

Flutter 实战篇：仿豆瓣电影APP

现在开始 Flutter 实战，我们的目的是仿写一个豆瓣电影APP。

在仿写的过程中，我们将学会：

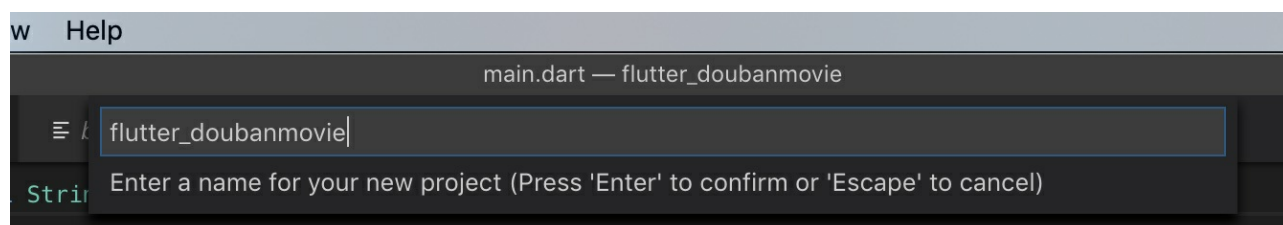
- 如何写UI和布局
- 如何添加第三方库
- 如何发起请求并解析数据
- 如何异步编程
- 页面之间如何跳转及如何传递参数

Widget 布局的实现

本节用 Flutter 实现豆瓣电影APP的节目。从这里你会学到如何使用 Flutter 的 Widget 去布局，以及遇到问题时如何解决。

创建 Flutter 工程

在 VS Code 中创建 Flutter 工程，工程名字为：
flutter_doubanmovie。



创建完成后，首先要对工程里的默认代码进行修改。

- 将 title 改为 豆瓣电影

在类 MyApp 里改为：

```
home: MyHomePage(title: '豆瓣电影'),
```

- `_MyHomePageState`

`_MyHomePageState` 里的 UI 代码要删掉，而且因为豆瓣电影没有 AppBar，所以 AppBar 也要删除，除删除之后的 `_MyHomePageState` 为：

```
class _MyHomePageState extends
State<MyHomePage> {

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        ),
      );
  }
}
```

做完这些操作之后，在运行 `flutter_doubanmovie`，得到的是一个空页面。

仿写豆瓣电影APP界面

写界面的正常流程应该是按照设计稿来实现，但是因为我们没有设计稿，所以我们按照豆瓣电影APP界面的截图来仿写。

首先分析豆瓣电影APP的界面：

可以看到 豆瓣电影APP 的界面主要有两部分组成：

1. 底部的 Navigation Bar
2. 上面显示的内容页面，包括：热映、找片、我的

所以，接下来我们先实现底部的 Navigation Bar，然后在实现要显示的内容页面。

实现底部 Navigation Bar

底部的 Navigation Bar 使用 Flutter 的 BottomNavigationBar 来实现，BottomNavigationBar 这个 Widget 前面没有介绍过，但是没有关系，我们通过查看 BottomNavigationBar 的构造函数就可以知道如何使用。

使用 Scaffold 的 bottomNavigationBar 属性

首先给 Scaffold 的 bottomNavigationBar 属性赋值：

```
return Scaffold(  
  ...  
  bottomNavigationBar: BottomNavigationBar(  
    ...  
  ),  
  ...  
)
```

然后在实现 BottomNavigationBar

为了使用 BottomNavigationBar，先看 BottomNavigationBar 的构造函数：

```
class BottomNavigationBar extends StatefulWidget
{
  BottomNavigationBar({
    Key key,
    @required this.items,
    this.onTap,
    this.currentIndex = 0,
    BottomNavigationBarType type,
    this.fixedColor,
    this.iconSize = 24.0,
  })
  ...
}
```

