

# 使用 ScopedModel 对豆瓣电影 App 进行重构

本节使用 ScopedModel 对豆瓣电影 App 进行重构。

### 重构后的工程路径

StateManager/flutter\_doubanmovie\_scopedmodel

### ScopedModel

ScopedModel 是一个可以对状态进行管理的第三方库,该库是从 Fuchsia 的代码库中提取的。

## ScopedModel 的使用

使用 ScopedModel 进行状态管理,要用到这个库提供的三个类,掌握这三个类的使用,就掌握了 ScopedModel。这三个类分别是:

- Model
- ScopedModel
- ScopedModelDescendant

#### Model

Model 是类,用来存储全局状态,当状态发生变化时,调用 notifyListeners() 方法,就会通知给依赖这个状态的子 Widget,引起子 Widget 的创建。我们需要继承 Model 类来写自己的 Models,例如 我们写一个 UserModel,里面可以用来存储用户的用户名和 token。

#### ScopedModel



#### ScopedModelDescendant

ScopedModelDescendant 也是 Widget,用于需要依赖 Model 里状态的子 Widget 中,而且会自动订阅 Model 里状态的变化,当 Model 里状态发生变化时,就会触发 ScopedModelDescendant 的重建。

#### 子Widget 中获取 Model 的方法

有两种方法可以在子 Widget 中获取 Model:

- 1. 使用 ScopedModelDescendant,可以获取 Model,并且在 Model 变化时会重建。
- 2. 使用 ScopedModel.of 的静态方法获取 Model 实例,但是无法收到 Model 变化的通知。如果需要频繁使用,可以在 Model 里写一个静态方法,下面会讲到。

## 使用 ScopedModel 重构

在 pubspec.yaml 里添加 ScopedModel 库的依赖:

```
dependencies:
    ...
    scoped_model: ^1.0.1
```

在 VS Code 里选择保存后,会自动下载依赖库。

第一步,是创建 CityModel 类,里面存储 curCity 的值,同时写一个可以设置 curCity 值的方法,并且在这个方法里,当 curCity 的值设置完后,还需要调用 notifyListeners():

```
class CityModel extends Model {
   String curCity;

void setCurCity(String city) {
   if (curCity != city) {
     curCity = city;

     //通知状态发生变化
     notifyListeners();
```



因为第一次打开的时候 curCity 需要从本地读取数据,这个读取数据的操作比较耗时,所以必须得异步操作,所以 CityModel 里的 curCity 的值默认只能是空的,读取到数据后在赋值,使得状态发生变化,从而触发 Widget 的重建,代码需要这么写:

```
class CityModel extends Model {
   void initData() async {
     final prefs = await SharedPreferences.getInstance(); //获取 prefs
     String city = prefs.getString('curCity'); //获取 key 为 curCity 的值
     if (city != null && city.isNotEmpty) {
       //如果有值
       setCurCity(city);
     } else {
       //如果没有值,则使用默认值
       setCurCity('深圳');
     }
   }
 }
在 CityModel 在建一个静态方法、获取 CityModel 的实例:
 class CityModel extends Model {
   . . .
   static CityModel of(BuildContext context) => ScopedModel.of<CityModel>(context)
 }
第二步,使用 ScopedModel<CityModel>() 作为父 Widget, 代码如下:
 class _MyHomePageState extends State<MyHomePage> {
   @override
   Widget build(BuildContext context) {
     return Scaffold(
       body: ScopedModel<CityModel>(
```

model: cityModel,

child: \_widgetItems[\_selectedIndex],



```
}
}
```

第三步,开始对依赖 curCity 的 Widget 进行重构,在 HotWidget 里需要使用 ScopedModelDescendant 来包原来的 HotWidget:

```
class HotWidgetState extends State<HotWidget> {
 @override
  void initState() {
   // TODO: implement initState
    super.initState();
    print('HotWidgetState initState');
  }
  @override
 Widget build(BuildContext context) {
    // TODO: implement build
    print('HotWidgetState build');
    return ScopedModelDescendant<CityModel>(
   );
}
```

