

第 10 章

后台默默的劳动者——探究服务

记得在我上大学的时候，iPhone 是属于少数人才拥有的稀有物品，Android 甚至还没面世，那个时候全球的手机市场是由诺基亚统治着的。当时我觉得诺基亚的 Symbian 操作系统做得特别出色，因为比起一般的手机，它可以支持后台功能。那个时候能够一边打着电话、听着音乐，一边在后台挂着 QQ 是件非常酷的事情。所以我也曾经单纯地认为，支持后台的手机就是智能手机。

而如今，Symbian 早已风光不再，Android 和 iOS 占据了大部分的智能市场份额，Windows Phone 也占据了一小部分，目前已是三分天下的局面。在这三大智能手机操作系统中，iOS 和 Windows Phone 一开始都是不支持后台的，后来逐渐意识到这个功能的重要性，才加入了后台功能。而 Android 则是沿用了 Symbian 的老习惯，从一开始就支持后台功能，这使得应用程序即使在关闭的情况下仍然可以在后台继续运行。不管怎么说，后台功能属于四大组件之一，其重要程度不言而喻，那么我们自然要好好学习一下它的用法了。

10.1 服务是什么

服务（Service）是 Android 中实现程序后台运行的解决方案，它非常适合去执行那些不需要和用户交互而且还要求长期运行的任务。服务的运行不依赖于任何用户界面，即使程序被切换到后台，或者用户打开了另外一个应用程序，服务仍然能够保持正常运行。

不过需要注意的是，服务并不是运行在一个独立的进程当中的，而是依赖于创建服务时所在的应用程序进程。当某个应用程序进程被杀掉时，所有依赖于该进程的服务也会停止运行。

另外，也不要被服务的后台概念所迷惑，实际上服务并不会自动开启线程，所有的代码都是默认运行在主线程当中的。也就是说，我们需要在服务的内部手动创建子线程，并在这里执行具体的任务，否则就有可能出现主线程被阻塞住的情况。那么本章的第一堂课，我们就先来学习一下关于 Android 多线程编程的知识。

10.2 Android 多线程编程

熟悉 Java 的你，对多线程编程一定不会陌生吧。当我们需要执行一些耗时操作，比如说发起一条网络请求时，考虑到网速等其他原因，服务器未必会立刻响应我们的请求，如果不将这类操作放在子线程里去运行，就会导致主线程被阻塞住，从而影响用户对软件的正常使用。那么就让我们从线程的基本用法开始学习吧。

10.2.1 线程的基本用法

Android 多线程编程其实并不比 Java 多线程编程特殊，基本都是使用相同的语法。比如说，定义一个线程只需要新建一个类继承自 `Thread`，然后重写父类的 `run()` 方法，并在里面编写耗时逻辑即可，如下所示：

```
class MyThread extends Thread {

    @Override
    public void run() {
        // 处理具体的逻辑
    }
}
```

那么该如何启动这个线程呢？其实也很简单，只需要 `new` 出 `MyThread` 的实例，然后调用它的 `start()` 方法，这样 `run()` 方法中的代码就会在子线程当中运行了，如下所示：

```
new MyThread().start();
```

当然，使用继承的方式耦合性有点高，更多的时候我们都会选择使用实现 `Runnable` 接口的方式来定义一个线程，如下所示：

```
class MyThread implements Runnable {

    @Override
    public void run() {
        // 处理具体的逻辑
    }
}
```

如果使用了这种写法，启动线程的方法也需要进行相应的改变，如下所示：

```
MyThread myThread = new MyThread();
new Thread(myThread).start();
```

可以看到，`Thread` 的构造函数接收一个 `Runnable` 参数，而我们 `new` 出的 `MyThread` 正是一个实现了 `Runnable` 接口的对象，所以可以直接将它传入到 `Thread` 的构造函数里。接着调用 `Thread` 的 `start()` 方法，`run()` 方法中的代码就会在子线程当中运行了。

当然，如果你不想专门再定义一个类去实现 `Runnable` 接口，也可以使用匿名类的方式，这种写法更为常见，如下所示：

```
new Thread(new Runnable() {
    @Override
    public void run() {
        // 处理具体的逻辑
    }
}).start();
```

以上几种线程的使用方式相信你都不会感到陌生，因为在 Java 中创建和启动线程也是使用同样的方式。了解了线程的基本用法后，下面我们来看一下 Android 多线程编程与 Java 多线程编程不同的地方。

10.2.2 在子线程中更新 UI

和许多其他的 GUI 库一样，Android 的 UI 也是线程不安全的。也就是说，如果想要更新应用程里的 UI 元素，则必须在主线程中进行，否则就会出现异常。

眼见为实，让我们通过一个具体的例子来验证一下吧。新建一个 `AndroidThreadTest` 项目，然后修改 `activity_main.xml` 中的代码，如下所示：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/change_text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Change Text" />

    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:text="Hello world"
        android:textSize="20sp" />

</RelativeLayout>
```

布局文件中定义了两个控件，`TextView` 用于在屏幕的正中央显示一个 `Hello world` 字符串，`Button` 用于改变 `TextView` 中显示的内容，我们希望在点击 `Button` 后可以把 `TextView` 中显示的字符串改成 `Nice to meet you`。

接下来修改 `MainActivity` 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {  
  
    private TextView text;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        text = (TextView) findViewById(R.id.text);  
        Button changeText = (Button) findViewById(R.id.change_text);  
        changeText.setOnClickListener(this);  
    }  
  
    @Override  
    public void onClick(View v) {  
        switch (v.getId()) {  
            case R.id.change_text:  
                new Thread(new Runnable() {  
                    @Override  
                    public void run() {  
                        text.setText("Nice to meet you");  
                    }  
                }).start();  
                break;  
            default:  
                break;  
        }  
    }  
}
```

可以看到，我们在 Change Text 按钮的点击事件里面开启了一个子线程，然后在子线程中调用 TextView 的 `setText()` 方法将显示的字符串改成 Nice to meet you。代码的逻辑非常简单，只不过我们是在子线程中更新 UI 的。现在运行一下程序，并点击 Change Text 按钮，你会发现程序果然崩溃了，如图 10.1 所示。



图 10.1 在子线程中更新 UI 导致崩溃

然后观察 logcat 中的错误日志，可以看出是由于在子线程中更新 UI 所导致的，如图 10.2 所示。

```
android.view.ViewRootImpl$CalledFromWrongThreadException: Only the original
thread that created a view hierarchy can touch its views.
```

图 10.2 崩溃的详细信息

由此证实了 Android 确实是不允许在子线程中进行 UI 操作的。但是有些时候，我们必须在子线程里去执行一些耗时任务，然后根据任务的执行结果来更新相应的 UI 控件，这该如何是好呢？

对于这种情况，Android 提供了一套异步消息处理机制，完美地解决了在子线程中进行 UI 操作的问题。本小节中我们先来学习一下异步消息处理的使用方法，下一小节中再去分析它的原理。

修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    public static final int UPDATE_TEXT = 1;

    private TextView text;

    private Handler handler = new Handler() {

        public void handleMessage(Message msg) {
            switch (msg.what) {
                case UPDATE_TEXT:
                    // 在这里可以进行 UI 操作
            }
        }
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        text = findViewById(R.id.text);
        text.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        if (v == text) {
            handler.sendMessage(handler.obtainMessage(UPDATE_TEXT));
        }
    }
}
```

```

        text.setText("Nice to meet you");
        break;
    default:
        break;
    }
}

};

...

@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.change_text:
            new Thread(new Runnable() {
                @Override
                public void run() {
                    Message message = new Message();
                    message.what = UPDATE_TEXT;
                    handler.sendMessage(message); // 将 Message 对象发送出去
                }
            }).start();
            break;
        default:
            break;
    }
}
}

```

这里我们先是定义了一个整型常量 `UPDATE_TEXT`，用于表示更新 `TextView` 这个动作。然后新增一个 `Handler` 对象，并重写父类的 `handleMessage()` 方法，在这里对具体的 `Message` 进行处理。如果发现 `Message` 的 `what` 字段的值等于 `UPDATE_TEXT`，就将 `TextView` 显示的内容改成 `Nice to meet you`。

下面再来看一下 `Change Text` 按钮的点击事件中的代码。可以看到，这次我们并没有在子线程里直接进行 UI 操作，而是创建了一个 `Message` (`android.os.Message`) 对象，并将它的 `what` 字段的值指定为 `UPDATE_TEXT`，然后调用 `Handler` 的 `sendMessage()` 方法将这条 `Message` 发送出去。很快，`Handler` 就会收到这条 `Message`，并在 `handleMessage()` 方法中对它进行处理。注意此时 `handleMessage()` 方法中的代码就是在主线程当中运行的了，所以我们可以放心地在这里进行 UI 操作。接下来对 `Message` 携带的 `what` 字段的值进行判断，如果等于 `UPDATE_TEXT`，就将 `TextView` 显示的内容改成 `Nice to meet you`。

现在重新运行程序，可以看到屏幕的正中央显示着 `Hello world`。然后点击一下 `Change Text` 按钮，显示的内容就被替换成 `Nice to meet you`，如图 10.3 所示。

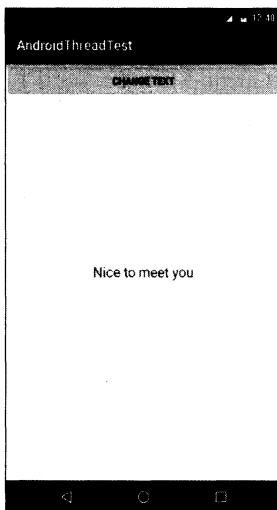


图 10.3 成功替换显示的文字

这样你就已经掌握了 Android 异步消息处理的基本用法，使用这种机制就可以出色地解决掉在子线程中更新 UI 的问题。不过恐怕你对它的工作原理还不是很清楚，下面我们就来分析一下 Android 异步消息处理机制到底是如何工作的。

10.2.3 解析异步消息处理机制

Android 中的异步消息处理主要由 4 个部分组成：Message、Handler、MessageQueue 和 Looper。其中 Message 和 Handler 在上一小节中我们已经接触过了，而 MessageQueue 和 Looper 对于你来说还是全新的概念，下面我就对这 4 个部分进行一下简要的介绍。

1. Message

Message 是在线程之间传递的消息，它可以在内部携带少量的信息，用于在不同线程之间交换数据。上一小节中我们使用到了 Message 的 what 字段，除此之外还可以使用 arg1 和 arg2 字段来携带一些整型数据，使用 obj 字段携带一个 Object 对象。

2. Handler

Handler 顾名思义也就是处理者的意思，它主要是用于发送和处理消息的。发送消息一般是使用 Handler 的 sendMessage() 方法，而发出的消息经过一系列地辗转处理后，最终会传递到 Handler 的 handleMessage() 方法中。

3. MessageQueue

MessageQueue 是消息队列的意思，它主要用于存放所有通过 Handler 发送的消息。这部分消息会一直存在于消息队列中，等待被处理。每个线程中只会有一个 MessageQueue 对象。

4. Looper

Looper 是每个线程中的 MessageQueue 的管家，调用 Looper 的 `loop()` 方法后，就会进入到一个无限循环当中，然后每当发现 MessageQueue 中存在一条消息，就会将它取出，并传递到 Handler 的 `handleMessage()` 方法中。每个线程中也只会有一个 Looper 对象。

了解了 Message、Handler、MessageQueue 以及 Looper 的基本概念后，我们再来把异步消息处理的整个流程梳理一遍。首先需要在主线程当中创建一个 Handler 对象，并重写 `handleMessage()` 方法。然后当子线程中需要进行 UI 操作时，就创建一个 Message 对象，并通过 Handler 将这条消息发送出去。之后这条消息会被添加到 MessageQueue 的队列中等待被处理，而 Looper 则会一直尝试从 MessageQueue 中取出待处理消息，最后分发回 Handler 的 `handleMessage()` 方法中。由于 Handler 是在主线程中创建的，所以此时 `handleMessage()` 方法中的代码也会在主线程中运行，于是我们在这里就可以安心地进行 UI 操作了。整个异步消息处理机制的流程示意图如图 10.4 所示。

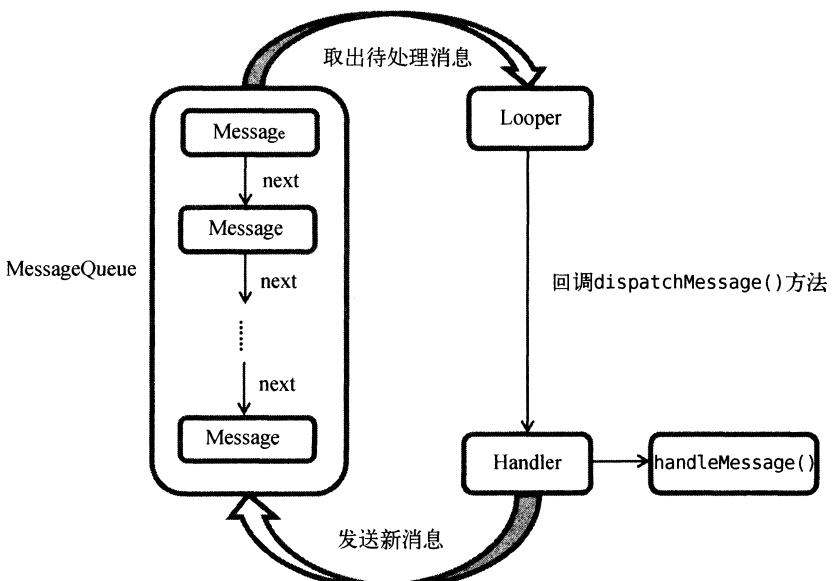


图 10.4 异步消息处理机制流程示意图

一条 Message 经过这样一个流程的辗转调用后，也就从子线程进入到了主线程，从不能更新 UI 变成了可以更新 UI，整个异步消息处理的核心思想也就是如此。

而我们在 9.2.1 小节中使用到的 `runOnUiThread()` 方法其实就是一个异步消息处理机制的接口封装，它虽然表面上看起来用法更为简单，但其实背后的实现原理和图 10.4 中的描述是一模一样的。

10.2.4 使用 AsyncTask

不过为了更加方便我们在子线程中对 UI 进行操作，Android 还提供了另外一些好用的工具，比如 `AsyncTask`。借助 `AsyncTask`，即使你对异步消息处理机制完全不了解，也可以十分简单地从子线程切换到主线程。当然，`AsyncTask` 背后的实现原理也是基于异步消息处理机制的，只是 Android 帮我们做了很好的封装而已。

首先来看一下 `AsyncTask` 的基本用法，由于 `AsyncTask` 是一个抽象类，所以如果我们想使用它，就必须要创建一个子类去继承它。在继承时我们可以为 `AsyncTask` 类指定 3 个泛型参数，这 3 个参数的用途如下。

- `Params`。在执行 `AsyncTask` 时需要传入的参数，可用于在后台任务中使用。
- `Progress`。后台任务执行时，如果需要在界面上显示当前的进度，则使用这里指定的泛型作为进度单位。
- `Result`。当任务执行完毕后，如果需要对结果进行返回，则使用这里指定的泛型作为返回值类型。

因此，一个最简单的自定义 `AsyncTask` 就可以写成如下方式：

```
class DownloadTask extends AsyncTask<Void, Integer, Boolean> {  
    ...  
}
```

这里我们把 `AsyncTask` 的第一个泛型参数指定为 `Void`，表示在执行 `AsyncTask` 的时候不需要传入参数给后台任务。第二个泛型参数指定为 `Integer`，表示使用整型数据来作为进度显示单位。第三个泛型参数指定为 `Boolean`，则表示使用布尔型数据来反馈执行结果。

当然，目前我们自定义的 `DownloadTask` 还是一个空任务，并不能进行任何实际的操作，我们还需要去重写 `AsyncTask` 中的几个方法才能完成对任务的定制。经常需要去重写的方法有以下 4 个。

1. `onPreExecute()`

这个方法会在后台任务开始执行之前调用，用于进行一些界面上的初始化操作，比如显示一个进度条对话框等。

2. `doInBackground(Params...)`

这个方法中的所有代码都会在子线程中运行，我们应该在这里去处理所有的耗时任务。任务一旦完成就可以通过 `return` 语句来将任务的执行结果返回，如果 `AsyncTask` 的第三个泛型参数指定的是 `Void`，就可以不返回任务执行结果。注意，在这个方法中是不可以进行 UI 操作的，如果需要更新 UI 元素，比如说反馈当前任务的执行进度，可以调用 `publishProgress(Progress...)` 方法来完成。

3. onProgressUpdate(Progress...)

当在后台任务中调用了 `publishProgress(Progress...)` 方法后，`onProgressUpdate(Progress...)` 方法就会很快被调用，该方法中携带的参数就是在后台任务中传递过来的。在这个方法中可以对 UI 进行操作，利用参数中的数值就可以对界面元素进行相应的更新。

4. onPostExecute(Result)

当后台任务执行完毕并通过 `return` 语句进行返回时，这个方法就很快会被调用。返回的数据会作为参数传递到此方法中，可以利用返回的数据来进行一些 UI 操作，比如说提醒任务执行的结果，以及关闭掉进度条对话框等。

因此，一个比较完整的自定义 `AsyncTask` 就可以写成如下方式：

```
class DownloadTask extends AsyncTask<Void, Integer, Boolean> {

    @Override
    protected void onPreExecute() {
        progressDialog.show(); // 显示进度对话框
    }

    @Override
    protected Boolean doInBackground(Void... params) {
        try {
            while (true) {
                int downloadPercent = doDownload(); // 这是一个虚构的方法
                publishProgress(downloadPercent);
                if (downloadPercent >= 100) {
                    break;
                }
            }
        } catch (Exception e) {
            return false;
        }
        return true;
    }

    @Override
    protected void onProgressUpdate(Integer... values) {
        // 在这里更新下载进度
        progressDialog.setMessage("Downloaded " + values[0] + "%");
    }

    @Override
    protected void onPostExecute(Boolean result) {
        progressDialog.dismiss(); // 关闭进度对话框
        // 在这里提示下载结果
        if (result) {
            Toast.makeText(context, "Download succeeded", Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(context, "Download failed", Toast.LENGTH_SHORT).show();
        }
    }
}
```

```
    }  
}
```

在这个 DownloadTask 中，我们在 `doInBackground()` 方法里去执行具体的下载任务。这个方法里的代码都是在子线程中运行的，因而不会影响到主线程的运行。注意这里虚构了一个 `doDownload()` 方法，这个方法用于计算当前的下载进度并返回，我们假设这个方法已经存在了。在得到了当前的下载进度后，下面就该考虑如何把它显示到界面上了，由于 `doInBackground()` 方法是在子线程中运行的，在这里肯定不能进行 UI 操作，所以我们可以调用 `publishProgress()` 方法并将当前的下载进度传进来，这样 `onProgressUpdate()` 方法就会很快被调用，在这里就可以进行 UI 操作了。

当下载完成后，`doInBackground()` 方法会返回一个布尔型变量，这样 `onPostExecute()` 方法就会很快被调用，这个方法也是在主线程中运行的。然后在这里我们会根据下载的结果来弹出相应的 Toast 提示，从而完成整个 DownloadTask 任务。

简单来说，使用 `AsyncTask` 的诀窍就是，在 `doInBackground()` 方法中执行具体的耗时任务，在 `onProgressUpdate()` 方法中进行 UI 操作，在 `onPostExecute()` 方法中执行一些任务的收尾工作。

如果想要启动这个任务，只需编写以下代码即可：

```
new DownloadTask().execute();
```

以上就是 `AsyncTask` 的基本用法，怎么样，是不是感觉简单方便了许多？我们并不需要去考虑什么异步消息处理机制，也不需要专门使用一个 `Handler` 来发送和接收消息，只需要调用一下 `publishProgress()` 方法，就可以轻松地从子线程切换到 UI 线程了。

在本章的最佳实践环节，我们会对下载这个功能进行完整的实现。

10.3 服务的基本用法

了解了 Android 多线程编程的技术之后，下面就让我们进入到本章的正题，开始对服务的相关内容进行学习。作为 Android 四大组件之一，服务也少不了有很多非常重要的知识点，那我们自然要从最基本的用法开始学习了。

10.3.1 定义一个服务

首先看一下如何在项目中定义一个服务。新建一个 ServiceTest 项目，然后右击 `com.example.servicetest` → `New` → `Service` → `Service`，会弹出如图 10.5 所示的窗口。

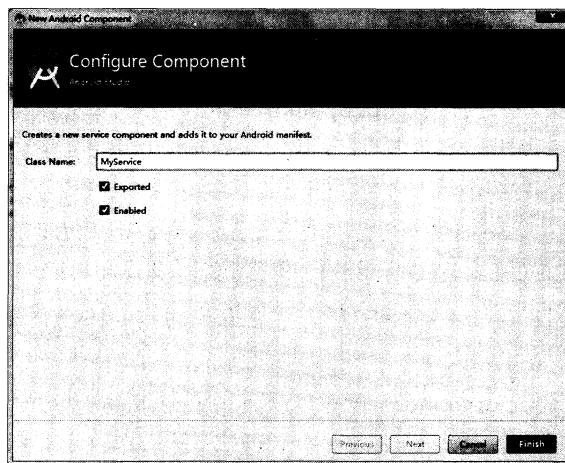


图 10.5 创建服务的窗口

可以看到，这里我们将服务命名为 MyService，**Exported** 属性表示是否允许除了当前程序之外的其他程序访问这个服务，**Enabled** 属性表示是否启用这个服务。将两个属性都勾中，点击 Finish 完成创建。

现在观察 MyService 中的代码，如下所示：

```
public class MyService extends Service {

    public MyService() {
    }

    @Override
    public IBinder onBind(Intent intent) {
        throw new UnsupportedOperationException("Not yet implemented");
    }
}
```

可以看到，MyService 是继承自 Service 类的，说明这是一个服务。目前 MyService 中可以算是空空如也，但有一个 onBind()方法特别醒目。这个方法是 Service 中唯一的一个抽象方法，所以必须要在子类里实现。我们会在后面的小节中使用到 onBind()方法，目前可以暂时将它忽略掉。

既然是定义一个服务，自然应该在服务中去处理一些事情了，那处理事情的逻辑应该写在哪里呢？这时就可以重写 Service 中的另外一些方法了，如下所示：

```
public class MyService extends Service {

    ...
    @Override
```

```

public void onCreate() {
    super.onCreate();
}

@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    return super.onStartCommand(intent, flags, startId);
}

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

}

```

可以看到，这里我们又重写了 `onCreate()`、`onStartCommand()` 和 `onDestroy()` 这 3 个方法，它们是每个服务中最常用到的 3 个方法了。其中 `onCreate()` 方法会在服务创建的时候调用，`onStartCommand()` 方法会在每次服务启动的时候调用，`onDestroy()` 方法会在服务销毁的时候调用。

通常情况下，如果我们希望服务一旦启动就立刻去执行某个动作，就可以将逻辑写在 `onStartCommand()` 方法里。而当服务销毁时，我们又应该在 `onDestroy()` 方法中去回收那些不再使用的资源。

另外需要注意，每一个服务都需要在 `AndroidManifest.xml` 文件中进行注册才能生效，不知道你有没有发现，这是 Android 四大组件共有的特点。不过相信你已经猜到了，智能的 Android Studio 早已自动帮我们将这一步完成了。打开 `AndroidManifest.xml` 文件瞧一瞧，代码如下所示：

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.servicetest">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        ...
        <service
            android:name=".MyService"
            android:enabled="true"
            android:exported="true">
        </service>
    </application>

</manifest>

```

这样的话，就已经将一个服务完全定义好了。

10.3.2 启动和停止服务

定义好了服务之后，接下来就应该考虑如何去启动以及停止这个服务。启动和停止的方法当然你也不会陌生，主要是借助 Intent 来实现的，下面就让我们在 ServiceTest 项目中尝试去启动以及停止 MyService 这个服务。

首先修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/start_service"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Start Service" />

    <Button
        android:id="@+id/stop_service"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Stop Service" />

</LinearLayout>
```

这里我们在布局文件中加入了两个按钮，分别是用于启动服务和停止服务的。

然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button startService = (Button) findViewById(R.id.start_service);
        Button stopService = (Button) findViewById(R.id.stop_service);
        startService.setOnClickListener(this);
        stopService.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.start_service:
                Intent startIntent = new Intent(this, MyService.class);
                startService(startIntent); // 启动服务
                break;
            case R.id.stop_service:
                Intent stopIntent = new Intent(this, MyService.class);
                stopService(stopIntent); // 停止服务
                break;
        }
    }
}
```

```

        stopService(stopIntent); // 停止服务
        break;
    default:
        break;
    }
}

}

```

可以看到，这里在 `onCreate()` 方法中分别获取到了 Start Service 按钮和 Stop Service 按钮的实例，并给它们注册了点击事件。然后在 Start Service 按钮的点击事件里，我们构建出了一个 `Intent` 对象，并调用 `startService()` 方法来启动 `MyService` 这个服务。在 Stop Service 按钮的点击事件里，我们同样构建出了一个 `Intent` 对象，并调用 `stopService()` 方法来停止 `MyService` 这个服务。`startService()` 和 `stopService()` 方法都是定义在 `Context` 类中的，所以我们在活动里可以直接调用这两个方法。注意，这里完全是由活动来决定服务何时停止的，如果没有点击 Stop Service 按钮，服务就会一直处于运行状态。那服务有没有什么办法让自己停下来呢？当然可以，只需要在 `MyService` 的任何一个位置调用 `stopSelf()` 方法就能让这个服务停止下来了。

那么接下来又有一个问题需要思考了，我们如何才能证实服务已经成功启动或者停止了呢？最简单的方法就是在 `MyService` 的几个方法中加入打印日志，如下所示：

```

public class MyService extends Service {

    ...

    @Override
    public void onCreate() {
        super.onCreate();
        Log.d("MyService", "onCreate executed");
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Log.d("MyService", "onStartCommand executed");
        return super.onStartCommand(intent, flags, startId);
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.d("MyService", "onDestroy executed");
    }
}