参数名字

参数类型

key	Key	Widget 的标识
items	List<BottomNavigationBarItem>	底部导航栏的选项 选项的类型是 BottomNavigatic
onTap	ValueChanged<int>	底部导航栏选项的 是选项的 index
currentIndex	int	当前选中的选项的 定义底部导航栏的 个值: BottomNavigatic 项少于4个时, 默

type	BottomNavigationBarType	fixedColor 不为空时，背景色就是 fixedColor，否则就是 ThemeData.primaryColor。当 BottomNavigationBarType.fixed 时，所有的选项都是导航栏的背景色。
fixedColor	Color	当 BottomNavigationBarType.fixed 时，选中的选项的背景色。
iconSize	double	BottomNavigationBarItem 的图标大小。

BottomNavigationBarItem

这里还有一个新的 Widget: BottomNavigationBarItem，我们在看一下 BottomNavigationBarItem 如何使用，它的构造函数为：

```
class BottomNavigationBarItem {
  const BottomNavigationBarItem({
    @required this.icon,
    this.title,
    Widget activeIcon,
    this.backgroundColor,
  })
  ...
}
```

参数名字	参数类型	意义	必选 or 可选
icon	Widget	图标	必选
title	String	标题	可选
activeIcon	Widget	激活时的图标	可选
backgroundColor	Color	背景色	可选

icon	Widget 选项的 Icon	必选
title	Widget 选项的 标题	可选
activeIcon	Widget 如果 activeIcon 没有设置的话，选中的时候会显示 icon	可选
backgroundColor	当 BottomNavigationBarType 的 type 是 shifting 时才有用，选中时会成为导航栏的背景色	可选

所以为 BottomNavigationBar 添加 BottomNavigationBarItem 就可以写成：

```

@override
Widget build(BuildContext context) {
  return Scaffold(
    body: Center(
    ),
    bottomNavigationBar: BottomNavigationBar(
      items: [
        BottomNavigationBarItem(icon:
Icon(Icons.school), title: Text('热映')),
        BottomNavigationBarItem(icon:
Icon(Icons.panorama_fish_eye), title: Text('找
片')),
        BottomNavigationBarItem(icon:
Icon(Icons.people), title: Text('我的'))
      ],//BottomNavigationBarItem 的 icon 用
的是 Icon Widget实现的，这里是随便找的图标，先实现功能，
后续在改成和 豆瓣电影 的一样
      currentIndex: 0,//默认选中的 index
      fixedColor: Colors.black, //选中时颜色
变为黑色
      type:
BottomNavigationBarType.fixed,//类型为 fixed
      onTap: _onItemTapped,
    ),
  );
}

void _onItemTapped(int index) {
}

```

写到这里，底部导航栏就已经实现了，但是显示的页面还没有实现。

实现内容页面

这里要实现内容页面，包括：热映、找片、我的，这里先用 Text 简单实现一下，并用 `_widgetItems` 的数组保存，接下来就要让 `_widgetItems` 和 当前选中的选项建立联系，这时候就需要用到 `_onItemTapped` 事件。

