第 33 章

33 跟踪设备的地理位置

本章，我们将创建一个名为RunTracker的应用。RunTracker利用设备的GPS，跟踪记录并显示用户的旅程。用户的旅程可能是在Big Nerd Ranch Bootcamp深林里的一次穿行，一次自驾游，或是一次海航。RunTracker应用可记录所有诸如此类的旅程。

首版RunTracker应用仅可从GPS获取位置更新，然后在屏幕上显示设备的当前位置。最终，完成版RunTracker应用可显示实时定位用户而形成的旅程路线图。

33.1 启动 RunTracker 项目

使用以下配置，创建一个新的Android应用，如图33-1所示。

图33-1 创建RunTracker应用

注意，新建应用向导界面与以往有两点不同。

- 最低SDK版本提升到了API 9级。
- 编译时使用了最新的Google API，而不是原来Android版本的API。为使用地图，我们需要Google API。

如看不到最新目标版本的Google APIs可选项，可通过Android SDK管理器进行下载。选择Window → Android SDK Manager菜单项，弹出Android SDK管理器界面，如图33-2所示，选择Google APIs，然后点击安装软件包按钮。

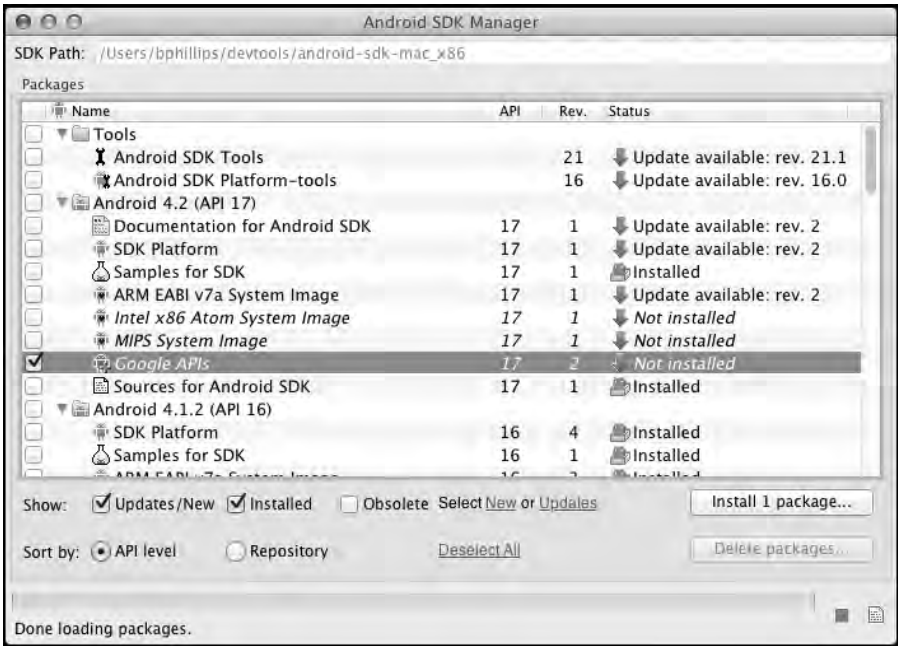

图33-2 安装适应于SDK 4.2的Google API

安装完成后，可在SDK版本选择框中看到可选的Google API。

如同其他项目的创建，通过向导创建一个名为`RunActivity`的空白activity。

33.1.1 创建RunActivity

`RunActivity`（以及RunTracker应用中的其他activity）需继承`SingleFragmentActivity`类。将SingleFragmentActivity.java和activity_fragment.xml文件分别复制进com.bignerdranch.android.runtracker类包和res/layout/目录。

然后，打开RunActivity.java，调整`RunActivity`类为`SingleFragmentActivity`的子类，如代码清单33-1所示。`RunActivity`类负责托管`RunFragment`类实例。目前`RunFragment`类还不存在，稍后我们会创建它。

