# 一. gradle是什么?

gradle跟ant/maven一样，是一种依赖管理/自动化构建工具。但是跟ant/maven不一样，它并没有使用xml语言，而是采用了Groovy语言，这使得它更加简洁、灵活，更加强大的是，gradle完全兼容maven和ivy。更多详细介绍可以看它的官网：http://www.gradle.org/

# 二.为什么使用gradle？

更容易重用资源和代码;

可以更容易创建不同的版本的程序，多个类型的apk包；

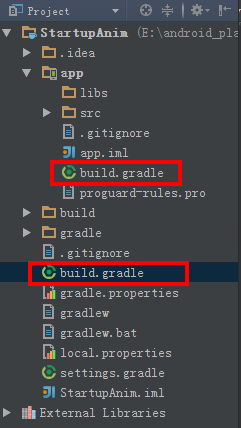
更容易配置，扩展;

更好的IDE集成;

# 三.gradle入门需知

###### 1.基本配置：

首先明确gradle跟maven一样，也有一个配置文件，maven里面是叫pom.xml，而在gradle中是叫build.gradle。Android Studio中的android项目通常至少包含两个build.gradle文件，一个是project范围的，另一个是module范围的，由于一个project可以有多个module，所以每个module下都会对应一个build.gradle。这么说有点抽象，看下面这个图：



这是一个android工程的project视图，上面那个是module下的build.gradle文件。下面那个是project下的build.gradle文件。这两个文件是有区别的，project下的build.gradle是基于整个project的配置，而module下的build.gradle是每个模块自己的配置。下面看下这两个build.gradle里面的内容：

project#build.gradle:

// Top-level build file where you can add configuration options common to all sub-projects/modules.

buildscript {

//构建过程依赖的仓库

repositories {

jcenter()

}

//构建过程需要依赖的库

dependencies {

//下面声明的是gradle插件的版本

classpath 'com.android.tools.build:gradle:1.1.0'

// NOTE: Do not place your application dependencies here; they belong

// in the individual module build.gradle files

}

}

//这里面配置整个项目依赖的仓库,这样每个module就不用配置仓库了

allprojects {

repositories {

jcenter()

}

}

注： repositories需要声明两次，这其实是由于它们作用不同，buildscript中的仓库是gradle脚本自身需要的资源，而allprojects下的仓库是项目所有模块需要的资源。所以千万不要配错了。

module#build.gradle:

//声明插件，这是一个android程序，如果是android库，应该是com.android.library

apply plugin: 'com.android.application'

android {

//安卓构建过程需要配置的参数

compileSdkVersion 21//编译版本

buildToolsVersion "21.1.2"//buildtool版本

defaultConfig {//默认配置，会同时应用到debug和release版本上

applicationId "com.taobao.startupanim"//包名

minSdkVersion 15

targetSdkVersion 21

versionCode 1

versionName "1.0"

}

buildTypes {

//这里面可以配置debug和release版本的一些参数，比如混淆、签名配置等

release {

//release版本

minifyEnabled false//是否开启混淆

proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'

//混淆文件位置

}

}

}

dependencies {

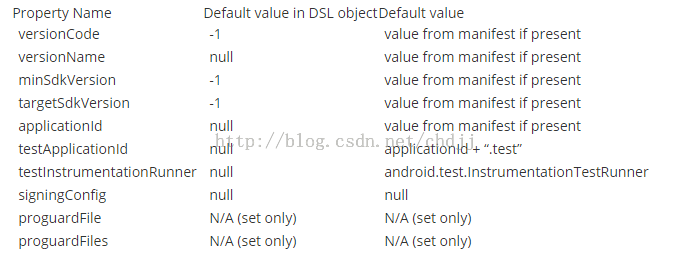
//模块依赖

compile fileTree(dir: 'libs', include: ['\*.jar'])//依赖libs目录下所有jar包

compile 'com.android.support:appcompat-v7:21.0.3'//依赖appcompat库

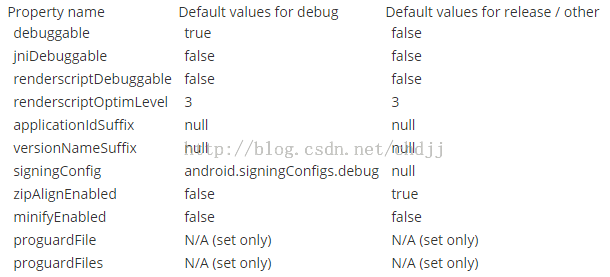
}

defaultConfig中是一些基本配置，它会同时应用到debug/release版本上，下面列举了所有可配项及对应的值：



buildTypes结点很重要，这里可以配置构建的版本的一些参数，默认有两个构建版本release/debug，当然你可以自定义一个构建版本，比如叫foo,然后通过gradlew assembleFoo就可以生成对应的apk了。

buildTypes里还有很多可配置项，下面列举了所有可配项以及debug/release版本的默认值：



下面看下其他一些与gradle相关的文件:

