Android 组件化开发预习资料

前言

该预习资料中包含一些组件化开发中的重要知识点，忘大家在课前可以稍微浏览下，以助于在听课的时候更容易上手掌握。

为什么一定要掌握组件化开发？

其实在讲到为什么一定要掌握组件化开发的时候，也要刨一刨单一模块开发的坑：

1. **单一模块开发中耦合太严重**
   类与类之间藕断丝连。
2. **单一模块编译速度太慢**
   所以的业务逻辑都在一个模块中，每次修改哪怕一个变量，在测试的时候都要编译整个工程。
3. **无法去做到功能复用**
   正是因为耦合太严重，一些功能在进行重用的时候需要去一个类一个类中理清楚，太麻烦。
4. **团队开发不便利**
   开发大项目的时候都是团队开发，但是单一模块注定是团队开发的死敌。

so,单一模块从上分析还是有很多不足的，所以在这种背景下出现了组件化开发，接下来说说组件化开发的优势(其实就是弥补了单一模块的不足)

1. **业务模块解耦**
   组件化开发中，会根据业务来拆分模块，每个模块之间没有任何的耦合，这样就能够和好的解决单一模块耦合严重的问题。
2. **极大提高工程编译速度**
   所以的业务逻辑都在一个模块中，每次修改哪怕一个变量，在测试的时候都要编译整个工程。
3. **组件化是功能重用的基石**
   每个业务逻辑模块是彼此独立，如果在新项目中又需要用到这个功能，可以直接复制过去使用，不需要进行任何解耦。
4. **团队开发神器**
   如果是团队开发的话，可以每个人或着每个项目小组负责一个模块即可，无需关注其他的功能模块，这样就可以减少沟通成本，提高开发效率。

