Flutter(三)之Flutter的基础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：这个章节本来打算讲解Flutter的渲染原理，但是学习初期过多的讲解原理性的内容，并不利于大家快速入门和上手，做出一些带效果的内容；
>
> 所以，我打算换一种思路，先讲解一些组件的用法，让大家习惯Flutter的开发过程和模式，再回头去巩固原理性的知识；
>
> 另外，在讲解这些Widget的时候，我并不打算将所有的属性一一列出，因为没有意义，也记不住；
>
> 我后面打算有一个专题是 **关于Flutter布局** 的，会选出一些好看的布局界面带着大家一起来完成：美团页面、京东页面、B站页面等等，某些我目前没有讲到的属性，后面应用的会再进行讲解；

1. 文本Widget

在Android中，我们使用TextView，iOS中我们使用UILabel来显示文本；

Flutter中，我们使用Text组件控制文本如何展示；

1.1. 普通文本展示

在Flutter中，我们可以将文本的控制显示分成两类：

- **控制文本布局的参数：** 如文本对齐方式 textAlign、文本排版方向 textDirection，文本显示最大行数 maxLines、文本截断规则 overflow 等等，这些都是构造函数中的参数；
- **控制文本样式的参数：** 如字体名称 fontFamily、字体大小 fontSize、文本颜色 color、文本阴影 shadows 等等，这些参数被统一封装到了构造函数中的参数 style 中；

下面我们来看一下其中一些属性的使用：

```
class MyHomeBody extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Text(
      "《定风波》 苏轼 \n莫听穿林打叶声，何妨吟啸且徐行。\n竹杖芒鞋轻胜马，谁怕？一蓑烟雨任平生。"
,
      style: TextStyle(
        fontSize: 20,
        color: Colors.purple
      ),
    );
  }
}
```

展示效果如下：

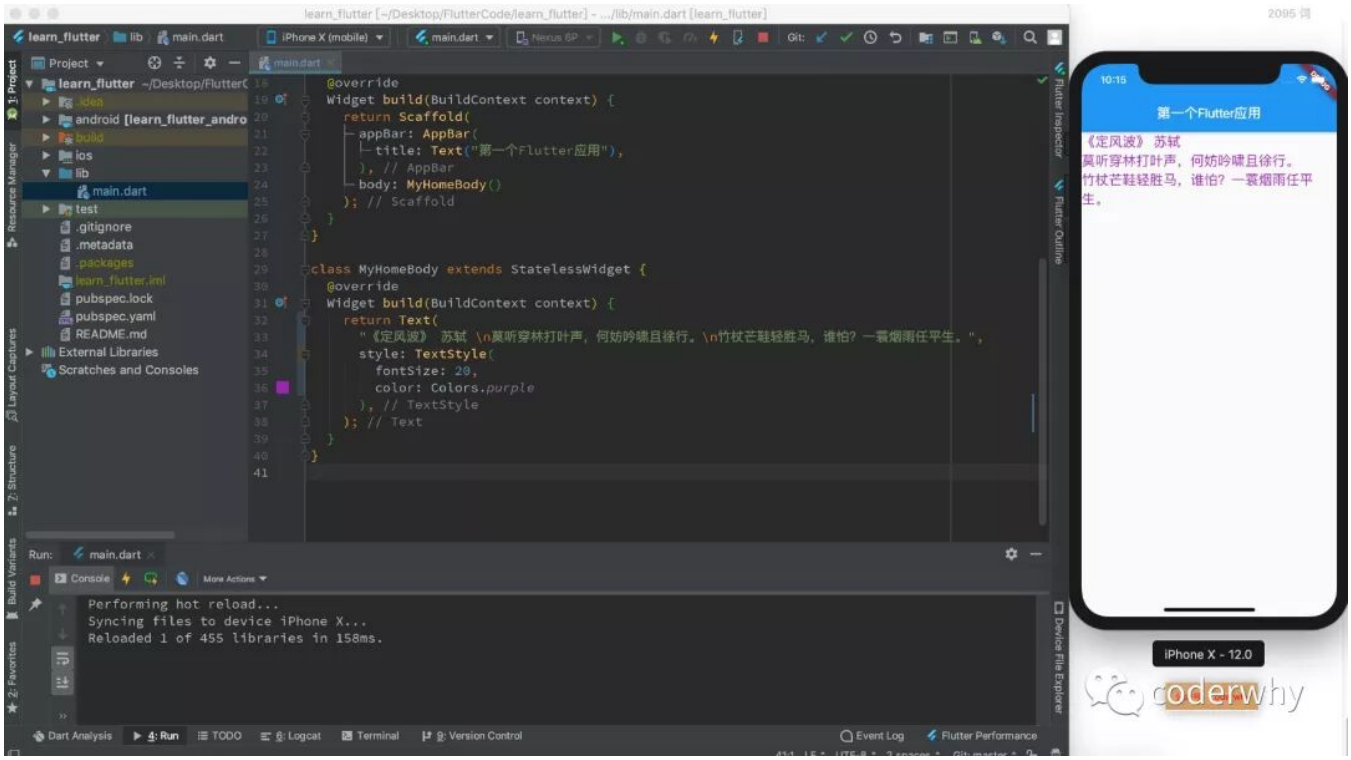

image-20190902101522966

我们可以通过一些属性来改变Text的布局：

- textAlign：文本对齐方式，比如TextAlign.center
- maxLines：最大显示行数，比如1
- overflow：超出部分显示方式，比如TextOverflow.ellipsis
- textScaleFactor：控制文本缩放，比如1.24

代码如下：

```
class MyHomeBody extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Text(
      "《定风波》 苏轼 \n莫听穿林打叶声，何妨吟啸且徐行。\n竹杖芒鞋轻胜马，谁怕？一蓑烟雨任平生。"
,
      textAlign: TextAlign.center, // 所有内容都居中对齐
      maxLines: 3, // 显然 "生。" 被删除了
      overflow: TextOverflow.ellipsis, // 超出部分显示...
//      textScaleFactor: 1.25,
      style: TextStyle(
        fontSize: 20,
        color: Colors.purple
      ),
    );
  }
}
```

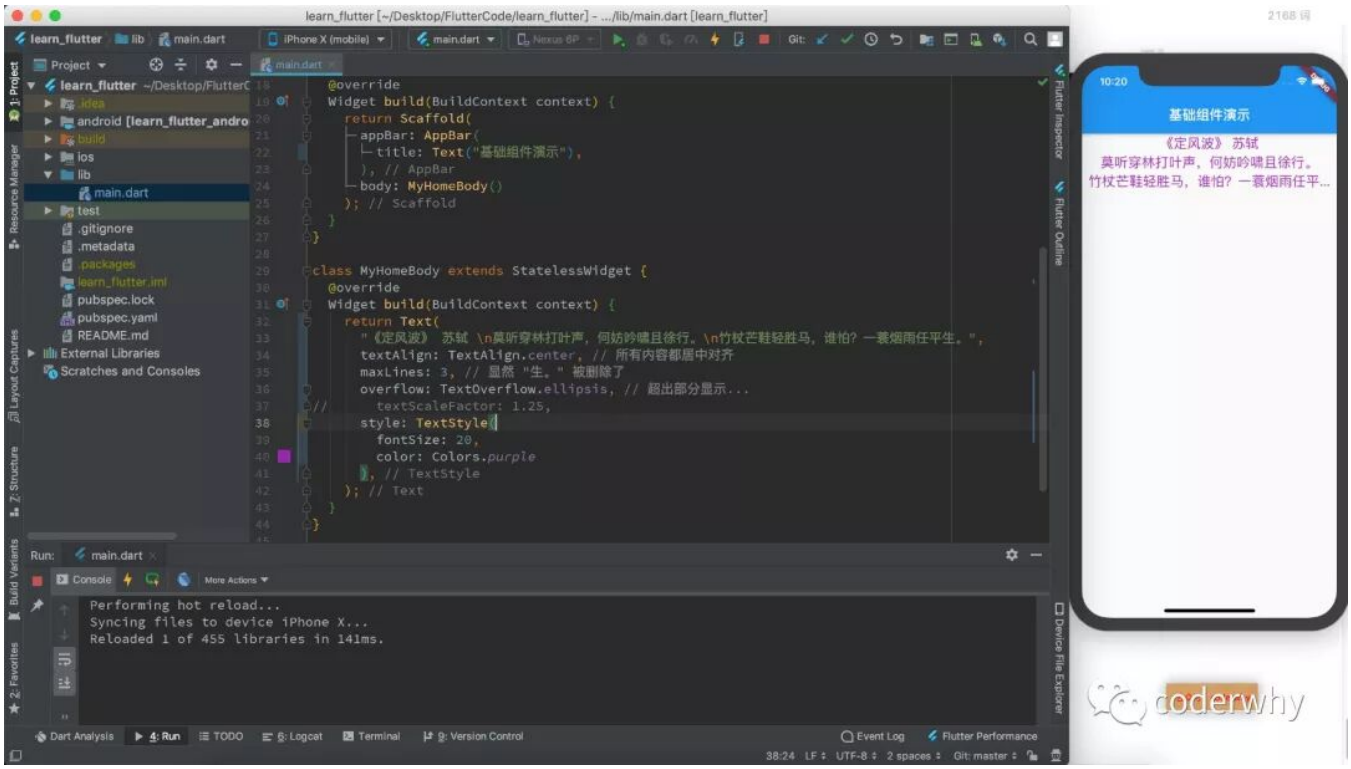

image-20190902102035307

1.2. 富文本展示

前面展示的文本，我们都应用了相同的样式，如果我们希望给他们不同的样式呢?

