# **混合开发应用场景**

目前在国内开发纯Flutter的应用还是比较少的，绝大部分使用Flutter来开发的公司也都是使用混合开发。那么混合开发的主要使用场景有哪些呢？

* 作为独立的页面进行加入，可以从原生页面跳转到Flutter页面，也可以从Flutter页面跳转到原生页面。



* 作为页面的一部分嵌入。



# **创建Flutter module**

既然是做混合开发，那么我们肯定是由Android原生项目的。假如native项目的路径是这样的：flutter/flutter\_hybrid/native,那么我们需要在native上一层目录flutter\_hybrid中创建Flutter module。

cd flutter/flutter\_hybrid/*//创建支持AndroidX的flutter\_module*

flutter **create** --androidx -t **module** flutter\_module *//创建不支持AndroidX的flutter\_module*

flutter **create** -t **module** flutter\_module

所以我们在创建模块的时候首先要确定native项目是不是已经支持AndroidX，如果支持就需要加上 --androidx 参数。

输入后控制台打印如下：

$ flutter create -t module flutter\_module

Creating project flutter\_module...

 flutter\_module/test/widget\_test.dart (created)

 flutter\_module/flutter\_module.iml (created)

 flutter\_module/.gitignore (created)

 flutter\_module/.metadata (created)

 flutter\_module/pubspec.yaml (created)

 flutter\_module/README.md (created)

 flutter\_module/lib/main.dart (created)

 flutter\_module/flutter\_module\_android.iml (created)

 flutter\_module/.idea/libraries/Flutter\_for\_Android.xml (created)

 flutter\_module/.idea/libraries/Dart\_SDK.xml (created)

 flutter\_module/.idea/modules.xml (created)

 flutter\_module/.idea/workspace.xml (created)

Running "flutter pub get" **in** flutter\_module... 1.2s

Wrote 12 files.

All done!

Your module **code** is **in** flutter\_module/lib/main.dart.

看到All done就表示我们项目创建好了。整个module目录和原生Flutter基本一样，主要就是Android、iOS的宿主工程和lib目录以及pubspec.yaml文件。

# **添加Flutter module依赖**

module项目创建好后就需要添加到Android项目中了。我们打开Android项目的setting.gradle文件，添加如下代码：

setBinding(new Binding([gradle: this]))

evaluate(new File(

 settingsDir.parentFile,

 //flutter\_module即为创建的模块名称

 'flutter\_module/.android/include\_flutter.groovy'

))

setBinding与evaluate允许Flutter模块包括它自己在内的任何Flutter插件，在settings.gradle中以类似 :flutter、package\_info、:video\_player的方式存在。

然后打开app/build.gradle在dependencies标签中添加依赖：

**implementation** project(':flutter')

这样两步就完成了依赖的添加，这里为什么添加的叫“flutter” 而不是 “flutter\_module”呢？因为项目编译完成后会在Android项目的目录下生成叫Flutter的目录，这就是需要我们依赖的。还有个需要注意是gradle中的minSdkVersion必须要大于等于16，因为这个flutter支持的最低版本。同时添加使用java8来编译。在app/build.gradle中的android标签中添加：

compileOptions {

 sourceCompatibility JavaVersion.VERSION\_1\_8

 targetCompatibility JavaVersion.VERSION\_1\_8

}

# **调用Flutter module**

依赖完成后我们就可以调用flutter模块来创建UI了。Flutter为我们提供了两种方式调用，一种是createView，以view的形式加载。另一种是createFragment，以Android中的fragment的形式加载。

createView方式：

public **class** **MainActivity** **extends** **AppCompatActivity** {

 **@Override**

 **protected** void onCreate(**@Nullable** **Bundle** savedInstanceState) {

 **super**.onCreate(savedInstanceState);*// setContentView(R.layout.activity\_main);*

 **FlutterView** flutterView = **Flutter**.createView(**this**, getLifecycle(), "initialRoute");

 setContentView(flutterView);

 }

}

createFragment方式：

public **class** **MainActivity** **extends** **AppCompatActivity** {

 **@Override**

 **protected** void onCreate(**@Nullable** **Bundle** savedInstanceState) {

 **super**.onCreate(savedInstanceState);

 setContentView(**R**.layout.activity\_main);

 **FragmentTransaction** transaction = getSupportFragmentManager().beginTransaction();

 *//container为activity\_main布局中的占位符FrameLayout*

 transaction.replace(**R**.id.container, **Flutter**.createFragment("initialRoute"));

 transaction.commit();

 }

}

这样就将Flutter默认的首页加载到应用上了。



从上面两部分代码中我们可以看到都有一个 “initialRoute” 参数，这个参数是用来告诉Dart代码在Flutter视图中显示哪个小部件。下面我们就来修改module中的main.dart代码来加载我们自己的页面。