用 `_selectedIndex` 表示当前选中的是哪个选项，显示的页面就是 `_widgetItems[_selectedIndex]`，`_selectedIndex` 默认为 0，当选中的选项卡变化时，会触发 `_onItemTapped` 事件，然后刷新页面，实现的代码如下：

```
int _selectedIndex = 0;

final _widgetItems = [Text('热映'), Text('找片'),
Text('我的')];

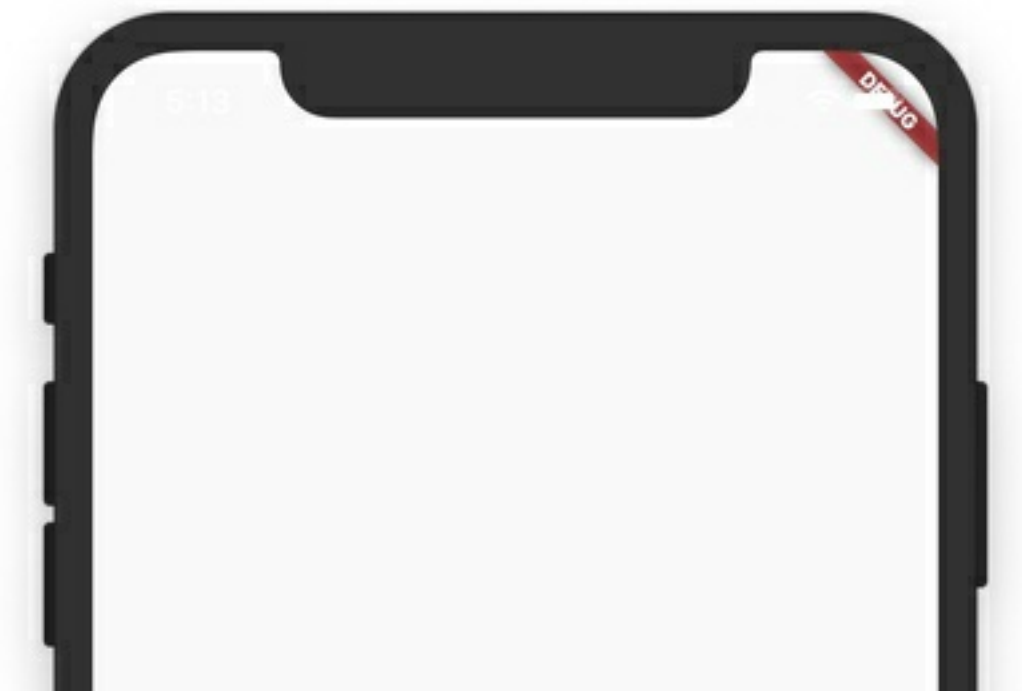
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: Center(
      child:
_widgetItems[_selectedIndex], //选中不同的选项显示不同的
的界面
    ),
    bottomNavigationBar: BottomNavigationBar(
      items: [
        BottomNavigationBarItem(icon:
Icon(Icons.school), title: Text('热映')),
        BottomNavigationBarItem(icon:
Icon(Icons.panorama_fish_eye), title: Text('找
片')),
        BottomNavigationBarItem(icon:
Icon(Icons.people), title: Text('我的'))
      ], //BottomNavigationBarItem 的 icon 用
```

