Bugly Android 应用升级 SDK 使用指南

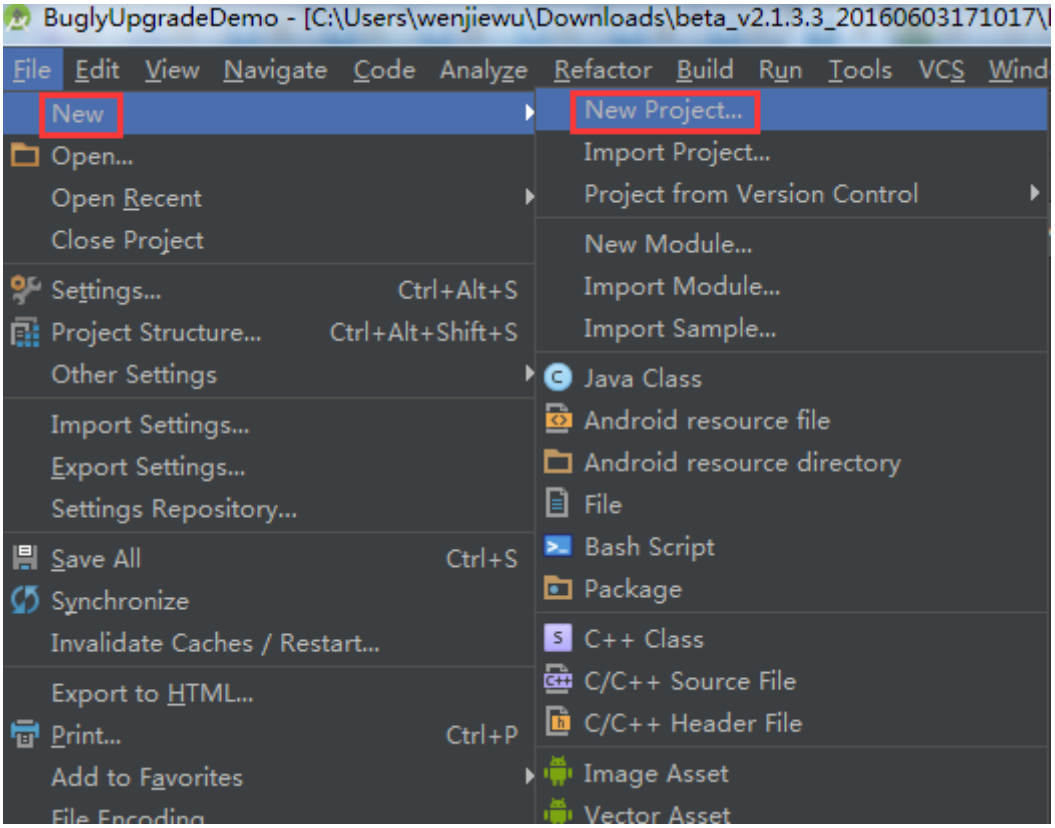
概述

升级功能是专为App的灰度升级而开发的组件，在bugly内测页面配置好App的更新策略，策略指定的老版本App在启动时会自动检测更新并提示升级，为团队的应用分发，灰度内测提供一站式解决方案。

