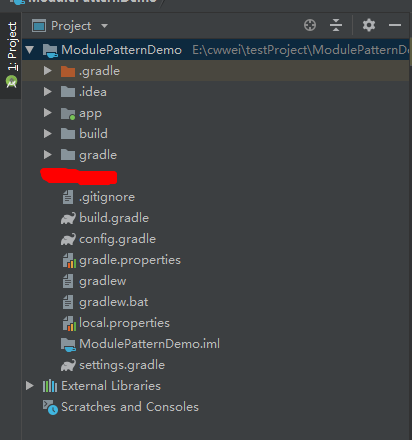
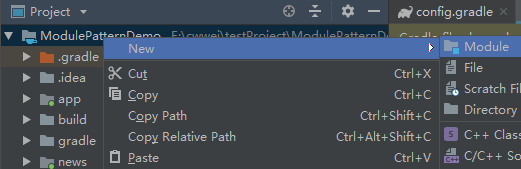
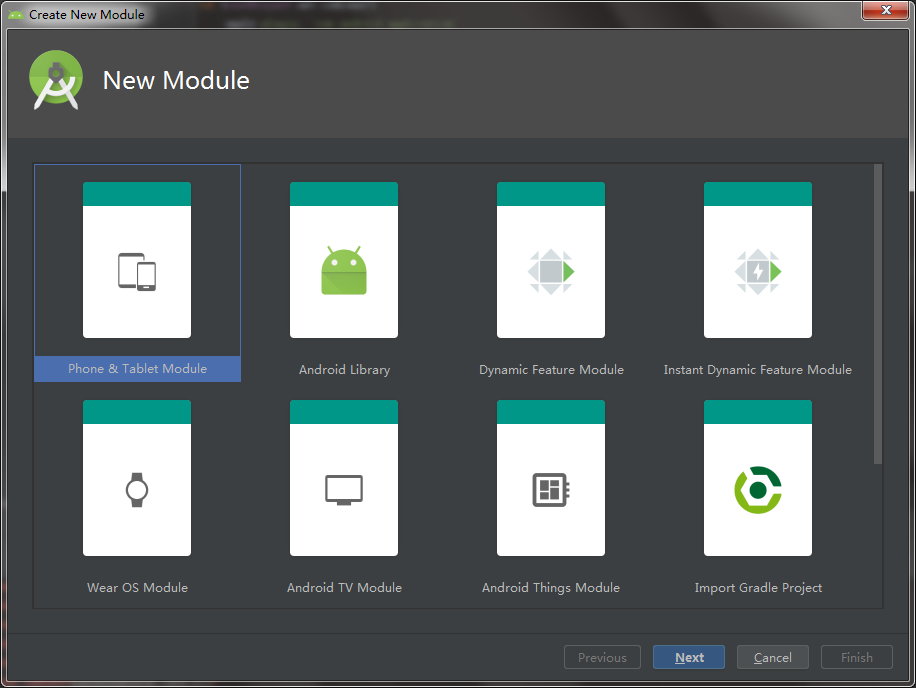
AS版本：3.3

1、创建一个壳APP：



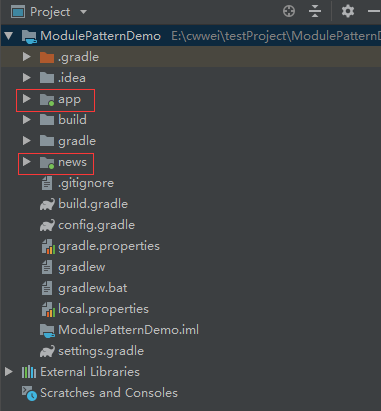
2、在本工程新建一个module，例如news。





（注意，如果next后minimum sdk选不了，返回上一步，再进入。）

3、最后工程有2个可独立运行的app



组件化实现

1、首先在工程根目录新建一个配置文件，如上面的config.gradle：

ext{  
 //变量的定义：xxx1 = ..., 或数组里 xxx2 : ..., 引用分别为：$xxx1, ["xxx2"]  
 isAlone = true  
// isAlone = false  
// kotlin\_version = '1.3.21'  
  
 android = [  
 compileSdkVersion : 28,  
 minSdkVersion : 19,  
 targetSdkVersion : 28,  
 versionCode : 1,  
 versionName : "1.0"  
 ]  
  
 dependencies = [  
 "androidx-appcompat" : 'androidx.appcompat:appcompat:1.0.0-beta01',  
 "androidx-core" : 'androidx.core:core-ktx:1.1.0-alpha04',  
 "androidx-constraintlayout" : 'androidx.constraintlayout:constraintlayout:1.1.3',  
 "androidx-test-runner" : 'androidx.test:runner:1.1.0-alpha4',  
 "androidx-test-espresso" : 'androidx.test.espresso:espresso-core:3.1.0-alpha4',  
 "junit" : 'junit:junit:4.12',  
 ]  
  
}

定义了一些版本信息和依赖（这些和组件化无关，有关的是isAlone）。

如何使用？

首先要在工程根目录的build.gradle文件的开头使用：

apply from: "config.gradle"

