目录

新义 SDK

标签：SDK

v3.2.0 2025.02.26

基础信息

- 开发者：新义互联（北京）科技有限公司
- 隐私声明
- 主要功能：用于从我们的系统获取广告

SDK下载

通常情况下，不需要下载后使用，直接使用 maven 库引用方式就可以（后面有详细说明），如果因为某些特殊情况必须下载开发包，可以通过下面的地址进行下载：https://nexus.mobrtb.com/repository/sdk/com/xy/sdk/adtalos-sdk/3.2.0/adtalos-sdk-3.2.0.aar

文档更新记录

版本	作者	时间
v3.0.0	马秉尧	2024.07.18
v3.0.1	马秉尧	2024.07.30
v3.1.0	马秉尧	2024.08.01
v3.2.0	马秉尧	2025.02.26

开发者后台

访问地址 https://developer.mobrtb.com

开发者后台为开发者主要提供以下功能

- 账号注册与登录
- 应用管理
- 广告位管理

账号注册与登录

访问登录页面 https://developer.mobrtb.com/web/#/login

进入注册页面 填写相关的信息, 进行账号注册

应用管理

通过左侧导航, 进入应用信息下的应用列表页面, 该页面可以查看应用列表数据, 并进行查看,添加,编辑等的操作

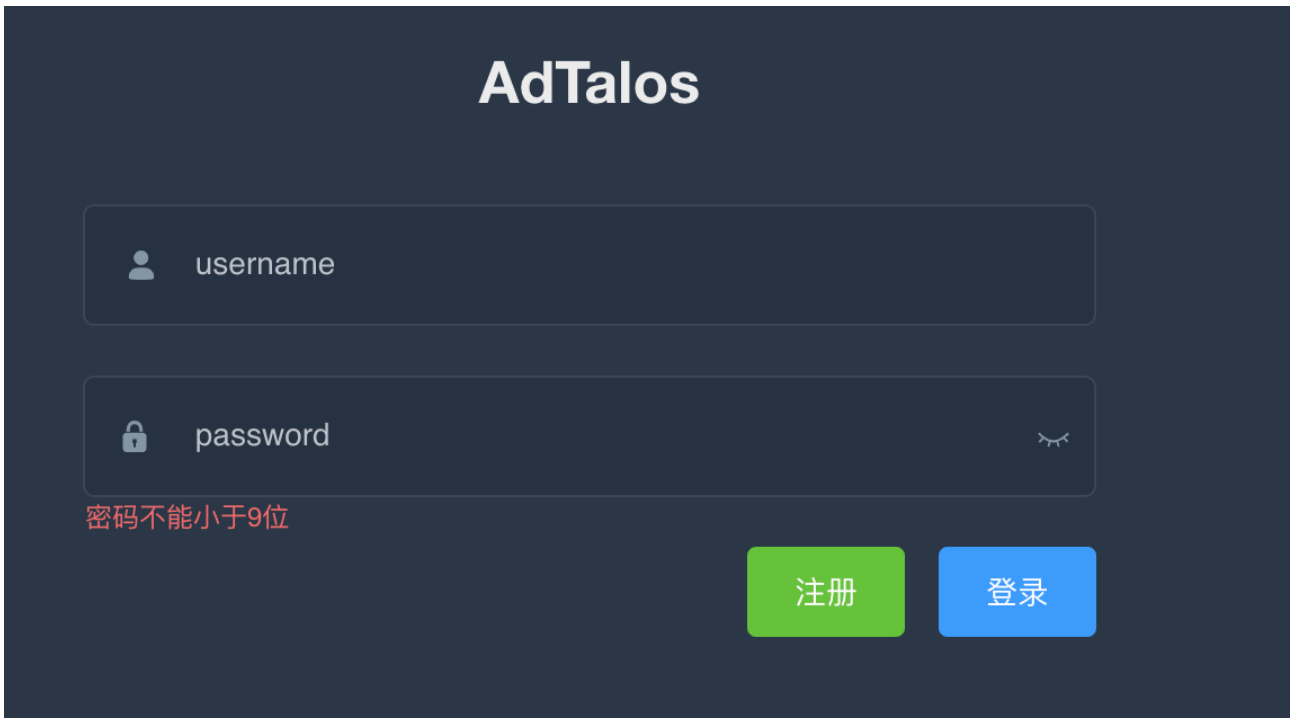

Figure 1: 登录页面

应用添加　通过应用列表页面上方的添加按钮, 进入到应用添加页面

::: tip 注意 接入方式请选择SDK :::

应用状态　在应用列表页可以查看应用的状态, 应用有审核中, 正常, 审核未通过, 暂停四种状态

- 在应用添加之后, 应用状态为审核中, 运营人员会对您的应用进行审核
- 若您的应用未审核通过, 应用状态会变为审核未通过
- 若您的应用审核通过, 应用状态会变为正常
- 若您的应用暂时不想投放广告, 可以点击列表中的暂停按钮, 应用状态会变为暂停

应用编辑　通过应用列表中的编辑按钮, 可以进入到应用编辑页面

::: tip 注意 某些关键信息, 比如操作系统, 包名等, 一旦应用审核通过, 是不可以再次变更的 :::

