浅析总结 AS 中 Gradle 配置运行

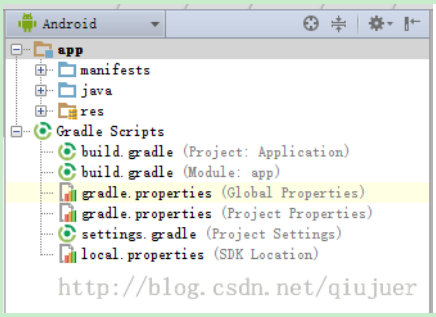
Gradle

Gradle 是以 Groovy 语言为基础，面向Java应用为主。基于DSL（领域特定语言）语法的自动化构建工具。

依赖管理

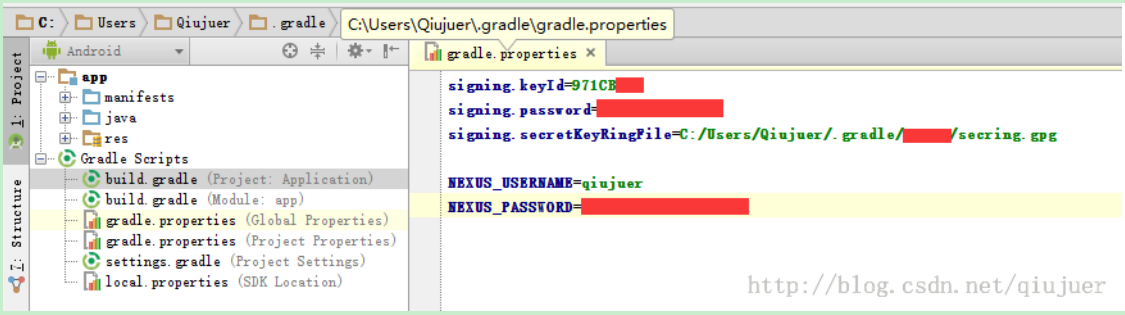
支持多方式依赖管理：包括从 maven 远程仓库、 nexus 私服、 ivy 仓库以及本地文件系统的 jars 或者 dirs 。这也是我最喜欢的地方，操作简单。

新项目



一个新的项目中就包含这些文件，build 是两个，一个项目一个是 APP Model 的。另外在 APP 中可以看见有一个 manifest 文件夹，这意味着着可以有多 AndroidManifest 文件。

另外值得一说的是 gradle.properties 文件也是含有两个，但是却是一个是全局，一个是项目的；这与上面的 Build 文件有何区别？区别在于全局文件存在于 C:\Users\用户名\.gradle 文件夹中，该文件有可能没有，需要自己创建，创建后所有项目都将具有访问权限，在该文件中一般保存的是项目的一些变量等，如果是无关紧要的变量可以保存在项目文件中，如果是用户名密码等变量则需要保存在全局文件中。



至于项目的配置文件一般是空的。

local.properties

[java] view plaincopy

```
## This file is automatically generated by Android Studio.
# Do not modify this file -- YOUR CHANGES WILL BE ERASED!
#
# This file should *NOT* be checked into Version Control Systems,
# as it contains information specific to your local configuration.
#
# Location of the SDK. This is only used by Gradle.
# For customization when using a Version Control System, please read the
# header note.
sdk.dir=D\:\\ToolKits\\Android\\sdk
```

