Bugly Android热更新详解

完整接入流程

- 打基准包安装并上报联网（注：填写唯一的tinkerId）
- 对基准包的bug修复（可以是Java代码变更，资源的变更）
- 修改基准包路径、填写补丁包tinkerId、mapping文件路径、resId文件路径
- 执行tinkerPatchRelease打Release版本补丁包
- 选择app/build/outputs/patch目录下的补丁包并上传（注：不要选择tinkerPatch目录下的补丁包，不然上传会有问题）
- 编辑下发补丁规则，点击立即下发
- 重启基准包，请求补丁策略（SDK会自动下载补丁并合成）
- 再次重启基准包，检验补丁应用结果
- 查看页面，查看激活数据的变化

普通打包

1、编译基准包

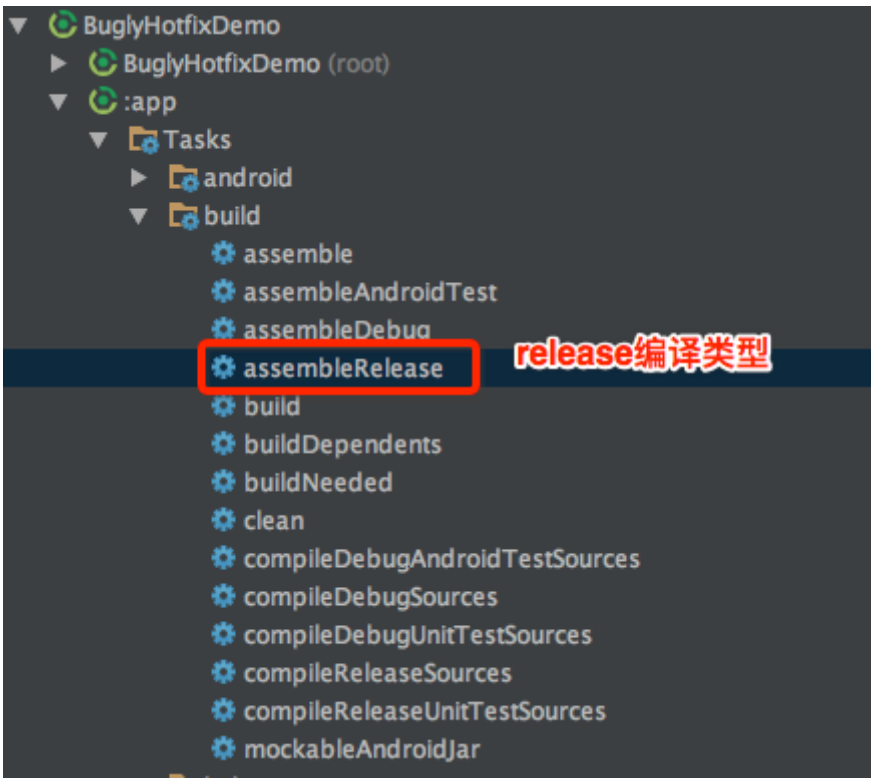
配置基准包的tinkerId

```
apply plugin: 'com.tencent.bugly.tinker-support'

def bakPath = file("${buildDir}/bakApk/")

def appName = "app-0113-10-45-57"

/**
 * 对于插件各参数的详细解析请参考
 */
tinkerSupport {

    // 开启tinker-support插件，默认值true
    enable = true

    // 指定归档目录，默认值当前module的子目录tinker
    autoBackupApkDir = "${bakPath}"

    // 是否启用覆盖tinkerPatch配置功能，默认值false
    // 开启后tinkerPatch配置不生效，即无需添加tinkerPatch
    overrideTinkerPatchConfiguration = true

    // 编译补丁包时，必需指定基线版本的apk，默认值为空
    // 如果为空，则表示不是进行补丁包的编译
    // @{link tinkerPatch.oldApk }
    baseApk =  "${bakPath}/${appName}/app-release.apk"

    // 对应tinker插件applyMapping
    baseApkProguardMapping = "${bakPath}/${appName}/app-release-mapping.txt"

    // 对应tinker插件applyResourceMapping
    baseApkResourceMapping = "${bakPath}/${appName}/app-release-R.txt"

    // 基线版本和补丁版本都需要修改这个参数，示例: 1.0.1-base 1.0.1-patch
    tinkerId = "1.0.1-base"

}
```

