Android Gradle详细讲解

文档变更记录：

|  |  |  |
| --- | --- | --- |
| 变更日期 | 变更原因 | 记录人 |
| 2018年9月19日星期三 | 创建 | LC |
| 2019年4月22日星期一 | 添加解决Android Studio文件同步错误的办法 | LC |

文档说明：

鉴于Android Gradle规则比较多，网络上资料比较分散，Android应用的构建完全依赖于Gradle，遇到问题时常束手无措，遂整理出此文档以供参考。

目录

**[1. 前言](#_Toc18828_WPSOffice_Level1)** **[1](#_Toc18828_WPSOffice_Level1)**

[1.1. Gradle简介](#_Toc7605_WPSOffice_Level2) [1](#_Toc7605_WPSOffice_Level2)

[1.2. Android Gradle简介](#_Toc7732_WPSOffice_Level2) [3](#_Toc7732_WPSOffice_Level2)

**[2. 关于GradleWrapper和gradle版本](#_Toc7605_WPSOffice_Level1)** **[3](#_Toc7605_WPSOffice_Level1)**

[2.1. 其他](#_Toc389_WPSOffice_Level2) [4](#_Toc389_WPSOffice_Level2)

**[3. 常见配置](#_Toc7732_WPSOffice_Level1)** **[4](#_Toc7732_WPSOffice_Level1)**

[3.1. 常用配置](#_Toc18196_WPSOffice_Level2) [4](#_Toc18196_WPSOffice_Level2)

**[4. 关于Local.properties](#_Toc389_WPSOffice_Level1)** **[5](#_Toc389_WPSOffice_Level1)**

**[5. 关于defaultConfig](#_Toc18196_WPSOffice_Level1)** **[6](#_Toc18196_WPSOffice_Level1)**

**[6. 关于ProductFlavor](#_Toc3596_WPSOffice_Level1)** **[6](#_Toc3596_WPSOffice_Level1)**

[6.1. defaultConfig](#_Toc3596_WPSOffice_Level2) [6](#_Toc3596_WPSOffice_Level2)

[6.2. 多apk生成](#_Toc32680_WPSOffice_Level2) [6](#_Toc32680_WPSOffice_Level2)

**[7. 关于apk自定义命名](#_Toc32680_WPSOffice_Level1)** **[7](#_Toc32680_WPSOffice_Level1)**

[7.1. 多渠道打包](#_Toc28414_WPSOffice_Level2) [7](#_Toc28414_WPSOffice_Level2)

[7.2. 动态引入组件](#_Toc19822_WPSOffice_Level2) [7](#_Toc19822_WPSOffice_Level2)

**[8. 关于依赖](#_Toc28414_WPSOffice_Level1)** **[8](#_Toc28414_WPSOffice_Level1)**

[8.1. 动态添加依赖](#_Toc500_WPSOffice_Level2) [8](#_Toc500_WPSOffice_Level2)

**[9. SourceSet](#_Toc19822_WPSOffice_Level1)** **[10](#_Toc19822_WPSOffice_Level1)**

**[10. 其他](#_Toc500_WPSOffice_Level1)** **[11](#_Toc500_WPSOffice_Level1)**

[10.1. BuildConfig](#_Toc13650_WPSOffice_Level2) [11](#_Toc13650_WPSOffice_Level2)

[10.2. applicationId和package属性值的关系](#_Toc10007_WPSOffice_Level2) [11](#_Toc10007_WPSOffice_Level2)

**[11. 关于Build](#_Toc13650_WPSOffice_Level1)** **[11](#_Toc13650_WPSOffice_Level1)**

[11.1. 关于build慢的问题](#_Toc29209_WPSOffice_Level2) [11](#_Toc29209_WPSOffice_Level2)

[11.2. 参考Build步骤](#_Toc9300_WPSOffice_Level2) [12](#_Toc9300_WPSOffice_Level2)

[11.3. 解决依赖库版本冲突](#_Toc14156_WPSOffice_Level2) [12](#_Toc14156_WPSOffice_Level2)

**[12. 常见错误与参考解决办法](#_Toc10007_WPSOffice_Level1)** **[12](#_Toc10007_WPSOffice_Level1)**

[12.1. 找不到原因的错误](#_Toc5911_WPSOffice_Level2) [12](#_Toc5911_WPSOffice_Level2)

