Flutter(五)之Flutter滚动Widget

原创 coderwhy coderwhy

> 前言一：由于Mac档期问题，最后一周更新Flutter，下周开始更新TypeScript；
>
> **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：列表是移动端经常使用的一种视图展示方式，在Flutter中提供了ListView和GridView。
>
> 为了可能展示出更好的效果，我这里提供了一段Json数据，所以我们可以先学习一下Json解析。

一. JSON读取和解析

> 在开发中，我们经常会使用本地JSON或者从服务器请求数据后回去到JSON，拿到JSON后通常会将JSON转成Model对象来进行后续的操作，因为这样操作更加的方便，也更加的安全。
>
> 所以学习JSON的相关操作以及读取JSON后如何转成Model对象对于Flutter开发也非常重要。

1.1. JSON资源配置

JSON也属于一种资源，所以在使用之前需要先进行相关的配置

我们之前在学习使用Image组件时，用到了本地图片，本地图片必须在 `pubspec.yaml` 中进行配置：

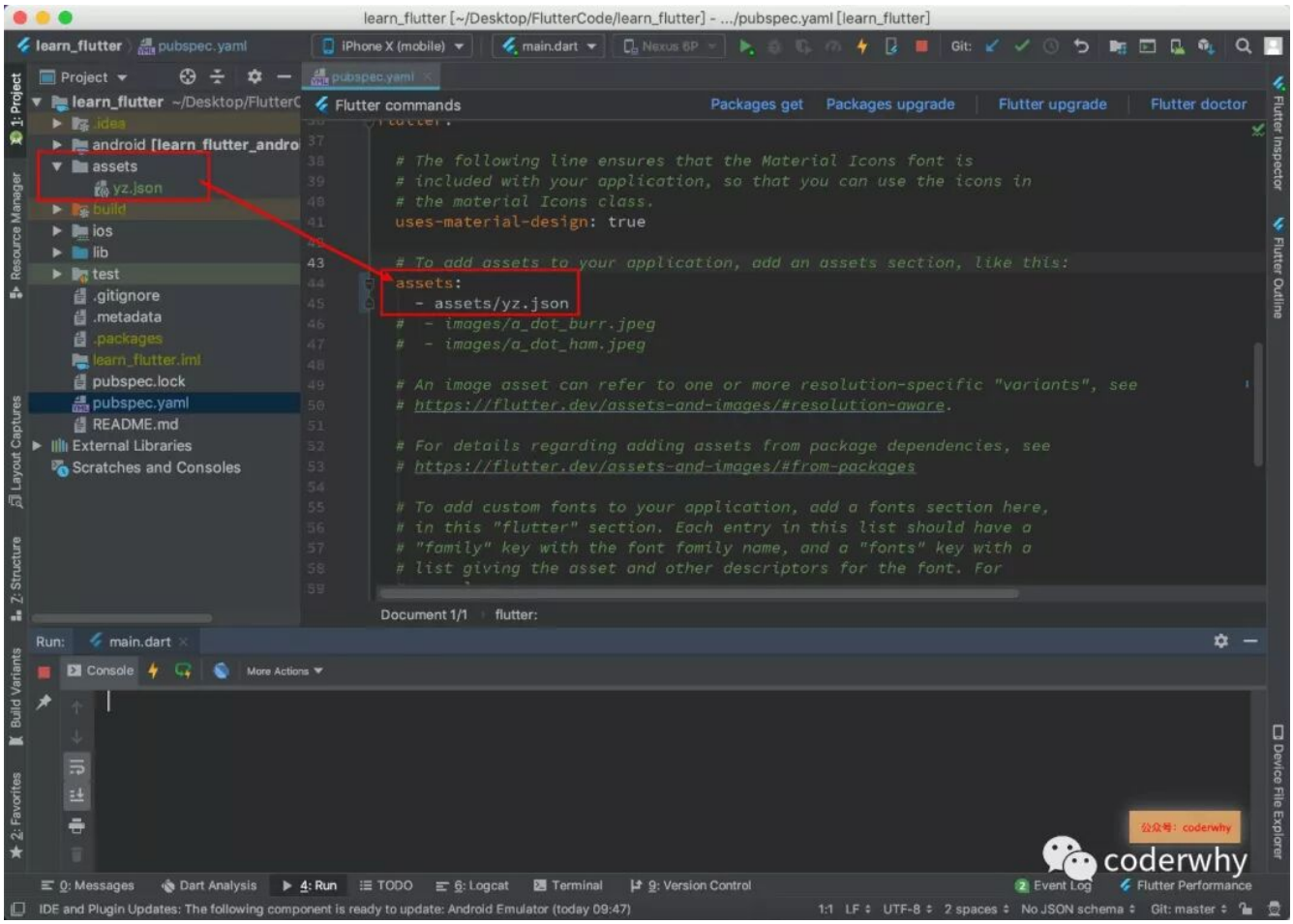

image-20190904144032396

1.2. JSON读取解析

JSON资源读取

如果我们希望读取JSON资源，可以使用 `package:flutter/services.dart` 包中的 `rootBundle` 。

在 `rootBundle` 中有一个 `loadString` 方法，可以去加载JSON资源

- 但是注意，查看该方法的源码，你会发现这个操作是一个异步的。
- 关于Future和async，这里就不再展开讲解，可以去查看之前的dart语法。

```
Future<String> loadString(String key, { bool cache = true }) async
{
  ...省略具体代码，可以自行查看源码
}
```

代码如下：(不要试图拷贝这个代码去运行，是没办法运行的)

```
import 'package:flutter/services.dart' show rootBundle;

// 打印读取的结果是一个字符串
rootBundle.loadString("assets/yz.json").then((value) => print(value));
```

JSON字符串转化

拿到JSON字符串后，我们需要将其转成成我们熟悉的List和Map类型。

我们可以通过 `dart:convert` 包中的 `json.decode` 方法将其进行转化

代码如下：

```
// 1.读取json文件
String jsonString = await rootBundle.loadString("assets/yz.json")
;

// 2.转成List或Map类型
final jsonResult = json.decode(jsonString);
```

对象Model定义

将JSON转成了List和Map类型后，就可以将List中的一个个Map转成Model对象，所以我们需要定义自己的Model

```
class Anchor {
  String nickname;
  String roomName;
  String imageUrl;

  Anchor({
    this.nickname,
    this.roomName,
    this.imageUrl
  });

  Anchor.withMap(Map<String, dynamic> parsedMap) {
    this.nickname = parsedMap["nickname"];
    this.roomName = parsedMap["roomName"];
    this.imageUrl = parsedMap["roomSrc"];
  }
}
```

1.3. JSON解析代码

上面我们给出了解析的一个个步骤，下面我们给出完整的代码逻辑

这里我单独创建了一个anchor.dart的文件，在其中定义了所有的相关代码：

- 之后外界只需要调用我内部的 `getAnchors` 就可以获取到解析后的数据了

```dart
import 'package:flutter/services.dart' show rootBundle;
import 'dart:convert';
import 'dart:async';

class Anchor {
  String nickname;
  String roomName;
  String imageUrl;

  Anchor({
    this.nickname,
    this.roomName,
    this.imageUrl
  });

  Anchor.withMap(Map<String, dynamic> parsedMap) {
    this.nickname = parsedMap["nickname"];
    this.roomName = parsedMap["roomName"];
    this.imageUrl = parsedMap["roomSrc"];
  }
}

Future<List<Anchor>> getAnchors() async {
  // 1.读取json文件
  String jsonString = await rootBundle.loadString("assets/yz.json")
;

  // 2.转成List或Map类型
  final jsonResult = json.decode(jsonString);

  // 3.遍历List，并且转成Anchor对象放到另一个List中
  List<Anchor> anchors = new List();
  for (Map<String, dynamic> map in jsonResult) {
    anchors.add(Anchor.withMap(map));
  }
  return anchors;
}
```

二. ListView组件

> 移动端数据量比较大时，我们都是通过列表来进行展示的，比如商品数据、聊天列表、通信录、朋友圈等。
>
> 在Android中，我们可以使用ListView或RecyclerView来实现，在iOS中，我们可以通过UITableView来实现。
>
> 在Flutter中，我们也有对应的列表Widget，就是ListView。