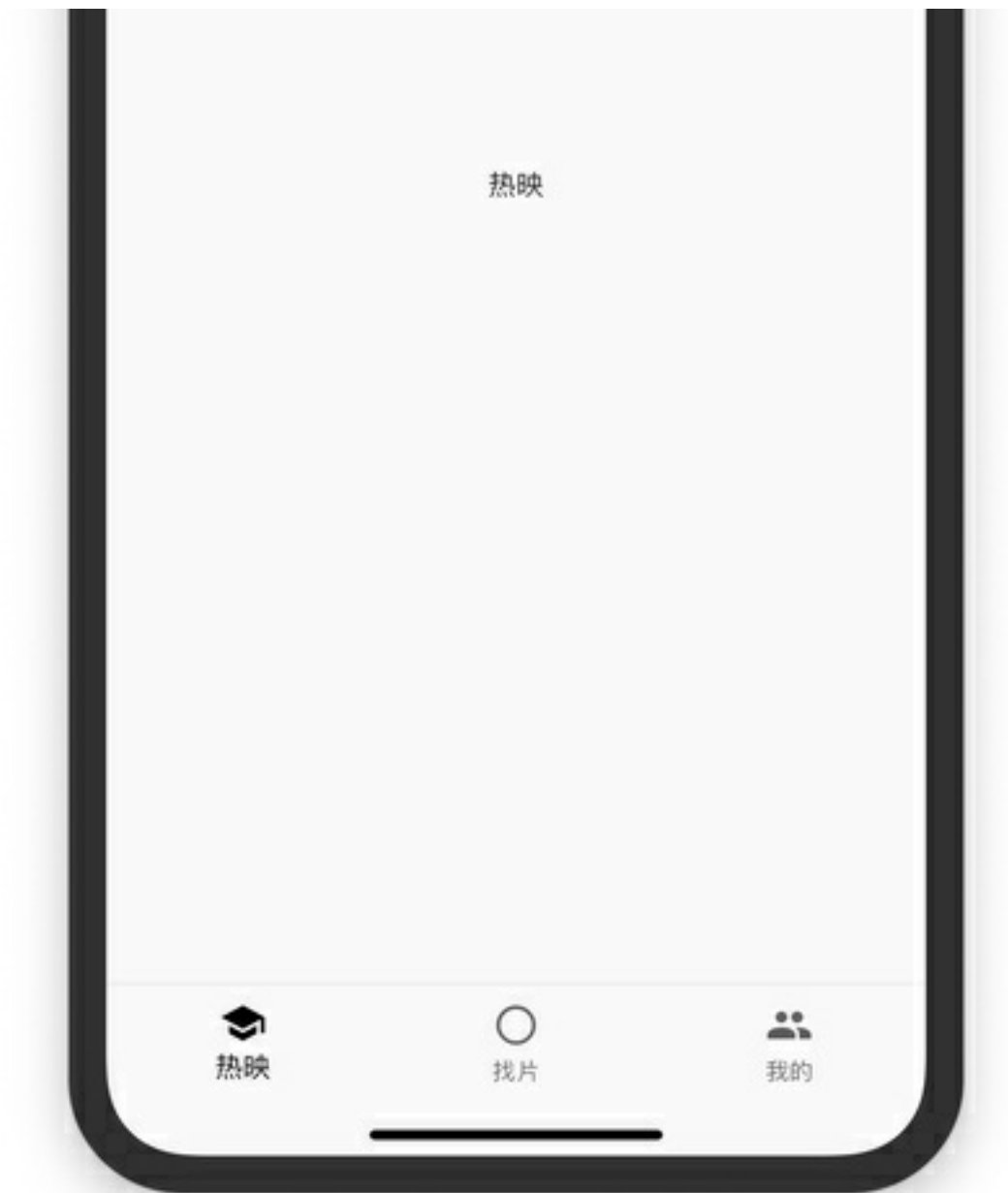

的是 Icon Widget实现的，这里是随便找的图标，先实现功能，后续在改成和 豆瓣电影 的一样

```
        currentIndex: _selectedIndex, //默认选中
        的 index
        fixedColor: Colors.black, //选中时颜色
        变为黑色
        type:
        BottomNavigationBarType.fixed, //类型为 fixed
        onTap: _onItemTapped,
    ),
);
}

void _onItemTapped(int index) {
    setState(() {
        _selectedIndex = index; //刷新界面
    });
}
```

运行后的效果为：





将内容页面提取出来在单独的类里实现

前面的内容页面都是用 Text 实现，而且存储在 main.dart 里，当内容页面的内容变得越来越复杂，势必会造成 main.dart 里的代码越来越多，不好管理，所以现在要把内容页面单独提取出一个文件来实现。

1. 热映页面

在 lib 目录下，右击，选择 New File,输入 HotWidget.dart，新建一个文件 HotWidget.dart，在这个文件里实现类 HotWidget

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

class HotWidget extends StatefulWidget {
  @override
  State<StatefulWidget> createState() {
    // TODO: implement createState
    return HotWidgetState();
  }
}

class HotWidgetState extends State<HotWidget> {
  @override
  Widget build(BuildContext context) {
    // TODO: implement build
    return Center(
      child: Text('热映'),
    );
  }
}
```

2. 找片页面

在 lib 目录下，新建一个文件 MoviesWidget.dart，在这个文件里实现类 MoviesWidget:

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

class MoviesWidget extends StatefulWidget {
  @override
  State<StatefulWidget> createState() {
    // TODO: implement createState
    return MoviesWidgetState();
  }
}

class MoviesWidgetState extends
State<MoviesWidget> {
  @override
  Widget build(BuildContext context) {
    // TODO: implement build
    return Center(
      child: Text('找片'),
    );
  }
}
```

