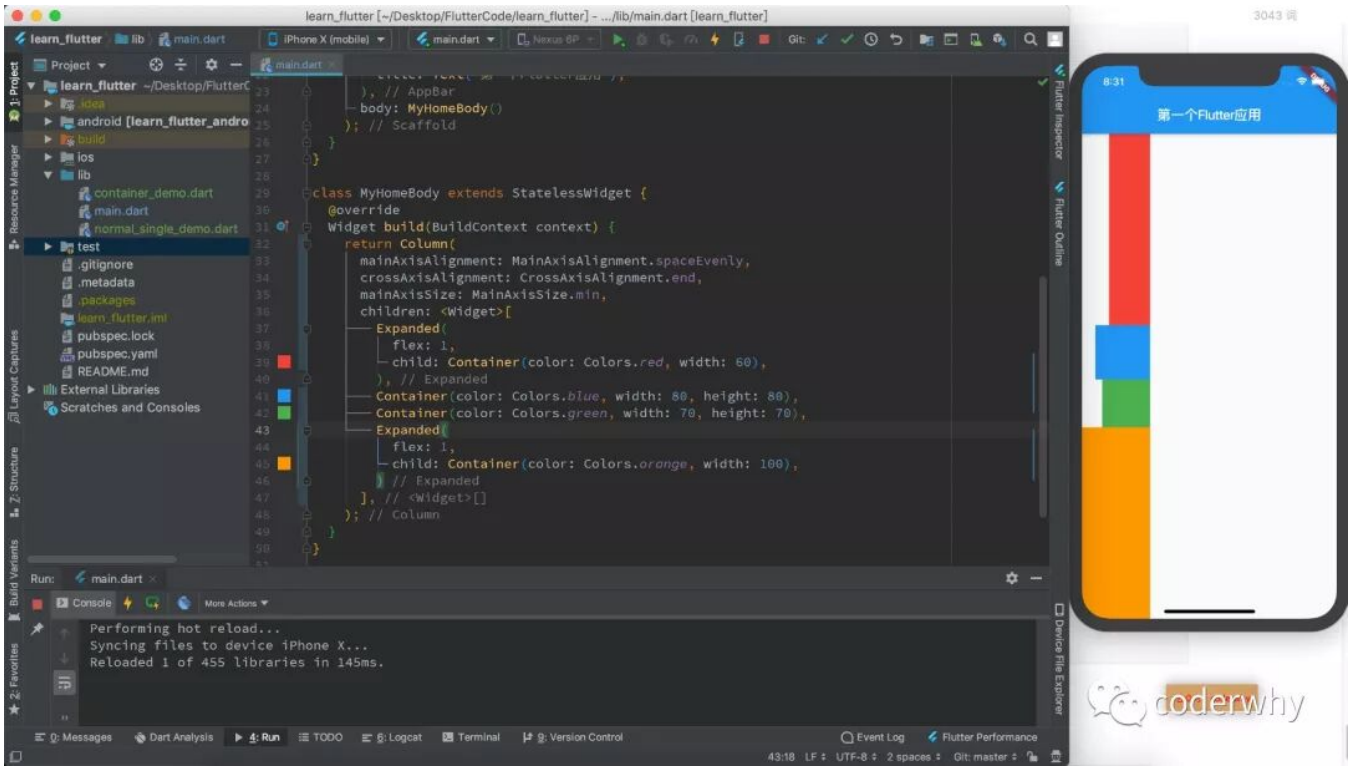
我们直接看它的源码：我们发现和Row属性是一致的，不再解释

```
Column({
  Key key,
  MainAxisAlignment mainAxisAlignment = MainAxisAlignment.start,
  MainAxisSize mainAxisSize = MainAxisSize.max,
  CrossAxisAlignment crossAxisAlignment = CrossAxisAlignment.center,
  TextDirection textDirection,
  VerticalDirection verticalDirection = VerticalDirection.down,
  TextBaseline textBaseline,
  List<Widget> children = const <Widget>[],
})
```

2.2.2. Column演练

我们直接将Row的代码中Row改为Column，查看代码运行效果：

```
class MyHomeBody extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Column(
      mainAxisAlignment: MainAxisAlignment.spaceEvenly,
      crossAxisAlignment: CrossAxisAlignment.end,
      mainAxisSize: MainAxisSize.min,
      children: <Widget>[
        Expanded(
          flex: 1,
          child: Container(color: Colors.red, width: 60),
        ),
        Container(color: Colors.blue, width: 80, height: 80),
        Container(color: Colors.green, width: 70, height: 70)
,
        Expanded(
          flex: 1,
          child: Container(color: Colors.orange, width: 100),
        )
      ],
    );
  }
}
```



2.3. Stack组件

> 在开发中，我们多个组件很有可能需要重叠显示，比如在一张图片上显示文字或者一个按钮等。
>
> 在Android中可以使用Frame来实现，在Web端可以使用绝对定位，在Flutter中我们需要使用层叠布局Stack。

2.3.1. Stack介绍

我们还是通过源码来看一下Stack有哪些属性：

```
Stack({
  Key key,
  this.alignment = AlignmentDirectional.topStart,
  this.textDirection,
  this.fit = StackFit.loose,
  this.overflow = Overflow.clip,
  List<Widget> children = const <Widget>[],
})
```

