<https://www.jb51.net/article/256293.htm>

<https://blog.csdn.net/Jason_HD/article/details/128789601>

<https://blog.csdn.net/xinpings/article/details/58688361?spm=1001.2014.3001.5506>

<https://blog.51cto.com/u_12228/11052088>

目前看这三个就够了

安卓配置多渠道包error：All flavors must now belong to a named flavor dimension

需要配置 flavorDimensions "version"

productFlavors {

flavorDimensions "minSdkVersion"//这行放外面好像也行，别的帖子看见过，没测试

// Define separate dev and prod product flavors.

juliang {

minSdkVersion 21

}

common {

minSdkVersion 21

}

}

android多渠道打包使用不同的资源、依赖、代码

android多渠道打包使用不同的资源、依赖、代码1android多渠道打包使用不同的资源、依赖、代码2android多渠道打包使用不同的资源、依赖、代码3android多渠道打包使用不同的资源、依赖、代码4

需求是这样来的，在发布我的app时，在国内的其他市场发布时都能正常通过，但到了百度市场时被拒绝了，给出的理由是个人合作者的app不能包含有支付功能，app里包含了支付的依赖。我就郁闷了，事实上我的app是没有支付功能的，只有分享功能，分享到微信及QQ的功能，所包含的依赖也是为实现此功能的。顺便也吐槽一下百度。。

吐槽归吐槽，app还是得发布的，那只能打包一个不包含分享功能的app了。

这就要用到多渠道打包了，不想用新建多个module的方式来做，就使用gradle配置来实现，实施的时候走了不少的弯路和花了不少时间，在此做记录及分享给有需要的人。

遇到的主要问题是：选择不同的渠道时需要加载不同的依赖并使用不同的代码，这二个地方都有一些坑。

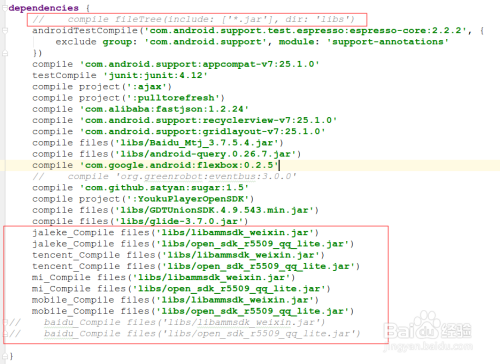
工具/原料

android studio 2.3

gradle

方法/步骤

建立不同的渠道，主要通过设置productFlavors来实现的，如图设置了tencent\_、baudu\_等几个渠道，记住已经设置好的渠道名，下面的步骤需要用到。



android多渠道打包使用不同的资源、依赖、代码

编辑build.gradle的依赖（dependencies），这一步要实现不同的渠道加载不同的依赖。

这里有个需要注意的地方，一般都有这么一句：

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

这句的作用是自动加载libs里的所有的.jar文件，这一句一定要去掉，否则实现不了不同渠道加载不同的jar的目的。去掉这句之后其他的jar只能通过compile 'com.github.satyan:sugar:1.5'的方式加载。

不同渠道加载不同的依赖就要在compile上做文章了，格式是：渠道名+Compile，这样就是渠道独有的，如tencent\_Compile就是tencent\_渠道独有的加载。

如图中的这二句：

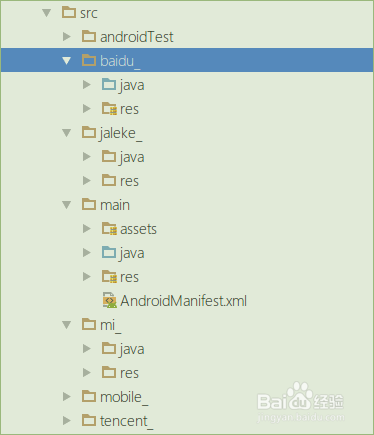
// baidu\_Compile files('libs/libammsdk\_weixin.jar')