[12.2. 同步失败](#_Toc11407_WPSOffice_Level2) [12](#_Toc11407_WPSOffice_Level2)

[12.3. Java文件出现棕红色时钟，文件无法跳转](#_Toc4305_WPSOffice_Level2) [12](#_Toc4305_WPSOffice_Level2)

[12.4. 禁止使用动态版本依赖](#_Toc27769_WPSOffice_Level2) [13](#_Toc27769_WPSOffice_Level2)

[12.5. 找不到库文件](#_Toc7396_WPSOffice_Level2) [13](#_Toc7396_WPSOffice_Level2)

[12.6. 奇怪错误1](#_Toc27502_WPSOffice_Level2) [13](#_Toc27502_WPSOffice_Level2)

[12.7. BuildTools版本问题](#_Toc8864_WPSOffice_Level2) [13](#_Toc8864_WPSOffice_Level2)

[12.8. 依赖库版本不一致问题](#_Toc3595_WPSOffice_Level2) [13](#_Toc3595_WPSOffice_Level2)

[12.9. appt错误](#_Toc27767_WPSOffice_Level2) [14](#_Toc27767_WPSOffice_Level2)

**[参考网：](#_Toc29209_WPSOffice_Level1)** **[16](#_Toc29209_WPSOffice_Level1)**

**[参考书籍：](#_Toc9300_WPSOffice_Level1)** **[16](#_Toc9300_WPSOffice_Level1)**

**[1、 Android Gradle 权威指南.pdf](#_Toc14156_WPSOffice_Level1)** **[16](#_Toc14156_WPSOffice_Level1)**

# 前言

## Gradle简介

Gradle本身提供一些基本的概念和整体核心的框架，其他用于描述真实使用场景逻辑的都以插件扩展的方式来实现。比如构建Java应用，就是通过Gradle内置的Java插件来实现的，AndroidGradle插件就是基于其内置Java插件实现的。

把插件应用到项目中，会帮助你在项目的构建过程中做很多事情，比如：测试、编译、打包；可以添加以来配置到项目中；也可以对项目进行一些约定，比如应用Java插件之后，约定src/main/java目录下是我们源代码存放的位置，在编译的时候也是编译这个目录下的Java源文件。我们只需要安装插件约定的方式，使用其提供的任务、方法或扩展，就可以对我们的项目进行构建。（参考[链接](https://blog.csdn.net/qq_33689414/article/details/53152212)）

### 关于task

一个Project包含多个task，一个Task就是一个原子性的操作，比如一次编译或一次打包。Android Gradle插件基本包含了所有的Java插件功能，比如assemble、check、build等，此外，还添加了connectedCheck、deviceCheck、lint、install和uninstall等任务。一般我们常用的任务是build，assemble, clan，lint，check等，这些任务可以打包生成apk。

### 关于插件

#### 插件功能

阿斯蒂芬

#### 引入插件

引入插件是通过Project.apply()方式完成的，apply方法有好几种用法，且插件分为二进制和脚本插件。二进制插件应用方式如代码1所示。

apply **plugin**: **'com.google.gms.google-services'** *// 1)*apply **plugin**: **'com.android.application'** *// 2)*apply **plugin**: org.gradle.api.plugins.JavaPlugin *// 3)*apply **plugin**: JavaPlugin *// 4)*apply **plugin**: **'java'** *// 5)*apply **from**: **'version.gradle'** *// 6)*apply{ *// 7)* plugin **'java'**}

代码 1

在代码1中，第5行中的’java’是Java插件的plugin id，其对应的类型是第3行，包ort.gradle.api.plugins是默认导入的，所以第3、4、5、7行代码等价，是引入Java插件的四种写法，且都是二进制引入。第6行代码是脚本插件的引入方式，其余都是二进制引入方式。

应用第三方插件，必须在buildscript{}里配置其classpath才能使用，比如Android Gradle插件就属于Android发布的第三方插件，可以在项目工程根目录下的build.gradle里面配置（为各个子工程共有，如此，子工程便不需各自重复配置），也可以在各个子工程里面配置，如代码2所示。

buildscript {  
 repositories {  
 google()  
 jcenter()  
 }  
 dependencies {  
 classpath **'com.android.tools.build:gradle:3.2.0-alpha08'** classpath **'com.google.gms:google-services:3.2.0'**

