第 19 章

19 相机I：取景器

记录办公室陋习时，如果能以现场照片佐证，问题解决起来就会容易很多。接下来的两章，使用系统自带的Camera API，为CriminalIntent应用添加拍摄作案现场照片的功能。

Camera API功能虽然强大，但要用好它并不容易。不仅要编写大量的实现代码，还要苦苦挣扎着学习和理解一大堆全新概念。因此，很容易产生的一个疑问就是："只是拍张快照，难道就没有便捷的标准接口可以使用吗？"

答案是肯定的。我们可以通过隐式intent与照相机进行交互。大多数Android设备都会内置相机应用。相机应用会自动侦听由`MediaStore.ACTION_IMAGE_CAPTURE`创建的intent。第21章将介绍如何使用隐式的intent。

很不幸，截止本书写作时，在大多数设备上，隐式intent的相机接口有一个bug，会导致用户无法保存全尺寸的照片。因此，对于那些只需要缩略图的应用来说，隐式intent完全可以满足要求。然而，CriminalIntent应用需要的是全尺寸的作案现场图片，别无选择，我们只能去学习使用Camera API了。

本章将要创建一个基于fragment的activity，然后使用`SurfaceView`类配合相机硬件来实时展示现场的视频预览，如图19-1所示。

图19-1　viewfinder中的相机实时预览

图19-2展示了稍后会创建的新对象。

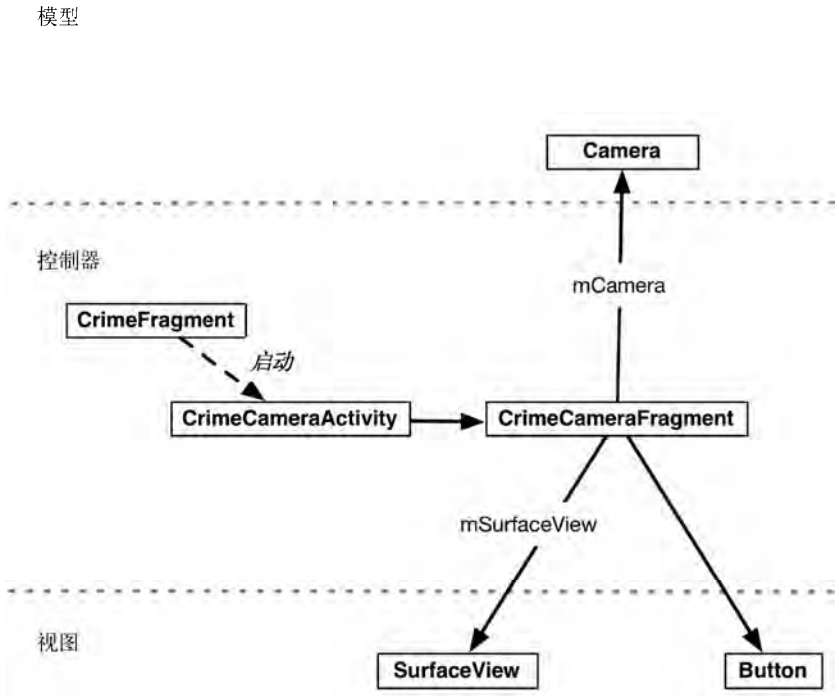

图19-2 CriminalIntent应用相机部分的对象图解

Camera实例提供了对设备相机硬件级别的调用。相机是一种独占性资源：一次只能有一个activity能够调用相机。

SurfaceView实例是相机的取景器。SurfaceView是一种特殊的视图，可直接将要显示的内容渲染输出到设备的屏幕上。

首先，我们会创建CrimeCameraFragment视图的布局、CrimeCameraFragment类及CrimeCameraActivity类。然后，在CrimeCameraFragment类中创建并管理一个用来拍照的取景器。最后，配置CrimeFragment启动CrimeCameraActivity实例。

19.1 创建 Fragment 布局

以FrameLayout为根元素，创建一个名为fragment_crime_camera.xml的布局文件。然后参照图19-3完成各组件的添加。

可以看到，新建布局文件中，FrameLayout只包含唯一一个LinearLayout子元素，这会导致Android Lint报出没有用处的LinearLayout警告信息。暂时忽略它，第20章将为FrameLayout添加第二个子视图。

