Bugly Android热更新使用指南

Bugly文档资料

介绍

热更新能力是Bugly为解决开发者紧急修复线上bug，而无需重新发版让用户无感知就能把问题修复的一项能力。Bugly目前采用**微信Tinker**的开源方案，开发者只需要集成我们提供的SDK就可以实现自动下载补丁包、合成、并应用补丁的功能，我们也提供了热更新管理后台让开发者对每个版本补丁进行管理。

添加插件依赖

工程根目录下“build.gradle”文件中添加：

```
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        // tinker gradle插件
        classpath ('com.tencent.tinker:tinker-patch-gradle-plugin:1.7.5')

        // tinkersupport插件
        classpath "com.tencent.bugly:tinker-support:1.0.1"
    }
}
```

注意：当前我们版本需要你指定tinker插件版本为1.7.5，避免因为插件版本的变更导致补丁包的生成的问题。

在app module的“build.gradle”文件中添加（示例配置）：

```
apply plugin: 'com.tencent.bugly.tinker-support'
// tinker-support 默认已添加配置apply plugin: 'com.tencent.bugly.tinker'

// 注意：必须要配置tinker-support
tinkerSupport {
    enable = true
}

// 全局信息相关配置项
tinkerPatch {
    oldApk = "YOUR_OLD_APK" //打补丁包时必选， 基准包路径

    ignoreWarning = false // 可选，默认false

    useSign = true // 可选，默认true， 验证基准apk和patch签名是否一致

    // 编译相关配置项
    buildConfig {
        applyMapping = "YOUR_OLD_APK_PROGUARD_FILE" //  可选，设置mapping文件，建议保持旧apk的proguard混淆方式
        applyResourceMapping = "YOUR_OLD_APK_RES_MAPPING"  // 可选，设置R.txt文件，通过旧apk文件保持ResId的分配
        tinkerId = "YOUR_TINKER_ID"  // 必选，当前版本的唯一标识，可以是git版本号、versionName
    }

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

    // lib相关的配置项
    lib {
         pattern = ["lib/armeabi/*.so"]
    }

    // res相关的配置项
    res {
        pattern = ["res/*", "assets/*", "resources.arsc", "AndroidManifest.xml"]
        ignoreChange = ["assets/sample_meta.txt"]
        largeModSize = 100
    }

    // 用于生成补丁包中的'package_meta.txt'文件
    packageConfig {
        configField("patchMessage", "tinker is sample to use")

        configField("platform", "all")

        configField("patchVersion", "1.0")
    }
```

```
    // 7zip路径配置项，执行前提是useSign为true
    sevenZip {
        zipArtifact = "com.tencent.mm:SevenZip:1.1.10" // optional
        //  path = "/usr/local/bin/7za" // optional
    }
}
```

tinker插件参数示例

参考tinker-sample-android中的gradle配置:https://github.com/Tencent/tinker/blob/master/tinker-sample-android/app/build.gradle

gradle参数说明

tinker插件包含以下项的配置

- tinkerPatch - 全局配置项
- buildConfig - 编译相关配置项
- dex - dex相关配置
- lib - lib相关配置
- res - res相关配置
- packageConfig - package_meta.txt文件配置
- sevenZip - 7zip路径配置项

更详细的配置说明请参考Tinker-接入指南。

如果你以前使用tinker插件，你必须要在使用tinkerPatch配置前配置tinkerSupport，否则你打出的补丁包将不能上传到Bugly平台。

如果你想更加方便使用插件配置，可以直接使用我们提供的tinker-support插件进行配置，详细参考tinker-support插件的使用介绍。

集成SDK

gradle配置

在app module的“build.gradle”文件中添加（示例配置）：

```
 dependencies {
          // 多dex配置
          compile "com.android.support:multidex:1.0.1"
          compile 'com.tencent.bugly:crashreport_upgrade:latest.release'//其中latest.release指代最新版本号，也可以指定明确的版本号，例如1.2.0
      }
```

集成Bugly升级SDK之后，我们需要按照以下方式自定义ApplicationLike来实现Application的代码（以下是示例）：

自定义Application

```
public class SampleApplication extends TinkerApplication {
    public SampleApplication() {
        super(ShareConstants.TINKER_ENABLE_ALL, "xxx.xxx.SampleApplicationLike",
                "com.tencent.tinker.loader.TinkerLoader", false);
    }
}
```