广告位管理

通过左侧导航, 可以进入应用信息下的广告位列表页, 该页面可以查看广告位列表数据, 并进行查看,添加,编辑的操作

广告位添加　通过广告位列表页面上方的添加按钮, 进入到广告位添加页面

Figure 2: 注册

Figure 3: 应用管理

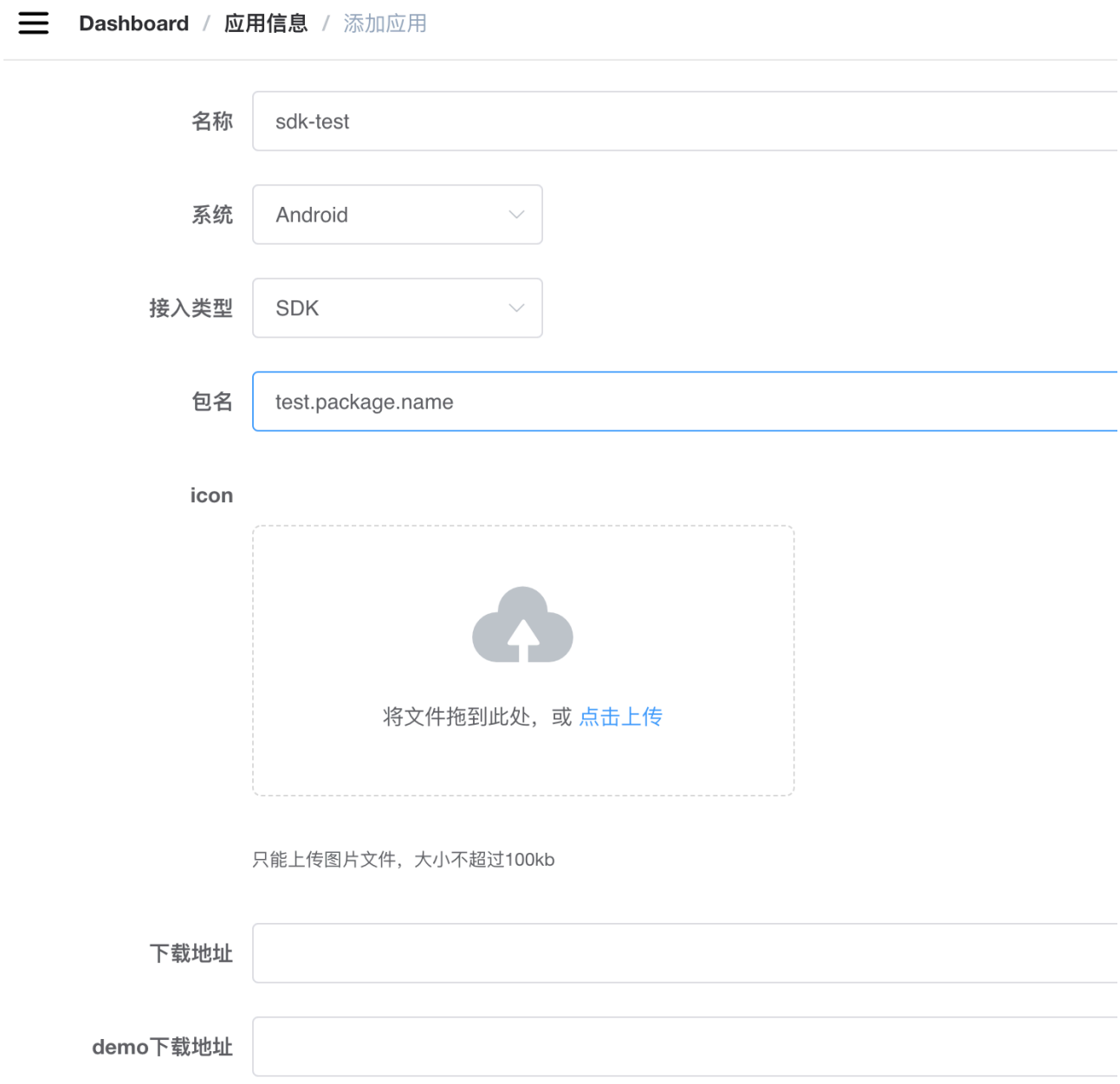

Figure 4: 应用添加

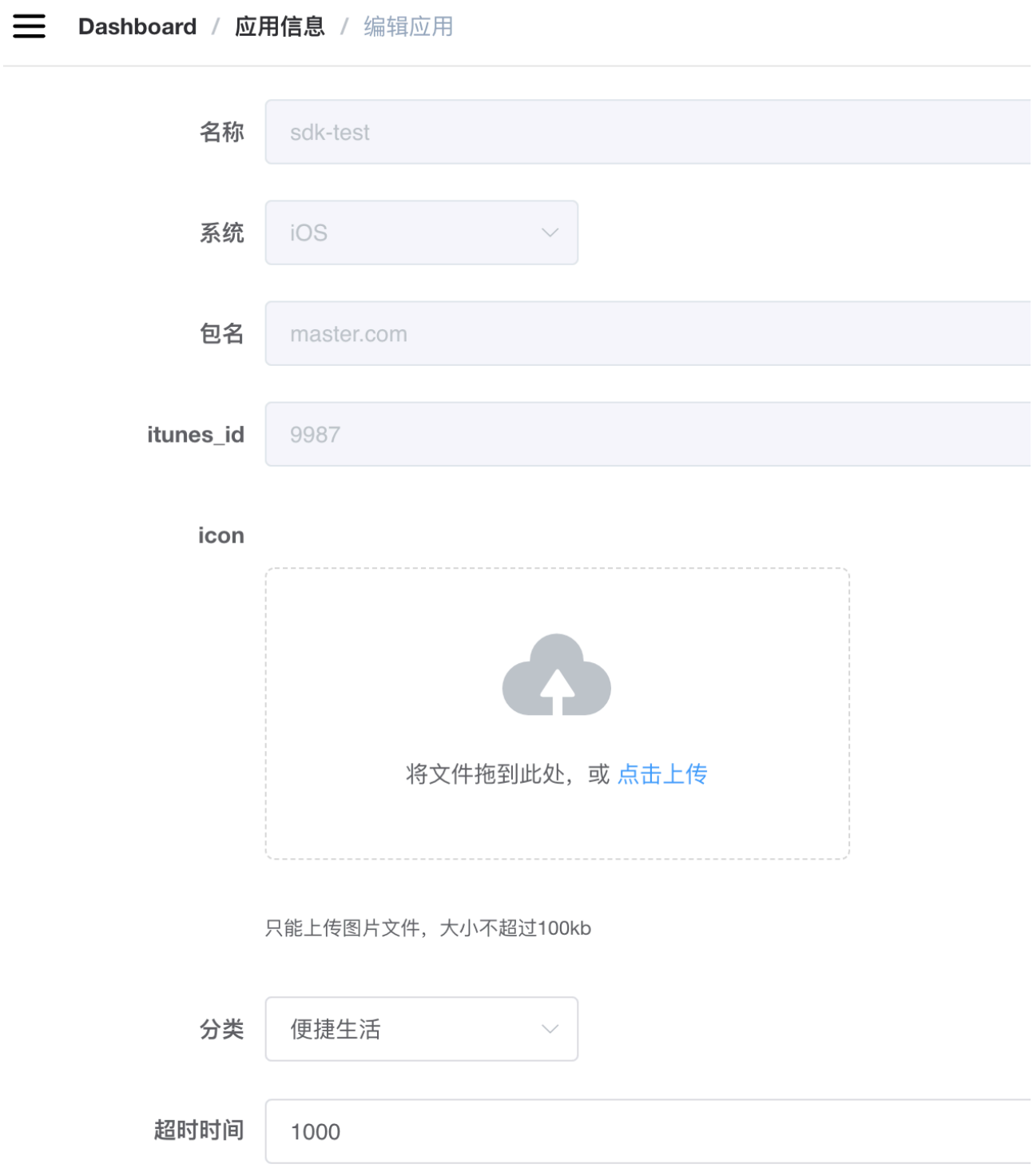

Figure 5: 应用编辑

Figure 6: 广告位添加

Figure 7: 广告位编辑

广告位编辑 通过广告位列表中的编辑按钮, 可以进入到广告位编辑页面

::: tip 注意 某些关键信息, 比如名称,广告形式等, 一旦保存成功, 是不可以再次变更的 :::

