**package** com.google.zxing.activity;  
  
**import** android.app.Activity;  
**import** android.app.ProgressDialog;  
**import** android.content.Context;  
**import** android.content.Intent;  
**import** android.content.res.AssetFileDescriptor;  
**import** android.graphics.Bitmap;  
**import** android.media.AudioManager;  
**import** android.media.MediaPlayer;  
**import** android.media.MediaPlayer.OnCompletionListener;  
**import** android.net.Uri;  
**import** android.os.Bundle;  
**import** android.os.Handler;  
**import** android.os.Vibrator;  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.text.TextUtils;  
**import** android.view.SurfaceHolder;  
**import** android.view.SurfaceHolder.Callback;  
**import** android.view.SurfaceView;  
**import** android.view.View;  
**import** android.widget.Button;  
**import** android.widget.ImageButton;  
**import** android.widget.Toast;  
  
**import** com.google.zxing.BarcodeFormat;  
**import** com.google.zxing.BinaryBitmap;  
**import** com.google.zxing.ChecksumException;  
**import** com.google.zxing.DecodeHintType;  
**import** com.google.zxing.FormatException;  
**import** com.google.zxing.NotFoundException;  
**import** com.google.zxing.R;  
**import** com.google.zxing.Result;  
**import** com.google.zxing.camera.CameraManager;  
**import** com.google.zxing.common.HybridBinarizer;  
**import** com.google.zxing.decoding.CaptureActivityHandler;  
**import** com.google.zxing.decoding.InactivityTimer;  
**import** com.google.zxing.decoding.RGBLuminanceSource;  
**import** com.google.zxing.qrcode.QRCodeReader;  
**import** com.google.zxing.util.BitmapUtil;  
**import** com.google.zxing.util.Constant;  
**import** com.google.zxing.view.ViewfinderView;  
**import** com.gyf.barlibrary.ImmersionBar;  
  
**import** java.io.IOException;  
**import** java.util.Hashtable;  
**import** java.util.Vector;  
  
  
*/\*\*  
 \* Initial the camera  
 \*  
 \** ***@author*** *Ryan.Tang  
 \*/***public class** CaptureActivity **extends** AppCompatActivity **implements** Callback {  
  
 **private static final int *REQUEST\_CODE\_SCAN\_GALLERY*** = 100;  
  
 **private** CaptureActivityHandler **handler**;  
 **private** ViewfinderView **viewfinderView**;  
 **private** ImageButton **back**;  
 **private** ImageButton **btnFlash**;  
 **private** Button **btnAlbum**; *// 相册* **private boolean isFlashOn** = **false**;  
 **private boolean hasSurface**;  
 **private** Vector<BarcodeFormat> **decodeFormats**;  
 **private** String **characterSet**;  
 **private** InactivityTimer **inactivityTimer**;  
 **private** MediaPlayer **mediaPlayer**;  
 **private boolean playBeep**;  
 **private static final float *BEEP\_VOLUME*** = 0.10f;  
 **private boolean vibrate**;  
 **private** ProgressDialog **mProgress**;  
 **private** Bitmap **scanBitmap**;  
 **private** ImmersionBar **mImmersionBar**;  
  
 **public static void** start(Activity context) {  
 Intent view = **new** Intent(context, CaptureActivity.**class**);  
 context.startActivityForResult(view, Constant.***REQ\_QR\_CODE***);  
 }  
  
 @Override  
 **public void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 **mImmersionBar** = ImmersionBar.with(**this**).statusBarDarkFont(**true**).keyboardEnable(**true**);  
 **mImmersionBar**.init();  
 setContentView(R.layout.activity\_scanner);  
 CameraManager.*init*(getApplication());  
 **viewfinderView** = (ViewfinderView) findViewById(R.id.viewfinder\_content);  
 **back** = (ImageButton) findViewById(R.id.btn\_back);  
 **back**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 finish();  
 }  
 });  
  
 **btnFlash** = (ImageButton) findViewById(R.id.btn\_flash);  
 **btnFlash**.setOnClickListener(**flashListener**);  
  
 **btnAlbum** = (Button) findViewById(R.id.btn\_album);  
 **btnAlbum**.setOnClickListener(**albumOnClick**);  
  
 **hasSurface** = **false**;  
 **inactivityTimer** = **new** InactivityTimer(**this**);  
  
 }  
  
 **private** View.OnClickListener **albumOnClick** = **new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 *//打开手机中的相册* Intent innerIntent = **new** Intent(Intent.ACTION\_GET\_CONTENT); *//"android.intent.action.GET\_CONTENT"* innerIntent.setType(**"image/\*"**);  
 startActivityForResult(innerIntent, ***REQUEST\_CODE\_SCAN\_GALLERY***);  
 }  
 };  
  
 @Override  
 **protected void** onActivityResult(**final int** requestCode, **int** resultCode, Intent data) {  
 **if** (resultCode == RESULT\_OK) {  
 **switch** (requestCode) {  
 **case *REQUEST\_CODE\_SCAN\_GALLERY***:  
 handleAlbumPic(data);  
 **break**;  
 }  
 }  
 **super**.onActivityResult(requestCode, resultCode, data);  
 }  
  
 */\*\*  
 \* 处理选择的图片  
 \*  
 \** ***@param data*** *\*/* **private void** handleAlbumPic(Intent data) {  
 *//获取选中图片的路径* **final** Uri uri = data.getData();  
  
 **mProgress** = **new** ProgressDialog(CaptureActivity.**this**);  
 **mProgress**.setMessage(**"正在扫描..."**);  
 **mProgress**.setCancelable(**false**);  
 **mProgress**.show();  
 runOnUiThread(**new** Runnable() {  
 @Override  
 **public void** run() {  
 Result result = scanningImage(uri);  
 **mProgress**.dismiss();  
 **if** (result != **null**) {  
 Intent resultIntent = **new** Intent();  
 Bundle bundle = getIntent().getExtras();  
 **if** (bundle == **null**) {  
 bundle = **new** Bundle();  
 }  
 bundle.putString(Constant.***INTENT\_EXTRA\_KEY\_QR\_SCAN***, result.getText());  
  
 resultIntent.putExtras(bundle);  
 CaptureActivity.**this**.setResult(RESULT\_OK, resultIntent);  
 finish();  
 } **else** {  
 Toast.makeText(CaptureActivity.**this**, R.string.note\_identify\_failed, Toast.LENGTH\_SHORT).show();  
 }  
 }  
 });  
 }  
  
 */\*\*  
 \* 扫描二维码图片的方法  
 \*  
 \** ***@param uri*** *\** ***@return*** *\*/* **public** Result scanningImage(Uri uri) {  
 **if** (uri == **null**) {  
 **return null**;  
 }  
 Hashtable<DecodeHintType, String> hints = **new** Hashtable<>();  
 hints.put(DecodeHintType.CHARACTER\_SET, **"UTF8"**); *//设置二维码内容的编码* **scanBitmap** = BitmapUtil.*decodeUri*(**this**, uri, 500, 500);  
 RGBLuminanceSource source = **new** RGBLuminanceSource(**scanBitmap**);  
 BinaryBitmap bitmap1 = **new** BinaryBitmap(**new** HybridBinarizer(source));  
 QRCodeReader reader = **new** QRCodeReader();  
 **try** {  
 **return** reader.decode(bitmap1, hints);  
 } **catch** (NotFoundException e) {  
 e.printStackTrace();  
 } **catch** (ChecksumException e) {  
 e.printStackTrace();  
 } **catch** (FormatException e) {  
 e.printStackTrace();  
 }  
 **return null**;  
 }  
  
 @Override  
 **protected void** onResume() {  
 **super**.onResume();  
 SurfaceView surfaceView = (SurfaceView) findViewById(R.id.scanner\_view);  
 SurfaceHolder surfaceHolder = surfaceView.getHolder();  
 **if** (**hasSurface**) {  
 initCamera(surfaceHolder);  
 } **else** {  
 surfaceHolder.addCallback(**this**);  
 surfaceHolder.setType(SurfaceHolder.SURFACE\_TYPE\_PUSH\_BUFFERS);  
 }  
 **decodeFormats** = **null**;  
 **characterSet** = **null**;  
  
 **playBeep** = **true**;  
 AudioManager audioService = (AudioManager) getSystemService(AUDIO\_SERVICE);  
 **if** (audioService.getRingerMode() != AudioManager.RINGER\_MODE\_NORMAL) {  
 **playBeep** = **false**;  
 }  
 initBeepSound();  
 **vibrate** = **true**;  
  
 }  
  
 @Override  
 **protected void** onPause() {  
 **super**.onPause();  
 **if** (**handler** != **null**) {  
 **handler**.quitSynchronously();  
 **handler** = **null**;  
 }  
 CameraManager.*get*().closeDriver();  
 }  
  
 @Override  
 **protected void** onDestroy() {  
 **inactivityTimer**.shutdown();  
 **if** (**mImmersionBar** != **null**)  
 **mImmersionBar**.destroy();  
 **super**.onDestroy();  
  
 }  
  
 */\*\*  
 \* Handler scan result  
 \*  
 \** ***@param result*** *\** ***@param barcode*** *\*/* **public void** handleDecode(Result result, Bitmap barcode) {  
 **inactivityTimer**.onActivity();  
 playBeepSoundAndVibrate();  
 String resultString = result.getText();  
 **if** (TextUtils.isEmpty(resultString)) {  
 Toast.makeText(CaptureActivity.**this**, R.string.note\_scan\_failed, Toast.LENGTH\_SHORT).show();  
 } **else** {  
 Intent resultIntent = **new** Intent();  
 Bundle bundle = getIntent().getExtras();  
 **if** (bundle == **null**) {  
 bundle = **new** Bundle();  
 }  
 bundle.putString(Constant.***INTENT\_EXTRA\_KEY\_QR\_SCAN***, resultString);  
 resultIntent.putExtras(bundle);  
 **this**.setResult(RESULT\_OK, resultIntent);  
 }  
 CaptureActivity.**this**.finish();  
 }  
  
 **private void** initCamera(SurfaceHolder surfaceHolder) {  
 **try** {  
 CameraManager.*get*().openDriver(surfaceHolder);  
 } **catch** (IOException ioe) {  
 **return**;  
 } **catch** (RuntimeException e) {  
 **return**;  
 }  
 **if** (**handler** == **null**) {  
 **handler** = **new** CaptureActivityHandler(**this**, **decodeFormats**,  
 **characterSet**);  
 }  
 }  
  
 @Override  
 **public void** surfaceChanged(SurfaceHolder holder, **int** format, **int** width,  
 **int** height) {  
  
 }  
  
 @Override  
 **public void** surfaceCreated(SurfaceHolder holder) {  
 **if** (!**hasSurface**) {  
 **hasSurface** = **true**;  
 initCamera(holder);  
 }  
  
 }  
  
 @Override  
 **public void** surfaceDestroyed(SurfaceHolder holder) {  
 **hasSurface** = **false**;  
  
 }  
  
 **public** ViewfinderView getViewfinderView() {  
 **return viewfinderView**;  
 }  
  
 **public** Handler getHandler() {  
 **return handler**;  
 }  
  
 **public void** drawViewfinder() {  
 **viewfinderView**.drawViewfinder();  
  
 }  
  
 **private void** initBeepSound() {  
 **if** (**playBeep** && **mediaPlayer** == **null**) {  
 *// The volume on STREAM\_SYSTEM is not adjustable, and users found it  
 // too loud,  
 // so we now play on the music stream.* setVolumeControlStream(AudioManager.STREAM\_MUSIC);  
 **mediaPlayer** = **new** MediaPlayer();  
 **mediaPlayer**.setAudioStreamType(AudioManager.STREAM\_MUSIC);  
 **mediaPlayer**.setOnCompletionListener(**beepListener**);  
  
