

GNSS TOOL APP 二次开发接入指南

1. 文档概述

本文档面向第三方 Android APP 开发人员，说明如何接入 GNSS TOOL APP 输出的高精度定位数据、NMEA 原始报文和测绘扩展数据。

GNSS TOOL APP 支持以下 5 种数据接入方式：

1. Android 原生 Mock 高精度位置接入；
2. Mock 位置 Extras 附加分类 NMEA 报文接入；
3. Android 系统标准 NMEA 监听器接入；
4. AIDL 跨进程数据接入；
5. 全局广播定位数据接入。

第三方 APP 可根据业务需求选择对应通道：

接入方式	是否支持坐标	是否支持 NMEA	是否支持平面坐标	是否依赖 Mock 权限	推荐场景
Mock 高精度位置	支持	不支持	不支持	是	普通高精度定位
Mock Extras 附加 NMEA	支持	支持分类单条 NMEA	不支持	是	轻量 NMEA 读取
系统 NMEA 监听器	支持	支持完整 NMEA 流	不支持	是	原生 NMEA 接口适配
AIDL 跨进程接入	支持	支持完整 NMEA 流	支持	否	专业测绘、高频采集
全局广播接入	支持	不支持	支持	否	轻量坐标读取

2. 前置条件

接入前请确认以下条件已满足：

1. 设备已安装 GNSS TOOL APP；
2. GNSS TOOL APP 已成功连接 GNSS 板卡；
3. 支持的连接方式支持蓝牙、USB、串口、Wi-Fi 或内置 PDA 板卡；
4. 第三方 APP 运行环境为 Android 8.0 及以上；
5. GNSS TOOL APP 已获取有效定位解；
6. 对应的数据分发开关已在 GNSS TOOL APP 设置中开启。

如需使用 Mock 相关通道，还需满足以下条件：

1. 设备已开启开发者选项；
2. 在系统“选择模拟位置信息应用”中选择 GNSS TOOL；
3. 第三方 APP 已申请定位权限；
4. GNSS TOOL 已开启系统位置播发相关开关。

3. Android Manifest 配置

3.1 基础定位权限

使用 Android 原生定位接口、Mock 位置或系统 NMEA 监听器时，需要声明定位权限：

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

对于高精度定位业务，应优先申请 ACCESS_FINE_LOCATION 权限。

3.2 Android 11 及以上包可见性配置

如果第三方 APP 需要绑定 GNSS TOOL 的 AIDL 服务，建议在 AndroidManifest.xml 中增加包可见性声明：

```
<queries>
<package android:name="com.sinognss.gnsstool" />
</queries>
```

该配置用于保证 Android 11 及以上系统能够正常发现并绑定 GNSS TOOL APP 服务。

3.3 AIDL 服务访问说明

AIDL 通道依赖 GNSS TOOL APP 暴露的 Binder 服务。第三方 APP 需要确保：

1. GNSS TOOL APP 已安装；
2. GNSS TOOL APP 已开启 AIDL 播发开关；
3. 第三方 APP 中的 AIDL 文件包名和接口定义与 GNSS TOOL 提供文件保持一致；
4. 绑定服务时使用正确的包名和 Service Action。