这里 ScopedModelDescendant 有三个参数:

child

child 的类型为 Widget,是可选的,这个 child 是不需要依赖 Model 的 Widget,将会 传递给 builder。

builder

builder 的类型为 ScopedModelDescendantBuilder,是一个函数,是必选的,定义 为:



```
typedef Widget ScopedModelDescendantBuilder<T extends Model>(
 BuildContext context,
 Widget child,
```





函数的参数有 context、child、model,在函数的内部可以访问这三个参数, builder 里 的 child 就是上面提到的 child, 在 builder 通过 modle 参数访问状态数据, 例如 modle.curCity, 最后返回一个 Widget。

rebuildOnChange

rebuildOnChange 是 bool 类型,是可选的,表示当状态发生变化的时候是否需要重 建, 默认是 true, 除非特别情况, 这里应该永远是 true。

所以继续重构代码,将 HotWidgetState 里的 curCity 变量删掉,并替换成 model.curCity :

```
return ScopedModelDescendant<CityModel>(
     builder: (context, child, model) {
       if (model.curCity != null && model.curCity.isNotEmpty) {
         //如果 curCity 不为空
         return ...
       } else {
         //如果 curCity 为空
         return ...
       }
     },
    );
```

这个时候 \_jumpToCitysWidget() 方法会报错,给这个方法增加一个 city 的参数,从 调用的地方传过来,因为这个方法里会改变 CityModel 里 curCity 的值,需要调用 CityModel 的 setCurCity() 方法,所以还要增加一个 CityModel 的参数,最后代码就 是:

```
void _jumpToCitysWidget(CityModel model,String city) async {
 var selectCity =
     await Navigator.pushNamed(context, '/Citys', arguments: city);
 if (selectCity == null) return;
 final prefs = await SharedPreferences.getInstance();
  prefs.setString('curCity', selectCity); //存取数据
 model.setCurCity(selectCity);
}
```



```
jumpToCitysWidget(model,model.curCity);
```

重构完 HotWidget 后,同理对 HotMoviesListWidget、CitysWidget 进行重构。

因为 HotMoviesListWidget 里的 curCity 的值需要在 build 之前就要用到,而且 HotMoviesListWidget 也没必要监听 CityModel 的变化,因为 HotMoviesListWidget 的父 Widget 是 HotWidget, HotWidget 已经对 CityModel 进行了变化, HotWidget 重建的时 候也会引起 HotMoviesListWidget 的重建,所以 HotMoviesListWidget 里就不用 ScopedModelDescendant<CityModel>(...) 了,而是用 CityModel.of(context),而 且因为要用到 context,和前面使用 InheritedWidget 一样的理由, getData() 放到 didChangeDependencies() 里, 重构后的代码为:

```
class HotMoviesListWidget extends StatefulWidget {
  HotMoviesListWidget() {}
  @override
  State<StatefulWidget> createState() {
    // TODO: implement createState
    return HotMoviesListWidgetState();
  }
}
class HotMoviesListWidgetState extends State<HotMoviesListWidget>
    with AutomaticKeepAliveClientMixin {
  List<HotMovieData> hotMovies = new List<HotMovieData>();
 @override
  void initState() {
   // TODO: implement initState
    super.initState();
  }
  @override
  void didChangeDependencies() {
    // TODO: implement didChangeDependencies
    super.didChangeDependencies();
    _getData();
  }
  void getData() async {
    List<HotMovieData> serverDataList = new List();
    var response = await http.get(
        https://api.douban.com/v2/movie/in_theaters?apikey=0b2bdeda43b5688921839
```



```
}
}
```

HotMoviesListWidget 的构造函数去掉了参数,所以之前调用的地方要把参数去掉。

CitysWidget 这里就可以重构,也可以不重构,因为原先这里的 curCity 的值是通过参数 传过来的,耦合性就比较低,但是传参还是挺麻烦的,这里我们还是重构一下吧。