 AssetFileDescriptor file = getResources().openRawResourceFd(  
 R.raw.beep);  
 **try** {  
 **mediaPlayer**.setDataSource(file.getFileDescriptor(),  
 file.getStartOffset(), file.getLength());  
 file.close();  
 **mediaPlayer**.setVolume(***BEEP\_VOLUME***, ***BEEP\_VOLUME***);  
 **mediaPlayer**.prepare();  
 } **catch** (IOException e) {  
 **mediaPlayer** = **null**;  
 }  
 }  
 }  
  
 **private static final long *VIBRATE\_DURATION*** = 200L;  
  
 **private void** playBeepSoundAndVibrate() {  
 **if** (**playBeep** && **mediaPlayer** != **null**) {  
 **mediaPlayer**.start();  
 }  
 **if** (**vibrate**) {  
 Vibrator vibrator = (Vibrator) getSystemService(VIBRATOR\_SERVICE);  
 vibrator.vibrate(***VIBRATE\_DURATION***);  
 }  
 }  
  
 */\*\*  
 \* When the beep has finished playing, rewind to queue up another one.  
 \*/* **private final** OnCompletionListener **beepListener** = **new** OnCompletionListener() {  
 @Override  
 **public void** onCompletion(MediaPlayer mediaPlayer) {  
 mediaPlayer.seekTo(0);  
 }  
 };  
  
 */\*\*  
 \* 闪光灯开关按钮  
 \*/* **private** View.OnClickListener **flashListener** = **new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 **try** {  
 **boolean** isSuccess = CameraManager.*get*().setFlashLight(!**isFlashOn**);  
 **if** (!isSuccess) {  
 Toast.makeText(CaptureActivity.**this**, R.string.note\_no\_flashlight, Toast.LENGTH\_SHORT).show();  
 **return**;  
 }  
 **if** (**isFlashOn**) {  
 *// 关闭闪光灯* **btnFlash**.setImageResource(R.drawable.flash\_off);  
 **isFlashOn** = **false**;  
 } **else** {  
 *// 开启闪光灯* **btnFlash**.setImageResource(R.drawable.flash\_on);  
 **isFlashOn** = **true**;  
 }  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 };  
}

*/\*  
 \* Copyright (C) 2008 ZXing authors  
 \*  
 \* Licensed under the Apache License, Version 2.0 (the "License");  
 \* you may not use this file except in compliance with the License.  
 \* You may obtain a copy of the License at  
 \*  
 \* http://www.apache.org/licenses/LICENSE-2.0  
 \*  
 \* Unless required by applicable law or agreed to in writing, software  
 \* distributed under the License is distributed on an "AS IS" BASIS,  
 \* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
 \* See the License for the specific language governing permissions and  
 \* limitations under the License.  
 \*/***package** com.google.zxing.decoding;  
  
**import** android.app.Activity;  
**import** android.content.Intent;  
**import** android.graphics.Bitmap;  
**import** android.net.Uri;  
**import** android.os.Bundle;  
**import** android.os.Handler;  
**import** android.os.Message;  
**import** android.util.Log;  
  
**import** com.google.zxing.BarcodeFormat;  
**import** com.google.zxing.R;  
**import** com.google.zxing.Result;  
**import** com.google.zxing.activity.CaptureActivity;  
**import** com.google.zxing.camera.CameraManager;  
**import** com.google.zxing.view.ViewfinderResultPointCallback;  
  
**import** java.util.Vector;  
  
  
*/\*\*  
 \* This class handles all the messaging which comprises the state machine for capture.  
 \*/***public final class** CaptureActivityHandler **extends** Handler {  
  
 **private static final** String ***TAG*** = CaptureActivityHandler.**class**.getSimpleName();  
  
 **private final** CaptureActivity **activity**;  
 **private final** DecodeThread **decodeThread**;  
 **private** State **state**;  
  
 **private enum** State {  
 ***PREVIEW***,  
 ***SUCCESS***,  
 ***DONE*** }  
  
 **public** CaptureActivityHandler(CaptureActivity activity, Vector<BarcodeFormat> decodeFormats,  
 String characterSet) {  
 **this**.**activity** = activity;  
 **decodeThread** = **new** DecodeThread(activity, decodeFormats, characterSet,  
 **new** ViewfinderResultPointCallback(activity.getViewfinderView()));  
 **decodeThread**.start();  
 **state** = State.***SUCCESS***;  
 *// Start ourselves capturing previews and decoding.* CameraManager.*get*().startPreview();  
 restartPreviewAndDecode();  
 }  
  
 @Override  
 **public void** handleMessage(Message message) {  
 **int** id = message.what;  
 **if** (id == R.id.auto\_focus) {  
 *//Log.d(TAG, "Got auto-focus message");  
 // When one auto focus pass finishes, start another. This is the closest thing to  
 // continuous AF. It does seem to hunt a bit, but I'm not sure what else to do.* **if** (**state** == State.***PREVIEW***) {  
 CameraManager.*get*().requestAutoFocus(**this**, R.id.auto\_focus);  
 }  
 } **else if** (id == R.id.restart\_preview) {  
 Log.d(***TAG***, **"Got restart preview message"**);  
 restartPreviewAndDecode();  
 } **else if** (id == R.id.decode\_succeeded) {  
 Log.d(***TAG***, **"Got decode succeeded message"**);  
 **state** = State.***SUCCESS***;  
 Bundle bundle = message.getData();  
  
 */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/* Bitmap barcode = bundle == **null** ? **null** :  
 (Bitmap) bundle.getParcelable(DecodeThread.***BARCODE\_BITMAP***);*//���ñ����߳�* **activity**.handleDecode((Result) message.obj, barcode);*//���ؽ��  
 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/* } **else if** (id == R.id.decode\_failed) {  
 *// We're decoding as fast as possible, so when one decode fails, start another.* **state** = State.***PREVIEW***;  
 CameraManager.*get*().requestPreviewFrame(**decodeThread**.getHandler(), R.id.decode);  
 } **else if** (id == R.id.return\_scan\_result) {  
 Log.d(***TAG***, **"Got return scan result message"**);  
 **activity**.setResult(Activity.RESULT\_OK, (Intent) message.obj);  
 **activity**.finish();  
 } **else if** (id == R.id.launch\_product\_query) {  
 Log.d(***TAG***, **"Got product query message"**);  
 String url = (String) message.obj;  
 Intent intent = **new** Intent(Intent.ACTION\_VIEW, Uri.parse(url));  
 intent.addFlags(Intent.FLAG\_ACTIVITY\_CLEAR\_WHEN\_TASK\_RESET);  
 **activity**.startActivity(intent);  
 }  
 }  
  
 **public void** quitSynchronously() {  
 **state** = State.***DONE***;  
 CameraManager.*get*().stopPreview();  
 Message quit = Message.obtain(**decodeThread**.getHandler(), R.id.quit);  
 quit.sendToTarget();  
 **try** {  
 **decodeThread**.join();  
 } **catch** (InterruptedException e) {  
 *// continue* }  
  
 *// Be absolutely sure we don't send any queued up messages* removeMessages(R.id.decode\_succeeded);  
 removeMessages(R.id.decode\_failed);  
 }  
  
 **private void** restartPreviewAndDecode() {  
 **if** (**state** == State.***SUCCESS***) {  
 **state** = State.***PREVIEW***;  
 CameraManager.*get*().requestPreviewFrame(**decodeThread**.getHandler(), R.id.decode);  
 CameraManager.*get*().requestAutoFocus(**this**, R.id.auto\_focus);  
 **activity**.drawViewfinder();  
 }  
 }  
  
}

*/\*  
 \* Copyright (C) 2010 ZXing authors  
 \*  
 \* Licensed under the Apache License, Version 2.0 (the "License");  
 \* you may not use this file except in compliance with the License.  
 \* You may obtain a copy of the License at  
 \*  
 \* http://www.apache.org/licenses/LICENSE-2.0  
 \*  
 \* Unless required by applicable law or agreed to in writing, software  
 \* distributed under the License is distributed on an "AS IS" BASIS,  
 \* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
 \* See the License for the specific language governing permissions and  
 \* limitations under the License.  
 \*/***package** com.google.zxing.decoding;  
  
**import** android.content.Intent;  
**import** android.net.Uri;  
  
**import** com.google.zxing.BarcodeFormat;  
  
**import** java.util.Arrays;  
**import** java.util.List;  
**import** java.util.Vector;  
**import** java.util.regex.Pattern;  
  
**final class** DecodeFormatManager {  
  
 **private static final** Pattern ***COMMA\_PATTERN*** = Pattern.compile(**","**);  
  
 **static final** Vector<BarcodeFormat> ***PRODUCT\_FORMATS***;  
 **static final** Vector<BarcodeFormat> ***ONE\_D\_FORMATS***;  
 **static final** Vector<BarcodeFormat> ***QR\_CODE\_FORMATS***;  
 **static final** Vector<BarcodeFormat> ***DATA\_MATRIX\_FORMATS***;  
  
 **static** {  
 ***PRODUCT\_FORMATS*** = **new** Vector<BarcodeFormat>(5);  
 ***PRODUCT\_FORMATS***.add(BarcodeFormat.UPC\_A);  
 ***PRODUCT\_FORMATS***.add(BarcodeFormat.UPC\_E);  
 ***PRODUCT\_FORMATS***.add(BarcodeFormat.EAN\_13);  
 ***PRODUCT\_FORMATS***.add(BarcodeFormat.EAN\_8);  
 ***ONE\_D\_FORMATS*** = **new** Vector<BarcodeFormat>(***PRODUCT\_FORMATS***.size() + 4);  
 ***ONE\_D\_FORMATS***.addAll(***PRODUCT\_FORMATS***);  
 ***ONE\_D\_FORMATS***.add(BarcodeFormat.CODE\_39);  
 ***ONE\_D\_FORMATS***.add(BarcodeFormat.CODE\_93);  
 ***ONE\_D\_FORMATS***.add(BarcodeFormat.CODE\_128);  
 ***ONE\_D\_FORMATS***.add(BarcodeFormat.ITF);  
 ***QR\_CODE\_FORMATS*** = **new** Vector<BarcodeFormat>(1);  
 ***QR\_CODE\_FORMATS***.add(BarcodeFormat.QR\_CODE);  
 ***DATA\_MATRIX\_FORMATS*** = **new** Vector<BarcodeFormat>(1);  
 ***DATA\_MATRIX\_FORMATS***.add(BarcodeFormat.DATA\_MATRIX);  
 }  
  
 **private** DecodeFormatManager() {  
 }  
  
 **static** Vector<BarcodeFormat> parseDecodeFormats(Intent intent) {  
 List<String> scanFormats = **null**;  
 String scanFormatsString = intent.getStringExtra(Intents.Scan.***SCAN\_FORMATS***);  
 **if** (scanFormatsString != **null**) {  
 scanFormats = Arrays.asList(***COMMA\_PATTERN***.split(scanFormatsString));  
 }  
 **return** *parseDecodeFormats*(scanFormats, intent.getStringExtra(Intents.Scan.***MODE***));  
 }  
  
 **static** Vector<BarcodeFormat> parseDecodeFormats(Uri inputUri) {  
 List<String> formats = inputUri.getQueryParameters(Intents.Scan.***SCAN\_FORMATS***);  
 **if** (formats != **null** && formats.size() == 1 && formats.get(0) != **null**) {  
 formats = Arrays.asList(***COMMA\_PATTERN***.split(formats.get(0)));  
 }  
 **return** *parseDecodeFormats*(formats, inputUri.getQueryParameter(Intents.Scan.***MODE***));  
 }  
  
