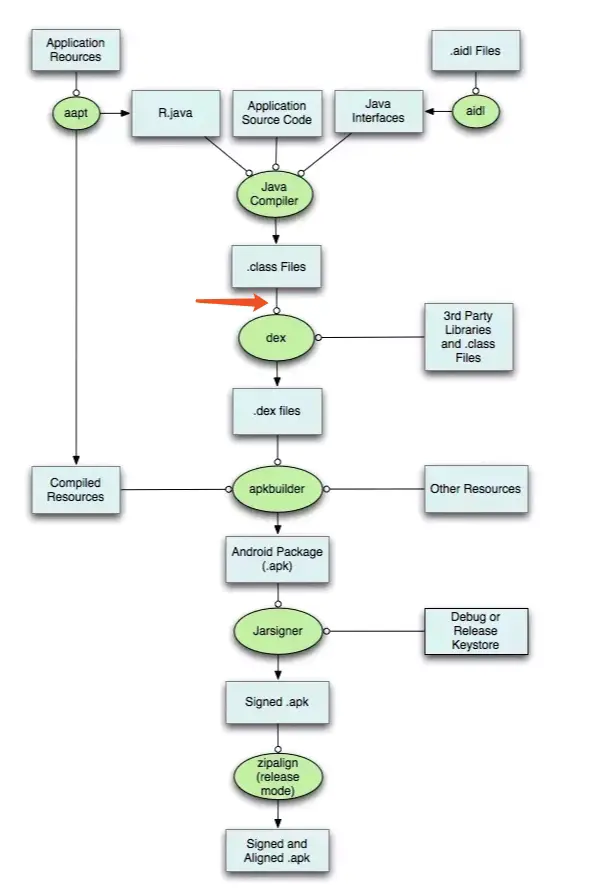
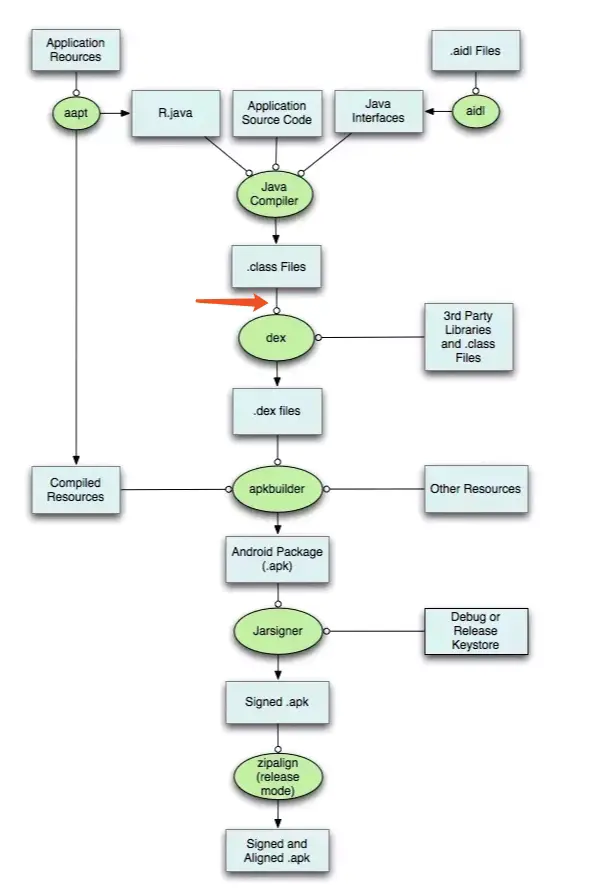
# Android Gradle Transform 详解

我们先来看下Android应用程序打包流程：



通过上图可知，我们只要在图中红色箭头处拦截（生成class文件之后，dex文件之前），就可以拿到当前应用程序中所有的.class文件，再去借助ASM之类的库，就可以遍历这些.class文件中所有方法，再根据一定的条件找到需要的目标方法，最后进行修改并保存，就可以插入我们的埋点代码。