```

现在可以运行一下程序来进行测试了，程序的主界面如图 10.6 所示。

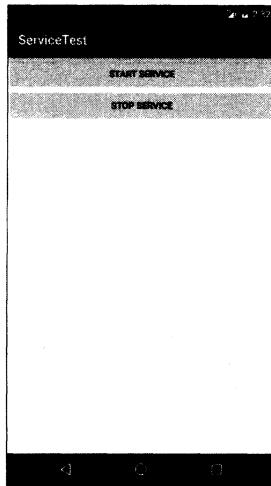


图 10.6 ServiceTest 的主界面

点击一下 Start Service 按钮，观察 logcat 中的打印日志，如图 10.7 所示。

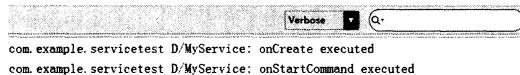


图 10.7 启动服务时的打印日志

MyService 中的 `onCreate()` 和 `onStartCommand()` 方法都执行了，说明这个服务确实已经启动成功了，并且你还可以在 `Settings→Developer options→Running services` 中找到它，如图 10.8 所示。

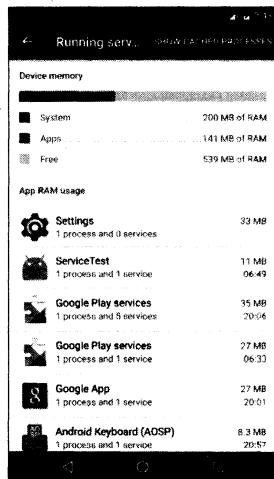


图 10.8 正在运行的服务列表

然后再点击一下 Stop Service 按钮，观察 logcat 中的打印日志，如图 10.9 所示。

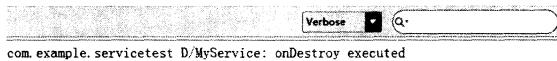


图 10.9 停止服务时的打印日志

由此证明，MyService 确实已经成功停止下来了。

话说回来，虽然我们已经学会了启动服务以及停止服务的方法，不知道你心里现在有没有一个疑惑，那就是 `onCreate()` 方法和 `onStartCommand()` 方法到底有什么区别呢？因为刚刚点击 Start Service 按钮后两个方法都执行了。

其实 `onCreate()` 方法是在服务第一次创建的时候调用的，而 `onStartCommand()` 方法则在每次启动服务的时候都会调用，由于刚才我们是第一次点击 Start Service 按钮，服务此时还未创建过，所以两个方法都会执行，之后如果你再连续多点击几次 Start Service 按钮，你就会发现只有 `onStartCommand()` 方法可以得到执行了。

10.3.3 活动和服务进行通信

上一小节中我们学习了启动和停止服务的方法，不知道你有没有发现，虽然服务是在活动里启动的，但在启动了服务之后，活动与服务基本就没有什么关系了。确实如此，我们在活动里调用了 `startService()` 方法来启动 MyService 这个服务，然后 MyService 的 `onCreate()` 和 `onStartCommand()` 方法就会得到执行。之后服务会一直处于运行状态，但具体运行的是什么逻辑，活动就控制不了了。这就类似于活动通知了服务一下：“你可以启动了！”然后服务就去忙自己的事情了，但活动并不知道服务到底做了什么事情，以及完成得如何。

那么有没有什么办法能让活动和服务的关系更紧密一些呢？例如在活动中指挥服务去干什么，服务就去干什么。当然可以，这就需要借助我们刚刚忽略的 `onBind()` 方法了。

比如说，目前我们希望在 MyService 里提供一个下载功能，然后在活动中可以决定何时开始下载，以及随时查看下载进度。实现这个功能的思路是创建一个专门的 `Binder` 对象来对下载功能进行管理，修改 MyService 中的代码，如下所示：

```
public class MyService extends Service {
    private DownloadBinder mBinder = new DownloadBinder();
    class DownloadBinder extends Binder {
        public void startDownload() {
            Log.d("MyService", "startDownload executed");
        }
        public int getProgress() {
            Log.d("MyService", "getProgress executed");
        }
    }
}
```

```

        return 0;
    }

}

@Override
public IBinder onBind(Intent intent) {
    return mBinder;
}

...
}

```

可以看到，这里我们新建了一个 `DownloadBinder` 类，并让它继承自 `Binder`，然后在它的内部提供了开始下载以及查看下载进度的方法。当然这只是两个模拟方法，并没有实现真正的功能，我们在这两个方法中分别打印了一行日志。

接着，在 `MyService` 中创建了 `DownloadBinder` 的实例，然后在 `onBind()` 方法里返回了这个实例，这样 `MyService` 中的工作就全部完成了。

下面就要看一看，在活动中如何去调用服务里的这些方法了。首先需要在布局文件里新增两个按钮，修改 `activity_main.xml` 中的代码，如下所示：

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    ...

    <Button
        android:id="@+id/bind_service"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Bind Service" />

    <Button
        android:id="@+id/unbind_service"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Unbind Service" />

</LinearLayout>

```

这两个按钮分别是用于绑定服务和取消绑定服务的，那到底谁需要去和服务绑定呢？当然就是活动了。当一个活动和服务绑定了之后，就可以调用该服务里的 `Binder` 提供的方法了。修改 `MainActivity` 中的代码，如下所示：

```

public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    private MyService.DownloadBinder downloadBinder;

```

```

private ServiceConnection connection = new ServiceConnection() {

    @Override
    public void onServiceDisconnected(ComponentName name) {
    }

    @Override
    public void onServiceConnected(ComponentName name, IBinder service) {
        downloadBinder = (MyService.DownloadBinder) service;
        downloadBinder.startDownload();
        downloadBinder.getProgress();
    }
};

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    ...

    Button bindService = (Button) findViewById(R.id.bind_service);
    Button unbindService = (Button) findViewById(R.id.unbind_service);
    bindService.setOnClickListener(this);
    unbindService.setOnClickListener(this);
}

@Override
public void onClick(View v) {
    switch (v.getId()) {
        ...
        case R.id.bind_service:
            Intent bindIntent = new Intent(this, MyService.class);
            bindService(bindIntent, connection, BIND_AUTO_CREATE); // 绑定服务
            break;
        case R.id.unbind_service:
            unbindService(connection); // 解绑服务
            break;
        default:
            break;
    }
}
}

```

可以看到，这里我们首先创建了一个 ServiceConnection 的匿名类，在里面重写了 onServiceConnected()方法和 onServiceDisconnected()方法，这两个方法分别会在活动与服务成功绑定以及解除绑定的时候调用。在 onServiceConnected()方法中，我们又通过向下转型得到了 DownloadBinder 的实例，有了这个实例，活动和服务之间的关系就变得非常紧密了。现在我们可以在活动中根据具体的场景来调用 DownloadBinder 中的任何 public()方法，即实现了指挥服务干什么服务就去干什么的功能。这里仍然只是做了个简单的测试，在 onServiceConnected()方法中调用了 DownloadBinder 的 startDownload()和 getProgress()方法。

当然，现在活动和服务其实还没进行绑定呢，这个功能是在 Bind Service 按钮的点击事件里完成的。可以看到，这里我们仍然是构建出了一个 Intent 对象，然后调用 `bindService()` 方法将 MainActivity 和 MyService 进行绑定。`bindService()` 方法接收 3 个参数，第一个参数就是刚刚构建出的 Intent 对象，第二个参数是前面创建出的 ServiceConnection 的实例，第三个参数则是一个标志位，这里传入 `BIND_AUTO_CREATE` 表示在活动和服务进行绑定后自动创建服务。这会使得 MyService 中的 `onCreate()` 方法得到执行，但 `onStartCommand()` 方法不会执行。

然后如果我们想解除活动和服务之间的绑定该怎么办呢？调用一下 `unbindService()` 方法就可以了，这也是 Unbind Service 按钮的点击事件里实现的功能。

现在让我们重新运行一下程序吧，界面如图 10.10 所示。

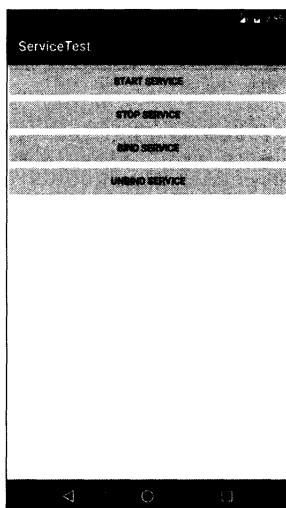


图 10.10 ServiceTest 新的主界面

点击一下 Bind Service 按钮，然后观察 logcat 中的打印日志，如图 10.11 所示。

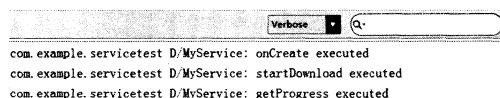


图 10.11 绑定服务时的打印日志

可以看到，首先是 MyService 的 `onCreate()` 方法得到了执行，然后 `startDownload()` 和 `getProgress()` 方法都得到了执行，说明我们确实已经在活动里成功调用了服务里提供的方法了。

另外需要注意，任何一个服务在整个应用程序范围内都是通用的，即 MyService 不仅可以和 MainActivity 绑定，还可以和任何一个其他的活动进行绑定，而且在绑定完成后它们都可以获取到相同的 DownloadBinder 实例。

10.4 服务的生命周期

之前我们学习过了活动以及碎片的生命周期。类似地，服务也有自己的生命周期，前面我们使用到的 `onCreate()`、`onStartCommand()`、`onBind()` 和 `onDestroy()` 等方法都是在服务的生命周期内可能回调的方法。

一旦在项目的任何位置调用了 Context 的 `startService()` 方法，相应的服务就会启动起来，并回调 `onStartCommand()` 方法。如果这个服务之前还没有创建过，`onCreate()` 方法会先于 `onStartCommand()` 方法执行。服务启动了之后会一直保持运行状态，直到 `stopService()` 或 `stopSelf()` 方法被调用。注意，虽然每调用一次 `startService()` 方法，`onStartCommand()` 就会执行一次，但实际上每个服务都只会存在一个实例。所以不管你调用了多少次 `startService()` 方法，只需调用一次 `stopService()` 或 `stopSelf()` 方法，服务就会停止下来了。

另外，还可以调用 Context 的 `bindService()` 来获取一个服务的持久连接，这时就会回调服务中的 `onBind()` 方法。类似地，如果这个服务之前还没有创建过，`onCreate()` 方法会先于 `onBind()` 方法执行。之后，调用方可以获取到 `onBind()` 方法里返回的 `IBinder` 对象的实例，这样就能自由地和服务进行通信了。只要调用方和服务之间的连接没有断开，服务就会一直保持运行状态。

当调用了 `startService()` 方法后，又去调用 `stopService()` 方法，这时服务中的 `onDestroy()` 方法就会执行，表示服务已经销毁了。类似地，当调用了 `bindService()` 方法后，又去调用 `unbindService()` 方法，`onDestroy()` 方法也会执行，这两种情况都很好理解。但是需要注意，我们是完全有可能对一个服务既调用了 `startService()` 方法，又调用了 `bindService()` 方法的，这种情况下该如何才能让服务销毁掉呢？根据 Android 系统的机制，一个服务只要被启动或者被绑定了之后，就会一直处于运行状态，必须要让以上两种条件同时不满足，服务才能被销毁。所以，这种情况下要同时调用 `stopService()` 和 `unbindService()` 方法，`onDestroy()` 方法才会执行。

这样你就已经把服务的生命周期完整地走了一遍。

10.5 服务的更多技巧

以上所学的都是关于服务最基本的一些用法和概念，当然也是最常用的。不过，仅仅满足于此显然是不够的，关于的更多高级使用技巧还在等着我们呢，下面就赶快去看一看吧。

10.5.1 使用前台服务

服务几乎都是在后台运行的，一直以来它都是默默地做着辛苦的工作。但是服务的系统优先级还是比较低的，当系统出现内存不足的情况时，就有可能会回收掉正在后台运行的服务。如果你希望服务可以一直保持运行状态，而不会由于系统内存不足的原因导致被回收，就可以考虑使

用前台服务。前台服务和普通服务最大的区别就在于，它会一直有一个正在运行的图标在系统的状态栏显示，下拉状态栏后可以看到更加详细的信息，非常类似于通知的效果。当然有时候你也可能不仅仅是为了防止服务被回收掉才使用前台服务的，有些项目由于特殊的需求会要求必须使用前台服务，比如说彩云天气这款天气预报应用，它的服务在后台更新天气数据的同时，还会在系统状态栏一直显示当前的天气信息，如图 10.12 所示。



图 10.12 彩云天气的前台服务效果

那么我们就来看一下如何才能创建一个前台服务吧，其实并不复杂，修改 MyService 中的代码，如下所示：

```
public class MyService extends Service {  
    ...  
  
    @Override  
    public void onCreate() {  
        super.onCreate();  
        Log.d("MyService", "onCreate executed");  
        Intent intent = new Intent(this, MainActivity.class);  
        PendingIntent pi = PendingIntent.getActivity(this, 0, intent, 0);  
        Notification notification = new NotificationCompat.Builder(this)  
            .setContentTitle("This is content title")  
            .setContentText("This is content text")  
            .setWhen(System.currentTimeMillis())  
            .setSmallIcon(R.mipmap.ic_launcher)  
            .setLargeIcon(BitmapFactory.decodeResource(getResources(),  
                R.mipmap.ic_launcher))  
            .setContentIntent(pi)  
            .build();  
        startForeground(1, notification);  
    }  
}
```

```
    }  
    ...  
}
```

可以看到，这里只是修改了 `onCreate()` 方法中的代码，相信这部分代码你会非常眼熟。没错！这就是我们在第 8 章中学习的创建通知的方法。只不过这次在构建出 `Notification` 对象后并没有使用 `NotificationManager` 来将通知显示出来，而是调用了 `startForeground()` 方法。这个方法接收两个参数，第一个参数是通知的 id，类似于 `notify()` 方法的第一个参数，第二个参数则是构建出的 `Notification` 对象。调用 `startForeground()` 方法后就会让 `MyService` 变成一个前台服务，并在系统状态栏显示出来。

现在重新运行一下程序，并点击 Start Service 或 Bind Service 按钮，`MyService` 就会以前台服务的模式启动了，并且在系统状态栏会显示一个通知图标，下拉状态栏后可以看到该通知的详细内容，如图 10.13 所示。

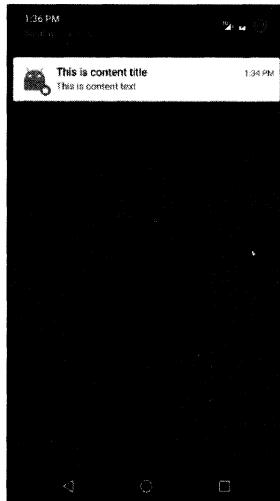


图 10.13 前台服务的状态栏效果

前台服务的用法就这么简单，只要你在第 8 章中将通知的用法掌握好了，学习本节的知识一定会特别轻松。

10.5.2 使用 IntentService

话说回来，在本章一开始的时候我们就已经知道，服务中的代码都是默认运行在主线程当中的，如果直接在服务里去处理一些耗时的逻辑，就很容易出现 ANR (Application Not Responding) 的情况。

所以这个时候就需要用到 Android 多线程编程的技术了，我们应该在服务的每个具体的方法里开启一个子线程，然后在这里去处理那些耗时的逻辑。因此，一个比较标准的服务就可以写成如下形式：

```
public class MyService extends Service {
    ...
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        new Thread(new Runnable() {
            @Override
            public void run() {
                // 处理具体的逻辑
            }
        }).start();
        return super.onStartCommand(intent, flags, startId);
    }
}
```

但是，这种服务一旦启动之后，就会一直处于运行状态，必须调用 `stopService()` 或者 `stopSelf()` 方法才能让服务停止下来。所以，如果想要实现让一个服务在执行完毕后自动停止的功能，就可以这样写：

```
public class MyService extends Service {
    ...
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        new Thread(new Runnable() {
            @Override
            public void run() {
                // 处理具体的逻辑
                stopSelf();
            }
        }).start();
        return super.onStartCommand(intent, flags, startId);
    }
}
```

虽说这种写法并不复杂，但是总会有一些程序员忘记开启线程，或者忘记调用 `stopSelf()` 方法。为了可以简单地创建一个异步的、会自动停止的服务，Android 专门提供了一个 `IntentService` 类，这个类就很好地解决了前面所提到的两种尴尬，下面我们就来看一下它的用法。

新建一个 `MyIntentService` 类继承自 `IntentService`，代码如下所示：

```

public class MyIntentService extends IntentService {

    public MyIntentService() {
        super("MyIntentService"); // 调用父类的有参构造函数
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        // 打印当前线程的 id
        Log.d("MyIntentService", "Thread id is " + Thread.currentThread().getId());
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.d("MyIntentService", "onDestroy executed");
    }
}

```

这里首先要提供一个无参的构造函数，并且必须在其内部调用父类的有参构造函数。然后要在子类中去实现 `onHandleIntent()` 这个抽象方法，在这个方法中可以去处理一些具体的逻辑，而且不用担心 ANR 的问题，因为这个方法已经是在子线程中运行的了。这里为了证实一下，我们在 `onHandleIntent()` 方法中打印了当前线程的 id。另外根据 `IntentService` 的特性，这个服务在运行结束后应该是会自动停止的，所以我们又重写了 `onDestroy()` 方法，在这里也打印了一行日志，以证实服务是不是停止掉了。

接下来修改 `activity_main.xml` 中的代码，加入一个用于启动 `MyIntentService` 这个服务的按钮，如下所示：

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    ...

    <Button
        android:id="@+id/start_intent_service"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Start IntentService" />

</LinearLayout>

```

然后修改 `MainActivity` 中的代码，如下所示：

```

public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    ...

```

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    ...
    Button startIntentService = (Button) findViewById(R.id.start_intent_
        service);
    startIntentService.setOnClickListener(this);
}

@Override
public void onClick(View v) {
    switch (v.getId()) {
        ...
        case R.id.start_intent_service:
            // 打印主线程的 id
            Log.d("MainActivity", "Thread id is " + Thread.currentThread() .
                getId());
            Intent intentService = new Intent(this, MyIntentService.class);
            startService(intentService);
            break;
        default:
            break;
    }
}
}

```

可以看到，我们在 Start IntentService 按钮的点击事件里面去启动 MyIntentService 这个服务，并在这里打印了一下主线程的 id，稍后用于和 IntentService 进行比对。你会发现，其实 IntentService 的用法和普通的服务没什么两样。

最后不要忘记，服务都是需要在 AndroidManifest.xml 里注册的，如下所示：

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.servicetest">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        ...
        <service android:name=".MyIntentService" />

    </application>

</manifest>

```

当然你也可以使用 Android Studio 提供的快捷方式来创建 IntentService，不过由于这样会自动生成一些我们用不到的代码，因此这里我采用了手动创建的方式。

现在重新运行一下程序，界面如图 10.14 所示。

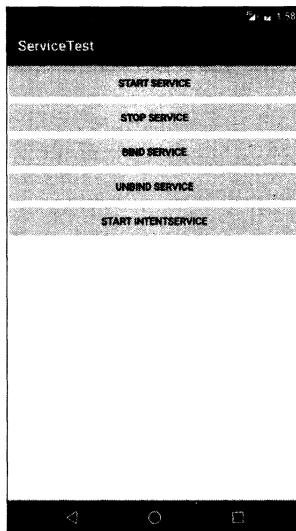


图 10.14 ServiceTest 更新后的主界面

点击 Start IntentService 按钮后，观察 logcat 中的打印日志，如图 10.15 所示。

```
Verbose
/com.example.servicetest D/MainActivity: Thread id is 1
/com.example.servicetest D/MyIntentService: Thread id is 154
/com.example.servicetest D/MyIntentService: onDestroy executed
```

图 10.15 启动 IntentService 时的打印日志

可以看到，不仅 MyIntentService 和 MainActivity 所在的线程 id 不一样，而且 `onDestroy()` 方法也得到了执行，说明 MyIntentService 在运行完毕后确实自动停止了。集开启线程和自动停止于一身，IntentService 还是博得了不少程序员的喜爱。

好了，关于服务的知识点你已经学得够多了，下面就让我们进入到本章的最佳实践环节吧。

10.6 服务的最佳实践——完整版的下载示例

本章中你已经掌握了很多关于服务的使用技巧，但是当在真正的项目里需要用到服务的时候，可能还会有一些棘手的问题让你不知所措。因此，下面我们就来综合运用一下，尝试实现一个在服务中经常会使用到的功能——下载。

本节中我们将要编写一个完整版的下载示例，其中会涉及第 7 章、第 8 章、第 9 章和第 10 章的部分内容，算是目前为止综合程度最高的一个例子了。准备好了吗？创建一个 ServiceBestPractice 项目，然后开始本节的学习之旅吧。

首先我们需要将项目中会使用到的依赖库添加好，编辑 app/build.gradle 文件，在 dependencies 闭包中添加如下内容：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:24.2.1'
    testCompile 'junit:junit:4.12'
    compile 'com.squareup.okhttp3:okhttp:3.4.1'
}
```

这里只需添加一个 OkHttp 的依赖就行了，待会儿在编写网络相关的功能时，我们将使用 OkHttp 来进行实现。

接下来需要定义一个回调接口，用于对下载过程中的各种状态进行监听和回调。新建一个 DownloadListener 接口，代码如下所示：

```
public interface DownloadListener {
    void onProgress(int progress);
    void onSuccess();
    void onFailed();
    void onPause();
    void onCancel();
}
```

可以看到，这里我们一共定义了 5 个回调方法，onProgress() 方法用于通知当前的下载进度，onSuccess() 方法用于通知下载成功事件，onFailed() 方法用于通知下载失败事件，onPaused() 方法用于通知下载暂时事件，onCancelled() 方法用于通知下载取消事件。

回调接口定义好了之后，下面我们就可以开始编写下载功能了。这里我准备使用本章中刚学的 AsyncTask 来进行实现，新建一个 DownloadTask 继承自 AsyncTask，代码如下所示：

```
public class DownloadTask extends AsyncTask<String, Integer, Integer> {
    public static final int TYPE_SUCCESS = 0;
    public static final int TYPE_FAILED = 1;
    public static final int TYPE_PAUSED = 2;
    public static final int TYPE_CANCELED = 3;

    private DownloadListener listener;
```

```
private boolean isCanceled = false;

private boolean isPaused = false;

private int lastProgress;

public DownloadTask(DownloadListener listener) {
    this.listener = listener;
}

@Override
protected Integer doInBackground(String... params) {
    InputStream is = null;
    RandomAccessFile savedFile = null;
    File file = null;
    try {
        long downloadedLength = 0; // 记录已下载的文件长度
        String downloadUrl = params[0];
        String fileName = downloadUrl.substring(downloadUrl.lastIndexOf("/"));
        String directory = Environment.getExternalStoragePublicDirectory
            (Environment.DIRECTORY_DOWNLOADS).getPath();
        file = new File(directory + fileName);
        if (file.exists()) {
            downloadedLength = file.length();
        }
        long contentLength = getContentLength(downloadUrl);
        if (contentLength == 0) {
            return TYPE_FAILED;
        } else if (contentLength == downloadedLength) {
            // 已下载字节和文件总字节相等，说明已经下载完成了
            return TYPE_SUCCESS;
        }
    OkHttpClient client = new OkHttpClient();
    Request request = new Request.Builder()
        // 断点下载，指定从哪个字节开始下载
        .addHeader("RANGE", "bytes=" + downloadedLength + "-")
        .url(downloadUrl)
        .build();
    Response response = client.newCall(request).execute();
    if (response != null) {
        is = response.body().byteStream();
        savedFile = new RandomAccessFile(file, "rw");
        savedFile.seek(downloadedLength); // 跳过已下载的字节
        byte[] b = new byte[1024];
        int total = 0;
        int len;
        while ((len = is.read(b)) != -1) {
            if (isCanceled) {
                return TYPE_CANCELED;
            } else if (isPaused) {
                return TYPE_PAUSED;
            } else {
                total += len;
            }
        }
    }
}
```

```
        savedFile.write(b, 0, len);
        // 计算已下载的百分比
        int progress = (int) ((total + downloadedLength) * 100 /
            contentLength);
        publishProgress(progress);
    }
}
response.body().close();
return TYPE_SUCCESS;
}
} catch (Exception e) {
    e.printStackTrace();
} finally {
    try {
        if (is != null) {
            is.close();
        }
        if (savedFile != null) {
            savedFile.close();
        }
        if (isCanceled && file != null) {
            file.delete();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
return TYPE_FAILED;
}

@Override
protected void onProgressUpdate(Integer... values) {
    int progress = values[0];
    if (progress > lastProgress) {
        listener.onProgress(progress);
        lastProgress = progress;
    }
}

@Override
protected void onPostExecute(Integer status) {
    switch (status) {
        case TYPE_SUCCESS:
            listener.onSuccess();
            break;
        case TYPE_FAILED:
            listener.onFailed();
            break;
        case TYPE_PAUSED:
            listener.onPause();
            break;
        case TYPE_CANCELED:
            listener.onCanceled();
            break;
        default:
```

