flutter 学习二

1. 课前复习

- 常用组件的使用
  - Image
  - Text
  - 按钮相关组件（MaterialButton、RaiseButton、FlatButton、DropdownButton、悬浮按钮、IconButton）
  - 单组件容器布局组件（Container、Padding、Center、Align）
  - 多组件容器布局组件（Row、Column、Stack、Wrap）
- 手势
- ListView的使用

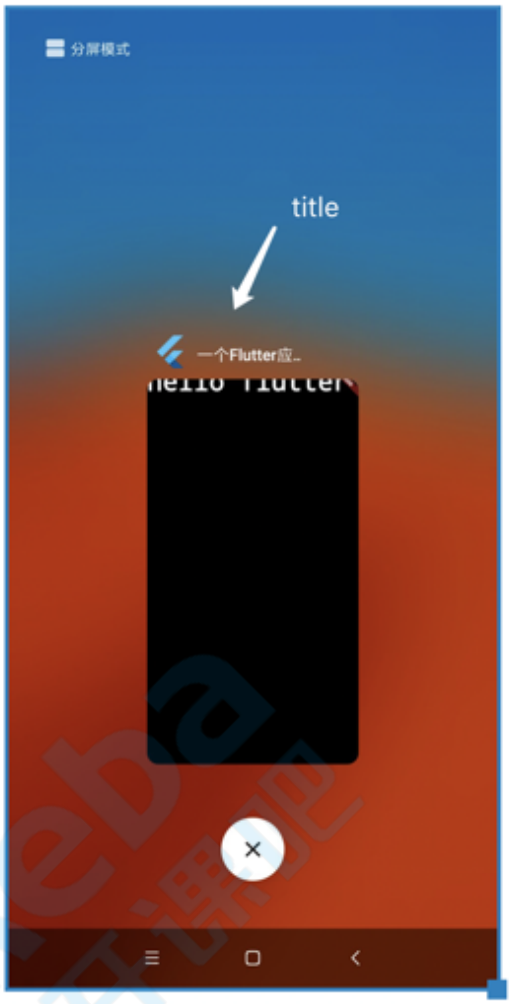
2. 课堂目标

- App结构和导航组件
  - MaterialApp应用组件
  - Scaffold脚手架组件
  - Flutter主题
- 可滚动组件
  - ListView
  - GridView
  - SingleChildScrollView
  - Table

3. 知识点

App结构和导航组件

1. **MaterialApp**：封装了应用程序实现Material Design所需要的一些Widget，实际是一种设计风格，里面会有已有的一些组件（eg：theme）
   - title: 该属性会在`Android`应用管理器的App上方显示，对于`ios`设备是没有效果的



- home: `Widget`类型，这是在应用程序正常启动时首先显示的Widget，除非指定了`initialRoute`。如果`initialRoute`显示失败，也该显示该Widget。
- theme: `ThemeData`类型，定义应用所使用的主题颜色，可以指定一个主题中每个控件的颜色
- routes: `Map<String, WidgetBuilder>`类型，是应用的顶级路由表，当使用`Navigator.pushNamed`进行命名路由的跳转时，会在此路表中进行查找并跳转
- initialRoute: `String`类型，初始化路由
- onGenerateRoute: `RouteFactory`类型，路由回调函数。当通过Navigator.of(context).pushNamed跳转的时候，如果routes查找不到会调用这个方法

2. **Scaffold**： 实现了基本的 Material Design 布局结构

- appBar：显示在界面顶部的一个 AppBar
- body：当前界面所显示的主要内容 Widget
  - drawer：抽屉菜单控件
  - bottomNavigationBar：显示在页面底部的导航栏，`items`必须大于2个

