重要信息!

从2020年3月1日开始，AppsFlyer已弃用旧版SDK版本。要了解哪些版本已弃用、旧版的影响以及如何更新到最新的SDK版本，请点击此处。

SDK 版本: 5.2.0 (发布说明)

1. 概述

作为AppsFlyer完整归因解决方案的一部分，AppsFlyer Android SDK提供了应用程序安装和事件记录功能。该SDK经过市场验证，安全，轻便且易于嵌入。

应用程序嵌入SDK后，您可以记录安装，会话和其他应用内事件（例如，应用内购买，游戏级别等）来评估用户参与度，甚至投资回报率！

1.1 SDK集成-您需要做什么

选项卡	目的	完成后
SDK集成 （ **强制** ）	添加并配置SDK	■ 应用面板记录到一个新的自然安装 ■ 应用面板记录到一个新的非自然安装
核心API (**推荐**)	测量应用内事件、收入、执行深度链接、统计转化数据	■ 使应用内事件和收入显示在控制面板上 ■ 实现深度链接
其他API （ **推荐** ）	可选API的实现和使用 我们建议检查该列表，因为一些可选API可能对您的商务计划至关重要，例如统计卸载或推荐归因。	■ 可以统计卸载、推荐安装、用户与推送通知的互动度等

选项卡	目的	完成后
API参考	开发人员SDK API的快速参考	

1.2 SDK与Android平台的兼容性

- Android OS 4.0及更高版本
- 非移动Android平台，例如智能电视，包括亚马逊的Fire TV
- Android应用的第三方应用商店，例如亚马逊，百度

此标签为应用程序开发人员参考，对如何实现和初始化AppsFlyer SDK进行说明描述，完成此标签后，您将在应用的AppsFlyer后台中看到两次安装，一次自然安装和一次非自然安装。

2.将SDK添加到您的应用中

2.1 将 SDK 添加到项目中

有两种方法可以将SDK添加到您的应用中：

- 使用Gradle（推荐）
- 手动添加SDK

使用Gradle

1

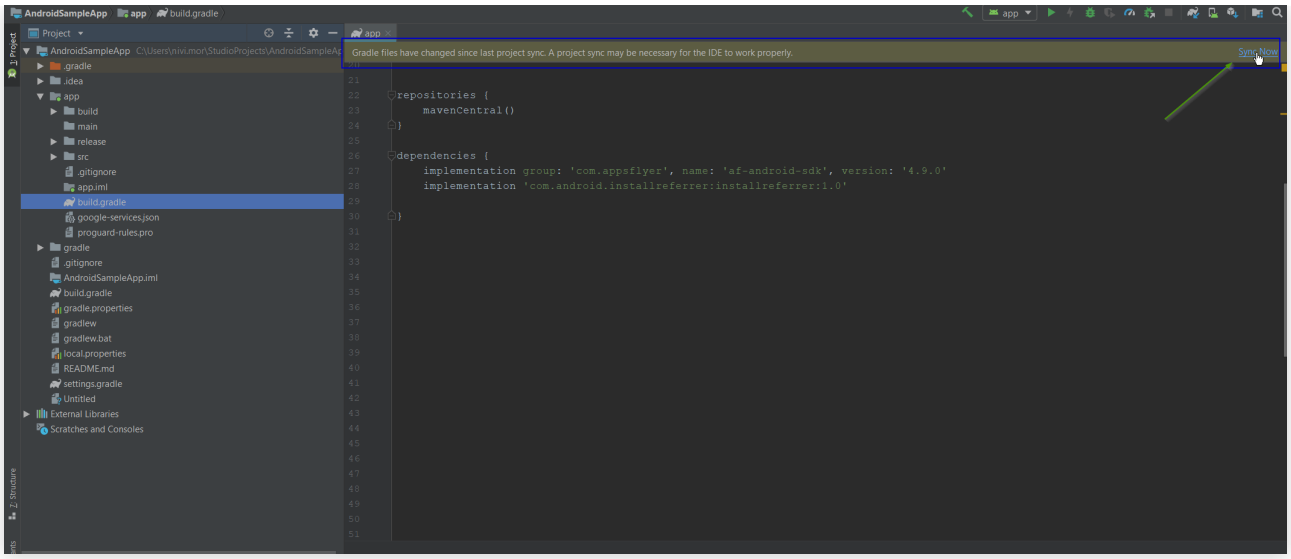
```
repositories {
  mavenCentral()
}
```

将以下代码添加到 `dependencies` 之前的 **/app/build.gradle** 模块：

2 添加最新版本的AppsFlyer SDK 为dependency。你可以在 此处找到最新版本.

```
dependencies {
//make sure to use the latest SDK version: https://mvnrepository.com/artifact/com.appsflyer/af-
    implementation 'com.appsflyer:af-android-sdk:5.0.0'
}
```

3 同步项目以检索依赖项-请参见以下截图：



手动添加SDK

1. 下载AF-Android-SDK.jar
2. 将其添加到项目

2.2 添加Android Install Referrer 到您的App中

Android Install Referrer可提高归因准确性，防止安装欺诈等。从AppsFlyer Android SDK版本4.8.6开始受支持。

> **注意**
>
> Google 在2020年3月份弃用了 BroadcastReceiver 为了不影响归因准确性，请您参考文档2.2部分，实施接入installreferrer。您可能仍需要使用BroadcastReceiver进行Google市场外商店的归因。请您与上架应用程序的Google市场外商店进行核对。

以下有两种方式可以将Install Referrer 添加到您的App中

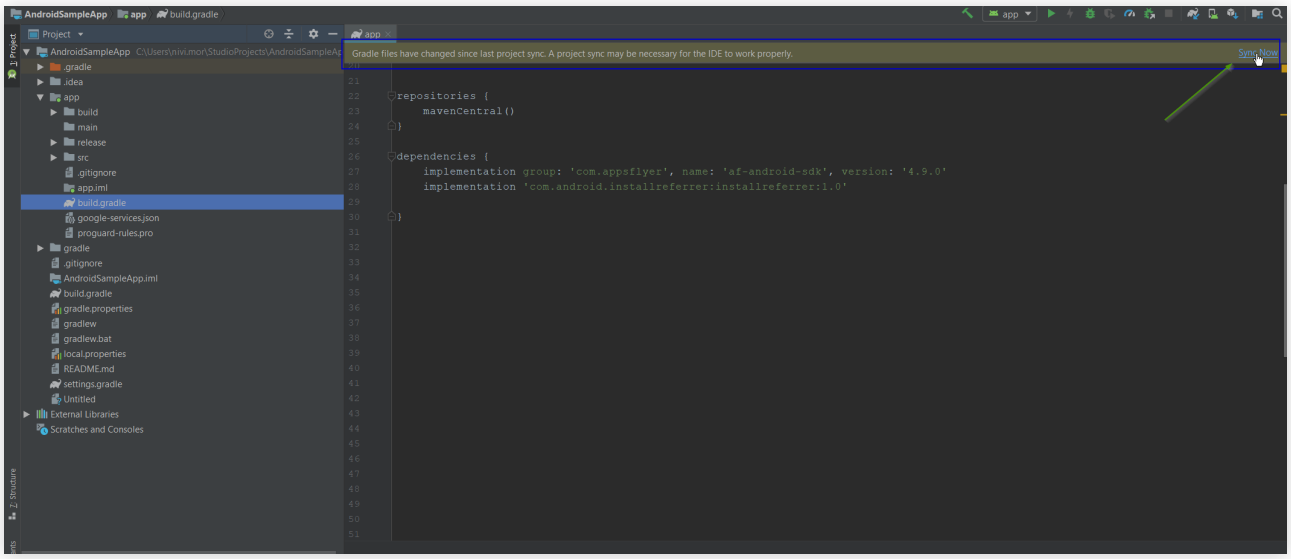
- 使用Gradle（推荐）
- 手动添加 Install Referrer

通过Gradle来添加Install Referrer

把安卓安装Referrer添加为dependency.您可以在 此处找到最新版本.

```
dependencies {
//make sure to use the latest SDK version: https://mvnrepository.com/artifact/com.appsflyer/af-
    implementation 'com.appsflyer:af-android-sdk:5.+'
    implementation 'com.android.installreferrer:installreferrer:1.0'
}
```

2 同步项目以检索依赖项-请参见以下截图：



如果您使用的是ProGuard，并且想使用Google的新referrer API，请设置以下ProGuard规则： -

```
dontwarn com.android.installreferrer
```