注意：这个类集成TinkerApplication类，这里面不做任何操作，所有Application的代码都会放到ApplicationLike继承类当中

参数解析

参数1：tinkerFlags 表示Tinker支持的类型 dex only、library only or all suuport，default: TINKER_ENABLE_ALL

参数2：delegateClassName Application代理类 这里填写你自定义的ApplicationLike

参数3：loaderClassName Tinker的加载器，使用默认即可

参数4：tinkerLoadVerifyFlag 加载dex或者lib是否验证md5，默认为false

自定义ApplicationLike

```
public class SampleApplicationLike extends DefaultApplicationLike {

    public static final String TAG = "Tinker.SampleApplicationLike";

    public SampleApplicationLike(Application application, int tinkerFlags,
            boolean tinkerLoadVerifyFlag, long applicationStartElapsedTime,
            long applicationStartMillisTime, Intent tinkerResultIntent, Resources[] re
sources,
            ClassLoader[] classLoader, AssetManager[] assetManager) {
        super(application, tinkerFlags, tinkerLoadVerifyFlag, applicationStartElapsedT
ime,
                applicationStartMillisTime, tinkerResultIntent, resources, classLoade
r,
                assetManager);
    }


    @Override
    public void onCreate() {
        super.onCreate();
        // 这里实现SDK初始化，appId替换成你的在Bugly平台申请的appId
        Bugly.init(getApplication(), "900029763", true);
    }


    @TargetApi(Build.VERSION_CODES.ICE_CREAM_SANDWICH)
    @Override
    public void onBaseContextAttached(Context base) {
        super.onBaseContextAttached(base);
        // you must install multiDex whatever tinker is installed!
        MultiDex.install(base);

        // 安装tinker
        // TinkerManager.installTinker(this); 替换成下面Bugly提供的方法
        Beta.installTinker(this);
    }

    @TargetApi(Build.VERSION_CODES.ICE_CREAM_SANDWICH)
    public void registerActivityLifecycleCallback(Application.ActivityLifecycleCallbac
ks callbacks) {
        getApplication().registerActivityLifecycleCallbacks(callbacks);
    }

}
```

> 注意：tinker需要你开启MultiDex,你需要在dependencies中进行配置 `compile "com.android.support:multidex:1.0.1"` 才可以使用MultiDex.install方法； SampleApplicationLike这个类是Application的代理类，以前所有在Application的实现必须要全部拷贝到这里，在 `onCreate` 方法调用SDK的初始化方法，在 `onBaseContextAttached` 中调用 `Beta.installTinker(this);` 。

统一初始化方法

```
    Bugly.init(getApplicationContext(), "注册时申请的APPID", false);
```

AndroidManifest.xml配置

1. 权限配置

```
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.READ_LOGS" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

2. Activity配置

```
<activity
    android:name="com.tencent.bugly.beta.ui.BetaActivity"
    android:theme="@android:style/Theme.Translucent" />
```

3. 配置FileProvider（Android N之后配置）

注意：如果您的工程targetSdkVersion为N或者以后的版本，必须要在AndroidManifest.xml文件中配置FileProvider来访问共享路径的文件。如果您暂时不需要适配Android 7.0以上的设备，将targetSdkVersion修改为低于24即可。

```
 <provider
    android:name="android.support.v4.content.FileProvider"
    android:authorities="${applicationId}.fileProvider"
    android:exported="false"
    android:grantUriPermissions="true">
    <meta-data
        android:name="android.support.FILE_PROVIDER_PATHS"
        android:resource="@xml/provider_paths"/>
</provider>
```

${applicationId}请替换为您的包名，例如com.bugly.upgrade.demo。这里要注意一下，FileProvider类是在support-v4包中的，检查你的工程是否引入该类库。

在res目录新建xml文件夹，创建provider_paths.xml文件如下：

