北京基调网络股份有限公司

听云平台部署说明

听云 APP Android SDK Gradle 部署说明

Chengmy
2017/11/27

前言

产品版本

与本手册相对应的产品版本如下所示。

产品名称	产品版本	手册版本
Android 探针	2.4.4	V1.0
Android 探针	2.5.0	V2.0
Android 探针	2.5.5	V2.1
Android 探针	2.5.7	V2.2
Android 探针	2.5.9	V2.3
Android 探针	2.7.0	V2.4
Android 探针	2.7.1	V2.5

内容介绍

本手册主要介绍了 Android SDK 探针的 Gradle 部署方法及步骤。

读者对象

本手册适用于以下人员：

- Android 开发工程师
- Android 测试工程师
- 网络监控工程师
- 系统维护工程师

约定

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
🔔	以本标志开始的文本表示有潜在风险，如果忽视这些文本，可能导致设备损坏、数据丢失、设备性能降低或不可预知的结果。

符号	说明
🕮	以本标志开始的文本是正文的附加信息，是对正文的强调和补充。

修改记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有版本的更新内容。

版本 V2.5 （2017-11-27）

- 修改

位置	描述
使用 Gradle 方式构建	移除 2.1.5 在 MainActivity 中初始化 SDK
使用 Gradle 方式构建	新增 Cordova 及 React Native 工程相关依赖构建

版本 V2.4 （2017-09-25）

- 修改

位置	描述
配置本地文件	移除配置 OkHttp 开关相关内容，增加项目嵌码控制开关

版本 V2.3 （2017-07-27）

- 修改

位置	描述
配置 OKHttp 控制开关	OKHttp2.0 系列请使用 2.1.0 以上版本
配置 WebView 控制开关	通过调用接口嵌码 WebView

版本 V2.2 （2017-06-22）

- 新增

位置	描述
高级功能	控制台版本提示

版本 V2.1 （2017-05-25）

- 修改

位置	描述
安装探针	无需添加 SO 库即可收集 TCP 数据
高级功能	用户自定义 ID 修改接口限制

版本 V2.0 （2016-10-27）

- 新增

位置	描述
简介	简介
高级功能	高级功能

版本 V1.0 （2014-06-18）

手册初次发行

目录

1 简介

Android SDK 探针支持如下数据采集：

- Http/Https 协议数据收集
- 崩溃/ANR 异常数据收集
- 事件性能数据收集
- 视图性能数据收集
- 进程内存和 CPU 使用率

Android SDK 探针支持如下协议类库：

- HttpURLConnection
- Android HttpClient[1]*4.0*
- Apache HttpClient (> 4.0)
- Volley+OkHttpClient
- OkHttp
- Retrofit
- WebView（原生）

Android SDK 探针支持如下系统版本：

- Android 2.0 版本~Android7.0 版本

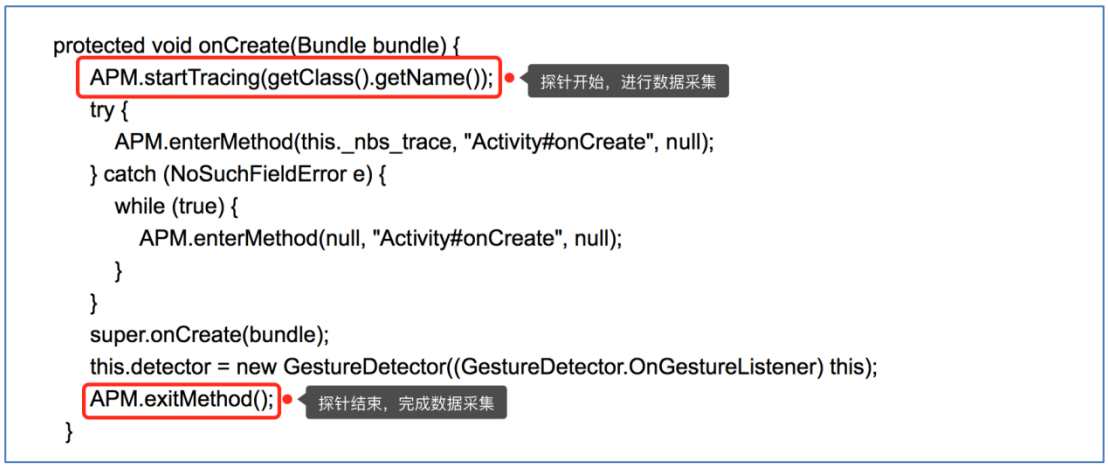
Android SDK 通过在指定的方法中嵌码去采集下列方法中的数据。可以通过报表服务器查询各种方法的数据和调用次数。