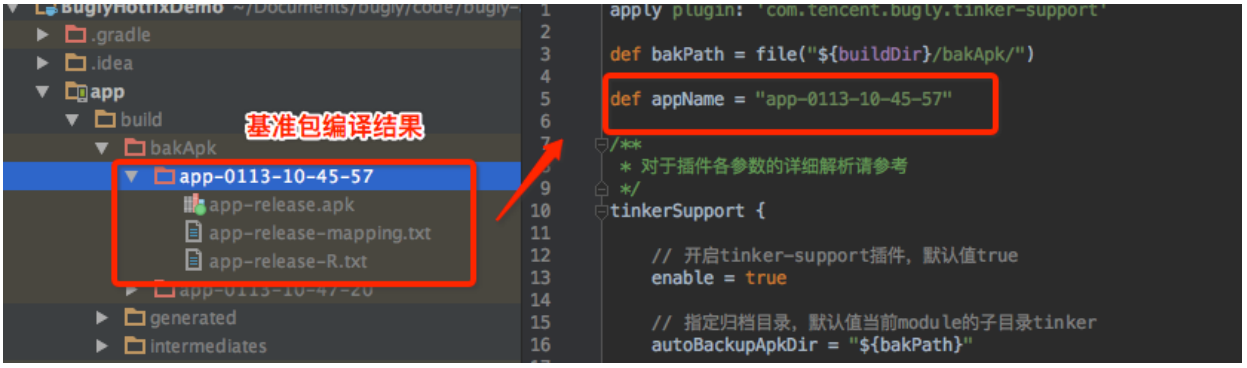
tinkerId最好是一个唯一标识，例如git版本号、versionName等等。 如果你要测试热更新，你需要对基线版本进行联网上报。

> 这里强调一下，基线版本配置一个唯一的tinkerId，而这个基线版本能够应用补丁的前提是集成过热更新SDK，并启动上报过联网，这样我们后台会将这个tinkerId对应到一个目标版本，例如tinkerId = “bugly_1.0.0” 对应了一个目标版本是1.0.0，基于这个版本打的补丁包就能匹配到目标版本。

执行`assembleRelease`编译生成基准包：



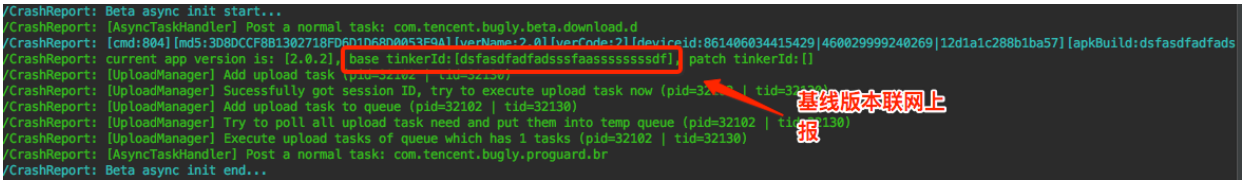
这个会在build/outputs/bakApk路径下生成每次编译的基准包、混淆配置文件、资源Id文件，如下图所示：



> **实际应用中，请注意保存线上发布版本的基准apk包、mapping文件、R.txt文件，如果线上版本有bug，就可以借助我们tinker-support插件进行补丁包的生成。**

启动apk，上报联网数据

我们每次冷启动都会请求补丁策略，会上报当前版本号和tinkerId，这样我们后台就能将这个唯一的tinkerId对应到一个版本，大家测试的时候可以打开logcat查看我们的日志，如下图所示：



2、对基线版本的bug修复

未修复前

```java
public class BugClass {

    public String bug() {
        // 这段代码会报空指针异常
        String str = null;
        Log.e("BugClass", "get string length:" + str.length());
        return "This is a bug class";
    }
}
```