```
<?xml version="1.0" encoding="utf-8"?>
<paths xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- /storage/emulated/0/Download/${applicationId}/.beta/apk-->
    <external-path name="beta_external_path" path="Download/"/>
    <!--/storage/emulated/0/Android/data/${applicationId}/files/apk/-->
    <external-path name="beta_external_files_path" path="Android/data/"/>
</paths>
```

这里配置的两个外部存储路径是升级SDK下载的文件可能存在的路径，一定要按照上面格式配置，不然可能会出现错误。

混淆配置

为了避免混淆SDK，在Proguard混淆文件中增加以下配置：

```
-dontwarn com.tencent.bugly.**
-keep public class com.tencent.bugly.**{*;}
```

使用范例

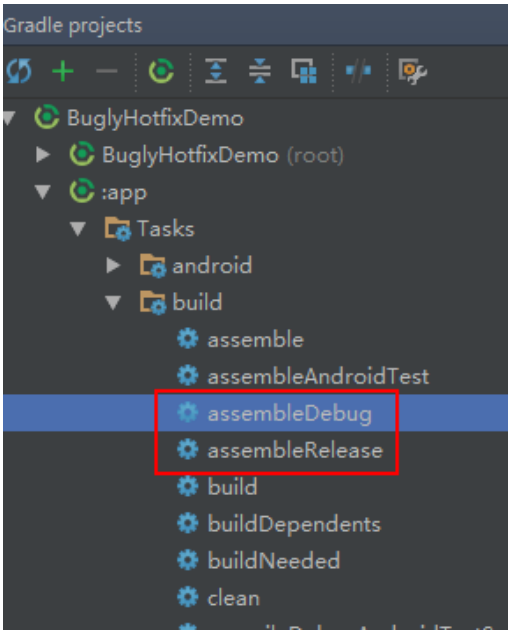
一、编译基准包

配置基准包的tinkerId

```
// 编译相关配置项
buildConfig {
    applyMapping = getApplyMappingPath() //  可选，设置mapping文件，建议保持旧apk的proguard混淆方式
    applyResourceMapping = getApplyResourceMappingPath() // 可选，设置R.txt文件，通过旧apk文件保持ResId的分配
    tinkerId = "55557784ce2aabfc5255da4a3c10319f8219c374" // 必选，默认为null
}
```

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。

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。

二、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;
        int 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

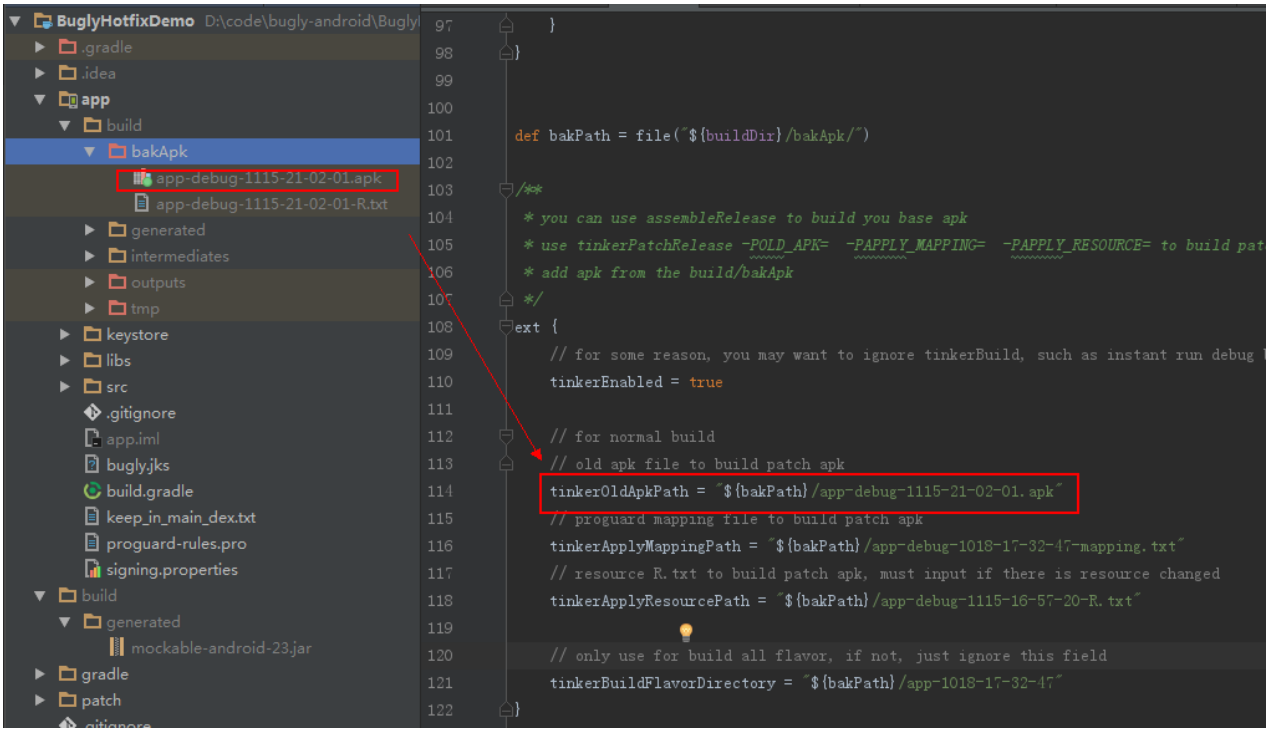
```java
public class BugClass {

