

NFC 集成指南

概述

NFC (Near Field Communication) 功能允许用户通过触碰 NFC 标签来快速拉起 App，自动解析标签中存储的蓝牙 MAC 地址，并直接连接打印机。整个流程如下：

```
用户触碰 NFC 标签
→ App 被拉起 (冷启动/热启动)
→ 解析 URL 中的 BMAC 参数
→ SDK 蓝牙连接打印机
→ UI 更新连接状态
```

NFC 标签中存储的 URL 格式示例：

```
http://hprint.hpdt.com/hereset/index.html?Model=RTP-
808BW&SN=R808BW262000442&BMAC=28:D4:1E:73:A4:19
```

其中 `BMAC` 参数即为打印机的蓝牙 MAC 地址。

Step 1: 配置 AndroidManifest.xml

1.1 添加 NFC 权限

在 `<manifest>` 标签内添加：

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

1.2 声明 NFC 硬件特性（可选）

添加 `required="false"` 表示 App 也可以在没有 NFC 硬件的设备上运行：

```
<uses-feature android:name="android.hardware.nfc" android:required="false" />
```

1.3 设置 Activity 启动模式

为了确保 NFC 拉起时 Activity 只有一个实例，避免重复创建，需要设置

```
launchMode="singleTask"：
```

```

<activity
    android:name=".MainActivity"
    android:launchMode="singleTask"
    android:exported="true"
    android:label="@string/app_name">

```

1.4 添加 NFC NDEF_DISCOVERED Intent Filter

该 intent-filter 用于匹配 NFC 标签中的 NDEF URI 记录。当标签包含 `http://hprint.hpert.com/...` 格式的 URL 时，系统会将 Intent 分发给此 Activity：

```

<intent-filter>
    <action android:name="android.nfc.action.NDEF_DISCOVERED" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:scheme="http" />
    <data android:scheme="https" />
    <data android:host="hprint.hpert.com" />
</intent-filter>

```

完整 AndroidManifest 参考

```

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

    <!-- NFC 权限 -->
    <uses-permission android:name="android.permission.NFC" />
    <uses-feature android:name="android.hardware.nfc" android:required="false" />

    <!-- 已有蓝牙、USB 等权限保持不变 -->
    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
    <uses-permission android:name="android.permission.BLUETOOTH_SCAN" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    <application ...>
        <activity
            android:name=".MainActivity"
            android:launchMode="singleTask"
            android:exported="true"
            android:label="@string/app_name">

            <!-- 桌面启动入口 -->
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>

```

```

<!-- NFC NDEF_DISCOVERED -->
<intent-filter>
    <action android:name="android.nfc.action.NDEF_DISCOVERED" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:scheme="http" />
    <data android:scheme="https" />
    <data android:host="hprint.hpdt.com" />
</intent-filter>

<!-- Deep Link -->
<intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
    <data android:scheme="http"
          android:host="hprint.hpdt.com"
          android:pathPrefix="/hereset/index.html" />
</intent-filter>
</activity>
</application>
</manifest>

```

Step 2: 处理 NFC Intent

NFC 触碰拉起 App 有两种情况：

- **冷启动**：App 未运行，系统创建新进程并调用 `onCreate()`，此时 Intent 从 `onCreate` 的 `intent` 参数传入。
- **热启动**：App 已在后台运行，系统调用 `onNewIntent()` 将新的 Intent 传入。

因此需要在两个入口都处理 NFC Intent。

2.1 冷启动处理（onCreate）

在 `onCreate` 中调用 NFC 处理方法：

```

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    // ... 其他初始化代码 ...

    // 处理冷启动时的 NFC Intent
    handleNfcIntent(intent)
}

```

2.2 热启动处理（onNewIntent）

重写 `onNewIntent` 方法：

```
override fun onNewIntent(intent: Intent) {
    super.onNewIntent(intent)
    handleNfcIntent(intent)
}
```