2.1. ListView基础

2.1.1 ListView基本使用

ListView可以沿一个方向（垂直或水平方向，默认是垂直方向）来排列其所有子Widget。

一种最简单的使用方式是直接将所有需要排列的子Widget放在ListView的children属性中即可。

我们来看一下直接使用ListView的代码演练：

- 为了让文字之间有一些间距，我使用了Padding Widget

```
class MyHomeBody extends StatelessWidget {
  final TextStyle textStyle = TextStyle(fontSize: 20, color: Colors.redAccent);

  @override
  Widget build(BuildContext context) {
    return ListView(
      children: <Widget>[
        Padding(
          padding: const EdgeInsets.all(8.0),
          child: Text("人的一切痛苦，本质上都是对自己无能的愤怒。", style: textStyle),
        ),
        Padding(
          padding: const EdgeInsets.all(8.0),
          child: Text("人活在世界上，不可以有偏差；而且多少要费点劲儿，才能把自己保持到理性的轨道上。", style: textStyle),
        ),
        Padding(
          padding: const EdgeInsets.all(8.0),
          child: Text("我活在世上，无非想要明白些道理，遇见些有趣的事。", style: textStyle),
        )
      ],
    );
  }
}
```

2.2.2. ListTile的使用

在开发中，我们经常见到一种列表，有一个图标或图片（Icon），有一个标题（Title），有一个子标题（Subtitle），还有尾部一个图标（Icon）。

这个时候，我们可以使用ListTile来实现：

```
class MyHomeBody extends StatelessWidget {

  @override
  Widget build(BuildContext context) {
    return ListView(
      children: <Widget>[
        ListTile(
          leading: Icon(Icons.people, size: 36,),
          title: Text("联系人"),
          subtitle: Text("联系人信息"),
          trailing: Icon(Icons.arrow_forward_ios),
        ),
        ListTile(
          leading: Icon(Icons.email, size: 36,),
          title: Text("邮箱"),
          subtitle: Text("邮箱地址信息"),
          trailing: Icon(Icons.arrow_forward_ios),
        ),
        ListTile(
          leading: Icon(Icons.message, size: 36,),
          title: Text("消息"),
          subtitle: Text("消息详情信息"),
          trailing: Icon(Icons.arrow_forward_ios),
        ),
        ListTile(
          leading: Icon(Icons.map, size: 36,),
          title: Text("地址"),
          subtitle: Text("地址详情信息"),
          trailing: Icon(Icons.arrow_forward_ios),
        )
      ],
    );
  }
}
```

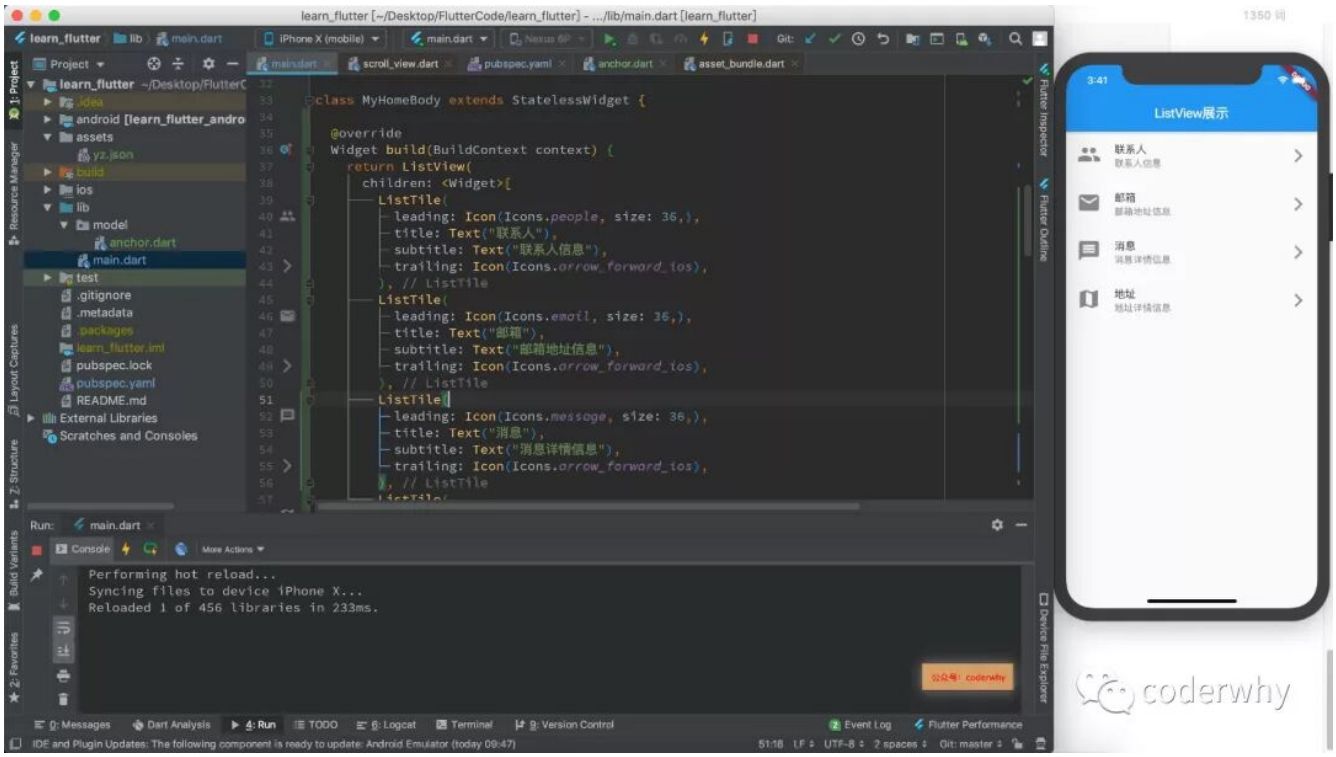

image-20190904154142133

2.2.3. 垂直方向滚动

我们可以通过设置 `scrollDirection` 参数来控制视图的滚动方向。

我们通过下面的代码实现一个水平滚动的内容：

- 这里需要注意，我们需要给Container设置width，否则它是没有宽度的，就不能正常显示。
- 或者我们也可以给ListView设置一个itemExtent，该属性会设置滚动方向上每个item所占据的宽度。

```
class MyHomeBody extends StatelessWidget {

  @override
  Widget build(BuildContext context) {
    return ListView(
      scrollDirection: Axis.horizontal,
      itemExtent: 200,
      children: <Widget>[
        Container(color: Colors.red, width: 200),
        Container(color: Colors.green, width: 200),
        Container(color: Colors.blue, width: 200),
        Container(color: Colors.purple, width: 200)
,
        Container(color: Colors.orange, width: 200)
,
      ],
    );
  }
}
```

