一、NFC 基础知识

1.1 概念

NFC是Near Field Communication缩写，即近距离无线通讯技术。可以在移动设备、消费类电子产品、PC 和智能控件工具间进行近距离无线通信。

NFC 通常的通信距离是 4 厘米或更短，工作频率是 13.56MHz，传输速率是 106kbit/s ~ 848kbit/s。

通过 NFC 技术，可以使 Android 设备与 NFC Tag 之间或者其他 Android 设备之间传输小数据量的数据。

NFC Tag 分很多种，其中简单的只提供读写段，有的只能读、不能写；复杂的 Tag 可以支持一些数学运算，通过加密硬件来控制对 Tag 中特定数据段的读写；甚至一些 Tag 上有简单的操作系统，允许与 Tag 上执行的代码进行一些相对复杂的交互。

NFC 总是在一个发起者和一个被动目标之间发生。发起者发出近场无线电波，这个近场可以给被动目标供电。

发起者一般为 Android 设备，被动的目标包括不需要电源的标签、卡等，也可以是有电源的设备，如 Android 手机。

NFC 技术为手机支付提供了技术基础。

与蓝牙和 WIFI 技术相比，NFC 的通信带宽和距离都要小得多，但是它成本低，不需要电源支持，这些都是得天独厚的应用推广条件。

2.1 NFC 通信模式

2.1 NFC的三种工作模式

NFC的工作模式有三种，分别是读卡器模式（Reader/writer mode）、仿真卡模式(Card Emulation Mode)、点对点模式（P2P mode）。

2.1.1 读卡器模式

数据在NFC芯片中，可以简单理解成“刷标签”。本质上就是通过支持NFC的手机或其它电子设备从带有NFC芯片的标签、贴纸、名片等媒介中读写信息。通常NFC标签是不需要外部供电的。当支持NFC的外设向NFC读写数据时，它会发送某种磁场，而这个磁场会自动的向NFC标签供电。

2.1.2 仿真卡模式

数据在支持NFC的手机或其它电子设备中，可以简单理解成“刷手机”。本质上就是将支持NFC的手机或其它电子设备当成借记卡、公交卡、门禁卡等IC卡使用。基本原理是将相应IC卡中的信息凭证封装成数据包存储在支持NFC的外设中 。
在使用时还需要一个NFC射频器（相当于刷卡器）。将手机靠近NFC射频器，手机就会接收到NFC射频器发过来的信号，在通过一系列复杂的验证后，将IC卡的相应信息传入NFC射频器，最后这些IC卡数据会传入NFC射频器连接的电脑，并进行相应的处理（如电子转帐、开门等操作）。

2.1.3 点对点模式

该模式与蓝牙、红外差不多，用于不同NFC设备之间进行数据交换，不过这个模式已经没有有“刷”的感觉了。其有效距离一般不能超过4厘米，但传输建立速度要比红外和蓝牙技术快很多，传输速度比红外块得多，如过双方都使用Android4.2，NFC会直接利用蓝牙传输。这种技术被称为Android Beam。所以使用Android Beam传输数据的两部设备不再限于4厘米之内。
点对点模式的典型应用是两部支持NFC的手机或平板电脑实现数据的点对点传输，例如，交换图片或同步设备联系人。因此，通过NFC，多个设备如数字相机，计算机，手机之间，都可以快速连接，并交换资料或者服务。

NFC 的工作原理

1.3 NFC 与其他近距离通信技术的对比

当下的近距无线通信技术除 NFC 外，主要还包括射频识别（RFID)、蓝牙（Bluetooth）、紫蜂（ZigBee）、红外、 Wi-Fi 等技术。以上各项技术都有各自的优缺点，下图给出了 NFC 以及其他几种短距离无线通信技术在所列频段上性能的比较。

名称	NFC	RFID	ZigBee	蓝牙	红外	Wi-Fi
传输速率	424 kbps	2 Mbps	100 kbps	1 Mbps	115 kbps	11 ~ 54 Mbps
传输距离	1 ~ 20 cm	< 3 m	2 ~ 20 m	10 ~ 100 m	1 m	10 ~ 200 m
频　段	13.56 MHz	2.4 GHz	2.4 GHz	2.4 GHz	980 mm 红外光	2.4 GHz
功　耗	10 mA	低	5 mA	20 mA	低	10 ~ 50 mA
安全性	极高	中等	中等	高	无	低
成　本	低	中	中	中	低	高

对比项	NFC	蓝牙	红外
网络类型	点对点	单点对多点	点对点
有效距离	<=0.1m	<=10m，最新的蓝牙4.0有效距离可达100m	一般在1m以内，热技术连接，不稳定
传输速度	最大424kbps	最大24Mbps	慢速115.2kbps，快速4Mbps
建立时间	<0.1s	6s	0.5s
安全性	安全，硬件实现	安全，软件实现	不安全，使用IRFM时除外
通信模式	主动-主动/被动	主动-主动	主动-主动
成本	低	中	低

https://blog.csdn.net/qq_41522183

上图可以看出，NFC 技术具有极高的安全性，在短距离通信中具有性能优势，更重要的是成本较低。

与蓝牙相比：NFC操作简单，配对迅速

与RFID相比：NFC适用范围广泛、可读可写，能直接集成在手机中

与红外线相比：数据传输较快、安全性高、能耗低

NFC应用实例

在日常生活中我们接触过的有移动支付、电子票务、移动身份识别、防伪等应用场景。比如我们坐地铁刷卡，出入小区的门禁，再比如人所皆知的茅台、五粮液酒的防伪都使用了 NFC 功能。

Android 提供了 android.nfc 和 android.nfc.tech 包，它们对 NFC 技术进行了支持。常用类介绍如下：

1. NfcManager

Android 设备的 NFC 适配器管理器，可以通过 getSystemService(String) 获得对象实例。

NfcManager 可以获取到当前 Android 设备支持的所有 NFC 适配器列表。

2. NfcAdapter

代表设备的 NFC 适配器。NFC 适配器是进行 NFC 操作的入口。

通常情况下，每个 Android 设备只有一个 NFC 适配器，可以通过 NfcAdapter.getDefaultAdapter(android.content.Context) 方法或者 NfcManager.getDefaultAdapter() 方法来获取当前 Android 设备的 NFC 适配器。

3. NdefMessage

代表 NDEF 消息。NDEF 是 NFC Forum 定义的标准数据结构，用于在设备和 NFC Tags 之间传递数据。一个 NdefMessage 对象包含多个 NdefRecord 对象。

4. NdefRecord

代表一条记录。每条 NDEF 记录都有一个 MIME 数据类型，比如文本、URL、智慧型海报等。NDEF 记录被存放在 NDEF 消息中。

5. Tag

表示被检测到的 NFC 目标，可以是一个标签、一个卡片、一个钥匙扣等。

Android.nfc.tech 包中包含对 NFC Tag 进行查询和 I/O 操作的类。

如果 Android 设备没有关闭掉 NFC 功能，当设备的屏幕没有被锁定时，Android 设备会一直搜寻附件的 NFC Tag。当一个 NFC Tag 被检测到，一个包含该 NFC Tag 信息的 Tag 对象将被创建并且封装到一个 Intent 里，然后 NFC 发布系统将这个 Intent 用 startActivity 发送到已注册的用于处理这种类型的 Intent 的 Activity 中进行处理。

当 Android 设备扫描到一个 NFC Tag，通用的行为是自动搜寻最合适的 Activity 处理这个包含 Tag 对象的 Intent，而不需要用户来选择哪个 Activity 来处理。

因为设备扫描 NFC Tag 是在很短的范围和时间内，如果让用户选择的话，就有可能需要移动设备，这样将会打断这个扫描过程。因此，开发者应该开发只处理需要处理的 Tag 的 Activity，以防止发生让用户选择使用哪个 Activity 来处理的情况。

Android 系统提供了一个 Tag 发布系统（Tag Dispatch System）帮助分析扫描到的 NFC Tag，从中提取相关数据，封装到 Intent 并且定位到对这些数据有兴趣的 Activity。