其中包含了你的 sdk 配置，当然你还可以配置 ndk 路径；格式与 sdk 一样。

settings.gradle

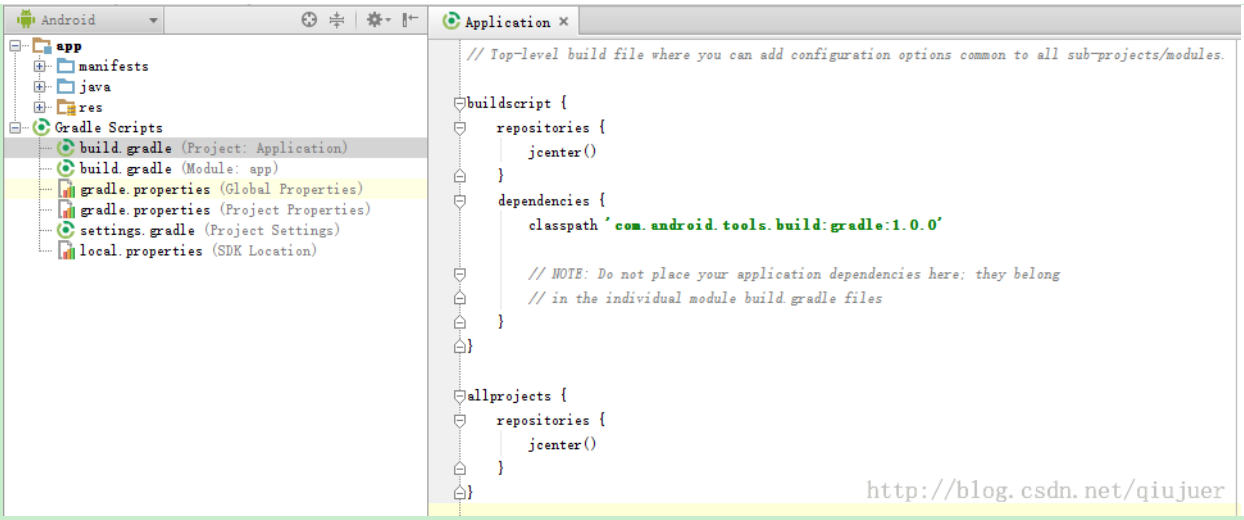
[java] view plaincopy

```
include ':app'
```

