一、二维码相关知识

1.1 一维码

所谓一维码，也就是条形码，超市里的条形码--这个相信大家都非常熟悉，**条形码实际上就是一串数字**，它上面存储了商品的序列号。

一维条码的宽度记载着数据，而其长度没有记载数据。

1.2 二维码

二维码又称二维条码，常见的二维码为 QR Code，QR 全称 Quick Response，是一个近几年来移动设备上超流行的一种编码式，它比传统的 Bar Code 条形码能存更多的信息，也能表示更多的数据类型。

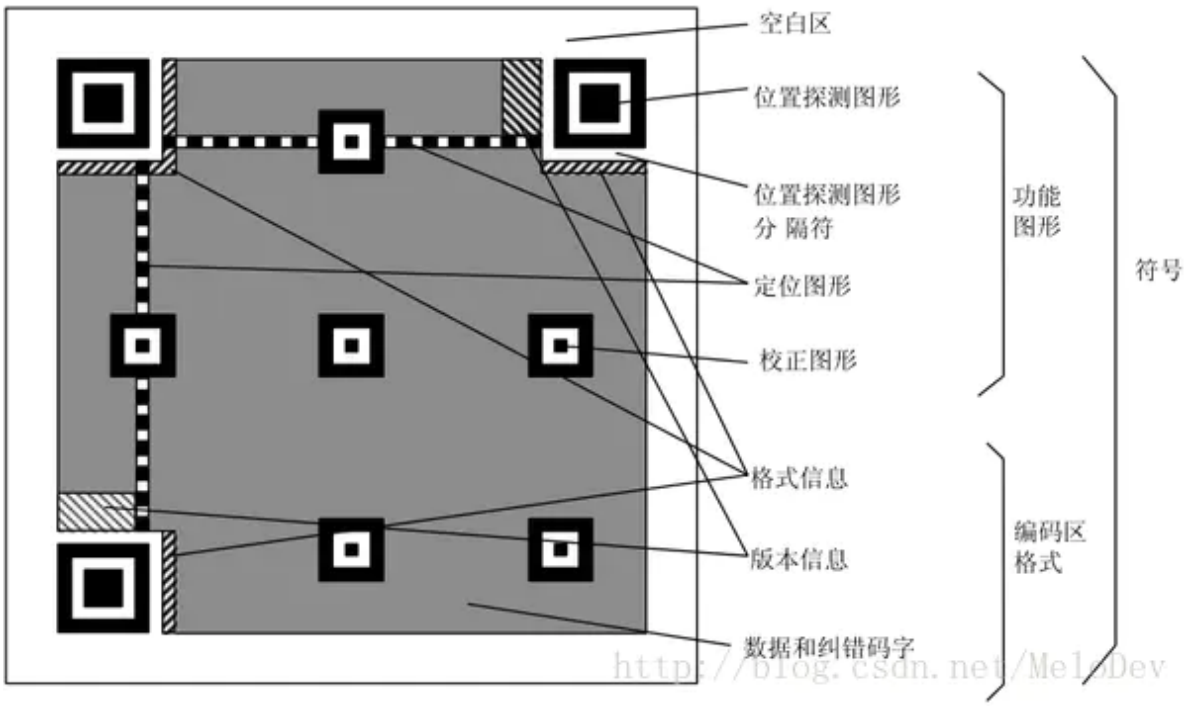
二维条码的长度、宽度均记载着数据，可以存储数字、字符串、图片、文件等，比如我们可以把 `www.baidu.com` 存储在二维码中，扫码二维码我们就可以获取到百度的地址。

