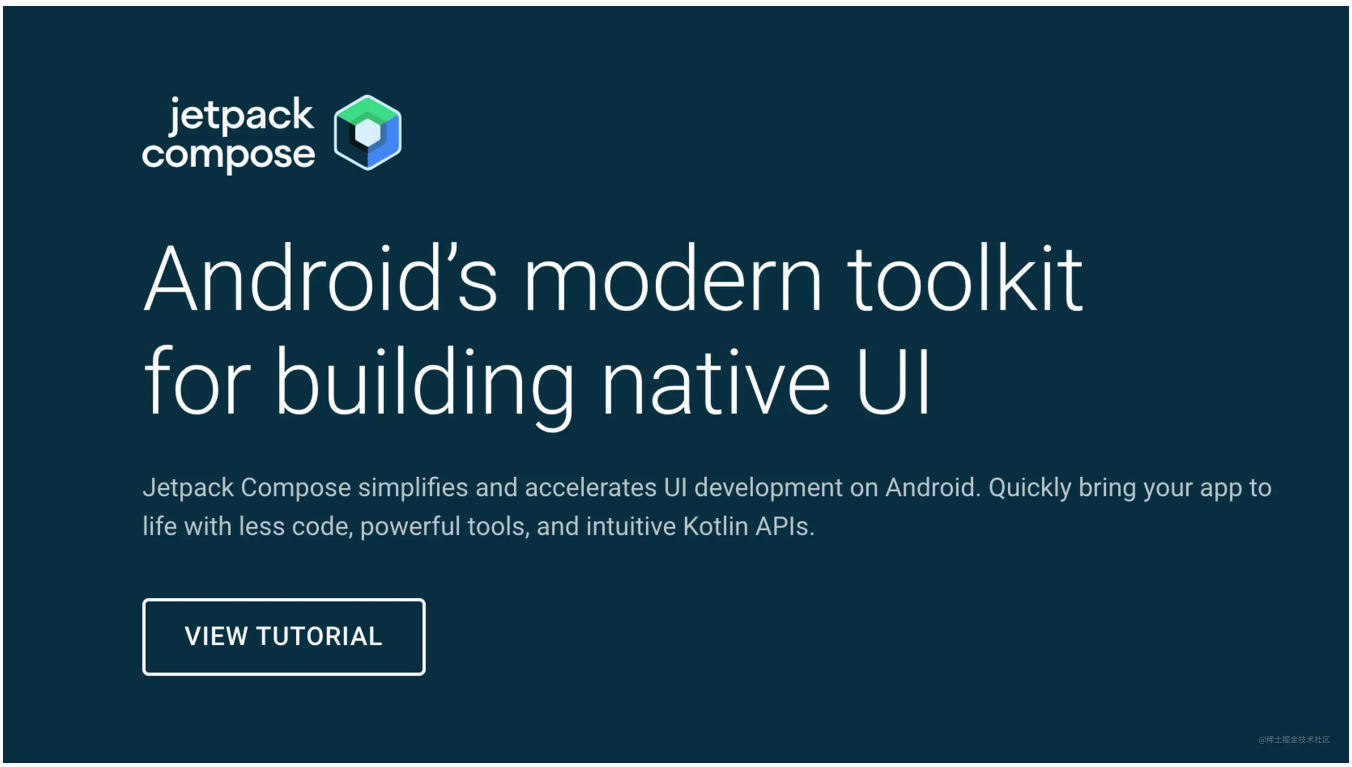
Android全新UI编程 - Jetpack Compose 超详细教程 第1弹

HyejeanMOON LV.4
2020年07月11日 16:54 · 阅读 22359

关注



1. 简介

`Jetpack Compose` 是在2019Google i/O大会上发布的新的库。 `Compose` 库是用响应式编程的方式对View进行构建,可以用更少更直观的代码，更强大的功能，能提高开发速度（这一段是谷歌自己说的）。 说实话， `View/Layout` 的模式对安卓工程师来说太过于熟悉，对于学习曲线陡峭的 `Jetpack Compose` 能不能很好的普及还是有所担心。

