第 20 章

相机 II：拍摄并处理照片

本章将从相机预览里拍摄照片并保存为JPEG格式的本地文件。然后，将照片与Crime关联起来并显示在CrimeFragment的视图中。如果需要，用户也可以选择在DialogFragment中查看大尺寸版本的图片，如图20-1所示。

图20-1　Crime的缩略图以及大尺寸图片展示

20.1　拍摄照片

首先，我们来升级CrimeCameraFragment的布局，为其添加一个进度指示条组件。相机拍摄照片的过程可能比较耗时，有时需要用户等一会儿，为了不让用户失去耐心，添加进度指示条非常有必要。

在fragment_crime_camera.xml布局文件中，参照图20-2，添加FrameLayout和ProgressBar组件。

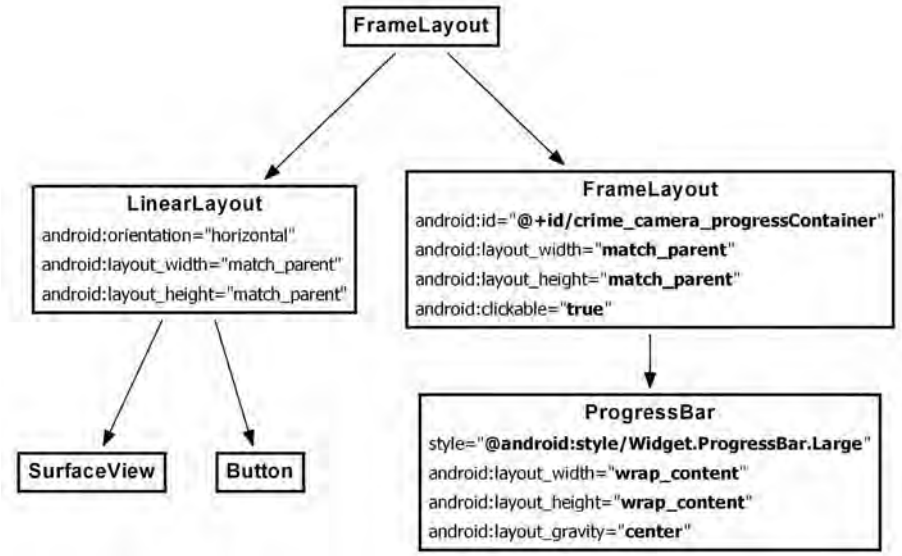

图20-2　添加FrameLayout和ProgressBar组件（fragment_crime_camera.xml）

代替默认的普通大小圆形进度条，@android:style/Widget.ProgressBar.Large样式将创建一个粗大的圆形旋转进度条，如图20-3所示。

图20-3　旋转的进度条

FrameLayout（包括它的ProgressBar子组件）的初始状态设置为不可见。只有在用户点击Take!按钮开始拍照时才可见。

注意，FrameLayout组件的宽、高属性都设置为了match_parent，而根元素FrameLayout

会按照各子组件定义的顺序叠放它们。因此，包含ProgressBar组件的FrameLayout会完全遮挡住同级的LinearLayout兄弟组件。

FrameLayout组件可见时，用户依然能够看到LinearLayout组件包含的子组件。只有ProgressBar组件确实会遮挡其他组件。然而，通过设置FrameLayout组件的宽、高属性值为match_parent以及设置android:clickable="true"，可以确保FrameLayout组件能够截获（仅截获但不响应）任何触摸事件。这样，可阻止用户与LinearLayout组件包含的子组件交互，尤其是可以阻止用户再次点击Take!拍照按钮。

返回到CrimeCameraFragment.java中，为FrameLayout组件添加成员变量，然后通过资源ID引用它并设置为不可见状态，如代码清单20-1所示。

代码清单20-1 配置使用FrameLayout视图（CrimeCameraFragment.java）

```
public class CrimeCameraFragment extends Fragment {
    ...
    private View mProgressContainer;

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

        mProgressContainer = v.findViewById(R.id.crime_camera_progressContainer);
        mProgressContainer.setVisibility(View.INVISIBLE);

        ...

        return v;
    }

...
}
```

20.1.1 实现相机回调方法

既然进度条的添加设置已完成，接下来实现从相机的实时预览中捕获一帧图像，然后将它保存为JPEG格式的文件。要拍摄一张照片，需调用以下见名知意的Camera方法：

```
public final void takePicture(Camera.ShutterCallback shutter,
    Camera.PictureCallback raw,
    Camera.PictureCallback jpeg)
```

ShutterCallback回调方法会在相机捕获图像时调用，但此时，图像数据还未处理完成。第一个PictureCallback回调方法是在原始图像数据可用时调用，通常来说，是在加工处理原始图像数据且没有存储之前。第二个PictureCallback回调方法是在JPEG版本的图像可用时调用。

我们可以实现以上接口并传入takePicture(...)方法。如果不打算实现任何接口，直接传入三个空值即可。

在CriminalIntent应用中，实现ShutterCallback回调方法以及JPEG版本的PictureCallback回调方法。图20-4展示了这些对象之间的交互关系。