CitysWidget 的重构有一点麻烦,因为 CitysWidget 是通过路由创建的,因此 CitysWidget 的父 Widget 和 HotWidget 的父 Widget 不同,所以为了让 CitysWidget 访问到 Model, 一方面要用 ScopedModel<CityModel>(),同时 HotWidget 和 CitysWidget 要公用同一 个 CityModel 的实例,所以重构后的代码如下:

```
CityModel cityModel = CityModel();
class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      . . .
      routes: {
        '/Citys': (context) => ScopedModel<CityModel>(
              model: cityModel,
              child: ScopedModelDescendant<CityModel>(
                builder: (context, child, model) {
                  return CitysWidget();
                },
              ),
            ),
      },
    );
  }
}
class MyHomePage extends StatefulWidget {
}
class MyHomePageState extends State<MyHomePage> {
```



```
return Scaffold(
       body: ScopedModel<CityModel>(
         model: cityModel,
         child: _widgetItems[_selectedIndex],
       ), //选中不同的选项显示不同的界面,
     );
   }
curCity 就使用 CityModel.of(context) 来获取, CitysWidget 重构后的代码为:
 class CityWidgtState extends State<CitysWidget> {
   String curCity;
   @override
   Widget build(BuildContext context) {
     // TODO: implement build
     curCity = CityModel.of(context).curCity;
   }
  }
同时把传递参数的代码删掉,在 HotWidget 的 _jumpToCitysWidget() 方法,重构后的
代码为:
 void _jumpToCitysWidget(CityModel model,String city) async {
     var selectCity =
         await Navigator.pushNamed(context, '/Citys');
   }
```

接下来在看一下,能不能把 UI 逻辑和业务逻辑分离,同样看一下 HotMoviesListWidget 里, 既包含了一个请求数据, 又包含了数据的显示, 可以考虑把数据的请求和显示分开, 数 据请求部分如下:

```
void _getData() async {
  List<HotMovieData> serverDataList = new List();
   var response = await http.get(
```





```
//成功获取数据
 if (response.statusCode == 200) {
    var responseJson = json.decode(response.body);
    for (dynamic data in responseJson['subjects']) {
      HotMovieData hotMovieData = HotMovieData.fromJson(data);
      serverDataList.add(hotMovieData);
    }
    setState(() {
      hotMovies = serverDataList;
   }):
 }
}
```

但是这部分应该放在哪呢? 其遇到的问题和 InheritedWidget 的状态管理框是一样的,这里 既不能放在 CityModel 里,因为这部分其实是本地状态, 创建一个新的 Model,会把原来 的本地状态变为全局状态,导致代码复杂话,不适合扩展,从中也可以看到 ScopedModel 的状态管理框架并没有考虑 UI 逻辑和业务逻辑分离的问题。

#### 分析

在使用 ScopedModel 重构完后,对比一下一个好的状态管理框架应该具有的条件:

- 能管理好全局状态和本地状态
- UI 逻辑和业务逻辑应该是分离的
- 在框架的帮助下可以写出高质量的代码
- 框架应该提升 App 的性能
- 框架要容易理解,便于扩展

我们发现 ScopedModel 只能管理全局状态,而且没有涉及到 UI 逻辑和业务逻辑的分离。

# 总结

通过 ScopedModel 对豆瓣电影 App 的重构,我们可以发现 ScopedModel 也可以对全局 状态进行管理,相对于 InheritedWidget 来说,有如下优点:

- 可以对全局状态进行管理
- 数据只能从上到下传递,也可以从下到上传递



- 无法管理本地状态
- 随着 App 变大, 代码维护也会变得越来越难。

所以,ScopedModel 只能在简单的 App 里使用,大型 App 绝对不能使用 ScopedModel 对状态进行管理。

#### 留言

评论将在后台进行审核, 审核通过后对所有人可见

#### 代来

这一章是不是有问题啊、大佬、没有写完。刚好看到这里麻烦更新一下

**a** 0 收起评论 26天前

> 小德\_REN [w] 德艺双馨 @ Single 好的,24小时内就会更新 26天前

评论审核通过后显示

评论