其余代码省略。。。

代码 2

在代码2中，buildscript{}块是一个在构建项目之前，为项目进行前期准备和初始化相关配置依赖的地方，配置好所需的依赖，就可以应用插件了（见代码块1，第2行）。classpath的内容格式为：group:name:version，即用冒号隔开。

### Java Gradle插件简介

爱迪生发

## Android Gradle简介

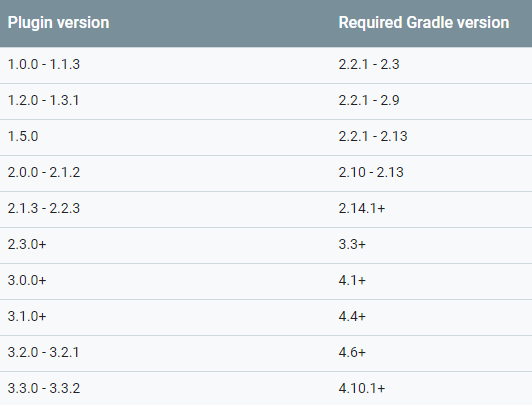
Android插件基于Java插件，即包含了Java插件的所有功能。配置块android{}就是Android Gradle插件为Project对象添加的一个扩展，所有的Android Gradle工程的配置都在android{}这个唯一入口中。每个项目至少有2个build.gradle文件，一个是项目级的，另一个是模块级的。

Android工程Gradle 配置文件包含如下：

|  |  |
| --- | --- |
| 名称 | 简介 |
| build.gradle(工程级) | 位于项目根目录下，于定义适用于项目中所有模块的构建配置。  例如指定Gradle插件版本：  classpath **'com.android.tools.build:gradle:3.2.0'**  配置各个子模块共用版本号：  project.ext {   MINIMUM\_SDK\_VERSION = 14  TARGET\_SDK\_VERSION = 27  TOOLS\_VERSION = **"27.0.3"** SUPPORT\_VERSION = **"27.0.2"** } |
| Build.gradle（模块级） |  |
| Gradle.properties | Gradle运行环境配置文件。可配置java运行环境：  **org.gradle.jvmargs**=**-Xmx1536m** |
| Gradle-wrapper.properties |  |
| Local.properties | 为构建系统配置本地环境属性，例如 SDK 安装路径。（不应手动修改该文件，不需要添加到git版本库）  也可配置一些本地化变量（[后续详细讲解](#_关于Local.properties)）。 |
| Progard-rules.pro | 定义混淆规则 |
| Setting.gradle | 位于项目根目录下，用于指示gradle在build应用的时候需要将哪些模块包括在内。例如：include ‘:app’。 |
| Gradlew.bat | 自动完成gradle的Windows脚本 |

# 关于GradleWrapper和gradle版本

Gradle版本问题导致无法编译时常发生，在配置版本的时候，需要对照gradle和Gradle插件对应表，如下所示：



例如设置gradle版本为4.6，设置插件版本为3.2.0：

**distributionUrl**=**https\://services.gradle.org/distributions/gradle-4.6-all.zip**

classpath **'com.android.tools.build:gradle:3.2.0'**

## 其他

# 常见配置

## 常用配置

|  |  |
| --- | --- |
| 属性 | 简介 |
| BuildToolVersion | Android构建工具的版本。可以在Android SDK目录里看到，是一个工具包，包括appt，dex等工具。  Android Gradle插件各个版本各自有一个最低Android SDK Build Tool 的版本要求。 |
| compileSdkVersion | 不会影响运行时行为，建议用最新sdk编译。必须大于support library的版本号。 |
| targetSdkVersion | 决定了系统api的行为，即只要targetSdkVersion保持不变，在任何版本的系统上，行为都不会改变。 |
| minSdkVersion | App最低的系统版本要求，大于等于所依赖库的minSdkVersion。 |

# 关于Local.properties

此文件在Android Studio中是用来配置SDK目录的,也可以在文件中配置一些本地化的变量。这个文件不会被提交到版本控制中。为了安全起见签名文件不可以放在项目中，那么可能每个开发人员的存放地方不同，这种情况下就可以使用local.properties来配置路径然后在app目录下的build.gradle中引用，示例如下：