手动添加 Install Referrer

1 下载安装referrer aar
2 将其添加到项目
3 将以下权限添加到manifest中：

```
com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE
```

2.3设置所需权限

添加permissions有助于提高指纹归因的比率。它还允许SDK发送更多数据，例如Wifi和运营商网络数据。您可以在原始数据报告找到此数据，并将其用于分析和优化活动。

添加所需权限

1 将以下权限添加到AndroidManifest.xml：

```
<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" />
<!-- Optional : -->
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

2.4设置BroadcastReceiver以从Google Play获取数据

注意

Google将于2020年3月弃用 BroadcastReceiver 。为了不影响归因准确性，请您参考文档2.2部分，实施接入installreferrer。您可能仍需要使用BroadcastReceiver进行Google市场外商店的归因。请您与上架应用程序的Google市场外商店进行核对。

BroadcastReceiver从Google Play获取 AppsFlyer用于归因 。使用BroadcastReceiver可以提高归因率。

实施安装引用广播接收器，有如下两种方案：

使用单个 Broadcast Receiver

如果您没有在AndroidManifest.xml的 `INSTALL_REFERRER` 上设置receiver监听，则在application标签内添加以下接收器：

```
<application>

    <receiver android:name="com.appsflyer.SingleInstallBroadcastReceiver" android:exported="true">
        <intent-filter>
            <action android:name="com.android.vending.INSTALL_REFERRER" />
        </intent-filter>
    </receiver>

</application>
```

使用多个 Broadcast Receiver

如果您已经有一个receiver在INSTALL_REFERRER上监听，则AppsFlyer有一个可以自动向其他所有receiver广播INSTALL_REFERRER的解决方案。在AndroidManifest.xml中，将以下receiver添加为INSTALL_REFERRER的第一接收器，并确保receiver标签在application标签内：

```
<application>

    <receiver android:name="com.appsflyer.MultipleInstallBroadcastReceiver" android:exported="true">
        <intent-filter>
            <action android:name="com.android.vending.INSTALL_REFERRER" />
        </intent-filter>
    </receiver>

</application>
```

建议

如果将接收器添加到 AndroidManifest.xml 后出现错误 "Unresolved class SingleInstallBroadcastReceiver", 请确保首先生成应用程序。

3.SDK初始化

本节介绍如何初始化AppsFlyer Android SDK。

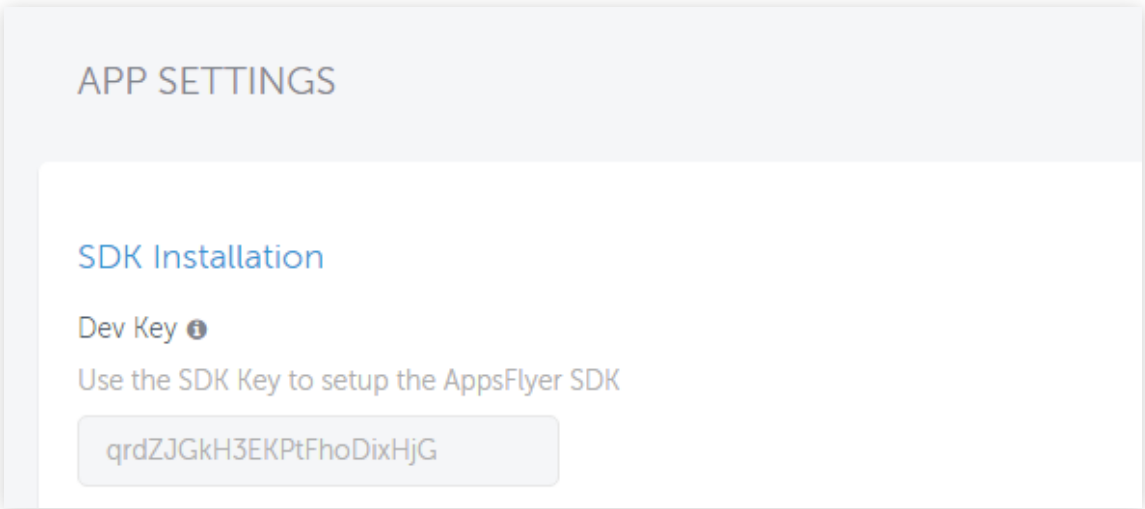
3.1检索dev key （开发者密钥）

AppsFlyer使用唯一的dev key来标识您的帐户。dev key是强制性的，因为它允许SDK安全地发送和检索属于您AppsFlyer帐户的数据。

重要提示：初始化SDK时，使用正确的dev key至关重要。使用错误的dev key会影响从SDK发送的所有流量，并导致归因和报告问题。

获取您的dev key：

1. 转到应用控制面板。
2. 在控制面板的 ***配置*** 单击 ***应用设置*** 。
3. 复制您的dev key。



3.2初始化SDK

我们建议在应用程序的全局应用程序类（global application state）中初始化SDK。这使SDK可以在所有方案中进行初始化，包括深度链接。

下述步骤发生在应用的全球应用类（global application class）中。

1 在应用的global class中，导入以下库

```
import android.app.Application;
import android.util.Log;
import com.appsflyer.AppsFlyerLib;
import com.appsflyer.AppsFlyerConversionListener;
import java.util.Map;
```

2 在全局类内，设一个变量记录您的dev key，推荐命名为AF_DEV_KEY。

重要：初始化SDK时，使用正确的dev key至关重要。使用错误的dev key会影响从SDK发送的所有流量，并导致归因和报告问题。

Java

```
public class AFApplication extends Application {
    private static final String AF_DEV_KEY = "qrdZGj123456789";

    //...

}
```

Kotlin

```
class AFApplication : Application() {
    private val devKey = "qrdZGj123456789";


    //...

}
```

3 在global class内部，在调用 `super.onCreate()` 后，实施`AppsFlyerConversionListener`.请参阅下面步骤4中的代码。

4 在全局类中，在 `ConversionListener` ，初始化SDK并调用 `startTracking` 方法

Java

```
public class AFApplication extends Application {
    private static final String AF_DEV_KEY = "qrdZGj123456789";

    @Override
    public void onCreate() {
        super.onCreate();
        AppsFlyerConversionListener conversionListener = new AppsFlyerConversionListener() {


            @Override
            public void onConversionDataSuccess(Map<String, Object> conversionData) {

                    for (String attrName : conversionData.keySet()) {
                    Log.d("LOG_TAG", "attribute: " + attrName + " = " + conversionData.get(attr
                }
            }

            @Override
            public void onConversionDataFail(String errorMessage) {
           Log.d("LOG_TAG", "error getting conversion data: " + errorMessage);
            }

            @Override
            public void onAppOpenAttribution(Map<String, String> conversionData) {

                for (String attrName : conversionData.keySet()) {
                    Log.d("LOG_TAG", "attribute: " + attrName + " = " + conversionData.get(attr
                }

            }

            @Override
            public void onAttributionFailure(String errorMessage) {
                Log.d("LOG_TAG", "error onAttributionFailure : " + errorMessage);
            }
        };

        AppsFlyerLib.getInstance().init(AF_DEV_KEY, conversionListener, getApplicationContext()
        AppsFlyerLib.getInstance().startTracking(this);

    }
}
```

Kotlin

```
class AFApplication : Application() {
    private val devKey = "qrdZGj123456789";

    override fun onCreate() {
        super.onCreate()
        val conversionDataListener  = object : AppsFlyerConversionListener{
            override fun onConversionDataSuccess(data: MutableMap<String, Any>?) {
                data?.let { cvData ->
                    cvData.map {
                        Log.i(LOG_TAG, "conversion_attribute:  ${it.key} = ${it.value}")
                    }
                }
            }

            override fun onConversionDataFail(error: String?) {
                Log.e(LOG_TAG, "error onAttributionFailure :  $error")
            }

            override fun onAppOpenAttribution(data: MutableMap<String, String>?) {
                data?.map {
                    Log.d(LOG_TAG, "onAppOpen_attribute: ${it.key} = ${it.value}")
                }
            }

            override fun onAttributionFailure(error: String?) {
                Log.e(LOG_TAG, "error onAttributionFailure :  $error")
            }
        }

