第 3 章

Scaffold 组件的详细综述

Scaffold 在英文中的意思为脚手架、建筑架，在 Flutter 应用程序中，也可称为脚手架，也就是说在 Flutter 应用开发中，通过 Scaffold 可以搭建页面的基本结构，一个页面可以理解为由 3 个部分组成：header 头部，或者称之为标题栏；body 体，可称之为内容主体页面；bottom 脚，可称之为页面的尾部，例如 bottomBar。

对于 Scaffold 来讲，AppBar 就是它的头，body 中配置加载的 Widget 就是它的体，底部菜单栏就是它的尾部，如图 3-1 所示。

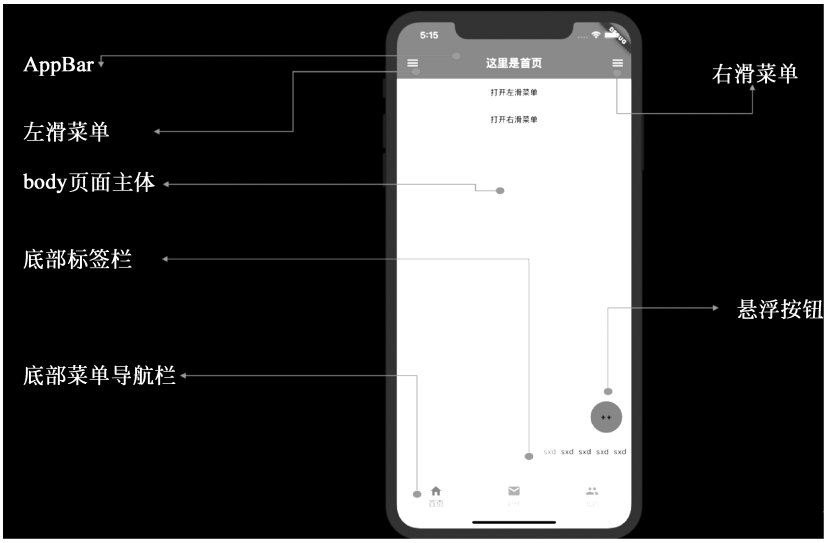

图 3-1　Scaffold 组件结构图

3.1　Scaffold 的基本使用

在实际应用开发中，当创建一个移动应用程序项目时，一般会设置一个启动初始化页面，那么这个页面一般没有标题，也没有底部内容区，只有一个内容 body 主区，在 Flutter 项目中，通过 main 函数，执行 runApp 方法通过 MaterialApp 组件来构建一个根布局 Widget，

然后通过 home 或者其他属性来配置这个启动页面如图 3-2 所示，代码如下：

```
//3.1 /lib/code2/main_data21.dart
//Scaffold 的基本使用,内容主体页面
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';

//应用入口
main() => runApp(themeDataFunction());

MaterialApp themeDataFunction() {
  return MaterialApp(home: FirstPage(),);
}
class FirstPage extends StatefulWidget {
  @override
  State<StatefulWidget> createState() {
    return FirstThemeState();
  }
}
class FirstThemeState extends State<FirstPage> {
  @override
  Widget build(BuildContext context) {
    //Scaffold 用来搭建页面的主体结构
    return Scaffold(
      //页面的主内容区
      //可以是单独的 StatefulWidget,也可以是当前页面构建的,如 Text 文本组件
      body: Center(child: Text("启动页面"),),);
  }
}
```

所以对于 Scaffold 来讲，它实现了基本的 Material Design 设计结构，一般可以用作单页面的父结构，在实际项目开发中，由标题栏与页面主体 body 构成的页面也是比较常见的，如图 3-3 所示，在 Flutter 中也可通过 Scaffold 来实现，代码如下：

```
//3.1 /lib/code2/main_data22.dart
//Scaffold 的基本使用,有 AppBar 的页面
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';

//应用入口
main() => runApp(themeDataFunction());

MaterialApp themeDataFunction() {
  return MaterialApp(home: FirstPage(),);
}
```

```
class FirstPage extends StatefulWidget {
  @override
  State < StatefulWidget > createState() {
    return FirstThemeState();
  }
}

class FirstThemeState extends State < FirstPage > {
  @override
  Widget build(BuildContext context) {
    //Scaffold 用来搭建页面的主体结构
    return Scaffold(
      //页面的头部
      appBar: AppBar(title: Text("标题"),),
      //页面的主内容区
      //可以是单独的 StatefulWidget,也可以是当前页面构建的,如 Text 文本组件
      body: Center(child: Text("显示日期"),),);
  }
}
```

图 3-2　启动页面

图 3-3　常用页面格局

Scaffold 组件的属性 appBar 用来配置一个 AppBar,用来构成页面的头部,类似于 Android 原生开发中的 AppBar 与 ToolBar,以及类似于 iOS 原生开发中的 UINavigationBar。

3.2 FloatingActionButton 的详细配置

FloatingActionButton 悬浮按钮，简称 FAB(下文中会使用简称)，Scaffold 组件中属性 FloatingActionButton 用来配置页面右下角的悬浮按钮功能，一般是一个圆形，中间有一个图标，悬浮按钮起源于 Material Design 设计理念的 z 轴概念，效果像是浮在页面之上，同 Android 原生开发中的 FloatingActionButton，在 Scaffold 中配置 FAB 按钮，程序运行效果与原生开发一致，如图 3-4 所示，悬浮按钮在 Flutter 中的实现代码如下：

```
//2.3 /lib/code2/main_data23.dart
//Scaffold 悬浮按钮
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';

//应用入口
main() => runApp(themeDataFunction());

MaterialApp themeDataFunction() {
  return MaterialApp(home: FirstPage(),);
}

class FirstPage extends StatefulWidget {
  @override
  State<StatefulWidget> createState() {
    return FirstThemeState();
  }
}
class FirstThemeState extends State<FirstPage> {
  @override
  Widget build(BuildContext context) {
    //Scaffold 用来搭建页面的主体结构
    return Scaffold(
      //页面的头部
      appBar: AppBar(title: Text("标题"),),
      //页面的主内容区
      //可以是单独的 StatefulWidget,也可以是当前页面构建的,如 Text 文本组件
      body: Center(child: Text("显示日期"),),
      //悬浮按钮
      floatingActionButton: FloatingActionButton(
        //一般建议使用 Icon
        child: Icon(Icons.add), onPressed: () {
          print("单击了悬浮按钮");
      },),);
```

```
    }
  }
```