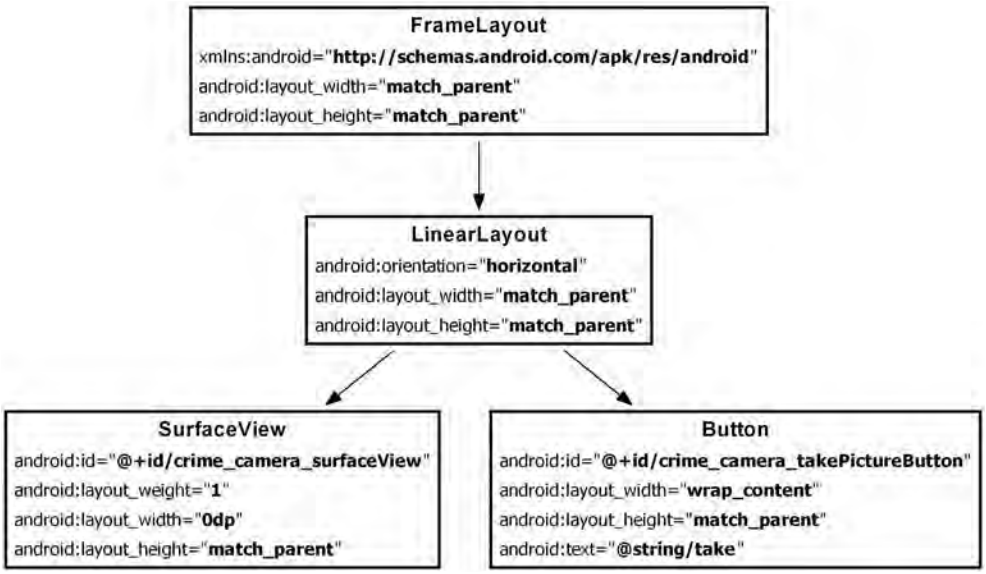

图19-3　CrimeCameraFragment的布局（fragment_crime_camera.xml）

在LinearLayout组件定义中，我们使用layout_width与layout_weight的属性组合来布置它的子视图。因为设置的android:layout_width="wrap_content"属性值，Button组件仅占用了自己所需的空间，而按照android:layout_width="0dp"的属性值，SurfaceView组件不占用任何空间。不过从剩余空间的角度来说，因为使用了layout_weight属性，所以SurfaceView组件使用了Button组件以外的全部空间。

图19-4展示了新建布局的预览界面。

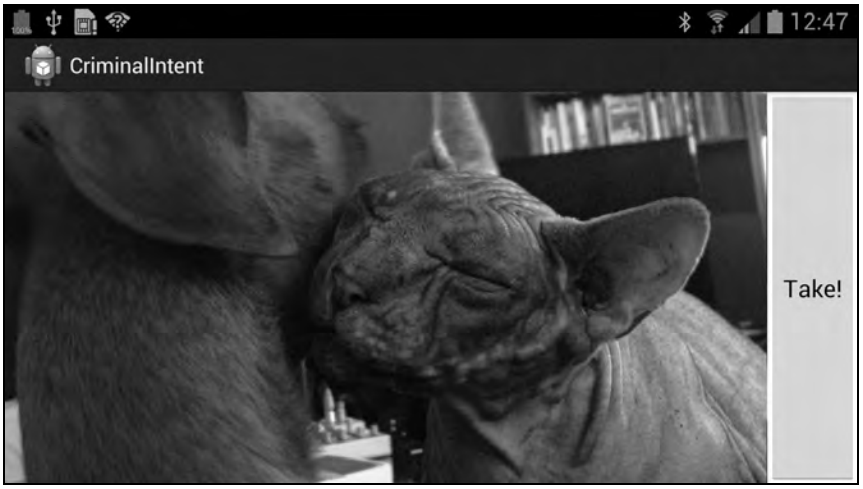

图19-4　取景器与按钮界面

在strings.xml中，为按钮的文本添加字符串资源，如代码清单19-1所示。

代码清单19-1 为相机按钮添加字符串资源（strings.xml）

```
    ...
    <string name="show_subtitle">Show Subtitle</string>
    <string name="subtitle">Sometimes tolerance is not a virtue.</string>
    <string name="take">Take!</string>

</resources>
```

19.2 创建 CrimeCameraFragment

以android.support.v4.app.Fragment为超类，创建一个名为CrimeCameraFragment的新类。在随后打开的CrimeCameraFragment.java中，增加如代码清单19-2所示的变量。然后，覆盖onCreateView(...)方法，实例化布局并引用各组件。现在，先为按钮设置一个事件监听器，用户点击按钮时，退出当前托管activity并回到列表项明细界面。

代码清单19-2 初始相机fragment类（CrimeCameraFragment.java）

```
public class CrimeCameraFragment extends Fragment {
    private static final String TAG = "CrimeCameraFragment";

    private Camera mCamera;
    private SurfaceView mSurfaceView;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime_camera, parent, false);

        Button takePictureButton = (Button)v
            .findViewById(R.id.crime_camera_takePictureButton);
        takePictureButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                getActivity().finish();
            }
        });

        mSurfaceView = (SurfaceView)v.findViewById(R.id.crime_camera_surfaceView);

        return v;
    }
}
```