这样module或app里的build.gradle就可以使用config.gradle的变量了，例如在news：

if (rootProject.ext.isAlone){  
 apply plugin: 'com.android.application'  
} else{  
 apply plugin: 'com.android.library'  
}

或

compileSdkVersion rootProject.ext.android["compileSdkVersion"]

isAlone用来编译时判断news module时做为app还是library。

2、改造news的AndroidManifest.xml。

module作为组件和app时，这个文件必须要修改的。

a、在news的src/main下新建一个module目录，复制一份AndroidManifest.xml；

b、在news的build.gradle的android层配置，这样不同的模式会读取不同的配置文件：

sourceSets {  
 main {  
 if (rootProject.ext.isAlone){  
 manifest.srcFile 'src/main/module/AndroidManifest.xml'  
// java.srcDirs = ['src/main/java', 'src/main/module/java']  
// res.srcDirs = ['src/main/res', 'src/main/module/res']  
 } else {  
 //作为组件  
 manifest.srcFile 'src/main/AndroidManifest.xml'  
 }  
 }  
 }

c、作为组件的AndroidManifest.xml，application标签我这里只保留主题属性。

<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 package="com.example.news">  
  
 <application  
 android:theme="@style/AppTheme">  
 <activity android:name=".NewsMainActivity"/>  
 </application>

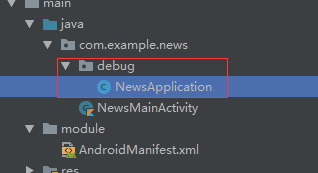
</manifest>

3、改造全局Context。

作为独立app时，news需要有自己的application，而作为组件（library）时，使用的是全局（壳app）的application，因此也需要修改。

类似修改AndroidManifest.xml，在java的包下新建debug包，再建一个NewsApplication.java，

NewsApplication继承Application，module的AndroidManifest.xml设置使用这个application。



但是作为组件时又不想使用这个NewsApplication，可以这样配置：

sourceSets {  
 main {  
 if (rootProject.ext.isAlone){  
 manifest.srcFile 'src/main/module/AndroidManifest.xml'  
// java.srcDirs = ['src/main/java', 'src/main/module/java']  
// res.srcDirs = ['src/main/res', 'src/main/module/res']  
 } else {  
 //作为组件  
 manifest.srcFile 'src/main/AndroidManifest.xml'  
 //排除debug文件夹中的所有Java文件  
 java {  
 exclude 'debug/\*\*'  
 }  
 }  
 }  
 }

那么作为组件时，使用的application时哪个？

在common组件（下面第6点说到）定义了一个BaseApplication，所有组件及壳app声明的application都继承这个BaseApplication。当app启动时，BaseApplication会被动地实例化。

这样会有个问题：news作为组件时，它的activity被其它组件或壳app启动时有可能带参数。而作为独立app启动时没有参数。

可以在debug目录添加一个启动的activity，不带界面，然后传递参数去跳转到目标activity。

问题：A组件是登录业务，B组件是登录后的业务。B作为独立app时，网络请求无法成功，因为需要token。

解决：独立app时可以在独立的application创建一个模拟登录，这样就不需要再在已写好的activity来模拟登录，也不实际。

4、第三方lib依赖问题

问题1：我们的项目肯定要依赖AndroidSupport库，而一些第三方也可能已经依赖了AndroidSupport，这个会导致重复依赖。

解决：修改一下依赖方式：

implementation rootProject.ext.dependencies["recyclerview"] 改为：

implementation(rootProject.ext.dependencies["recyclerview"]){  
 exclude module: 'support-v4'//根据组件名排除  
 exclude group: 'android.support.v4'//根据包名排除  
}

问题2：如果有多个组件都依赖一些开源库，这样每个组件都要添加一遍，尤其版本改变时更麻烦。

解决：使用一个common库管理这些开源库，每个组件依赖这个common库。（后面再改造）

5、组件间通信和调用

目前，壳app和其它组件、其它组件之间都不能相互跳转及携带数据。需要一个“路由”。

github上路由的开源库有： Arouter ，**ActivityRouter，**[NewFastFrame](https://github.com/HelloChenJinJun/NewFastFrame)。

（可以先了解下第6点的common组件）

这里以**ActivityRouter为例：**

1、在common组件添加依赖

api rootProject.ext.dependencies["router"]  
annotationProcessor rootProject.ext.dependencies["annotationProcessor"]

注意1，AS版本原因可能需要在module的build.gradle的android-defaultconfig层添加：

// 声明需要使用注解功能  
// javaCompileOptions { annotationProcessorOptions { includeCompileClasspath = true } }

注意2，注解依赖是annotationProcessor不是其它。

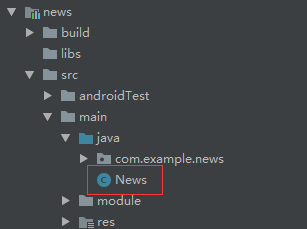
2、在壳app的manifest配置：

<activity android:name="com.github.mzule.activityrouter.router.RouterActivity"  
 android:theme="@android:style/Theme.NoDisplay">  
 <intent-filter>  
 <action android:name="android.intent.action.VIEW" />  
  