// baidu\_Compile files('libs/open\_sdk\_r5509\_qq\_lite.jar')

这二句被注释了，所以baidu\_渠道不会加载这二个jar了。

android多渠道打包使用不同的资源、依赖、代码

在项目（module）的src新建已渠道名称命名的文件夹，名字一定要跟渠道名一样，级别跟main文件夹是一样的，里面独有文件独有资源的文件结构要跟main一致。（不要在AS里创建，会自动给加个main文件夹，导致文件找不到，直接在文件系统中创建就行，创建好如果不刷新，就多等一会，早晚会刷新的）



这里有二种文件，一种是需要用到的资源文件，如各种xml文件或图片等，一种是java代码。这里我遇到了一个坑。一开始我以为渠道文件夹里的文件在编译时会自动替换掉main文件夹里的相应文件，xml文件和图片等资源确实是这样的，没有问题；但到了java文件时，文件就出现了，如图中ShareActivity.java文件，在各个渠道文件夹里有这个文件，在main里也有这个文件，但在编译时就会提示类重复，编译不通过。哈哈。。说好的自动替换呢，怎么还编译不了呢，郁闷了。后来经过多次实验得出结论，只要把main里的java文件删除就能正常编译了。这是遇到的最大的坑，花了不少时间。后来才得知，各个渠道文件夹和main文件夹之间的关系不是单纯的替换，而是优先使用渠道的东西，编译时相当于是合并，xml等文件相当于直接替换，但java却不是这样的。我只能只要理解了，哈哈

android多渠道打包使用不同的资源、依赖、代码

**Android studio gradle中分渠道加载res、libraries及Class**

前段时间遇到这样的需求，在App里使用了地图功能，对应海外版使用Android内置google的地图服务。而对于国内版（大家都知道的），我们使用了高德地图。而在导入高德地图的时候需要加入对应的sdk及.so文件，这样最后打包的大小会增加2M左右。但海外版并不需要这部分资源，故需要根据不同的渠道加载不同的jar、res、及不同的类，有助于保持代码的干净，控制包的大小。

不多说了，直接上步骤！

1）首先要获创建不同的productFlavors，便于分渠道打包和调试；

productFlavors {

// Define separate dev and prod product flavors.

dev {

minSdkVersion 21

...

}

Chinese\_dev {

minSdkVersion 21

...

}

1

2

3

4

5

6

7

8

9

10

2）要获取当前Flavor()，用于选择相应的资源：

def getCurrentFlavor() {

Gradle gradle = getGradle()

String taskStr = gradle.getStartParameter().getTaskRequests().toString()

println("taskStr:" + taskStr)

println()

Pattern pattern;

if (taskStr.contains("assemble")) {

pattern = Pattern.compile("assemble(\\w+)(Release|Debug)")

} else {

pattern = Pattern.compile("generate(\\w+)(Release|Debug)")

}

Matcher matcher = pattern.matcher(taskStr)

if (matcher.find()) {

returnmatcher.group(1);

}

return ""

}

3）判断是否是自己需要的Flavor,并选择libs的路径（这里需要创建不同名字的libs包，如这里可以把高德的相关jar放入cn\_libs目录下）：

def buildLibs() {

String currentFlavor = getCurrentFlavor()

println("current Flavor:" + currentFlavor)

booleanisChinese = "Chinese\_dev".equalsIgnoreCase(currentFlavor))

println("--- is Chinese ---" + isChinese)

if (isChinese) {

return'cn\_libs'

}

return 'libs'

}

4）在dependencies中加载时调用buildLibs方法来取出要加载的资源包：

dependencies {

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

5）为了完整的让不同的资源分开，还需要把AndroidManifest.xml及resource文件都分开。在main目录的同级目录下可创建两个单独的目录，如分别是cn及en，并放入对应的代码及资源。

对于AndroidManifest.xml，关键资源使用 tools:node=“merge”：

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools">

<uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" tools:node="merge"/>

<uses-permission android:name="android.permission.CHANGE\_WIFI\_STATE" tools:node="merge"/>