图 3-4　悬浮按钮使用

3.2.1　FloatingActionButton 的类型

对于 FloatingActionButton 的属性 child，它是用来配置按钮上显示子 Widget 的，能够配置使用的范畴比较大，源码中推荐使用 Icon，但笔者认为诸如图 3-4 中所示效果，FAB 适用于强调操作类型，一个图标简单而容易理解，在互联网产品设计中，简单直白而不复杂，也是一种思路。

FAB 有 3 种类型：regular、mini、extended。regular 在这里与 mini 是相对的，不同之处在于 mini 类型是缩小的版本，默认创建的 FAB 是 regular 类型的，创建的 FAB 的 width 和 height 都为 56.0，可通过配置 FAB 的属性 mini 为 true，将创建的 FAB 类型由默认的 regular 标准尺寸修改为 mini 类型，此时创建的 FAB 的 width 和 height 都为 46.0，效果如图 3-5 所示，代码如下：

```
//3.2.1 /lib/code2/main_data24.dart
//FloatingActionButton 的类型:regular 类型
```

```
//此处通过 new Object 方式来创建的悬浮按钮为 regular 类型
FloatingActionButton buildFAB1(){
  return FloatingActionButton(
    //一般建议使用 Icon
    child: Icon(Icons.add), onPressed: () {
    print("单击了悬浮按钮");
  },);
}

//3.2.1 FloatingActionButton 的类型: mini 类型
//此处通过 new Object 方式来创建的悬浮按钮为 mini 类型
FloatingActionButton buildFAB2() {
  return FloatingActionButton(
    //一般建议使用 Icon
    child: Icon(Icons.add), onPressed: () {
    print("单击了悬浮按钮");
  },
    //默认是 false
    mini: true,);
}
```

图 3-5　悬浮按类型对比

对于 extended 类型，可以通过 FloatingActionButton. extended 方式来创建，也可以通过构造函数来创建，不过要指定属性 isExtended 的值为 true，与前两者不同的是它可以指定一个 Label 来显示文本，效果如图 3-6 所示，同理也限制了子 Widget 为 Icon 类型，对应添加子 Widget 的方式已不是 child，而是封装成 icon 属性，代码如下：

```
//3.2.1 /lib/code2/main_data24.dart
//FloatingActionButton 的类型:extended 类型
//通过 FloatingActionButton.extended 方式来创建悬浮按钮
FloatingActionButton buildFAB3() {
  return FloatingActionButton.extended(
    //通过 icon 来配置显示的图标
    icon: Icon(Icons.add), onPressed: () {
    print("单击了悬浮按钮");
  },
    //通过 labe 来添加文本信息
    label: Text("添加信息"),);
}
```

图 3-6　extended 类型 FAB

3.2.2 FloatingActionButton的常用属性使用分析

属性 tooltip 用来配置 FAB 长按时的提示文本，如图 3-7 所示。

图 3-7 FAB 的 tooltip 提示

图 3-7 所示效果对应代码如下：

```
//3.2.2 /lib/code2/main_data24.dart
//FloatingActionButton 的常用属性使用分析
FloatingActionButton buildFAB4(){
  return FloatingActionButton(
    child: Icon(Icons.add), onPressed: () { print("单击了悬浮按钮");},
  tooltip: "这里是 FAB!",
  );
}
```

属性 backgroundColor 用来配置 FAB 的背景色，splashColor 用来配置 FAB 单击时的水波纹颜色，foregroundColor 用来配置 FAB 的文字与图标中内容的颜色，如图 3-8 所示，代码如下：

```
//3.2.2 /lib/code2/main_data25.dart
//FloatingActionButton 的常用属性使用分析:颜色配置
```

```
FloatingActionButton buildFAB5(){
  return FloatingActionButton(
    child: Icon(Icons.add),
    //单击事件响应
    onPressed: () {},
    //背景色为红色
    backgroundColor: Colors.red,
    //单击水波纹颜色为黄色
    splashColor: Colors.yellow,
    //前景色为紫色
    foregroundColor: Colors.deepPurple,
  );
}
```

图 3-8　FAB 的颜色配置

对于颜色配置，这也属于主题配置的范畴，在第 2 章 MaterialApp 组件中提到默认主题配置 ThemeData，为应变 App 中的多主题样式使用，笔者在这里建议还是在 MaterialApp 组件中通过 FloatingActionButtonThemeData 来配置 FAB 的使用样式，对于 FAB 来讲，MaterialApp 配置对应代码如下：

```
//3.2.2 /lib/code2/main_data26.dart
//FloatingActionButton 的常用属性使用分析,MaterialApp 中主题配置
```

```
MaterialApp(
  home: FirstPage(),
  //主题配置
  theme: ThemeData(
    //FAB 悬浮按钮主题样式配置
    floatingActionButtonTheme: FloatingActionButtonThemeData(
      //背景色为红色
      backgroundColor: Colors.red,
      //单击水波纹颜色为黄色
      splashColor: Colors.yellow,
      //前景色为紫色
      foregroundColor: Colors.deepPurple,
      //默认显示下的阴影高度
      elevation: 6.0,
      //点按下去时阴影的高度
      highlightElevation: 10.0,
      //不可被单击时的阴影的高度
      disabledElevation: 1.0,
    ),
  ),);
```

3.2.3 FloatingActionButton 的 shape 属性分析

shape 用来配置 FAB 的形状，在 FAB 创建时，默认的形状是圆形的，默认使用的是 CircleBorder，创建的是圆形的 FAB，对于 CircleBorder 还可以设置边框，如图 3-9 所示，代码如下：

```
//3.2.3 /lib/code2/main_data27.dart
//FAB 的 shape 属性分析，默认创建使用的 shape
ShapeBorder fabShapeBorder1() {
  //圆形
  return CircleBorder(
    //配置边框
      side: BorderSide(
        //边框颜色
          color: Colors.blue,
          //边框的宽度
          width: 4.0,
          //边框的样式，solid 为实线，none 为不显示边框
          style: BorderStyle.solid),
  );
}
```