参数解析：

- alignment：此参数决定如何去对齐没有定位（没有使用Positioned）或部分定位的子widget。所谓部分定位，在这里**特指没有在某一个轴上定位：**left、right为横轴，top、bottom为纵轴，只要包含某个轴上的一个定位属性就算在该轴上有定位。
- textDirection：和Row、Wrap的textDirection功能一样，都用于决定alignment对齐的参考系即：textDirection的值为 `TextDirection.ltr` ，则alignment的 `start` 代表左， `end` 代表右；textDirection的值为 `TextDirection.rtl` ，则alignment的 `start` 代表右， `end` 代表左。
- fit：此参数用于决定 **没有定位** 的子widget如何去适应Stack的大小。 `StackFit.loose` 表示使用子widget的大小， `StackFit.expand` 表示扩伸到Stack的大小。
- overflow：此属性决定如何显示超出Stack显示空间的子widget，值为 `Overflow.clip` 时，超出部分会被剪裁（隐藏），值为 `Overflow.visible` 时则不会。

2.3.2. Stack演练

Stack会经常和Positioned一起来使用，Positioned可以决定组件在Stack中的位置，用于实现类似于Web中的绝对定位效果。

我们来看一个简单的演练：

```
class MyHomeBody extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Stack(
      children: <Widget>[
        Container(
          color: Colors.purple,
          width: 300,
          height: 300,
        ),
        Positioned(
          left: 20,
          top: 20,
          child: Icon(Icons.favorite, size: 50, color: Colors.white)
        ),
        Positioned(
          bottom: 20,
          right: 20,
          child: Text("你好啊，李银河", style: TextStyle(fontSize: 20, color: Colors.white)),
        )
      ],
    );
  }
}
```

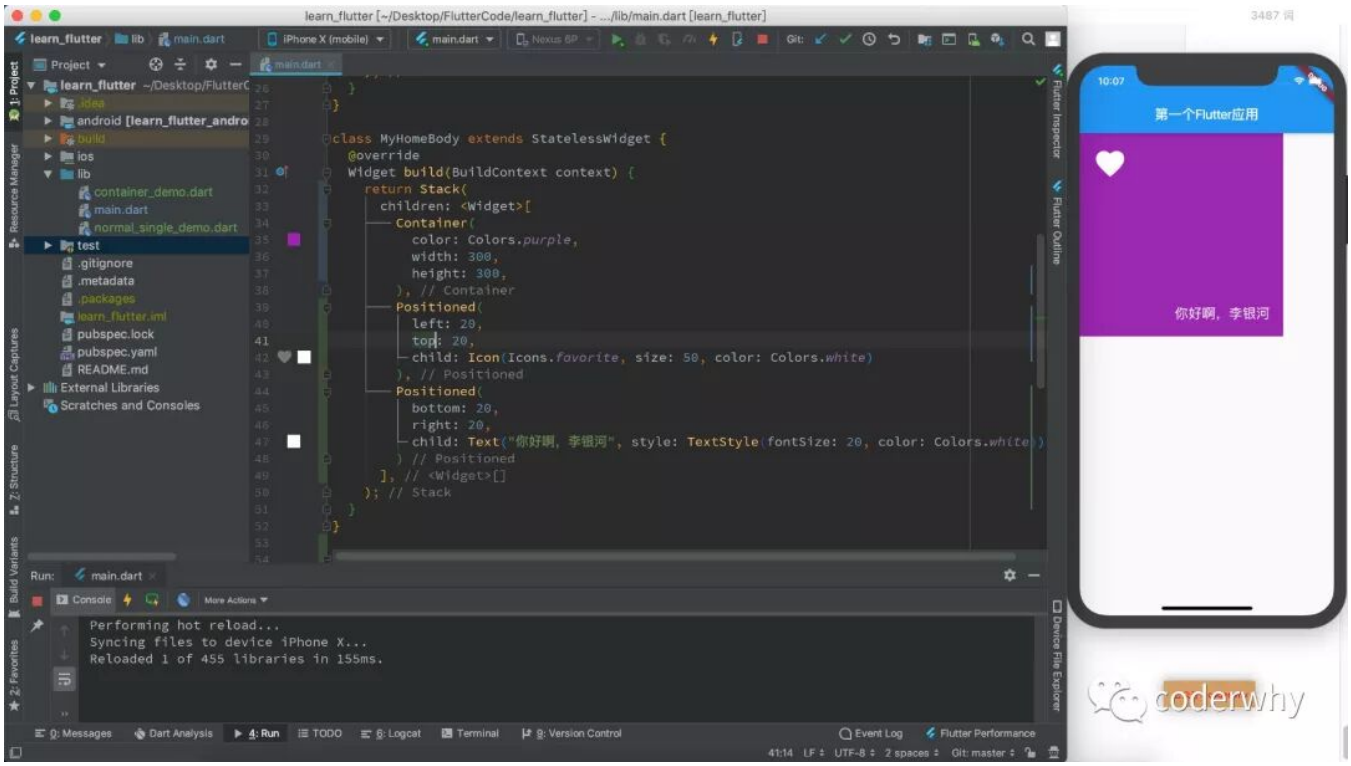


image-20190903220757633

注意：Positioned组件只能在Stack中使用。

> 备注：所有内容首发于公众号，之后除了Flutter也会更新其他技术文章，TypeScript、React、Node、uniapp、mpvue、数据结构与算法等等，也会更新一些自己的学习心得等，欢迎大家关注

coderwhy

微信扫描二维码，关注我的公众号

公众号

参考资料

[1] Flutter布局组件: *https://flutter.dev/docs/development/ui/widgets/layout*