- 应用响应时间
- DNS 解析时间
- TCP 建连时间
- SSL 握手时间
- 首包时间

- 访问量

Android SDK 探针也可以采集线程堆栈，数据库、自定义参数和 HTTP 请求参数。

1.1 Android SDK Theory



听云 SDK 通过虚拟机技术在应用打包编译过程中对应用采样点（Http 标准协议和 Https 标准协议）进行嵌码操作，该操作会在协议类库方法前后部署听云 SDK 探针，该过程不会影响用户代码逻辑。

每当 App 启动时，听云 Agent 开始工作。应用有网络请求时，通过之前部署的听云 SDK 探针以一定的采集频率来采集数据，并对采集的数据进行汇总后，上传到服务器（报表展现）。

应用退出到后台或用户关闭 App 时，听云 Agent 停止工作，以便减少不必要的流量消耗。

1.2 Android SDK 体积增量

- 应用 App 嵌码后体积增量为 300KB 左右。

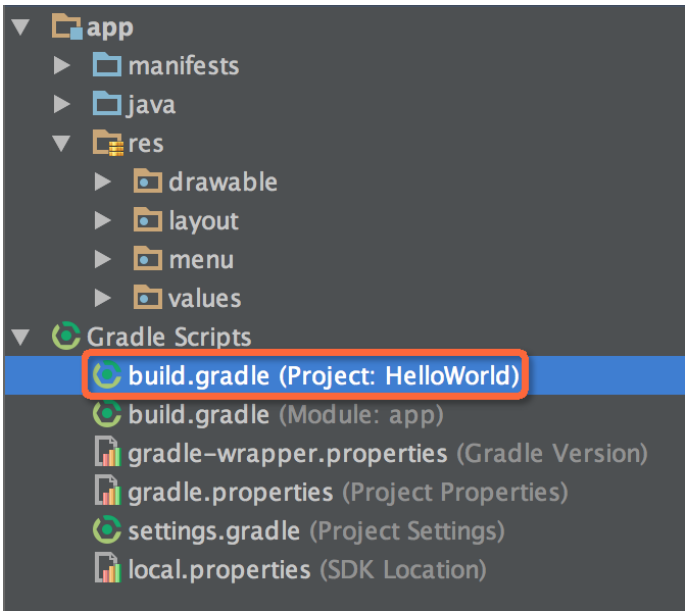
2 Gradle 方式部署

2.1 使用 Gradle 构建

2.1.1 Android Studio 工程相关依赖构建

操作员需要确保已经安装了 Gradle 构建环境和 AS 开发环境

1. 打开项目根目录下的 build.gradle（Project）文件

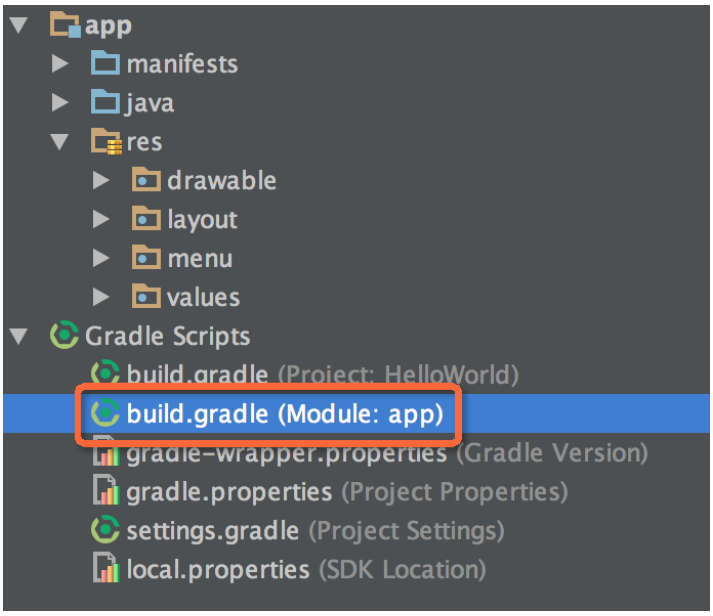


2. 在 buildscript 模块中加入代码

```
classpath 'com.networkbench.newlens.agent.android:agent-gradle-plugin:TingYun Version'//TingYun_Version 为当前听云 APP 版本号
```

```
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:1.0.0'
        classpath "com.networkbench.newlens.agent.android:agent-gradle-plugin:2.2.5"
    }
}

allprojects {
    repositories {
        jcenter()
    }
}
```