 **private static** Vector<BarcodeFormat> parseDecodeFormats(Iterable<String> scanFormats,  
 String decodeMode) {  
 **if** (scanFormats != **null**) {  
 Vector<BarcodeFormat> formats = **new** Vector<BarcodeFormat>();  
 **try** {  
 **for** (String format : scanFormats) {  
 formats.add(BarcodeFormat.valueOf(format));  
 }  
 **return** formats;  
 } **catch** (IllegalArgumentException iae) {  
 *// ignore it then* }  
 }  
 **if** (decodeMode != **null**) {  
 **if** (Intents.Scan.***PRODUCT\_MODE***.equals(decodeMode)) {  
 **return *PRODUCT\_FORMATS***;  
 }  
 **if** (Intents.Scan.***QR\_CODE\_MODE***.equals(decodeMode)) {  
 **return *QR\_CODE\_FORMATS***;  
 }  
 **if** (Intents.Scan.***DATA\_MATRIX\_MODE***.equals(decodeMode)) {  
 **return *DATA\_MATRIX\_FORMATS***;  
 }  
 **if** (Intents.Scan.***ONE\_D\_MODE***.equals(decodeMode)) {  
 **return *ONE\_D\_FORMATS***;  
 }  
 }  
 **return null**;  
 }  
  
}

*/\*  
 \* Copyright (C) 2010 ZXing authors  
 \*  
 \* Licensed under the Apache License, Version 2.0 (the "License");  
 \* you may not use this file except in compliance with the License.  
 \* You may obtain a copy of the License at  
 \*  
 \* http://www.apache.org/licenses/LICENSE-2.0  
 \*  
 \* Unless required by applicable law or agreed to in writing, software  
 \* distributed under the License is distributed on an "AS IS" BASIS,  
 \* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
 \* See the License for the specific language governing permissions and  
 \* limitations under the License.  
 \*/***package** com.google.zxing.decoding;  
  
**import** android.os.Bundle;  
**import** android.os.Handler;  
**import** android.os.Looper;  
**import** android.os.Message;  
**import** android.util.Log;  
  
**import** com.google.zxing.BinaryBitmap;  
**import** com.google.zxing.DecodeHintType;  
**import** com.google.zxing.MultiFormatReader;  
**import** com.google.zxing.R;  
**import** com.google.zxing.ReaderException;  
**import** com.google.zxing.Result;  
**import** com.google.zxing.activity.CaptureActivity;  
**import** com.google.zxing.camera.CameraManager;  
**import** com.google.zxing.camera.PlanarYUVLuminanceSource;  
**import** com.google.zxing.common.HybridBinarizer;  
  
**import** java.util.Hashtable;  
  
  
**final class** DecodeHandler **extends** Handler {  
  
 **private static final** String ***TAG*** = DecodeHandler.**class**.getSimpleName();  
  
 **private final** CaptureActivity **activity**;  
 **private final** MultiFormatReader **multiFormatReader**;  
  
 DecodeHandler(CaptureActivity activity, Hashtable<DecodeHintType, Object> hints) {  
 **multiFormatReader** = **new** MultiFormatReader();  
 **multiFormatReader**.setHints(hints);  
 **this**.**activity** = activity;  
 }  
  
 @Override  
 **public void** handleMessage(Message message) {  
 **int** i = message.what;  
 **if** (i == R.id.decode) {  
 *//Log.d(TAG, "Got decode message");* decode((**byte**[]) message.obj, message.arg1, message.arg2);  
 } **else if** (i == R.id.quit) {  
 Looper.myLooper().quit();  
 }  
 }  
  
 */\*\*  
 \* Decode the data within the viewfinder rectangle, and time how long it took. For efficiency,  
 \* reuse the same reader objects from one decode to the next.  
 \*  
 \** ***@param data*** *The YUV preview frame.  
 \** ***@param width*** *The width of the preview frame.  
 \** ***@param height*** *The height of the preview frame.  
 \*/* **private void** decode(**byte**[] data, **int** width, **int** height) {  
 **long** start = System.currentTimeMillis();  
 Result rawResult = **null**;  
  
 *//modify here* **byte**[] rotatedData = **new byte**[data.**length**];  
 **for** (**int** y = 0; y < height; y++) {  
 **for** (**int** x = 0; x < width; x++) {  
 rotatedData[x \* height + height - y - 1] = data[x + y \* width];  
 }  
 }  
 **int** tmp = width; *// Here we are swapping, that's the difference to #11* width = height;  
 height = tmp;  
  
 PlanarYUVLuminanceSource source = CameraManager.*get*().buildLuminanceSource(rotatedData, width, height);  
 BinaryBitmap bitmap = **new** BinaryBitmap(**new** HybridBinarizer(source));  
 **try** {  
 rawResult = **multiFormatReader**.decodeWithState(bitmap);  
 } **catch** (ReaderException re) {  
 *// continue* } **finally** {  
 **multiFormatReader**.reset();  
 }  
  
 **if** (rawResult != **null**) {  
 **long** end = System.currentTimeMillis();  
 Log.d(***TAG***, **"Found barcode ("** + (end - start) + **" ms):\n"** + rawResult.toString());  
 Message message = Message.obtain(**activity**.getHandler(), R.id.decode\_succeeded, rawResult);  
 Bundle bundle = **new** Bundle();  
 bundle.putParcelable(DecodeThread.***BARCODE\_BITMAP***, source.renderCroppedGreyscaleBitmap());  
 message.setData(bundle);  
 *//Log.d(TAG, "Sending decode succeeded message...");* message.sendToTarget();  
 } **else** {  
 Message message = Message.obtain(**activity**.getHandler(), R.id.decode\_failed);  
 message.sendToTarget();  
 }  
 }  
  
}

*/\*  
 \* Copyright (C) 2008 ZXing authors  
 \*  
 \* Licensed under the Apache License, Version 2.0 (the "License");  
 \* you may not use this file except in compliance with the License.  
 \* You may obtain a copy of the License at  
 \*  
 \* http://www.apache.org/licenses/LICENSE-2.0  
 \*  
 \* Unless required by applicable law or agreed to in writing, software  
 \* distributed under the License is distributed on an "AS IS" BASIS,  
 \* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
 \* See the License for the specific language governing permissions and  
 \* limitations under the License.  
 \*/***package** com.google.zxing.decoding;  
  
**import** android.os.Handler;  
**import** android.os.Looper;  
  
**import** com.google.zxing.BarcodeFormat;  
**import** com.google.zxing.DecodeHintType;  
**import** com.google.zxing.ResultPointCallback;  
**import** com.google.zxing.activity.CaptureActivity;  
  
**import** java.util.Hashtable;  
**import** java.util.Vector;  
**import** java.util.concurrent.CountDownLatch;  
  
*/\*\*  
 \* This thread does all the heavy lifting of decoding the images.  
 \* �����߳�  
 \*/***final class** DecodeThread **extends** Thread {  
  
 **public static final** String ***BARCODE\_BITMAP*** = **"barcode\_bitmap"**;  
 **private final** CaptureActivity **activity**;  
 **private final** Hashtable<DecodeHintType, Object> **hints**;  
 **private** Handler **handler**;  
 **private final** CountDownLatch **handlerInitLatch**;  
  
 DecodeThread(CaptureActivity activity,  
 Vector<BarcodeFormat> decodeFormats,  
 String characterSet,  
 ResultPointCallback resultPointCallback) {  
  
 **this**.**activity** = activity;  
 **handlerInitLatch** = **new** CountDownLatch(1);  
  
 **hints** = **new** Hashtable<DecodeHintType, Object>(3);  
  
 **if** (decodeFormats == **null** || decodeFormats.isEmpty()) {  
 decodeFormats = **new** Vector<BarcodeFormat>();  
 decodeFormats.addAll(DecodeFormatManager.***ONE\_D\_FORMATS***);  
 decodeFormats.addAll(DecodeFormatManager.***QR\_CODE\_FORMATS***);  
 decodeFormats.addAll(DecodeFormatManager.***DATA\_MATRIX\_FORMATS***);  
 }  
  
 **hints**.put(DecodeHintType.POSSIBLE\_FORMATS, decodeFormats);  
  
 **if** (characterSet != **null**) {  
 **hints**.put(DecodeHintType.CHARACTER\_SET, characterSet);  
 }  
  
 **hints**.put(DecodeHintType.NEED\_RESULT\_POINT\_CALLBACK, resultPointCallback);  
 }  
  
 Handler getHandler() {  
 **try** {  
 **handlerInitLatch**.await();  
 } **catch** (InterruptedException ie) {  
 *// continue?* }  
 **return handler**;  
 }  
  
 @Override  
 **public void** run() {  
 Looper.prepare();  
 **handler** = **new** DecodeHandler(**activity**, **hints**);  
 **handlerInitLatch**.countDown();  
 Looper.loop();  
 }  
  
}

*/\*  
 \* Copyright (C) 2010 ZXing authors  
 \*  
 \* Licensed under the Apache License, Version 2.0 (the "License");  
 \* you may not use this file except in compliance with the License.  
 \* You may obtain a copy of the License at  
 \*  
 \* http://www.apache.org/licenses/LICENSE-2.0  
 \*  
 \* Unless required by applicable law or agreed to in writing, software  
 \* distributed under the License is distributed on an "AS IS" BASIS,  
 \* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
 \* See the License for the specific language governing permissions and  
 \* limitations under the License.  
 \*/***package** com.google.zxing.decoding;  
  
**import** android.app.Activity;  
**import** android.content.DialogInterface;  
  
*/\*\*  
 \* Simple listener used to exit the app in a few cases.  
 \*  
 \*/***public final class** FinishListener  
 **implements** DialogInterface.OnClickListener, DialogInterface.OnCancelListener, Runnable {  
  
 **private final** Activity **activityToFinish**;  
  
 **public** FinishListener(Activity activityToFinish) {  
 **this**.**activityToFinish** = activityToFinish;  
 }  
  
 **public void** onCancel(DialogInterface dialogInterface) {  
 run();  
 }  
  
 **public void** onClick(DialogInterface dialogInterface, **int** i) {  
 run();  
 }  
  
 **public void** run() {  
 **activityToFinish**.finish();  
 }  
  
}

*/\*  
 \* Copyright (C) 2010 ZXing authors  
 \*  
 \* Licensed under the Apache License, Version 2.0 (the "License");  
 \* you may not use this file except in compliance with the License.  
 \* You may obtain a copy of the License at  
 \*  
 \* http://www.apache.org/licenses/LICENSE-2.0  
 \*  
 \* Unless required by applicable law or agreed to in writing, software  
 \* distributed under the License is distributed on an "AS IS" BASIS,  
 \* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
 \* See the License for the specific language governing permissions and  
 \* limitations under the License.  
 \*/***package** com.google.zxing.decoding;  
  
**import** android.app.Activity;  
  
**import** java.util.concurrent.Executors;  
**import** java.util.concurrent.ScheduledExecutorService;  
**import** java.util.concurrent.ScheduledFuture;  
**import** java.util.concurrent.ThreadFactory;  
**import** java.util.concurrent.TimeUnit;  
  
*/\*\*  
 \* Finishes an activity after a period of inactivity.  
 \*/***public final class** InactivityTimer {  
  
 **private static final int *INACTIVITY\_DELAY\_SECONDS*** = 5 \* 60;  
  
 **private final** ScheduledExecutorService **inactivityTimer** =  
 Executors.newSingleThreadScheduledExecutor(**new** DaemonThreadFactory());  
 **private final** Activity **activity**;  
 **private** ScheduledFuture<?> **inactivityFuture** = **null**;  
  
 **public** InactivityTimer(Activity activity) {  
 **this**.**activity** = activity;  
 onActivity();  
 }  
  
 **public void** onActivity() {  
 cancel();  
 **inactivityFuture** = **inactivityTimer**.schedule(**new** FinishListener(**activity**),  
 ***INACTIVITY\_DELAY\_SECONDS***,  
 TimeUnit.SECONDS);  
 }  
  