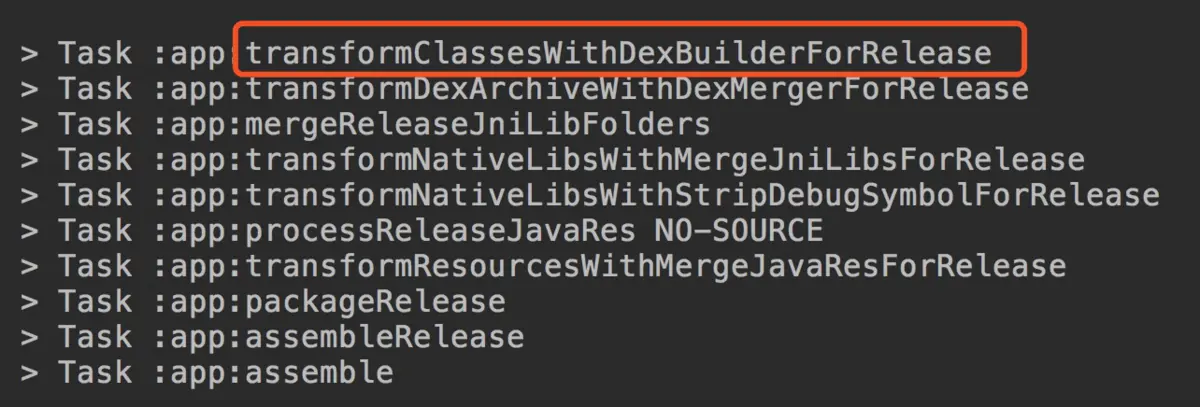
Google从 Android Gradle 1.5.0 开始，提供了Transform API。通过Transform API，允许第三方以插件的形式，在Android应用程序打包成dex文件之前的编译过程中操作.class文件。我们只要实现一套Transform，去遍历所有.class文件的所有方法，然后进行修改（在特定的listener回调中插入埋点代码），再对源文件进行替换，即可以达到插入代码的目的。



# Gradle Transform概述

Gradle Transform是Android官方提供给开发者在项目构建阶段（.class -> .dex转换期间）用来修改.class文件的一套标准API，即把输入的.class文件转变成目标字节码文件。目前比较经典的应用是字节码插桩、代码注入等。

我们build一个项目，会打印出如下日志，红框框住的部分就是一个Transform的名称



通过上张图可以看到原生就带了一系列Transform供使用，那么这些Transform是怎么组织在一起的呢？

每个Transform其实都是一个gradle task，Android编译器中的TaskManager将每个Transform串连起来，第一个Transform接收来自javac编译的结果，以及已经拉取到在本地的第三方依赖（jar、aar），还有resource资源，注意，这里的resource并非android项目中的res资源，而是asset目录下的资源。 这些编译的中间产物，在Transform组成的链条上流动，每个Transform节点可以对class进行处理再传递给下一个Transform。我们常见的混淆，Desugar等逻辑，它们的实现如今都是封装在一个个Transform中，而我们自定义的Transform，会插入到这个Transform链条的最前面。

最终，我们定义的Transform会被转化成一个个TransformTask，在Gradle编译时调用。

# Transform两个基础概念

TransformInput

TransformOutputProvider

## TransformInput

TransformInput是指输入文件的一个抽象，包括：

## DirectoryInput集合

是指以源码的方式参与项目编译的所有目录结构及其目录下的源码文件

JarInput集合

是指以jar包方式参与项目编译的所有本地jar包和远程jar包（此处的jar包包括aar）

TransformOutputProvider

之Transform的输出，通过它可以获取到输出路径等信息

# Transform.java

先来了解下Transform类，定义如下

public abstract class Transform {

public Transform() {  
 }  
  
 // Transform名称  
 public abstract String getName();  
  
 public abstract Set<ContentType> getInputTypes();  
  
 public Set<ContentType> getOutputTypes() {  
 return this.getInputTypes();  
 }  
  
 public abstract Set<? super Scope> getScopes();  
  
  
 public abstract boolean isIncremental();  
  
 */\*\** ***@deprecated*** *\*/* @Deprecated  
 public void transform(Context context, Collection<TransformInput> inputs, Collection<TransformInput> referencedInputs, TransformOutputProvider outputProvider, boolean isIncremental) throws IOException, TransformException, InterruptedException {  
 }  
  
 public void transform(TransformInvocation transformInvocation) throws TransformException, InterruptedException, IOException {  
 this.transform(transformInvocation.getContext(), transformInvocation.getInputs(), transformInvocation.getReferencedInputs(), transformInvocation.getOutputProvider(), transformInvocation.isIncremental());  
 }  
  
 public boolean isCacheable() {  
 return false;  
 }  
  
 ...  
}

# Transform#getName()

Transform名称，上面build日志红框框住的部分就是Transform名称

transformClassesWithDexBuilderForDebug

那么最终的名字是如何构成的呢？

在gradle plugin的源码中有一个叫TransformManager的类，这个类管理着所有的Transform的子类，里面有一个方法叫getTaskNamePrefix，在这个方法中就是获得Task的前缀，以transform开头，之后拼接ContentType，这个ContentType代表着这个Transform的输入文件的类型，类型主要有两种，一种是Classes，另一种是Resources，ContentType之间使用And连接，拼接完成后加上With，之后紧跟的就是这个Transform的Name，name在getName()方法中重写返回即可。

TransformManager#getTaskNamePrefix()代码如下：

static String getTaskNamePrefix(Transform transform) {

StringBuilder sb = new StringBuilder(100);  
 sb.append("transform");  
 sb.append((String)transform.getInputTypes().stream().map((inputType) -> {  
 return CaseFormat.UPPER\_UNDERSCORE.to(CaseFormat.UPPER\_CAMEL, inputType.name());  
 }).sorted().collect(Collectors.joining("And")));  
 sb.append("With");  
 StringHelper.appendCapitalized(sb, transform.getName());  
 sb.append("For");  
 return sb.toString();  
}