- 比如《定风波》我希望字体更大一点，并且是黑色字体，并且有加粗效果；
- 比如 苏轼 我希望是红色字体；

如果希望展示这种混合样式，那么我们可以利用分片来进行操作（在Android中，我们可以使用SpannableString，在iOS中，我们可以使用NSAttributedString完成，了解即可）

代码如下：

```
class MyHomeBody extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Text.rich(
      TextSpan(
        children: [
          TextSpan(text: "《定风波》", style: TextStyle(fontSize: 25, fontWeight: FontWeight.bold, color: Colors.black)),
          TextSpan(text: "苏轼", style: TextStyle(fontSize: 18, color: Colors.redAccent)),
          TextSpan(text: "\n莫听穿林打叶声，何妨吟啸且徐行。\n竹杖芒鞋轻胜马，谁怕？一蓑烟雨任平生。")
        ],
      ),
      style: TextStyle(fontSize: 20, color: Colors.purple),
      textAlign: TextAlign.center,
    );
  }
}
```

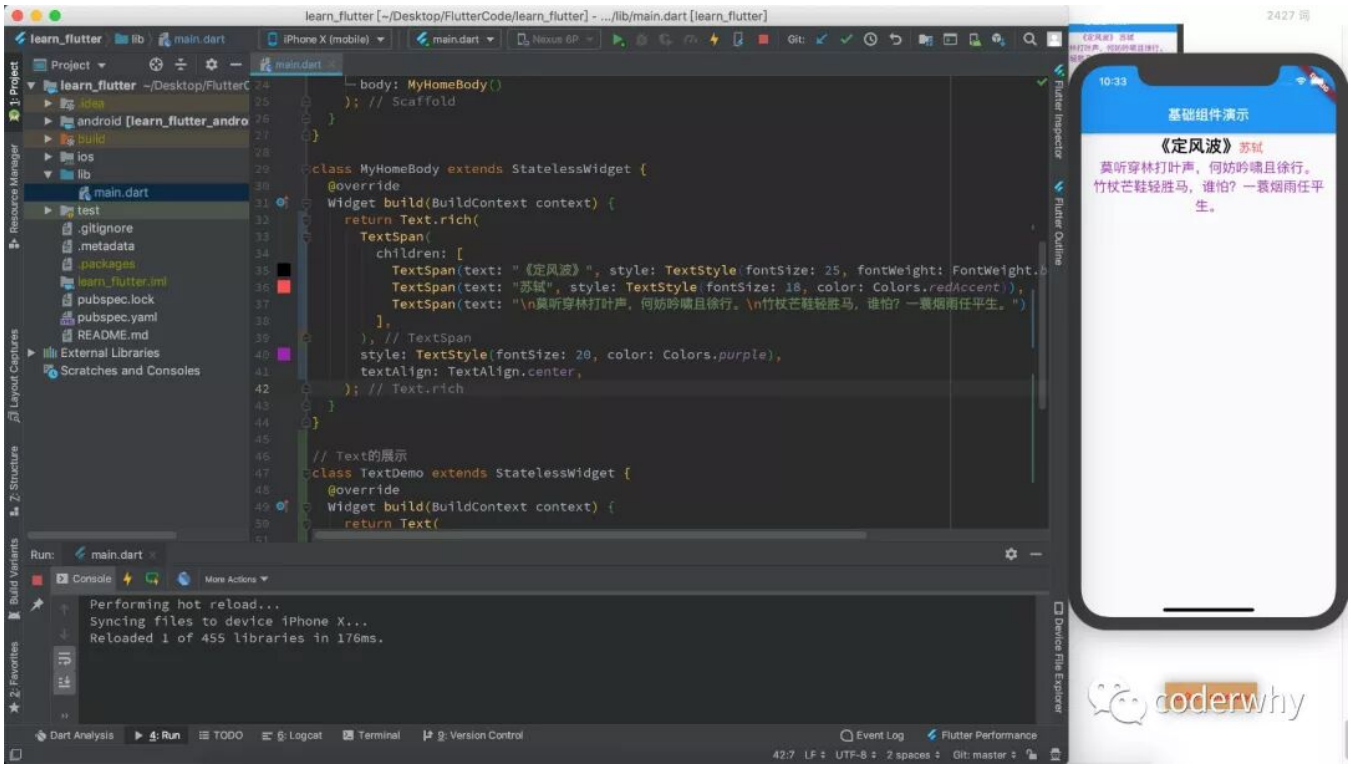

image-20190902103333353

二. 按钮Widget

2.1. 按钮的基础

Material widget库中提供了多种按钮Widget如FloatingActionButton、RaisedButton、FlatButton、OutlineButton等

我们直接来对他们进行一个展示：

```
class MyHomeBody extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Column(
      children: <Widget>[
        FloatingActionButton(
          child: Text("FloatingActionButton"),
          onPressed: () {
            print("FloatingActionButton Click")
;
          },
        ),
        RaisedButton(
          child: Text("RaisedButton"),
          onPressed: () {
            print("RaisedButton Click");
          },
        ),
        FlatButton(
          child: Text("FlatButton"),
          onPressed: () {
            print("FlatButton Click");
          },
        ),
        OutlineButton(
          child: Text("OutlineButton"),
          onPressed: () {
            print("OutlineButton Click");
          },
        )
      ],
    );
  }
}
```

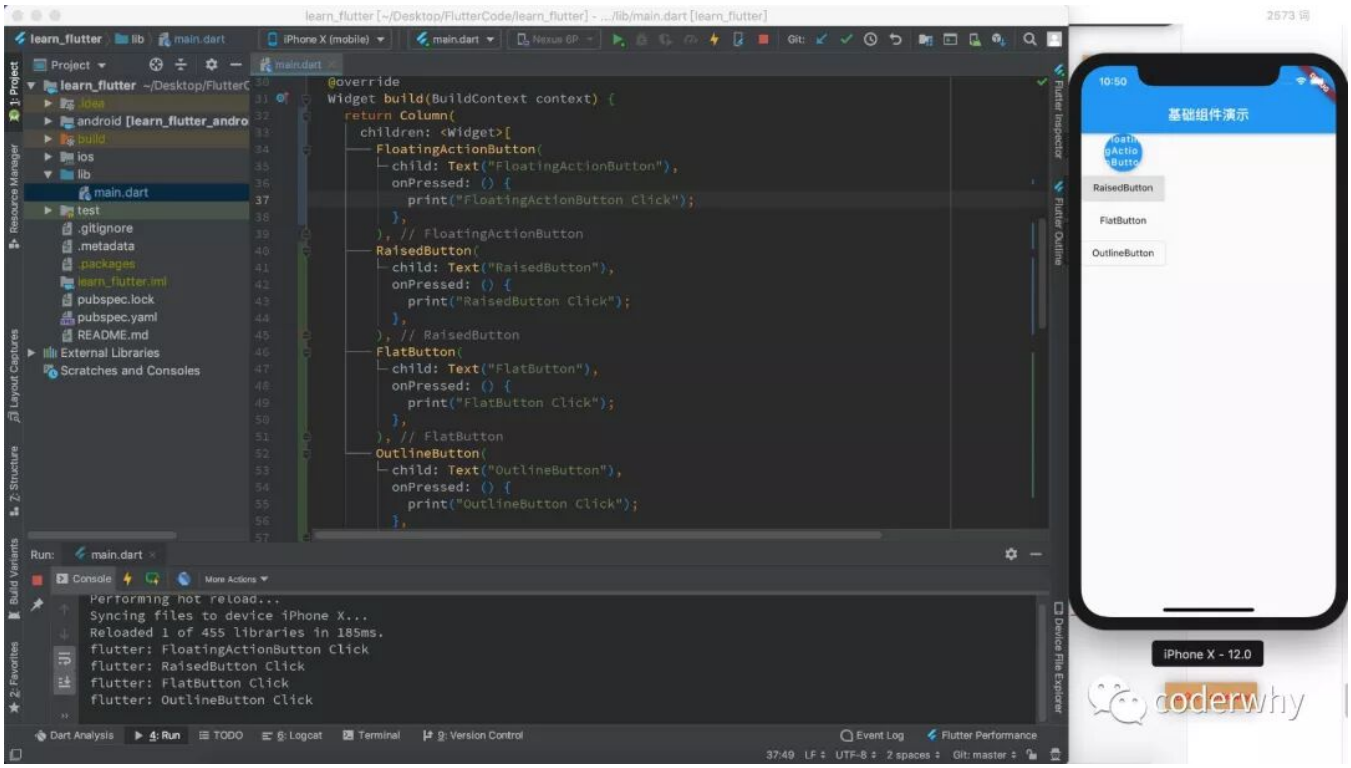

image-20190902105017343

2.2. 自定义样式

前面的按钮我们使用的都是默认样式，我们可以通过一些属性来改变按钮的样式

```
RaisedButton(
  child: Text("同意协议", style: TextStyle(color: Colors.white)),
  color: Colors.orange, // 按钮的颜色
  highlightColor: Colors.orange[700], // 按下去高亮颜色
  shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(20)), // 圆角的实
现
  onPressed: () {
    print("同意协议");
  },
)
```