```

        break;
    }
}

public void pauseDownload() {
    isPaused = true;
}

public void cancelDownload() {
    isCanceled = true;
}

private long getContentLength(String downloadUrl) throws IOException {
    OkHttpClient client = new OkHttpClient();
    Request request = new Request.Builder()
        .url(downloadUrl)
        .build();
    Response response = client.newCall(request).execute();
    if (response != null && response.isSuccessful()) {
        long contentLength = response.body().contentLength();
        response.close();
        return contentLength;
    }
    return 0;
}
}

```

这段代码就比较长了，我们需要一步步地进行分析。首先看一下 `AsyncTask` 中的 3 个泛型参数：第一个泛型参数指定为 `String`，表示在执行 `AsyncTask` 的时候需要传入一个字符串参数给后台任务；第二个泛型参数指定为 `Integer`，表示使用整型数据来作为进度显示单位；第三个泛型参数指定为 `Integer`，则表示使用整型数据来反馈执行结果。

接下来我们定义了 4 个整型常量用于表示下载的状态，`TYPE_SUCCESS` 表示下载成功，`TYPE_FAILED` 表示下载失败，`TYPE_PAUSED` 表示暂停下载，`TYPE_CANCELED` 表示取消下载。然后在 `DownloadTask` 的构造函数中要求传入一个刚刚定义的 `DownloadListener` 参数，我们待会就会将下载的状态通过这个参数进行回调。

接着就是要重写 `doInBackground()`、`onProgressUpdate()` 和 `onPostExecute()` 这 3 个方法了，我们之前已经学习过这 3 个方法各自的作用，因此在这里它们各自所负责的任务也是明确的：`doInBackground()` 方法用于在后台执行具体的下载逻辑，`onProgressUpdate()` 方法用于在界面上更新当前的下载进度，`onPostExecute()` 用于通知最终的下载结果。

那么先来看一下 `doInBackground()` 方法，首先我们从参数中获取到了下载的 URL 地址，并根据 URL 地址解析出了下载的文件名，然后指定将文件下载到 `Environment.DIRECTORY_DOWNLOADS` 目录下，也就是 SD 卡的 `Download` 目录。我们还要判断一下 `Download` 目录中是

不是已经存在要下载的文件了，如果已经存在的话则读取已下载的字节数，这样就可以在后面启用断点续传的功能。接下来先是调用了 `getContentLength()` 方法来获取待下载文件的总长度，如果文件长度等于 0 则说明文件有问题，直接返回 `TYPE_FAILED`，如果文件长度等于已下载文件长度，那么就说明文件已经下载完了，直接返回 `TYPE_SUCCESS` 即可。紧接着使用 OkHttp 来发送一条网络请求，需要注意的是，这里在请求中添加了一个 header，用于告诉服务器我们想要从哪个字节开始下载，因为已下载过的部分就不需要再重新下载了。接下来读取服务器响应的数据，并使用 Java 的文件流方式，不断从网络上读取数据，不断写入到本地，一直到文件全部下载完成为止。在这个过程中，我们还要判断用户有没有触发暂停或者取消的操作，如果说有的话则返回 `TYPE_PAUSED` 或 `TYPE_CANCELED` 来中断下载，如果没有的话则实时计算当前的下载进度，然后调用 `publishProgress()` 方法进行通知。暂停和取消操作都是使用一个布尔型的变量来进行控制的，调用 `pauseDownload()` 或 `cancelDownload()` 方法即可更改变量的值。

接下来看一下 `onProgressUpdate()` 方法，这个方法就简单得多了，它首先从参数中获取到当前的下载进度，然后和上一次的下载进度进行对比，如果有变化的话则调用 `DownloadListener` 的 `onProgress()` 方法来通知下载进度更新。

最后是 `onPostExecute()` 方法，也非常简单，就是根据参数中传入的下载状态来进行回调。下载成功就调用 `DownloadListener` 的 `onSuccess()` 方法，下载失败就调用 `onFailed()` 方法，暂停下载就调用 `onPaused()` 方法，取消下载就调用 `onCanceled()` 方法。

这样我们就把具体的下载功能完成了，下面为了保证 `DownloadTask` 可以一直在后台运行，我们还需要创建一个下载的服务。右击 `com.example.servicebestpractice` → `New` → `Service` → `Service`，新建 `DownloadService`，然后修改其中的代码，如下所示：

```
public class DownloadService extends Service {  
  
    private DownloadTask downloadTask;  
  
    private String downloadUrl;  
  
    private DownloadListener listener = new DownloadListener() {  
        @Override  
        public void onProgress(int progress) {  
            getNotificationManager().notify(1, getNotification("Downloading...",  
                progress));  
        }  
  
        @Override  
        public void onSuccess() {  
            downloadTask = null;  
            // 下载成功时将前台服务通知关闭，并创建一个下载成功的通知  
            stopForeground(true);  
            getNotificationManager().notify(1, getNotification("Download Success",  
                -1));  
            Toast.makeText(DownloadService.this, "Download Success",  
                Toast.LENGTH_SHORT).show();  
        }  
    };  
}
```

```
}

@Override
public void onFailed() {
    downloadTask = null;
    // 下载失败时将前台服务通知关闭，并创建一个下载失败的通知
    stopForeground(true);
    getNotificationManager().notify(1, getNotification("Download Failed",
        -1));
    Toast.makeText(DownloadService.this, "Download Failed",
        Toast.LENGTH_SHORT).show();
}

@Override
public void onPause() {
    downloadTask = null;
    Toast.makeText(DownloadService.this, "Paused", Toast.LENGTH_SHORT).
        show();
}

@Override
public void onCancel() {
    downloadTask = null;
    stopForeground(true);
    Toast.makeText(DownloadService.this, "Cancelled", Toast.LENGTH_SHORT).
        show();
}

private DownloadBinder mBinder = new DownloadBinder();

@Override
public IBinder onBind(Intent intent) {
    return mBinder;
}

class DownloadBinder extends Binder {

    public void startDownload(String url) {
        if (downloadTask == null) {
            downloadUrl = url;
            downloadTask = new DownloadTask(listener);
            downloadTask.execute(downloadUrl);
            startForeground(1, getNotification("Downloading...", 0));
            Toast.makeText(DownloadService.this, "Downloading...", Toast.
                LENGTH_SHORT).show();
        }
    }

    public void pauseDownload() {
        if (downloadTask != null) {
            downloadTask.pauseDownload();
        }
    }
}
```

```

    }

    public void cancelDownload() {
        if (downloadTask != null) {
            downloadTask.cancelDownload();
        } else {
            if (downloadUrl != null) {
                // 取消下载时需将文件删除，并将通知关闭
                String fileName = downloadUrl.substring(downloadUrl.
                    lastIndexOf("/"));
                String directory = Environment.getExternalStoragePublicDirectory
                    (Environment.DIRECTORY_DOWNLOADS).getPath();
                File file = new File(directory + fileName);
                if (file.exists()) {
                    file.delete();
                }
                getNotificationManager().cancel(1);
                stopForeground(true);
                Toast.makeText(DownloadService.this, "Canceled",
                    Toast.LENGTH_SHORT).show();
            }
        }
    }

    private NotificationManager getNotificationManager() {
        return (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
    }

    private Notification getNotification(String title, int progress) {
        Intent intent = new Intent(this, MainActivity.class);
        PendingIntent pi = PendingIntent.getActivity(this, 0, intent, 0);
        NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
        builder.setSmallIcon(R.mipmap.ic_launcher);
        builder.setLargeIcon(BitmapFactory.decodeResource(getResources(),
            R.mipmap.ic_launcher));
        builder.setContentIntent(pi);
        builder.setContentTitle(title);
        if (progress > 0) {
            // 当 progress 大于或等于 0 时才显示下载进度
            builder.setContentText(progress + "%");
            builder.setProgress(100, progress, false);
        }
        return builder.build();
    }
}

```

这段代码同样也比较长，我们还是得耐心慢慢看。首先这里创建了一个 `DownloadListener` 的匿名类实例，并在匿名类中实现了 `onProgress()`、`onSuccess()`、`onFailed()`、`onPaused()` 和 `onCanceled()` 这 5 个方法。在 `onProgress()` 方法中，我们调用 `getNotification()` 方法构

建了一个用于显示下载进度的通知，然后调用 `NotificationManager` 的 `notify()` 方法去触发这个通知，这样就可以在下拉状态栏中实时看到当前下载的进度了。在 `onSuccess()` 方法中，我们首先是将正在下载的前台通知关闭，然后创建一个新的通知用于告诉用户下载成功了。其他几个方法也都是类似的，分别用于告诉用户下载失败、暂停和取消这几个事件。

接下来为了要让 `DownloadService` 可以和活动进行通信，我们又创建了一个 `DownloadBinder`。`DownloadBinder` 中提供了 `startDownload()`、`pauseDownload()` 和 `cancelDownload()` 这 3 个方法，那么顾名思义，它们分别是用于开始下载、暂停下载和取消下载的。在 `startDownload()` 方法中，我们创建了一个 `DownloadTask` 的实例，把刚才的 `DownloadListener` 作为参数传入，然后调用 `execute()` 方法开启下载，并将下载文件的 URL 地址传入到 `execute()` 方法中。同时，为了让这个下载服务成为一个前台服务，我们还调用了 `startForeground()` 方法，这样就会在系统状态栏中创建一个持续运行的通知了。接着往下看，`pauseDownload()` 方法中的代码就非常简单了，就是简单地调用了一下 `DownloadTask` 中的 `pauseDownload()` 方法。`cancelDownload()` 方法中的逻辑也基本类似，但是要注意，取消下载的时候我们需要将正在下载的文件删除掉，这一点和暂停下载是不同的。

另外，`DownloadService` 类中所有使用到的通知都是调用 `getNotification()` 方法进行构建的，这个方法中的代码我们之前基本都是学过的，只有一个 `setProgress()` 方法没有见过。`setProgress()` 方法接收 3 个参数，第一个参数传入通知的最大进度，第二个参数传入通知的当前进度，第三个参数表示是否使用模糊进度条，这里传入 `false`。设置完 `setProgress()` 方法，通知上就会有进度条显示出来了。

现在下载的服务也已经成功实现，后端的工作基本都完成了，那么接下来我们开始编写前端的部分。修改 `activity_main.xml` 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/start_download"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Start Download" />

    <Button
        android:id="@+id/pause_download"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Pause Download" />

    <Button
        android:id="@+id/cancel_download"
        android:layout_width="match_parent"
```

```

    android:layout_height="wrap_content"
    android:text="Cancel Download" />

</LinearLayout>

```

布局文件还是非常简单的，这里在 LinearLayout 中放置了 3 个按钮，分别用于开始下载、暂停下载和取消下载。

然后修改 MainActivity 中的代码，如下所示：

```

public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    private DownloadService.DownloadBinder downloadBinder;

    private ServiceConnection connection = new ServiceConnection() {

        @Override
        public void onServiceDisconnected(ComponentName name) {
        }

        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            downloadBinder = (DownloadService.DownloadBinder) service;
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button startDownload = (Button) findViewById(R.id.start_download);
        Button pauseDownload = (Button) findViewById(R.id.pause_download);
        Button cancelDownload = (Button) findViewById(R.id.cancel_download);
        startDownload.setOnClickListener(this);
        pauseDownload.setOnClickListener(this);
        cancelDownload.setOnClickListener(this);
        Intent intent = new Intent(this, DownloadService.class);
        startService(intent); // 启动服务
        bindService(intent, connection, BIND_AUTO_CREATE); // 绑定服务
        if (ContextCompat.checkSelfPermission(MainActivity.this, Manifest.
            permission.WRITE_EXTERNAL_STORAGE)!= PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(MainActivity.this, new
                String[]{ Manifest.permission.WRITE_EXTERNAL_STORAGE }, 1);
        }
    }

    @Override
    public void onClick(View v) {
        if (downloadBinder == null) {
            return;
        }
        switch (v.getId()) {

```

```

        case R.id.start_download:
            String url = "https://raw.githubusercontent.com/guolindev/eclipse/
                         master/eclipse-inst-win64.exe";
            downloadBinder.startDownload(url);
            break;
        case R.id.pause_download:
            downloadBinder.pauseDownload();
            break;
        case R.id.cancel_download:
            downloadBinder.cancelDownload();
            break;
        default:
            break;
    }
}

@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions,
                                       int[] grantResults) {
    switch (requestCode) {
        case 1:
            if (grantResults.length > 0 && grantResults[0] != PackageManager.
                PERMISSION_GRANTED) {
                Toast.makeText(this, "拒绝权限将无法使用程序", Toast.LENGTH_SHORT).
                    show();
                finish();
            }
            break;
        default:
    }
}

@Override
protected void onDestroy() {
    super.onDestroy();
    unbindService(connection);
}
}

```

可以看到，这里我们首先创建了一个 `ServiceConnection` 的匿名类，然后在 `onServiceConnected()`方法中获取到 `DownloadBinder` 的实例，有了这个实例，我们就可以在活动中调用服务提供的各种方法了。

接下来看一下 `onCreate()`方法，在这里我们对各个按钮都进行了初始化操作并设置了点击事件，然后分别调用了 `startService()`和 `bindService()`方法来启动和绑定服务。这一点至关重要，因为启动服务可以保证 `DownloadService`一直在后台运行，绑定服务则可以让 `MainActivity` 和 `DownloadService` 进行通信，因此两个方法调用都必不可少。在 `onCreate()`方法的最后，我们还进行了 `WRITE_EXTERNAL_STORAGE` 的运行时权限申请，因为下载文件是要下载到 SD 卡的 `Download` 目录下的，如果没有这个权限的话，我们整个程序都无法正常工作。

接下来的代码就非常简单了，在 `onClick()` 方法中我们对点击事件进行判断，如果点击了开始按钮就调用 `DownloadBinder` 的 `startDownload()` 方法，如果点击了暂停按钮就调用 `pauseDownload()` 方法，如果点击了取消按钮就调用 `cancelDownload()` 方法。`startDownload()` 方法中你可以传入任意的下载地址，这里我使用了一个 Eclipse 的下载地址，以此向这个 Android 平台上曾经最出色的开发工具致敬。

另外还有一点需要注意，如果活动被销毁了，那么一定要记得对服务进行解绑，不然就有可能会造成内存泄漏。这里我们在 `onDestroy()` 方法中完成了解绑操作。

现在只差最后一步了，我们还需要在 `AndroidManifest.xml` 文件中声明使用到的权限。当然除了权限之外，`MainActivity` 和 `DownloadService` 也是需要声明的，不过 Android Studio 应该早就帮我们将这两个组件声明好了，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.servicebestpractice">

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service
            android:name=".DownloadService"
            android:enabled="true"
            android:exported="true" />
    </application>

</manifest>
```

其中，由于我们的程序使用到了网络和访问 SD 卡的功能，因此需要声明 `INTERNET` 和 `WRITE_EXTERNAL_STORAGE` 这两个权限。

这样所有代码就都编写完了，现在终于可以运行一下程序了，如图 10.16 所示。

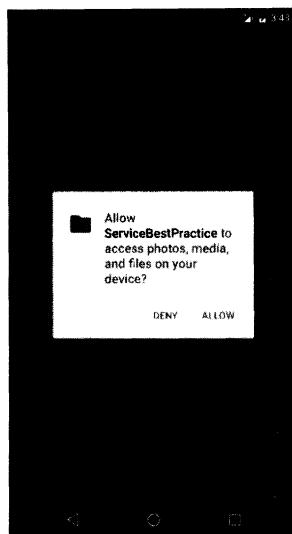


图 10.16 申请访问 SD 卡权限

程序一启动立刻就会申请访问 SD 卡的权限,这里我们点击 ALLOW,然后点击 Start Download 按钮就可以开始下载了。下载过程中可以下拉系统状态栏查看实时的下载进度,如图 10.17 所示。

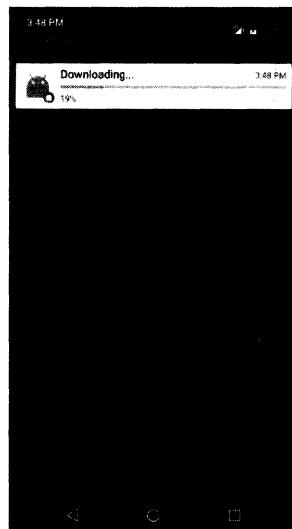


图 10.17 查看实时的下载进度

同时,我们还可以点击 Pause Download 或 Cancel Download,甚至于断网操作来测试这个下载程序的健壮性。最终下载完成后会弹出一个 Download Success 的通知,然后我们可以通过任意一个文件浏览器来查看一下 SD 卡的 Download 目录,如图 10.18 所示。

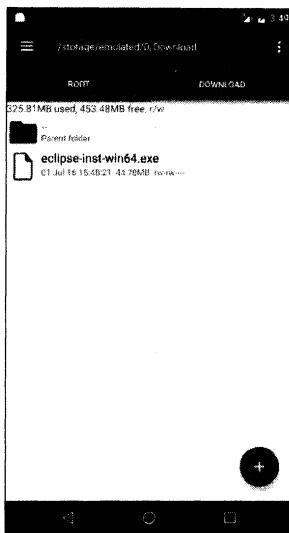


图 10.18 查看 SD 卡的 Download 目录

可以看到，文件已经成功下载下来了。

当然，我们还可以做一些更加丰富的操作，比如说再次点击 Start Download 按钮，你会发现程序会立刻弹出一个 Download Success 的提示，因为它检测到文件已经下载完成了，因而不会再重新去下载一遍。如果我们点击 Cancel Download 按钮先将下载文件删除掉，然后再点击 Start Download 按钮，你就会发现程序又会开始重新下载了。

总体来说，这个下载示例的稳定性还是挺不错的，而且综合性很强，将这个示例完全掌握了之后，你的水平肯定又更进一步了。

好了，最佳实践部分到此结束，下面我们就来回顾一下本章所学的内容吧。

10.7 小结与点评

在本章中，我们学习了很多与服务相关的重要知识点，包括 Android 多线程编程、服务的基本用法、服务的生命周期、前台服务和 IntentService 等。这些内容已经覆盖了大部分你在日常开发中可能用到的服务技术，再加上最佳实践部分学习的下载示例程序，相信以后不管遇到什么样的服务难题，你都能从容解决。

另外，本章同样是有里程碑式的纪念意义的，因为我们已经将 Android 中的四大组件全部学完，并且本书的内容也学习一大半了。对于你来说，现在你已经脱离了 Android 初级开发者的身份，并应该具备了独立完成很多功能的能力。

那么后面我们应该再接再厉，争取进一步提升自身的能力，所以现在还不是放松的时候，下一章中我们准备去学习一下 Android 特色开发的相关内容。

第 11 章

Android 特色开发——基于位置的服务

现在你已经学会了非常多的 Android 技能，并且通过这些技能你完全可以编写出相当不错的应用程序了。不过本章中，我们将要学习一些全新的 Android 技术，这些技术有别于传统的 PC 或 Web 领域的应用技术，是只有在移动设备上才能实现的。

说到只有在移动设备上才能实现的技术，很容易就让人联想到基于位置的服务（Location Based Service）。由于移动设备相比于电脑可以随身携带，我们通过地理定位的技术就可以随时得知自己所在的位置，从而围绕这一点开发出很多有意思的应用。本章中我们就将针对这一点进行讨论，学习一下基于位置的服务究竟是如何实现的。

11.1 基于位置的服务简介

基于位置的服务简称 LBS，随着移动互联网的兴起，这个技术在最近的几年里十分火爆。其实它本身并不是什么时髦的技术，主要的工作原理就是利用无线电通讯网络或 GPS 等定位方式来确定出移动设备所在的位置，而这种定位技术早在很多年前就已经出现了。

那为什么 LBS 技术直到最近几年才开始流行呢？这主要是因为，在过去移动设备的功能极其有限，即使定位到了设备所在的位置，也就仅仅只是定位到了而已，我们并不能在位置的基础上进行一些其他的操作。而现在就大大不同了，有了 Android 系统作为载体，我们可以利用定位出的位置进行许多丰富多彩的操作。比如说天气预报程序可以根据用户所在的位置自动选择城市，发微博的时候我们可以向朋友们晒一下自己在哪里，不认识路的时候随时打开地图就可以查询路线，等等。

介绍了这么多，相信你已经按捺不住了吧？我们马上就要开始本章的学习之旅，但在开始之前，还有一些事情是你必须要知道的。

首先你要清楚，基于位置的服务所围绕的核心就是要先确定出用户所在的位置。通常有两种技术方式可以实现：一种是通过 GPS 定位，一种是通过网络定位。GPS 定位的工作原理是基于手机内置的 GPS 硬件直接和卫星交互来获取当前的经纬度信息，这种定位方式精确度非常高，

但缺点是只能在室外使用，室内基本无法接收到卫星的信号。网络定位的工作原理是根据手机当前网络附近的三个基站进行测速，以此计算出手机和每个基站之间的距离，再通过三角定位确定出一个大概的位置，这种定位方式精确度一般，但优点是在室内室外都可以使用。

Android 对这两种定位方式都提供了相应的 API 支持，但是由于一些特殊原因，Google 的网络服务在中国不可访问，从而导致网络定位方式的 API 失效。而 GPS 定位虽然不需要网络，但是必须要在室外才可以使用，因此你在室内开发的时候很有可能会遇到不管使用哪种定位方式都无法成功定位的情况。

基于以上原因，我决定就不在本书中讲解 Android 原生定位 API 的用法了，而是使用一些国内第三方公司的 SDK。目前国内在这一领域做得比较好的一个是百度，一个是高德，本章我们就来学习一下百度在 LBS 方面提供的丰富多彩的功能。

11.2 申请 API Key

要想在自己的应用程序里使用百度的 LBS 功能，首先必须申请一个 API Key。你得拥有一个百度账号才能进行申请，我相信大多数人早就已经拥有了吧？如果你还没有的话，赶快去注册一个吧。

有了百度账号之后，我们就可以申请成为一名百度开发者了，登录你的百度账号，并打开 <http://developer.baidu.com/user/reg> 这个网址，在这里填写一些注册信息即可，如图 11.1 所示。

* 类型：
 个人 公司

* 开发者来源：
 开发者

* 开发者姓名：

* 开发者简介：

* Email 地址：
 修改

* 手机号：
 重新发送 (5 秒)

* 验证码：

开发者官方网站：

品牌 LOGO：
112px*54px, 支持PNG/JPG/GIF格式, 应用提交至PC
Web渠道时进行展示

我已阅读并同意百度开放云平台注册协议

提交

图 11.1 填写开发者信息

只需填写带有“*”号的那部分内容就足够了，接下来点击提交，会显示如图 11.2 所示的界面。



图 11.2 验证邮箱

接着点击“去我的邮箱”，将会进入到我们刚才填写的邮箱当中，这时收件箱中应该会有一封刚刚收到的邮件，这就是百度发送给我们的验证邮件，点击邮件当中的链接就可以完成注册了，如图 11.3 所示。

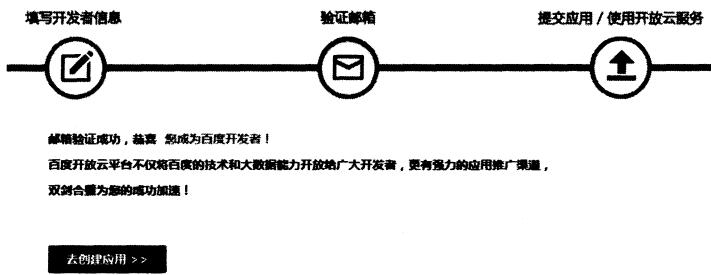


图 11.3 成为百度开发者

到此一切顺利！这样你就已经成为一名百度开发者了。接着访问 <http://lbsyun.baidu.com/apiconsole/key> 这个地址，然后同意百度开发者协议，会看到如图 11.4 所示的界面。



图 11.4 百度 LBS 开放平台主界面

由于这是一个刚刚注册的账号，所以目前的应用列表是空的。接下来点击创建应用就可以去申请 API Key 了，应用名称可以随便填，应用类型选择 Android SDK，启用服务保持默认即可，如图 11.5 所示。

应用名称： LBSTest

应用类型： Android SDK ▼

启用服务：

<input type="checkbox"/> 云检索API	<input checked="" type="checkbox"/> Javascript API	<input checked="" type="checkbox"/> Place API v2
<input checked="" type="checkbox"/> Geocoding API v2	<input checked="" type="checkbox"/> IP定位API	<input checked="" type="checkbox"/> 路线交通API
<input checked="" type="checkbox"/> Android地图SDK	<input checked="" type="checkbox"/> Android导航离线SDK	<input checked="" type="checkbox"/> Android导航SDK
<input checked="" type="checkbox"/> 静态图API	<input checked="" type="checkbox"/> 全景静态图API	<input checked="" type="checkbox"/> 坐标转换API
<input checked="" type="checkbox"/> 网络API	<input checked="" type="checkbox"/> 全景URL API	<input checked="" type="checkbox"/> Android导航 HUD SDK
<input checked="" type="checkbox"/> 云逆地理编码API	<input checked="" type="checkbox"/> Routematrix API	

* 发布版SHA1： 请输入发布版SHA1。

开发版SHA1： 请输入开发版SHA1。

* 包名： 请输入包名。

安全码： 输入sha1和包名后自动生成

Android SDK安全码组成：SHA1+包名。[\(查看详细配置方法\)](#)

新申请的Mobile与Browser类型的ak不再支持云存储接口的访问，如要使用云存储，请申请Server类型ak。

提交

图 11.5 创建应用界面

那么，这个发布版 SHA1 和开发版 SHA1 又是个什么东西呢？这是我们申请 API Key 所必须填写的一个字段，它指的是打包程序时所用签名文件的 SHA1 指纹，可以通过 Android Studio 查看到。打开 Android Studio 中的任意一个项目，点击右侧工具栏的 Gradle→项目名→:app→Tasks→android，如图 11.6 所示。

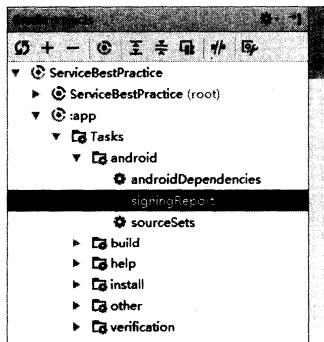


图 11.6 查看内置 Gradle Tasks

这里展示了一个 Android Studio 项目中所有内置的 Gradle Tasks，其中 signingReport 这个 Task 就可以用来查看签名文件信息。双击 signingReport，结果如图 11.7 所示。

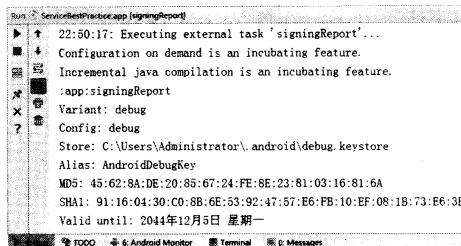


图 11.7 signingReport Task 的执行结果

其中，91:16:04:30:C0:8B:6E:53:92:47:57:E6:FB:10:EF:08:1B:73:E6:3E 就是我们所需的 SHA1 指纹了，当然你的 Android Studio 中显示的指纹和我的肯定是不一样的。另外需要注意，目前我们使用的是 debug.keystore 文件所生成的指纹，这是 Android 自动生成的一个用于测试的签名文件。而当你的应用程序发布时还需要创建一个正式的签名文件，如果要得到它的指纹，可以在 cmd 中输入如下命令：

