第 36 章

使用地图 36

RunTracker应用的下一项实施任务是在地图上显示用户的旅程路线。使用最新的Google Maps API（版本2.0），可轻松实现该任务。本章，我们将新建一个`RunMapFragment`类，用以展示用户的旅程路线以及表明旅程起点和终点的交互式地图标注。

不过，在开始应用开发之前，我们必须首先设置项目使用地图API。

36.1 添加 Maps API 给 RunTracker 应用

Maps API（2.0版本）由Google Play services SDK提供，并且对开发环境和应用均有一定的要求。

36.1.1 使用物理设备测试地图

Google Play services SDK（或者说Maps API）要求物理设备最低运行Android 2.2版本系统，并且安装了Google Play store。 Maps API不支持虚拟设备。

36.1.2 安装使用 Google Play services SDK

要在项目中使用Maps API，首先需安装Google Play services SDK功能扩展包，并配置其库项目供应用使用。按照以下步骤应能完成SDK的安装和配置，如遇问题，请访问http://developer.android.com/google/play-services/网页获取最新安装与配置信息。

(1) 打开Android SDK管理器，在安装包选择区域的Extras部分，选择安装Google Play services功能扩展。安装完成后，可在Android SDK目录的extras/google/ google_play_services目录看到安装文件。

(2) 在Ecipse中，使用File → Import... → Existing Android Code Into Workspace菜单项，将Google Play services库项目副本导入到工作区。库项目存放在Google Play services内extra目录中的libproject/google-play-services_lib目录下。在项目导入向导页面，记得选择Copy projects into workspace选项，以便在应用中使用库项目的副本。

(3) 打开RunTracker项目的Properties窗口，选择左边的Android选项，并在右边窗口的Library区域添加库项目引用。点击Add…按钮，然后选择google-play-services_lib库项目。

36.1.3　获取 Google Maps API key

要使用Maps API，还需为应用创建一个API key。此过程需要好几步操作才能完成。具体操作可访问网页https://developers.google.com/maps/documentation/android/start，然后按照操作指令完成API key的获取。

36.1.4　更新 RunTracker 应用的 manifest 配置文件

Google Play services和Maps API要能正常工作，还需在manifest配置文件中完成使用权限、API key等配置。参照代码清单36-1，将加亮部分的XML代码添加到RunTracker的manifest配置文件中。注意，以MAPS_RECEIVE结尾的使用权限，需添加RunTracker应用的全路径包名。

既然已打开配置文件，那就顺便完成RunMapActivity（稍后会创建）的配置。

代码清单36-1　Maps API使用配置（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" />

  <permission
    android:name="com.bignerdranch.android.runtracker.permission.MAPS_RECEIVE"
    android:protectionLevel="signature"/>
  <uses-permission
    android:name="com.bignerdranch.android.runtracker.permission.MAPS_RECEIVE"/>

  <uses-permission android:name="android.permission.INTERNET"/>
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
  <uses-permission
    android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>
  <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>

  <uses-feature android:required="true"
    android:name="android.hardware.location.gps" />
  <uses-feature
    android:required="true"
    android:glEsVersion="0x00020000"/>
  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme">
    <activity android:name=".RunListActivity"
      android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
```

```
        <activity android:name=".RunActivity"
          android:label="@string/app_name" />
        <activity android:name=".RunMapActivity"
          android:label="@string/app_name" />
        <receiver android:name=".TrackingLocationReceiver"
          android:exported="false">
          <intent-filter>
            <action
              android:name="com.bignerdranch.android.runtracker.ACTION_LOCATION"/>
           </intent-filter>
        </receiver>
        <meta-data
          android:name="com.google.android.maps.v2.API_KEY"
          android:value="your-maps-API-key-here"/>
      </application>

    </manifest>
```

36.2 在地图上显示用户的地理位置

明确应用的需求后，现在我们来实现地图的展示。MapFragment和SupportMapFragment是Maps API的两个类。通过创建这两个类的子类，我们可方便地配置MapView和GoogleMap（与MapView关联的模型对象）。

参照代码清单36-2，以SupportMapFragment为父类，创建RunMapFragment类。

代码清单36-2 基本RunMapFragment类（RunMapFragment.java）