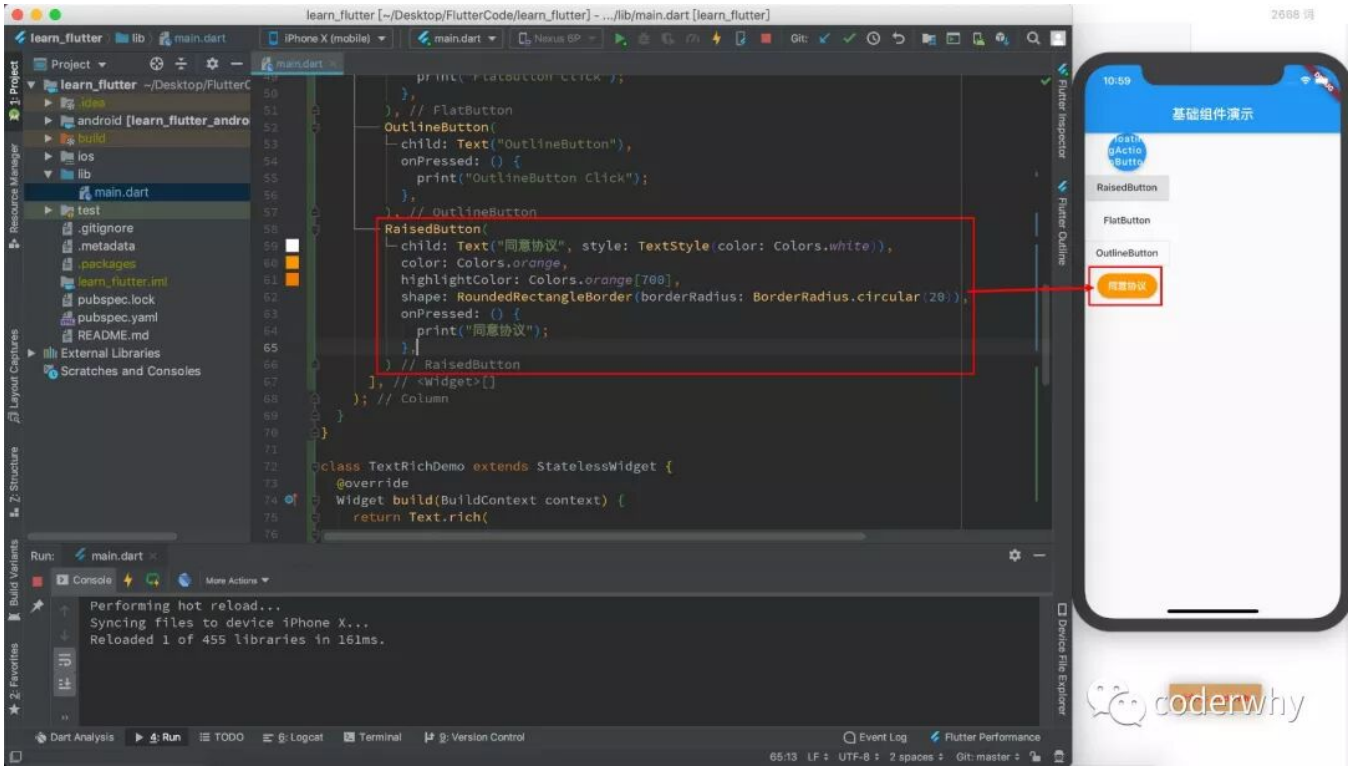

image-20190902110011756

事实上这里还有一个比较常见的属性：elevation，用于控制阴影的大小，很多地方都会有这个属性，大家可以自行演示一下

三. 图片Widget

图片可以让我们的应用更加丰富多彩，Flutter中使用Image组件

Image组件有很多的构造函数，我们这里主要学习两个：

- Image.assets：加载本地资源图片；
- Image.network：加载网络中的图片；

3.1. 加载网络图片

相对来讲，Flutter中加载网络图片会更加简单，直接传入URL并不需要什么配置，所以我们先来看一下Flutter中如何加载网络图片。

我们先来看看Image有哪些属性可以设置：

```
const Image({
  ...
  this.width, //图片的宽
  this.height, //图片高度
  this.color, //图片的混合色值
  this.colorBlendMode, //混合模式
  this.fit,//缩放模式
  this.alignment = Alignment.center, //对齐方式
  this.repeat = ImageRepeat.noRepeat, //重复方
式
  ...
})
```

- `width`、`height`：用于设置图片的宽、高，当不指定宽高时，图片会根据当前父容器的限制，尽可能的显示其原始大小，如果只设置`width`、`height`的其中一个，那么另一个属性默认会按比例缩放，但可以通过下面介绍的`fit`属性来指定适应规则。
- `fit`：该属性用于在图片的显示空间和图片本身大小不同时指定图片的适应模式。适应模式是在`BoxFit`中定义，它是一个枚举类型，有如下值：
  - `fill`：会拉伸填充满显示空间，图片本身长宽比会发生变化，图片会变形。
  - `cover`：会按图片的长宽比放大后居中填满显示空间，图片不会变形，超出显示空间部分会被剪裁。
  - `contain`：这是图片的默认适应规则，图片会在保证图片本身长宽比不变的情况下缩放以适应当前显示空间，图片不会变形。
  - `fitWidth`：图片的宽度会缩放到显示空间的宽度，高度会按比例缩放，然后居中显示，图片不会变形，超出显示空间部分会被剪裁。
  - `fitHeight`：图片的高度会缩放到显示空间的高度，宽度会按比例缩放，然后居中显示，图片不会变形，超出显示空间部分会被剪裁。
  - `none`：图片没有适应策略，会在显示空间内显示图片，如果图片比显示空间大，则显示空间只会显示图片中间部分。
- `color`和`colorBlendMode`：在图片绘制时可以对每一个像素进行颜色混合处理，`color`指定混合色，而`colorBlendMode`指定混合模式；
- `repeat`：当图片本身大小小于显示空间时，指定图片的重复规则。

我们对其中某些属性做一个演练：

- 注意，这里我用了一个Container，大家可以把它理解成一个UIView或者View，就是一个容器；
- 后面我会专门讲到这个组件的使用；

```
class MyHomeBody extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Center(
      child: Container(
        child: Image.network(
          "http://img0.dili360.com/ga/M01/48/3C/wKgBy1kj49qAMVd7ADKmuZ9jug8377.tub.jpg"
,
          alignment: Alignment.topCenter,
          repeat: ImageRepeat.repeatY,
          color: Colors.red,
          colorBlendMode: BlendMode.colorDodge,
        ),
        width: 300,
        height: 300,
        color: Colors.yellow,
      ),
    );
  }
}
```

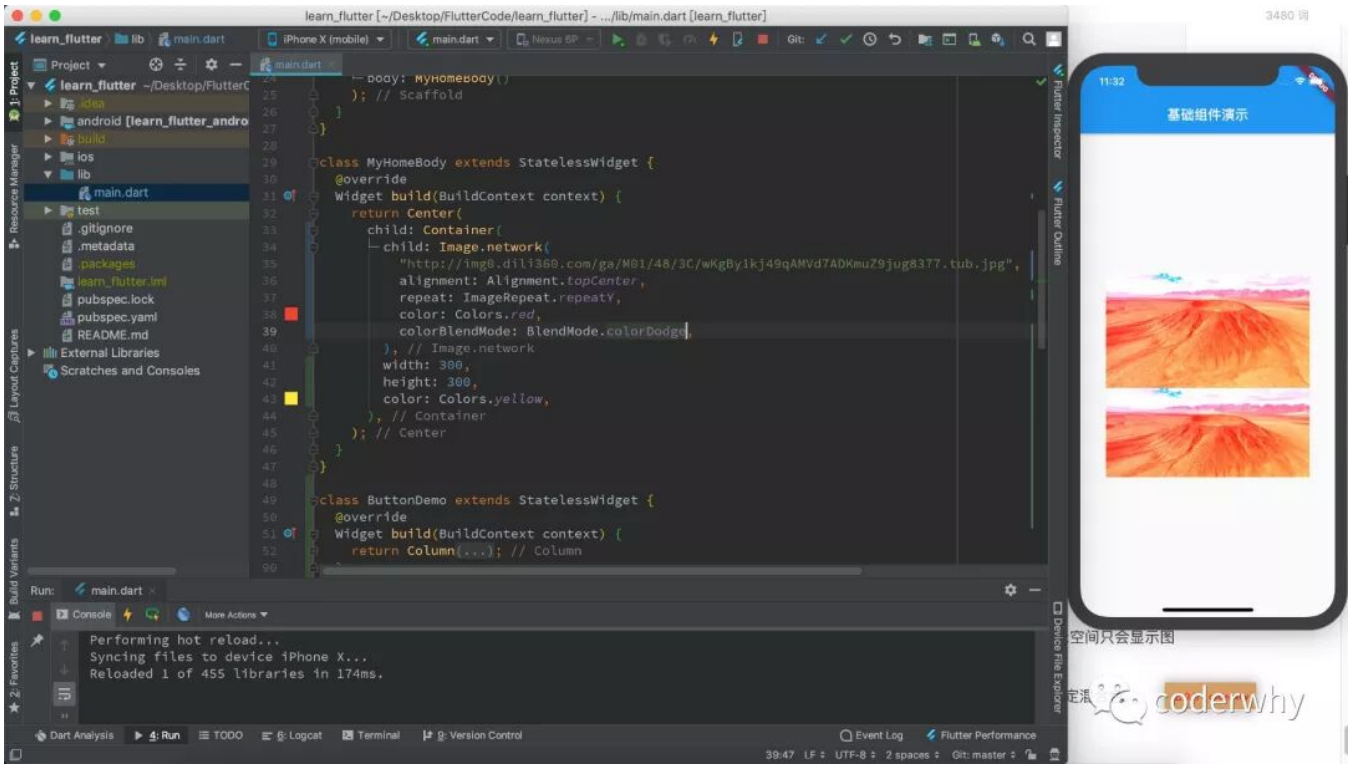

image-20190902113310213

3.2. 加载本地图片

加载本地图片稍微麻烦一点，需要将图片引入，并且进行配置

```
class MyHomeBody extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Center(
      child: Container(
        width: 300,
        height: 300,
        color: Colors.yellow,
        child: Image.asset("images/test.jpeg")
,
      ),
    );
  }
}
```

