蚂蚁科技

扫一扫
使用指南

文档版本：20231226

蚂蚁集团
ANT GROUP

法律声明

商标声明

蚂蚁集团 ANT GROUP及其他蚂蚁集团相关的商标均为蚂蚁集团所有。本文档涉及的第三方的注册商标，依法由权利人所有。

免责声明

由于产品版本升级、调整或其他原因，本文档内容有可能变更。蚂蚁集团保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在蚂蚁集团授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过蚂蚁集团授权渠道下载、获取最新版的用户文档。如因文档使用不当造成的直接或间接损失，本公司不承担任何责任。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	**单击设置> 网络> 设置网络类型。**
粗体	表示按键、菜单、页面名称等UI元素。	**在结果确认页面，单击确定。**
`Courier字体`	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.扫一扫

1.1. 扫一扫简介

扫一扫（Scan）是 mPaaS 提供的扫码组件，源于支付宝的扫码能力。该组件秉承了支付宝精准、快速的扫码能力，能够迅速识别出条形码并准确地获得条码中的信息。

组件功能

扫一扫组件支持扫描二维条形码（二维码）和一维条形码（条码）。

二维条形码（二维码）

- Gen0（普通二维码）：

- Gen1（visualead 自定义码）：

一维条形码（条码）

- EAN8：

- EAN13：

- EAN14：

- EAN18：

- EAN128：

- ISBN：

- ISSN：

- Code39：

- Code128：

- UPC-A：

- UPC-E：

- ITF-14：

产品优势

mPaaS 的扫一扫功能，在同等条件下，和业界领先的同类产品相比，在扫码的识别速度、识别率等能力上均占有优势。

识别速度快

在同等距离、同等光源的情况下，mPaaS 扫一扫对二维码/条形码的识别速度快于同类产品。

识别能力强

依赖于特有的模糊处理和数据评估矫正，同类产品的相册调用其扫码组件 API 无法识别出的图片，mPaaS 扫一扫也能够识别出来。

- 这张是同类产品的摄像头可以识别，但是其相册调用扫码组件 API 无法识别的图片。

- 以下是同类产品完全不能识别的二维码。

1.2. 接入 Android

1.2.1. 快速开始

本文介绍的是在 Android 中接入扫一扫 SDK 的操作步骤。

> **说明**
>
> 自 2020 年 6 月 28 日起，mPaaS 停止维护 10.1.32 基线。请升级到 10.1.60、10.1.68 或 10.2.3 基线。扫一扫支持 **原生 AAR** 和 **组件化（Portal&Bundle）**两种接入方式。文本将介绍在 10.2.3、10.1.68、10.1.60 基线下如何使用扫码功能。自 mPaaS 10.1.68.33 版本基线起，扫一扫支持全屏模式下的多码识别。自 mPaaS 10.2.3 版本基线起，扫一扫新增 AI 识别小码并自动放大的功能。

前置条件

- 若采用原生 AAR 方式接入，需先完成 将 mPaaS 添加到您的项目中 的前提条件和后续相关步骤。
- 若采用组件化方式接入，需先完成 组件化接入流程 。

添加 SDK

10.2.3

如需使用 AI 识别小码并自动放大功能，请安装 **扫一扫 AI** 组件。

原生 AAR 方式

参考 AAR 组件管理，通过 **组件管理（AAR）** 在工程中安装 **扫一扫/扫一扫 AI** 组件。

组件化方式

在 Portal 和 Bundle 工程中通过 **组件管理** 安装 **扫一扫/扫一扫 AI** 组件。更多信息，参考 管理组件依赖。

10.1.68/10.1.60

原生 AAR 方式

参考 AAR 组件管理，通过 **组件管理（AAR）** 在工程中安装 **扫码** 组件。

组件化方式

在 Portal 和 Bundle 工程中通过 **组件管理** 安装 **扫码** 组件。更多信息，参考 管理组件依赖。

使用扫一扫功能

10.2.3/10.1.68

使用全屏扫码功能

```
ScanRequest scanRequest = new ScanRequest();
MPScan.startMPaasScanFullScreenActivity(this, scanRequest, new MPScanCallbackAdapter()
{
    @Override
    public boolean onScanFinish(final Context context, MPScanResult mpScanResult, final
MPScanStarter mpScanStarter) {
        Toast.makeText(getApplicationContext(),
                mpScanResult != null ? mpScanResult.getText() : "没有识别到码", Toast.LENG
TH_SHORT).show();
        ((Activity) context).finish();
        // 返回 true 表示该回调已消费，不需要再次回调
        return true;
    }
});
```

使用窗口扫码功能

在 mPaaS 10.1.68 基线上使用窗口扫码功能（旧标准 UI），若扫码失败直接返回扫码界面，若扫码成功将获取二维码的 URL 信息。

```
ScanRequest scanRequest = new ScanRequest();
scanRequest.setScanType(ScanRequest.ScanType.QRCODE);
MPScan.startMPaasScanActivity(this, scanRequest, new ScanCallback() {
    @Override
    public void onScanResult(final boolean isProcessed, final Intent result) {
        if (!isProcessed) {
            // 扫码界面点击物理返回键或左上角返回键
            return;
        }
        // 注意：本回调是在子线程中执行
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                if (result == null || result.getData() == null) {
                    // 扫码失败
                    return;
                }
                // 扫码成功
                String url = result.getData().toString();
            }
        });
    }
});
```

10.1.60

在 10.1.60 基线上使用扫码功能，若扫码失败直接返回扫码界面，若扫码成功将获取二维码的 URL 信息。

```
ScanService service = LauncherApplicationAgent
    .getInstance().getMicroApplicationContext()
    .findServiceByInterface(ScanService.class.getName());

ScanRequest scanRequest = new ScanRequest();
scanRequest.setScanType(ScanRequest.ScanType.QRCODE);

service.scan(this, scanRequest, new ScanCallback() {
    @Override
    public void onScanResult(boolean isProcessed, final Intent result) {
        if (!isProcessed) {
            // 扫码界面点击物理返回键或左上角返回键
            return;
        }
        // 注意：本回调是在子线程中执行
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                if (result == null || result.getData() == null) {
                    // 扫码失败
                    return;
                }
                // 扫码成功
                String url = result.getData().toString();
            }
        });
    }
});
```

1.2.2. 进阶指南

窗口扫码表示在旧的标准 UI 下使用扫码功能。若需使用支持多码识别的全屏扫码功能，请将 mPaaS 基线版本升级至 10.1.68.33 及以上。

下图是扫一扫支持的三种 UI 扫码样式。

标准 UI 下使用扫一扫

全屏扫码

> **说明**
>
> 多码识别功能只支持在标准 UI 下使用。

如需连续扫码，即扫码识别成功后不退出继续识别，可根据如下代码来实现。

```
ScanRequest scanRequest = new ScanRequest();
        MPScan.startMPaasScanFullScreenActivity(this, scanRequest, new
MPScanCallbackAdapter() {
            @Override
            public boolean onScanFinish(Context context, MPScanResult mpScanResult,
final MPScanStarter mpScanStarter) {
                new android.app.AlertDialog.Builder(context)
                        .setMessage(mpScanResult != null ? mpScanResult.getText() : "没有
识别到码")
                        .setPositiveButton(R.string.confirm, new
DialogInterface.OnClickListener() {
                            @Override
                            public void onClick(DialogInterface dialog, int which) {
                                mpScanStarter.restart();
                            }
                        })
                        .create()
                        .show();
                // 返回 false 表示该回调未消费，下次识别继续回调
                return false;
            }
        });
```

重写 `MPScanCallbackAdapter` 的其他方法来监听其他事件：

```
MPScan.startMPaasScanFullScreenActivity(this, scanRequest, new MPScanCallbackAdapter()
{
    @Override
    public boolean onScanFinish(final Context context, MPScanResult mpScanResult, final
MPScanStarter mpScanStarter) {
        return true;
    }

    @Override
    public boolean onScanError(Context context, MPScanError error) {
        // 识别错误
        return super.onScanError(context, error);
    }

    @Override
    public boolean onScanCancel(Context context) {
        // 识别取消
        return super.onScanCancel(context);
    }
});
```

在启动全屏扫码功能前，可根据如下代码设置启动参数。

```
ScanRequest scanRequest = new ScanRequest();

// 设置提示文字
scanRequest.setViewText("提示文字");

// 设置打开手电筒提示文字
scanRequest.setOpenTorchText("打开手电筒");

// 设置关闭手电筒提示文字
scanRequest.setCloseTorchText("关闭手电筒");

// 设置扫码识别类型
// 该设置仅对直接扫码生效，对识别相册图片无效
scanRequest.setRecognizeType(
    ScanRequest.RecognizeType.QR_CODE,     // 二维码
    ScanRequest.RecognizeType.BAR_CODE,    // 条形码
    ScanRequest.RecognizeType.DM_CODE,     // DM 码
    ScanRequest.RecognizeType.PDF417_Code // PDF417 码
); // 不设置，则默认识别前三种

// 设置隐藏相册按钮
scanRequest.setNotSupportAlbum(true);

// 设置多码标记图片
scanRequest.setMultiMaMarker(R.drawable.green_arrow);

// 设置多码提示文字
scanRequest.setMultiMaTipText("点击绿色箭头选择码");

// 设置选中单个码后的圆点颜色
scanRequest.setMaTargetColor("#32CD32");

// 开启 AI 识别小码并自动放大，仅 10.2.3 及以上基线支持，需接入扫一扫 AI 组件
scanRequest.setEnableAI(true);

// 设置延时提示文案，仅 10.2.3 及以上基线支持
scanRequest.setDelayTipText("延时x秒弹出toast");

// 设置延时提示时间，单位毫秒，仅 10.2.3 及以上基线支持
scanRequest.setDelayTipTime(5000);
```

窗口扫码

使用在窗口扫码功能时，可根据如下代码设置启动参数。

```
ScanRequest scanRequest = new ScanRequest();

// 设置扫码页 UI 风格
scanRequest.setScanType(ScanRequest.ScanType.QRCODE);  // 二维码风格
scanRequest.setScanType(ScanRequest.ScanType.BARCODE); // 条形码风格，默认

// 设置扫码界面 title
scanRequest.setTitleText("标准扫码");

// 设置扫码窗口下提示文字
scanRequest.setViewText("提示文字");

// 设置打开手电筒提示文字，仅 10.1.60 及以上基线支持
scanRequest.setOpenTorchText("打开手电筒");

// 设置关闭手电筒提示文字，仅 10.1.60 及以上基线支持
scanRequest.setCloseTorchText("关闭手电筒");

// 设置扫码识别类型，仅 10.1.60.6+ 和 10.1.68.2+ 基线支持
// 该设置仅对直接扫码生效，对识别相册图片无效
scanRequest.setRecognizeType(
    ScanRequest.RecognizeType.QR_CODE,     // 二维码
    ScanRequest.RecognizeType.BAR_CODE,    // 条形码
    ScanRequest.RecognizeType.DM_CODE,     // DM 码
    ScanRequest.RecognizeType.PDF417_Code // PDF417 码
); // 不设置，则默认识别前三种

// 设置透明状态栏（在 Android 4.4+ 系统上生效），仅 10.1.68.15+ 基线支持
scanRequest.setTranslucentStatusBar(true);

// 设置隐藏相册按钮，仅 10.1.68.22+ 基线支持
scanRequest.setNotSupportAlbum(true);
```