代码清单33-1　初始RunActivity（RunActivity.java）

```
public class RunActivity extends SingleFragmentActivity {

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

}
```

33.1.2　创建RunFragment

接下来，完成用户界面以及初始版本RunFragment的创建。RunFragment的UI负责显示当前旅程的基本数据和地理位置，且还带有两个启停跟踪的按钮，如图33-3所示。

图33-3　初始RunTracker的UI

1. 添加字符串资源

首先，参照代码清单33-2所示代码，添加应用所需的字符串资源。打开res/values/strings.xml文件，添加以下字符串资源。有三个字符串将在本章后面使用，这里也先一并完成添加。

代码清单33-2　RunTracker的字符串资源（strings.xml）

```
<resources>
  <string name="app_name">RunTracker</string>
  <string name="started">Started:</string>
  <string name="latitude">Latitude:</string>
  <string name="longitude">Longitude:</string>
```

```
    <string name="altitude">Altitude:</string>
    <string name="elapsed_time">Elapsed Time:</string>
    <string name="start">Start</string>
    <string name="stop">Stop</string>
    <string name="gps_enabled">GPS Enabled</string>
    <string name="gps_disabled">GPS Disabled</string>
    <string name="cell_text">Run at %1$s</string>
</resources>
```

2. 获取布局文件

为保持布局文件版面清爽，这里我们将使用一个TableLayout组件。TableLayout组件包含五个TableRow和一个LinearLayout。每个TableRow又包含两个TextView：一个用于显示标题，一个用于显示运行时的数据。LinearLayout则包含两个Button。

整个布局包含的组件我们都已经学习使用过。为避免从头做起，可从本书随书代码中获取该布局（<http://www.bignerdranch.com/solutions/AndroidProgramming.zip>）。下载解压代码压缩包后，找到33_Location/RunTracker/res/layout/fragment_run.xml，然后将它复制到项目的res/layout目录中。

3. 创建RunFragment类

现在我们来创建RunFragment类。该类的初始版本仅完成了UI的显示以及对各组件的引用，如代码清单33-3所示。

代码清单33-3 RunFragment的初始代码（RunFragment.java）

```
public class RunFragment extends Fragment {
    private Button mStartButton, mStopButton;
    private TextView mStartedTextView, mLatitudeTextView,
        mLongitudeTextView, mAltitudeTextView, mDurationTextView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_run, container, false);

        mStartedTextView = (TextView)view.findViewById(R.id.run_startedTextView);
        mLatitudeTextView = (TextView)view.findViewById(R.id.run_latitudeTextView);
        mLongitudeTextView = (TextView)view.findViewById(R.id.run_longitudeTextView)
        mAltitudeTextView = (TextView)view.findViewById(R.id.run_altitudeTextView);
        mDurationTextView = (TextView)view.findViewById(R.id.run_durationTextView);

        mStartButton = (Button)view.findViewById(R.id.run_startButton);

        mStopButton = (Button)view.findViewById(R.id.run_stopButton);

        return view;
    }
}
```

运行应用，确认用户界面如图33-3所示。

33.2　地理位置与 LocationManager

应用框架搭建完毕后，下面我们来实现应用的具体功能。Android系统中的地理位置数据是由LocationManager系统服务提供的。该系统服务向所有需要地理位置数据的应用提供数据更新。更新数据的传送通常采用两种方式。

其中，使用LocationListener接口可能是最直接的一种方式。通过onLocationChanged(Location)方法，该接口提供的信息有：地理位置数据更新、状态更新以及定位服务提供者启停状态的通知消息。

如只需将地理位置数据发送给应用中的单个组件，使用LocationListener接口会很方便。例如，如果只想在RunFragment中显示地理位置数据更新，提供LocationListener接口实现给LocationManager类的requestLocationUpdates(...)或requestSingleUpdate(...)方法即可。