```
public class RunMapFragment extends SupportMapFragment {
    private static final String ARG_RUN_ID = "RUN_ID";

    private GoogleMap mGoogleMap;

    public static RunMapFragment newInstance(long runId) {
        Bundle args = new Bundle();
        args.putLong(ARG_RUN_ID, runId);
        RunMapFragment rf = new RunMapFragment();
        rf.setArguments(args);
        return rf;
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
        View v = super.onCreateView(inflater, parent, savedInstanceState);

        // Stash a reference to the GoogleMap
        mGoogleMap = getMap();
        // Show the user's location
        mGoogleMap.setMyLocationEnabled(true);

        return v;
    }

}
```

如同RunFragment类中的处理，RunMapFragment的newInstance(long)方法使用传入的旅程ID，为自身的新实例设置了附加参数。稍后，在获取地理位置信息列表时会用到该附加参数。

onCreateView(...)实现方法调用超类版本的同名方法，获取并返回了一个视图。不用疑惑，我们这么做主要是为了初始化RunMapFragment的GoogleMap实例。GoogleMap是与MapView绑定的模型对象，可利用它对地图进行各种附加配置。比如，在当前初始版本的RunMapFragment类中，我们就调用了setMyLocationEnabled(boolean)方法，以允许用户在地图上查看并导航至自身地理位置。

为托管新建的RunMapFragment，使用代码清单36-3所示代码，创建一个简单的RunMapActivity类（已在manifest配置文件中添加了引用）。

代码清单36-3　托管RunMapFragment的新建RunMapActivity（RunMapActivity.java）

```
public class RunMapActivity extends SingleFragmentActivity {
    /** A key for passing a run ID as a long */
    public static final String EXTRA_RUN_ID =
        "com.bignerdranch.android.runtracker.run_id";

    @Override
    protected Fragment createFragment() {
        long runId = getIntent().getLongExtra(EXTRA_RUN_ID, -1);
        if (runId != -1) {
            return RunMapFragment.newInstance(runId);
        } else {
            return new RunMapFragment();
        }
    }

}
```

现在需编写代码，在有可用的旅程记录时，从RunFragment中启动RunMapActivity。为此，我们还需在布局页面上添加触发按钮。但依惯例，首先应完成按钮所需字符串资源的添加，如代码清单36-4所示。

代码清单36-4　UI显示所需的字符串资源（res/values/strings.xml）

```
  <string name="new_run">New Run</string>
  <string name="map">Map</string>
  <string name="run_start">Run Start</string>
  <string name="run_started_at_format">Run started at %s</string>
  <string name="run_finish">Run Finish</string>
  <string name="run_finished_at_format">Run finished at %s</string>
</resources>
```

更新RunFragment的布局，完成Map按钮的添加，如代码清单36-5所示。

代码清单36-5　添加Map按钮（fragment_run.xml）

```
    <Button android:id="@+id/run_stopButton"
      android:layout_width="0dp"
      android:layout_height="wrap_content"
```

```
        android:layout_weight="1"
        android:text="@string/stop"
        />
      <Button android:id="@+id/run_mapButton"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/map"
        />
    </LinearLayout>
</TableLayout>
```

现在，我们来配置RunFragment与新按钮相关联，并正确设置Map按钮的启用或禁用状态，如代码清单36-6所示。

代码清单36-6 配置关联Map按钮（RunFragment.java）

```
public class RunFragment extends Fragment {
    ...

    private RunManager mRunManager;

    private Run mRun;
    private Location mLastLocation;

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

    ...