4. 接入方式一：Android 原生 Mock 高精度位置

4.1 功能说明

该方式通过 Android 原生定位接口获取 GNSS TOOL 注入到系统 GPS Provider 的高精度位置数据。可获取字段包括：

- 纬度；
- 经度；
- 高程；
- 精度；
- 速度；
- 方向；
- 时间戳。

该方式不包含 NMEA 报文，也不包含平面坐标、卫星数、差分延迟等

测绘扩展字段。

4.2 适用场景

适用于仅需要高精度经纬度和高程的普通定位场景，例如：

- 高精度地图定位；
- 轨迹记录；
- 位置展示；
- 替代系统 GPS 定位源；
- 不需要 NMEA 报文的第三方 APP。

4.3 GNSS TOOL 开关依赖

需在 GNSS TOOL APP 中开启：**系统播发高精度位置**，同时需要在系统开发者选项中选择 GNSS TOOL 作为模拟位置应用。

4.4 代码示例

```
class MockLocationActivity : AppCompatActivity() {

    private lateinit var locationManager: LocationManager

    lateinit var binding: ActivityGnssLocationBinding

    private val locationListener = object : LocationListener {

        override fun onLocationChanged(location: Location) {
            // 高精度大地坐标
            val lat = location.latitude
            val lon = location.longitude
            val alt = location.altitude

            // 精度信息
            val horizontalAcc = location.accuracy
```

```

        val verticalAcc = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            location.verticalAccuracyMeters
        } else {
            null
        }

        // 速度与方向
        val speed = location.speed
        val bearing = location.bearing

        // 时间戳
        val time = location.time

        val locationTxt = "lat=$lat, lon=$lon, alt=$alt, hAcc=$horizontalAcc, vAcc=$verticalAcc,
speed=$speed, bearing=$bearing, time=$time"
        Log.d(
            "GNSS MOCK",
            locationTxt
        )

        runOnUiThread {
            binding.tvLocation.setText(locationTxt)
        }

        location.extras?.let {
            parseNmea(it)
        }
    }

    override fun onProviderEnabled(provider: String) {
        Log.d("GNSS MOCK", "Provider enabled: $provider")
    }

    override fun onProviderDisabled(provider: String) {
        Log.w("GNSS MOCK", "Provider disabled: $provider")
    }
}

```

```

private fun parseNmea(bundle: Bundle) {

    val gga = bundle.getString("gpgga")
    val rmc = bundle.getString("gnrmc")
    val vtg = bundle.getString("gnavtg")
    val gsa = bundle.getString("gngsa")
    val gsv = bundle.getString("ngsv")
    val gll = bundle.getString("ngll")
    val gst = bundle.getString("ngst")

    gga?.let {
        Log.d("GNSS_NMEA_GGA", it)
        // 可根据 GGA 判断固定解状态、卫星数、高程等
    }

    gst?.let {
        Log.d("GNSS_NMEA_GST", it)
        // 可根据 GST 判断定位误差统计信息
    }

    rmc?.let {
        Log.d("GNSS_NMEA_RMC", it)
        // 可根据 RMC 读取速度、航向、UTC 时间等
    }

    runOnUiThread {
        binding.tvNmea.setText("$gga\n$rmc\n$vtg\n$gsa\n$gsv\n$gll\n$gst")
    }
}

private val nmeaLocationListener = object : LocationListener {

    override fun onLocationChanged(location: Location) {

    }

}

```

```

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = ActivityGnssLocationBinding.inflate(layoutInflater)
    setContentView(binding.root)

    locationManager = getSystemService(Context.LOCATION_SERVICE) as LocationManager

    requestMockLocation()

}

private fun requestMockLocation() {
    if (
        ActivityCompat.checkSelfPermission(
            this,
            Manifest.permission.ACCESS_FINE_LOCATION
        ) != PackageManager.PERMISSION_GRANTED
    ) {
        ActivityCompat.requestPermissions(
            this,
            arrayOf(Manifest.permission.ACCESS_FINE_LOCATION),
            1001
        )
        return
    }

    locationManager.requestLocationUpdates(
        LocationManager.GPS_PROVIDER,
        100L,
        0f,
        locationListener,
        Looper.getMainLooper()
    )
}

override fun onDestroy() {
    super.onDestroy()
    locationManager.removeUpdates(locationListener)
}

```

```
}  
}
```

4.5 注意事项

1. 第三方 APP 必须申请定位权限；
2. GNSS TOOL 必须被系统设置为模拟位置应用；
3. GNSS TOOL 必须已连接设备并持续输出定位结果；
4. 如果只需要坐标，优先使用该方式，接入成本最低。

5. 接入方式二：Mock 位置 Extras 附加分类 NMEA 报文

5.1 功能说明

该方式在 Android 原生 Location 回调的基础上，通过 Location.getExtras() 读取 GNSS TOOL 附加的分类 NMEA 报文。支持读取的 NMEA 类型包括：

- GGA；
- RMC；
- VTG；
- GSA；
- GSV；
- GLL；
- GST。

该方式适合需要少量 NMEA 报文辅助判断定位状态的场景。

5.2 GNSS TOOL 开关依赖

需在 GNSS TOOL APP 中开启：**系统位置附加 NMEA 报文**，同时需要开启 Mock 位置相关能力。

5.3 MockBundleKey 对照表

Key 字符串	对应报文类型	说明
gpgga	\$GPGGA	定位质量、卫星数、高程、差分状态
gnrmc	\$GNRMC	推荐最小定位信息、时间、速度、航向
gnvtg	\$GNVTG	地面航向与速度
gngsa	\$GNGSA	DOP 与参与解算卫星
gngsv	\$GNGSV	可见卫星信息
gnll	\$GNLL	经纬度与 UTC 时间
gngst	\$GNGST	定位误差统计信息