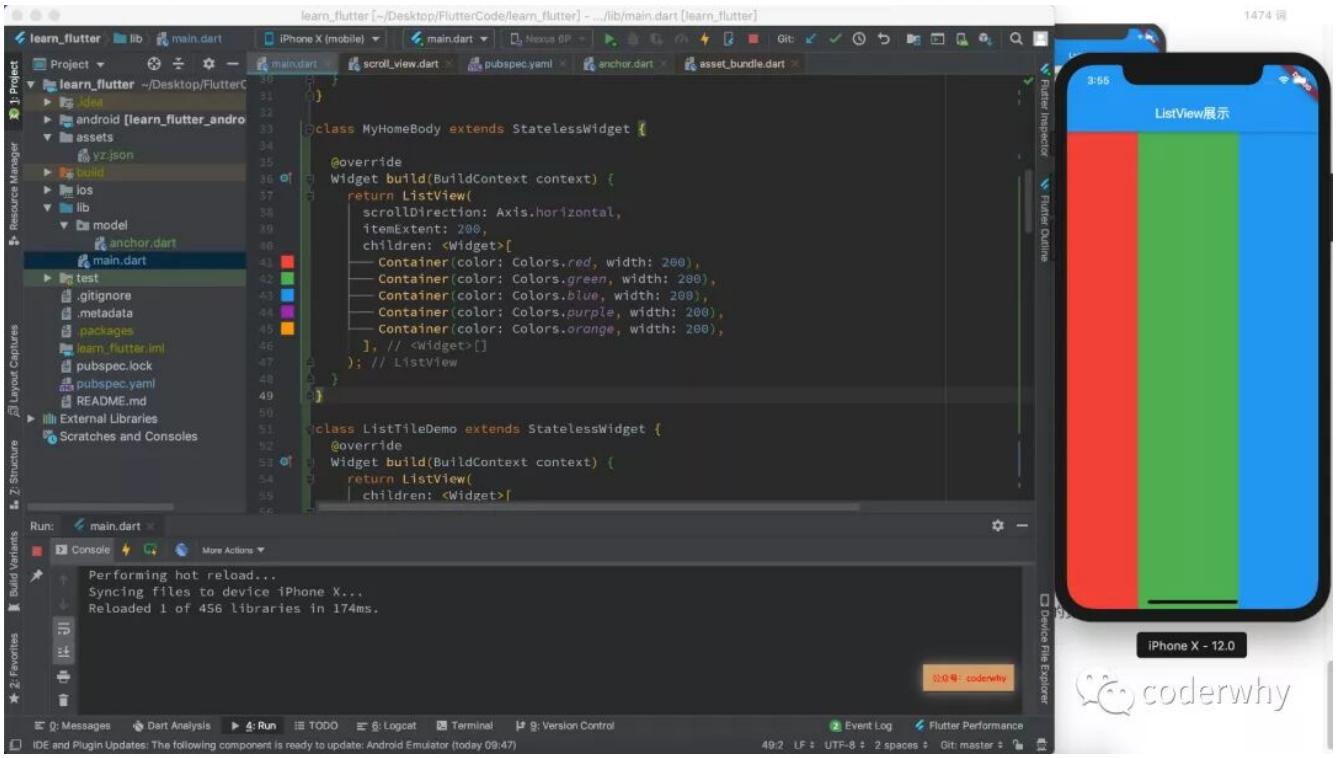

image-20190904155511007

2.2. ListView.build

> 通过构造函数中的children传入所有的子Widget有一个问题：默认会创建出所有的子Widget。
>
> 但是对于用户来说，一次性构建出所有的Widget并不会有什么差异，但是对于我们的程序来说会产生性能问题，而且会增加首屏的渲染时间。
>
> 我们可以ListView.build来构建子Widget，提供性能。

2.2.1. ListView.build基本使用

ListView.build适用于子Widget比较多的场景，该构造函数将创建子Widget交给了一个抽象的方法，交给ListView进行管理，ListView会在真正需要的时候去创建子Widget，而不是一开始就全部初始化好。

该方法有两个重要参数：

- itemBuilder：列表项创建的方法。当列表滚动到对应位置的时候，ListView会自动调用该方法来创建对应的子Widget。类型是IndexedWidgetBuilder，是一个函数类型。
- itemCount：表示列表项的数量，如果为空，则表示ListView为无限列表。

我们还是通过一个简单的案例来认识它：

```
class MyHomeBody extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return ListView.builder(
      itemCount: 100,
      itemExtent: 80,
      itemBuilder: (BuildContext context, int index) {
        return ListTile(title: Text("标题$index"), subtitle: Text("详情内容$index"))
;
      }
    );
  }
}
```

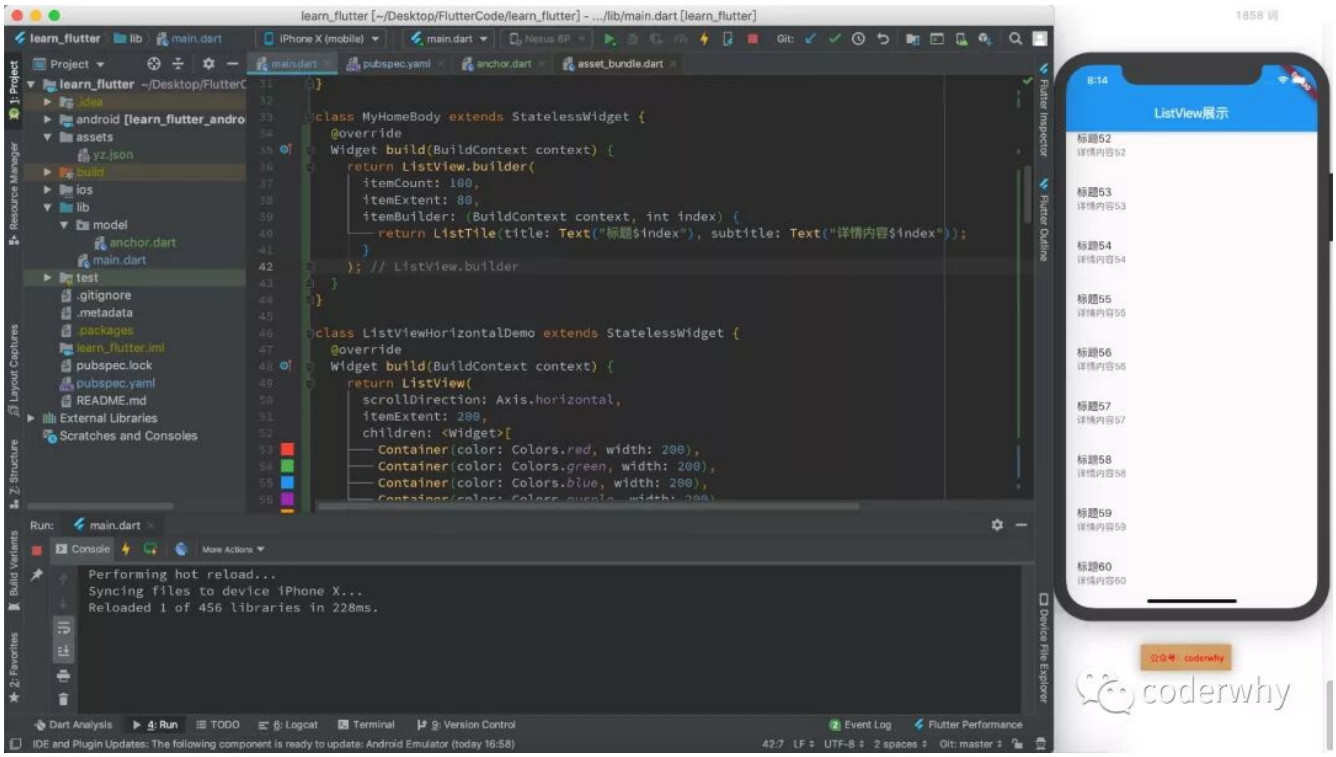

image-20190904201433097

2.2.2. ListView.build动态数据

在之前，我们搞了一个yz.json数据，我们现在动态的来通过JSON数据展示一个列表。

思考：这个时候是否依然可以使用 `StatelessWidget` ：

答案：不可以，因为当前我们的数据是异步加载的，刚开始界面并不会展示数据（没有数据），后面从JSON中加载出来数据（有数据）后，再次展示加载的数据。

- 这里是有状态的变化的，从无数据，到有数据的变化。
- 这个时候，我们需要使用 `StatefulWidget` 来管理组件。

展示代码如下：

```dart
import 'model/anchor.dart';

...省略中间代码
class MyHomeBody extends StatefulWidget {
  @override
  State<StatefulWidget> createState() {
    return MyHomeBodyState();
  }
}

class MyHomeBodyState extends State<MyHomeBody> {
  List<Anchor> anchors = [];

  // 在初始化状态的方法中加载数据
  @override
  void initState() {
    getAnchors().then((anchors) {
      setState(() {
        this.anchors = anchors;
      });
    });

    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    return ListView.builder(
      itemBuilder: (BuildContext context, int index) {
        return Padding(
          padding: EdgeInsets.all(8),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: <Widget>[
              Image.network(
                anchors[index].imageUrl,
                fit: BoxFit.fitWidth,
                width: MediaQuery.of(context).size.width,
              ),
              SizedBox(height: 8),
              Text(anchors[index].nickname, style: TextStyle(fontSize: 20),)
,
              SizedBox(height: 5),
              Text(anchors[index].roomName)
            ],
          ),
        );
      },
    );
  }
}
```