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

        mMapButton = (Button)view.findViewById(R.id.run_mapButton);
        mMapButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent i = new Intent(getActivity(), RunMapActivity.class);
                i.putExtra(RunMapActivity.EXTRA_RUN_ID, mRun.getId());
                startActivity(i);
            }
        });
        updateUI();

        return view;
    }

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

    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()));
            mMapButton.setEnabled(true);
        } else {
            mMapButton.setEnabled(false);
        }
        mDurationTextView.setText(Run.formatDuration(durationSeconds));

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

完成以上工作后，RunTracker应能显示地图及用户当前的地理位置了。运行应用，加载一项旅程，然后点击Map按钮。如果设备能够连接网络，且位于Big Nerd Ranch办公室附近，则可看到类似图36-1的应用运行效果图。

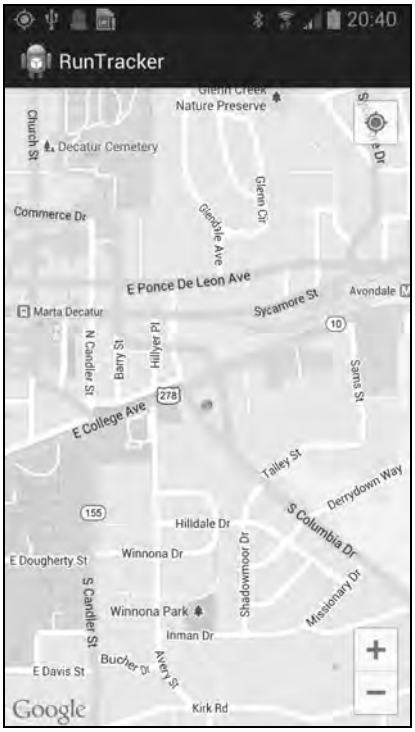

图36-1　RunTracker应用的定位功能

36.3　显示旅程路线

接下来的任务是显示用户的旅程路线。有了Maps API，这很简单，但我们需要获取地理位置

列表以便知道如何画出这条线。分别在RunDatabaseHelper和RunManager类中添加一个方法，以提供一个封装地理位置信息的LocationCursor，如代码清单36-7所示。

代码清单36-7 查询单个旅程的地理记录（RunDatabaseHelper.java）

```
public LocationCursor queryLocationsForRun(long runId) {
    Cursor wrapped = getReadableDatabase().query(TABLE_LOCATION,
            null,
            COLUMN_LOCATION_RUN_ID + " = ?", // Limit to the given run
            new String[]{ String.valueOf(runId) },
            null, // group by
            null, // having
            COLUMN_LOCATION_TIMESTAMP + " asc"); // order by timestamp
    return new LocationCursor(wrapped);
}
```

queryLocationsForRun(long)方法与前面SQLite那章的queryLastLocationForRun (long)方法十分类似，不过该方法首先以升序对地理位置进行排序，然后再返回排序后的全部结果。

我们也可在RunManager中使用这个方法，为RunMapFragment提供可以增色UI的数据，如代码清单36-8所示。

代码清单36-8 查询旅程的地理位置记录（第二部分）（RunManager.java）

```
public LocationCursor queryLocationsForRun(long runId) {
    return mHelper.queryLocationsForRun(runId);
}
```

现在，RunMapFragment可使用该方法加载地理位置数据了。自然地，我们会想到使用Loader来避免在主线程上执行数据库查询。因此，新建一个LocationListCursorLoader类来处理这项任务，如代码清单36-9所示。

代码清单36-9 地理位置记录loader（LocationListCursorLoader.java）

```
public class LocationListCursorLoader extends SQLiteCursorLoader {
    private long mRunId;

    public LocationListCursorLoader(Context c, long runId) {
        super(c);
        mRunId = runId;
    }

    @Override
    protected Cursor loadCursor() {
        return RunManager.get(getContext()).queryLocationsForRun(mRunId);
    }
}
```