```
keytool -list -v -keystore <签名文件路径>
```

然后输入正确的密码就可以了。创建签名文件的方法我们将在第 15 章中学习。

那么也就是说，现在得到的这个 SHA1 指纹实际上是一个开发版的 SHA1 指纹，不过因为暂时我们还没有一个发布版的 SHA1 指纹，因此这两个值都填成一样的就可以了。最后还剩下一个包名选项，虽然目前我们的应用程序还不存在，但可以先将包名预定下来，比如就叫 com.example.lbtest，这样所有的内容就都填写完整了，如图 11.8 所示。

应用名称：LBTest

应用类型：Android SDK

启用服务：

- 云检索API
- Javascript API
- Place API v2
- Geocoding API v2
- IP定位API
- 路线交通API
- Android地图SDK
- 全景静态图API
- Android导航SDK
- 静态图API
- 全景URL API
- 坐标转换API
- 唐僧API
- 全景HUD API
- Android导航HUD SDK
- 云逆地理编码API
- Routematrix API

* 发布版SHA1：91:16:04:30:C0:8B:6E:53:92:47:57:E6:FB:10:EF:08:1B:73:E6:3E

开发版SHA1：91:16:04:30:C0:8B:6E:53:92:47:57:E6:FB:10:EF:08:1B:73:E6:3E 输入正确

* 包名：com.example.lbtest

安全码：
91:16:04:30:C0:8B:6E:53:92:47:57:E6:FB:10:EF:08:1B:73:E6:3E:com.example.lbtest
91:16:04:30:C0:8B:6E:53:92:47:57:E6:FB:10:EF:08:1B:73:E6:3E:com.example.lbtest

Android SDK安全码组成：SHA1+包名。(查看详细配置方法)

新申请的Mobile与Browser类型的ak不再支持云存储接口的访问，如要使用云存储，请申请Server类型ak。

提交

图 11.8 填写完整所有创建应用的信息

接下来点击提交，应用就应该创建成功了，如图 11.9 所示。

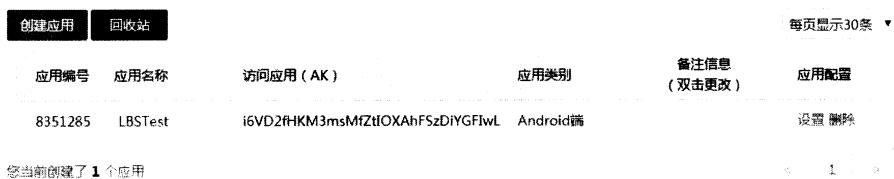


图 11.9 查看已创建的应用

其中，i6VD2fHKM3msMfZtIOXAhFSzDiYGFJwL 就是申请到的 API Key，有了它就可以进行后续的 LBS 开发工作了，那么我们马上开始吧。

11.3 使用百度定位

现在正是趁热打铁的好时机，新建一个 LBSTest 项目，包名应该就会自动被命名为 com.example.lbstest。另外需要注意，本章中所写的代码建议你都在手机上运行，虽然模拟器中也提供了模拟地理位置的功能，但在手机上可以得到真实的位置数据，你的感受会更加深刻。

11.3.1 准备 LBS SDK

在开始编码之前，我们还需要先将百度 LBS 开放平台的 SDK 准备好，下载地址是：<http://lbsyun.baidu.com/sdk/download>。本章中我们会用到基础地图和定位功能这两个 SDK，将它们勾选上，然后点击“开发包”下载按钮即可，如图 11.10 所示。



图 11.10 下载 SDK 界面

下载完成后对该压缩包解压，其中会有一个 libs 目录，这里面的内容就是我们所需要的一切了，如图 11.11 所示。

名称	类型	大小
arm64-v8a	文件夹	
armeabi	文件夹	
armeabi-v7a	文件夹	
x86	文件夹	
x86_64	文件夹	
BaiduLBS_Android.jar	Executable Jar File	1,232 KB

图 11.11 压缩包 libs 目录下的内容

libs 目录下的内容又分为两部分，BaiduLBS_Android.jar 这个文件是 Java 层要使用到的，其他子目录下的 so 文件是 Native 层要用到的。so 文件是用 C/C++语言进行编写，然后再用 NDK 编译出来的。当然这里我们并不需要去编写 C/C++的代码，因为百度都已经做好了封装，但是我们需要将 libs 目录下的每一个文件都放置到正确的位置。

首先观察一下当前的项目结构，你会发现 app 模块下面有一个 libs 目录，这里就是用来存放所有的 Jar 包的，我们将 BaiduLBS_Android.jar 复制到这里，如图 11.12 所示。

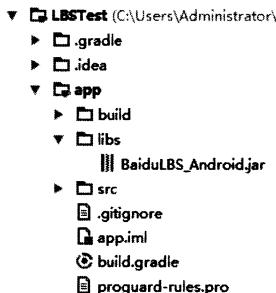


图 11.12 将 Jar 包放置到 libs 目录中

接下来展开 src/main 目录，右击该目录→New→Directory，再创建一个名为 jniLibs 的目录，这里就是专门用来存放 so 文件的，然后把压缩包里的其他所有目录直接复制到这里，如图 11.13 所示。

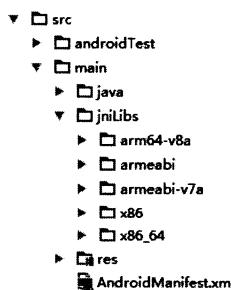


图 11.13 将 so 文件放置到 jniLibs 目录中

另外，虽然所有新创建的项目中，app/build.gradle 文件都会默认配置以下这段声明：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    ...
}
```

这表示会将 libs 目录下所有以.jar 结尾的文件添加到当前项目的引用中。但是由于我们是直接将 Jar 包复制到 libs 目录下的，并没有修改 gradle 文件，因此不会弹出我们平时熟悉的 Sync Now 提示。这个时候必须手动点击一下 Android Studio 顶部工具栏中的 Sync 按钮（图 11.14 中最左边的按钮），不然项目将无法引用到 Jar 包中提供的任何接口。

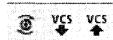


图 11.14 Android Studio 顶部工具栏

点击 Sync 按钮之后，libs 目录下的 jar 文件就会多出一个向右的箭头，这就表示项目已经能引用到这些 Jar 包了，如图 11.15 所示。



图 11.15 Jar 包引用成功

好了，这样我们就把 LBS 的 SDK 都准备好了，接下来开始编码吧。

11.3.2 确定自己位置的经纬度

首先修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TextView
        android:id="@+id/position_text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</LinearLayout>
```

布局文件中的内容非常简单，只有一个 TextView 控件，用于稍后显示当前位置的经纬度信息。

然后修改 AndroidManifest.xml 文件中的代码，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.lbstest">

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

```

<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>

<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <meta-data
        android:name="com.baidu.lbsapi.API_KEY"
        android:value="i6VD2fHKM3msMfZtIOXAhFSzDiYGFiwl" />

    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <service android:name="com.baidu.location.f" android:enabled="true"
        android:process=":remote">
    </service>
</application>

</manifest>

```

AndroidManifest.xml 文件改动比较多，我们来仔细阅读一下。可以看到，这里首先添加了很多行权限声明，每一个权限都是百度 LBS SDK 内部要用到的。然后在<application>标签的内部添加了一个<meta-data>标签，这个标签的 android:name 部分是固定的，必须填 com.baidu.lbsapi.API_KEY， android:value 部分则应该填入我们在 11.2 节申请到的 API Key。最后，还需要再注册一个 LBS SDK 中的服务，不用对这个服务的名字感到疑惑，因为百度 LBS SDK 中的代码都是混淆过的。

接下来修改 MainActivity 中的代码，如下所示：

```

public class MainActivity extends AppCompatActivity {

    public LocationClient mLocationClient;

    private TextView positionText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {

```

```
super.onCreate(savedInstanceState);
mLocationClient = new LocationClient(getApplicationContext());
mLocationClient.registerLocationListener(new MyLocationListener());
setContentView(R.layout.activity_main);
positionText = (TextView) findViewById(R.id.position_text_view);
List<String> permissionList = new ArrayList<>();
if (ContextCompat.checkSelfPermission(MainActivity.this, Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
    permissionList.add(Manifest.permission.ACCESS_FINE_LOCATION);
}
if (ContextCompat.checkSelfPermission(MainActivity.this, Manifest.permission.READ_PHONE_STATE) != PackageManager.PERMISSION_GRANTED) {
    permissionList.add(Manifest.permission.READ_PHONE_STATE);
}
if (ContextCompat.checkSelfPermission(MainActivity.this, Manifest.permission.WRITE_EXTERNAL_STORAGE) != PackageManager.PERMISSION_GRANTED) {
    permissionList.add(Manifest.permission.WRITE_EXTERNAL_STORAGE);
}
if (!permissionList.isEmpty()) {
    String [] permissions = permissionList.toArray(new String[permissionList.size()]);
    ActivityCompat.requestPermissions(MainActivity.this, permissions, 1);
} else {
    requestLocation();
}
}

private void requestLocation() {
    mLocationClient.start();
}

@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions,
    int[] grantResults) {
    switch (requestCode) {
        case 1:
            if (grantResults.length > 0) {
                for (int result : grantResults) {
                    if (result != PackageManager.PERMISSION_GRANTED) {
                        Toast.makeText(this, "必须同意所有权限才能使用本程序",
                            Toast.LENGTH_SHORT).show();
                        finish();
                        return;
                    }
                }
                requestLocation();
            } else {
                Toast.makeText(this, "发生未知错误", Toast.LENGTH_SHORT).show();
                finish();
            }
            break;
        default:
    }
}
```

```

public class MyLocationListener implements BDLocationListener {

    @Override
    public void onReceiveLocation(BDLocation location) {
        StringBuilder currentPosition = new StringBuilder();
        currentPosition.append("纬度: ").append(location.getLatitude()).
            append("\n");
        currentPosition.append("经线: ").append(location.getLongitude()).
            append("\n");
        currentPosition.append("定位方式: ");
        if (location.getLocType() == BDLocation.TypeGpsLocation) {
            currentPosition.append("GPS");
        } else if (location.getLocType() == BDLocation.TypeNetworkLocation) {
            currentPosition.append("网络");
        }
        positionText.setText(currentPosition);
    }
}

```

可以看到，在 `onCreate()` 方法中，我们首先创建了一个 `LocationClient` 的实例，`LocationClient` 的构建函数接收一个 `Context` 参数，这里调用 `getApplicationContext()` 方法来获取一个全局的 `Context` 参数并传入。然后调用 `LocationClient` 的 `registerLocationListener()` 方法来注册一个定位监听器，当获取到位置信息的时候，就会回调这个定位监听器。

接下来看一下这里运行时权限的用法，由于我们在 `AndroidManifest.xml` 中声明了很多权限，参考一下 7.2.1 小节中的危险权限表格可以发现，其中 `ACCESS_COARSE_LOCATION`、`ACCESS_FINE_LOCATION`、`READ_PHONE_STATE`、`WRITE_EXTERNAL_STORAGE` 这 4 个权限是需要进行运行时权限处理的，不过由于 `ACCESS_COARSE_LOCATION` 和 `ACCESS_FINE_LOCATION` 都属于同一个权限组，因此两者只要申请其一就可以了。那么怎样才能在运行时一次性申请 3 个权限呢？这里我们使用了一种新的用法，首先创建一个空的 `List` 集合，然后依次判断这 3 个权限有没有被授权，如果没被授权就添加到 `List` 集合中，最后将 `List` 转换成数组，再调用 `ActivityCompat.requestPermissions()` 方法一次性申请。

除此之外，`onRequestPermissionsResult()` 方法中对权限申请结果的逻辑处理也和之前有所不同，这次我们通过一个循环将申请的每个权限都进行了判断，如果有任何一个权限被拒绝，那么就直接调用 `finish()` 方法关闭当前程序，只有当所有权限都被用户同意了，才会调用 `requestLocation()` 方法开始地理位置定位。

`requestLocation()` 方法中的代码比较简单，只是调用了一下 `LocationClient` 的 `start()` 方法就能开始定位了。定位的结果会回调到我们前面注册的监听器当中，也就是 `MyLocationListener`。观察一下 `MyLocationListener` 的 `onReceiveLocation()` 方法中，在这里我们通过 `BDLocation` 的 `getLatitude()` 方法获取当前位置的纬度，通过 `getLongitude()` 方法获

取当前位置的经度，通过 `getLocType()` 方法获取当前的定位方式，最终将结果组装成一个字符串，显示到 `TextView` 上面。

现在我们可以来运行一下程序了，如图 11.16 所示。毫无疑问，打开程序首先就会弹出运行时权限的申请对话框，注意看对话框的底部，提示我们一共有 3 项权限申请，当前是第 1 项，授权了第 1 项后就会显示第 2 项，这里我们全部点击允许，然后就会立刻开始定位了，结果如图 11.17 所示。



图 11.16 运行时权限申请对话框



图 11.17 地理位置定位的结果

可以看到，设备当前的经纬度信息已经成功定位出来了。

不过，在默认情况下，调用 `LocationClient` 的 `start()` 方法只会定位一次，如果我们正在快速移动中，怎样才能实时更新当前的位置呢？为此，百度 LBS SDK 提供了一系列的设置方法，来允许我们更改默认的行为，修改 `MainActivity` 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    ...

    private void requestLocation() {
        initLocation();
        mLocationClient.start();
    }

    private void initLocation(){
        LocationClientOption option = new LocationClientOption();
        option.setScanSpan(5000);
        mLocationClient.setLocOption(option);
    }

    @Override
```

```

protected void onDestroy() {
    super.onDestroy();
    mLocationClient.stop();
}

...
}

```

这里增加了一个 `initLocation()` 方法，在 `initLocation()` 方法中我们创建了一个 `LocationClientOption` 对象，然后调用它的 `setScanSpan()` 方法来设置更新的间隔。这里传入了 5000，表示每 5 秒钟会更新一下当前的位置。

最后要记得，在活动被销毁的时候一定要调用 `LocationClient` 的 `stop()` 方法来停止定位，不然程序会持续在后台不停地进行定位，从而严重消耗手机的电量。

现在重新运行一下程序，然后拿着手机随处移动，你会发现界面上的经纬度信息也会跟着一起变化的。

11.3.3 选择定位模式

还记得在本章刚开始的时候说过，Android 中主要有两种定位方式吗？一种是通过 GPS 定位，一种是通过网络定位。而从上一小节中的例子中应该可以看出，我们一直是使用的网络定位。那么如何才能切换到精确度更高的 GPS 定位呢？本小节我们就来学习一下。

首先，GPS 定位功能必须要由用户主动去启用才行，不然任何应用程序都无法使用 GPS 获取到手机当前的位置信息。进入手机的设置→位置信息，如图 11.18 所示。



图 11.18 位置信息设置界面

我们可以通过顶部的开关来控制定位功能是开启还是关闭，另外，点击“模式”可以选择具体的定位模式，如图 11.19 所示。

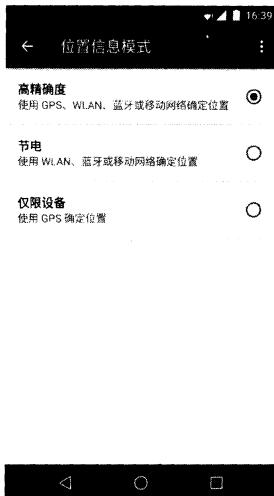


图 11.19 选择具体的定位模式

其中，高精确度模式表示允许使用 GPS、无线网络、蓝牙或移动网络来进行定位，节电模式表示仅允许使用无线网络、蓝牙或移动网络来进行定位，而仅限设备模式表示仅允许使用 GPS 来进行定位。也就是说，如果我们想要使用 GPS 定位功能，这里必须要选择高精确度模式，或者仅限设备模式。

当然，你并不需要担心一旦启用 GPS 定位功能后，手机的电量就会直线下滑，这只是表明你已经同意让应用程序来对你的手机进行 GPS 定位了，但只有当定位操作真正开始的时候，才会影响到手机的电量。

开启了 GPS 定位功能之后，再回来看一下代码。我们可以在 `initLocation()` 方法中对百度 LBS SDK 的定位模式进行指定，一共有 3 种模式可选：`Hight_Accuracy`、`Battery_Saving` 和 `Device_Sensors`。`Hight_Accuracy` 表示高精确度模式，会在 GPS 信号正常的情况下优先使用 GPS 定位，在无法接收 GPS 信号的时候使用网络定位。`Battery_Saving` 表示节电模式，只会使用网络进行定位。`Device_Sensors` 表示传感器模式，只会使用 GPS 进行定位。其中，`Hight_Accuracy` 是默认的模式，也就是说，我们即使不修改任何代码，只要拿着手机走到室外去，让手机可以接收到 GPS 信号，就会自动切换到 GPS 定位模式了。

当然我们也可以强制指定只使用 GPS 进行定位，修改 `MainActivity` 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {
```

```
...
```

```
private void initLocation(){
    LocationClientOption option = new LocationClientOption();
    option.setLocationMode(LocationClientOption.LocationMode.Device_Sensors);
    mLocationClient.setLocOption(option);
}

...
}

}
```

这里调用了 `setLocationMode()` 方法来将定位模式指定成传感器模式，也就是说只能使用 GPS 进行定位。重新运行一下程序，然后拿着你的手机走到室外去，结果如图 11.20 所示。



图 11.20 GPS 定位的结果

11.3.4 看得懂的位置信息

话说回来，刚才我们虽然成功获取到了设备当前位置的经纬度信息，但遗憾的是，这种经纬度的值一般人是根本看不懂的，相信谁也无法立刻答出南纬 25 度、东经 148 度是什么地方吧？为了能够更加直观地阅读，我们还需要学习一下如何获取看得懂的位置信息。

幸运的是，百度 LBS SDK 在这方面提供了非常好的支持，我们只需要进行一些简单的接口调用就能得到当前位置各种丰富的地址信息，下面就来一起看一下吧。

修改 `MainActivity` 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    ...
    private void initLocation(){
        LocationClientOption option = new LocationClientOption();
    }
}
```

```

        option.setScanSpan(5000);
        option.setIsNeedAddress(true);
        mLocationClient.setLocOption(option);
    }

    ...

    public class MyLocationListener implements BDLocationListener {

        @Override
        public void onReceiveLocation(BDLocation location) {
            StringBuilder currentPosition = new StringBuilder();
            currentPosition.append("纬度: ").append(location.getLatitude()).
                append("\n");
            currentPosition.append("经线: ").append(location.getLongitude()).
                append("\n");
            currentPosition.append("国家: ").append(location.getCountry()).
                append("\n");
            currentPosition.append("省: ").append(location.getProvince()).
                append("\n");
            currentPosition.append("市: ").append(location.getCity()).append("\n");
            currentPosition.append("区: ").append(location.getDistrict()).
                append("\n");
            currentPosition.append("街道: ").append(location.getStreet()).
                append("\n");
            currentPosition.append("定位方式: ");
            if (location.getLocType() == BDLocation.TypeGpsLocation) {
                currentPosition.append("GPS");
            } else if (location.getLocType() == BDLocation.TypeNetworkLocation) {
                currentPosition.append("网络");
            }
            positionText.setText(currentPosition);
        }
    }
}

```

首先在 `initLocation()` 方法中，我们调用了 `LocationClientOption` 的 `setIsNeedAddress()` 方法，并传入 `true`，这就表示我们需要获取当前位置详细的地址信息。

接下来在 `MyLocationListener` 的 `onReceiveLocation()` 方法就可以获取到各种丰富的地址信息了，调用 `getCountry()` 方法可以得到当前所在国家，调用 `getProvince()` 方法可以得到当前所在省份，以此类推。另外还有一点需要注意，由于获取地址信息一定需要用到网络，因此即使我们将定位模式指定成了 `Device_Sensors`，也会自动开启网络定位功能。

现在重新运行一下程序，结果如图 11.21 所示。

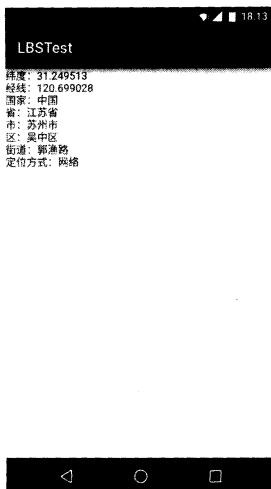


图 11.21 获取到当前位置的地址信息

可以看到，手机当前位置的地址信息已经成功显示出来了。如果你带着手机移动了较远的距离，界面上显示的位置也会跟着一起变化的。

11.4 使用百度地图

现在手机地图的应用真的可以算得上是非常广泛了，和 PC 上的地图相比，手机地图能够随时随地进行查看，并且轻松构建出行路线，使用起来明显更加地方便。但是你有没有想过，其实我们在自己的应用程序里也是可以加入地图功能的，比如优步中使用的就是百度地图。本节我们就来学习一下这方面的知识。

11.4.1 让地图显示出来

由于在上一节中我们已经将 LBS SDK 全部准备好了，其中就包括了地图功能，因此这里就不用再去下载百度地图的 SDK 了。

那么我们直接在 LBSTest 项目的基础上进行开发，修改 `activity_main.xml` 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TextView
        android:id="@+id/position_text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```

        android:visibility="gone" />

<com.baidu.mapapi.map.MapView
    android:id="@+id/bmapView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:clickable="true" />

</LinearLayout>

```

这里在布局文件中新放置了一个MapView控件，并让它充满整个屏幕。这个MapView是由百度提供的自定义控件，所以在使用它的时候需要将完整的包名加上。另外，之前用于显示定位信息的TextView现在暂时用不到了，我们将它的visibility属性指定成gone，让它在界面上隐藏起来。

接下来修改 MainActivity 中的代码，如下所示：

```

public class MainActivity extends AppCompatActivity {

    ...

    private MapView mapView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mLocationClient = new LocationClient(getApplicationContext());
        mLocationClient.registerLocationListener(new MyLocationListener());
        SDKInitializer.initialize(getApplicationContext());
        setContentView(R.layout.activity_main);
        mapView = (MapView) findViewById(R.id.bmapView);
        ...
    }

    ...

    @Override
    protected void onResume() {
        super.onResume();
        mapView.onResume();
    }

    @Override
    protected void onPause() {
        super.onPause();
        mapView.onPause();
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        mLocationClient.stop();
    }
}

```

```
    mapView.onDestroy();  
}  
  
...  
  
}
```

可以看到，这里的代码也非常简单。首先需要调用 SDKInitializer 的 `initialize()` 方法来进行初始化操作，`initialize()` 方法接收一个 `Context` 参数，这里我们调用 `getApplicationContext()` 方法来获取一个全局的 `Context` 参数并传入。注意初始化操作一定要在 `setContentView()` 方法前调用，不然的话就会出错。接下来我们调用 `findViewById()` 方法获取到了 `MapView` 的实例，这个实例在后面的功能当中还会用到。

另外还需要重写 `onResume()`、`onPause()` 和 `onDestroy()` 这 3 个方法，在这里对 `MapView` 进行管理，以保证资源能够及时地得到释放。

好了，就是这么简单。现在重新运行一下程序，百度地图就应该成功显示出来了，如图 11.22 所示。



图 11.22 让百度地图显示出来

11.4.2 移动到我的位置

地图是成功显示出来了，但也许这并不是你想要的。因为这是一张默认的地图，显示的是北京市中心的位置，而你可能希望看到更加精细的地图信息，比如说自己所在位置的周边环境。显然，通过缩放和移动的方式来慢慢找到自己的位置是一种很愚蠢的做法。那么本小节我们就来学习一下，如何才能在地图中快速移动到自己的位置。

百度 LBS SDK 的 API 中提供了一个 `BaiduMap` 类，它是地图的总控制器，调用 `MapView` 的 `getMap()` 方法就能获取到 `BaiduMap` 的实例，如下所示：

```
BaiduMap baiduMap = mapView.getMap();
```

有了 `BaiduMap` 后，我们就能对地图进行各种各样的操作了，比如设置地图的缩放级别以及将地图移动到某一个经纬度上。

百度地图将缩放级别的取值范围限定在 3 到 19 之间，其中小数点位的值也是可以取的，值越大，地图显示的信息就越精细。比如我们想要将缩放级别设置成 12.5，就可以这样写：

```
MapStatusUpdate update = MapStatusUpdateFactory.zoomTo(12.5f);
baiduMap.animateMapStatus(update);
```

其中 `MapStatusUpdateFactory` 的 `zoomTo()` 方法接收一个 `float` 型的参数，就是用于设置缩放级别的，这里我们传入 12.5f。`zoomTo()` 方法返回一个 `MapStatusUpdate` 对象，我们把这个对象传入 `BaiduMap` 的 `animateMapStatus()` 方法当中即可完成缩放功能。

那么怎样才能让地图移动到某一个经纬度上呢？这就需要借助 `LatLng` 类了。其实 `LatLng` 并没有什么太多的用法，主要就是用于存放经纬度值的，它的构造方法接收两个参数，第一个参数是纬度值，第二个参数是经度值。之后调用 `MapStatusUpdateFactory` 的 `newLatLng()` 方法将 `LatLng` 对象传入，`newLatLng()` 方法返回的也是一个 `MapStatusUpdate` 对象，我们再把这个对象传入 `BaiduMap` 的 `animateMapStatus()` 方法当中，就可以将地图移动到指定的经纬度上了，写法如下：

```
LatLng ll = new LatLng(39.915, 116.404);
MapStatusUpdate update = MapStatusUpdateFactory.newLatLng(ll);
baiduMap.animateMapStatus(update);
```

上述代码就实现了将地图移动到北纬 39.915 度、东经 116.404 度这个位置的功能。

了解了这些知识之后，接下来再去实现将地图快速移动到自己位置的功能就变得非常简单了。首先我们可以利用在 11.3 节中所学的定位技术来获得自己当前位置的经纬度，之后再按照上述的方法来将地图移动到指定的位置就可以了。

那么下面我们就来继续完善 `LBSTest` 这个项目，加入“移动到我的位置”这个功能。修改 `MainActivity` 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    ...
    private BaiduMap baiduMap;
    private boolean isFirstLocate = true;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```

super.onCreate(savedInstanceState);
mLocationClient = new LocationClient(getApplicationContext());
mLocationClient.registerLocationListener(new MyLocationListener());
SDKInitializer.initialize(getApplicationContext());
setContentView(R.layout.activity_main);
mapView = (MapView) findViewById(R.id.bmapView);
baiduMap = mapView.getMap();
...
}

private void navigateTo(BDLocation location) {
    if (isFirstLocate) {
        LatLng ll = new LatLng(location.getLatitude(), location.getLongitude());
        MapStatusUpdate update = MapStatusUpdateFactory.newLatLng(ll);
        baiduMap.animateMapStatus(update);
        update = MapStatusUpdateFactory.zoomTo(16f);
        baiduMap.animateMapStatus(update);
        isFirstLocate = false;
    }
}

public class MyLocationListener implements BDLocationListener {

    @Override
    public void onReceiveLocation(BDLocation location) {
        if (location.getLocType() == BDLocation.TypeGpsLocation
            || location.getLocType() == BDLocation.TypeNetworkLocation) {
            navigateTo(location);
        }
    }
}
}