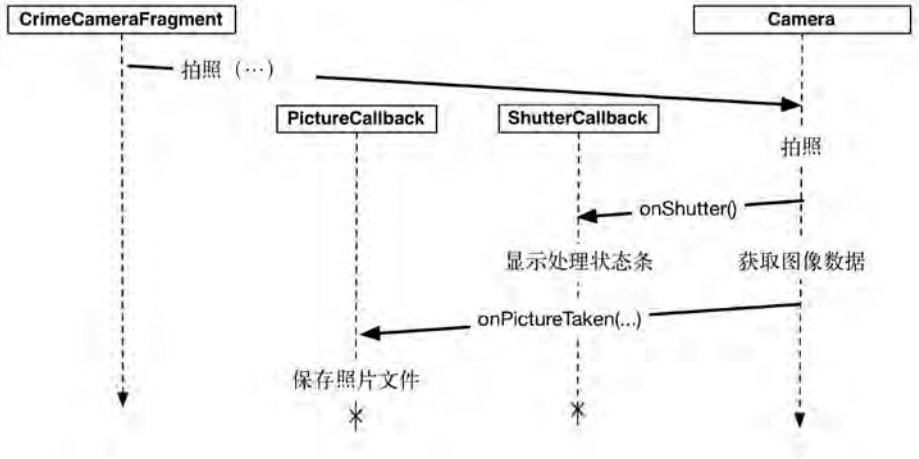

图20-4　在CrimeCameraFragment中拍照

下面是需要实现的两个接口，每个接口含有一个待实现的方法：

```
public static interface Camera.ShutterCallback {
    public abstract void onShutter();
}

public static interface Camera.PictureCallback {
    public abstract void onPictureTaken (byte[] data, Camera camera);
}
```

在CrimeCameraFragment.java中，实现Camera.ShutterCallback接口显示进度条视图，实现Camera.PictureCallback接口命名并保存已拍摄的JPEG图片文件，如代码清单20-2所示。

代码清单20-2　实现传入takePicture(...)方法的接口（CrimeCameraFragment.java）

```
...

private View mProgressContainer;

private Camera.ShutterCallback mShutterCallback = new Camera.ShutterCallback() {
    public void onShutter() {
        // Display the progress indicator
        mProgressContainer.setVisibility(View.VISIBLE);
    }
};

private Camera.PictureCallback mJpegCallback = new Camera.PictureCallback() {
    public void onPictureTaken(byte[] data, Camera camera) {
        // Create a filename
        String filename = UUID.randomUUID().toString() + ".jpg";
        // Save the jpeg data to disk
        FileOutputStream os = null;
        boolean success = true;

        try {
            os = getActivity().openFileOutput(filename, Context.MODE_PRIVATE);
```

```
                os.write(data);
            } catch (Exception e) {
                Log.e(TAG, "Error writing to file " + filename, e);
                success = false;
            } finally {
                try {
                    if (os != null)
                        os.close();
                } catch (Exception e) {
                    Log.e(TAG, "Error closing file " + filename, e);
                    success = false;
                }
            }

            if (success) {
                Log.i(TAG, "JPEG saved at " + filename);
            }
            getActivity().finish();
        }
    };

    ...
```

在onPictureTaken(...)方法中，创建了一个UUID字符串作为图片文件名。然后，使用Java I/O类打开一个输出流，将从Camera传入的JPEG数据写入文件。如果一切操作顺利，程序会输出一条文件保存成功的日志。

注意，代表进度指示条的mProgressContainer变量没有再设置回不可见状态。既然在onPictureTaken(...)方法中，fragment视图最终会随着activity的销毁而销毁。那也就没必要再关心mProgressContainer变量的处理了。

完成了回调方法的处理，接下来就是修改Take!按钮的监听器方法，实现对takePicture(...)方法的调用。对于没有实现的接收处理原始图像数据的回调方法，记得传入null值，如代码清单20-3所示。

代码清单20-3 实现takePicture(...)按钮单击事件方法 (CrimeCameraFragment.java)

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

        takePictureButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                getActivity().finish();
                if (mCamera != null) {
                    mCamera.takePicture(mShutterCallback, null, mJpegCallback);
                }
            }
        });

        ...

        return v;
    }
```

20.1.2　设置图片尺寸大小

相机需要知道创建多大尺寸的图片。设置图片尺寸与设置预览尺寸一样。可以调用以下Camera.Parameters方法获得可用的图片尺寸的列表：

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

在surfaceChanged(...)方法中，使用getBestSupportedSize(...)方法获得支持的适用于Surface的图片尺寸。最后将获得的尺寸设置为相机要创建的图片尺寸，如代码清单20-4所示。

代码清单20-4　调用getBestSupportedSize(...)方法设置图片尺寸（CrimeCameraFragment.java）

```
        ...

        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 = getBestSupportedSize(parameters.getSupportedPreviewSizes(), w, h);
            parameters.setPreviewSize(s.width, s.height);
            s = getBestSupportedSize(parameters.getSupportedPictureSizes(), w, h);
            parameters.setPictureSize(s.width, s.height);
            mCamera.setParameters(parameters);