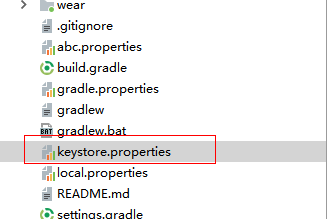
在local.properties配置如下，其中sd.dir是自动生成的，无需改动：

**sdk.dir**=**C\:\\Users\\Administrator\\AppData\\Local\\Android\\Sdk  
RELEASE\_KEY\_FILE\_PATH**=**E\:\\**

在build.gradle中配置如下，通过java的Properties对象读取local.properties的键值映射内容：

signingConfigs{  
 autoSigning{  
 **def** properties = **new** Properties()  
 **def** inputStream = project.rootProject.file(**"local.properties"**).newDataInputStream();  
 properties.load(inputStream)  
  
 keyAlias **'myAlias'** keyPassword **'myPwd'** storeFile file(properties.getProperty(**'RELEASE\_KEY\_FILE\_PATH'**))  
 storePassword **'myStorePwd'** }  
}

亦或新建一个keystore.properties文件如下：



在其中定义一些本地配置：

**storeFile**=**keystore.jks  
storePassword**=**myStorePassword  
keyAlias**=**myKeyAlias  
keyPassword**=**mykeyPassword**

# 关于defaultConfig

属性如下：

# 关于ProductFlavor

属性如下：

|  |  |
| --- | --- |
| 属性 | 简介 |
| [dimension](http://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.ProductFlavor.html" \l "com.android.build.gradle.internal.dsl.ProductFlavor:dimension) |  |

## defaultConfig

DefaultConfig是一个ProductFlavor，负责定义所有的默认配置，如果一个ProductFlavor没有被特殊定义配置的话，默认就会使用defaultConfig{}块指定的配置，比如包名、版本号、版本名称等，属性如下：

|  |  |
| --- | --- |
| 属性 | 简介 |
| [applicationId](http://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.ProductFlavor.html" \l "com.android.build.gradle.internal.dsl.ProductFlavor:applicationId) |  |
| [versionCode](http://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.ProductFlavor.html" \l "com.android.build.gradle.internal.dsl.ProductFlavor:versionCode) | 表明app内部版本号，用于控制app升级。 |
| [versionName](http://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.ProductFlavor.html" \l "com.android.build.gradle.internal.dsl.ProductFlavor:versionName) | 表明app应用的版本名称。 |
| minSdkVersion | 指定app最低支持的Android操作系统版本 |
| targetSdkVersion |  |
| SigingConfig | 配置默认签名信息 |
| ProguardFile | 配置混淆文件 |

## 多apk生成

Android Gradle支持基于屏幕分辨率、Android版本、设备类型（平板或手机）、设备硬件（是否有摄像头）等生成多个apk文件，而发布的时候这些apk共享一个应用ID和下载入口，如此可以缩小apk大小。

# 关于apk自定义命名

有时候会遇到“The APK file does not exist on disk”的错误，导致apk无法安装。原因在于命名规则中包含动态部分，由于Gradle在执行编译命令和安装命令有时间差，导致编译出来的apk名称和安装时获取到的apk名称不一致，所以它就报找不到指定的apk文件了。那么应该这样命名呢？如代码3所示（Gradle插件3.2以下略有不同）。

android.applicationVariants.all { variant ->  
 variant.outputs.each {  
 **if** (variant.buildType.name.equals(**"release"**)) {  
 variant.outputs.all {  
 **def** application = **"PowerSecurity"  
 def** versionCode = variant.getVersionCode()  
 **def** versionName = variant.getVersionName()  
 **def** channel = variant.productFlavors[0].manifestPlaceholders.get(**"CHANNEL\_VALUE"**)  
 **def** newName = **"**${application}**\_**${versionName}**\_**${versionCode}**\_**${channel}**.apk"** outputFileName = newName;  
*// }* }  
 }  
 }  
}

代码3列出了导出的aar包的命名方法，需要注意的是命名代码块必须放在android{}下面，不能放在buildTypes{}或其他子配置里面。如果需要对app包命名则把”libraryVariants”改为applicationVariants即可。

## 多渠道打包

Android Gradle中，定义了一个叫做Build Variant的概念，一个Build Variant = Build Type（Release或Debug） + Product Flavor（各种渠道，例如google、badu等），参考[链接](https://www.cnblogs.com/travellife/p/Gradle-shi-xian-Android-duo-qu-dao-ding-zhi-hua-da.html)， 参考[链接](https://www.cnblogs.com/youxilua/archive/2013/05/20/3087935.html)，