19.3 创建 CrimeCameraActivity

以SingleFragmentActivity为超类，创建一个名为CrimeCameraActivity的新类。在CrimeCameraActivity.java中，覆盖createFragment()方法返回一个CrimeCameraFragment，如代码清单19-3所示。

代码清单19-3　创建相机的activity类（CrimeCameraActivity.java）

```
public class CrimeCameraActivity extends SingleFragmentActivity {
    @Override
    protected Fragment createFragment() {
        return new CrimeCameraFragment();
    }
}
```

声明activity并添加允许调用相机设置

接下来，除了声明CrimeCameraActivity类外，还需要在配置文件中增加uses-permission元素节点以获得使用相机的权限。

参照代码清单19-4，更新AndroidManifest.xml配置文件。

代码清单19-4　声明相机的activity并添加允许调用相机设置（AndroidManifest.xml）

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.bignerdranch.android.criminalintent"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="16"/>
  <uses-permission android:name="android.permission.CAMERA" />
  <uses-feature android:name="android.hardware.camera" />

  <application

          ...

    <activity android:name=".CrimeCameraActivity"
      android:screenOrientation="landscape"
      android:label="@string/app_name">
    </activity>

  </application>

</manifest>
```

uses-feature元素用来指定应用使用的某项特色设备功能。通过android.hardware.camera特色功能的设置，可以保证只有那些配备相机功能的设备才能够看到你发布在Google Play上的应用。

注意，在activity的声明中，为了防止用户在调整角度取景拍照时，设备屏幕随意旋转，我们使用android:screenOrientation属性强制activity界面总是以水平模式展现。

属性android:screenOrientation还有很多其他可选属性值。例如，可以设置activity与其父类保持一致的显示方位，也可以选择在设备处于不同方向时，根据设备感应器感应只以水平模式显示。可以查看开发文档的<activity>元素获取更多其他可用属性值信息。

19.4 使用相机 API

目前为止，我们一直在处理基本的activity创建工作。现在，是时候学习理解相机相关的概念并着手使用相机类了。

19.4.1 打开并释放相机

首先，来进行相机资源的管理。我们已经在CrimeCameraFragment中添加了一个Camera实例。相机是一种系统级别的重要资源，因此，很关键的一点就是，需要时使用，用完及时释放。如果忘记释放，除非重启设备，否则其他应用将无法使用相机。

以下是将要用来管理Camera实例的方法：

```
public static Camera open(int cameraId)
public static Camera open()
public final void release()
```

其中open(int)方法是在API级别第9级引入的，因此，如果设备的API级别小于第9级，那么就只能使用不带参数的open()方法。

在CrimeCameraFragment生命周期中，我们应该在onResume()和onPause()回调方法中打开和释放相机资源。这两个方法可确定用户能够同fragment视图交互的时间边界，只有在用户能够同fragment视图交互时，相机才可以使用。（注意，即使fragment首次开始出现在屏幕上，onResume()方法也会被调用。）

在CrimeCameraFragment.onResume()方法中，使用Camera.open(int)静态方法来初始化相机。然后传入参数0打开设备可用的第一相机（通常指的是后置相机）。如果设备没有后置相机（如Nexus 7机型），那么前置相机将会打开。

对于API级别第8级的设备来说，需要调用不带参数的Camera.open()方法。针对onResume()方法使用@TargetApi注解保护，然后检查设备的编译版本，根据设备不同的版本号确定是调用Camera.open(0)方法还是Camera.open()方法，如代码清单19-5所示。

代码清单19-5 在onResume()方法中打开相机（CrimeCameraFragment.java）

```
@TargetApi(9)
@Override
public void onResume() {
    super.onResume();
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.GINGERBREAD) {
        mCamera = Camera.open(0);
    } else {
        mCamera = Camera.open();
    }
}
```

（Android Lint可能会警告说正在主线程上打开相机。这属于正常警示，在学习第26章介绍的多线程相关知识前，暂时忽略它们。）

Fragment被销毁时，应该及时释放相机资源，以便于其他应用需要时可以使用。覆盖

onPause()方法释放相机资源，如代码清单19-6所示。

代码清单19-6　实现生命周期方法（CrimeCameraFragment.java）

```
public void onResume() {
    super.onResume();
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.GINGERBREAD) {
        mCamera = Camera.open(0);
    } else {
        mCamera = Camera.open();
    }
}

