第 5 章

Flutter 框架基本组件的使用

Flutter 框架设计目标为给用户呈现丰富多彩的外观。我们将从基本的组件开始，逐步讲解 Flutter 中各种组件的用法。这一章，主要讲解 Container 容器组件、Text 文本组件、Icon 图标组件、Image 图片组件、屏幕适配、生命周期、StatefulWidget 有状态组件和 StatelessWidget 无状态组件等。

5.1 Flutter 架构组成

Flutter 被设计为一个可扩展的分层系统，各个组件相互独立，上层组件依赖下层组件。组件无法越权访问更底层的内容，框架层中的部分都是可选且可替代的。

Flutter 框架结构图分为三部分：Framework、Engine 和 Embedder，如图 5-1 所示。

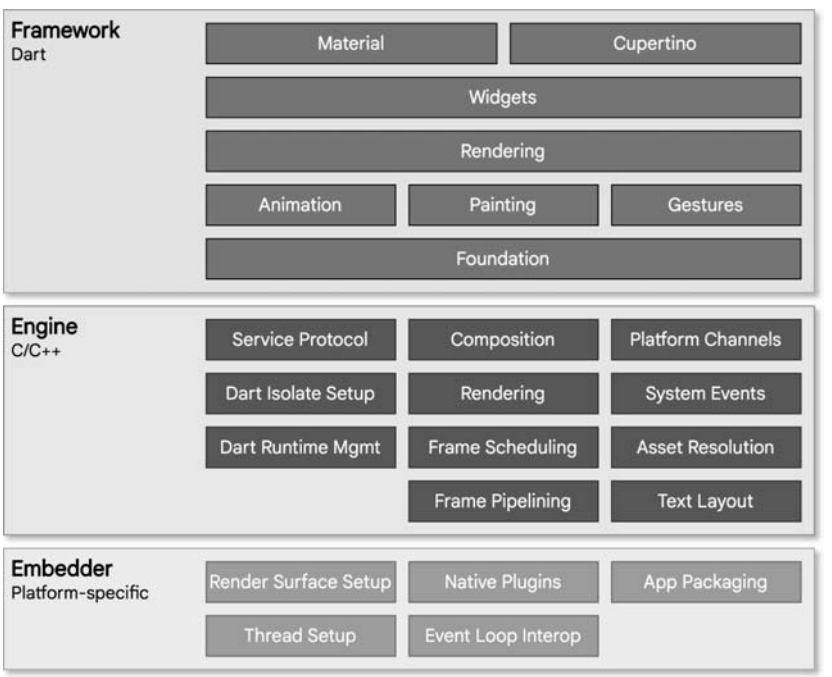

图 5-1　Flutter 架构层

Framework 指使用 Dart 编写的 Flutter 框架，其中 Material 指实现了材质设计(Material Design)的 Flutter 组件，是官方推荐使用的组件类型。在使用时需要导入 import package:flutter/material.dart 包。本书也是以 Material Design 为主讲解 Flutter 组件的使用。

Cupertino 为 iOS 风格的组件，它有很多组件与 Material Design 类似。

Widgets 库是构建 Flutter 框架的基础，Rendering 库实现了 Flutter 的布局和后台绘制，Animation 库实现了 Flutter 的动画功能，Painting 库包括包装了 Flutter 引擎的绘制 API，实现的功能如绘制缩放图像、阴影之间的插值、绘制方框周围的边框等。Gestures 库主要包含和手势识别相关的类，如单击、拖曳、长按、悬浮等。foundation 库中定义的功能是 Flutter 框架所有其他层使用的最低级别的实用程序类和函数。

Engine 为 Flutter 引擎，使用 C/C++编写，并提供了 Flutter 应用所需的原码。当需要绘制新一帧的内容时，引擎将负责对需要合成的场景进行栅格化。它提供了 Flutter 核心 API 的底层实现，包括图形(通过 Skia)、文本布局、文件及网络 IO、辅助功能支持、插件架构和 Dart 运行环境及编译环境的工具链。如 Platform Channels 平台通道，可以实现把 Flutter 嵌入一个已经存在的应用中，实现 Flutter 框架的 Dart 语言与 kotlin 和 Swift 的相互通信。