            ...
        }
    });
```

运行CriminalIntent应用，然后点击Take!按钮。在LogCat中，创建一个以CrimeCameraFragment为标签的过滤器，查看图片文件的保存位置。

目前为止，CrimeCameraFragment类完全具备了拍照及保存文件的功能。相机API的相关开发工作全部完成了。本章接下来的部分将重点介绍CrimeFragment类的开发完善，从而将图片与应用的其他部分进行整合。

20.2　返回数据给 CrimeFragment

CrimeFragment类要能使用图片，需要将文件名从CrimeCameraFragment回传给它。图20-5展示了CrimeFragment与CrimeCameraFragment之间的交互过程。

首先，CrimeFragment以接收返回值的方式启动CrimeCameraActivity。图片拍摄完成后，CrimeCameraFragment会以图片文件名作为extra创建一个intent，并调用setResult(...)方法。然后，ActivityManager会调用onActivityResult(...)方法将intent转发给CrimePagerActivity。最后，CrimePagerActivity的FragmentManager会调用CrimeFragment.onActivityResult(...)方法，将intent转发给CrimeFragment。

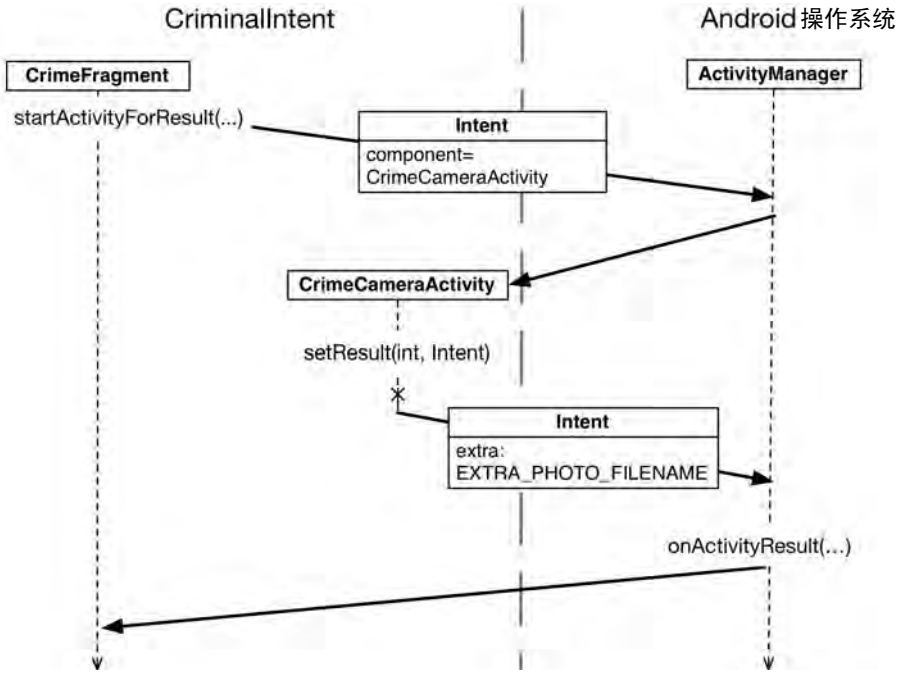

图20-5 使用CrimeCameraActivity设置回传信息

20.2.1 以接收返回值的方式启动CrimeCameraActivity

当前，CrimeFragment只是直接启动CrimeCameraActivity。在CrimeFragment.java中，新增一个请求码常量，然后修改拍照按钮的监听器方法，以需要接收返回值的方式启动CrimeCameraActivity，如代码清单20-5所示。

代码清单20-5 以接收返回值的方式启动CrimeCameraActivity（CrimeFragment.java）

```
public class CrimeFragment extends Fragment {
    ...
    private static final int REQUEST_DATE = 0;
    private static final int REQUEST_PHOTO = 1;

    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {

        ...

        mPhotoButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                // Launch the camera activity
                Intent i = new Intent(getActivity(), CrimeCameraActivity.class);
                startActivity(i);
```

```
                startActivityForResult(i, REQUEST_PHOTO);
            }
        });

        ...
    }
```

20.2.2　在CrimeCameraFragment中设置返回值

CrimeCameraFragment会将图片文件名放置在extra中并附加到intent上，然后传入CrimeCameraActivity.setResult(int, Intent)方法。在CrimeCameraFragment.java中，新增一个extra常量。然后，在onPictureTaken(...)方法中，判断照片处理状态，如果照片保存成功，就创建一个intent并设置结果代码为RESULT_OK，反之，则设置结果代码为RESULT_CANCELED，如代码清单20-6所示。

代码清单20-6　新增照片文件名extra（CrimeCameraFragment.java）

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

    public static final String EXTRA_PHOTO_FILENAME =
        "com.bignerdranch.android.criminalintent.photo_filename";

    ...

    private Camera.PictureCallback mJpegCallback = new Camera.PictureCallback() {
        public void onPictureTaken(byte[] data, Camera camera) {
            ...
            try {
                ...
            } catch (Exception e) {
                ...
            } finally {
                ...
            }
            Log.i(TAG, "JPEG saved at " + filename);
            // Set the photo filename on the result intent
            if (success) {
                Intent i = new Intent();
                i.putExtra(EXTRA_PHOTO_FILENAME, filename);
                getActivity().setResult(Activity.RESULT_OK, i);
            } else {
                getActivity().setResult(Activity.RESULT_CANCELED);
            }
            getActivity().finish();
        }
    };

    ...
}
```

20.2.3　在CrimeFragment中获取照片文件名

最后，CrimeFragment会使用照片文件名更新CriminalIntent应用的模型层和视图层。在

CrimeFragment.java中，覆盖onActivityResult(...)方法，检查结果并获取照片文件名。然后，为CrimeFragment类新增一个用于日志记录的TAG，如果照片文件名获取成功，就输出结果日志，如代码清单20-7所示。

