Flutter(一)之Flutter开发初体验

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。

一. 创建Flutter项目

> 创建Flutter项目有两种方式： `通过命令行创建` 和 `通过开发工具创建`

1.1. 通过命令行创建

通过命令行创建非常简单，在终端输入以下命令即可：

- **注意：**Flutter的名称不要包含特殊的字符，另外不可以使用驼峰标识
- 创建完之后使用自己喜欢的开发工具打开即可

```
flutter create learn_flutter
```

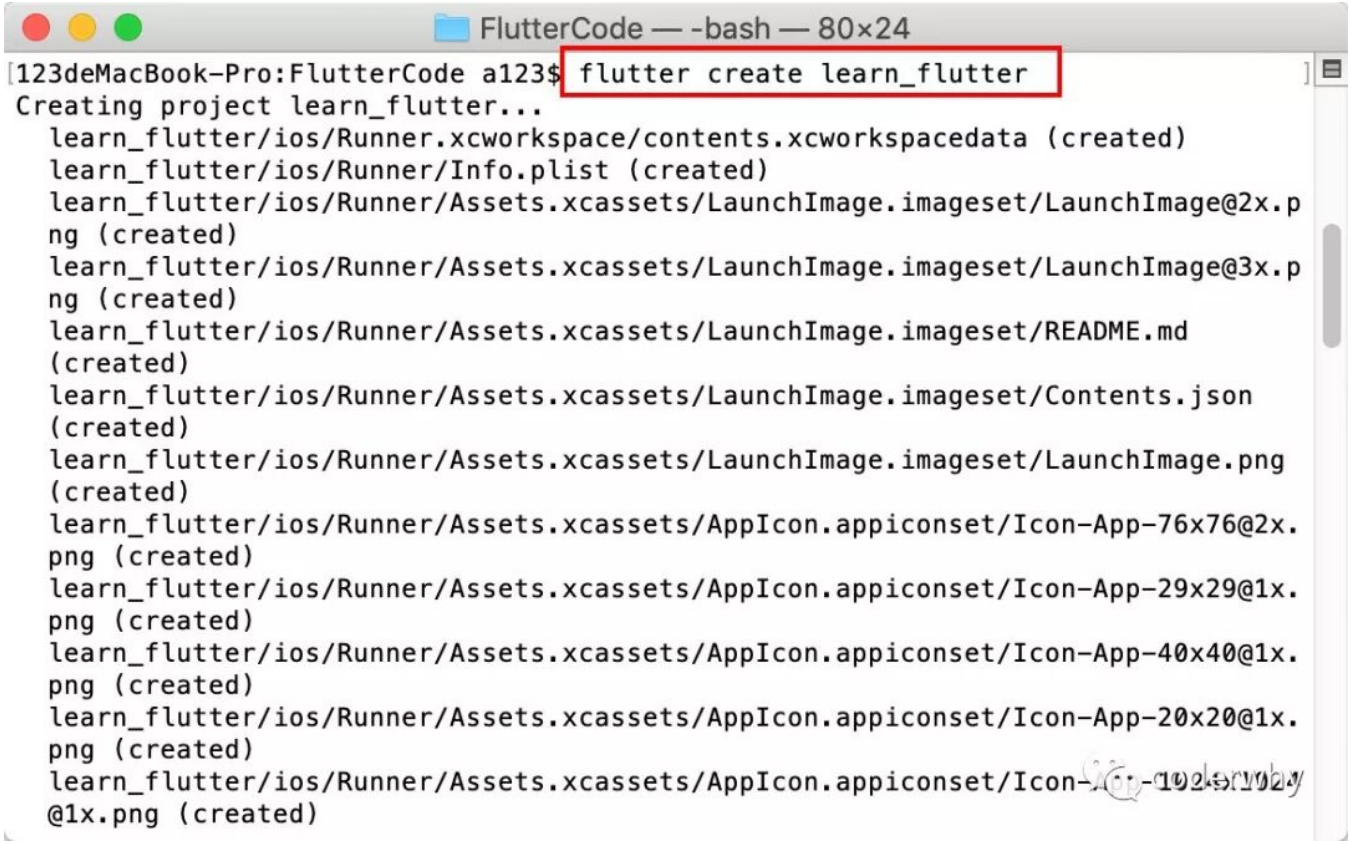

image-20190915164546394

1.2. 通过开发工具创建

我这里也可以直接通过 **Android Studio** 来进行创建：

- 选择 `Start a new Flutter project`，之后填写相关的信息即可，这里不再赘述

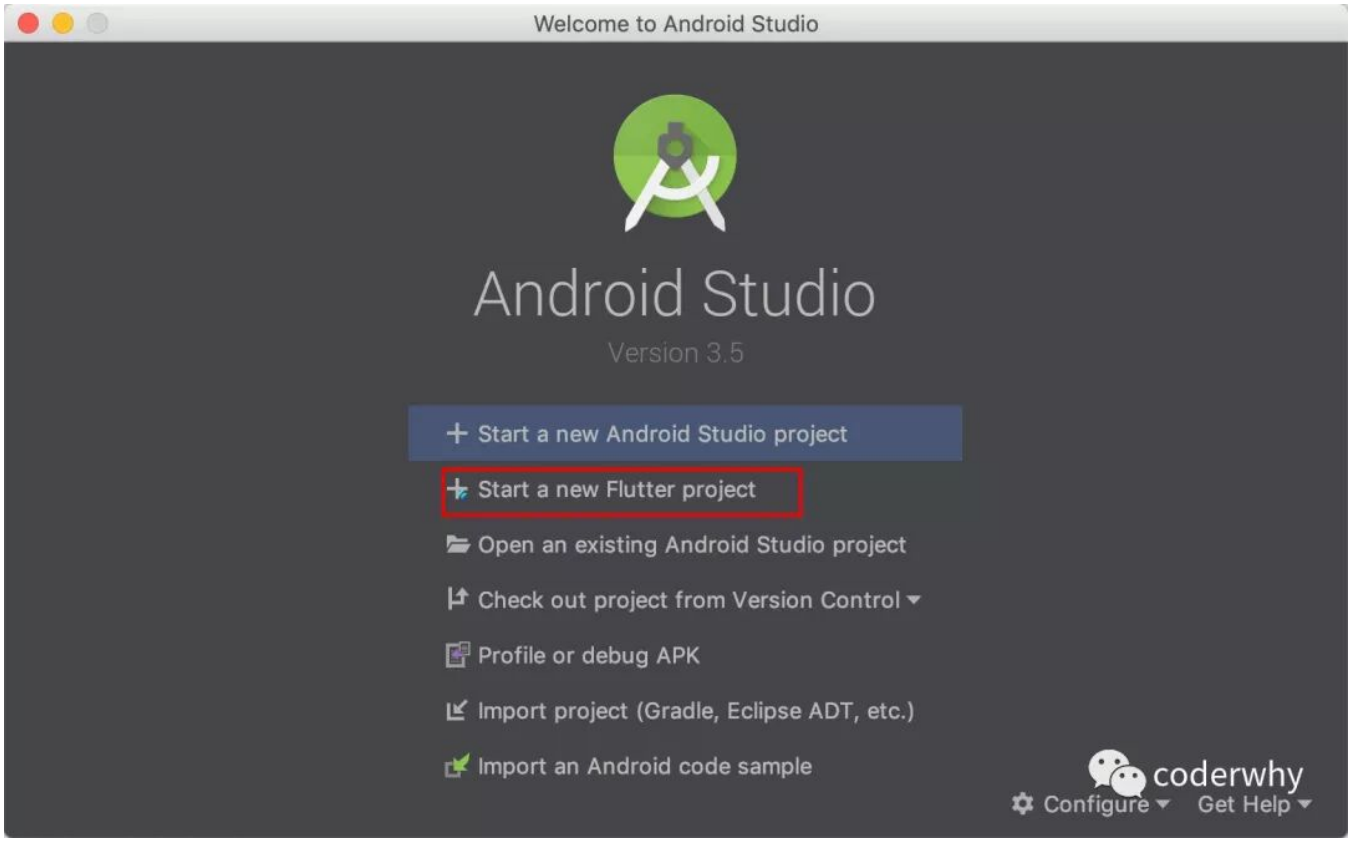

image-20190901200434719

1.3. 默认程序分析

我们讲创建的应用起来跑在模拟器上（我这里选择iPhone模拟器，Android也可以），会看到如下效果：

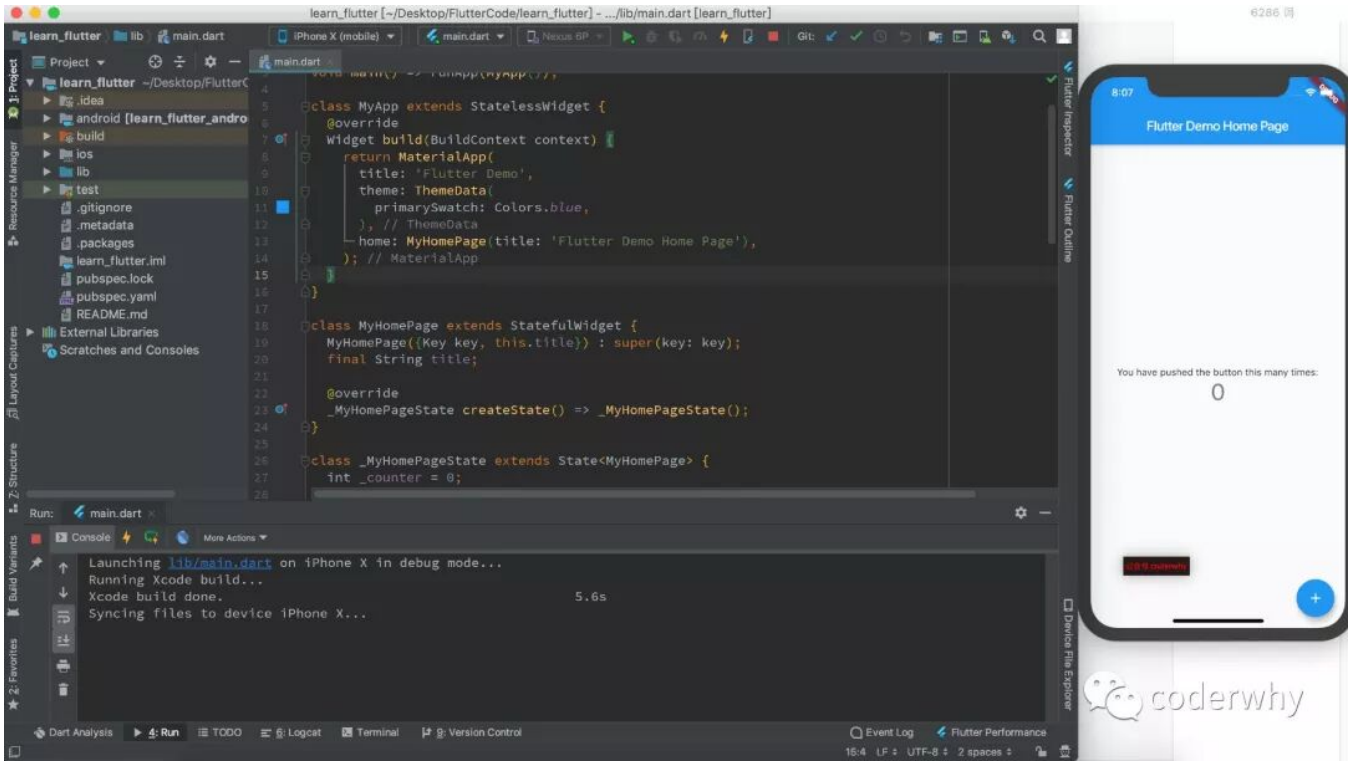

image-20190901200718627

默认项目分析：

- 我们之前已经分析过目录结构了，在目录下有一个 `lib` 文件夹，里面会存放我们编写的Flutter代码；
- 打开发现里面有一个 `main.dart` ，它是我们Flutter启动的 `入口文件` ，里面有 `main函数` ；

默认代码分析：