        AppsFlyerLib.getInstance().init(devKey, conversionDataListener, applicationContext)
        AppsFlyerLib.getInstance().startTracking(this)
    }
}
```

3.3注册该global application class

在AndroidManifest.xml文件的 `application` 标签内，添加以下行：

```
android:name="APP.PACKAGE.NAME.AFApplication"
```

- APP.PACAKGE.NAME-用应用包名替换。
- AFApplication-用应用global class中设置的名称替换它。

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.appsflyer.sample">


    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <!-- Optional : -->
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />

    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.INTERNET" />

    <uses-permission android:name="android.permission.WAKE_LOCK" />

    <application
        android:name="com.appsflyer.sample.AFApplication"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name="com.appsflyer.sample.MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>

        </activity>

        <receiver
            android:name="com.appsflyer.SingleInstallBroadcastReceiver"
            android:exported="true">
            <intent-filter>
                <action android:name="com.android.vending.INSTALL_REFERRER" />
            </intent-filter>
        </receiver>

    </application>

</manifest>
```

manifest.xml中的这一行告诉应用程序什么是全局应用程序。如上所述，这样做使AppsFlyer SDK可以在整个应用程序中全局访问。

4.测试安装

现在，您可以通过模拟自然安装和非自然安装来测试SDK集成。

4.1将测试设备列入白名单

在开始测试安装之前， 把测试设备加入白名单。

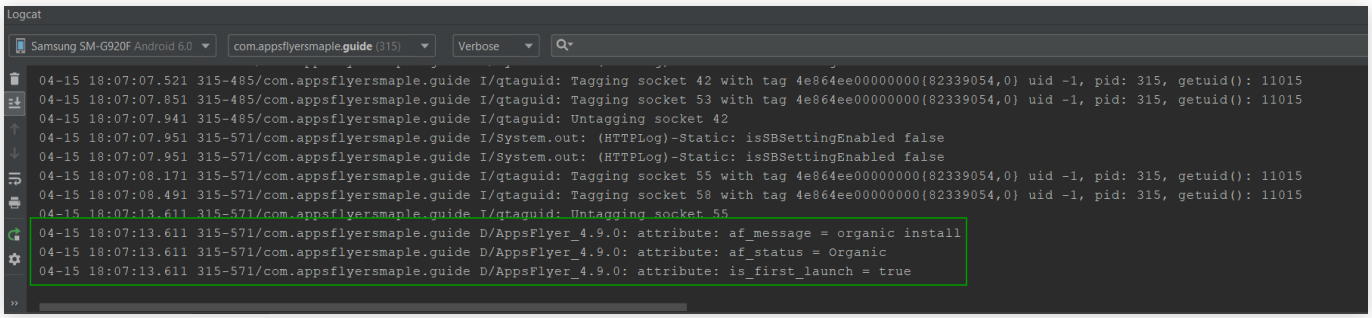
4.2模拟自然安装

自然安装是未被归因的安装，通常是直接前往应用商店安装的用户。

要模拟自然安装：

1. 确保已将移动设备连接到计算机。
2. 在Android Studio中，打开Logcat。
3. 在Android Studio中，将应用安装到设备或模拟器上。
4. 等待应用启动。
5. 在Logcat中，查找该应用的包名。

您应该看到以下内容：



屏幕截图中突出显示的部分表示SDK报告了自然安装。此数据来自AFApplication类中的 `onConversionDataSuccess` 方法。获取转换数据在 本指南后面将进行讨论 。

注意： 从SDK Version5开始， `onConversionDataSuccess` 是获取转换数据的方法的名称。如果使用的SDK版本低于5.0.0，则该方法的名称为 `onInstallConversionDataLoaded` 。我们建议您升级到SDK 5.0.0。要了解更多信息，请点击这里 。

此时该应用的***数据总览***面板上会出现一个自然安装。

如果您未在应用程序的仪表板上看到安装，请参阅我们的 SDK故障排除指南 。

4.3模拟非自然安装

非自然安装通常是在广告互动之后的可归因安装。您可以使用归因链接模拟非自然安装。

模拟非自然安装：

1. 在manifest中，找出您的应用包名，例如**com.company.app**。
2. 在下面的URL中，将<PACKAGE_NAME>替换为您的应用包名：

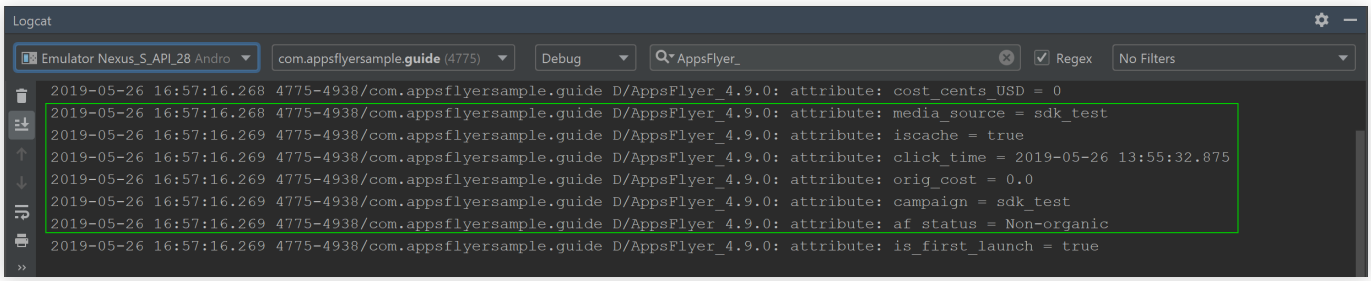
   ```
   https://app.appsflyer.com/<PACKAGE_NAME>?pid=sdk_test&c=sdk_test
   ```

   `pid` 参数代表媒体渠道名称。参数 `c` 表示广告系列名称。
3. 将此URL发送到设备。可以通过电子邮件或WhatsApp发送
4. 在设备上，单击URL。

- 如果该应用已在应用商店上架，那您将跳转至应用商店。**不要**从应用商店下载并安装该应用。继续执行步骤5。
- 如果该应用未在应用商店上架并且仍在开发中，则屏幕会显示'在应用商店中找不到这个应用'。继续执行步骤5。

5 在Android Studio中，打开Logcat。
6 使用USB数据线将设备连接到计算机。
7 通过Android Studio在设备上安装该应用。
8 在Logcat中，查找该应用的包名。

您应该看到以下内容：



现在在应用的***数据总览***面板里可以看到一个非自然安装。

集成SDK的已知问题

请参阅以下内容，以了解有关集成SDK时可能出现的问题以及如何解决这些问题的更多信息。

ProGuard警告

如果您使用的是ProGuard，并且遇到有关我们的 `AFKeystoreWrapper` 类的警告，则将以下代码添加到ProGuard规则文件中：

```
-keep class com.appsflyer.** { *; }
```

备份规则

如果您在AndroidManifest.xml的<application>标签内添加 `android:fullBackupContent="true"` ，您可能会收到错误提示：

```
Manifest merger failed : Attribute application@fullBackupContent value=(true)
```

要解决此错误，请在AndroidManifest.xml文件的<application>标签中添加 `tools:replace="android:fullBackupContent` 。