代码清单20-7 获取照片文件名（CrimeFragment.java）

```
public class CrimeFragment extends Fragment {
    private static final String TAG = "CrimeFragment"
    public static final String EXTRA_CRIME_ID =
        "com.bignerdranch.android.criminalintent.crime_id";
    ...

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (resultCode != Activity.RESULT_OK) return;

        if (requestCode == REQUEST_DATE) {
            Date date = (Date)data
                .getSerializableExtra(DatePickerFragment.EXTRA_DATE);
            mCrime.setDate(date);
            updateDate();
        } else if (requestCode == REQUEST_PHOTO) {
            // Create a new Photo object and attach it to the crime
            String filename = data
                .getStringExtra(CrimeCameraFragment.EXTRA_PHOTO_FILENAME);
            if (filename != null) {
                Log.i(TAG, "filename: " + filename);
            }
        }
    }
```

运行CriminalIntent应用。在CrimeCameraActivity中拍摄一张照片。然后检查LogCat，确认CrimeFragment成功获取了照片文件名。

有了CrimeFragment获取的照片文件名，接下来还有一些事情要做。

更新模型层：首先需编写一个封装照片文件名的Photo类。还需给Crime类添加一个Photo类型的mPhoto属性。CrimeFragment将使用照片文件名创建一个Photo对象，然后使用它设置Crime的mPhoto属性。

更新CrimeFragment的视图：需要为CrimeFragment的布局增加一个ImageView组件，然后在ImageView视图上显示Crime的照片缩略图。

显示全尺寸版的图片：需要创建一个名为ImageFragment的DialogFragment子类，然后使用它显示指定路径的照片。

20.3 更新模型层

图20-6展示了CrimeFragment、Crime以及Photo类三者之间的关系。

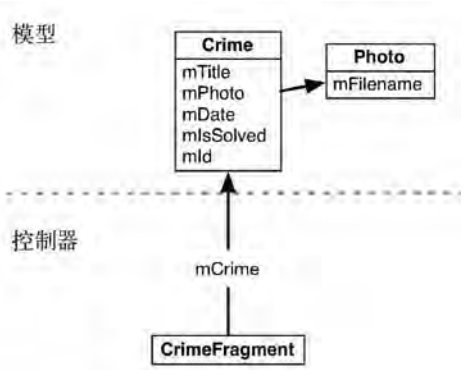

图20-6　模型层对象与CrimeFragment

20.3.1　新增Photo类

以默认的java.lang.Object为超类，在com.bignerdranch.android.criminalintent包中创建一个名为Photo的新类。

在Photo.java中，参照代码清单20-8添加需要的变量和方法。

代码清单20-8　Photo新建类（Photo.java）

```
...

public class Photo {
    private static final String JSON_FILENAME = "filename";

    private String mFilename;

    /** Create a Photo representing an existing file on disk */
    public Photo(String filename) {
        mFilename = filename;
    }

    public Photo(JSONObject json) throws JSONException {
        mFilename = json.getString(JSON_FILENAME);
    }

    public JSONObject toJSON() throws JSONException {
        JSONObject json = new JSONObject();
        json.put(JSON_FILENAME, mFilename);
        return json;
    }

    public String getFilename() {
        return mFilename;
    }
}
```

注意，Photo类有两个构造方法。第一个构造方法根据给定的文件名创建一个Photo对象。第二个构造方法是一个JSON序列化方法，在保存以及加载Photo类型的数据时，Crime会用到它。

20.3.2 为Crime添加photo属性

现在，我们来更新Crime类，包含一个Photo对象并将其序列化为JSON格式，如代码清单20-9所示。

代码清单20-9 Crime照片（Crime.java）

```
public class Crime {
    ...
    private static final String JSON_DATE = "date";
    private static final String JSON_PHOTO = "photo";

    ...
    private Date mDate = new Date();
    private Photo mPhoto;

    ...

    public Crime(JSONObject json) throws JSONException {
        ...
        mDate = new Date(json.getLong(JSON_DATE));
        if (json.has(JSON_PHOTO))
            mPhoto = new Photo(json.getJSONObject(JSON_PHOTO));
    }

    public JSONObject toJSON() throws JSONException {
        JSONObject json = new JSONObject();
        ...
        json.put(JSON_DATE, mDate.getTime());
        if (mPhoto != null)
            json.put(JSON_PHOTO, mPhoto.toJSON());
        return json;
    }

    ...

    public Photo getPhoto() {
        return mPhoto;
    }

    public void setPhoto(Photo p) {
        mPhoto = p;
    }

}
```

20.3.3 设置photo属性

在CrimeFragment.java中，修改onActivityResult(...)方法，在其中新建一个Photo对象并设置给当前的Crime，如代码清单20-10所示。

代码清单20-10 处理新照片（CrimeFragment.java）

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode != Activity.RESULT_OK) return;
```

```
        if (requestCode == REQUEST_DATE) {
                Date date = (Date)data
                    .getSerializableExtra(DatePickerFragment.EXTRA_DATE);
                mCrime.setDate(date);
                updateDate();
            } else if (requestCode == REQUEST_PHOTO) {
            // Create a new Photo object and attach it to the crime
            String filename = data
                .getStringExtra(CrimeCameraFragment.EXTRA_PHOTO_FILENAME);
            if (filename != null) {
                Log.i(TAG, "filename: " + filename);