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
```

```
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: Scaffold(
        appBar: AppBar(
          title: Text('首页'),
          // centerTitle: false,
        ),
        drawer: Drawer(
          child: Column(
            children: <Widget>[
              DrawerItem(1, '选项1'),
              DrawerItem(2, '选项2'),
              DrawerItem(3, '选项3'),
              DrawerItem(4, '选项4'),
              DrawerItem(5, '选项5')
            ],
          ),
        ),
        bottomNavigationBar: BottomNavigationBar(
          type: BottomNavigationBarType.fixed,
          currentIndex: 1,
          items: [
            new BottomNavigationBarItem(
                  icon: Icon(Icons.account_balance), title: Text('标题一')),
            new BottomNavigationBarItem(
                icon: Icon(Icons.contacts), title: Text('标题二')),
            new BottomNavigationBarItem(
                icon: Icon(Icons.library_music), title: Text('标题三'))
          ],
        ),
        body: Center(
          child: Text('THIS IS BODY'),
        ),
      ),
    );
  }
}
```

3. **Flutter主题**： 使用主题可以在App里面共享颜色和字体样式。在Flutter里面有两种方式来使用主题，一种是全局范围的、一种是使用`Theme` Widget， `Theme` Widget可以在App的某个部分使用主题。全局的主题其实也就是`MaterialApp`将 `Theme`做为根widget了。

```
ThemeData({
  Brightness brightness, //深色还是浅色
  MaterialColor primarySwatch, //主题颜色样本
  Color primaryColor, //主色，决定导航栏颜色
```

```
  Color accentColor, //次级色，决定大多数Widget的颜色，如进度条、开关等。
  Color cardColor, //卡片颜色
  Color dividerColor, //分割线颜色
  ButtonThemeData buttonTheme, //按钮主题
  Color cursorColor, //输入框光标颜色
  Color dialogBackgroundColor,//对话框背景颜色
  String fontFamily, //文字字体
  TextTheme textTheme,// 字体主题，包括标题、body等文字样式
  IconThemeData iconTheme, // Icon的默认样式
  TargetPlatform platform, //指定平台，应用特定平台控件风格
  ...
})
```

- 创建全局主题：`MaterialApp` 接收一个theme的参数，类型为`ThemeData`，为App提供统一的颜色和字体。支持的参数可以在这里查看

```
new MaterialApp(
    title: title,
    theme: new ThemeData(
        brightness: Brightness.dark,
        primaryColor: Colors.lightBlue[800],
    ),
);
```

- 创建局部主题：如果想为某个页面使用不同于App的风格，可以使用`Theme`来覆盖App的主题

```
new Theme(
      data: new ThemeData(
          accentColor: Colors.yellow,
      ),
      child: new Text('Hello World'),
  );
```

- 覆盖（扩展）主题：如果不想覆盖所有的样式，可以继承App的主题，只覆盖部分样式，使用`copyWith`方法。

```
new Theme(
    data: Theme.of(context).copyWith(accentColor: Colors.yellow),
    child: new Text('use copyWith method'),
);
```

可滚动组件

1. **GridView**：网格布局，适用于多行多列的情况
   - 方式一：GridView.count

> 这种方式如果指定单个widget的宽高是不会起作用的，因为这里已经指定了每一行分成几列以及宽高比，还有边距等等

```
GridView.count(
      //水平子Widget之间间距
      crossAxisSpacing: 10.0,
      //垂直子Widget之间间距
      mainAxisSpacing: 30.0,
      //GridView内边距
      padding: EdgeInsets.all(10.0),
      //一行的Widget数量
      crossAxisCount: 2,
      //子Widget宽高比例
      childAspectRatio: 2.0,
      //子Widget列表
      children: widgetList(),
 );
```

- 方式二：GridView.builder

```
GridView.builder(
    itemCount: datas.length,
    //SliverGridDelegateWithFixedCrossAxisCount 构建一个横轴固定数量
Widget
    gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
      //横轴元素个数
        crossAxisCount: 3,
        //纵轴间距
        mainAxisSpacing: 20.0,
        //横轴间距
        crossAxisSpacing: 10.0,
        //子组件宽高长度比例
        childAspectRatio: 1.0),
    itemBuilder: (BuildContext context, int index) {
      //Widget Function(BuildContext context, int index)
      return getItemContainer(datas[index]);
    });
```

- 方式三：GridView.builder（SliverGridDelegateWithMaxCrossAxisExtent）