如果您指定了自己的备份规则，请通过添加以下规则将它们与AppsFlyer规则手动合并：

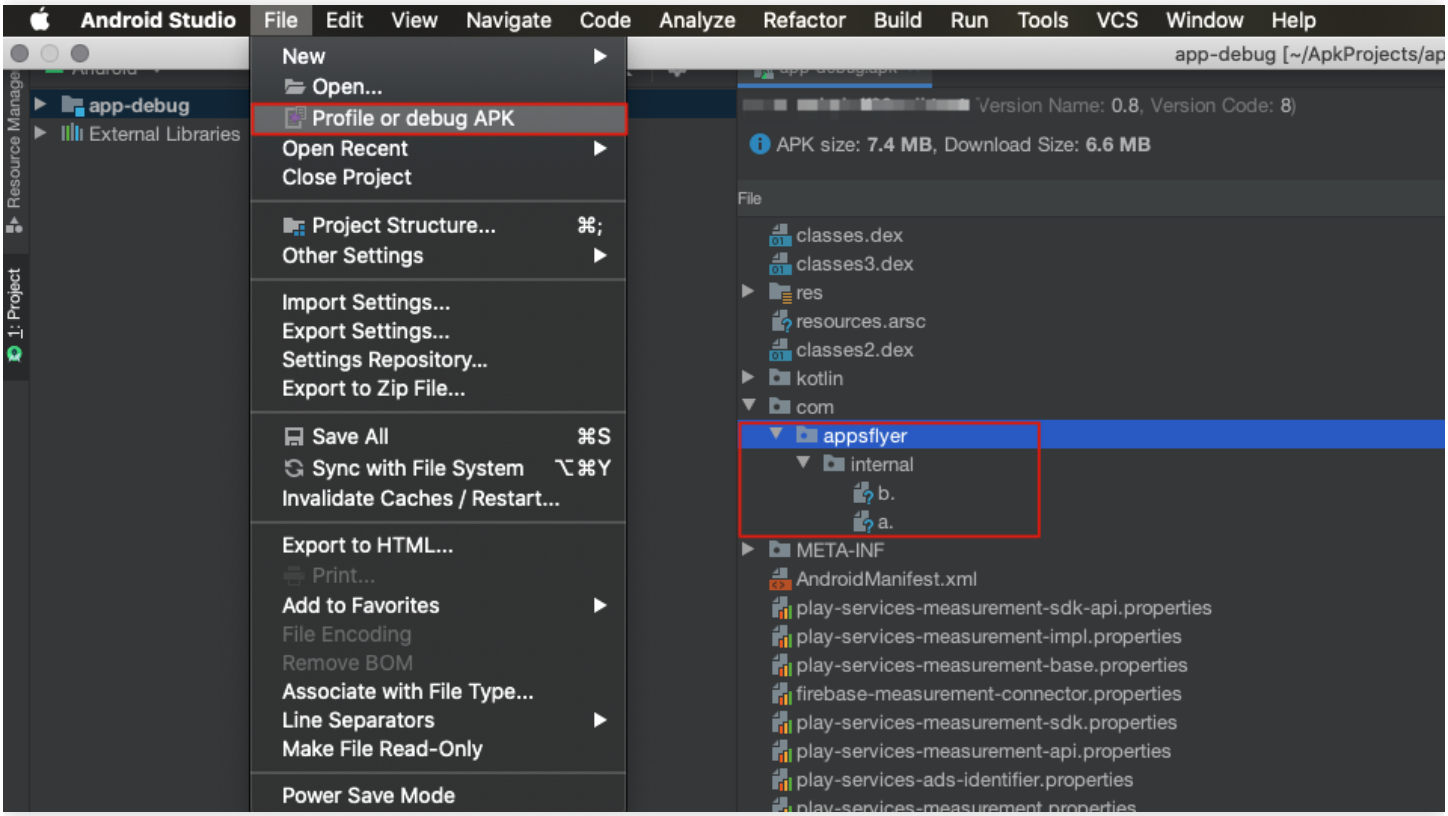
```
<full-backup-content>
    ...//your custom rules
    <exclude domain="sharedpref" path="appsflyer-data"/>
</full-backup-content>
```

缺少资源文件

如果您使用的Android SDK 为5.0.0或更高版本，请确保在APK中，除了 ***classes.dex*** 和 ***resources*** 文件外，您的文件夹 ***com > appsflyer > internal*** 中同样包含文件 ***a.*** 和 ***b.*** 。

如果这些文件丢失，则SDK无法向我们的服务器发出网络请求。

在Android Studio中打开您的APK来检查您是否有必须的文件。请参阅以下屏幕截图以供参考。



此标签说明了如何记录应用内事件和收入、以及如何设置深度链接。

记录应用内事件和收入可让您衡量用户质量。深度链接可以为用户提供更好的用户体验。

该选项卡是给开发人员的说明，但是市场营销人员的参与也必不可少，原因如下：

- 应当由市场营销人员决定需要记录的应用内事件以衡量用户质量。
- 市场营销人员必须要有访问AppsFlyer控制面板的权限才可以设置OneLink进行深度链接。

5.记录应用内事件

应用内事件可助您深入了解应用里正在发生的事。我们建议您花些时间定义要记录的事件。记录应用内事件有助于您衡量KPI，例如ROI（投资回报率）和LTV（生命周期价值）。

有几种方法可以记录应用内事件。最常见的方法是通过我们在本文中讨论的通过SDK发送事件。要了解其他记录应用内事件的方法，请参见我们的应用内事件概述指南 。

如果您的应用属于某个行业，例如旅行、游戏、电子商务等，您可以参考每个行业的推荐应用内事件列表。

5.1应用内事件名称和参数

SDK具有两个与应用程序内事件相关的接口：

- **AFInAppEventType** - 应用内**事件名称**常量。
- **AFInAppEventParameterName** -应用内**事件参数名称**常量。

出于以下原因，我们强烈建议使用这两个接口：

- 标准命名使得AppsFlyer可以将事件自动映射到SRN，例如Facebook，Google，Twitter和Snapchat。
- 向后兼容-如果AppsFlyer决定更改任何事件或事件参数的名称，则您的实现是向后兼容的。

要使用这两个接口，请导入它们：

```
import com.appsflyer.AFInAppEventParameterName;
import com.appsflyer.AFInAppEventType;
```

这里是推荐事件名称和结构列表 。

5.2记录收入

您可以通过任何应用内事件发送收入。使用 `af_revenue` （ `AFInAppEventParameterName.REVENUE` ）事件参数将收入包含在应用内事件中。您可以使用任何数值（正数或负数）填充它。

`af_revenue` 是AppsFlyer原始数据和控制面板上唯一统计为收入的事件参数。欲了解更多详情请点击这里 。

发送收入事件时，请记住以下几点：

- 如果设置货币代码（请参见下面的示例），则该代码应为 3个字符的ISO 4217代码 。(默认值为美元)。
- 您可以通过调用以下方法为所有事件设置货币代码： `AppsFlyer.setCurrencyCode ( " zzz " )` 要了解有关货币设置、显示和货币换算的信息，请参见我们的收入货币指南。
- 收入值不应包含逗号、货币符号或文本。例如，收入事件应类似于 1234.56。

示例： 有收入的应用内购买事件

Java

```
Map<String, Object> eventValue = new HashMap<String, Object>();
eventValue.put(AFInAppEventParameterName.REVENUE,1234.56);
eventValue.put(AFInAppEventParameterName.CONTENT_TYPE,"Shirt");
eventValue.put(AFInAppEventParameterName.CONTENT_ID,"1234567");
eventValue.put(AFInAppEventParameterName.CURRENCY,"USD");
AppsFlyerLib.getInstance().trackEvent(getApplicationContext() , AFInAppEventType.PURCHASE , eventVal
```

Kotlin

```
val eventValue = HashMap<String, Any>()
eventValue.put(AFInAppEventParameterName.REVENUE,1234.56)
eventValue.put(AFInAppEventParameterName.CONTENT_TYPE,"Shirt")
eventValue.put(AFInAppEventParameterName.CONTENT_ID,"1234567")
eventValue.put(AFInAppEventParameterName.CURRENCY,"USD")
AppsFlyerLib.getInstance().trackEvent(getApplicationContext() , AFInAppEventType.PURCHASE , eventVal
```

上面的购买事件具有与之相关的收入$ 1234.56，在控制面板上显示为收入。

记录负收入

在某些情况下，您想记录负收入。

例如，用户收到你售出的鞋子的退款。

Java

```
Map<String, Object> eventValue = new HashMap<String, Object>();
eventValue.put(AFInAppEventParameterName.REVENUE,-200);
eventValue.put(AFInAppEventParameterName.CONTENT_TYPE,"shoes");
eventValue.put(AFInAppEventParameterName.CONTENT_ID,"4875");
eventValue.put(AFInAppEventParameterName.CURRENCY,"USD");
AppsFlyerLib.getInstance().trackEvent(getApplicationContext() , "cancel_purchase" , eventValue);
```

Kotlin

```
val eventValue = HashMap<String, Any>()
eventValue.put(AFInAppEventParameterName.REVENUE, -200)
eventValue.put(AFInAppEventParameterName.CONTENT_TYPE, "category_a")
eventValue.put(AFInAppEventParameterName.CONTENT_ID, "1234567")
eventValue.put(AFInAppEventParameterName.CURRENCY, "USD")
AppsFlyerLib.getInstance().trackEvent(applicationContext,  "cancel_purchase", eventValue)
```

注意

请注意上面代码中的以下内容:

1. 收入值前面有一个减号
2. 事件名称具有 "cancel_purchase" 的唯一值-允许您在控制面板和原始数据报告中识别负收入事件

5.3应用内购买验证

AppsFlyer的SDK为应用内购买事件提供服务器验证服务。要验证购买事件，请调用 `validateAndTrackInAppPurchase` 。

该方法调用将自动生成一个 `af_purchase` 应用内事件。

```
public static void validateAndTrackInAppPurchase(Context context,
String publicKey, String signature, String purchaseData,
String price, String currency, HashMap<String, String> additionalParameters);
```

方法参数

- 字符串publicKey-Google Developer Console的公共密钥。
- 字符串签名–交易签名（购买完成后从Google API返回）。
- 字符串PurchaseData –以JSON格式显示的已购产品（购买完成后从Google API返回）。
- 字符串收入–向AppsFlyer报告应用内事件收入。
- 字符串货币-报告给AppsFlyer的应用内事件货币。
- HashMap<String, String> 额外参数 – 出现在原始数据event_value字段的额外应用内事件参数。

购买验证成功和失败回调

如果您想知道验证购买是否成功，请在application类中实现registerValidatorListener。该监听器有两个回调块，一个用于“成功”，一个用于“失败”（出于任何原因，包括验证失败）。

Java

```
AppsFlyerLib.getInstance().registerValidatorListener(this,new
   AppsFlyerInAppPurchaseValidatorListener() {
     public void onValidateInApp() {
       Log.d(TAG, "Purchase validated successfully");
     }
     public void onValidateInAppFailure(String error) {
       Log.d(TAG, "onValidateInAppFailure called: " + error);
     }
});
```

Kotlin

```
AppsFlyerLib.getInstance().registerValidatorListener(this, object : AppsFlyerInAppPurchaseValidatorL
    override fun onValidateInApp() {
        Log.d(LOG_TAG, "Purchase validated successfully")
    }

    override fun onValidateInAppFailure(error: String) {
        Log.d(LOG_TAG, "onValidateInAppFailure called: $error")
   }
})
```

购买验证的用法示例:

Java

```
// Purchase object is returned by Google API in onPurchasesUpdated() callback
private void handlePurchase(Purchase purchase) {
    Log.d(LOG_TAG, "Purchase successful!");
    Map<String, String> additional_event_values = new HashMap<>();
    additional_event_values.put("some_parameter", "some_value");
    String price= "10";
    String currency = "USD";
    AppsFlyerLib.getInstance().validateAndTrackInAppPurchase(getApplicationContext(), PUBLIC_KEY, pu
}
```

Kotlin

```
// Purchase object is returned by Google API in onPurchasesUpdated() callback
private fun handlePurchase(Purchase purchase) {
   Log.d(LOG_TAG, "Purchase successful!");
   val additional_event_values = HashMap<String, String>()
   additional_event_values.put("some_parameter", "some_value");
   val price= "10";
   val currency = "USD";
   AppsFlyerLib.getInstance().validateAndTrackInAppPurchase(this, PUBLIC_KEY, purchase.getSignature(
}
```

验证应用内购买会自动将应用内购买事件发送到AppsFlyer。参见下面event_value参数中传递的示例数据:

```
{
   "some_parameter":"some_value", // from additional_event_values
   "af_currency":"USD", // from currency
   "af_content_id":"test_id", // from purchase
   "af_revenue":"10", // from revenue
   "af_quantity":"1", // from purchase
   "af_validated":true // flag that AF verified the purchase
}
```

注意

调用 `validateAndTrackInAppPurchase` 会自动生成一个**af_purchase** 应用内事件。自己另外发送此事件会创建事件重复上报机制。

5.4 应用内事件限制

- 事件名称：最多45个字符
- 事件值：不得超过1000个字符-如超过这个长度，我们可能会截断它
- 定价和收入：仅使用数字和小数点，例如5或5.2
- 价格和收入最多可以到小数点后5位，如5.12345
- 从Android SDK V4.8.1开始，应用内事件和其他SDK API都支持非英语字符。

5.5 记录应用内事件的示例

您可以通过调用 `trackEvent` 来记录应用内事件的名称和参数值。有关更多详细信息，请参见应用内事件文档。

以下是有关如何记录购买事件的简单示例。关于每个行业的现成代码片段列表，请参阅详细应用内事件指南

示例：应用内购买活动

Java

```
Map<String,Object> eventValues = new HashMap<>();
eventValues.put(AFInAppEventParameterName.REVENUE, 1200);
eventValues.put(AFInAppEventParameterName.CURRENCY, "JPY");
eventValues.put(AFInAppEventParameterName.CONTENT_TYPE, "Shoes");
AppsFlyerLib.getInstance().trackEvent(this, AFInAppEventType.PURCHASE, eventValues);
```

Kotlin

```
val eventValues = HashMap<String, Any>();
eventValues.put(AFInAppEventParameterName.REVENUE, 1200)
eventValues.put(AFInAppEventParameterName.CURRENCY, "JPY")
eventValues.put(AFInAppEventParameterName.CONTENT_TYPE, "Shoes")
AppsFlyerLib.getInstance().trackEvent(this, AFInAppEventType.PURCHASE, eventValues)
```

5.6记录离线应用内事件

如果用户在互联网连接不可用时启动事件，Appsflyer仍然可以记录该事件。它是这样工作的：

1. SDK 将事件发送到 AppsFlyer 的服务器并等待响应。

2 如果 SDK 没有收到响应200, 则该事件将存储在缓存中。
3 收到下一个响应200后, 存储的事件将重新发送到服务器。
4 如果缓存中有多个事件, 它们将被立即按序发送到服务器。

> **注意**
>
> SDK 的缓存最多可以存储40个事件, 这意味着只保存脱机发生的前40个事件。所有之后的事件都会被丢弃，直到下一个相应200。
>
> 原始数据中显示的事件时间是设备再次联机后事件发送到AppsFlyer的时间。不是事件发生的实际时间。

5.7记录应用内事件处理成功和失败的信息

您可以在记录应用内事件时设置监听器。监听器可以给两种场景定义逻辑:

- 应用内事件已成功记录。
- 记录应用内事件时发生错误。

Java

```
AppsFlyerLib.getInstance().trackEvent(getApplicationContext(), AFInAppEventType.PURCHASE, eventValue
      @Override
      public void onTrackingRequestSuccess() {
        Log.d(AppsFlyerLibCore.LOG_TAG, "onTrackingRequestSuccess");
      }
      @Override
      public void onTrackingRequestFailure(String error) {
        Log.d(AppsFlyerLibCore.LOG_TAG, "onTrackingRequestFailure: " + error);
      }
 });
```

Kotlin

```
AppsFlyerLib.getInstance().trackEvent(this@MainActivity, AFInAppEventType.PURCHASE, emptyMap(), obje
    override fun onTrackingRequestSuccess() {
       Log.d(AppsFlyerLibCore.LOG_TAG, "onTrackingRequestSuccess")

    override fun onTrackingRequestFailure(error: String) {
       Log.d(AppsFlyerLibCore.LOG_TAG, "onTrackingRequestFailure called: $error")
    }
})
```

6. 用OneLink进行深度链接

OneLink是AppsFlyer的对平台归因、 **跳转和深度链接**的解决方案。

6.1设备检测和跳转

OneLink在设备点击时检测到设备类型，并将用户重定向到匹配的目的地，例如Google Play商店、iOS应用商店、第三方应用商店、或者网页。

OneLink重定向指南讨论了如何实现多平台归因链接（无需SDK编码）。这也是深层链接的基础。

6.2 深度链接

深度链接使您可以将用户发送到特定活动，并为他们提供自定义内容。该功能在运行再营销广告系列时特别有用。

要运用OneLink进行深度链接， **有权访问AppsFlyer控制面板的市场营销人员和有权访问该应用的开发人员必须共同合作** 。

请参阅我们的用于深度链接的URI Scheme设置指南 。

6.3 延迟深度连结

延迟深度链接可以深度链接新用户，并在他们安装应用后为其展示自定义内容。这不同于常规的深度链接，在常规深度链接中，应用必须已经安装在用户设备上。

要用OneLink实现延迟深度链接，开发人员需要有访问AppsFlyer控制面板的权限。

延迟深度链接的设置与深度链接相同。唯一的区别是，您需要在应用中实现其他逻辑，以便在用户安装和启动应用后将他们深度链接到自定义内容。

请参阅深度链接设置指南了解更多信息。

6.4 获取深度链接数据

在每次安装或深度链接发生后，SDK都会为您提供转化或互动数据。您可以使用此数据并配置代码逻辑来触发自定义内容或应用内的特定活动

请参阅深度链接设置了解更多信息。

7. 获取转化数据

在SDK层级，您可以实时的获取每一次新安装用户的归因数据

通过这个方式，你可以为用户提供个性化的内容，或者将他们带到App内的指定activity页面 (请查看本篇文章里的 deferred deep linking).这会大幅提高用户与App的互动程度。

