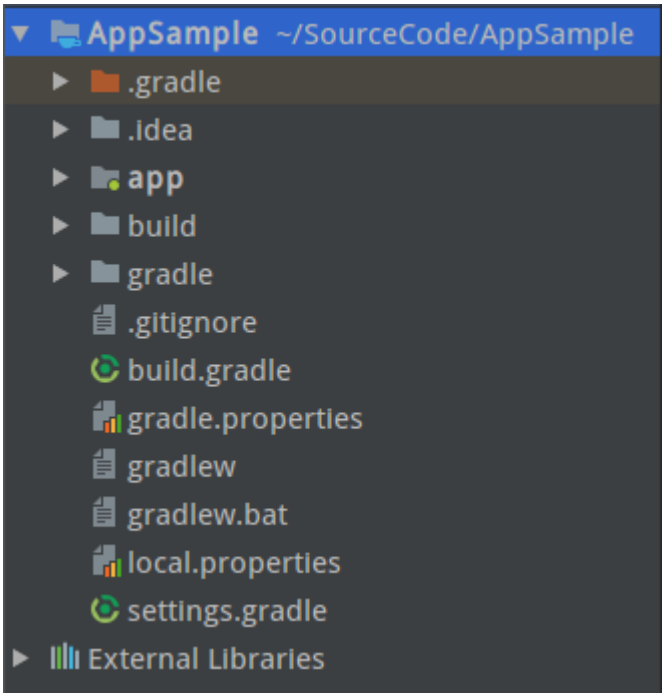
- 前言
- 为什么要组件化
- 开始组件化实践
- 组件的通信
- 关于组件合并的坑
- 关于合并资源差异化
- 小结

前言

关于组件化的开发之前有同学分享过一次，不知道在座的各位有木有在实际项目中实践呢？

为什么要组件化

> 之前小爱的项目最初立项的时候只有一个app模块，就类似于我现在新建的项目工程

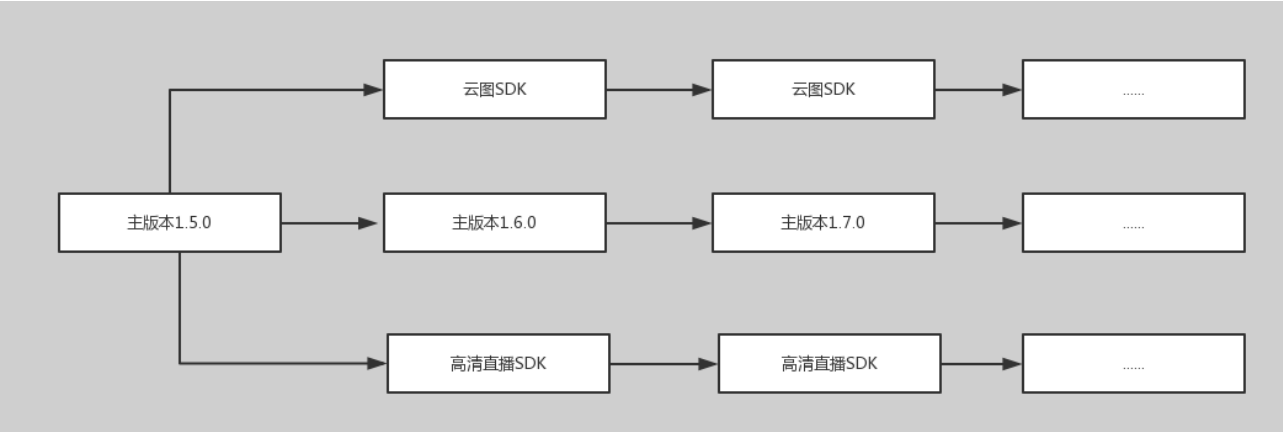


> 可以看到这种工程结构比较单一，但是随着项目的不断迭代，功能越来越多，已经很难满足我们现在的业务了。
>
> 小爱直播现在除了自己的主版本以外，还需要将主版本部分功能抽离成sdk，提供给合作方接入(云图，高清直播，潘多拉魔盒等等)，除此之外，合作方可能会作不同程度的定制，云图需要短视频模块，但不需要娃娃机功能，高清直播觉得体积太大只要直播功能，并且希望动态下载库并加载的功能。那么基于这种方式还能友好的玩耍吗？