SDK用法

在这里，我们以创建一个简单的演示程序为例，介绍新义 SDK 的基本用法。

基本用法

创建应用 在 Android Studio 中，选择新建项目（New Project...）。

项目名称和包名按照实际需要填写。

语言没有限制，你可以根据自己的喜好选择 Java 或者 Kotlin。这里我们选 Java。

Minimum API level 这里需要注意，虽然我们的 SDK 最低支持的版本为 API 11：Android 3.0。但是因为广告是使用 WebView 来显示的，而 Android 5.0 以下的系统的 WebView 对 HTML5 支持的很不好，如果系统版本小于 Android 5.0 的话，可能大部分广告都不能正常展示。所以，如果您的 App 支持 Android 5.0 以下的系统，那么最好在 App 中判断一下当前运行的系统版本，当系统版本大于等于 Android 5.0 再调用广告 SDK 来显示广告。

修改 build.gradle 配置 接下来打开 Module: app 的 build.gradle 文件或者 settings.gradle 中。在配置部分加入：

```
repositories {
    maven { url 'https://nexus.mobrtb.com/repository/sdk' }
}
```

然后在 dependencies 中，加入：

```
    implementation 'com.xy.sdk:adtalos-sdk:3.2.0'
```

编辑完之后，看起来大致是这样的：

build.gradle

```
plugins {
    id 'com.android.application'
}
android {
    compileSdk 35
    defaultConfig {
        applicationId "com.xy.demo"
        minSdk 21
        targetSdk 35
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
               proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-
rules.pro'
        }
    }
    namespace 'com.xy.demo'
}

dependencies {
    implementation 'com.xy.sdk:adtalos-sdk:3.2.0'
    implementation 'androidx.appcompat:appcompat:1.7.0'
    implementation 'com.google.android.material:material:1.12.0'
}
```

settings.gradle

```
pluginManagement {
    repositories {
        google()
        mavenCentral()
        gradlePluginPortal()
    }
}
dependencyResolutionManagement {
    repositories {
        google()
        mavenCentral()
        maven { url 'https://nexus.mobrtb.com/repository/sdk' }
        maven { url 'https://developer.huawei.com/repo/' }
    }
}
include ':app'
rootProject.name='Demo'
```

::: tip 注意： 这里 `com.xy.sdk` 是默认的包名。如果需要定制包名，您可以在使用时，替换为您在 cms 开发者面板中填写的包名。 :::

新义 SDK 的最新版本可以从这里直接查看：

https://nexus.mobrtb.com/service/rest/repository/browse/sdk

在跟上面包名相对应的目录下，你可以找到属于您的专属 SDK 开发包。

修改 AndroidManifest.xml 配置 在 `application` 中，加入 `android:usesCleartextTraffic="true"` ，以允许可以通过 http 接收广告资源。

在 `activity` 中，加入 `android:configChanges="orientation|screenSize|keyboard|keyboardHidden"` ，避免因手机旋转导致 `onCreate` 反复执行而造成广告被多次重复加载，影响用户体验。

修改之后的 AndroidManifest.xml 文件看上去大致是这样的：

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-permission android:name="android.permission.REQUEST_INSTALL_PACKAGES" />

    <application
        android:name=".DemoApplication"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:usesCleartextTraffic="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".NativeActivity"
            android:configChanges="orientation|screenSize|keyboard|keyboardHidden"
            android:label="@string/title_activity_native"
            android:theme="@style/AppTheme.NoActionBar" />
        <activity
            android:name=".BannerActivity"
            android:configChanges="orientation|screenSize|keyboard|keyboardHidden"
            android:label="@string/title_activity_banner"
            android:theme="@style/AppTheme.NoActionBar" />
```

```
        <activity
            android:name=".MainActivity"
            android:configChanges="orientation|screenSize|keyboard|keyboardHidden"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

合规使用说明

- 合规使用说明

请求服务区域设置 自 2.7.3 版本之后，SDK 增加了区域设置，而且一直到 3.1.0 版本的默认设置都是 `Area.GLOBAL` 。如果跑国内流量，可以通过 `SDK.setArea` 方法进行区域设置，例如：

```
SDK.setArea(Area.CHINA);
```

将请求地址区域改为国内。

但是从 3.2.0 版本之后，默认设置已改回国内，所以使用最新版本时，不需要修改该设置。

创建横幅、原生广告

创建横幅和原生广告的方式是相同的，它们都是使用相同的 `com.xy.sdk.View` 来创建，只是属性值有所区别。

::: tip 注意 这里 `com.xy.sdk` 是默认的包名。您在使用时，请替换为您在 cms 开发者面板中填写的包名。下同 :::

通过 XML 创建横幅广告 在 Android 的 xml 布局文件中，在需要放置横幅广告的地方，插入以下代码即可：

```
        <com.xy.sdk.View
            android:id="@+id/banner_view"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            app:unit_size="banner_size"
            app:unit_id="209A03F87BA3B4EB82BEC9E5F8B41383" />
```

其中 `unit_size` 设置为横幅广告的尺寸（ `"banner_size"` ）， `unit_id` 为横幅广告位的 id，注意不要填错。

通过 XML 创建原生广告 在 Android 的 xml 布局文件中，在需要放置原生广告的地方，插入以下代码即可：

```
        <com.xy.sdk.View
            android:id="@+id/native_view"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            app:unit_size="native_size"
            app:unit_id="98738D91D3BB241458D3FAE5A5BF7D34" />
```

其中 `unit_size` 设置为原生广告的尺寸（ `"native_size"` ），其中 `unit_id` 为横幅广告位的 id，注意不要填错。

其它属性 除了 `unit_size`，`unit_id` 属性之外，还有 2 个特别的属性：`unit_width` 和 `unit_height`。

这 2 个属性跟 `unit_size` 一样都是用来设置广告尺寸的。

对于原生广告 `unit_size` 还可以设置其它的一些预设枚举值：

- `native_2to1_size`
- `native_3to2_size`
- `native_16to9_size`
- `native_4to3_size`
- `native_1to1_size`
- `native_11to4_size`
- 如果上面这些预设值都不合适，你可以使用 `unit_width` 和 `unit_height` 来设置合适的广告宽高值。