- 这是一个计数器的案例程序，点击右下角的 `+` 符号，上面显示的数字会递增；
- 但是我们第一次接触main.dart中的代码，可能会发现很多 `不认识` 的代码，不知道这个内容是如何编写出来的；

作为初学者，我的建议是将其中所有的代码全部删除掉，从零去创建里面的代码，这样我们才能对Flutter应用程序的结构非常清晰；

二. 开始Flutter代码

2.1. Hello World

2.1.1. Hello World的需求

做任何的开发，我们都是从祖传的 `Hello World` 开始，那么现在我们的需求来了：

- 在界面中心位置，显示一个Hello World；

2.1.2. Hello World的实现

下面，我们就动手开始编写Hello World：

```
import 'package:flutter/material.dart';

main(List<String> args) {
  runApp(Text("Hello World", textDirection: TextDirection.ltr));
}
```

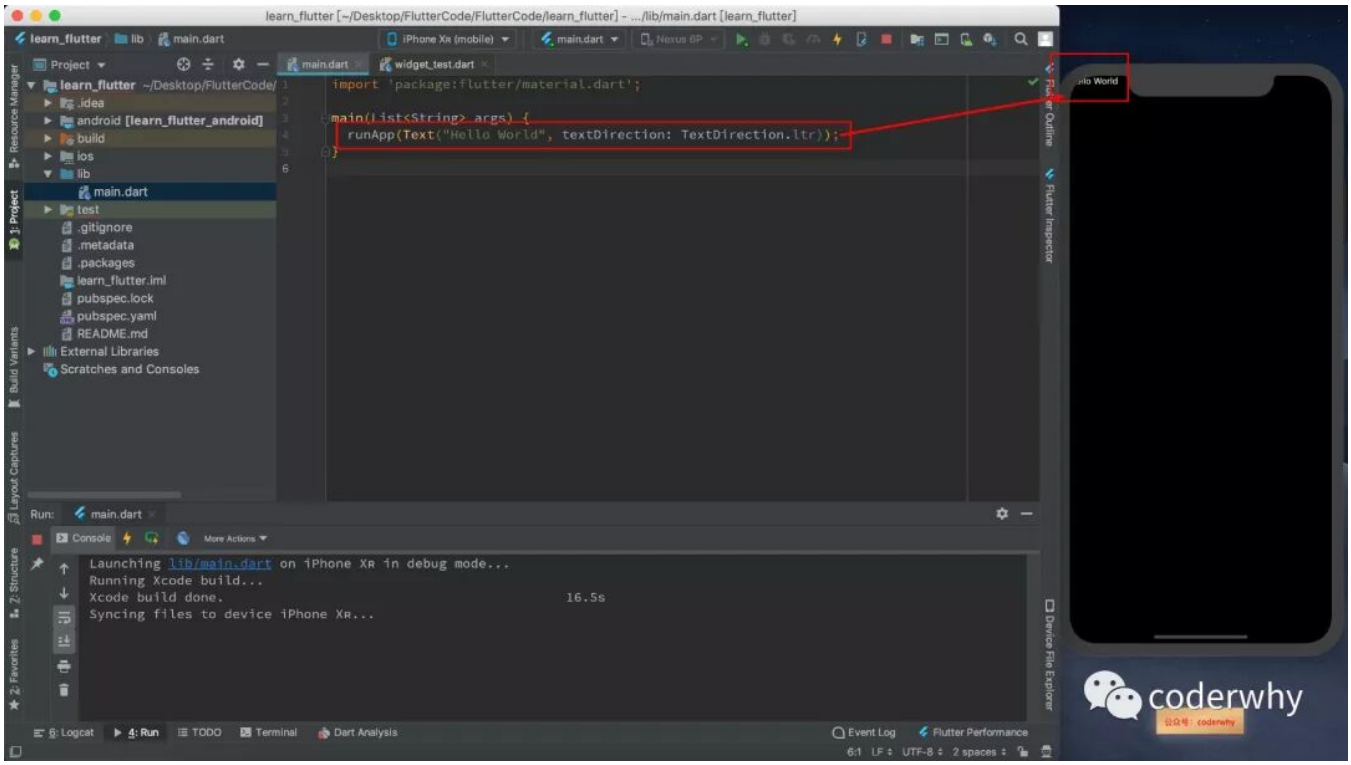

image-20190916212127281

当然，上面的代码我们已经实现了在界面上显示Hello World：

- 但是 `没有居中` ， `字体也有点小` ；
- 这些问题，我们放到后面再来解决，先搞懂目前的几行代码；

上面的代码我们有一些比较熟悉，有一些并不清楚是什么：

- 比如我们知道Dart程序的 `入口都是main函数` ，而Flutter是Dart编写的，所以入口也是main函数；
- 但是我们导入的 `Material是什么呢` ？
- 另外，我们在main函数中调用了一个 `runApp()函数` 又是什么呢?