之前的做法：由主分支切出对应的几个不同的sdk分支

```
sdk_yuntu
sdk_hanju
sdk_mobiletv
....
```

于是就有了下图



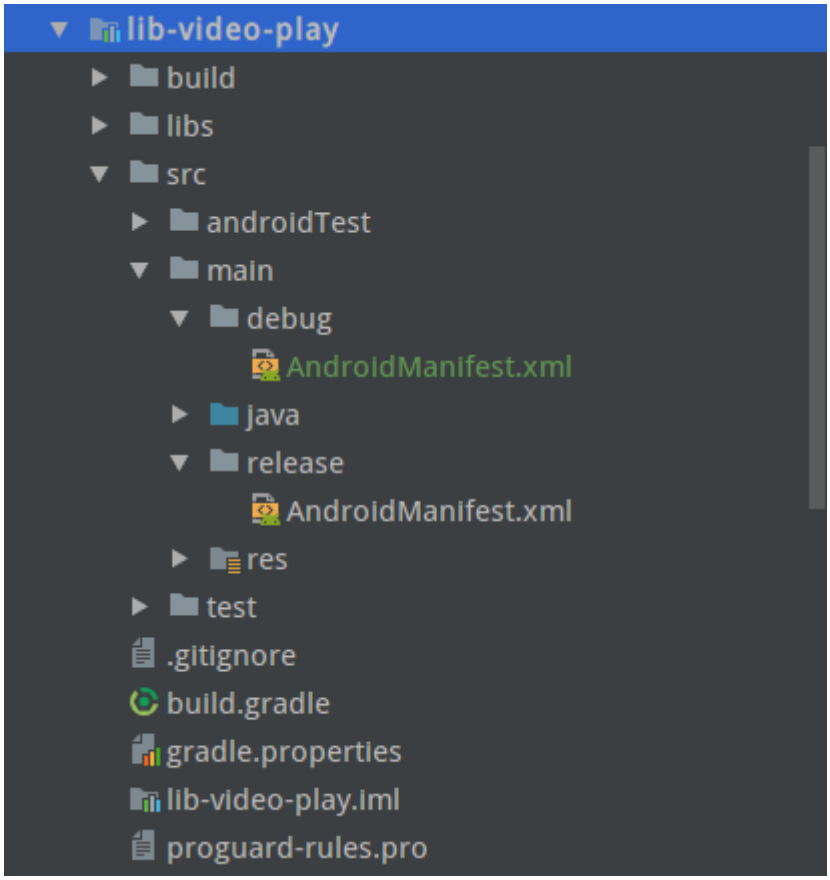
> 可以看到，这种方式只能解决一时之需，导致的一个后果就是，不同的分支往后差异会越来越大，就产生了一个新的问题，每次合作方需要更新的时候，会有很多的差异需要合并，并且有些在云图sdk分支修改的bug，并没有及时同步到其他分支上，又会导致同样的问题产生，耗费大量的时间，做了很多重复的工作。

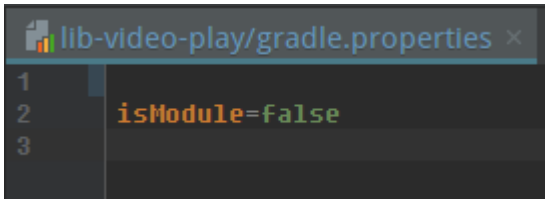
- 思考

> 有了上述的诸多问题，我们能否有一种解决方案，基于主版本分支可以自由灵活的合并我们的模块提供给合作方，而又不影响或者少影响我们主工程的业务呢？

开始组件化实践

我以其中一个短视频播放组件为例





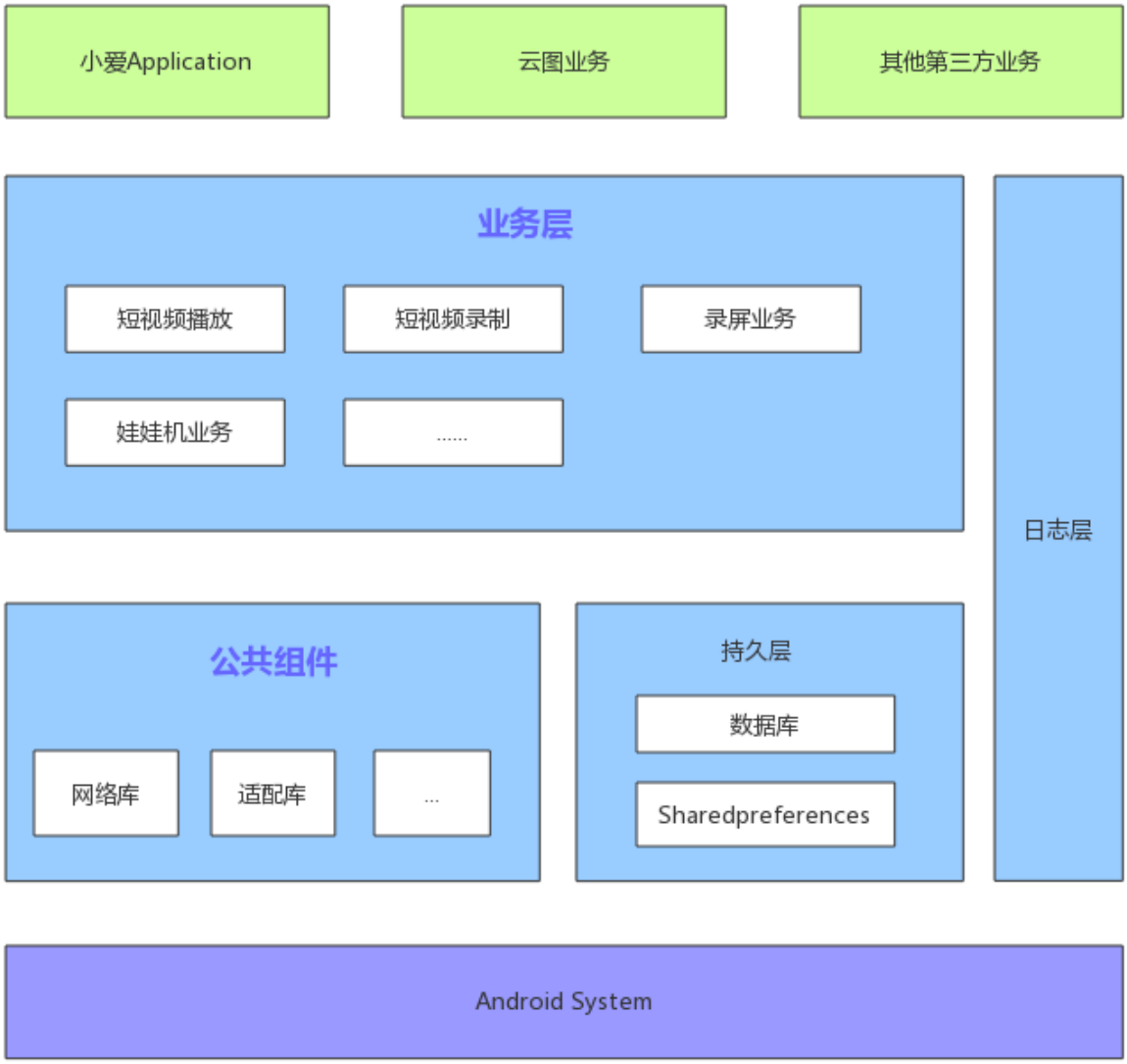
> 新建一个组件lib-video-play
>
> 修改当前组件中的gradle.properties文件
>
> 修改对应的build.gradle文件

```
if (isModule.toBoolean()) {
    apply plugin: 'com.android.application'
} else {
    apply plugin: 'com.android.library'
}
```

```
sourceSets {
    main {
        if (isModule.toBoolean()) {
            manifest.srcFile 'src/main/debug/AndroidManifest.xml'
        } else {
            manifest.srcFile 'src/main/release/AndroidManifest.xml'
        }
    }
}
```