2.3 NFC Intent 处理核心方法

```
private fun handleNfcIntent(intent: Intent) {
    // 只处理 NDEF_DISCOVERED 和 VIEW 两种 Action
    val action = intent.action
    if (action != NfcAdapter.ACTION_NDEF_DISCOVERED && action !=
Intent.ACTION_VIEW) {
        return
    }

    val uri = intent.data ?: return
    val url = uri.toString()
    Log.d("TAG", "NFC/DeepLink received: $url")

    // 从 URL 中提取蓝牙 MAC 地址
    val mac = parseMacFromNfcUrl(url)
    if (mac == null) {
        Toast.makeText(this, "NFC 数据无效, 未找到打印机地址",
Toast.LENGTH_LONG).show()
        return
    }

    Toast.makeText(this, "NFC 发现打印机: $mac", Toast.LENGTH_SHORT).show()

    // 开始蓝牙连接
    isConnectingState.value = true
    loadingMessageState.value = "NFC 蓝牙连接中..."
    lifecycleScope.launch {
        try {
            connectBluetooth(mac)
            updateConnectionStatus()
        } finally {
            isConnectingState.value = false
        }
    }
}
```

Step 3: 解析 URL 中的蓝牙 MAC 地址

NFC 标签中的 URL 包含打印机的蓝牙 MAC 地址，通过 `BMAC` 查询参数传递。

URL 格式：

```
http://hprint.hpert.com/hereset/index.html?Model=RTP-808BW&SN=R808BW262000442&BMAC=28:D4:1E:73:A4:19
```

解析方法：

```
/**
 * 从 NFC URL 中解析 BMAC (蓝牙 MAC 地址)
 * URL 格式: http://hprint.hpert.com/hereset/index.html?
Model=xxx&SN=xxx&BMAC=xx:xx:xx:xx:xx:xx
 */
private fun parseMacFromNfcUrl(url: String): String? {
    return try {
        val uri = Uri.parse(url)
        uri.getQueryParameter("BMAC")
    } catch (e: Exception) {
        Log.e("TAG", "Failed to parse NFC URL", e)
        null
    }
}
```

使用 `Uri.getQueryParameter("BMAC")` 可以直接提取指定参数的值，无需手动正则匹配。

Step 4: 使用 SDK 蓝牙连接

解析到 MAC 地址后，调用 SDK 的蓝牙连接接口进行连接。

4.1 蓝牙连接方法

```
private suspend fun connectBluetooth(address: String) {
    // 设置连接中状态
    ConnectionHolder.setConnecting()

    // 创建蓝牙连接对象
    val connection = BluetoothConnection()

    // 执行连接
    when (val result =
        connection.connect(ConnectionParams.Bluetooth(address))) {
        is Result.Success -> {
            // 连接成功，保存连接状态
            ConnectionHolder.setConnection(
                connection,
                ConnectionHolder.ConnectionType.BLUETOOTH,
```

```
        address
    )
    Toast.makeText(this, "连接成功", Toast.LENGTH_SHORT).show()
}
is Result.Failure -> {
    // 连接失败，保存错误信息
    ConnectionHolder.setError(result.error.message)
    Toast.makeText(
        this,
        "连接失败: ${result.error.message}",
        Toast.LENGTH_SHORT
    ).show()
}
}
}
```

4.2 ConnectionHolder 说明

`ConnectionHolder` 是一个全局单例，用于管理打印机连接状态，提供以下功能：

方法	说明
<code>setConnection(connection, type, info)</code>	设置连接成功状态
<code>setConnecting()</code>	设置连接中状态
<code>setError(message)</code>	设置连接失败状态
<code>isConnected()</code>	检查是否已连接
<code>disconnect()</code>	断开连接

连接类型枚举：

```
enum class ConnectionType {
    NONE, BLUETOOTH, WIFI, USB, SERIAL
}
```

Step 5: UI 更新

连接过程中和连接完成后，需要更新 UI 反馈给用户。

5.1 连接中状态

使用加载弹窗提示用户正在连接：

```
isConnectingState.value = true
loadingMessageState.value = "NFC 蓝牙连接中..."
```

在 Compose UI 中显示加载弹窗：

```
if (isLoading) {
    AlertDialog(
        onDismissRequest = {},
        confirmButton = {},
        title = { Text("请稍候...") },
        text = {
            Row(verticalAlignment = Alignment.CenterVertically) {
                CircularProgressIndicator(modifier = Modifier.size(20.dp))
                Spacer(modifier = Modifier.width(8.dp))
                Text(loadingMessage)
            }
        }
    )
}
```

5.2 连接完成状态

连接完成后调用 `updateConnectionStatus()` 刷新 UI：

```
private fun updateConnectionStatus() {
    isConnectedState.value = ConnectionHolder.isConnected()
    connectionStatusText.value = if (isConnectedState.value) {
        "已连接：${ConnectionHolder.connectionType.name}"
    } else {
        "请连接打印机"
    }
}
```

UI 中的状态卡片会根据连接状态自动改变颜色（已连接=主题色，未连接=灰色）。

5.3 错误提示

连接失败时通过 Toast 显示具体错误信息，方便用户排障。

完整代码参考

以下是 MainActivity 中与 NFC 相关的完整代码：

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
```

```

        super.onCreate(savedInstanceState)
        // ... 初始化代码 ...