```

这里并没有新增多少代码，主要是加入了一个 `navigateTo()` 方法。这个方法中的代码也很好理解，先是将 `BDLocation` 对象中的地理位置信息取出并封装到 `LatLng` 对象中，然后调用 `MapStatusUpdateFactory` 的 `newLatLng()` 方法并将 `LatLng` 对象传入，接着将返回的 `MapStatusUpdate` 对象作为参数传入到 `BaiduMap` 的 `animateMapStatus()` 方法当中，和上面介绍的用法是一模一样的。并且这里为了让地图信息可以显示得更加丰富一些，我们将缩放级别设置成了 16。另外还有一点需要注意，上述代码当中我们使用了一个 `isFirstLocate` 变量，这个变量的作用是为了防止多次调用 `animateMapStatus()` 方法，因为将地图移动到我们当前的位置只需要在程序第一次定位的时候调用一次就可以了。

写好了 `navigateTo()` 方法之后，剩下的事情就简单了，当定位到设备当前位置的时候，我们在 `onReceiveLocation()` 方法中直接把 `BDLocation` 对象传给 `navigateTo()` 方法，这样就能够让地图移动到设备所在的位置了。

现在重新运行一下程序，结果如图 11.23 所示。

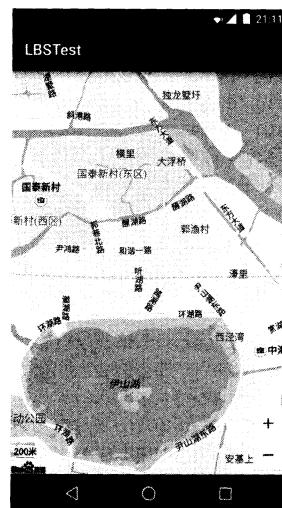


图 11.23 将地图移动到设备所在的位置

11.4.3 让“我”显示在地图上

现在我们已经可以让地图显示我们周边的环境了，但是相信在你平时使用手机地图时应该会注意到，通常情况下手机地图上应该都会有一个小光标，用于显示设备当前所在的位置，并且如果设备正在移动的话，那么这个光标也会跟着一起移动。那么我们现在就继续对现有代码进行扩展，让“我”能够显示在地图上。

百度 LBS SDK 当中提供了一个 `MyLocationData.Builder` 类，这个类是用来封装设备当前所在位置的，我们只需将经纬度信息传入到这个类的相应方法当中就可以了，如下所示：

```
MyLocationData.Builder locationBuilder = new MyLocationData.Builder();
locationBuilder.latitude(39.915);
locationBuilder.longitude(116.404);
```

`MyLocationData.Builder` 类还提供了一个 `build()` 方法，当我们把要封装的信息都设置完成之后，只需要调用它的 `build()` 方法，就会生成一个 `MyLocationData` 的实例，然后再将这个实例传入到 `BaiduMap` 的 `setMyLocationData()` 方法当中，就可以让设备当前的位置显示在地图上了，写法如下：

```
MyLocationData locationData = locationBuilder.build();
baiduMap.setMyLocationData(locationData);
```

大体思路就是这个样子，下面我们开始来实现一下，修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {
```

1

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mLocationClient = new LocationClient(getApplicationContext());
    mLocationClient.registerLocationListener(new MyLocationListener());
    SDKInitializer.initialize(getApplicationContext());
    setContentView(R.layout.activity_main);
    mapView = (MapView) findViewById(R.id.bmapView);
    baiduMap = mapView.getMap();
    baiduMap.setMyLocationEnabled(true);
    ...
}

private void navigateTo(BDLocation location) {
    if (isFirstLocate) {
        LatLng ll = new LatLng(location.getLatitude(), location.getLongitude());
        MapStatusUpdate update = MapStatusUpdateFactory.newLatLng(ll);
        baiduMap.animateMapStatus(update);
        update = MapStatusUpdateFactory.zoomTo(16f);
        baiduMap.animateMapStatus(update);
        isFirstLocate = false;
    }
    MyLocationData.Builder locationBuilder = new MyLocationData.Builder();
    locationBuilder.latitude(location.getLatitude());
    locationBuilder.longitude(location.getLongitude());
    MyLocationData locationData = locationBuilder.build();
    baiduMap.setMyLocationData(locationData);
}

@Override
protected void onDestroy() {
    super.onDestroy();
    mLocationClient.stop();
    mapView.onDestroy();
    baiduMap.setMyLocationEnabled(false);
}
}

```

可以看到，在 `navigateTo()` 方法中，我们添加了 `MyLocationData` 的构建逻辑，将 `Location` 中包含的经度和纬度分别封装到了 `MyLocationData.Builder` 当中，最后把 `MyLocationData` 设置到了 `BaiduMap` 的 `setMyLocationData()` 方法当中。注意这段逻辑必须写在 `isFirstLocate` 这个 `if` 条件语句的外面，因为让地图移动到我们当前的位置只需要在第一次定位的时候执行，但是设备在地图上显示的位置却应该是随着设备的移动而实时改变的。

另外，根据百度地图的限制，如果我们想要使用这一功能，一定要事先调用 `BaiduMap` 的 `setMyLocationEnabled()` 方法将此功能开启，否则设备的位置将无法在地图上显示。而在程序退出的时候，也要记得将此功能给关闭掉。

就是这么简单，现在重新运行一下程序，结果如图 11.24 所示。

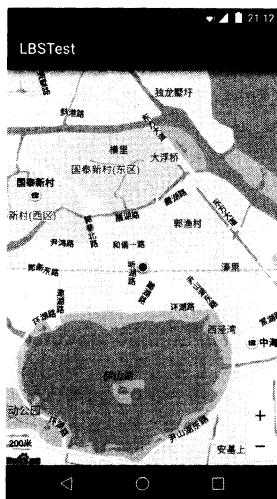


图 11.24 让“我”显示在地图上

这样的话，用户就可以非常清晰地看出自己当前是在哪里了。

关于百度 LBS SDK 的用法我就准备介绍这么多，现在你已经算是成功入门了。如果想要更加深入地研究百度 LBS 的各种用法，可以到官方网站上面参考开发指南，地址是：<http://lbsyun.baidu.com>。另外，百度 LBS SDK 的版本未来随时都有可能更新，也许更新之后会导致书上的例子无法正常运行，因此除了照着图书学习之外，根据官网的开发指南来进行学习也是非常重要的，因为官方文档永远都是最新的。

好了，本章的主体内容到这里就结束了。下面我们将再次进入本书的特殊环节，学习一下关于 Git 的高级用法。

11.5 Git 时间——版本控制工具的高级用法

现在的你对于 Git 应该完全不会感到陌生了吧，通过了之前两节内容的学习，你已经掌握了很多 Git 中常用的命令，像提交代码这种简单的操作相信肯定是最不倒你的。

那么打开 Git Bash，并进入到 LBSTest 这个项目的根目录，然后执行提交操作：

```
git init
git add .
git commit -m "First Commit."
```

这样就将准备工作完成了，下面就让我们开始学习关于 Git 的高级用法。

11.5.1 分支的用法

分支是版本控制工具中比较高级且比较重要的一个概念，它主要的作用就是在现有代码的基础上开辟一个分叉口，使得代码可以在主干线上同时进行开发，且相互之间不会影响。分支的工作原理示意图如图 11.25 所示。

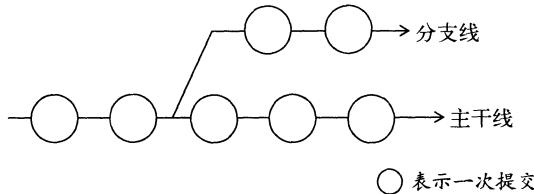


图 11.25 分支的工作原理示意图

你也许会有疑惑，为什么需要建立分支呢？只在主干线上进行开发不是挺好的吗？没错，通常情况下，只在主干线上进行开发是完全没有问题的，不过一旦涉及出版本的情况，如果不建立分支的话，你就会非常地头疼。举个简单的例子吧，比如说你们公司研发了一款不错的软件，最近刚刚完成，并推出了 1.0 版本。但是领导是不会让你们闲着的，马上提出了新的需求，让你们投入到了 1.1 版本的开发工作当中。过了几个星期，1.1 版本的功能已完成了一半，但是这个时候有用户反馈，之前上线的 1.0 版本发现了几个重大的 bug，严重影响软件的正常使用。领导也相当重视这个问题，要求你们立刻修复这些 bug，并重新发布 1.0 版本，但这个时候你就非常为难了，你会发现根本没法去修复这些 bug。因为现在 1.1 版本已开发一半了，如果在现有代码的基础上修复这些 bug，那么更新的 1.0 版本将会带有一半 1.1 版本的功能！

进退两难了是不是？但是如果你使用了分支的话，就完全不会存在这个让人头疼的问题。你只需要在发布 1.0 版本的时候建立一个分支，然后在主干线上继续开发 1.1 版本的功能。当 1.0 版本上发现任何 bug 的时候，就在分支线上进行修改，然后发布新的 1.0 版本，并记得将修改后的代码合并到主干线上。这样的话，不仅可以轻松解决掉 1.0 版本存在的 bug，而且保证了主干线上的代码也已经修复了这些 bug，当 1.1 版本发布时就不会有同样的 bug 存在了。

说了这么多，相信你也已经意识到分支的重要性了，那么我们马上来学习一下如何在 Git 中操作分支吧。

分支的英文名是 `branch`，如果想要查看当前的版本库当中有哪些分支，可以使用 `git branch` 这个命令，结果如图 11.26 所示。



图 11.26 查看所有分支

由于目前 LBSTest 项目中还没有创建过任何分支，因此只有一个 `master` 分支存在，这也就是前面所说的主干线。接下来我们尝试去创建一个分支，命令如下：

```
git branch version1.0
```

这样就创建了一个名为 version1.0 的分支，我们再次输入 `git branch` 这个命令来检查一下，结果如图 11.27 所示。



图 11.27 再次查看所有分支

可以看到，果然有一个叫作 version1.0 的分支出现了。你会发现，`master` 分支的前面有一个“*”号，说明目前我们的代码还是在 `master` 分支上的，那么怎样才能切换到 `version1.0` 这个分支上呢？其实也很简单，只需要使用 `checkout` 命令即可，如下所示：

```
git checkout version1.0
```

再次输入 `git branch` 来进行检查，结果如图 11.28 所示。



图 11.28 查看切换分支后的结果

可以看到，我们已经把代码成功切换到 `version1.0` 这个分支上了。

需要注意的是，在 `version1.0` 分支上修改并提交的代码将不会影响到 `master` 分支。同样的道理，在 `master` 分支上修改并提交的代码也不会影响到 `version1.0` 分支。因此，如果我们在 `version1.0` 分支上修复了一个 bug，在 `master` 分支上这个 bug 仍然是存在的。这时将修改的代码一行行复制到 `master` 分支上显然不是一种聪明的做法，最好的办法就是使用 `merge` 命令来完成合并操作，如下所示：

```
git checkout master
git merge version1.0
```

仅仅这样简单的两行命令，就可以把在 `version1.0` 分支上修改并提交的内容合并到 `master` 分支上了。当然，在合并分支的时候还有可能出现代码冲突的情况，这个时候你就需要静下心来慢慢地找出并解决这些冲突，Git 在这里就无法帮助你了。

最后，当我们不再需要 `version1.0` 这个分支的时候，可以使用如下命令将这个分支删除掉：

```
git branch -D version1.0
```

11.5.2 与远程版本库协作

可以这样说，如果你是一个人在开发，那么使用版本控制工具就远远无法发挥出它真正强大的功能。没错，所有版本控制工具最重要的一个特点就是可以使用它来进行团队合作开发。每个人的电脑上都会有一份代码，当团队的某个成员在自己的电脑上编写完成了某个功能后，就将代

码提交到服务器，其他的成员只需要将服务器上的代码同步到本地，就能保证整个团队所有人的代码都相同。这样的话，每个团队成员就可以各司其职，大家共同来完成一个较为庞大的项目。

那么如何使用 Git 来进行团队合作开发呢？这就需要有一个远程的版本库，团队的每个成员都从这个版本库中获取到最原始的代码，然后各自进行开发，并且以后每次提交的代码都同步到远程版本库上就可以了。另外，团队中的每个成员最好都要养成经常从版本库中获取最新代码的习惯，不然的话，大家的代码就很有可能经常出现冲突。

比如说现在有一个远程版本库的 Git 地址是 <https://github.com/example/test.git>，就可以使用如下的命令将代码下载到本地：

```
git clone https://github.com/example/test.git
```

之后你在这份代码的基础上进行了一些修改和提交，那么怎样才能把本地修改的内容同步到远程版本库上呢？这就需要借助 `push` 命令来完成了，用法如下所示：

```
git push origin master
```

其中 `origin` 部分指定的是远程版本库的 Git 地址，`master` 部分指定的是同步到哪一个分支上，上述命令就完成了将本地代码同步到 <https://github.com/example/test.git> 这个版本库的 `master` 分支上的功能。

知道了将本地的修改同步到远程版本库上的方法，接下来我们看一下如何将远程版本库上的修改同步到本地。Git 提供了两种命令来完成此功能，分别是 `fetch` 和 `pull`，`fetch` 的语法规则和 `push` 是差不多的，如下所示：

```
git fetch origin master
```

执行这个命令后，就会将远程版本库上的代码同步到本地，不过同步下来的代码并不会合并到任何分支上去，而是会存放到一个 `origin/master` 分支上，这时我们可以通过 `diff` 命令来查看远程版本库上到底修改了哪些东西：

```
git diff origin/master
```

之后再调用 `merge` 命令将 `origin/master` 分支上的修改合并到主分支上即可，如下所示：

```
git merge origin/master
```

而 `pull` 命令则是相当于将 `fetch` 和 `merge` 这两个命令放在一起执行了，它可以从远程版本库上获取最新的代码并且合并到本地，用法如下所示：

```
git pull origin master
```

也许你现在对远程版本库的使用还是感觉比较抽象，没关系，因为暂时我们只是了解了一下命令的用法，还没进行实践，在第 14 章当中，你将会对远程版本库的用法有更深一层的认识。

11.6 小结与点评

不得不说，本章中学到的知识应该还算是蛮有趣的吧？在这次的 Android 特色开发环节中，我们主要学习了基于位置服务的工作原理和用法，借助百度提供的 LBS SDK，我们可以随时确定自己当前位置的经纬度，并且还能获取到具体的省、市、区、街道等地址。之后又学习了百度地图的用法，不仅成功地将地图信息显示了出来，还综合利用了前面所学到的定位技术实现了一个较为完整的例子。

除了基于位置的服务之外，本章 Git 时间中继续对 Git 的用法进行了更深一步的探究，使得我们对分支和远程版本库的使用都有了一定层次的了解。

那么关于 Android 特色开发的内容就讲到这里，下一章中我们将会学习 Android 5.0 系统中新增的一套全新的知识点——Material Design。

第 12 章

最佳的 UI 体验——Material Design 实战

其实长久以来，大多数人都认为 Android 系统的 UI 并不算美观，至少没有 iOS 系统的美观。以至于很多 IT 公司在进行应用界面设计的时候，为了保证双平台的统一性，强制要求 Android 端的界面风格必须和 iOS 端一致。这种情况在现实工作当中实在是太常见了，虽然我认为这是非常不合理的。因为对于一般用户来说，他们不太可能会在两个操作系统上分别去使用同一个应用，但是却必定会在同一个操作系统上使用不同的应用。因此，同一个操作系统中各个应用之间的界面统一性要远比一个应用在双平台的界面统一性重要得多，只有这样，才能给使用者带来更好的用户体验。

但问题在于，Android 标准的界面设计风格并不是特别被大众所接受，很多公司都觉得自己完全可以设计出更加好看的局面，从而导致 Android 平台的界面风格长期难以得到统一。为了解决这个问题，谷歌也是祭出了杀手锏，在 2014 年 Google I/O 大会上重磅推出了一套全新的界面设计语言——Material Design。

本章我们就将对 Material Design 进行一次深入的学习。

12.1 什么是 Material Design

Material Design 是由谷歌的设计工程师们基于传统优秀的设计原则，结合丰富的创意和科学技术所发明的一套全新的界面设计语言，包含了视觉、运动、互动效果等特性。那么谷歌凭什么认为 Material Design 就能解决 Android 平台界面风格不统一的问题呢？一言以蔽之，好看！

没错，这次谷歌在界面设计上确实是下足了功夫，很多媒体评论，Material Design 的出现使得 Android 首次在 UI 方面超越了 iOS。按照正常的思维来想，如果各个公司都无法设计出比 Material Design 更出色的界面风格，那么它们就应该理所当然地使用 Material Design 来设计界面，从而也就能解决 Android 平台界面风格不统一的问题了。

为了做出表率，谷歌从 Android 5.0 系统开始，就将所有内置的应用都使用 Material Design 风格来进行设计。这里我随便截了两张图，你可以先欣赏一下，如图 12.1 所示。

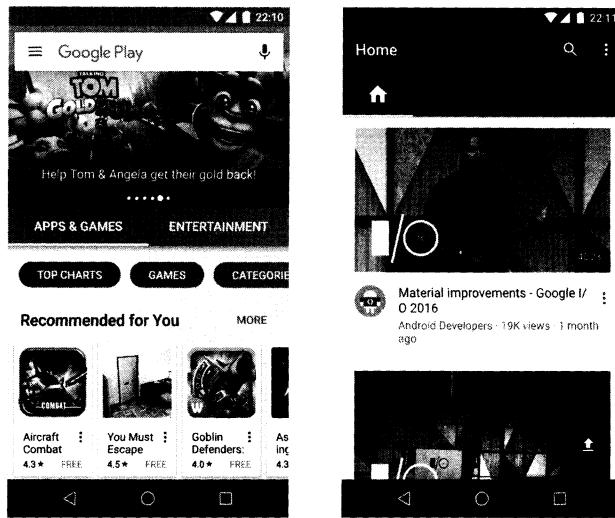


图 12.1 使用 Material Design 设计的应用

其中，左边的应用是 Play Store，右边的应用是 YouTube。可以看出，它们的界面都十分美观，而它们正是使用 Material Design 来进行设计的。

不过，在重磅推出之后，Material Design 的普及程度却不能说是特别理想。因为这只是一个推荐的设计规范，主要是面向 UI 设计人员的，而不是面向开发者的。很多开发者可能根本就搞不清楚什么样的界面和效果才叫 Material Design，就算搞清楚了，实现起来也会很费劲，因为不少 Material Design 的效果是很难实现的，而 Android 中却几乎没有提供相应的 API 支持，一切都需要靠开发者自己从零写起。

谷歌当然也意识到了这个问题，于是在 2015 年的 Google I/O 大会上推出了一个 Design Support 库，这个库将 Material Design 中最具代表性的一些控件和效果进行了封装，使得开发者在即使不了解 Material Design 的情况下也能非常轻松地将自己的应用 Material 化。本章中我们就将对 Design Support 这个库进行深入的学习，并且配合一些其他的控件来完成一个优秀的 Material Design 应用。

新建一个 MaterialTest 项目，然后我们马上开始吧！

12.2 Toolbar

Toolbar 将会是我们接触的第一个 Material 控件。虽说对于 Toolbar 你暂时应该还是比较陌生的，但是对于它的另一个相关控件 ActionBar，你就应该有点熟悉了。

回忆一下，我们曾经在 3.4.1 小节为了使用一个自定义的标题栏，而把系统原生的 ActionBar 隐藏掉。没错，每个活动最顶部的那个标题栏其实就是 ActionBar，之前我们编写的所有程序里一直都有 ActionBar 的身影。

不过 ActionBar 由于其设计的原因，被限定只能位于活动的顶部，从而不能实现一些 Material Design 的效果，因此官方现在已经不再建议使用 ActionBar 了。那么本书中我也就不准备再介绍 ActionBar 的用法了，而是直接讲解现在更加推荐使用的 Toolbar。

Toolbar 的强大之处在于，它不仅继承了 ActionBar 的所有功能，而且灵活性很高，可以配合其他控件来完成一些 Material Design 的效果，下面我们就来具体学习一下。

首先你要知道，任何一个新建的项目，默认都是会显示 ActionBar 的，这个想必你已经见识过太多次了。那么这个 ActionBar 到底是从哪里来的呢？其实这是根据项目中指定的主题来显示的，打开 AndroidManifest.xml 文件看一下，如下所示：

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    ...
</application>
```

可以看到，这里使用 `android:theme` 属性指定了一个 AppTheme 的主题。那么这个 AppTheme 又是在哪里定义的呢？打开 res/values/styles.xml 文件，代码如下所示：

```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>
</resources>
```

这里定义了一个叫 AppTheme 的主题，然后指定它的 parent 主题是 Theme.AppCompat.Light.DarkActionBar。这个 DarkActionBar 是一个深色的 ActionBar 主题，我们之前的项目中自带的 ActionBar 就是因为指定了这个主题才出现的。

而现在我们准备使用 Toolbar 来替代 ActionBar，因此需要指定一个不带 ActionBar 的主题，通常有 Theme.AppCompat.NoActionBar 和 Theme.AppCompat.Light.NoActionBar 这两种主题可选。其中 Theme.AppCompat.NoActionBar 表示深色主题，它会将界面的主体颜色设成深色，陪衬颜色设成淡色。而 Theme.AppCompat.Light.NoActionBar 表示淡色主题，它会将界面的主体颜色设成

淡色，陪衬颜色设成深色。具体的效果你可以自己动手试一试，这里由于我们之前的程序一直都是以淡色为主的，那么我就选用淡色主题了，如下所示：

```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

</resources>
```

然后观察一下 AppTheme 中的属性重写，这里重写了 colorPrimary、colorPrimaryDark 和 colorAccent 这 3 个属性的颜色。那么这 3 个属性分别代表着什么位置的颜色呢？我用语言比较难描述清楚，还是通过一张图来理解一下吧，如图 12.2 所示。

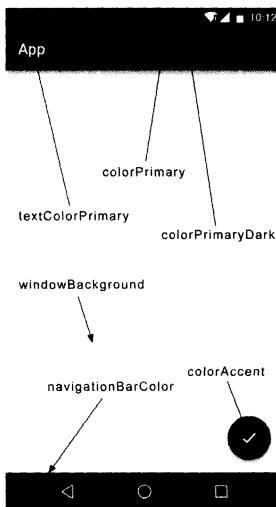


图 12.2 各属性指定颜色的位置

可以看到，每个属性所指定颜色的位置直接一目了然了。

除了上述 3 个属性之外，我们还可以通过 textColorPrimary、windowBackground 和 navigationBarColor 等属性来控制更多位置的颜色。不过唯独 colorAccent 这个属性比较难理解，它不只是用来指定这样一个按钮的颜色，而是更多表达了一个强调的意思，比如一些控件的选中状态也会使用 colorAccent 的颜色。

现在我们已经将 ActionBar 隐藏起来了，那么接下来看一看如何使用 Toolbar 来替代 ActionBar。修改 activity_main.xml 中的代码，如下所示：

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:background="?attr/colorPrimary"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
        app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />

</FrameLayout>
```

虽然这段代码不长，但是里面着实有不少技术点是需要我们去仔细琢磨一下的。首先看一下第 2 行，这里使用 `xmlns:app` 指定了一个新的命名空间。思考一下，正是由于每个布局文件都会使用 `xmlns:android` 来指定一个命名空间，因此我们才能一直使用 `android:id`、`android:layout_width` 等写法，那么这里指定了 `xmlns:app`，也就是说现在可以使用 `app:attribute` 这样的写法了。但是为什么这里要指定一个 `xmlns:app` 的命名空间呢？这是由于 Material Design 是在 Android 5.0 系统中才出现的，而很多的 Material 属性在 5.0 之前的系统中并不存在，那么为了能够兼容之前的老系统，我们就不能使用 `android:attribute` 这样的写法了，而是应该使用 `app:attribute`。

接下来定义了一个 Toolbar 控件，这个控件是由 `appcompat-v7` 库提供的。这里我们给 Toolbar 指定了一个 id，将它的宽度设置为 `match_parent`，高度设置为 `actionBar` 的高度，背景色设置为 `colorPrimary`。不过下面的部分就稍微有点难理解了，由于我们刚才在 `styles.xml` 中将程序的主题指定成了淡色主题，因此 Toolbar 现在也是淡色主题，而 Toolbar 上面的各种元素就会自动使用深色系，这是为了和主体颜色区别开。但是这个效果看起来就会很差，之前使用 `ActionBar` 时文字都是白色的，现在变成黑色的会很难看。那么为了能让 Toolbar 单独使用深色主题，这里我们使用 `android:theme` 属性，将 Toolbar 的主题指定成了 `ThemeOverlay.AppCompat.Dark.ActionBar`。但是这样指定完了之后又会出现新的问题，如果 Toolbar 中有菜单按钮（我们在 2.2.5 小节中学过），那么弹出的菜单项也会变成深色主题，这样就再次变得十分难看，于是这里使用了 `app:popupTheme` 属性单独将弹出的菜单项指定成了淡色主题。之所以使用 `app:popupTheme`，是因为 `popupTheme` 这个属性是在 Android 5.0 系统中新增的，我们使用 `app:popupTheme` 的话就可以兼容 Android 5.0 以下的系统了。

如果你觉得上面的描述很绕的话，可以自己动手做一做试验，看看不指定上述主题会是什么样的效果，这样你会理解得更加深刻。

写完了布局，接下来我们修改 `MainActivity`，代码如下所示：

```
public class MainActivity extends AppCompatActivity {
```