这个类有一个会造成空指针的方法。

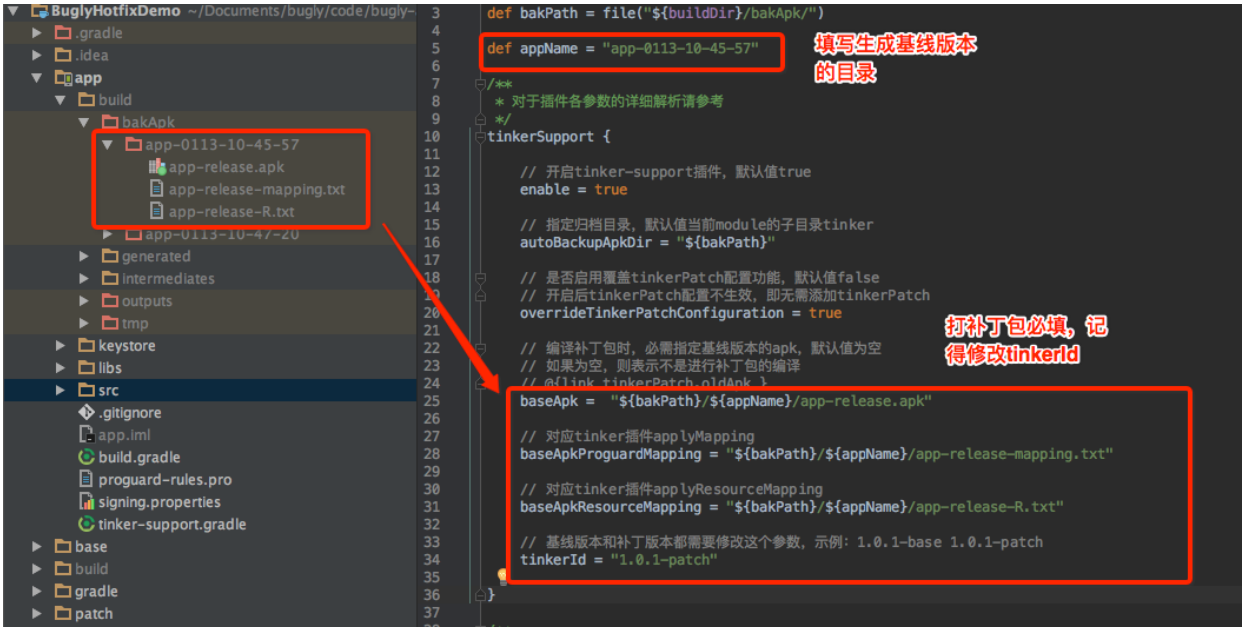
修复后

```java
public class BugClass {

    public String bug() {
        // 这段代码会报空指针异常
        // String str = null;
        // Log.e("BugClass", "get string length:" + str.length());
        return "This is a fixed bug class";
    }
}
```