```
GridView.builder(
      itemCount: datas.length,
      itemBuilder: (BuildContext context, int index) {
        return getItemContainer(datas[index]);
      },
      gridDelegate: SliverGridDelegateWithMaxCrossAxisExtent(
        //单个子Widget的水平最大宽度
        maxCrossAxisExtent: 200,
```

```
        //水平单个子Widget之间间距
        mainAxisSpacing: 20.0,
        //垂直单个子Widget之间间距
        crossAxisSpacing: 10.0
      ),
    );
```

> 对于`SliverGridDelegateWithMaxCrossAxisExtent`而言，水平方向元素个数不再固定，其水平个数也就是有几列，由`maxCrossAxisExtent`和屏幕的宽度以及`padding`和`mainAxisSpacing`等决定。

- 方式四：GridView.custom

```
@override
  Widget build(BuildContext context) {
    List<String> datas = getDataList();
    return GridView.custom(
        gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
            crossAxisCount: 3, mainAxisSpacing: 10.0,
crossAxisSpacing: 20.0, ),
        childrenDelegate: SliverChildBuilderDelegate((context,
position) {
          return getItemContainer(datas[position]);
        }, childCount: datas.length));
  }
```

2. **SingleChildScrollView**：可滑动的view，类似于原生和RN中的ScrollView，其只能包含一个子元素

   常用属性：

   - **scrollDirection**：滚动方向，默认是垂直
   - **reverse**：是否按照阅读方向相反的方向滑动。
   - **padding**：填充距离
   - **primary**：是否使用 widget 树中默认的 PrimaryScrollController 。当滑动方向为垂直方向（scrollDirection值为Axis.vertical）并且controller没有指定时，primary默认为true
   - **physics**：此属性接受一个ScrollPhysics对象，它决定可滚动Widget如何响应用户操作，比如用户滑动完抬起手指后，继续执行动画；或者滑动到边界时，如何显示。默认情况下，Flutter会根据具体平台分别使用不同的ScrollPhysics对象，应用不同的显示效果，如当滑动到边界时，继续拖动的话，在iOS上会出现弹性效果，而在Android上会出现微光效果。如果你想在所有平台下使用同一种效果，可以显式指定，Flutter SDK中包含了两个ScrollPhysics的子类可以直接使用： ClampingScrollPhysics→Android下微光效果 / BouncingScrollPhysics→iOS下弹性效果
   - **controller**：此属性接受一个ScrollController对象。ScrollController的主要作用是控制滚动位置和监听滚动事件
   - **child**：子元素

3. **Table**：表格组件

   常用属性：

- **columnWidths**：每一列的宽度
- **defaultColumnWidth**：默认的每一列宽度值，默认情况下均分
- **textDirection**：文字方向，一般无需考虑
- **border**：表格边框
- **defaultVerticalAlignment**：每一个cell的垂直方向的alignment。 top：被放置在的顶部； middle：垂直居中； bottom：放置在底部； baseline：文本baseline对齐； fill：充满整个cell
- **textBaseline**：defaultVerticalAlignment为baseline的时候，会用到这个属性

```
Table(
  children: items,
  columnWidths: <int, TableColumnWidth>{
    0: FixedColumnWidth(100.0),
    1: FixedColumnWidth(40.0),
    2: FixedColumnWidth(100.0),
  },
  border: TableBorder.all(
      color: Color(0xffdddddd), width: 1.0, style: BorderStyle.solid),
)
```

4. 总结

- Flutter整个界面的构成元素
- 列表种类

5. 作业 && 答疑

完成如下界面

6. 下节课内容

- 路由的使用
- 数据持久化
- 动画
- Provider