3. 打开项目工程主模块下的 build.gradle（Module）文件



4. 在文件中引入 mavenCentral()

```
repositories {
mavenCentral()
}
```

5. 在 dependencies 模块中添加代码

```
compile
"com.networkbench.newlens.agent.android:nbs.newlens.agent:TingYun_Version"
// TingYun_Version 为当前听云 APP 版本号
```

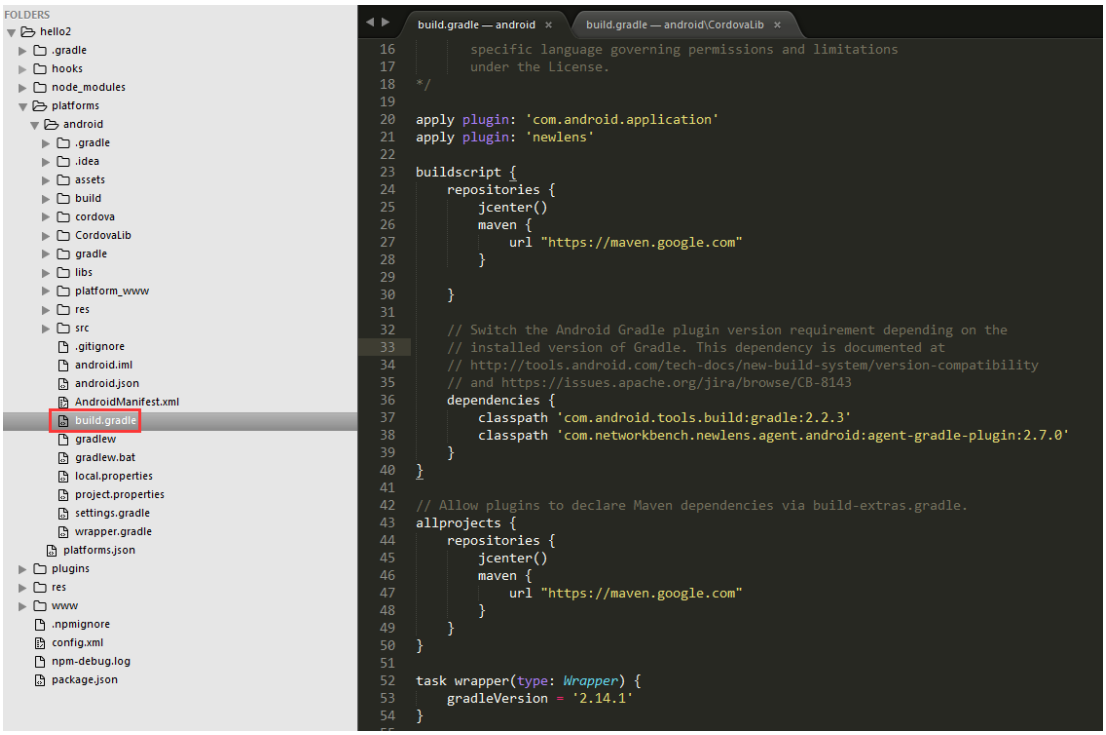
6. 添加听云符号表插件

```
apply plugin:'newlens'
```

```
apply plugin: 'android'
apply plugin: 'newlens'

android {
    compileSdkVersion 21
    buildToolsVersion "21.1.2"

    defaultConfig {
        applicationId "com.hello.nbs.helloworld"
        minSdkVersion 14
        targetSdkVersion 21
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }

}

repositories {
    mavenCentral()
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:21.0.3'
    compile "com.networkbench.newlens.agent.android:nbs.newlens.agent:2.2.5"
}
```

