# TODO

<https://blog.csdn.net/yhaolpz/article/details/51346242?utm_source=blogxgwz3>

https://blog.csdn.net/omnispace/article/details/51922816?utm\_source=blogxgwz1

LocationManagerService API的Hook解析

<https://blog.csdn.net/u012439416/article/details/76360375>

Android 系统中 Location Service 的实现与架构

https://www.ibm.com/developerworks/cn/opensource/os-cn-android-location/index.html

# API接口

表 1. android.location 包中的类和接口

| **名称** | **类型** | **说明** |
| --- | --- | --- |
| GpsStatus.Listener | 接口 | 用于接受 GPS 状态改变时的通知。 |
| GpsStatus.NmeaListener | 接口 | 用于接受 Nmea（为海用电子设备制定的格式）信息。 |
| LocationListener | 接口 | 用于接受位置信息改变时的通知。 |
| Address | 类 | 用于描述地址信息。 |
| Criteria | 类 | 用于选择 LocationProvider。 |
| Geocoder | 类 | 用于处理地理位置的编码。 |
| GpsSatellite | 类 | 用于描述 GPS 卫星的状态。 |
| GpsStatus | 类 | 用于描述 GPS 设备的状态。 |
| Location | 类 | 用于描述地理位置信息，包括经度，纬度，海拔，方向等信息。 |
| LocationManager | 类 | 用于获取和调用定位服务。 |
| LocationProvider | 类 | 描述 Location Provider 的超类，Location Provider 是真正用来获取位置信息的组件。Location Provider 的实现主要可以分为两类：一种依赖于 GPS 设备，另一种依赖网络状态。 |

## 使用定位服务提供的 API

Ds

|  |
| --- |
| public class MainActivity extends Activity {     // 测试使用的日志 Tag   private static final String TAG = "LocationService API Demo";   // 将在 onCreate 中被初始化   private LocationManager locationManager;   // 接受位置更新的监听器   protected final LocationListener locationListener =                                   new            LocationListener() {       // 当位置发生变化时，输出位置信息     public void onLocationChanged(Location location) {       Log.d(TAG, "Location changed to: " + getLocationInfo(location));     }       public void onProviderDisabled(String provider) {       Log.d(TAG, provider + " disabled.");     }       public void onProviderEnabled(String provider) {       Log.d(TAG, provider + " enabled.");     }       public void onStatusChanged(String provider, int status,  Bundle extras){       Log.d(TAG, provider + " status changed.");     }   };     @Override   protected void onCreate(Bundle savedInstanceState) {     super.onCreate(savedInstanceState);     setContentView(R.layout.activity\_main);     // 获取 LocationManager     locationManager = (LocationManager)getSystemService(LOCATION\_SERVICE);   }     @Override   protected void onResume() {     super.onResume();     // 指定一个 Provider     String currentProvider = LocationManager.NETWORK\_PROVIDER;     Log.d(TAG, "CurrentProvider: " + currentProvider);     // 获取 Provider 最后一个记录的地址信息     Location lastKnownLocation = locationManager  .getLastKnownLocation(currentProvider);     if (lastKnownLocation != null) {       Log.d(TAG, "LastKnownLocation: "     + getLocationInfo(lastKnownLocation));     } else {       Log.d(TAG, "Last Location Unkown!");     }     // 注册监听器接受位置更新     locationManager.requestLocationUpdates(currentProvider, 0, 0,  locationListener);   }     @Override   protected void onPause() {     super.onPause();     // 移除监听器     locationManager.removeUpdates(locationListener);     Log.d(TAG, "LocationListener: " + locationListener + " removed.");   }     /\*\*    \* 将 Location 对象转换成字符串形式方便显示    \*    \* @param location    \*            Location 对象    \* @return 字符串形式的表示    \*/   private String getLocationInfo(Location location) {     String info = "";     info += "Longitude:" + location.getLongitude();     info += ", Latitude:" + location.getLatitude();     if (location.hasAltitude()) {       info += ", Altitude:" + location.getAltitude();     }     if (location.hasBearing()) {       info += ", Bearing:" + location.getBearing();     }     return info;   }  } |

这段代码的说明如下： 在 Activity 显示的时候首先尝试获取通过网络定位的 Location Provider 记录的最后一次定位信息，然后在系统中注册一个监听器来监听位置信息的变更，这里的 API 都是使用定位服务最常用的。