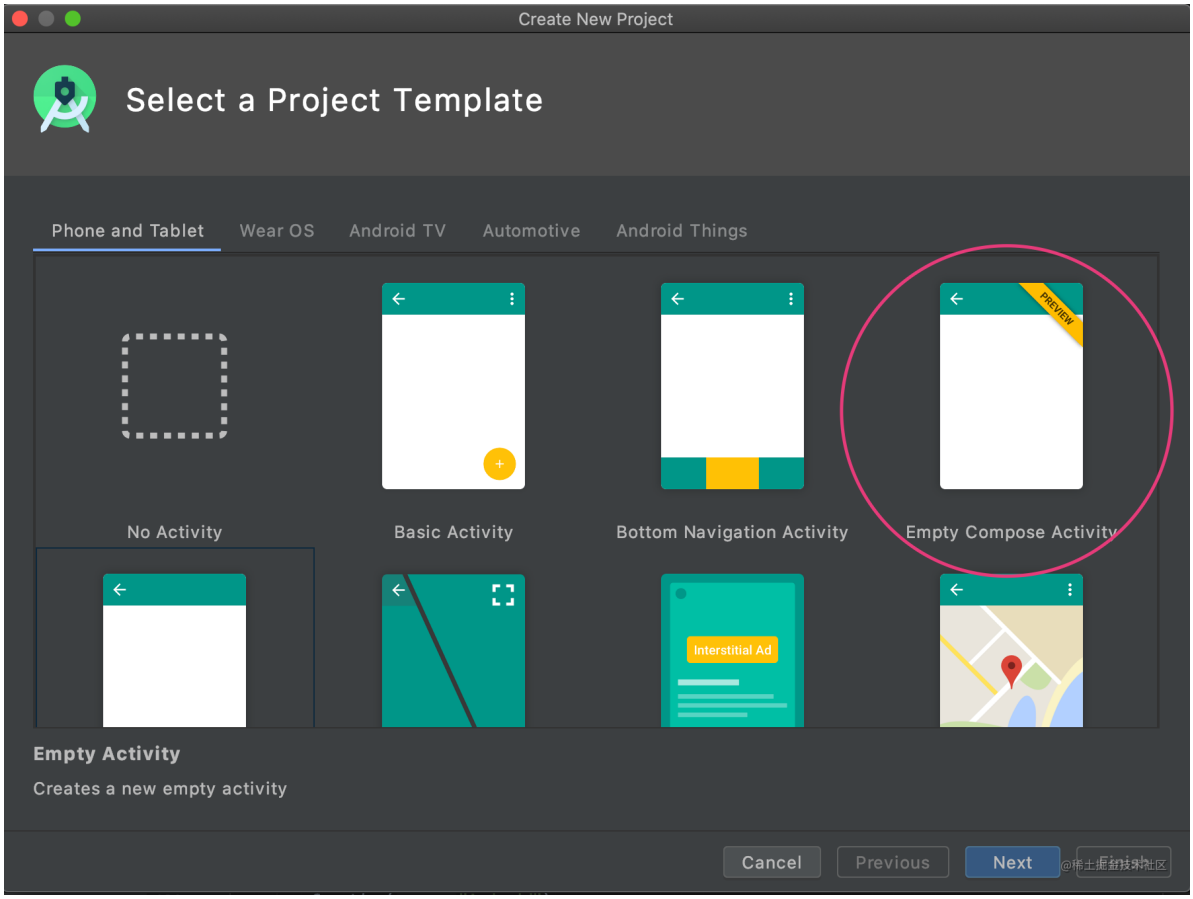
因为 `Jetpack Compose` 的内容比较多，我会分成多个文章来进行介绍。 内容包括常用UI组件的使用， `Flow` 和 `Compose` 的结合使用，以及如何构建MVVM应用。 还有， `Compose` 的API还没有完全的确定下来，如果有API的修改，我也会对文章进行修订，所以敬请放心。

第1弹将会介绍如何创建 `Compose` 应用以及基本注解， `Compose` 方法的使用。 好了，闲话不多说，开整!