二维条码的种类很多，不同的机构开发出的二维条码具有不同的结构以及编写、读取方法。

- 堆叠式/行排式二维条码，如，Code 16K、Code 49、PDF417（如下图）等。
- 矩阵式二维码，最流行莫过于 QR CODE，

1.3 二维码的结构

二维条码有一维条码没有的“**定位点**”和“**容错机制**”。容错机制在即使没有辨识到全部的条码、或是说条码有污损时，也可以正确地还原条码上的信息。



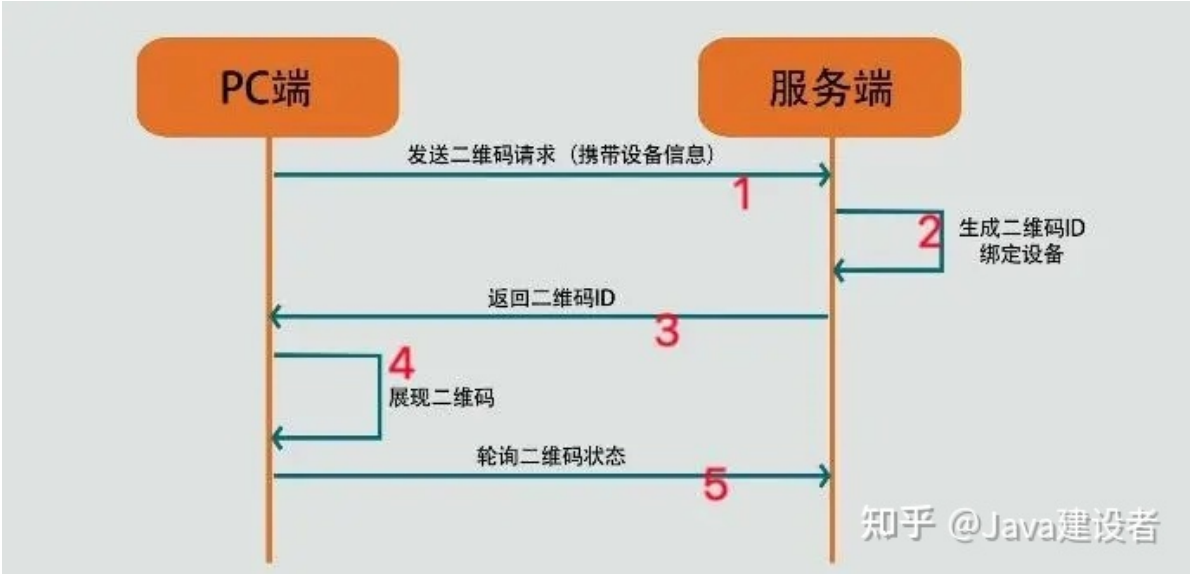
二维码一共有 40 个尺寸。官方叫版本 Version。

Version 1 是 21 x 21 的矩阵，Version 2 是 25 x 25 的矩阵，Version 3 是 29 的尺寸，每增加一个 version，就会增加 4 的尺寸，公式是：(V-1)*4 + 21（V是版本号） 最高 Version 40，(40-1)*4+21 = 177，所以最高是 177 x 177 的正方形。