                Photo p = new Photo(filename);
                mCrime.setPhoto(p);
                Log.i(TAG, "Crime: " + mCrime.getTitle() + " has a photo");
            }
        }
    }
```

运行CriminalIntent应用并拍摄一张照片。然后查看LogCat，确认Crime拥有这张新拍的照片。

可能有人会问，为什么要创建一个Photo类，而不是简单地添加一个文件名属性给Crime类。直接添加文件名属性虽然可行，但新建Photo类可以帮助处理更多任务，如显示照片名称或处理触摸事件。显然，要处理这些事情，我们需要一个单独的类。

20.4　更新 CrimeFragment 的视图

完成了模型层的更新，我们来着手更新CrimeFragment的视图层。特别要提到的是，CrimeFragment将会在ImageView上显示照片缩略图。

图20-7　添加了ImageView组件的CrimeFragment

20.4.1 添加ImageView组件

打开layout/fragment_crime.xml布局文件，对照图20-8添加ImageView组件。

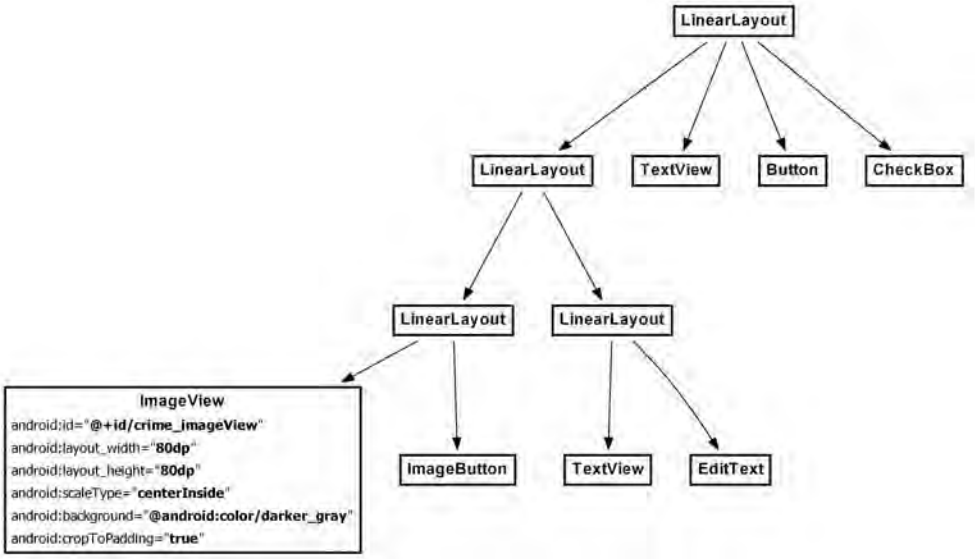

图20-8 添加了ImageView组件的CrimeFragment布局（layout/fragment_crime.xml）

我们还需要一个带有ImageView组件的水平模式布局，如图20-9所示。

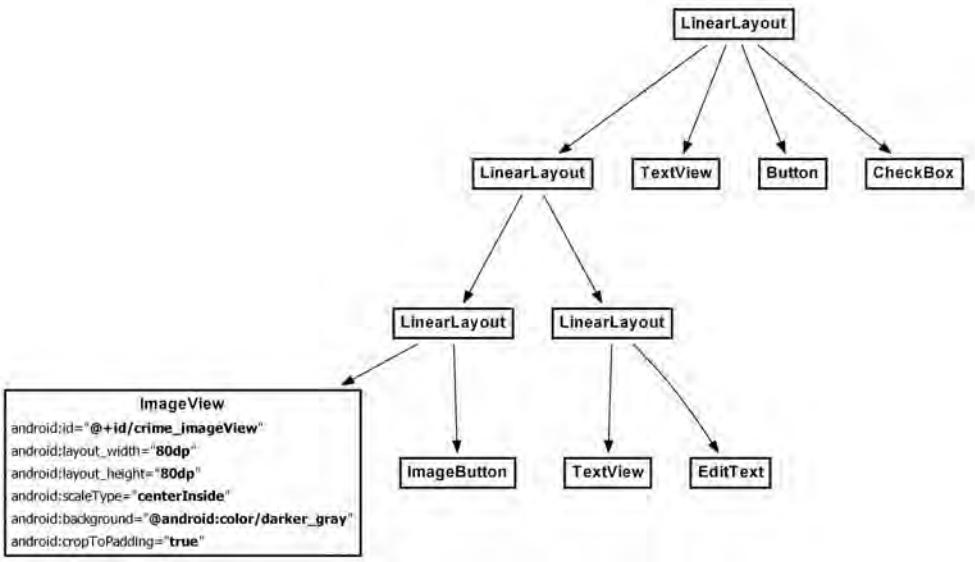

图20-9 带有ImageView组件的水平模式布局（layout-land/fragment_crime.xml）

在CrimeFragment.java中，创建一个成员变量，然后在onCreateView(...)方法中以资源ID引用ImageView视图，如代码清单20-11所示。

代码清单20-11　配置ImageButton（CrimeFragment.java）

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

    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {

        ...

        mPhotoButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                // Launch the camera activity
                Intent i = new Intent(getActivity(), CrimeCameraActivity.class);
                startActivityForResult(i, REQUEST_PHOTO);
            }
        });

        mPhotoView = (ImageView)v.findViewById(R.id.crime_imageView);

        ...
    }
```