该文件中就仅仅只包含了一句话，在你的项目中如果有多个 Model 存在的时候，就可以选择包含哪些进行编译。

build.gradle

项目：



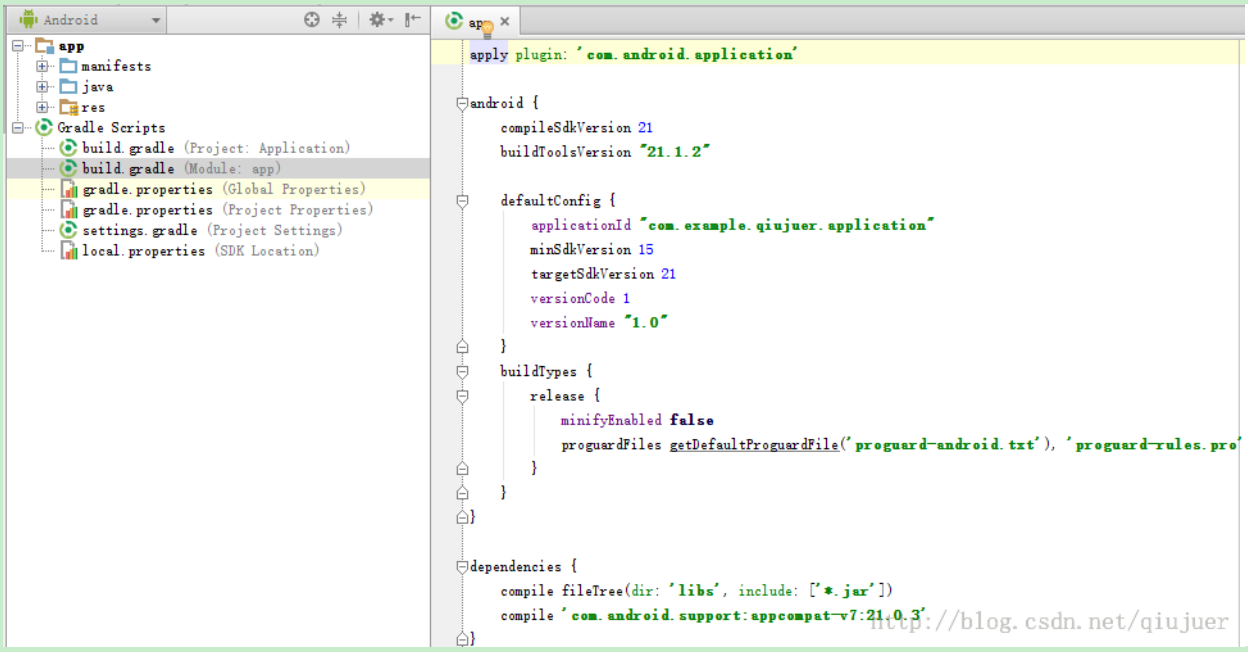
两个大的包围一看就明了，一个是为编译准备的，一个是为所有项目准备的。

其中，Repositories 配置的是上面所说的依赖管理的东西，也就是依赖管理的服务器。默认是 jcenter() 你可以添加其他，多个之间不干扰。

dependencies 这个也是依赖管理的东西，上面是指定依赖管理的服务器，这个就是具体依赖什么库。

联合起来也就是，依赖 jcenter() 服务中的 gradle 库，其包名是：“com.android.tools.build”，版本是：1.0.0 版本。

APP Model



在这个中基本可以按照名称来知晓其作用。

第一行：

[java] view plaincopy

```
1. apply plugin: 'com.android.application'
```

表示的是添加插件，其是可以理解为该 model 为一个 com.android.application 程序，也就是应用程序，如果你的 Model 是一个库，那么自然也就是：

[java] view plaincopy

```
1. apply plugin: 'com.android.library'
```

dependencies：

这个也就是所谓的依赖了，在这里不光可以进行远程依赖（上面所说的方法），也可以本地依赖：

[java] view plaincopy

```
1. compile fileTree(include: ['*.jar'], dir: 'libs')
```

这句话也就是说编译时依赖 libs 文件夹下的所有 jar 文件。

[java] view plaincopy

```
1. compile project(':library')
```

这样一句话是什么意思？这也是依赖，不过依赖的是一个 model，前面说了在一个项目中可以有多个 model，这句话的意思也就是依赖一个本项目中 名称为 library 的 model 库。

[java] view plaincopy

```
compile 'com.android.support:appcompat-v7:21.0.3'
```

至于这句话也就是依赖一个远程的库了，这个库的作用是在低版本中使用一定的 Material Design 的东西。

其他一些介绍我以前发过文章可以看看，包括依赖 JNI 本地 aar 等等：

- [Android][Android Studio] Gradle项目中添加JNI生成文件(.so文件)
- [Android][Android Studio] *.jar 与 *.aar 的生成与*.aar导入项目方法

详细说说 android 部分：

先来看看基本完整的一个：

[java] view plaincopy

```
android {
    compileSdkVersion ANDROID_BUILD_TARGET_SDK_VERSION as int
    buildToolsVersion ANDROID_BUILD_TOOLS_VERSION

    defaultConfig {
    }

    buildTypes {
    }

    compileOptions {
    }

    sourceSets {
    }

    lintOptions {
    }

    productFlavors {
        flavor1 {
        }

        flavor2 {
        }
    }
    signingConfigs {
        release {
            storeFile file("x.keystore")
            storePassword "xxx"
            keyAlias "xxxx"
            keyPassword "xxx"
        }
    }
}
```

可以看见如果是完整的是有很多可以配置的地方；还让我一个个道来：

[java] view plaincopy

```
compileSdkVersion 21
buildToolsVersion "21.1.2"
```

这两个就是指定的编译SDK以及编辑工具版本，具体可以打开你的 SDK Manager 看看。

defaultConfig

这个自然就是默认配置了，既然是默认配置那么久相当于全局配置，也就是说这里边配置的下面的 buildTypes 中也将自动继承了。

在这个中可以放入很多的控制，如下面 buildTypes/release 中的配置你也可以放到其中：

[java] view plaincopy

```
defaultConfig {
    applicationId "com.example.qiujuer.application"
    minSdkVersion 15
    targetSdkVersion 21
    versionCode 1
    versionName "1.0"

    ndk {
        moduleName "genius"
        cFlags "-DANDROID_NDK -D_RELEASE"
        ldLibs "m", "log", "jnigraphics"
        abiFilters "all"
    }
}
```

在这里，首先进行了一个 applicationId 配置，该配置不是必须，但 库类型的 Model 将无此配置。

下面自然也就是 最小的SDK版本为 15，目标版本为：21 也就是说其中的代码你使用的全是API21中的 Android。再有就是当前的版本代码，版本名称，在 Eclipse 中这两个属性是在 AndroidManifest.xml 文件中，在这里把其提出来单独配置就是为了下面你可以在不同发布版本中配置不同的值。