二、手机扫码登录流程解析

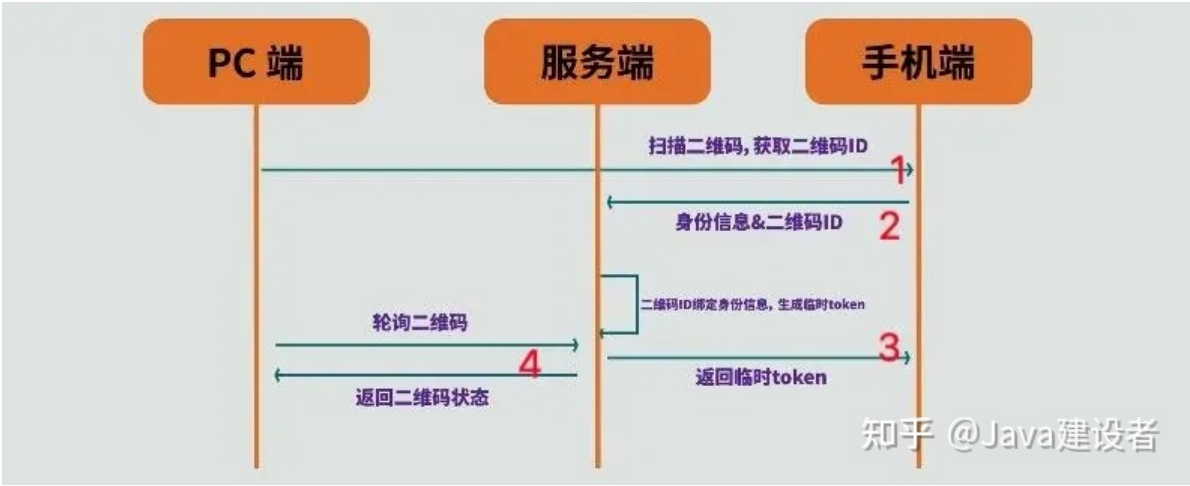
以手机端扫码登录PC端为例：

2.1 二维码准备



1. PC端向服务端发起请求，告诉服务端，我要生成用户登录的二维码，并且把PC端设备信息也传递给服务端
2. 服务端收到请求后，它生成二维码ID，并将二维码ID与PC端设备信息进行绑定
3. 然后把二维码ID返回给PC端
4. PC端收到二维码ID后，生成二维码(二维码中肯定包含了ID)
5. 为了及时知道二维码的状态，客户端在展现二维码后，PC端不断的轮询服务端，比如每隔一秒就轮询一次，请求服务端告诉当前二维码的状态及相关信息

2.2 扫描状态切换



1. 用户用手机去扫描PC端的二维码，通过二维码内容取到其中的二维码ID
2. 再调用服务端API将移动端的身份信息与二维码ID一起发送给服务端
3. 服务端接收到后，它可以将身份信息与二维码ID进行绑定，生成临时token。然后返回给手机端
4. 因为PC端一直在轮询二维码状态，所以这时候二维码状态发生了改变，它就可以在界面上把二维码状态更新为已扫描

为什么需要返回给手机端一个临时token呢？临时token与token一样，它也是一种身份凭证，不同的地方在于它只能用一次，用过就失效。

在第三步骤中返回临时 token，为的就是手机端在下一步操作时，可以用它作为凭证。以此确保扫码，登录两步操作是同一部手机端发出的。

2.3 状态确认

1. 手机端在接收到临时token后会弹出确认登录界面，用户点击确认时，手机端携带临时token用来调用服务端的接口，告诉服务端，我已经确认
2. 服务端收到确认后，根据二维码ID绑定的设备信息与账号信息，生成用户PC端登录的token
3. 这时候PC端的轮询接口，它就可以得知二维码的状态已经变成了"已确认"，并且从服务端可以获取到用户登录的token
4. 到这里，登录就成功了，后端PC端就可以用token去访问服务端的资源了

三、Android 生成二维码 + 扫描二维码功能

3.1 添加依赖库

```
implementation 'com.journeyapps:zxing-android-embedded:4.3.0'
```

集成第三方库 ZXing Android Embedded ，该库要求 Android SDK 24+

3.2 开启硬件加速

因为库中使用了 TextureView，所以要求打开硬件加速：

```
    <application android:hardwareAccelerated="true" ... >
```

如果项目中由于其他原因需要关闭硬件加速，则可以单独在扫描二维码的 Activity 上开启硬件加速。

例如：

```
        <activity
            android:name=".qrcode.CustomScanActivity"
            android:hardwareAccelerated="true"
            android:launchMode="singleTop" />
```