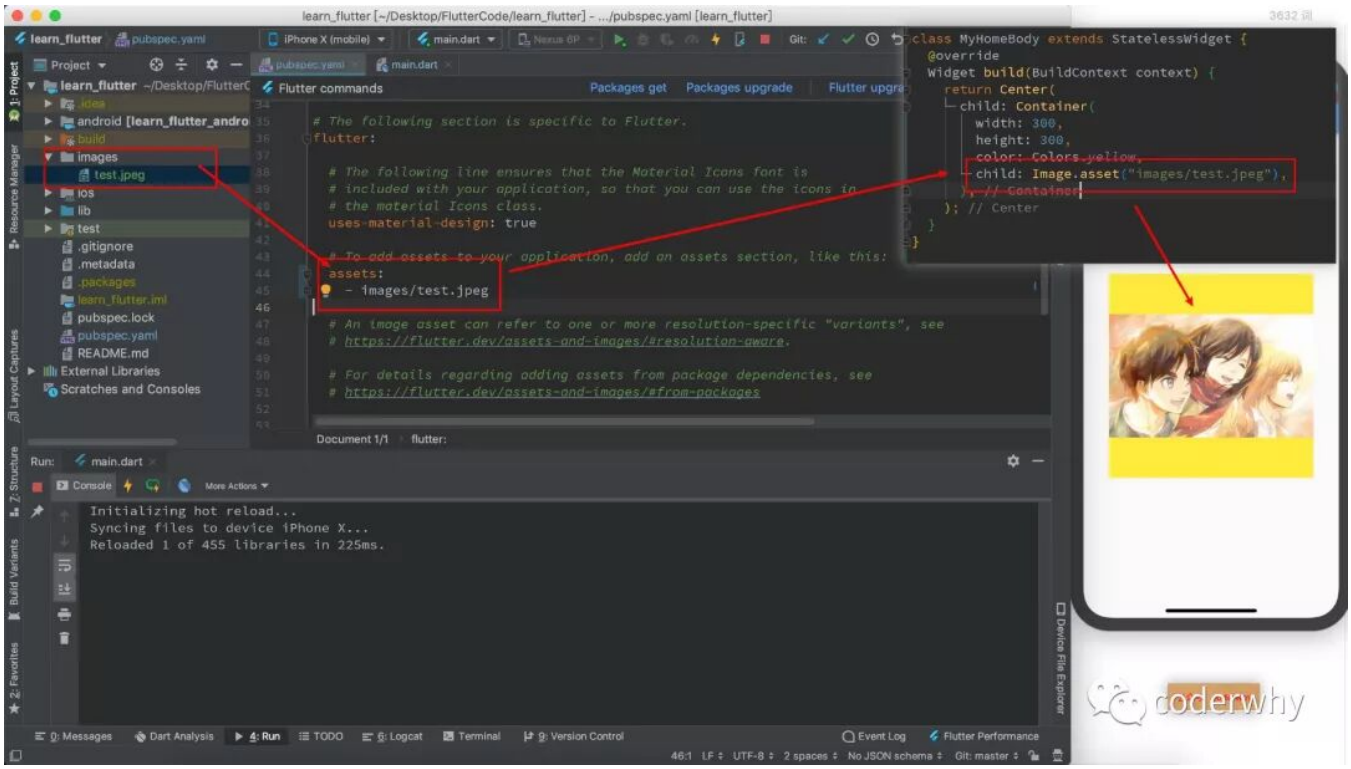

image-20190902114616699

3.3. 实现圆角图像

> 在Flutter中实现圆角效果也是使用一些Widget来实现的。

3.3.1. 实现圆角头像

方式一：CircleAvatar

CircleAvatar可以实现圆角头像，也可以添加一个子Widget：

```
const CircleAvatar({
  Key key,
  this.child, // 子Widget
  this.backgroundColor, // 背景颜色
  this.backgroundImage, // 背景图像
  this.foregroundColor, // 前景颜色
  this.radius, // 半径
  this.minRadius, // 最小半径
  this.maxRadius, // 最大半径
})
```

我们来实现一个圆形头像：

- 注意一：这里我们使用的是NetworkImage，因为backgroundImage要求我们传入一个ImageProvider；
  - ImageProvider是一个抽象类，事实上所有我们前面创建的Image对象都有包含 `image` 属性，该属性就是一个ImageProvider
- 注意二：这里我还在里面添加了一个文字，但是我在文字外层包裹了一个Container；
  - 这里Container的作用是为了可以控制文字在其中的位置调整；

```
class HomeContent extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Center(
      child: CircleAvatar(
        radius: 100,
        backgroundImage: NetworkImage("https://tva1.sinaimg.cn/large/006y8mN6gy1g7aa03bmfpj3069069mx8.jpg")
,
        child: Container(
          alignment: Alignment(0, .5),
          width: 200,
          height: 200,
          child: Text("兵长利威尔")
        ),
      ),
    );
  }
}
```

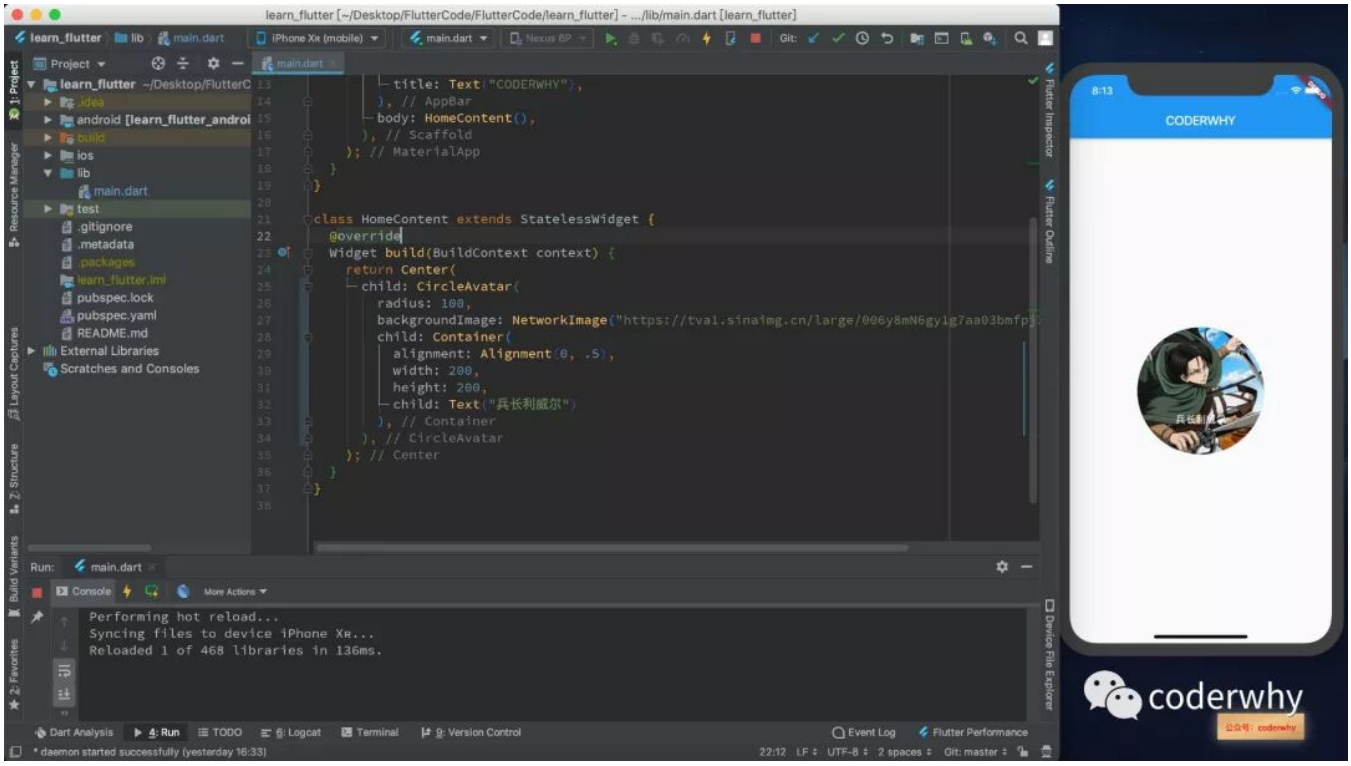

image-20190924081343639

方式二：ClipOval

ClipOval也可以实现圆角头像，而且通常是在只有头像时使用

```
class HomeContent extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Center(
      child: ClipOval(
        child: Image.network(
          "https://tva1.sinaimg.cn/large/006y8mN6gy1g7aa03bmfpj3069069mx8.jpg"
,
          width: 200,
          height: 200,
        ),
      ),
    );
  }
}
```