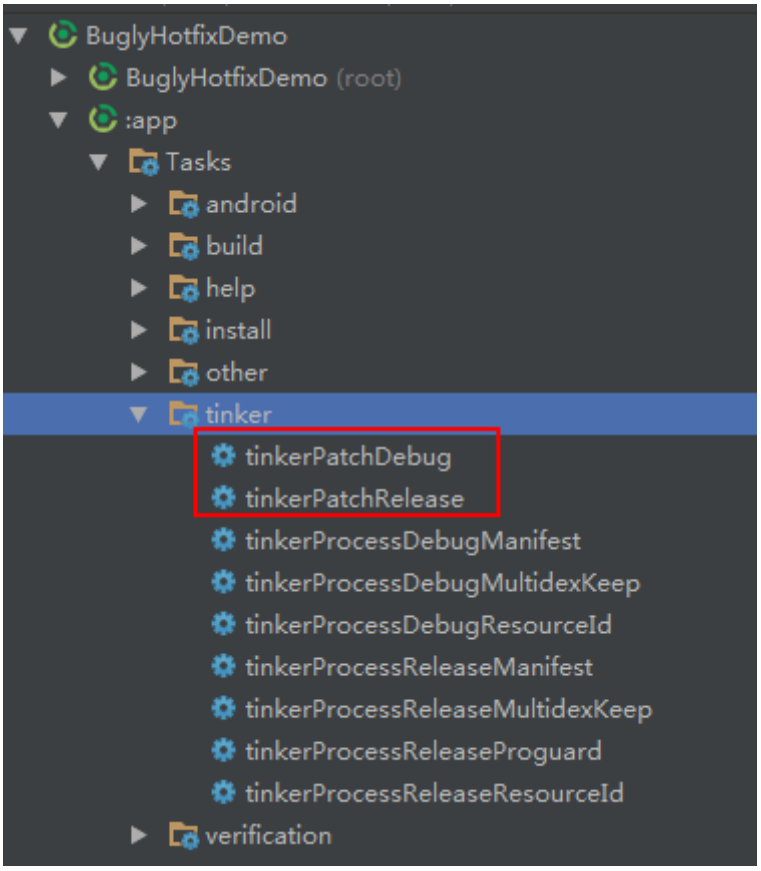
对产生bug的类进行修复，作为补丁下次覆盖基线版本的类。

3、根据基线版本生成补丁包

修改待修复apk路径、mapping文件路径、resId文件路径

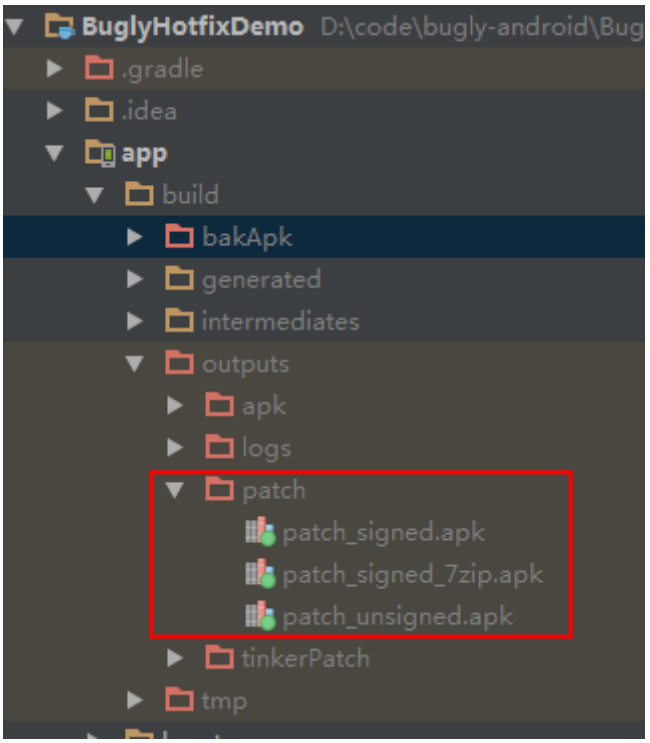


执行构建补丁包的task



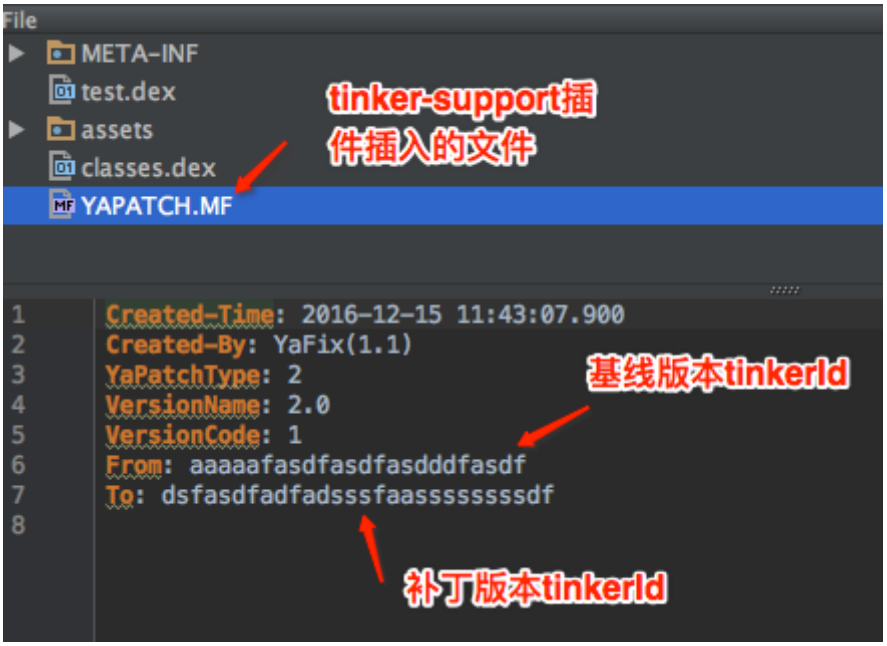
如果你要生成不同编译环境的补丁包，只需要执行Tinker插件生成的task，比如`tinkerPatchRelease`就能生成release编译环境的补丁包。