`unit_width` 和 `unit_height` 的值为整数，没有单位。

`native_size` 是一个特殊的值，其它取值都是固定宽高比的，而 `native_size` 是自适应内容宽高的。如果您的广告宽高比尺寸是固定的，请不要使用 `native_size` 这个尺寸预设值。

获取对象 在 Java 代码中，通过：

```
com.xy.sdk.View bannerView = findViewById(R.id.banner_view);
com.xy.sdk.View nativeView = findViewById(R.id.native_view);
```

可以获取到该广告对象。

直接通过 Java 代码创建横幅、原生广告

```
import android.app.Activity;
import android.view.ViewGroup;
import android.os.Bundle;

import com.xy.sdk.Size;
import com.xy.sdk.View;

import static android.view.ViewGroup.LayoutParams.MATCH_PARENT;
import static android.view.ViewGroup.LayoutParams.WRAP_CONTENT;

...
public class MainActivity extends Activity {
...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ...
        ViewGroup container = findViewById(R.id.container);
        ...
        View bannerView = new View(this);
        bannerView.setSize(Size.BANNER);
        container.addView(bannerView, MATCH_PARENT, WRAP_CONTENT);
        bannerView.load("CA868C57C6D992A2DDAAB7EAEE63D18B");

        View nativeView = new View(this);
        nativeView.setSize(Size.NATIVE_3TO2);
        container.addView(nativeView, MATCH_PARENT, WRAP_CONTENT);
        nativeView.load("98738D91D3BB241458D3FAE5A5BF7D34");
        ...
    }
```

```
...
}
```

资源回收 为了能够及时回收资源，在该广告对象所在的 Activity 对象中，加入以下方法：

```
@Override
protected void onDestroy() {
    bannerView.destroy();
    nativeView.destroy();
    super.onDestroy();
}

@Override
protected void onPause() {
    bannerView.pause();
    nativeView.pause();
    super.onPause();
}

@Override
protected void onResume() {
    bannerView.resume();
    nativeView.resume();
    super.onResume();
}
```

如果不加以上语句，对于带有视频的原生广告，视频可能仍然会在后台播放而无法关闭。

setSize 方法 该方法与使用 XML 中的 `app:unit_size`，`app:unit_width` 和 `app:unit_height` 属性设置广告尺寸是等价的。

load 方法 该方法用于设置广告位 id，并且加载广告。该方法只能执行一次。

多次执行该方法时，只有第一次设置的广告位 id 有效，后面的设置无效，而且也不会重复加载广告。

另外，当通过 XML 的 `app:unit_id` 属性设置了广告位 id 的情况下，试图通过执行该方法来修改广告位 id 也是无效的。

`load` 方法还有一个重载版本，重载版本的第一个参数跟上面的方法一致，第二个参数表示是否自动显示，如果您需要提前加载，延迟显示，可以调用该方法时，设置改参数的值为 `false`，则广告加载完之后，不会自动显示，当调用 `show` 或者 `render` 方法时才会显示，显示之前可以通过 `isLoaded` 方法来判断是否已经加载完毕，但通常你不需要调用它，因为在 `show` 和 `render` 方法内部有同样的判断，除非您需要在 `if` 语句的 `else` 语句中有其它的操作。

show 和 render 方法的异同点 `show` 和 `render` 方法都应该在 `onLoaded` 事件发生之后调用才会有效，在此之前调用是无效的。

它们两个的作用都是在广告加载之后，开发者可以手动控制广告的显示。但它们之间是有区别的。

`show` 方法在展示广告之后，会自动进行展示上报。而 `render` 方法不会自动进行展示上报。

`show` 方法会在展示广告之前检查广告视图是否处于可见状态，只有当处于可见状态时，`show` 方法才会进行广告展示并上报。而 `render` 方法不会做该检查，而是直接将广告内容显示在广告视图中。

`render` 方法因为不会进行展示上报，因此，开发者需要手动调用 `impressionReport` 方法进行展示上报。

impressionReport 方法 `impressionReport` 方法应该在 `onRendered` 事件发生之后调用才会有效，在此之前调用是无效的。

`impressionReport` 方法是与 `render` 方法对应的，如果不使用 `render` 方法，也不要使用 `impressionReport` 方法进行手动展示上报。

开发者不论在何种情况下，对 `impressionReport` 方法进行调用，都会进行展示上报，即使在广告处于不可见状态时也会进行展示上报，该方法不会在客户端对展示上报进行过滤，但是会将跟广告可见状态相关的各种数据上报给服务器，如果服务器发现展示上报时广告处于不可见状态，会判定为无效展示，如果无效展示过多，会判定为作弊行为。

开发者使用该方法时，请按照正确的步骤进行，否则，后果自负。

横幅广告轮转显示 默认情况下，广告不会自动轮转显示，如果需要对横幅广告开启自动轮转显示，可以使用 `setCarouselModeEnabled` 方法开启该功能，这样横幅广告就会根据后台设置的轮转时间（15秒或30秒），进行自动轮转显示了。

横幅广告轮转动画 默认情况下，横幅广告是直接显示出来的，如果需要在广告轮转时，有一个滑入滑出的动画效果，可以使用 `setAnimationEnabled` 方法开启该功能。

原生广告视频控制 对于原生广告可以通过 `getVideoController` 方法返回一个 `VideoController` 对象。

hasVideo 方法 如果原生广告中包含视频，该方法返回 `true` ，否则返回 `false` 。

play 方法 如果原生广告中包含视频，可以使用该方法开始视频播放。

pause 方法 如果原生广告中包含视频，可以使用该方法暂停视频播放。

mute 方法 如果原生广告中包含视频，可以通过该方法控制视频是否静音。设置为 `true` 为静音，设置为 `false` 取消静音。

isPlaying 方法 用来判断视频是否处于播放状态。

isEnded 方法 用来判断视频是否已播放完毕。

getMetadata 方法 如果原生广告中包含视频，通过该方法可以返回该视频的元信息。

注意: 只有当视频元信息加载完毕之后，该方法才能返回有效值，否则，返回 `null` 。

原生模板样式 原生广告的样式通常需要跟用户 app 的风格一致，因此，我们对原生广告提供了自定义 HTML [模板]功能（实际上我们对所有类型的广告都提供了自定义 HTML [模板]功能，只是其它类型的广告，即使不使用该功能也基本能满足需求）。因此，当您使用原生广告时，最好是按照自己 app 的风格来定制自己专用的模板。尤其是原生视频广告，默认模板使用的是 HTML 默认的视频控制栏，其美观度欠佳，您很可能需要自己定制样式才能满足需求。