<application

tools:node="merge">

<meta-data

android:name="com.amap.api.v2.apikey"

android:value="key"/>

<service android:name="com.amap.api.location.APSService"/>

</application>

</manifest>

2.对于代码class的替换，若要根据flavor来切换代码，则需要不同的flavor的java包，并在其下创建下创建相同的类名，并且在main下不能再有该类，则在打包时只打对应的class文件，其他的则丢弃。如：

| Flavor A

| ——-> com.fan.util

| ————> class A

| main

| ——>com.fan.util

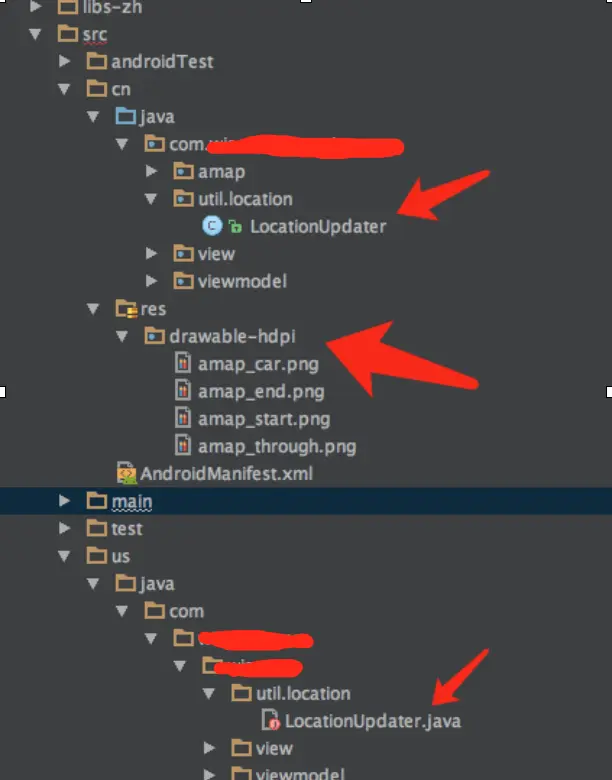
| ————>不能再有class A

| Flavor B

| ——>com.fan.util

| ————->class A

创建好后的目录大概长这样：



3.在gradle文件中配置java.srcDirs,AndroidManifest.xml及resource的位置:

sourceSets {

main {

jniLibs.srcDirs = ['libs']

java.srcDirs = ['src/main/java']

res.srcDirs = ['src/main/res']

}

dev {

java.srcDirs = ['src/us/java/']

}

Chinese\_dev {

jniLibs.srcDirs = ['libs-zh']

manifest.srcFile 'src/cn/AndroidManifest.xml'

java.srcDirs = ['src/cn/java/']

res.srcDirs = ['src/cn/res']

