

# Flutter 功能类 Widget

功能类Widget 指的是 非UI Widget，具有一定功能的 Widget。Flutter 中功能类Widget 有很多，本节主要讲如下的功能类 Widget：

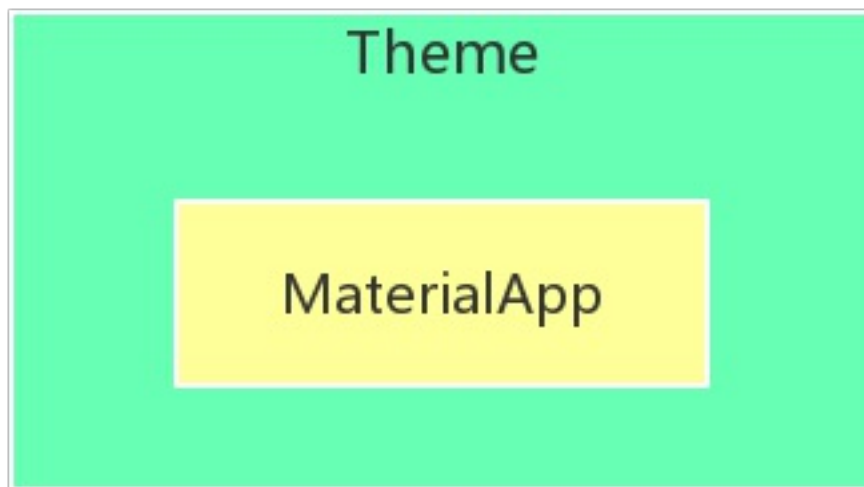
- Theme
- MediaQuery
- Navigator
- InheritedWidget

## Theme

Theme Widget 的功能是为 MaterialApp 定义主题数据，如导航栏颜色、标题字体、Icon样式。

## Theme 在 MaterialApp 中的位置

在看代码的时候，你可能没有看到 Theme 的使用，实际上是在使用 MaterialApp 时，会自动将 Theme Widget 加上去：



Theme 会嵌套 MaterialApp。

## 在 Flutter 中获取 Theme 的实例

要想在 Flutter 中获取 Theme 的实例，首先 根Widget 必须是 MaterialApp，然后在 MaterialApp 的子Widget 里运行：

```
Theme.of(context);
```

返回的类型是 ThemeData，Theme 的所有数据都存储在 ThemeData 里。

## ThemeData 说明

ThemeData 的构造函数为：

```
factory ThemeData({  
    Brightness brightness,  
    MaterialColor primarySwatch,
```

```
Color primaryColor,  
Brightness primaryColorBrightness,  
Color primaryColorLight,  
Color primaryColorDark,  
Color accentColor,  
Brightness accentColorBrightness,  
Color canvasColor,  
Color scaffoldBackgroundColor,  
Color bottomAppBarColor,  
Color cardColor,  
Color dividerColor,  
Color highlightColor,  
Color splashColor,  
InteractiveInkFeatureFactory splashFactory,  
Color selectedRowColor,  
Color unselectedWidgetColor,  
Color disabledColor,  
Color buttonColor,  
ButtonThemeData buttonTheme,  
Color secondaryHeaderColor,  
Color textSelectionColor,  
Color cursorColor,  
Color textSelectionHandleColor,  
Color backgroundColor,  
Color dialogBackgroundColor,  
Color indicatorColor,  
Color hintColor,  
Color errorColor,  
Color toggleableActiveColor,  
String fontFamily,  
TextTheme textTheme,  
TextTheme primaryTextTheme,  
TextTheme accentTextTheme,
```

```
InputDecorationTheme inputDecorationTheme,  
IconThemeData iconTheme,  
IconThemeData primaryIconTheme,  
IconThemeData accentIconTheme,  
SliderThemeData sliderTheme,  
TabBarTheme tabBarTheme,  
CardTheme cardTheme,  
ChipThemeData chipTheme,  
TargetPlatform platform,  
MaterialTapTargetSize materialTapTargetSize,  
PageTransitionsTheme pageTransitionsTheme,  
AppBarTheme appBarTheme,  
BottomAppBarTheme bottomAppBarTheme,  
ColorScheme colorScheme,  
DialogTheme dialogTheme,  
Typography typography,  
CupertinoThemeData cupertinoOverrideTheme  
}) {  
  ...  
}
```

