# Bugly Android热更新使用指南



# 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。

**为什么使用Bugly热更新？**

- 无需关注Tinker是如何合成补丁的
- 无需自己搭建补丁管理后台
- 无需考虑后台下发补丁策略的任何事情
- 无需考虑补丁下载合成的时机，处理后台下发的策略
- 我们提供了更加方便集成Tinker的方式
- 我们通过HTTPS及签名校验等机制保障补丁下发的安全性
- 丰富的下发维度控制，有效控制补丁影响范围
- 我们提供了应用升级一站式解决方案

# 第一步：添加插件依赖

工程根目录下“build.gradle”文件中添加：

```
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        // tinkersupport插件，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.
0.3
        classpath "com.tencent.bugly:tinker-support:latest.release"
    }
}
```

注意：当前我们版本需要你指定tinker插件版本为1.7.6，避免因为插件版本的变更导致补丁包的生成的问题。 自**SDK 1.2.2**版本起无需再配tinker插件的classpath。

# 第二步：集成SDK

**gradle配置**

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

```
 dependencies {

          compile "com.android.support:multidex:1.0.1" // 多dex配置
          compile 'com.tencent.bugly:crashreport_upgrade:latest.release'//其中latest.re
lease指代最新版本号，也可以指定明确的版本号，例如1.2.1
}
```

在app module的“build.gradle”文件中添加：

```
// 依赖插件脚本
apply from: 'tinker-support.gradle'
```

**tinker-support.gradle内容如下所示（示例配置）：**

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

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

def appName = "app-0111-15-18-41"

/**
 * 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"

    // 唯一标识当前版本
    tinkerId = "1.0.1-base"

    // 是否开启代理Application，设置之后无须改造Application，默认为false
    enableProxyApplication = false
}
```

更详细的配置项参考tinker-support配置说明

# 第三步：初始化SDK

## enableProxyApplication = false 的情况

> 这是Tinker推荐的接入方式，一定程度上会增加接入成本，但具有更好的兼容性。

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

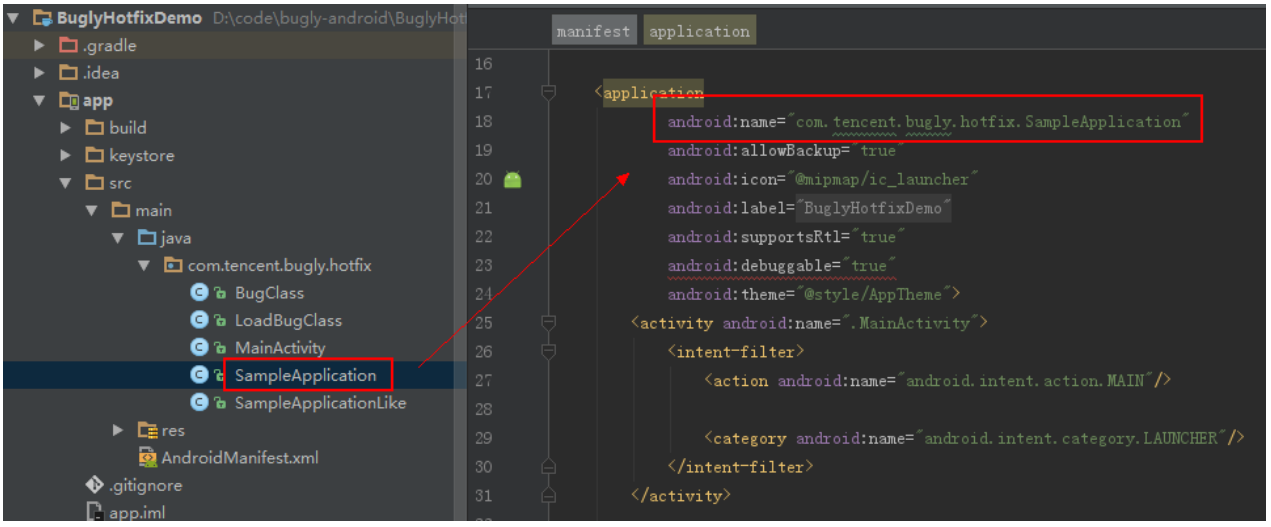
参数1：tinkerFlags 表示Tinker支持的类型 dex only、library only or all suuport，default: TINKER_ENABLE_ALL

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

我们需要您将以前的Applicaton配置为继承TinkerApplication的类：




**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);` 。

## enableProxyApplication = true 的情况

```
public class MyApplication extends Application {

    @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(this, "900029763", true);
    }

    @Override
    protected void attachBaseContext(Context base) {
        super.attachBaseContext(base);

        // 安装tinker
        Beta.installTinker();
    }

}
```

> 注：无须你改造Application，主要是为了降低接入成本，我们插件会动态替换AndroidMinifest文件中的Application为我们定义好用于反射真实Application的类（需要您接入**SDK 1.2.2版本** 和 **插件版本 1.0.3**以上）。

**统一初始化方法**

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

# 第四步：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"/>
```

注意：如果你也想使用升级功能，你必须要进行2、3项的配置，而如果你只想使用热更新能力，你只需要配置权限即可。

**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：如果您想兼容Android N或者以上的设备，必须要在AndroidManifest.xml文件中配置FileProvider来访问共享路径的文件。

```
 <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。

# 第五步：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.**{*;}
```

如果你使用了support-v4包，你还需要配置以下混淆规则：

```
-keep class android.support.**{*;}
```

# 使用范例

大家如果想测试验证热更新能力，请仔细查看以下文档：

热更新使用范例

# tinker-support插件使用介绍

| 参数名 | 默认值 | 参数说明 |
| --- | --- | --- |

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 |
| autoBackupApkDir | ‘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：<br>1. minSdkVersion小于14，但是`dexMode`的值为”raw”;<br>2. 新编译的安装包出现新增的四大组件(Activity, BroadcastReceiver…)；<br>3. 定义在dex.loader用于加载补丁的类不在main dex中;<br>4. 定义在dex.loader用于加载补丁的类出现修改；<br>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。** |

| 参数名 | 默认值 | 参数说明 |
| --- | --- | --- |
| 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’。<br>对于’raw’模式，我们将会保持输入dex的格式。<br>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()。 |
| 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。<br>这里需要定义的类有：<br>1. 你自己定义的Application类；<br>2. Tinker库中用于加载补丁包的部分类，即com.tencent.tinker.loader.*；<br>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；<br>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。** |
| 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。 |
| enableProxyApplication | false | 是否使用proxy/delegate application 模式 |