# 源码解析

## 定位服务的实现架构图

整个定位服务的架构如图 1 所示。该结构共分为四层：

最上面是应用层，即 android.location 包中包含的内容，是以 Java 语言提供的 API。

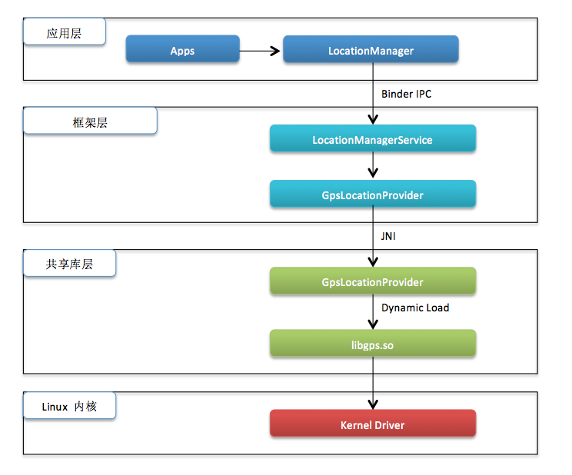
第二层是框架层，这一层包含了系统服务的实现，主要由 Java 语言来实现。

第三层是共享库层，本层由 C 以及 C++ 语言实现 , 框架层与共享库层使用 JNI 进行衔接。

最下面一层是 Linux 内核层 , 整个 Android 系统都是以 Linux 内核为基础的。

从上至下它们是逐层依赖的关系，每层依赖下面一层完成其所需提供的服务。

图 1. 定位服务的实现架构图



## 启动(startOtherServices)

ServerThread 中 run 方法的代码片段

**boolean** disableLocation = SystemProperties.getBoolean(**"config.disable\_location"**, **false**);

**if** (!disableLocation) {  
 location = **new** LocationManagerService(context);  
 ServiceManager.addService(Context.LOCATION\_SERVICE, location);  
 countryDetector = **new** CountryDetectorService(context);  
 ServiceManager.addService(Context.COUNTRY\_DETECTOR, countryDetector);  
}

## 主要字段

获取位置信息可以选择不同的 Location Provider，每个 Location Provider 可能会记录最近一次的定位信息。同时，我们也可以使用监听器来主动获取位置更新通知。所有的这些功能，都是在 LocationManagerService 中实现的。

## LocationManagerService 中的主要字段

| **类型** | **名称** | **说明** |
| --- | --- | --- |
| HashMap | mLastWriteTime | 记录最后一次位置更新的时间，这是一个以 Location Provider 名称为键的映射。 |
| Set | mEnabledProviders | 有效的 Location Provider。 |
| Set | mDisabledProviders | 无效的 Location Provider。 |
| boolean | sProvidersLoaded | Location Provider 是否已经被加载。 |
| String | mNetworkLocationProviderPackageName | 提供以网络方式进行定位服务的包名。 |
| String | mGeocodeProviderPackageName | 提供地理位置码服务的包名。 |
| LocationWorkerHandler | mLocationHandler | 这是一个 Handler，用来处理位置信息更新和包更新两种消息。 |
| LocationProviderProxy | mNetworkLocationProvider | 以网络方式提供定位服务的 Location Provider 的代理。 |
| LocationProviderInterface | mGpsLocationProvider | 依赖于 GPS 模块实现定位的 Location Provider。 |
| ArrayList | mProviders | 所有的 Location Provider。 |
| HashMap | mProvidersByName | 所有的 Location Provider，以名字为键存储在映射中。 |
| Object | mLock | 作为内部实现的锁使用。 |
| HashMap | mLastKnownLocation | 最近一次的定位信息，以 Location Provider 的名称为键的映射。 |
| PackageMonitor | mPackageMonitor | 监测器，监测服务包更新事件，并发送消息给 mLocationHandler。 |

从这些字段中我们可以看出，LocationManagerService 中的主要内容都是围绕着 Location Provider 而实现的。

### 构造函数

**public** LocationManagerService(Context context) {  
 **super**();  
 mContext = context;  
 mAppOps = (AppOpsManager)context.getSystemService(Context.APP\_OPS\_SERVICE);  
  