2. 教程

2.1 创建新的项目或导入库

Jetpack Compose 是从 Android Studio 4.2 开始支持的，所以需要通过4.2(现在是canary版本)创建新的项目或者添加导入库。这里按照创建新的项目来进行介绍。



根据上图所示，在创建新的项目时需要选择 Empty Compose Activity 。

此时模块中的 build.gradle 文件会新增下列的库的依赖。

rust 复制代码

```
dependencies {
    ...
    implementation 'androidx.ui:ui-layout:"${compose_version}"'
    implementation 'androidx.ui:ui-material:"${compose_version}"'
    implementation 'androidx.ui:ui-tooling:"${compose_version}"'
    ...
}
```

还有在模块的 build.gradle 文件中新增下列的设置。

复制代码

```
android {
    ...
```

```
    buildFeatures {
        compose true
    }

    composeOptions {
        kotlinCompilerExtensionVersion "${compose_version}"

        kotlinCompilerVersion "1.3.70-dev-withExperimentalGoogleExtensions-20200424"
    }
}
```

2.2 UI相关

2.2.1 @Compose

所有关于构建View的方法都必须添加 `@Compose` 的注解才可以。并且 `@Compose` 跟协程的 `Suspend` 的使用方法比较类似,被 `@Compose` 的注解的方法只能在同样被 `@Comopse` 注解的方法中才能被调用。

kotlin 复制代码

```kotlin
@Composable
fun Greeting(name: String) {
    Text(text = "Hello $name!")
}
```

2.2.2 @Preview

加上 `@Preview` 注解的方法可以在不运行App的情况下就可以确认布局的情况。

`@Preview` 的注解中比较常用的参数如下:

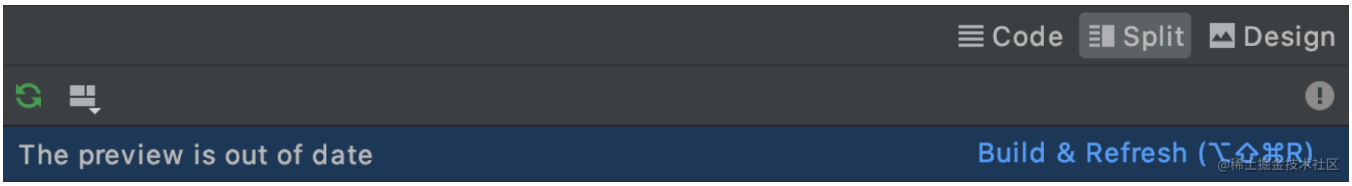
1. `name: String` : 为该Preview命名，该名字会在布局预览中显示。
2. `showBackground: Boolean` : 是否显示背景，true为显示。
3. `backgroundColor: Long` : 设置背景的颜色。
4. `showDecoration: Boolean` : 是否显示Statusbar和Toolbar，true为显示。
5. `group: String` : 为该Preview设置group名字，可以在UI中以group为单位显示。
6. `fontScale: Float` : 可以在预览中对字体放大，范围是从0.01。
7. `widthDp: Int` : 在Compose中渲染的最大宽度，单位为dp。
8. `heightDp: Int` : 在Compose中渲染的最大高度，单位为dp。

上面的参数都是可选参数，还有像背景设置等的参数并**不是**对实际的App进行设置，只是对Preview中的背景进行设置，为了更容易看清布局。

less 复制代码

```
@Preview(showBackground = true, name = "Home UI", showDecoration = true)
@Composable
fun DefaultPreview() {
    MyApplicationTheme {
        Greeting("Android")
    }
}
```

在IDE的右上角有 `Code` ， `Split` , `Design` 三个选项。分别是只显示代码，同时显示代码和布局和只显示布局。
当更改跟UI相关的代码时，会显示如下图的一个横条通知，点击 `Build&Refresh` 即可更新显示所更改代码的UI。



2.2.3 setContent

`setContent` 的作用是和zai `Layout/View` 中的 `setContentView` 是一样的。
`setContent` 的方法也是有 `@Compose` 注解的方法。所以，在 `setContent` 中写入关于UI的 `@Compopse` 方法，即可在Activity中显示。

kotlin 复制代码

```
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            JetpackComposeDemoTheme {
                Greeting("Android")
            }
        }
    }
```