        // 处理冷启动时的 NFC Intent
        handleNfcIntent(intent)
    }

    override fun onNewIntent(intent: Intent) {
        super.onNewIntent(intent)
        handleNfcIntent(intent)
    }

    /**
     * 处理 NFC Intent: 解析 URL, 提取 MAC, 蓝牙连接
     */
    private fun handleNfcIntent(intent: Intent) {
        val action = intent.action
        if (action != NfcAdapter.ACTION_NDEF_DISCOVERED && action !=
Intent.ACTION_VIEW) {
            return
        }

        val uri = intent.data ?: return
        val url = uri.toString()
        Log.d("TAG", "NFC/DeepLink received: $url")

        val mac = parseMacFromNfcUrl(url)
        if (mac == null) {
            Toast.makeText(this, "NFC 数据无效, 未找到打印机地址",
Toast.LENGTH_LONG).show()
            return
        }

        Toast.makeText(this, "NFC 发现打印机: $mac", Toast.LENGTH_SHORT).show()

        // 开始蓝牙连接
        isConnectingState.value = true
        loadingMessageState.value = "NFC 蓝牙连接中..."
        lifecycleScope.launch {
            try {
                connectBluetooth(mac)
                updateConnectionStatus()
            } finally {
                isConnectingState.value = false
            }
        }
    }

    /**

```

```

        * 从 NFC URL 中解析 BMAC
        * URL: http://hprint.hpert.com/hereset/index.html?
Model=xxx&SN=xxx&BMAC=xx:xx:xx:xx:xx:xx
    */
    private fun parseMacFromNfcUrl(url: String): String? {
        return try {
            val uri = Uri.parse(url)
            uri.getQueryParameter("BMAC")
        } catch (e: Exception) {
            Log.e("TAG", "Failed to parse NFC URL", e)
            null
        }
    }

    /**
    * SDK 蓝牙连接
    */
    private suspend fun connectBluetooth(address: String) {
        ConnectionHolder.setConnecting()
        val connection = BluetoothConnection()
        when (val result =
connection.connect(ConnectionParams.Bluetooth(address))) {
            is Result.Success -> {
                ConnectionHolder.setConnection(
                    connection,
                    ConnectionHolder.ConnectionType.BLUETOOTH,
                    address
                )
                Toast.makeText(this, "连接成功", Toast.LENGTH_SHORT).show()
            }
            is Result.Failure -> {
                ConnectionHolder.setError(result.error.message)
                Toast.makeText(
                    this,
                    "连接失败: ${result.error.message}",
                    Toast.LENGTH_SHORT
                ).show()
            }
        }
    }
}

```

需要导入的包

```
import android.nfc.NfcAdapter
import android.net.Uri
import android.bluetooth.BluetoothDevice
import com.printer.sdk.connection.BluetoothConnection
import com.printer.sdk.connection.ConnectionParams
import com.printer.sdk.model.Result
```

注意事项

1. **Android 12+ (API 31+) 蓝牙权限**：如果 `targetSdk >= 31`，需要同时声明 `BLUETOOTH_CONNECT` 和 `BLUETOOTH_SCAN` 权限，并在运行时动态申请。
2. **位置权限**：蓝牙扫描需要 `ACCESS_FINE_LOCATION` 或 `ACCESS_COARSE_LOCATION` 权限（Android 11 及以下版本）。
3. **NFC 权限是普通权限**：`NFC` 权限属于 `normal` 级别，在 `AndroidManifest` 中声明即可，无需运行时申请。
4. **模拟器测试**：Android 模拟器默认不支持 NFC，建议在真机上测试。
5. **URL 匹配范围**：`NDEF_DISCOVERED` 的 `intent-filter` 中如果只写 `scheme=http` 和 `host=hprint.hpert.com` 而不加 `pathPrefix`，可以匹配该域名下的所有路径，更具通用性。
6. **连接回调**：蓝牙连接是耗时操作，务必在协程中执行，避免阻塞 UI 线程。