Flutter状态管理学习手册(一)——ScopedModel

WinDin

一、ScopedModel简介

ScopedModel属于入门级别的状态管理框架，它的思想比较简单，参考官方文档便可以很容易理解其中构架。

在Flutter中Lifting state up(状态提升)是十分必要的，状态提升可以理解为把组件之间相互共享的状态提取出来放在一个较高层级中管理的一种思想。ScopedModel提供了对于这种状态管理的便利。

二、ScopedModel中的三个概念

ScopedModel主要有三个重要的概念，也是其中的三个类：Model、ScopedModel和ScopedModelDescendant。ScopedModel基本上通过这三个类实现其功能。 Descendant 子孙 Scoped 审视，仔细研究

Model是封装状态和状态操作的地方。我们可以将想要的数据存放在Model当中并且将对数据操作，如添加删除的相关方法放在这里。Model还提供了一个notifyListeners()方法，它的作用是当数据发生改变时，可以通过调用notifyListeners()方法通知界面进行更新。

ScopedModel是一个用于保存Model的Widget。通常ScopedModel会一个应用的入口处作为父布局使用，并以Model作为参数传

入，使得`ScopedModel`持有`Model`。

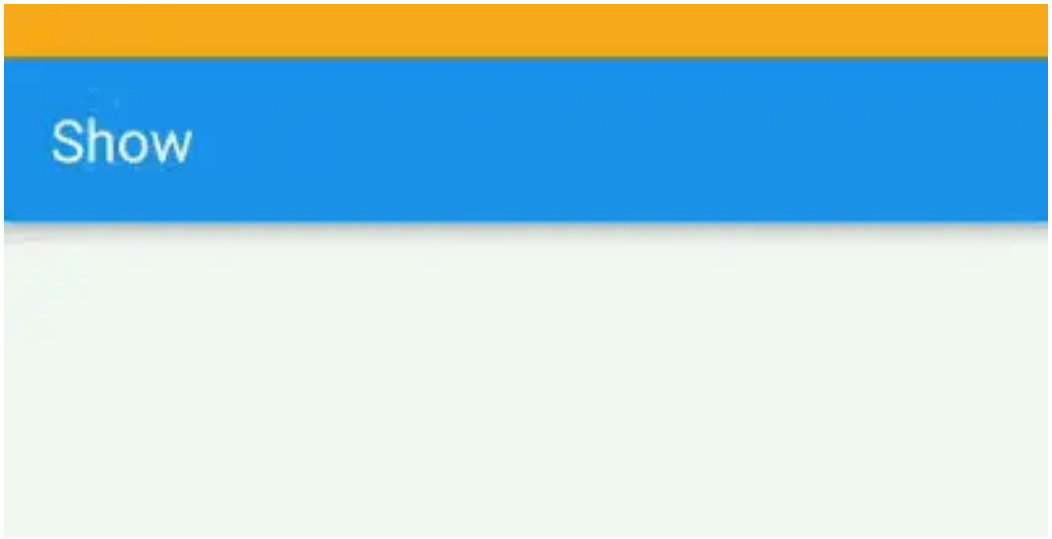
在`ScopedModel`的子布局中，可以通过`ScopedModel.of<Model>(context)`方法来获取`Model`。

`ScopedModelDescendant`，顾名思义，是`ScopedModel`的派生物。同样的，它也是一个`Widget`。`ScopedModelDescendant`会作为`ScopedModel`下的子布局存在，它的主要作用是响应状态更新。

`ScopedModelDescendant`中存在`builder`函数，这个函数会在`Model`的`notifyListeners()`发生时被调用，从而根据`Model`中的数据生成相应的界面。

三、ScopedModel的实践

这里以常见的获取列表选择列表为例子。一个页面用于展示选中项和跳转到列表，一个页面用于显示列表。



1. 引入 `scoped_model` 第三方库

在根目录的 `pubspec.yaml` 文件的 `dependencies` 中加入依赖

```
dependencies:
```

```
...
scoped_model: ^1.0.0
```

2. 定义`Model`

创建一个`ListModel`类，这个类需要继承`scoped_model`包里的`Model`类。

`ListModel`类中包含三个状态：列表初始化标志、列表数据、选中的列表项。

```
bool _init = false; // 列表初始化标志
List<String> _list = []; // 列表数据
String _selected = '未选中'; // 选中的列表项
```