}

}

然后就可以通过Build Variants愉快的玩耍了！！！

————————————————

版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。

android 制作差分包命令 android差异化打包 转载

香奈儿2024-06-05 12:48:57

文章标签android 制作差分包命令android差异化打包gradle差异化打包差异化文章分类Android移动开发

Android 中实现差异化打包权威指南

一.差异化打包的使用场景

思考：

一个项目为多个不同的渠道商开发，渠道商都要求显示自己的Logo，怎么设计项目结构 ------某个图片资源不同（或者其他的资源不同）

如果某一个渠道商表示去掉某一个功能，怎么处理 ------ 某个逻辑判断不同

如果某一个渠道商需要添加一个自己的宣传页，怎么处理 ------入口不同

如果渠道商的部分页面不同，怎么组织项目 -------存在逻辑和页面不同

项目要上线到不同的应用市场，需要统计在不同应用市场的下载情况和使用情况 -----在manifest中配置不同的渠道号

解决以上问题的处理方案：

新建多个项目-----不利于代码维护，如果项目出错，需要修改所有相关工程

新建多个分支-----不利于代码维护，如果项目出错，需要修改所有相关分支

差异化打包 ------ 修改公共部分，一处修改，其他渠道都同步修改

差异化打包用于主线相同，但是同时存在部分功能不同的情况，可以在较小的代码修改的情况下，实现不同的功能

二.差异化打包的使用

1. flavorDimensions多维度要求

flavorDimensions配置多维度的配置渠道

比如现在有一个项目，不同的渠道商的要求不同，而且不同的渠道商又有不同的机型，不同的机型的功能也存在差异。此时仅仅使用渠道商一个维度来差异化打包是不够的，需要通过多维度配置不同版本。

flavorDimensions配置在defaultConfig中， 代码中配置了CHANNEL和 MACHINE\_MODE两个维度 ，代码如下

defaultConfig {

flavorDimensions "CHANNEL", "MACHINE\_MODE"

}

注意：

defaultConfig中配置的维度在productFlavors必须全部配置，否则项目会编译不通过

如果仅仅只有一个维度，flavorDimensions也需要配置，可以配置为default，此时productFlavors中可以不配置维度名称

flavorDimensions中维度具有优先级，优先级是从高到低

2. productFlavors多渠道配置

productFlavors {

T1 {

dimension "MACHINE\_MODE"

buildConfigField "String", "MACHINE\_CPU", "\"CPU5200\""

buildConfigField "int", "MACHINE\_TYPE", "1"

}

T2 {

dimension "MACHINE\_MODE"

buildConfigField "String", "MACHINE\_CPU", "\"CPU5300\""

buildConfigField "int", "MACHINE\_TYPE", "2"

}

wandoujia {

dimension "CHANNEL"

applicationId = "com.djt.productflavordemo.wandoujia"

defaultConfig {

versionNameSuffix "-wandoujia"

versionName "1.0.20200727.1"

}

manifestPlaceholders = [AppName: "豌豆荚", CheckUpdateAppId: "com.djt.productflavordemo.wandoujia"]

}

huawei {

dimension "CHANNEL"

applicationId = "com.djt.productflavordemo.huawei"

defaultConfig {

versionNameSuffix "-huawei"

versionName "1.0.20200727.2"

}

manifestPlaceholders = [AppName: "华为应用商店", CheckUpdateAppId: "com.djt.productflavordemo.wandoujia"]

}

}

sourceSets {

wandoujia {

res.srcDirs = ['src/wandoujia/res']

java {

srcDirs "src/wandoujia/java"

}

}

huawei {

res.srcDirs = ['src/huawei/res']

java {

srcDirs "src/huawei/java"

}

}

}

以上配置的productFlavors生成的版本如下：

android 制作差分包命令 android差异化打包\_android 制作差分包命令

（1）flavorDimensions和productFlavors的关系：

在AS中打开Build Variants构建差异化视图，左侧是模块，右侧是模块存在的差异化版本列表，可以选择一个变体版本作为当前运行的版本。

版本名称命名规则与flavorDimensions配置的维度优先级有关：：渠道名称+机型+Debug/Release

渠道名称和机型的先后顺序与flavorDimensions中CHANNEL和 MACHINE\_MODE的优先级一致

每一个版本必须且仅能配置一个维度，否则编译失败

（2）productFlavors中配置的值与BuildConfig之间的对应关系

public final class BuildConfig {

public static final boolean DEBUG = Boolean.parseBoolean("true");

public static final String APPLICATION\_ID = "com.djt.productflavordemo.huawei"; //应用的包名，对应`productFlavors`的applicationId

public static final String BUILD\_TYPE = "debug";

public static final String FLAVOR = "huaweiT1"; //变体的名称："CHANNEL"+"MACHINE\_MODE",与`flavorDimensions`保持一致

public static final int VERSION\_CODE = 1;

public static final String VERSION\_NAME = "1.0.20200727.2-huawei"; //versionName和 versionNameSuffix结合