然而，RunTracker应用没这么简单。不管用户界面是否存在（如应用在后台运行），应用都需要持续定位用户地理位置。当然，我们也可使用stickyService，但stickyService本身复杂难用。而且对RunTracker应用来说，这种方式也不够轻量级。因此，这里我们使用自Android 2.3（GingerBread）开始引入的PendingIntentAPI。

使用PendingIntent获取地理位置数据更新，我们实际是要求LocationManager在将来某个时点帮忙发送某种类型的Intent。这样，即使应用组件，甚至是整个应用进程都销毁了，LocationManager仍会一直发送intent，直到要求它停止并按需启动新组件响应它们。利用这种优势，即使持续进行设备定位，也可以避免应用消耗过多的资源。

为管理与LocationManager的通讯，以及后续更多有关当前旅程的细节，创建一个名为RunManager的单例类，如代码清单33-4所示。

代码清单33-4　RunManager单例类的初始代码（RunManager.java）

```
public class RunManager {
    private static final String TAG = "RunManager";

    public static final String ACTION_LOCATION =
        "com.bignerdranch.android.runtracker.ACTION_LOCATION";

    private static RunManager sRunManager;
    private Context mAppContext;
    private LocationManager mLocationManager;

    // The private constructor forces users to use RunManager.get(Context)
    private RunManager(Context appContext) {
        mAppContext = appContext;
        mLocationManager = (LocationManager)mAppContext
            .getSystemService(Context.LOCATION_SERVICE);
    }

    public static RunManager get(Context c) {
        if (sRunManager == null) {
```

```
            // Use the application context to avoid leaking activities
            sRunManager = new RunManager(c.getApplicationContext());
        }
        return sRunManager;
    }

    private PendingIntent getLocationPendingIntent(boolean shouldCreate) {
        Intent broadcast = new Intent(ACTION_LOCATION);
        int flags = shouldCreate ? 0 : PendingIntent.FLAG_NO_CREATE;
        return PendingIntent.getBroadcast(mAppContext, 0, broadcast, flags);
    }

    public void startLocationUpdates() {
        String provider = LocationManager.GPS_PROVIDER;

        // Start updates from the location manager
        PendingIntent pi = getLocationPendingIntent(true);
        mLocationManager.requestLocationUpdates(provider, 0, 0, pi);
    }

    public void stopLocationUpdates() {
        PendingIntent pi = getLocationPendingIntent(false);
        if (pi != null) {
            mLocationManager.removeUpdates(pi);
            pi.cancel();
        }
    }

    public boolean isTrackingRun() {
        return getLocationPendingIntent(false) != null;
    }

}
```

注意，RunManager有三个公共实例方法。它们是RunManager的基本API。RunManager可启停地理位置更新，并让我们知晓用户旅程跟踪是否正在进行。

在startLocationUpdates()方法中，我们明确要求LocationManager通过GPS定位装置提供实时的定位数据更新。requestLocationUpdates(String, long, float, PendingIntent)方法需要四个参数，其中最小等待时间（以毫秒为单位）以及最短移动距离（以米为单位）可用于决定发送下一次定位数据更新要移动的距离和要等待的时间。

以能接受的度以及良好的用户体验为基准，这些参数应调整设置为最佳值。对于RunTracker应用来说，用户需尽可能准确地知道他们的地理位置以及曾经的足迹。这也是为什么我们硬编码了GPS提供者，并要求尽可能频繁地更新实时数据。

另一方面，对于那些只需要知道用户当前大致位置的应用，设置较大值不仅不会影响用户体验，还可减少设备电力的消耗。

地理位置更新发生时，私有的getLocationPendingIntent(boolean)方法会创建用于广播的intent。我们使用一个定制操作名识别应用内的事件。通过shouldCreate标志参数，我们告诉PendingIntent.getBroadcast(...)方法是否应该在系统内创建新PendingIntent。