至于这里的 ndk 部分，这个就是我额外加入的，其作用是可以直接编译 NDK 代码，不需要自己执行，具体详见：[Android] 环境配置之Android Studio开发NDK

buildTypes

在这里进行配置的是你的编译配置，可以看见这里有一个 release ，当然也就是有 debug 部分，两个部分配置都是一样。

在这里主要进行的配置是是否进行代码混淆，所以有一个代码混淆的开关，以及代码混淆的具体文件，文件有两种，无论哪种都行。

compileOptions

很多人或许不知道这个部分是干什么的，其是看看下面就明了了：

[java] view plaincopy

```
compileOptions {
    sourceCompatibility JavaVersion.VERSION_1_7
    targetCompatibility JavaVersion.VERSION_1_7
```

```
4. }
```

在这里你可以进行 Java 的版本配置，以便使用对应版本的一些新特性。

sourceSets

这个部分，看名字应该有个大概意思就是说源码设置，其是很多从 Eclipse 中迁移过来的代码，大部分中都将带有这个设置，因为 Eclipse的文件夹与 AS 不尽相同，所以需要手动指定。

[java] view plaincopy

```
1. sourceSets {
2.     main {
3.         manifest.srcFile 'AndroidManifest.xml'
4.         java.srcDirs = ['src']
5.         resources.srcDirs = ['src']
6.         aidl.srcDirs = ['src']
7.         renderscript.srcDirs = ['src']
8.         res.srcDirs = ['res']
9.         assets.srcDirs = ['assets']
10.        jniLibs.srcDirs = ['libs']
11.    }
12.
13. }
```

以上是一些常用的设置，其中最后一个是引用 *.so 文件的时候使用的方法。

lintOptions

这个其实应该写到最后的，因为这个是设置 编译的 lint 开关。

程序在buid的时候，会执行lint检查，有任何的错误或者警告提示，都会终止构建，我们可以将其关掉。

[java] view plaincopy

```
1. lintOptions {
2.     abortOnError false
3. }
```

productFlavors

在这里你可以设置你的产品发布的一些东西，比如你现在一共软件需要发布到不同渠道，且不同渠道中的包名不同，那么可以在此进行配置；甚至可以设置不同的 AndroidManifest.xml 文件。

[java] view plaincopy

```
1. productFlavors {
2.     flavor1 {
3.         packageName='com.example.qiujuer.application1'
4.         manifest.srcFile 'exampleapk/AndroidManifest1.xml'
5.     }
6.
7.     flavor2 {
8.         packageName='com.example.qiujuer.application2'
9.         manifest.srcFile 'exampleapk/AndroidManifest2.xml'
10.    }
11. }
```

不过，对于这个我并不常用，可以说基本没有用。

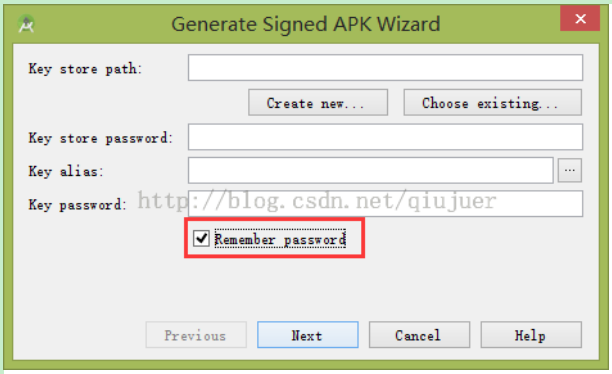
signingConfigs

这个相信大伙都知道吧，就是为包签名的配置，你可以设置具体的签名文件，签名密码等等：

[java] view plaincopy

```
1. signingConfigs {
2.     release {
3.         storeFile file("x.keystore")
4.         storePassword "xxx"
5.         keyAlias "xxxx"
6.         keyPassword "xxx"
7.     }
8. }
```

这个可以不用自己创建，你可以点击 build/generate signed apk，在其中选择你的文件或者创建签名文件，设置密码等等，然后选择记住密码，然后就会看见有这个配置了。