Embedder 指平台嵌入层，用于呈现所有 Flutter 内容的原生系统应用，它充当着宿主操作系统和 Flutter 之间的粘合剂的角色。当启动一个 Flutter 应用时，嵌入层会提供一个入口，初始化 Flutter 引擎，获取用户界面和栅格化线程，创建 Flutter 可以写入的纹理。嵌入层同时负责管理应用的生命周期，包括输入的操作(例如鼠标、键盘和触控)、窗口大小的变化、线程管理和平台消息的传递等。Flutter 拥有 Android、iOS、Windows、macOS 和 Linux 的平台嵌入层，当然，开发者可以创建自定义的嵌入层。

5.2 MaterialApp Flutter 材质应用

Flutter 界面设计主要以 Material Design 风格应用为主，Material 是一种移动端和网页端通用的视觉设计语言。Flutter 提供了丰富的具有 Material 风格的组件，通过它可以使用更多 Material 的特性，相应的网址为 https://material-io.cn/。

MaterialApp 包含材料设计应用程序通常需要的许多组件，如导航、路由、主页、样式、多语言设置、调试等。MaterialApp 的主要组成功能如下：

```
//第 5 章 5.1 MaterialApp 中的主要属性及用法
MaterialApp({
  this.navigatorKey,                                    //导航键,key 的作用是提高复用性能
  this.home,                                            //主页
  this.routes = const <String, WidgetBuilder>{},        //路由
  this.initialRoute,                                    //初始路由名称
  this.onGenerateRoute,                                 //路由构造
```

```
  this.onUnknownRoute,                                    //未知路由
  this.builder,                                           //建造者
  this.title = '',                                        //应用的标题
  this.onGenerateTitle,                                   //生成标题回调函数
  this.color,                                             //App 颜色
  this.theme,                                             //样式主题
  this.darkTheme,                                         //主机暗色模式
  this.themeMode = ThemeMode.system,                      //样式模式
  this.locale,                                            //本地化
  this.localizationsDelegates,                            //本地化代理
  this.supportedLocales = const <Locale>[Locale('en', 'US')],  //本地化处理
  this.deBugShowMaterialGrid = false,                     //是否调试显示材质网格
  this.showPerformanceOverlay = false,                    //是否显示性能叠加
  this.checkerboardRasterCacheImages = false,             //检查缓存图片的情况
  this.checkerboardOffscreenLayers = false,               //检查不必要的 setlayer
  this.showSemanticsDeBugger = false,                     //是否显示语义调试器
  this.deBugShowCheckedModeBanner = true,                 //是否在右上角显示 Debug 标记
});
```

5.3 Scaffold 脚手架

Scaffold 是一个页面布局脚手架，实现了基本的 Material 布局，继承自 StatefulWidget 有状态组件。像大部分的应用页面，包含标题栏、主体内容、底部导航菜单或者侧滑抽屉菜单等，每次都重复写这些内容会大大降低开发效率，所以 Flutter 框架提供了具有 Material 风格的 Scaffold 页面布局结构，可以很快地搭建出这些元素。与 Scaffold 相关的组件如下：

```
//第 5 章 5.2 Scaffold 主要属性及用法
const Scaffold({
   this.appBar,                                    //标题栏
   this.body,                                      //主体部分
   this.floatingActionButton,                      //悬浮按钮
   this.floatingActionButtonLocation,              //悬浮按钮位置
   this.floatingActionButtonAnimator,              //悬浮按钮动画
   this.persistentFooterButtons,                   //固定在下方显示的按钮
   this.drawer,                                    //左侧侧滑抽屉菜单
   this.endDrawer,                                 //右侧侧滑抽屉菜单
   this.bottomNavigationBar,                       //底部导航栏
   this.bottomSheet,                               //底部拉出菜单
   this.backgroundColor,                           //背景色
   this.resizeToAvoidBottomPadding,                //自动适应底部
   this.resizeToAvoidBottomInset,                  //重新计算 body 布局空间大小，避免被遮挡
   this.primary = true,            //是否显示在屏幕顶部，默认值为 true，将显示到顶部状态栏
   this. onDrawerChanged,          //侧滑抽屉菜单改变回调函数
  })
```

5.4　标题栏的显示

Scaffold 是 Material 库中提供的一个组件，它提供了默认的导航栏、标题和包含主屏幕组件树的 body 属性等特性，body 属性为导航栏下的区域，它是面积最大的区域。丰富的外观是由层层嵌套复杂的组件树组成的。