# Transform#getInputTypes()

需要处理的数据类型，有两种枚举类型

1. CLASSES

代表处理的 java 的 class 文件，返回TransformManager.CONTENT\_CLASS

1. RESOURCES

代表要处理 java 的资源，返回TransformManager.CONTENT\_RESOURCES

# Transform#getScopes()

指 Transform 要操作内容的范围，官方文档 Scope 有 7 种类型：

1. EXTERNAL\_LIBRARIES ： 只有外部库
2. PROJECT ： 只有项目内容
3. PROJECT\_LOCAL\_DEPS ： 只有项目的本地依赖(本地jar)
4. PROVIDED\_ONLY ： 只提供本地或远程依赖项
5. SUB\_PROJECTS ： 只有子项目
6. SUB\_PROJECTS\_LOCAL\_DEPS： 只有子项目的本地依赖项(本地jar)
7. ESTED\_CODE ：由当前变量(包括依赖项)测试的代码

如果要处理所有的class字节码，返回TransformManager.SCOPE\_FULL\_PROJECT

# Transform#isIncremental()

增量编译开关

当我们开启增量编译的时候，相当input包含了changed/removed/added三种状态，实际上还有notchanged。需要做的操作如下：

1. NOTCHANGED: 当前文件不需处理，甚至复制操作都不用；
2. ADDED、CHANGED: 正常处理，输出给下一个任务；
3. REMOVED: 移除outputProvider获取路径对应的文件。

# Transform#transform()

public void transform(@NonNull TransformInvocation transformInvocation)

throws TransformException, InterruptedException, IOException {  
 // Just delegate to old method, for code that uses the old API.  
 //noinspection deprecation  
 this.transform(transformInvocation.getContext(), transformInvocation.getInputs(),  
 transformInvocation.getReferencedInputs(),  
 transformInvocation.getOutputProvider(),  
 transformInvocation.isIncremental());  
}

注意点

1. 如果拿取了getInputs()的输入进行消费，则transform后必须再输出给下一级
2. 如果拿取了getReferencedInputs()的输入，则不应该被transform
3. 是否增量编译要以transformInvocation.isIncremental()为准

Transform#isCacheable()

如果我们的transform需要被缓存，则为true，它被TransformTask所用到

# Transform编写模板

AspectJTransform.groovy代码如下：

class AspectJTransform extends Transform {

final String NAME = "JokerwanTransform"  
  
 @Override  
 String getName() {  
 return NAME  
 }  
  
 @Override  
 Set<QualifiedContent.ContentType> getInputTypes() {  
 return TransformManager.CONTENT\_CLASS  
 }  
  
 @Override  
 Set<? super QualifiedContent.Scope> getScopes() {  
 return TransformManager.SCOPE\_FULL\_PROJECT  
 }  
  
 @Override  
 boolean isIncremental() {  
 return false  
 }  
  
 @Override  
 void transform(TransformInvocation transformInvocation) throws TransformException, InterruptedException, IOException {  
 super.transform(transformInvocation)  
  
 // OutputProvider管理输出路径，如果消费型输入为空，你会发现OutputProvider == null  
 TransformOutputProvider outputProvider = transformInvocation.getOutputProvider();  
  
 transformInvocation.inputs.each { TransformInput input ->  
 input.jarInputs.each { JarInput jarInput ->  
 // 处理Jar  
 processJarInput(jarInput, outputProvider)  
 }  
  
 input.directoryInputs.each { DirectoryInput directoryInput ->  
 // 处理源码文件  
 processDirectoryInputs(directoryInput, outputProvider)  
 }  
 }  
 }  
  
 void processJarInput(JarInput jarInput, TransformOutputProvider outputProvider) {  
 File dest = outputProvider.getContentLocation(  
 jarInput.getFile().getAbsolutePath(),  
 jarInput.getContentTypes(),  
 jarInput.getScopes(),  
 Format.JAR)  
  
 // to do some transform  
  
 // 将修改过的字节码copy到dest，就可以实现编译期间干预字节码的目的了   
 FileUtils.copyFiley(jarInput.getFile(), dest)  
 }  
  
 void processDirectoryInputs(DirectoryInput directoryInput, TransformOutputProvider outputProvider) {  
 File dest = outputProvider.getContentLocation(directoryInput.getName(),  
 directoryInput.getContentTypes(), directoryInput.getScopes(),  
 Format.DIRECTORY)  
 // 建立文件夹   
 FileUtils.forceMkdir(dest)  
  
 // to do some transform  
  
 // 将修改过的字节码copy到dest，就可以实现编译期间干预字节码的目的了   
 FileUtils.copyDirectory(directoryInput.getFile(), dest)  
 }  
}

链接：<https://www.jianshu.com/p/cf90c557b866>

# 目录1

## 目录2

### 目录3

#### 目录4

##### 目录5