 **private void** cancel() {  
 **if** (**inactivityFuture** != **null**) {  
 **inactivityFuture**.cancel(**true**);  
 **inactivityFuture** = **null**;  
 }  
 }  
  
 **public void** shutdown() {  
 cancel();  
 **inactivityTimer**.shutdown();  
 }  
  
 **private static final class** DaemonThreadFactory **implements** ThreadFactory {  
 **public** Thread newThread(Runnable runnable) {  
 Thread thread = **new** Thread(runnable);  
 thread.setDaemon(**true**);  
 **return** thread;  
 }  
 }  
  
}

*/\*  
 \* Copyright (C) 2008 ZXing authors  
 \*  
 \* Licensed under the Apache License, Version 2.0 (the "License");  
 \* you may not use this file except in compliance with the License.  
 \* You may obtain a copy of the License at  
 \*  
 \* http://www.apache.org/licenses/LICENSE-2.0  
 \*  
 \* Unless required by applicable law or agreed to in writing, software  
 \* distributed under the License is distributed on an "AS IS" BASIS,  
 \* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
 \* See the License for the specific language governing permissions and  
 \* limitations under the License.  
 \*/***package** com.google.zxing.decoding;  
  
*/\*\*  
 \* This class provides the constants to use when sending an Intent to Barcode Scanner.  
 \* These strings are effectively API and cannot be changed.  
 \*/***public final class** Intents {  
 **private** Intents() {  
 }  
  
 **public static final class** Scan {  
 */\*\*  
 \* Send this intent to open the Barcodes app in scanning mode, find a barcode, and return  
 \* the results.  
 \*/* **public static final** String ***ACTION*** = **"com.google.zxing.client.android.SCAN"**;  
  
 */\*\*  
 \* By default, sending Scan.ACTION will decode all barcodes that we understand. However it  
 \* may be useful to limit scanning to certain formats. Use Intent.putExtra(MODE, value) with  
 \* one of the values below ({****@link*** *#PRODUCT\_MODE}, {****@link*** *#ONE\_D\_MODE}, {****@link*** *#QR\_CODE\_MODE}).  
 \* Optional.  
 \*  
 \* Setting this is effectively shorthnad for setting explicit formats with {****@link*** *#SCAN\_FORMATS}.  
 \* It is overridden by that setting.  
 \*/* **public static final** String ***MODE*** = **"SCAN\_MODE"**;  
  
 */\*\*  
 \* Comma-separated list of formats to scan for. The values must match the names of  
 \* {****@link*** *com.google.zxing.BarcodeFormat}s, such as {****@link*** *com.google.zxing.BarcodeFormat#EAN\_13}.  
 \* Example: "EAN\_13,EAN\_8,QR\_CODE"  
 \*  
 \* This overrides {****@link*** *#MODE}.  
 \*/* **public static final** String ***SCAN\_FORMATS*** = **"SCAN\_FORMATS"**;  
  
 */\*\*  
 \** ***@see*** *com.google.zxing.DecodeHintType#CHARACTER\_SET  
 \*/* **public static final** String ***CHARACTER\_SET*** = **"CHARACTER\_SET"**;  
  
 */\*\*  
 \* Decode only UPC and EAN barcodes. This is the right choice for shopping apps which get  
 \* prices, reviews, etc. for products.  
 \*/* **public static final** String ***PRODUCT\_MODE*** = **"PRODUCT\_MODE"**;  
  
 */\*\*  
 \* Decode only 1D barcodes (currently UPC, EAN, Code 39, and Code 128).  
 \*/* **public static final** String ***ONE\_D\_MODE*** = **"ONE\_D\_MODE"**;  
  
 */\*\*  
 \* Decode only QR codes.  
 \*/* **public static final** String ***QR\_CODE\_MODE*** = **"QR\_CODE\_MODE"**;  
  
 */\*\*  
 \* Decode only Data Matrix codes.  
 \*/* **public static final** String ***DATA\_MATRIX\_MODE*** = **"DATA\_MATRIX\_MODE"**;  
  
 */\*\*  
 \* If a barcode is found, Barcodes returns RESULT\_OK to onActivityResult() of the app which  
 \* requested the scan via startSubActivity(). The barcodes contents can be retrieved with  
 \* intent.getStringExtra(RESULT). If the user presses Back, the result code will be  
 \* RESULT\_CANCELED.  
 \*/* **public static final** String ***RESULT*** = **"SCAN\_RESULT"**;  
  
 */\*\*  
 \* Call intent.getStringExtra(RESULT\_FORMAT) to determine which barcode format was found.  
 \* See Contents.Format for possible values.  
 \*/* **public static final** String ***RESULT\_FORMAT*** = **"SCAN\_RESULT\_FORMAT"**;  
  
 */\*\*  
 \* Setting this to false will not save scanned codes in the history.  
 \*/* **public static final** String ***SAVE\_HISTORY*** = **"SAVE\_HISTORY"**;  
  
 **private** Scan() {  
 }  
 }  
  
 **public static final class** Encode {  
 */\*\*  
 \* Send this intent to encode a piece of data as a QR code and display it full screen, so  
 \* that another person can scan the barcode from your screen.  
 \*/* **public static final** String ***ACTION*** = **"com.google.zxing.client.android.ENCODE"**;  
  
 */\*\*  
 \* The data to encode. Use Intent.putExtra(DATA, data) where data is either a String or a  
 \* Bundle, depending on the type and format specified. Non-QR Code formats should  
 \* just use a String here. For QR Code, see Contents for details.  
 \*/* **public static final** String ***DATA*** = **"ENCODE\_DATA"**;  
  
 */\*\*  
 \* The type of data being supplied if the format is QR Code. Use  
 \* Intent.putExtra(TYPE, type) with one of Contents.Type.  
 \*/* **public static final** String ***TYPE*** = **"ENCODE\_TYPE"**;  
  
 */\*\*  
 \* The barcode format to be displayed. If this isn't specified or is blank,  
 \* it defaults to QR Code. Use Intent.putExtra(FORMAT, format), where  
 \* format is one of Contents.Format.  
 \*/* **public static final** String ***FORMAT*** = **"ENCODE\_FORMAT"**;  
  
 **private** Encode() {  
 }  
 }  
  
 **public static final class** SearchBookContents {  
 */\*\*  
 \* Use Google Book Search to search the contents of the book provided.  
 \*/* **public static final** String ***ACTION*** = **"com.google.zxing.client.android.SEARCH\_BOOK\_CONTENTS"**;  
  
 */\*\*  
 \* The book to search, identified by ISBN number.  
 \*/* **public static final** String ***ISBN*** = **"ISBN"**;  
  
 */\*\*  
 \* An optional field which is the text to search for.  
 \*/* **public static final** String ***QUERY*** = **"QUERY"**;  
  
 **private** SearchBookContents() {  
 }  
 }  
  
 **public static final class** WifiConnect {  
 */\*\*  
 \* Internal intent used to trigger connection to a wi-fi network.  
 \*/* **public static final** String ***ACTION*** = **"com.google.zxing.client.android.WIFI\_CONNECT"**;  
  
 */\*\*  
 \* The network to connect to, all the configuration provided here.  
 \*/* **public static final** String ***SSID*** = **"SSID"**;  
  
 */\*\*  
 \* The network to connect to, all the configuration provided here.  
 \*/* **public static final** String ***TYPE*** = **"TYPE"**;  
  
 */\*\*  
 \* The network to connect to, all the configuration provided here.  
 \*/* **public static final** String ***PASSWORD*** = **"PASSWORD"**;  
  
 **private** WifiConnect() {  
 }  
 }  
  
  
 **public static final class** Share {  
 */\*\*  
 \* Give the user a choice of items to encode as a barcode, then render it as a QR Code and  
 \* display onscreen for a friend to scan with their phone.  
 \*/* **public static final** String ***ACTION*** = **"com.google.zxing.client.android.SHARE"**;  
  
 **private** Share() {  
 }  
 }  
}

*/\*  
 \* Copyright 2009 ZXing authors  
 \*  
 \* Licensed under the Apache License, Version 2.0 (the "License");  
 \* you may not use this file except in compliance with the License.  
 \* You may obtain a copy of the License at  
 \*  
 \* http://www.apache.org/licenses/LICENSE-2.0  
 \*  
 \* Unless required by applicable law or agreed to in writing, software  
 \* distributed under the License is distributed on an "AS IS" BASIS,  
 \* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
 \* See the License for the specific language governing permissions and  
 \* limitations under the License.  
 \*/***package** com.google.zxing.decoding;  
  
**import** android.graphics.Bitmap;  
**import** android.graphics.BitmapFactory;  
  
**import** com.google.zxing.LuminanceSource;  
  
**import** java.io.FileNotFoundException;  
  
*/\*\*  
 \* This class is used to help decode images from files which arrive as RGB data  
 \* from Android bitmaps. It does not support cropping or rotation.  
 \*  
 \*/***public final class** RGBLuminanceSource **extends** LuminanceSource {  
  
 **private final byte**[] **luminances**;  
  
 **public** RGBLuminanceSource(String path) **throws** FileNotFoundException {  
 **this**(*loadBitmap*(path));  
 }  
  
 **public** RGBLuminanceSource(Bitmap bitmap) {  
 **super**(bitmap.getWidth(), bitmap.getHeight());  
  
 **int** width = bitmap.getWidth();  
 **int** height = bitmap.getHeight();  
  
 **int**[] pixels = **new int**[width \* height];  
 bitmap.getPixels(pixels, 0, width, 0, 0, width, height);  
  
 *// In order to measure pure decoding speed, we convert the entire image  
 // to a greyscale array  
 // up front, which is the same as the Y channel of the  
 // YUVLuminanceSource in the real app.* **luminances** = **new byte**[width \* height];  
 **for** (**int** y = 0; y < height; y++) {  
 **int** offset = y \* width;  
 **for** (**int** x = 0; x < width; x++) {  
 **int** pixel = pixels[offset + x];  
 **int** r = (pixel >> 16) & 0xff;  
 **int** g = (pixel >> 8) & 0xff;  
 **int** b = pixel & 0xff;  
 **if** (r == g && g == b) {  
 *// Image is already greyscale, so pick any channel.* **luminances**[offset + x] = (**byte**) r;  
 } **else** {  
 *// Calculate luminance cheaply, favoring green.* **luminances**[offset + x] = (**byte**) ((r + g + g + b) >> 2);  
 }  
 }  
 }  
 }  
  
  
 @Override  
 **public byte**[] getRow(**int** y, **byte**[] row) {  
 **if** (y < 0 || y >= getHeight()) {  
 **throw new** IllegalArgumentException(**"Requested row is outside the image: "** + y);  
 }  
 **int** width = getWidth();  
 **if** (row == **null** || row.**length** < width) {  
 row = **new byte**[width];  
 }  
  
 System.arraycopy(**luminances**, y \* width, row, 0, width);  
 **return** row;  
 }  
  
 *// Since this class does not support cropping, the underlying byte array  
 // already contains  
 // exactly what the caller is asking for, so give it to them without a copy.* @Override  
 **public byte**[] getMatrix() {  
 **return luminances**;  
 }  
  
 **private static** Bitmap loadBitmap(String path) **throws** FileNotFoundException {  
 Bitmap bitmap = BitmapFactory.decodeFile(path);  
 **if** (bitmap == **null**) {  
 **throw new** FileNotFoundException(**"Couldn't open "** + path);  
 }  
 **return** bitmap;  
 }  
  