自定义 UI 下使用扫一扫

请参考 代码示例。

自定义 UI 升级适配

自 10.1.68.5 和 10.1.60.11 起，扫一扫 SDK 新增了类 MPScanner 以及相关接口，用来替代此前自定义扫码需要使用的 BQCScanCallback、MaScanCallback 等原始接口。相比原始接口，MPScanner 提供了完整的封装性、简洁易懂的 API，以及更多新特性的支持（例如环境亮度不足的回调），强烈推荐您使用 MPScanner 来开发自定义扫码页。如果您仍然在使用 BQCScanCallback、MaScanCallback 等原始接口，当您从低版本升级时可能需要适配以下变更：

- 10.1.68.22 版本：`MaScanCallback` 类、`BQCScanCallback` 类、`IOnMaSDKDecodeInfo` 类新增部分接口，您只需空实现这些接口即可，其中 `MaScanCallback.onMaCodeInterceptor` 方法返回 false 。
- 10.1.60.6 版本：`BQCScanCallback` 类新增部分接口，您只需空实现这些接口即可。
- 10.1.60 版本：`BQCScanCallback` 类新增部分接口，您只需空实现这些接口即可。
- 10.1.20 版本：`MaScanCallback` 类接口变更如下：`void onResultMa(MaScanResult maScanResult)` 变更为 `void onResultMa(MultiMaScanResult multiMaScanResult)` 您可以按照以下方式获取 `MaScanResult` ：

```
MaScanResult maScanResult = multiMaScanResult.maScanResults[0];
```

自定义 UI API 说明

MPScanner

自定义 UI 相关的设置内容如下：

```
/**
 * 设置显示相机内容的 View
 * 推荐在 {@link MPScanListener} 的 onConfiguration 方法中调用
 *
 * @param textureView 自定义扫码页中的 TextureView
 */
public void setDisplayView(TextureView textureView);

/**
 * 设置扫描识别的区域
 *
 * @param rect 识别的区域
 */
public void setScanRegion(Rect rect);

/**
 * 设置扫描监听器
 */
public void setMPScanListener(MPScanListener mpScanListener);

/**
 * 设置识别图像灰度值监听器
 */
public void setMPImageGrayListener(MPImageGrayListener mpImageGrayListener);

/**
 * 获取 Camera 对象
 *
 * @return Camera 对象
 */
public Camera getCamera();

/**
 * 设置识别的码类型
 * 仅对直接扫码生效，对从 bitmap 中识别码无效
 *
 *
 * @param recognizeTypes BAR_CODE 条形码；
 *                       QR_CODE 二维码；
 *                       DM_CODE DM 码；
 *                       PDF417_CODE PDF417 码；
 *                       不设置则默认识别前三种
 */
public void setRecognizeType(MPRecognizeType... recognizeTypes);
```

自定义 UI 相关的扫描内容如下：

```
/**
 * 打开相机并开始扫描
 *
 * 首次进入页面时或相机关闭状态下调用
 */
public void openCameraAndStartScan();

/**
 * 关闭相机并停止扫描
 */
public void closeCameraAndStopScan();

/**
 * 开始扫描
 *
 * 不会更改相机状态，需在相机打开的状态下调用才能生效
 */
public void startScan();

/**
 * 停止扫描
 *
 * 不会更改相机状态
 */
public void stopScan();

/**
 * 从 bitmap 中识别码
 *
 * @param bitmap 需要识别的 bitmap
 * @return 识别结果
 */
public MPScanResult scanFromBitmap(Bitmap bitmap);
```

其他：

```
/**
 * 打开或关闭手电筒
 *
 * @return 调用方法后，手电筒是否打开
 */
public boolean switchTorch();

/**
 * 打开手电筒
 */
public void openTorch();

/**
 * 关闭手电筒
 */
public void closeTorch();

/**
 * 播放默认的“哔哔”声
 */
public void beep();

/**
 * 释放资源
 *
 * 请在 onDestroy 中调用
 */
public void release();
```

MPScanListener

```
/**
 * 扫描参数配置完成
 */
void onConfiguration();

/**
 * 扫描识别开始
 */
void onStart();

/**
 * 识别成功
 *
 * @param result 识别结果
 */
void onSuccess(MPScanResult result);

/**
 * 识别错误
 *
 * @param error 错误
 */
void onError(MPScanError error);
```

MPImageGrayListener

```
/**
 * 获取识别图像的平均灰度值
 *
 * 正常范围大约在 50-140 之间，
 * 当灰度值低于或高于正常范围时，通常意味着环境亮度过低或过高，可以提示用户打开或关闭手电筒
 * 注意：该方法在识别过程中会不断被调用
 *
 * @param gray 图像的平均灰度值
 */
void onGetImageGray(int gray);
```

MPScanResult

```
/**
 * 识别结果字符串
 */
private String text;

/**
 * 识别的码类型
 */
private MPRecognizeType mpRecognizeType;
```

1.2.3. 使用教程

1.2.3.1. 总览

扫一扫支持原生 AAR 接入、mPaaS Inside 接入和组件化接入三种接入方式。如果想要像使用其他 SDK 一样简单地接入并使用 mPaaS，推荐使用原生 AAR 接入方式。

原生 AAR 接入方式是指采用原生 Android AAR 打包方案，更贴近 Android 开发者的技术栈。开发者无需了解 mPaaS 相关的打包知识，通过 mPaaS Android Studio 插件即可将 mPaaS 集成到开发者的项目中。该方式降低了开发者的接入成本，能够让开发者更轻松地使用 mPaaS。

为了方便您快速熟悉并掌握原生 AAR 接入方式，本教程以原生 AAR 接入方式为例，指导您快速接入扫一扫组件并使用扫码功能。

本教程一共包含以下五个部分：

1. 在 Android Studio 创建应用
2. 在 mPaaS 控制台创建应用
3. 原生 AAR 方式接入工程
4. 标准 UI 下使用扫码功能
5. 自定义 UI 下使用扫码功能

您将学会

- 如何创建一个通过单击按钮弹出 Toast 的安卓应用。
- 如何接入原生 AAR。
- 如何在标准 UI 下使用扫码功能。
- 如何在自定义 UI 下使用扫码功能。

您将需要

1. 配置开发环境（本教程以 Windows 下的开发环境为例进行说明）。
2. 网络浏览器（建议您使用 Chrome 浏览器）。
3. 一部安卓手机（系统版本为安卓 4.3 或更新的版本）及配套的数据线。您也可以选择使用模拟器进行调试，本教程以模拟器为例。

1.2.3.2. 在 Android Studio 创建应用

在本节您将创建一个通过点击按钮弹出 Toast 的应用，并获得 APK 格式的安装包。

该过程主要分为四个步骤：

1. 创建工程
2. 编写代码
3. 创建签名文件并给工程添加签名
4. 在手机上安装应用

如果您已经有了一个原生的 Android 开发工程并完成了签名，那么您可以跳过本教程，直接 在 mPaaS 控制台创建应用。

创建工程

1. 打开 Android Studio，点击 **File** > **New** > **New Project**。
2. 在弹出的新建工程窗口中，选择 **Empty Activity**，点击 **Next**。
3. 输入 **Name**、**Package name**（可以使用默认值）、**Save location**。在此处 **Name** 以 **Scan Application** 为例。选择 **Minimum SDK** 为 **API 18: Android 4.3 (Jelly Bean)**。

> **说明**
>
> API 18: Android 4.3（Jelly Bean）是 mPaaS 支持的最低版本，您在实际生产中可以根据需要进行选择。

4. 点击 **Finish**，即可完成 **创建工程**。

编写代码

1. 打开 `activity_main.xml` 文件，参照如下代码添加按钮。

```
<Button
    android:id="@+id/button"
    android:layout_width="101dp"
    android:layout_height="50dp"
    android:layout_marginStart="142dp"
    android:layout_marginTop="153dp"
    android:layout_marginBottom="151dp"
    android:text="Button"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

2. 打开 `MainActivity` 类，添加按钮的点击事件。

```
    findViewById(R.id.button).setOnClickListener(new View.OnClickListener(){
        @Override
        public void onClick(View v) {
            Toast.makeText(MainActivity.this, "Hello mPaaS!",
Toast.LENGTH_SHORT).show();
        }
    });
```

3. 编译成功后，您已完成 **编写代码**。

创建签名文件并给工程添加签名

1. 在 Android Studio 中点击 **Build** > **Generate Signed Bundle / APK**。
2. 在弹出的窗口中选择 **APK**，点击 **Next**。
3. 选择 **Create new**。
4. 填入相应信息后，点击 **OK**，即可完成创建签名。您可在指定的 **Key store path** 中获得生成的签名文件。
5. 内容自动填充后，点击 **Next** 开始对工程添加签名。
6. 根据需要选择 **Build Variants**，Build Variants 信息需要牢记，因为在使用加密文件的时候需要选择和生成时一致的类型。随后勾选 **V1（Jar Signature）**加密版本。V1（Jar Signature）为必选项，V2（Full APK Signature）可按需选择。
7. 点击 **Finish**。打包完成后在工程文件夹下的 `debug` 文件夹（`~\MyHApplication\app\debug`）中，即可获得该应用签名后的 APK 安装包。在本教程中，安装包名为 `app-debug.apk`。

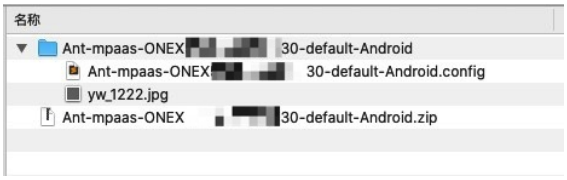
在手机上安装应用

1. 连接手机到电脑，并开启手机的 USB 调试模式。
2. 运行工程。
3. 点击 **BUTTON**，弹出Toast，即表示应用安装成功且实现了预期功能。至此，您已完成 **在手机上安装应用**。

1.2.3.3. 在 mPaaS 控制台创建应用

本文介绍的是在 mPaaS 控制台创建应用的操作步骤。

1. 打开网络浏览器，登录 mPaaS 控制台 。
2. 创建 mPaaS 应用，输入项目名并单击 **创建**。
3. 在 **应用列表** 页，单击 **Scan Application**（应用名称）打开 **应用详情** 页，单击 **Android 代码配置** ，打开 **将 mPaaS 接入到我的应用** 页面。
4. 在 **将 mPaaS 接入到我的应用** 页，单击 **下载配置文件**，打开 **代码配置** 页。输入 **Package Name**（应用包名）（此处以 **com.mpaas.demo** 为例），上传编译并添加签名后的 APK 安装包。关于快速生成签名后的 APK 相关信息，请参见 生成控制台用签名 APK。
5. 在 **代码配置** 页，填写完成后，单击 **下载配置**，即可获取 mPaaS 的配置文件。配置文件是一个压缩包文件。该压缩包包含一个 `.config` 文件以及一个 `yw_1222.jpg` 的加密图片。



1.2.3.4. 原生 AAR 方式接入工程

本文介绍如何将工程通过原生 AAR 的方式接入 mPaaS。

操作步骤

1. 在 Android Studio 中选择 **mPaaS** > **原生 AAR 接入**。
2. 在界面右侧弹出的窗口中，选择 **导入 App 配置** 下方的 **开始导入**。
3. 在弹出的 **导入 mPaaS 配置文件** 窗口中，选择 **我已经从控制台上下载配置文件，准备导入到工程** 。
4. 选择在控制台创建 mPaaS 应用后下载的 配置文件，点击 **Finish**。
5. 随后会提示配置文件导入成功。
6. 点击界面右侧 **接入/升级基线** 下方的 **开始配置**。
7. 在弹出的 **选择 mPaaS 基线版本** 窗口中，选择 10.1.68 基线，点击 **OK**，即可接入 mPaaS SDK。

> **说明**
>
> 再次点击 **开始配置** 可升级基线。

8. 点击界面右侧 **配置/更新组件** 下方的 **开始配置**。
9. 在弹出的组件列表中，勾选 **扫码**，并点击 **OK**，即可将扫码组件添加至工程。至此您已完成通过原生 AAR 方式接入工程到 mPaaS。

后续步骤

- 标准 UI 下使用扫码功能：将标准 UI 扫码的能力添加到工程中，并设置扫码界面的 Title。
- 自定义 UI 下使用扫码功能：将自定义 UI 扫码的能力添加到工程中。

> **说明**
>
> 您可以在您的项目中分别使用标准 UI 下的扫码能力和自定义 UI 下的扫码能力，也可以选择其中一个。点此下载 后续步骤中使用的代码示例。

1.2.3.5. 标准 UI 下使用扫码功能

本文将引导您将标准 UI 扫码的能力添加到工程中，并介绍如何设置扫码界面的 Title。

标准 UI 下使用扫一扫

1. 打开 Android Studio 在 `activity_main.xml` 文件中，重新设置 Button 样式并修改 Button 的 id 为 `standard_ui_btn` 。