3. 我的页面

在 lib 目录下，新建一个文件 MineWidget.dart，在这个文件里实现类 MineWidget:

```

import 'package:flutter/material.dart';

class MineWidget extends StatefulWidget {
  @override
  State<StatefulWidget> createState() {
    // TODO: implement createState
    return MineWidgetState();
  }
}

class MineWidgetState extends
State<MineWidget> {
  @override
  Widget build(BuildContext context) {
    // TODO: implement build
    return Center(
      child: Text('我的'),
    );
  }
}

```

4. 改造 main.dart

将页面独立分开之后，还需要改造 main.dart，首先将 _widgetItems 改造为：

```

final _widgetItems = [HotWidget(),
MoviesWidget(), MineWidget()];

```

同时 Scaffold 的 body 改为：

```

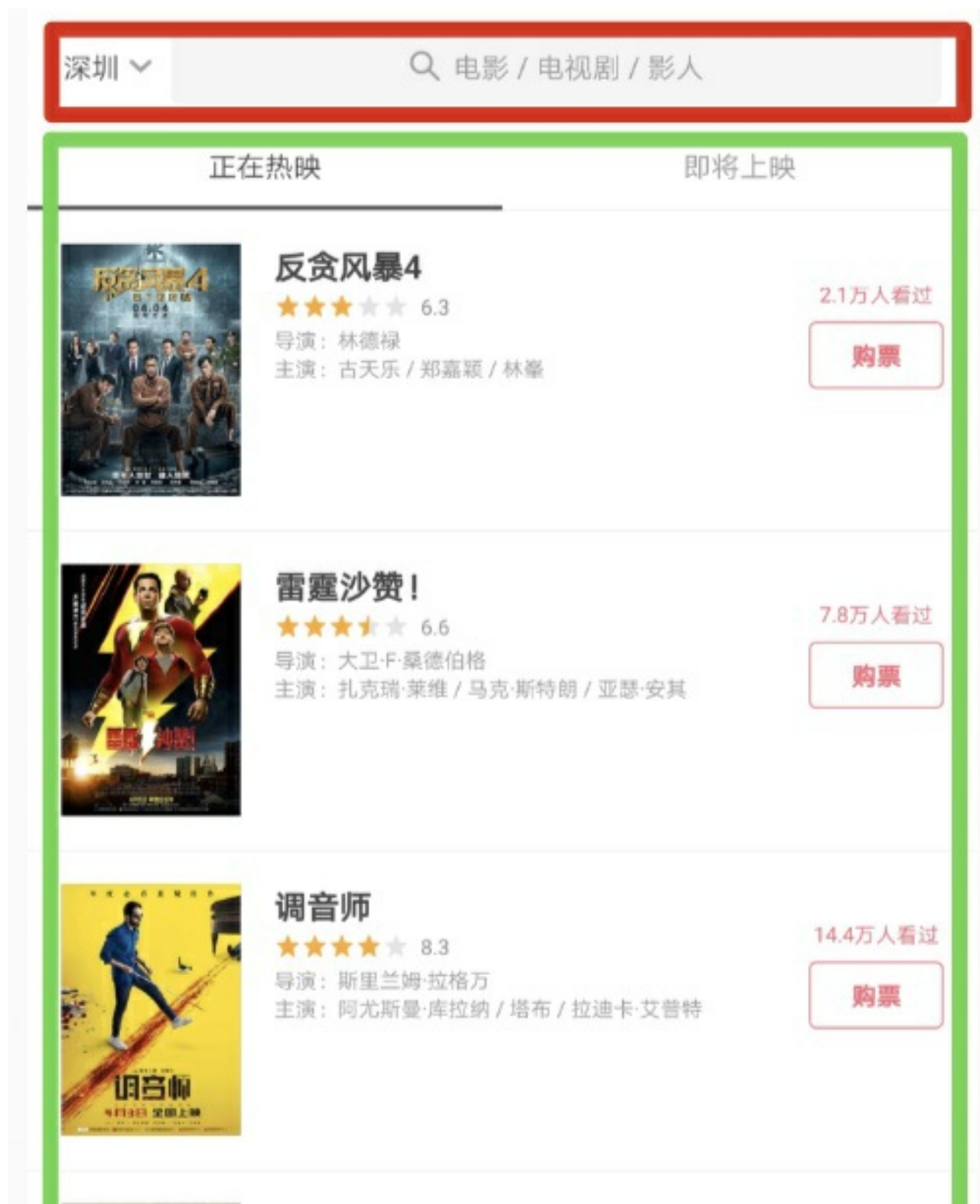
body: _widgetItems[_selectedIndex], //选中不同的
选项显示不同的界面，