如果同时有多个 Activity 都可以对封装了 Tag 数据的 Intent 进行处理，那么会出现一个选择列表，让用户来选择要处理的 Activity。

Tag 发布系统定义了三种 Intent，按照顺序优先级从高到低说明如下。

1）android.nfc.action.NDEF_DISCOVERED

当一个包含 NDEF 负载的 Tag 被检测到时，该 Intent 被启动，这是最高优先级的 Intent。

如果检测到的是一个未知的 Tag 或者不包含 NDEF 负载的 Tag，那么该 Intent 不会被启动。

若 NDEF_DISCOVERED Intent 已经被启动，则 TECH_DISCOVERED和TAG_DISCOVERED Intent 将不会被启动。

处理该 Intent 的 Activity 需要对应设置 intent-filter 属性，例如：

```
    <intent-filter>
        <action android:name="android.nfc.action.NDEF_DISCOVERED" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="text/plain"/>
    </intent-filter>
```

表明当前 Activity 可以处理 NDEF_DISCOVERED 类型的 Intent，但是其携带数据的类型需要是“text/plain”类型。

2）android.nfc.action.TECH_DISCOVERED

当一个包含 NDEF 负载的 Tag 被检测到，并且没有 Activity 处理 NDEF_DISCOVERED Intent 时，该 Intent 会被启动。若该 Intent 被启动，则 TAG_DISCOVERED 不会被启动。

3）android.nfc.action.TAG_DISCOVERED

当一个包含 NDEF 负载的 Tag 被检测到，并且没有 Activity 处理 NDEF_DISCOVERED 和 TECH_DISCOVERED Intent 时，或者 Tag 被检测为未知的，该 Intent 被启动。

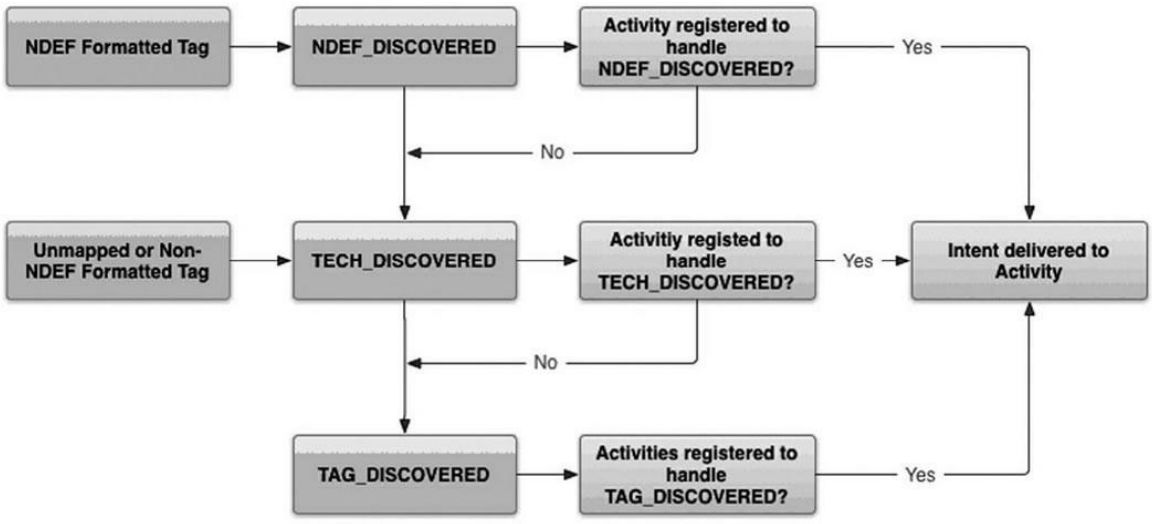


图 1 Tag发布系统运行图

要进行 NFC 访问，需要在工程的 AndroidManifest.xml 文件中添加如下代码：

1）用于获取 NFC 硬件访问权限。

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

2）指定最小 SDK 版本的代码。
支持 NFC 功能的最小 SDK 版本为 API Level 9，但是仅支持有限的 Tag 发布和访问 NDEF 信息，不支持其他 Tag 的输入输出操作。

API Level 10 增加了对 Tag 的读写方式，并添加了前台 NDEF 推数据的方式。

API Level 14 提供了将 NDEF 数据传送到其他设备的方式。建立 SDK 的最小版本要高于 10。

```
<uses-sdk android:minSdkVersion="10"/>
```

3）设置 uses-feature 属性，以便在 Google Play Store 发布时，仅使具有 NFC 硬件的设备可以搜索到。

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

近距离无线通信 (NFC) 是一组近距离无线技术，通常只有在距离不超过 4 厘米时才能启动连接。借助 NFC，您可以在 NFC 标签与 Android 设备之间或者两台 Android 设备之间共享小型负载。

标签的复杂度可能各有不同。简单标签仅提供读取和写入语义，有时可使用一次性可编程区域将卡片设置为只读。较复杂的标签可提供数学运算，还可使用加密硬件来验证对扇区的访问权限。最为复杂的标签可包含操作环境，允许与针对标签执行的代码进行复杂的互动。存储在标签中的数据也可以采用多种格式编写，但许多 Android 框架 API 都基于名为 NDEF（NFC 数据交换格式）的 NFC Forum 标准。

支持 NFC 的 Android 设备同时支持以下三种主要操作模式：

> 读取器/写入器模式：支持 NFC 设备读取和/或写入被动 NFC 标签和贴纸。
> 点对点模式：支持 NFC 设备与其他 NFC 对等设备交换数据；Android Beam 使用的就是此操作模式。
> 卡模拟模式：支持 NFC 设备本身充当 NFC 卡。然后，可以通过外部 NFC 读取器（例如 NFC 销售终端）访问模拟 NFC 卡。

二、卡片相关知识

2.1 卡片的分类

2.1.1 ID 卡

ID 卡又叫身份识别卡，**只有一个ID号，不可以存储任何数据**，是一种**不可写入的感应式卡。**卡号在封卡前写入后不可再更改，绝对确保卡号的唯一性和安全性。

ID卡与磁卡一样，都仅仅使用了“卡的号码”而已，卡内除了卡号外，无任何保密功能，其“卡号”是公开、裸露的。所以说ID卡就是“感应式磁卡”。

应用：主要应用在门禁系统、企业工牌

2.1.2 IC 卡

IC 卡又称集成电路卡，是智能卡的总称，它是将一个微电子芯片嵌入符合 ISO 7816 标准的卡基中，做成卡片形式。

IC 卡带有存储器可读写，普通的 IC 卡也叫储存器卡、逻辑加密卡。**IC卡数据的读取，写入均需相应的密码认证，数据可以分区，不同区用于不同的功能，可以有不同的密码保护。**IC 卡的加密和反复读写的特性，其安全性远大于ID卡，主要应用在地铁乘车费、校园一卡通、门禁卡。

IC卡所记录内容可反复擦写，分为**接触式和非接触式IC卡。**

射频IC卡的有两种卡型——Type A和type B型。其主要的区别在于载波调制深度及二进制数的编码方式，**二代身份证属于 type B型卡。**

2.1.3 M1 卡

M1，全称 Mifare classic 1K，是菲利浦下属子公司恩智浦出品的芯片缩写，目前该公司的M1芯片与国产芯片相兼容，利用PVC封装M1芯片、感应天线，然后压制成型后而制作的卡即是智能卡行业所说的M1卡，**属于非接触式IC卡。**

M1卡，优点是可读可写的多功能卡，缺点是：价格稍贵，感应距离短，

适合非定额消费系统、停车场系统、门禁考勤系统等。

2.1.4 CPU 卡

CPU 卡芯片是一个微处理器，它的功能相当于一台微型计算机。

如果不强调的话，**CPU卡也是IC卡的一种，是高级版的IC卡，具有信息处理的功能**，优点是存储空间大、读取速度快、支持一卡多用功能等特点 。如果不强调无线的话，电话卡中 SIM 卡就是典型的 CPU 卡。