3.3 权限申请

需要申请摄像头权限。如果使用保存二维码到本地的功能，则还需要申请读写文件权限。

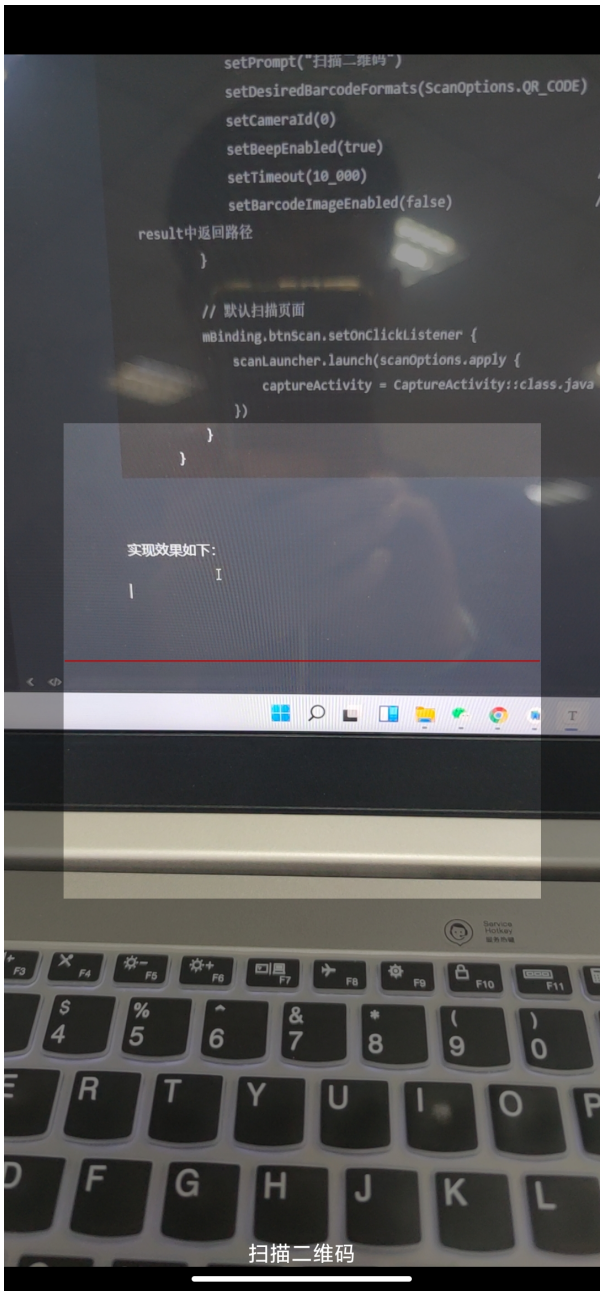
```
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

3.4 调用默认的扫描页面

3.4.1 调用代码

```
    @SuppressLint("SetTextI18n")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(mBinding.root)

        val scanLauncher = registerForActivityResult(
            ScanContract()
        ) { scanResult ->
            if (scanResult.contents == null) {
                Toast.makeText(this, "未扫描到结果", Toast.LENGTH_LONG).show()
            } else {
                "$scanResult".log()
                mBinding.tvScanResult.text = "扫描结果：\n ${scanResult.contents}"
            }
        }

        val scanOptions = ScanOptions().apply {
            setPrompt("扫描二维码")
            setDesiredBarcodeFormats(ScanOptions.QR_CODE)  // 图形码的格式：商品码、
一维码、二维码、数据矩阵、全部类型
            setCameraId(0)                                 // 0 后置摄像头  1 前置
摄像头(针对手机而言)
            setBeepEnabled(true)                           // 开启成功声音
            setTimeout(10_000)                             // 设置超时时间
            setBarcodeImageEnabled(false)                  // 是否保存图片，扫描成功
会截取扫描框的图形保存到手机并在result中返回路径
        }