预览修改后的布局，或者运行CriminalIntent应用，确保ImageView组件已正确添加。

20.4.2　图像处理

相机拍摄的照片尺寸通常都很大，需要预先处理，然后才能在ImageView视图上显示。手机制造商每年新推出的手机都带有越来越强大的相机。对用户来说，这是好事。但对于开发者来说，这很让人头痛。

本书写作时，主流Android手机都带有800万像素的照相机组件。大尺寸的图片很容易耗尽应用的内存。因此，加载图片前，需要编写代码缩小图片。图片使用完毕，也需要编写代码清理删除它。

1. 添加处理过的图片到imageview视图

在com.bignerdranch.android.criminalintent包中创建一个名为PictureUtils的新类。然后，在PictureUtils.java中，添加如代码清单20-12所示的方法，将图片缩放到设备默认的显示尺寸。

代码清单20-12　添加PictureUtils类（PictureUtils.java）

```
public class PictureUtils {
    /**
     * Get a BitmapDrawable from a local file that is scaled down
     * to fit the current Window size.
```

```
     */
    @SuppressWarnings("deprecation")
    public static BitmapDrawable getScaledDrawable(Activity a, String path) {
        Display display = a.getWindowManager().getDefaultDisplay();
        float destWidth = display.getWidth();
        float destHeight = display.getHeight();

        // Read in the dimensions of the image on disk
        BitmapFactory.Options options = new BitmapFactory.Options();
        options.inJustDecodeBounds = true;
        BitmapFactory.decodeFile(path, options);

        float srcWidth = options.outWidth;
        float srcHeight = options.outHeight;

        int inSampleSize = 1;
        if (srcHeight > destHeight || srcWidth > destWidth) {
            if (srcWidth > srcHeight) {
                inSampleSize = Math.round(srcHeight / destHeight);
            } else {
                inSampleSize = Math.round(srcWidth / destWidth);
            }
        }

        options = new BitmapFactory.Options();
        options.inSampleSize = inSampleSize;

        Bitmap bitmap = BitmapFactory.decodeFile(path, options);
        return new BitmapDrawable(a.getResources(), bitmap);
    }
}
```

注意，Display.getWidth()和Display.getHeight()方法已被弃用。本章末尾将介绍更多有关代码弃用的知识。

如果能将图片缩放至完美匹配ImageView视图的尺寸，那自然最好了。然而，我们通常无法及时获得用来显示图片的视图尺寸。例如，在onCreateView(...)方法中，就无法获得ImageView视图的尺寸。设备的默认屏幕大小是固定可知的，因此，稳妥起见，可以缩放图片至设备的默认显示屏大小。注意，用来显示图片的视图可能会小于默认的屏幕显示尺寸，但大于屏幕默认的显示尺寸则肯定不行。

接下来，在CrimeFragment类中，新增一个私有方法，将缩放后的图片设置给ImageView视图，如代码清单20-13所示。

代码清单20-13 添加showPhoto()方法（CrimeFragment.java）

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
    ...
}

private void showPhoto() {
    // (Re)set the image button's image based on our photo
    Photo p = mCrime.getPhoto();
    BitmapDrawable b = null;
```

```
        if (p != null) {
            String path = getActivity()
                .getFileStreamPath(p.getFilename()).getAbsolutePath();
            b = PictureUtils.getScaledDrawable(getActivity(), path);
        }
        mPhotoView.setImageDrawable(b);
    }
```

在CrimeFragment.java中，新增onStart()实现方法，只要CrimeFragment的视图一出现在屏幕上，就调用showPhoto()方法显示图片，如代码清单20-14所示。

代码清单20-14　加载图片（CrimeFragment.java）

```
...

    private void showPhoto() {
        // (Re)set the image button's image based on our photo
        Photo p = mCrime.getPhoto();
        BitmapDrawable b = null;
        if (p != null) {
            String path = getActivity()
                .getFileStreamPath(p.getFilename()).getAbsolutePath();
            b = PictureUtils.getScaledDrawable(getActivity(), path);
        }
        mPhotoButton.setImageDrawable(b);
    }

    @Override
    public void onStart() {
        super.onStart();
        showPhoto();
    }
```

在CrimeFragment.onActivityResult(...)方法中，同样调用showPhoto()方法，以确保用户从CrimeCameraActivity返回后，ImageView视图可以显示用户所拍照片，如代码清单20-15所示。

代码清单20-15　在onActivityResult(...)方法中调用showPhoto()方法（CrimeFragment.java）

```
    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (resultCode != Activity.RESULT_OK) return;

        if (requestCode == REQUEST_PHOTO) {
            // Create a new Photo object and attach it to the crime
            String filename = data
                .getStringExtra(CrimeCameraFragment.EXTRA_PHOTO_FILENAME);
            if (filename != null) {
                Photo p = new Photo(filename);
                mCrime.setPhoto(p);
                showPhoto();
                Log.i(TAG, "Crime: " + mCrime.getTitle() + "has a photo");
            }
        }
    }
```

2. 卸载图片

在PictureUtils类中添加清理方法，清理ImageView的BitmapDrawable，如代码清单20-16所示。

代码清单20-16 清理工作（PictureUtils.java）

```
public class PictureUtils {
    /**
     * ...
     */
    @SuppressWarnings("deprecation")
    public static BitmapDrawable getScaledDrawable(Activity a, String path) {
        ...
    }

    public static void cleanImageView(ImageView imageView) {
        if (!(imageView.getDrawable() instanceof BitmapDrawable))
            return;