> 基本上这三步就可以实现了该组件在application和library之间的切换了。

小爱直播项目的架构图



组件的通信

> 1.使用隐式跳转
>
> 2.使用事件总线(EventBus,Rxbus等)
>
> 3.使用开源的ARouter来实现

目前小爱直播项目是采用的第三种方案，方便扩展。

关于组件合并的痛

> 前面组件化的实践和通信已经基本落地实践了，一般的应用到这里就可以了，但是由于业务的需要，小爱除了自己打包成主版本apk发布市场以外，还需要抽离部分业务模块给第三方app使用，组件化只是我们的第一步，如何应对这种变化的业务需求呢？

聪明的你肯定很快就会想到了，那么基于组件化的这种方式有两种可行的方案：

> 1.将各个独立的组件分别打包成对应的aar，提供给第三方，但是又涉及到一个问题，那就是混淆的问题，如果直接分别提供原始的aar包，那么源代码几乎等于完全暴露，如果分别混淆，又会存在一个问题，公共组件中常用的工具类被混淆，上层的 短视频 这些组件就会找不到对应的类。
>
> ```
> lib-common.aar
> lib-video-play.aar
> ...
> ```
>
> 2.将各个独立的组件合并混淆成一个aar文件，提供给第三方使用，简便快捷。
> 遗憾的是Android官方并没有提供这种合并的操作，但是发现github上有作者开源了一个合并脚本[fat-aar.gradle](fat-aar.gradle)，这个脚本的作用实际就是合并我们的多个组件为一个aar
>
> 新建一个新的module名字为live-yuntu，在对应的build.gradle配置中，配置如下

```
apply from: "../fat-aar.gradle"


embedded project(':lib-common')
embedded project(':app')
embedded project(':lib-video-play')
```

```
def getArtifactFileName() {
    return "${POM_ARTIFACT_ID}-${VERSION_NAME}.aar"
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:25.3.1'
    testCompile 'junit:junit:4.12'

    embedded project(':lib-common')
    embedded project(':app')
    embedded project(':lib-video-play')
}
```