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

    if (mCamera != null) {
        mCamera.release();
        mCamera = null;
    }
}
```

注意，调用release()方法之前，首先要确保存在Camera实例。调用相机相关代码前，都应该作这样的检查。要知道，即使是请求获取相机的使用权限，相机也可能无法获得。比如，另一个activity正在使用它，或者因为是虚拟设备，相机根本就不存在。总之，不管是什么原因，相机实例不存在时，空值检查可以防止应用意外崩溃。

19.4.2　SurfaceView、SurfaceHolder与Surface

SurfaceView类实现了SurfaceHolder接口。在CrimeCameraFragment.java中，增加以下代码获取SurfaceView的SurfaceHolder实例，如代码清单19-7所示。

代码清单19-7　获取SurfaceHolder实例（CrimeCameraFragment.java）

```
@Override
@SuppressWarnings("deprecation")
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {

    ...

    mSurfaceView = (SurfaceView)v.findViewById(R.id.crime_camera_surfaceView);
    SurfaceHolder holder = mSurfaceView.getHolder();
    // setType() and SURFACE_TYPE_PUSH_BUFFERS are both deprecated,
    // but are required for Camera preview to work on pre-3.0 devices.
    holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);

    return v;
}
```

setType(...)方法和SURFACE_TYPE_PUSH_BUFFERS常量都已被弃用，因此，对于废弃代码，编译器会提示警告信息。Eclipse也会将弃用代码打上删除线标示出来。

既然已经是弃用的代码，为什么还要使用它们呢？在运行Honeycomb之前版本的设备上，相机预览能够工作离不开setType(...)方法以及SURFACE_TYPE_PUSH_BUFFERS常量的支持。在代码清单19-7中，我们使用@SuppressWarnings注解来消除弃用代码相关的警告信息。这样处理似乎比较怪异，但这是处理弃用代码与兼容性问题的最佳方式。有关Android中弃用代码的更深入讨论，请参见第20章末尾的深入学习部分。

SurfaceHolder是我们与Surface对象联系的纽带。Surface对象代表着原始像素数据的缓冲区。

Surface对象也有生命周期：SurfaceView出现在屏幕上时，会创建Surface；SurfaceView从屏幕上消失时，Surface随即被销毁。Surface不存在时，必须保证没有任何内容要在它上面绘制。

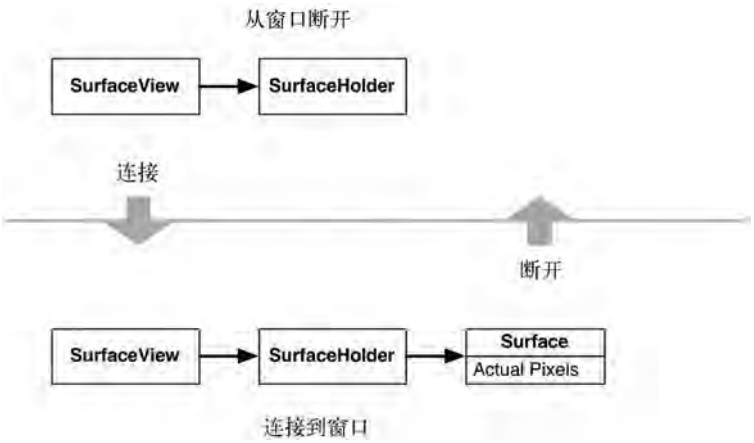

图19-5 SurfaceView、SurfaceHolder及Surface

不像其他视图对象，SurfaceView及其协同工作对象都不会自我绘制内容。对于任何想将内容绘制到Surface缓冲区的对象，我们称其为Surface的*客户端*。在CrimeCameraFragment类中，Camera实例是Surface的客户端。

记住，Surface不存在时，必须保证没有任何内容要在Surface的缓冲区中绘制。图19-6展示了需要处理的两种可能情况，Surface创建完成后，需要将Camera连接到SurfaceHolder上；Surface销毁后，再将Camera从SurfaceHolder上断开。

为完成以上任务，SurfaceHolder提供了另一个接口：SurfaceHolder.Callback。该接口监听Surface生命周期中的事件，这样就可以控制Surface与其客户端协同工作。

以下是SurfaceHolder.Callback接口中的三个方法。

- public abstract void surfaceCreated(SurfaceHolder holder)
  包含SurfaceView的视图层级结构被放到屏幕上时调用该方法。这里也是Surface与其客户端进行关联的地方。

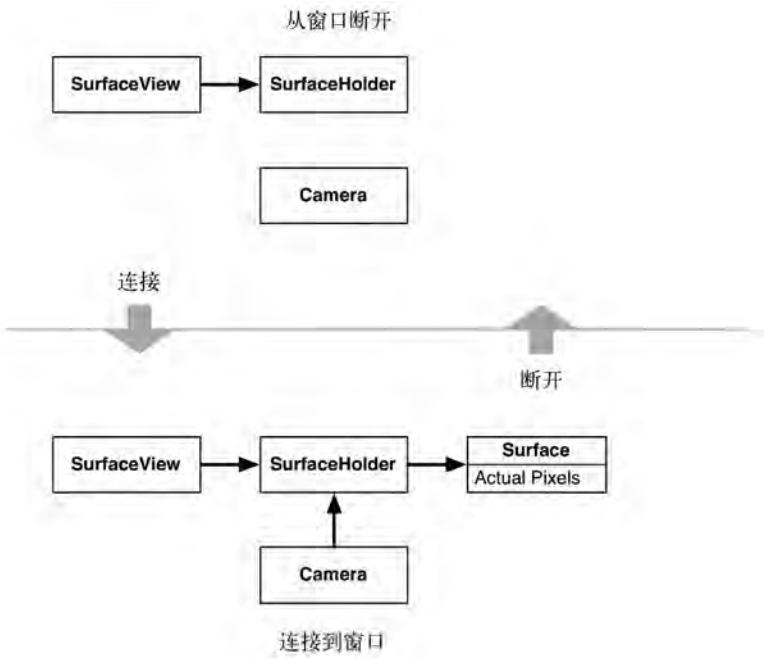

图19-6　理想的工作状态

- `public abstract void surfaceChanged(SurfaceHolder holder, int format, int width, int height)`
  Surface首次显示在屏幕上时调用该方法。通过传入的参数，可以知道Surface的像素格式以及它的宽度和高度。该方法内可以通知Surface的客户端，有多大的绘制区域可以使用。
- `public abstract void surfaceDestroyed(SurfaceHolder holder)`
  SurfaceView从屏幕上移除时，Surface也随即被销毁。通过该方法，可以通知Surface的客户端停止使用Surface。

以下是用来响应Surface生命周期事件的三个Camera方法。

- `public final void setPreviewDisplay(SurfaceHolder holder)`
  该方法用来连接Camera与Surface。我们将在surfaceCreated()方法中调用它。
- `public final void startPreview()`
  该方法用来在Surface上绘制帧。我们将在surfaceChanged(...)方法中调用它。
- `public final void stopPreview()`
  该方法用来停止在Surface上绘制帧。我们将在surfaceDestroyed()方法中调用它。

在CrimeCameraFragment.java中，实现SurfaceHolder.Callback接口，使得相机预览与Surface生命周期方法能够协同工作，如代码清单19-8所示。

代码清单19-8 实现SurfaceHolder.Callback接口（CrimeCameraFragment.java）