我们设置两个route，分别展示route1Widget，和route2Widget，当没有匹配的时候展示提醒文字。

**import** 'package:flutter/material.dart';**import** 'dart:ui';

void main() => runApp(**MyApp**(

 *//通过window.defaultRouteName获取从native传递过来的参数,需要导入dart:ui包*

 initParams: window.defaultRouteName,

 ));

**class** **MyApp** **extends** **StatelessWidget** {

 **final** **String** initParams;

 **MyApp**({**Key** key, **this**.initParams}) : **super**(key: key);

 **@override**

 **Widget** build(**BuildContext** context) {

 **return** **MaterialApp**(

 title: 'Flutter\_Android混合开发',

 theme: **ThemeData**(

 primarySwatch: **Colors**.blue,

 ),

 home: **HomePage**(initParams: initParams),

 );

 }

}

**class** **HomePage** **extends** **StatefulWidget** {

 **final** **String** initParams;

 const **HomePage**({**Key** key, **this**.initParams}) : **super**(key: key);

 **@override**

 \_HomePageState createState() => \_HomePageState();

}

**class** **\_HomePageState** **extends** **State<HomePage>** {

 **@override**

 **Widget** build(**BuildContext** context) {

 **return** **Scaffold**(

 body: **Center**(

 child: \_widgetRoute(widget.initParams),

 ),

 );

 }

}

*///路由转发***Widget** \_widgetRoute(**String** route) {

 switch (route) {

 **case** "route1":

 **return** route1Widget();

 **case** "route2":

 **return** route2Widget();

 **default**:

 **return** notFoundWidget();

 }

}

**Widget** route1Widget() {

 **return** **Center**(

 child: **Text**(

 "this is route1Widget",

 style: **TextStyle**(color: **Colors**.red, fontSize: 20),

 ),

 );

}

**Widget** route2Widget() {

 **return** **Center**(

 child: **Text**(

 "this is route2Widget",

 style: **TextStyle**(color: **Colors**.blue, fontSize: 20),

 ),

 );

}

**Widget** notFoundWidget() {

 **return** **Center**(

 child: **Text**(

 "未匹配到路由111",

 style: **TextStyle**(fontSize: 40),

 ),

 );

}

我们现在将加载Flutter时的initialRoute参数替换为 “route1”，那页面将加载route1Widget，替换为 “route2”，将加载route2Widget。否则将展示notFoundWidget。当然我们可以直接传路由参数，但是因为这个参数本身是一个字符串，所以我们可以来搞事情。比如传递一个json串，那么是不是可以做很多事呢？这里我就不贴demo了，因为和上面的逻辑基本一样，大家可以去试试看。

# **热重启/重新加载**

大家在写纯Flutter应用的时候，知道是有热重启/重新加载功能的，但是在做混合开发的过程中，你会发现热重启/重新加载功能失效了。那么如何在混合开发中开启热重启/重新加载功能呢？

* 首先接入我们的设备或者模拟器
* 将我们的App关闭，退出后台，在terminal中运行 flutter attach命令

$ flutter attach

Waiting **for** a connection **from** Flutter **on** Android SDK built **for** x86...

此时就在等待设备的连接。这里要注意的是，如果电脑连接了多台设备需要使用 -d 命令来指定一台设备，参数为设备的id。

flutter attach -d '你的设备id'

* 然后启动我们的应用会看到控制台输出：

Done.

Syncing files **to** device Android SDK built **for** x86... 1,393ms

**To** hot reload changes **while** running, press "r". **To** hot restart (and rebuild state), press "R".

An Observatory debugger and profiler on Android SDK built **for** x86 is available at: http:*//127.0.0.1:59354/zRsDBfpesrk=/***For** a more detailed help message, press "h". **To** detach, press "d"; **to** quit, press "q".

这样就表示我们连接成功了。在输出的日志中也告诉了我们如何使用热重启/重新加载功能。

在Terminal中输入以下命令​：​

r : 热加载；

R : 热重启；

h : 获取帮助；

d : 断开连接；

q : 退出；

这里的的 d 和 q 的命令都有退出调试，区别在于 d 命令只是单纯的断开而 q 命令会将应用退到后台。

# **调试Dart代码**

同样在混合开发过程中我们如何调试dart代码呢？

* 关闭我们的应用
* 点击Android Studio工具栏上的Flutter Attach按钮(需要安装Flutter与Dart插件)



* 启动我们的应用

接下来就可以像调试普通Flutter项目一样来调试混合开发模式下的Dart代码了。

# **总结**

以上就是如何在Android原生项目中接入Flutter模块的基础讲解，主要就是模块的创建、依赖、调用以及调试等等。其它的像iOS接入Flutter模块，Android项目和Flutter项目之间的通信以及iOS项目和Flutter之间的通信都将在之后的文章中进行讲解。因为写在一篇中篇幅太长，朋友们读起来也累。所以后续还会有至少三篇相关的文章和大家见面。动动手关注公众号，即时获取相关文章的推送。