下面，我们对不认识的代码进行一些分析。

2.2. 代码分析

2.2.1. runApp和Widget

runApp 是Flutter内部提供的一个函数，当我们启动一个Flutter应用程序时就是从调用这个函数开始的

- 我们可以点到runApp的源码，查看到该函数
- 我们暂时不分析具体的源码（因为我发现过多的理论，对于初学者来说并不友好）

```
void runApp(Widget app) {
  ...省略代码
}
```

该函数让我们传入一个东西： **Widget** ？

我们先说 **Widget的翻译** ：

- Widget在国内有很多的翻译；
- 做过Android、iOS等开发的人群，喜欢将它翻译成 **控件** ；
- 做过Vue、React等开发的人群，喜欢将它翻译成 **组件** ；
- 如果我们使用Google，Widget翻译过来应该是 **小部件** ；
- 没有说哪种翻译一定是对的，或者一定是错的，但是我个人更倾向于 **小部件或者组件** ；

Widget 到底什么东西呢?

- 我们学习Flutter，从一开始就可以有一个基本的认识： **Flutter中万物皆Widget（万物皆可盘）** ；
- 在我们iOS或者Android开发中，我们的界面有很多种类的划分：应用（Application）、视图控制器（View Controller）、活动（Activity）、View（视图）、Button（按钮）等等；
- 但是在Flutter中， **这些东西都是不同的Widget而已** ；
- 也就是我们整个应用程序中 `所看到的内容` 几乎都是Widget，甚至是 `内边距的设置` ，我们也需要使用一个叫 `Padding的Widget` 来做；

runApp函数让我们传入的就是一个Widget：

- 但是我们现在没有Widget，怎么办呢?
- 我们可以导入Flutter默认已经给我们提供的Material库，来使用其中的很多内置Widget；

2.2.2. Material设计风格

material是什么呢?

- material是Google公司推行的一套 `设计风格` ，或者叫 `设计语言` 、 `设计规范` 等；
- 里面有非常多的设计规范，比如 `颜色` 、 `文字的排版` 、 `响应动画与过度` 、 `填充` 等等；
- 在Flutter中高度集成了 `Material风格的Widget` ；
- 在我们的应用中，我们可以直接使用这些Widget来创建我们的应用（后面会用到很多）；

Text小部件分析：

- 我们可以使用Text小部件来完成文字的显示；
- 我们发现Text小部件继承自StatelessWidget，StatelessWidget继承自Widget；
- 所以我们可以将Text小部件传入到runApp函数中
- 属性非常多，但是我们已经学习了Dart语法，所以你会发现只有this.data属性是必须传入的。

```
class Text extends StatelessWidget {
  const Text(
    this.data, {
    Key key,
    this.style,
    this.strutStyle,
    this.textAlign,
    this.textDirection,
    this.locale,
    this.softWrap,
    this.overflow,
    this.textScaleFactor,
    this.maxLines,
    this.semanticsLabel,
    this.textWidthBasis,
  });
}
```

StatelessWidget简单介绍：

- StatelessWidget继承自Widget；
- 后面我会更加详细的介绍它的用法；

```
abstract class StatelessWidget extends Widget {
 // ...省略代码
}
```

2.3. 代码改进

2.3.1. 改进界面样式

我们发现现在的代码并不是我们想要的最终结果：

- 我们可能希望文字居中显示，并且可以大一些；
- **居中显示：** 需要使用另外一个Widget， `Center` ；
- **文字大一些：** 需要给Text文本设置一些样式；

我们修改代码如下：

- 我们在Text小部件外层包装了一个Center部件，让Text作为其child；
- 并且，我们给Text组件设置了一个属性：style，对应的值是TextStyle类型；

```
import 'package:flutter/material.dart';

main(List<String> args) {
  runApp(
    Center(
      child: Text(
        "Hello World",
        textDirection: TextDirection.ltr,
        style: TextStyle(fontSize: 36),
      ),
    )
  );
}
```

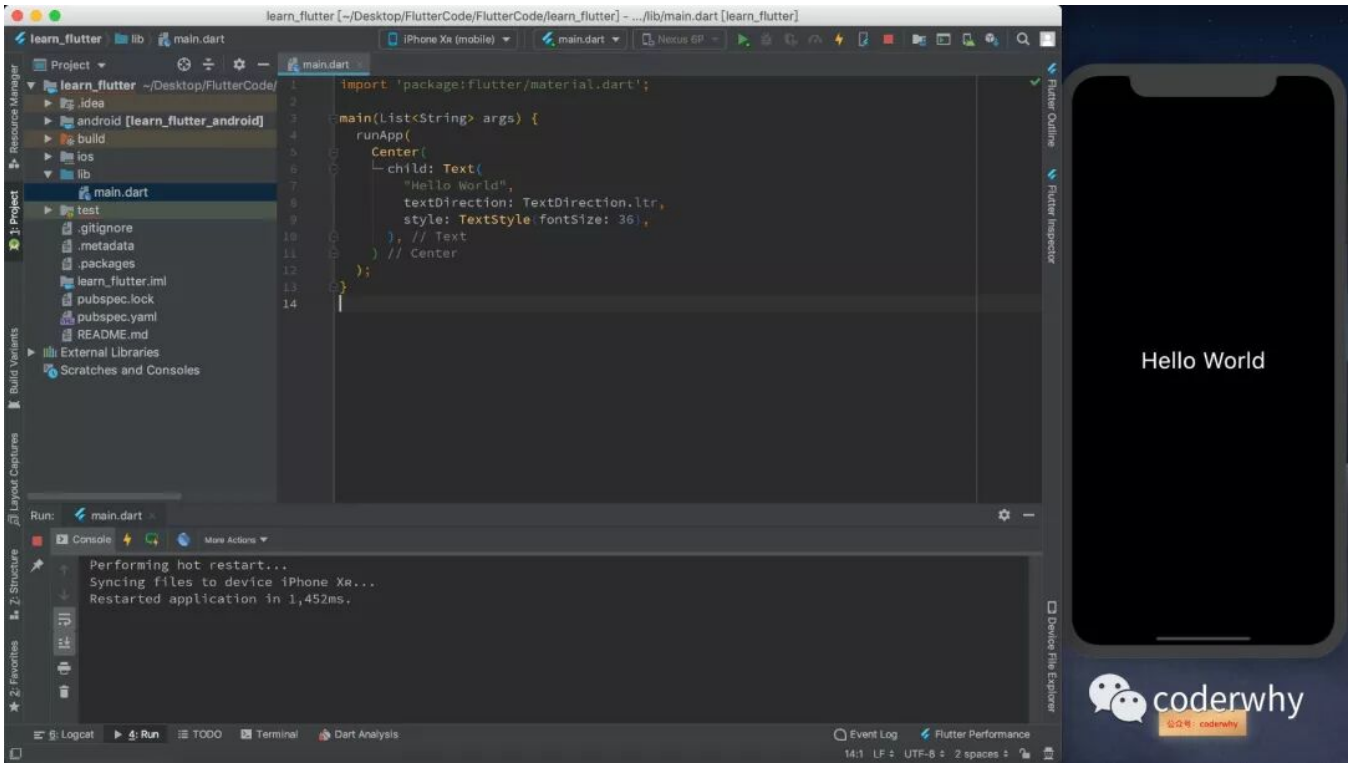

image-20190916215857058

2.3.2. 改进界面结构

目前我们虽然可以显示HelloWorld，但是我们发现最底部的背景是黑色，并且我们的页面并不够结构化。

- 正常的App页面应该有一定的结构，比如通常都会有 `导航栏` ，会有一些 `背景颜色` 等

在开发当中，我们并不需要从零去搭建这种结构化的界面，我们可以使用Material库，直接使用其中的一些封装好的组件来完成一些结构的搭建。

我们通过下面的代码来实现：

```
import 'package:flutter/material.dart';

main(List<String> args) {
  runApp(
    MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text("CODERWHY"),
        ),
        body: Center(
          child: Text(
            "Hello World",
            textDirection: TextDirection.ltr,
            style: TextStyle(fontSize: 36),
          ),
        ),
      ),
    )
  );
}
```