使用 Scaffold 在移动端的导航栏上显示标题，内容为 Home，代码如下：

```
//第 5 章 5.3 Flutter 应用中标题的显示
MaterialApp(                                                    //第 1 行
   home: Scaffold(                                              //第 2 行
      appBar: AppBar(                                           //第 3 行
         title: const Text('Home'),                             //第 4 行
      ),
   ),
   deBugShowCheckedModeBanner: false,                           //第 7 行
)
```

上述关键代码表示的含义如下：

第 1 行表示使用 MaterialApp 构造页面界面布局。

第 2 行表示在屏幕的主要区域使用了 Scaffold 脚手架实现布局。

第 3 行表示应用标题栏，使用了 Scaffold 组件的 appBar 属性。

第 4 行表示 appBar 属性通过 AppBar 组件的 title 属性，通过 Text 文本组件将内容显示为 Home 的标题。

第 7 行表示隐藏标题栏上的 Debug 调试图标。

显示的效果如图 5-2 所示。

图 5-2　Flutter 导航栏中显示标题

由于使用了 MaterialApp，导入的库为 import 'package:flutter/material.dart';，完整的 main.dart 源代码如下：

```
//第 5 章 5.4 Flutter 导航栏标题的显示
import 'package:flutter/material.dart';

void main() {
  runApp(Center(
    child: Chapter5Demo(),
  ));
}

class Chapter5Demo extends StatelessWidget {
  const Chapter5Demo({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text("Home"),
        ),
      ),
    );
  }
}
```

程序的运行设置类似于 3.2 节，可以在 VS Code 中输入，运行上面的代码以观看效果。

5.5 Container 容器组件

Container 容器组件，是用得比较多的组件，常用的属性有颜色、位置、长度、宽度等，Container 容器使用 child 属性包含其他的组件。

Container 常见的属性见表 5-1。

表 5-1 Container 常见属性及含义

width	设置容器的宽度
height	设置容器的高度
margin	容器与周围其他组件的距离
padding	容器内部组件与边界及其他组件的距离
alignment	容器内子节点组件的对齐方式
color	容器的颜色
child	包含其他子组件
decoration	对容器组件的样式进行修饰
transform	在绘制图形前对图形进行转换

在上面代码的基础上可以对 Scaffold 增加 body 属性，修改后的代码如下：

```
//第 5 章 5.5 Container 容器的使用
@override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text("Home"),
        ),                                                    //在此行下增加代码
        body: Container(                                      //增加的第 1 行
          margin: const EdgeInsets.all(20.0),                 //增加的第 2 行
          color: Colors.red,                                  //增加的第 3 行
          width: 96.0,                                        //增加的第 4 行
          height: 96.0,                                       //增加的第 5 行
        ),                                                    //增加的第 6 行
      ),
    );
  }
```

在上面的代码中，增加的第 1 行代码表示在 Scoffold 使用 body 属性添加容器属性。

增加的第 2 行代码表示容器的空白上下左右都为 20，这里的数字表示逻辑像素。

增加的第 3 行代码表示容器的颜色为红色。

增加的第 4 行代码表示容器的宽度为 96。

增加的第 5 行代码表示容器的高度为 96。

上述代码表示在屏幕的左上角显示一个容器，颜色为红色，宽度为 96，高度为 96，容器和四周的空白为 20，标题栏显示为 Home，效果如图 5-3 所示。

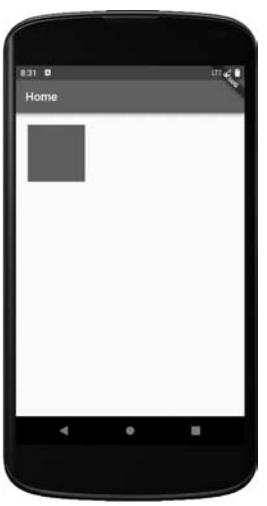

图 5-3　容器的显示

5.6 文本 Text 组件

文本 Text 组件主要用于显示文本内容。在容器中可以显示文本信息，只要加入一行语句就可以了。通过热加载(Hot Reload)技术自动在虚拟机的容器上显示文本内容，代码如下：

```
//第 5 章 5.6 Flutter 应用容器中 Text 文本的显示
body: Container(
        margin: const EdgeInsets.all(20.0),
        color: Colors.red,
        width: 96.0,
        height: 96.0,
        child: Text("你好"),                              //此处为增加的语句
      ),
```