可以通过修改 shape 的值来修改 FAB 的形状，如图 3-10 所示，代码如下：

```
//3.2.3 /lib/code2/main_data28.dart
//FloatingActionButton 的常用属性使用分析，shape 属性分析
```

```
MaterialApp(
  home: FirstPage(),
  //主题配置
  theme: ThemeData(
    ... //省略
    //FAB悬浮按钮主题样式配置
    floatingActionButtonTheme: FloatingActionButtonThemeData(
    ... //省略
      //用来指定FAB的形状
      shape:RoundedRectangleBorder(),
    ),
  ),);
```

图 3-9　设置边框的悬浮按钮

对于 RoundedRectangleBorder 圆角矩形来讲，也可以使用 side 来配置边框，使用方式与 CircleBorder 中的 side 配置的 BorderSide 一致，不过它还可以配置矩形四周的圆角，如图 3-11 所示，代码如下：

```
//3.2.3 /lib/code2/main_data29.dart
//FAB的shape属性分析,使用圆角矩形 shape
ShapeBorder fabShapeBorder2() {
```

```
    //圆形
    return RoundedRectangleBorder(
      //设置四周的圆角
      borderRadius: BorderRadius.circular(10),
    );
  }
```

图 3-10　修改 shape 的悬浮按钮

borderRadius 取值为 BorderRadius.circular，可理解为将当前矩形的上下左右 4 个边角同时设置为圆角，对于圆角一般由 x 轴方向的一个半径和 y 轴方向的一个半径决定，circular 指定的就是由这两个半径值保持一致所限制形成的圆角，如图 3-11 所示。

BorderRadius 也可以单独指定一个角的圆角，如图 3-12 所示，代码如下：

```
//3.2.3  /lib/code2/main_data29.dart
//FAB 的 shape 属性分析,使用圆角矩形 shape
ShapeBorder fabShapeBorder3() {
  //圆形
  return RoundedRectangleBorder(
    //设置左上角的圆角
    borderRadius: BorderRadius.only(topLeft: Radius.elliptical(10, 20)),
  );
}
```

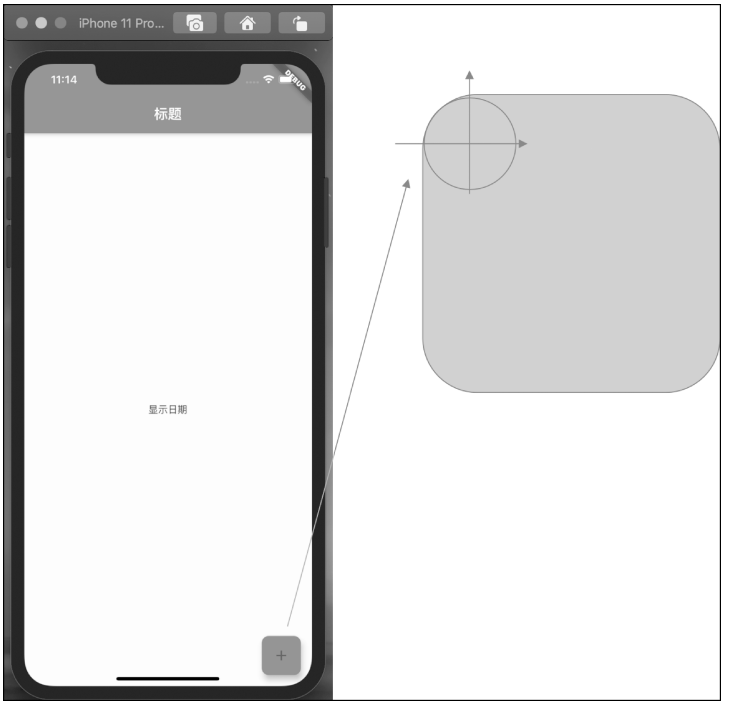

图 3-11　设置四周圆角

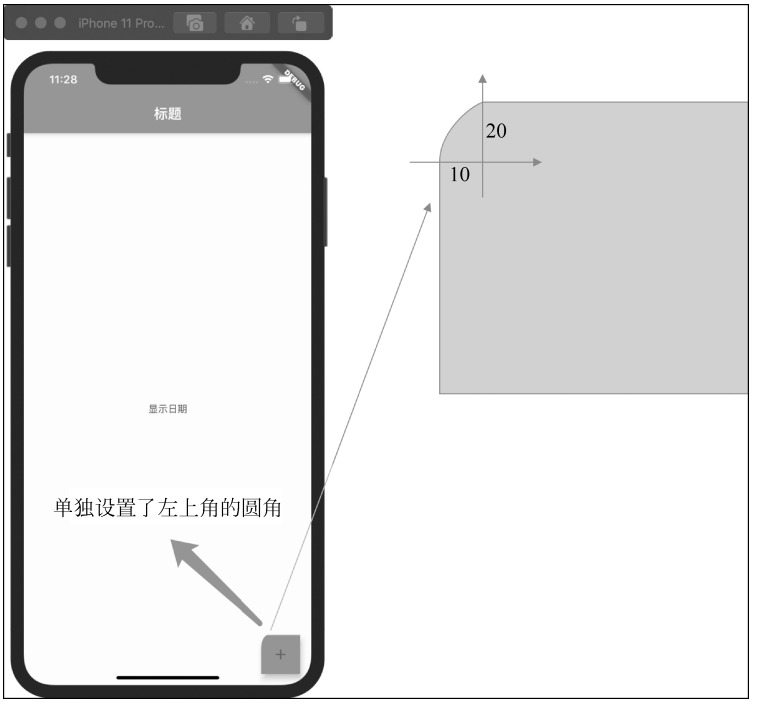

图 3-12　单独设置左上角圆角

对于 Scaffold 的属性 floatingActionButtonLocation 是用来配置 FAB 的位置的，Scaffold 配置的 FAB 可选位置如图 3-13 所示。