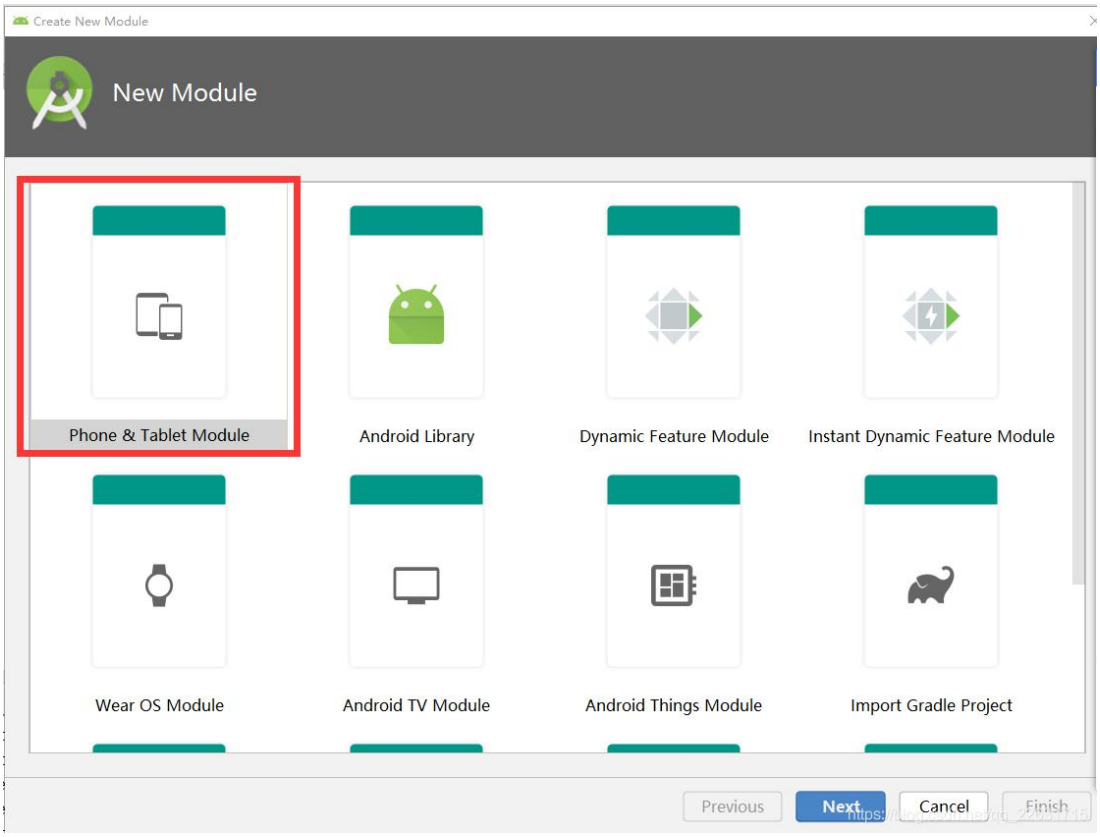
组件化开发的注意事项

1. **要注意包名和资源文件命名冲突问题**
   组件化开发虽然会分将业务逻辑拆分为一个个模块，但是最终打包发布的时候依然要打包到主模块中，所以要注意包名和资源名字的命名问题。
2. **Gradle 中的版本号的统一管理**
   在组件化开发中一定会存在多个模块，每个模块都会有一个 build.gradle 文件，所以必须对每个模块的 build.gradle 进行统一的管理。
3. **组件在 Application 和 Library 之间如何做到随意切换**
   因为每个模块在开发过程中都是 Application，但是最终打包发布的时候都要变为 Library，所以每个模块都需要在 Application 与 Library 之间进行随意的切换。
4. **AndroidManifest.xml 文件的区分**
   正是因为模块要在 Application 与 Library 之间进行切换，所以在不同的状态下要加载不同的 AndroidManifest.xml。
5. **Library 不能在 Gradle 文件中有 applicationId**
   当模块是 Application 的时候，在它的 build.gradle 文件中会有 ApplicationId，但是当它成为 Library 之后，ApplicationId 就不能有了，所以要怎么去进行控制这种情况。