增加的语句表示可以通过容器的 child 属性添加子组件，此处添加的子组件为 Text 文本组件。对文本组件中字体的大小、颜色、行高、间距等也可以设置调整，将在第 7 章 Flutter 样式中进行具体讲解。

5.7 图标 Icon 组件

Flutter 自带有大量的图标，可以以多种形式显示，可以从中选择我们需要的图标，以适合应用的要求。官方网址为 https://www.fluttericon.com。在容器中显示一个心形图标，颜色为品红色，大小为 72。可以修改前述代码中的 child 中的内容，代码如下：

```
//第 5 章 5.7 Flutter 应用容器中 Icon 图标的显示
body: Container(
        margin: const EdgeInsets.all(20.0),
        color: Colors.yellow,
        width: 200,
        height: 200,
        child: Icon(                                      //添加图标组件
          Icons.favorite,                                 //心形图标
          color: Colors.pink,                             //颜色为品红
          size: 72.0,                                     //心形图标大小为 72
        ),
      ),
```

运行效果如图 5-4 所示。

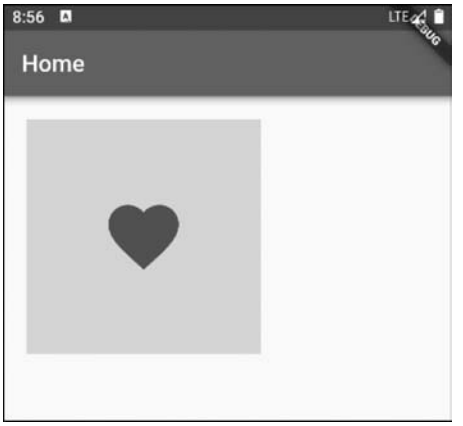

图 5-4　在容器中显示心形图标

5.8　图片 Image 组件

可以使用 Image 组件通过内存、文件、资源或网络等方式访问图片，实现图片的显示。Flutter 主要支持以下格式的图片，如 JPEG、PNG、GIF、BMP、Animated GIF 等。

Image 组件的主要属性如下：

```
//第 5 章 5.8 Image 图片组件中主要属性的用法
const Image({
   required ImageProvider < Object > image,        //要显示的图像
   ImageFrameBuilder? frameBuilder,                //创建图像生成器函数,例如可以设置动画效果
   ImageLoadingBuilder? loadingBuilder,            //正在加载时的效果
   ImageErrorWidgetBuilder? errorBuilder,          //加载图片错误时调用此函数
   String? semanticLabel,                          //对图像的描述
   double? width,                                  //图像宽度的设置
   double? height,                                 //图像高度的设置
   Color? color,                 //如果非空,则使用 colorBlendMod 将此颜色与每个图像像素混合
   Animation < double >? opacity, //如果非空,则在绘制到画布上之前,动画中的值将与每个图像
                                  //像素的不透明度相乘
   BlendMode? colorBlendMode,                              //颜色与此图像组合
   BoxFit? fit,                                            //图像在布局中的显示方式
   AlignmentGeometry alignment = Alignment.center,         //图像在内部的对齐方式
   ImageRepeat repeat = ImageRepeat.noRepeat,              //当空间多余时,是否重复显示图像
   Rect? centerSlice,                                      //图像适应目标的方式
   bool isAntiAlias = false,                               //是否使用抗锯齿绘制图像
   FilterQuality filterQuality = FilterQuality.low         //图像的渲染质量
})
```

5.8.1 网络图片的显示

可以在 Flutter 应用中显示网络图片。在移动端加载网络图片时，首先要配置工程的网络请求权限。在 VS Code 工程中打开 AndroidManifest.xml 文件(路径：android/app/src/main)，在 application 上面添加以下内容：

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
```

其次，定义变量_url，指明网络图片的地址(此为示例，由于网络图片地址可能会有变动，读者也可以自己在网络上找到一张图片地址替换书中的图片网络地址)。

最后，添加 Image.network(_url, fit：BoxFit.fill,)代码实现在 Flutter 程序中显示图片，其中 fit：BoxFit.fill 表示图片填充整个容器，代码如下：