}

**package** com.google.zxing.encoding;  
  
**import** android.graphics.Bitmap;  
**import** android.graphics.Canvas;  
  
**import** com.google.zxing.BarcodeFormat;  
**import** com.google.zxing.EncodeHintType;  
**import** com.google.zxing.MultiFormatWriter;  
**import** com.google.zxing.WriterException;  
**import** com.google.zxing.common.BitMatrix;  
**import** com.google.zxing.qrcode.QRCodeWriter;  
**import** com.google.zxing.qrcode.decoder.ErrorCorrectionLevel;  
  
**import** java.util.HashMap;  
**import** java.util.Hashtable;  
**import** java.util.Map;  
  
*/\*\*  
 \** ***@author*** *Ryan Tang  
 \*  
 \*/***public final class** EncodingHandler {  
 **private static final int *BLACK*** = 0xff000000;  
  
 **public static** Bitmap createQRCode(String str, **int** widthAndHeight) **throws** WriterException {  
 Hashtable<EncodeHintType, String> hints = **new** Hashtable<EncodeHintType, String>();  
 hints.put(EncodeHintType.CHARACTER\_SET, **"utf-8"**);  
 BitMatrix matrix = **new** MultiFormatWriter().encode(str,  
 BarcodeFormat.QR\_CODE, widthAndHeight, widthAndHeight);  
 **int** width = matrix.getWidth();  
 **int** height = matrix.getHeight();  
 **int**[] pixels = **new int**[width \* height];  
  
 **for** (**int** y = 0; y < height; y++) {  
 **for** (**int** x = 0; x < width; x++) {  
 **if** (matrix.get(x, y)) {  
 pixels[y \* width + x] = ***BLACK***;  
 }  
 }  
 }  
 Bitmap bitmap = Bitmap.createBitmap(width, height,  
 Bitmap.Config.ARGB\_8888);  
 bitmap.setPixels(pixels, 0, width, 0, 0, width, height);  
 **return** bitmap;  
 }  
  
 */\*\*  
 \* 创建二维码  
 \*  
 \** ***@param content*** *content  
 \** ***@param widthPix*** *widthPix  
 \** ***@param heightPix*** *heightPix  
 \** ***@param logoBm*** *logoBm  
 \** ***@return*** *二维码  
 \*/* **public static** Bitmap createQRCode(String content, **int** widthPix, **int** heightPix, Bitmap logoBm) {  
 **try** {  
 **if** (content == **null** || **""**.equals(content)) {  
 **return null**;  
 }  
 *// 配置参数* Map<EncodeHintType, Object> hints = **new** HashMap<>();  
 hints.put(EncodeHintType.CHARACTER\_SET, **"utf-8"**);  
 *// 容错级别* hints.put(EncodeHintType.ERROR\_CORRECTION, ErrorCorrectionLevel.H);  
 *// 图像数据转换，使用了矩阵转换* BitMatrix bitMatrix = **new** QRCodeWriter().encode(content, BarcodeFormat.QR\_CODE, widthPix,  
 heightPix, hints);  
 **int**[] pixels = **new int**[widthPix \* heightPix];  
 *// 下面这里按照二维码的算法，逐个生成二维码的图片，  
 // 两个for循环是图片横列扫描的结果* **for** (**int** y = 0; y < heightPix; y++) {  
 **for** (**int** x = 0; x < widthPix; x++) {  
 **if** (bitMatrix.get(x, y)) {  
 pixels[y \* widthPix + x] = 0xff000000;  
 } **else** {  
 pixels[y \* widthPix + x] = 0xffffffff;  
 }  
 }  
 }  
 *// 生成二维码图片的格式，使用ARGB\_8888* Bitmap bitmap = Bitmap.createBitmap(widthPix, heightPix, Bitmap.Config.ARGB\_8888);  
 bitmap.setPixels(pixels, 0, widthPix, 0, 0, widthPix, heightPix);  
 **if** (logoBm != **null**) {  
 bitmap = *addLogo*(bitmap, logoBm);  
 }  
 *//必须使用compress方法将bitmap保存到文件中再进行读取。直接返回的bitmap是没有任何压缩的，内存消耗巨大！* **return** bitmap;  
 } **catch** (WriterException e) {  
 e.printStackTrace();  
 }  
 **return null**;  
 }  
  
 */\*\*  
 \* 在二维码中间添加Logo图案  
 \*/* **private static** Bitmap addLogo(Bitmap src, Bitmap logo) {  
 **if** (src == **null**) {  
 **return null**;  
 }  
 **if** (logo == **null**) {  
 **return** src;  
 }  
 *//获取图片的宽高* **int** srcWidth = src.getWidth();  
 **int** srcHeight = src.getHeight();  
 **int** logoWidth = logo.getWidth();  
 **int** logoHeight = logo.getHeight();  
 **if** (srcWidth == 0 || srcHeight == 0) {  
 **return null**;  
 }  
 **if** (logoWidth == 0 || logoHeight == 0) {  
 **return** src;  
 }  
 *//logo大小为二维码整体大小的1/5* **float** scaleFactor = srcWidth \* 1.0f / 5 / logoWidth;  
 Bitmap bitmap = Bitmap.createBitmap(srcWidth, srcHeight, Bitmap.Config.ARGB\_8888);  
 **try** {  
 Canvas canvas = **new** Canvas(bitmap);  
 canvas.drawBitmap(src, 0, 0, **null**);  
 canvas.scale(scaleFactor, scaleFactor, srcWidth / 2, srcHeight / 2);  
 canvas.drawBitmap(logo, (srcWidth - logoWidth) / 2, (srcHeight - logoHeight) / 2, **null**);  
*// canvas.save(Canvas.ALL\_SAVE\_FLAG);* canvas.save();  
 canvas.restore();  
 } **catch** (Exception e) {  
 bitmap = **null**;  
 e.getStackTrace();  
 }  
 **return** bitmap;  
 }  
}

**package** com.google.zxing.util;  
  
**import** android.content.ContentResolver;  
**import** android.content.Context;  
**import** android.graphics.Bitmap;  
**import** android.graphics.BitmapFactory;  
**import** android.net.Uri;  
**import** android.util.Log;  
  
**import** java.io.IOException;  
**import** java.io.InputStream;  
  
**public class** BitmapUtil {  
 */\*\*  
 \* 读取一个缩放后的图片，限定图片大小，避免OOM  
 \** ***@param uri*** *图片uri，支持“file://”、“content://”  
 \** ***@param maxWidth*** *最大允许宽度  
 \** ***@param maxHeight*** *最大允许高度  
 \** ***@return*** *返回一个缩放后的Bitmap，失败则返回null  
 \*/* **public static** Bitmap decodeUri(Context context, Uri uri, **int** maxWidth, **int** maxHeight) {  
 BitmapFactory.Options options = **new** BitmapFactory.Options();  
 options.inJustDecodeBounds = **true**; *//只读取图片尺寸  
 readBitmapScale*(context, uri, options);  
  
 *//计算实际缩放比例* **int** scale = 1;  
 **for** (**int** i = 0; i < Integer.MAX\_VALUE; i++) {  
 **if** ((options.outWidth / scale > maxWidth &&  
 options.outWidth / scale > maxWidth \* 1.4) ||  
 (options.outHeight / scale > maxHeight &&  
 options.outHeight / scale > maxHeight \* 1.4)) {  
 scale++;  
 } **else** {  
 **break**;  
 }  
 }  
  
 options.inSampleSize = scale;  
 options.inJustDecodeBounds = **false**;*//读取图片内容* options.inPreferredConfig = Bitmap.Config.RGB\_565; *//根据情况进行修改* Bitmap bitmap = **null**;  
 **try** {  
 bitmap = *readBitmapData*(context, uri, options);  
 } **catch** (Throwable e) {  
 e.printStackTrace();  
 }  
 **return** bitmap;  
 }  
  
 **private static void** readBitmapScale(Context context, Uri uri, BitmapFactory.Options options) {  
 **if** (uri == **null**) {  
 **return**;  
 }  
 String scheme = uri.getScheme();  
 **if** (ContentResolver.SCHEME\_CONTENT.equals(scheme) ||  
 ContentResolver.SCHEME\_FILE.equals(scheme)) {  
 InputStream stream = **null**;  
 **try** {  
 stream = context.getContentResolver().openInputStream(uri);  
 BitmapFactory.decodeStream(stream, **null**, options);  
 } **catch** (Exception e) {  
 Log.w(**"readBitmapScale"**, **"Unable to open content: "** + uri, e);  
 } **finally** {  
 **if** (stream != **null**) {  
 **try** {  
 stream.close();  
 } **catch** (IOException e) {  
 Log.e(**"readBitmapScale"**, **"Unable to close content: "** + uri, e);  
 }  
 }  
 }  
 } **else if** (ContentResolver.SCHEME\_ANDROID\_RESOURCE.equals(scheme)) {  
 Log.e(**"readBitmapScale"**, **"Unable to close content: "** + uri);  
 } **else** {  
 Log.e(**"readBitmapScale"**, **"Unable to close content: "** + uri);  
 }  
 }  
  
 **private static** Bitmap readBitmapData(Context context, Uri uri, BitmapFactory.Options options) {  
 **if** (uri == **null**) {  
 **return null**;  
 }  
 Bitmap bitmap = **null**;  
 String scheme = uri.getScheme();  
 **if** (ContentResolver.SCHEME\_CONTENT.equals(scheme) ||  
 ContentResolver.SCHEME\_FILE.equals(scheme)) {  
 InputStream stream = **null**;  
 **try** {  
 stream = context.getContentResolver().openInputStream(uri);  
 bitmap = BitmapFactory.decodeStream(stream, **null**, options);  
 } **catch** (Exception e) {  
 Log.e(**"readBitmapData"**, **"Unable to open content: "** + uri, e);  
 } **finally** {  
 **if** (stream != **null**) {  
 **try** {  
 stream.close();  
 } **catch** (IOException e) {  
 Log.e(**"readBitmapData"**, **"Unable to close content: "** + uri, e);  
 }  
 }  
 }  
 } **else if** (ContentResolver.SCHEME\_ANDROID\_RESOURCE.equals(scheme)) {  
 Log.e(**"readBitmapData"**, **"Unable to close content: "** + uri);  
 } **else** {  
 Log.e(**"readBitmapData"**, **"Unable to close content: "** + uri);  
 }  
 **return** bitmap;  
 }  
}

**package** com.google.zxing.util;  
  
**public class** Constant {  
 **public static final int *REQ\_QR\_CODE*** = 11002;  
 **public static final int *REQ\_PERM\_CAMERA*** = 11003;  
  
 **public static final** String ***INTENT\_EXTRA\_KEY\_QR\_SCAN*** = **"qr\_scan\_result"**;  
}

**package** com.google.zxing.util;  
  
**import** android.graphics.Bitmap;  
**import** android.graphics.Color;  
  
**import** com.google.zxing.BarcodeFormat;  
**import** com.google.zxing.EncodeHintType;  
**import** com.google.zxing.MultiFormatWriter;  
**import** com.google.zxing.WriterException;  
**import** com.google.zxing.common.BitMatrix;  
**import** com.google.zxing.qrcode.decoder.ErrorCorrectionLevel;  
  
**import** java.util.HashMap;  
**import** java.util.Map;  
  
**public class** QrCodeGenerator {  
  
 **public static** Bitmap getQrCodeImage(String data, **int** width, **int** height) {  
 **if** (data == **null** || data.length() == 0) {  
 **return null**;  
 }  
 Map<EncodeHintType, Object> hintsMap = **new** HashMap<>(3);  
 hintsMap.put(EncodeHintType.CHARACTER\_SET, **"utf-8"**);  
 hintsMap.put(EncodeHintType.ERROR\_CORRECTION, ErrorCorrectionLevel.H);  
 hintsMap.put(EncodeHintType.MARGIN, 0);  
 **try** {  
 BitMatrix bitMatrix = **new** MultiFormatWriter().encode(data, BarcodeFormat.QR\_CODE, width, height, hintsMap);  
 Bitmap bitmap = *bitMatrix2Bitmap*(bitMatrix);  
 **return** bitmap;  
 } **catch** (WriterException e) {  
 e.printStackTrace();  
 }  
 **return null**;  
 }  
  