    public String bug() {
        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。

三、根据基线版本生成补丁包

修改基线版本apk路径



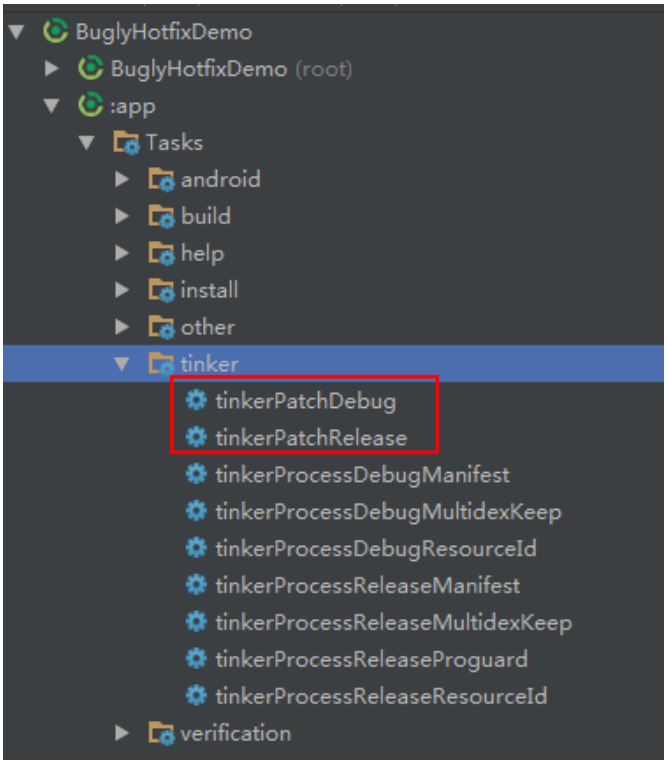
注：这个只是示例，你需要按照你自身的配置指定oldApk路径。 具体参考tinker-sample-android

填写补丁包的tinkerId、applyMapping、applyResourceMapping

```
// 编译相关配置项
buildConfig {
    applyMapping = getApplyMappingPath() //  可选，设置mapping文件，建议保持旧apk的proguard混淆方式
    applyResourceMapping = getApplyResourceMappingPath() // 可选，设置R.txt文件，通过旧apk文件保持ResId的分配
      tinkerId = "55557784ce2aabfc5255da4a3c10319f8219c374" // 必选，默认为null
    tinkerId = "a8d7e290b04cc8e11453dc60bbcf93347ed2feb8"
}
```

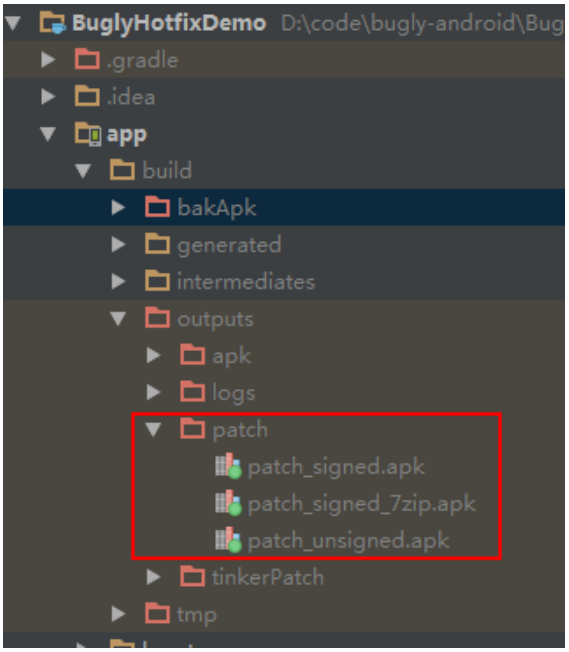
注意：如果你是编译release版本的补丁包，你需要填入基线版本生成的mapping和resId映射文件。