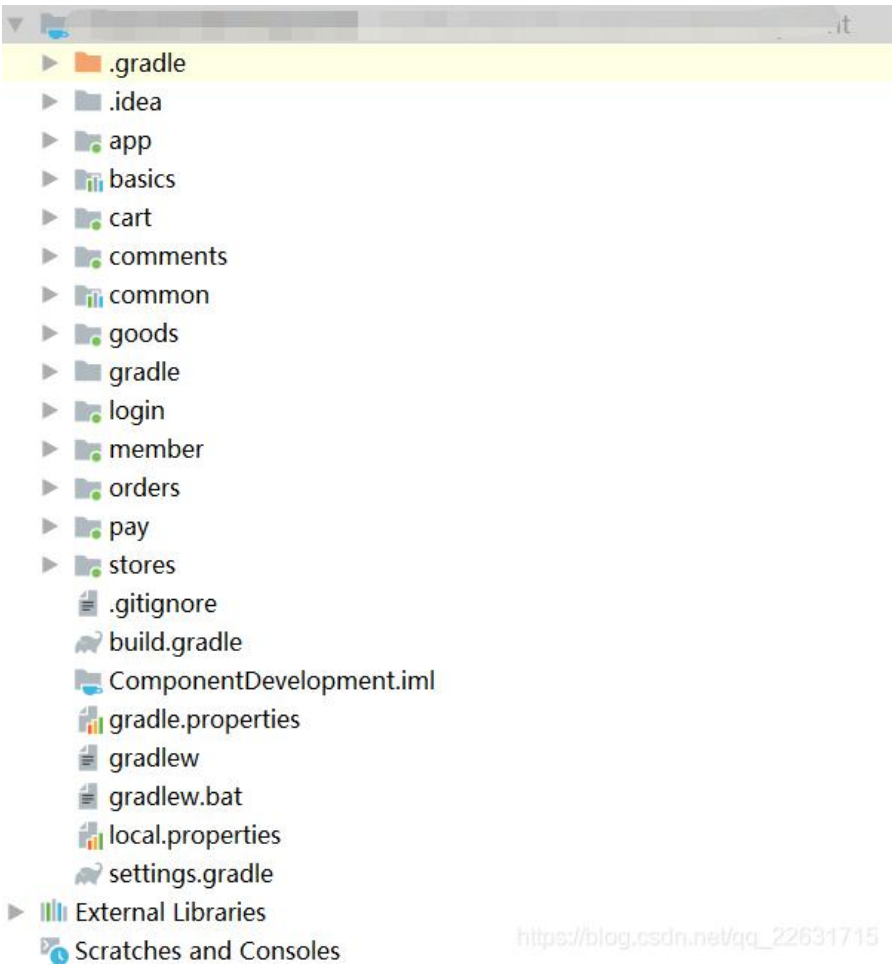
如何把项目组件化

如果是老项目就设计到项目迁移的问题，如果是新项目那就直接可以在创建项目的时候就进行组件化开发。具体步骤如下：

第一步：把业务模块划分好之后，创建相对应的 module


Create New Module
New Module
Phone & Tablet Module
Android Library
Dynamic Feature Module
Instant Dynamic Feature Module
Wear OS Module
Android TV Module
Android Things Module
Import Gradle Project
Previous
Next
Cancel
Finish

最终效果如下：例子



注意：在创建 module 以及 Activity 的时候，命名最好有自己的规则（防止类名重复以及资源名字重复）

第二步：把所有 module 中的版本号以及一些需要统一管理的内容进行统一管理（通过定义全局变量的方式，常用有三种方式）

1. 直接定义在 gradle.properties 文件中，如图：

```
#所有定义的东西  都是字符串类型
# 最小SDK版本
MIN_SDK_VERSION = 21
#目标设备SDK版本
TAR_SDK_VERSION = 28
#SDK编译版本
COMPILER_SDK_VESION = 28

BUILD_TOOL_VESION = 29.0.0
#当前依赖的support库
APP_COMPAT = com.android.support:appcompat-v7:28.0.0
#所有模块的版本号
VERSION_CODE = 1
#所有模块的版本名字
VERSION_NAME = 1.0
```


使用方式：

```
android {
    compileSdkVersion TAR_SDK_VERSION.toInteger()
    buildToolsVersion BUILD_TOOL_VESION


    defaultConfig {
        applicationId "com.maniu.member"
        minSdkVersion MIN_SDK_VERSION.toInteger()
        targetSdkVersion TAR_SDK_VERSION.toInteger()
        versionCode VERSION_CODE.toInteger()
        versionName VERSION_NAME

        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"

    }
```


注意：定义在 gradle.properties 文件中的全局变量都是 String 类型，使用的时候请先转换为需要的类型。

2. 定义在工程的 build.gradle 文件中，如图：

```
task clean(type: Delete) {
    delete rootProject.buildDir
}

ext {
    compileSdkVersion = 28
    buildToolsVersion = "29.0.0"
    minSdkVersion = 21
    targetSdkVersion = 28
    versionCode = 1
    versionName = '1.0'
    APP_COMPAT = "com.android.support:appcompat-v7:28.0.0"
}
```

使用方式：

```
android {
    compileSdkVersion rootProject.ext.compileSdkVersion
    buildToolsVersion rootProject.ext.buildToolsVersion

    defaultConfig {
        applicationId "com.maniu.member"
        minSdkVersion rootProject.ext.minSdkVersion
        targetSdkVersion rootProject.ext.targetSdkVersion
        versionCode rootProject.ext.versionCode
        versionName rootProject.ext.versionName
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
```


3. 自定义 gradle 文件，如图：

config.gradle

```
ext {
    android = [
            compileSdkVersion: 28,
            buildToolsVersion: "29.0.0",
            minSdkVersion    : 21,
            targetSdkVersion : 28,
            versionCode : 1,
            versionName : '1.0'
    ]
    dependencies = [
            appcompatV7:': com.android.support:appcompat-v7:28.0.0',
    ]
}
```


定义好之后还需要是主工程的 build.gradle 进行应用，如图：

```
// Top-level build file where you can add configuration options common to al
apply from: "config.gradle"

buildscript {
    repositories {
        google()
        jcenter()

    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.4.1'

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}
```