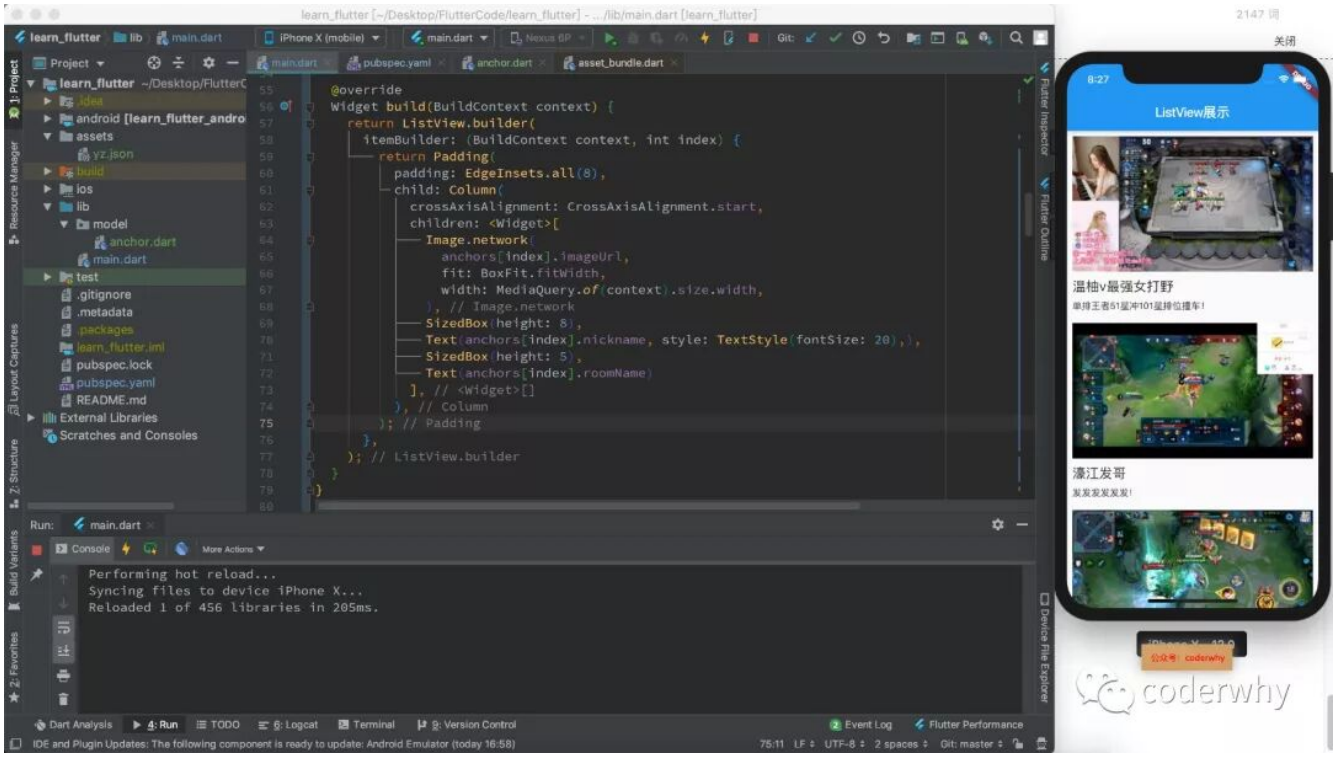

image-20190904202722900

2.2.3. ListView.separated

`ListView.separated` 可以生成列表项之间的分割器，它除了比 `ListView.builder` 多了一个 `separatorBuilder` 参数，该参数是一个分割器生成器。

下面我们看一个例子：奇数行添加一条蓝色下划线，偶数行添加一条红色下划线：

```
class MySeparatedDemo extends StatelessWidget {
  Divider blueColor = Divider(color: Colors.blue);
  Divider redColor = Divider(color: Colors.red);

  @override
  Widget build(BuildContext context) {
    return ListView.separated(
      itemBuilder: (BuildContext context, int index) {
        return ListTile(
          leading: Icon(Icons.people),
          title: Text("联系人${index+1}"),
          subtitle: Text("联系人电话${index+1}"),
        );
      },
      separatorBuilder: (BuildContext context, int index)
{
        return index % 2 == 0 ? redColor : blueColor;
      },
      itemCount: 100
    );
  }
}
```

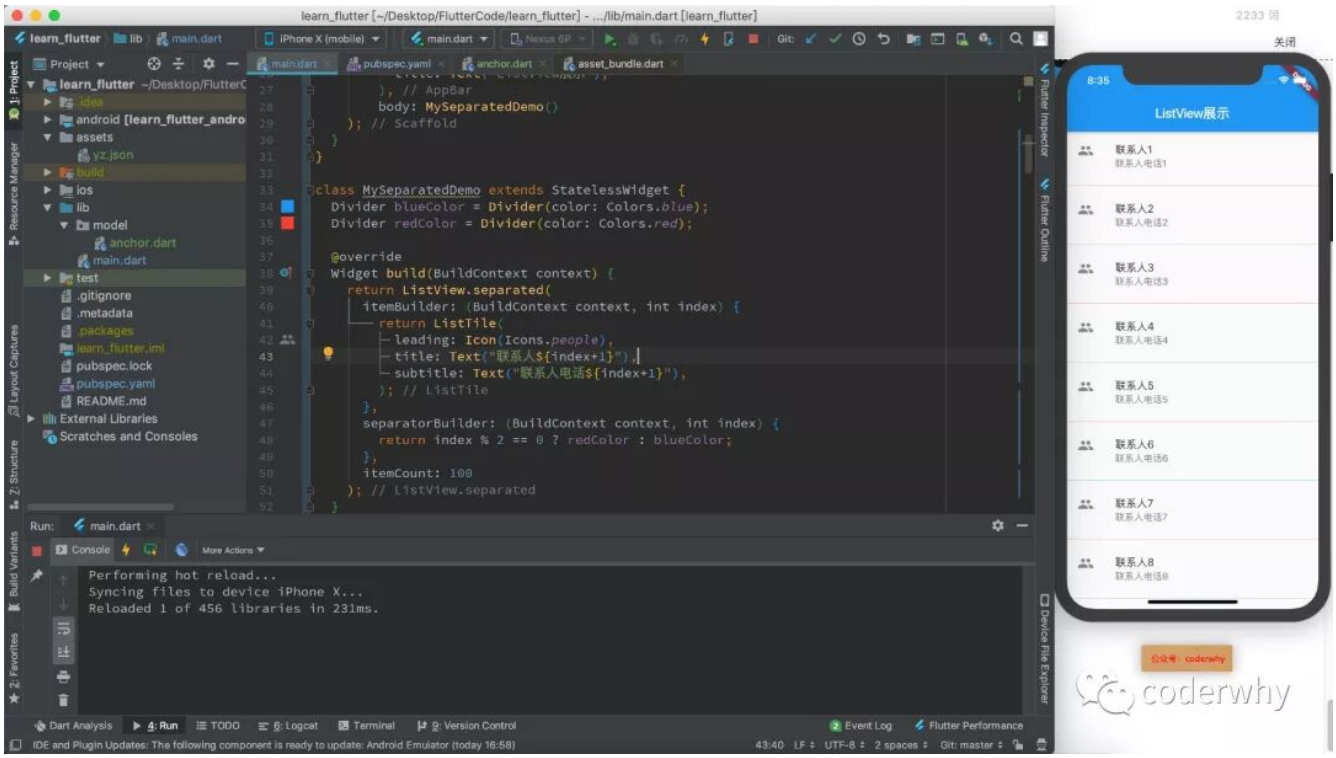

image-20190904203539235

三. GridView组件

> GridView用于展示多列的展示，在开发中也非常常见，比如直播App中的主播列表、电商中的商品列表等等。
>
> 在Flutter中我们可以使用GridView来实现，使用方式和ListView也比较相似。

3.1. GridView构造函数

我们先学习GridView构造函数的使用方法

一种使用GridView的方式就是使用构造函数来创建，和ListView对比有一个特殊的参数：`gridDelegate`

`gridDelegate` 用于控制交叉轴的item数量或者宽度，需要传入的类型是SliverGridDelegate，但是它是一个抽象类，所以我们需要传入它的子类：

SliverGridDelegateWithFixedCrossAxisCount

```
SliverGridDelegateWithFixedCrossAxisCount({
  @required double crossAxisCount, // 交叉轴的item个数
  double mainAxisSpacing = 0.0, // 主轴的间距
  double crossAxisSpacing = 0.0, // 交叉轴的间距
  double childAspectRatio = 1.0, // 子Widget的宽高比
})
```

代码演练：

```dart
class MyGridCountDemo extends StatelessWidget {

  List<Widget> getGridWidgets() {
    return List.generate(100, (index) {
      return Container(
        color: Colors.purple,
        alignment: Alignment(0, 0),
        child: Text("item$index", style: TextStyle(fontSize: 20, color: Colors.white)),
      );
    });
  }

  @override
  Widget build(BuildContext context) {
    return GridView(
      gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
        crossAxisCount: 3,
        mainAxisSpacing: 10,
        crossAxisSpacing: 10,
        childAspectRatio: 1.0
      ),
      children: getGridWidgets(),
    );
  }
}
```