CPU 卡可适用于金融、保险、交警、政府行业等多个领域。

2.1.5 RFID卡

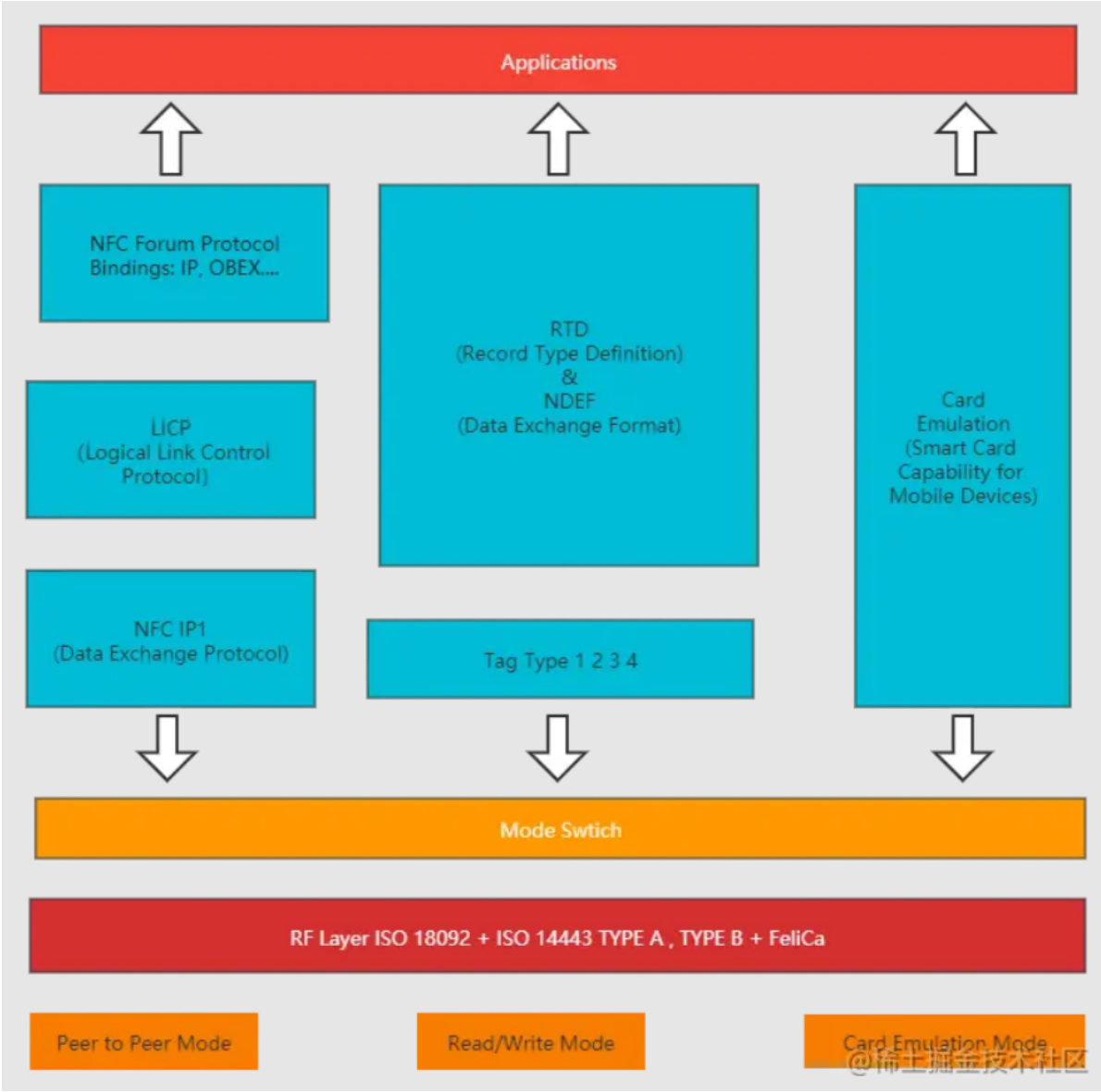
是指非接触式类电子卡片/标签，包括有ID卡、IC卡和NFC卡以及其它等电子卡/标签。他们主要的区别在于工作频段。

卡类型	芯片名称	频率	重复擦写	备注说明
IC卡	M1卡	13.56MHZ	✗	普通卡，0块（卡号块）出厂为锁定状态，不可用于复制，只能授权，物业卡为些卡，俗称授权卡
IC卡	UID卡	13.56MHZ	✔	经典复制卡，可以反复擦写，无法穿防火墙
IC卡	CUID卡	13.56MHZ	✔	穿防火墙IC复制卡，对于UID卡复制成功却刷卡无效的情况下，可以使用此卡复制
IC卡	FUID卡	13.56MHZ	✗	一次性穿防火墙IC复制卡，对于CUID卡复制成功刷卡无效的情况下，可以使用此卡复制
IC卡	UFUID卡	13.56MHZ	✗	一次性穿防火墙IC复制卡，但区别在于此卡支持手动锁卡，没有锁卡之前就是一张UID卡，锁卡之后不支持擦写
IC卡	GTU卡	13.56MHZ	✔	可以自动复位数据的IC复制卡，用于滚动码防复制系统，支持反复擦写
IC卡	GDMIC卡	13.56MHZ	✔	可以自动复位数据的IC复制卡，用于滚动码防复制系统，支持反复擦写，功能同GTU卡，生产厂商不同
ID卡	T5577卡 EM4305卡 5200卡	125KHZ	✔	可擦写复制的ID卡
ID卡	F8268卡	125KHZ	✔	穿防火墙ID复制卡，用于ID卡复制成功刷卡无效的情况

三、NFC 协议基础

3.1 NFC 技术标准

NFC技术标准包括四层，如图：



RT Layer ISO 层

射频层，包括一些射频协议（ISO 18092，ISO 1443 TYPE A，ISO 1443 TYPE B 和 Felica)

Mode Switch 层

模式切换层，可以理解为衔接层，射频层协议和 NFC 协议层之间的映射，标准的切换

NFC Protocol 层

P2P模式、卡模拟模式、数据交互协议(NDEF)、记录类型定义协议(RTD)、Type 类型等

Applications 层

应用层

3.2 NFC 常见的标准规范

- ISO 14443(A/B)
- NFCIP-1
- MIFARE
- Felica

3.3 NFC Tag

3.3.1 NFC 论坛定义的 Tag

属性	类型 1	类型 2	类型 3	类型 4	类型 5
标准	ISO/IEC 14443A	ISO/IEC 14443A	ISO/IEC 18092 JIS X 6319-4 FELICA	ISO/IEC 14443A ISO/IEC 14443B	ISO/IEC 15693
存储器	96 字节到 2 KB	48 字节到 2 KB	2 KB	32 KB	高达 8 KB
数据速率	106 kbit/s	106 kbit/s	212 kbit/s, 424 kbit/s	106 kbit/s, 212 kbit/s, 424 kbit/s	26.48 kbit/s
能力	读 重写 只读	读 重写 只读	读 重写 只读	读 重写 只读 工厂配置	读 重写 只读
防冲突	无	有	有	有	有
注释	简单，性价比高	-	成本更高，适合 于复杂应用	-	疏耦合区域

Type 1

基于ISO14443A。具有可读、重新写入的能力，用户可将其配置为只读。存储能力为96字节，用来存网址URL或其他小量数据。然而，内存可被扩充到2k字节。通信速度为106 kbit/s。

典型的芯片类Topaz 512 (BCM20203)

Type 2

也是基于ISO14443A，具有可读、重新写入的能力，用户可将其配置为只读。其基本内存大小为48字节，但可被扩充到2k字节。通信速度也是106 kbit/s。

典型的芯片类型为Mifare Ultralight / Ultralight C, NTag203/210/213/215/216等。

Type 3

数据通讯速度为212 Kbit/s。故此类标签较为适合较复杂的应用，成本较高。

Type 4

此类标签被定义为与ISO14443A、B标准兼容。制造时被预先设定为可读/可重写、或者只读。芯片内存更大，通信速度介于106 kbit/s和424 kbit/s之间。适用于多种应用典型的芯片类型为 DESFire系列，容量2K,4K,8K不等。