## 动态引入组件

参考代码如下：

**def** allComponents = [**"app"**, **"facebook"**]  
allComponents.forEach(  
 { name ->  
 **if** (name.equals(**""**)){  
 include **":**${name}**"** project(**":**${name}**"**).projectDir == getComponentDir(name)  
 }  
 }  
)

# 关于依赖

依赖分为三种类型模块依赖、远程二进制依赖、本地二进制依赖。查看所有依赖命令：

Gradlew androidDependencies

查看某个子模块（例如app-dir）的依赖：

Gradlew :app-dir:denpendencies

模块依赖，例如compile project(':mylibrary')；远程二进制依赖，例如compile 'com.android.support:appcompat-v7:27.1.1'；本地二进制依赖，例如compile fileTree(dir: 'libs', include: ['\*.jar'])，如下：

repositories {  
 flatDir {  
 dirs **'libs'** *// 本地库目录，'以build.gradle所在目录为根目录的相对路径'* }  
 maven { url **"https://s3.amazonaws.com/moat-sdk-builds"** }  
 ivy{  
 url **"xxx"** }  
 maven { url **'https://maven.google.com'** }  
}

## 动态添加依赖

经常由于所依赖的版本不一致，发生各种编译或者运行时错误，以com.android.support库为例，有两个办法，办法一如下所示，在app模块的build.config的最外层添加如下代码，指定所有版本为26.0.0：

configurations.all {  
 resolutionStrategy {  
 eachDependency { details ->  
 *// Force all of the primary support libraries to use the same version.* **if** (details.requested.group == **'com.android.support'**) {  
 details.useVersion **"26.0.0"** }  
 }  
 }  
}

指定版本为2.1.2。

configurations.all {  
 resolutionStrategy {  
 eachDependency { details ->  
 *// Force all the error-prone dependencies to use the same version.* **if** (details.requested.group == **'com.google.errorprone'** &&  
 details.requested.name.startsWith(**'error\_prone\_'**)) {  
 details.useVersion **'2.1.2'** }  
 }  
 }  
}

办法二如下，在添加依赖的时候分别添加版本号变量：

compile **"com.android.support:percent:**$support\_version**"**

其中support\_version变量定义在工程的build.config中buildscript的最顶层：

buildscript {  
 ext.support\_version = **'27.1.0'** repositories {

省略...

参考代码如下：

**boolean** useSource = gradle.startParameter.taskNames.any {  
 it.contains(**"assemble"**) || it.contains(**"install"**) || it.contains(**"upload"**)  
}  
subProject.forEach { subProject ->  
 **if** (useSource) {  
 android.sourceSets.**main** {  
 java.srcDirs += file(**"../**${subProject}**/src/main/java"**)  
 res.srcDirs += file(**"../**${subProject}**/src/main/res"**)  
 }  
 } **else** {  
 dependencies { implementation project(**":**$subProject**"**) }  
 }  
}

# SourceSet

sourceSet可以为不同的编译包指定不同的源文件（包括manifest、java、res等）。示例如下指定不同的manifest文件：

sourceSets{  
 main{  
 **if** (**isLibrary**.toBoolean()){  
 **manifest**.srcFile **"src/main/AndroidManifest.xml"** }**else** {  
 **manifest**.srcFile **"src/main/AndroidManifest.xml"** }  
 }  
 google{  
 **manifest**.srcFile **"src/google/AndroidManifest.xml"** }  
 baidu{  
 **manifest**.srcFile **"src/baidu/AndroidManifest.xml"** }  
}  
flavorDimensions **"channel"**productFlavors{  
 google{  
 dimension **"channel"** }  
 baidu{  
 dimension **"channel"** }  
}

# 其他

## BuildConfig

Android Gradle提供了buildConfigField(String type, String name, String value)让我们可以添加自己定义的常量到BuildConfig中，可以为不同的渠道或者不同构建类型在BuildConfig中定义不同的值。

## applicationId和package属性值的关系

App模块Build.gradle中的applicationId是指应用的id，package仅仅只是资源路径和源码目录。

# 关于Build

## 关于build慢的问题

### 加快DEX生成速度

Android Studio提供dexOption区块以便于我们配置DEX构建属性，加快DEX文件的生成速度，代码如下：

dexOptions {

preDexLibraries true

maxProcessCount 8

javaMaxHeapSize "2048m"