2.1.2 Cordova 工程相关依赖构建

1. 打开项目 android 目录下的 build.gradle 文件



2. 添加代码

```
apply plugin:'newlens'
```

```
apply plugin: 'com.android.application'
apply plugin: 'newlens'

buildscript {
    repositories {
        jcenter()
        maven {
            url "https://maven.google.com"
        }

    }
```

3. 在 buildscript 中添加以下代码

```
classpath
'com.networkbench.newlens.agent.android:agent-gradle-plugin:TingYun Version
'//TingYun_Version 为当前听云 APP 版本号
```

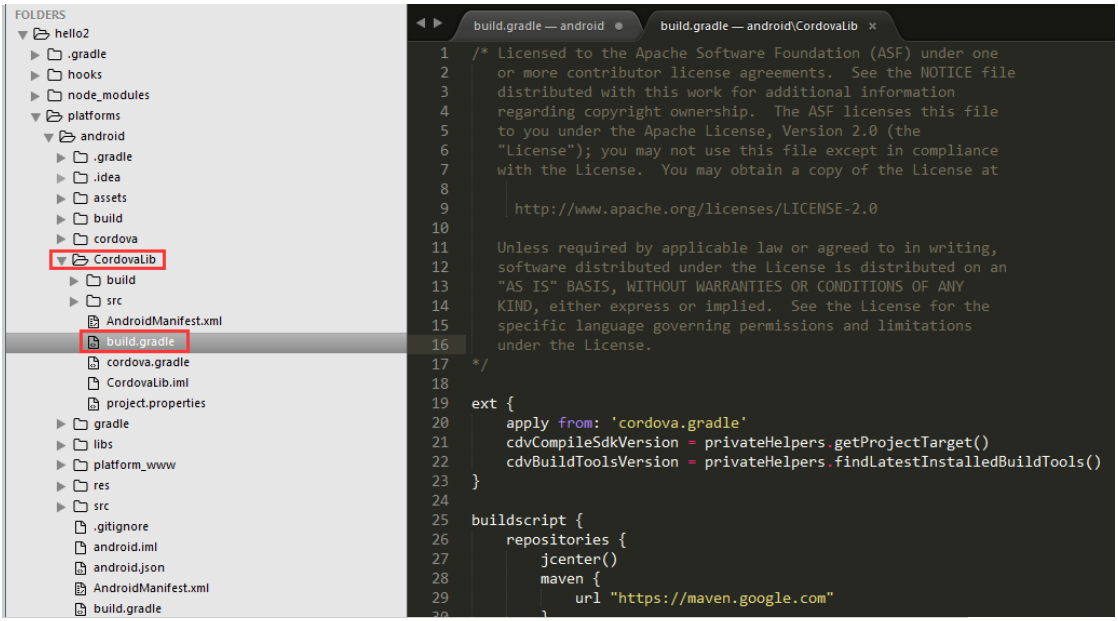
```
buildscript {
    repositories {
        jcenter()
        maven {
            url "https://maven.google.com"
        }

    }

    dependencies {
        classpath 'com.android.tools.build:gradle:2.2.3'
        classpath 'com.networkbench.newlens.agent.android:agent-gradle-plugin:2.7.0'
    }
}
```

4. 在 dependencies 中添加以下代码

```
compile
'com.networkbench.newlens.agent.android:nbs.newlens.agent:TingYun_Version'/
/TingYun_Version 为当前听云 APP 版本号
```

```
dependencies {
    compile 'com.networkbench.newlens.agent.android:nbs.newlens.agent:2.7.0'
    compile fileTree(dir: 'libs', include: '*.jar')
    // SUB-PROJECT DEPENDENCIES START
    debugCompile(project(path: "CordovaLib", configuration: "debug"))
    releaseCompile(project(path: "CordovaLib", configuration: "release"))
    // SUB-PROJECT DEPENDENCIES END
}
```