> 这个关键字由脚本定义，主要用来标识我们要合并的module，关于fat-aar合并的操作步骤：

1.合并Manifests清单文件
2.合并混淆配置文件
3.生成组件的R文件
4.组合本地依赖的jar包
5.合并R文件

> 根据上面的命令就可以合并多个组件，生成最终的aar包，至于脚本的具体实现，时间关系，不细说，大家可以下去看看里面的内容

第一个问题：

> 由于之前项目的gradle插件版本是2.3.3，合并出来的aar包，给其他工程依赖，导致运行的时候出现如下错误：

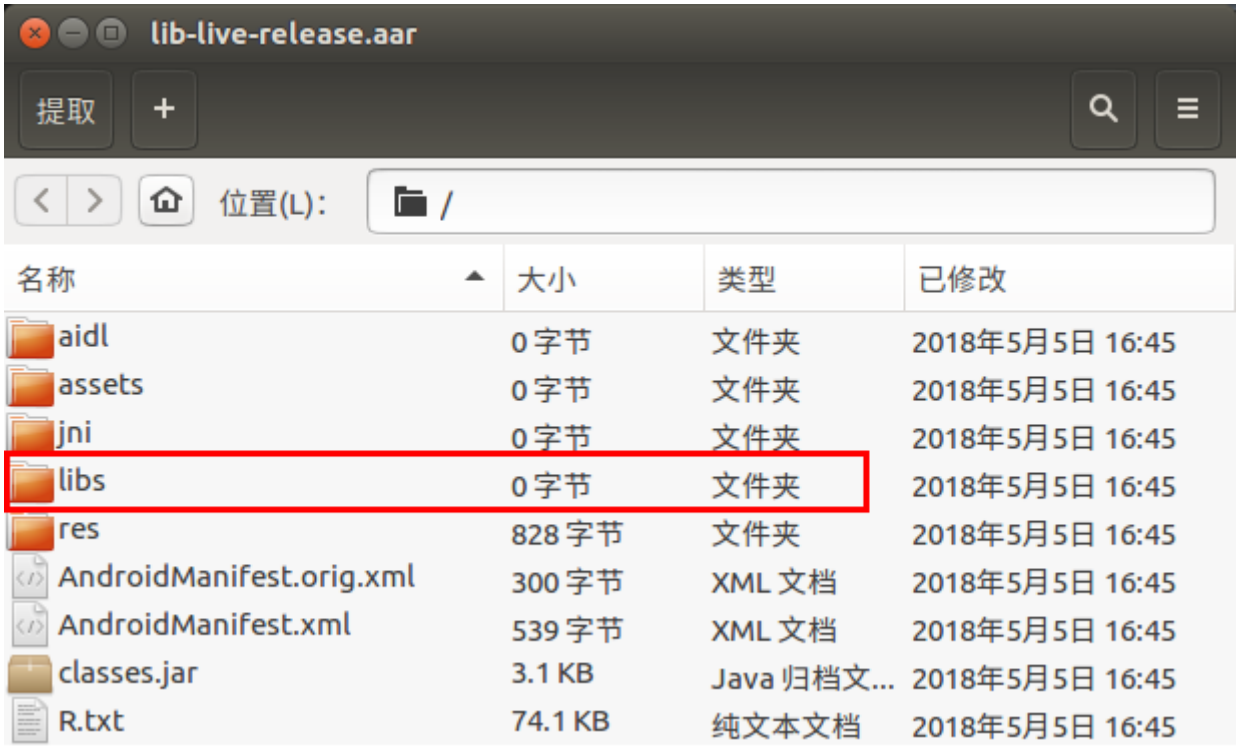
```
E/AndroidRuntime: FATAL EXCEPTION: main
                  Process: com.byhook.sample, PID: 18061
                  java.lang.NoClassDefFoundError: Failed resolution of: Lcom/fungo/common/R$layout;
                      at com.fungo.common.CommonActivity.onCreate(CommonActivity.java:19)
                      at android.app.Activity.performCreate(Activity.java:6013)
                      at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1108)
                      at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2359)
                      at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2466)
                      at android.app.ActivityThread.access$1200(ActivityThread.java:152)
                      at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1341)
                      at android.os.Handler.dispatchMessage(Handler.java:102)
                      at android.os.Looper.loop(Looper.java:135)
                      at android.app.ActivityThread.main(ActivityThread.java:5538) <2 internal calls>
                      at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:958)
                      at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:753)
                   Caused by: java.lang.ClassNotFoundException: Didn't find class "com.fungo.common.R$layout"
                      at dalvik.system.BaseDexClassLoader.findClass(BaseDexClassLoader.java:56)
                      at java.lang.ClassLoader.loadClass(ClassLoader.java:511)
                      at java.lang.ClassLoader.loadClass(ClassLoader.java:469) <10 more...> <2 internal calls
                   Suppressed: java.lang.ClassNotFoundException: com.fungo.common.R$layout
                      at java.lang.Class.classForName(Native Method)
                      at java.lang.BootClassLoader.findClass(ClassLoader.java:781)
                      at java.lang.BootClassLoader.loadClass(ClassLoader.java:841)
                      at java.lang.ClassLoader.loadClass(ClassLoader.java:504)
```