 *// Let the package manager query which are the default location  
 // providers as they get certain permissions granted by default.* PackageManagerInternal packageManagerInternal = LocalServices.getService(  
 PackageManagerInternal.**class**);  
 packageManagerInternal.setLocationPackagesProvider(  
 **new** PackageManagerInternal.PackagesProvider() {  
 @Override  
 **public** String[] getPackages(**int** userId) {  
 **return** mContext.getResources().getStringArray(  
 com.android.internal.R.array.config\_locationProviderPackageNames);  
 }  
 });  
}

//利用PackageManager查询默认的LocationProvider

*<!--  
 Sets the package names whose certificates should be used to  
 verify location providers are allowed to be loaded.  
-->*<**string-array name="config\_locationProviderPackageNames" translatable="false"**>  
 <**item**>com.google.android.gms</**item**>  
 <**item**>com.android.location.fused</**item**>  
</**string-array**>

<**string-array name="config\_locationProviderPackageNames" translatable="false"**>  
 *<!-- The standard AOSP fused location provider -->* <**item**>com.android.location.fused</**item**>  
</**string-array**>

## systemRunning

sd

public void systemRunning() {

synchronized (mLock) {

mPackageManager = mContext.getPackageManager();

mPowerManager = (PowerManager) mContext.getSystemService(Context.POWER\_SERVICE);

//用于处理位置信息和包更新两种消息

mLocationHandler = new LocationWorkerHandler(BackgroundThread.get().getLooper());

//Android平台提供粗细两种精度的位置信息，其中粗精度的位置信息由LocationFudger根据

//细精度的位置信息进行一定的数字模糊处理后得到

mLocationFudger = new LocationFudger(mContext, mLocationHandler);

//系统有一个黑白名单用于禁用某些特定的LP，在黑白名单中，LP有其java对应的包名指定

mBlacklist = new LocationBlacklist(mContext, mLocationHandler);

mBlacklist.init();

//创建地理围栏对象

mGeofenceManager = new GeofenceManager(mContext, mBlacklist);

....

// 创建及加载系统中所有的LP

loadProvidersLocked();

//根据设置中的开关情况，开启或禁止某一个LP，在此函数中，各个LP中的enable/disable函数被调用

updateProvidersLocked();

}

//监听设置数据库的变化，字段LOCATION\_PROVIDER\_ALLOWED，当用户点击设置  
//菜单按钮时，会改变该字段的值

Settings.Secure.LOCATION\_PROVIDERS\_ALLOWED

Settings.Global.LOCATION\_BACKGROUND\_THROTTLE\_INTERVAL\_MS

Settings.Global.LOCATION\_BACKGROUND\_THROTTLE\_PACKAGE\_WHITELIST

### loadProvidersLocked

---------------------

private void loadProvidersLocked() {

//创建passiveProvider，该LP始终是enable的，它自己不能更新位置信息，

//而是靠其他的LP来触发

//PassiveProvider的位置更新是由LMS接收其他LP的位置更新通知后，主动调用LocationProvider的、

//updateLocation函数来完成

PassiveProvider passiveProvider = new PassiveProvider(this);

addProviderLocked(passiveProvider);

//mEnabledProviders用于保存被启用的LP

mEnabledProviders.add(passiveProvider.getName());

mPassiveProvider = passiveProvider;

//mEnabledProviders用于保存所有被启用的LP

#### GnssLocationProvider

**if** (GnssLocationProvider.isSupported()) {  
 *// Create a gps location provider* GnssLocationProvider gnssProvider = **new** GnssLocationProvider(mContext, **this**,  
 mLocationHandler.getLooper());  
 mGnssSystemInfoProvider = gnssProvider.getGnssSystemInfoProvider();  
 mGnssBatchingProvider = gnssProvider.getGnssBatchingProvider();  
 mGnssStatusProvider = gnssProvider.getGnssStatusProvider();  
 mNetInitiatedListener = gnssProvider.getNetInitiatedListener();  
 addProviderLocked(gnssProvider);  
 mRealProviders.put(LocationManager.GPS\_PROVIDER, gnssProvider);  
 mGnssMeasurementsProvider = gnssProvider.getGnssMeasurementsProvider();  
 mGnssNavigationMessageProvider = gnssProvider.getGnssNavigationMessageProvider();  
 mGpsGeofenceProxy = gnssProvider.getGpsGeofenceProxy();  
}