```

实现热映页面

因为热映页面现在是用 Text 来代替，所以现在开始实现热映页面。

热映页面分析





可以看到热映页面在垂直方向上有两个部分：

1. 搜索栏
2. TabBar 与 TabBarView

所以，这里可以用 Column 来实现。

在看搜索栏：



在水平方向上有三个部分，所以可以用 Row 来实现

在 TabBar 里的数据流内容可以用 ListView 来实现。

搜索栏实现

搜索栏在水平方向上的3个部分，可以用 Text、Icon、TextField 这3个 Widget 来实现，你可能会这么写：

```
Row(  
  children: <Widget>[  
    Text(  
      '深圳',  
      style: TextStyle(fontSize: 16),  
    ),  
    Icon(Icons.arrow_drop_down),  
    TextField()  
  ],  
)
```

但是这样写会报错：

```
flutter: ═══ EXCEPTION CAUGHT BY RENDERING  
LIBRARY
```

```
flutter: The following assertion was thrown  
during performLayout():  
flutter: BoxConstraints forces an infinite width.  
flutter: These invalid constraints were provided  
to RenderRepaintBoundary's layout() function by  
the  
flutter: following function, which probably  
computed the invalid constraints in question:  
flutter:  
_RenderDecoration._layout.layoutLineBox  
(package:flutter/src/material/input_decorator.dart  
:819:11)  
flutter: The offending constraints were:  
flutter:   BoxConstraints(w=Infinity,  
0.0<=h<=80.0)
```


因为 Row 要求它的 非flexible的子Widget 必须要有明确的宽度，为了给 flexible的子Widget 分配大小，但是 TextField 没有明确的大小，TextField 是 match_parent 的，所以这里需要用 Expanded 来嵌套 TextField，使用方式如下：

```
Row(  
  children: <Widget>[  
    Text(  
      '深圳',  
      style: TextStyle(fontSize: 16),  
    ),  
    Icon(Icons.arrow_drop_down),  
    Expanded(  
      flex: 1,  
      child: TextField(),  
    ),  
  ],  
)
```

运行后的效果为：



可以看到 TextField 的样式不符合要求，所以还要在设置 TextField 的样式，使用 InputDecoration 来设置 TextField 的样式：

```

TextField(
  textAlign: TextAlign.center,
  style: TextStyle(fontSize: 16),
  decoration: InputDecoration(
    hintText: '\uE8b6 电影 / 电视剧 / 影人',
    hintStyle: TextStyle(fontFamily:
'MaterialIcons', fontSize: 16),
    contentPadding: EdgeInsets.only(top: 8,
bottom: 8),
    border: OutlineInputBorder(
      borderSide: BorderSide.none,
      borderRadius:
BorderRadius.all(Radius.circular(5)),
    ),
    filled: true,
    fillColor: Colors.black12
  ),
)

```

最后搜索栏就是这样子：



因为 hintText 是 String 类型，为了显示搜索的 icon，使用了 Unicode 字符，并且设置 MaterialIcons 的字体库，就可以用 Unicode 来表示 icon。要查看特定 icon 的 Unicode 字符，可以在 Icons 里找到，然后看它的值就能找到它的 Unicode 字符，例如这个搜索 icon，它是 Icons.search，值为：

```

static const IconData search = IconData(0xe8b6,
fontFamily: 'MaterialIcons');