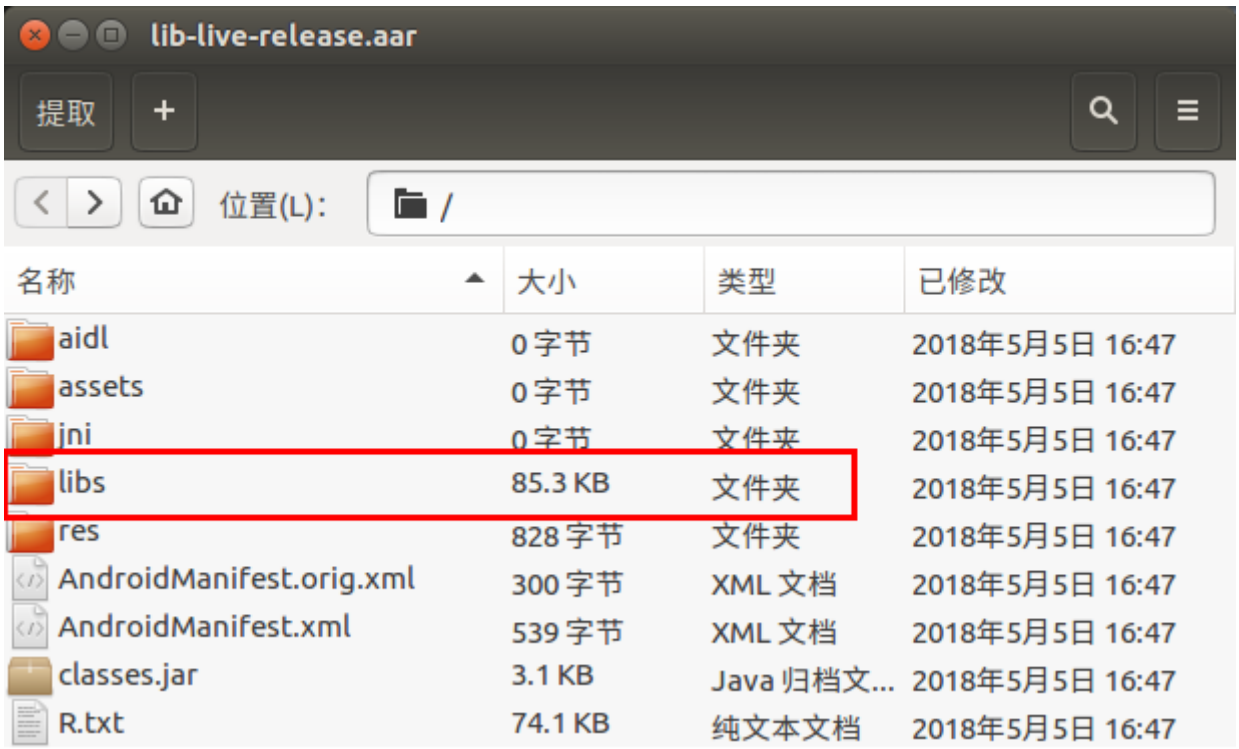
解决方案：

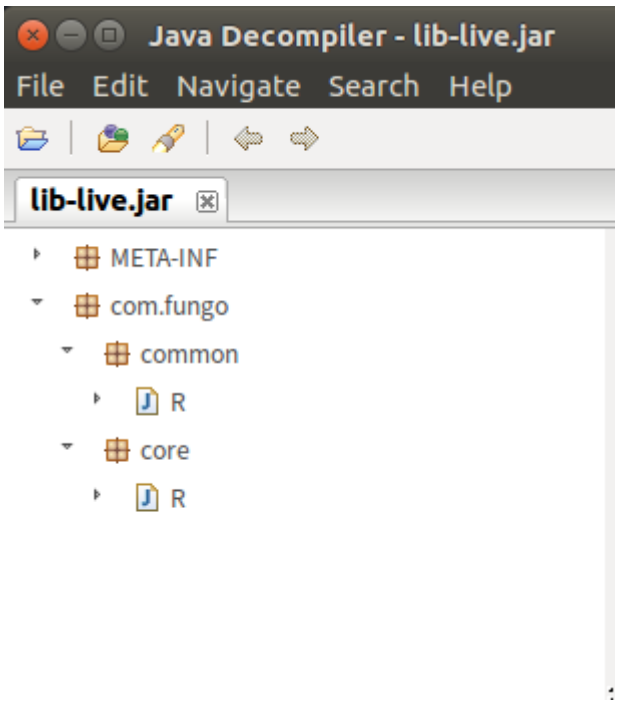
```
// Top-level build file where you can add configuration options common to all sub-
buildscript {

    repositories {
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:2.2.3'


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

allprojects {
    repositories {
        jcenter()
    }
}
```

```
// Embedded aar files dependencies
ext.embeddedAarFiles = new ArrayList<ResolvedArtifact>()
// List of embedded R classes
ext.embeddedRClasses = new ArrayList()

// Change backslash to forward slash on windows
ext.build_dir = buildDir.path.replace(File.separator, '/');
ext.root_dir = project.rootDir.absolutePath.replace(File.separator, '/');
ext.exploded_aar_dir = "$build_dir/intermediates/exploded-aar";
ext.classs_release_dir = "$build_dir/intermediates/classes/release";
ext.bundle_release_dir = "$build_dir/intermediates/bundles/release";
ext.manifest_aaapt_dir = "$build_dir/intermediates/manifests/aapt/release";
ext.generated_rsrc_dir = "$build_dir/generated/source/r/release";

ext.base_r2x_dir = "$build_dir/fat-aar/release/";

def gradleVersionStr = GradleVersion.current().getVersion();

ext.gradleApiVersion = "2.2".toFloat();   //gradleVersionStr.substring(0, gradle

println "Gradle version: " + gradleVersionStr;
```

> 合并成功，并且依赖到其他工程中也没有刚才的R文件相关的错误了，但是问题的根源还是没有找到。为了搞清楚原因，我对改版本号之前和之后分别合并了一份aar文件，进行对比。







所以这个问题的解决方案有两种：

> 1.像我上面那样修改gradle插件的版本号为2.2.3，并修改对应的脚本内的版本号
>
> 2.如果需要高版本2.2.3以上，可以考虑修改fat-aar脚本，使之合并R文件相关的jar包
>
> 目前小爱直播中使用的是第一种方案，第二种方案只是笔者的思路，从上述几张图的分析可以看出，找不到R相关的错误，是因为相关的R文件的jar

> 包没有合并进来，因此是否可以再次修改fat-aar.gradle脚本，将生成的jar包合并进来呢？这个大家下去之后可以思考一下?