 **private static** Bitmap bitMatrix2Bitmap(BitMatrix matrix) {  
 **int** w = matrix.getWidth();  
 **int** h = matrix.getHeight();  
 **int**[] rawData = **new int**[w \* h];  
 **for** (**int** i = 0; i < w; i++) {  
 **for** (**int** j = 0; j < h; j++) {  
 **int** color = Color.WHITE;  
 **if** (matrix.get(i, j)) {  
 color = Color.BLACK;  
 }  
 rawData[i + (j \* w)] = color;  
 }  
 }  
 Bitmap bitmap = Bitmap.createBitmap(w, h, Bitmap.Config.RGB\_565);  
 bitmap.setPixels(rawData, 0, w, 0, 0, w, h);  
 **return** bitmap;  
 }  
  
}

*/\*  
 \* Copyright (C) 2009 ZXing authors  
 \*  
 \* Licensed under the Apache License, Version 2.0 (the "License");  
 \* you may not use this file except in compliance with the License.  
 \* You may obtain a copy of the License at  
 \*  
 \* http://www.apache.org/licenses/LICENSE-2.0  
 \*  
 \* Unless required by applicable law or agreed to in writing, software  
 \* distributed under the License is distributed on an "AS IS" BASIS,  
 \* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
 \* See the License for the specific language governing permissions and  
 \* limitations under the License.  
 \*/***package** com.google.zxing.view;  
  
**import** com.google.zxing.ResultPoint;  
**import** com.google.zxing.ResultPointCallback;  
  
**public final class** ViewfinderResultPointCallback **implements** ResultPointCallback {  
 **private final** ViewfinderView **viewfinderView**;  
  
 **public** ViewfinderResultPointCallback(ViewfinderView viewfinderView) {  
 **this**.**viewfinderView** = viewfinderView;  
 }  
  
 **public void** foundPossibleResultPoint(ResultPoint point) {  
 **viewfinderView**.addPossibleResultPoint(point);  
 }  
  
}

*/\*  
 \* Copyright (C) 2008 ZXing authors  
 \*  
 \* Licensed under the Apache License, Version 2.0 (the "License");  
 \* you may not use this file except in compliance with the License.  
 \* You may obtain a copy of the License at  
 \*  
 \* http://www.apache.org/licenses/LICENSE-2.0  
 \*  
 \* Unless required by applicable law or agreed to in writing, software  
 \* distributed under the License is distributed on an "AS IS" BASIS,  
 \* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
 \* See the License for the specific language governing permissions and  
 \* limitations under the License.  
 \*/***package** com.google.zxing.view;  
  
**import** android.content.Context;  
**import** android.content.res.TypedArray;  
**import** android.graphics.Bitmap;  
**import** android.graphics.Canvas;  
**import** android.graphics.ComposeShader;  
**import** android.graphics.LinearGradient;  
**import** android.graphics.Paint;  
**import** android.graphics.PorterDuff;  
**import** android.graphics.RadialGradient;  
**import** android.graphics.Rect;  
**import** android.graphics.RectF;  
**import** android.graphics.Shader;  
**import** android.graphics.SweepGradient;  
**import** android.util.AttributeSet;  
**import** android.view.View;  
  
**import** com.google.zxing.R;  
**import** com.google.zxing.ResultPoint;  
**import** com.google.zxing.camera.CameraManager;  
  
**import** java.util.Collection;  
**import** java.util.HashSet;  
  
  
*/\*\*  
 \* This view is overlaid on top of the camera preview. It adds the viewfinder rectangle and partial  
 \* transparency outside it, as well as the laser scanner animation and result points.  
 \** ***@author*** *dswitkin@google.com (Daniel Switkin)  
 \*/***public final class** ViewfinderView **extends** View {  
 **private static final int**[] ***SCANNER\_ALPHA*** = {0, 64, 128, 192, 255, 192, 128, 64};  
 **private static final long *ANIMATION\_DELAY*** = 10L;  
 **private static final int *OPAQUE*** = 0xFF;  
 **private static final int *CORNER\_RECT\_WIDTH*** = 10; *//扫描区边角的宽* **private static final int *CORNER\_RECT\_HEIGHT*** = 50; *//扫描区边角的高* **private static final int *SCANNER\_LINE\_MOVE\_DISTANCE*** = 5; *//扫描线移动距离* **private static final int *SCANNER\_LINE\_HEIGHT*** = 10; *//扫描线宽度* **private final** Paint **paint**;  
 **private** Bitmap **resultBitmap**;  
 *//模糊区域颜色* **private final int maskColor**;  
 **private final int resultColor**;  
 *//扫描区域边框颜色* **private final int frameColor**;  
 *//扫描线颜色* **private final int laserColor**;  
 *//四角颜色* **private final int cornerColor**;  
 *//扫描点的颜色* **private final int resultPointColor**;  
 **private int scannerAlpha**;  
 *//扫描区域提示文本* **private final** String **labelText**;  
 *//扫描区域提示文本颜色* **private final int labelTextColor**;  
 **private final float labelTextSize**;  
 **private final float labelTextMarginTop**;  
  
 **public static int** *scannerStart* = 0;  
 **public static int** *scannerEnd* = 0;  
  
 **private** Collection<ResultPoint> **possibleResultPoints**;  
 **private** Collection<ResultPoint> **lastPossibleResultPoints**;  
  
 *// This constructor is used when the class is built from an XML resource.* **public** ViewfinderView(Context context, AttributeSet attrs) {  
 **super**(context, attrs);  
  
 *//初始化自定义属性信息* TypedArray array = context.obtainStyledAttributes(attrs, R.styleable.ViewfinderView);  
 **laserColor** = array.getColor(R.styleable.ViewfinderView\_laser\_color, 0x00FF00);  
 **cornerColor** = array.getColor(R.styleable.ViewfinderView\_corner\_color, 0x00FF00);  
 **frameColor** = array.getColor(R.styleable.ViewfinderView\_frame\_color, 0xFFFFFF);  
 **resultPointColor** = array.getColor(R.styleable.ViewfinderView\_result\_point\_color, 0xC0FFFF00);  
 **maskColor** = array.getColor(R.styleable.ViewfinderView\_mask\_color, 0x60000000);  
 **resultColor** = array.getColor(R.styleable.ViewfinderView\_result\_color, 0xB0000000);  
 **labelTextColor** = array.getColor(R.styleable.ViewfinderView\_label\_text\_color, 0x90FFFFFF);  
 **labelText** = array.getString(R.styleable.ViewfinderView\_label\_text);  
 **labelTextSize** = array.getDimension(R.styleable.ViewfinderView\_label\_text\_size, 36f);  
 **labelTextMarginTop** = array.getDimension(R.styleable.ViewfinderView\_label\_text\_margin\_top, 0);  
  
 *// Initialize these once for performance rather than calling them every time in onDraw().* **paint** = **new** Paint();  
 **paint**.setAntiAlias(**true**);  
 **scannerAlpha** = 0;  
 **possibleResultPoints** = **new** HashSet<ResultPoint>(5);  
  
  
 }  
  
 @Override  
 **public void** onDraw(Canvas canvas) {  
 Rect frame = CameraManager.*get*().getFramingRect();  
 **if** (frame == **null**) {  
 **return**;  
 }  
 **if** (*scannerStart* == 0 || *scannerEnd* == 0) {  
 *scannerStart* = frame.top;  
 *scannerEnd* = frame.bottom;  
 }  
  
 **int** width = canvas.getWidth();  
 **int** height = canvas.getHeight();  
 *// Draw the exterior (i.e. outside the framing rect) darkened* drawExterior(canvas, frame, width, height);  
  
  
 **if** (**resultBitmap** != **null**) {  
 *// Draw the opaque result bitmap over the scanning rectangle* **paint**.setAlpha(***OPAQUE***);  
 canvas.drawBitmap(**resultBitmap**, frame.left, frame.top, **paint**);  
 } **else** {  
 *// Draw a two pixel solid black border inside the framing rect  
// drawFrame(canvas, frame);  
 // 绘制边角* drawCorner(canvas, frame);  
 *//绘制提示信息* drawTextInfo(canvas, frame);  
 *// Draw a red "laser scanner" line through the middle to show decoding is active* drawLaserScanner(canvas, frame);  
  
 Collection<ResultPoint> currentPossible = **possibleResultPoints**;  
 Collection<ResultPoint> currentLast = **lastPossibleResultPoints**;  
 **if** (currentPossible.isEmpty()) {  
 **lastPossibleResultPoints** = **null**;  
 } **else** {  
 **possibleResultPoints** = **new** HashSet<ResultPoint>(5);  
 **lastPossibleResultPoints** = currentPossible;  
 **paint**.setAlpha(***OPAQUE***);  
 **paint**.setColor(**resultPointColor**);  
 **for** (ResultPoint point : currentPossible) {  
 canvas.drawCircle(frame.left + point.getX(), frame.top + point.getY(), 6.0f, **paint**);  
 }  
 }  
 **if** (currentLast != **null**) {  
 **paint**.setAlpha(***OPAQUE*** / 2);  
 **paint**.setColor(**resultPointColor**);  
 **for** (ResultPoint point : currentLast) {  
 canvas.drawCircle(frame.left + point.getX(), frame.top + point.getY(), 3.0f, **paint**);  
 }  
 }  
  
 *// Request another update at the animation interval, but only repaint the laser line,  
 // not the entire viewfinder mask.  
 //指定重绘区域，该方法会在子线程中执行* postInvalidateDelayed(***ANIMATION\_DELAY***, frame.left, frame.top, frame.right, frame.bottom);  
 }  
 }  
  
 *//绘制文本* **private void** drawTextInfo(Canvas canvas, Rect frame) {  
 **paint**.setColor(**labelTextColor**);  
 **paint**.setTextSize(**labelTextSize**);  
 **paint**.setTextAlign(Paint.Align.CENTER);  
 canvas.drawText(**labelText**, frame.left + frame.width() / 2, frame.bottom + ***CORNER\_RECT\_HEIGHT*** \* 1.5f + **labelTextMarginTop**, **paint**);  
 }  
  
  
 *//绘制边角* **private void** drawCorner(Canvas canvas, Rect frame) {  
 **paint**.setColor(**cornerColor**);  
 *//左上* canvas.drawRect(frame.left, frame.top, frame.left + ***CORNER\_RECT\_WIDTH***, frame.top + ***CORNER\_RECT\_HEIGHT***, **paint**);  
 canvas.drawRect(frame.left, frame.top, frame.left + ***CORNER\_RECT\_HEIGHT***, frame.top + ***CORNER\_RECT\_WIDTH***, **paint**);  
 *//右上* canvas.drawRect(frame.right - ***CORNER\_RECT\_WIDTH***, frame.top, frame.right, frame.top + ***CORNER\_RECT\_HEIGHT***, **paint**);  
 canvas.drawRect(frame.right - ***CORNER\_RECT\_HEIGHT***, frame.top, frame.right, frame.top + ***CORNER\_RECT\_WIDTH***, **paint**);  
 *//左下* canvas.drawRect(frame.left, frame.bottom - ***CORNER\_RECT\_WIDTH***, frame.left + ***CORNER\_RECT\_HEIGHT***, frame.bottom, **paint**);  
 canvas.drawRect(frame.left, frame.bottom - ***CORNER\_RECT\_HEIGHT***, frame.left + ***CORNER\_RECT\_WIDTH***, frame.bottom, **paint**);  
 *//右下* canvas.drawRect(frame.right - ***CORNER\_RECT\_WIDTH***, frame.bottom - ***CORNER\_RECT\_HEIGHT***, frame.right, frame.bottom, **paint**);  
 canvas.drawRect(frame.right - ***CORNER\_RECT\_HEIGHT***, frame.bottom - ***CORNER\_RECT\_WIDTH***, frame.right, frame.bottom, **paint**);  
 }  
  