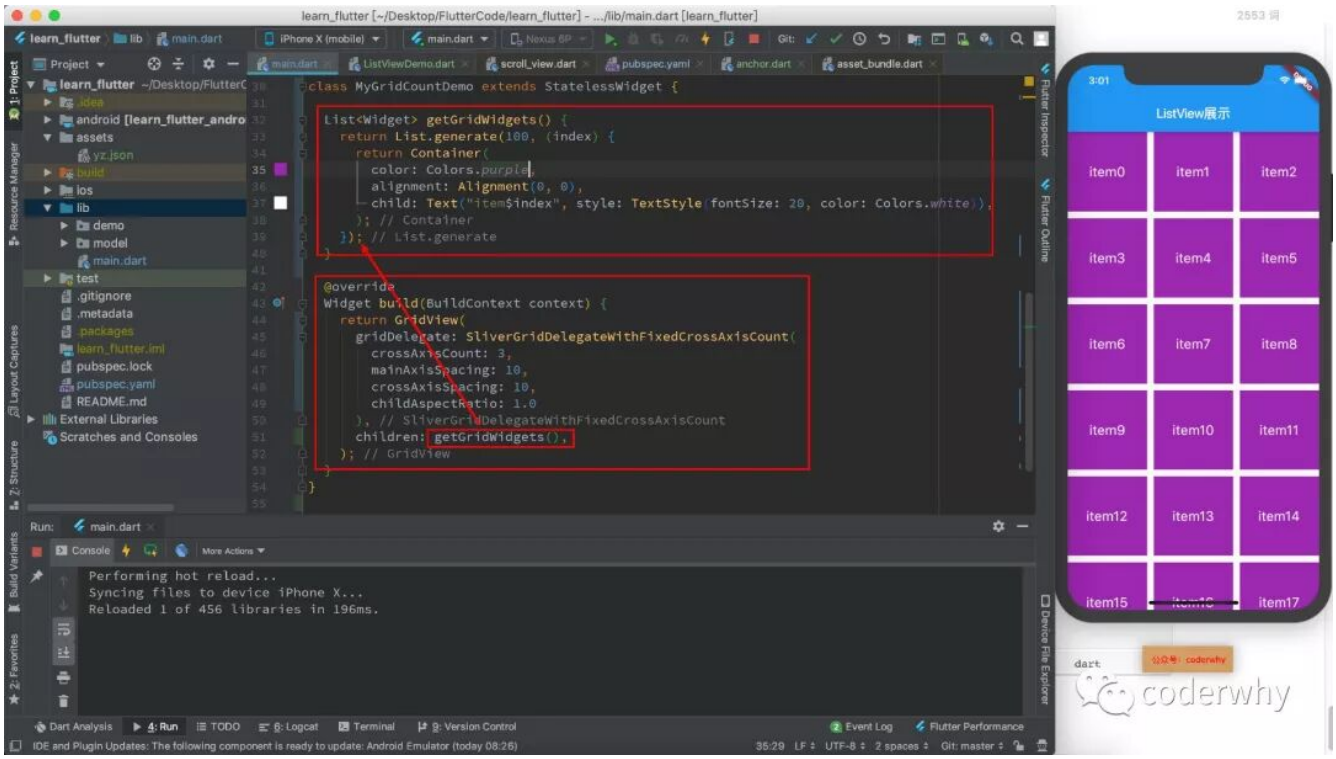

image-20190905150128115

SliverGridDelegateWithMaxCrossAxisExtent

```dart
SliverGridDelegateWithMaxCrossAxisExtent({
  double maxCrossAxisExtent, // 交叉轴的item宽度
  double mainAxisSpacing = 0.0, // 主轴的间距
  double crossAxisSpacing = 0.0, // 交叉轴的间距
  double childAspectRatio = 1.0, // 子Widget的宽高比
})
```

代码演练：

```
class MyGridExtentDemo extends StatelessWidget {

  List<Widget> getGridWidgets() {
    return List.generate(100, (index) {
      return Container(
        color: Colors.purple,
        alignment: Alignment(0, 0),
        child: Text("item$index", style: TextStyle(fontSize: 20, color: Colors.white)),
      );
    });
  }

  @override
  Widget build(BuildContext context) {
    return GridView(
      gridDelegate: SliverGridDelegateWithMaxCrossAxisExtent(
        maxCrossAxisExtent: 150,
        mainAxisSpacing: 10,
        crossAxisSpacing: 10,
        childAspectRatio: 1.0
      ),
      children: getGridWidgets(),
    );
  }
}
```

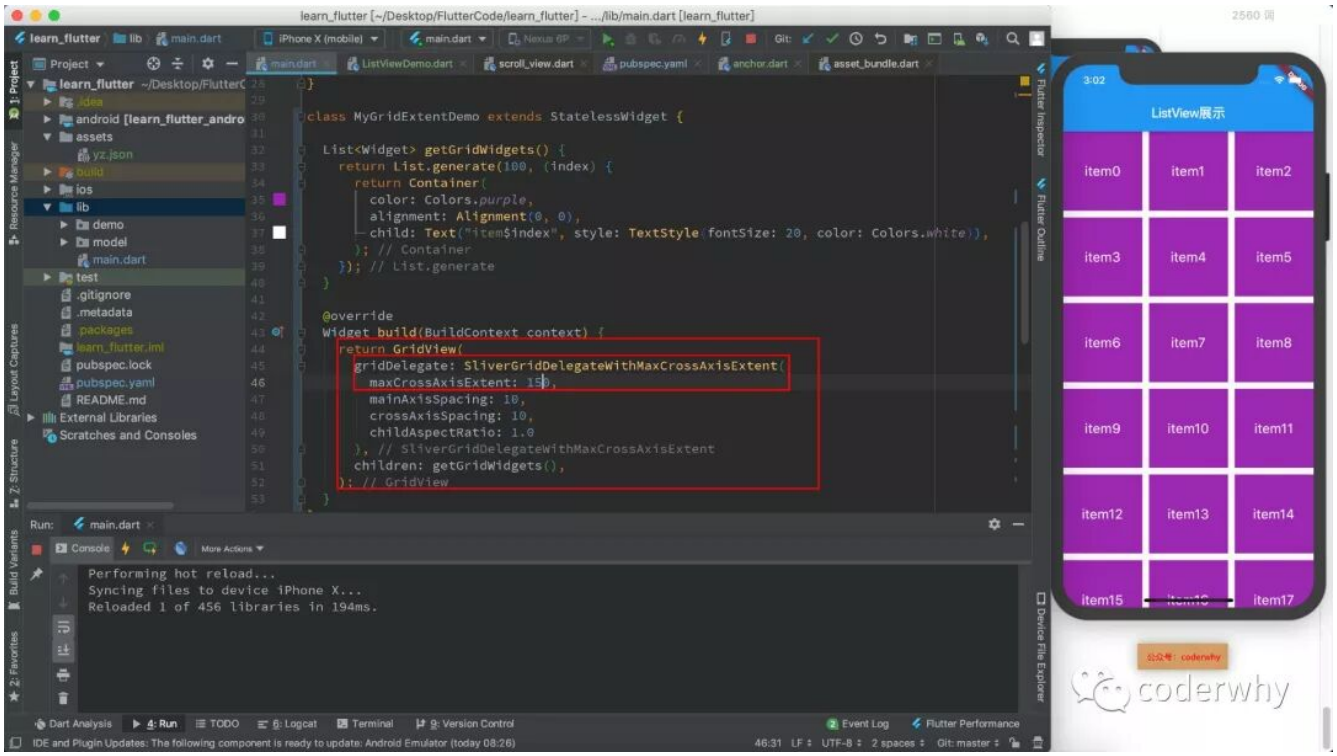

image-20190905150313598

前面两种方式也可以不设置delegate

可以分别使用：`GridView.count构造函数` 和 `GridView.extent` 构造函数实现相同的效果，这里不再赘述。

3.2. GridView.build

和ListView一样，使用构造函数会一次性创建所有的子Widget，会带来性能问题，所以我们可以使用 `GridView.build` 来交给GridView自己管理需要创建的子Widget。

我们直接使用之前的数据来进行代码演练：

```dart
class _GridViewBuildDemoState extends State<GridViewBuildDemo> {
  List<Anchor> anchors = [];

  @override
  void initState() {
    getAnchors().then((anchors) {
      setState(() {
        this.anchors = anchors;
      });
    });
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.all(8.0),
      child: GridView.builder(
        shrinkWrap: true,
        physics: ClampingScrollPhysics(),
        gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
          crossAxisCount: 2,
          mainAxisSpacing: 10,
          crossAxisSpacing: 10,
          childAspectRatio: 1.2
        ),
        itemCount: anchors.length,
        itemBuilder: (BuildContext context, int index) {
          return Container(
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: <Widget>[
                Image.network(anchors[index].imageUrl),
                SizedBox(height: 5),
                Text(anchors[index].nickname, style: TextStyle(fontSize: 16),),
                Text(anchors[index].roomName, maxLines: 1, overflow: TextOverflow.ellipsis,)
              ],
            ),
          );
        }
      ),
    );
  }
}
```