        // 默认扫描页面
        mBinding.btnScan.setOnClickListener {
            scanLauncher.launch(scanOptions.apply {
                captureActivity = CaptureActivity::class.java
            })
        }
    }
```

3.4.2 效果图


setPrompt("扫描二维码")
setDesiredBarcodeFormats(ScanOptions.QR_CODE)
setCameraId(0)
setBeepEnabled(true)
setTimeout(10_000)
setBarcodeImageEnabled(false)
result中返回路径
}
// 默认扫描页面
mBinding.btnScan.setOnClickListener {
scanLauncher.launch(scanOptions.apply {
captureActivity = CaptureActivity::class.java
})
}
}
实现效果如下:
扫描二维码

3.5 自定义扫描页面

通过查看该库的示例和源码，自定义扫描页面需要一下几个步骤：

1. 根据需求自定义遮罩层 View
2. 创建一个 xml 布局并引入库中的组件和自定义的遮罩层布局
3. 创建一个新的 Activity，加载第二步中的 xml 布局，处理初始化操作和声明周期相关的操作
4. 调用时只需要把默认的扫描页面更改为第二步中的 Activity 即可

3.5.1 自定义遮罩层

```
public class CustomViewfinderView extends ViewfinderView {

    //重绘时间间隔
    public static final long INT_ANIMATION_DELAY = 12;

    /* ****************************************     边角线相关属性
************************************************/
```

```
    //"边角线长度/扫描边框长度"的占比 (比例越大，线越长)
    public float mLineRate = 0.1F;
    //边角线厚度 (建议使用dp)
    public float mLineDepth = dp2px(4);
    //边角线颜色
    public int mLineColor;

    /* ****************************************    扫描线相关属性
**********************************************/
    //扫描线起始位置
    public int mScanLinePosition = 0;
    //扫描线厚度
    public float mScanLineDepth = dp2px(4);
    //扫描线每次移动距离
    public float mScanLineDy = dp2px(3);
    //渐变线
    public LinearGradient mLinearGradient;
    //图形paint
    public Paint mBitmapPaint;
    ///颜色在渐变中所占比例，此处均衡渐变
    public float[] mPositions = new float[]{0f, 0.5f, 1f};
    //线性梯度各个位置对应的颜色值
    public int[] mScanLineColor = new int[]{0x00000000, Color.WHITE,
0x00000000};

    //扫描框宽、高
    public float mScanFrameWidth;
    public float mScanFrameHeight;

    public CustomViewfinderView(Context context, AttributeSet attrs) {
        super(context, attrs);
        TypedArray typedArray = context.obtainStyledAttributes(attrs,
R.styleable.CustomViewfinderView);
        mLineColor =
typedArray.getColor(R.styleable.CustomViewfinderView_lineColor, Color.YELLOW);
        mScanLineColor[1] =
typedArray.getColor(R.styleable.CustomViewfinderView_cornerColor, Color.YELLOW);
        mScanFrameWidth =
typedArray.getDimension(R.styleable.CustomViewfinderView_scanFrameWidth,
dp2px(160));
        mScanFrameHeight =
typedArray.getDimension(R.styleable.CustomViewfinderView_scanFrameHeight,
dp2px(160));
        typedArray.recycle();
        mBitmapPaint = new Paint();
        mBitmapPaint.setAntiAlias(true);
    }

    @SuppressLint({"DrawAllocation"})
    @Override
    public void onDraw(Canvas canvas) {
        refreshSizes();
        if (framingRect == null || previewSize == null) {
            return;
        }

        final Rect frame = framingRect;
```

```
        final int width = getWidth();
        final int height = getHeight();

        //绘制扫描框外部遮罩
        paint.setColor(resultBitmap != null ? resultColor : maskColor);
        canvas.drawRect(0, 0, width, frame.top, paint);
        canvas.drawRect(0, frame.top, frame.left, frame.bottom + 1, paint);
        canvas.drawRect(frame.right + 1, frame.top, width, frame.bottom + 1,
paint);
        canvas.drawRect(0, frame.bottom + 1, width, height, paint);