 *//绘制扫描线* **private void** drawLaserScanner(Canvas canvas, Rect frame) {  
 **paint**.setColor(**laserColor**);  
 *//扫描线闪烁效果  
// paint.setAlpha(SCANNER\_ALPHA[scannerAlpha]);  
// scannerAlpha = (scannerAlpha + 1) % SCANNER\_ALPHA.length;  
// int middle = frame.height() / 2 + frame.top;  
// canvas.drawRect(frame.left + 2, middle - 1, frame.right - 1, middle + 2, paint);  
 //线性渐变* LinearGradient linearGradient = **new** LinearGradient(  
 frame.left, *scannerStart*,  
 frame.left, *scannerStart* + ***SCANNER\_LINE\_HEIGHT***,  
 shadeColor(**laserColor**),  
 **laserColor**,  
 Shader.TileMode.MIRROR);  
  
 RadialGradient radialGradient = **new** RadialGradient(  
 (**float**) (frame.left + frame.width() / 2),  
 (**float**) (*scannerStart* + ***SCANNER\_LINE\_HEIGHT*** / 2),  
 360f,  
 **laserColor**,  
 shadeColor(**laserColor**),  
 Shader.TileMode.MIRROR);  
  
 SweepGradient sweepGradient = **new** SweepGradient(  
 (**float**) (frame.left + frame.width() / 2),  
 (**float**) (*scannerStart* + ***SCANNER\_LINE\_HEIGHT***),  
 shadeColor(**laserColor**),  
 **laserColor**);  
  
 ComposeShader composeShader = **new** ComposeShader(radialGradient, linearGradient, PorterDuff.Mode.ADD);  
  
 **paint**.setShader(radialGradient);  
 **if** (*scannerStart* <= *scannerEnd*) {  
 *//矩形  
// canvas.drawRect(frame.left, scannerStart, frame.right, scannerStart + SCANNER\_LINE\_HEIGHT, paint);  
 //椭圆* RectF rectF = **new** RectF(frame.left + 2 \* ***SCANNER\_LINE\_HEIGHT***, *scannerStart*, frame.right - 2 \* ***SCANNER\_LINE\_HEIGHT***, *scannerStart* + ***SCANNER\_LINE\_HEIGHT***);  
 canvas.drawOval(rectF, **paint**);  
 *scannerStart* += ***SCANNER\_LINE\_MOVE\_DISTANCE***;  
 } **else** {  
 *scannerStart* = frame.top;  
 }  
 **paint**.setShader(**null**);  
 }  
  
 *//处理颜色模糊* **public int** shadeColor(**int** color) {  
 String hax = Integer.toHexString(color);  
 String result = **"20"** + hax.substring(2);  
 **return** Integer.valueOf(result, 16);  
 }  
  
 *// 绘制扫描区边框 Draw a two pixel solid black border inside the framing rect* **private void** drawFrame(Canvas canvas, Rect frame) {  
 **paint**.setColor(**frameColor**);  
 canvas.drawRect(frame.left, frame.top, frame.right + 1, frame.top + 2, **paint**);  
 canvas.drawRect(frame.left, frame.top + 2, frame.left + 2, frame.bottom - 1, **paint**);  
 canvas.drawRect(frame.right - 1, frame.top, frame.right + 1, frame.bottom - 1, **paint**);  
 canvas.drawRect(frame.left, frame.bottom - 1, frame.right + 1, frame.bottom + 1, **paint**);  
 }  
  
 *// 绘制模糊区域 Draw the exterior (i.e. outside the framing rect) darkened* **private void** drawExterior(Canvas canvas, Rect frame, **int** width, **int** height) {  
 **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**);  
 }  
  
 **public void** drawViewfinder() {  
 **resultBitmap** = **null**;  
 invalidate();  
 }  
  
 */\*\*  
 \* Draw a bitmap with the result points highlighted instead of the live scanning display.  
 \*  
 \** ***@param barcode*** *An image of the decoded barcode.  
 \*/* **public void** drawResultBitmap(Bitmap barcode) {  
 **resultBitmap** = barcode;  
 invalidate();  
 }  
  
 **public void** addPossibleResultPoint(ResultPoint point) {  
 **possibleResultPoints**.add(point);  
 }  
  
}

<**manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 package="com.google.zxing"**>  
  
 <**uses-permission android:name="android.permission.INTERNET"** />  
 <**uses-permission android:name="android.permission.VIBRATE"** />  
 <**uses-permission android:name="android.permission.CAMERA"** />  
  
 <**uses-feature android:name="android.hardware.camera.autofocus"** />  
  
</**manifest**>

apply **plugin**: **'com.android.library'**android {  
 compileSdkVersion 28  
  
 defaultConfig {  
 minSdkVersion 16  
 targetSdkVersion 28  
 versionCode 1  
 versionName **"1.0"** testInstrumentationRunner **"android.support.test.runner.AndroidJUnitRunner"** }  
  
 buildTypes {  
 release {  
 minifyEnabled **false** proguardFiles getDefaultProguardFile(**'proguard-android.txt'**), **'proguard-rules.pro'** }  
 }  
  
}  
  
dependencies {  
 implementation fileTree(**dir**: **'libs'**, **include**: [**'\*.jar'**])  
 implementation **'com.android.support:appcompat-v7:28.0.0'** implementation **'com.google.zxing:core:3.4.0'** testImplementation **'junit:junit:4.12'** implementation **'com.android.support.constraint:constraint-layout:1.1.3'** *//沉浸式* implementation rootProject.ext.dependencies[**"barlibrary"**]  
 androidTestImplementation **'com.android.support.test:runner:1.0.2'** androidTestImplementation **'com.android.support.test.espresso:espresso-core:3.0.2'**}