## ThemeData 的使用

ThemeData 的数据是通过 MaterialApp 的 theme 参数来使用的，如：

```
MaterialApp(  
  title: "Flutter Demo",  
  theme: ThemeData(  
    primaryColor: Colors.blue,  
  ),  
  ...  
)
```

## Theme.of(context) 的使用

Theme.of(context) 会获得 MaterialApp 的主题数据 ThemeData，这些数据都是 final 的，所以是只读的，不能修改。

这里写一个例子，读取 MaterialApp 的 primaryColor 用于 Text 的颜色。

### 代码所在位置

flutter\_widget\_demo/lib/features/ThemeFeatureWidget.dart

### 完整代码

```
import 'package:flutter/material.dart';

void main() => runApp(ThemeFeatureWidget());

class ThemeFeatureWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: "Flutter Demo",
      theme: ThemeData(
        primaryColor: Colors.red,
      ),
      home: Scaffold(
        appBar: AppBar(title: Text("Flutter 功能类Widget -- Theme")),
        body: ChildText(),
      );
  }
}

class ChildText extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    // TODO: implement build
    return Text("Hello Flutter",
      style: TextStyle(color: Theme.of(context).primaryColor));
  }
}
```

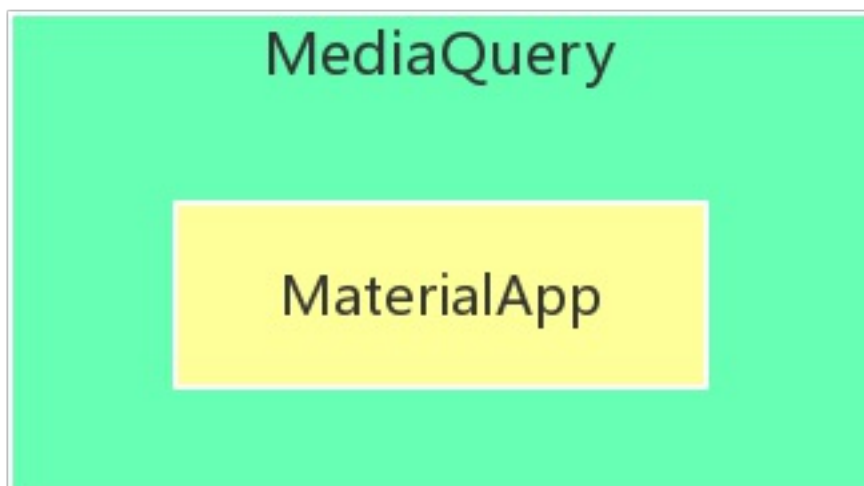
运行效果为：

# MediaQuery

MediaQuery Widget 的功能是查询一些数据，例如整个屏幕的宽度、高度、设备像素比等数据

## MediaQuery 在 MaterialApp 中的位置

MediaQuery 同 Theme 一样，只看代码，你可能没有看到 MediaQuery 的使用，但实际上在使用 MaterialApp 时，会自动将 MediaQuery Widget 加上去：



MediaQuery 会嵌套 MaterialApp。

## 在 Flutter 中获取 MediaQuery 的实例

要想在 Flutter 中获取 MediaQuery 的实例，首先 根Widget 必须是 MaterialApp，然后在 MaterialApp 的子Widget 里运行：

```
MediaQuery.of(context);
```