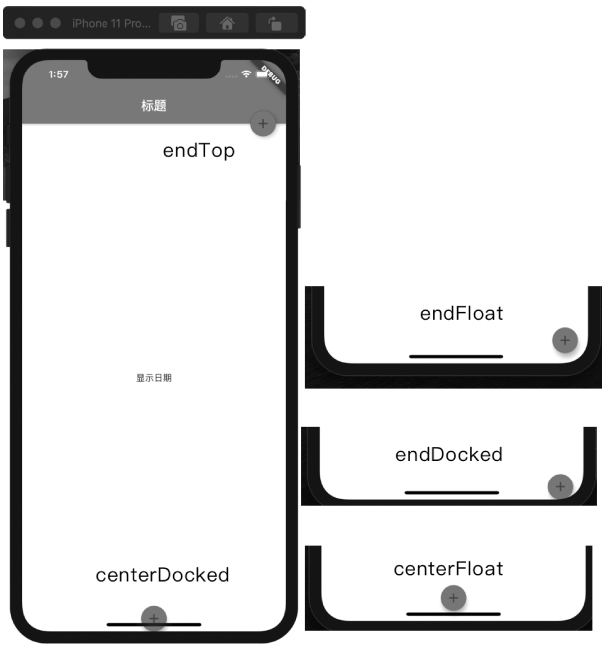

图 3-13　FAB 的位置说明

3.3　Drawer 配置侧拉页面

Scaffold 的 drawer 用来配置页面左侧侧拉页面，属性 endDrawer 配置右侧侧拉页面，对于 drawer 来讲，它接收的是一个 Widget，可以是一个容器，也可以是一个 Text，还可以是一 StatefulWidget，在实际项目开发中，侧拉页面一般是由一个 ListView 或是一个 Column 线性布局多个条目，如图 3-14 所示，当配置了 Scaffold 的 drawer 内容时，就会在 AppBar 的左侧多出一个菜单按钮，当单击这个按钮时，就会从左侧向右滑出一个侧拉页面，同时用户可从页面左侧向右侧滑动而触发出这个页面。

当配置了 Scaffold 的 endDrawer 属性时，在 AppBar 的右侧多出一个菜单按钮，同理单击时，会有一个页面从当前页面的右侧滑出来，用户也可以通过从当前页面右侧边缘向左滑动而将这个页面触发出来，代码如下：

```
//3.3  /lib/code2/main_data30.dart

//drawer 配置侧拉页面
import 'package:flutter/cupertino.dart';
```

```
import 'package:flutter/material.dart';
//应用入口
main() => runApp(themeDataFunction());
MaterialApp themeDataFunction() {
  return MaterialApp(
    home: FirstPage6(),);
}
class FirstPage6 extends StatefulWidget {
  @override
  State<StatefulWidget> createState() {
    return FirstThemeState();
  }
}
class FirstThemeState extends State<FirstPage6> {
  @override
  Widget build(BuildContext context) {
    //Scaffold 用来搭建页面的主体结构
    return Scaffold(
      //左侧侧拉页面
      drawer:buildDrawer() ,
      //右侧侧拉页面
      endDrawer: buildDrawer(),
      //页面的头部
      appBar: AppBar(title: Text("标题"),),
      //页面的主内容区
      //可以是单独的 StatefulWidget,也可以是当前页面构建的,如 Text 文本组件
      body: Center(child: Text("显示日期"),),
      );
  }
  //封装方法
  Container buildDrawer(){
    //Container 可看作一个容器,用来包裹一些 Widget
    return Container(
      //背景颜色
      color: Colors.white,
      width: 200,
      //Column 可以让子 Widget 在垂直方向线性排列
      child: Column(
        children:<Widget>[
          Container(color: Colors.blue,height: 200,child: Text("这是一个 Text"),),
          Container(color: Colors.red,height: 200,child: Text("这是一个 Text2"),),
        ],
      ),
    );
  }
}
```

图 3-14　配置侧拉页面

3.3.1　用户信息组件 UserAccountsDrawerHeader

UserAccountsDrawerHeader 从组件名字中就可得知此组件是专门用来配置侧拉页面中用户账号信息的，如图 3-15 所示，代码如下：

```
//3.3 /lib/code2/main_data31.dart
//drawer 配置侧拉页面的 UserAccountsDrawerHeader 组件
Container buildDrawer2() {
  //Container 可看作一个容器，用来包裹一些 Widget
  return Container(
    //背景颜色
    color: Colors.white,
    width: 200,
    //Column 可以让子 Widget 在垂直方向线性排列
    child: Column(
      children: <Widget>[
```

```
          UserAccountsDrawerHeader(
            //显示的二级标题
            accountEmail: Text('928 *** 994@qq.com'),
            //显示的小标题
            accountName: Text('这里是 Drawer'),
            //小箭头单击响应
            onDetailsPressed: () {},
            //当前显示的背景图片
            currentAccountPicture: CircleAvatar(
              child: Icon(Icons.message),
            ),
          ),
        ],
      ),
    );
  }
```

图 3-15　用户信息组件的侧拉页面

UserAccountsDrawerHeader 显示的布局样式是定义好的一个范式，在实际应用开发中似乎用得较少，因为很少有设计师会将用户的一个简介信息通过这样的方式展现出来，这个组件还有一个 decoration 属性，用来设置边框样式。

3.3.2 DrawerHeader

在 3.3.1 节中，使用 UserAccountsDrawerHeader 来设置并显示侧拉页面中用户的简介信息，它的固定样式如图 3-15 所示，在实际开发中与设计师眼中的布局样式可能不相符合，此时，可通过 DrawerHeader 来构建一个自定义的页面，效果如图 3-16 所示，其属性 child 用来配置显示的内容，curve 用来配置侧拉页面滑出的动画效果，代码如下：

```
//3.3 /lib/code2/main_data32.dart
//drawer 配置侧拉页面,自定义的 DrawerHeader
Container buildDrawer() {
  //Container 可看作一个容器,用来包裹一些 Widget
  return Container(
    //背景颜色
    color: Colors.white,
    width: 200,
    //Column 可以让子 Widget 在垂直方向线性排列
    child: Column(
      children: <Widget>[
        DrawerHeader(
          //设置 Header 的上下左右内边距为 0
          padding: EdgeInsets.zero,
          //显示的具体内容
          child: buildDrawerBody(),
          //侧拉页面滑出的动画效果,先快后慢
          curve: Curves.fastOutSlowIn,
        ),
      ],
    ),
  );
}

//Stack 为帧布局页面,Widget 可重合显示
Stack buildDrawerBody() {
  return Stack(
    children: <Widget>[
      //可以用来设置背景图片
      Container(color: Colors.grey,),
      //Align 用来对齐组件
      Align(
        //底部左对齐
        alignment: FractionalOffset.bottomLeft,
        //设置用户的显示信息
```

```
          child: Container(
            //底部外边距为 12
            margin: EdgeInsets.only(bottom: 12),
            child: Row(
              children: <Widget>[
                SizedBox(width: 12,),
                //圆形的头像
                CircleAvatar(
                  child: Icon(Icons.message),
                ),
                SizedBox(width: 16,),
                Column(

                  //使用 Column 线性布局包裹子 Widget 大小
                  mainAxisSize: MainAxisSize.min,
                  children: <Widget>[
                    Text("这里是用户的姓名"),
                    SizedBox(height: 8,),
                    Text("这里是用户的简介"),
                  ],
                ),
              ],
            ),),)
        ],
      );
    }
```