}

preDexLibraries声明是否预先编译依赖库，从而加快构建速度，实质是通过延时清除已生成的依赖库的构建文件，从而提高构建速度，根据使用情况合理配置。maxProcessCount设置进程运行过程中可以使用的最大线程数。默认值为4。javaMaxHeapSize设置DEX编译器的最大堆大小，堆或者栈都是用于存放暂时不用的垃圾，当内存不足时，垃圾回收机制会清除过时的缓存，堆大小决定垃圾清除的频率，影响着构建的速度。设置如下：

**org.gradle.jvmargs**=**-Xmx4608M**

## 参考Build步骤

建议用gradlew命名完成整个build任务（也可配置gradle-2.14.1的环境变量）。

## 解决依赖库版本冲突

强制设置依赖库版本：

configurations.all{  
 resolveStrategy{  
 force **'org.hamcrest:hamcrest-core:1.3'** }  
}

查看依赖库命令：gradlew -q app:dependencies

# 常见错误与参考解决办法

## 找不到原因的错误

有时候会遇到很奇怪的错误提示，或者没有提示（Run Configurations里面找不到Module），导致当前工程无法完成编译。可以尝试的解决办法是手动删除当前的build下的编译文件，随后执行“Invalid cache/Restart”，还是无法编译则修改gradle和gradle-wrapper版本。

## 同步失败

错误信息：

Unfortunately you can't have non-Gradle Java modules and Android-Gradle modules in one project

建议尝试以下方式解决：

## Java文件出现棕红色时钟，文件无法跳转

在打开工程的时候，停在Indexing过程中，文件无法跳转，并出现如下提示：

Argument for @NotNull parameter 'message' of com/android/tools/idea/gradle/project/sync/GradleSyncState.syncFailed must not be null

建议尝试以下方式解决此问题：

* 点击File->sync Project width gradle Files和File->Sync width File System.
* 或者点击File->invalidate Cache/System.
* 或者重命名当前工程文件的父目录，然后用Android studio再次打开此工程，或许能解决此问题。

## 禁止使用动态版本依赖

相关网址[链接](https://blog.danlew.net/2015/09/09/dont-use-dynamic-versions-for-your-dependencies/)。动态版本依赖可能会导致库的行为不可控（即会出现各种奇奇怪怪的编译或运行错误），尽量用如下形式依赖，确保依赖版本的统一性和确定性：

api **"com.android.support:appcompat-v7:**$project.SUPPORT\_VERSION**"**

## 找不到库文件

添加仓库或者更换翻墙工具可以解决问题。

## 奇怪错误1

错误提示：<item name.....

解决办法：删除manifest里面的所有注释。

## BuildTools版本问题

错误日志如下：

E/AndroidRuntime: FATAL EXCEPTION: main

java.lang.NoSuchMethodError: No static method getFont(Landroid/content/Context;ILandroid/util/TypedValue;ILandroid/widget/TextView;)Landroid/graphics/Typeface; in class Landroid/support/v4/content/res/ResourcesCompat; or its super classes (declaration of 'android.support.v4.content.res.ResourcesCompat' appears in

解决办法：

把compileSdkVersion和buildToolsVersion改为和所有的“com.android.support:\*”版本改为一致。即如果support版本为27，那么compileSdkVersion和buildToolsVersion版本都必须为27。

## 依赖库版本不一致问题

提示错误：

# [Duplicate zip entry [classes.jar:android/support/design/widget/CoordinatorLayout$Behavior.class]](https://stackoverflow.com/questions/49044782/duplicate-zip-entry-classes-jarandroid-support-design-widget-coordinatorlayout)

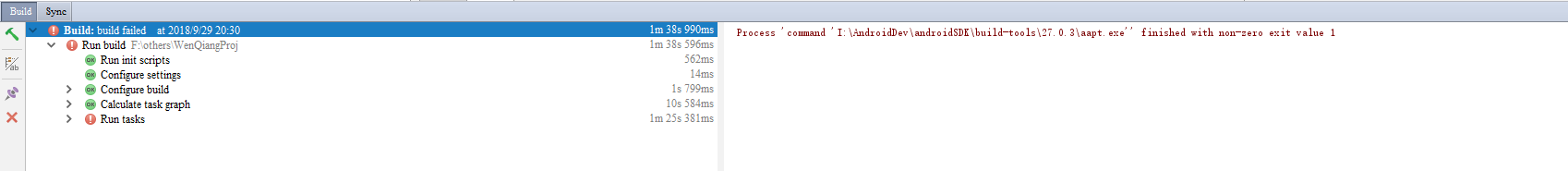
解决步骤:

* 运行命令查看依赖版本库：gradlew -q :app:dependencies > dependencies.txt
* 用ctrl+F查找dependencies.txt里面的各个support版本信息情况。
* 把所有的support版本改为->指向的版本（参考[8.1](#_动态添加依赖)），且按照[12.7](#_BuildTools版本问题)的要求把BuildToolsVersion改为一致，就能编译通过了。

## appt错误

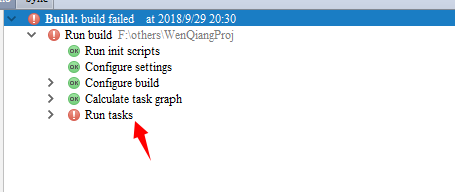
有时候，突然出现appt错误（资源文件错误），错误详情，如下：

Process 'command 'I:\AndroidDev\androidSDK\build-tools\27.0.3\aapt.exe'' finished with non-zero exit value 1

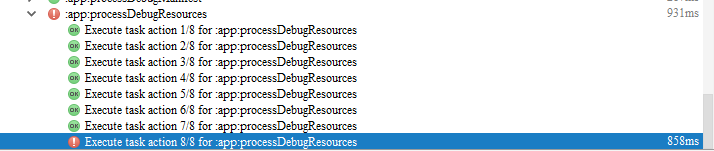


出现此问题有很多原因，参考解决过程如下：

点击如下红色警告：



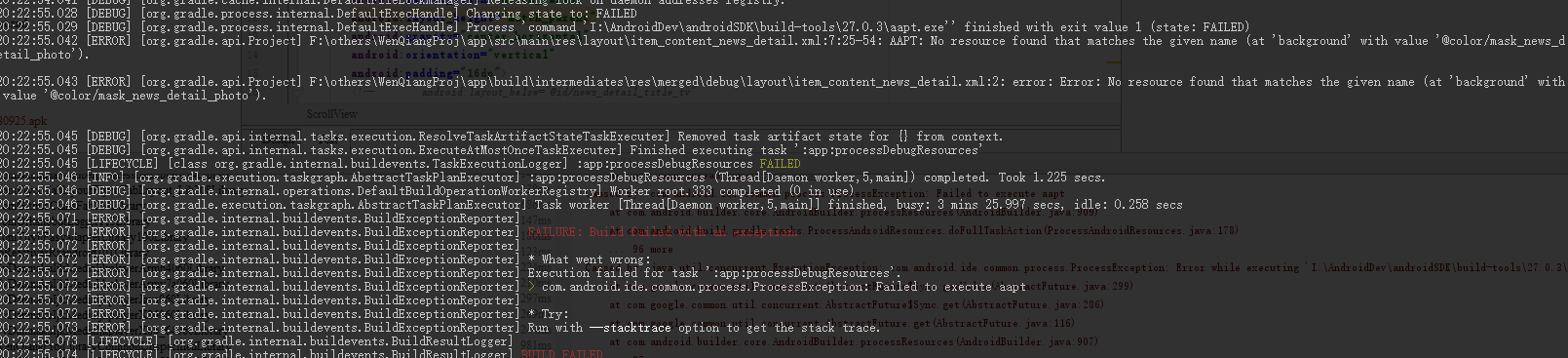
展开如下所示：



可知道是在执行任务processDebugResources出现的错误，在控制台下，转到项目的根目录，执行如下命令：



得到错误的详细信息如下所示：



根据上图错误提示，修改即可。

附录1

参考网：

1. Android Plugin DSL Reference：<http://google.github.io/android-gradle-dsl/current/index.html>
2. ProductFlavor：

<http://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.ProductFlavor.html>

1. gradle.properties：<https://docs.gradle.org/current/userguide/build_environment.html>
2. 中文官方：<http://ask.android-studio.org/?/article/7>

参考书籍：

1. Android Gradle 权威指南.pdf