```
 <?xml version="1.0" encoding="utf-8"?>
 <androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
     xmlns:app="http://schemas.android.com/apk/res-auto"
     xmlns:tools="http://schemas.android.com/tools"
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     tools:context=".MainActivity">

     <Button
         android:id="@+id/standard_ui_btn"
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:layout_marginTop="48dp"
         android:background="#108EE9"
         android:gravity="center"
         android:text="标准 UI 下使用扫一扫"
         android:textColor="#ffffff"
         app:layout_constraintEnd_toEndOf="parent"
         app:layout_constraintHorizontal_bias="0.498"
         app:layout_constraintStart_toStartOf="parent"
         app:layout_constraintTop_toTopOf="parent" />

 </androidx.constraintlayout.widget.ConstraintLayout>
```

2. 在 `MainActivity` 类重写单击按钮事件，通过单击按钮实现扫码功能。代码如下所示：

```
    private ScanRequest scanRequest = new ScanRequest();
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        findViewById(R.id.standard_ui_btn).setOnClickListener(new
View.OnClickListener(){
            @Override
            public void onClick(View v) {
                MPScan.startMPaasScanActivity(MainActivity.this, scanRequest, new Sca
nCallback() {
                    @Override
                    public void onScanResult(final boolean isProcessed, final Intent
result) {
                        if (!isProcessed) {
                            // 扫码界面单击物理返回键或左上角返回键
                            return;
                        }
                        // 注意：本回调是在子线程中执行
                        runOnUiThread(new Runnable() {
                            @Override
                            public void run() {
                                if (result == null || result.getData() == null) {
                                    // 扫码失败
                                    Toast.makeText(MainActivity.this, "扫码失败，请重试！
", Toast.LENGTH_SHORT).show();
                                    return;
                                }
                                // 扫码成功
                                new AlertDialog.Builder(MainActivity.this)
                                        .setMessage(result.getData().toString())
                                        .setPositiveButton(R.string.confirm, null)
                                        .create()
                                        .show();
                            }
                        });
                    }
                });

            }
        });
    }
```

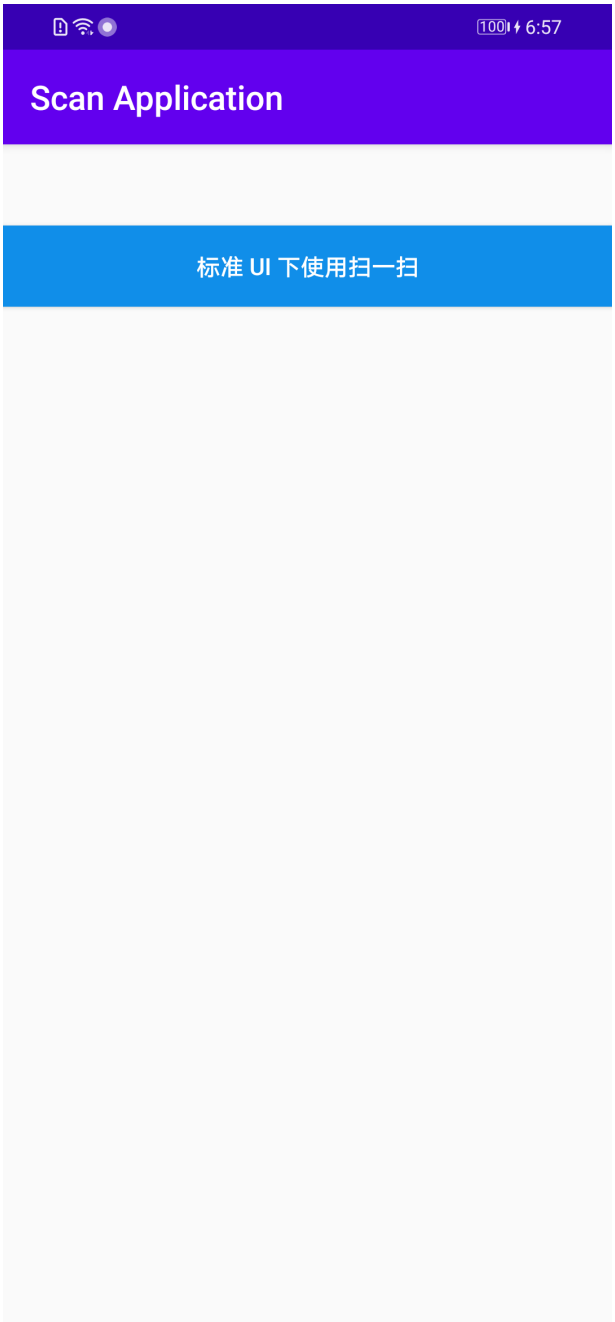
3. 在工程的 `AndroidManifest.xml` 中添加读写权限和网络访问权限。

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

4. 在工程主 Module 下的 `build.gradle(:app)` 中添加以下配置：

```
apply plugin: 'com.android.application'
apply plugin: 'com.alipay.apollo.baseline.config'

android {
    compileSdkVersion 29
    buildToolsVersion "29.0.3"

    defaultConfig {
        applicationId "com.example.h5application"
        minSdkVersion 18
        targetSdkVersion 26
        ndk{
            abiFilters 'armeabi'
        }
        multiDexEnabled true
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation platform("com.mpaas.android:$mpaas_artifact:$mpaas_baseline")
    implementation fileTree(dir: "libs", include: ["*.jar"])
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    implementation 'com.mpaas.android:nebula'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'

}
```

5. 编译并运行工程后在手机上安装应用。打开应用后界面如下：


6:57
Scan Application
标准 UI 下使用扫一扫

6. 点击 **标准 UI 下使用扫一扫** 即可使用标准 UI 下的扫码功能。

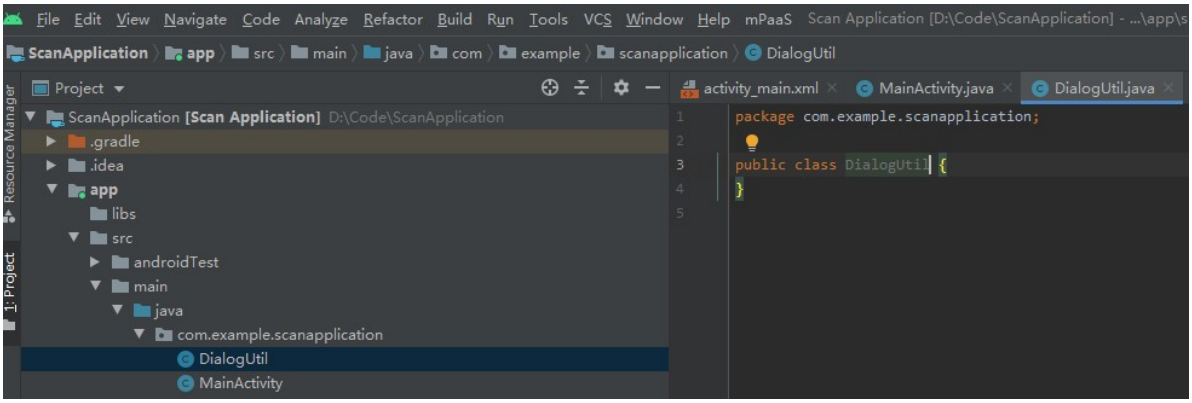
7. 扫描如下二维码，界面会弹出该二维码的信息。

设置扫码界面 Title

1. 在 `activity_main.xml` 文件中，添加 Button，并设置 Button 的 id 为 `btn_title` 。

```
<Button
    android:id="@+id/btn_title"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="128dp"
    android:background="#108EE9"
    android:gravity="center"
    android:text="标准 UI 下设置扫码界面 Title"
    android:textColor="#ffffff"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

2. 在 `com.example.scanapplication` 包中创建 `DialogUtil` 类。



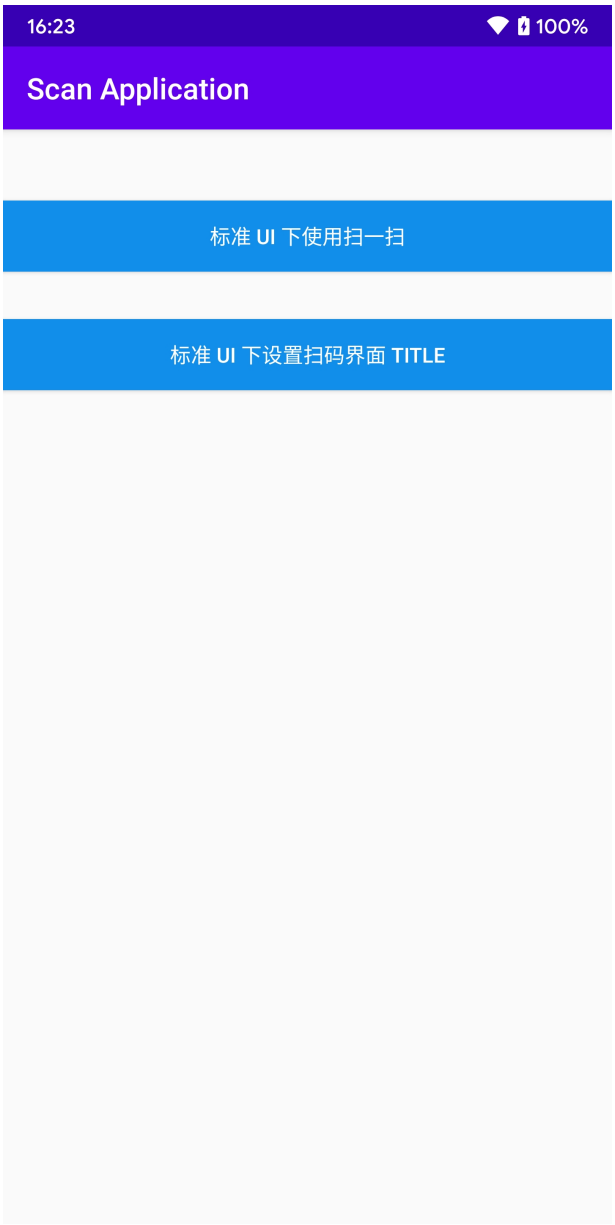
3. 在 `DialogUtil` 类中设置扫码界面样式。

```
public interface PromptCallback {
    void onConfirm(String msg);
}

public static void prompt(Activity activity, final PromptCallback callback) {
    final EditText edit = new EditText(activity);
    new AlertDialog.Builder(activity)
            .setTitle("输入文字")
            .setView(edit)
            .setPositiveButton("确定"
                    , new DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                            if (callback != null) {
                                String text = edit.getText().toString().trim();
                                callback.onConfirm(text);
                            }
                            dialog.dismiss();
                        }
                    })
            .setNegativeButton("取消", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    dialog.dismiss();
                }
            })
            .create()
            .show();
}
```