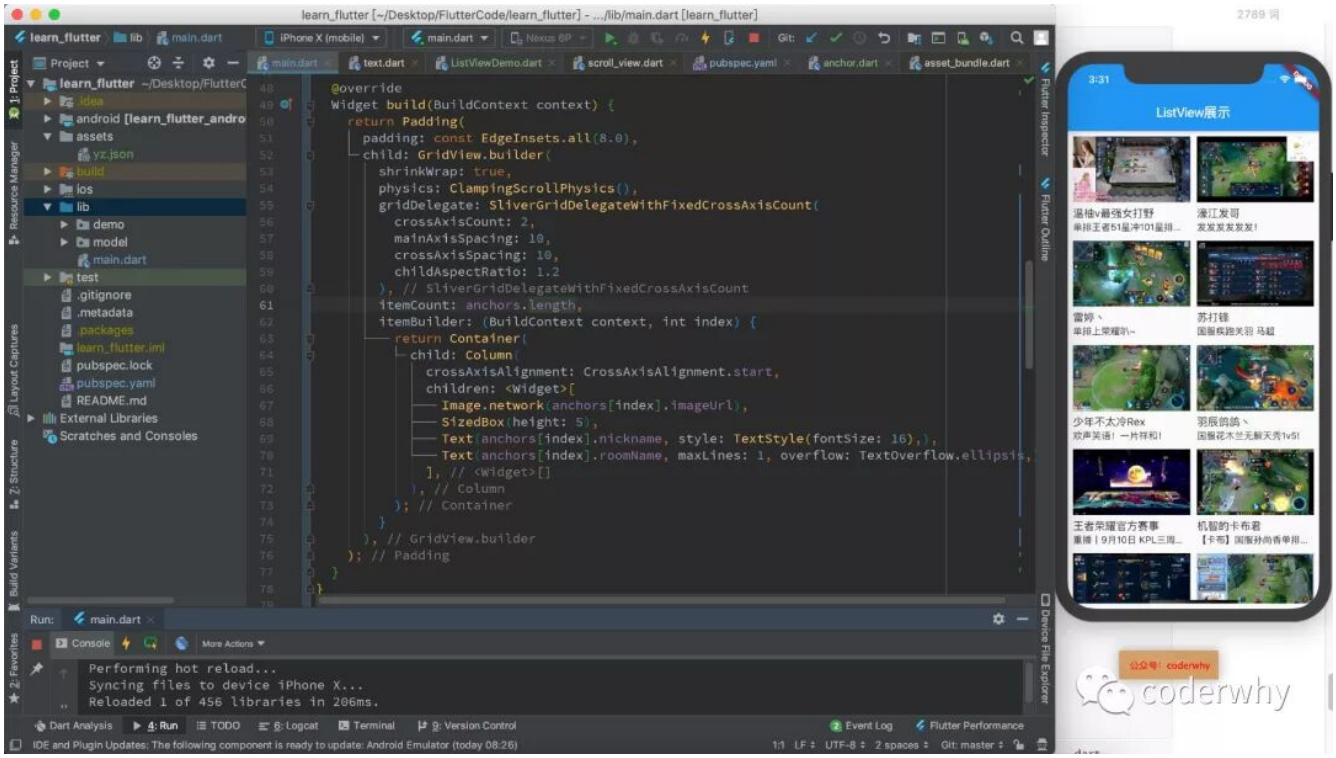

image-20190905153201888

四. Slivers

我们考虑一个这样的布局：一个滑动的视图中包括一个标题视图（HeaderView），一个列表视图（ListView），一个网格视图（GridView）。

我们怎么可以让它们做到统一的滑动效果呢？使用前面的滚动是很难做到的。

Flutter中有一个可以完成这样滚动效果的Widget：CustomScrollView，可以统一管理多个滚动视图。

在CustomScrollView中，每一个独立的，可滚动的Widget被称之为Sliver。

补充：Sliver可以翻译成裂片、薄片，你可以将每一个独立的滚动视图当做一个小裂片。

4.1. Slivers的基本使用

因为我们需要把很多的Sliver放在一个CustomScrollView中，所以CustomScrollView有一个slivers属性，里面让我们放对应的一些Sliver：

- SliverList：类似于我们之前使用过的ListView；
- SliverFixedExtentList：类似于SliverList只是可以设置滚动的高度；
- SliverGrid：类似于我们之前使用过的GridView；
- SliverPadding：设置Sliver的内边距，因为可能要单独给Sliver设置内边距；
- SliverAppBar：添加一个AppBar，通常用来作为CustomScrollView的HeaderView；
- SliverSafeArea：设置内容显示在安全区域（比如不让齐刘海挡住我们的内容）

我们简单演示一下：SliverGrid+SliverPadding+SliverSafeArea的组合

```
class HomeContent extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return CustomScrollView(
      slivers: <Widget>[
        SliverSafeArea(
          sliver: SliverPadding(
            padding: EdgeInsets.all(8),
            sliver: SliverGrid(
              gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
                crossAxisCount: 2,
                crossAxisSpacing: 8,
                mainAxisSpacing: 8,
              ),
              delegate: SliverChildBuilderDelegate(
                (BuildContext context, int index) {
                  return Container(
                    alignment: Alignment(0, 0),
                    color: Colors.orange,
                    child: Text("item$index"),
                  );
                },
                childCount: 20
              ),
            ),
          ),
        )
      ],
    );
  }
}
```