Type 5

此类标签，是近几年被定义的最新类型的NFC标签，对应的RFID协议是ISO15693系列RFID芯片。NFC Forum引入此系列的芯片，是为了满足日益增长的各种远距离、小型化的NFC标签及其应用。

3.3.2 NXP 特定的 TAG 类型

除了 NFC 论坛定义的 Tag 类型之外，其他厂商也提供了其他的自定义私有 Tag 类型，其中用的最广泛的是 NXP 的 Mifare Classical Tag：

Mifare Classical Tag 基于 ISO 14443A 标准，

- 可读可写，可配置成只读
- 可变内存 `192/768/3584 bytes`
- 传输速率 `106kb/s`
- 支持数据冲突保护
- 市场上有兼容的产品，如 NXP MIFARE Classic 1K 、MIFARE Classic 4K 和 Classic Mini

3.4 NDEF 协议

3.4.1 NDEF 概述

NDEF，全称 NFC Data Exchange Format，是**轻量级的紧凑的二进制格式**，是 NFC 论坛定义的用于 NFC 数据交换的通用数据格式。

NDEF 使 NFC 的各种功能更加容易的使用各种支持的标签类型进行数据传输，这是由于 NDEF 已经封装了 NFC 标签的种类细节信息，因此应用不用关心是在与何种标签进行通信。

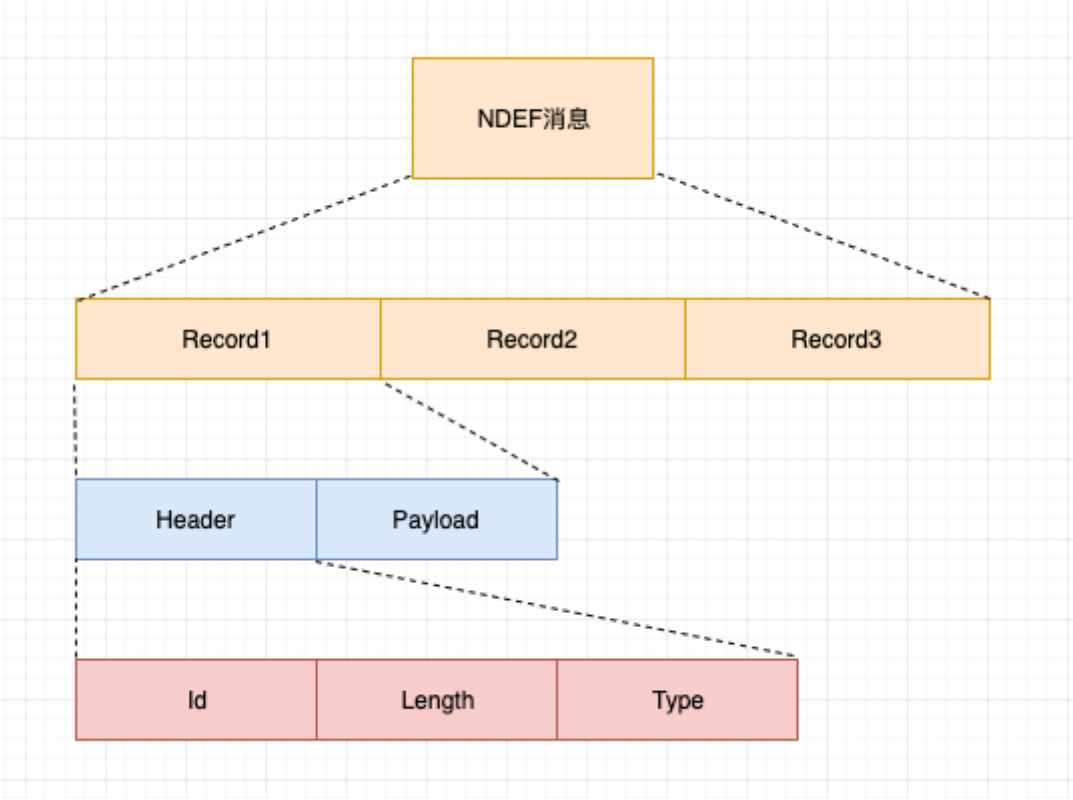
NDEF 的主要目的有：

- 封装任意形式的文件和实体（如加密数据，XML文件等）
- 封装未知大小的文件和实体
- 组合按某种顺序出现的多个文件和实体（如含有附件的标准文件）
- 同时需要注意小负载的封装不应该增加系统的负荷

使用场景：

上层应用产生由一个或多个文件生成的NDEF信息，该消息交由底层LLC层传送给对方，对方可以接受后直接处理或作为中间阶段写入Tag中。当其他设备接近该tag时，会读到该tag中的内容，并把读到的NDEF消息传给上层应用分析和处理。

3.4.2 NDEF 的组成



NDEF 消息主要由若干个记录（Record ）组成，Record 中的内容遵循 RTD（Record Type Definition）协议。

Record 报头包含3个部分：Identifier、Length、和 Type 。

Record 的内容可以是 URL，MIME 或者 NFC 自定义的数据类型。使用 NFC 自定义的数据类型，载荷内容（Payload）必须被定义在一个 NFC 记录类型定义（RTD）文档中。

NDEF 头部组成:

表 2-2 NDEF 头部组成

名　称	NDEF 协议对应	含　义
Length	PAYLOAD LENGTH	Payload 的长度，单位是字节（octet）
Type	Type Value: TNF TYPE	类型域，用来指定载荷的类型
Identifier	ID LENGTH ID	可选的指定载荷是否带有一个 NDEF 记录