2.2.4 *Theme

在创建新的Compose项目时会自动创建一个 `项目名+Theme` 的 `@Compose` 方法。 我们可以通过更改颜色来完成对主题颜色的设置。 生成的Theme方法的代码如下。

ini 复制代码

```
private val DarkColorPalette = darkColorPalette(
        primary = purple200,
        primaryVariant = purple700,
        secondary = teal200
)

private val LightColorPalette = lightColorPalette(
        primary = purple500,
        primaryVariant = purple700,
        secondary = teal200

        /* Other default colors to override
    background = Color.White,
    surface = Color.White,
    onPrimary = Color.White,
    onSecondary = Color.Black,
    onBackground = Color.Black,
    onSurface = Color.Black,
    */
)

@Composable
fun JetpackComposeDemoTheme(darkTheme: Boolean = isSystemInDarkTheme(), content: @Composable
    val colors = if (darkTheme) {
        DarkColorPalette
    } else {
        LightColorPalette
    }

    MaterialTheme(
            colors = colors,
            typography = typography,
            shapes = shapes,
            content = content
    )
}
```

Theme方法中有正常主题和Dark主题的颜色设置，里面还有关于 `MeterialTheme` 的设置。

关于Theme方法的用法如下。

scss 复制代码

```
        setContent {
            JetpackComposeDemoTheme {
                Greeting("Android")
            }
        }
```

在 JetpackComposeDemoTheme 里面的所有UI方法都会应用上述主题中指定的颜色。

2.2.4 Modifier

Modifier 是各个 Compose 的UI组件一定会用到的一个类。它是被用于设置UI的摆放位置，padding等信息的类。关于 Modifier 相关的设置实在是太多，在这里只介绍会经常用到的。

- padding 设置各个UI的padding。padding的重载的方法一共有四个。

```scss
Modifier.padding(10.dp) // 给上下左右设置成同一个值
Modifier.padding(10.dp, 11.dp, 12.dp, 13.dp) // 分别为上下左右设值
Modifier.padding(10.dp, 11.dp) // 分别为上下和左右设值
Modifier.padding(InnerPadding(10.dp, 11.dp, 12.dp, 13.dp))// 分别为上下左右设值
```

这里设置的值必须为 Dp ， Compose 为我们在Int中扩展了一个方法 dp ， 帮我们转换成 Dp 。

- plus 可以把其他的Modifier加入到当前的Modifier中。

```scss
Modifier.plus(otherModifier) // 把otherModifier的信息加入到现有的modifier中
```

- fillMaxHeight , fillMaxWidth , fillMaxSize 类似于 match_parent ,填充整个父layout。

```scss
Modifier.fillMaxHeight() // 填充整个高度
```

- width , heigh , size 设置Content的宽度和高度。

```scss
Modifier.width(2.dp) // 设置宽度
Modifier.height(3.dp)  // 设置高度
Modifier.size(4.dp, 5.dp) // 设置高度和宽度
```

- widthIn , heightIn , sizeIn 设置Content的宽度和高度的最大值和最小值。

```scss
Modifier.widthIn(2.dp) // 设置最大宽度
Modifier.heightIn(3.dp) // 设置最大高度
Modifier.sizeIn(4.dp, 5.dp, 6.dp, 7.dp) // 设置最大最小的宽度和高度
```

- gravity 在 Column 中元素的位置。

```scss
```

```
Modifier.gravity(Alignment.CenterHorizontally) // 横向居中
Modifier.gravity(Alignment.Start) // 横向居左
Modifier.gravity(Alignment.End) // 横向居右
```

- `rtl`，`ltr` 开始布局UI的方向。

arduino 复制代码

```
Modifier.rtl  // 从右到左
Modifier.ltr  // 从左到右
```

Modifier的方法都返回Modifier的实例的链式调用，所以只要连续调用想要使用的方法即可。

kotlin 复制代码

```
@Composable
fun Greeting(name: String) {
    Text(text = "Hello $name!", modifier = Modifier.padding(20.dp).fillMaxSize())
}
```

2.2.5 Column，Row

正如其名字一样， `Column` 和 `Row` 可以理解为在 `View/Layout` 体系中的纵向和横向的 `ViewGroup` 。
需要传入的参数一共有四个。

- `Modifier` 用上述的方法传入已经按需求设置好的Modifier即可。
- `Arrangement.Horizontal`，`Arrangement.Vertical` 需要给 `Row` 传入 `Arrangement.Horizontal`，为 `Column` 传入 `Arrangement.Vertical` 。 这些值决定如何布置内部UI组件。
  可传入的值为 `Center`，`Start`，`End`，`SpaceEvenly`，`SpaceBetween`，`SpaceAround` 。
  重点解释一下 `SpaceEvenly`，`SpaceBetween`，`SpaceAround` 。