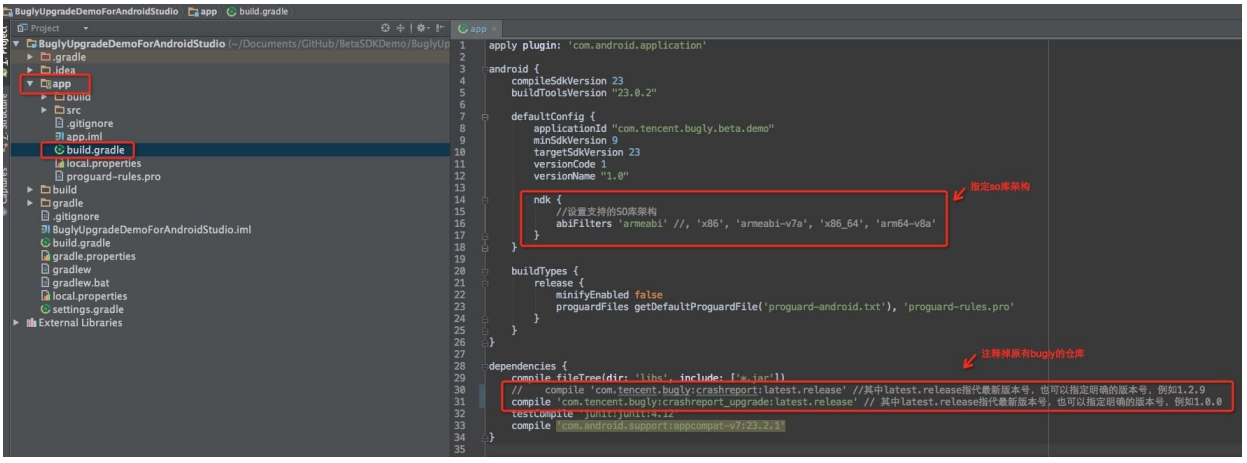
自动导入（推荐）

使用Android Studio创建project

【File】-> 【New】->【New Project】



gradle配置（重要）



配置示例（路径app/build.gradle）：

```
  android {
        defaultConfig {
          ndk {
            //设置支持的SO库架构
            abiFilters 'armeabi' //, 'x86', 'armeabi-v7a', 'x86_64', 'arm
64-v8a'
          }
        }
      }
      dependencies {
          //注释掉原有bugly的仓库
          //compile 'com.tencent.bugly:crashreport:latest.release'//其中la
test.release指代最新版本号，也可以指定明确的版本号，例如2.3.2
          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
          compile 'com.tencent.bugly:nativecrashreport: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2.2.0
      }
```

后续更新升级SDK时，只需变更配置脚本中的版本号即可。

> 注意：
> 升级SDK已经集成crash上报功能，已经集成Bugly的用户需要注释掉原来Bugly的jcenter库；
> 已经配置过符号表的Bugly用户保留原有符号表配置；
> Bugly SDK（2.1.5及以上版本）已经将Java Crash和Native Crash捕获功能分开，如果想使用NDK库，需要配置：
> compile ‘com.tencent.bugly:nativecrashreport:latest.release’