一个标准的 NDEF Record：

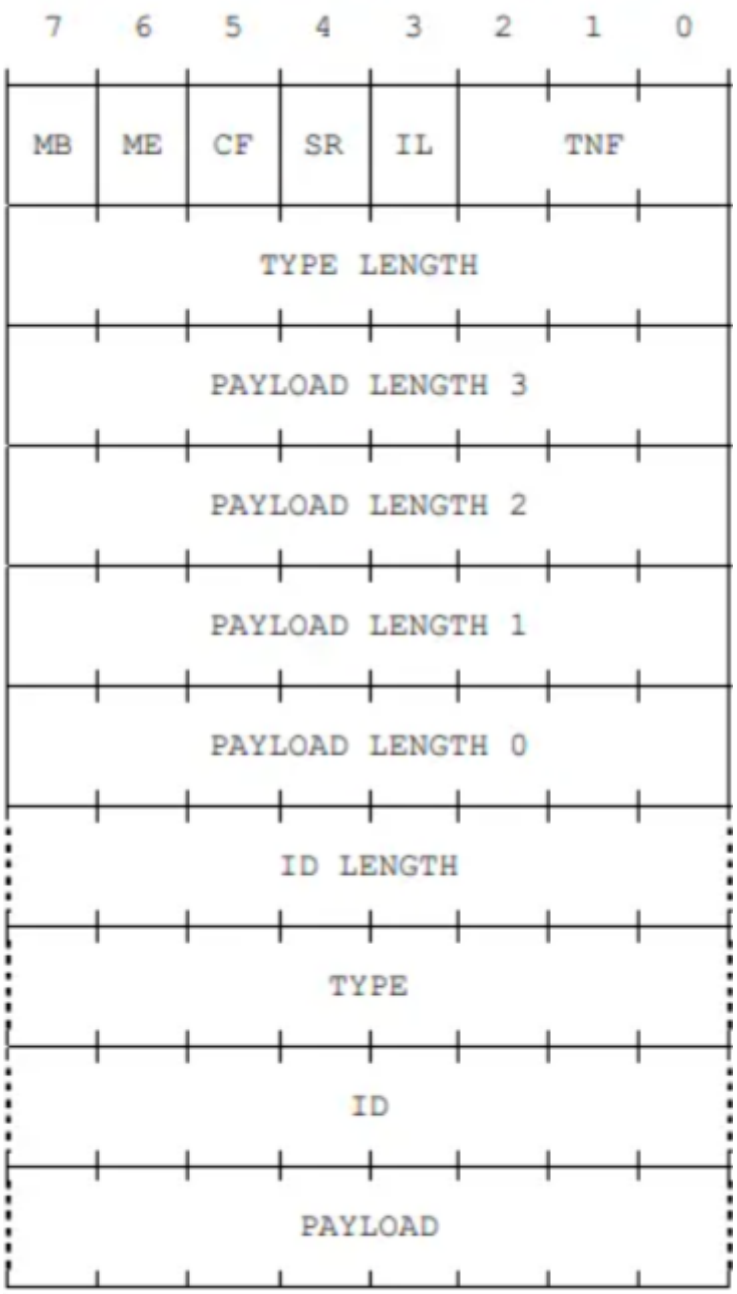
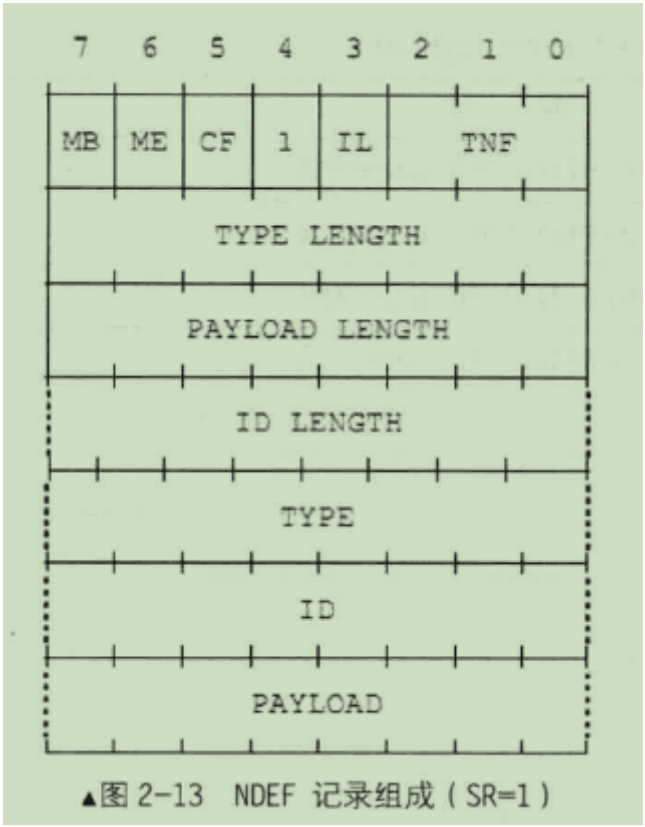


Figure 3. NDEF Record Layout

(从左往右，7代表第0位)

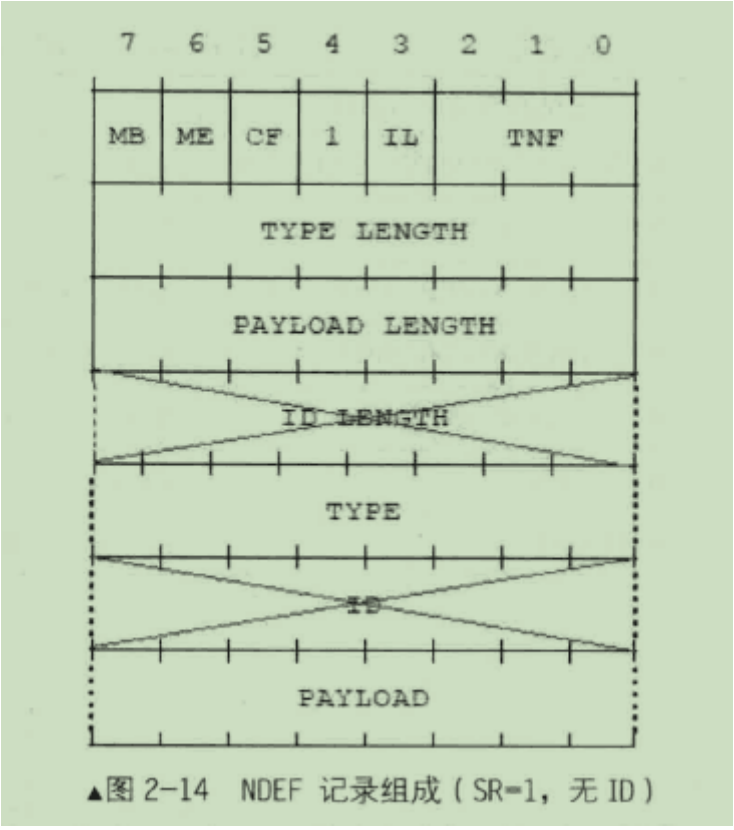
- 第一个字节是一些标志位，具体含义看下图 [**Record 的第一个字节的含义**] 介绍
- 第二个字节是 `TYPE LENGTH`，表示 `Type` 的长度
- 第三个字节是 `PAYLOAD LENGTH` 表示 `payload` 的长度，如果 `SR` 为 0，则 `PAYLOAD` 长度占 `4` 个字节，否则占 `1` 个字节
- 紧跟着的是 ID 长度

当 SR=1 时，表示该记录是短记录，此时记录组成图如下：



▲图 2-13 NDEF 记录组成（SR=1）

当 IL=1 时，表示该记录没有 ID_LENGTH 和 ID 域，如下图（该记录是 SR=1&&IL=1的情况）：



▲图 2-14 NDEF 记录组成（SR=1，无 ID）

Record 的第一个字节的含义：

表 2-3 NDEF 记录第一字节含义

Label	Size	Means
MB（Message Begin）	1 bit	1/true 表示该记录是首记录，the first record in NDEF message 0/false 表示该记录非首记录
ME（Message End）	1 bit	1 表示该记录是尾记录 0 表示该记录前还有其他记录，more records are ahead
CF（Chunk Flag）	1 bit	0 表示该记录未被切块 1 表示该记录被切块
SR（Short Record）	1 bit	0 表示该记录是长记录，PAYLOAD LENGTH 占 4Byte 1 表示该记录是短记录，PAYLOAD LENGTH 占 1Byte
IL/ID_LENGTH （identification length）	1 bit	0 表示该记录没有 ID_LENGTH 和 ID 域，is omitted 1 表示该记录有 ID_LENGTH 和 ID 域
TNF（Type Name Format）	3 bits	Type 类型，如表所示（TNF Field Values） 如 0x01(即二进制 001)表示 well-known type

第一个字节后三位`TNF`的类型，具体含义如下：

表 2-4 NDEF TNF 类型

值	类型名称格式（TNF）	映　射
0x00	TNF_EMPTY	退化到 ACTION_TECH_DISCOVERED 类型的 Intent 对象
0x01	TNF_WELL_KNOWN	依赖于类型字段中设置的记录类型定义（RTD）的 MIME 类型或 URI
0x02	TNF_MIME_MEDIA	基于类型字段的 MIME 类型
0x03	TNF_ABSOLUTE_URI	基于类型字段的 URI
0x04	TNF_EXTERNAL_TYPE	基于类型字段中 URN 的 URI。URN 是缩短的格式（<domain_name>:<service_name>）被编码到 NDEF 类型中 Android 会把这个 URN 映射成以下格式的 URI:vnd.android.nfc://ext/<domain_name>:<service_name>
0x05	TNF_UNKNOWN	退化到 ACTION_TECH_DISCOVERED 类型的 Intent 对象
0x06	TNF_UNCHANGED	退化到 ACTION_TECH_DISCOVERED 类型的 Intent 对象
0x07	Reserved	保留

TNF 为 Type 的类型，该类型定义了 TNF_ABSOLUTE_URI 等7种类型的 Type 类型，其中，TNF_WELL_KNOWN 中又定义了一系列的 RTD。