执行构建补丁包的task



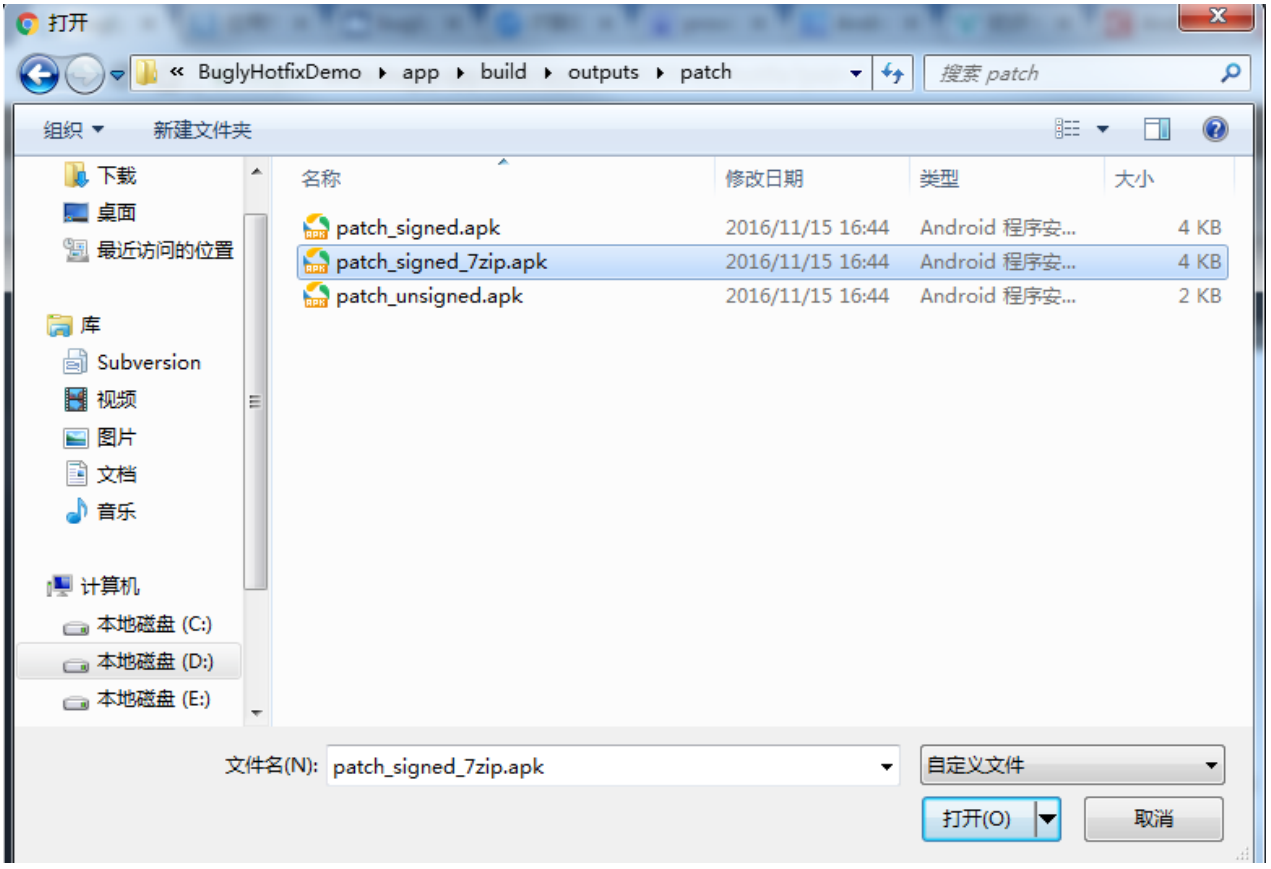
如果你要生成不同编译环境的补丁包，只需要执行Tinker插件生成的task，比如`tinkerPatchRelease`就能生成release编译环境的补丁包。