1.gradle.properties：

从名字上就知道它是一个配置文件，没错，这里面可以定义一些常量供build.gradle使用，比如可以配置签名相关信息如keystore位置，密码，keyalias等。

2.settings.gradle：

这个文件是用来配置多模块的，比如你的项目有两个模块module-a,module-b,那么你就需要在这个文件中进行配置，格式如下：

include ':module-a',':module-b'

3.gradle文件夹：

这里面有两个文件，gradle-wrapper.jar和gradle-wrapper.properties,它们就是gradle wrapper。gradle项目都会有，你可以通过命令gradle init来创建它们（前提是本地安装了gradle并且配置到了环境变量中）。

4.gradlew和gradlew.bat:

这分别是linux下的shell脚本和windows下的批处理文件，它们的作用是根据gradle-wrapper.properties文件中的distributionUrl下载对应的gradle版本。这样就可以保证在不同的环境下构建时都是使用的统一版本的gradle，即使该环境没有安装gradle也可以，因为gradle wrapper会自动下载对应的gradle版本。

gradlew的用法跟gradle一模一样，比如执行构建gradle build命令，你可以用gradlew build。gradlew即gradle wrapper的缩写。

###### 2.gradle仓库：

gradle有三种仓库，maven仓库，ivy仓库以及flat本地仓库。声明方式如下：

maven{

url "..."

}

ivy{

url "..."

}

flatDir{

dirs 'xxx'

}

有一些仓库提供了别名，可直接使用：

repositories{

mavenCentral()

jcenter()

mavenLocal()

}

###### 3.gradle任务：

gradle中有一个核心概念叫任务，跟maven中的插件目标类似。gradle的android插件提供了四个顶级任务

assemble 构建项目输出

check 运行检测和测试任务

build 运行assemble和check

clean 清理输出任务

执行任务可以通过gradle/gradlew+任务名称的方式执，执行一个顶级任务会同时执行与其依赖的任务，比如你执行

**gradlew assemble**

它通常会执行:

**gradlew assembleDebug**

**gradlew assembleRelease**

这时会在你项目的build/outputs/apk或者build/outputs/aar目录生成输出文件

注：linux下执行构建任务需要首先更改gradlew脚本的权限，然后才能执行该脚本：

**chmod +x gradlew**

**./gradlew assemble**

可以通过：

**gradlew tasks**

列出所有可用的任务。在Android Studio中可以打开右侧gradle视图查看所有任务。

# 四.常见问题

###### 1.导入本地jar包：

跟eclipse不太一样，android studio导入本地jar除了将jar包放到模块的libs目录中以外，还得在该模块的build.gradle中进行配置，配置方式是在dependencies结点下进行如下声明：

**compile files('libs/xxx.jar')**

如果libs下有多个jar文件，可以这样声明：

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

###### 2.导入maven库：

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

可见，格式为

**compile 'groupId:artifactId:version'**

###### 3.导入某个project：

你的app是多模块的，假设有两个模块app和module-A,并且app模块是依赖module-A的，这时候我们就需要在app模块的build.gradle中的dependencies结点下配置依赖：

**compile project(':module-A')**

并且你需要在settings.gradle中把module-A模块包含进来：

**include ':module-A',':app'**

此外，这种情况下module-A模块是作为库存在的，因而它的build.gradle中的插件声明通常应该是这样的：

**apply plugin: 'com.android.library'**

而且，作为library的模块module-A的build.gradle文件的defaultConfig中是不允许声明applicationId的，这点需要注意。

###### 4.声明三方maven仓库：

可能你项目需要的一些库文件是在你们公司的私服上，这时候repositories中仅有jcenter就不行了，你还需要把私服地址配到里面来，注意，应该配到project的build.gradle中的allprojects结点下或者直接配到某个模块中如果仅有这个模块用到。