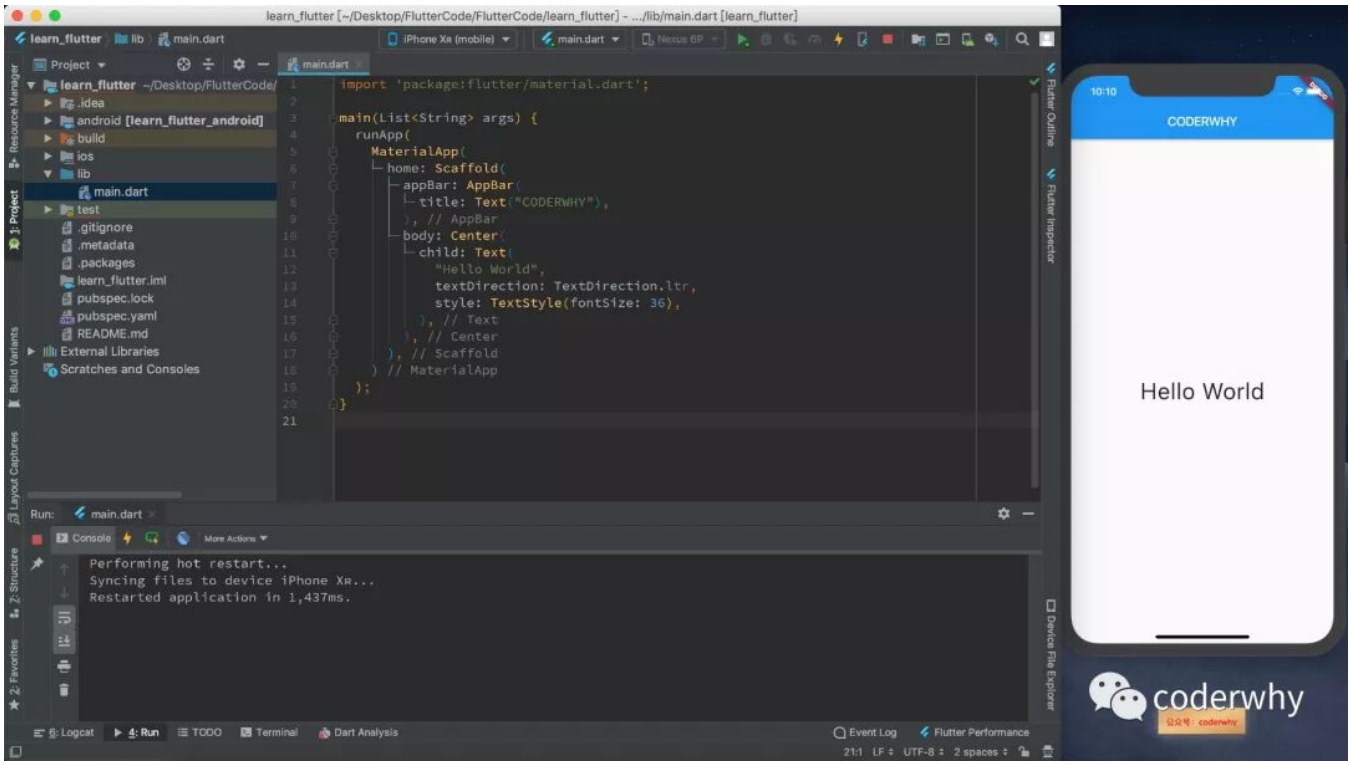

image-20190916221014543

在最外层包裹一个 **MaterialApp**

- 这意味着整个应用我们都会采用MaterialApp风格的一些东西，方便我们对应用的设计，并且目前我们使用了其中两个属性；

- title：这个是定义在Android系统中打开多任务切换窗口时显示的标题；（暂时可以不写）
- home：是该应用启动时显示的页面，我们传入了一个Scaffold；

Scaffold 是什么呢?

- 翻译过来是 `脚手架` ，脚手架的作用就是搭建页面的基本结构；
- 所以我们给MaterialApp的home属性传入了一个Scaffold对象，作为启动显示的Widget；
- Scaffold也有一些属性，比如 `appBar` 和 `body` ；
- appBar是用于设计导航栏的，我们传入了一个 `title属性` ；
- body是页面的内容部分，我们传入了之前已经创建好的Center中包裹的一个Text的Widget；

2.3.3. 进阶案例实现

我们可以让界面中存在更多的元素：

- 写到这里的时候，你可能已经发现 `嵌套太多` 了，不要着急，我们后面会对代码重构的

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
```

```dart
import 'package:flutter/material.dart';

main(List<String> args) {
  runApp(
    MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text("CODERWHY"),
        ),
        body: Center(
          child: Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: <Widget>[
              Checkbox(
                value: true,
                onChanged: (value) => print("Hello World")),
              Text(
                "同意协议",
                textDirection: TextDirection.ltr,
                style: TextStyle(fontSize: 20),
              )
            ],
          ),
        ),
      ),
    )
  );
}
```

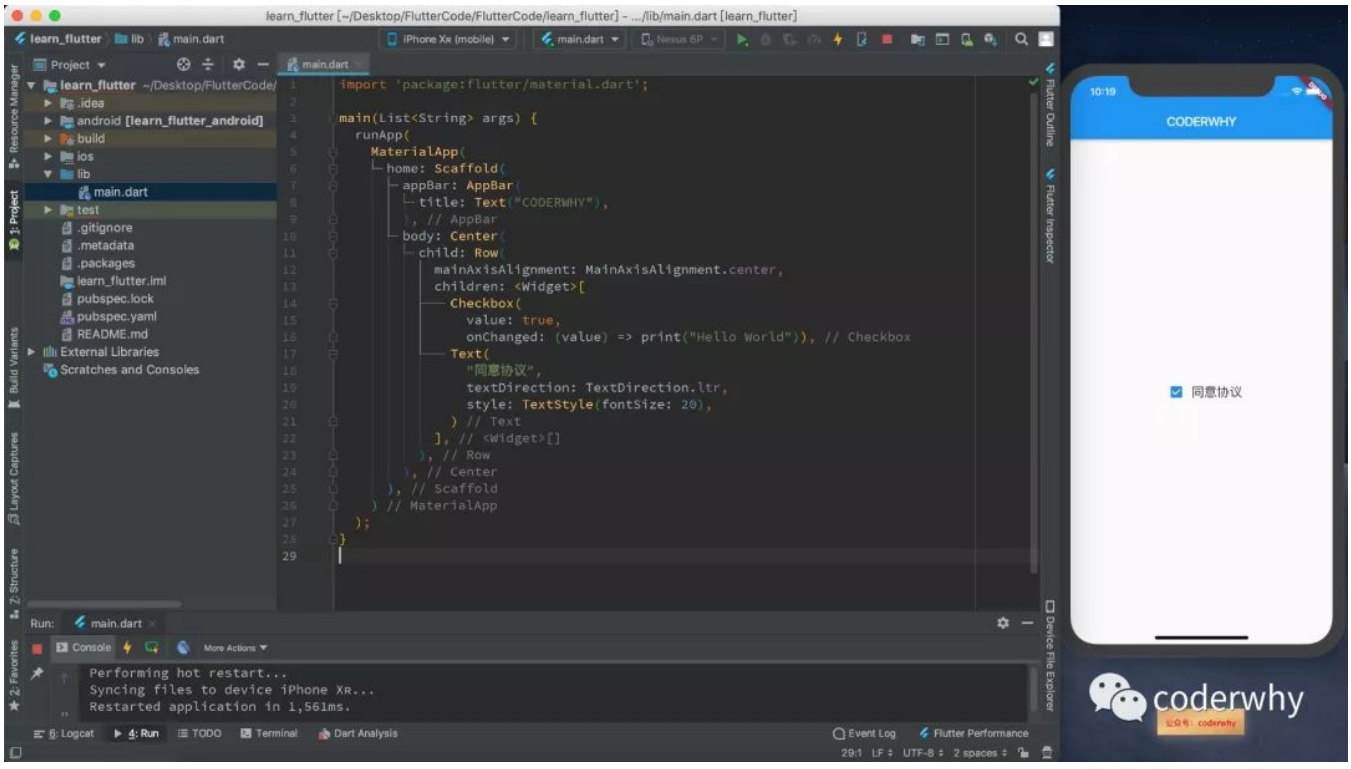

image-20190916221925380

2.4. 代码重构

2.4.1. 创建自己的Widget

很多学习Flutter的人，都会被Flutter的 嵌套 劝退，当代码嵌套过多时，结构很容易看不清晰。

这里有两点我先说明一下：

- 1、Flutter整个开发过程中就是形成一个Widget树，所以形成嵌套是很正常的。
- 2、关于Flutter的代码缩进，更多开发中我们使用的是2个空格（前端开发2个空格居多，你喜欢4个也没问题）

但是，我们开发一个这么简单的程序就出现如此多的嵌套，如果应用程序更复杂呢?

- 我们可以对我们的代码进行封装，将它们封装到自己的Widget中，创建自己的Widget；

如何创建自己的Widget呢?

- 在Flutter开发中，我们可以继承自StatelessWidget或者StatefulWidget来创建自己的Widget类；
- **StatelessWidget：** 没有状态改变的Widget，通常这种Widget仅仅是做一些展示工作而已；
- **StatefulWidget：** 需要保存状态，并且可能出现状态改变的Widget；

在上面的案例中对代码的重构，我们使用StatelessWidget即可，所以我们接下来学习一下如果利用StatelessWidget来对我们的代码进行重构；

StatefulWidget我们放到后面的一个案例中来学习；

2.4.2. StatelessWidget

StatelessWidget通常是一些没有状态（State，也可以理解成data）需要维护的Widget：

- 它们的数据通常是直接写死（放在Widget中的数据，必须被定义为final，为什么呢？我在下一个章节讲解StatefulWidget会讲到）；
- 从parent widget中传入的而且一旦传入就不可以修改；
- 从InheritedWidget获取来使用的数据（这个放到后面会讲解）；

我们来看一下创建一个StatelessWidget的格式：

- 1、让自己创建的Widget继承自StatelessWidget；
- 2、StatelessWidget包含一个必须重写的方法：build方法；