使用方式：

```
android {
    compileSdkVersion rootProject.ext.android.compileSdkVersion
    buildToolsVersion rootProject.ext.android.buildToolsVersion


    defaultConfig {
        applicationId "com.maniu.member"
        minSdkVersion rootProject.ext.android.minSdkVersion
        targetSdkVersion rootProject.ext.android.targetSdkVersion
        versionCode rootProject.ext.android.versionCode
        versionName rootProject.ext.android.versionName
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
```


第三步：如何让每个 module 都能在 Application 和 Library 之间进行随意的切换(其实很简单，定义一个 boolean 类型的全局变量当做开关)

config.gradle

```
ext {
    android = [
            compileSdkVersion: 28,
            buildToolsVersion: "29.0.0",
            minSdkVersion    : 21,
            targetSdkVersion : 28,
            versionCode : 1,
            versionName : '1.0',
            is_application : false
    ]
    dependencies = [
            appcompatV7:'com.android.support:appcompat-v7:28.0.0',
    ]
}
```

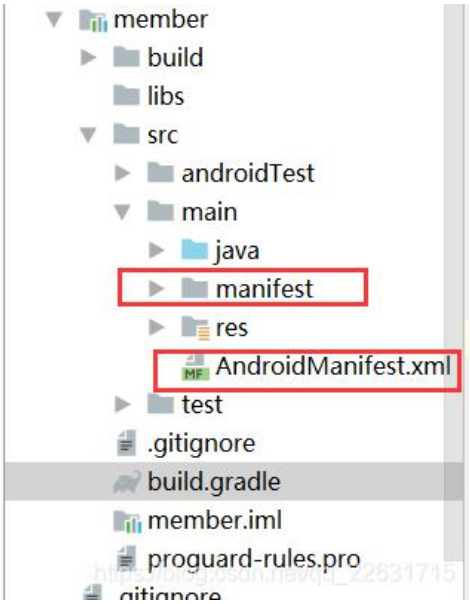


```
if(rootProject.ext.android.is_application){
    apply plugin: 'com.android.application'
}else{
    apply plugin: 'com.android.library'
}

android {
    compileSdkVersion rootProject.ext.android.compileSdkVersion
    buildToolsVersion rootProject.ext.android.buildToolsVersion


    defaultConfig {
        if(rootProject.ext.android.is_application){
            applicationId "com.maniu.member"
        }
        minSdkVersion rootProject.ext.android.minSdkVersion
        targetSdkVersion rootProject.ext.android.targetSdkVersion
        versionCode rootProject.ext.android.versionCode
        versionName rootProject.ext.android.versionName
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
```


注意：只有当 module 是 Application 的时候才具有 applicationId，所以这里也要进行处理。

第四步：Application 和 Library 所加载 AndroidManifest.xml 文件要区分，因为他们对 AndroidManifest.xml 文件的要求不一样。

处理方式很简单，直接复制一个 AndroidManifest.xml 文件，一份在 module 是 Application 的时候用，一份在 module 是 Library 的时候用。如图：



Library 下的 AndroidManifest.xml 文件

AndroidManifest.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.maniu.member">

    <application
        >
        <activity android:name=".MemberActivity">
        </activity>
    </application>

</manifest>
```


在 module 的 build.gradle 文件中区分使用

```
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),
        }
    }

    sourceSets{
        main{
            if(rootProject.ext.android.is_application){
                manifest.srcFile 'src/main/AndroidManifest.xml'
            }else{
                manifest.srcFile 'src/main/manifest/AndroidManifest.xml'
            }
        }
    }
```

第五步：将其他业务逻辑 module 注入到主 module 中（注意 Application 是不能依赖其他的 Application，所以要记得判断依赖的时候，其他模块是否是 Application）。

```
//只有当所有业务逻辑模块都是library的时候才依赖他们
if(!rootProject.ext.android.is_application){
    implementation project(path: ':member')
}
```

到此为止，一个组件化项目的架子就搭好了。注意：以上的操作都针对所有的业务逻辑 **module**，所有业务逻辑 **module** 都要进行这样处理。