5. 打开 CordovaLib 的 build.gradle 文件



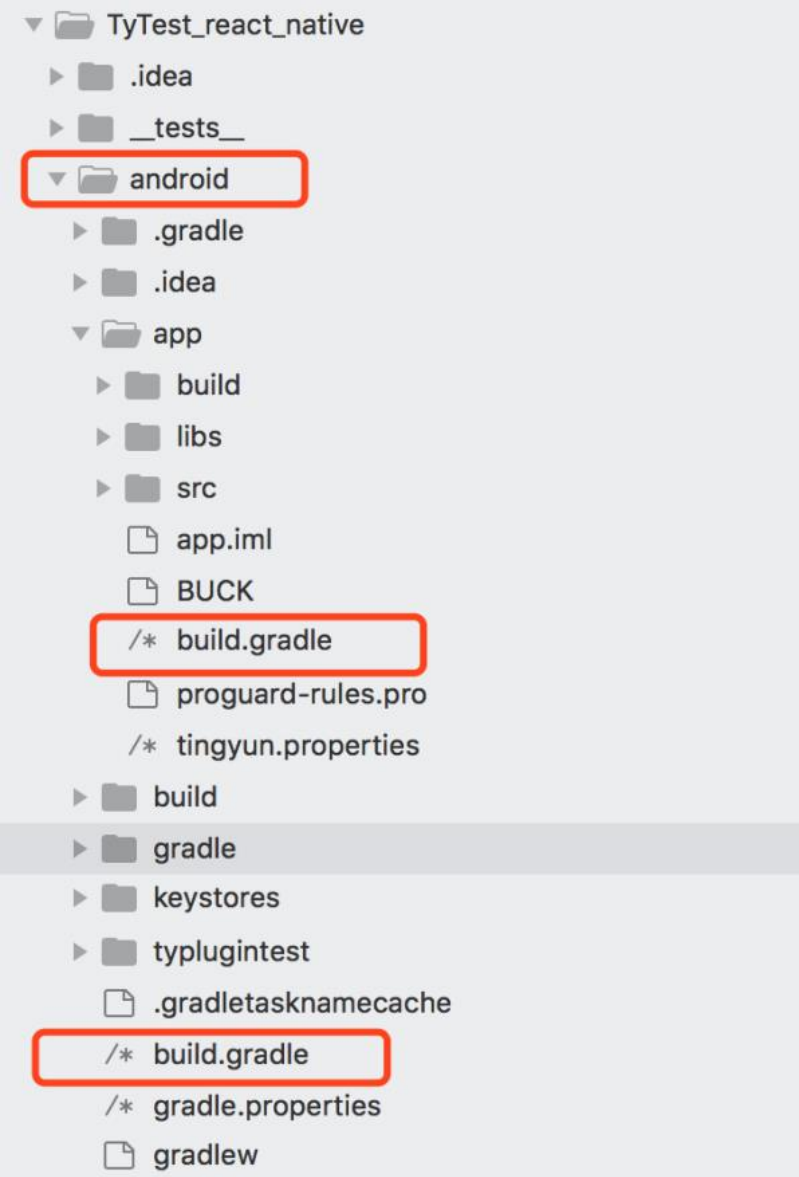
6. 添加以下代码

```
apply plugin: 'newlens'

allprojects {
    repositories {
        mavenCentral()
        jcenter()
//     mavenCentral()和 jcenter()配置任意一个均可
}

dependencies {
    provided
'com.networkbench.newlens.agent.android:nbs.newlens.agent:TingYun_Version'
//此处为 provided 依赖，TingYun_Version 为当前听云 APP 版本号
}
```

```
apply plugin: 'com.android.library'
apply plugin: 'com.github.dcendents.android-maven'
apply plugin: 'com.jfrog.bintray'
apply plugin: 'newlens'

allprojects {
    repositories {
        mavenCentral()
        jcenter()
    }
}

dependencies {
    provided 'com.networkbench.newlens.agent.android:nbs.newlens.agent:2.7.0'
}
```

2.1.3 React Native 工程相关依赖构建

React Native 工程嵌码和普通的 android 项目一样的，项目目录如下：



可以参照 2.1.1Android Studio 工程相关依赖构建进行配置。

2.1.4 配置应用权限

构建完成后，请在待监测的 App 工程的 AndroidMainfest.xml 文件中增加以下的权限

```
<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 PHONE STATE"/>
<uses-permission android:name="android.permission.READ LOGS" />
<uses-permission android:name="android.permission.INTERNET" />
```

2.1.5 插入初始化探针代码

1. 在嵌码项目工程的“Application”中 import NBSAppAgent 类

```
import com.networkbench.agent.impl.NBSAppAgent;
```

2. 在“Application”中的 onCreate()方法（如未找到该方法请新增 onCreate()）中初始化 Android SDK

```
NBSAppAgent.setLicenseKey("AppKey").withLocationServiceEnabled(true).startI
nApplication(this.getApplicationContext());//Appkey 请从官网获取
```