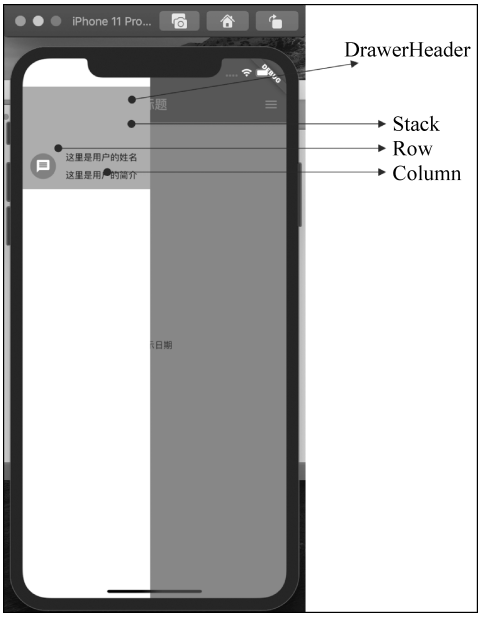

图 3-16　自定义实现 DrawerHeader

3.3.3 单击按钮打开与关闭侧拉页面

在实际项目开发中，当配置了左侧拉页面时，如果要自定义 AppBar 默认打开侧拉页面的显示按钮，如图 3-17 所示需要配置 AppBar 中的 leading 属性，代码如下：

```
//3.3.3 /lib/code2/main_data33.dart
//单击按钮打开与关闭侧拉页面，修改默认 AppBar 的按钮配置
class FirstThemeState extends State<FirstPage6> {
  @override
  Widget build(BuildContext context) {
    //Scaffold 用来搭建页面的主体结构
    return Scaffold(
      //左侧侧拉页面
      drawer: buildDrawer(),
      //右侧侧拉页面
      endDrawer: buildDrawer(),
      //页面的头部
      appBar: AppBar(title: Text("标题"),
      leading: InkWell(onTap: (){

      },child: Icon(Icons.add),),),
      //页面的主内容区
      //可以是单独的 StatefulWidget，也可以是当前页面构建的，如 Text 文本组件
      body: Center(child: Text("显示日期"),),
    );
  }
```

图 3-17 自定义 leading 按钮

当不需要显示这个按钮时，可将 leading 值赋为 null。

当配置了 leading 自定义的按钮样式后，就需要在按钮的单击事件中配置打开侧拉页面的代码，代码如下：

```
//打开左侧拉页面
Scaffold.of(context).openDrawer();
```

当直接在配置的按钮中调用打开侧拉页面的方法时，有时会出报错信息，日志代码如下：

```
═══════════ Exception caught by gesture ═══════════
The following assertion was thrown while handling a gesture:
Scaffold.of() called with a context that does not contain a Scaffold.
```

这里因为在 Scaffold 所配置的按钮中直接引用了 context，这个 context 是由 build 方法传入的，与 Scaffold 是同一级别的，从源码中可以看到 Scaffold 的 of 方法会根据当前传入的 context 通过 findAncestorStateOfType 方法去找当前对应的 ScaffoldState 实例，找到后成功返回，找不到则抛出异常。

而在这个 findAncestorStateOfType 方法中，是从当前 of 方法中传入的 context 的父节点开始查找的。而从 build 方法中传入的 context 的父节点中并没有 ScaffoldState 实例，因为 Scaffold 与 build 方法传入的 context 是在同一节点中。

解决方案是在 Scaffold 的 AppBar 的 leading 中再使用 Builder 组件包裹，然后再调用 openDrawer 方法，此时就可以正常打开侧拉页面了，代码如下：

```
//3.3.3 /lib/code2/main_data34.dart
//单击按钮打开与关闭侧拉页面，配置 Scaffold.of(context)
class FirstThemeState extends State<FirstPage6> {
  @override
  Widget build(BuildContext context) {
    //Scaffold 用来搭建页面的主体结构
    return Scaffold(
      //禁用页面的侧滑效果
      drawerEdgeDragWidth: 0,
      //左侧侧拉页面
      drawer: buildDrawer(),
      //右侧侧拉页面
      endDrawer: buildDrawer(),
      //页面的头部
      appBar: AppBar(title: Text("标题"),
        //使用 Builder 包裹后，在 Scaffold.of 中使用 Builder 回调 context
        //这个 context 的父级就是 ScaffoldState 了
```

```
          leading: Builder(builder: (BuildContext context) {
            return InkWell(onTap: () {
              //打开左侧拉页面
              Scaffold.of(context).openDrawer();
            }, child: Icon(Icons.add),);
          }),),
        body: ...  //省略
      );
    }
```

打开侧拉页面后，单击页面的遮罩层，侧拉页面会自动收缩回去，有时项目要求在页面中单击一个按钮，然后打开一个新的页面，在打开新的页面时，当前的侧拉页面要关闭，关闭侧拉页面的代码如下：

```
Navigator.of(context).pop();
```

在实际应用项目开发中，有时不需要通过左滑或者右滑将侧拉页面滑出来，这里就需要禁用，此时可通过配置 Scaffold 组件的 drawerEdgeDragWidth 为 0 来实现。