要从Android SDK获取AppsFlyer的转换数据，请执行 `AppsFlyerConversionListener` 。

Java

```
import android.app.Application;
import com.appsflyer.AppsFlyerLib;
import com.appsflyer.AppsFlyerConversionListener;
import java.util.Map;
import android.util.Log;

public class AFApplication extends Application {
    private static final String AF_DEV_KEY = "qrdZGj123456789";

    @Override
    public void onCreate() {
        super.onCreate();
        AppsFlyerConversionListener conversionListener = new AppsFlyerConversionListener() {


            @Override
            public void onConversionDataSuccess(Map<String, String> conversionData) {

                    for (String attrName : conversionData.keySet()) {
                    Log.d("LOG_TAG", "attribute: " + attrName + " = " + conversionData.get(attrName));
                }
            }

            @Override
            public void onConversionDataFail(String errorMessage) {
           Log.d("LOG_TAG", "error getting conversion data: " + errorMessage);
            }

            @Override
            public void onAppOpenAttribution(Map<String, String> conversionData) {

                for (String attrName : conversionData.keySet()) {
                    Log.d("LOG_TAG", "attribute: " + attrName + " = " + conversionData.get(attrName));
                }

            }

            @Override
            public void onAttributionFailure(String errorMessage) {
                Log.d("LOG_TAG", "error onAttributionFailure : " + errorMessage);
            }
        };

        AppsFlyerLib.getInstance().init(AF_DEV_KEY, conversionListener, getApplicationContext());
        AppsFlyerLib.getInstance().startTracking(this);
    }
}
```

Kotlin

```
import com.appsflyer.AppsFlyerConversionListener
import com.appsflyer.AppsFlyerLib
import com.appsflyer.AppsFlyerLibCore.LOG_TAG

class AFApplication : Application() {
    private val devKey = "qrdZGj123456789";

    override fun onCreate() {
        super.onCreate()
        val conversionDataListener  = object : AppsFlyerConversionListener{
            override fun onConversionDataSuccess(data: MutableMap<String, Any>?) {
                data?.let { cvData ->
                    cvData.map {
                        Log.i(LOG_TAG, "conversion_attribute:  ${it.key} = ${it.value}")
                    }
                }
            }

            override fun onConversionDataFail(error: String?) {
                Log.e(LOG_TAG, "error onAttributionFailure :  $error")
            }

            override fun onAppOpenAttribution(data: MutableMap<String, String>?) {
                data?.map {
                    Log.d(LOG_TAG, "onAppOpen_attribute: ${it.key} = ${it.value}")
                }
            }

            override fun onAttributionFailure(error: String?) {
                Log.e(LOG_TAG, "error onAttributionFailure :  $error")
            }
        }

        AppsFlyerLib.getInstance().init(devKey, conversionDataListener, applicationContext)
        AppsFlyerLib.getInstance().startTracking(this)
    }
}
```

`AppsFlyerConversionListener` 接口中两个最重要的API是：

- `onInstallConversionData` 提供新安装的转化数据。**注意：** 从SDK Version 第5版开始，`onConversionDataSuccess` 是获取转化数据方法的名称。如果使用的SDK版本低于5.0.0，则该方法的名称为 `onInstallConversionDataLoaded` 。您需要将SDK升级到5.0.0版本要了解更多信息，请点击这里 。
- `onAppOpenAttribution` 在启动现有应用时（手动或通过深度链接）提供再营销的转化数据。

要了解有关转化数据的更多信息，请参见关于转化数据方案指南。

8.归因

第三方应用商店的安卓应用

使用AppsFlyer，您可以归因第三方商店的Android应用安装。这样一来，您就可以在无法使用Google Play的市场中推广应用并触达更大的受众

关于第三方应用商店安装归因的详情，请参阅此处。

预装应用的归因

在预安装广告系列中，应用所有者可以要求设备制造商在设备出厂前预装他们的应用。

使用AppsFlyer可以轻松对预装应用进行归因。用户首次启动您的应用时，AppsFlyer会将安装归因给作为媒体渠道的设备制造商。

如需了解详情，请点击此处。

监测卸载

AppsFlyer允许您测量不同渠道的用户卸载率。卸载评估可以帮助您根据这一重要KPI分析和优化广告系列。
要了解如何设置卸载监测，请在 此处阅读.

设置跟踪请求监听器

如果要收到 AppsFlyer 服务器成功接收跟踪请求的确认, 则可以实现 Appsflyer TrackingRequestListener 侦听器。

对于 SDK 发出的归因请求的每200个响应, 就会调用 onTrackingrequest

onTrackingRequestFailure(String error) 会被调用, 如果响应作为错误字符串返回。

实施实例

Java

```
AppsFlyerLib.getInstance().startTracking(this.getApplication(),"devKey", myListener());
private AppsFlyerTrackingRequestListener myListener() {
  return new AppsFlyerTrackingRequestListener() {
    @Override public void onTrackingRequestSuccess() {
      Log.d("Debug", "Got 200 response from server");
    }  @Override public void onTrackingRequestFailure(String error) {
      Log.d("Debug", error);
      }
    };
  }
```

Kotlin

```
AppsFlyerLib.getInstance().startTracking(this,"devKey", myListener())
private fun myListener() : AppsFlyerTrackingRequestListener{
   return object : AppsFlyerTrackingRequestListener{
      override fun onTrackingRequestSuccess() {
      Log.d("Debug", "Got 200 response from server");
   }
   override fun onTrackingRequestFailure(error: String?) {
     Log.d("Debug", error);
   }
  }
}
```

设置其他自定义数据

setAdditionalData API 需要在 SDK 级别与若干外部合作伙伴平台对接，包括 Segment、Adobe 和 Urban Airship。仅在平台对接文章明确说明需要 ***setAdditionalData*** API 时，才使用该 API。

`setAdditionalData` 代码示例：

Java

```
HashMap<String,Object> CustomDataMap = new HashMap<>();
CustomDataMap.put("custom_param_1","value_of_param_1");
AppsFlyerLib.getInstance().setAdditionalData(CustomDataMap);
```

Kotlin

```
val customDataMap = HashMap<String, Any>()
customDataMap.put("custom_param_1","value_of_param_1")
AppsFlyerLib.getInstance().setAdditionalData(customDataMap)
```

9. 会话

自定义会话间的时间

默认情况下，在2次应用启动之间必须至少间隔 5秒，才能将其计为2次独立的应用启动 (更多有关应用启动计数的信息)。

使用以下API设置会话之间的最短时间：

```
appsflyerlib. setmintimemebetweensession (intseconds);
```

将两次启动之间的自定义时间设置为**高值 可能严重影响依赖会话数据的API** （例如深度链接） 。

面向实用应用的后台会话

您可以使用此SDK方法报告新的用户会话。例如，这可能对于在后台运行的工具类应用很方便。

在活动的onCreate（）中使用此API：

```
public void reportTrackSession(Context context);
```

用法示例：

```
AppsFlyerLib.getInstance().reportTrackSession(context);
```

10.自有媒体

解决已包装的深度链接 URL

某些第三方服务，例如电子邮件服务提供商用他们自己的点击记录域名把链接包在电子邮件中。有些甚至允许您设置自己的点击记录域名。如果OneLink包在此类域名中，则可能会限制其功能。

要解决此问题，您可以使用 `setResolveDeepLinkURLs` API。使用此API从启动应用程序的点击域中获取OneLink。请确保在 SDK 初始化之前调用此 API。

例如，您有三个点击域重定向到您的OneLink，即https://mysubdomain.onelink.me/abCD。使用此API获取您点击域重定向到的OneLink。此API方法接收SDK解析的域列表。在SDK初始化之前添加以下代码。

```
AppsFlyerLib.getInstance().setResolveDeepLinkURLs("clickdomain.com", "myclickdomain.com", "anothercl
```