```
class MyStatelessWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return <返回我们的Widget要渲染的Widget，比如一个Text Widget>;
  }
}
```

build方法的解析：

- Flutter在拿到我们自己创建的StatelessWidget时，就会执行它的build方法；
- 我们需要在build方法中告诉Flutter，我们的Widget希望渲染什么元素，比如一个Text Widget；
- StatelessWidget没办法主动去执行build方法，当我们使用的数据发生改变时，build方法会被重新执行；

build方法什么情况下被执行呢？：

- 1、当我们的StatelessWidget第一次被插入到Widget树中时（也就是第一次被创建时）；
- 2、当我们的父Widget（parent widget）发生改变时，子Widget会被重新构建；
- 3、如果我们的Widget依赖InheritedWidget的一些数据，InheritedWidget数据发生改变时；

2.4.3. 重构案例代码

现在我们就可以通过StatelessWidget来对我们的代码进行重构了

- 因为我们的整个代码都是一些数据展示，没有数据的改变，使用StatelessWidget即可；
- 另外，为了体现更好的封装性，我对代码进行了两层的拆分，让代码结构看起来更加清晰；（具体的拆分方式，我会在后面的案例中不断的体现出来，目前我们先拆分两层）

重构后的代码如下：

```dart
import 'package:flutter/material.dart';

main(List<String> args) {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text("CODERWHY"),
        ),
        body: HomeContent(),
      ),
    )
  }
}

class HomeContent extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Center(
      child: Row(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          Checkbox(
              value: true,
              onChanged: (value) => print("Hello World")),
          Text(
            "同意协议",
            textDirection: TextDirection.ltr,
            style: TextStyle(fontSize: 20),
          )
        ],
      ),
    );
  }
}
```

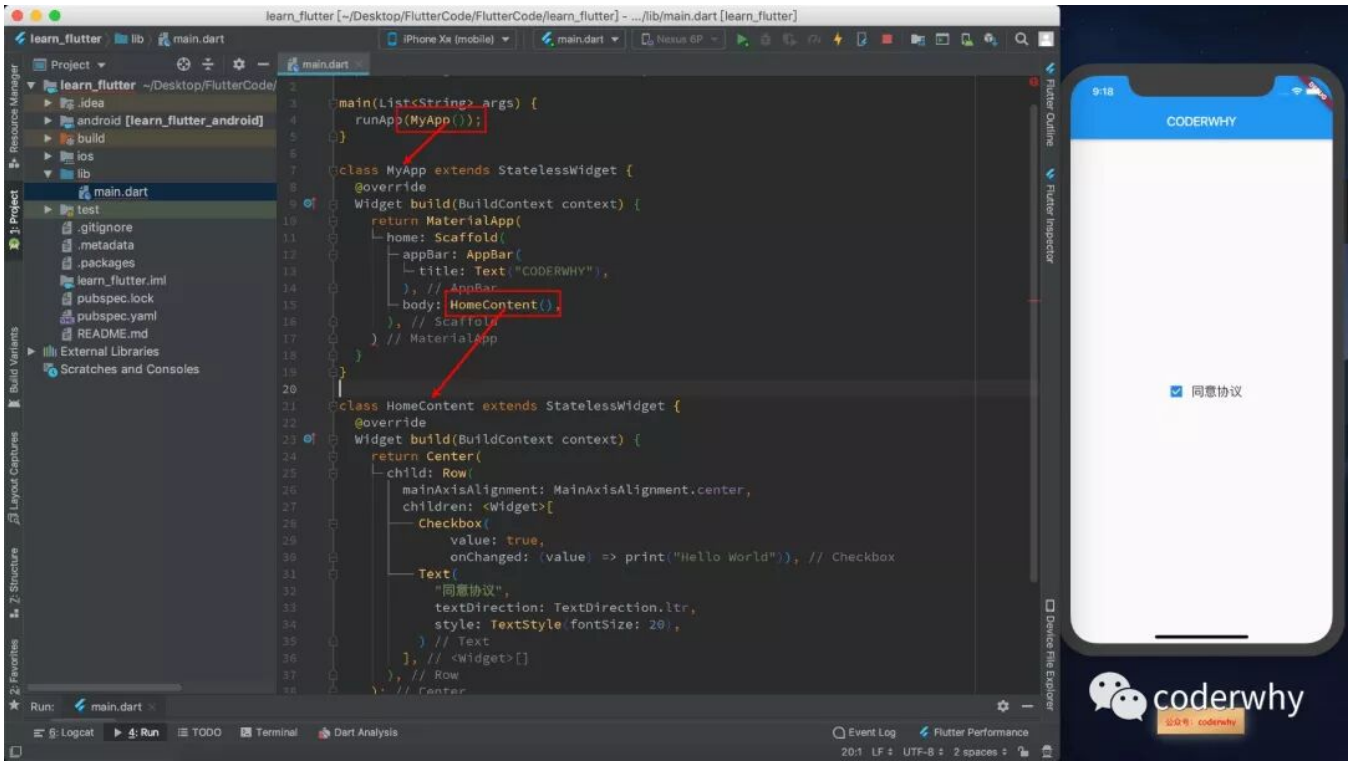

image-20190917091913208

三. 案例练习

3.1. 案例最终效果

我们先来看一下案例的最终展示效果：

- 这个效果中我们会使用很多没有接触的Widget；
- 没有关系，后面这些常用的Widget我会一个个讲解；
- 这个案例最主要的目的还是让大家更加熟悉Flutter的开发模式以及自定义Widget的封装过程；

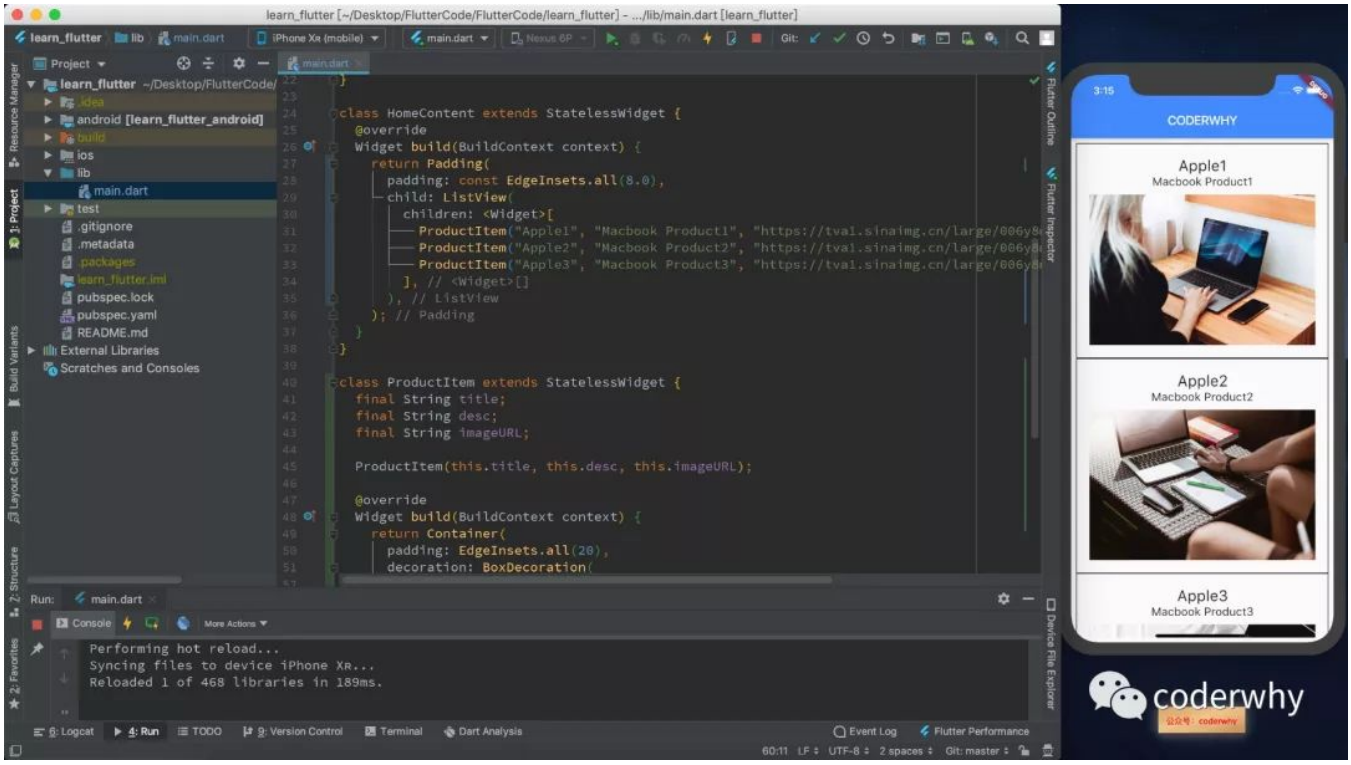

image-20190917151554241

3.2. 自定义Widget

在我们的案例中，很明显一个产品的展示就是一个大的Widget，这个Widget包含如下Widget：

- 标题的Widget：使用一个Text Widget完成；

- 描述的Widget：使用一个Text Widget完成；
- 图片的Widget：使用一个Image Widget完成；
- 上面三个Widget要垂直排列，我们可以使用一个Column的Widget（上一个章节中我们使用了一次Row是水平排列的）

另外，三个展示的标题、描述、图片都是不一样的，所以我们可以让Parent Widget来决定内容：

- 创建三个成员变量保存父Widget传入的数据