5.4 代码示例

```
private fun parseNmea(bundle: Bundle) {

    val gga = bundle.getString("gpgga")
    val rmc = bundle.getString("gnrmc")
    val vtg = bundle.getString("gnvtg")
    val gsa = bundle.getString("gngsa")
    val gsv = bundle.getString("gngsv")
    val gll = bundle.getString("gnll")
    val gst = bundle.getString("gngst")

    gga?.let {
        Log.d("GNSS_NMEA_GGA", it)
        // 可根据 GGA 判断固定解状态、卫星数、高程等
    }

    gst?.let {
        Log.d("GNSS_NMEA_GST", it)
        // 可根据 GST 判断定位误差统计信息
    }

    rmc?.let {
```

```

        Log.d("GNSS_NMEA_RMC", it)
        // 可根据 RMC 读取速度、航向、UTC 时间等
    }

    runOnUiThread {
        binding.tvNmea.setText("$gga\n$rmc\n$vtg\n$gsa\n$gsv\n$gll\n$gst")
    }
}

```

5.5 注意事项

1. 该方式读取的是分类后的单条 NMEA 报文；
2. 并不等同于完整连续的 NMEA 原始数据流；
3. 第三方 APP 仍然需要通过 Android 原生定位接口接收 Location；
4. 该方式仍受 Mock 模拟位置权限限制；
5. 如果业务需要完整原始 NMEA 流，建议使用 AIDL 或系统 NMEA 监听器方式。

6. 接入方式三：系统标准 NMEA 监听器

6.1 功能说明

该方式通过 Android 标准 `OnNmeaMessageListener` 接收完整 NMEA 原始报文流。

第三方 APP 使用 Android 系统原生接口：

```
locationManager.addNmeaListener(...)
```

即可监听 GNSS TOOL 经系统转发后的 NMEA 报文。

6.2 前置要求

该方式需要设备系统底层支持 GNSS TOOL 的 NMEA 转发能力。

接入前请确认：

1. 手持设备系统已完成 GNSS TOOL NMEA 转发定制；

2. GNSS TOOL APP 已开启系统播发 NMEA 报文；
3. 第三方 APP 已申请定位权限；
4. 设备已获取有效定位解。

6.3 GNSS TOOL 开关依赖

需在 GNSS TOOL APP 中开启：系统播发 NMEA 报文。

6.4 代码示例

```
class NmeaListenerActivity : AppCompatActivity() {

    private lateinit var locationManager: LocationManager

    private lateinit var binding: ActivityGnssLocationBinding

    private val nmeaListener = OnNmeaMessageListener { message, timestamp ->
        // message 为系统转发的 NMEA 原始报文
        Log.d("GNSS_NMEA_RAW", "timestamp=$timestamp, message=$message")
        runOnUiThread {
            binding.tvNmea.text = "NMEA:${message}"
        }
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityGnssLocationBinding.inflate(layoutInflater)
        setContentView(binding.root)
        locationManager = getSystemService(Context.LOCATION_SERVICE) as LocationManager

        registerNmeaListener()
    }

    private fun registerNmeaListener() {
        if (
            ActivityCompat.checkSelfPermission(
                this,
                Manifest.permission.ACCESS_FINE_LOCATION
            ) == PackageManager.PERMISSION_GRANTED
        ) {
            locationManager.requestLocationUpdates(
                LocationManager.NETWORK_PROVIDER,
                0,
                0,
                nmeaListener
            )
        }
    }
}
```

```

        ) != PackageManager.PERMISSION_GRANTED
    ){
        ActivityCompat.requestPermissions(
            this,
            arrayOf(Manifest.permission.ACCESS_FINE_LOCATION),
            1002
        )
        return
    }

    locationManager.addNmeaListener(nmeaListener)
}

override fun onDestroy() {
    super.onDestroy()
    locationManager.removeNmeaListener(nmeaListener)
}
}