*<?***xml version="1.0" encoding="UTF-8"***?>*<**module external.linked.project.id=":zxing-lib" external.linked.project.path="$MODULE\_DIR$" external.root.project.path="$MODULE\_DIR$/.." external.system.id="GRADLE" type="JAVA\_MODULE" version="4"**>  
 <**component name="FacetManager"**>  
 <**facet type="android-gradle" name="Android-Gradle"**>  
 <**configuration**>  
 <**option name="GRADLE\_PROJECT\_PATH" value=":zxing-lib"** />  
 </**configuration**>  
 </**facet**>  
 <**facet type="android" name="Android"**>  
 <**configuration**>  
 <**option name="SELECTED\_BUILD\_VARIANT" value="debug"** />  
 <**option name="ASSEMBLE\_TASK\_NAME" value="assembleDebug"** />  
 <**option name="COMPILE\_JAVA\_TASK\_NAME" value="compileDebugSources"** />  
 <**afterSyncTasks**>  
 <**task**>generateDebugSources</**task**>  
 </**afterSyncTasks**>  
 <**option name="ALLOW\_USER\_CONFIGURATION" value="false"** />  
 <**option name="MANIFEST\_FILE\_RELATIVE\_PATH" value="/src/main/AndroidManifest.xml"** />  
 <**option name="RES\_FOLDER\_RELATIVE\_PATH" value="/src/main/res"** />  
 <**option name="RES\_FOLDERS\_RELATIVE\_PATH" value="file://$MODULE\_DIR$/src/main/res;file://$MODULE\_DIR$/build/generated/res/rs/debug;file://$MODULE\_DIR$/build/generated/res/resValues/debug"** />  
 <**option name="ASSETS\_FOLDER\_RELATIVE\_PATH" value="/src/main/assets"** />  
 <**option name="PROJECT\_TYPE" value="1"** />  
 </**configuration**>  
 </**facet**>  
 </**component**>  
 <**component name="NewModuleRootManager" LANGUAGE\_LEVEL="JDK\_1\_7"**>  
 <**output url="file://$MODULE\_DIR$/build/intermediates/classes/debug"** />  
 <**output-test url="file://$MODULE\_DIR$/build/intermediates/classes/test/debug"** />  
 <**exclude-output** />  
 <**content url="file://$MODULE\_DIR$"**>  
 <**sourceFolder url="file://$MODULE\_DIR$/build/generated/source/apt/debug" isTestSource="false" generated="true"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/build/generated/source/aidl/debug" isTestSource="false" generated="true"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/build/generated/source/buildConfig/debug" isTestSource="false" generated="true"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/build/generated/source/rs/debug" isTestSource="false" generated="true"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/build/generated/res/rs/debug" type="java-resource"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/build/generated/res/resValues/debug" type="java-resource"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/build/generated/source/apt/androidTest/debug" isTestSource="true" generated="true"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/build/generated/source/aidl/androidTest/debug" isTestSource="true" generated="true"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/build/generated/source/buildConfig/androidTest/debug" isTestSource="true" generated="true"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/build/generated/source/rs/androidTest/debug" isTestSource="true" generated="true"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/build/generated/res/rs/androidTest/debug" type="java-test-resource"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/build/generated/res/resValues/androidTest/debug" type="java-test-resource"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/build/generated/source/apt/test/debug" isTestSource="true" generated="true"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/debug/res" type="java-resource"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/debug/resources" type="java-resource"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/debug/assets" type="java-resource"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/debug/aidl" isTestSource="false"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/debug/java" isTestSource="false"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/debug/rs" isTestSource="false"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/debug/shaders" isTestSource="false"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/androidTestDebug/res" type="java-test-resource"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/androidTestDebug/resources" type="java-test-resource"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/androidTestDebug/assets" type="java-test-resource"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/androidTestDebug/aidl" isTestSource="true"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/androidTestDebug/java" isTestSource="true"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/androidTestDebug/rs" isTestSource="true"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/androidTestDebug/shaders" isTestSource="true"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/testDebug/res" type="java-test-resource"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/testDebug/resources" type="java-test-resource"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/testDebug/assets" type="java-test-resource"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/testDebug/aidl" isTestSource="true"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/testDebug/java" isTestSource="true"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/testDebug/rs" isTestSource="true"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/testDebug/shaders" isTestSource="true"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/main/res" type="java-resource"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/main/resources" type="java-resource"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/main/assets" type="java-resource"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/main/aidl" isTestSource="false"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/main/java" isTestSource="false"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/main/rs" isTestSource="false"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/main/shaders" isTestSource="false"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/androidTest/res" type="java-test-resource"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/androidTest/resources" type="java-test-resource"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/androidTest/assets" type="java-test-resource"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/androidTest/aidl" isTestSource="true"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/androidTest/java" isTestSource="true"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/androidTest/rs" isTestSource="true"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/androidTest/shaders" isTestSource="true"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/test/res" type="java-test-resource"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/test/resources" type="java-test-resource"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/test/assets" type="java-test-resource"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/test/aidl" isTestSource="true"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/test/java" isTestSource="true"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/test/rs" isTestSource="true"** />  
 <**sourceFolder url="file://$MODULE\_DIR$/src/test/shaders" isTestSource="true"** />  
 <**excludeFolder url="file://$MODULE\_DIR$/build/generated/not\_namespaced\_r\_class\_sources"** />  
 <**excludeFolder url="file://$MODULE\_DIR$/build/generated/source/r"** />  
 <**excludeFolder url="file://$MODULE\_DIR$/build/intermediates/attr"** />  
 <**excludeFolder url="file://$MODULE\_DIR$/build/intermediates/bundles"** />  
 <**excludeFolder url="file://$MODULE\_DIR$/build/intermediates/check-manifest"** />  
 <**excludeFolder url="file://$MODULE\_DIR$/build/intermediates/classes"** />  
 <**excludeFolder url="file://$MODULE\_DIR$/build/intermediates/incremental"** />  
 <**excludeFolder url="file://$MODULE\_DIR$/build/intermediates/intermediate-jars"** />  
 <**excludeFolder url="file://$MODULE\_DIR$/build/intermediates/javaPrecompile"** />  
 <**excludeFolder url="file://$MODULE\_DIR$/build/intermediates/jniLibs"** />  
 <**excludeFolder url="file://$MODULE\_DIR$/build/intermediates/manifests"** />  
 <**excludeFolder url="file://$MODULE\_DIR$/build/intermediates/res"** />  
 <**excludeFolder url="file://$MODULE\_DIR$/build/intermediates/rs"** />  
 <**excludeFolder url="file://$MODULE\_DIR$/build/intermediates/shaders"** />  
 <**excludeFolder url="file://$MODULE\_DIR$/build/intermediates/transforms"** />  
 <**excludeFolder url="file://$MODULE\_DIR$/build/outputs"** />  
 <**excludeFolder url="file://$MODULE\_DIR$/build/tmp"** />  
 </**content**>  
 <**orderEntry type="jdk" jdkName="Android API 28 Platform" jdkType="Android SDK"** />  
 <**orderEntry type="sourceFolder" forTests="false"** />  
 <**orderEntry type="library" name="Gradle: com.android.support:support-fragment:28.0.0@aar" level="project"** />  
 <**orderEntry type="library" name="Gradle: com.android.support:localbroadcastmanager:28.0.0@aar" level="project"** />  
 <**orderEntry type="library" name="Gradle: com.android.support:documentfile:28.0.0@aar" level="project"** />  
 <**orderEntry type="library" name="Gradle: com.android.support:animated-vector-drawable:28.0.0@aar" level="project"** />  
 <**orderEntry type="library" name="Gradle: com.android.support:slidingpanelayout:28.0.0@aar" level="project"** />  
 <**orderEntry type="library" name="Gradle: android.arch.lifecycle:viewmodel:1.1.1@aar" level="project"** />  
 <**orderEntry type="library" name="Gradle: com.android.support:appcompat-v7:28.0.0@aar" level="project"** />  
 <**orderEntry type="library" scope="TEST" name="Gradle: com.android.support.test.espresso:espresso-idling-resource:3.0.2@aar" level="project"** />  
 <**orderEntry type="library" name="Gradle: com.android.support:loader:28.0.0@aar" level="project"** />  
 <**orderEntry type="library" name="Gradle: android.arch.core:runtime:1.1.1@aar" level="project"** />  
 <**orderEntry type="library" scope="TEST" name="Gradle: com.android.support.test:runner:1.0.2@aar" level="project"** />  
 <**orderEntry type="library" name="Gradle: android.arch.lifecycle:livedata-core:1.1.1@aar" level="project"** />  
 <**orderEntry type="library" name="Gradle: com.android.support:collections:28.0.0@jar" level="project"** />  
 <**orderEntry type="library" scope="TEST" name="Gradle: com.squareup:javawriter:2.1.1@jar" level="project"** />  
 <**orderEntry type="library" name="Gradle: com.android.support:cursoradapter:28.0.0@aar" level="project"** />  
 <**orderEntry type="library" name="Gradle: android.arch.lifecycle:runtime:1.1.1@aar" level="project"** />  
 <**orderEntry type="library" scope="TEST" name="Gradle: com.android.support.test.espresso:espresso-core:3.0.2@aar" level="project"** />  
 <**orderEntry type="library" name="Gradle: com.android.support:support-compat:28.0.0@aar" level="project"** />  
 <**orderEntry type="library" name="Gradle: com.android.support:support-core-ui:28.0.0@aar" level="project"** />  
 <**orderEntry type="library" name="Gradle: com.android.support:asynclayoutinflater:28.0.0@aar" level="project"** />  
 <**orderEntry type="library" name="Gradle: com.android.support.constraint:constraint-layout:1.1.3@aar" level="project"** />  
 <**orderEntry type="library" name="Gradle: com.android.support:print:28.0.0@aar" level="project"** />  
 <**orderEntry type="library" scope="TEST" name="Gradle: com.google.code.findbugs:jsr305:2.0.1@jar" level="project"** />  
 <**orderEntry type="library" scope="TEST" name="Gradle: javax.inject:javax.inject:1@jar" level="project"** />  
 <**orderEntry type="library" name="Gradle: android.arch.core:common:1.1.1@jar" level="project"** />  
 <**orderEntry type="library" name="Gradle: com.android.support:versionedparcelable:28.0.0@aar" level="project"** />  
 <**orderEntry type="library" name="Gradle: com.google.zxing:core:3.4.0@jar" level="project"** />  
 <**orderEntry type="library" scope="TEST" name="Gradle: junit:junit:4.12@jar" level="project"** />  
 <**orderEntry type="library" scope="TEST" name="Gradle: org.hamcrest:hamcrest-core:1.3@jar" level="project"** />  
 <**orderEntry type="library" name="Gradle: com.android.support:viewpager:28.0.0@aar" level="project"** />  
 <**orderEntry type="library" name="Gradle: com.android.support:support-core-utils:28.0.0@aar" level="project"** />  
 <**orderEntry type="library" name="Gradle: com.android.support:support-vector-drawable:28.0.0@aar" level="project"** />  
 <**orderEntry type="library" scope="TEST" name="Gradle: com.android.support.test:monitor:1.0.2@aar" level="project"** />  
 <**orderEntry type="library" scope="TEST" name="Gradle: org.hamcrest:hamcrest-library:1.3@jar" level="project"** />  
 <**orderEntry type="library" scope="TEST" name="Gradle: org.hamcrest:hamcrest-integration:1.3@jar" level="project"** />  
 <**orderEntry type="library" name="Gradle: com.android.support:support-annotations:28.0.0@jar" level="project"** />  
 <**orderEntry type="library" name="Gradle: com.gyf.barlibrary:barlibrary:2.3.0@aar" level="project"** />  
 <**orderEntry type="library" name="Gradle: android.arch.lifecycle:common:1.1.1@jar" level="project"** />  
 <**orderEntry type="library" name="Gradle: com.android.support:interpolator:28.0.0@aar" level="project"** />  
 <**orderEntry type="library" name="Gradle: android.arch.lifecycle:livedata:1.1.1@aar" level="project"** />  
 <**orderEntry type="library" name="Gradle: com.android.support:drawerlayout:28.0.0@aar" level="project"** />  
 <**orderEntry type="library" name="Gradle: com.android.support:coordinatorlayout:28.0.0@aar" level="project"** />  
 <**orderEntry type="library" name="Gradle: com.android.support:customview:28.0.0@aar" level="project"** />  
 <**orderEntry type="library" name="Gradle: com.android.support:swiperefreshlayout:28.0.0@aar" level="project"** />  
 <**orderEntry type="library" name="Gradle: com.android.support.constraint:constraint-layout-solver:1.1.3@jar" level="project"** />  
 <**orderEntry type="library" scope="TEST" name="Gradle: net.sf.kxml:kxml2:2.3.0@jar" level="project"** />  
 </**component**>  
</**module**>

apply **from**: **"config.gradle"**buildscript {  
   
 repositories {  
 google()  
 jcenter()  
 maven { url **"https://jitpack.io"** }  
 maven { url **"https://dl.bintray.com/thelasterstar/maven/"** }  
 }  
 dependencies {  
 classpath **'com.android.tools.build:gradle:3.0.1'** classpath **"io.objectbox:objectbox-gradle-plugin:2.3.4"** *// NOTE: Do not place your application dependencies here; they belong  
 // in the individual module build.gradle files* }  
}  
  
allprojects {  
 repositories {  
 google()  
 jcenter()  
 maven { url **"https://jitpack.io"** }  
 maven { url **"https://dl.bintray.com/thelasterstar/maven/"** }  
 }  
}  
  
task clean(**type**: Delete) {  
 delete rootProject.buildDir  
}

ext {  
 android = [  
 **compileSdkVersion**: 28,  
 **applicationId** : **"com.jlkf.ks\_link\_wm126"**,  
 **minSdkVersion** : 19,  
 **targetSdkVersion** : 28,  
 **versionCode** : 1,  
 **versionName** : **"1.0.0"** ]  
  
 dependVersion = [  
 **support**: **"28.0.0"** ]  
  
 dependencies = [  
 *//android-support* **"support"** : **"com.android.support"**,  
 **"support-v4"** : **"com.android.support:support-v4:**${dependVersion.**support**}**"**,  
 **"appcompat-v7"** : **"com.android.support:appcompat-v7:**${dependVersion.**support**}**"**,  
 **"design"** : **"com.android.support:design:**${dependVersion.**support**}**"**,  
 **"recyclerview"** : **"com.android.support:recyclerview-v7:**${dependVersion.**support**}**"**,  
 **"cardview"** : **"com.android.support:cardview-v7:**${dependVersion.**support**}**"**,  
  
 **"multidex"** : **"com.android.support:multidex:1.0.3"**,  
 *//http* **"retrofit2"** : **"com.squareup.retrofit2:retrofit:2.5.0"**,  
 **"converter-scalars"** : **"com.squareup.retrofit2:converter-scalars:2.1.0"**,  
 **"converter-gson"** : **"com.squareup.retrofit2:converter-gson:2.5.0"**,  
 **"adapter-rxjava2"** : **"com.squareup.retrofit2:adapter-rxjava2:2.4.0"**,  
  
 *//rxjava* **"rxjava2"** : **"io.reactivex.rxjava2:rxjava:2.2.8"**,  
 **"rxandroid"** : **"io.reactivex.rxjava2:rxandroid:2.1.1"**,  
 **"autodispose"** : **"com.uber.autodispose:autodispose:1.0.0-RC3"**,  
 **"autodispose-android-archcomponents"**: **"com.uber.autodispose:autodispose-android-archcomponents:1.0.0-RC3"**,  
  
 *//dagger* **"dagger"** : **"com.google.dagger:dagger:2.23.2"**,  
 **"dagger-compiler"** : **"com.google.dagger:dagger-compiler:2.23.2"**,  
 **"dagger-android-compiler"** : **"com.google.dagger:dagger-android-processor:2.23.2"**,  
  
 *//bugly* **"bugly"** : **"com.tencent.bugly:crashreport:3.0.0"**,  
  
 *//ui* **"constraint-layout"** : **"com.android.support.constraint:constraint-layout:1.1.3"**,  
 **"barlibrary"** : **"com.gyf.barlibrary:barlibrary:2.3.0"**,  
 **"BaseRecyclerViewAdapterHelper"** : **"com.github.CymChad:BaseRecyclerViewAdapterHelper:2.9.41"**,  
 **"glide"** : **"com.github.bumptech.glide:glide:4.9.0"**,  
 **"glide-compiler"** : **"com.github.bumptech.glide:compiler:4.9.0"**,  
 **"SmartRefreshLayout"** : **"com.scwang.smartrefresh:SmartRefreshLayout:1.1.0-alpha-28"**,  
 **"SmartRefreshHeader"** : **"com.scwang.smartrefresh:SmartRefreshHeader:1.1.0-alpha-28"**,  
 **"menuRecylcerView"** : **"com.yanzhenjie.recyclerview:support:1.3.2"**,  
 **"toast"** : **"com.hjq:toast:6.0"**,  
 *//弹出框选择器* **"PickerView"** : **'com.contrarywind:Android-PickerView:4.1.8'**,  
 **"objectbox"** : **"io.objectbox:objectbox-android:2.3.4"**,  
 **"objectbrowser"** : **"io.objectbox:objectbox-android-objectbrowser:2.3.4"**,  
 **"kprogresshud"** : **"com.kaopiz:kprogresshud:1.2.0"**,  
  
 ]  
}