```
class ProductItem extends StatelessWidget {
  final String title;
  final String desc;
  final String imageURL;

  ProductItem(this.title, this.desc, this.imageURL);

  @override
  Widget build(BuildContext context) {
    return Column(
      children: <Widget>[
        Text(title, style: TextStyle(fontSize: 24)),
        Text(desc, style: TextStyle(fontSize: 18)),
        Image.network(imageURL)
      ],
    );
  }
}
```

3.3. 列表数据展示

现在我们就可以创建三个ProductItem来让他们展示了：

- MyApp和上一个章节是一致的，没有任何改变；
- HomeContent中，我们使用了一个Column，因为我们创建的三个ProductItem是垂直排列的

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      theme: ThemeData(
        primaryColor: Colors.blueAccent
      ),
      home: Scaffold(
        appBar: AppBar(
          title: Text("CODERWHY"),
        ),
        body: HomeContent(),
      ),
    );
  }
}

class HomeContent extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Column(
      children: <Widget>[
        ProductItem("Apple1", "Macbook Product1", "https://tva1.sinaimg.cn/large/006y8mN6gy1g72j6nk1d4j30u00k0n0j.jpg"),
        ProductItem("Apple2", "Macbook Product2", "https://tva1.sinaimg.cn/large/006y8mN6gy1g72imm9u5zj30u00k0adf.jpg"),
        ProductItem("Apple3", "Macbook Product3", "https://tva1.sinaimg.cn/large/006y8mN6gy1g72imqlouhj30u00k00v0.jpg"),
      ],
    );
  }
}
```

运行效果如下：

- 错误信息：下面出现了黄色的斑马线；
- 这是因为在Flutter的布局中，内容是不能超出屏幕范围的，当超出时不会自动变成滚动效果，而是会报下面的错误；

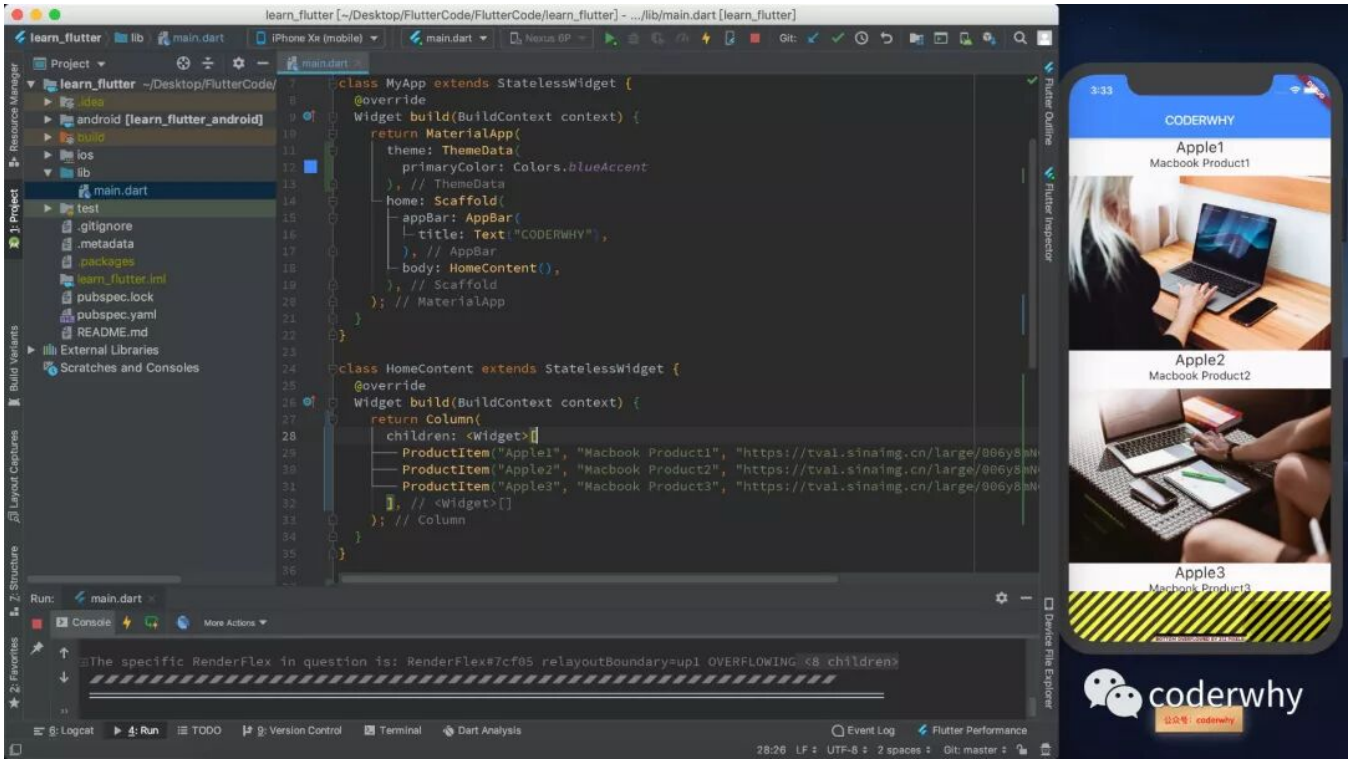

image-20190917153343504

如何可以解决这个问题呢?