生成的补丁包在**build/outputs/patch**目录下：



大家这里可能会有一个疑问，补丁版本是怎么匹配到目标版本的，可以双击patch包，我们提供的插件会在tinker生成的patch包基础上插入一个MF文件：

```
----------------------Tinker patch end-----------------------
------ Tinker Support ------
Tinker patch output dir: /Users/devilwwj/Documents/bugly/code/bugly-android/BuglyHotfixDemo/app/build/outputs/tinkerPatch/release
Get TINKER_ID = aaaaafasdfasdfasdddfasdf, NEW_TINKER_ID = dsfasdfadfadsssfaasssssssdf
Generate patch description file: YAPATCH.MF
Tinker patch file: /Users/devilwwj/Documents/bugly/code/bugly-android/BuglyHotfixDemo/app/build/outputs/tinkerPatch/release/patch_signed.apk
Copy patch_signed.apk to /Users/devilwwj/Documents/bugly/code/bugly-android/BuglyHotfixDemo/app/build/outputs/patch/release/patch_signed.apk
Add YAPATCH.MF into the file: patch_signed.apk
Tinker patch file: /Users/devilwwj/Documents/bugly/code/bugly-android/BuglyHotfixDemo/app/build/outputs/tinkerPatch/release/patch_signed_7zip.apk
Copy patch_signed_7zip.apk to /Users/devilwwj/Documents/bugly/code/bugly-android/BuglyHotfixDemo/app/build/outputs/patch/release/patch_signed_7zip.apk
Add YAPATCH.MF into the file: patch_signed_7zip.apk
Tinker patch file: /Users/devilwwj/Documents/bugly/code/bugly-android/BuglyHotfixDemo/app/build/outputs/tinkerPatch/release/patch_unsigned.apk
Copy patch_unsigned.apk to /Users/devilwwj/Documents/bugly/code/bugly-android/BuglyHotfixDemo/app/build/outputs/patch/release/patch_unsigned.apk
Add YAPATCH.MF into the file: patch_unsigned.apk
Delete the patch description file: YAPATCH.MF
------ Tinker Support end ------

BUILD SUCCESSFUL
```



4、上传补丁包到平台

上传补丁包到平台并下发编辑规则


Bugly
buglyDemo
异常上报
热更新
全量更新
热更新
发布新补丁
所有状
目标版本
补丁
null.0
0.34
6.0.9
0.5
6.0.8
0.37
6.0.6
0.36