        //绘制4个角
        paint.setColor(mLineColor);
        canvas.drawRect(frame.left, frame.top, frame.left + frame.width() *
mLineRate, frame.top + mLineDepth, paint);
        canvas.drawRect(frame.left, frame.top, frame.left + mLineDepth, frame.top
+ frame.height() * mLineRate, paint);

        canvas.drawRect(frame.right - frame.width() * mLineRate, frame.top,
frame.right, frame.top + mLineDepth, paint);
        canvas.drawRect(frame.right - mLineDepth, frame.top, frame.right,
frame.top + frame.height() * mLineRate, paint);

        canvas.drawRect(frame.left, frame.bottom - mLineDepth, frame.left +
frame.width() * mLineRate, frame.bottom, paint);
        canvas.drawRect(frame.left, frame.bottom - frame.height() * mLineRate,
frame.left + mLineDepth, frame.bottom, paint);

        canvas.drawRect(frame.right - frame.width() * mLineRate, frame.bottom -
mLineDepth, frame.right, frame.bottom, paint);
        canvas.drawRect(frame.right - mLineDepth, frame.bottom - frame.height() *
mLineRate, frame.right, frame.bottom, paint);


        if (resultBitmap != null) {
            // Draw the opaque result bitmap over the scanning rectangle
            paint.setAlpha(CURRENT_POINT_OPACITY);
            canvas.drawBitmap(resultBitmap, null, frame, paint);
        } else {
            // 绘制渐变扫描线
            mScanLinePosition += mScanLineDy;
            if (mScanLinePosition >= frame.height()) {
                mScanLinePosition = 0;
            }
            mLinearGradient = new LinearGradient(frame.left, frame.top +
mScanLinePosition, frame.right, frame.top + mScanLinePosition, mScanLineColor,
mPositions, Shader.TileMode.CLAMP);
            paint.setShader(mLinearGradient);
            canvas.drawRect(frame.left, frame.top + mScanLinePosition,
frame.right, frame.top + mScanLinePosition + mScanLineDepth, paint);
            paint.setShader(null);

            //绘制资源图片扫描线
//            Rect lineRect = new Rect();
//            lineRect.left = frame.left;
//            lineRect.top = frame.top + mScanLinePosition;
//            lineRect.right = frame.right;
//            lineRect.bottom = frame.top + dp2px(6) + mScanLinePosition;
```

```
//            Bitmap bitmap =
BitmapFactory.decodeResource(getResources(),R.drawable.img_line);
//            canvas.drawBitmap(bitmap, null, lineRect, mBitmapPaint);


            //============绘制扫描时小圆点，效果为默认=======================

            final float scaleX = this.getWidth() / (float) previewSize.width;
            final float scaleY = this.getHeight() / (float) previewSize.height;
            // draw the last possible result points
            if (!lastPossibleResultPoints.isEmpty()) {
                paint.setAlpha(CURRENT_POINT_OPACITY / 2);
                paint.setColor(resultPointColor);
                float radius = POINT_SIZE / 2.0f;
                for (final ResultPoint point : lastPossibleResultPoints) {
                    canvas.drawCircle(
                            (int) (point.getX() * scaleX),
                            (int) (point.getY() * scaleY),
                            radius, paint
                    );
                }
                lastPossibleResultPoints.clear();
            }

            // draw current possible result points
            if (!possibleResultPoints.isEmpty()) {
                paint.setAlpha(CURRENT_POINT_OPACITY);
                paint.setColor(resultPointColor);
                for (final ResultPoint point : possibleResultPoints) {
                    canvas.drawCircle(
                            (int) (point.getX() * scaleX),
                            (int) (point.getY() * scaleY),
                            POINT_SIZE, paint
                    );
                }

                // swap and clear buffers
                final List<ResultPoint> temp = possibleResultPoints;
                possibleResultPoints = lastPossibleResultPoints;
                lastPossibleResultPoints = temp;
                possibleResultPoints.clear();
            }