4. 在 `MainActivity` 类中编写代码。通过点击 `btn_title` 按钮实现设置扫码界面 Title 的功能。代码如下所示：

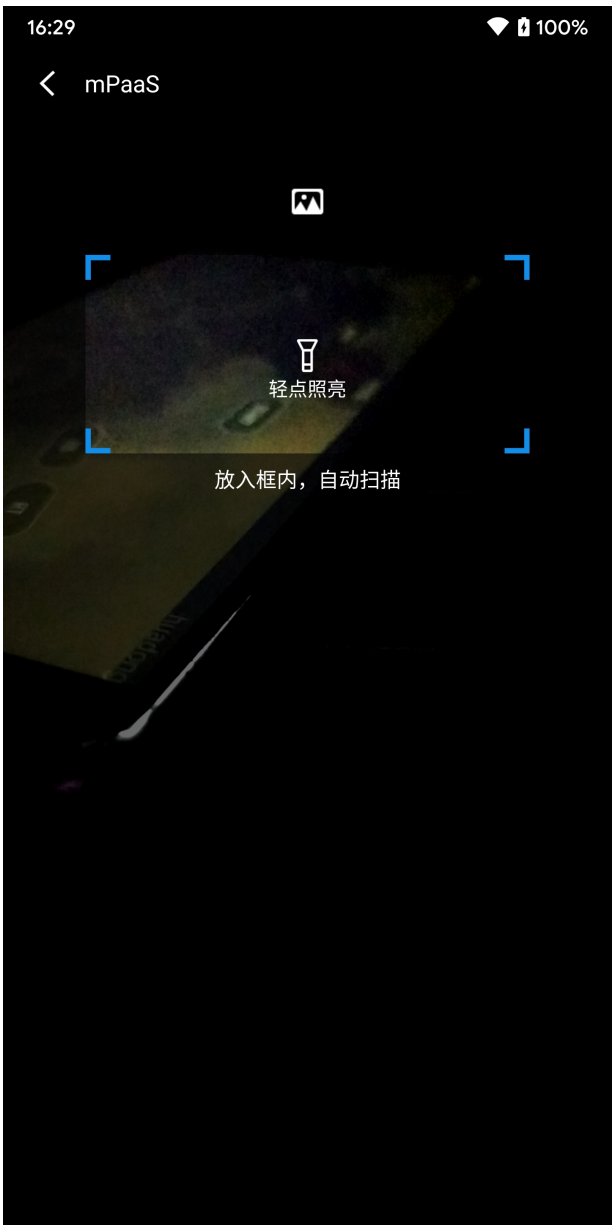
```
    findViewById(R.id.btn_title).setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            DialogUtil.prompt(MainActivity.this, new DialogUtil.PromptCallback() {
                @Override
                public void onConfirm(String msg) {
                    scanRequest.setTitleText(msg);
                }
            });
        }
    });
```

5. 编译工程后，在手机上安装应用。打开应用后界面如下：



6. 点击 **标准 UI 下设置扫码界面 Title**，输入您要显示的 Title 信息，这里输入 **mPaaS**，单击 **确定** 完成。

7. 单击 **标准 UI 下使用扫一扫**，扫码页面左上角的 Title 显示第 6 步输入的 Title 信息，标准 UI 下设置扫码 Title 成功。



1.2.3.6. 自定义 UI 下使用扫码功能

本文将引导您绘制自定义 UI 界面并将自定义 UI 扫码的能力添加到工程中。

如需在自定义 UI 下使用扫码功能，请参考 代码示例。

该过程主要分为以下四个步骤：

1. 创建依赖工程
2. 在依赖工程中创建定义 UI 界面
3. 在依赖工程中使用扫码功能
4. 在主工程中调用自定义 UI 下的扫码功能

操作步骤

创建依赖工程

1. 单击 **File** > **New** > **New Module**。
2. 选择 **Android Library**，单击 **Next**。
3. 输入 **Module name**，单击 **Finish**。

在依赖工程中创建定义 UI 界面

1. 在 `custom` 的 `com.example.custom` 包中创建 `widget` 包。在 `widget` 包中添加 `APSurfaceTexture` 类，让其继承 `SurfaceTexture` 类，以获取图像流。

```
public class APSurfaceTexture extends SurfaceTexture {

 private static final String TAG = "APSurfaceTexture";

 public SurfaceTexture mSurface;

 public APSurfaceTexture() {
     super(0);
 }

 @TargetApi(Build.VERSION_CODES.JELLY_BEAN)
 @Override
 public void attachToGLContext(int texName) {
     mSurface.attachToGLContext(texName);
 }

 @TargetApi(Build.VERSION_CODES.JELLY_BEAN)
 @Override
 public void detachFromGLContext() {
     try {
         mSurface.detachFromGLContext();
     } catch (Exception ex) {
         try {
             Method nativeMethod =
SurfaceTexture.class.getDeclaredMethod("nativeDetachFromGLContext");
             nativeMethod.setAccessible(true);
             int retCode = (Integer) nativeMethod.invoke(mSurface);
             LoggerFactory.getTraceLogger().debug(TAG, "nativeDetachFromGLContext
invoke retCode:" + retCode);
         } catch (Exception e) {
             LoggerFactory.getTraceLogger().error(TAG, "nativeDetachFromGLContext
invoke exception:" + e.getMessage());
         }
         LoggerFactory.getTraceLogger().error(TAG, "mSurface.detachFromGLContext() ex
ception:" + ex.getMessage());
     }
 }

 @Override
 public boolean equals(Object o) {
     return mSurface.equals(o);
 }

 @Override
```

```
 public long getTimestamp() {
     return mSurface.getTimestamp();
 }

 @Override
 public void getTransformMatrix(float[] mtx) {
     mSurface.getTransformMatrix(mtx);
 }

 @Override
 public void release() {
     super.release();
     mSurface.release();
 }

 @Override
 public int hashCode() {
     return mSurface.hashCode();
 }

 @TargetApi(Build.VERSION_CODES.KITKAT)
 @Override
 public void releaseTexImage() {
     mSurface.releaseTexImage();
 }

 @TargetApi(Build.VERSION_CODES.ICE_CREAM_SANDWICH_MR1)
 @Override
 public void setDefaultBufferSize(int width, int height) {
     mSurface.setDefaultBufferSize(width, height);
 }

 @Override
 public void setOnFrameAvailableListener(OnFrameAvailableListener listener) {
     mSurface.setOnFrameAvailableListener(listener);
 }

 @Override
 public String toString() {
     return mSurface.toString();
 }

 @Override
 public void updateTexImage() {
     mSurface.updateTexImage();
 }
}
```

2. 在 `custom` 的 `widget` 包中添加 `APTextureView` 类，让其继承 `TextureView` 类，实现图像流的显示。

```
public class APTextureView extends TextureView {

 private static final String TAG = "APTextureView";
```

```
 private Field mSurfaceField;

 public APTextureView(Context context) {
     super(context);
 }

 public APTextureView(Context context, AttributeSet attrs) {
     super(context, attrs);
 }

 public APTextureView(Context context, AttributeSet attrs, int defStyleAttr) {
     super(context, attrs, defStyleAttr);
 }

 @Override
 protected void onDetachedFromWindow() {
     try {
         super.onDetachedFromWindow();
     } catch (Exception ex) {
         LoggerFactory.getTraceLogger().error(TAG, "onDetachedFromWindow exception:"
+ ex.getMessage());
     }
 }

 @Override
 public void setSurfaceTexture(SurfaceTexture surfaceTexture) {
     super.setSurfaceTexture(surfaceTexture);
     afterSetSurfaceTexture();
 }

 private void afterSetSurfaceTexture() {
     LoggerFactory.getTraceLogger().debug(TAG, "afterSetSurfaceTexture
Build.VERSION.SDK_INT:" + Build.VERSION.SDK_INT);
     if (Build.VERSION.SDK_INT < 16 || Build.VERSION.SDK_INT > 20) {
         return;
     }

     try {
         if (mSurfaceField == null) {
             mSurfaceField = TextureView.class.getDeclaredField("mSurface");
             mSurfaceField.setAccessible(true);
         }

         SurfaceTexture innerSurface = (SurfaceTexture) mSurfaceField.get(this);
         if (innerSurface != null) {
             if (!(innerSurface instanceof APSurfaceTexture)) {
                 APSurfaceTexture wrapSurface = new APSurfaceTexture();
                 wrapSurface.mSurface = innerSurface;
                 mSurfaceField.set(this, wrapSurface);
                 LoggerFactory.getTraceLogger().debug(TAG, "afterSetSurfaceTexture wra
p mSurface");
             }
         }
```

```
        } catch (Exception ex) {
            LoggerFactory.getTraceLogger().error(TAG, "afterSetSurfaceTexture
exception:" + ex.getMessage());
        }
    }
}
```

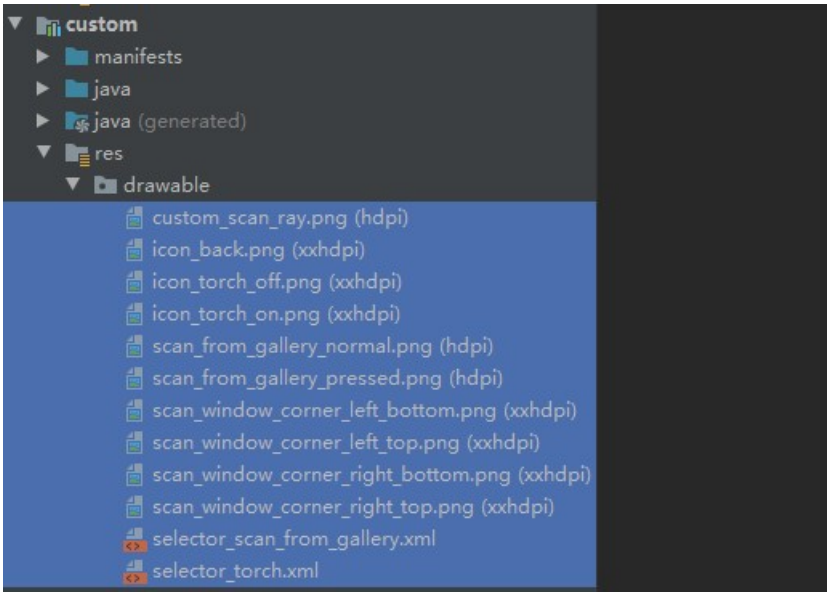
3. 在 `com.example.custom` 包中创建 `Utils` 类，实现图片的转换。

```
public class Utils {

 private static String TAG = "Utils";

 public static void toast(Context context, String msg) {
     Toast.makeText(context, msg, Toast.LENGTH_SHORT).show();
 }

 public static Bitmap changeBitmapColor(Bitmap bitmap, int color) {
     int bitmap_w = bitmap.getWidth();
     int bitmap_h = bitmap.getHeight();
     int[] arrayColor = new int[bitmap_w * bitmap_h];

     int count = 0;
     for (int i = 0; i < bitmap_h; i++) {
         for (int j = 0; j < bitmap_w; j++) {

             int originColor = bitmap.getPixel(j, i);
             // 非透明区域
             if (originColor != 0) {
                 originColor = color;
             }

             arrayColor[count] = originColor;
             count++;
         }
     }
     return Bitmap.createBitmap(arrayColor, bitmap_w, bitmap_h,
Bitmap.Config.ARGB_8888);
 }

 public static Bitmap uri2Bitmap(Context context, Uri uri) {
     Bitmap bitmap = null;
     InputStream in;
     try {
         in = context.getContentResolver().openInputStream(uri);
         if (in != null) {
             bitmap = BitmapFactory.decodeStream(in);
             in.close();
         }
     } catch (Exception e) {
         LoggerFactory.getTraceLogger().error(TAG, "uri2Bitmap: Exception " +
e.getMessage());
     }
     return bitmap;
 }
}
```