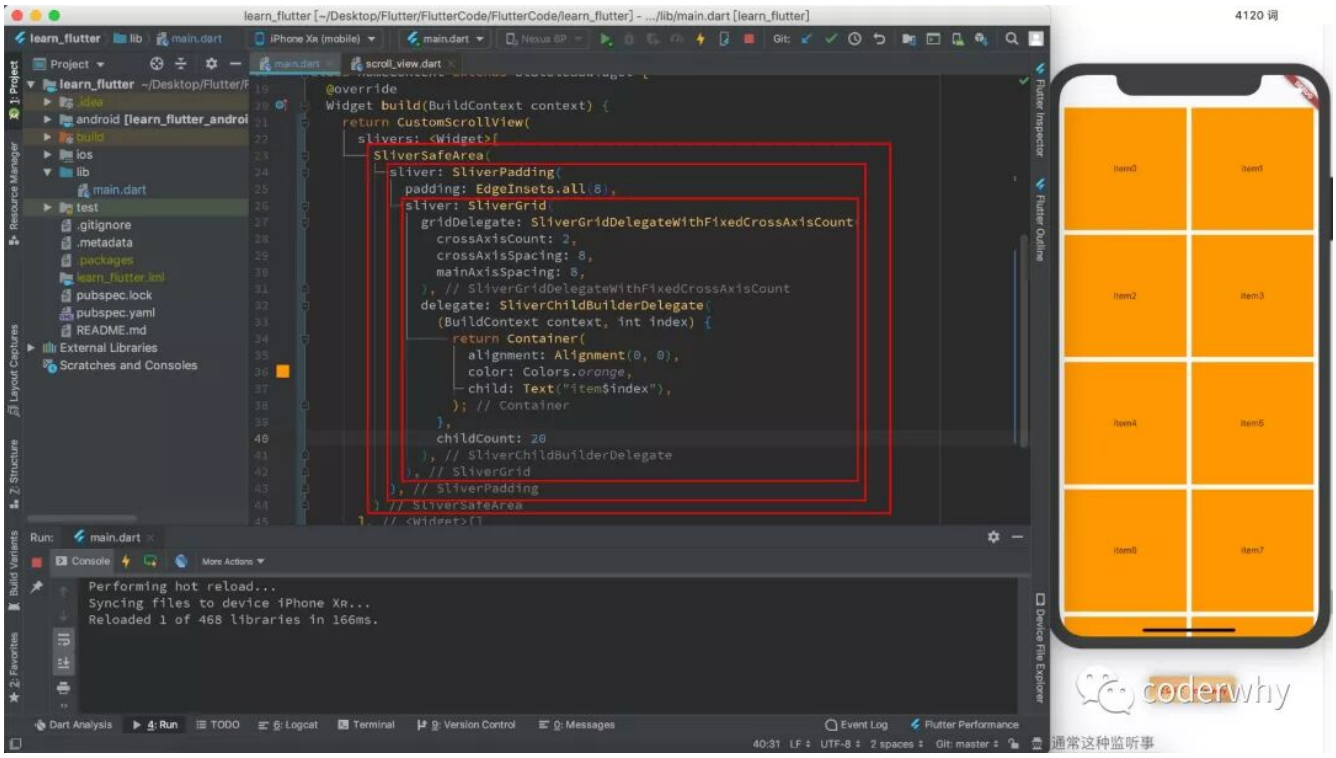

image-20191001180054605

4.2. Slivers的组合使用

这里我使用官方的示例程序，将SliverAppBar+SliverGrid+SliverFixedExtentList做出如下界面：

```dart
class HomeContent extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return showCustomScrollView();
  }

  Widget showCustomScrollView() {
    return new CustomScrollView(
      slivers: <Widget>[
        const SliverAppBar(
          expandedHeight: 250.0,
          flexibleSpace: FlexibleSpaceBar(
            title: Text('Coderwhy Demo'),
            background: Image(
              image: NetworkImage(
                "https://tva1.sinaimg.cn/large/006y8mN6gy1g72j6nk1d4j30u00k0n0j.jpg"
,
              ),
              fit: BoxFit.cover,
            ),
          ),
        ),
        new SliverGrid(
          gridDelegate: new SliverGridDelegateWithMaxCrossAxisExtent(
            maxCrossAxisExtent: 200.0,
            mainAxisSpacing: 10.0,
            crossAxisSpacing: 10.0,
            childAspectRatio: 4.0,
          ),
          delegate: new SliverChildBuilderDelegate(
                (BuildContext context, int index) {
              return new Container(
                alignment: Alignment.center,
                color: Colors.teal[100 * (index % 9)],
                child: new Text('grid item $index'),
              );
            },
            childCount: 10,
          ),
        ),
        SliverFixedExtentList(
          itemExtent: 50.0,
          delegate: SliverChildBuilderDelegate(
                (BuildContext context, int index) {
              return new Container(
                alignment: Alignment.center,
                color: Colors.lightBlue[100 * (index % 9)],
                child: new Text('list item $index'),
              );
            },
            childCount: 20
          ),
        ),
      ],
    );
  }
}
```

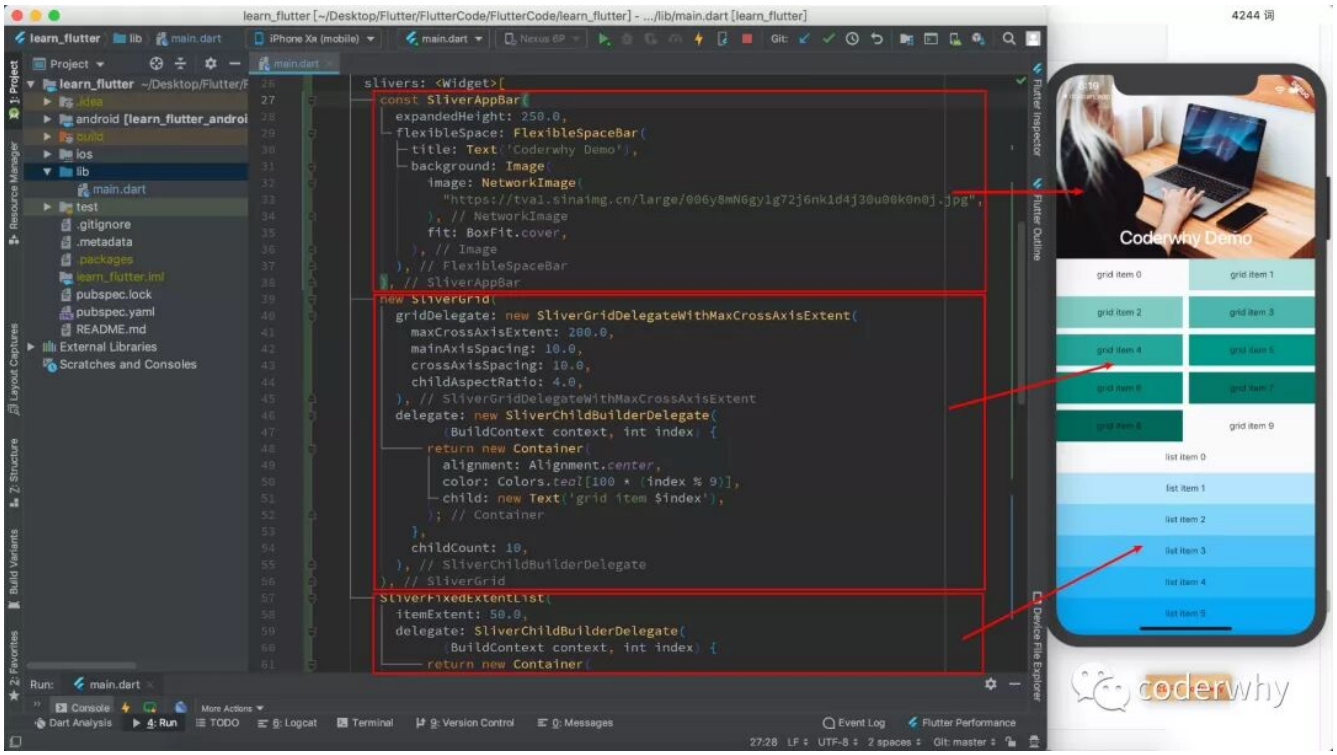

image-20191001181959972

五. 监听滚动事件

对于滚动的视图，我们经常需要监听它的一些滚动事件，在监听到的时候去做对应的一些事情。

比如视图滚动到底部时，我们可能希望做上拉加载更多；

比如滚动到一定位置时显示一个回到顶部的按钮，点击回到顶部的按钮，回到顶部；

比如监听滚动什么时候开始，什么时候结束；

在Flutter中监听滚动相关的内容由两部分组成：ScrollController和ScrollNotification。

5.1. ScrollController

在Flutter中，Widget并不是最终渲染到屏幕上的元素（真正渲染的是RenderObject），因此通常这种监听事件以及相关的信息并不能直接从Widget中获取，而是必须通过对应的Widget的Controller来实现。

ListView、GridView的组件控制器是ScrollController，我们可以通过它来获取视图的滚动信息，并且可以调用里面的方法来更新视图的滚动位置。

另外，通常情况下，我们会根据滚动的位置来改变一些Widget的状态信息，所以ScrollController通常会和StatefulWidget一起来使用，并且会在其中控制它的初始化、监听、销毁等事件。

我们来做一个案例，当滚动到1000位置的时候，显示一个回到顶部的按钮：

- `jumpTo(double offset)` 、 `animateTo(double offset,...)` ：这两个方法用于跳转到指定的位置，它们不同之处在于，后者在跳转时会执行一个动画，而前者不会。
- ScrollController间接继承自Listenable，我们可以根据ScrollController来监听滚动事件。

```dart
class MyHomePage extends StatefulWidget {
  @override
  State<StatefulWidget> createState() => MyHomePageState();
}

class MyHomePageState extends State<MyHomePage> {
  ScrollController _controller;
  bool _isShowTop = false;

  @override
  void initState() {
    // 初始化ScrollController
    _controller = ScrollController();

    // 监听滚动
    _controller.addListener(() {
      var tempSsShowTop = _controller.offset >= 1000;
      if (tempSsShowTop != _isShowTop) {
        setState(() {
          _isShowTop = tempSsShowTop;
        });
      }
    });

    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("ListView展示"),
      ),
      body: ListView.builder(
        itemCount: 100,
        itemExtent: 60,
        controller: _controller,
        itemBuilder: (BuildContext context, int index) {
          return ListTile(title: Text("item$index"));
        }
      ),
      floatingActionButton: !_isShowTop ? null : FloatingActionButton(
        child: Icon(Icons.arrow_upward),
        onPressed: () {
          _controller.animateTo(0, duration: Duration(milliseconds: 1000), curve: Curves.ease);
        },
      ),
    );
  }
}
```