Case

一些常用的操作配置说完了，来说说，一个简单的小Case 。

在 APP Model build.gradle 文件根部我们加上：

[java] view plaincopy

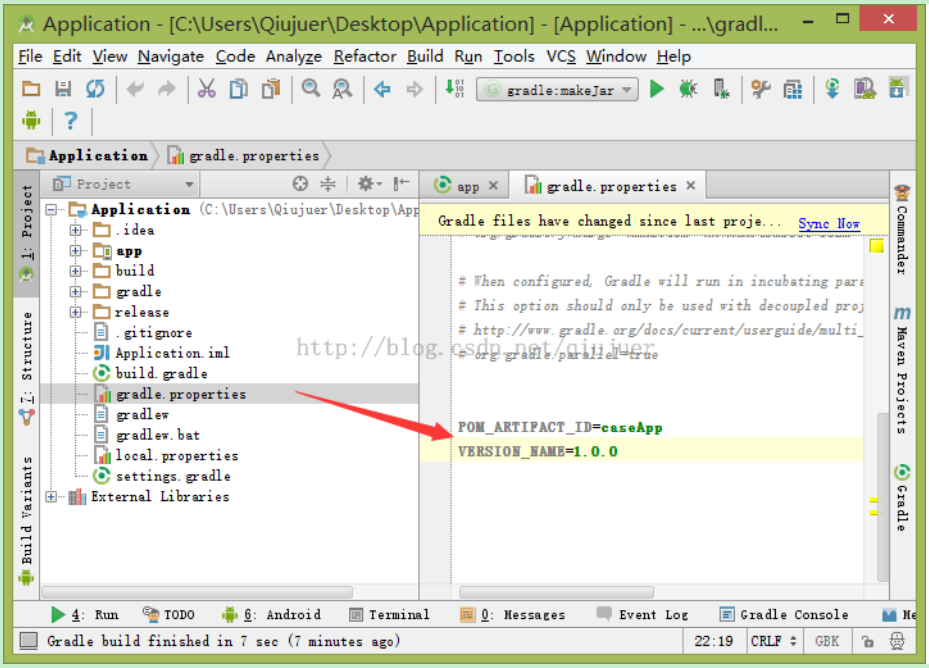
```
task clearApk(type: Delete) {
    delete '../release/' + POM_ARTIFACT_ID + '_' + VERSION_NAME + '.apk'
}

task makeApk(type: Copy) {
    from('build/outputs/apk/')
    into('../release/')
    include('app-debug.apk')
    rename('app-debug.apk', POM_ARTIFACT_ID + '_' + VERSION_NAME + '.apk')
}
makeApk.dependsOn(clearApk, build)
```

代码分为3个部分，分别是删除，拷贝，以及将其连接起来的一个设置。

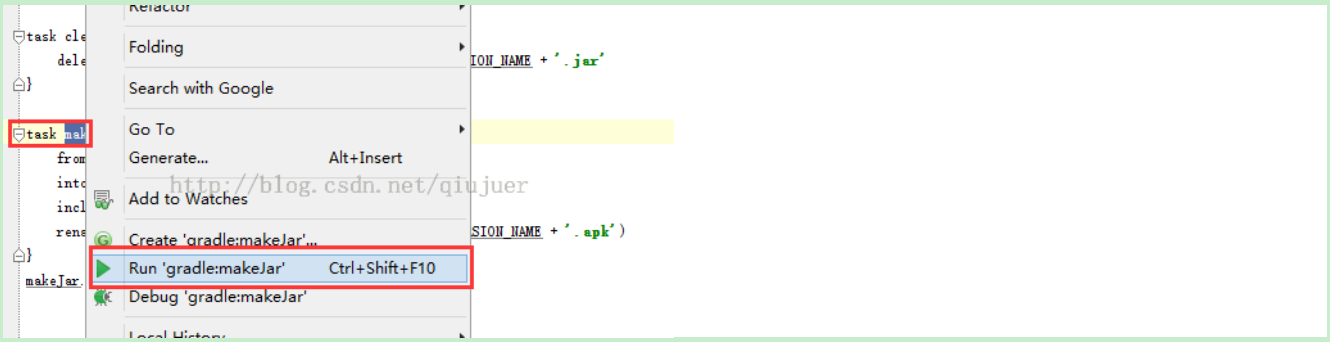
先说说这个 Case 的目的，其目的是拷贝 build/outputs/apk下面的debug apk文件到项目根目录的 release 文件夹下，并且更名。