表 2-5 TNF_WELL_KNOWN 所支持的 RTD 和它们的映射

记录类型定义（RTD）	映　射
RTD_ALTERNATIVE_CARRIER	退化到 ACTION_TECH_DISCOVERED 类型的 Intent 对象
RTD_HANDOVER_CARRIER	退化到 ACTION_TECH_DISCOVERED 类型的 Intent 对象
RTD_HANDOVER_REQUEST	退化到 ACTION_TECH_DISCOVERED 类型的 Intent 对象
RTD_HANDOVER_SELECT	退化到 ACTION_TECH_DISCOVERED 类型的 Intent 对象
RTD_SMART_POSTER	基于负载解析的 URI
RTD_TEXT	text/plain 类型的 MIME
RTD_URI	基于有效负载的 URI

Record 是 NDEF 消息中的最小单元。为避免发送端发送数据过大导致接收端无法处理，需要对 Record 进行分片处理，并将第一个Record 中的CF标记置1，中间的 Record 也要设置CF为1，且 type_length 和 IL 都为0，TNF 为 0x6（unchanged），结束的 Record 中CF 清零，且 type_length 和 IL 都为0。

3.5 RTD 协议

3.5.1 概述

RTD，全称 NFC Record Type Definition ， NFC 记录类型的定义。

3.5.2 常见的 RTD 类型

1. NFC 文本 RTD（T），可携带 unicode 字符串，记录描述文本信息。
2. NFC URI RTD（U），存储网络地址，邮件或者电话号码
3. NFC 智能海报 RTD（Sp），用于将 URL，短信或者电话号码编入NFC论坛标签，及如何在这些设备之间传递这些信息。
4. NFC 通用控制 RTD
5. NFC 签名 RTD

RTD_TEXT

RTD_TEXT 即文本记录类型，用来存储 tag 中的文本信息。

内容格式如下：

3.2.1 Syntax

The NFC Forum Well Known Type [NDEF], [NFC RTD] for the Text record is “T” (in NFC binary encoding: 0x54).

The data content is as follows:

Table 2. Text Record Content Syntax

Offset (bytes)	Length (bytes)	Content
0	1	Status byte. See Table 3.
1	<n>	ISO/IANA language code. Examples: “fi”, “en-US”, “fr-CA”, “jp”. The encoding is US-ASCII.
n+1	<m>	The actual text. Encoding is either UTF-8 or UTF-16, depending on the status bit.

The Status bit encodings are as described in Table 3. Any value marked RFU SHALL be ignored, and any software writing these bits SHALL use the value zero for these bits.

Table 3. Status Byte Encodings

Bit number (0 is LSB)	Content
7	0: The text is encoded in UTF-8 1: The text is encoded in UTF16
6	RFU (MUST be set to zero)
5..0	The length of the IANA language code.

@稀土掘金技术社区

第一个表中第一个字节是状态码，具体规则看第二个表，如果**第7位(最高位)是1，说明是UTF16编码，然后0到5位是语言编码的长度，第6位是固定为0。**

如果判断出`TYPE`是`TEXT`类型的，后续内容就按照以上图标中的格式进行解析。

RTD_URI

RTD_URI 用来描述从 NFC 兼容的 tag 中取得一个 URI，或者在两个 NFC 设备之间传递 URI 数据，同时也提供另一种在另一个 NFC 元素（如智能海报）里存储 URI 的方法。

内容格式如下：

3.2.1 URI Record Type

The Well Known Type for an URI record is “U” (0x55 in the NDEF binary representation).

The structure of an URI record is described below.

Table 2. URI Record Contents

Name	Offset	Size	Value	Description
Identifier code	0	1 byte	URI identifier code	The URI identifier code, as specified in Table 3.
URI field	1	N	UTF-8 string	The rest of the URI, or the entire URI (if identifier code is 0x00).

@稀土掘金技术社区

Identiffier code 为 URI 前缀标识符，其值如下图。

如果`Type`是`URI`类型，后面第一个字节解析如下图(`URI`前缀)，截取协议中部分，总共有`255`种，

3.2.2 URI Identifier Code

In order to shorten the URI, the first byte of the record data describes the protocol field of an URI. The following table MUST be used to encode and decode the URI, though applications MAY use the 0x00 value to denote no prefixing when encoding, regardless of whether there actually is a suitable abbreviation code.

For explanations of the different protocols, please refer to the protocol documentations themselves. NFC devices are not required to support any particular protocol.

Table 3. Abbreviation Table

Decimal	Hex	Protocol
0	0x00	N/A. No prepending is done, and the URI field contains the unabridged URI.
1	0x01	http://www.
2	0x02	https://www.
3	0x03	http://
4	0x04	https://
5	0x05	tel:
6	0x06	mailto:
7	0x07	ftp://anonymous:anonymous@
8	0x08	ftp://ftp.
9	0x09	ftps://

@稀土掘金技术社区

RTD_Smart Poster

RTD_Smart Poster 是将 URL，SMS 等信息综合到了一个Tag中，RTD_Smart Poster 的记录内容实际上就是 RTD_TEXT 和 RTD_URI 等记录的组合。

Table 4. Example for a Simple URI

Offset	Content	Length	Explanation
0	0xD1	1	NDEF header. TNF = 0x01 (Well Known Type). SR=1, MB=1, ME=1
1	0x02	1	Record name length (2 bytes)
2	0x12	1	Length of the Smart Poster data (18 bytes)
3	“Sp”	2	The record name
5	0xD1	1	NDEF header. TNF = 0x01, SR=1, MB=1, ME=1
6	0x01	1	Record name length (1 byte)
7	0x0E	1	The length of the URI payload (14 bytes)
8	“U”	1	Record type: “U”
9	0x01	1	Abbreviation: “http://www.”
10	“nfc-forum.org”	13	The URI itself.

@稀土掘金技术社区

3.6 NDEF 解析实例

3.6.1 RTD_TEXT 解析

假设目前又一个纯 RTD_TEXT 类型的 NEDF Hex 串为：

> D1 01 0F 54 02 65 6E 68 65 6C 6C 6F 2C 77 6F 72 6C 64 21

表 2-6 RTD_TEXT NDEF 解析实例

Hex 值	解　析	说　明
D1 = 11010001 B	1 该 NDEF 只有 1 个 Record 2 该 NDEF 为短记录，且无 ID 和 ID_Length 3 NFC Forum well-known type [NFC RTD]	参考图 2-12 和表 2-3，NDEF 记录标准 根据 NDEF Record 中第一字节定义
01	Type_Length = 1	参考表 2-8 和表 2-9
0F	Payload_Lenght = 15	参考表 2-8 和表 2-9
54	Type = 54 = "T", Type 为文本记录类型	参考表 2-8 和表 2-9 由 Type_Length=1 决定
02	语言码长度 IANA=2	参考表 2-8 和表 2-9，Payload 内容

续表

Hex 值	解　析	说　明
65 6E	65 6E = "en"，即编码为 US-ASCII 码	参考表 2-8 和表 2-9，Payload 内容
68 65 6C 6C 6F 2C 77 6F 72 6C 64 21	hello,world!	参考表 2-8 和表 2-9 Payload 内容

即最终的解析结果为"hello，world!"