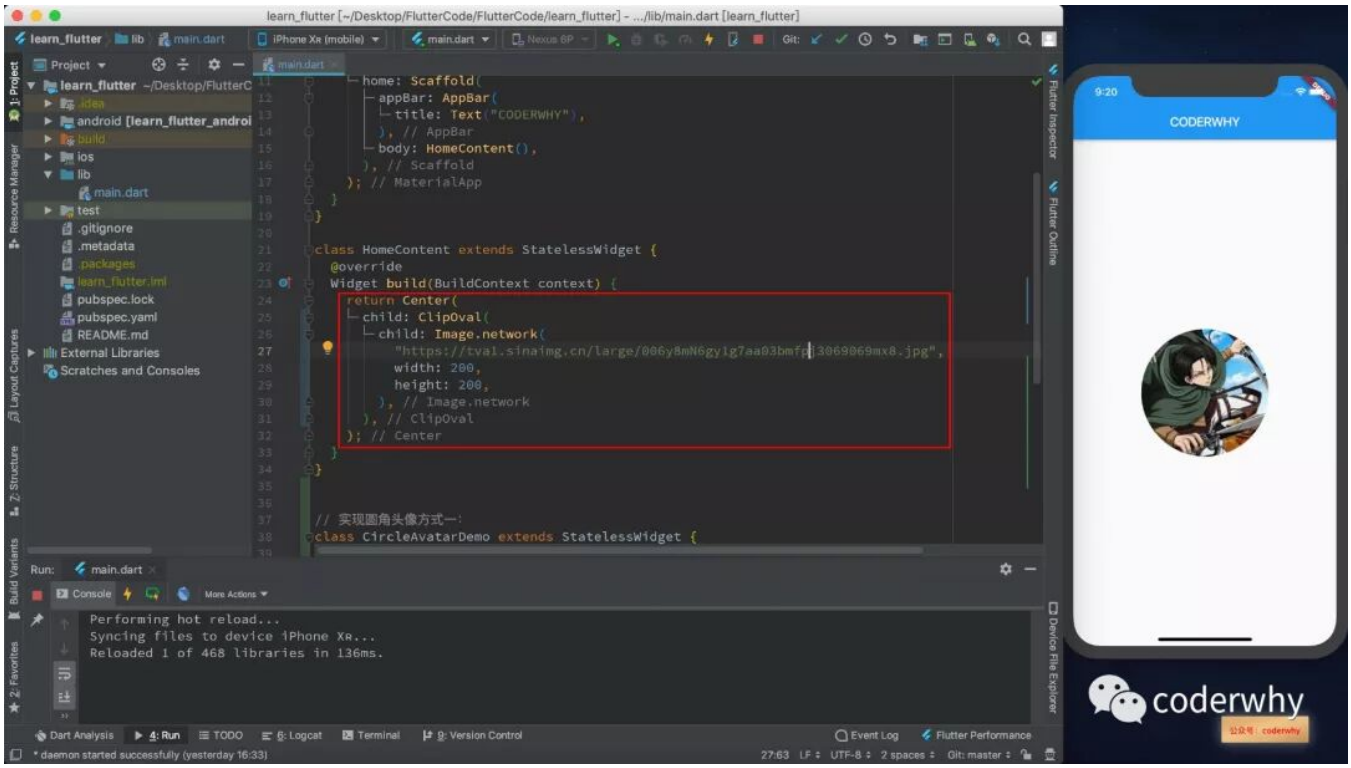

image-20190924092127687

实现方式三：Container+BoxDecoration

这种方式我们放在讲解Container时来讲这种方式

3.3.2. 实现圆角图片

方式一：ClipRRect

ClipRRect用于实现圆角效果，可以设置圆角的大小。

实现代码如下，非常简单：

```
class HomeContent extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Center(
      child: ClipRRect(
        borderRadius: BorderRadius.circular(10),
        child: Image.network(
          "https://tva1.sinaimg.cn/large/006y8mN6gy1g7aa03bmfpj3069069mx8.jpg"
,
          width: 200,
          height: 200,
        ),
      ),
    );
  }
}
```

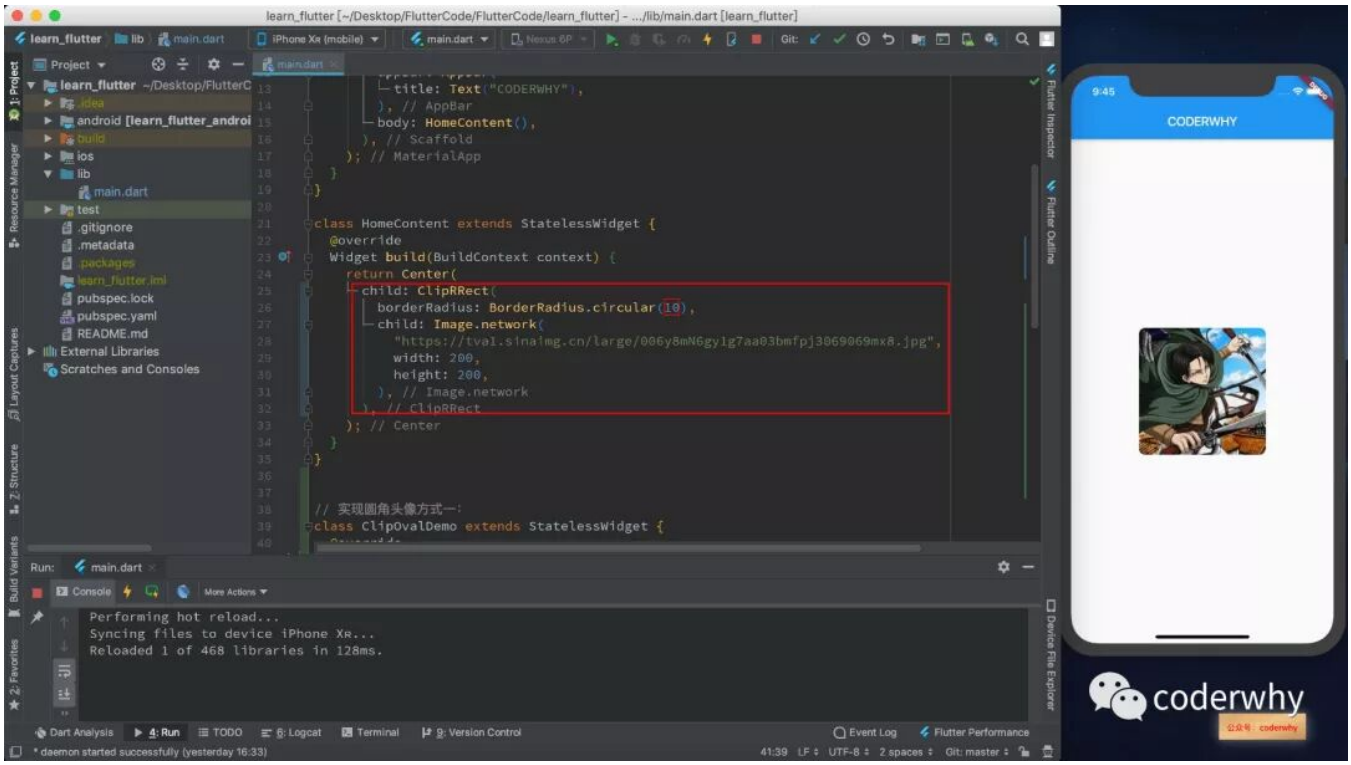

image-20190924094516174

方式二：Container+BoxDecoration

这个也放到后面讲解Container时讲解

四. 表单Widget

> 和用户交互的其中一种就是输入框，比如注册、登录、搜索，我们收集用户输入的内容将其提交到服务器。

4.1. TextField的使用

4.1.1. TextField的介绍

TextField用于接收用户的文本输入，它提供了非常多的属性，我们来看一下源码：

- 但是我们没必要一个个去学习，很多时候用到某个功能时去查看是否包含某个属性即可

```
const TextField({
  Key key,
  this.controller,
  this.focusNode,
  this.decoration = const InputDecoration(),
  TextInputType keyboardType,
  this.textInputAction,
  this.textCapitalization = TextCapitalization.none,
  this.style,
  this.strutStyle,
  this.textAlign = TextAlign.start,
  this.textAlignVertical,
  this.textDirection,
  this.readOnly = false,
  ToolbarOptions toolbarOptions,
  this.showCursor,
  this.autofocus = false,
  this.obscureText = false,
  this.autocorrect = true,
  this.maxLines = 1,
  this.minLines,
  this.expands = false,
  this.maxLength,
  this.maxLengthEnforced = true,
  this.onChanged,
  this.onEditingComplete,
  this.onSubmitted,
  this.inputFormatters,
  this.enabled,
  this.cursorWidth = 2.0,
  this.cursorRadius,
  this.cursorColor,
  this.keyboardAppearance,
  this.scrollPadding = const EdgeInsets.all(20.0),
  this.dragStartBehavior = DragStartBehavior.start,
  this.enableInteractiveSelection = true,
  this.onTap,
  this.buildCounter,
  this.scrollController,
  this.scrollPhysics,
})
```

我们来学习几个比较常见的属性：

- 一些属性比较简单：`keyboardType` 键盘的类型，`style` 设置样式，`textAlign` 文本对齐方式，`maxLength` 最大显示行数等等；
- `decoration` ：用于设置输入框相关的样式
  - icon：设置左边显示的图标
  - labelText：在输入框上面显示一个提示的文本
  - hintText：显示提示的占位文字
  - border：输入框的边框，默认底部有一个边框，可以通过InputBorder.none删除掉
  - filled：是否填充输入框，默认为false
  - fillColor：输入框填充的颜色
- `controller` ：
- `onChanged` ：监听输入框内容的改变，传入一个回调函数
- `onSubmitted` ：点击键盘中右下角的down时，会回调的一个函数