```

所以搜索的 Unicode 字符为: e8b6, 在字符串里用 \uE8B6 表示。

TabBar 与 TabBarView 的实现

TabBar 与 TabBarView 用 DefaultTabController 来实现, DefaultTabController 的 children 里面包含了 TabBar 和 TabBarView, 同时 DefaultTabController 为了占用剩下的空间, 也需要用 Expanded 来嵌套, 最后完整的代码如下:

```
class HotWidgetState extends State<HotWidget> {
  @override
  Widget build(BuildContext context) {
    // TODO: implement build
    return Column(
      mainAxisAlignment: MainAxisAlignment.start,
      children: <Widget>[
        Container(
          height: 80,
          alignment: Alignment.bottomCenter,
          padding: EdgeInsets.only(left: 20,
right: 20),
          child: Row(
            children: <Widget>[
              Text(
                '深圳',
                style: TextStyle(fontSize: 16),
              ),
              Icon(Icons.search),
              Expanded(
                flex: 1,
                child: TextField(
                  textAlign: TextAlign.center,
                  style: TextStyle(fontSize: 16),
```

```

        decoration: InputDecoration(
            hintText: '\uE8b6 电影 / 电视
剧 / 影人',
            hintStyle:
                TextStyle(fontFamily:
'MaterialIcons', fontSize: 16),
            contentPadding:
EdgeInsets.only(top: 8, bottom: 8),
            border: OutlineInputBorder(
                borderSide:
BorderSide.none,
                borderRadius:
BorderRadius.all(Radius.circular(5)),
            ),
            filled: true,
            fillColor: Colors.black12),
    ),
    ],
),
),
Expanded(
    flex: 1,
    child: DefaultTabController(
        length: 2,
        child: Column(
            children: <Widget>[
                Container(
                    constraints:
BoxConstraints.expand(height: 50),
                    child: TabBar(
                        unselectedLabelColor:
Colors.black12,

```

```

        labelColor:
Colors.black87,
        indicatorColor:
Colors.black87,
        tabs: <Widget>[Tab(text:
'正在热映'), Tab(text: '即将上映')]),
    ),
    Expanded(
      child: Container(
        child: TabBarView(
          children: <Widget>[
            Center(
              child: Text('正在热
映'),
            ),
            Center(
              child: Text('即将上
映'),
            ),
          ],
        ),
      ),
    ),
  ],
),
);
}
}