```

6.5 注意事项

1. 该方式依赖设备系统定制；
2. 原生 Android 系统默认不会将 Mock Location Extras 中的 NMEA 字段转发至 addNmeaListener()；
3. 如果设备未完成系统适配，即使代码正确，也可能无法收到 NMEA 数据；
4. 如果设备不支持该方式，建议改用 AIDL 通道获取完整 NMEA 报文。

7. 接入方式四：AIDL 跨进程数据接入

7.1 功能说明

AIDL 通道通过 Binder 跨进程通信方式，将 GNSS TOOL 中的定位数据、NMEA 报文和测绘扩展字段推送给第三方 APP。

该方式不依赖 Android Mock Location 权限，也不依赖系统 NMEA 转

发定制，是专业测绘类 APP 推荐使用的接入方式。

7.2 AIDL 通道优势

AIDL 通道支持以下能力：

1. 不受模拟位置权限限制；
2. 支持高频、稳定的数据推送；
3. 支持完整 NMEA 原始报文；
4. 支持平面 NEZ 坐标；
5. 支持天线高修正高程；
6. 支持 GNSS 板卡原始高程；
7. 支持跟踪卫星数、解算卫星数；
8. 支持差分类型、差分延迟等测绘专用参数。

7.3 GNSS TOOL 开关依赖

需在 GNSS TOOL APP 中开启：**AIDL 播发位置信息**。

7.4 接入步骤

第三方 APP 接入 AIDL 通道时，按以下步骤进行：

1. 拷贝 GNSS TOOL 提供的 AIDL 文件至第三方项目；
2. 保持 AIDL 文件包名、目录结构与 GNSS TOOL 提供文件一致；
3. 编译项目，生成 AIDL 接口代理类；
4. 绑定 GNSS TOOL AIDL 服务；
5. 注册 IGnssDataListener 回调；
6. 在回调中接收定位数据和 NMEA 报文；
7. 页面销毁或业务结束时注销监听并解绑服务。

需要拷贝的 AIDL 文件包括：

IGnssDataManager.aidl

IGnssDataListener.aidl

7.5 AIDL 服务信息

GNSS TOOL AIDL 服务信息如下：

项目	值
包名	com.sinognss.gnsstool
Service Action	com.sinognss.sm.mock.service.GnssMockAidlService
通信方式	Binder / AIDL
数据回调	IGnssDataListener

7.6 AIDL 完整接入代码

```
class GnssAidlClient{
    private val context: Context
} {

    private var gnssDataManager: I GnssDataManager? = null
    private var isBound = false

    var onLocation: ((GnssDataBean) -> Unit)? = null
    var onNmea: ((String) -> Unit)? = null

    private val aidlListener = object : I GnssDataListener.Stub() {

        /**
         * 接收全套测绘定位数据
         */
        override fun onGnssDataReceive(data: Bundle?) {
            data ?. return

            val bean = GnssDataBean.parse(data)

            // 大地坐标
```

```

        val lat = bean.lat
        val lon = bean.lon

        // 高程信息
        val altAntenna = bean.alt    // 地面高程，已叠加天线高
        val altRaw = bean.oAlt       // GNSS 板卡原始相位中心高程

        // 局部平面坐标
        val n = bean.n
        val e = bean.e
        val z = bean.z

        // 解算状态
        val diffType = bean.diffType
        val trackSat = bean.trackSatNum
        val solutionSat = bean.solutionSatNum
        val diffDelay = bean.diffDelay
        val utcTime = bean.utcTime

        Log.d(
            "GNSS_AIDL_DATA",
            "lat=$lat, lon=$lon, alt=$altAntenna, rawAlt=$altRaw, n=$n, e=$e, z=$z, diffType=$diffType, trackSat=$trackSat, solutionSat=$solutionSat, diffDelay=$diffDelay, utcTime=$utcTime"
        )

        onLocation?.invoke(bean)
    }

    /**
     * 接收完整原始 NMEA 报文
     */
    override fun onNmeaReceive(nmea: String?) {
        if (nmea.isNullOrEmpty()) return
        onNmea?.invoke(nmea)
        Log.d("GNSS_AIDL_NMEA", nmea)
    }
}