        // Clean up the view's image for the sake of memory
        BitmapDrawable b = (BitmapDrawable)imageView.getDrawable();
        b.getBitmap().recycle();
        imageView.setImageDrawable(null);
    }
}
```

Bitmap.recycle()方法的调用需要一些解释。Android开发文档暗示不需要调用Bitmap.recycle()方法，但实际上需要。因此，下面给出技术说明。

Bitmap.recycle()方法释放了bitmap占用的原始存储空间。这也是bitmap对象最核心的部分。（取决于具体的Android系统版本，原始存储空间可大可小。Honeycomb以前，它存储了Java Bitmap的所有数据。）

如果不主动调用recycle()方法释放内存，占用的内存也会被清理。但是，它是在将来某个时点在finalizer中清理，而不是在bitmap自身的垃圾回收时清理。这意味着很可能在finalizer调用之前，应用已经耗尽了内存资源。

finalizer的执行有时不太靠谱，且这类bug很难跟踪或重现。因此，如果应用使用的图片文件很大，最好主动调用recycle()方法，以避免可能的内存耗尽问题。

在CrimeFragment类中，添加onStop()方法，并在其中调用cleanImageView(...)方法清理内存，如代码清单20-17所示。

代码清单20-17 卸载图片（CrimeFragment.java）

```
@Override
public void onStart() {
    super.onStart();
    showPhoto();
}

@Override
public void onStop() {
    super.onStop();
    PictureUtils.cleanImageView(mPhotoView);
}
```

在onStart()方法中加载图片，然后在onStop()方法中卸载图片是一种好习惯。这些方法标志着用户可以看到activity的时间点。如果改在onResume()方法和onPause()方法中加载和卸

载图片，用户体验可能会很糟糕。

暂停的activity也可能部分可见，比如说，非全屏的activity视图显示在暂停的activity视图之上时。如果使用了`onResume()`方法和`onPause()`方法，那么图像消失后，因为没有被全部遮住，它又显示在了屏幕上。所以说，最佳实践就是，activity的视图一出现时就加载图片，然后等到activity再也不可见的情况下，再对它们进行卸载。

运行CriminalIntent应用。拍摄一张照片并确认它显示在`Imageview`视图上。然后退出应用并重新启动它。确认进入同一`Crime`明细界面时，`Imageview`视图上的图片仍可正常显示。

按照`CrimeCameraActivity`的初始显示方向，最好是以水平模式进行拍照。然而，如果不小心使用了竖直模式，拍照按钮上的图片可能无法按正确的方向显示。请通过本章第一个挑战练习修正该问题。

20.5　在 DialogFragment 中显示大图片

本章，CriminalIntent应用开发的最后环节是让用户查看`Crime`的大尺寸照片，如图20-10所示。

图20-10　显示较大图片的`DialogFragment`

以DialogFragment为父类，在com.bignerdranch.android.criminalintent包中创建一个名为ImageFragment的新类。

ImageFragment类需要知道Crime照片的文件路径。在ImageFragment.java中，新增一个newInstance(String)方法，该方法接受照片文件路径并放置到argument bundle中，如代码清单20-18所示。

代码清单20-18 创建ImageFragment（ImageFragment.java）

```
public class ImageFragment extends DialogFragment {
    public static final String EXTRA_IMAGE_PATH =
        "com.bignerdranch.android.criminalintent.image_path";

    public static ImageFragment newInstance(String imagePath) {
        Bundle args = new Bundle();
        args.putSerializable(EXTRA_IMAGE_PATH, imagePath);

        ImageFragment fragment = new ImageFragment();
        fragment.setArguments(args);
        fragment.setStyle(DialogFragment.STYLE_NO_TITLE, 0);

        return fragment;
    }
}
```

通过设置fragment的样式为DialogFragment.STYLE_NO_TITLE，获得一个如图20-10所示的简洁用户界面。

ImageFragment不需要显示AlertDialog视图自带的标题和按钮。如果fragment不需要显示标题和按钮，要实现显示大图片的对话框，采用覆盖onCreateView(...)方法并使用简单视图的方式，要比覆盖onCreateDialog(...)方法并使用Dialog更简单、快捷且灵活。

在ImageFragment.java中，覆盖onCreateView(...)方法创建ImageView并从argument获取文件路径。然后获取缩小版的图片并设置给ImageView。最后，只要图片不再需要，就主动覆盖onDestroyView()方法以释放内存，如代码清单20-19所示。

代码清单20-19 创建ImageFragment（ImageFragment.java）

```
public class ImageFragment extends DialogFragment {
    public static final String EXTRA_IMAGE_PATH =
        "com.bignerdranch.android.criminalintent.image_path";

    public static ImageFragment newInstance(String imagePath) {
        ...
    }

    private ImageView mImageView;

    @Override
    public View onCreateView(LayoutInflater inflater,
            ViewGroup parent, Bundle savedInstanceState) {
        mImageView = new ImageView(getActivity());
        String path = (String)getArguments().getSerializable(EXTRA_IMAGE_PATH);
```

```
            BitmapDrawable image = PictureUtils.getScaledDrawable(getActivity(), path);

            mImageView.setImageDrawable(image);

            return mImageView;
        }

        @Override
        public void onDestroyView() {
            super.onDestroyView();
            PictureUtils.cleanImageView(mImageView);
        }
    }
```

最后，我们需要从CrimeFragment弹出显示图片的对话框。在CrimeFragment.java中，添加一个监听器方法给mPhotoView。在实现方法里，创建一个ImageFragment实例，然后通过调用ImageFragment的show(...)方法，将它添加给CrimePagerActivity的FragmentManager。另外，还需要一个字符串常量，用来唯一定位FragmentManager中的ImageFragment，如代码清单20-20所示。

代码清单20-20　显示ImageFragment界面（CrimeFragment.java）

```
public class CrimeFragment extends Fragment {

    ...
    private static final String DIALOG_IMAGE = "image";

    ...