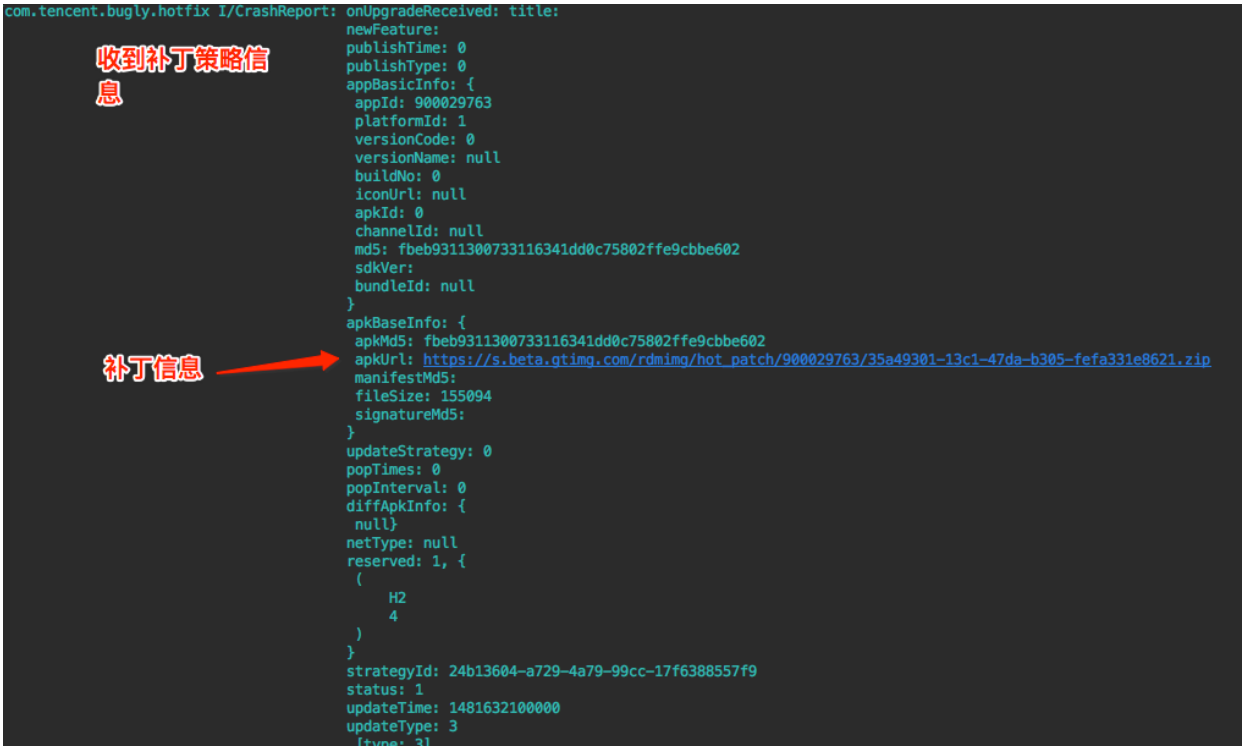

打开
BuglyHotfixDemo ▸ app ▸ build ▸ outputs ▸ patch
搜索 patch
组织
新建文件夹
下载
桌面
最近访问的位置
库
Subversion
视频
图片
文档
音乐
计算机
本地磁盘 (C:)
本地磁盘 (D:)
本地磁盘 (E:)
名称
修改日期
类型
大小
patch_signed.apk 2016/11/15 16:44 Android 程序安... 4 KB
patch_signed_7zip.apk 2016/11/15 16:44 Android 程序安... 4 KB
patch_unsigned.apk 2016/11/15 16:44 Android 程序安... 2 KB
文件名(N): patch_signed_7zip.apk
自定义文件
打开(O)
取消

点击 发布新补丁 ，上传前面生成的patch包，我们平台会自动为你匹配到目标版本，你可以选择下发范围（开发设备、全量设备、自定义），填写完备注之后，点击立即下发让补丁生效，这样你就可以在客户端当中收到我们的策略，SDK会自动帮你把补丁包下到本地。