```

```

private val serviceConnection = object : ServiceConnection {

    override fun onServiceConnected(name: ComponentName, service: IBinder) {
        gnssDataManager = IGnssDataManager.Stub.asInterface(service)

        try {
            gnssDataManager?.registerGnssDataListener(aidlListener)
            Log.d("GNSS_AIDL", "AIDL service connected and listener registered")
        } catch (e: RemoteException) {
            Log.e("GNSS_AIDL", "registerGnssDataListener failed", e)
        }
    }

    override fun onServiceDisconnected(name: ComponentName) {
        gnssDataManager = null
        isBound = false
        Log.w("GNSS_AIDL", "AIDL service disconnected")
    }

    override fun onBindingDied(name: ComponentName) {
        gnssDataManager = null
        isBound = false
        Log.w("GNSS_AIDL", "AIDL binding died")
    }

    override fun onNullBinding(name: ComponentName) {
        gnssDataManager = null
        isBound = false
        Log.w("GNSS_AIDL", "AIDL null binding")
    }
}

/**
 * 绑定 GNSS TOOL AIDL 服务
 */
fun bindAidlService() {

```

```

        val intent = Intent().apply {
            component = ComponentName(
                "com.sinognss.gnssstool",
                "com.sinognss.sm.mock.service.GnssMockAidlService"
            )
        }
        // 绑定远程服务
        isBound = context.bindService(intent, serviceConnection!!, Context.BIND_AUTO_CREATE)

        Log.d("GNSS_AIDL", "bindService result=$isBound")
    }

    /**
     * 注销监听并解绑服务
     */
    fun unbindAidlService() {
        try {
            gnssDataManager?.unregisterGnssDataListener(aidlListener)
        } catch (e: RemoteException) {
            Log.e("GNSS_AIDL", "unregisterGnssDataListener failed", e)
        }

        if (isBound) {
            context.unbindService(serviceConnection)
            isBound = false
        }

        gnssDataManager = null
    }
}

```

7.7 Activity 中使用示例

```

class AidlDemoActivity : AppCompatActivity() {

    private lateinit var gnssAidlClient: GnssAidlClient

    lateinit var binding: ActivityGnssLocationBinding

```

```

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = ActivityGnssLocationBinding.inflate(layoutInflater)
    setContentView(binding.root)

    gnssAidlClient = GnssAidlClient(this)
    gnssAidlClient.bindAidlService()
    gnssAidlClient.onLocation = {
        runOnUiThread {
            binding.tvLocation.setText(it.toString())
        }
    }
    gnssAidlClient.onNmea = {
        runOnUiThread {
            binding.tvNmea.setText(it.toString())
        }
    }
}

}

override fun onDestroy() {
    super.onDestroy()
    gnssAidlClient.unbindAidlService()
}
}

```

7.8 GnssDataBean 字段说明

字段	类型	说明
lat	Double	WGS84 大地纬度
lon	Double	WGS84 大地经度
alt	Double	地面高程，已叠加天线高
oAlt	Double	GNSS 板卡原始相位中心高程
n	Double	当地平面北向坐标
e	Double	当地平面东向坐标

z	Double	当地平面垂直方向坐标
utcTime	Long	UTC 时间戳，单位 ms
diffType	Int	差分解状态码
trackSatNum	Int	跟踪卫星总数
solutionSatNum	Int	参与解算卫星数
diffDelay	Double / Float	差分数据延迟时长

7.9 diffType 状态码说明

diffType	含义
0	无效解
1	单点解
2	差分解
3	浮动解
4	固定解
10	基准站模式

7.10 AIDL 注意事项

1. AIDL 回调运行在线程池中，如需更新 UI，应切换至主线程；
2. 第三方 APP 应在页面销毁或服务停止时注销监听；
3. 需要处理 RemoteException；
4. 需要处理 GNSS TOOL APP 被关闭、升级或服务重启导致的断连；
5. 如果绑定失败，应检查包名、Service Action、AIDL 文件包名和 GNSS TOOL 开关状态；
6. 高频采集场景建议优先使用 AIDL 通道。

8. 接入方式五：全局广播接收定位数据

8.1 功能说明

全局广播通道通过 Android Broadcast 机制向第三方 APP 分发定位数

据。

该方式接入简单，不需要绑定服务，也不依赖 Mock 模拟位置权限，适合轻量级读取坐标的业务。

8.2 广播常量

项目	值
Action	com.sinognss.gnss.location.ACTION_GNSS_LOCATION
数据载体	Intent Extras
数据解析	GnssDataBean.parse(bundle)
是否包含 NMEA	不包含