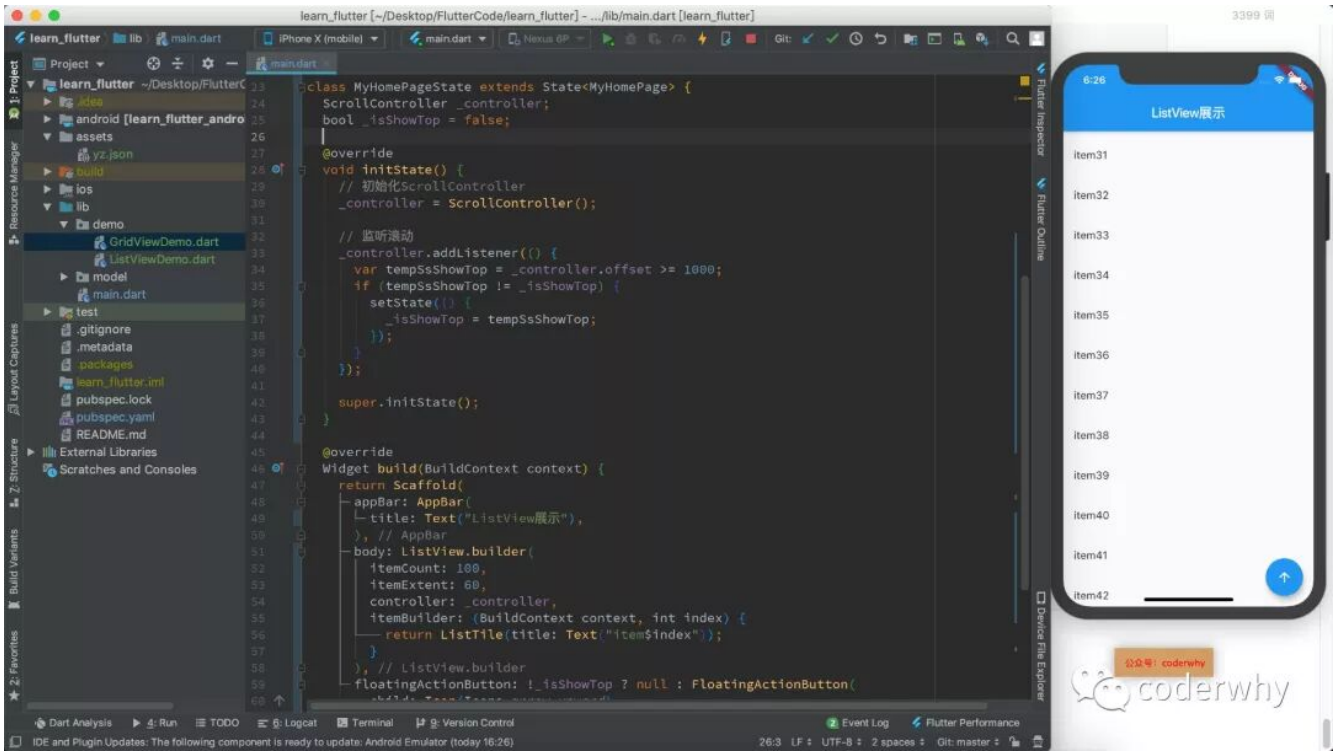

image-20190905182638821

5.2. NotificationListener

如果我们希望监听什么时候开始滚动，什么时候结束滚动，这个时候我们可以通过 `NotificationListener` 。

- NotificationListener是一个Widget，模板参数T是想监听的通知类型，如果省略，则所有类型通知都会被监听，如果指定特定类型，则只有该类型的通知会被监听。
- NotificationListener需要一个onNotification回调函数，用于实现监听处理逻辑。
- 该回调可以返回一个布尔值，代表是否阻止该事件继续向上冒泡，如果为 `true` 时，则冒泡终止，事件停止向上传播，如果不返回或者返回值为 `false` 时，则冒泡继续。

案例: 列表滚动, 并且在中间显示滚动进度

```dart
class MyHomeNotificationDemo extends StatefulWidget {
  @override
  State<StatefulWidget> createState() => MyHomeNotificationDemoState();
}

class MyHomeNotificationDemoState extends State<MyHomeNotificationDemo> {
  int _progress = 0;

  @override
  Widget build(BuildContext context) {
    return NotificationListener(
      onNotification: (ScrollNotification notification) {
        // 1.判断监听事件的类型
        if (notification is ScrollStartNotification) {
          print("开始滚动.....");
        } else if (notification is ScrollUpdateNotification) {
          // 当前滚动的位置和总长度
          final currentPixel = notification.metrics.pixels;
          final totalPixel = notification.metrics.maxScrollExtent;
          double progress = currentPixel / totalPixel;
          setState(() {
            _progress = (progress * 100).toInt();
          });
          print("正在滚动: ${notification.metrics.pixels} - ${notification.metrics.maxScrollExtent}")
;
        } else if (notification is ScrollEndNotification) {
          print("结束滚动....");
        }
        return false;
      },
      child: Stack(
        alignment: Alignment(.9, .9),
        children: <Widget>[
          ListView.builder(
            itemCount: 100,
            itemExtent: 60,
            itemBuilder: (BuildContext context, int index) {
              return ListTile(title: Text("item$index"));
            }
          ),
          CircleAvatar(
            radius: 30,
            child: Text("$_progress%"),
            backgroundColor: Colors.black54,
          )
        ],
      ),
    );
  }
}
```

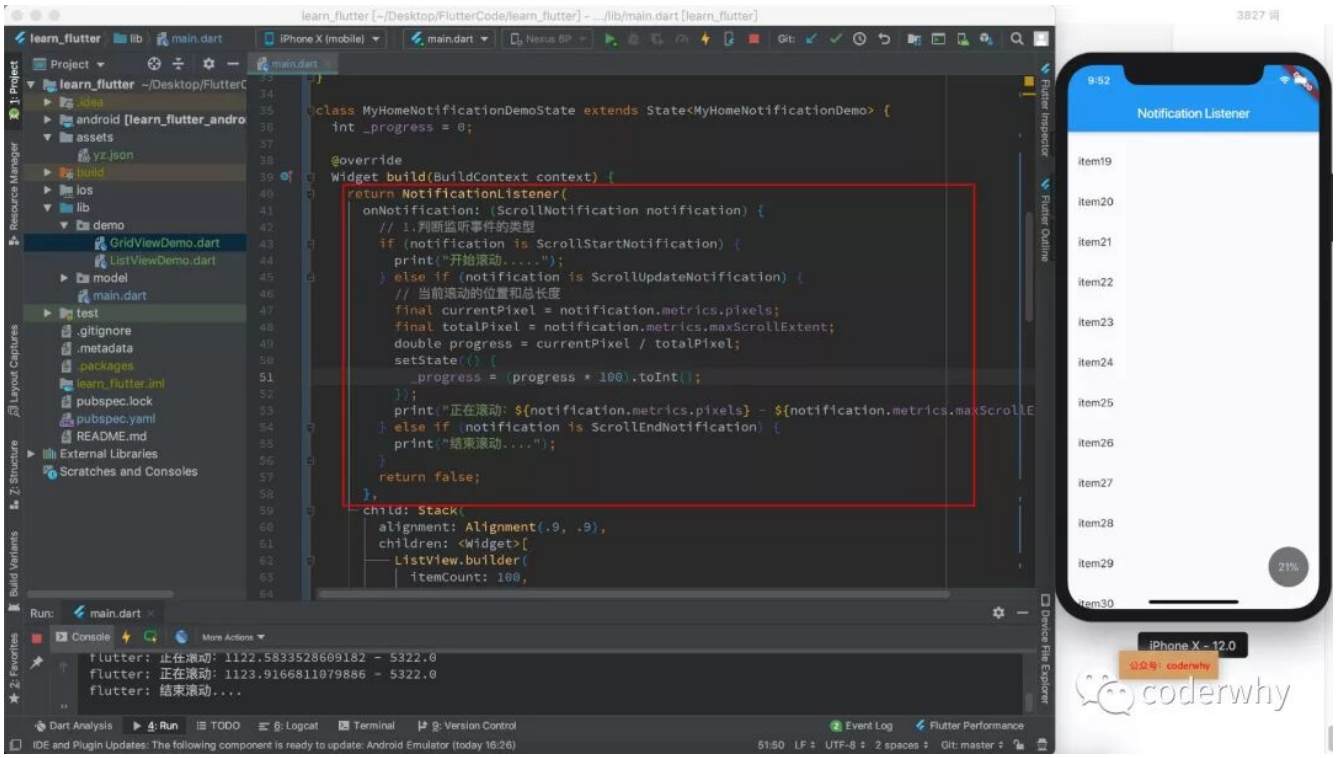

image-20190905215240084

> 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