第二个问题：

> 降级之后合并的库被其他工程引用，使用本地的模块没有问题了，但是一旦有些模块使用了第三方的类比如okhttp,glide等等，就会出现
>
> ```
> ClassNotFoundException
> ```
>
> 这就是合并打包的第二个坑，因为合并的aar工程并没有将我们的一些依赖合并进来，导致其他工程引用的时候，出现找不到类的错误。
>
> 第一种：我们可以将我们依赖的库告诉合作方，让他们依赖，而我们自己只提供本地的aar包

```
dependencies {
    compile fileTree(include: ['*.jar'], dir: 'libs')
    compile fileTree(include: ['*.jar'], dir: 'libs/linkedme')
    testCompile 'junit:junit:4.12'
    compile rootProject.ext.dependencies["appcompat-v7"]
    compile rootProject.ext.dependencies["design"]
    compile rootProject.ext.dependencies["palette-v7"]
    compile rootProject.ext.dependencies["glide"]
    compile rootProject.ext.dependencies["fresco"]
    compile rootProject.ext.dependencies["fresco-webp"]
    compile rootProject.ext.dependencies["fresco-anim"]
    compile rootProject.ext.dependencies["fresco-animwebp"]
    compile rootProject.ext.dependencies["fresco-gif"]
    compile rootProject.ext.dependencies["fresco-okhttp"]
    compile rootProject.ext.dependencies["fastjson"]
    compile rootProject.ext.dependencies["okhttp"]
    compile rootProject.ext.dependencies["gson"]
    compile rootProject.ext.dependencies["nineoldandroids"]
    compile rootProject.ext.dependencies["android-gif-drawable"]
    compile rootProject.ext.dependencies["PhotoView"]
    compile rootProject.ext.dependencies["greendao"]
    compile rootProject.ext.dependencies["support-dynamic-animation"]
    compile rootProject.ext.dependencies["oss-android-sdk"]
    compile(rootProject.ext.dependencies["rxbus"], {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
```

> 第二种：我们自己合并本地的远程依赖库生成对应的pom.xml文件，然后一起打包上传到maven上去

```
compile 'com.fungo.loveshow:live-yuntu:x.x.x'
```

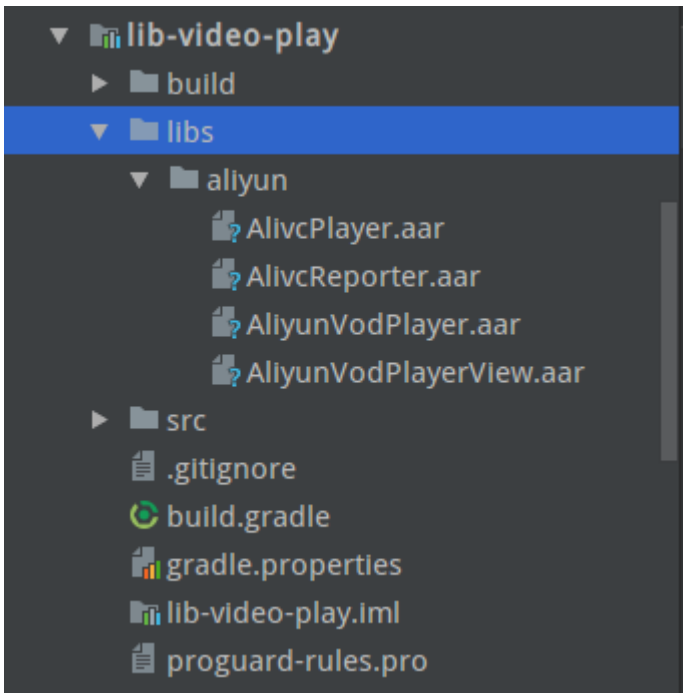
合并发布到**maven**的问题

> 刚刚也讲了合并aar的一些问题，笔者也发现fat-aar的作者也提供了相关的合并操作的脚本publish.gradle，既然有了现成的轮子，我们就直接跑呗！

```
androidLibs(MavenPublication) {
    groupId = GROUP
    artifactId POM_ARTIFACT_ID
    version = VERSION_NAME

    artifact "${project.buildDir}/outputs/aar/${getArtifactFileName()}" //bundleRelease

    List<String> embedList = new ArrayList<>();
    Map<String, ResolvedDependency> depList = new LinkedHashMap<>();

    //List all embedded dependencies
    configurations.embedded.allDependencies.each {
        def depName = String.format("%s:%s", it.group, it.name)
        embedList.add(depName);
    }

    //Collect all first level dependencies except embedded ones
    configurations.compile.resolvedConfiguration.firstLevelModuleDependencies.each {
        ResolvedDependency dep ->
            def depName = String.format("%s:%s", dep.moduleGroup, dep.moduleName)
            if (!embedList.contains(depName) && !depList.containsKey(depName)) {
                depList.put(depName, dep)
            }
    }
```

> 脚本的实现基本可以看到，是遍历了 `embedded` 指定的组件，获取其中的依赖，然后生成相应的pom.xml文件:

```
depList.values().each {
    ResolvedDependency dep ->
        def hasGroup = dep.moduleGroup != null
        def hasName = (dep.moduleName != null || "unspecified".equals(dep.moduleName))
        def hasVersion = dep.moduleVersion != null

        if (hasGroup && hasName && hasVersion) {
            def dependencyNode = dependenciesNode.appendNode('dependency')
            dependencyNode.appendNode('groupId', dep.moduleGroup)
            dependencyNode.appendNode('artifactId', dep.moduleName)
            dependencyNode.appendNode('version', dep.moduleVersion)
        }
}
```