```
        ...
        SurfaceHolder holder = mSurfaceView.getHolder();
        // setType() and SURFACE_TYPE_PUSH_BUFFERS are both deprecated,
        // but are required for Camera preview to work on pre-3.0 devices.
        holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);

        holder.addCallback(new SurfaceHolder.Callback() {

            public void surfaceCreated(SurfaceHolder holder) {
                // Tell the camera to use this surface as its preview area
                try {
                    if (mCamera != null) {
                        mCamera.setPreviewDisplay(holder);
                    }
                } catch (IOException exception) {
                    Log.e(TAG, "Error setting up preview display", exception);
                }
            }

            public void surfaceDestroyed(SurfaceHolder holder) {
                // We can no longer display on this surface, so stop the preview.
                if (mCamera != null) {
                    mCamera.stopPreview();
                }
            }

            public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {
                if (mCamera == null) return;

                // The surface has changed size; update the camera preview size
                Camera.Parameters parameters = mCamera.getParameters();
                Size s = null; // To be reset in the next section
                parameters.setPreviewSize(s.width, s.height);
                mCamera.setParameters(parameters);
                try {
                    mCamera.startPreview();
                } catch (Exception e) {
                    Log.e(TAG, "Could not start preview", e);
                    mCamera.release();
                    mCamera = null;
                }
            }
        });

        return v;
    }
```

注意，预览启动失败时，我们通过异常控制机制释放了相机资源。任何时候，打开相机并完成任务后，必须记得及时释放它，即使是在发生异常时。

在surfaceChanged(...)实现方法中，我们设置相机预览大小为空。在确定可接受的预览大小前，这只是一个临时赋值。相机的预览大小不能随意设置，如果设置了不可接受的值，应用将会抛出异常。

19.4.3　确定预览界面大小

首先，通过Camera.Parameters嵌套类获取系统支持的相机预览尺寸列表。Camera.Parameters类包括下列方法：

```
public List<Camera.Size> getSupportedPreviewSizes()
```

该方法返回android.hardware.Camera.Size类实例的一个列表，每个实例封装了一个具体的图片宽高尺寸。

要找到适合Surface的预览尺寸，可以将列表中的预览尺寸与传入surfaceChanged(...)方法的Surface的宽、高进行比较。

在CrimeCameraFragment类中，添加代码清单19-9所示的方法。该方法接受一组预览尺寸，然后找出具有最大数目像素的尺寸。要说明的是，这里计算最佳尺寸的实现代码并不优雅，但它能够很好地满足我们的使用需求。

代码清单19-9　找出设备支持的最佳尺寸（CrimeCameraFragment.java）