4.1.2. TextField的样式以及监听

我们来演示一下TextField的decoration属性以及监听：

```dart
class HomeContent extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Container(
      padding: EdgeInsets.all(20),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          TextFieldDemo()
        ],
      ),
    );
  }
}

class TextFieldDemo extends StatefulWidget {
  @override
  _TextFieldDemoState createState() => _TextFieldDemoState();
}

class _TextFieldDemoState extends State<TextFieldDemo> {
  @override
  Widget build(BuildContext context) {
    return TextField(
      decoration: InputDecoration(
        icon: Icon(Icons.people),
        labelText: "username",
        hintText: "请输入用户名",
        border: InputBorder.none,
        filled: true,
        fillColor: Colors.lightGreen
      ),
      onChanged: (value) {
        print("onChanged:$value");
      },
      onSubmitted: (value) {
        print("onSubmitted:$value");
      },
    );
  }
}
```

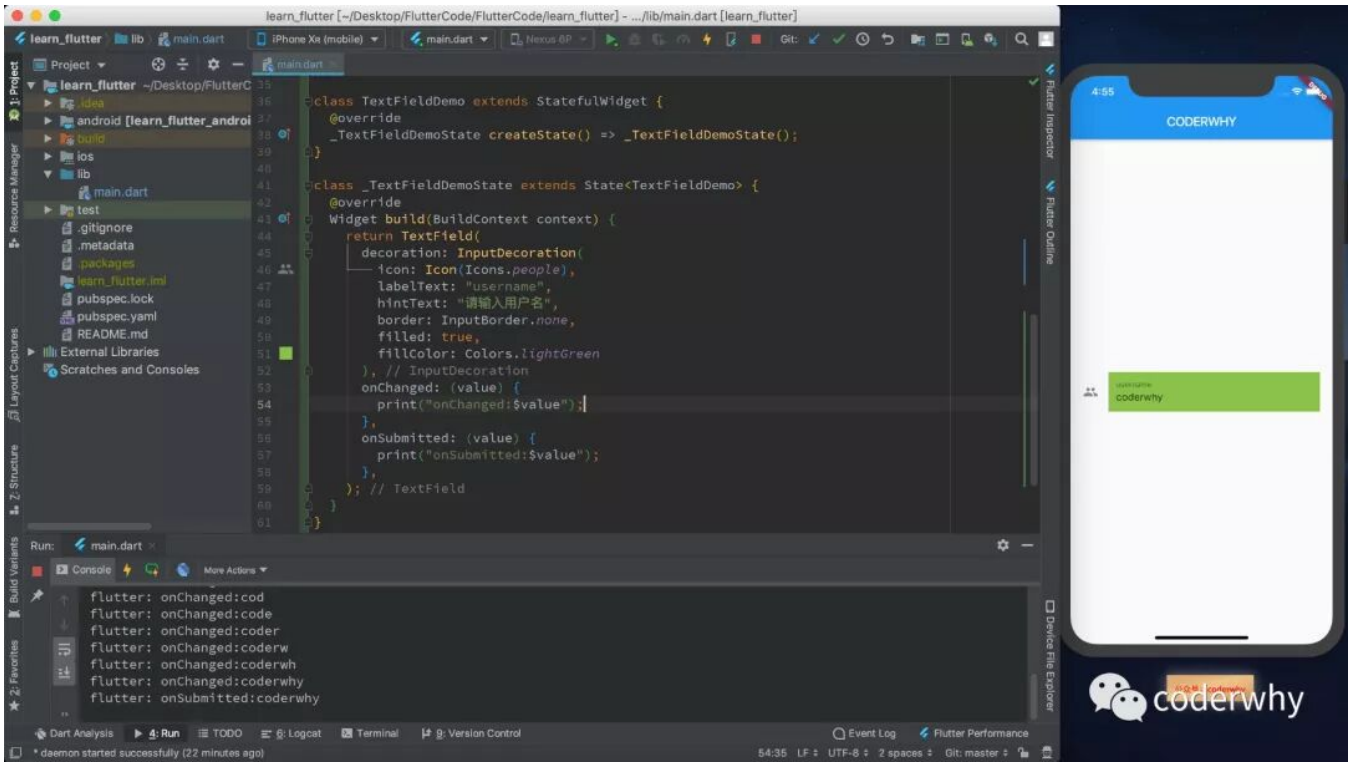

image-20190923165526780

4.1.3. TextField的controller

我们可以给TextField添加一个控制器（Controller），可以使用它设置文本的初始值，也可以使用它来监听文本的改变；

事实上，如果我们没有为TextField提供一个Controller，那么会Flutter会默认创建一个TextEditingController的，这个结论可以通过阅读源码得到：

```
  @override
  void initState() {
    super.initState();
    // ...其他代码
    if (widget.controller == null)
      _controller = TextEditingController();
  }
```

我们也可以自己来创建一个Controller控制一些内容：

```
class _TextFieldDemoState extends State<TextFieldDemo> {
  final textEditingController = TextEditingController();

  @override
  void initState() {
    super.initState();

    // 1.设置默认值
    textEditingController.text = "Hello World";

    // 2.监听文本框
    textEditingController.addListener(() {
      print("textEditingController:${textEditingController.text}")
;
    });
  }

  // ...省略build方法
}
```

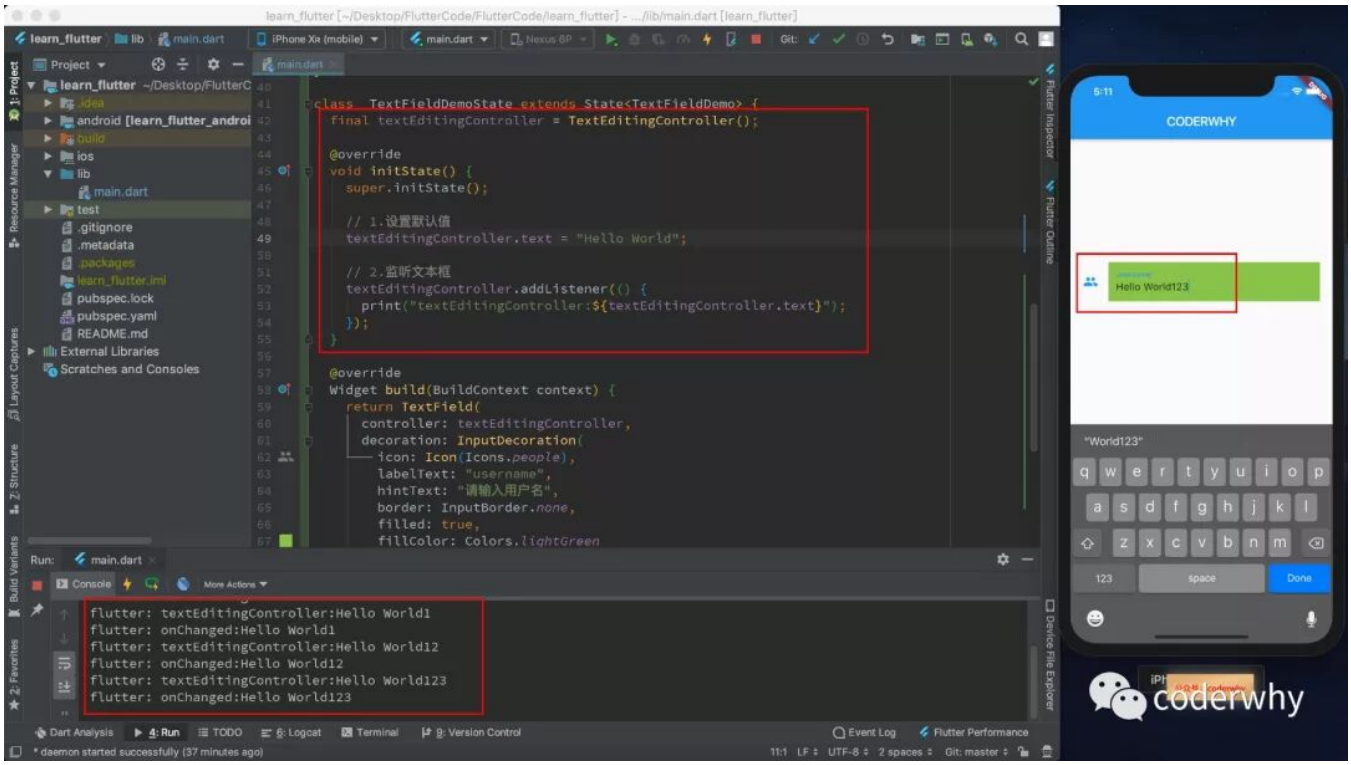

image-20190923171132816

4.2. Form表单的使用

> 在我们开发注册、登录页面时，通常会有多个表单需要同时获取内容或者进行一些验证，如果对每一个TextField都分别进行验证，是一件比较麻烦的事情。
>
> 做过前端的开发知道，我们可以将多个input标签放在一个form里面，Flutter也借鉴了这样的思想：我们可以通过Form对输入框进行分组，统一进行一些操作。

4.2.1. Form表单的基本使用

Form表单也是一个Widget，可以在里面放入我们的输入框。

但是Form表单中输入框必须是FormField类型的

- 我们查看刚刚学过的TextField是继承自StatefulWidget，并不是一个FormField类型；
- 我们可以使用TextFormField，它的使用类似于TextField，并且是继承自FormField的；