```
//第 5 章 5.9 网络图片的显示
//指明图片地址
static const String _url = "http://image.biaobaiju.com/uploads/20180531/02/1527706049-
cVJjelLyiS.jpg";
...
body: Container(
        margin: const EdgeInsets.all(20.0),
        width: 200,
        height: 200,
child: Image.network(                                          //添加 Image 组件，以显示图像
         _url,
fit: BoxFit.fill,
loadingBuilder: (BuildContext context, Widget child,
              ImageChunkEvent? loadingProgress) {              //图像加载过程的处理
            if (loadingProgress == null) {                     //为空显示图像
              return child;
            }
            return Text('加载中…');                            //加载图像过程显示文本提示内容
          },
       ),
     ),
```

运行结果如图 5-5 所示。

loadingBuilder()函数表示在加载过程中，可以使用替代文本。由于网络上有的图像较大且网络速度慢等原因，加上文本提示后，可以让用户感觉到正在加载图像，这样用户的体验会更好。在 loadingBuilder()中，判断是否图像已加载完成，如果未加载完成，则显示文本信息'加载中…'，如果已完成，则显示图像。

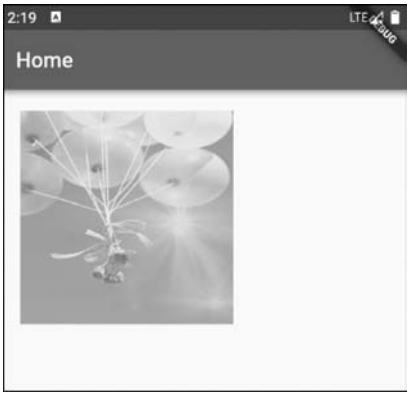

图 5-5　在容器中显示图像

Image 组件中的 fit 属性表示图像的显示方式，如是否拉伸、填充等。常见的值见表 5-2。

表 5-2　Image fit 属性的类型

fill	全图显示且填充满，图片可能会拉伸
contain	全图显示但不充满，显示原比例
cover	显示可能拉伸，也可能裁剪，充满
fitWidth	显示可能拉伸，也可能裁剪，宽度充满
fitHeight	显示可能拉伸，也可能裁剪，高度充满
scaleDown	在目标框内对齐图像（默认情况下，居中），如有必要，缩小图像的比例，以确保图像适合于框内

5.8.2　显示本地图片

本地图片的显示方式及步骤如下：

第 1 步，在 VS Code 资源管理器中单击新建文件夹图标，新建 asset/images 文件夹，并将图片拖动到这个文件夹中，如图 5-6 所示。

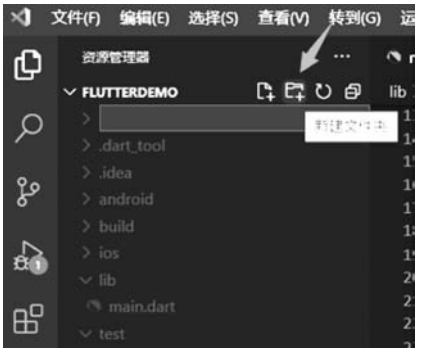

图 5-6　新建图片文件夹

第 2 步，如果是从网上下载的图片，则可能默认为锁定状态。要选中该文件，右击并选择文件属性解除锁定，如图 5-7 所示。

第 3 步，打开 pubspec.yaml 文件，加入如图 5-6 所输入的表示图片位置的内容（此处为 asset/images/）。一定要注意下面的格式，可以采用删除图中注释左边“#”和空格的方式生成下面的代码，如图 5-8 所示。

图 5-7　解除文件锁定

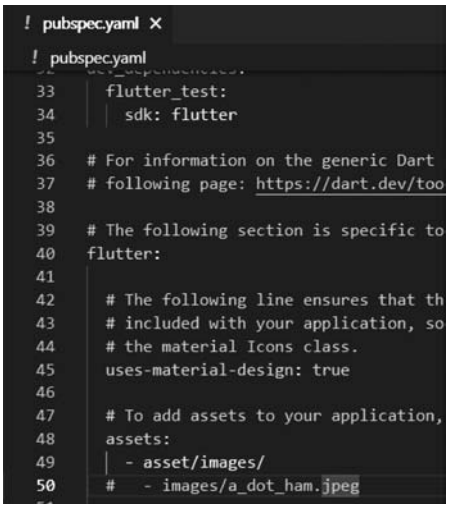

图 5-8　在 pubspec.yaml 文件中指明图片位置

第 4 步，在程序中加入如下语句，注意图片的名称和位置一定不要写错，与图 5-8 所示及实际的图片位置三者要保持一致。