    @Override
    @TargetApi(11)
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        ...

        mPhotoView = (ImageView)v.findViewById(R.id.crime_imageView);
        mPhotoView.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                Photo p = mCrime.getPhoto();
                if (p == null)
                    return;

                FragmentManager fm = getActivity()
                    .getSupportFragmentManager();
                String path = getActivity()
                    .getFileStreamPath(p.getFilename()).getAbsolutePath();
                ImageFragment.newInstance(path)
                    .show(fm, DIALOG_IMAGE);
            }
        });

        ...
    }
```

运行CriminalIntent应用。拍摄一张照片，确认可以清楚地看到那些令人震惊的案发现场照。

20.6 挑战练习：Crime 照片的显示方向

有时候，用户会以竖直模式拍摄照片。查阅API开发文档，寻找探测设备方向的方法。在Photo类中保存照片拍摄时的设备方向，然后在CrimeFragment类和ImageFragment类中，根据设备方向正确旋转它。

20.7 挑战练习：删除照片

当前，虽然可以替换Crime的照片，但旧照片依然占据着存储空间。在CrimeFragment类的onActivityResult(int, int, Intent)方法中添加代码，检查并删除目标Crime已存在的照片文件。

如果对拍摄的照片不满意，用户只能新拍一张照片去替换老照片。作为另一项挑战练习，实现让用户直接删除现有照片。在CrimeFragment类中，实现长按缩略图，触发一个上下文菜单或进入上下文操作模式，然后选择Delete Photo菜单项，从磁盘、模型层以及ImageView中删除照片。

20.8 深入学习：Android 代码的废弃处理

第19章，在设置相机预览尺寸时，我们使用了一个弃用方法和一个弃用常量。本章同样也使用了弃用方法。好奇多思的读者可能会问，既然是弃用的方法，为什么还要使用呢?

要回答这个问题，首先要搞明白部分API的弃用到底意味着什么。如果某些东西被弃用，这就意味着再也不需要它们了。有时，我们不再需要某些方法提供的功能，这就会导致弃用发生。第19章添加的SurfaceHolder.setType(int)方法和SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS常量就属于这种情况。在旧系统版本的设备上，根据具体的使用方式，SurfaceHolder需要做相应的配置。而对于现在的新版本系统，这种状况已不复存在，自然setType(...)方法就再也无用武之地了。

此外，基于某种原因，由于有了较新版本的替代方法，原来的老方法自然也就弃用了。例如，BitmapDrawable类就有一个因bug较多而弃用的BitmapDrawable(Bitmap)构造方法。再者，从设计的角度看，某些旧方法的弃用主要是因为需要引入更简约实用的新方法。比如，View.setBackgroundDrawable(Drawable)方法就是这样一个例子。同样的情况还有本章使用过的Display.getWidth()和Display.getHeight()方法。这两个方法现在已经被getSize(Point)方法取代，这样一来，原来顺序调用getWidth()和getHeight()方法时出现的一些bug得到了彻底解决。

不同系统平台对代码弃用的处理也各不相同。其中，比较有代表性的两种极端处理方式分别来自于我们熟悉的系统平台——微软与苹果。

在微软平台上，API虽然弃用了，但它们不会被删除。这是因为，对微软来说，他们认为运行在各种系统版本上的程序越多越好。任何引入公共API，都会得到永远的支持。甚至，为了保证向后兼容性，那些有较多bug、没有写入文档的特性也都一直保留。显然，这种处理方式有时

会给人带来不好的感觉。

而在苹果平台上，弃用的API通常很快就会从操作系统中移除。苹果苛求于干净完美的系统，为达目的，他们不在意删除了多少弃用的API。因而，苹果系统一直给人简洁清爽的感觉。唯一的问题就是，为防止老程序突然无法运行，用户需要不断地更新升级到最新系统。

在苹果平台上，如果希望同时支持新旧系统，可以按以下方式编写代码：

```
float destWidth;
float destHeight;

if (Build.VERSION.SDK_INT > Build.VERSION_CODES.HONEYCOMB_MR2) {
    Point size;
    display.getSize(size);
    destWidth = size.x;
    destHeight = size.y;
} else {
    destWidth = display.getWidth();
    destHeight = display.getHeight();
}
```

这是因为在苹果平台上，`getWidth()`和`getHeight()`方法随时可能被删除，所以，必须小心避免在新系统中使用它们。

虽然Android系统的API弃用处理方式与微软不完全相同，但应该说还是比较接近的。每一版本的Android SDK都最大化地与以前版本的SDK兼容。这意味着弃用的API方法几乎都不会删除。因此，不用太担心使用了旧方法。本书将根据实际需要使用弃用方法，并且通过注解的方式禁止Android Lint报出兼容性警告信息。为编写出更加简洁、优雅的代码，请尽量不要使用弃用的API。