public static final String FLAVOR\_CHANNEL = "huawei"; //所属渠道名称

public static final String FLAVOR\_MACHINE\_MODE = "T1";//所属机型名称

// Fields from product flavor: T1

public static final String MACHINE\_CPU = "CPU5200"; //productFlavors配置的String类型常量值

public static final int MACHINE\_TYPE = 1;//productFlavors配置的int常量值

}

注意：

BuildConfig文件中的常量值在productFlavors配置的时候，int类型的值必须加双引号，String类型的值必须添加转义符号的双引号。

T1 {

dimension "MACHINE\_MODE"

buildConfigField "String", "MACHINE\_CPU", "\"CPU5200\""

buildConfigField "int", "MACHINE\_TYPE", "1"

}

（3）sourceSets 配置各变体的代码文件的路径和资源文件的路径

需要在main文件夹同层级文件夹新建wandoujia和huawei文件夹，文件夹的内部文件结构要和main文件夹的子文件结构保持一致，但是仅仅需要新建将要使用到的文件结构。

sourceSets {

wandoujia {

res.srcDirs = ['src/wandoujia/res']

java {

srcDirs "src/wandoujia/java"

}

}

huawei {

res.srcDirs = ['src/huawei/res']

java {

srcDirs "src/huawei/java"

}

}

}

Java文件树对比

android 制作差分包命令 android差异化打包\_android 制作差分包命令\_02

res文件树对比

android 制作差分包命令 android差异化打包\_gradle差异化打包\_03

（4）使用versionNameSuffix的优点

不同的渠道商编译的APK的版本号都添加相应渠道商的后缀，方便测试人员识别多个渠道编译的APK，以免APK混淆

3. 代码文件如何实现差异化

使用场景：该版本与主线版本逻辑基本相同，只有部分逻辑不同，可以获取BuildConfig的值进行区分当前版本

代码如下：

val sb = StringBuffer()

when (BuildConfig.FLAVOR\_MACHINE\_MODE) {

"T1" -> Toast.makeText(

this@MainActivity,

"机器名称:${BuildConfig.FLAVOR\_MACHINE\_MODE}",

Toast.LENGTH\_SHORT

).show()

"T2" -> Toast.makeText(

this@MainActivity,

"机器名称:${BuildConfig.FLAVOR\_MACHINE\_MODE}",

Toast.LENGTH\_SHORT

).show()

"T3" -> Toast.makeText(

this@MainActivity,

"机器名称:${BuildConfig.FLAVOR\_MACHINE\_MODE}",

Toast.LENGTH\_SHORT

).show()

}

sb.append("机器名称:${BuildConfig.FLAVOR\_MACHINE\_MODE}\n")

if (BuildConfig.MACHINE\_TYPE == 1) {

sb.append("机型:${BuildConfig.MACHINE\_TYPE}\n")

} else {

sb.append("机型:${BuildConfig.MACHINE\_TYPE}\n")

}

if (BuildConfig.MACHINE\_CPU == "CPU5200") {

sb.append("机型:${BuildConfig.MACHINE\_CPU}\n")

} else {

}

sb.append(MetaUtils.getMetaStrValue(this, "CheckUpdateAppId")+"\n")

sb.append(getString(R.string.app\_name))

content.text = sb.toString()

使用场景：该版本部分类与主线版本差异较大，需要把整个类提取出来。

把提取出来的类，保存在对应的模块的对应文件夹下。

如果在某一个模块下提取了某个文件为差异化部分，那么在所有的差异化模块都要把这个文件提取出来

4. 资源文件如何实现差异化

如果仅仅是资源文件不同，直接在res对应的文件夹下面新建一个文件相同文件名的文件就好。在实际打包过程中，差异化文件夹下面的res文件会和main主线的res文件合并，APK在实际调用过程中，会优先调用差异化包中的文件，在差异化包中没有对应文件或者资源，会调用main主线中的资源