3.6.2 RTD_URI 解析

假设目前又一个纯 RTD_URI 类型的 NEDF Hex 串为：

> D1 01 0A 55 01 62 61 69 64 75 2E 63 6F 6D

表 2-7　RTD_URI NDEF 解析实例

Hex 值	解　析	说　明
D1 = 11010001 B	1 该 NDEF 只有 1 个 Record 2 该 NDEF 为短记录，且无 ID 和 ID_Length 3 NFC Forum well-known type [NFC RTD]	参考图 2-12 和表 2-3，NDEF 记录标准根据 NDEF Record 中第一字节定义
01	Type_Length = 1	参考表 2-10 和表 2-11
0A	Payload_Lenght = 10	参考表 2-10 和表 2-11
55	Type = 55 = "U", Type 为 URI 记录类型 URI Record Type Definition	参考表 2-10 和表 2-11 由 Type_Length=1 决定
01	http://www.	参考表 2-10 和表 2-11
62 61 69 64 75 2E 63 6F 6D	baidu.com	参考表 2-10 和表 2-11，Payload 内容

四、Android 中的 NFC 相关知识

将 `NDEF` 数据与 `Android` 结合使用时，会有两个主要用例

- 从 `NFC` 标签读取 `NDEF` 数据
  - 从 `NFC` 标签读取 `NDEF` 数据的操作由标签调度系统进行处理，该系统会分析已发现的 `NFC` 标签，对相应数据进行适当分类，然后启动对分类后的数据感兴趣的应用。如果某个应用想要处理扫描到的 `NFC` 标签，则可以声明 Intent 过滤器，并请求对数据进行处理
- 使用 Android Beam™ 将 `NDEF` 消息从一台设备传输到另一台设备
  - 借助 `Android Beam™` 功能，设备可以将 `NDEF` 消息推送到另一台设备，方法是将两台设备靠在一起。与蓝牙等其他无线技术相比，这种互动可提供更简便的数据发送方式，因为使用 `NFC` 时无需手动发现设备并将其配对。当两台设备之间的距离近到一定范围内时，系统会自动开始连接。`Android Beam` 功能通过一组 `NFC API` 提供，因此任何应用都可以在设备间传输信息。例如，通讯录、浏览器和 `YouTube` 应用可使用 `Android Beam` 在多台设备之间共享联系人信息、网页和视频

注意：要下载完整的 `NDEF` 规范，请转到 NFC Forum 规范和应用文档网站，并参阅创建常见类型的 NDEF 记录，查看有关如何构造 NDEF 记录的示例。

4.1 Nfc 开发相关的类和包

Android对NFC的支持主要在 android.nfc 和android.nfc.tech 两个包中。

android.nfc 包中主要类如下：

- NfcManager

用于管理Android设备中指出的所有NFC Adapter，因为大多安卓设备只支持一个NFC Adapter的原因，所有在这种情况下可以直接使用getDefaultAapater 获取系统支持的Adapter。

- NfcAdapter

为一个NFC Adapter 对象，用于定义一个Intent使系统在检测到NFC Tag时通知你定义的Activity，并提供用来注册forground tag 消息发送的方法等。

- NdefMessage

描述NDEF格式的信息，实际上我们写入NFC标签的就是NdefMessage对象。

- NdefRecord

描述NDEF信息的一个信息段，一个NdefMessage可能包含一个或者多个NdefRecord。

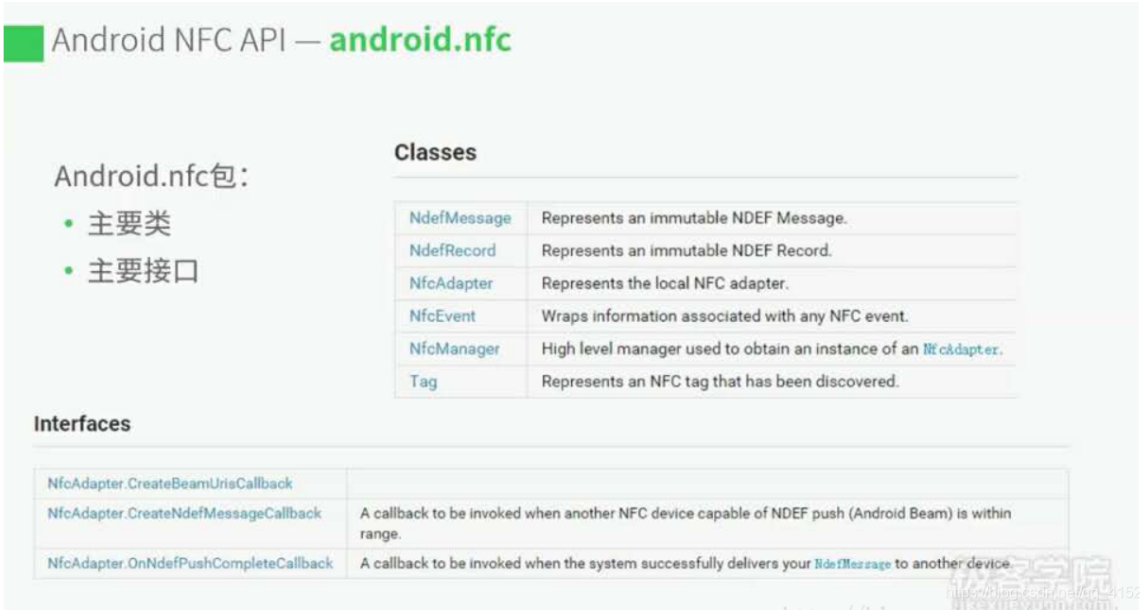
NdefMessage 和 NdefRecord 是 Android NFC 技术的核心类，无论读写NDEF格式的NFC标签，还是通过Android Beam技术传递Ndef格式的数据，都需要这两个类。

- Tag

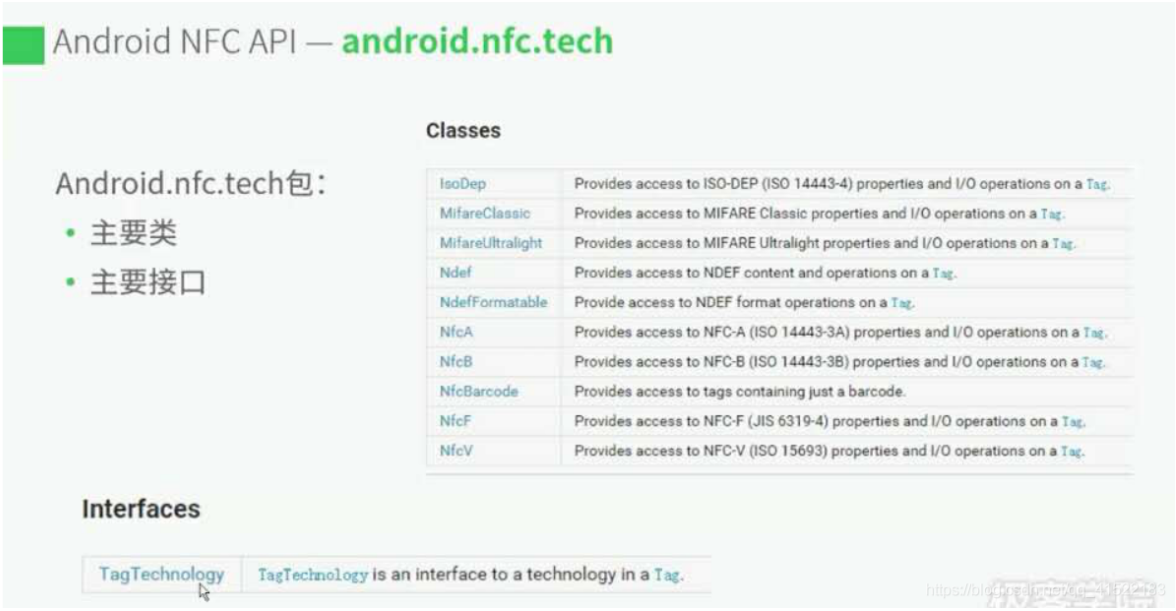
代表一个被动式Tag对象，可以代表一个标签，卡片，钥匙扣等。当Android设备检测到一个Tag时，会创建一个Tag对象，将其放在Intent对象，然后发送到相应的Activity。

android.nfc.tech 包中则定义了可以对Tag进行的读写操作的类，这些类按照其使用的技术类型可以分成不同的类如：NfcA, NfcB, NfcF,以及MifareClassic 等。

Android.nfc主要类和主要接口：



Android.nfc.tech 主要类和主要接口



4.2 Android Beam

在Android 中将NDEF数据从一台设备发送到另一部NFC设备的功能的实现是通过Android Beam的技术来实现的。Android Beam是一个基于近场通信所做的新功能，是一款应用程序，旨在最大程度地利用NFC技术，可让用户对几乎任何东西进行分享，无论是联系人、图片、网页链接还是YouTube链接。