4. 在 `custom` 中创建 `res > values > attrs.xml` 文件并添加如下代码。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
 <declare-styleable name="scan">
     <attr name="shadowColor" format="color" />
 </declare-styleable>
</resources>
```

5. 在 `custom` 的 `res > drawable` 文件夹中粘贴如下 资源文件。



6. 在 `custom` 的 `widget` 包中添加 `FinderView` 类，让其继承 `View` 类，并添加如下代码。实现扫码窗口、边角及周边阴影的绘制功能。

```
public class FinderView extends View {

private static final int DEFAULT_SHADOW_COLOR = 0x96000000;

private int scanWindowLeft, scanWindowTop, scanWindowRight, scanWindowBottom;
private Bitmap leftTopCorner, rightTopCorner, leftBottomCorner, rightBottomCorner;
private Paint paint;
private int shadowColor;

public FinderView(Context context, AttributeSet attrs, int defStyle) {
    super(context, attrs, defStyle);
    init(context, attrs);
}

public FinderView(Context context, AttributeSet attrs) {
    super(context, attrs);
    init(context, attrs);
}

private void init(Context context, AttributeSet attrs) {
    applyConfig(context, attrs);
    setVisibility(INVISIBLE);
    initCornerBitmap(context);

    paint = new Paint();
```

```
    paint = new Paint();
    paint.setAntiAlias(true);
}

private void applyConfig(Context context, AttributeSet attrs) {
    if (attrs != null) {
        TypedArray typedArray = context.obtainStyledAttributes(attrs,
R.styleable.scan);
        shadowColor = typedArray.getColor(R.styleable.scan_shadowColor,
DEFAULT_SHADOW_COLOR);
        typedArray.recycle();
    }
}
//初始化扫码窗口边角样式
private void initCornerBitmap(Context context) {
    Resources res = context.getResources();
    leftTopCorner = BitmapFactory.decodeResource(res,
R.drawable.scan_window_corner_left_top);
    rightTopCorner = BitmapFactory.decodeResource(res,
R.drawable.scan_window_corner_right_top);
    leftBottomCorner = BitmapFactory.decodeResource(res,
R.drawable.scan_window_corner_left_bottom);
    rightBottomCorner = BitmapFactory.decodeResource(res,
R.drawable.scan_window_corner_right_bottom);
}

@Override
public void draw(Canvas canvas) {
    super.draw(canvas);
    drawShadow(canvas);
    drawCorner(canvas);
}
//绘制扫码窗口边角样式
private void drawCorner(Canvas canvas) {
    paint.setAlpha(255);
    canvas.drawBitmap(leftTopCorner, scanWindowLeft, scanWindowTop, paint);
    canvas.drawBitmap(rightTopCorner, scanWindowRight - rightTopCorner.getWidth(), sc
anWindowTop, paint);
    canvas.drawBitmap(leftBottomCorner, scanWindowLeft, scanWindowBottom -
leftBottomCorner.getHeight(), paint);
    canvas.drawBitmap(rightBottomCorner, scanWindowRight -
rightBottomCorner.getWidth(), scanWindowBottom - rightBottomCorner.getHeight(), paint
);
}
//绘制扫码周边阴影
private void drawShadow(Canvas canvas) {
    paint.setColor(shadowColor);
    canvas.drawRect(0, 0, getWidth(), scanWindowTop, paint);
    canvas.drawRect(0, scanWindowTop, scanWindowLeft, scanWindowBottom, paint);
    canvas.drawRect(scanWindowRight, scanWindowTop, getWidth(), scanWindowBottom, pai
nt);
    canvas.drawRect(0, scanWindowBottom, getWidth(), getHeight(), paint);
}

/**
```

```
 * 根据 RayView 的位置决定扫码窗口的位置
 */
public void setScanWindowLocation(int left, int top, int right, int bottom) {
    scanWindowLeft = left;
    scanWindowTop = top;
    scanWindowRight = right;
    scanWindowBottom = bottom;
    invalidate();
    setVisibility(VISIBLE);
}

public void setShadowColor(int shadowColor) {
    this.shadowColor = shadowColor;
}
//设置扫码窗口边角颜色
public void setCornerColor(int angleColor) {
    leftTopCorner = Utils.changeBitmapColor(leftTopCorner, angleColor);
    rightTopCorner = Utils.changeBitmapColor(rightTopCorner, angleColor);
    leftBottomCorner = Utils.changeBitmapColor(leftBottomCorner, angleColor);
    rightBottomCorner = Utils.changeBitmapColor(rightBottomCorner, angleColor);
}
}
```

7. 在 `custom` 的 `widget` 包中添加 `RayView` 类，让其继承 `ImageView` 类，并添加如下代码。实现扫描射线的绘制功能。

```
public class RayView extends ImageView {

private FinderView mFinderView;
private ScaleAnimation scanAnimation;
private int[] location = new int[2];

public RayView(Context context, AttributeSet attrs) {
    super(context, attrs);
}

public RayView(Context context) {
    super(context);
}

@Override
protected void onLayout(boolean changed, int left, int top, int right, int bottom) {
    super.onLayout(changed, left, top, right, bottom);

    // 设置 FinderView 中扫码窗口的位置
    getLocationOnScreen(location);
    if (mFinderView != null) {
        mFinderView.setScanWindowLocation(location[0], location[1], location[0] + get
Width(), location[1] + getHeight());
    }
}

public void startScanAnimation() {
    setVisibility(VISIBLE);
    if (scanAnimation == null) {
        scanAnimation = new ScaleAnimation(1.0f, 1.0f, 0.0f, 1.0f);
        scanAnimation.setDuration(3000L);
        scanAnimation.setFillAfter(true);
        scanAnimation.setRepeatCount(Animation.INFINITE);
        scanAnimation.setInterpolator(new AccelerateDecelerateInterpolator());
    }
    startAnimation(scanAnimation);
}

public void stopScanAnimation() {
    setVisibility(INVISIBLE);
    if (scanAnimation != null) {
        this.clearAnimation();
        scanAnimation = null;
    }
}

public void setFinderView(FinderView FinderView) {
    mFinderView = FinderView;
}
}
```

8. 在 `custom` 的 `res` 中创建 `layout > File > view_scan.xml` 文件，并添加如下代码，绘制扫描页面的布局界面。

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

    <com.example.custom.widget.FinderView
        android:id="@+id/finder_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:gravity="center_vertical"
        android:orientation="horizontal">

        <ImageView
            android:id="@+id/back"
            android:layout_width="48dp"
            android:layout_height="48dp"
            android:scaleType="center"
            android:src="@drawable/icon_back" />

        <TextView
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:gravity="center"
            android:text="@string/custom_title"
            android:textColor="#ffffff"
            android:textSize="16sp" />

        <ImageView
            android:id="@+id/gallery"
            android:layout_width="34dp"
            android:layout_height="34dp"
            android:layout_marginEnd="10dp"
            android:layout_marginRight="10dp"
            android:scaleType="fitXY"
            android:src="@drawable/selector_scan_from_gallery" />

        <ImageView
            android:id="@+id/torch"
            android:layout_width="34dp"
            android:layout_height="34dp"
            android:layout_marginEnd="10dp"
            android:layout_marginRight="10dp"
            android:scaleType="fitXY"
            android:src="@drawable/selector_torch" />
    </LinearLayout>

    <com.example.custom.widget.RayView
        android:id="@+id/ray_view"
        android:layout_width="270dp"
        android:layout_height="280dp"
```

```
        android:layout_centerInParent="true"
        android:background="@drawable/custom_scan_ray" />

    <TextView
        android:id="@+id/tip_tv"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/ray_view"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dp"
        android:includeFontPadding="false"
        android:text="@string/scan_tip"
        android:textColor="#7fffffff"
        android:textSize="14sp" />

</merge>
```

9. 在 `widget` 包中添加 `ScanView` 类，让其继承 `RelativeLayout` 类，并添加如下代码。实现扫码相关的 View 与扫码引擎的交互功能。

```
public class ScanView extends RelativeLayout {

private RayView mRayView;

public ScanView(Context context) {
    super(context);
    init(context);
}

public ScanView(Context context, AttributeSet attrs) {
    super(context, attrs);
    init(context);
}

public ScanView(Context context, AttributeSet attrs, int defStyle) {
    super(context, attrs, defStyle);
    init(context);
}

private void init(Context ctx) {
    LayoutInflater.from(ctx).inflate(R.layout.view_scan, this, true);
    FinderView finderView = (FinderView) findViewById(R.id.finder_view);
    mRayView = (RayView) findViewById(R.id.ray_view);
    mRayView.setFinderView(finderView);
}

public void onStartScan() {
    mRayView.startScanAnimation();
}

public void onStopScan() {
    mRayView.stopScanAnimation();
}

public float getCropWidth() {
```

```
    return mRayView.getWidth() * 1.1f;
}

public Rect getScanRect(Camera camera, int previewWidth, int previewHeight) {
    if (camera == null) {
        return null;
    }
    int[] location = new int[2];
    mRayView.getLocationOnScreen(location);
    Rect r = new Rect(location[0], location[1],
            location[0] + mRayView.getWidth(), location[1] + mRayView.getHeight());
    Camera.Size size;
    try {
        size = camera.getParameters().getPreviewSize();
    } catch (Exception e) {
        return null;
    }
    if (size == null) {
        return null;
    }
    double rateX = (double) size.height / (double) previewWidth;
    double rateY = (double) size.width / (double) previewHeight;
    // 裁剪框大小 = 网格动画框大小 * 1.1
    int expandX = (int) (mRayView.getWidth() * 0.05);
    int expandY = (int) (mRayView.getHeight() * 0.05);
    Rect resRect = new Rect(
            (int) ((r.top - expandY) * rateY),
            (int) ((r.left - expandX) * rateX),
            (int) ((r.bottom + expandY) * rateY),
            (int) ((r.right + expandX) * rateX));

    Rect finalRect = new Rect(
            resRect.left < 0 ? 0 : resRect.left,
            resRect.top < 0 ? 0 : resRect.top,
            resRect.width() > size.width ? size.width : resRect.width(),
            resRect.height() > size.height ? size.height : resRect.height());

    Rect rect1 = new Rect(
            finalRect.left / 4 * 4,
            finalRect.top / 4 * 4,
            finalRect.right / 4 * 4,
            finalRect.bottom / 4 * 4);

    int max = Math.max(rect1.right, rect1.bottom);
    int diff = Math.abs(rect1.right - rect1.bottom) / 8 * 4;

    Rect rect2;
    if (rect1.right > rect1.bottom) {
        rect2 = new Rect(rect1.left, rect1.top - diff, max, max);
    } else {
        rect2 = new Rect(rect1.left - diff, rect1.top, max, max);
    }
    return rect2;
}
```

```
}
```

0. 在 `custom` 的 `layout` 文件夹中创建 `activity_custom_scan.xml` 文件并添加如下代码。绘制自定义扫码功能的主界面。