~~if (GpsLocationProvider.isSupported()) {~~

~~//GpsLocationProvider使用GPS定位是最准确的定位方式~~

~~//GPSLocationProvider有LMS创建并加载，运行在LMS所在的system\_process中，属于系统提供的LP服务~~

~~GpsLocationProvider gpsProvider = new GpsLocationProvider(mContext, this,~~

~~mLocationHandler.getLooper());~~

~~mGpsStatusProvider = gpsProvider.getGpsStatusProvider();~~

~~mNetInitiatedListener = gpsProvider.getNetInitiatedListener();~~

~~//保存GpsLocationProvider~~

~~addProviderLocked(gpsProvider);~~

~~//GpsLocationProvider真实的位置提供者，将其保存在mRealProviders中~~

~~mRealProviders.put(LocationManager.GPS\_PROVIDER, gpsProvider);~~

~~mGpsMeasurementsProvider = gpsProvider.getGpsMeasurementsProvider();~~

~~mGpsNavigationMessageProvider = gpsProvider.getGpsNavigationMessageProvider();~~

~~mGpsGeofenceProxy = gpsProvider.getGpsGeofenceProxy();~~

~~}~~

#### locationProviderPackageNames

//config\_locationProviderPackageNames存储了第三方LP的java包名，Android原生代码中该值为

//“com.android.location.fused”，FusedLP对应的源码路径为framework/base/packages/FusedLocation

//

Resources resources = mContext.getResources();

ArrayList<String> providerPackageNames = new ArrayList<String>();

String[] pkgs = resources.getStringArray(

com.android.internal.R.array.config\_locationProviderPackageNames);

if (pkgs != null) providerPackageNames.addAll(Arrays.asList(pkgs));

//加载应用程序使用的LP服务时，LMS将检查它们的签名信息及版本信息

ensureFallbackFusedProviderPresentLocked(providerPackageNames);

#### networkProvider

//加载networkLP，创建以网络方式进行定位的LP

//使用网络实现定位服务的代理，网络定位依赖于手机信号的基站或者WiFi接入点作为定位的基础

<**bool name="config\_enableNetworkLocationOverlay" translatable="false"**>true</**bool**>  
*<!-- Package name providing network location support. Used only when  
 config\_enableNetworkLocationOverlay is false. -->*<**string name="config\_networkLocationProviderPackageName" translatable="false"**>@null</**string**>

LocationProviderProxy networkProvider = LocationProviderProxy.createAndBind(

mContext,

LocationManager.NETWORK\_PROVIDER,

NETWORK\_LOCATION\_SERVICE\_ACTION,

com.android.internal.R.bool.config\_enableNetworkLocationOverlay,

com.android.internal.R.string.config\_networkLocationProviderPackageName,

com.android.internal.R.array.config\_locationProviderPackageNames,

mLocationHandler);

if (networkProvider != null) {

mRealProviders.put(LocationManager.NETWORK\_PROVIDER, networkProvider);

//应用程序实现的LP保存在mProxyProviders

mProxyProviders.add(networkProvider);

addProviderLocked(networkProvider);

} else {

Slog.w(TAG,  "no network location provider found");

}

#### fusedLocationProvider

//加载FusedLP

//属于系统提供的应用程序，其内部将使用其他的LP

LocationProviderProxy fusedLocationProvider = LocationProviderProxy.createAndBind(

mContext,

LocationManager.FUSED\_PROVIDER,

FUSED\_LOCATION\_SERVICE\_ACTION,

com.android.internal.R.bool.config\_enableFusedLocationOverlay,

com.android.internal.R.string.config\_fusedLocationProviderPackageName,

com.android.internal.R.array.config\_locationProviderPackageNames,

mLocationHandler);

if (fusedLocationProvider != null) {

addProviderLocked(fusedLocationProvider);

mProxyProviders.add(fusedLocationProvider);

//FusedLP默认处于启动状态

mEnabledProviders.add(fusedLocationProvider.getName());

mRealProviders.put(LocationManager.FUSED\_PROVIDER, fusedLocationProvider);

} else {

Slog.e(TAG, "no fused location provider found",

new IllegalStateException("Location service needs a fused location provider"));

}

#### mGeocodeProvider

//mGeocodeProvider由第三方应用程序提供，一般与NetWorkLP位于同一个应用程序中

mGeocodeProvider = GeocoderProxy.createAndBind(mContext,

com.android.internal.R.bool.config\_enableGeocoderOverlay,

com.android.internal.R.string.config\_geocoderProviderPackageName,

com.android.internal.R.array.config\_locationProviderPackageNames,

mLocationHandler);

if (mGeocodeProvider == null) {

Slog.e(TAG,  "no geocoder provider found");

}

.....