最后，在isTrackingRun()实现方法中，调用getLocationPendingIntent(false)方法进

行空值判断，以确定PendingIntent是否已在操作系统中登记。

33.3　接收定位数据更新 broadcast

实现以发送broadcastIntent的方式获取地理位置数据更新后，接下来就该处理数据的接收了。无论是在前台还是在后台运行，RunTracker应用都必须能接收到更新数据。因此，最好使用登记在manifest配置文件中的独立BroadcastReceiver处理数据接收。

简单起见，创建一个LocationReceiver记录接收到的地理位置更新数据，如代码清单33-5所示。

代码清单33-5　基本LocationReceiver（LocationReceiver.java）

```
public class LocationReceiver extends BroadcastReceiver {
  private static final String TAG = "LocationReceiver";

  @Override
  public void onReceive(Context context, Intent intent) {
      // If you got a Location extra, use it
      Location loc = (Location)intent
          .getParcelableExtra(LocationManager.KEY_LOCATION_CHANGED);
      if (loc != null) {
          onLocationReceived(context, loc);
          return;
      }
      // If you get here, something else has happened
      if (intent.hasExtra(LocationManager.KEY_PROVIDER_ENABLED)) {
          boolean enabled = intent
              .getBooleanExtra(LocationManager.KEY_PROVIDER_ENABLED, false);
          onProviderEnabledChanged(enabled);
      }
  }

  protected void onLocationReceived(Context context, Location loc) {
      Log.d(TAG, this + " Got location from " + loc.getProvider() + ": "
          + loc.getLatitude() + ", " + loc.getLongitude());
  }

  protected void onProviderEnabledChanged(boolean enabled) {
      Log.d(TAG, "Provider " + (enabled ? "enabled" : "disabled"));
  }

}
```

在onReceive(Context, Intent)实现方法中，LocationManager打包了附加额外信息的intent。LocationManager.KEY_LOCATION_CHANGED键值可指定一个表示最新更新的Location实例。如果接收到地理位置数据更新，则调用onLocationReceived(Context, Location)方法，记录下服务提供者名字以及相应的经纬度数据。

LocationManager也可以传入一个具有KEY_PROVIDER_ENABLED键值的布尔类型extra信息。如果收到这样的extra信息，则可调用onProviderEnabled(boolean)方法记录下它。最终，我们将继承LocationReceiver类，并利用onLocationReceived(...)以及onProviderEnabled(...)方法

执行更多有用的任务。

在RunTracker应用的manifest配置文件中，完成LocationReceiver类的声明登记。顺便也完成ACCESS_FINE_LOCATION使用权限以及GPS硬件uses-feature节点的添加，如代码清单33-6所示。

代码清单33-6 添加定位使用权限（AndroidManifest.xml）

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.bignerdranch.android.runtracker"
  android:versionCode="1"
  android:versionName="1.0">

  <uses-sdk android:minSdkVersion="9" android:targetSdkVersion="15" />
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
  <uses-feature android:required="true"
    android:name="android.hardware.location.gps"/>

  <application android:label="@string/app_name"
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:theme="@style/AppTheme">
    <activity android:name=".RunActivity"
      android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <receiver android:name=".LocationReceiver"
      android:exported="false">
      <intent-filter>
        <action android:name="com.bignerdranch.android.runtracker.ACTION_LOCATION"/>
      </intent-filter>
    </receiver>

  </application>

</manifest>
```

现在，我们已完成了请求和接收地理位置数据更新的实现代码。最后应该是提供用户界面以启停定位服务，并显示接收的定位数据。

33.4 使用定位数据刷新 UI 显示

为验证定位数据更新是否能正常工作，在与RunManager通讯的RunFragment类中，为Start和Stop按钮提供单击事件监听器。另外再添加一个简单的updateUI()方法调用。如代码清单33-7所示。

代码清单33-7 启停定位数据更新服务（RunFragment.java）