            //============绘制扫描时小圆点，效果为默认 end=======================
        }

        //定时刷新扫描框
        postInvalidateDelayed(INT_ANIMATION_DELAY,
                frame.left - POINT_SIZE,
                frame.top - POINT_SIZE,
                frame.right + POINT_SIZE,
                frame.bottom + POINT_SIZE);

    }

    protected void refreshSizes() {
        if (cameraPreview == null) {
            return;
```

```
        }
        //添加设置边框大小代码
        cameraPreview.setFramingRectSize(new Size((int) mScanFrameWidth, (int)
mScanFrameHeight));

        Rect framingRect = cameraPreview.getFramingRect();
        Size previewSize = cameraPreview.getPreviewSize();
        if (framingRect != null && previewSize != null) {
            this.framingRect = framingRect;
            this.previewSize = previewSize;
        }
    }

    private int dp2px(int dp) {
        float density = getContext().getResources().getDisplayMetrics().density;
        return (int) (dp * density + 0.5f);
    }
}
```

3.5.2 创建 xml 布局

custom_barcode_scanner.xml

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

    <com.journeyapps.barcodescanner.BarcodeView
        android:id="@+id/zxing_barcode_surface"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <com.cs.android.qrcode.CustomViewfinderView
        android:id="@+id/zxing_viewfinder_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:cornerColor="@color/WHITE"
        app:lineColor="@color/WHITE"
        app:scanFrameHeight="180dp"
        app:scanFrameWidth="180dp" />

    <TextView
        android:id="@+id/zxing_status_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|center_horizontal"
        android:background="@color/zxing_transparent"
        android:text="@string/zxing_msg_default_status"
        android:textColor="@color/zxing_status_text" />
</merge>
```

activity_custom_scan.xml

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

    <com.journeyapps.barcodescanner.DecoratedBarcodeView
        android:id="@+id/decorateBarcodeView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:zxing_scanner_layout="@layout/custom_barcode_scanner" />

</FrameLayout>
```

3.5.3 创建 Activity

```
class CustomScanActivity : BaseActivity() {
    private val mBinding: ActivityCustomScanBinding by lazy {
        ActivityCustomScanBinding.inflate(layoutInflater)
    }

    private lateinit var capture: CaptureManager

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(mBinding.root)

        capture = CaptureManager(this, mBinding.decorateBarcodeView).apply {
            initializeFromIntent(intent, savedInstanceState)
            decode()
        }
    }

    override fun onResume() {
        super.onResume()
        capture.onResume()
    }

    override fun onPause() {
        super.onPause()
        capture.onPause()
    }

    override fun onDestroy() {
        super.onDestroy()
        capture.onDestroy()
    }

    override fun onSaveInstanceState(outState: Bundle) {
        super.onSaveInstanceState(outState)
        capture.onSaveInstanceState(outState)
    }