}

#### flpHardwareProvider

asd

*// bind to fused hardware provider if supported  
// in devices without support, requesting an instance of FlpHardwareProvider will raise an  
// exception, so make sure we only do that when supported*FlpHardwareProvider flpHardwareProvider;  
**if** (FlpHardwareProvider.isSupported()) {  
 flpHardwareProvider = FlpHardwareProvider.getInstance(mContext);  
 FusedProxy fusedProxy = FusedProxy.createAndBind(  
 mContext,  
 mLocationHandler,  
 flpHardwareProvider.getLocationHardware(),  
 com.android.internal.R.bool.config\_enableHardwareFlpOverlay,  
 com.android.internal.R.string.config\_hardwareFlpPackageName,  
 com.android.internal.R.array.config\_locationProviderPackageNames);  
 **if** (fusedProxy == **null**) {  
 Slog.d(TAG, **"Unable to bind FusedProxy."**);  
 }  
} **else** {  
 flpHardwareProvider = **null**;  
 Slog.d(TAG, **"FLP HAL not supported"**);  
}

#### geofence provider

GeofenceProxy provider = GeofenceProxy.createAndBind(  
 mContext,com.android.internal.R.bool.config\_enableGeofenceOverlay,  
 com.android.internal.R.string.config\_geofenceProviderPackageName,  
 com.android.internal.R.array.config\_locationProviderPackageNames,  
 mLocationHandler,  
 mGpsGeofenceProxy,  
 flpHardwareProvider != **null** ? flpHardwareProvider.getGeofenceHardware() : **null**);  
**if** (provider == **null**) {  
 Slog.d(TAG, **"Unable to bind FLP Geofence proxy."**);  
}

#### ind to hardware activity recognition

sd

**boolean** activityRecognitionHardwareIsSupported = ActivityRecognitionHardware.isSupported();  
ActivityRecognitionHardware activityRecognitionHardware = **null**;  
**if** (activityRecognitionHardwareIsSupported) {  
 activityRecognitionHardware = ActivityRecognitionHardware.getInstance(mContext);  
} **else** {  
 Slog.d(TAG, **"Hardware Activity-Recognition not supported."**);  
}  
ActivityRecognitionProxy proxy = ActivityRecognitionProxy.createAndBind(  
 mContext,  
 mLocationHandler,  
 activityRecognitionHardwareIsSupported,  
 activityRecognitionHardware,  
 com.android.internal.R.bool.config\_enableActivityRecognitionHardwareOverlay,  
 com.android.internal.R.string.config\_activityRecognitionHardwarePackageName,  
 com.android.internal.R.array.config\_locationProviderPackageNames);  
**if** (proxy == **null**) {  
 Slog.d(TAG, **"Unable to bind ActivityRecognitionProxy."**);  
}

#### testProviderStrings

d

String[] testProviderStrings = resources.getStringArray(  
 com.android.internal.R.array.config\_testLocationProviders);  
**for** (String testProviderString : testProviderStrings) {  
 String fragments[] = testProviderString.split(**","**);  
 String name = fragments[0].trim();  
 **if** (mProvidersByName.get(name) != **null**) {  
 **throw new** IllegalArgumentException(**"Provider \""** + name + **"\" already exists"**);  
 }  
 ProviderProperties properties = **new** ProviderProperties(  
 Boolean.parseBoolean(fragments[1]) */\* requiresNetwork \*/*,  
 Boolean.parseBoolean(fragments[2]) */\* requiresSatellite \*/*,  
 Boolean.parseBoolean(fragments[3]) */\* requiresCell \*/*,  
 Boolean.parseBoolean(fragments[4]) */\* hasMonetaryCost \*/*,  
 Boolean.parseBoolean(fragments[5]) */\* supportsAltitude \*/*,  
 Boolean.parseBoolean(fragments[6]) */\* supportsSpeed \*/*,  
 Boolean.parseBoolean(fragments[7]) */\* supportsBearing \*/*,  
 Integer.parseInt(fragments[8]) */\* powerRequirement \*/*,  
 Integer.parseInt(fragments[9]) */\* accuracy \*/*);  
 addTestProviderLocked(name, properties);  
}