然后，在RunMapFragment中，使用刚才新建的loader加载地理位置数据，如代码清单36-10所示。

代码清单36-10 在RunMapFragment中加载地理位置数据（RunMapFragment.java）

```
public class RunMapFragment extends SupportMapFragment
    implements LoaderCallbacks<Cursor> {
```

```
    private static final String ARG_RUN_ID = "RUN_ID";
    private static final int LOAD_LOCATIONS = 0;

    private GoogleMap mGoogleMap;
    private LocationCursor mLocationCursor;

    ...

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

        // Check for a Run ID as an argument, and find the run
        Bundle args = getArguments();
        if (args != null) {
            long runId = args.getLong(ARG_RUN_ID, -1);
            if (runId != -1) {
                LoaderManager lm = getLoaderManager();
                lm.initLoader(LOAD_LOCATIONS, args, this);
            }
        }
    }

    ...

    @Override
    public Loader<Cursor> onCreateLoader(int id, Bundle args) {
        long runId = args.getLong(ARG_RUN_ID, -1);
        return new LocationListCursorLoader(getActivity(), runId);
    }

    @Override
    public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {
        mLocationCursor = (LocationCursor)cursor;
    }

    @Override
    public void onLoaderReset(Loader<Cursor> loader) {
        // Stop using the data
        mLocationCursor.close();
        mLocationCursor = null;
    }

}
```

以上代码可以看到，RunMapFragment类的 LocationCursor实例变量会持有一组地理位置数据。稍后，这些数据将会用于在地图上绘制旅程路线。

数据加载完毕后，onLoaderReset(Loader<Cursor>)实现方法负责关闭cursor并清除对其的引用。正常情况下，应在用户离开地图fragment界面后，LoaderManager停止loader时调用该方法。

这样的处理逻辑简单清晰，但该方法无法处理设备旋转时的cursor关闭。而这正是我们想要的结果。

现在我们来处理本章的核心部分：在GoogleMap上显示旅程路线。添加一个配置旅程路线的updateUI()方法，然后在onLoadFinished(...)方法中调用它，如代码清单36-11所示。

代码清单36-11 在地图上显示旅程路线（RunMapFragment.java）

```
private void updateUI() {
    if (mGoogleMap == null || mLocationCursor == null)
        return;

    // Set up an overlay on the map for this run's locations
    // Create a polyline with all of the points
    PolylineOptions line = new PolylineOptions();
    // Also create a LatLngBounds so you can zoom to fit
    LatLngBounds.Builder latLngBuilder = new LatLngBounds.Builder();
    // Iterate over the locations
    mLocationCursor.moveToFirst();
    while (!mLocationCursor.isAfterLast()) {
        Location loc = mLocationCursor.getLocation();
        LatLng latLng = new LatLng(loc.getLatitude(), loc.getLongitude());
        line.add(latLng);
        latLngBuilder.include(latLng);
        mLocationCursor.moveToNext();
    }
    // Add the polyline to the map
    mGoogleMap.addPolyline(line);
    // Make the map zoom to show the track, with some padding
    // Use the size of the current display in pixels as a bounding box
    Display display = getActivity().getWindowManager().getDefaultDisplay();
    // Construct a movement instruction for the map camera
    LatLngBounds latLngBounds = latLngBuilder.build();
    CameraUpdate movement = CameraUpdateFactory.newLatLngBounds(latLngBounds,
            display.getWidth(), display.getHeight(), 15);
    mGoogleMap.moveCamera(movement);
}

@Override
public Loader<Cursor> onCreateLoader(int id, Bundle args) {
    long runId = args.getLong(ARG_RUN_ID, -1);
    return new LocationListCursorLoader(getActivity(), runId);
}

@Override
public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {
    mLocationCursor = (LocationCursor)cursor;
    updateUI();
}
```

updateUI()方法使用了Maps API中的一些支持类和方法。首先，它创建了PolylineOptions实例和LatLngBounds.Builder实例。PolylineOptions实例的作用是创建用于地图的线条；而LatLngBounds.Builder实例则用于在所有数据都收集完时，创建一个供地图缩放的包围框。

然后，遍历LocationCursor，并利用坐标为每个Location创建一个LatLng。在处理cursor中的下一条数据前，将LatLng添加到PolylineOptions实例和LatLngBounds实例中。

按照以上方式处理完所有的地理位置数据后，调用GoogleMap的addPolyline(PolylineOptions)方法，将线条绘制到地图上。

接接下来的任务是缩放地图以显示整个旅程路线。“camera”负责处理在地图上移动的操作。

打包在CameraUpdate实例中的移动指令是用来调整camera的，我们将其传入moveCamera(CameraUpdate)方法实现在地图上移动。而CameraUpdateFactory的newLatLngBounds(LatLngBounds, int, int, int)方法则负责创建一个实例，将camera的移动限制在刚才添加的包围框内。