```
public class RunFragment extends Fragment {

    private RunManager mRunManager;
```

```
    private Button mStartButton, mStopButton;
    private TextView mStartedTextView, mLatitudeTextView,
        mLongitudeTextView, mAltitudeTextView, mDurationTextView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
        mRunManager = RunManager.get(getActivity());
    }

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

        mStartButton = (Button)view.findViewById(R.id.run_startButton);
        mStartButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mRunManager.startLocationUpdates();
                updateUI();
            }
        });

        mStopButton = (Button)view.findViewById(R.id.run_stopButton);
        mStopButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mRunManager.stopLocationUpdates();
                updateUI();
            }
        });

        updateUI();


        return view;
    }

    private void updateUI() {
        boolean started = mRunManager.isTrackingRun();

        mStartButton.setEnabled(!started);
        mStopButton.setEnabled(started);
    }
}
```

完成以上操作后，可再次运行RunTracker应用，并通过LogCat窗口，观察位置数据更新的接收。为获得理想结果，可使用DDMS视图里的模拟器控制，发送模拟位置更新给模拟器，或携带设备外出，等待GPS的刷新。首次获得数据更新通常需要几分钟的时间。在LogCat窗口滚动查看位置数据细节信息。如嫌麻烦，也可修改代码并提供关键字信息，然后在LogCat窗口中使用过滤器，锁定查看目标信息。

使用LogCat窗口进行定位跟踪查看，并不是特别得人性化。为在用户界面显示定位数据更新的细节信息，可在`RunFragment`类中继承`LocationReceiver`类，实现保存`Location`数据，并更

新UI。有了存储在新的Run实例中的这些数据，我们就可显示当前旅程的开始日期和持续时间。实现一个简单的保存开始日期的Run类，添加计算持续时间的方法，并将持续时间格式化显示为字符串信息，如代码清单33-8所示。

代码清单33-8 基本Run类（Run.java）

```
public class Run {
    private Date mStartDate;

    public Run() {
        mStartDate = new Date();
    }

    public Date getStartDate() {
        return mStartDate;
    }

    public void setStartDate(Date startDate) {
        mStartDate = startDate;
    }

    public int getDurationSeconds(long endMillis) {
        return (int)((endMillis - mStartDate.getTime()) / 1000);
    }

    public static String formatDuration(int durationSeconds) {
        int seconds = durationSeconds % 60;
        int minutes = ((durationSeconds - seconds) / 60) % 60;
        int hours = (durationSeconds - (minutes * 60) - seconds) / 3600;
        return String.format("%02d:%02d:%02d", hours, minutes, seconds);
    }

}
```

现在更新RunFragment类代码，以关联使用新建的Run类，如代码清单33-9所示。

代码清单33-9 显示地理位置更新数据（RunFragment.java）

```
public class RunFragment extends Fragment {

    private BroadcastReceiver mLocationReceiver = new LocationReceiver() {

        @Override
        protected void onLocationReceived(Context context, Location loc) {
            mLastLocation = loc;
            if (isVisible())
                updateUI();
        }

        @Override
        protected void onProviderEnabledChanged(boolean enabled) {
            int toastText = enabled ? R.string.gps_enabled : R.string.gps_disabled;
            Toast.makeText(getActivity(), toastText, Toast.LENGTH_LONG).show();
        }

    };

    private RunManager mRunManager;
```

```
    private Run mRun;
    private Location mLastLocation;
    private Button mStartButton, mStopButton;

    ...