如果您有好的样式，也可以推荐给我们，如果您推荐的样式能够满足大多数用户需求的话，我们会考虑更新默认的模板。

创建开屏、插屏、激励视频广告

开屏、插屏、激励视频都是通过创建一个 `com.xy.sdk.Controller` 类的对象来实现的。

::: tip 注意 这里 `com.xy.sdk` 是默认的包名。您在使用时，请替换为您在 cms 开发者面板中填写的包名。 :::

下面是创建开屏广告的代码：

```
// 创建并加载开屏广告对象，参数为开屏广告的广告位 id。
Controller splash = new Controller("5C3DD65A809B08A2D6CF3DEFBC7E09C7", Controller.SPLASH);
...
// 在合适的位置显示广告。
splash.show();
```

使用 `isLoaded` 方法可以判断广告是否已经加载完毕。注意: 调用 `show` 方法之后，会首先自动调用 `isLoaded` 方法判断广告是否加载完成，如果完成就显示，如果没有完成，会每隔 200 毫秒再次判断，直到广告显示出来。如果重试加载失败或者超过 3 秒钟超时，则停止继续判断。

另外，3 秒只是一个默认的超时时间，如果觉得默认超时判断时间太长或太短，可以使用 `show(timeout)` 方法来自行调整超时时间， `timeout` 是判断加载完成的超时时间，单位是毫秒，上面的 `show` 方法相当于 `show(3000)` ，注意该值不要设置的过大或过小。如果过小，广告可能永远不会被展示出来。

创建插屏广告和激励视频广告跟上面的方式类似，只需要把 `Controller.SPLASH` 分别换成: `Controller.INTERSTITIAL` 和 `Controller.REWARDED_VIDEO` ，就可以了，相关的广告位 id，也需要修改为插屏广告和激励视频广告的广告位 id。

这三种广告在显示方式上有所区别。 `SPLASH` 只有竖屏显示方式。 `INTERSTITIAL` 是根据当前屏幕方向来决定横竖屏显示方式。 `REWARDED_VIDEO` 广告在开发者后台可以设置横竖屏方式。

如果想载入和显示分离，可以先调用 `load` 方法。需要显示广告的时候再调用 `show` 方法。

沉浸模式 开屏、插屏、激励视频广告在显示时，默认开启沉浸模式。如果不想使用沉浸模式，可以使用 `setImmersiveMode` 方法，设置为 `false` 即可。

广告落地页设置

广告落地页有以下一些设置，这些设置都是可选的。

导航按钮工具栏

```
LandingPageActivity.setDisplayActionBarEnabled(true);
```

通过该设置可以开启导航按钮工具栏显示，默认是不显示的。

换页动画效果

```
LandingPageActivity.setAnimationEnabled(true);
```

通过该设置可以开启落地页内的换页动画效果，开启后，页面前进，后退会有滑入滑出的动画效果。

落地页全屏显示

```
LandingPageActivity.setFullScreenEnabled(true);
```

通过该设置可以开启或关闭落地页全屏显示效果，注意: 默认就是全屏显示。

落地页方向设置

```
LandingPageActivity.setOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
```

通过该设置可以设置落地页的显示方向，默认是不指定方向（可以随设备旋转而旋转）。

高级用法

新义 SDK 除了能够以默认的方式来展示广告之外，还提供了进一步自定义您的广告行为的能力。

您可以在广告生命周期内加入许多事件，如加载、打开、关闭等等。

另外，广告的展示方式您也可以通过自定义模板的方式来实现完全定制。

事件

标准事件 AdListener 该接口定义如下：

```
public interface Listener {
    // 请求广告前触发
    void onBeforeRequest();
    // 当广告渲染完毕后触发
    void onRendered();
    // 当广告展示完毕后触发
    void onImpressionFinished();
    // 对于横幅、原生广告，当重试超过设置的自动重试次数后仍然没有加载成功时触发。
    // 对于开屏，插屏，激励视频广告，当重试超过设置的自动重试次数后仍然没有加载成功，
    // 或者当超过展示超时（默认3秒）后仍然没有加载完成时触发。
    void onImpressionFailed();
    // 当广告展示过程中发生错误时触发
    void onImpressionReceivedError(int errorCode, String description);
    // 当广告加载完毕后触发
    void onLoaded();
    // 当广告加载失败后触发
    void onFailedToLoad(Exception e);
    // 对于横幅、原生广告，当落地页被全屏打开、遮挡住广告条时触发。
    // 对于开屏，插屏，激励视频广告，当这些广告本身显示时触发。
    void onOpened();
    // 当广告中的有效连接被点击时触发。
    void onClicked();
    // 当用户点击打开其他应用（如外部浏览器）时调用，从而在后台运行当前应用时触发。
    void onLeftApplication();
    // 对于横幅、原生广告，当落地页遮挡住广告条的落地页关闭时触发。
    // 对于开屏，插屏，激励视频广告，当这些广告本身关闭时触发。
    void onClosed();
}
```

不论何种形式的广告，都可以通过 `setListener` 方法来设置上面的事件，通过 `getListener` 方法获取已经设置的事件。

该接口中的每个事件方法都提供了空的默认实现，开发者在实现该接口时，只需要实现自己关心的事件方法即可，不必全部实现。

横幅或原生广告关闭事件 View.OnCloseListener 在 `View` 这个类上，我们定义了一个 `OnCloseListener` 事件：

```
    public interface OnCloseListener {
        boolean onClose();
    }
```

默认情况下，横幅和原生广告是不显示关闭按钮的。但是开发者可以在后台给横幅或原生广告设置是否显示关闭按钮，并且可以设置显示位置，大小，点击范围，当设置了显示关闭按钮后，用户可以通过点击横幅、原生广告上的关闭按钮来关闭该广告。当该广告被关闭时，该事件被触发。

开发者可以调用 `setOnCloseListener` 方法来设置该事件，通过 `getOnCloseListener` 方法获取已经设置的事件。

视频相关事件 VideoListener 该事件定义如下：

```
public interface VideoListener {
    // 当视频元数据被加载完毕后触发
    void onVideoLoad(VideoController.Metadata metadata);
    // 当视频第一次开始播放时触发
    void onVideoStart();
    // 当视频状态由未播放、播放结束或暂停状态变为播放状态时触发
    void onVideoPlay();
    // 当视频播放暂停时触发
```

```
    void onVideoPause();
    // 当视频播放结束时触发
    void onVideoEnd();
    // 当视频音量改变或者静音状态改变时触发
    void onVideoVolumeChange(double volume, boolean muted);
    // 当视频播放时，会不断触发该事件
    void onVideoTimeUpdate(double currentTime, double duration);
    // 当视频发生错误时触发
    void onVideoError();
    // 当视频中断时触发
    void onVideoBreak();
}
```