手动导入aar

我们也支持手动集成升级SDK，毋须按照下面步骤进行接入。

下载SDK库文件

SDK下载

点击下载即可，解压缩会发现包含以下文件：

- 需要集成的aar包
- 接入文档
- Demo示例

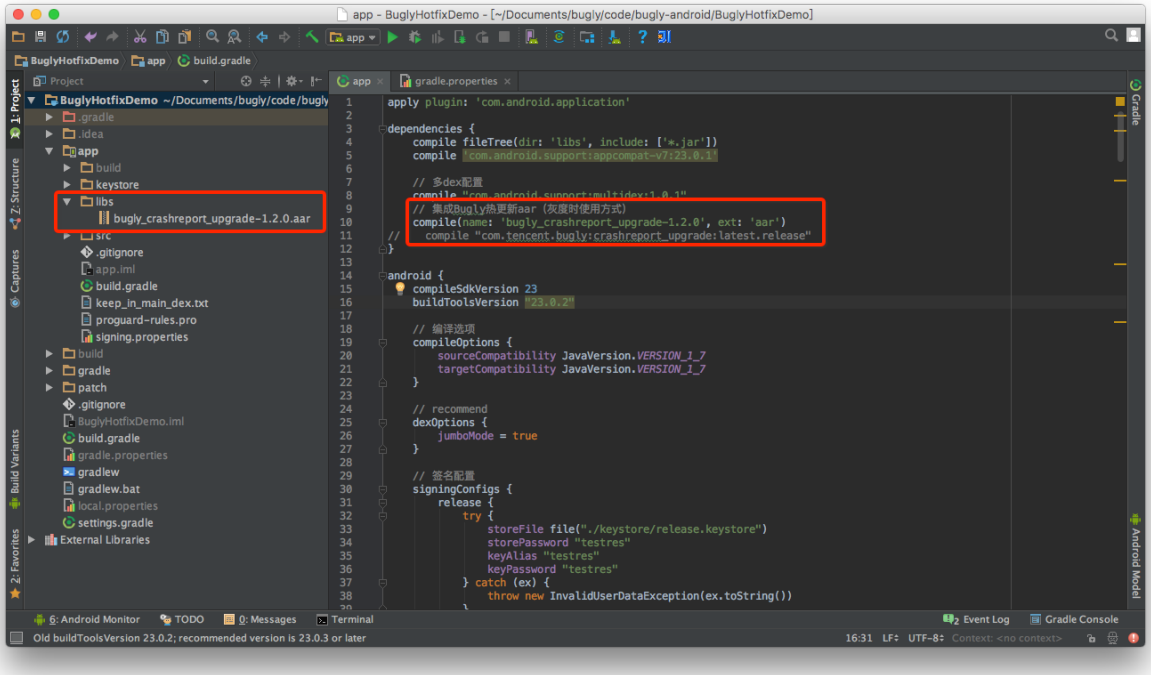
构建aar编译路径

Android Studio工程

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile(name: 'bugly_crashreport_upgrade-1.2.0', ext: 'aar')    }

android {
    repositories {
        flatDir {
            dirs 'libs'
        }
}
```

如下图所示：



注：升级SDK自1.2.0版本将不再支持Eclipse集成方式，建议使用gradle远程仓库集成和aar导入集成。

参数配置

在AndroidMainfest.xml中进行以下配置：

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" />
<!--保存资源到SD卡-->
<uses-permission android:name="android.permission.WRITE_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。

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.**{*;}
```

> 注意：
> 已经接入Bugly SDK的用户需要先删除原Bugly SDK的jar包；
> android4.1以上的工程必须把jar包放在libs目录下，否则会出现NoClassDefFoundError错误;
>
> 如果您的工程有Native代码（C/C++）或者集成了其他第三方SO库，建议下载Bugly的NDK动态库。
>
> Bugly NDK包含多个架构的SO库：
>
> - armeabi
> - armeabi-v7a
> - arm64-v8a
> - x86
> - x86_64
>
> **在集成Bugly SO库时，请注意只保留支持的架构SO库。**

测试验证

SDK初始化

注意：如果您之前使用过Bugly SDK，请将以下这句注释掉。

```
CrashReport.initCrashReport(getApplicationContext(), "注册时申请的APPID", false);
```

统一初始化方法：

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

参数解析：

参数1：上下文对象

参数2：注册时申请的APPID

参数3：是否开启debug模式，true表示打开debug模式，false表示关闭调试模式

> 提示：已经接入Bugly用户改用上面的初始化方法,不影响原有的crash上报功能;
> init方法会自动检测更新，不需要再手动调用Beta.checkUpgrade(), 如需增加自动检查时机可以使用Beta.checkUpgrade(false,false);
>
> 参数1：isManual 用户手动点击检查，非用户点击操作请传false
>
> 参数2：isSilence 是否显示弹窗等交互，[true:没有弹窗和toast] [false:有弹窗或toast]