```
/** A simple algorithm to get the largest size available. For a more
 * robust version, see CameraPreview.java in the ApiDemos
 * sample app from Android. */
private Size getBestSupportedSize(List<Size> sizes, int width, int height) {
    Size bestSize = sizes.get(0);
    int largestArea = bestSize.width * bestSize.height;
    for (Size s : sizes) {
        int area = s.width * s.height;
        if (area > largestArea) {
            bestSize = s;
            largestArea = area;
        }
    }
    return bestSize;
}
```

在surfaceChanged(...)方法里调用该方法设置预览尺寸，如代码清单19-10所示。

代码清单19-10　调用getBestSupportedSize(...)方法（CrimeCameraFragment.java）

```
...
holder.addCallback(new SurfaceHolder.Callback() {

    ...

    public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {
        // The surface has changed size; update the camera preview size
        Camera.Parameters parameters = mCamera.getParameters();
        Size s = null;
        Size s = getBestSupportedSize(parameters.getSupportedPreviewSizes(), w, h);
        parameters.setPreviewSize(s.width, s.height);
        mCamera.setParameters(parameters);
        try {
            mCamera.startPreview();
        } catch (Exception e) {
            Log.e(TAG, "Could not start preview", e);
            mCamera.release();
```

```
                mCamera = null;
            }
        }
    });
```

19.4.4 启动 CrimeCameraActivity

要使用取景器，需要在CrimeFragment用户界面添加一个相机调用按钮。单击相机按钮，CrimeFragment将启动一个CrimeCameraActivity实例。图19-7展示了已添加相机按钮的CrimeFragment视图界面。

图19-7 添加了相机按钮的CrimeFragment

要实现图19-7所示的组件重排后的用户界面，需要添加三个LinearLayout和一个ImageButton。参照图19-8完成对CrimeFragment默认布局的调整。

参照图19-9完成类似的水平布局调整。

在CrimeFragment类中，新增一个成员变量，通过资源ID引用图片按钮，然后为其设置OnClickListener，启动CrimeCameraActivity，如代码清单19-11所示。

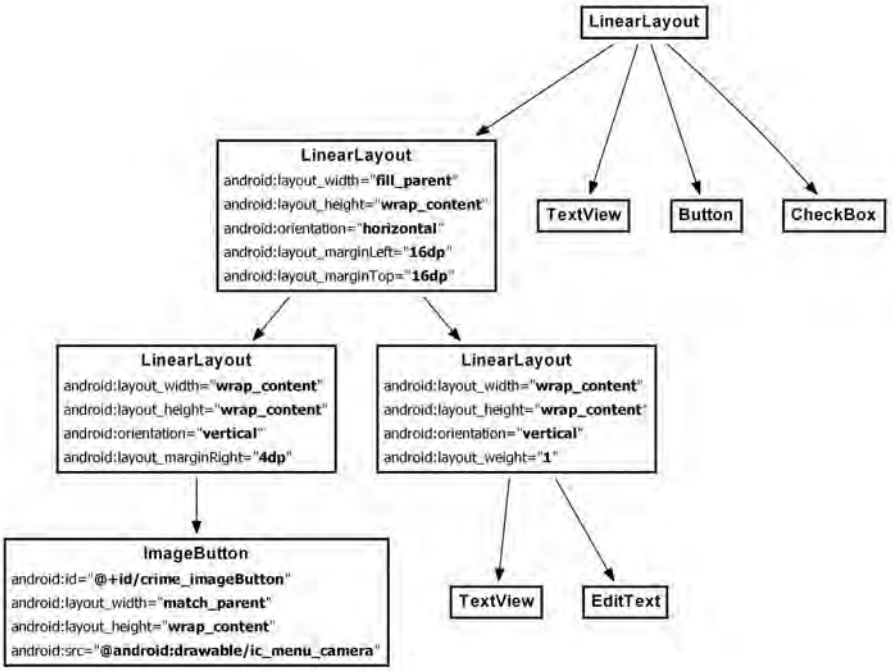

图19-8　添加相机按钮并重新布置布局（layout/fragment_crime.xml）

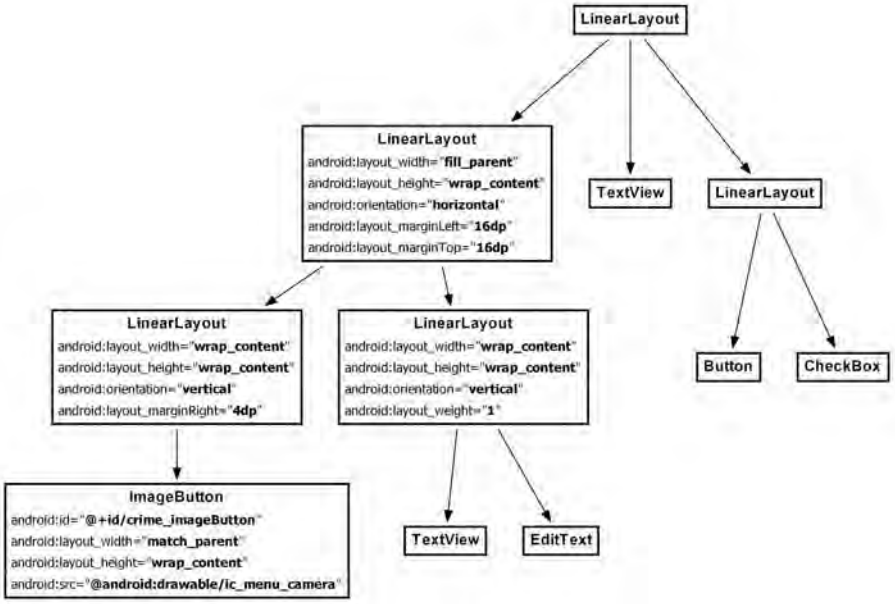

图19-9　添加相机按钮并重新布置布局（layout-land/fragment_crime.xml）

代码清单19-11 启动CrimeCameraActivity（CrimeFragment.java）