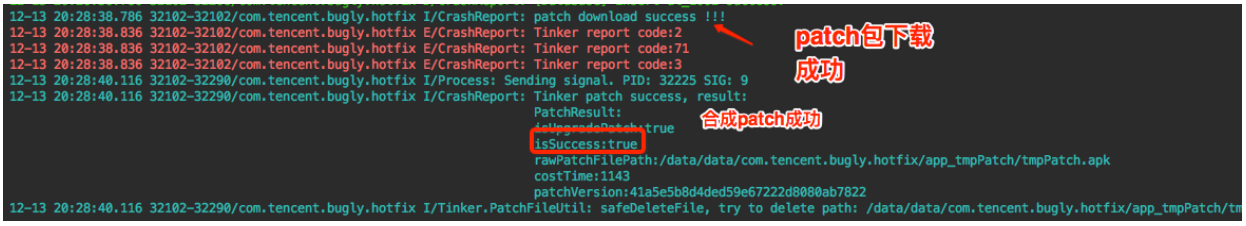
5、测试补丁应用效果

启动app应用patch

如果匹配到目标版本，后台就会下发补丁策略，可以在logcat看到如下日志：



下载成功之后，我们会立即去合成补丁，可以看到patch合成的日志：



重启app查看效果

注：我们方案是基于Tinker方案的实现，需要下次启动才能让补丁生效。

多渠道打包

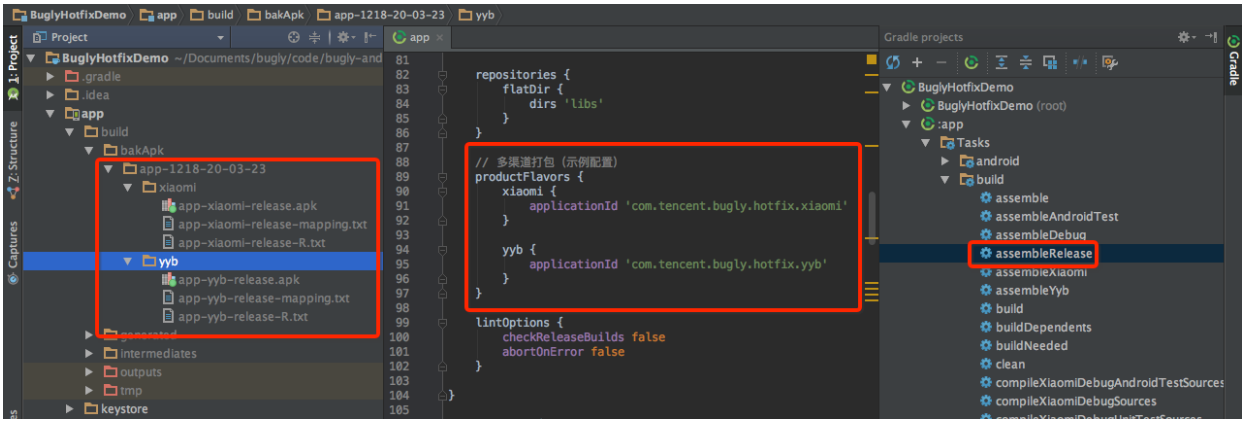
tinker是支持我们打多渠道的，建议大家按照以下步骤进行最佳实践：

1. 配置productFlavors

```
android {
    ...

    // 多渠道打包（示例配置）
    productFlavors {
        xiaomi {
            applicationId 'com.tencent.bugly.hotfix.xiaomi'
        }

        yyb {
            applicationId 'com.tencent.bugly.hotfix.yyb'
        }
    }

  ...

}
```

2. 执行`assembleRelease`生成基线apk

按照普通打包方式正常配置基线版本的tinkerId，然后执行assembleRelease生成不同渠道的apk，会在工程中build/bakApk/生成如下图所示文件：



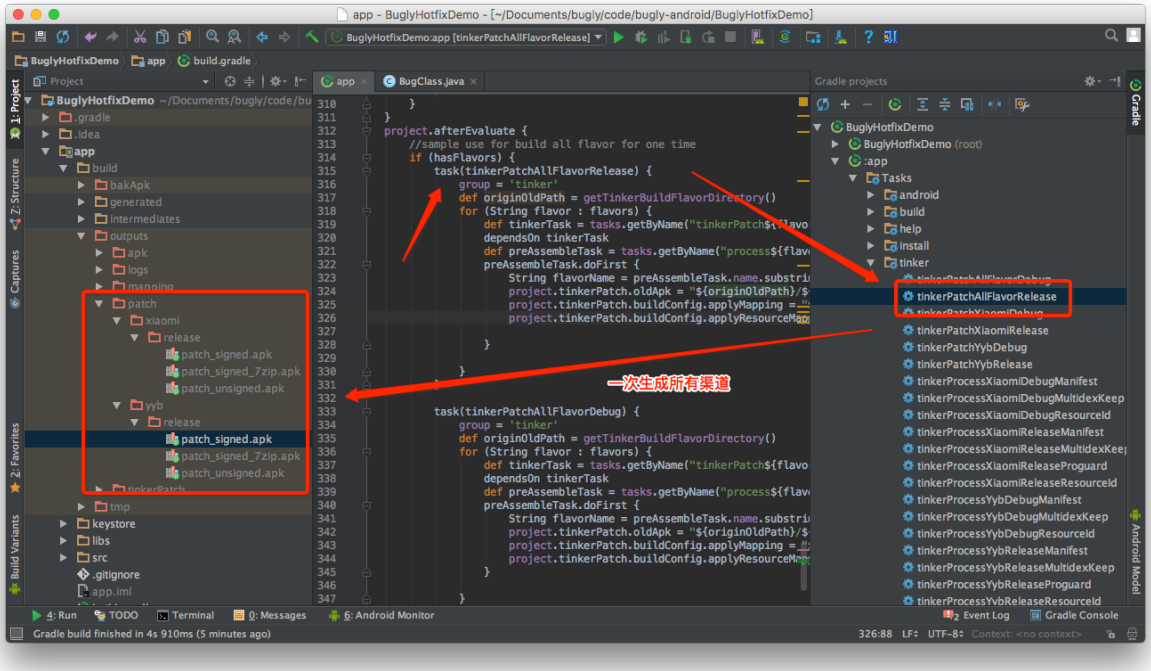
3. 打渠道补丁包配置

```
ext {
    // for some reason, you may want to ignore tinkerBuild, such as insta
nt run debug build?
    tinkerEnabled = true

    // for normal build
    // old apk file to build patch apk
    // tinkerOldApkPath = "${bakPath}/app-release-1215-11-41-02.apk"
    // proguard mapping file to build patch apk
    // tinkerApplyMappingPath = "${bakPath}/app-release-1215-11-41-02-map
ping.txt"
    // resource R.txt to build patch apk, must input if there is resource
 changed
    // tinkerApplyResourcePath = "${bakPath}/app-release-1215-11-41-02-R.
txt"

    // only use for build all flavor, if not, just ignore this field
    tinkerBuildFlavorDirectory = "${bakPath}/app-1218-20-03-23"
}
```