- 我们将Column换成ListView即可；
- ListView可以让自己的子Widget变成滚动的效果；

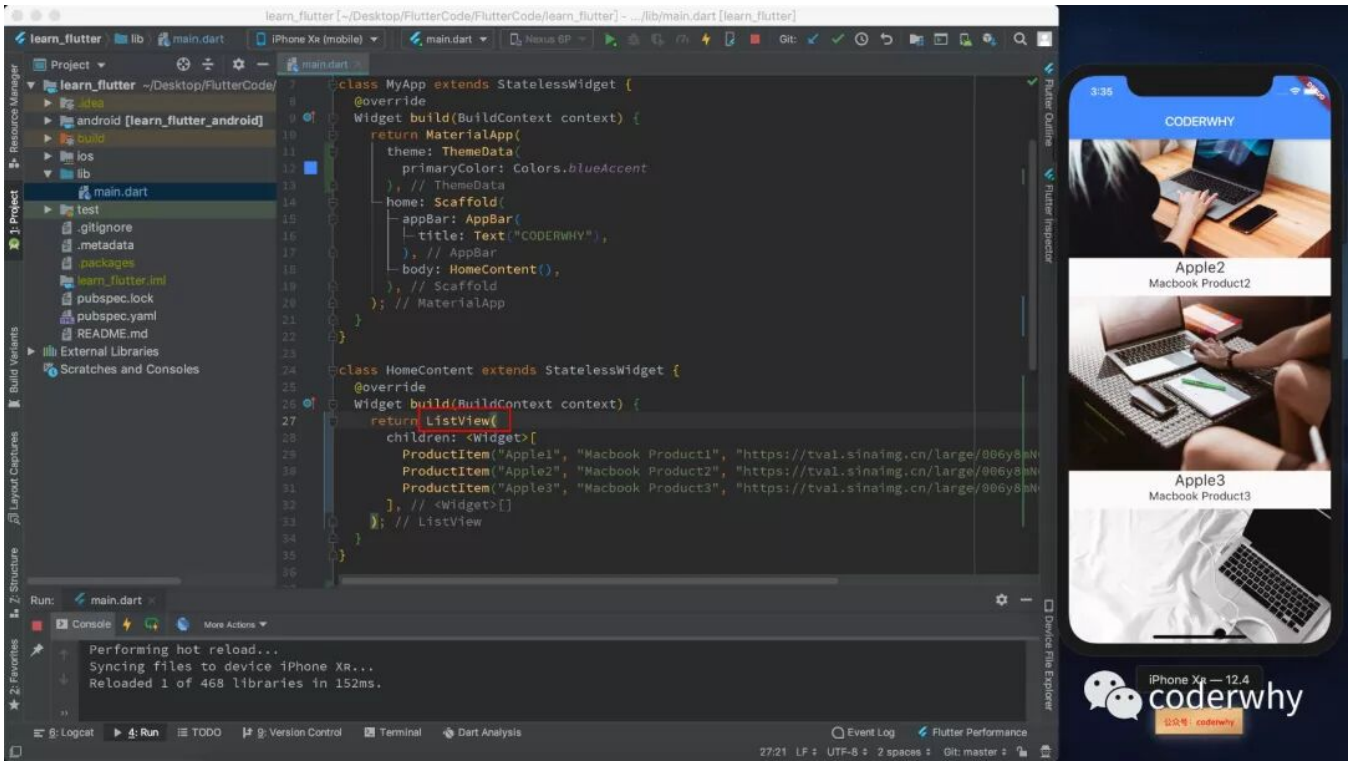

image-20190917153558093

3.4. 案例细节调整

3.4.1. 界面整体边距

如果我们希望整个内容距离屏幕的边缘有一定的间距，怎么做呢?

- 我们需要使用另外一个Widget：Padding，它有一个padding属性用于设置边距大小；
- 没错，设置内边距也是使用Widget，这个Widget就是Padding；

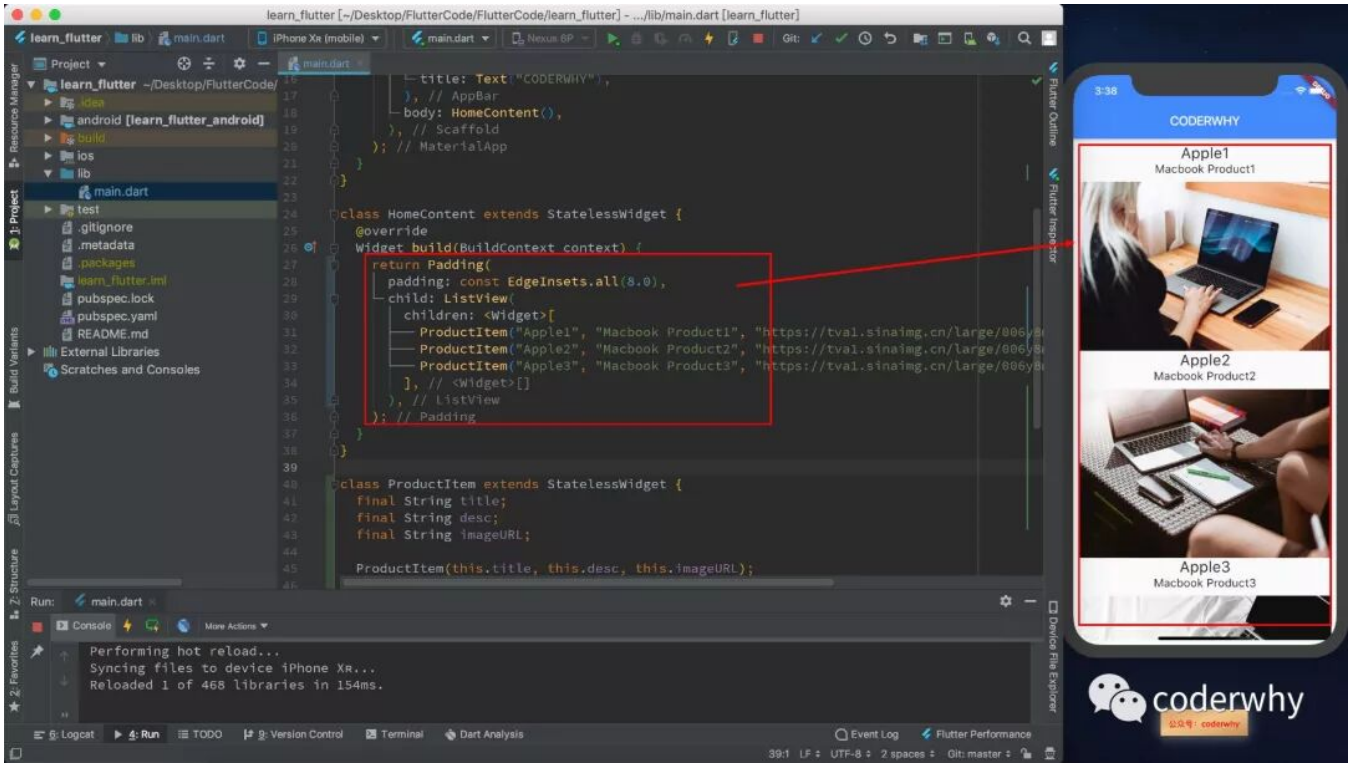

image-20190917153942819

3.4.2. 商品内边距和边框

我们现在希望给所有的商品也添加一个内边距，并且还有边框，怎么做呢?