```
public class CrimeFragment extends Fragment {
    ...
    private ImageButton mPhotoButton;

    ...
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime, parent, false);

        ...

        mPhotoButton = (ImageButton)v.findViewById(R.id.crime_imageButton);
        mPhotoButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent i = new Intent(getActivity(), CrimeCameraActivity.class);
                startActivity(i);
            }
        });

        return v;
    }
```

对于不带相机的设备，拍照按钮（mPhotoButton）应该禁用。可以查询PackageManager确认设备是否带有相机。在onCreateView(...)方法中，针对没有相机的设备，添加禁用拍照按钮的代码，如代码清单19-12所示。

代码清单19-12 检查设备是否带有相机（CrimeFragment.java）

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_crime, parent, false);

    ...

    mPhotoButton = (ImageButton)v.findViewById(R.id.crime_imageButton);
    mPhotoButton.setOnClickListener(new View.OnClickListener() {
        ...
    });

    // If camera is not available, disable camera functionality
    PackageManager pm = getActivity().getPackageManager();
    boolean hasACamera = pm.hasSystemFeature(PackageManager.FEATURE_CAMERA) ||
            pm.hasSystemFeature(PackageManager.FEATURE_CAMERA_FRONT) ||
            Build.VERSION.SDK_INT < Build.VERSION_CODES.GINGERBREAD ||
            Camera.getNumberOfCameras() > 0;
    if (!hasACamera) {
            mPhotoButton.setEnabled(false);
        }


        ...
    }
```

获取到PackageManager后，调用hasSystemFeature(String)方法并传入表示设备特色功能的常量。FEATURE_CAMERA常量代表后置相机，而FEATURE_CAMERA_FRONT常量代表前置相机。对于没有相机的设备，调用ImageButton按钮的setEnabled(false)属性方法。

运行CriminalIntent应用。查看某项Crime明细记录，然后点击拍照按钮查看相机实时预览画面。拍照功能将会在下一章完成，现在点击Take!按钮将返回CrimeFragment视图，如图19-10所示。

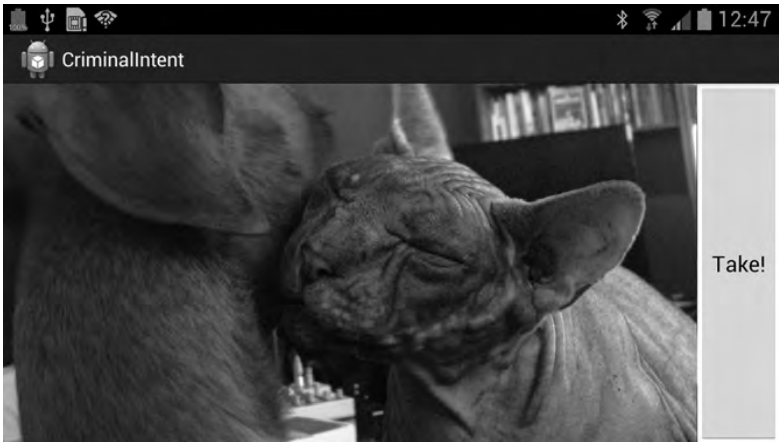

图19-10　来自相机的实时预览画面

前面我们已经在配置文件中强制CrimeCameraActivity界面总是以水平模式展现。尝试旋转设备，可以看到，即使设备处于竖直模式，预览和拍照按钮都被锁定以水平模式展现了。