> 实际使用过程中，依然存在问题。那就是我们依赖的第三方，并不都是gradle远程依赖的，有些类似娃娃机和短视频这些库，实际是aar包的形式本地依赖
>
> 这些aar包直接合并进入我们的sdk会存在问题，用上述脚步直接生成会将本地的aar包给合并进去，导致发布maven仓库的时候，远程依赖不了这些aar包报错

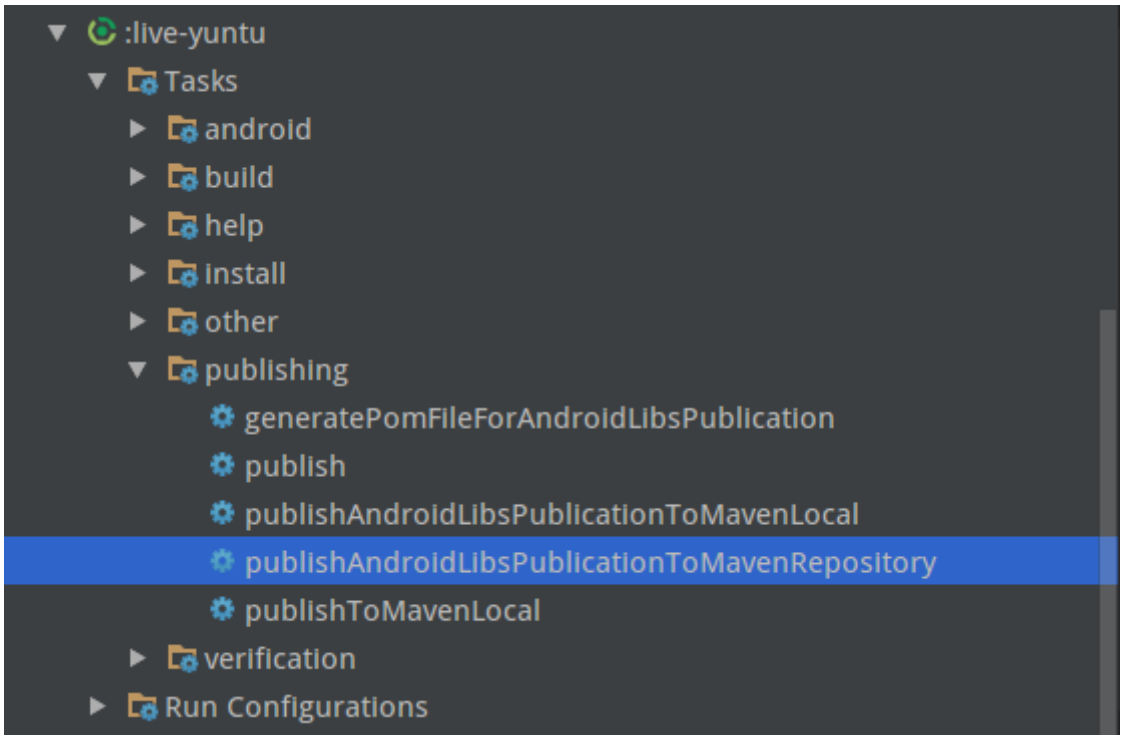
```
<dependency>
  <groupId>com.ksyun.media</groupId>
  <artifactId>libksylive-armv7a</artifactId>
  <version>2.6.3</version>
</dependency>
<dependency>
  <groupId/>
  <artifactId>aliyun/AlivcPlayer</artifactId>
  <version/>
</dependency>
<dependency>
  <groupId/>
  <artifactId>aliyun/AliyunVodPlayer</artifactId>
  <version/>
</dependency>
<dependency>
  <groupId/>
  <artifactId>aliyun/AliyunVodPlayerView</artifactId>
  <version/>
</dependency>
<dependency>
  <groupId/>
  <artifactId>aliyun/AlivcReporter</artifactId>
  <version/>
```