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
}

}

```

这里关键的代码只有两句，首先通过 `findViewById()` 得到 `Toolbar` 的实例，然后调用 `setSupportActionBar()` 方法并将 `Toolbar` 的实例传入，这样我们就做到既使用了 `Toolbar`，又让它的外观与功能都和 `ActionBar` 一致了。

现在运行一下程序，效果如图 12.3 所示。

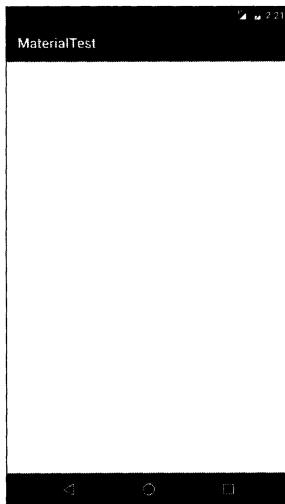


图 12.3 Toolbar 的标准界面

这个标题栏我们再熟悉不过了，虽然看上去和之前的标题栏没什么两样，但其实它已经是 `Toolbar` 而不是 `ActionBar` 了。因此它现在也具备了实现 Material Design 效果的能力，这个我们在后面就会学到。

接下来我们再学习一些 `Toolbar` 比较常用的功能吧，比如修改标题栏上显示的文字内容。这段文字内容是在 `AndroidManifest.xml` 中指定的，如下所示：

```

<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity>

```

```

    android:name=".MainActivity"
    android:label="Fruits">
    ...
</activity>
</application>

```

这里给 `activity` 增加了一个 `android:label` 属性, 用于指定在 Toolbar 中显示的文字内容, 如果没有指定的话, 会默认使用 `application` 中指定的 `label` 内容, 也就是我们的应用名称。

不过只有一个标题的 Toolbar 看起来太单调了, 我们还可以再添加一些 `action` 按钮来让 Toolbar 更加丰富一些, 这里我提前准备了几张图片来作为按钮的图标, 将它们放在了 `drawable-xxhdpi` 目录下。现在右击 `res` 目录 → New → Directory, 创建一个 `menu` 文件夹。然后右击 `menu` 文件夹 → New → Menu resource file, 创建一个 `toolbar.xml` 文件, 并编写如下代码:

```

<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/backup"
        android:icon="@drawable/ic_backup"
        android:title="Backup"
        app:showAsAction="always" />
    <item
        android:id="@+id/delete"
        android:icon="@drawable/ic_delete"
        android:title="Delete"
        app:showAsAction="ifRoom" />
    <item
        android:id="@+id/settings"
        android:icon="@drawable/ic_settings"
        android:title="Settings"
        app:showAsAction="never" />
</menu>

```

可以看到, 我们通过`<item>`标签来定义 `action` 按钮, `android:id` 用于指定按钮的 id, `android:icon` 用于指定按钮的图标, `android:title` 用于指定按钮的文字。

接着使用 `app:showAsAction` 来指定按钮的显示位置, 之所以这里再次使用了 `app` 命名空间, 同样是为了能够兼容低版本的系统。`showAsAction` 主要有以下几种值可选: `always` 表示永远显示在 Toolbar 中, 如果屏幕空间不够则不显示; `ifRoom` 表示屏幕空间足够的情况下显示在 Toolbar 中, 不够的话就显示在菜单当中; `never` 则表示永远显示在菜单当中。注意, Toolbar 中的 `action` 按钮只会显示图标, 菜单中的 `action` 按钮只会显示文字。

接下来的做法就和 2.2.5 小节中的完全一致了, 修改 `MainActivity` 中的代码, 如下所示:

```

public class MainActivity extends AppCompatActivity {
    ...
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.toolbar, menu);
    }
}

```

```
        return true;
    }

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.backup:
            Toast.makeText(this, "You clicked Backup", Toast.LENGTH_SHORT).
                show();
            break;
        case R.id.delete:
            Toast.makeText(this, "You clicked Delete", Toast.LENGTH_SHORT).
                show();
            break;
        case R.id.settings:
            Toast.makeText(this, "You clicked Settings", Toast.LENGTH_SHORT).
                show();
            break;
        default:
    }
    return true;
}

}
```

非常简单，我们在 `onCreateOptionsMenu()` 方法中加载了 `toolbar.xml` 这个菜单文件，然后在 `onOptionsItemSelected()` 方法中处理各个按钮的点击事件。现在重新运行一下程序，效果如图 12.4 所示。

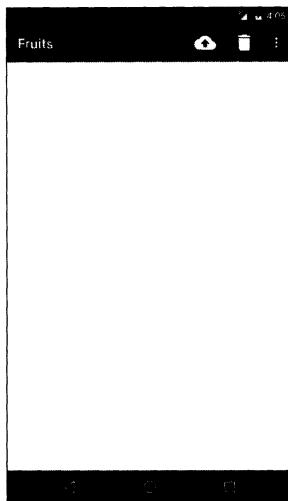


图 12.4 带有 action 按钮的 Toolbar

可以看到，Toolbar 上面现在显示了两个 action 按钮，这是因为 Backup 按钮指定的显示位置

是 always，Delete 按钮指定的显示位置是 ifRoom，而现在屏幕空间很充足，因此两个按钮都会显示在 Toolbar 中。另外一个 Settings 按钮由于指定的显示位置是 never，所以不会显示在 Toolbar 中，点击一下最右边的菜单按钮来展开菜单项，你就能找到 Settings 按钮了。另外这些 action 按钮都是可以响应点击事件的，你可以自己去试一试。

好了，关于 Toolbar 的内容就先讲这么多吧。当然 Toolbar 的功能还远远不只这些，不过我们显然无法在一节当中就把所有的用法全部学完，后面会结合其他控件来挖掘 Toolbar 的更多功能。

12.3 滑动菜单

滑动菜单可以说是 Material Design 中最常见的效果之一了，在许多著名的应用（如 Gmail、Google+等）中，都有滑动菜单的功能。虽说这个功能看上去好像挺复杂的，不过借助谷歌提供的各种工具，我们可以很轻松地实现非常炫酷的滑动菜单效果，那么我们马上开始吧。

12.3.1 DrawerLayout

所谓的滑动菜单就是将一些菜单选项隐藏起来，而不是放置在主屏幕上，然后可以通过滑动的方式将菜单显示出来。这种方式既节省了屏幕空间，又实现了非常好的动画效果，是 Material Design 中推荐的做法。

不过如果我们全靠自己去实现上述功能的话，难度恐怕就很大了。幸运的是，谷歌提供了一个 DrawerLayout 控件，借助这个控件，实现滑动菜单简单又方便。

先来简单介绍一下 DrawerLayout 的用法吧。首先它是一个布局，在布局中允许放入两个直接子控件，第一个子控件是主屏幕中显示的内容，第二个子控件是滑动菜单中显示的内容。因此，我们就可以对 activity_main.xml 中的代码做如下修改：

```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
            app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />
    
```

```
</FrameLayout>

<TextView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_gravity="start"
    android:text="This is menu"
    android:textSize="30sp"
    android:background="#FFF" />

</android.support.v4.widget.DrawerLayout>
```

可以看到，这里最外层的控件使用了 DrawerLayout，这个控件是由 support-v4 库提供的。DrawerLayout 中放置了两个直接子控件，第一个子控件是 FrameLayout，用于作为主屏幕中显示的内容，当然里面还有我们刚刚定义的 Toolbar。第二个子控件这里使用了一个 TextView，用于作为滑动菜单中显示的内容，其实使用什么都可以，DrawerLayout 并没有限制只能使用固定的控件。

但是关于第二个子控件有一点需要注意，`layout_gravity` 这个属性是必须指定的，因为我们需要告诉 DrawerLayout 滑动菜单是在屏幕的左边还是右边，指定 `left` 表示滑动菜单在左边，指定 `right` 表示滑动菜单在右边。这里我指定了 `start`，表示会根据系统语言进行判断，如果系统语言是从左往右的，比如英语、汉语，滑动菜单就在左边，如果系统语言是从右往左的，比如阿拉伯语，滑动菜单就在右边。

没错，只需要改动这么多就可以了，现在重新运行一下程序，然后在屏幕的左侧边缘向右拖动，就可以让滑动菜单显示出来了，如图 12.5 所示。

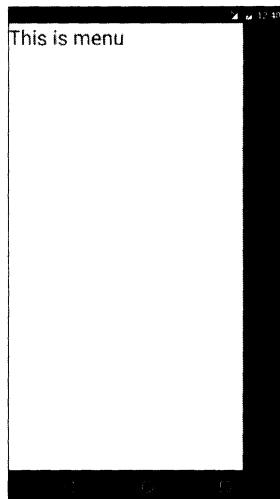


图 12.5 显示滑动菜单界面

然后向左滑动菜单，或者点击一下菜单以外的区域，都可以让滑动菜单关闭，从而回到主界面。无论是展示还是隐藏滑动菜单，都是有非常流畅的动画过渡的。

可以看到，我们只是稍微改动了一下布局文件，就能实现如此炫酷的效果，是不是觉得挺激动呢？不过现在的滑动菜单还有点问题，因为只有在屏幕的左侧边缘进行拖动时才能将菜单拖出来，而很多用户可能根本就不知道有这个功能，那么该怎么提示他们呢？

Material Design 建议的做法是在 Toolbar 的最左边加入一个导航按钮，点击了按钮也会将滑动菜单的内容展示出来。这样就相当于给用户提供了两种打开滑动菜单的方式，防止一些用户不知道屏幕的左侧边缘是可以拖动的。

下面我们开始来实现这个功能。首先我准备了一张导航按钮的图标 `ic_menu.png`，将它放在了 `drawable-xxhdpi` 目录下。然后修改 `MainActivity` 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private DrawerLayout mDrawerLayout;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        mDrawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
        ActionBar actionBar = getSupportActionBar();
        if (actionBar != null) {
            actionBar.setDisplayHomeAsUpEnabled(true);
            actionBar.setHomeAsUpIndicator(R.drawable.ic_menu);
        }
    }

    ...

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case android.R.id.home:
                mDrawerLayout.openDrawer(GravityCompat.START);
                break;
            ...
            default:
        }
        return true;
    }
}
```

这里我们并没有改动多少代码，首先调用 `findViewById()` 方法得到了 `DrawerLayout` 的实例，然后调用 `getSupportActionBar()` 方法得到了 `ActionBar` 的实例，虽然这个 `ActionBar` 的具

体实现是由 Toolbar 来完成的。接着调用 ActionBar 的 `setDisplayHomeAsUpEnabled()` 方法让导航按钮显示出来，又调用了 `setHomeAsUpIndicator()` 方法来设置一个导航按钮图标。实际上，Toolbar 最左侧的这个按钮就叫作 HomeAsUp 按钮，它默认的图标是一个返回的箭头，含义是返回上一个活动。很明显，这里我们将它默认的样式和作用都进行了修改。

接下来在 `onOptionsItemSelected()` 方法中对 HomeAsUp 按钮的点击事件进行处理，HomeAsUp 按钮的 id 永远都是 `android.R.id.home`。然后调用 DrawerLayout 的 `openDrawer()` 方法将滑动菜单展示出来，注意 `openDrawer()` 方法要求传入一个 Gravity 参数，为了保证这里的行为和 XML 中定义的一致，我们传入了 `GravityCompat.START`。

现在重新运行一下程序，效果如图 12.6 所示。

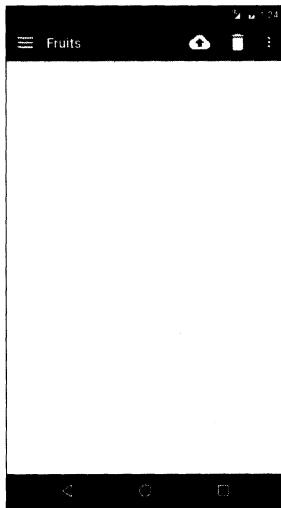


图 12.6 显示 HomeAsUp 按钮

可以看到，在 Toolbar 的最左边出现了一个导航按钮，用户看到这个按钮就知道这肯定是可以点击的。现在点击一下这个按钮，滑动菜单界面就会再次展示出来了。

12.3.2 NavigationView

目前我们已经成功实现了滑动菜单功能，其中滑动功能已经做得非常好了，但是菜单却还很丑，毕竟菜单页面仅仅使用了一个 `TextView`，非常单调。有对比才会有落差，我们看一下 Google+ 的滑动菜单页面是长什么样的，如图 12.7 所示。

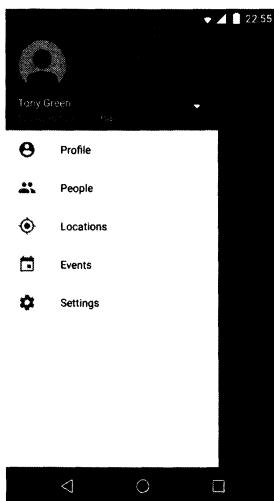


图 12.7 Google+的滑动菜单页面

经过对比之后是不是觉得我们的滑动菜单页面更丑了？不过没关系，优化滑动菜单页面，这就是我们本小节的全部目标。

事实上，你可以在滑动菜单页面定制任意的布局，不过谷歌给我们提供了一种更好的方法——使用 `NavigationView`。`NavigationView` 是 Design Support 库中提供的一个控件，它不仅是严格按照 Material Design 的要求来进行设计的，而且还可以将滑动菜单页面的实现变得非常简单。接下来我们就学习一下 `NavigationView` 的用法。

首先，既然这个控件是 Design Support 库中提供的，那么我们就需要将这个库引入到项目中才行。打开 `app/build.gradle` 文件，在 `dependencies` 闭包中添加如下内容：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:24.2.1'
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:design:24.2.1'
    compile 'de.hdodenhof:circleimageview:2.1.0'
}
```

这里添加了两行依赖关系，第一行就是 Design Support 库，第二行是一个开源项目 `CircleImageView`，它可以用来轻松实现图片圆形化的功能，我们待会就会用到它。`CircleImageView` 的项目主页地址是：<https://github.com/hdodenhof/CircleImageView>。

在开始使用 `NavigationView` 之前，我们还需要提前准备好两个东西：`menu` 和 `headerLayout`。`menu` 是用来在 `NavigationView` 中显示具体的菜单项的，`headerLayout` 则是用来在 `NavigationView` 中显示头部布局的。

我们先来准备 `menu`，这里我事先找了几张图片来作为按钮的图标，并将它们放在了

drawable-xxhdpi 目录下。然后右击 menu 文件夹→New→Menu resource file，创建一个 nav_menu.xml 文件，并编写如下代码：

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:checkableBehavior="single">
        <item
            android:id="@+id/nav_call"
            android:icon="@drawable/nav_call"
            android:title="Call" />
        <item
            android:id="@+id/nav_friends"
            android:icon="@drawable/nav_friends"
            android:title="Friends" />
        <item
            android:id="@+id/nav_location"
            android:icon="@drawable/nav_location"
            android:title="Location" />
        <item
            android:id="@+id/nav_mail"
            android:icon="@drawable/nav_mail"
            android:title="Mail" />
        <item
            android:id="@+id/nav_task"
            android:icon="@drawable/nav_task"
            android:title="Tasks" />
    </group>
</menu>
```

我们首先在`<menu>`中嵌套了一个`<group>`标签，然后将 group 的 `checkableBehavior` 属性指定为 `single`。`group` 表示一个组，`checkableBehavior` 指定为 `single` 表示组中的所有菜单项只能单选。

那么下面我们来看一下这些菜单项吧。这里一共定义了 5 个 item，分别使用 `android:id` 属性指定菜单项的 id，`android:icon` 属性指定菜单项的图标，`android:title` 属性指定菜单项显示的文字。就是这么简单，现在我们已经把 menu 准备好了。

接下来应该准备 headerLayout 了，这是一个可以随意定制的布局，不过我并不想将它做得太复杂。这里简单起见，我们就在 headerLayout 中放置头像、用户名、邮箱地址这 3 项内容吧。

说到头像，那我们还需要再准备一张图片，这里我找了一张宠物图片，并把它放在了 drawable-xxhdpi 目录下。另外这张图片最好是一张正方形图片，因为待会我们会把它圆形化。然后右击 layout 文件夹→New→Layout resource file，创建一个 nav_header.xml 文件。修改其中的代码，如下所示：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="180dp"
    android:padding="10dp"
    android:background="?attr/colorPrimary">
```

```

<de.hdodenhof.circleimageview.CircleImageView
    android:id="@+id/icon_image"
    android:layout_width="70dp"
    android:layout_height="70dp"
    android:src="@drawable/nav_icon"
    android:layout_centerInParent="true" />

<TextView
    android:id="@+id/username"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:text="tonygrendev@gmail.com"
    android:textColor="#FFF"
    android:textSize="14sp" />

<TextView
    android:id="@+id/mail"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_above="@+id/username"
    android:text="Tony Green"
    android:textColor="#FFF"
    android:textSize="14sp" />

</RelativeLayout>

```

可以看到，布局文件的最外层是一个 `RelativeLayout`，我们将它的宽度设为 `match_parent`，高度设为 `180dp`，这是一个 `NavigationView` 比较适合的高度，然后指定它的背景色为 `colorPrimary`。

在 `RelativeLayout` 中我们放置了 3 个控件，`CircleImageView` 是一个用于将图片圆形化的控件，它的用法非常简单，基本和 `ImageView` 是完全一样的，这里给它指定了一张图片作为头像，然后设置为居中显示。另外两个 `TextView` 分别用于显示用户名和邮箱地址，它们都用到了一些 `RelativeLayout` 的定位属性，相信肯定难不倒你吧？

现在 `menu` 和 `headerLayout` 都准备好了，我们终于可以使用 `NavigationView` 了。修改 `activity_main.xml` 中的代码，如下所示：

```

<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"

```

```

        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:background="?attr/colorPrimary"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
        app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />

    </FrameLayout>

    <android.support.design.widget.NavigationView
        android:id="@+id/nav_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        app:menu="@menu/nav_menu"
        app:headerLayout="@layout/nav_header" />

</android.support.v4.widget.DrawerLayout>

```

可以看到，我们将之前的 TextView 换成了 NavigationView，这样滑动菜单中显示的内容也就变成 NavigationView 了。这里又通过 app:menu 和 app:headerLayout 属性将我们刚才准备好的 menu 和 headerLayout 设置了进去，这样 NavigationView 就定义完成了。

NavigationView 虽然定义完成了，但是我们还要去处理菜单项的点击事件才行。修改 MainActivity 中的代码，如下所示：

```

public class MainActivity extends AppCompatActivity {

    private DrawerLayout mDrawerLayout;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        mDrawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
        NavigationView navView = (NavigationView) findViewById(R.id.nav_view);
        ActionBar actionBar = getSupportActionBar();
        if (actionBar != null) {
            actionBar.setDisplayHomeAsUpEnabled(true);
            actionBar.setHomeAsUpIndicator(R.drawable.ic_menu);
        }
        navView.setCheckedItem(R.id.nav_call);
        navView.setNavigationItemSelectedListener(new NavigationView.OnNavigation
            ItemSelectedListener() {
            @Override
            public boolean onNavigationItemSelected(MenuItem item) {
                mDrawerLayout.closeDrawers();
                return true;
            }
        });
    }
}

```

```
...
```

```
}
```

代码还是比较简单的，这里首先获取到了 `NavigationView` 的实例，然后调用它的 `setCheckedItem()` 方法将 `Call` 菜单项设置为默认选中。接着调用了 `setNavigationItemSelectedListener()` 方法来设置一个菜单项选中事件的监听器，当用户点击了任意菜单项时，就会回调到 `onNavigationItemSelected()` 方法中。我们可以在这个方法中写相应的逻辑处理，不过这里我并没有附加任何逻辑，只是调用了 `DrawerLayout` 的 `closeDrawers()` 方法将滑动菜单关闭，这也是合情合理的做法。

现在可以重新运行一下程序了，点击一下 `Toolbar` 左侧的导航按钮，效果如图 12.8 所示。

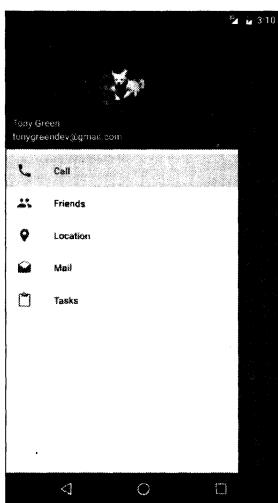


图 12.8 `NavigationView` 界面

怎么样？这样的滑动菜单页面，你无论如何也不能说它丑了吧？Material Design 的魅力就在这里，它真的是一种非常美观的设计理念，只要你按照它的各种规范和建议来设计界面，最终做出来的程序就是特别好看的。

相信你对现在做出来的效果也一定十分满意吧？不过不要满足于现状，后面我们会实现更加炫酷的效果。跟紧脚步，继续学习。

12.4 悬浮按钮和可交互提示

立面设计是 Material Design 中一条非常重要的设计思想，也就是说，按照 Material Design 的理念，应用程序的界面不仅仅只是一个平面，而应该是有立体效果的。在官方给出的示例中，最简单且最具代表性的立面设计就是悬浮按钮了，这种按钮不属于主界面平面的一部分，而是位于另外一个维度的，因此就会给人一种悬浮的感觉。

本节中我们会对这个悬浮按钮的效果进行学习，另外还会学习一种可交互式的提示工具。关于提示工具，我们之前一直都是使用的 Toast，但是 Toast 只能用于告知用户某某事情已经发生了，用户却不能对此做出任何的响应，那么今天我们就将在这一方面进行扩展。

12.4.1 FloatingActionButton

FloatingActionButton 是 Design Support 库中提供的一个控件，这个控件可以帮助我们比较轻松地实现悬浮按钮的效果。其实在之前的图 12.2 中，我们就已经预览过悬浮按钮是长什么样子的了，它默认会使用 colorAccent 来作为按钮的颜色，我们还可以通过给按钮指定一个图标来表明这个按钮的作用是什么。

下面开始来具体实现。首先仍然需要提前准备好一个图标，这里我放置了一张 ic_done.png 到 drawable-xxhdpi 目录下。然后修改 activity_main.xml 中的代码，如下所示：

```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
            app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />

        <android.support.design.widget.FloatingActionButton
            android:id="@+id/fab"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="bottom|end"
            android:layout_margin="16dp"
            android:src="@drawable/ic_done" />
    </FrameLayout>

    ...
</android.support.v4.widget.DrawerLayout>
```

可以看到，这里我们在主屏幕布局中加入了一个 FloatingActionButton。这个控件的用法并没有什么特别的地方，`layout_width` 和 `layout_height` 属性都指定成 `wrap_content`，`layout_gravity` 属性指定将这个控件放置于屏幕的右下角，其中 `end` 的工作原理和之前的

`start` 是一样的，即如果系统语言是从左往右的，那么 `end` 就表示在右边，如果系统语言是从右往左的，那么 `end` 就表示在左边。然后通过 `layout_margin` 属性给控件的四周留点边距，紧贴着屏幕边缘肯定是不好看的，最后通过 `src` 属性给 `FloatingActionButton` 设置了一个图标。

没错，就是这么简单，现在我们就可以来运行一下了，效果如图 12.9 所示。

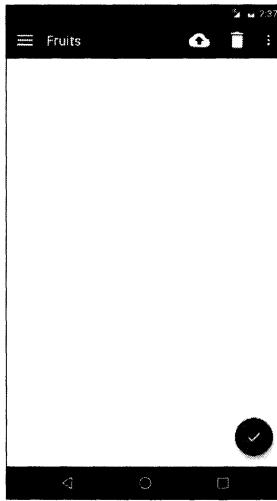


图 12.9 悬浮按钮的效果

一个漂亮的悬浮按钮就在屏幕的右下方出现了。

如果你仔细观察的话，会发现这个悬浮按钮的下面还有一点阴影。其实这很好理解，因为 `FloatingActionButton` 是悬浮在当前界面上的，既然是悬浮，那么就理所应当会有投影，Design Support 库连这种细节都帮我们考虑到了。

说到悬浮，其实我们还可以指定 `FloatingActionButton` 的悬浮高度，如下所示：

```
<android.support.design.widget.FloatingActionButton  
    android:id="@+id/fab"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_gravity="bottom|end"  
    android:layout_margin="16dp"  
    android:src="@drawable/ic_done"  
    app:elevation="8dp" />
```

这里使用 `app:elevation` 属性来给 `FloatingActionButton` 指定一个高度值，高度值越大，投影范围也越大，但是投影效果越淡，高度值越小，投影范围也越小，但是投影效果越浓。当然这些效果的差异其实都不怎么明显，我个人感觉使用默认的 `FloatingActionButton` 效果就已经足够了。

接下来我们看一下 `FloatingActionButton` 是如何处理点击事件的，毕竟，一个按钮首先要能

点击才有意义。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private DrawerLayout mDrawerLayout;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        ...
        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(MainActivity.this, "FAB clicked", Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```

如果你在期待 FloatingActionButton 会有什么特殊用法的话，那可能就要让你失望了，它和普通的 Button 其实没什么两样，都是调用 `setOnClickListener()` 方法来注册一个监听器，当点击按钮时，就会执行监听器中的 `onClick()` 方法，这里我们在 `onClick()` 方法中弹出了一个 Toast。

现在重新运行一下程序，并点击 FloatingActionButton，效果如图 12.10 所示。

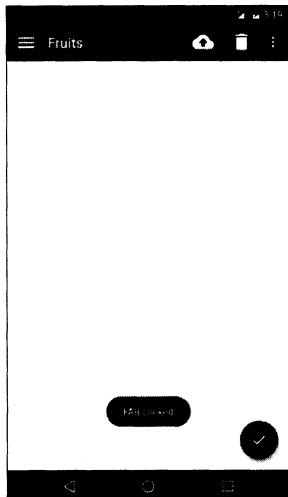


图 12.10 处理 FloatingActionButton 的点击事件

12.4.2 Snackbar

现在我们已经掌握了 FloatingActionButton 的基本用法，不过在上一小节处理点击事件的时候，仍然是使用 Toast 来作为提示工具的，本小节中我们就来学习一个 Design Support 库提供的更加先进的提示工具——Snackbar。

首先要明确，Snackbar 并不是 Toast 的替代品，它们两者之间有着不同的应用场景。Toast 的作用是告诉用户现在发生了什么事情，但同时用户只能被动接收这个事情，因为没有什么办法能让用户进行选择。而 Snackbar 则在这方面进行了扩展，它允许在提示当中加入一个可交互按钮，当用户点击按钮的时候可以执行一些额外的逻辑操作。打个比方，如果我们在执行删除操作的时候只弹出一个 Toast 提示，那么用户要是误删了某个重要数据的话肯定会十分抓狂吧，但是如果我们在增加一个 Undo 按钮，就相当于给用户提供了一种弥补措施，从而大大降低了事故发生的概率，提升了用户体验。

Snackbar 的用法也非常简单，它和 Toast 是基本相似的，只不过可以额外增加一个按钮的点击事件。修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private DrawerLayout mDrawerLayout;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ...
        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Snackbar.make(view, "Data deleted", Snackbar.LENGTH_SHORT)
                    .setAction("Undo", new View.OnClickListener() {
                        @Override
                        public void onClick(View v) {
                            Toast.makeText(MainActivity.this, "Data restored",
                                Toast.LENGTH_SHORT).show();
                        }
                    })
                    .show();
            }
        });
        ...
    }
}
```