3.4 BottomNavigationBar 配置底部导航栏菜单

在 Scaffold 中，通过 bottomNavigationBar 来配置底部导航栏。结合 BottomNavigationBar 组件来实现，创建一个实战，效果如图 3-18 所示。代码如下：

```
//3.3 /lib/code2/main_data35.dart
//bottomNavigationBar 配置底部导航栏菜单
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';

//应用入口
main() =>
    runApp(MaterialApp(
      home: FirstPage(),),);

class FirstPage extends StatefulWidget {
  @override
  State<StatefulWidget> createState() {
    return FirstThemeState();
  }
}
class FirstThemeState extends State<FirstPage> {
  @override
```

```
Widget build(BuildContext context) {
  //Scaffold 用来搭建页面的主体结构
  return Scaffold(
    //页面的头部
    appBar: AppBar(title: Text("标题"),),
    //页面的主内容区
    //可以是单独的 StatefulWidget,也可以是当前页面构建的,如 Text 文本组件
    body: Center(child: Text("当前选中的页面是 $ _tabIndex"),),
    //底部导航栏
    bottomNavigationBar: buildBottomNavigation(),
  );
}

//选中的标签
int _tabIndex = 0;
//底部导航栏使用的图标
List < Icon > normalIcon = [
  Icon(Icons.home),
  Icon(Icons.message),
  Icon(Icons.people)
];
//底部导航栏使用的标题文字
List < String > normalTitle =[
  "首页",
  "消息",
  "我的"
];

//构建底部导航栏
BottomNavigationBar buildBottomNavigation(){
  //创建一个 BottomNavigationBar
  return new BottomNavigationBar(
    items: < BottomNavigationBarItem >[
      new BottomNavigationBarItem(
          icon: normalIcon[0], title: Text(normalTitle[0])),
      new BottomNavigationBarItem(
          icon:  normalIcon[1], title: Text(normalTitle[1])),
      new BottomNavigationBarItem(
          icon:  normalIcon[2], title: Text(normalTitle[2])),
    ],
    //显示效果
    type: BottomNavigationBarType.fixed,
    //当前选中的页面
    currentIndex: _tabIndex,
    //导航栏的背景颜色
    backgroundColor: Colors.white,
    //固定图标与文字的颜色
```

```
    //fixedColor: Colors.deepPurple,
          //选中时图标与文字的颜色
          selectedItemColor: Colors.blue,
          //未选中时图标与文字的颜色
          unselectedItemColor: Colors.grey,
          //图标的大小
          iconSize: 24.0,
          //单击事件
          onTap: (index) {
            setState(() { _tabIndex = index; });
          },
        );
      }
    }
```

图 3-18 BottomNavigationBar 导航栏

3.4.1 items 属性分析

BottomNavigationBar 的属性 items 用来配置底部导航栏的每一个选项，它接收的是 BottomNavigationBarItem 组件类型，关于 BottomNavigationBarItem 的属性分析如表 3-1 所示。

表 3-1　BottomNavigationBarItem 的属性

属　性	说　明	类　型
icon	默认显示的图标	Widget
activeIcon	选中时显示的图标	Widget
title	对应的标题	Widget
backgroundColor	shifting 时的背景颜色模式下的背景颜色	Color

3.4.2　type 属性分析

BottomNavigationBar 的 type 属性影响的是底部导航栏切换时的动画效果，源码代码如下：

```
//3.4.2 type 属性分析
enum BottomNavigationBarType {
  //The [BottomNavigationBar]'s [BottomNavigationBarItem]s have fixed width.
  fixed,

  //The location and size of the [BottomNavigationBar] [BottomNavigationBarItem]s
  //animate and labels fade in when they are tapped.
  shifting,
}
```

bottomNavigationBar 默认配置的 type 是 fixed，fixed 限制的切换效果是在导航栏菜单切换时图标和文字标题会有微缩放的动画效果，而对于 shifting 来讲，切换动画效果更明显，在 shifting 模式下，只有当前选中的 item 的图标与文字才会显示出来，而未选中的 item 的标题文字是隐藏的，如图 3-19 所示。

图 3-19　shifting 模式

3.4.3 bottomNavigationBar 结合独立的 StatefulWidget 使用

在图 3-18 所示效果中，页面 body 显示的只是一个 Widget，在实际项目开发中，这里往往配置的是单独的页面，在这里笔者创建了 3 个页面，代码如下：

```
//3.4.3  /lib/code2/main_data36.dart

//bottomNavigationBar 结合独立的 StatefulWidget 使用,首页面
class ScaffoldHomeItemPage extends StatefulWidget {
  //页面标识
  int pageIndex;
   //构造函数
  ScaffoldHomeItemPage(this.pageIndex,{Key key}) : super(key: key);
  @override
  State < StatefulWidget > createState() {
    return ScaffoldHomeItemState();
  }
}
//3.4.3  /lib/code2/main_data36.dart
class ScaffoldHomeItemState extends State < ScaffoldHomeItemPage > {
  //页面创建时初始化函数
  @override
  void initState() {
    super.initState();
    print("页面创建 ${widget.pageIndex}");
  }

  @override
  Widget build(BuildContext context) {
    return Center(child: Text("当前页面标识为 ${widget.pageIndex}"),
    );
  }

  //页面销毁时回调函数
  @override
  void dispose() {
    super.dispose();
    print("页面消失 ${widget.pageIndex}");
  }
}
```

第 2 个页面与第 3 个页面源码与第 1 个页面效果一样，然后在 bottomNavigationBar 中配置使用，代码如下：

```
//3.4.3  /lib/code2/main_data36.dart
List<Widget> bodyWidgetList = [
  ScaffoldHomeItemPage(0),
  ScaffoldHomeItemPage1(1),
  ScaffoldHomeItemPage2(2),
];

@override
Widget build(BuildContext context) {
  //Scaffold 用来搭建页面的主体结构
  return Scaffold(
    //页面的头部
    appBar: AppBar(title: Text("标题"),),
    //页面的主内容区
    //可以是单独的 StatefulWidget,也可以是当前页面构建的,如 Text 文本组件
    body:bodyWidgetList[_tabIndex],
    //底部导航栏
    bottomNavigationBar: buildBottomNavigation(),
  );
}
```

_tabIndex 是底部导航栏当前选中的页面角标，每当底部导航栏进行切换时，调用 setState 方法进行页面刷新，在 Scaffold 的 body 中也会动态地从当前配置页面的 bodyWidgetList 中取出对应的页面进行渲染。

3.4.4　bottomNavigationBar 页面保活解决方案

在 3.4.3 节的实例中，默认显示的页面是_tabIndex 为 0 首页面，然后依次单击“消息”、“我的”陆续加载第 2 个页面与第 3 个页面，body 中的页面正常切换，如图 3-20 所示。

在 Android Studio 中的日志输出控制台可看到的日志如下：

```
flutter: 页面创建 0
flutter: 页面创建 1
flutter: 页面消失 0
flutter: 页面创建 2
flutter: 页面消失 1
```