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <com.mpaas.aar.demo.custom.widget.APTextureView
        android:id="@+id/surface_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <com.mpaas.aar.demo.custom.widget.ScanView
        android:id="@+id/scan_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</FrameLayout>
```

在依赖工程中使用扫码功能

1. 在 `custom` 的 `com.example.custom` 包中添加 `ScanHelper` 类，并添加如下代码。调用扫码功能以及获取扫码结果的回调结果。

```
public class ScanHelper {

 private static class Holder {
     private static ScanHelper instance = new ScanHelper();
 }

 private ScanCallback scanCallback;

 private ScanHelper() {
 }

 public static ScanHelper getInstance() {
     return Holder.instance;
 }

 public void scan(Context context, ScanCallback scanCallback) {
     if (context == null) {
         return;
     }
     this.scanCallback = scanCallback;
     context.startActivity(new Intent(context, CustomScanActivity.class));
 }

 void notifyScanResult(boolean isProcessed, Intent resultData) {
     if (scanCallback != null) {
         scanCallback.onScanResult(isProcessed, resultData);
         scanCallback = null;
     }
 }

 public interface ScanCallback {
     void onScanResult(boolean isProcessed, Intent result);
 }
}
```

2. 在 `custom` 的 `com.example.custom` 包中添加 `CustomScanActivity` 类，让其继承 `Activity` 类。设置界面沉浸模式并创建资源文件对应的 `View` 和 `Button` 。

```
public class CustomScanActivity extends Activity {
 private final String TAG = CustomScanActivity.class.getSimpleName();
 private static final int REQUEST_CODE_PERMISSION = 1;
 private static final int REQUEST_CODE_PHOTO = 2;
 private ImageView mTorchBtn;
 private APTextureView mTextureView;
 private ScanView mScanView;
 private boolean isFirstStart = true;
 private boolean isPermissionGranted;
 private boolean isScanning;
 private boolean isPaused;
 private Rect scanRect;
 private MPScanner mpScanner;
 @Override
 protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_custom_scan);

        // 设置沉浸模式
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT) {
            getWindow().setFlags(
                    WindowManager.LayoutParams.FLAG_TRANSLUCENT_STATUS,
                    WindowManager.LayoutParams.FLAG_TRANSLUCENT_STATUS);
        }

        mTextureView = findViewById(R.id.surface_view);
        mScanView = findViewById(R.id.scan_view);
        mTorchBtn = findViewById(R.id.torch);

    }

     @Override
    public void onPause() {
        super.onPause();

    }

    @Override
    public void onResume() {
        super.onResume();

    }

    @Override
    public void onDestroy() {
        super.onDestroy();

    }

    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[] permission
s, @NonNull int[] grantResults) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);

    }
     @Override
    public void onBackPressed() {
        super.onBackPressed();

    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

    }
}
```

3. 实现打开手机相册的功能。

i. 在 `CustomScanActivity` 中创建 `pickImageFromGallery` 方法。

```
private void pickImageFromGallery() {
 Intent intent = new Intent(Intent.ACTION_GET_CONTENT);
 intent.setType("image/*");
 startActivityForResult(intent, REQUEST_CODE_PHOTO);
}
```

ii. `onCreate` 方法中添加 `gallery` 的单击事件，并调用 `pickImageFromGallery` 方法。

```
 findViewById(R.id.gallery).setOnClickListener(new View.OnClickListener() {
     @Override
     public void onClick(View v) {
         pickImageFromGallery();
     }
 });
```

4. 实现切换手电开关的功能。

i. 在 `CustomScanActivity` 中创建 `switchTorch` 方法。

```
 private void switchTorch() {
     boolean torchOn = mpScanner.switchTorch();
     mTorchBtn.setSelected(torchOn);
 }
```

ii. `onCreate` 方法中添加 `mTorchBtn` 的单击事件，并调用 `switchTorch` 方法。

```
mTorchBtn.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
             switchTorch();
         }
     });
```

5. 在 `CustomScanActivity` 中创建 `notifyScanResult` 方法，`onBackPressed` 中调用 `notifyScanResult` 方法。

```
 private void notifyScanResult(boolean isProcessed, Intent resultData) {
     ScanHelper.getInstance().notifyScanResult(isProcessed, resultData);
 }

 @Override
 public void onBackPressed() {
     super.onBackPressed();
     notifyScanResult(false, null);
 }
```

6. 在 `CustomScanActivity` 的 `onCreate` 方法中添加 `back` 的单击事件，并调用 `onBackPressed` 方法。

```
    findViewById(R.id.back).setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            onBackPressed();
        }
    });
```

7. 在 `CustomScanActivity` 中创建 `initMPScanner` 方法，并使用 `mpScanner` 对象的 `setRecognizeType` 方法设置识别码的类型。

```
private void initMPScanner() {
    mpScanner = new MPScanner(this);
    mpScanner.setRecognizeType(
            MPRecognizeType.QR_CODE,
            MPRecognizeType.BAR_CODE,
            MPRecognizeType.DM_CODE,
            MPRecognizeType.PDF417_CODE
    );
}
```

8. 在 `CustomScanActivity` 中创建 `onScanSuccess` 方法，并实现如下代码。

```
private void onScanSuccess(final MPScanResult result) {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            if (result == null) {
                notifyScanResult(true, null);
            } else {
                Intent intent = new Intent();
                intent.setData(Uri.parse(result.getText()));
                notifyScanResult(true, intent);
            }
            CustomScanActivity.this.finish();
        }
    });
}
```

9. 在 `CustomScanActivity` 中创建 `initScanRect` 方法，初始化扫描功能。

i. 调用 `mpScanner` 对象的 `getCamera` 方法获取 `Camera` 对象并调用 `mpScanner` 对象的 `setScanRegion` 方法设置扫描区域。

```
private void initScanRect() {
 if (scanRect == null) {
     scanRect = mScanView.getScanRect(
             mpScanner.getCamera(), mTextureView.getWidth(),
mTextureView.getHeight());

     float cropWidth = mScanView.getCropWidth();
     LoggerFactory.getTraceLogger().debug(TAG, "cropWidth: " + cropWidth);
     if (cropWidth > 0) {
         // 预览放大 = 屏幕宽 / 裁剪框宽
         WindowManager wm = (WindowManager)
getSystemService(Context.WINDOW_SERVICE);
         float screenWith = wm.getDefaultDisplay().getWidth();
         float screenHeight = wm.getDefaultDisplay().getHeight();
         float previewScale = screenWith / cropWidth;
         if (previewScale < 1.0f) {
             previewScale = 1.0f;
         }
         if (previewScale > 1.5f) {
             previewScale = 1.5f;
         }
         LoggerFactory.getTraceLogger().debug(TAG, "previewScale: " +
previewScale);
         Matrix transform = new Matrix();
         transform.setScale(previewScale, previewScale, screenWith / 2,
screenHeight / 2);
         mTextureView.setTransform(transform);
     }
 }
 mpScanner.setScanRegion(scanRect);
}
```

ii. 使用 `mpScanner` 对象的 `setMPScanListener` 方法实现扫描监听器的功能。

```
mpScanner.setMPScanListener(new MPScanListener() {
    @Override
    public void onConfiguration() {
        mpScanner.setDisplayView(mTextureView);
    }

    @Override
    public void onStart() {
        if (!isPaused) {
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    if (!isFinishing()) {
                        initScanRect();
                        mScanView.onStartScan();
                    }
                }
            });
        }
    }

    @Override
    public void onSuccess(MPScanResult mpScanResult) {
        mpScanner.beep();
        onScanSuccess(mpScanResult);
    }

    @Override
    public void onError(MPScanError mpScanError) {
        if (!isPaused) {
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    Utils.toast(CustomScanActivity.this,
getString(R.string.camera_open_error));
                }
            });
        }
    }
});
```

iii. 使用 `mpScanner` 对象的 `setMPImageGrayListener` 方法实现识别图像灰度值的监听功能。

```
mpScanner.setMPImageGrayListener(new MPImageGrayListener() {
    @Override
    public void onGetImageGray(int gray) {
        // 注意：该回调在昏暗环境下可能会连续多次执行
        if (gray < MPImageGrayListener.LOW_IMAGE_GRAY) {
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    Utils.toast(CustomScanActivity.this, "光线太暗，请打开手电筒");
                }
            });
        }
    }
});
}
```

0. 在 `CustomScanActivity` 中分别创建 `startScan` 和 `stopScan` 方法，实现开启和关闭相机扫码权限。

```
private void startScan() {
    try {
        mpScanner.openCameraAndStartScan();
        isScanning = true;
    } catch (Exception e) {
        isScanning = false;
        LoggerFactory.getTraceLogger().error(TAG, "startScan: Exception " +
e.getMessage());
    }
}

private void stopScan() {
    mpScanner.closeCameraAndStopScan();
    mScanView.onStopScan();
    isScanning = false;
    if (isFirstStart) {
        isFirstStart = false;
    }
}
```

1. 在 `CustomScanActivity` 中创建 `onPermissionGranted` 方法、`checkCameraPermission` 方法和 `scanFromUri` 方法。

```
private void onPermissionGranted() {
    isPermissionGranted = true;
    startScan();
}

private void checkCameraPermission() {
    if (PermissionChecker.checkSelfPermission(
            this, Manifest.permission.CAMERA) !=
PermissionChecker.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions(this, new String[]
{Manifest.permission.CAMERA}, REQUEST_CODE_PERMISSION);
    } else {
        onPermissionGranted();
    }
}

private void scanFromUri(Uri uri) {
    final Bitmap bitmap = Utils.uri2Bitmap(this, uri);
    if (bitmap == null) {
        notifyScanResult(true, null);
        finish();
    } else {
        new Thread(new Runnable() {
            @Override
            public void run() {
                MPScanResult mpScanResult = mpScanner.scanFromBitmap(bitmap);
                mpScanner.beep();
                onScanSuccess(mpScanResult);
            }
        }, "scanFromUri").start();
    }
}
```

2. 在 `CustomScanActivity` 的 `onCreate` 方法中调用 `checkCameraPermission` 方法检查相机权限。

```
checkCameraPermission();
```

3. 在 `CustomScanActivity` 的 `onPause` 、 `onResume` 、 `onDestroy` 、 `onRequestPermissionsResult` 和 `onActivityResult` 方法中分别添加如下内容。

```
@Override
public void onPause() {
    super.onPause();
    isPaused = true;
    if (isScanning) {
        stopScan();
    }
}

@Override
public void onResume() {
    super.onResume();
    isPaused = false;
    if (!isFirstStart && isPermissionGranted) {
        startScan();
    }
}

@Override
public void onDestroy() {
    super.onDestroy();
    mpScanner.release();
}
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions
, @NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    if (requestCode == REQUEST_CODE_PERMISSION) {
        int length = Math.min(permissions.length, grantResults.length);
        for (int i = 0; i < length; i++) {
            if (TextUtils.equals(permissions[i], Manifest.permission.CAMERA)) {
                if (grantResults[i] != PackageManager.PERMISSION_GRANTED) {
                    Utils.toast(this, getString(R.string.camera_no_permission));
                } else {
                    onPermissionGranted();
                }
                break;
            }
        }
    }
      @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (data == null) {
        return;
    }
    if (requestCode == REQUEST_CODE_PHOTO) {
        scanFromUri(data.getData());
    }
}
}
```

4. 在 `custom` 的 `AndroidManifest.xml` 文件中设置 `CustomScanActivity` 为 `custom` 的主入

口。

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

    <application>
        <activity
            android:name=".CustomScanActivity"
            android:configChanges="orientation|keyboardHidden|navigation"
            android:exported="false"
            android:launchMode="singleTask"
            android:screenOrientation="portrait"
            android:theme="@android:style/Theme.NoTitleBar"
            android:windowSoftInputMode="adjustResize|stateHidden" />
    </application>

</manifest>
```