8.3 GNSS TOOL 开关依赖

需在 GNSS TOOL APP 中开启：广播播发位置信息。

8.4 动态广播接收示例

```
class GnssBroadcastActivity : AppCompatActivity() {

    private lateinit var binding: ActivityGnssLocationBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityGnssLocationBinding.inflate(layoutInflater)
        setContentView(binding.root)
    }

    private val gnssReceiver = object : BroadcastReceiver() {

        override fun onReceive(context: Context?, intent: Intent?) {
            if (intent?.action != "com.sinognss.gnss.location.ACTION_GNSS_LOCATION") {
                return
            }
        }
    }
}
```

```

    }

    val bundle = intent.extras ?: return
    val data = GnssDataBean.parse(bundle)

    val lat = data.lat
    val lon = data.lon
    val alt = data.alt

    val n = data.n
    val e = data.e
    val z = data.z

    val diffType = data.diffType
    val trackSatNum = data.trackSatNum
    val solutionSatNum = data.solutionSatNum

    Log.d(
        "GNSS_BROADCAST",
        "lat=$lat, lon=$lon, alt=$alt, n=$n, e=$e, z=$z, diffType=$diffType, trackSat=$trackSatNum,
solutionSat=$solutionSatNum"
    )

    binding.tvLocation.setText(data.toString())
}

}

override fun onStart() {
    super.onStart()
    registerGnssBroadcast()
}

override fun onStop() {
    super.onStop()
    unregisterGnssBroadcast()
}

private fun registerGnssBroadcast() {

```

```

val filter = IntentFilter("com.sinognss.gnss.location.ACTION_GNSS_LOCATION")

if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
    registerReceiver(
        gnssReceiver,
        filter,
        Context.RECEIVER_EXPORTED
    )
} else {
    registerReceiver(gnssReceiver, filter)
}
}

private fun unregisterGnssBroadcast() {
    try {
        unregisterReceiver(gnssReceiver)
    } catch (e: IllegalArgumentException) {
        Log.w("GNSS_BROADCAST", "Receiver already unregistered")
    }
}
}

```

8.5 静态广播说明

对于定位类高频数据，不建议优先使用静态广播接收方式。推荐使用动态注册方式，并在页面或业务生命周期内注册与注销。

如确需静态注册，应在 `AndroidManifest.xml` 中声明接收器，并确认系统版本、后台限制和广播发送方式是否满足业务要求。

8.6 限制说明

1. 广播通道不传输 NMEA 报文；
2. 广播通道存在系统队列调度延迟；
3. 后台进程被系统回收后，接收稳定性弱于 AIDL；
4. 高频采集、测绘作业、长时间后台运行场景不建议使用广播通道；
5. 仅需要轻量读取坐标时，可选择该方式。

9. 接入方式选择建议

9.1 仅需要经纬度、高程、精度

推荐使用：Android 原生 Mock 高精度位置

原因：

- 接入成本最低；
- 第三方 APP 使用系统原生定位接口即可；
- 适合普通定位展示和轨迹记录。

9.2 需要读取 GGA、GST、RMC 等少量 NMEA 报文

推荐使用：Mock 位置 Extras 附加分类 NMEA 报文

原因：

- 无需绑定服务；
- 无需解析完整 NMEA 流；
- 可直接在 Location Extras 中读取分类报文。

9.3 需要完整 NMEA 原始报文流

推荐优先级如下：

1. 若设备系统已定制支持，使用系统标准 NMEA 监听器；
2. 若设备系统未定制，使用 AIDL 通道。

9.4 需要测绘平面坐标和完整扩展字段

推荐使用：AIDL 跨进程数据接入

原因：

- 字段最完整；
- 实时性最好；
- 不依赖 Mock 权限；
- 支持完整 NMEA；

- 适合专业测绘、高频采集、离线采集等场景。

9.5 仅需要轻量读取坐标

推荐使用：全局广播接收定位数据

原因：

- 接入简单；
- 不依赖 Mock 权限；
- 无需绑定 AIDL 服务；
- 适合工具类 APP 或临时读取坐标。

10. 常见问题排查

10.1 Mock 位置获取不到数据

请按以下顺序检查：

1. 系统开发者选项是否已开启；
2. “选择模拟位置信息应用”是否已选择 GNSS TOOL；
3. GNSS TOOL 是否已连接 设备；
4. GNSS TOOL 是否已有有效定位解；
5. GNSS TOOL 是否已开启“系统播发高精度位置”；
6. 第三方 APP 是否已申请 ACCESS_FINE_LOCATION 权限；
7. 第三方 APP 是否使用 LocationManager.GPS_PROVIDER 监听位置；
8. 系统定位服务是否已开启。