- 我们可以使用一个Container的Widget，它里面有padding属性，并且可以通过decoration来设置边框；
- Container我们也会在后面详细来讲，我们先用起来；

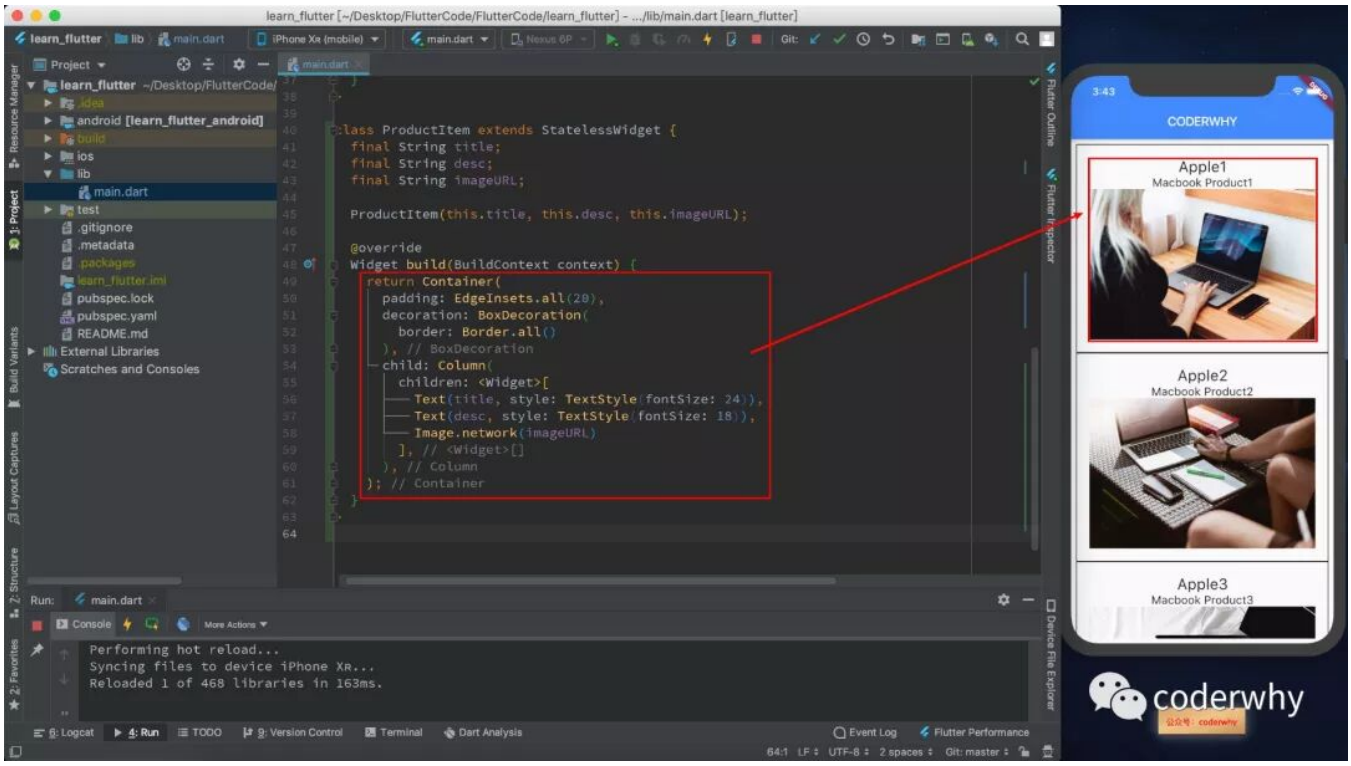

image-20190917154359364

3.4.3. 文字图片的间距

我们希望给图片和文字之间添加一些间距，怎么做呢？

- 方式一：给图片或者文字添加一个向上的内边距或者向下的内边距；
- 方式二：使用SizedBox的Widget，设置一个height属性，可以增加一些距离；

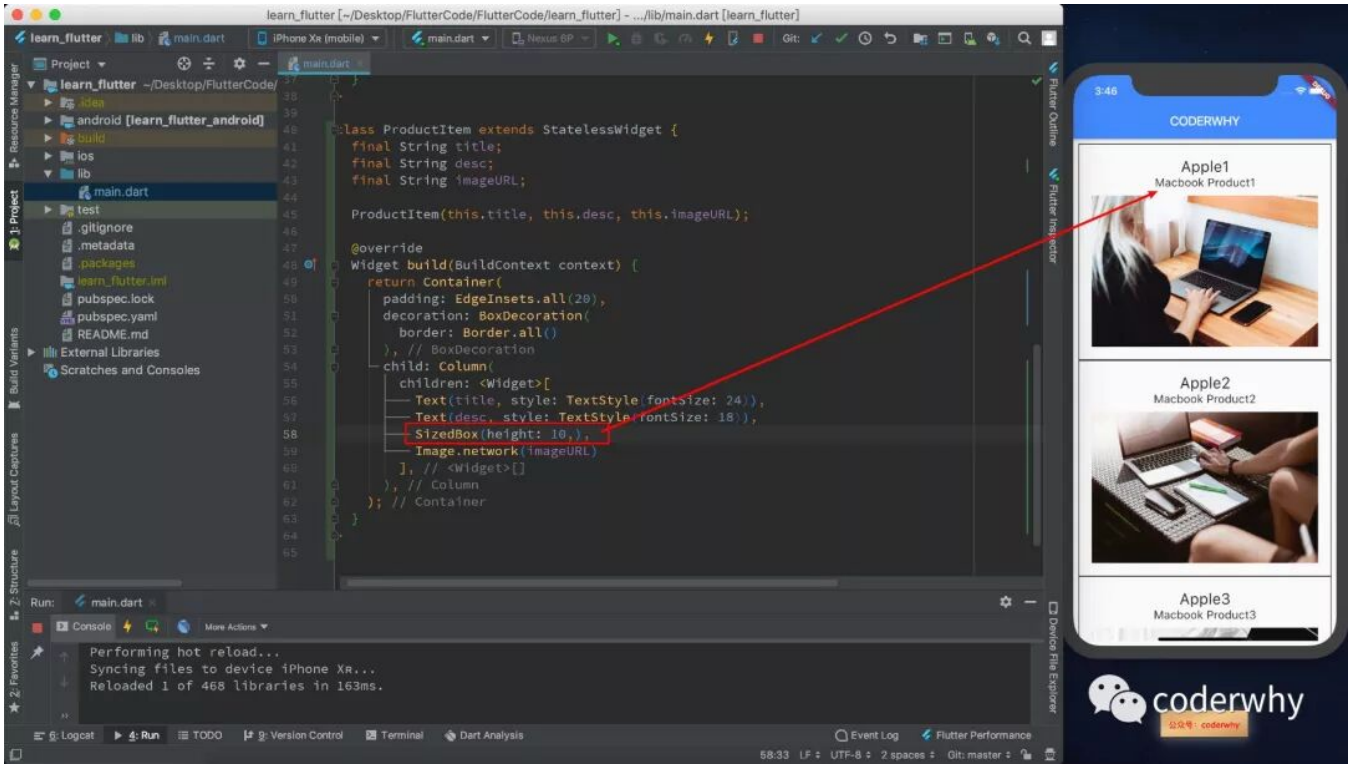

image-20190917154630828

3.5. 最终实现代码

最后，我给出最终实现代码：

```
1
2
```

```dart
import 'package:flutter/material.dart';

main(List<String> args) {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      theme: ThemeData(
        primaryColor: Colors.blueAccent
      ),
      home: Scaffold(
        appBar: AppBar(
          title: Text("CODERWHY"),
        ),
        body: HomeContent(),
      ),
    );
  }
}

class HomeContent extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.all(8.0),
      child: ListView(
        children: <Widget>[
          ProductItem("Apple1", "Macbook Product1", "https://tva1.sinaimg.cn/large/006y8mN6gy1g72j6nk1d4j30u00k0n0j.jpg"),
          ProductItem("Apple2", "Macbook Product2", "https://tva1.sinaimg.cn/large/006y8mN6gy1g72imm9u5zj30u00k0adf.jpg"),
          ProductItem("Apple3", "Macbook Product3", "https://tva1.sinaimg.cn/large/006y8mN6gy1g72imqlouhj30u00k00v0.jpg"),
        ],
      ),
    );
  }
}

class ProductItem extends StatelessWidget {
  final String title;
  final String desc;
  final String imageURL;

  ProductItem(this.title, this.desc, this.imageURL);

  @override
  Widget build(BuildContext context) {
    return Container(
      padding: EdgeInsets.all(20),
      decoration: BoxDecoration(
        border: Border.all()
      ),
      child: Column(
        children: <Widget>[
          Text(title, style: TextStyle(fontSize: 24)),
          Text(desc, style: TextStyle(fontSize: 18)),
          SizedBox(height: 10,),
          Image.network(imageURL)
        ],
      ),
    );
  }
}
```

> 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

coderwhy

公众号