 <category android:name="android.intent.category.DEFAULT" />  
 <category android:name="android.intent.category.BROWSABLE" />  
  
 <data android:scheme="module" /> <!-- 改成自己的scheme -->  
 </intent-filter>  
</activity>

注意：scheme的值和url有关，Routers.open(MainActivity.this, "**module**://girls");。

3、声明组件

在每一个业务组件的java文件的根目录下创建一个类，用 **注解@Module** 声明这个业务组件：



@Module("news")  
public class News {  
}

同时在壳app的application类添加注解，组件得到管理：

@Modules({"main", "news"})  
public class MyApplication extends BaseApplication {  
}

4、给组件的activity定义url

@Router("news")  
class NewsMainActivity : AppCompatActivity() {

这样就可以在整个项目的任何地方调用activity：

Routers.open(MainActivity.this, "module://girls");

6、common组件主要是管理组件和壳app的公共资源。

例如依赖开源库、权限、base类（例如baseapplication）等。

这个组件只是一个library，不需要判断isAlone；

common组件的AndroidManifest.xml只包含包名和整个app使用的权限：

<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 package="com.example.common">  
  
 <uses-permission android:name="android.permission.INTERNET"/>  
  
 <!--<application android:allowBackup="true"-->  
 <!--android:label="@string/app\_name"-->  
 <!--android:icon="@mipmap/ic\_launcher"-->  
 <!--android:roundIcon="@mipmap/ic\_launcher\_round"-->  
 <!--android:supportsRtl="true"-->  
 <!--android:theme="@style/AppTheme"/>-->  
</manifest>

common组件的build.gradle：

apply plugin: 'com.android.library'  
  
android {  
 compileSdkVersion rootProject.ext.android["compileSdkVersion"]  
  
  
  
 defaultConfig {  
// applicationId "com.example.common"  
 minSdkVersion rootProject.ext.android["minSdkVersion"]  
 targetSdkVersion rootProject.ext.android["compileSdkVersion"]  
 versionCode rootProject.ext.android["versionCode"]  
 versionName rootProject.ext.android["versionName"]  
  
// testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"  
  
 }  
  
// buildTypes {  
// release {  
// minifyEnabled false  
// proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'  
// }  
// }  
  
}  
  
dependencies {  
 implementation fileTree(dir: 'libs', include: ['\*.jar'])  
  
// implementation 'com.android.support:appcompat-v7:28.0.0'  
// testImplementation 'junit:junit:4.12'  
// androidTestImplementation 'com.android.support.test:runner:1.0.2'  
// androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'

//在此添加第三方开源库

api rootProject.ext.dependencies["recyclerview"]

......  
}

现在可以将news组件的第三方依赖移动到common的build.gradle里了，并在news组件的依赖里添加：

implementation project(':common')

插曲：

问题：关于module（或app）使用library的依赖时，AS3.x以前使用compile，3.x以后加入api和implement。

implement声明依赖只限于本模块使用，所以如果要其它组件也能使用common的依赖，应该要使用api或compile（https://blog.csdn.net/yzhg\_1005/article/details/82759014）

7、壳app

一个空壳module，不能有activity，没有任何业务代码，但有几点要在这个module声明：

a、声明app的应用application，并继承baseapplication；

b、manifest.xml：声明app的图标、名称、application name等

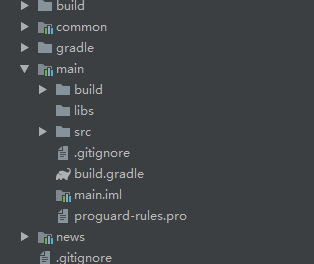
c、build.gradle：始终是application，配置app的打包、签名等

作为特殊的组件，壳app独立模式下只需依赖common组件，集成模式下要依赖所有在应用application中声明的组件（文章说集成模式不再需要依赖common组件，这跟api、implement有关吧）。

总之，壳app中manifest有的属性，业务组件都不能有。

8、main组件

和其它业务组件一样，不同的是整个app的启动activity放在这里。



9、其它参考后面的连接

编译注意：

1、独立编译后，再切换为集成编译，clear一下project；

2、创建module时，布局的类型可能会出错，重新创建。

3、发现壳app不能在AS点击图标运行，没有将main组件依赖进来。

if (rootProject.ext.isAlone){  
 implementation project(':common')  
} else{  
 implementation project(':main')  
 implementation project(':news')  
}

集成模式时，如果每个组件都用implement添加common的依赖，壳app将报错。因为上面的这个依赖语句，至少要有一个组件用api或compile依赖common，壳app才能依赖common。

4、作为组件时，每个组件的工程里使用的manifest不能声明applic的name属性。

5、组件间资源名冲突，前缀都用组件名。

最后：用activityrouter发现activity跳转不了，不知道啥原因，用arouter试下。。。。。。。

https://blog.csdn.net/guiying712/article/details/55213884