对于原生视频广告来说，可以通过 `getVideoController` 方法获取视频控制器，然后通过视频控制器对象上的 `setVideoListener` 方法来设置该事件，通过 `getVideoListener` 方法来获取已经设置的事件。

对于开屏，插屏，激励视频广告，可以直接通过 `Controller` 对象上的 `setVideoListener` 和 `getVideoListener` 方法来设置获取该事件。

该接口中的每个事件方法也都提供了空的默认实现，开发者在实现该接口时，只需要实现自己关心的事件方法即可，不必全部实现。

自定义事件 CustomListener 和 DefaultCustomListener 除了上面的事件以外，我们还为开发者提供了自定义事件的功能。

自定义事件是为开发者准备的 Java 代码跟 HTML 广告模板中 JavaScript 代码交互的接口。

自定义事件的接口定义非常简单：

```
public interface DefaultCustomListener {
    void on(String name, String data);
}
public interface CustomListener {
    void on(String data);
}
```

其中参数 `name` 是用户在 HTML 广告模板中，通过 JavaScript 中传递给 Java 自定义事件的事件名。

其中参数 `data` 是用户在 HTML 广告模板中，通过 JavaScript 中传递给 Java 自定义事件的数据。

`DefaultCustomListener` 跟 `CustomListener` 可以同时设置。

`CustomListener` 的优先级比 `DefaultCustomListener` 高，如果用户设置了与事件名对应的 `CustomListener`，则该事件则不会再发给 `DefaultCustomListener`。所有没有设置 `CustomListener` 的用户自定义事件都会发给 `DefaultCustomListener` 来处理。

不论何种形式的广告，我们都提供了以下 5 个用于操作自定义事件的方法：

- `setDefaultCustomListener`
- `getDefaultCustomListener`
- `setCustomListener`
- `getCustomListener`
- `removeCustomListener`

setDefaultCustomListener 方法 该方法的接口定义为：

```
public void setDefaultCustomListener(DefaultCustomListener listener)
```

其中参数 `listener` 为默认自定义事件处理器。

getDefaultCustomListener 方法 该方法的接口定义为：

```
public DefaultCustomListener getDefaultCustomListener()
```

其返回值为默认自定义事件处理器。

setCustomListener 方法 该方法的接口定义为：

```
public void setCustomListener(String name, CustomListener listener)
```

其中参数 `name` 为自定义事件名， `listener` 为该事件名对应的事件处理器。一个自定义事件只能添加一个事件处理器。

getCustomListener 该方法的接口定义为：

```
public CustomListener getCustomListener(String name)
```

其中参数 `name` 为自定义事件名，返回值为该事件名对应的事件处理器。

removeCustomListener 该方法的接口定义为：

```
public void removeCustomListener(String name)
```

其中参数 `name` 为自定义事件名，该方法执行后，删除与该事件名对应的事件处理器。

自定义事件的具体应用方法，我们在[模板]一章会进行详细说明。

模板

模板基本用法 我们的广告是基于 html 模板来实现的。根据不同形式的广告，我们提供了一组默认的模板，您可以在：https://github.com/adtalos/ads-sdk-templates 上直接看到它们。您也可以帮助我们一起来对他们进行改进。

另外，我们对每一种形式的每个广告位都开放了自定义 html 模板的功能，您可以在[开发者后台]中编辑广告位时看到该选项。

模板变量 模板中可以使用的变量有以下几个：

含义	变量
广告标题	`{{ .Title }}`
广告副标题	`{{ .Subtitle }}`
广告描述	`{{ .Description }}`
按钮文字	`{{ .ButtonText }}`
广告主名	`{{ .AdvertiserName }}`
目标地址	`{{ .TargetURL }}`
图标地址	`{{ .IconURL }}`
视频地址	`{{ .VideoURL }}`
Logo地址	`{{ .LogoURL }}`
图片1	`{{ .Image1 }}`
图片2	`{{ .Image2 }}`
图片3	`{{ .Image3 }}`
HTML片段	`{{ .HTMLSnippet }}`

含义	变量
是否是下载类广告	`{{ .IsDownloadAd }}`
是否包含原生素材	`{{ .IsNativeAd }}`
语言	`{{ .Language }}`

注意，变量是否有值跟广告主返回的广告素材有关，某些变量并不一定总是有值，因此，在编写自定义模板时需要注意判断。

JavaScript 与 Java 交互 我们的 SDK 提供了一个 JavaScript 对象 `adtalos` ，您可以在 HTML 模板中使用它来实现跟 Java 的交互。该对象上目前提供了 1 个方法： `dispatch` 。

`dispatch` 方法是跟[事件]中的自定义事件来结合使用的。 `dispatch` 方法有两个参数：

```
adtalos.dispatch(name, data)
```

其中 `name` 对应的是自定义事件时， `setCustomListener` 、 `getCustomListener` 和 `removeCustomListener` 方法中的 `name` 参数，或 `DefaultCustomListener` 接口中 `on` 方法的 `name` 参数。

而 `data` 对应的是，自定义事件中 `on` 方法的 `data` 参数。

这两个参数都是字符串类型。如果希望通过该方法，传递 JSON 数据给 Java 自定义事件，只需要在 JavaScript 中用 `JSON.stringify` 方法将 JavaScript 中的对象序列化为 JSON 字符串，然后通过 `data` 参数传给 Java 自定义事件，然后在 Java 的自定义事件的 `on` 方法中，用 `new JSONObject(data)` 来还原为 `JSONObject` 即可。传递数组参数的方法也是类似。

另外，需要注意，不要自定义 `close` 事件，因为该事件已经在 SDK 内部被使用，重新定义该事件可能会导致广告无法被关闭。

模板设计注意事项 原生广告的 HTML 模板中，body 的高度不要设置为 100%。如果设置为 100%，在代码中如果又同时设置了 `unit_size` 为 `native_size` 的话，广告会显示不出来。高度值最好是使用 `vw` 为单位进行设置，这样可以在横竖屏转化时，可以保持广告的纵横比。

竞价

从 3.2.0 版本开始，SDK 增加了竞价支持。不管是 `View` （横幅和原生广告）还是 `Controller` （开屏、插屏、激励视频广告）都增加了以下几个方法：

```
public int getPrice()

public boolean sendWinNotice()

public boolean sendWinNotice(int secondPrice)

public boolean sendLossNotice(int lossReason)
```

这些方法都需要在广告载入之后才有效。另外，需要注意，竞价广告的广告位的竞价类型需要设置为竞价，否则这些方法也无效。

在创建竞价广告时，还需要注意，创建横幅和原生广告时，需要直接通过 Java 代码创建横幅、原生广告，并且使用 `load` 方法时，第二个参数需要设置为 false，这样才能在广告加载后，不会立刻曝光。