    override fun onRequestPermissionsResult(
```

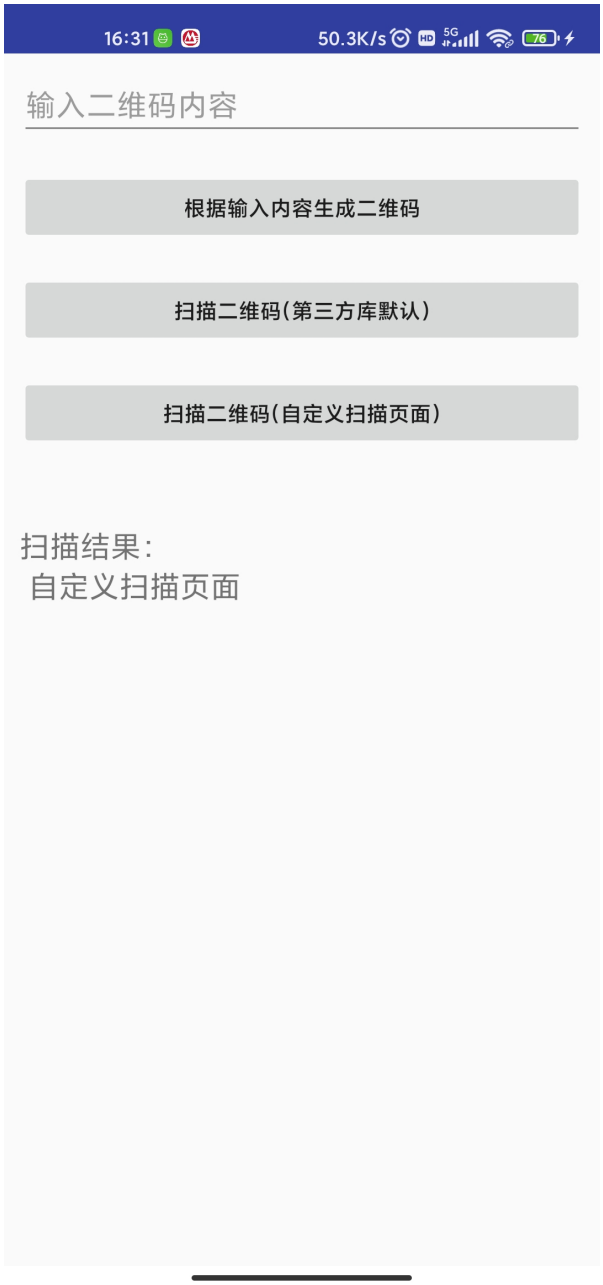
```
        requestCode: Int,
        permissions: Array<out String>,
        grantResults: IntArray
    ) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults)
        capture.onRequestPermissionsResult(requestCode, permissions,
grantResults)
    }

    override fun onKeyDown(keyCode: Int, event: KeyEvent?): Boolean {
        return mBinding.decorateBarcodeView.onKeyDown(keyCode, event) ||
super.onKeyDown(keyCode, event)
    }
}
```

3.5.4 设置扫描时使用自定义页面

```
    @SuppressLint("SetTextI18n")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(mBinding.root)


        val scanLauncher = registerForActivityResult(
            ScanContract()
        ) { scanResult ->
            if (scanResult.contents == null) {
                Toast.makeText(this, "未扫描到结果", Toast.LENGTH_LONG).show()
            } else {
                "$scanResult".log()
                mBinding.tvScanResult.text = "扫描结果：\n ${scanResult.contents}"
            }
        }

        val scanOptions = ScanOptions().apply {
            setPrompt("扫描二维码")
            setDesiredBarcodeFormats(ScanOptions.QR_CODE)  // 图形码的格式：商品码、
一维码、二维码、数据矩阵、全部类型
            setCameraId(0)                                 // 0 后置摄像头  1 前置
摄像头(针对手机而言)
            setBeepEnabled(true)                           // 开启成功声音
            setTimeout(10_000)                             // 设置超时时间
            setBarcodeImageEnabled(false)                  // 是否保存图片，扫描成功
会截取扫描框的图形保存到手机并在result中返回路径
        }

        // 自定义扫描页面
        mBinding.btnScanCustom.setOnClickListener {
            scanLauncher.launch(scanOptions.apply {
                captureActivity = CustomScanActivity::class.java
            })
        }
    }
```

3.5.5 效果图


16:30
43.5K/s
scanLauncher.launch(s
captureActivity =
})
}
}
实现效果如下：
扫描二维码



3.6 生成二维码

3.6.1 相关代码

```
    /**
     * 生成二维码
     *
     * No customization of the image is currently supported, including changing
colors or padding.
     * If you require more customization, copy and modify the source for the
encoder.
     */
    private fun generateBarcode(content: String): Bitmap? {
        val barcodeEncoder = BarcodeEncoder()
        return try {
```

```
            barcodeEncoder.encodeBitmap(content, BarcodeFormat.QR_CODE, 500,
500)
        } catch (e: Exception) {
            e.printStackTrace()
            null
        }
    }
```

3.6.2 效果图