这这里，以像素为单位，我们传入当前屏幕尺寸作为地图的近似大小尺寸。为使二者完美契合，我们指定了一个15单位像素的填充。相较于newLatLngBounds(LatLngBounds, int, int, int)方法，还有一个简单版本的newLatLngBounds(LatLngBounds, int)方法可供使用。但是，如果在MapView通过绘制过程完成自身尺寸确定之前就调用该简单版本的方法，它会抛出IllegalStateException异常。既然无法保证调用updateUI()方法时MapView能否准备就绪，因此只能使用屏幕尺寸作为安全的假想值了。

完成以上全部工作后，再次运行RunTracker应用，我们应能看到一条代表旅程路线的美丽黑线。如需进一步美化线条，可查阅PolylineOptions类的更多相关信息加以实现。

36.4　为旅程添加开始和结束地图标注

在结束本章全部开发工作之前，我们再来完成一件锦上添花的任务：添加地图标注，以醒目显示旅程的开始和结束点。另外，再为地图标注配上相关文字信息，实现在用户点击地图标注时弹出一个信息窗。

在updateUI()方法中添加以下加亮代码，如代码清单36-12所示。

代码清单36-12　附带信息的开始和结束地图标注（RunMapFragment.java）

```
private void updateUI() {
    if (mGoogleMap == null || mLocationCursor == null)
        return;

    // Set up an overlay on the map for this run's locations
    // Create a polyline with all of the points
    PolylineOptions line = new PolylineOptions();
    // Also create a LatLngBounds so you can zoom to fit
    LatLngBounds.Builder latLngBuilder = new LatLngBounds.Builder();
    // Iterate over the locations
    mLocationCursor.moveToFirst();
    while (!mLocationCursor.isAfterLast()) {
        Location loc = mLocationCursor.getLocation();
        LatLng latLng = new LatLng(loc.getLatitude(), loc.getLongitude());

        Resources r = getResources();

        // If this is the first location, add a marker for it
        if (mLocationCursor.isFirst()) {
            String startDate = new Date(loc.getTime()).toString();
            MarkerOptions startMarkerOptions = new MarkerOptions()
                .position(latLng)
                .title(r.getString(R.string.run_start))
                .snippet(r.getString(R.string.run_started_at_format, startDate));
            mGoogleMap.addMarker(startMarkerOptions);
```

```
        } else if (mLocationCursor.isLast()) {
            // If this is the last location, and not also the first, add a marker
            String endDate = new Date(loc.getTime()).toString();
            MarkerOptions finishMarkerOptions = new MarkerOptions()
                .position(latLng)
                .title(r.getString(R.string.run_finish))
                .snippet(r.getString(R.string.run_finished_at_format, endDate));
            mGoogleMap.addMarker(finishMarkerOptions);
        }

        line.add(latLng);
        latLngBuilder.include(latLng);
        mLocationCursor.moveToNext();
    }
    // Add the polyline to the map
    mGoogleMap.addPolyline(line);
```

以上代码中，我们为每个旅程的开始和结束点都创建了一个MarkerOptions实例，用于保存位置、标题以及位置简介文字。用户点击地图标注时，标题和位置简介文字可显示在弹出的信息窗中。

这里，我们使用了默认的地图标注图标。如有需要，也可使用icon(BitmapDescriptor)和BitmapDescriptorFactory方法，创建不同颜色或具有定制图像的地图标注。这两个方法还提供有多种可选的标准颜色（或称色调）。

运行RunTracker应用，来段旅行进行测试吧！祝旅途愉快！

36.5 挑战练习：实时数据更新

当前，RunMapFragment会在地图上显示过去某个时间点的用户旅程路线。一个完美的旅程跟踪记录应用应能够快速实时地更新显示旅程路线。在RunMapFragment中，使用LocationReceiver子类及时响应新的地理位置，并重新绘制地图上的旅程路线。在重新绘制之前，别忘了先清除旧的旅程路线以及对应的地图标注。