我们通过Form的包裹，来实现一个注册的页面：

```
class FormDemo extends StatefulWidget {
  @override
  _FormDemoState createState() => _FormDemoState();
}

class _FormDemoState extends State<FormDemo> {
  @override
  Widget build(BuildContext context) {
    return Form(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          TextFormField(
            decoration: InputDecoration(
              icon: Icon(Icons.people),
              labelText: "用户名或手机号"
            ),
          ),
          TextFormField(
            obscureText: true,
            decoration: InputDecoration(
              icon: Icon(Icons.lock),
              labelText: "密码"
            ),
          ),
          SizedBox(height: 16,),
          Container(
            width: double.infinity,
            height: 44,
            child: RaisedButton(
              color: Colors.lightGreen,
              child: Text("注 册", style: TextStyle(fontSize: 20, color: Colors.white),),
              onPressed: () {
                print("点击了注册按钮");
              },
            ),
          )
        ],
      ),
    );
  }
}
```

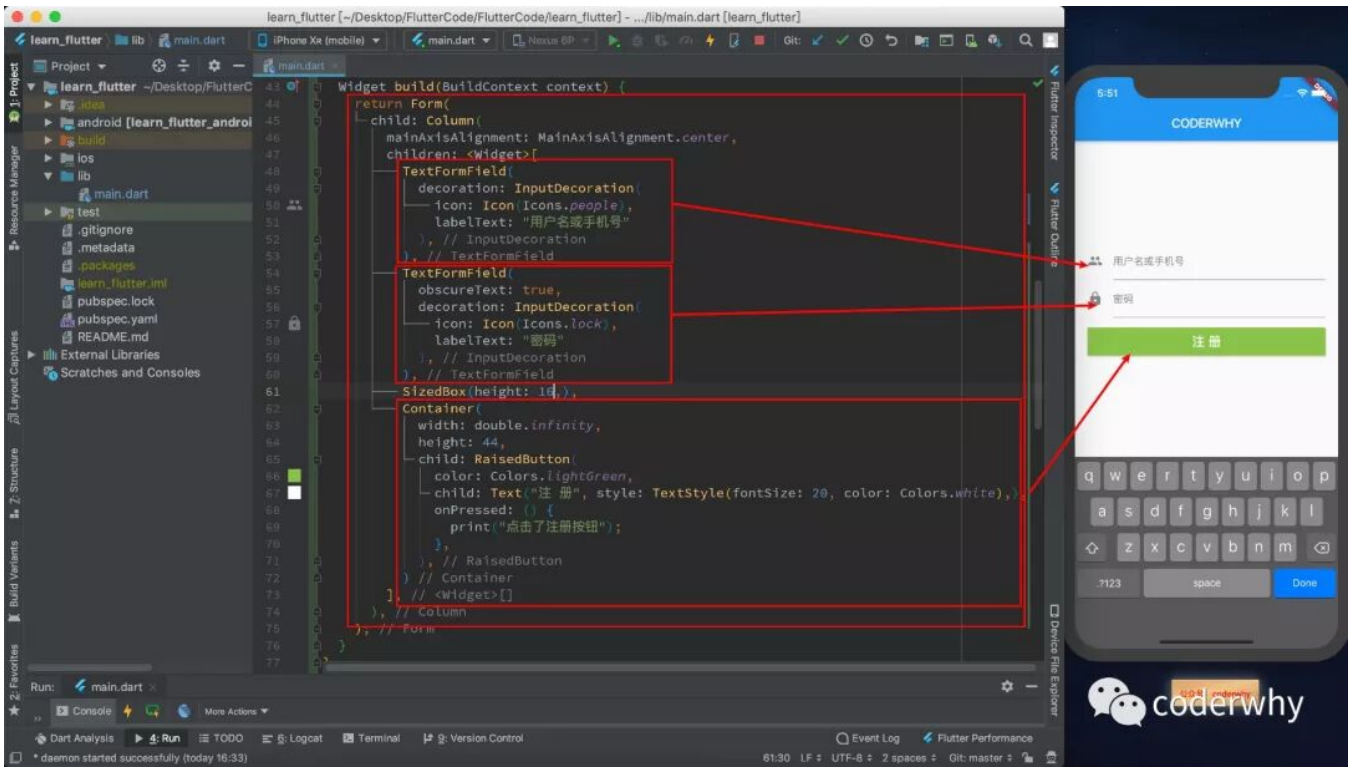

image-20190923175224983

4.2.2. 保存和获取表单数据

有了表单后，我们需要在点击注册时，可以同时获取和保存表单中的数据，怎么可以做到呢?

- 1、需要监听注册按钮的点击，在之前我们已经监听的onPressed传入的回调中来做即可。（当然，如果嵌套太多，我们待会儿可以将它抽取到一个单独的方法中）
- 2、监听到按钮点击时，同时获取 `用户名` 和 `密码` 的表单信息。

如何同时获取 `用户名` 和 `密码` 的表单信息?

- 如果我们调用 `Form的State对象` 的save方法，就会调用Form中放入的TextFormField的onSave回调：

```
TextFormField(
  decoration: InputDecoration(
    icon: Icon(Icons.people),
    labelText: "用户名或手机号"
  ),
  onSaved: (value) {
    print("用户名: $value");
  },
),
```

- 但是，我们有没有办法可以在点击按钮时，拿到 `Form对象` 来调用它的save方法呢?

知识点：在Flutter如何可以获取一个通过一个引用获取一个StatefulWidget的State对象呢?

答案：通过绑定一个GlobalKey即可。

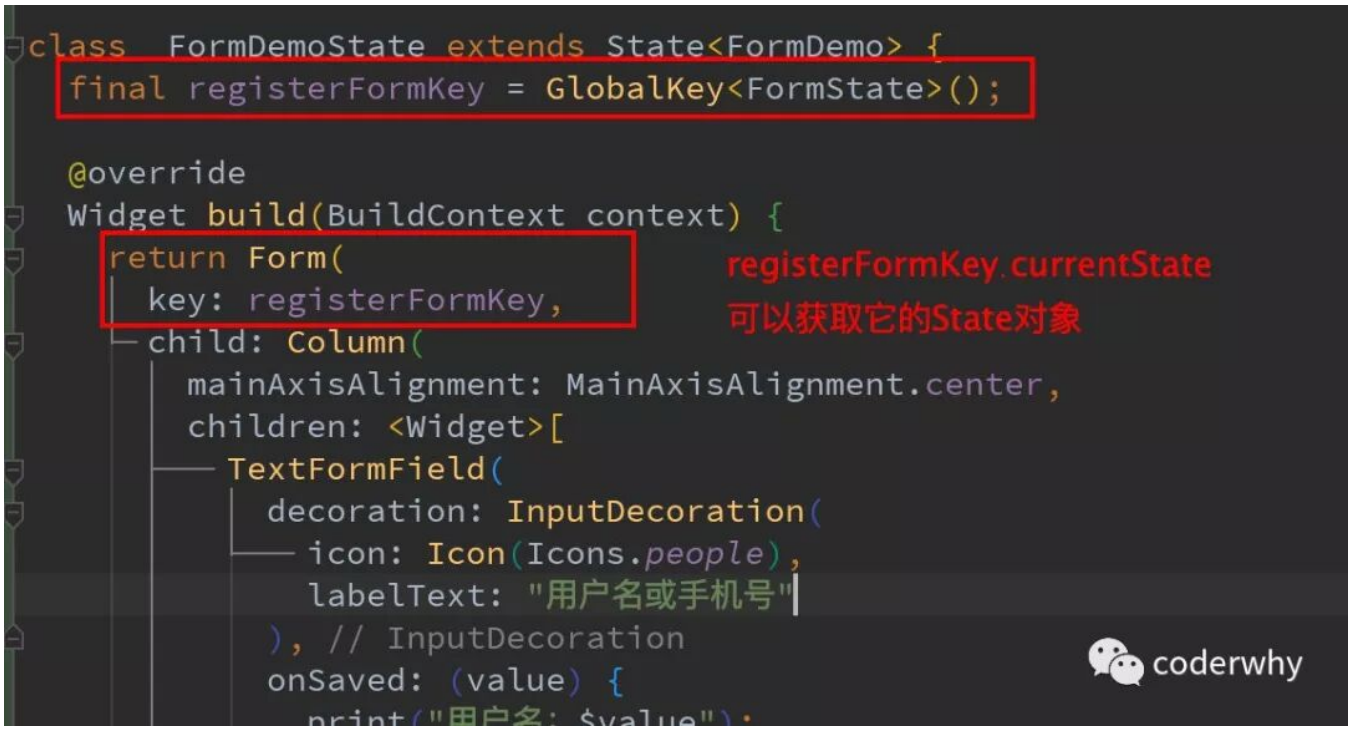

image-20190923202433788

案例代码演练：

```dart
class FormDemo extends StatefulWidget {
  @override
  _FormDemoState createState() => _FormDemoState();
}

class _FormDemoState extends State<FormDemo> {
  final registerFormKey = GlobalKey<FormState>();
  String username, password;

  void registerForm() {
    registerFormKey.currentState.save();

    print("username:$username password:$password");
  }

  @override
  Widget build(BuildContext context) {
    return Form(
      key: registerFormKey,
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          TextFormField(
            decoration: InputDecoration(
              icon: Icon(Icons.people),
              labelText: "用户名或手机号"
            ),
            onSaved: (value) {
              this.username = value;
            },
          ),
          TextFormField(
            obscureText: true,
            decoration: InputDecoration(
              icon: Icon(Icons.lock),
              labelText: "密码"
            ),
            onSaved: (value) {
              this.password = value;
            },
          ),
          SizedBox(height: 16,),
          Container(
            width: double.infinity,
            height: 44,
            child: RaisedButton(
              color: Colors.lightGreen,
              child: Text("注 册", style: TextStyle(fontSize: 20, color: Colors.white),),
              onPressed: registerForm,
            ),
          )
        ],
      ),
    );
  }
}
```