但是细心的朋友应该会看见其中有两个参数：POM_ARTIFACT_ID VERSION_NAME 这两个从哪里来？

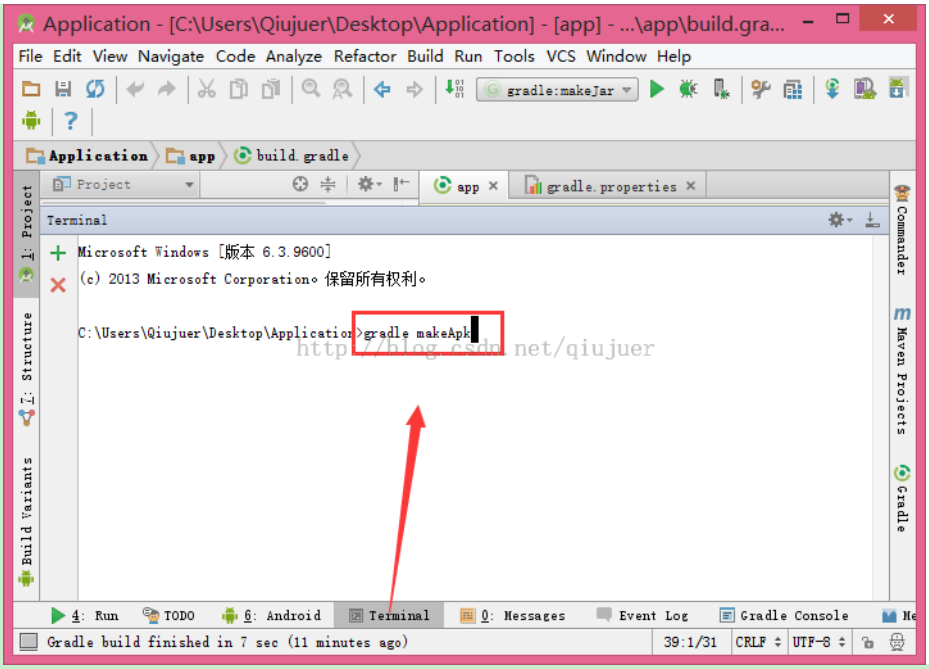
前面最开始讲过：gradle.properties 文件，这两个就是写在项目根部的 gradle.properties 文件中：



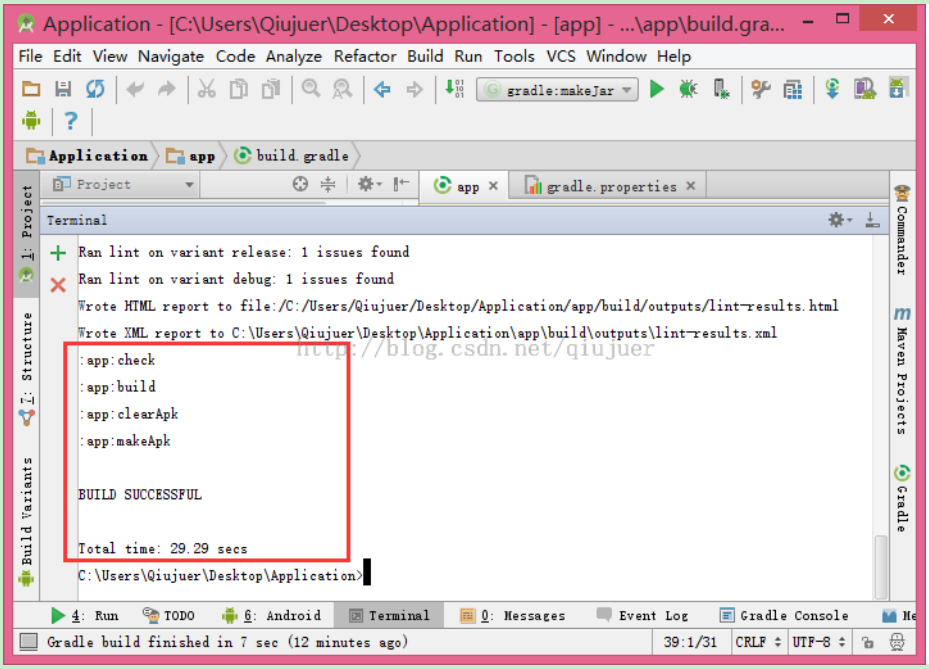
写好了，怎么运行呢？两种方式，第一种，代码 task 上右击，run():



第二张，命令行方式：



输入后回车，等待执行完成，成功后会出现：



现在看看项目中：