上面的代码使您可以在保留OneLink功能的同时使用点击域名。点击域名负责启动应用。该API会从这些点击域名中获取OneLink，然后您可以使用此OneLink中的数据进行深度链接和自定义用户内容。

记录推送通知

借助AppsFlyer，您可以记录作为再营销广告重要部分的推送通知。

要启用此功能，请在点击通知后启动的每个活动的 `onCreate` 方法内调用 `sendPushNotificationData` ：

```
AppsFlyerLib.getInstance().sendPushNotificationData(this);
```

有关推送通知测量的更多信息，请参阅此处 。

用户邀请归因

允许您的现有用户邀请他们的朋友和联系人作为新用户加入应用，这可能是应用增长的关键因素。使用AppsFlyer，您可以在应用中归因和记录源自用户邀请的安装。

有关详细信息，请参见用户邀请归因文章 。

交叉推广归因

交叉推广应用可能是增加应用安装量的主要增长因素。AppsFlyer使您可以归因和记录源自当前应用用户的交叉推广安装。

有关详细信息，请点击此处参阅交叉促销归因文章。

11. 用户标识符

获取AppsFlyer ID

将为每个新增安装创建一个AppsFlyer ID。您可以将AppsFlyer ID用于多种用途：

- 发送服务器到服务器应用内事件 。
- 将AppsFlyer ID与您后台的用户记录进行匹配。
- 合并 pull 和 push API时映射条目。

使用以下API获取AppsFlyer ID：

```
public String getAppsFlyerUID(Context context);
```

使用示例：

```
String appsFlyerId = AppsFlyerLib.getInstance().getAppsFlyerUID(this);
```

设置客户用户 ID

把自有的Customer User ID发给AppsFlyer可以交叉引用自有唯一ID、AppsFlyer ID以及其他标识符。可以在AppsFlyer 原始数据报告中看到Customer User ID。您还可以在回传API中使用它，以便与内部ID进行交叉引用。
要设置您的客户用户 ID：

```
public void setCustomerUserId(String id);
```

使用示例：

```
AppsFlyerLib.getInstance().setCustomerUserId("myId");
```

我们建议在应用流早期设置Customer User ID，因为它仅会关联设置之后上报的事件：

- 如果在调用 `startTracking` 之前调用 `setCustomerUserId` ，则Customer User ID将出现在安装和应用内事件的原始数据报告中。
- 如果在此之后设置，则“Customer User ID”仅关联该设置之后记录的事件。

获取客户用户 ID

为避免在首次启动后再次设置“Customer User ID”值，并减少向服务器请求Customer User ID的次数，可以使用以下方法检查其值是否为空：

```
AppsFlyerProperties.getInstance().getString(AppsFlyerProperties.APP_USER_ID)
```

了解客户用户 ID 的更多信息，请点击此处。

为 customerUserID 延迟 SDK 初始化

您可以延迟SDK初始化，直到设置了customerUserID。

指示 SDK 应延迟客户用户 ID 调用的初始化

```
AppsFlyerLib.getInstance().waitForCustomerUserId(true);
```

紧接 init () 方法之前。SDK 初始化的其他部分保持不变。

有了 customerUserID 之后，请调用

```
AppsFlyerLib.getInstance().setCustomerIdAndTrack("customer_id", this);
```

为SDK提供相关的客户用户ID并启动SDK。

代码应如下所示：

Java

```
public class AFApplication extends Application {
  private static final String AF_DEV_KEY = "qrdZGj123456789";
```

```
    @Override
    public void onCreate() {
      super.onCreate();
      AppsFlyerConversionListener conversionDataListener =
      new AppsFlyerConversionListener() {
        ...
      };
      AppsFlyerLib.getInstance().waitForCustomerUserId(true);
      AppsFlyerLib.getInstance().init(AF_DEV_KEY, getConversionListener(), getApplicationContext());
      AppsFlyerLib.getInstance().startTracking(this);
      // Do your magic to get the customerUserID
      // ...
      // any AppsFlyer SDK code invoked here will be discarded
     //Call the following API once the customerUserID is available:
  AppsFlyerLib.getInstance().setCustomerIdAndTrack("customer_id", this);
    }
}
```

Kotlin

```
class AFApplication: Application() {
 private val afDevKey = ""

 override fun onCreate() {
  super.onCreate()
  val conversionDataListener = object: AppsFlyerConversionListener {
   ...
  }
  AppsFlyerLib.getInstance().waitForCustomerUserId(true);
  AppsFlyerLib.getInstance().init(afDevKey, conversionDataListener, this)
  AppsFlyerLib.getInstance().startTracking(this)
  // Do your magic to get the customerUserID
  // ...
  // any AppsFlyer SDK code invoked here will be discarded
  // Call the following API once the customerUserID is available:
  AppsFlyerLib.getInstance().setCustomerIdAndTrack("customer_id", this)
 }
}
```

要了解有关将SDK初始化延迟到客户用户ID可用之后的更多信息，请在此处 。

⚠ 警告

仅在适合您的业务逻辑的情况下使用此 API。使用此 API 会提高数据差异的几率，并且可能使应用更容易暴露于诈骗中。

Google Advertising ID

从 SDK 4.8.0 版起，AppsFlyer 自动采集 `google_advertising_id`。

采集 Google Advertising ID 的要求仅涵盖 SDK 4.7.X 及以下版本。

OAID、IMEI和Android ID

必须收集至少一个唯一标识符以进行归因。可获得以下标识符：GAID、Android ID、IMEI和OAID。

GAID-Google Advertising ID

从包含Google Play服务的应用中自动收集。**如果GAID可用，则应用程序不应收集IMEI和Android ID，以避免违反Google Play政策** 。

IMEI 和 Android ID

默认禁用。仅适用于不包含Google Play服务的应用。
注意： 从于2019年末发布的Android 10（API级别29），对IMEI参数的访问受到限制。

要将这些ID发送到AppsFlyer：

1. 在应用上打开，收集设备IMEI和/或Android ID。
2. 在调用 `startTracking` 方法之前，请调用以下API：

```
AppsFlyerLib.getInstance().setImeiData("IMEI_HERE");
AppsFlyerLib.getInstance().setAndroidIdData("ANDROID_ID_HERE");
```

OAID- 打开广告商标识符

OAID实施指南

选择退出设备ID收集

开发人员可以选择以下方式退出收集IMEI、Android ID和OAID：

```
AppsFlyerLib.getInstance().setCollectIMEI(false);
AppsFlyerLib.getInstance().setCollectAndroidID(false);AppsFlyerLib.getInstance().setCollectOaid(fals
```

12. 用户隐私

退出

在某些极端情况下，您可能出于法律和隐私合规方面的考虑，希望关闭所有 SDK 追踪。这可以通过 ***isStopTracking*** API 来实现。调用此API后，SDK将停止运行，并且不再与AppsFlyer服务器通信。

```
AppsFlyerLib.getInstance().stopTracking(true,context);
```

有几种不同的方案为用户选择退出。**我们强烈建议您遵循与你的应用相关的方案的确切说明。**

在任何情况下，可调用相同 API 以重新激活 SDK，但会传递 False。

> **注意事项**

如果 `isStopTracking` 设置为 `true` 则不要调用 `startTracking`

要在 `stopTracking` 设置为 `false` 后再次开始跟踪, 请使用以下 SDK API:

```
AppsFlyerLib.getInstance().startTracking(getApplicationContext(), AF_DEV_KEY);
```

⚠ 警告

仅在您完全不想追踪该用户时，使用 stopTracking API。使用此 API 将严重影响您的归因、数据收集以及 深度链接 机制。

匿名用户数据

AppsFlyer 为您提供在 AppsFlyer 分析中匿名特定用户标示符的方法。该方法符合最新的隐私要求以及Facebook数据和隐私策略。其默认值为NO，表示默认情况下不执行匿名化。
在 SDK 初始化过程中，使用该 API 明确匿名用户安装、事件和互动：

```
public void setDeviceTrackingDisabled(boolean isDisabled);
```