配置方式：

repositories{

maven{

url="http://mvnrepo.xxx.com"

}

}

###### 5.依赖三方aar文件：

**compile 'com.aaa.xxx:core:1.0.1@aar'**

###### 6.将库项目导出为aar：

首先你的项目必须是一个库项目，build.gradle中进行配置：

**apply plugin : 'com.android.library'**

然后你可以在命令行中进到项目目录，执行如下gradle任务：

1

**gradlew assembleRelease**//确保该目录下有gradlew文件

生成的aar在/build/output/aar文件夹中

###### 7.引用本地aar：

首先将aar文件放到模块的libs目录下，然后在该模块的build.gradle中声明flat仓库：

repositories{

flatDir{

dirs 'libs'

}

}

最后在dependencies结点下依赖该aar模块：

dependencies{

compile (name:'xxx',ext:'aar')

}

###### 8.排除依赖：

当出现依赖冲突的时候可以通过排除依赖解决，具体方式如下：

compile (group:'xxx',name:'xxx',version:'xxx'){

exclude group:'xxx',module:'xxx'//module对应的就是artifactId

}

###### 9.多dex支持（打包65k方法数限制）

首先在build.gradle的buildConfig中增加如下配置：

multiDexEnabled true

接着，在dependencies结点下增加如下依赖：

dependencies{

compile 'com.android.support:multidex:1.0.0'

}

最后，让你的Application继承MultiDexApplication,如果你的应用没有声明Application，可以在manifest文件的application结点下增加name属性，值为android.support.multidex.MultiDexApplication。

详细内容参见官方文档。

10.自动移除不用资源

可以在buildTypes结点中增加如下配置：

buildTypes{

release{

minifyEnabled true

shrinkResources true

}

}

###### 11.忽略lint错误：

可以在build.gradle文件中的android结点下增加如下配置：

android{

lintOptions{

abortOnError false

}

}

###### 12.声明编译的java版本

可以在build.gradle文件中的android结点下增加如下配置：

compileOptions {

sourceCompatibility JavaVersion.VERSION\_1\_7

targetCompatibility JavaVersion.VERSION\_1\_7

}

###### 13.应用签名配置

首先在module的build.gradle中增加这些字段：

storeFiles:keystore文件存储位置,通常是.jks文件

storePassword 密码

keyAlias keystore别名

keyPassword 密码

具体配置方式为：

首先在build.gradle的android结点下增加如下配置：

signingConfigs {

//debug版本的签名配置,通常不用配，因为有默认的debug签名

debug {

}

release {

storeFile file("key.jks")

storePassword "123456"

keyAlias "mykey"

keyPassword "123456"

}

}

注：debug的默认签名为:

signingConfig android.signingCongfigs.debug

位置为

${home}\.android\debug.keystore

然后在buildTypes结点下的对应版本中添加上面的配置：

buildTypes{

release{

signingConfig signingConfigs.release

}

}

当然，release不是固定的名称，你可以随便取，比如这样：

android {

signingConfigs {

debug {

storeFile file("debug.keystore")

}

myConfig {

storeFile file("other.keystore")

storePassword "android"

keyAlias "androiddebugkey"

keyPassword "android"

}

}

buildTypes {

foo {

debuggable true

jniDebuggable true

signingConfig signingConfigs.myConfig

}

}

}

真实开发中，把密码配置到build.gradle中不是很好的做法，最好的做法是放在gradle.properties中:

RELEASE\_STOREFILE=xxx.jks

RELEASE\_STORE\_PASSWORD=123456

RELEASE\_KEY\_ALIAS=mykey

RELEASE\_KEY\_PASSWORD=123456

然后直接引用即可：

storeFile file(RELEASE\_STOREFILE)

storePassword RELEASE\_STORE\_PASSWORD

keyAlias RELEASE\_KEY\_ALIAS

keyPassword RELEASE\_KEY\_PASSWORD

14.定制buildConfig：

在build.gradle中配置

buildTypes{

release{

buildConfigField "string","type","\"release\""

}

debug{

buildConfigField "string","type","\"debug\""

}

}

这样就会在BuildConfig类中生成type字段：

//build/generate/source/buildConfig/release/包名/ 路径下的BuildConfig.java

public static final String type = "release"

//build/generate/source/buildConfig/debug/包名/ 路径下的BuildConfig.java

public static final String type = "debug"