可以看到，这里调用了 Snackbar 的 make() 方法来创建一个 Snackbar 对象，make() 方法的

第一个参数需要传入一个 View，只要是当前界面布局的任意一个 View 都可以，Snackbar 会使用这个 View 来自动查找最外层的布局，用于展示 Snackbar。第二个参数就是 Snackbar 中显示的内容，第三个参数是 Snackbar 显示的时长。这些和 Toast 都是类似的。

接着这里又调用了一个 `setAction()` 方法来设置一个动作，从而让 Snackbar 不仅仅是一个提示，而是可以和用户进行交互的。简单起见，我们在动作按钮的点击事件里面弹出一个 Toast 提示。最后调用 `show()` 方法让 Snackbar 显示出来。

现在重新运行一下程序，并点击悬浮按钮，效果如图 12.11 所示。

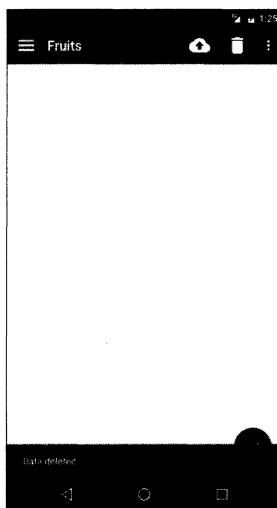


图 12.11 Snackbar 的效果

可以看到，Snackbar 从屏幕底部出现了，上面有我们所设置的提示文字，还有一个 Undo 按钮，按钮是可以点击的。过一段时间后 Snackbar 会自动从屏幕底部消失。

不管是出现还是消失，Snackbar 都是带有动画效果的，因此视觉体验也会比较好。

不过你有没有发现一个 bug，这个 Snackbar 竟然将我们的悬浮按钮给遮挡住了。虽说也不是什么重大的问题，因为 Snackbar 过一会儿就会自动消失，但这种用户体验总归是不友好的。有没有什么办法能解决一下呢？当然有，只需要借助 CoordinatorLayout 就可以轻松解决。

12.4.3 CoordinatorLayout

CoordinatorLayout 可以说是一个加强版的 FrameLayout，这个布局也是由 Design Support 库提供的。它在普通情况下的作用和 FrameLayout 基本一致，不过既然是 Design Support 库中提供的布局，那么就必然有一些 Material Design 的魔力了。

事实上，CoordinatorLayout 可以监听其所有子控件的各种事件，然后自动帮助我们做出最为

合理的响应。举个简单的例子，刚才弹出的 Snackbar 提示将悬浮按钮遮挡住了，而如果我们能让 CoordinatorLayout 监听到 Snackbar 的弹出事件，那么它会自动将内部的 FloatingActionButton 向上偏移，从而确保不会被 Snackbar 遮挡到。

至于 CoordinatorLayout 的使用也非常简单，我们只需要将原来的 FrameLayout 替换一下就可以了。修改 activity_main.xml 中的代码，如下所示：

```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.design.widget.CoordinatorLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
            app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />

        <android.support.design.widget.FloatingActionButton
            android:id="@+id/fab"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="bottom|end"
            android:layout_margin="16dp"
            android:src="@drawable/ic_done" />
    </android.support.design.widget.CoordinatorLayout>

    ...
</android.support.v4.widget.DrawerLayout>
```

由于 CoordinatorLayout 本身就是一个加强版的 FrameLayout，因此这种替换不会有任何的副作用。现在重新运行一下程序，并点击悬浮按钮，效果如图 12.12 所示。

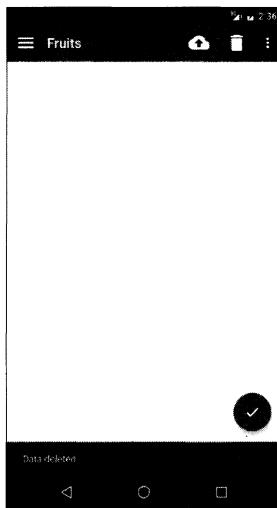


图 12.12 CoordinatorLayout 自动将悬浮按钮上移

可以看到，悬浮按钮自动向上偏移了 Snackbar 的同等高度，从而确保不会被遮挡住，当 Snackbar 消失的时候，悬浮按钮会自动向下偏移回到原来位置。

另外悬浮按钮的向上和向下偏移也是伴随着动画效果的，且和 Snackbar 完全同步，整体效果看上去特别赏心悦目。

不过我们回过头来再思考一下，刚才说的是 CoordinatorLayout 可以监听其所有子控件的各种事件，但是 Snackbar 好像并不是 CoordinatorLayout 的子控件吧，为什么它却可以被监听到呢？

其实道理很简单，还记得我们在 Snackbar 的 `make()` 方法中传入的第一个参数吗？这个参数就是用来指定 Snackbar 是基于哪个 View 来触发的，刚才我们传入的是 FloatingActionButton 本身，而 FloatingActionButton 是 CoordinatorLayout 中的子控件，因此这个事件就理所应当能被监听到了。你可以自己再做个试验，如果给 Snackbar 的 `make()` 方法传入一个 DrawerLayout，那么 Snackbar 就会再次遮挡住悬浮按钮，因为 DrawerLayout 不是 CoordinatorLayout 的子控件，CoordinatorLayout 也就无法监听到 Snackbar 的弹出和隐藏事件了。

本节的内容就到这里，接下来我们继续丰富 MaterialTest 项目，加入卡片式布局效果。

12.5 卡片式布局

虽然现在 MaterialTest 中已经应用了非常多的 Material Design 效果，不过你会发现，界面上最主要的一块区域还处于空白状态。这块区域通常都是用来放置应用的主体内容的，我准备使用一些精美的水果图片来填充这部分区域。

那么为了要让水果图片也能 Material 化，本节中我们将会学习如何实现卡片式布局的效果。

卡片式布局也是 Materials Design 中提出的一个新的概念，它可以让页面中的元素看起来就像在卡片中一样，并且还能拥有圆角和投影，下面我们就开始具体学习一下。

12.5.1 CardView

CardView是用于实现卡片式布局效果的重要控件，由appcompat-v7库提供。实际上，CardView也是一个FrameLayout，只是额外提供了圆角和阴影等效果，看上去会有立体的感觉。

我们先来看一下 CardView 的基本用法吧，其实非常简单，如下所示：

```
<android.support.v7.widget.CardView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    app:cardCornerRadius="4dp"
    app:elevation="5dp">
    <TextView
        android:id="@+id/info_text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>
</android.support.v7.widget.CardView>
```

这里定义了一个 CardView 布局，我们可以通过 `app:cardCornerRadius` 属性指定卡片圆角的弧度，数值越大，圆角的弧度也越大。另外还可以通过 `app:elevation` 属性指定卡片的高度，高度值越大，投影范围也越大，但是投影效果越淡，高度值越小，投影范围也越小，但是投影效果越浓，这一点和 FloatingActionButton 是一致的。

然后我们在 CardView 布局中放置了一个 TextView，那么这个 TextView 就会显示在一张卡片当中了，CardView 的用法就是这么简单。

但是我们显然不可能在如此宽阔的一块空白区域内只放置一张卡片，为了能够充分利用屏幕的空间，这里我准备综合运用一下第 3 章中学到的知识，使用 RecyclerView 来填充 MaterialTest 项目的主界面部分。还记得之前实现过的水果列表效果吗？这次我们将升级一下，实现一个高配版的水果列表效果。

既然是要实现水果列表，那么首先肯定需要准备许多张水果图片，这里我从网上挑选了一些精美的水果图片，将它们复制到了项目当中。

然后由于我们还需要用到 RecyclerView、CardView 这几个控件，因此必须在 `app/build.gradle` 文件中声明这些库的依赖才行：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:24.2.1'
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:design:24.2.1'
    compile 'de.hdodenhof:circleimageview:2.1.0'
    compile 'com.android.support:recyclerview-v7:24.2.1'
    compile 'com.android.support:cardview-v7:24.2.1'
```

```
        compile 'com.github.bumptech.glide:glide:3.7.0'
    }
```

注意上述声明的最后一行，这里添加了一个 Glide 库的依赖。Glide 是一个超级强大的图片加载库，它不仅可以用于加载本地图片，还可以加载网络图片、GIF 图片、甚至是本地视频。最重要的是，Glide 的用法非常简单，只需一行代码就能轻松实现复杂的图片加载功能，因此这里我们准备用它来加载水果图片。Glide 的项目主页地址是：<https://github.com/bumptech/glide>。

接下来开始具体的代码实现，修改 activity_main.xml 中的代码，如下所示：

```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.design.widget.CoordinatorLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
            app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />

        <android.support.v7.widget.RecyclerView
            android:id="@+id/recycler_view"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />

        <android.support.design.widget.FloatingActionButton
            android:id="@+id/fab"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="bottom|end"
            android:layout_margin="16dp"
            android:src="@drawable/ic_done" />
    </android.support.design.widget.CoordinatorLayout>

    ...
</android.support.v4.widget.DrawerLayout>
```

这里我们在 CoordinatorLayout 中添加了一个 RecyclerView，给它指定一个 id，然后将宽度和高度都设置为 match_parent，这样 RecyclerView 也就占满了整个布局的空间。

接着定义一个实体类 Fruit，代码如下所示：

```
public class Fruit {
```

```
private String name;  
private int imageId;  
  
public Fruit(String name, int imageId) {  
    this.name = name;  
    this.imageId = imageId;  
}  
  
public String getName() {  
    return name;  
}  
  
public int getImageId() {  
    return imageId;  
}  
}
```

Fruit 类中只有两个字段，name 表示水果的名字，imageId 表示水果对应图片的资源 id。

然后需要为 RecyclerView 的子项指定一个我们自定义的布局，在 layout 目录下新建 fruit_item.xml，代码如下所示：

```
<android.support.v7.widget.CardView  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:layout_margin="5dp"  
    app:cardCornerRadius="4dp">  
  
    <LinearLayout  
        android:orientation="vertical"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content">  
  
        <ImageView  
            android:id="@+id/fruit_image"  
            android:layout_width="match_parent"  
            android:layout_height="100dp"  
            android:scaleType="centerCrop" />  
  
        <TextView  
            android:id="@+id/fruit_name"  
            android:layout_width="wrap_content"  
            android:layout_height="wrap_content"  
            android:layout_gravity="center_horizontal"  
            android:layout_margin="5dp"  
            android:textSize="16sp" />  
    </LinearLayout>  
</android.support.v7.widget.CardView>
```

这里使用了 CardView 来作为子项的最外层布局，从而使得 RecyclerView 中的每个元素都是在卡片当中的。CardView 由于是一个 FrameLayout，因此它没有什么方便的定位方式，这里我们只好在 CardView 中再嵌套一个 LinearLayout，然后在 LinearLayout 中放置具体的内容。

内容倒也没有什么特殊的地方，就是定义了一个 ImageView 用于显示水果的图片，又定义了一个 TextView 用于显示水果的名称，并让 TextView 在水平方向上居中显示。注意在 ImageView 中我们使用了一个 scaleType 属性，这个属性可以指定图片的缩放模式。由于各张水果图片的长宽比例可能都不一致，为了让所有的图片都能填充满整个 ImageView，这里使用了 centerCrop 模式，它可以让图片保持原有比例填充满 ImageView，并将超出屏幕的部分裁剪掉。

接下来需要为 RecyclerView 准备一个适配器，新建 FruitAdapter 类，让这个适配器继承自 RecyclerView.Adapter，并将泛型指定为 FruitAdapter.ViewHolder，代码如下所示：

```
public class FruitAdapter extends RecyclerView.Adapter<FruitAdapter.ViewHolder> {

    private Context mContext;

    private List<Fruit> mFruitList;

    static class ViewHolder extends RecyclerView.ViewHolder {
        CardView cardView;
        ImageView fruitImage;
        TextView fruitName;

        public ViewHolder(View view) {
            super(view);
            cardView = (CardView) view;
            fruitImage = (ImageView) view.findViewById(R.id.fruit_image);
            fruitName = (TextView) view.findViewById(R.id.fruit_name);
        }
    }

    public FruitAdapter(List<Fruit> fruitList) {
        mFruitList = fruitList;
    }

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        if (mContext == null) {
            mContext = parent.getContext();
        }
        View view = LayoutInflater.from(mContext).inflate(R.layout.fruit_item,
                parent, false);
        return new ViewHolder(view);
    }

    @Override
    public void onBindViewHolder(ViewHolder holder, int position) {
        Fruit fruit = mFruitList.get(position);
        holder.fruitName.setText(fruit.getName());
        Glide.with(mContext).load(fruit.getImageId()).into(holder.fruitImage);
    }

    @Override
    public int getItemCount() {
        return mFruitList.size();
    }
}
```

```

    }

    @Override
    public int getItemCount() {
        return mFruitList.size();
    }
}

```

上述代码相信你一定很熟悉，和我们在第3章中编写的FruitAdapter几乎一模一样。唯一需要注意的是，在onBindViewHolder()方法中我们使用了Glide来加载水果图片。

那么这里就顺便来看一下Glide的用法吧，其实并没有太多好讲的，因为Glide的用法实在是太简单了。首先调用Glide.with()方法并传入一个Context、Activity或Fragment参数，然后调用load()方法去加载图片，可以是一个URL地址，也可以是一个本地路径，或者是一个资源id，最后调用into()方法将图片设置到具体某一个ImageView中就可以了。

那么我们为什么要使用Glide而不是传统的设置图片方式呢？因为这次我从网上找的这些水果图片像素都非常高，如果不进行压缩就直接展示的话，很容易就会引起内存溢出。而使用Glide就完全不需要担心这回事，因为Glide在内部做了许多非常复杂的逻辑操作，其中就包括了图片压缩，我们只需要安心按照Glide的标准用法去加载图片就可以了。

这样我们就将RecyclerView的适配器也准备好了，最后修改MainActivity中的代码，如下所示：

```

public class MainActivity extends AppCompatActivity {

    private DrawerLayout mDrawerLayout;

    private Fruit[] fruits = {new Fruit("Apple", R.drawable.apple), new Fruit("Banana",
        R.drawable.banana),
        new Fruit("Orange", R.drawable.orange), new Fruit("Watermelon", R.
            drawable.watermelon),
        new Fruit("Pear", R.drawable.pear), new Fruit("Grape", R.drawable.
            grape),
        new Fruit("Pineapple", R.drawable.pineapple), new Fruit("Strawberry",
            R.drawable.strawberry),
        new Fruit("Cherry", R.drawable.cherry), new Fruit("Mango", R.drawable.
            mango)};

    private List<Fruit> fruitList = new ArrayList<>();

    private FruitAdapter adapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        ...
        initFruits();
        RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recycler_view);

```

```

        GridLayoutManager layoutManager = new GridLayoutManager(this, 2);
        recyclerView.setLayoutManager(layoutManager);
        adapter = new FruitAdapter(fruitList);
        recyclerView.setAdapter(adapter);
    }

    private void initFruits() {
        fruitList.clear();
        for (int i = 0; i < 50; i++) {
            Random random = new Random();
            int index = random.nextInt(fruits.length);
            fruitList.add(fruits[index]);
        }
    }
}

```

在 MainActivity 中我们首先定义了一个数组，数组里面存放了很多个 Fruit 的实例，每个实例都代表着一种水果。然后在 initFruits()方法中，先是清空了一下 fruitList 中的数据，接着使用一个随机函数，从刚才定义的 Fruit 数组中随机挑选一个水果放入到 fruitList 当中，这样每次打开程序看到的水果数据都会是不同的。另外，为了让界面上的数据多一些，这里使用了一个循环，随机挑选 50 个水果。

之后的用法就是 RecyclerView 的标准用法了，不过这里使用了 GridLayoutManager 这种布局方式。在第 3 章中我们已经学过了 LinearLayoutManager 和 StaggeredLayoutManager，现在终于将所有的布局方式都补齐了。GridLayoutManager 的用法也没有什么特别之处，它的构造函数接收两个参数，第一个是 Context，第二个是列数，这里我们希望每一行中会有两列数据。

现在重新运行一下程序，效果如图 12.13 所示。

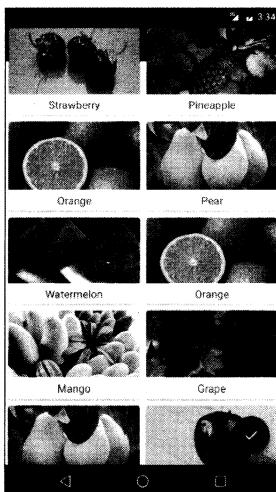


图 12.13 卡片式布局效果

可以看到，精美的水果图片成功展示出来了。每个水果都是在一张单独的卡片当中的，并且还拥有圆角和投影，是不是非常美观？另外，由于我们是使用随机的方式来获取水果数据的，因此界面上会有一些重复的水果出现，这属于正常现象。

当你陶醉于当前精美的界面的时候，你是不是忽略了一个细节？哎呀，我们的 Toolbar 怎么不见了！仔细观察一下原来是被 RecyclerView 给挡住了。这个问题又该怎么解决呢？这就需要借助到另外一个工具了——AppBarLayout。

12.5.2 AppBarLayout

首先我们来分析一下为什么 RecyclerView 会把 Toolbar 给遮挡住吧。其实并不难理解，由于 RecyclerView 和 Toolbar 都是放置在 CoordinatorLayout 中的，而前面已经说过，CoordinatorLayout 就是一个加强版的 FrameLayout，那么 FrameLayout 中的所有控件在不进行明确定位的情况下，默认都会摆放在布局的左上角，从而也就产生了遮挡的现象。其实这已经不是你第一次遇到这种情况了，我们在 3.3.3 小节学习 FrameLayout 的时候就早已见识过了控件与控件之间遮挡的效果。

既然已经找到了问题的原因，那么该如何解决呢？传统情况下，使用偏移是唯一的解决办法，即让 RecyclerView 向下偏移一个 Toolbar 的高度，从而保证不会遮挡到 Toolbar。不过我们使用的并不是普通的 FrameLayout，而是 CoordinatorLayout，因此自然会有一些更加巧妙的解决办法。

这里我准备使用 Design Support 库中提供的另外一个工具——AppBarLayout。AppBarLayout 实际上是一个垂直方向的 LinearLayout，它在内部做了很多滚动事件的封装，并应用了一些 Material Design 的设计理念。

那么我们怎样使用 AppBarLayout 才能解决前面的覆盖问题呢？其实只需要两步就可以了，第一步将 Toolbar 嵌套到 AppBarLayout 中，第二步给 RecyclerView 指定一个布局行为。修改 activity_main.xml 中的代码，如下所示：

```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.design.widget.CoordinatorLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <android.support.design.widget.AppBarLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content">

            <android.support.v7.widget.Toolbar
                android:id="@+id/toolbar"
                android:layout_width="match_parent"
                android:layout_height="?attr/actionBarSize"
```

```
        android:background="?attr/colorPrimary"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
        app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />
    </android.support.design.widget.AppBarLayout>

    <android.support.v7.widget.RecyclerView
        android:id="@+id/recycler_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="@string/appbar_scrolling_view_behavior" />

    ...
</android.support.design.widget.CoordinatorLayout>

...
</android.support.v4.widget.DrawerLayout>
```

可以看到，布局文件并没有什么太大的变化。我们首先定义了一个 AppBarLayout，并将 Toolbar 放置在了 AppBarLayout 里面，然后在 RecyclerView 中使用 `app:layout_behavior` 属性指定了一个布局行为。其中 `appbar_scrolling_view_behavior` 这个字符串也是由 Design Support 库提供的。

现在重新运行一下程序，你就会发现一切都正常了，如图 12.14 所示。

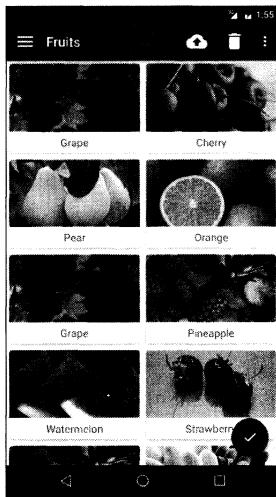


图 12.14 解决 RecyclerView 遮挡 Toolbar 的问题

虽说使用 AppBarLayout 已经成功解决了 RecyclerView 遮挡 Toolbar 的问题，但是刚才有提到过，说 AppBarLayout 中应用了一些 Material Design 的设计理念，好像从上面的例子完全体现不出来呀。事实上，当 RecyclerView 滚动的时候就已经将滚动事件都通知给 AppBarLayout 了，只是我们还没进行处理而已。那么下面就让我们来进一步优化，看看 AppBarLayout 到底能实现什

么样的 Material Design 效果。

当 AppBarLayout 接收到滚动事件的时候，它内部的子控件其实是可以指定如何去影响这些事件的，通过 `app:layout_scrollFlags` 属性就能实现。修改 `activity_main.xml` 中的代码，如下所示：

```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.design.widget.CoordinatorLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <android.support.design.widget.AppBarLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content">

            <android.support.v7.widget.Toolbar
                android:id="@+id/toolbar"
                android:layout_width="match_parent"
                android:layout_height="?attr/actionBarSize"
                android:background="?attr/colorPrimary"
                android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
                app:popupTheme="@style/ThemeOverlay.AppCompat.Light"
                app:layout_scrollFlags="scroll|enterAlways|snap" />
            </android.support.design.widget.AppBarLayout>

            ...
        </android.support.design.widget.CoordinatorLayout>

        ...
    </android.support.v4.widget.DrawerLayout>
```

这里在 Toolbar 中添加了一个 `app:layout_scrollFlags` 属性，并将这个属性的值指定成了 `scroll|enterAlways|snap`。其中，`scroll` 表示当 RecyclerView 向上滚动的时候，Toolbar 会跟着一起向上滚动并实现隐藏；`enterAlways` 表示当 RecyclerView 向下滚动的时候，Toolbar 会跟着一起向下滚动并重新显示。`snap` 表示当 Toolbar 还没有完全隐藏或显示的时候，会根据当前滚动的距离，自动选择是隐藏还是显示。

我们要改动的就只有这一行代码而已，现在重新运行一下程序，并向上滚动 RecyclerView，效果如图 12.15 所示。

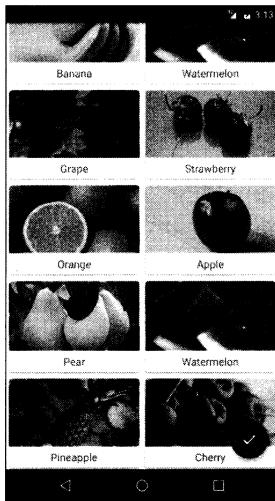


图 12.15 向上滚动 RecyclerView 隐藏 Toolbar

可以看到，随着我们向上滚动 RecyclerView，Toolbar 竟然消失了，而向下滚动 RecyclerView，Toolbar 又会重新出现。这其实也是 Material Design 中的一项重要设计思想，因为当用户在向上滚动 RecyclerView 的时候，其注意力肯定是在 RecyclerView 的内容上面的，这个时候如果 Toolbar 还占据着屏幕空间，就会在一定程度上影响用户的阅读体验，而将 Toolbar 隐藏则可以让阅读体验达到最佳状态。当用户需要操作 Toolbar 上的功能时，只需要轻微向下滚动，Toolbar 就会重新出现。这种设计方式，既保证了用户的最佳阅读效果，又不影响任何功能上的操作，Material Design 考虑得就是这么细致入微。

当然了，像这种功能，如果是使用 ActionBar 的话，那就完全不可能实现了，Toolbar 的出现为我们提供了更多的可能。

12.6 下拉刷新

下拉刷新这种功能早就不是什么新鲜的东西了，几乎所有的应用里都会有这个功能。不过市面上现有的下拉刷新功能在风格上都各不相同，并且和 Material Design 还有些格格不入的感觉。因此，谷歌为了让 Android 的下拉刷新风格能有一个统一的标准，于是在 Material Design 中制定了一个官方的设计规范。当然，我们并不需要去深入了解这个规范到底是什么样的，因为谷歌早就提供好了现成的控件，我们只需要在项目中直接使用就可以了。

SwipeRefreshLayout 就是用于实现下拉刷新功能的核心类，它是由 support-v4 库提供的。我们把想要实现下拉刷新功能的控件放置到 SwipeRefreshLayout 中，就可以迅速让这个控件支持下拉刷新。那么在 MaterialTest 项目中，应该支持下拉刷新功能的控件自然就是 RecyclerView 了。

由于 SwipeRefreshLayout 的用法也比较简单，下面我们就直接开始使用了。修改 activity_

main.xml 中的代码，如下所示：

```

<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.design.widget.CoordinatorLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        ...

        <android.support.v4.widget.SwipeRefreshLayout
            android:id="@+id/swipe_refresh"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            app:layout_behavior="@string/appbar_scrolling_view_behavior">

            <android.support.v7.widget.RecyclerView
                android:id="@+id/recycler_view"
                android:layout_width="match_parent"
                android:layout_height="match_parent" />
        </android.support.v4.widget.SwipeRefreshLayout>

        ...

    </android.support.design.widget.CoordinatorLayout>

    ...