隐藏状态栏和操作栏

如图19-10所示，activity的操作栏和状态栏遮挡了部分取景器窗口。一般来说，用户只关注取景器中的画面，而且也不会在拍照界面停留很久，因此，操作栏和状态栏不仅没有什么用处，甚至还会妨碍拍照取景，如果能隐藏它们那最好不过了。

有趣的是，我们只能在CrimeCameraActivity中而不能在CrimeCameraFragment中隐藏操作栏和状态栏。打开CrimeCameraActivity.java文件，参照代码清单19-13，在onCreate(Bundle)方法中添加隐藏功能代码。

代码清单19-13　配置activity（CrimeCameraActivity.java）

```
public class CrimeCameraActivity extends SingleFragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        // Hide the window title.
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        // Hide the status bar and other OS-level chrome
        getWindow().addFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN);

        super.onCreate(savedInstanceState);
    }
```

```
    @Override
    protected Fragment createFragment() {
        return new CrimeCameraFragment();
    }
}
```

为什么必须在activity中实现隐藏呢？在调用`Activity.setContentView(...)`方法（该方法是在`CrimeCameraActivity`类的`onCreate(Bundle)`超类版本方法中被调用的。）创建activity视图之前，就必须调用`requestWindowFeature(...)`方法及`addFlags(...)`方法。而fragment无法在其托管activity视图创建之前添加，因此，必须在activity里调用隐藏操作栏和状态栏的相关方法。

再次运行CriminalIntent应用。现在看到的是一个没有遮挡的取景器窗口，如图19-11所示。

图19-11 隐藏了状态栏和操作栏的activity画面

下一章将介绍更多camera API相关内容，实现在本地保存图像文件并在CrimeFragment视图中显示。

19.5 深入学习：以命令行的方式运行 activity

本章，通过在`CrimeFragment`界面添加拍照按钮启动`CrimeCameraActivity`，我们可以快速地测试相机调用代码。不过，有时候，在activity代码整合到应用之前，我们可能就需要测试新activity代码。

Android提供的一个快捷但不完善的方法是：修改配置文件中activity声明节点的`<intent-filter>`元素设置，替换启动activity为需要测试的activity。这样，应用启动时，要测试的activity就会出现。然而，这种方法有个缺点：在恢复原来的启动activity之前，可能不能使用应用的其他一些功能。

实际开发时，替换启动activity的方式并不一定总是合适的。例如，共同开发同一应用时，修改启动activity的方式就会给团队其他人员的测试带来麻烦，令人生厌。不过，Android考虑得很

全面，它还提供了另外一种好办法：使用adb工具从命令行启动activity。

要从命令行启动activity，首先要导出activity。打开AndroidManifest.xml配置文件，将下列属性设置添加到CrimeCameraActivity的activity声明中：

```
<activity android:name=".CrimeCameraActivity"
  android:exported="true"
  android:screenOrientation="landscape"
  android:label="@string/app_name">
</activity>
```

默认情况下，某个应用的activity只能从自己的应用里启动。将android:exported属性值设为true相当于告诉Android，其他应用也可以启动指定应用的activity。（如果将intent过滤器添加到activity的声明中，该activity的android:exported属性值会被自动设为true。）

接下来，在Android SDK安装目录的platform-tools子目录下找到adb工具。建议将platform-tools和tools子目录添加到命令行shell的路径中。

adb工具（Android Debug Bridge）是命令行迷的最爱。使用adb工具可以完成很多原来一直由Eclipse处理的事情。例如，监控LogCat，在设备上打开shell，浏览文件系统，上传下载文件，列出已连接设备以及重置adb。真是一个实用的好工具。

adb也可以用于多个设备，但只有一台设备运行时，使用起来会更容易。现在，关闭或断开任何其他模拟器或设备，如平常一样，在一台测试设备上运行CriminalIntent应用，然后使用下列神奇的语句从命令行启动CrimeCameraActivity。

```
$ adb shell am start -n com.bignerdranch.android.criminalintent/.CrimeCameraActivity
Starting: Intent { cmp=com.bignerdranch.android.criminalintent/.CrimeCameraActivity }
```

执行以上命令后，应该可以看到CrimeCameraActivity在模拟器或设备上运行了。以上命令语句是在设备上的shell中运行命令的一种快捷方式。它的实际运行结果与以下命令的执行结果完全相同：

```
$ adb shell

shell@android:/ $ am start -n com.bignerdranch.android.criminalintent/.CrimeCameraAct\
ivity
```

am（activity manager）是一个在设备上运行的命令行程序。它支持启动和停止Android组件（component）并从命令行发送intent。可以运行adb shell am指令，查看am工具能够完成的所有任务。