在主工程中调用自定义 UI 下的扫码功能

1. 在 `activity_main.xml` 文件中，添加 `Button` ，并设置 `Button` 的 ID 为 `custom_ui_btn` 。

```
<Button
    android:id="@+id/custom_ui_btn"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="208dp"
    android:background="#108EE9"
    android:gravity="center"
    android:text="自定义 UI 下使用扫一扫"
    android:textColor="#ffffff"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

2. 在 `MainActivity` 类中编写代码。添加 `custom_ui_btn` 按钮的单击事件。获取自定义 UI 界面，并使用自定义 UI 的扫码功能。代码如下所示：

```
findViewById(R.id.custom_ui_btn).setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            ScanHelper.getInstance().scan(MainActivity.this, new
ScanHelper.ScanCallback() {
                @Override
                public void onScanResult(boolean isProcessed, Intent result) {
                    if (!isProcessed) {
                        // 扫码界面单击物理返回键或左上角返回键
                        return;
                    }

                    if (result == null || result.getData() == null) {
                        Toast.makeText(MainActivity.this, "扫码失败，请重试！",
Toast.LENGTH_SHORT).show();
                        return;
                    }
                    new AlertDialog.Builder(MainActivity.this)
                            .setMessage(result.getData().toString())
                            .setPositiveButton(R.string.confirm, null)
                            .create()
                            .show();
                }
            });
        }
    });
```

3. 编译运行工程后，单击 **自定义 UI 下使用扫一扫** 后即可使用自定义 UI 下的扫码功能。
4. 扫描二维码，会弹出该二维码的信息。

1.3. 接入 iOS

1.3.1. 快速开始

扫一扫 SDK 是支付宝目前正在使用的识别二维码、条形码等功能的 SDK 。本文介绍如何将扫一扫组件接入到 iOS 客户端。扫一扫支持 **基于 mPaaS 框架接入** 、**基于已有工程且使用 mPaaS 插件接入** 以及 **基于已有工程且使用 CocoaPods 接入** 三种接入方式。您可以参考 接入方式介绍，根据实际业务情况选择合适的接入方式。

前置条件

您已经根据您的接入方式，将扫一扫组件 SDK 添加至工程。更多信息，请参见以下内容：

- 基于 mPaaS 框架接入
- 基于已有工程且使用 mPaaS 插件接入
- 基于已有工程且使用 CocoaPods 接入

添加SDK

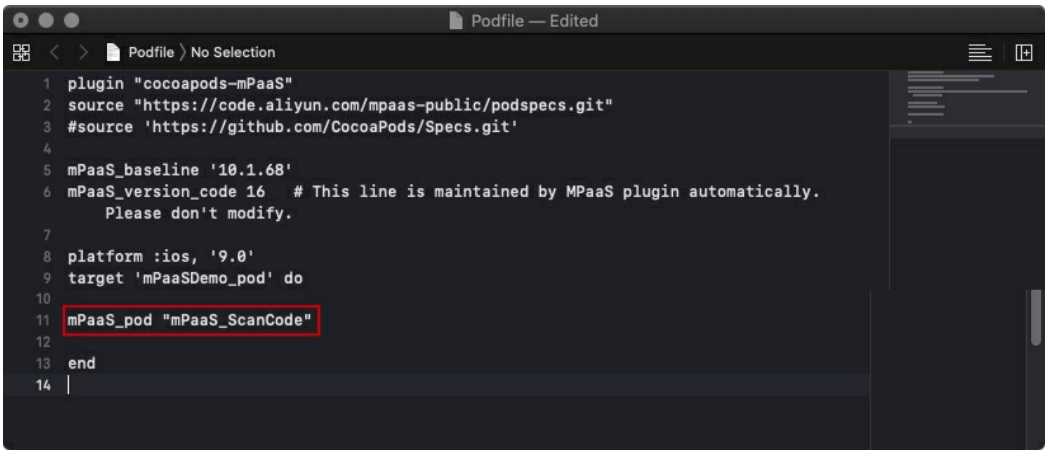
根据您采用的接入方式，请选择相应的添加方式。

- 使用 mPaaS Xcode Extension。此方式适用于采用了 **基于 mPaaS 框架接入** 或 **基于已有工程且使用 mPaaS 插件接入** 的接入方式。
  i. 单击 Xcode 菜单项 **Editor** > **mPaaS** > **编辑工程**，打开编辑工程页面。

ii. 选择 **扫码**，保存后点击 **开始编辑**，即可完成添加。

- 使用 cocoapods-mPaaS 插件。此方式适用于采用了 **基于已有工程且使用 CocoaPods 接入** 的接入方式。

  i. 在 Podfile 文件中，使用 `mPaaS_pod "mPaaS_ScanCode"` 添加扫码组件依赖。



  ii. 在命令行中执行 `pod install` 即可完成接入。

使用 SDK（ ≥ 10.1.68.17 ）

本文将结合 扫一扫 官方 Demo 介绍如何在 10.1.68.17 及以上版本的基线中使用扫一扫 SDK。

> **说明**
>
> 多码识别功能只支持在标准 UI 下使用。

操作步骤如下：

1. 唤起标准扫码页面并处理扫描结果。

```
  @interface MPScanDemoVC()<TBScanViewControllerDelegate>
  @property(nonatomic, strong) TBScanViewController *scanVC;
  @end
 - (void)defaultScan {
    TBScanViewController *vc = [[MPScanCodeAdapterInterface sharedInstance]
createDefaultScanPageWithallback:^(id  _Nonnull result, BOOL keepAlive) {
0000000000000000        // 处理扫描结果
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@""
message:result[@"resp_result"] delegate:self cancelButtonTitle:@"OK"
otherButtonTitles:nil, nil];
         alert.tag = 1999;
         [alert show];
     }];
     [self.navigationController pushViewController:vc animated:YES];
     self.scanVC =  vc;
 }
```

2. 持续扫码。

```
 - (void)alertView:(UIAlertView *)alertView clickedButtonAtIndex:
(NSInteger)buttonIndex {
     // 持续扫码
     [self.scanVC resumeScan];
 }
```

使用 SDK（ < 10.1.68.17 ）

本文将结合 扫一扫 官方 Demo 介绍如何在 10.1.68.17 以下版本的基线中使用扫一扫 SDK。

操作步骤如下：

1. 唤起扫码界面。

```
 @interface MPScanDemoVC()<TBScanViewControllerDelegate>
 @property(nonatomic, strong) TBScanViewController *scanVC;
 @end
 - (void)startDefauleScanViewController
 {
     TBScanViewController *vc = [[TBScanViewController alloc] init];
     vc.scanType = ScanType_All_Code;
     vc.delegate = self;
     [self.navigationController pushViewController:vc animated:YES];
     self.scanVC = vc;
 }
```

2. 处理扫描结果。

```
#pragma mark 处理扫描结果
-(void)didFind:(NSArray<TBScanResult*>*)resultArray
{
    if([resultArray count] > 0) {
        TBScanResult *result = resultArray.firstObject;
        NSString* content = result.data;

        dispatch_async(dispatch_get_main_queue(), ^{
            // 注意：扫码的结果是在子线程，如有 UI 相关操作，请切换到主线程
            UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@""
message:content delegate:self cancelButtonTitle:@"OK" otherButtonTitles:nil, nil];
            [alert show];
        });
    }
}
```

3. 持续扫码。

```
#pragma mark alert
- (void)alertView:(UIAlertView *)alertView clickedButtonAtIndex:
(NSInteger)buttonIndex {
    [self.scanVC resumeScan];
}
```

1.3.2. 进阶指南

本文将结合 扫一扫 官方 Demo 介绍如何在 10.1.60 及以上版本的基线中使用标准 UI 下的扫码功能和自定义 UI 下的扫码功能。

标准 UI 下使用扫一扫

在标准 UI 下修改扫码所在页面的参数。

```objectivec
- (void)custoDefaultScan {
   TBScanViewController *vc = [[MPScanCodeAdapterInterface sharedInstance]
createDefaultScanPageWithallback:^(id  _Nonnull result, BOOL keepAlive) {
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@""
message:result[@"resp_result"] delegate:self cancelButtonTitle:@"OK"
otherButtonTitles:nil, nil];
        alert.tag = 1001;
        [alert show];
    }];
    [self.navigationController pushViewController:vc animated:YES];
    self.scanVC =  vc;

    // 设置扫码界面 title
    vc.title = @"标准扫码";

    // 设置打开手电筒提示文字
    vc.torchStateNormalTitle = @"打开手电筒";

    // 设置关闭手电筒提示文字
    vc.torchStateSelectedTitle = @"关闭手电筒";

    // 设置扫码识别类型
    vc.scanType =  ScanType_QRCode;

    // 设置选择相册按钮
    vc.navigationItem.rightBarButtonItem = [[UIBarButtonItem alloc]
initWithImage:APCommonUILoadImage(@"camera") style:UIBarButtonItemStylePlain target:sel
f action:@selector(selectPhotos)];

}

- (void)selectPhotos
{
    [self.scanVC scanPhotoLibrary];
}
```

自定义 UI 下使用扫一扫

若您需要完全自定义扫码 UI 界面，可自定义扫码页并让其继承 TBScanViewController。

- 创建扫码页，并自定义扫码区。

```
   @interface MPScanCodeViewController : TBScanViewController
 <TBScanViewControllerDelegate>

   @end

   @implementation MPScanCodeViewController

   - (instancetype)init
   {
       if (self = [super init])
       {
```

```
        {
            self.delegate = self;
            self.scanType = ScanType_All_Code;
        }
        return self;
    }

    - (void)viewDidLoad {
        [super viewDidLoad];
        // Do any additional setup after loading the view.
        self.title = @"扫码";

        // 自定义扫码界面大小
        CGRect rect = [MPScanCodeViewController constructScanAnimationRect];
        self.rectOfInterest = rect;

        // 自定义相册按钮
        self.navigationItem.rightBarButtonItem = [[UIBarButtonItem alloc]
initWithTitle:@"选择相册" style:UIBarButtonItemStylePlain target:self
action:@selector(selectPhoto)];
    }

    + (CGRect)constructScanAnimationRect
    {
        CGSize screenXY = [UIScreen mainScreen].bounds.size;
        NSInteger focusFrameWH = screenXY.width / 320 * 220;//as wx
        int offet = 10;
        if (screenXY.height == 568)
            offet = 19;

        return CGRectMake((screenXY.width - focusFrameWH) / 2,
                          (screenXY.height - 64 - focusFrameWH - 83 - 50 - offet) / 2 +
64,
                          focusFrameWH,
                          focusFrameWH);
    }

    -(void)buildContainerView:(UIView*)containerView
    {
        // 自定义扫码框 view
        UIView* bg = [[UIView alloc] initWithFrame:containerView.bounds];
        [containerView addSubview:bg];
        CGRect rect = [MPScanCodeViewController constructScanAnimationRect];
        UIView* view = [[UIView alloc] initWithFrame:rect];
        view.backgroundColor = [UIColor orangeColor];
        view.alpha = 0.5;
        [bg addSubview:view];
    }

    - (void)selectPhoto
    {
        [self scanPhotoLibrary];
    }
```

- **处理扫码结果。**