从日志信息可以得出结论，在单击底部导航按钮进行页面切换时，当前配置的这几个页面会被销毁，在实际应用开发中，一般这种页面结构用在 App 的首页面，用户单击切换的频率相对高，所以每切换一次页面重新加载渲染一次，这种无论从设计方面还是从体验方面，以及从流量方面都相对不好，最好的结果就是这样的页面只创建一次就可以了。

为了实现页面只创建一次的程序设计，笔者在这里提出一个方案可以达到目的，首先

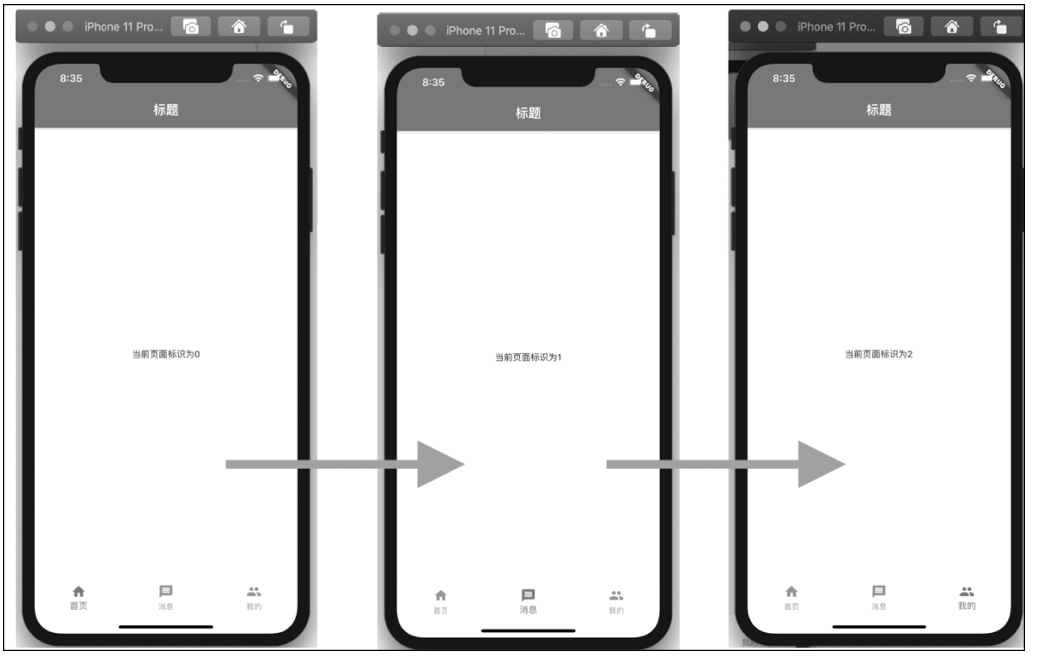

图 3-20　页面切换效果图

Scaffold 的 body 用来加载页面的主体信息，这个 body 加载一个帧布局 Stack，接着将这 3 个页面一次性全部加载并创建出来，然后将这 3 个页面重叠显示在帧布局中，再结合透明控制组件 Opacity 将当前选中的页面透明度设置为 1，从而显示出来，其他的页面透明度设置为 0，用户不可见，运行程序，最后再进行 3 个页面重复单击底部导航栏进行切换，日志输入如下：

```
Syncing files to device iPhone 11 Pro Max...
flutter: 页面创建 0
flutter: 页面创建 1
flutter: 页面创建 2
```

透明度结合帧布局确实实现了这一解决方案，代码如下：

```
//3.4.4 /lib/code2/main_data40.dart
//bottomNavigationBar 页面保活解决方案
@override
Widget build(BuildContext context) {
  //Scaffold 用来搭建页面的主体结构
  return Scaffold(
    //页面的头部
    appBar: AppBar(title: Text("标题"),),
```

```
      //页面的主内容区
      body:buildBodyFunction(),
      //底部导航栏
      bottomNavigationBar: buildBottomNavigation(),
    );
  }

  Widget buildBodyFunction(){
    //帧布局结合透明布局
  return Stack(children: <Widget>[
     Opacity(opacity: _tabIndex == 0?1:0,child: ScaffoldHomeItemPage(0),),
     Opacity(opacity: _tabIndex == 1?1:0,child: ScaffoldHomeItemPage(1),),
     Opacity(opacity: _tabIndex == 2?1:0,child: ScaffoldHomeItemPage(2),),
  ],);
  }
```

Opacity 组件用来设置子 Widget 的透明度，属性 opacity 接收一个 double 值，取值范围为 0.0～1.0，从透明到不透明，0.0 代表透明，1.0 代表不透明。

3.5　BottomAppBar 配置底部导航栏菜单

BottomAppBar 组件在 Flutter 中可用来配置底部导航菜单栏，同样可实现如图 3-20 所示的效果，代码如下：

```
//3.5 /lib/code2/main_data37.dart
//BottomAppBar 配置底部导航栏菜单
@override
Widget build(BuildContext context) {
   return Scaffold(
     //页面的头部
     appBar: AppBar(title: Text("标题"),),
     //页面的主内容区
     body:buildBodyFunction(),
     //底部导航栏
     bottomNavigationBar: buildBottomNavigation(),
   );
}
//构建底部导航栏
BottomAppBar buildBottomNavigation(){
  return BottomAppBar(
    //底部导航栏的背景
    color: Colors.white,
    //Row 中的子 Widget 在水平方向非线性排列
```

```
      child: Row(
        //使每一个子 Widget 平均分配 Row 的宽度
        mainAxisAlignment: MainAxisAlignment.spaceAround,
        children: <Widget>[
          buildBottomItem( 0, Icons.home, "首页"),
          buildBottomItem( 1, Icons.message, "消息"),
          buildBottomItem( 2, Icons.people, "我的"),
        ],
      ),
    );
  }
```

对于 buildBottomItem 方法是用来构建底部导航菜单栏的图标与标题排列的，如图 3-21 所示，底部导航栏通过线性布局 Column 与 Row 结合构成。