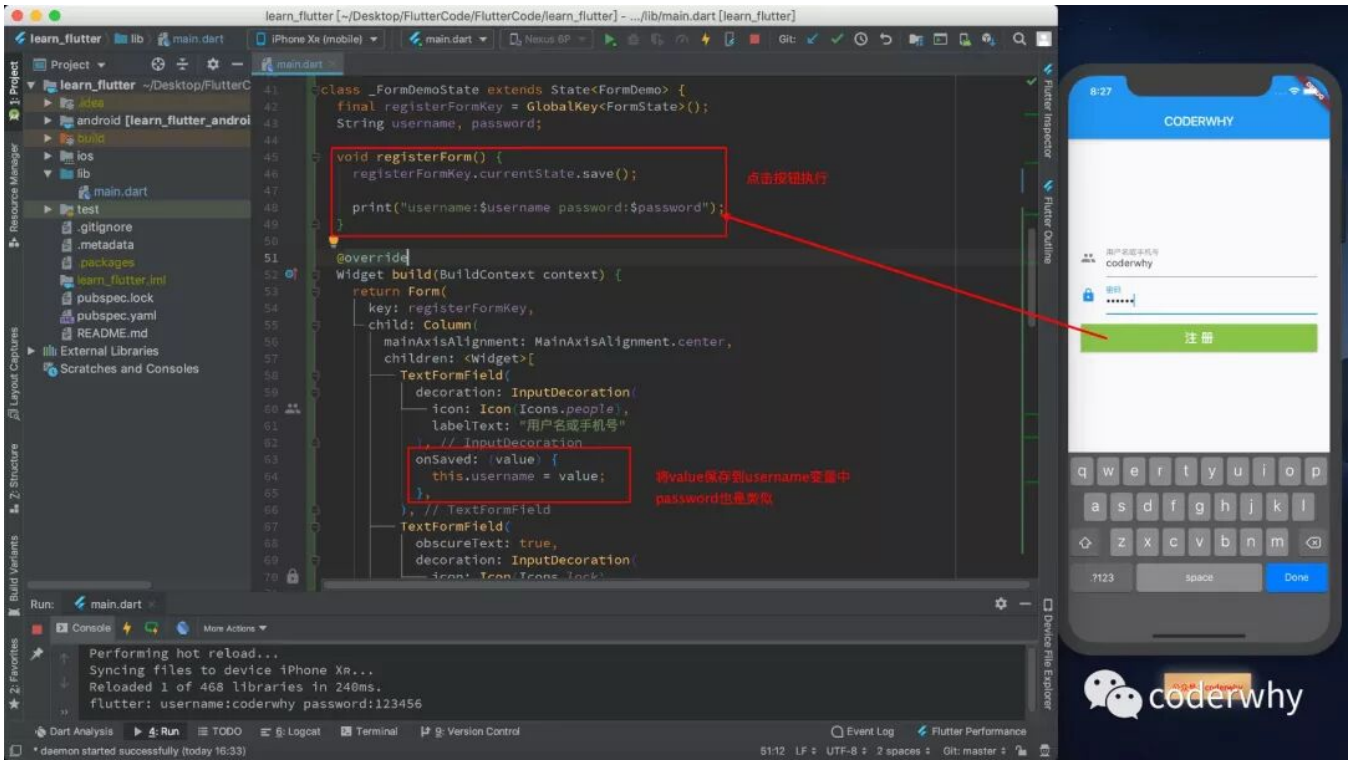

image-20190923202832219

4.2.3. 验证填写的表单数据

在表单中，我们可以添加 `验证器` ，如果不符合某些特定的规则，那么给用户一定的提示信息

比如我们需要账号和密码有这样的规则：账号和密码都不能为空。

按照如下步骤就可以完成整个验证过程：

- 1、为TextFormField添加validator的回调函数；
- 2、调用Form的State对象的validate方法，就会回调validator传入的函数；

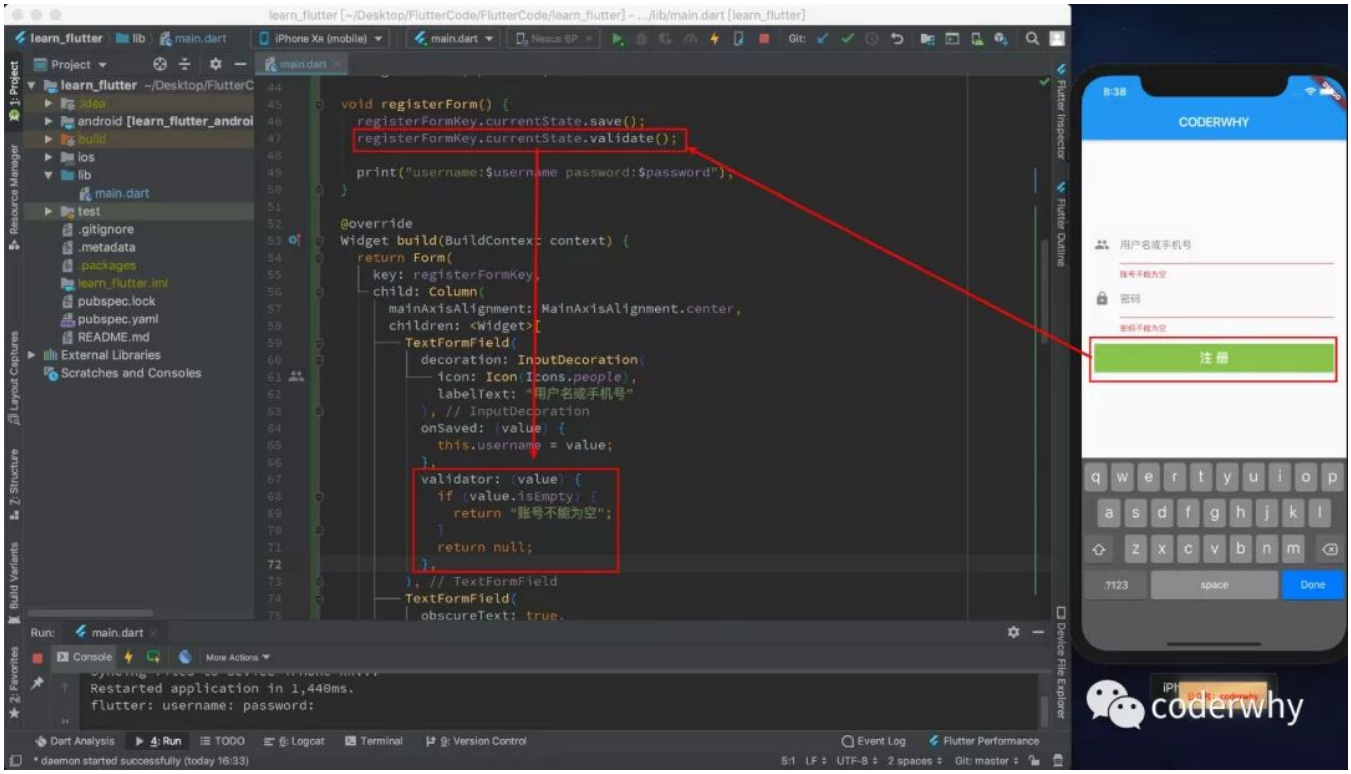

image-20190923203843492

也可以为TextFormField添加一个属性：autovalidate

- 不需要调用validate方法，会自动验证是否符合要求；

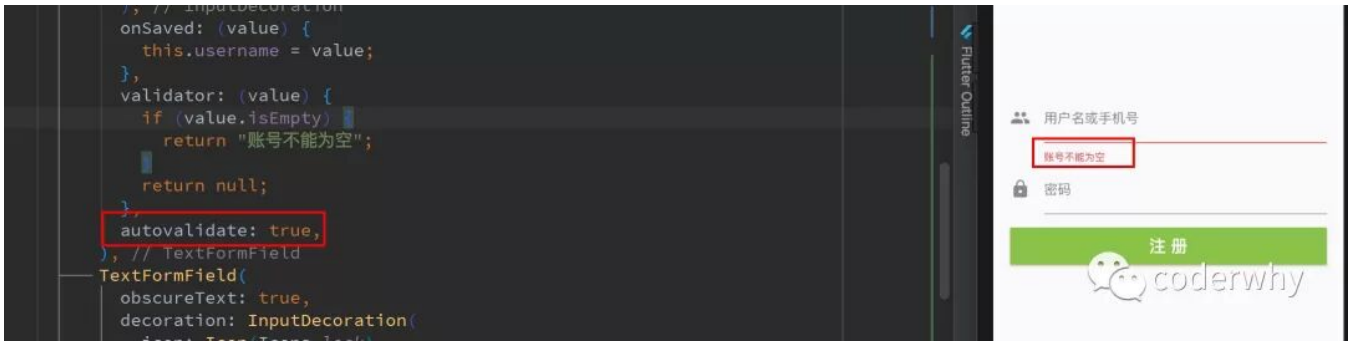

image-20190923204051768

> 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