适用范围：

适用所有的res文件

在适配图片的时候，必须在将图片放在适配机器对应的尺寸文件下，否则会调用用main主线中的资源

5. AndroidManifest文件如何实现差异化

manifest文件也可以实现差异化，在main主线中manifest属于所有项目的通用部分，变体版本文件下的manifest属于各自项目的独有部分。

main主线的manifest和变体版本文件下manifest配置不能冲突，否则编译会报错

main主线的manifest和变体版本文件下manifest配置组件的不同属性，结果是两个manifest的合并

三.其他与AndroidManifest内容补充：

（1）如何在AndroidManifest中使用build.gradle文件的赋值

在meta-data元素下通过${}引用

<meta-data

android:name="CheckUpdateAppId"

android:value="${CheckUpdateAppId}" />

1.

2.

3.

在build.gradle中使用manifestPlaceholders赋值，manifestPlaceholders接受的是一个数组类型，格式[常量名称1：常量值1，常量名称2，常量值2]

huawei {

dimension "CHANNEL"

applicationId = "com.djt.productflavordemo.huawei"

defaultConfig {

versionNameSuffix "-huawei"

versionName "1.0.20200727.2"

}

manifestPlaceholders = [AppName: "华为应用商店", CheckUpdateAppId: "com.djt.productflavordemo.wandoujia"]

（2）如何在代码块中使用AndroidManifest中的meta-data节点的数据

ApplicationInfo的metaData返回的是一个Bundle类型的对象，可以根据meta-data中要获取的值的类型，来调用Bundle对应的获取相应类型的值的方法。代码中定义了获取String，int和boolean类型值的方法

/\*\*

\* 获取meta的所有值

\* @param context

\* @return

\*/

public static Bundle getMetaBundle(Context context) {

try {

ApplicationInfo info = context.getPackageManager().getApplicationInfo(context.getPackageName(), PackageManager.GET\_META\_DATA);

return info.metaData;

} catch (Exception e) {

e.printStackTrace();

}

return null;

}

/\*\*

\* 获取meta的Str值

\* @param context

\* @param keyName

\* @return

\*/

public static String getMetaStrValue(Context context, String keyName) {

try {

Bundle metaBundle = getMetaBundle(context);

if(metaBundle != null) {

String msg = metaBundle.getString(keyName);

if (!TextUtils.isEmpty(msg)) {

return msg;

}

}

} catch (Exception e) {

e.printStackTrace();

}

return "";

}

/\*\*

\* 获取meta的Int值

\* @param context

\* @param keyName

\* @return

\*/

public static int getMetaIntValue(Context context, String keyName, int defaultValue) {

try {

Bundle metaBundle = getMetaBundle(context);

if(metaBundle != null) {

return metaBundle.getInt(keyName, defaultValue);

}

} catch (Exception e) {

e.printStackTrace();

}

return defaultValue;

}

/\*\*

\* 获取meta的Boolean值

\* @param context

\* @param keyName

\* @return

\*/

public static boolean getMetaBooleanValue(Context context, String keyName, boolean defaultValue) {

try {

Bundle metaBundle = getMetaBundle(context);

if(metaBundle != null) {

return metaBundle.getBoolean(keyName, defaultValue);

}

} catch (Exception e) {

e.printStackTrace();

}

return defaultValue;

}

注意：

meta-data可以动态获取，但是不可以在代码中动态设置。虽然我们可以获取 ApplicationInfo的metaData返回的是一个Bundle类型的对象，并可以对Bundle进行赋值，但是仅仅是修改了Bundle对象的值，实际获取meta-data的值不会修改。

（3）如何实现一个APK拥有多个入口，即在桌面上显示不同的图标和应用名

为需要作为入口的Activity添加Launcher属性；

为Activity添加label值和icon值，分别为入口名称和图标

<activity

android:name=".MainActivity"

android:label="豌豆荚"

android:icon="@drawable/ic\_launcher\_foreground">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

1.

如果不配置label值和icon值，则应用的图标和名称与Application中配置的相同