`SpaceEvenly`： 各个元素间的空隙为等比例。

`SpaceBetween`： 第一元素前和最后一个元素之后没有空隙，所有空隙都按等比例放入各个元素之间。

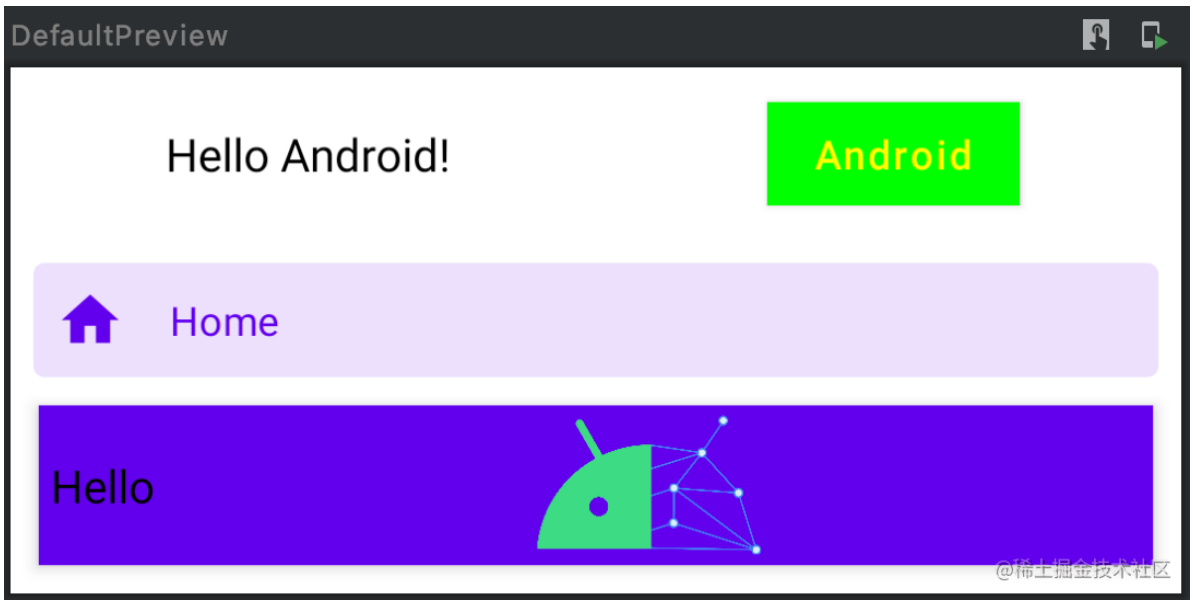
`SpaceAround`： 把整体中一半的空隙平分的放入第一元素前和最后一个元素之后，剩余的一半等比例的放入各个元素之间。

- `Alignment.Vertical`，`Alignment.Horizontal` 需要给 `Row` 传入 `Alignment.Vertical`，为 `Column` 传入 `Alignment.Horizontal`。 使用方法和 `Modifier` 的 `gravity` 中传入参数的用法是一样的，这里就略过了。

- `@Composable ColumnScope.() -> Unit` 需要传入标有 `@Compose` 的UI方法。但是这里我们会有lamda函数的写法来实现。

整体代码如下。

```scss 复制代码
Column {
        Row(modifier = Modifier.ltr.fillMaxWidth(),horizontalArrangement = Arrangement.Space
       // ..,...
    }
```



3. 其他

写的有点多，关于Android全新UI编程的第一篇就到这里。我会抓紧写下一篇的。

Jetpack Compose 官网

JetpackComposeDemo Github

Android全新UI编程 - Jetpack Compose 超详细教程：
第1弹

其他教程：
神一样的存在，Dagger Hilt !!
Android10的分区存储机制(Scoped Storage)适配教程
Android Jetpack Room的详细教程
Android的属性动画(Property Animation)详细教程
Android ConstraintLayout的易懂教程
Google的MergeAdapter的使用
Paging在Android中的应用
Android UI测试之Espresso
Android WorkManager的使用

本文使用 mdnice 排版

分类： Android 标签： Android