---------------------

作者：漆黑迷夜

来源：CSDN

原文：https://blog.csdn.net/lans1996/article/details/78144638

版权声明：本文为博主原创文章，转载请附上博文链接！

---------------------

作者：漆黑迷夜

来源：CSDN

原文：https://blog.csdn.net/lans1996/article/details/78144638

版权声明：本文为博主原创文章，转载请附上博文链接！

### LocationProviderInterface

上文中我们说了，Location Provider 是真正获取位置信息的模块。在 android.location 包中，用 LocationProvider 这个接口来描述。而这一接口是提供给应用层 API 使用的，在 LocationManagerService 中，Location Provider 使用另外一个接口来描述，这就是 com.android.location.provider. LocationProviderInterface，LocationManagerService 对于定位服务的实现均是通过调用 LocationProviderInterface 来完成的。

LocationProviderInterface 对象存储在名称为 mProviders 以及 mProvidersByName 的字段中（见表 2）。 LocationProviderInterface 接口的说明如表 3 所示：

##### 表 3. LocationProviderInterface 接口说明

| **名称** | **说明** |
| --- | --- |
| getName | 获取当前 Location Provider 的名称 |
| requiresNetwork | 该 Location Provider 是否需要网络 |
| requiresSatellite | 该 Location Provider 是否需要卫星 |
| requiresCell | 该 Location Provider 是否需要手机蜂窝信号 |
| hasMonetaryCost | 该 Location Provider 是否需要耗费金钱 |
| supportsAltitude | 该 Location Provider 是否支持海拔高度信息 |
| supportsSpeed | 该 Location Provider 是否支持速度信息 |
| supportsBearing | 该 Location Provider 是否支持方位信息 |
| getPowerRequirement | 获取该 Location Provider 的耗电量级别 |
| meetsCriteria | 该 Location Provider 是否能符合指定的 Criteria |
| getAccuracy | 获取该 Location Provider 的精度级别 |
| isEnabled | 查询有效状态 |
| enable | 使该 Location Provider 有效 |
| disable | 使该 Location Provider 无效 |
| getStatus | 获取该 Location Provider 的状态 |
| getStatusUpdateTime | 获取该 Location Provider 的状态更新时间 |
| enableLocationTracking | 使该 Location Provider 位置追踪有效 |
| requestSingleShotFix | 请求 Single Shot Fix |
| getInternalState | 获取该 Location Provider 的内部状态 |
| setMinTime | 设置最小时间 |
| updateNetworkState | 使该 Location Provider 更新网络状态 |
| updateLocation | 使该 Location Provider 更新位置 |
| sendExtraCommand | 使该 Location Provider 发送辅助的命令 |
| addListener | 增加监听器 |
| removeListener | 移除监听器 |

在 Android 源码中，实现 LocationProviderInterface 接口的类有四个，它们如表 4 所示：

##### 表 4. LocationProviderInterface 的实现类

| **名称** | **说明** |
| --- | --- |
| GpsLocationProvider | 使用 Gps 卫星定位，最准确的定位方式。 |
| PassiveProvider | 该 Provider 并不真正触发定位的更新，而是使用其他 Provider 来完成位置报告。 |
| LocationProviderProxy | 使用网络实现定位的服务的代理。网络定位依赖于手机信号的基站或者 Wifi 接入点作为定位的基础。注意该类只是个代理，并不包含真正的实现逻辑。 |
| MockProvider | 为了辅助测试的模拟实现类。 |

那么，在 LocationManagerService 中，对于 LocationProviderInterface 的加载是在什么时候完成的呢？上面我们已经看过 LocationManagerService 的构造方法了，并没有看到这部分内容。 其实，LocationManagerService 是一个线程类，除了构造函数以外，在其 run 方法中又完成了另外一部分的初始化工作，主要是调用其 initialize 方法。 在 initialize 方法中调用了 loadProviders 方法，loadProviders 这个方法中完成了 Location Provider 的加载工作。 该方法又经过同步加锁以及异常的包装，最终的实现方法是 \_loadProvidersLocked。 \_loadProvidersLocked 方法的代码如清单 5 所示：

# FusedLocation

# CountryDetectorService

# 参考