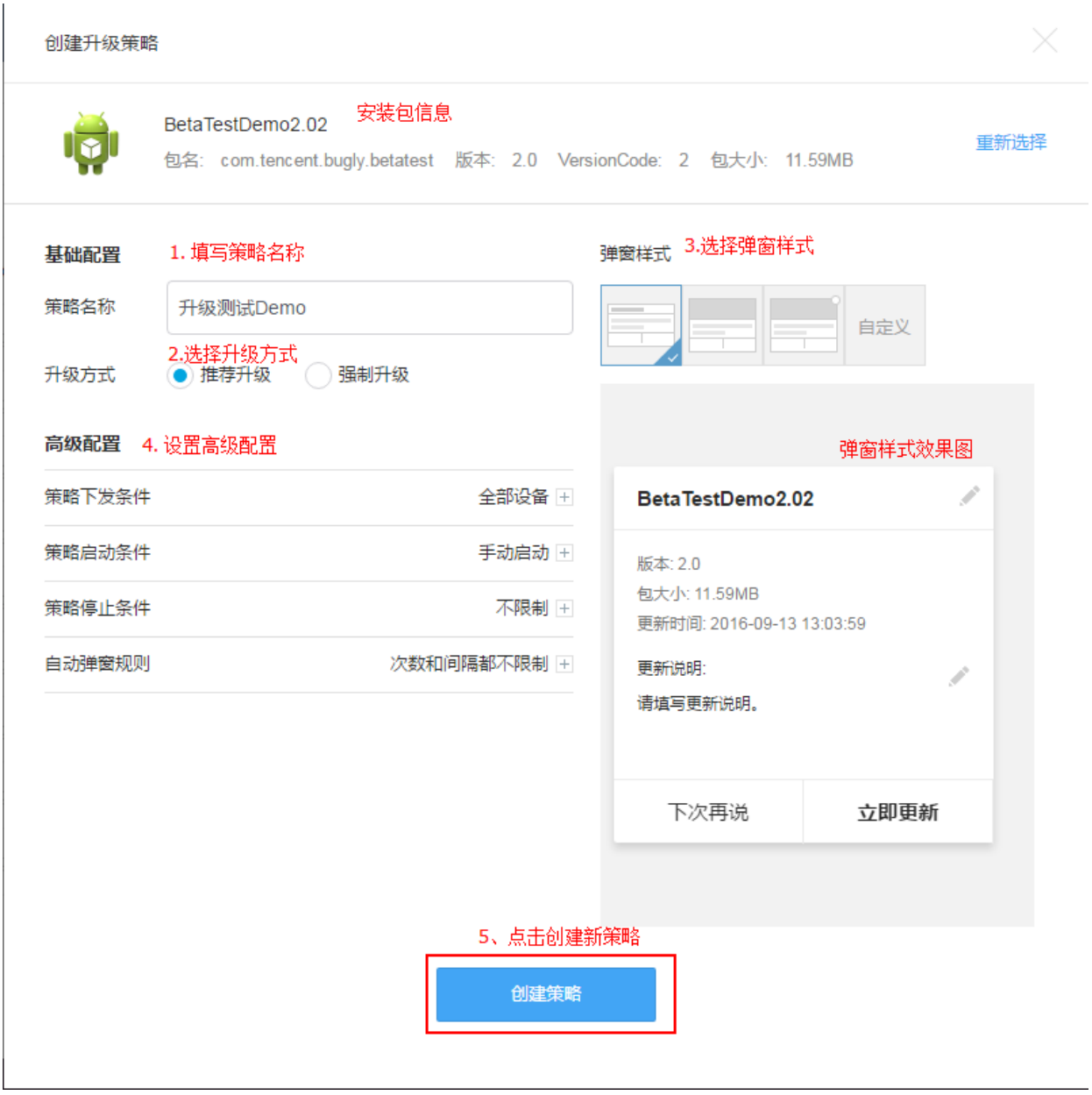
发布新版本

进入内测分发页面选择注册的APP，点击发布新版本，上传要升级的APP的版本（上传APP的versioncode必须不低于外发版本的versiocode，否则用户检测不到更新）

编辑版本信息

发布新升级

使用默认策略配置，点击创建策略



策略创建完成后会回到版本编辑页面，点击启动，使策略生效;

测试验证

完成步骤4.4中的策略配置，在本地安装配置过升级SDK的低版本APP，启动后（**请先杀掉进程**）等待一段时间(默认是3s)会弹出如下升级弹窗，表示SDK配置成功。

注： **如果你有以下需求，可以参考升级SDK高级配置**：

1. 设置自动初始化
2. 设置开关自动检查
3. 设置升级检查周期
4. 设置初始化延迟
5. 设置通知栏图标
6. 设置更新弹窗bannner图
7. 设置更新资源存储目录
8. 设置开启显示打断策略
9. 设置自定义UI
10. 设置升级对话框生命周期回调