在`Model`中不仅只有数据，还包括对数据操作的方法，这里定义两个操作方法，分别是选中列表项目和加载列表的方法，并且，这两个方法在更新数据后，需要调用`notifyListeners()`通知UI更新。

```

/**
 * 选中列表项
 */
void select(String selected) {
```

```
    _selected = selected;

    // 通知数据变更
    notifyListeners();
  }

  /**
   * 加载列表
   */
  void loadList() async {
    // 模拟网络请求
    await Future.delayed(Duration(milliseconds: 3000));
    _list = [
      '1\. Scoped Model',
      '2\. Scoped Model',
      '3\. Scoped Model',
      '4\. Scoped Model',
      '5\. Scoped Model',
      '6\. Scoped Model',
      '7\. Scoped Model',
      '8\. Scoped Model',
      '9\. Scoped Model',
      '10\. Scoped Model'
    ];

    _init = true;
    // 通知数据变更
    notifyListeners();
  }
```

3. UI布局

在UI上，使用`ScopedModel`作为根布局，提供`Model`，使用

ScopedModelDescendant作为子布局，响应Model。

首先，在main()方法中，创建ListModel实例，用ScopedModel包裹MyApp布局

```
void main() {

  // 创建Model实例
  ListModel listModel = ListModel();
  // 使用ScopedModel作为根布局
  runApp(ScopedModel(model: listModel, child: MyApp()));
}
```

为了体现状态提升这一概念，例子中使用两个页面，一个是ShowPage，另一个是ListPage。ShowPage用于显示选中的列表项目和提供跳转到ListPage的入口，ListPage用于加载显示列表。

在ShowPage中，显示ListModel中的选中项。

```
ScopedModelDescendant<ListModel>(
  builder: (context, child, model) {
    String selected = model.selected;
    return Text(selected);
  }
),
```

`ScopedModelDescendant`的泛型指明`ListModel`，它便会自动获取`ScopedModel`中的`ListModel`，在`builder: (context, child, model)`中即可通过其中的`model`参数获取状态，构建UI。

同样的，在`ListPage`中，通过`ScopedModelDescendant`来显示加载状态和列表。

```
body: ScopedModelDescendant<ListModel>(builder: (context,
child, model) {
        // 根据状态显示界面
        if (!model.isInit) {
          // 显示loading界面
          return buildLoad();
        } else {
          // 显示列表界面
          var list = model.list;
          return buildList(list);
        }
      }),
```

4. 状态改变

`ListPage`是一个`StatefulWidget`，所以可以在`initState()`方法中进行列表的加载工作。

```
@override
void initState() {
  super.initState();
  ListModel model = ScopedModel.of<ListModel>(context); //
获取ListModel
```

```
  if (!model.isInit) {
    model.loadList(); // 加载列表
  }
}
```

点击列表中的某一项时，会选中该项，这时调用`ListModel`中的`select(String)`方法，并返回上一个界面。

```
onTap: () {
  ListModel model = ScopedModel.of<ListModel>(context);
  model.select(list[index]);
  // 返回到上一级页面
  Navigator.pop(context);
},
```

完整代码可以参考 github.com/windinwork/...

四、ScopedModel的注意事项

1. `ScopedModelDescendant`的层级需要尽量低，可以避免大范围的UI重建。这里引用官方的例子。

```
// 错误示例
return ScopedModelDescendant<CartModel>(
  builder: (context, child, cart) {
    return HumongousWidget(
      // ...
      child: AnotherMonstrousWidget(
        // ...
        child: Text('Total price: ${cart.totalPrice}'),
```

```
      ),
    );
  },
);
```

```
// 正确示例
return HumongousWidget(
  // ...
  child: AnotherMonstrousWidget(
    // ...
    child: ScopedModelDescendant<CartModel>(
      builder: (context, child, cart) {
        return Text('Total price: ${cart.totalPrice}');
      },
    ),
  ),
);
```

五、总结

`ScopedModel`功能比较简单，使用`Model`保存状态和通知状态改变，使用`ScopedModel`提供`Model`，使用`ScopedModelDescendant`布局来响应状态变化，是一个十分适合入门者理解的状态管理模型。

参考目录

Simple app state management