返回的类型是 MediaQueryData。

## MediaQueryData 说明

MediaQueryData 的构造函数为：

```
class MediaQueryData {  
  const MediaQueryData({  
    this.size = Size.zero,  
    this.devicePixelRatio = 1.0,  
    this.textScaleFactor = 1.0,  
    this.platformBrightness = Brightness.light,  
    this.padding = EdgeInsets.zero,  
    this.viewInsets = EdgeInsets.zero,  
    this.alwaysUse24HourFormat = false,  
    this.accessibleNavigation = false,  
    this.invertColors = false,  
    this.disableAnimations = false,  
    this.boldText = false,  
  });  
  ...  
}
```

参数名字	参数类型	意义
------	------	----

--	--	--	--
----	----	----	----

size	Size	屏幕的逻辑像素大小
------	------	-----------

size.width		屏幕的宽度（逻辑像素）
------------	--	-------------

size.height		屏幕的高度（逻辑像素）
-------------	--	-------------

devicePixelRatio	double	像素比：每个逻辑像素代表的设备像素数
------------------	--------	--------------------

textScaleFactor	double	每个逻辑像素代表的字体像素数
-----------------	--------	----------------



| platformBrightness | Brightness | 平台的亮度模式,有两种模式:light和dark  
默认为light |  
padding	EdgeInsets	表示系统状态栏或者刘海屏的padding
viewInsets	EdgeInsets	表示设备键盘的padding
alwaysUse24HourFormat	bool	格式化时间时是否使用24小时格式
accessibleNavigation	bool	是否使用TalkBack或 VoiceOver等辅助功能服务与应用程序进行交互
invertColors	bool	设备是否反转平台的颜色。
现在只能在 iOS 上使用		
disableAnimations	bool	平台是否要求禁用动画
boldText	bool	平台是否请求使用粗体来绘制文本。

## MediaQuery.of(context) 的使用

MediaQuery.of(context) 获得的 MediaQueryData 数据里, 我们一般关心的是:

- Size
- devicePixelRatio

### Size

Size 是整个屏幕的宽高数据, 不管你在哪里调用 MediaQuery.of(context), 获得的都是整个屏幕的 Size。

### devicePixelRatio

devicePixelRatio 表示的是 每个逻辑像素代表的设备像素数。

设备像素数 = 逻辑像素数 \* devicePixelRatio

## 获取 子Widget 的宽高

要想获取 子Widget 的宽高，就不能使用 `MediaQuery.of(context)`，这个时候要用到 `GlobalKey`。

为 子Widget 设置 `GlobalKey`，然后通过 `GlobalKey` 获取 子Widget 的宽高。

这里写一个例子，获取屏幕的宽高和子Widget的宽高。

### 代码所在位置

`flutter_widget_demo/lib/features/MediaQueryFeatureWidget.c`

### 完整代码

```
import 'package:flutter/material.dart';

void main() => runApp(MediaQueryFeatureWidget());

GlobalKey _globalKey = GlobalKey();

class MediaQueryFeatureWidget extends
StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: "Flutter Demo",
      theme: ThemeData(
        primaryColor: Colors.blue,
      ),
      home: Scaffold(
        appBar: AppBar(title: Text("Flutter 功能
```

```
类Widget -- MediaQuery"))),  
    body: BodyWidget()),  
    );  
}  
}
```

```
class BodyWidget extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Column(  
      children: <Widget>[  
        Text(  
          "Hello Flutter",  
          key: _globalKey,  
        ),  
        RaisedButton(  
          child: Text('getSize'),  
          onPressed: () {  
            //获取屏幕的宽高  
            print("Screen width:" +  
MediaQuery.of(context).size.width.toString() +  
              " Screen height:" +  
MediaQuery.of(context).size.height.toString());  
            //获取子Widget 的宽高  
            print("Ttext width:" +  
_globalKey.currentContext.size.width.toString() +  
              " Screen height:" +  
_globalKey.currentContext.size.height.toString())  
            ;  
          }  
        ),  
      ],  
    );  
  }  
}
```