生成的补丁包在build/outputs/patch目录下：



四、上传补丁包到平台

上传补丁包到平台并下发编辑规则



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。

五、测试补丁应用效果

启动app应用patch


8:13
BuglyHotfixDemo
当前版本：6.0.0.3
显示结果
加载补丁
退出程序
很抱歉，“BuglyHotfixDemo”已停止运行。
确定

重启app查看效果

参数名	默认值	参数说明

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。

tinker-support插件使用介绍

如果你想使用我们的插件来配置的话，你可以不配置tinker插件的参数，而使用我们提供的参数，具体可以使用的配置项如下所示：

tinker-support插件配置

参数名	默认值	参数说明
enable	true	开启tinker-support插件
autoBackupApk	false	是否编译完成后，归档apk到指定目录

参数名	默认值	参数说明
backupApkDir	‘tinker’	指定归档目录，默认值当前module的子目录tinker
overrideTinkerPatchConfiguration	false	是否启用覆盖tinkerPatch配置功能,开启后tinkerPatch配置不生效，即无需添加tinkerPatch
baseApk	“”	对应tinker插件`oldApk`，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
ignoreWarning	false	如果出现以下的情况，并且ignoreWarning为false，我们将中断编译。因为这些情况可能会导致编译出来的patch包带来风险： 1. minSdkVersion小于14，但是`dexMode`的值为”raw”; 2. 新编译的安装包出现新增的四大组件(Activity, BroadcastReceiver…)； 3. 定义在dex.loader用于加载补丁的类不在main dex中; 4. 定义在dex.loader用于加载补丁的类出现修改； 5. resources.arsc改变，但没有使用applyResourceMapping编译。
patchSigning	true	对应tinker插件`userSign`，在运行过程中，我们需要验证基准apk包与补丁包的签名是否一致，我们是否需要为你签名。
baseApkProguardMapping	“”	对应tinker插件`applyMapping`，可选参数；在编译新的apk时候，我们希望通过保持旧apk的proguard混淆方式，从而减少补丁包的大小。这个只是推荐的，但`设置applyMapping会影响任何的assemble编译`。
baseApkResourceMapping	“”	对应tinker插件`applyResourceMapping`，可选参数；在编译新的apk时候，我们希望通过旧apk的`R.txt`文件保持ResId的分配，这样不仅`可以减少补丁包的大小`，同时也`避免由于ResId改变导致remote view异常`。
tinkerId	“”	在运行过程中，我们需要验证基准apk包的tinkerId是否等于补丁包的tinkerId。这个是决定补丁包能运行在哪些基准包上面，一般来说我们可以使用git版本号、versionName等等。
dexMode	“jar”	只能是’raw’或者’jar’。 对于’raw’模式，我们将会保持输入dex的格式。 对于’jar’模式，我们将会把输入dex重新压缩封装到jar。如果你的minSdkVersion小于14，你必须选择‘jar’模式，而且它更省存储空间，但是验证md5时比’raw’模式耗时()。

参数名	默认值	参数说明
dexPattern	[“classes*.dex”, “assets/secondary-dex-?.jar”]	需要处理dex路径，支持*、?通配符，必须使用’/’分割。路径是相对安装包的，例如/assets/…
dexLoader	[“com.tencent.tinker.loader.*”]	这一项非常重要，它定义了哪些类在加载补丁包的时候会用到。这些类是通过Tinker无法修改的类，也是一定要放在main dex的类。 这里需要定义的类有： 1. 你自己定义的Application类； 2. Tinker库中用于加载补丁包的部分类，即com.tencent.tinker.loader.*； 3. 如果你自定义了TinkerLoader，需要将它以及它引用的所有类也加入loader中； 4. 其他一些你不希望被更改的类，例如Sample中的BaseBuildInfo类。**这里需要注意的是，这些类的直接引用类也需要加入到loader中。或者你需要将这个类变成非preverify。**
usePreGeneratedPatchDex	false	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。
libPattern	[“lib/armeabi/*.so”]	需要处理lib路径，支持*、?通配符，必须使用’/’分割。与dex.pattern一致, 路径是相对安装包的，例如/assets/…
resPattern	[“res/*”, “assets/*”, “resources.arsc”, “AndroidManifest.xml”]	需要处理res路径，支持*、?通配符，必须使用’/’分割。与dex.pattern一致, 路径是相对安装包的，例如/assets/…，`务必注意的是，只有满足pattern的资源才会放到合成后的资源包。`
resIgnoreChange	[“assets/*_meta.txt”]	支持*、?通配符，必须使用’/’分割。若满足ignoreChange的pattern，在编译时会忽略该文件的新增、删除与修改。 **最极端的情况，ignoreChange与上面的pattern一致，即会完全忽略所有资源的修改。**
resLargeModSize	100	对于修改的资源，如果大于largeModSize，我们将使用bsdiff算法。这可以降低补丁包的大小，但是会增加合成时的复杂度。默认大小为100kb