使用示例：

```
AppsFlyerLib.getInstance().setDeviceTrackingDisabled(true);
```

可以把调用 `deviceTrackingDisabled` 设置为 **false** 来重新启动归因。

⚠ 警告

对用户匿名会严重影响您的归因信息。
只在法律要求您不得收集用户信息的地区使用此选项。

---

getAppsFlyerUID

描述	获取AppsFlyer ID。有关更多信息，请参见此处 。
方法签名	`public String getAppsFlyerUID(Context context);`
用法示例	`String appsFlyerId = AppsFlyerLib.getInstance().getAppsFlyerUID(this);`

onAppOpenAttribution

描述	通过深度链接打开应用时获取深度链接数据。
方法签名	`public void onAppOpenAttribution(Map<String, String> conversionData)`

onAppOpenAttributionFailure

描述	处理获取深度链接数据时的报错。
方法签名	`public void onAttributionFailure(String errorMessage)`

onConversionDataSuccess

描述	安装后获取转化数据。对于延迟深度链接很有用。 **注意：** 从SDK Version 第5版开始， `onConversionDataSuccess` 是获取转化数据方法的名称。如果使用的SDK版本低于5.0.0，则该方法的名称为 `onConversionDataReceived` 。我们建议您升级到SDK 5.0.0。要了解更多信息，请点击这里 。
方法签名	`public void onConversionDataSuccess(Map<String, String> conversionData)`

onConversionDataFail

描述	无法获取安装转化数据时处理错误。
方法签名	`public void onConversionDataFail(String errorMessage)`

onTrackingRequestFailure

描述	AppsFlyerTrackingRequestListener的回调方法。当SDK无法报告应用启动时调用。
方法签名	`public void onTrackingRequestFailure(String error);`
用法示例	请参阅设置归因请求监听器 。

onTrackingRequestSuccess

描述	AppsFlyerTrackingRequestListener的回调方法。当SDK成功报告应用启动时调用。
方法签名	`public void onTrackingRequestSuccess();`
用法示例	请参阅设置归因请求监听器 。

reportTrackSession

描述	如果您的应用是在后台运行的工具类应用的话，则报告会话。
方法签名	`public void reportTrackSession(Context context);`
用法示例	`AppsFlyerLib.getInstance().reportTrackSession(this);`

sendDeepLinkData

描述	此方法可用于任何深度链接的活动。 即使用户被深度链接到特定活动，此方法也可以确保您获取归因数据。 有关更多信息，请参见这里 。
方法签名	`public void sendDeepLinkData(Activity activity);`
用法示例	`AppsFlyerLib.getInstance().sendDeepLinkData(this);`

sendPushNotificationData

描述	测量并从推送通知活动中获取数据。有关更多信息，请参阅测量推送通知 。
方法签名	`public void sendPushNotificationData(Activity activity);`
用法示例	`AppsFlyerLib.getInstance().sendPushNotificationData(this);`

setAdditionalData

描述	添加要发送到外部合作伙伴平台的其他数据。
方法签名	`public void setAdditionalData(HashMap<String, Object> customData);`
用法示例	参见设置附加数据 。

setAndroidIdData

描述	将Android ID发给AppsFlyer。参见 OAID、IMEI和Android ID 。
方法签名	`public void setAndroidIdData(String aAndroidID);`
用法示例	`AppsFlyerLib.getInstance().setImeiData("Android ID");`

setAppInviteOneLink

描述	设置用于为用户邀请创建自定义归因链接的OneLink模板ID。
方法签名	`public void setAppInviteOneLink(String oneLinkId);`
用法示例	请参阅将 OneLink用于用户邀请归因 。

setCollectAndroidID

描述	表明是否应将Android ID发给AppsFlyer。
方法签名	`public void setCollectAndroidID(boolean isCollect);`
用法示例	参见 OAID、IMEI和Android ID 。

setCollectIMEI

描述	表明是否应将IMEI发给AppsFlyer。
方法签名	`public void setCollectIMEI(boolean isCollect);`

描述	表明是否应将IMEI发给AppsFlyer。
用法示例	参见 OAID、IMEI和Android ID 。

setCustomerIdAndTrack

描述	客户用户ID可用后，启动SDK。有关更多信息，请参见为客户用户ID延迟SDK初始化 。
方法签名	`public void setCustomerIdAndTrack(String id, @NonNull Context contex);`
用法示例	`AppsFlyerLib.getInstance().setCustomerIdAndTrack(customer_id", this);`

设置Customer User ID

描述	设置客户用户标识 (Customer User ID)。有关更多信息，请参见设置Customer User ID 。
方法签名	`public void setCustomerUserId(String id)`
用法示例	`AppsFlyerLib.getInstance().setCustomerUserId("customer_id")`

setDebugLog

描述	启用调试日志。请参阅Android 调试日志。
方法签名	`public void setDebugLog(boolean shouldEnable);`
用法示例	`AppsFlyerLib.getInstance().setDebugLog(true);`

setDeviceTrackingDisabled

描述	匿名化用户的安装、事件和会话。有关更多信息，请参见匿名化用户数据 。
方法签名	`public void setDeviceTrackingDisabled(boolean isDisabled);`

描述	匿名化用户的安装、事件和会话。有关更多信息，请参见匿名化用户数据 。
用法示例	`AppsFlyerLib.getInstance().setDeviceTrackingDisabled(true);`

setLogLevel

描述	设置AppsFlyer SDK日志级别。
方法签名	`public void void setLogLevel(AFLogger.LogLevel logLevel)`
用法示例	`AppsFlyerLib.getInstance().setLogLevel(AFLogger.LogLevel.INFO);`

setMinTimeBetweenSessions

描述	设置会话之间的最短时间。有关更多信息，请参阅自定义会话间歇。
方法签名	`public void setMinTimeBetweenSessions(int seconds);`
用法示例	`AppsFlyerLib.getInstance().setMinTimeBetweenSessions(10);`

setOaidData

描述	将OAID发给AppsFlyer。参见 OAID、IMEI和Android ID 。
方法签名	`public void setOaidData(String oaid);`
用法示例	`AppsFlyerLib.getInstance().setOaidData("OAID");`

setOutOfStore

描述	指定下载应用的备用应用商店。
方法签名	`public void setOutOfStore(String sourceName);`

描述	指定下载应用的备用应用商店。
用法示例	`AppsFlyerLib.getInstance().setOutOfStore("baidu");`

setPreinstallAttribution

描述	设置SDK在预安装的应用启动时报告预安装。
方法签名	`public void setPreinstallAttribution(String mediaSource, String campaign, String`
用法示例	请参阅用于Android的预装广告系列 。

setResolveDeepLinkURLs

描述	解决点击域名中的OneLink。有关更多信息，请参见深度链接URL包装解析 。
方法签名	`public void setResolveDeepLinkURLs(String... urls);`
用法示例	`AppsFlyerLib.getInstance().setResolveDeepLinkURLs("example.com", "click.example.`

startTracking

描述	在应用启动时启动SDK。有关更多信息，请参见初始化SDK 。
方法签名	`public void startTracking(Application application);`
用法示例	`AppsFlyerLib.getInstance().startTracking(this);`

stopTracking

描述	关闭所有SDK功能。有关更多信息，请参阅用户隐私退出 。
方法签名	`public void stopTracking(boolean isTrackingStopped, Context context);`

描述	关闭所有SDK功能。有关更多信息，请参阅用户隐私退出 。
用法示例	`AppsFlyerLib.getInstance().stopTracking(true, this);`

trackAppLaunch-已弃用

描述	**该方法已弃用。请改用 startTracking 。** 有两个功能： ■ 将应用启动事件即时发送到AppsFlyer。 ■ 如果使用stopTracking关闭了SDK功能，请重新启动SDK功能。
方法签名	`public void trackAppLaunch(Context context, String devKey);`
用法示例	`AppsFlyerLib.getInstance().trackAppLaunch(this, AF_DEV_KEY);`

trackEvent

描述	将应用内事件发给AppsFlyer。有关更多信息，请参阅记录应用内事件 。
方法签名	`public void trackEvent(Context context, String eventName, Map<String, Object> ev`
用法示例	`AppsFlyerLib.getInstance().trackEvent(this, AFInAppEventType.PURCHASE, eventValu`

updateServerUninstallToken

描述	对于将Firebase用于卸载测量以外目的的开发人员。有关更多信息，请参阅卸载测量 。
方法签名	`public void updateServerUninstallToken(Context context, String token)`
用法示例	`AppsFlyerlib.getInstance().updateServerUninstallToken(getApplicationContext(),`

waitForCustomerUserId

描述	延迟SDK初始化，直到设置了客户用户ID。
方法签名	`public void waitForCustomerUserId(boolean wait);`
用法示例	`AppsFlyerLib.getInstance().waitForCustomerUserId(true);`