只需要配置tinkerBuildFlavorDirectory即可，其他的几个字段不需要填写，这里会自动根据路径拼接;补丁包的tinkerId按照普通打包方式填写即可。

4.执行`tinkerPatchAllFlavorRelease`生成所有渠道补丁包

如下图所示：



5.测试应用补丁包

与普通打包一致。

加固打包（仅支持tinker 1.7.5以下）

> tinker的一般模式需要Dex的合成，它并不支持加固，一定要使用加固的app可以使用usePreGeneratedPatchDex模式。由于加固会改变apk的dex结构，所以生成补丁包时我们务必要使用加固前的apk。
> **但是需要注意的是，某些加固工具会将非exported的四大组件的类名替换，对于这部分类即使使用usePreGeneratedPatchDex也无法修改。对于360加固，MainActivity由于被提前加载，也无法修复。大家对于加固的情况，请仔细测试，能否支持与加固的方式有关联。**

1.提前生成dex配置

tinker是支持加固模式的，但需要你回退到Qzone方案 ，将usePreGeneratedPatchDex设置为true。

```
// dex相关配置项
dex {
    dexMode = "jar" // 可选，默认为jar
    usePreGeneratedPatchDex = true // 可选，默认为false
    pattern = ["classes*.dex",
               "assets/secondary-dex-?.jar"]
    // 必选
    loader = ["com.tencent.tinker.loader.*",
              "com.tencent.bugly.hotfix.SampleApplication",
          ]
}
```

> 是否提前生成dex，而非合成的方式。这套方案即**回退成Qzone的方案**，对于需要使用加固或者多flavor打包(建议使用其他方式生成渠道包)的用户可使用。但是这套方案需要**插桩**，**会造成Dalvik下性能损耗以及Art补丁包可能过大的问题，务必谨慎使用。另外一方面，这种方案在Android N之后可能会产生问题，建议过滤N之后的用户。**

2.将基准包进行加固

如果你的app需要进行加固，你需要将你打出的基准包上传到具体的加固平台进行加固，例如乐固，加固完成之后需要对apk进行**重签名**：

```
jarsigner -verbose -keystore <YOUR_KEYSTORE> -signedjar <SIGNED_APK> <UNSIGNED_APK> <KEY_ALIAS>
```

> 以上命令说明：
>
> -verbose：指定生成详细输出
>
> -keystore：指定证书存储路径
>
> -signedjar：改选项的三个参数分别为签名后的apk包、未签名的apk包、数字证书别名

3.根据加固的基准包生成补丁包

打patch包的操作跟普通打包方式一致。