在广告载入后（ `onLoaded` 事件或 `isLoaded` 方法返回 `true` 时），使用 `getPrice` 方法返回广告的千次曝光出价，单位为分，然后根据跟其他广告主返回广告的出价进行比较，来决定时候曝光。

如果竞价获胜，调用 `sendWinNotice` 方法来发送竞价获胜上报，如果不加参数，直接以出价作为成交价进行上报,即一价成交。如果调用带参数的版本，参数的价格作为成交价，一般来说该价格应该为第二高价，注意该

价格不应该大于 `getPrice` 返回的价格，否则不会上报竞价成功，并且返回 `false` 。如果竞胜上报成功，会返回 `true` ，同时后面再调用 `getPrice` 时，返回的价格也会变成 `sendWinNotice` 上报的成交价。竞价成功上报之后，需要调用 `show` 方法来进行曝光。

如果竞价失败，调用 `sendLossNotice` 方法来发送竞价失败上报，参数在 `com.xy.sdk.AdLossReason` 中有定义：

```
/** 竞价失败原因 */
public final class AdLossReason {
    /** 底价过滤 */
    public static final int BID_PRICE_FILTER = 1001;
    /** 竞价出价低于最高价 */
    public static final int PRICE_LOW_FILTER = 1002;
    /** 素材黑名单过滤 */
    public static final int ADM_BLACKLIST_FILTER = 1003;
    /** 竞品过滤 */
    public static final int COMPETE_FILTER = 1004;
    /** 超时过滤 */
    public static final int TIMEOUT_FILTER = 1005;
    /** 其它过滤 */
    public static final int OTHER_FILTER = 1006;
}
```

调用该方法后，如果上报成功，会返回 `true` ，并且返回的广告会被清空，广告载入状态也会变为未载入状态。如果在广告未载入状态下调用该方法，该方法会返回 `false` ，并且不会上报成功。调用该方法之后，不应该再调用 `show` 方法。

创建的广告对象最好只使用一次，竞价成功上报并展示曝光之后，如果需要再次竞价，需重新创建一个广告对象。竞价失败之后，如果需要再次请求广告，也重新创建一个广告对象。否则可能会无法再次载入广告。

FAQ

SDK 是否支持原生转横幅广告？ 答：支持。

SDK 是否支持原生转开屏广告？ 答：支持。

对于原生广告，在 onLoaded 之后，使用 render 方法，广告没有显示，一般是什么原因？ 答：如果广告的 `View` 没有添加到其它视图中时，只能将尺寸设置为 `Size.NATIVE` 这个尺寸，因为设置为其它尺寸的话，需要根据广告视图 `LayoutParams` 来计算实际尺寸，而当广告视图没有添加进其它视图时， `LayoutParams` 是不存在的，因此无法计算实际尺寸，也就无法正常显示广告了。解决方法很简单，在调用 `render` 之前将广告视图添加到其它视图中，或者将广告尺寸设置为 `Size.NATIVE` 这个尺寸就可以了。

是否可以在开屏广告的底部插入我们媒体自己的 Logo 等内容？ 答：在创建开屏广告位时，通过自定义 html[模板]就可以实现该功能，您甚至可以结合自定义[模板]和自定义[事件]来实现『会员去广告』等功能。

原生广告是否可以加上『不感兴趣』菜单？ 答：原生广告的默认模板目前只包含最基本的内容显示功能，诸如『不感兴趣』这样的菜单，您可以结合自定义[模板]和自定义[事件]来实现。

是否可以自定义广告的关闭按钮的大小，位置？ 答：在[开发者后台]中包含有这样的功能。对于横幅、开屏、插屏形式的广告，默认的样式基本上应该可以满足您的要求。如果您需要更加个性化的关闭按钮显示方式，可以通过自定义[模板]和自定义[事件]来实现，关闭的动作通过在 JavaScript 中调用 `adtalos.dispatch('close', '');` 方法来触发，在 Java 中已经对该自定义事件提供了实现，您不需要自己来实现。

请求失败时会自动重试吗？ 答：会的。自动重试次数默认为 5 次，开发者可以通过 `autoRetry` 方法来设置重试次数。设置为小于等于 0 的值则禁用失败自动重试。

如何禁止自动初始化 SDK，实现手动初始化 SDK? 答：为了方便用户使用，我们的 SDK 默认情况下是自动初始化的，不需要用户手动调用初始化方法。

但是在某些特殊情况下，用户如果不希望自动初始化，而是自己手动控制初始化，比如只针对高版本的 Android 进行初始化，对低版本的 Android 不做初始化。

针对上面这种情况下，我们增加了禁用自动初始化 SDK，实现手动初始化 SDK 的功能。方法如下：

首先禁止自动初始化 SDK，打开应用的 AndroidManifest.xml 文件。

在 `manifest` 标签中加入 `xmlns:tools="http://schemas.android.com/tools"` 这个属性。

然后在 `<application>...</application>` 中，加入以下代码：

```
        <provider
            android:name="com.xy.sdk.InitProvider"
            android:authorities="${applicationId}.initprovider"
            tools:replace="android:enabled"
            android:enabled="false"
            android:exported="false" />
```

::: tip 注意： 这里 `com.xy.sdk` 是默认的包名。您在使用时，请替换为您在 cms 开发者面板中填写的包名。 :::

这样做之后，自动初始化 SDK 的功能就被禁止了。

接下来是实现手动初始化，手动初始化可以最常见的方式是在 `Application` 中实现。例如：

```
package com.xy.sample;

import android.app.Application;

import com.xy.sdk.SDK;

public class SampleApplication extends Application {
    public void onCreate() {
        super.onCreate();
        SDK.init(getApplicationContext());
    }
}
```

上面代码中， `SDK.init(getApplicationContext());` 这一句代码是手动初始化 SDK 的关键。

针对上面这个类，我们只要在 AndroidManifest.xml 文件的 `application` 标签中，增加 `android:name=".SampleApplicati` 这个属性，就可以实现手动初始化 SDK 了。

默认情况下，仍然是自动初始化的。所以，如果没有上面的特殊需求，可以忽略上面的内容。

更新日志

3.2.0 版本

- SDK 区域设置默认值改回国内。
- SDK 增加了竞价支持。

3.1.0 版本

- SDK 增加了 onBeforeRequest 事件。

3.0.1 版本

- SDK 移除了获取电池信息，传感器信息的功能。

3.0.0 版本

- SDK 初始化移除了自动获取 IMEI，IMSI，OAID，Android ID，GAID，MAC 等设备号的功能，移除了自动获取地理位置信息，电池信息，传感器信息的功能。
- SDK 增加了手动设置 IMEI、OAID、AndroidID、GAID 的 API。
- SDK 增加了自动获取 IMEI、OAID、AndroidID、GAID 的开关 API。
- SDK 增加了是否使用地理位置信息，电池信息，传感器信息的开关 API。
- 将最低支持的 Android API 版本从 19 改为 21。
- 升级了 Android SDK 的相关依赖。
- 一些其他优化。