解决方案：

> 如图，因为本地依赖的第三方aar，使用此脚本合并生成pom.xml文件的时候，version是空的，所以可以根据这个条件过滤掉

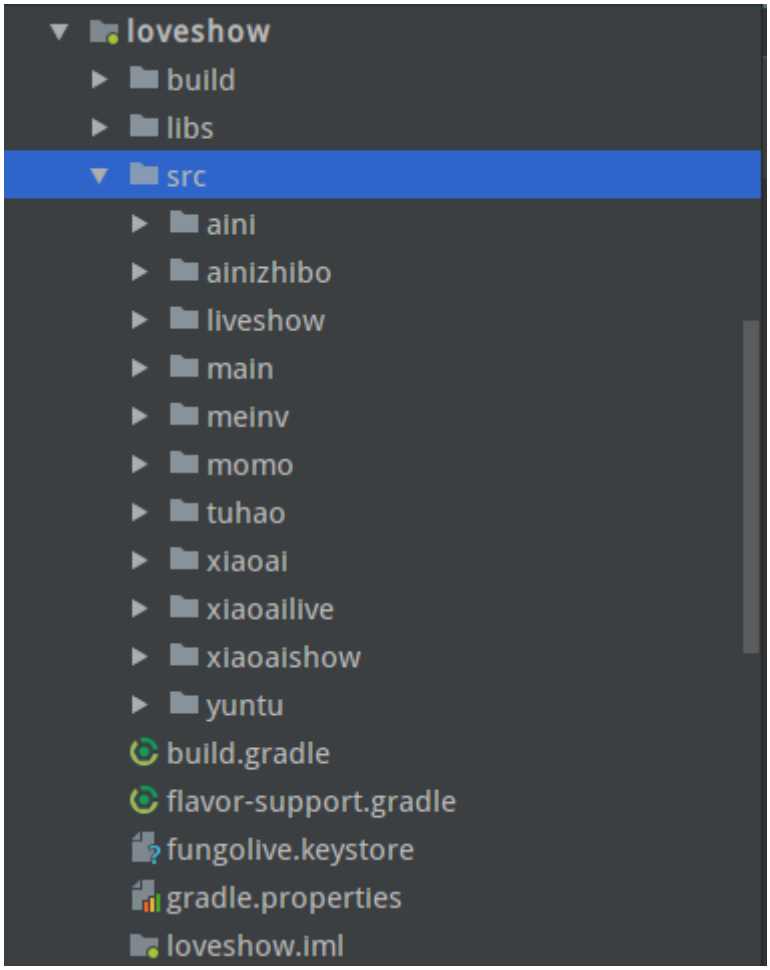
```
pom.withXml {
    def dependenciesNode = asNode().appendNode('dependencies')
    depList.values().each {
        ResolvedDependency dep ->
            def hasGroup = dep.moduleGroup != null
            def hasName = (dep.moduleName != null && !"unspecified".equals(dep.moduleName) && !"".equals(dep.moduleVersion))
            def hasVersion = (dep.moduleVersion != null && !"".equals(dep.moduleVersion) && !"unspecified".equals(dep.moduleVersion))

            if (hasGroup && hasName && hasVersion) {
                def dependencyNode = dependenciesNode.appendNode('dependency')
                dependencyNode.appendNode('groupId', dep.moduleGroup)
                dependencyNode.appendNode('artifactId', dep.moduleName)
                dependencyNode.appendNode('version', dep.moduleVersion)
            }
    }
}
```

> 这个时候我们基本就已经实现了组件之间的灵活组合合并的问题了。使用命令
>
> ```
> gradle clean asR
> ```
>
> 合并打包之后，发布到远程仓库即可



差异化资源的实现

> 现在有这种需求，云图需要在直播间加上一个领取爱钻的控件，但是又不能影响主版本，刚开始想在代码中作判定，但是后来想想如果后续合作的SDK都是如此，那代码得混成什么样？于是放弃了这个念头，但是仔细想想看，我们是否可以学习多马甲打包的思想，来实现呢？



> 其实实现思路比较简单，我们只需要记录扫描的资源文件名缓存起来，如果跟我们新增的资源，名字相同的时候，使用我们最新的资源即可。

```
task embedLibraryResources << {
    println "Running FAT-AAR Task :embedLibraryResources"

    def oldInputResourceSet = packageReleaseResources.inputResourceSets
    packageReleaseResources.conventionMapping.map("inputResourceSets") {
        getMergedInputResourceSets(oldInputResourceSet)
    }
}

/**
 * 本地资源
 * 由于合并module的时候
 * 资源文件无法替换
 */

private List getMergedInputResourceSets(List inputResourceSet) {
    //We need to do this trickery here since the class declared here and that used by the runtime
    //are different and results in class cast error
    def ResourceSetClass = inputResourceSet.get(0).class

    List newInputResourceSet = new ArrayList(inputResourceSet)

    HashMap<String, String> localRes = new HashMap<String, String>()
    String path = buildDir.path.replace("/build", "") + "/src/main/res"
    FileTree moduleRes = fileTree(dir: path, include: '**/*.xml')
    moduleRes.each { File file ->
        println("路径=" + file.getAbsolutePath() + "/" + file.getName())
        localRes.put(file.getName(), file.getAbsolutePath())
    }
```

```
FileTree libRes = fileTree(dir: "$aarPath/res", include: '**/*.xml')
            libRes.each { File file ->
                if (localRes.containsKey(file.getName())) {
```

```
                println("资源重复" + file.getAbsolutePath())
                file.delete()
            }
        }
```

```
def getArtifactFileName() {
    return "${POM_ARTIFACT_ID}-${VERSION_NAME}.aar"
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:25.3.1'
    testCompile 'junit:junit:4.12'
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })

    if (isModule.toBoolean()) {
        compile project(':lib-common')
        compile project(':lib-share')
        compile project(':app')
        compile project(':lib-video-play')
    } else {
        embedded project(':lib-common')
        //embedded project(':lib-share')
        embedded project(':app')
        embedded project(':lib-video-play')
    }

}
```

1.在gradle:2.2.3版本以上，合并出来的aar包，没有包含R文件相关的jar
2.本地合并的aar包，并没有包含远程依赖的一些库
3.如果组件中有第三方的aar包的时候，生成pom.xml需要过滤

> 到此，我们多组件合并的坑基本已经走的差不多了，当主版本业务更新了的时候，我们只需要根据下面的配置，来决定我们要合并哪几个组件，并发布到maven上去，然后让云图的同事更新下相关的版本号即可。基本告别了之前的多分支，代码迁移带来的诸多问题。

小结

> 第三方的轮子在我们看来学习的更多的是设计思想，但是并不是所有的轮子拿来都可以直接适用我们的业务的，因此在使用这些轮子的时候，最好还是知其所以然，基于现有的业务加以调整，或许能更适合我们的业务变迁。

纸上得来终觉浅，绝知此事要躬行！

谢谢大家