```
//第 5 章 5.10 本地资源图片的显示
Container(
        margin: const EdgeInsets.all(20.0),
        color: Colors.yellow,
        width: 200,
        height: 200,
        child: Image.asset("asset/images/coast.jpg"),//此处为增加的语句
      ),
```

5.8.3　加载图片过程中，显示进度条信息

一般情况下，图片加载是直接显示在屏幕上，前面的过程屏幕可能长时间为空白，尤其在加载网络上的图片时，这样会给用户不好的体验。在加载的过程中，也可以先显示替代的图片或进度信息，这样会给用户带来很好的外观体验。如有动画的效果会有更酷的体验，代码如下：

```
//第 5 章 5.11 加载网络图片时,显示进度条
body: Container(
        margin: const EdgeInsets.all(20.0),
        color: Colors.red,
        width: 200,
        height: 200,
        child: FadeInImage.assetNetwork(
           placeholder: "asset/images/loading.gif",
           image:" https://tenfei04.cfp.cn/creative/vcg/veer/800water/veer-147317368.jpg"),
      ),
```

FadeInImage 类的作用为在加载目标图像时显示占位符 placeholder 属性中的图像,然后在加载完成后占位符中的图像在新图像中淡出。placeholder 属性为图片显示前的加载图片,一般为进度条图片,image 属性为显示的图片地址。进度条图片是本地资源,也要复制到工程相应的位置上。FadeInImage 常用属性见表 5-3。

表 5-3　FadeInImage 常用属性

alignment	在图像边界内对齐图像
fadeOutCurve	占位符淡出动画的随时间变化曲线
fadeInCurve	图像淡入动画的随时间变化曲线
fadeOutDuration	占位符加载过程中动画持续时间
fadeInDuration	图像加载过程中动画持续时间
fit	图像填充容器的方式,如填满、居中等
key	控制一个小组件如何替换组件树中的另一个组件

在图像的显示过程中,默认情况下,不同的手机分辨率会使用不同类型的图片。在设备像素比为 2.0 的屏幕上,屏幕将使用如 images/2x/cat.png 中所示的图片文件,相应地,在设备像素比为 4.0 的屏幕上,将使用 3.5x 文件夹中的图片 cat.png。在 pubspec.yaml 文件中与设备像素比有关的代码如下。我们可以在实际使用过程中,在相应的位置添加不同大小的图片,以适应不同的屏幕。

```
flutter:
  assets:
    - images/cat.png
    - images/2x/cat.png
    - images/3.5x/cat.png
```

5.9　Flutter 按钮类型

在 Flutter 中,按钮的类型很多,常用的有以下几种:

(1) 可以由 ButtonStyle 统一定义风格的按钮,如 TextButton、OutlinedButton、

ElevatedButton。它们在单击时,界面有类似水的波纹动态效果。

(2) 带图标的按钮,如 IconButton、BackButton、CloseButton。

(3) 切换按钮 ToggleButton。

(4) Material Design 中的浮动按钮,如 FloatingActionButton。

5.9.1 TextButton 文本按钮

TextButton 文本按钮的表现形式为普通按钮,并且没有边框,带边框的则为 OutlinedButton 按钮。可以在工具栏、对话框或与其他内容内联时,使用文本按钮。

下面的示例代码为 TextButton 的实现形式。onPressed 为单击事件,如果为 null,则此项禁用。其中,child 显示按钮文本内容。style 属性用于定义按钮的风格,通过 TextButton 的静态方法 styleFrom 复制了原来的风格,并实现了字体的大小为 20。

```
//第 5 章 5.12 TextButton 文本按钮的使用
TextButton(
          style: TextButton.styleFrom(
            textStyle: const TextStyle(fontSize: 20),
          ),
          onPressed: () {},
          child: const Text('文本按钮'),
        ),
```

5.9.2 OutlinedButton 强调按钮

OutlinedButton 和 TextButton 类似,区别为 OutlinedButton 按钮是以边框的形式存在,代码如下:

```
//第 5 章 5.13 OutlinedButton 强调按钮的使用
OutlinedButton(
     onPressed: () {
       print('Received click');
     },
     child: const Text('有边框的按钮'),
   );
```

5.9.3 ElevatedButton 有阴影的按钮

ElevatedButton 有阴影的按钮的表现形式更为丰富,可以在长列表或其他空间位置使用,但不要在对话框或卡片本身有阴影效果的组件上使用。ElevatedButton 在显示时按钮的边界有阴影立体效果,按下时的动态效果更为明显,代码如下:

```
//第 5 章 5.14 ElevatedButton 带阴影的按钮的使用
ElevatedButton (
          style: ElevatedButton.styleFrom(
            textStyle: const TextStyle(fontSize: 20),
          ),
          onPressed: () {},
          child: const Text('点我'),
        ),
```

TextButton 文本按钮、OutlinedButton 强调按钮和 ElevatedButton 有阴影的按钮的外观表现形式如图 5-9 所示。

图 5-9　ElevatedButton、TextButton、OutlinedButton 按钮的显示

5.9.4　IconButton 图标按钮

图标按钮，顾名思义，带图标的按钮，常见的有后退按钮 BackButton、关闭按钮 CloseButton 等。可以放在任何位置，通常应用于屏幕最上面的导航栏 AppBar.actions 位置，用于单击完成一些事件。图 5-10 为星形图标按钮当单击时的实现效果，默认显示星形图标，当单击时有类似波浪向外扩散的阴影效果。

图 5-10　单击图标按钮的水波效果

icon 表示按钮图标的类型，color 表示按钮的颜色，iconSize 表示图标的大小，splashColor 表示按下这个按钮时，背景显示的动态颜色。onPressed 实现单击事件，代码如下：

```
//第 5 章 5.15IconButton 带图标的按钮的使用
IconButton(
          icon: const Icon(Icons.android),     //按钮图标的类型
          color: Colors.red,                   //按钮的颜色为红色
```

```
        iconSize: 96,                    //图标的大小为 96
        splashColor: Colors.blue,        //按下这个按钮时,背景显示的动态颜色为蓝色
        onPressed: () {},                //实现单击事件
      ),
```

5.9.5 FloatingActionButton 浮动按钮

FloatingActionButton 浮动操作按钮组件通常是一个圆形图标按钮,它悬停在屏幕内容之上,以执行应用程序中的相应操作。在 Scaffold 中使用 floatingActionButton 属性实现浮动操作按钮的显示效果,使用 floatingActionButtonLocation 属性设置按钮的位置,使用 floatingActionButtonAnimator 属性实现按钮位置变化的动画效果。浮动操作按钮组件在屏幕的显示效果如图 5-11 箭头所示。

图 5-11 FloatingActionButton 浮动按钮的显示

实现 FloatingActionButton 浮动按钮显示的关键代码如下:

```
//第 5 章 5.16 FloatingActionButton 浮动按钮的使用
@override
  Widget build(BuildContext context) {
    return Scaffold(                                    //定义 Scaffold
      appBar: AppBar(
        title: Text('FloatingActonButton 示例'),
      ),
      //floatingActionButtonAnimator: ,                  //实现浮动按钮的动画效果
      //(1)
      floatingActionButton: FloatingActionButton(
```

```
        onPressed: () {                                    //定义单击事件
         //单击浮动按钮,以执行一些事件
         },
        backgroundColor: Colors.blue,                      //定义背景颜色
        child: const Icon(Icons.add),                      //定义要显示的内容
      ),
      //(2)
      floatingActionButtonLocation: FloatingActionButtonLocation.endFloat,
      body: Center(
        child: FlutterLogo(),
      ),
    );
  }
```

上述关键标记代码表示的含义分别如下：

(1) 通过 FloatingActionButton 属性实现显示浮动按钮的功能。

(2) 通过 FloatingActionButtonLocation 类设置浮动按钮的位置，默认的常量值为 endFloat，表示位置在右下角的位置。其他常量可以控制浮动按钮在其他相应的位置，如 centerDocked 表示浮动按钮在底部选项卡的中间位置，startTop 表示在顶部左边的位置。