2.7.5 版本

- 完善隐私设置。
- 修复华为高版本手机可能的闪退或卡死问题。

2.7.4 版本

- 修复隐私配置链式调用。

2.7.3 版本

- 增加区域设置。

2.7.2 版本

- 修复自 2.6.0 版本之后，以非默认包名打包后无法正常使用的问题。

2.7.1 版本

- 修复某些手机型号获取OAID出错问题。
- 支持老版的OPPO手机获取OAID。

2.7.0 版本

- SDK 工具链更新
- 内置 OAID 获取功能（无需再单独设置）
- 适配海外需求（GAID 支持，英文模板支持）
- 增加用户信息配置
- 增加隐私配置
- 性能优化

2.6.0 版本

- 增加 BootMark 和 UpdateMark 上报字段支持。

2.5.0 版本

- 修正某些雨滴屏，刘海屏手机上，开屏广告无法真全屏显示的问题。

2.4.0 版本

- 增加 Deeplink 的上报支持。

2.3.0 版本

- 升级了 2G，3G，4G 网络的判断。
- 增加 App Links 支持。
- 升级到 AndroidX。

2.2.3 版本

- 优化了设备信息获取。
- 修正了一个空指针异常。

2.2.2 版本

- 增加一些新的日志字段，并修正 oaid 日志字段。

2.2.1 版本

- 修正与 Flutter 不兼容的问题。

2.2.0 版本

- 开屏，插屏，激励视频中增加 load 方法。方便开发者将加载和显示分离。

2.1.0 版本

- 增加了 onImpressionFailed 事件。
- 修正了开屏，插屏，激励视频中 autoRetry 设置无效的问题。

2.0.0 版本

- 对包名，类名和部分方法名进行了改名，防止某些市场提示 App 内含有广告。
- 开发者可以自己设置 sdk 开发包的包名，定制专属的 sdk 开发包。
- 不再强制开启 Java 8 支持。

1.8.0 版本

- 修正某些特殊的 HTML URL 插屏广告，竖屏时关闭按钮过大的问题。
- 修正广告落地页方向设置无效的问题。
- 修正某些 HTML 横幅广告无法显示的问题。
- 修正某些原生广告在屏幕旋转时无法等比缩放的问题。
- 修正某些原生广告在屏幕旋转时可能会显示白屏的问题。
- 先上报再触发事件，避免用户事件中发生异常影响上报。
- 取消 1.7.0 版本加入的自动申请权限功能，提供手动申请权限方法。
- 开发包体积优化。

1.7.1 版本

- 改善应用内落地页导航按钮工具栏的显示效果。

1.7.0 版本

- SDK 初始化时，自动请求各种需要的权限，开发者不再需要自己编写请求权限的代码。

1.6.0 版本

- 增加了 SDK.setOAID 方法，开发者可以用来自己设置 OAID。
- 增加了 AdActivity.setDisplayActionBarEnabled 方法，开发者可以用来设置应用内落地页是否显示导航按钮工具栏。

1.5.1 版本

- 视频控制器增加一个 isEnded 方法。

1.5.0 版本

- 优化插屏广告的关闭按钮和广告字样位置。
- 应用内打开的广告连接如果重定向到外部程序时，应用内浏览器自动关闭。
- 内置浏览器设置为默认全屏主题。
- 增加开屏视频、插屏视频的视频事件支持。
- 统一插屏、开屏、激励视频广告的构造方法。
- 取消 HTML 模板中的 adtalos.getWidth 和 adtalos.getHeight 方法（目前的默认模板已经不再依赖这两个方法了）。
- 增加自动重试次数设置。
- 视频控制器增加一个 isPlaying 方法。
- 修正插屏广告的横竖屏问题。
- 其它一些细节上的优化。

1.4.5 版本

- 修正上一个版本引起的 WebView 提前释放的问题。

1.4.4 版本

- 修正了通过 applicationContext 创建的 AdView，在 Android P 上点击崩溃的问题。

1.4.3 版本

- 优化广告请求尺寸计算方式。

1.4.2 版本

- 修正 Android 8.0 导致的开屏/插屏/激励视频广告闪退的问题。

1.4.1 版本

- 修正某几款特殊机型（小米 MIX2 和 MIX2S）下，没有获取到文件读写权限时，下载类广告点击会引起闪退的问题。

1.4.0 版本

- 提高开屏广告的曝光率。
- 为开屏/插屏/激励视频广告增加了 show(timeout) 方法。

1.3.4 版本

- 代码优化。

1.3.3 版本

- 修正非 AdSize.NATIVE 尺寸的原生广告，通过代码创建对象无法显示的问题。

1.3.2 版本

- 取消创建 AdView 时，要求 Context 必须是 Activity 的限制。

1.3.1 版本

- 针对开发者在 app 中可能调用了 pauseTimers 方法而未在合适的时机调用对应的 resumeTimers 方法导致全局 WebView 失效的问题而做出的兼容性修改，以保证在广告展示时能够自动调用 resumeTimers 方法来恢复 WebView 的功能。

1.3.0 版本

- 增加了手动初始化 SDK 的功能。

1.2.5 版本

- 修正了网络请求不兼容 Android 4.4 以下版本的问题。

1.2.4 版本

- 修正了 WebView 没有彻底释放的问题。

1.2.3 版本

- 修正了 1.2.x 版本中，一个无法正常请求广告的严重错误。

1.2.2 版本

- 修正 AdView 提前被释放时，可能导致 crash 的问题。

1.2.1 版本

- 修正 onAdRendered 个别时候不会被触发的问题。

1.2.0 版本

- 横幅和原生广告增加了动画开关方法。
- 原生广告默认关闭动画效果。

1.1.0 版本

- 横幅和原生广告新增了 render 方法。
- 横幅和原生广告新增了 impressionReport 方法。
- 新增了 onAdRendered 事件。

1.0.5 版本

- 完善崩溃信息捕获。

1.0.4 版本

- 修正一个空指针异常。

1.0.3 版本

- 所有广告类型都增加了底部右下角的广告字样。

1.0.2 版本

- 修复极少数老机型上激励视频广告暂停后无法继续播放的问题。

1.0.0 版本

- 正式版本发布。