3. 若无需采集地理位置，请使用以下配置

```
NBSAppAgent.setLicenseKey("AppKey").start(this.getApplicationContext());
//Appkey 请从官网获取
```

2.1.6 使用 Gradle 命令打包编译

```
gradle clean build
```

2.1.7 配置混淆

1. 发布前请在 proguard 混淆配置文件中增加以下内容，以免 tingyunSDK 不可用

```
# ProGuard configurations for NetworkBench Lens
-keep class com.networkbench.** { *; }
-dontwarn com.networkbench.**
-keepattributes Exceptions, Signature, InnerClasses
# End NetworkBench Lens
```

2. 若需要保留行号信息，请在 proguard.cfg 中添加以下内容

```
-keepattributes SourceFile,LineNumberTable
```

2.2 嵌码完整性校验

1. 数据收集服务器校验
2. 嵌码完成后可通过“LogCat”查看听云 SDK 日志输出结果，用以进行数据收集服务器校验 TAG 为 NBSAgent，标准日志输出结果如下所示：

```
NBSAgent start
NBSAgent enabled
NBSAgent V “TingYun_Version” //TingYun_Version 为当前 SDK 的版本号
connect success
```

3. 数据功能完整性校验
   嵌码完成后可通过“LogCat”查看听云 SDK 日志输出结果，用以进行数据功能完整性校验 TAG 为 TingYun，标准日志输出结果如下所示：

```
D/TingYun: Crash switch is true
D/TingYun: webView switch is true
D/TingYun: ANR monitor switch is true
D/TingYun: UserAction Switch  is true
D/TingYun: cdnSwitch Switch  is true
```

3 高级功能

3.1 用户自定义 ID

1. 功能说明
   用户自定义 ID 为当前用户设置唯一标示码，在任意位置均可设置 UserID。

2. 相关接口

```
//UserID 最多包含 64 个字符，支持中文、英文、数字、下划线，但不能包含空格或其他的转义字符
NBSAppAgent.setUserIdentifier("userIdentifier");
```

3. 代码示例

```
public class MainActivity extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        String userIdentifier = getUserID();
        NBSAppAgent.setLicenseKey("AppKey").withLocationServiceEnabled(true)
.start(this.getApplicationContext());
         NBSAppAgent.setUserIdentifier(userIdentifier);
      }
 }
```

3.2 面包屑

1. 功能说明
   面包屑能够更好的协助用户调查崩溃发生的原因，可以知晓用户发生崩溃之前的代码逻辑与崩溃轨迹结合使用能够更好的复现用户崩溃场景。

2. 相关接口

```
//最多包含 100 个字符，支持中文、英文、数字、下划线
NBSAppAgent.leaveBreadcrumb("keyPressed”);
NBSAppAgent.leaveBreadcrumb("loginDone");
```

3. 代码示例

```
public MyActivity extends Activity {
         public void onCreate(Bundle savedInstanceState) {
             super.onCreate(savedInstanceState);
             NBSAppAgent.setLicenseKey("AppKey").withLocationServiceEnabled(
true).start(this.getApplicationContext());
             NBSAppAgent.leaveBreadcrumb("login MyActivity onCreate");
         }
         public void onResume() {
             super.onResume();
             NBSAppAgent.leaveBreadcrumb("login MyActivity onResume");
         }
         public void logginPressed(View view) {
             NBSAppAgent.leaveBreadcrumb("login MyActivity logginPressed");
             new LoginAsyncTask.execute();
         }
```

```
        public void onStop() {
            super.onStop();
            NBSAppAgent.leaveBreadcrumb("login MyActivity onStop");
        }
    }
```

3.3 自定义 Event

1. 功能说明

   自定义事件用于统计 App 中的任意事件，开发者可以在 SDK 初始化后的任意位置添加自定义事件，并设置对应上传参数。如：真实用户操作时候点击某个功能按钮或触发了某个功能事件等。

2. 相关接口

```
//EVENT_ID 最多包含 32 个字符，支持中文、英文、数字、下划线，但不能包含空格或其他的转义字符
NBSAppAgent.onEvent(String EVENT_ID);
```

3. 代码示例

```
@Override
public void onClick(View v) {
    ……
      NBSAppAgent.onEvent("添加购物车");
    ……
}
```