```

运行的效果为：

3:55



深圳 ▾

🔍 电影 / 电视剧 / 影人

正在热映

即将上映

正在热映



热映



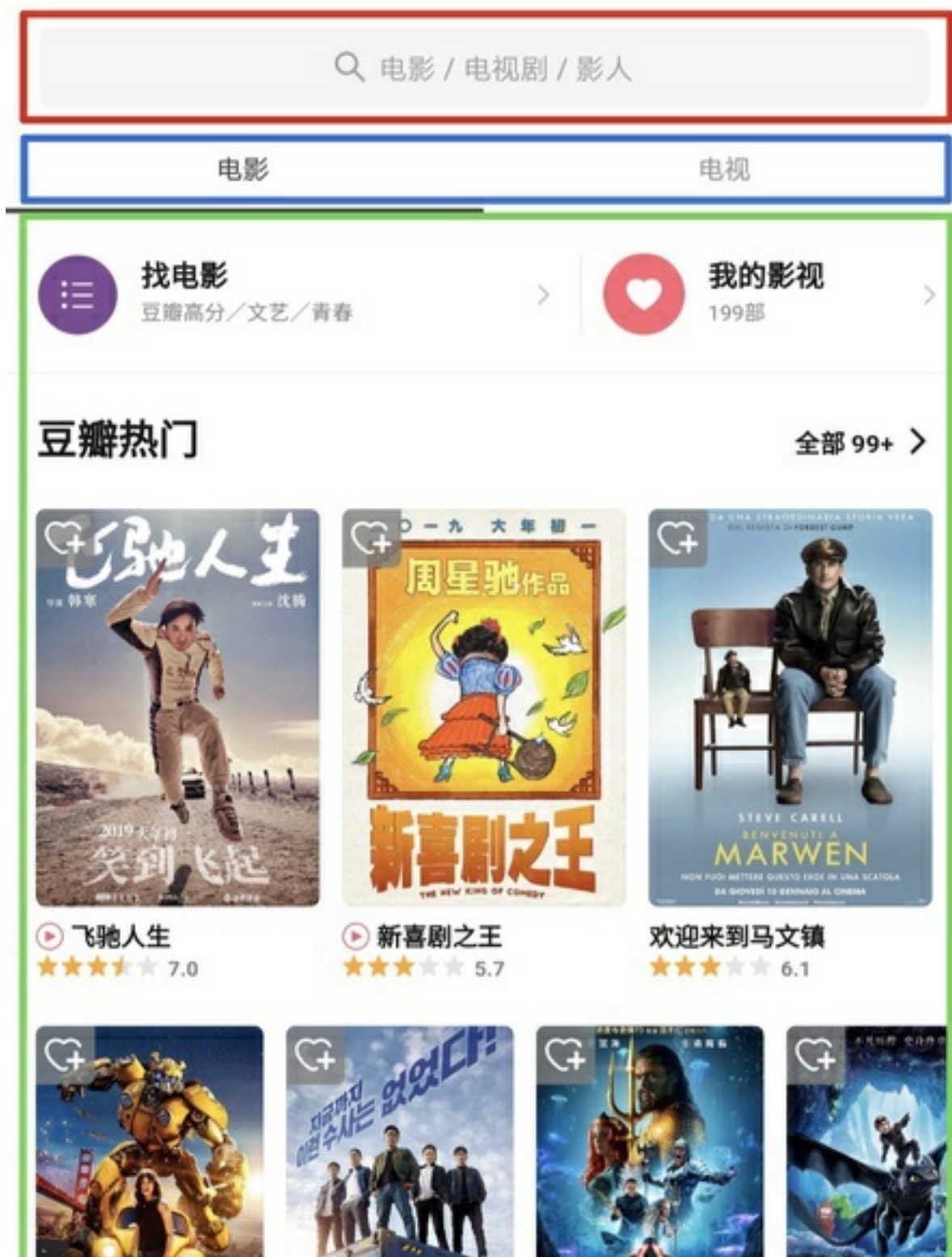
找片



我的

实现找片页面

找片页面分析





找片页面在垂直方向上也是有两个部分：

1. 搜索栏
2. TabBar 与 TabBarView

找片页面按照热映页面来实现就好了。

页面实现

在 MoviesWidget.dart 里修改：

```
class MoviesWidgetState extends
State<MoviesWidget> {
  @override
  Widget build(BuildContext context) {
    // TODO: implement build
    return Column(
      children: <Widget>[
        Container(
          height: 80,
          alignment: Alignment.bottomCenter,
          padding: EdgeInsets.only(left: 20,
right: 20),
          child: TextField(
```



```

        textAlign: TextAlign.center,
        style: TextStyle(fontSize: 16),
        decoration: InputDecoration(
            hintText: '\uE8b6 电影 / 电视剧 /
影人',
            hintStyle: TextStyle(fontFamily:
'MaterialIcons', fontSize: 16),
            contentPadding:
EdgeInsets.only(top: 8, bottom: 8),
            border: OutlineInputBorder(
                borderSide: BorderSide.none,
                borderRadius:
BorderRadius.all(Radius.circular(5)),
            ),
            filled: true,
            fillColor: Colors.black12),
    ),
    Expanded(
        flex: 1,
        child: DefaultTabController(
            length: 2,
            child: Column(
                children: <Widget>[
                    Container(
                        constraints:
BoxConstraints.expand(height: 50),
                        child: TabBar(
                            unselectedLabelColor:
Colors.black12,
                            labelColor:
Colors.black87,
                            indicatorColor:

```

```
Colors.black87,
        tabs: <Widget>[Tab(text:
'电影'), Tab(text: '电视')]),
    ),
    Expanded(
      child: Container(
        child: TabBarView(
          children: <Widget>[
            Center(
              child: Text('电影'),
            ),
            Center(
              child: Text('电视'),
            )
          ],
        ),
      ),
    ),
  ],
),
);
}
```

运行后的效果为：