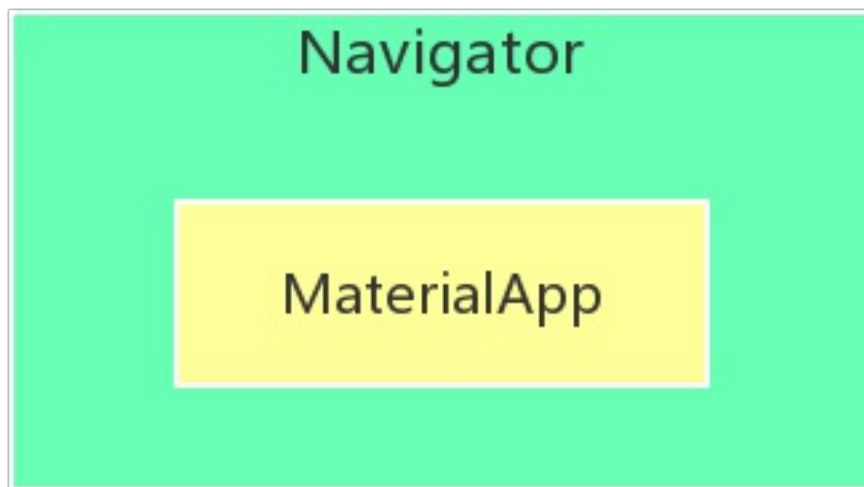
```
}  
  }  
  );  
  ],  
  )  
  },  
}
```

# Navigator

Navigator 是用来管理页面显示的 Widget，这些页面以堆栈的数据结构存储，当有新页面显示时，就会把上一个页面压入栈，所以栈底是最初的页面，栈顶是当前的页面。

## Navigator 在 MaterialApp 中的位置

Navigator 同 Theme 一样，只看代码，你可能没有看到 Navigator 的使用，但实际上在使用 MaterialApp 时，会自动将 Navigator Widget 加上去：



`Navigator` 会嵌套 `MaterialApp`。

## Routes（路由）

移动应用将全屏幕显示的叫做“screens”或者“pages”，在 Flutter 里叫做 `Routes`（路由），`Navigator` 就是用来管理 `Routes` 显示的 `Widget`。

## Navigator 的说明

`Navigator` 提供了操作 `Routes` 的两种方法：`Navigator.push` 和 `Navigator.pop`。

## 在 Flutter 中获取 Navigator 的实例

在 Flutter 中除了 `MaterialApp` 提供的 `Navigator`，也可以实现自定义的 `Navigator`，但是大部分情况下还是使用的 `MaterialApp` 提供的 `Navigator`。

首先 根Widget 必须是 MaterialApp，然后在 MaterialApp 的 子Widget 里运行：

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

返回的类型是 NavigatorState，是 Navigator Widget 的 State 类。

## NavigatorState 说明

以下是 NavigatorState 类的图：

以下是 Navigator 类的图：

可以看到 Navigator 类里的方法和 NavigatorState 类的方法都一样，所以 也可以直接使用 Navigator ，而不是 Navigator.of(context)。

这里 Navigator 就简单介绍完了， 具体使用会在后面介绍。

# InheritedWidget

InheritedWidget 可以高效的将数据在 Widget树 中向下传递，通常用来共享数据，Flutter中非常重要的一个功能 Widget。

前面介绍的 Theme 和 MediaQuery 之所以可以在 子Widget 内访问到数据，就是使用到了 InheritedWidget。

大家应该都有这样的经历，从 A 页面跳转到 B 页面，但 B 页面需要用到 A 页面的一些数据，这时候该怎么做呢？

通常做法是，A 页面打开 B 页面的时候，把数据从 A 页面传递给 B 页面，但这样做，会使 A 页面的数据和 B 页面的数据产生割裂，如果又要求 B 页面产生的结果又返回给 A 页面呢？这使得我们要用很大的精力去处理数据传输的问题。

InheritedWidget 的目的就是处理数据问题，可以将共享的数据在 InheritedWidget 里实现，这样 A 页面和 B 页面都可以直接访问数据并修改，数据一旦修改，就会触发依赖这个数据的 UI 的刷新。InheritedWidget 在 Flutter 开发中有很大的作用。

这里 InheritedWidget 就简单介绍完了，具体使用会在后面介绍。