3.4 自定义 Trace

由于自定义 Trace 是成对出现的，请勿跨方法、跨进程以及在异步加载和递归调用中使用该接口。

1. 功能说明

   听云 SDK 默认采集系统类和方法的性能数据，无法采集开发者自定义类和方法的性能数据。使用“自定义 Trace”接口就可以帮助开发者时刻了解所写代码的健壮性及其性能数据。如：开发者想要了解某个自定义方法的初始化耗时及性能消耗情况，就可以在该自定义方法前后添加“自定义 Trace”接口即可。

2. 相关接口

```
//Name 为当前方法所在方法名或自定义名称，支持中文、英文、数字、下划线，但不能包含空格或其他的转义字符
NBSAppAgent.beginTracer("String Name");
NBSAppAgent.endTracer("String Name");
```

3. 代码示例

```
//用户可以在 SDK 初始化后的任意方法前后添加自定义 Trace
public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      init();
    }
```

```
private void init() {
     //方法开始前添加 beginTracer
     NBSAppAgent.beginTracer("这是 Init 方法");
       try {
           ……
       } catch (NameNotFoundException e) {
           e.printStackTrace();
       }
     //方法结束后添加 endTracer
     NBSAppAgent.endTracer("这是 Init 方法");
    }
```

3.5 自定义 Log

收集 LogCat 信息,需要使用 READ_LOGS 权限,请将下列代码添加到应用程序的 AndroidManifest.xml 文件中，默认收集 50 行，最多收集 100 行日志。

< uses-permission android:name="android.permission.READ_LOGS" >

1. 功能说明
   开发者可通过 Android 日志系统的 LogCat，来收集和查看系统调试输出的信息，通过打印输出的 Log 信息来调查 Bug 发生时的应用程序信息，并通过听云 SDK 上传自定义 Log 日志。
2. 控制开关

```
NBSAppAgent.setLicenseKey("AppKey").withLocationServiceEnabled(true).enable
Logging(true).start(this.getApplicationContext());
```

3. 相关接口

```
NBSAppAgent.setLogging(int lineNumber);
NBSAppAgent.setLogging(String filter);
NBSAppAgent.setLogging(int lineNumber,String filter);
```

4. 代码示例

```
public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      // Enable logging
      NBSAppAgent.setLicenseKey("AppKey").withLocationServiceEnabled(true).
enableLogging(true).start(this.getApplicationContext());
         // Log last 100 messages
         NBSAppAgent.setLogging(100);
    }
```

3.6 自定义附加信息

1. 功能说明
   用户可以在初始化之后任意位置配置该接口，最多可添加 10 条附加信息，每条附加信息最大支持 100 个字节随崩溃上传。

2. 相关接口

```
NBSAppAgent.setUserCrashMessage(String key,String value);
```

3. 代码示例

```
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    NBSAppAgent.setLicenseKey("AppKey").withLocationServiceEnabled(true).s
tart(this.getApplicationContext());
     //初始化后的任意位置插入自定义附加信息
     NBSAppAgent.setUserCrashMessage("张三","13700001234");}
```

3.7 控制台版本提示

1. 功能说明

用户调试时会在控制台输出版本更新提示，当开启调试模式且当前使用 SDK 版本低于线上最新版本时，App 运行后打印输出。

注：项目工程只有 debug 模式才会开启此功能。

可通过在 AndroidMainfest.xml 文件中的 application 标签添加 debuggable 属性控制是否开启调试模式。

<application android:debuggable="true">

2. 相关接口

默认开启，若无需提示，可关闭控制台提示功能。

```
NBSAppAgent.closeLogForUpdateHint() ;
```

3. 代码示例

```
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    NBSAppAgent.setLicenseKey("AppKey").withLocationServiceEnabled(true).
closeLogForUpdateHint().start(this.getApplicationContext());
    //初始化时调用接口关闭版本提示功能
}
```

4 设置本地配置文件

4.1 新建 tingyun.properties 配置文件

在项目工程根目录及 Module 目录下，新建 tingyun.properties 配置文件即可

4.2 配置 Crash 反混淆（mapping）

听云 SDK 提供了两种方式进行反混淆设置

4.2.1 通过报表上传 mapping 文件

1. 报表中选择“崩溃”模块，在“崩溃历史记录列表”打开“dSYM/Mapping 文件管理”