# 五.常用编译选项配置:

1. Build Type分为debug release版本;
2. Product Flavor:

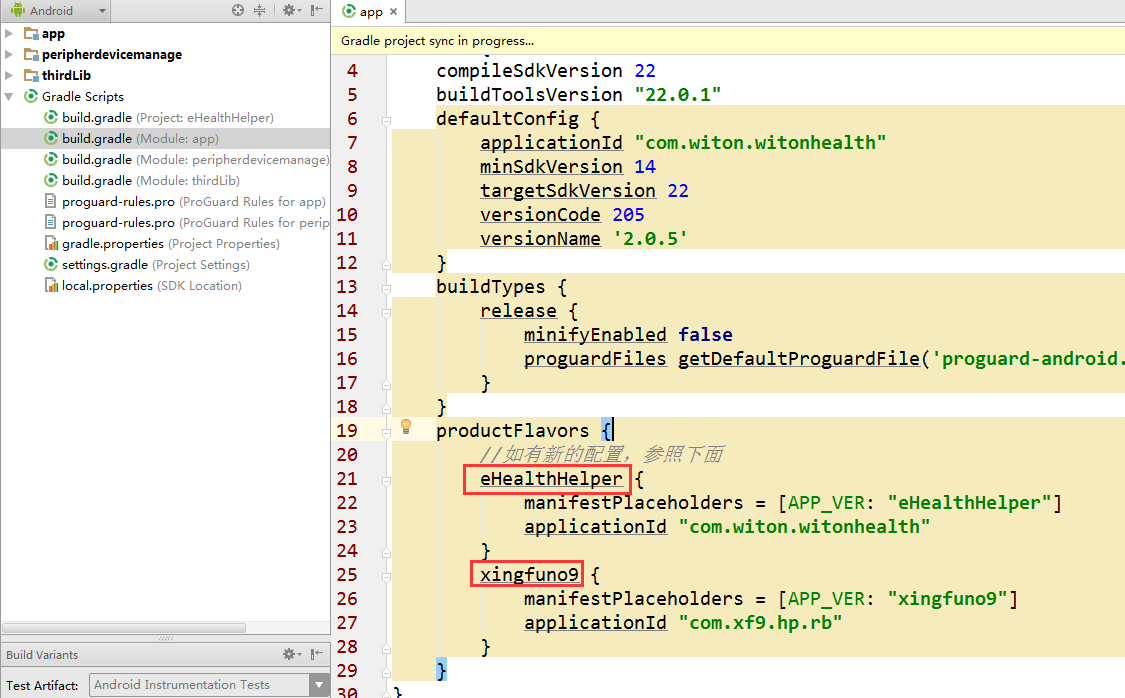
目的是为了同一份代码要打多个包，例如收费 pay 和免费 free，逻辑上有一些小区别，又不想通过逻辑判断这种丑陋的方式。或者你要实现所谓多渠道打包。

最终**Build Variant** = Build Type x Product Flavor

配置好了Build Type 和 Product Flavor 之后，Gradle 会生成若干个包，分别为：

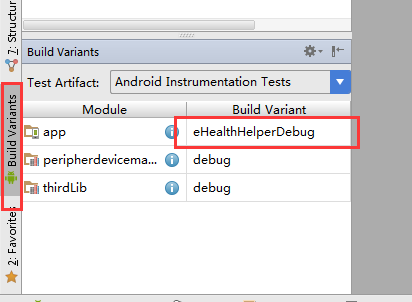
payDebug、payRelease、freeDebug、freeRelease

实例：([点此](http://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.ProductFlavor.html)查看所有的productFlavors选项)



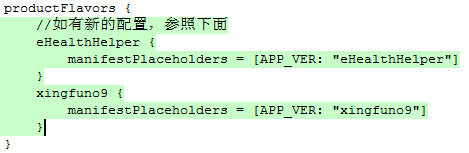
注意，对于java代码，如果要区分free和pay，则必须在两个目录（free、pay）中同时存在此份代码，并且在main中不能存在，否则编译时会产生冲突！

切换代码：



对于Product Flavor中的属性，还有一个比较常用：manifestPlaceholders,manifestxml中的占位符，编译时会替换文件中${APP\_VER}的字符，以便在Context中加载区分版本。

build.gradle:



AndroidManifest.xml:



edit by dong.liang