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

        mStartButton = (Button)view.findViewById(R.id.run_startButton);
        mStartButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mRunManager.startLocationUpdates();
                mRun = new Run();
                updateUI();
            }
        });

        ...
    }

    @Override
    public void onStart() {
        super.onStart();
        getActivity().registerReceiver(mLocationReceiver,
                new IntentFilter(RunManager.ACTION_LOCATION));
    }

    @Override
    public void onStop() {
        getActivity().unregisterReceiver(mLocationReceiver);
        super.onStop();
    }

    private void updateUI() {
        boolean started = mRunManager.isTrackingRun();

        if (mRun != null)
            mStartedTextView.setText(mRun.getStartDate().toString());

        int durationSeconds = 0;
        if (mRun != null && mLastLocation != null) {
            durationSeconds = mRun.getDurationSeconds(mLastLocation.getTime());
            mLatitudeTextView.setText(Double.toString(mLastLocation.getLatitude()));
            mLongitudeTextView.setText(Double.toString(mLastLocation.getLongitude()));
            mAltitudeTextView.setText(Double.toString(mLastLocation.getAltitude()));
        }
        mDurationTextView.setText(Run.formatDuration(durationSeconds));

        mStartButton.setEnabled(!started);
        mStopButton.setEnabled(started);
    }

}
```

以上代码包含了很多重要的实现细节信息。首先是添加的Run类和最后接收到的Location

数据的实例变量。其次是在updateUI()方法中用于更新用户界面的后台数据。定位更新服务一启动，Run类就立即完成初始化操作。

另外，我们也创建了一个存储在mLocationReceiver变量中的LocationReceiver匿名类，用于保存接收到的定位数据并更新UI显示，此外，在GPS服务提供者启动或停止时，还会有Toast提示消息显示。

最后，使用onStart()和onStop()实现方法结合用户可见的fragment，对receiver进行登记和撤销登记。同时也建议在onCreate(Bundle)和onDestroy()方法中完成这些任务，这样即使是应用用户界面退出前台而保持位置数据的接收时，mLastLocation变量也可一直包含最新的地理位置更新数据。

再次运行RunTracker应用，可看到显示在用户界面上的真实地理位置数据。

33.5 快速定位：最近一次地理位置

有时，用户并不想耗时几分钟去等待设备与神秘的太空定位卫星成功通讯并返回自己的地理位置信息。幸运的是，可利用LocationManager的最近一次地理位置（适应于各种定位方式，如GPS、WIFI网络、手机基站等），来消除这种等待的煎熬。

由于我们只使用GPS定位服务，因此使用最近一次地理位置信息最合适不过了，具体代码实现也比较简单直接，如代码清单33-10所示。唯一棘手的是，将最近一次地理位置信息取回用户界面。为完成该任务，可把自己看作是LocationManager，然后发送一个Intent。

代码清单33-10 获取最近一次地理位置（RunManager.java）

```
public void startLocationUpdates() {
    String provider = LocationManager.GPS_PROVIDER;

    // Get the last known location and broadcast it if you have one
    Location lastKnown = mLocationManager.getLastKnownLocation(provider);
    if (lastKnown != null) {
        // Reset the time to now
        lastKnown.setTime(System.currentTimeMillis());
        broadcastLocation(lastKnown);
    }

    // Start updates from the location manager
    PendingIntent pi = getLocationPendingIntent(true);
    mLocationManager.requestLocationUpdates(provider, 0, 0, pi);
}