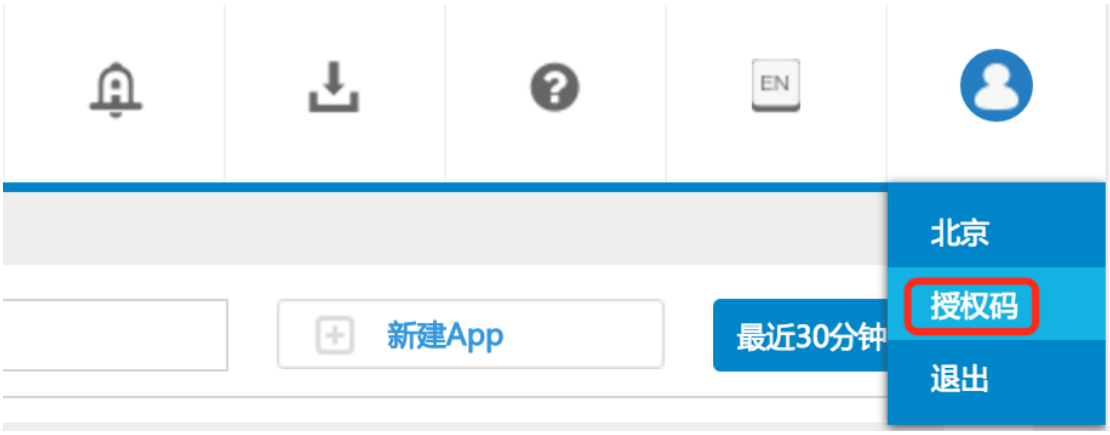
2. 找到所需版本，并上传本地 mapping 文件即可

4.2.2 通过 tingyun.properties 文件自动上传 mapping 文件

1. 添加如下配置到 tingyun.properties 文件

```
authKey=*听云 API 账号授权 Key，由报表系统生成*
appKey=*听云 AppKey*
mapping_file_auto_upload=true
```

2. 配置 Authkey 授权码



a. 登陆报表后选择“授权码”

b. 复制当前授权码并添加到 tingyun.properties 即可

3. 配置 AppKey

tingyun.properties 配置文件中的 AppKey 相同，在“修改设置”中获取即可

4. 配置 mapping_file_auto_upload 控制开关

```
mapping_file_auto_upload=true
```

5. 开启混淆器
   mapping_file_auto_upload 控制开关只有在启用混淆器的时候才会生效，开启控制开关后，听云 SDK 会将本地目录下的 mapping 文件自动上传到听云服务器。若未开启混淆器则该配置项不生效。

4.3 配置 WebView 控制开关

1. 请将如下配置添加到 tingyun.properties 配置文件中

```
webview=true
```

2. 采集 WebView 数据需调用 setWebViewClient 方法，如嵌码 App 中未调用该方法，请添加如下内容

```
webview.setWebViewClient(new WebViewClient(){});
```

3. 采集 WebView 数据需在 WebChromeClient 的 onProgressChanged 函数中调用接口：NBSWebChromeClient.initJSMonitor(view, newProgress);例子如下：

```
webview.setWebChromeClient(new WebChromeClient(){
 @Override
      public void onProgressChanged(WebView view, int newProgress) {
          NBSWebChromeClient.initJSMonitor(view, newProgress);
          super.onProgressChanged(view, newProgress);
      }
});
```

4. 采集 Cordova WebView 数据需在 CordovaLib 中的 SystemWebChromeClient 中添加 onProgressChanged 函数并调用接口：NBSWebChromeClient.initJSMonitor(view, newProgress);例子如下：

```
webview.setWebChromeClient(new WebChromeClient(){
 @Override
      public void onProgressChanged(WebView view, int newProgress) {
          NBSWebChromeClient.initJSMonitor(view, newProgress);
          super.onProgressChanged(view, newProgress);
      }
});
```

4.4 配置项目嵌码控制开关

1. 如果不需要在项目中嵌码，请在 tingyun.properties 配置文件中添加以下内容

> 注：此项配置适用于多个项目并行打包，未使用听云的项目可能会嵌入 SDK 代码。若出现此情况，请在未使用听云的项目根目录下放置配置文件。若无此情况，无需配置此项。

```
instrument=false
```