参数名	默认值	参数说明
configField	TINKER_ID, NEW_TINKER_ID	configField(“key”, “value”), 默认我们自动从基准安装包与新安装包的Manifest中读取tinkerId,并自动写入configField。在这里，你可以定义其他的信息，在运行时可以通过TinkerLoadResult.getPackageConfigByName得到相应的数值。但是建议直接通过修改代码来实现，例如BuildConfig。
sevenZipArtifact	“com.tencent.mm:SevenZip:1.1.10”	例如”com.tencent.mm:SevenZip:1.1.10”，将自动根据机器属性获得对应的7za运行文件，推荐使用。
sevenZipExecutePath	“/usr/local/bin/7za”	系统中的7za路径，例如”/usr/local/bin/7za”。path设置会覆盖zipArtifact，若都不设置，将直接使用7za去尝试。

FAQ

Q：之前使用Tinker怎么切换过来使用Bugly？

A：Bugly使用源码集成Tinker，如果之前集成过Tinker，你需要注释掉以下配置：

```
//    compile("com.tencent.tinker:tinker-android-lib:${TINKER_VERSION}") { changing =
true }
//    provided("com.tencent.tinker:tinker-android-anno:${TINKER_VERSION}") { changing
 = true }
```

插件配置不需要更改，只需要加上我们Bugly额外的tinker-support插件即可：

```
    // tinker gradle插件
    classpath ('com.tencent.tinker:tinker-patch-gradle-plugin:1.7.5')

    // tinkersupport插件
    classpath "com.tencent.bugly:tinker-support:1.0.0"
```

Q： 基线版本表示什么意思?

A：表示你需要修复apk的版本，比如你已经上线了某个版本的apk，你需要用一个唯一的tinkerId来标识这个版本，而补丁包也是基于这个版本打的。

Q：打一个补丁包需要改哪些东西？

A：

> 1. 修复bug的类、修改资源
> 2. 修改oldApk配置
> 3. 修改tinkerId

Q：tinkerId该怎么填？

A：在运行过程中，我们需要验证基准apk包的tinkerId是否等于补丁包的tinkerId。这个是决定补丁包能运行在哪些基准包上面，一般来说我们可以使用git版本号、versionName等等。

Q：两次传入的tinkerId是否一样？

A：不一样的，编译补丁包时，tinker会自动读取基准包AndroidManifest的tinkerId作为package_meta.txt中的TINKER_ID。将本次编译传入的tinkerId，作为package_meta.txt的NEW_TINKER_ID。

Q. 为什么我上传补丁提示我“未匹配到可用补丁的App版本”？

A：如果你的基线版本没有上报过联网，基于这个版本生成的补丁包就无法匹配到，请检查你的基线版本配置是否正确。

Q：我以前的是Bugly SDK，现在集成升级SDK会有什么影响？

A：不会有影响的，升级SDK内置Bugly功能模块，你只需要将初始化方法改为统一的`Bugly.init(getApplicationContext(), "注册时申请的APPID", false);`即可。

Q：你们是怎么定义开发设备的？

A：我们会提供接口`Bugly.setIsDevelopmentDevice(getApplicationContext(), true);`，我们后台就会将你当前设备识别为开发设备，如果设置为false则非开发设备，我们会根据这个配置进行策略控制。