10.2 Extras 读取不到 NMEA 报文

请检查：

1. GNSS TOOL 是否已开启“系统位置附加 NMEA 报文”；
2. Mock 高精度位置是否能正常收到；
3. location.extras 是否为空；

4. 读取的 key 是否与文档一致；
5. 当前板卡是否持续输出对应类型的 NMEA 报文。

10.3 addNmeaListener() 无数据

请检查：

1. 设备系统是否已完成 GNSS TOOL NMEA 转发定制；
2. GNSS TOOL 是否已开启“系统播发 NMEA 报文”；
3. 第三方 APP 是否已申请定位权限；
4. 系统定位服务是否已开启；
5. 当前设备是否支持标准 NMEA 转发能力。

如果设备未进行系统定制，建议改用 AIDL 通道获取完整 NMEA 报文。

10.4 AIDL 绑定失败

请检查：

1. GNSS TOOL APP 是否已安装；
2. GNSS TOOL APP 包名是否为 `com.sinognss.gnsstool`；
3. Service Action 是否为 `com.sinognss.sm.mock.service.GnssMockAidlService`；
4. Android 11 及以上是否配置了 `<queries>`；
5. AIDL 文件包名和接口定义是否与 GNSS TOOL 提供文件一致；
6. GNSS TOOL 是否已开启“AIDL 播发位置信息”；
7. `bindService()` 返回值是否为 `true`；
8. GNSS TOOL 是否正在运行或具备被外部绑定的服务能力。

10.5 AIDL 能收到位置，但收不到 NMEA

请检查：

1. GNSS TOOL 是否已开启 AIDL NMEA 播发能力；

2. 板卡是否正在输出 NMEA 报文；
3. `onNmeaReceive(nmea: String?)` 是否已正确实现；
4. 回调是否被异常中断；
5. 第三方 APP 是否在页面销毁时过早解绑服务。

10.6 广播接收不到数据

请检查：

1. GNSS TOOL 是否已开启“广播播发位置信息”；
2. Action 是否填写正确；
3. 动态广播是否已注册；
4. Android 13 及以上动态注册时是否设置了 `Context.RECEIVER_EXPORTED`；
5. 第三方 APP 是否处于可接收广播的生命周期内；
6. GNSS TOOL 是否已有有效定位解。

11. 最佳实践

1. 普通定位类 APP 优先使用 Mock 高精度位置方式；
2. 专业测绘类 APP 优先使用 AIDL 方式；
3. 需要完整 NMEA 且系统已定制时，可使用系统标准 NMEA 监听器；
4. 仅临时读取坐标时，可使用广播方式；
5. 高频采集、长时间后台运行、离线测绘作业不建议使用广播方式；
6. 使用 AIDL 时必须做好断连重连和生命周期释放；
7. 使用 Mock 方式时必须引导用户正确选择 GNSS TOOL 为模拟位置应用；
8. 使用 Location Extras 读取 NMEA 时，应做好 key 不存在或报文为空的兼容处理；
9. 所有通道均应判断 GNSS TOOL 是否已连接板卡并获得有效定位

解。

12. 总结

GNSS TOOL APP 为第三方 Android APP 提供了多种高精度定位数据接入方式。

各方式定位如下：

- Mock 高精度位置：适合最低成本接入高精度坐标；
- Mock Extras 附加 NMEA：适合读取少量分类 NMEA 报文；
- 系统标准 NMEA 监听器：适合已完成系统定制的原生 NMEA 接入；
- AIDL 跨进程接入：适合专业测绘、高频采集和完整数据字段接入；
- 全局广播接入：适合轻量级、低频率、临时读取坐标的场景。

第三方开发者可根据业务需求、系统权限、数据字段完整性和实时性要求，选择合适的数据接入方式完成 GNSS TOOL 高精度定位能力集成。