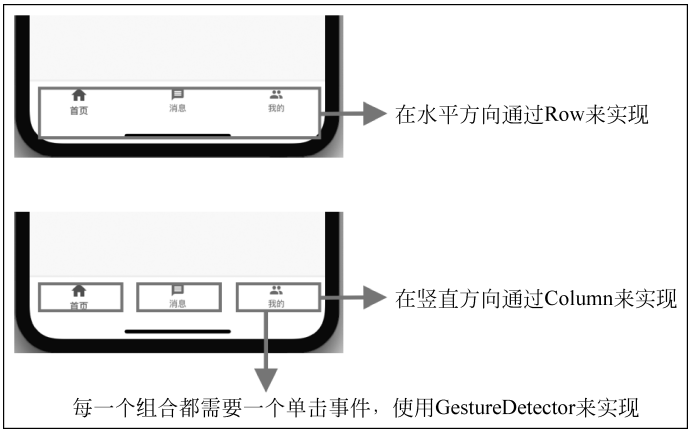

图 3-21　底部导航栏结构说明

对于图标与标题的组件，代码如下：

```
//3.5 /lib/code2/main_data37.dart
//BottomAppBar 配置底部导航栏菜单，图标与代码的组合
//[index]为每个条目对应的角标
//[iconData]为每个条目对应的图标
//[title]为每个条目对应的标题
buildBottomItem( int index, IconData iconData, String title) {
  //未选中状态的样式
  TextStyle textStyle = TextStyle(fontSize: 12.0,color: Colors.grey);
  MaterialColor iconColor = Colors.grey;
  double iconSize = 20;
  EdgeInsetsGeometry padding =   EdgeInsets.only(top: 8.0);
```

```
if(_tabIndex == index){
  //选中状态的文字样式
  textStyle = TextStyle(fontSize: 13.0,color: Colors.blue);
  //选中状态的按钮样式
  iconColor = Colors.blue;
  //状态图标的大小
  iconSize = 25;
  padding =   EdgeInsets.only(top: 6.0);
}
//上下竖直方向排列的图标与标题文字
Widget padItem = Padding(
    padding: padding,
    child: Container(
      color: Colors.white,
      child: Center(
        child: Column(
          children: <Widget>[
            Icon(
              iconData,
              color: iconColor,
              size: iconSize,
            ),
            Text(
              title,
              style: textStyle,
            )
          ],
        ),
      ),
    ),
  );
//Row 中通过 Expanded 进行权重布局排列
Widget item = Expanded(
  flex: 1,
  child: new GestureDetector(
    onTap: () {
      if (index != _tabIndex) {
        setState(() {
          _tabIndex = index;
        });
      }
    },
    child: SizedBox(
      height: 52,
      child: padItem,
```

```
        ),
      ),
    );
    return item;
  }
```

在 Scaffold 中，使用 BottomAppBar 与悬浮按钮 FloatingActionButton 结合使用，可以达到如图 3-22 所示页面效果，代码如下：

```
//3.5.1 /lib/code2/main_data38.dart
//BottomAppBar 结合悬浮按钮使用，Scaffold 中的配置
@override
Widget build(BuildContext context) {
  //Scaffold 用来搭建页面的主体结构
  return Scaffold(
    //页面的头部
    appBar: AppBar(title: Text("标题"),),
    //页面的主内容区
    body:buildBodyFunction(),
    //底部导航栏
    bottomNavigationBar: buildBottomNavigation(),
    //悬浮按钮的位置
    floatingActionButtonLocation: FloatingActionButtonLocation.centerDocked,
    //悬浮按钮
    floatingActionButton: FloatingActionButton(
      child: const Icon(Icons.add),
      onPressed: () {
          print("add press ");},
    ),
  );
}
```

这里需要注意的是悬浮按钮的位置需要配置为 Docked 模式，在对应的 BottomAppBar 组件中，也需要做相应的配置，代码如下：

```
//3.5.1 /lib/code2/main_data38.dart
//BottomAppBar 结合悬浮按钮使用，BottomAppBar 的配置
BottomAppBar buildBottomNavigation(){
  return BottomAppBar(
    //悬浮按钮与其他菜单栏的结合方式
    shape: CircularNotchedRectangle(),
    //FloatingActionButton 和 BottomAppBar 之间的差距
    notchMargin: 6.0,
```

```
        //底部导航栏的背景
        color: Colors.white,
        //Row 中的子 Widget 在水平方向非线性排列
        child: Row(
          //使每一个子 Widget 平均分配 Row 的宽度
          mainAxisAlignment: MainAxisAlignment.spaceAround,
          children: <Widget>[
            ...  //省略
          ],
        ),
      );
    }
```

图 3-22　不规则导航栏

3.6　底部标签栏 bottomSheet

Scaffold 的属性 bottomSheet 用来配置底部固定的标签样式提示栏，如图 3-23 所示。

bottomSheet 在这里接收的是一个 Widget，也就是说可以配置任意的 Widget，不过在实际项目开发中，如果使用了 bottomSheet，一般配置的是一个水平方向线性排列的内容，

图 3-23　bottomSheet 配置的标签样式提示栏

如使用 Row 布局或者 Stack 帧布局等，代码如下：

```
//3.6 /lib/code2/main_data39.dart
//底部标签栏 bottomSheet
@override
Widget build(BuildContext context) {
  //Scaffold 用来搭建页面的主体结构
  return Scaffold(
    //页面的头部
    appBar: AppBar(title: Text("标题"),),
    //页面的主内容区
    body:buildBodyFunction(),
    //固定的标签栏
    bottomSheet: Container(
      color: Colors.blue, height: 44, child: Row(children: <Widget>[
      Container(margin: EdgeInsets.only(left: 10, right: 2),
        color: Colors.white,
        child: Text("标签 1"),),
      Container(margin: EdgeInsets.only(left: 10, right: 2),
        color: Colors.white,
```

```
            child: Text("标签 1"),),
          Container(margin: EdgeInsets.only(left: 10, right: 2),
            color: Colors.white,
            child: Text("标签 1"),),
          Container(margin: EdgeInsets.only(left: 10, right: 2),
            color: Colors.white,
            child: Text("标签 1"),),

        ],),),
        //底部导航栏
        bottomNavigationBar: buildBottomNavigation(),
      );
    }
```

小结

本章详细介绍了用来构建页面主体结构的 Scaffold，笔者建议 push 的普通页面全部使用 Scaffold 来组装，一个独立的视图中建议只使用一个 Scaffold，在 PageView、TabBarView 中会有多个子 Widget，在 PageView 与 TabBarView 所在页面的 Widget 视图中使用 Scaffold 构建，其中在每个子项中建议不使用 Scaffold。