</android.support.v4.widget.DrawerLayout>

```

可以看到，这里我们在 RecyclerView 的外面又嵌套了一层 SwipeRefreshLayout，这样 RecyclerView 就自动拥有下拉刷新功能了。另外需要注意，由于 RecyclerView 现在变成了 SwipeRefreshLayout 的子控件，因此之前使用 `app:layout_behavior` 声明的布局行为现在也要移到 SwipeRefreshLayout 中才行。

不过这还没有结束，虽然 RecyclerView 已经支持下拉刷新功能了，但是我们还要在代码中处理具体的刷新逻辑才行。修改 MainActivity 中的代码，如下所示：

```

public class MainActivity extends AppCompatActivity {

    ...

    private SwipeRefreshLayout swipeRefresh;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ...

```

```

        swipeRefresh = (SwipeRefreshLayout) findViewById(R.id.swipe_refresh);
        swipeRefresh.setColorSchemeResources(R.color.colorPrimary);
        swipeRefresh.setOnRefreshListener(new SwipeRefreshLayout.
            OnRefreshListener() {
            @Override
            public void onRefresh() {
                refreshFruits();
            }
        });
    }

    private void refreshFruits() {
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    Thread.sleep(2000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        initFruits();
                        adapter.notifyDataSetChanged();
                        swipeRefresh.setRefreshing(false);
                    }
                });
            }
        }).start();
    }

    ...
}

```

这段代码应该还是比较好理解的，首先通过 `findViewById()` 方法拿到 `SwipeRefreshLayout` 的实例，然后调用 `setColorSchemeResources()` 方法来设置下拉刷新进度条的颜色，这里我们就使用主题中的 `colorPrimary` 作为进度条的颜色了。接着调用 `setOnRefreshListener()` 方法来设置一个下拉刷新的监听器，当触发了下拉刷新操作的时候就会回调这个监听器的 `onRefresh()` 方法，然后我们在这里去处理具体的刷新逻辑就可以了。

通常情况下，`onRefresh()` 方法中应该是去网络上请求最新的数据，然后再将这些数据展示出来。这里简单起见，我们就不和网络进行交互了，而是调用一个 `refreshFruits()` 方法进行本地刷新操作。`refreshFruits()` 方法中先是开启了一个线程，然后将线程沉睡两秒钟。之所以这么做，是因为本地刷新操作速度非常快，如果不将线程沉睡的话，刷新立刻就结束了，从而看不到刷新的过程。沉睡结束之后，这里使用了 `runOnUiThread()` 方法将线程切换回主线程，然后调用 `initFruits()` 方法重新生成数据，接着再调用 `FruitAdapter` 的 `notifyDataSetChanged()` 方法通知数据发生了变化，最后调用 `SwipeRefreshLayout` 的 `setRefreshing()` 方法并传入 `false`，用于表示刷新事件结束，并隐藏刷新进度条。

现在可以重新运行一下程序了，在屏幕的主界面向下拖动，会有一个下拉刷新的进度条出现，松手后就会自动进行刷新了，效果如图 12.16 所示。

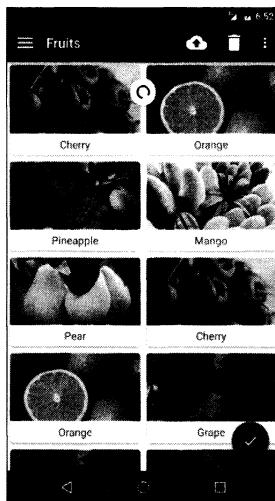


图 12.16 实现下拉刷新效果

下拉刷新的进度条只会停留两秒钟，之后就会自动消失，界面上的水果数据也会随之更新。

这样我们就把下拉刷新的功能也成功实现了，并且这就是 Material Design 中规定的最标准的下拉刷新效果，还有什么会比这个更好看呢？目前我们的项目中已经应用了众多 Material Design 的效果，Design Support 库中的常用控件也学了大半了。不过本章的学习之旅还没有结束，在最后的尾声部分，我们再来实现一个非常震撼的 Material Design 效果——可折叠式标题栏。

12.7 可折叠式标题栏

虽说我们现在的标题栏是使用 Toolbar 来编写的，不过它看上去和传统的 ActionBar 其实没什么两样，只不过可以响应 RecyclerView 的滚动事件来进行隐藏和显示。而 Material Design 中并没有限定标题栏必须是长这个样子的，事实上，我们可以根据自己的喜好随意定制标题栏的样式。那么本节中我们就来实现一个可折叠式标题栏的效果，需要借助 CollapsingToolbarLayout 这个工具。

12.7.1 CollapsingToolbarLayout

顾名思义，CollapsingToolbarLayout 是一个作用于 Toolbar 基础之上的布局，它也是由 Design Support 库提供的。CollapsingToolbarLayout 可以让 Toolbar 的效果变得更加丰富，不仅仅是展示一个标题栏，而是能够实现非常华丽的效果。

不过，CollapsingToolbarLayout 是不能独立存在的，它在设计的时候就被限定只能作为

AppBarLayout 的直接子布局来使用。而 AppBarLayout 又必须是 CoordinatorLayout 的子布局，因此本节中我们要实现的功能其实需要综合运用前面所学的各种知识。那么话不多说，这就开始吧。

首先我们需要一个额外的活动来作为水果的详情展示界面，右击 com.example.materialtest 包 → New → Activity → Empty Activity，创建一个 FruitActivity，并将布局名指定成 activity_fruit.xml，然后我们开始编写水果详情展示界面的布局。

由于整个布局文件比较复杂，这里我准备采用分段编写的方式。activity_fruit.xml 中的内容主要分为两部分，一个是水果标题栏，一个是水果内容详情，我们来一步步实现。

首先实现标题栏部分，这里使用 CoordinatorLayout 来作为最外层布局，如下所示：

```
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

</android.support.design.widget.CoordinatorLayout>
```

一开始的代码还是比较简单的，相信没有什么需要解释的地方。注意始终记得要定义一个 xmlns:app 的命名空间，在 Material Design 的开发中会经常用到它。

接着我们在 CoordinatorLayout 中嵌套一个 AppBarLayout，如下所示：

```
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.design.widget.AppBarLayout
        android:id="@+id/appBar"
        android:layout_width="match_parent"
        android:layout_height="250dp">
    </android.support.design.widget.AppBarLayout>

</android.support.design.widget.CoordinatorLayout>
```

目前为止也没有什么难理解的地方，我们给 AppBarLayout 定义了一个 id，将它的宽度指定为 match_parent，高度指定为 250dp。当然这里的高度值你可以随意指定，不过我尝试之后发现 250dp 的视觉效果比较好。

接下来我们在 AppBarLayout 中再嵌套一个 CollapsingToolbarLayout，如下所示：

```
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
```

```

<android.support.design.widget.AppBarLayout
    android:id="@+id/appBar"
    android:layout_width="match_parent"
    android:layout_height="250dp">

    <android.support.design.widget.CollapsingToolbarLayout
        android:id="@+id/collapsing_toolbar"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
        app:contentScrim="?attr/colorPrimary"
        app:layout_scrollFlags="scroll|exitUntilCollapsed">
    </android.support.design.widget.CollapsingToolbarLayout>

</android.support.design.widget.AppBarLayout>

</android.support.design.widget.CoordinatorLayout>

```

从现在开始就稍微有点难理解了，这里我们使用了新的布局 CollapsingToolbarLayout。其中，`id`、`layout_width` 和 `layout_height` 这几个属性比较简单，我就不解释了。`android:theme` 属性指定了一个 `ThemeOverlay.AppCompat.Dark.ActionBar` 的主题，其实对于这部分我们也并不陌生，因为之前在 `activity_main.xml` 中给 Toolbar 指定的也是这个主题，只不过这里要实现更加高级的 Toolbar 效果，因此需要将这个主题的指定提到上一层来。`app:contentScrim` 属性用于指定 CollapsingToolbarLayout 在趋于折叠状态以及折叠之后的背景色，其实 CollapsingToolbarLayout 在折叠之后就是一个普通的 Toolbar，那么背景色肯定应该是 `colorPrimary` 了，具体的效果我们待会儿就能看到。`app:layout_scrollFlags` 属性我们也是见过的，只不过之前是给 Toolbar 指定的，现在也移到外面来了。其中，`scroll` 表示 CollapsingToolbarLayout 会随着水果内容详情的滚动一起滚动，`exitUntilCollapsed` 表示当 CollapsingToolbarLayout 随着滚动完成折叠之后就保留在界面上，不再移出屏幕。

接下来，我们在 CollapsingToolbarLayout 中定义标题栏的具体内容，如下所示：

```

<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.design.widget.AppBarLayout
        android:id="@+id/appBar"
        android:layout_width="match_parent"
        android:layout_height="250dp">

        <android.support.design.widget.CollapsingToolbarLayout
            android:id="@+id/collapsing_toolbar"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
            app:contentScrim="?attr/colorPrimary"
            app:layout_scrollFlags="scroll|exitUntilCollapsed">
    </android.support.design.widget.CollapsingToolbarLayout>

```

```

<ImageView
    android:id="@+id/fruit_image_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:scaleType="centerCrop"
    app:layout_collapseMode="parallax" />

<android.support.v7.widget.Toolbar
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    app:layout_collapseMode="pin" />
</android.support.design.widget.CollapsingToolbarLayout>

</android.support.design.widget.AppBarLayout>

</android.support.design.widget.CoordinatorLayout>

```

可以看到，我们在 CollapsingToolbarLayout 中定义了一个 ImageView 和一个 Toolbar，也就意味着，这个高级版的标题栏将是由普通的标题栏加上图片组合而成的。这里定义的大多数属性我们都是见过的，就不再解释了，只有一个 app:layout_collapseMode 比较陌生。它用于指定当前控件在 CollapsingToolbarLayout 折叠过程中的折叠模式，其中 Toolbar 指定成 pin，表示在折叠的过程中位置始终保持不变，ImageView 指定成 parallax，表示会在折叠的过程中产生一定的错位偏移，这种模式的视觉效果会非常好。

这样我们就将水果标题栏的界面编写完成了，下面开始编写水果内容详情部分。继续修改 activity_fruit.xml 中的代码，如下所示：

```

<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.design.widget.AppBarLayout
        android:id="@+id/appBar"
        android:layout_width="match_parent"
        android:layout_height="250dp">
        ...
    </android.support.design.widget.AppBarLayout>

    <android.support.v4.widget.NestedScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="@string/appbar_scrolling_view_behavior">
        ...
    </android.support.v4.widget.NestedScrollView>
</android.support.design.widget.CoordinatorLayout>

```

水果内容详情的最外层布局使用了一个 NestedScrollView，注意它和 AppBarLayout 是平级的。

我们之前在 9.2.1 小节学过 ScrollView 的用法，它允许使用滚动的方式来查看屏幕以外的数据，而 NestedScrollView 在此基础之上还增加了嵌套响应滚动事件的功能。由于 CoordinatorLayout 本身已经可以响应滚动事件了，因此我们在它的内部就需要使用 NestedScrollView 或 RecyclerView 这样的布局。另外，这里还通过 `app:layout_behavior` 属性指定了一个布局行为，这和之前在 RecyclerView 中的用法是一模一样的。

不管是 ScrollView 还是 NestedScrollView，它们的内部都只允许存在一个直接子布局。因此，如果我们想要在里面放入很多东西的话，通常都会先嵌套一个 LinearLayout，然后再在 LinearLayout 中放入具体的内容就可以了，如下所示：

```
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    ...

    <android.support.v4.widget.NestedScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="@string/appbar_scrolling_view_behavior">

        <LinearLayout
            android:orientation="vertical"
            android:layout_width="match_parent"
            android:layout_height="wrap_content">
        </LinearLayout>

    </android.support.v4.widget.NestedScrollView>

</android.support.design.widget.CoordinatorLayout>
```

这里我们嵌套了一个垂直方向的 LinearLayout，并将 `layout_width` 设置为 `match_parent`，将 `layout_height` 设置为 `wrap_content`。

接下来在 LinearLayout 中放入具体的内容，这里我准备使用一个 TextView 来显示水果的内容详情，并将 TextView 放在一个卡片式布局当中，如下所示：

```
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    ...

    <android.support.v4.widget.NestedScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="@string/appbar_scrolling_view_behavior">
```

```

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <android.support.v7.widget.CardView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="15dp"
        android:layout_marginLeft="15dp"
        android:layout_marginRight="15dp"
        android:layout_marginTop="35dp"
        app:cardCornerRadius="4dp">

        <TextView
            android:id="@+id/fruit_content_text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="10dp" />
    </android.support.v7.widget.CardView>

</LinearLayout>
</android.support.v4.widget.NestedScrollView>

</android.support.design.widget.CoordinatorLayout>

```

这段代码也没有什么难理解的地方，都是我们学过的知识。需要注意的是，这里为了让界面更加美观，我在 CardView 和 TextView 上都加了一些边距。其中，CardView 的 marginTop 加了 35dp 的边距，这是为下面要编写的东西留出空间。

好的，这样就把水果标题栏和水果内容详情的界面都编写完了，不过我们还可以在界面上再添加一个悬浮按钮。这个悬浮按钮并不是必需的，根据具体的需求添加就可以了，如果加入的话，我们将免费获得一些额外的动画效果。

为了做出示范，我就准备在 activity_fruit.xml 中加入一个悬浮按钮了。这个界面是一个水果详情展示界面，那么我就加入一个表示评论作用的悬浮按钮吧。首先需要提前准备好一个图标，这里我放置了一张 ic_comment.png 到 drawable-xxhdpi 目录下。然后修改 activity_fruit.xml 中的代码，如下所示：

```

<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.design.widget.AppBarLayout
        android:id="@+id/appBar"
        android:layout_width="match_parent"
        android:layout_height="250dp">
        ...
    </android.support.design.widget.AppBarLayout>

```

```

<android.support.v4.widget.NestedScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior">
    ...
</android.support.v4.widget.NestedScrollView>

<android.support.design.widget.FloatingActionButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="16dp"
    android:src="@drawable/ic_comment"
    app:layout_anchor="@+id/appBar"
    app:layout_anchorGravity="bottom|end" />

</android.support.design.widget.CoordinatorLayout>

```

可以看到，这里加入了一个 FloatingActionButton，它和 AppBarLayout 以及 NestedScrollView 是平级的。FloatingActionButton 中使用 app:layout_anchor 属性指定了一个锚点，我们将锚点设置为 AppBarLayout，这样悬浮按钮就会出现在水果标题栏的区域内，接着又使用 app:layout_anchorGravity 属性将悬浮按钮定位在标题栏区域的右下角。其他一些属性都比较简单，就不再进行解释了。

好了，现在我们终于将整个 activity_fruit.xml 布局都编写完了，内容虽然比较长，但由于是分段编写的，并且每一步我都进行了详细的说明，相信你应该看得很明白吧。

界面完成了之后，接下来我们开始编写功能逻辑，修改 FruitActivity 中的代码，如下所示：

```

public class FruitActivity extends AppCompatActivity {

    public static final String FRUIT_NAME = "fruit_name";

    public static final String FRUIT_IMAGE_ID = "fruit_image_id";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fruit);
        Intent intent = getIntent();
        String fruitName = intent.getStringExtra(FRUIT_NAME);
        int fruitImageId = intent.getIntExtra(FRUIT_IMAGE_ID, 0);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        CollapsingToolbarLayout collapsingToolbar = (CollapsingToolbarLayout)
            findViewById(R.id.collapsing_toolbar);
        ImageView fruitImageView = (ImageView) findViewById(R.id.fruit_image_view);
        TextView fruitContentText = (TextView) findViewById(R.id.fruit_content_
            text);
        setSupportActionBar(toolbar);
        ActionBar actionBar = getSupportActionBar();
        if (actionBar != null) {
            actionBar.setDisplayHomeAsUpEnabled(true);
        }
        collapsingToolbar.setTitle(fruitName);
    }
}

```

```

        Glide.with(this).load(fruitImageId).into(fruitImageView);
        String fruitContent = generateFruitContent(fruitName);
        fruitContentText.setText(fruitContent);
    }

    private String generateFruitContent(String fruitName) {
        StringBuilder fruitContent = new StringBuilder();
        for (int i = 0; i < 500; i++) {
            fruitContent.append(fruitName);
        }
        return fruitContent.toString();
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case android.R.id.home:
                finish();
                return true;
        }
        return super.onOptionsItemSelected(item);
    }
}

```

FruitActivity 中的代码并不是很复杂。首先，在 `onCreate()` 方法中，我们通过 Intent 获取到传入的水果名和水果图片的资源 id，然后通过 `findViewById()` 方法拿到刚才在布局文件中定义的各个控件的实例。接着就是使用了 Toolbar 的标准用法，将它作为 ActionBar 显示，并启用 HomeAsUp 按钮。由于 HomeAsUp 按钮的默认图标就是一个返回箭头，这正是我们所期望的，因此就不用再额外设置别的图标了。

接下来开始填充界面上的内容，调用 CollapsingToolbarLayout 的 `setTitle()` 方法将水果名设置成当前界面的标题，然后使用 Glide 加载传入的水果图片，并设置到标题栏的 ImageView 上面。接着需要填充水果的内容详情，由于这只是个示例程序，并不需要什么真实的数据，所以我使用了一个 `generateFruitContent()` 方法将水果名循环拼接 500 次，从而生成了一个比较长的字符串，将它设置到了 TextView 上面。

最后，我们在 `onOptionsItemSelected()` 方法中处理了 HomeAsUp 按钮的点击事件，当点击了这个按钮时，就调用 `finish()` 方法关闭当前的活动，从而返回上一个活动。

所有工作都完成了吗？其实还差最关键的一步，就是处理 RecyclerView 的点击事件，不然的话我们根本就无法打开 FruitActivity。修改 FruitAdapter 中的代码，如下所示：

```

public class FruitAdapter extends RecyclerView.Adapter<FruitAdapter.ViewHolder> {

    ...

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        if (mContext == null) {
            mContext = parent.getContext();
        }
    }
}

```

```

View view = LayoutInflater.from(mContext).inflate(R.layout.fruit_item,
    parent, false);
final ViewHolder holder = new ViewHolder(view);
holder.cardView.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        int position = holder.getAdapterPosition();
        Fruit fruit = mFruitList.get(position);
        Intent intent = new Intent(mContext, FruitActivity.class);
        intent.putExtra(FruitActivity.FRUIT_NAME, fruit.getName());
        intent.putExtra(FruitActivity.FRUIT_IMAGE_ID, fruit.getImageId());
        mContext.startActivity(intent);
    }
});
return holder;
}
...
}

```

最关键的一步其实也是最简单的，这里我们给 CardView 注册了一个点击事件监听器，然后在点击事件中获取当前点击项的水果名和水果图片资源 id，把它们传入到 Intent 中，最后调用 `startActivity()` 方法启动 `FruitActivity`。

见证奇迹的时刻到了，现在重新运行一下程序，并点击界面上的任意一个水果，比如我点击了葡萄，效果如图 12.17 所示。

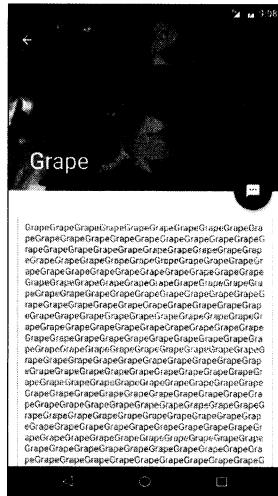


图 12.17 水果的详情展示界面

你没有看错，如此精美的界面就是我们亲手敲出来的。这个界面上的内容分为三部分，水果标题栏、水果内容详情和悬浮按钮，相信你一眼就能将它们区分出来吧。Toolbar 和水果背景图完美地融合到了一起，既保证了图片的展示空间，又不影响 Toolbar 的任何功能，那个向左的箭

头就是用来返回上一个活动的。

不过这并不是全部，真正的好戏还在后头。我们尝试向上拖动水果内容详情，你会发现水果背景图上的标题会慢慢缩小，并且背景图会产生一些错位偏移的效果，如图 12.18 所示。



图 12.18 向上拖动水果内容详情

这是由于用户想要查看水果的内容详情，此时界面的重点在具体的内容上面，因此标题栏就会自动进行折叠，从而节省屏幕空间。

继续向上拖动，直到标题栏变成完全折叠状态，效果如图 12.19 所示。



图 12.19 标题栏变成完全折叠状态

可以看到，标题栏的背景图片不见了，悬浮按钮也自动消失了，现在水果标题栏变成了一个最普通的 Toolbar。这是由于用户正在阅读具体的内容，我们需要给他们提供最充分的阅读空间。而如果这个时候向下拖动水果内容详情，就会执行一个完全相反的动画过程，最终恢复成图 12.17 的界面效果。

不知道你有没有被这个效果所感动呢？在这里，我真心地感谢 Material Design 送给我们的礼物。

12.7.2 充分利用系统状态栏空间

虽说现在水果详情展示界面的效果已经非常华丽了，但这并不代表我们不能再进一步地提升。观察一下图 12.17，你会发现水果的背景图片和系统的状态栏总有一些不搭的感觉，如果我们能将背景图和状态栏融合到一起，那这个视觉体验绝对能提升好几个档次。

只不过很可惜的是，在 Android 5.0 系统之前，我们是无法对状态栏的背景或颜色进行操作的，那个时候也还没有 Material Design 的概念。但是 Android 5.0 及之后的系统都是支持这个功能的，因此这里我们就来实现一个系统差异型的效果，在 Android 5.0 及之后的系统中，使用背景图和状态栏融合的模式，在之前的系统中使用普通的模式。

想要让背景图能够和系统状态栏融合，需要借助 `android:fitsSystemWindows` 这个属性来实现。在 CoordinatorLayout、AppBarLayout、CollapsingToolbarLayout 这种嵌套结构的布局中，将控件的 `android:fitsSystemWindows` 属性指定成 `true`，就表示该控件会出现在系统状态栏里。对应到我们的程序，那就是水果标题栏中的 ImageView 应该设置这个属性了。不过只给 ImageView 设置这个属性是没有用的，我们必须将 ImageView 布局结构中的所有父布局都设置上这个属性才可以，修改 `activity_fruit.xml` 中的代码，如下所示：

```
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true">

    <android.support.design.widget.AppBarLayout
        android:id="@+id/appBar"
        android:layout_width="match_parent"
        android:layout_height="250dp"
        android:fitsSystemWindows="true">

        <android.support.design.widget.CollapsingToolbarLayout
            android:id="@+id/collapsing_toolbar"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
            android:fitsSystemWindows="true"
            app:contentScrim="?attr/colorPrimary"
```

```

    app:layout_scrollFlags="scroll|exitUntilCollapsed">

    <ImageView
        android:id="@+id/fruit_image_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="centerCrop"
        android:fitsSystemWindows="true"
        app:layout_collapseMode="parallax" />

    ...
</android.support.design.widget.CollapsingToolbarLayout>

</android.support.design.widget.AppBarLayout>

...
</android.support.design.widget.CoordinatorLayout>

```

但是，即使我们将 `android:fitsSystemWindows` 属性都设置好了还是没有用的，因为还必须在程序的主题中将状态栏颜色指定成透明色才行。指定成透明色的方法很简单，在主题中将 `android:statusBarColor` 属性的值指定成`@android:color/transparent` 就可以了。但问题在于，`android:statusBarColor` 这个属性是从 API 21，也就是 Android 5.0 系统开始才有的，之前的系统无法指定这个属性。那么，系统差异型的功能实现就要从这里开始了。

右击 res 目录→New→Directory，创建一个 values-v21 目录，然后右击 values-v21 目录→New→Values resource file，创建一个 styles.xml 文件。接着对这个文件进行编写，代码如下所示：

```

<resources>

    <style name="FruitActivityTheme" parent="AppTheme">
        <item name="android:statusBarColor">@android:color/transparent</item>
    </style>

</resources>

```

这里我们定义了一个 FruitActivityTheme 主题，它是专门给 FruitActivity 使用的。FruitActivityTheme 的 parent 主题是 AppTheme，也就是说，它继承了 AppTheme 中的所有特性。然后我们在 FruitActivityTheme 中将状态栏的颜色指定成透明色，由于 values-v21 目录是只有 Android 5.0 及以上的系统才会去读取的，因此这么声明是没有问题的。

但是 Android 5.0 之前的系统却无法识别 FruitActivityTheme 这个主题，因此我们还需要对 values/styles.xml 文件进行修改，如下所示：

```

<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>

```

```
<item name="colorPrimaryDark">@color/colorPrimaryDark</item>
<item name="colorAccent">@color/colorAccent</item>
</style>

<style name="FruitActivityTheme" parent="AppTheme">
</style>

</resources>
```

可以看到，这里也定义了一个 FruitActivityTheme 主题，并且 parent 主题也是 AppTheme，但是它的内部是空的。因为 Android 5.0 之前的系统无法指定状态栏的颜色，因此这里什么都不用做就可以了。

最后，我们还需要让 FruitActivity 使用这个主题才可以，修改 AndroidManifest.xml 中的代码，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.materialtest">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        ...
        <activity
            android:name=".FruitActivity"
            android:theme="@style/FruitActivityTheme">
        </activity>
    </application>

</manifest>
```

这里使用 `android:theme` 属性单独给 FruitActivity 指定了 FruitActivityTheme 这个主题，这样我们就大功告成了。

现在只要是在 Android 5.0 及以上的系统运行 MaterialTest 程序，水果详情展示界面的效果就会如图 12.20 所示。

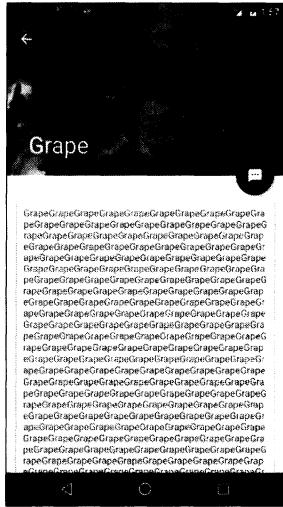


图 12.20 背景图和状态栏融合的效果

相信我，再对比一下图 12.17 的效果，这两种视觉体验绝对不是在一个档次上的。

12.8 小结与点评

学完了本章的所有知识，你有没有觉得无比兴奋呢？反正我是这么觉得的。本章我们的收获实在是太多了，从一个什么都没有的空项目，经过一章的学习，最后实现了一个功能如此丰富、界面如此华丽的应用，还有什么事情比这个更让我们有成就感吗？

本章中我们充分利用了 Design Support 库、support-v4 库、appcompat-v7 库，以及一些开源项目来实现一个高度 Material 化的应用程序，能将这些库中的相关控件熟练掌握，你的 Material Design 技术就算是合格了。

不过说到底，我仍然还是在以一个开发者的思维给你讲解 Material Design，侧重于如何去实现这些效果。而实际上，Material Design 的设计思维和设计理念才是更加重要的东西，当然这部分内容应该是 UI 设计人员去学习的，如果你也感兴趣的话，可以参考一下 Material Design 的官方文章：<https://material.google.com>。

现在你已经足足学习了 12 章的内容，对 Android 应用程序开发的理解应该比较深刻了。目前系统性的知识几乎都已经讲完了，但是还有一些零散的高级技巧在等待着你，那么就让我们赶快进入到下一章的学习当中吧。