```
-(void)didFind:(NSArray<TBScanResult*>*)resultArray
{
    TBScanResult *result = resultArray.firstObject;
    NSString* content = result.data;
    if (result.resultType == TBScanResultTypeQRCode) {
        content = [NSString stringWithFormat:@"qrcode:%@, hiddenData:%@,
TBScanQRCodeResultType:%@", result.data, result.hiddenData, [result.extData objectFor
Key:TBScanResultTypeQRCode]];
        NSLog(@"subType is %@, ScanType_QRCode is %@", @(result.subType),
@(ScanType_QRCode));
    } else if (result.resultType == TBScanResultTypeVLGen3Code) {
        content = [NSString stringWithFormat:@"gen3:%@", result.data];
        NSLog(@"subType is %@, ScanType_GEN3 is %@", @(result.subType),
@(ScanType_GEN3));
    } else if (result.resultType == TBScanResultTypeGoodsBarcode) {
        content = [NSString stringWithFormat:@"barcode:%@", result.data];
        NSLog(@"subType is %@, EAN13 is %@", @(result.subType), @(EAN13));
    } else if (result.resultType == TBScanResultTypeDataMatrixCode) {
        content = [NSString stringWithFormat:@"dm:%@", result.data];
        NSLog(@"subType is %@, ScanType_DATAMATRIX is %@", @(result.subType),
@(ScanType_DATAMATRIX));
    } else if (result.resultType == TBScanResultTypeExpressCode) {
        content = [NSString stringWithFormat:@"express:%@", result.data];
        NSLog(@"subType is %@, ScanType_FASTMAIL is %@", @(result.subType),
@(ScanType_FASTMAIL));
    }
    dispatch_async(dispatch_get_main_queue(), ^{
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"" message:content
delegate:self cancelButtonTitle:@"OK" otherButtonTitles:nil, nil];
        alert.tag = 9999;
        [alert show];
    });
}
```

- **持续扫码。**

```
- (void)alertView:(UIAlertView *)alertView clickedButtonAtIndex:
(NSInteger)buttonIndex{
    // 持续扫码
    [self resumeScan];
}
```

- **本地相册识别失败的回调。**

```
- (void)scanPhotoFailed
{
    // 相册识别失败的回调
    NSLog(@"scanPhotoFailed");
}
```

- **其他回调处理。**

```
- (void)cameraPermissionDenied
{
    [self.navigationController popViewControllerAnimated:YES];
}

- (void)cameraDidStart
{
    NSLog(@"started!!");
}

-(void)setTorchState:(TorchState)bState
{
    NSLog(@"TorchState:%lu", (unsigned long)bState);
}

-(void)userTrack:(NSString*)name
{
    NSLog(@"userTrack:%@", name);
}

-(void)userTrack:(NSString*)name args:(NSDictionary*)data
{
    NSLog(@"userTrack:%@, args:%@", name, data);
}

- (void)scanPhotoFailed
{
    // 相册识别失败的回调
    NSLog(@"scanPhotoFailed");
}
```

1.3.3. 多码识别

本文介绍如何将多码识别组件接入到 iOS 客户端。多码识别支持基于 mPaaS 框架接入、基于已有工程且使用 mPaaS 插件接入以及基于已有工程且使用 CocoaPods 接入三种接入方式。您可以参考 接入方式介绍，根据实际业务情况选择合适的接入方式。

前置条件

您已经根据您的接入方式，将扫一扫组件 SDK 添加至工程。更多信息，请参见以下内容：

- 基于 mPaaS 框架接入
- 基于已有工程且使用 mPaaS 插件接入
- 基于已有工程且使用 CocoaPods 接入

添加 SDK

根据您采用的接入方式，请选择相应的添加方式。

- 使用 mPaaS Xcode Extension。此方式适用于采用了 **基于 mPaaS 框架接入** 或 **基于已有工程且使用 mPaaS 插件接入**的接入方式。
  i. 单击 Xcode 菜单项 **Editor**> **mPaaS**> **编辑工程**> **升级基线**，切换工程到定制基线 `cp_change_28238` 或基线 10.2.3.5 以上版本。

> **说明**
>
> 如果 **升级基线**不可点，请确保工程配置文件已经导入，参考前置条件。

ii. 选择 **扫码**，保存后点击 **开始编辑**，即可完成添加。

- 使用 cocoapods-mPaaS 插件。此方式适用于采用了 **基于已有工程且使用 CocoaPods 接入** 的接入方式。
  i. 在 `Podfile` 文件中，
     a. 修改 mPaaS_baseline 为 `cp_change_28238` 或基线 10.2.3.5 以上版本。
     b. 使用 `mPaaS_pod "mPaaS_ScanCode"` 添加扫码组件依赖。

```
# mPaaS Pods Begin
plugin "cocoapods-mPaaS"
source "https://code.aliyun.com/mpaas-public/podspecs.git"
mPaaS_baseline 'cp_change_28238'  # 请将 x.x.x 替换成真实基线版本
mPaaS_version_code 4   # This line is maintained by MPaaS plugin automatically. Please don't modify.
# mPaaS Pods End
# ------------------------------------------------------------------
# Uncomment the next line to define a global platform for your project
# platform :ios, '9.0'

target 'MP_Demo' do
  # Comment the next line if you don't want to use dynamic frameworks
  use_frameworks!

  # Pods for MP_Demo
  mPaaS_pod "mPaaS_ScanCode"

end
```

  ii. 单击此处查看如何使用 CocoaPods，根据需要在命令行中执行 `pod install` 或 `pod update` 即可完成接入。

使用 SDK

打开默认扫码页面

本文将结合 扫一扫 官方 Demo 介绍如何在定制基线 `cp_change_28238` 或 10.2.3.5 以上版本的基线中使用扫一扫多码识别默认 UI SDK。

- 唤起默认扫码页面并处理扫描结果。

```
 #import <TBScanSDK/TBScanSDK.h>

 @interface MPScanDemoVC()

 @property(nonatomic, strong) TBScanViewController *scanVC;

 @end

- (void)defaultScan {

   // 是否显示相册入口
   [MPScanCodeAdapterInterface sharedInstance].shoulShowAlbum = NO;

   TBScanViewController *vc = [[MPScanCodeAdapterInterface sharedInstance]
createDefaultScanPageWithallback:^(id  _Nonnull result, BOOL keepAlive) {
       // 处理扫描结果
       UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@""
message:result[@"resp_result"] delegate:self cancelButtonTitle:@"OK"
otherButtonTitles:nil, nil];
        alert.tag = 1999;
        [alert show];
    }];

     // 设置扫码类型
     vc.scanType =  ScanType_Default_Code;

    [self.navigationController pushViewController:vc animated:YES];
    self.scanVC =  vc;
}
```

- 多码识别，持续扫码。

```
- (void)alertView:(UIAlertView *)alertView clickedButtonAtIndex:
(NSInteger)buttonIndex {
    // 持续扫码
    [self.scanVC resumeCaptureSession];
}
```

自定义 UI 的使用方式

本文将结合 扫一扫 官方 Demo 介绍如何在自定义 UI 下使用扫一扫多码识别 SDK。

自定义继承 TBScanViewController 的 ViewController

```
#import <UIKit/UIKit.h>

NS_ASSUME_NONNULL_BEGIN

@interface MPScanCodeViewController :
TBScanViewController<TBScanViewControllerDelegate>

@end

NS_ASSUME_NONNULL_END
```

初始化自定义扫码 ViewController

```
//自定义扫码入口
- (void)customScanAction
{
    MPScanCodeViewController *vc = [[MPScanCodeViewController alloc]
initWithConfig:@{}];
    [self.navigationController pushViewController:vc animated:YES];
}
```

```
@implementation MPScanCodeViewController

- (instancetype)initWithConfig:(NSDictionary *)config
{
    if (self = [super initWithConfig:config])
    {
        self.delegate = self;
        self.scanType = ScanType_All_Code;
    }
    return self;
}
```

> **重要**
>
> 初始化自定义扫码的 ViewController 只能使用 `-(instancetype)initWithConfig:(NSDictionary *)config;` 方式。

自定义扫码框

```
- (void)buildContainerView:(UIView*)containerView
{
    // 自定义扫码框 view
    UIView* bg = [[UIView alloc] initWithFrame:containerView.bounds];
    [containerView addSubview:bg];
    CGRect rect = [MPScanCodeViewController constructScanAnimationRect];
    UIView* view = [[UIView alloc] initWithFrame:rect];
    view.backgroundColor = [UIColor orangeColor];
    view.alpha = 0.5;
    [bg addSubview:view];
}
```

处理扫码结果

用户根据自己业务场景进行处理。

```
#pragma mark TBScanViewControllerDelegate

-(void)didFind:(NSArray<TBScanResult*>*)resultArray
{
    TBScanResult *result = resultArray.firstObject;
    NSString* content = result.data;
    if (result.resultType == TBScanResultTypeQRCode) {
        content = [NSString stringWithFormat:@"qrcode:%@, hiddenData:%@,
TBScanQRCodeResultType:%@", result.data, result.hiddenData, [result.extData
objectForKey:TBScanResultTypeQRCode]];
        NSLog(@"subType is %@, ScanType_QRCode is %@", @(result.subType),
@(ScanType_QRCode));
    } else if (result.resultType == TBScanResultTypeVLGen3Code) {
        content = [NSString stringWithFormat:@"gen3:%@", result.data];
        NSLog(@"subType is %@, ScanType_GEN3 is %@", @(result.subType),
@(ScanType_GEN3));
    } else if (result.resultType == TBScanResultTypeGoodsBarcode) {
        content = [NSString stringWithFormat:@"barcode:%@", result.data];
        NSLog(@"subType is %@, EAN13 is %@", @(result.subType), @(EAN13));
    } else if (result.resultType == TBScanResultTypeDataMatrixCode) {
        content = [NSString stringWithFormat:@"dm:%@", result.data];
        NSLog(@"subType is %@, ScanType_DATAMATRIX is %@", @(result.subType),
@(ScanType_DATAMATRIX));
    } else if (result.resultType == TBScanResultTypeExpressCode) {
        content = [NSString stringWithFormat:@"express:%@", result.data];
        NSLog(@"subType is %@, ScanType_FASTMAIL is %@", @(result.subType),
@(ScanType_FASTMAIL));
    }
    dispatch_async(dispatch_get_main_queue(), ^{
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"" message:content del
egate:self cancelButtonTitle:@"OK" otherButtonTitles:nil, nil];
        alert.tag = 9999;
        [alert show];
    });
}
```

1.4. 常见问题

本文介绍的是接入扫一扫过程中的常见问题。

扫一扫组件是否收费？

接入扫一扫组件不计费。但扫一扫组件中的日志埋点及日志上报功能依赖计费组件移动分析。例如在扫一扫组件中配置了日志埋点并开启日志上报功能，在使用过程中收集日志获取扫码次数、扫码成功次数、扫码失败次数等，用于对扫码性能进行监控和分析。根据移动分析组件的计费规则，会产生一定的费用，计费规则请参见 后付费模式；关闭该功能，则不产生费用，如需关闭，请参考文档 日志上报。

Android 工程使用原生 AAR 方式或 mPaaS Inside 方式接入时，如何初始化 mPaaS？

需要在 Application 中添加以下代码，若使用了热修复功能（QuinoxlessApplication），无需初始化 mPaaS。

```
public class MyApplication extends Application {
    @Override
    protected void attachBaseContext(Context base) {
        super.attachBaseContext(base);
        // mPaaS 初始化回调设置
        QuinoxlessFramework.setup(this, new IInitCallback() {
            @Override
            public void onPostInit() {
                        }
        });
    }
    @Override
    public void onCreate() {
        super.onCreate();
        // mPaaS 初始化
        QuinoxlessFramework.init();
    }
}
```

在 Android 10.1.68 基线中，启动扫码时卡死如何处理？

在 AAR 和 mPaaS Inside 模式下，如果您除了扫码组件还引用了其他组件，请进行 mPaaS 初始化，否则可能会导致主线程卡死。