private void broadcastLocation(Location location) {
    Intent broadcast = new Intent(ACTION_LOCATION);
    broadcast.putExtra(LocationManager.KEY_LOCATION_CHANGED, location);
    mAppContext.sendBroadcast(broadcast);
}
```

注意，代码中重置了从GPS定位服务提供者获取的地理位置时间戳。这可能不是用户想要的，是否需要重置将作为一个练习留给读者去解决。

也可向LocationManager请求来自不同定位服务提供者的最近一次地理位置，或使用getA-

llProviders()方法获知协同工作的定位服务提供者。如遍历查看所有最近一次地理位置信息，应查看其准确性，并确定是否为比较新的时间戳。如较为久远，可不采用这些数据信息。

33.6　在物理和虚拟设备上测试地理位置定位

即使对热爱户外运动的开发者而言，测试类似RunTracker的应用也充满了挑战。测试应能够获得准确跟踪的地理位置信息并保存下来。如果只是在附近随便逛逛，或是请朋友用自行车载着自己和测试设备在周边地区低速穿行，那么测试效果往往难以得到保证。

为解决这个问题，我们可发送地理位置测试数据给LocationManager，从而实现设备在别处的模拟。

要达到这种效果，最简单的方式是使用DDMS中的模拟器控制窗口。虽然这种方式只适用于虚拟设备，但我们既可以手动一次指定一处地理位置，也可使用GPX或KML文件模拟提供一系列不时变换的地理位置。

要在物理设备上测试地理位置定位，我们还需多花点功夫，但完全是有可能实现的。实现的基本过程如下。

- 获取ACCESS_MOCK_LOCATION权限。
- 使用LocationManager.addTestProvider(...)方法添加虚拟定位服务提供者。
- 使用setTestProviderEnabled(...)方法启动虚拟定位服务提供者。
- 使用setTestProviderStatus(...)方法设置初始状态。
- 使用setTestProviderLocation(...)方法发布地理位置数据。
- 使用removeTestProvider(...)方法移除虚拟定位服务提供者。

挺复杂是吗？没关系，我们早有准备。Big Nerd Ranch有一个简单的TestProvider项目，读者可下载安装到设备上，然后运行并配置管理虚拟定位服务提供者，以实现测试需求。

从Github网站https://github.com/bignerdranch/AndroidCourseResources下载TestProvider项目，然后将它导入Eclipse工具。

为使用TestProvider，我们还需填加一些代码到RunTracker项目。参照代码清单33-11，完成RunManager类的更新。

代码清单33-11　使用虚拟定位服务提供者（RunManager.java）

```
public class RunManager {
    private static final String TAG = "RunManager";

    public static final String ACTION_LOCATION =
        "com.bignerdranch.android.runtracker.ACTION_LOCATION";

    private static final String TEST_PROVIDER = "TEST_PROVIDER";

    private static RunManager sRunManager;
    private Context mAppContext;
    private LocationManager mLocationManager;
```

```
...

public void startLocationUpdates() {
    String provider = LocationManager.GPS_PROVIDER;
    // If you have the test provider and it's enabled, use it
    if (mLocationManager.getProvider(TEST_PROVIDER) != null &&
            mLocationManager.isProviderEnabled(TEST_PROVIDER)) {
        provider = TEST_PROVIDER;
    }
    Log.d(TAG, "Using provider " + provider);

    // get the last known location and broadcast it if you have one
    Location lastKnown = mLocationManager.getLastKnownLocation(provider);
    if (lastKnown != null) {
        // Reset the time to now
        lastKnown.setTime(System.currentTimeMillis());
        broadcastLocation(lastKnown);
    }
```

另外，TestProvider要能正常工作，我们还需打开Settings应用，在Developer options菜单中开启Allow mock locations设置，如图33-4所示。

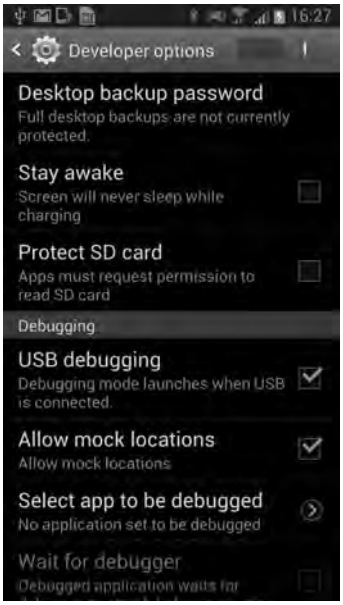

图33-4 允许使用地理位置模拟数据

设置完成后，在设备上运行TestProvider应用，点击按钮模拟地理位置数据更新。

然后运行RunTracker应用，查看接收到的模拟数据。（提示：模拟地理位置为美国的乔治亚州亚特兰大市。）测试任务完成后，记得关闭虚拟定位服务提供者，否则设备会搞不清楚真实的地理位置。