借助 Android Beam™ 功能，设备可以将 NDEF 消息推送到另一台设备，方法是将两台设备靠在一起。与蓝牙等其他无线技术相比，这种互动可提供更简便的数据发送方式，因为使用 NFC 时无需手动发现设备并将其配对。当两台设备之间的距离近到一定范围内时，系统会自动开始连接。Android Beam 功能通过一组 NFC API 提供，因此任何应用都可以在设备间传输信息。例如，通讯录、浏览器和 YouTube 应用可使用 Android Beam 在多台设备之间共享联系人信息、网页和视频。

4.3 标签调度系统

4.3.1 概念

NFC 标签调度写（NFC Tag Dispatch System）是一种通过预先定义好的 Tag 或 NDEF 消息来启动应用程序的机制，即，当扫描到一个 NFC Tag 时，如果 Intent 中注册了对应的应用程序，那么在处理该 Tag 信息时就会启动该应用。当存在多个可以处理该 Tag 信息的应用时，系统会弹出一个 Activity Choose ，供用户选择开启哪个应用。

一般，在Android 设备中，只要在设备的设置菜单中 NFC 未被禁用，Android 设备都会在非锁屏的状态下搜索 NFC 信息。**一旦检测到有 NFC Tag 的“触碰”动作，Tag 中的 type（NFC 类型）数据将会通过一个 Intent 进行装载。**此时，NFC 标签调度系统会自动调研能够处理该 Tag 信息的应用。因此，应用程序需要注册它们可以处理的 NFC 数据类型以便 NFC 标签调度系统调用。

在 Android 系统中，NFC 标签调度系统的工作流程为：

1. 解析 `NFC` 标签并确定 `MIME` 类型或 `URI`（后者用于标识标签中的数据负载）。
2. 将 `MIME` 类型或 `URI` 与负载一起封装到 `Intent` 中。
3. 根据 `Intent` 启动 `Activity`。

4.3.2 标签映射

NFC 标签涉及多种技术，也可以通过许多不同的方式将数据写入 NFC 标签中。Android 对 NFC Forum 定义的 NDEF 标准的支持最完备。

NDEF 数据封装在包含一条或多条记录（`NdefRecord`）的消息（`NdefMessage`）内。每条 NDEF 记录的格式都必须正确，符合您要创建的记录所属的类型对应的规范。

Android 还支持其他类型的不包含 NDEF 数据的标签，您可以使用 `android.nfc.tech` 软件包中的类处理这些标签。在处理这些其他类型的标签时，您需要通过编写自己的协议栈来与标签进行通信，因此，我们建议您尽可能使用 NDEF，以简化开发，同时最大限度地支持 Android 设备。

> 要下载完整的 NDEF 规范，请转到 NFC Forum 规范和应用文档网站，并参阅创建常见类型的 NDEF 记录，查看有关如何构造 NDEF 记录的示例。

当 Android 设备扫描包含 NDEF 格式数据的 NFC 标签时，它会解析该消息并尝试确定数据的 MIME 类型或起标识作用的 URI。为此，系统需要读取 `NdefMessage` 中的第一条 `NdefRecord`，以确定如何解读整个 NDEF 消息（一个 NDEF 消息可能具有多条 NDEF 记录）。

在格式正确的 NDEF 消息中，第一条 `NdefRecord` 包含以下字段：

3 位 TNF（类型名称格式）

表示如何解读可变长度类型字段。表 1 中介绍了有效的值。

可变长度类型

介绍了记录的类型。如果使用 `TNF_WELL_KNOWN`，那么请使用此字段来指定记录类型定义 (RTD)。表 2 中介绍了有效的 RTD 值。

可变长度 ID

记录的唯一标识符。此字段并不经常使用，但如果您需要对标签进行唯一标识，则可为其创建 ID。

可变长度负载

您要读取或写入的实际数据负载。一个 NDEF 消息可以包含多条 NDEF 记录，因此不要假定 NDEF 消息的第一条 NDEF 记录中就有完整的负载。

标签调度系统使用 TNF 和类型字段来尝试将 MIME 类型或 URI 映射到 NDEF 消息。如果成功映射，它会将相关信息与实际负载一起封装到 `ACTION_NDEF_DISCOVERED` Intent 内。不过，在某些情况下，标签调度系统无法根据第一条 NDEF 记录来确定数据的类型。如果 NDEF 数据无法映射到 MIME 类型或 URI，或者 NFC 标签不包含 NDEF 数据，就会出现上述情况。在此类情况下，标签调度系统会转而将含有标签技术相关信息的 `Tag` 对象及负载封装到 `ACTION_TECH_DISCOVERED` Intent 中。

表 1 介绍了标签调度系统如何将 TNF 和类型字段映射到 MIME 类型或 URI，还介绍了哪些 TNF 无法映射到 MIME 类型或 URI。如果无法映射，标签调度系统会回退到 `ACTION_TECH_DISCOVERED`。

例如，如果标签调度系统遇到 `TNF_ABSOLUTE_URI` 类型的记录，则会将该记录的可变长度类型字段映射到 URI 中。标签调度系统将此 URI 连同与标签有关的其他信息（如负载）一起封装在 `ACTION_NDEF_DISCOVERED` Intent 的数据字段中。另一方面，如果标签调度系统遇到 `TNF_UNKNOWN` 类型的记录，则会转而创建一个 Intent，用于封装与标签技术相关的信息。

表 1. 支持的 TNF 及其映射

类型名称格式 (TNF)	映射
`TNF_ABSOLUTE_URI`	基于类型字段的 URI。
`TNF_EMPTY`	回退到 `ACTION_TECH_DISCOVERED`。
`TNF_EXTERNAL_TYPE`	基于类型字段中 URN 的 URI。URN 以缩短形式 (`*<domain_name>:<service_name>*`) 编码到 NDEF 类型字段中。Android 会以如下形式将此映射到 URI：`vnd.android.nfc://ext/*<domain_name>:<service_name>*`。
`TNF_MIME_MEDIA`	基于类型字段的 MIME 类型。
`TNF_UNCHANGED`	在第一条记录中无效，因此会回退到 `ACTION_TECH_DISCOVERED`。
`TNF_UNKNOWN`	回退到 `ACTION_TECH_DISCOVERED`。
`TNF_WELL_KNOWN`	MIME 类型或 URI，具体取决于您在类型字段中设置的记录类型定义 (RTD)。如需详细了解可用的 RTD 及其映射，请参阅表 2。

表 2. TNF_WELL_KNOWN 支持的 RTD 及其映射：

记录类型定义 (RTD)	映射
`RTD_ALTERNATIVE_CARRIER`	回退到 `ACTION_TECH_DISCOVERED`。
`RTD_HANDOVER_CARRIER`	回退到 `ACTION_TECH_DISCOVERED`。
`RTD_HANDOVER_REQUEST`	回退到 `ACTION_TECH_DISCOVERED`。
`RTD_HANDOVER_SELECT`	回退到 `ACTION_TECH_DISCOVERED`。
`RTD_SMART_POSTER`	基于负载解析结果的 URI。
`RTD_TEXT`	`text/plain` 的 MIME 类型。
`RTD_URI`	基于负载的 URI。

4.3.3 如何将 NFC 标签分发到应用

当标签调度系统创建完用于封装 NFC 标签及其标识信息的 Intent 后，它会将该 Intent 发送给感兴趣的应用，由这些应用对其进行过滤。如果有多个应用可处理该 Intent，系统会显示 Activity 选择器，供用户选择要使用的 Activity。

标签调度系统定义了三种 Intent，按优先级从高到低列出如下：

1. **ACTION_NDEF_DISCOVERED**

   如果扫描到包含此 Intent 的 Activity，并且可识别其类型，则使用此 Intent 启动 Activity。这是优先级最高的 Intent，NFC标签调度系

   统会尽可能尝试使用此 Intent 启动 Activity，在找不到这个Intent时才会尝试使用其他 Intent。

2. **ACTION_TECH_DISCOVERED**

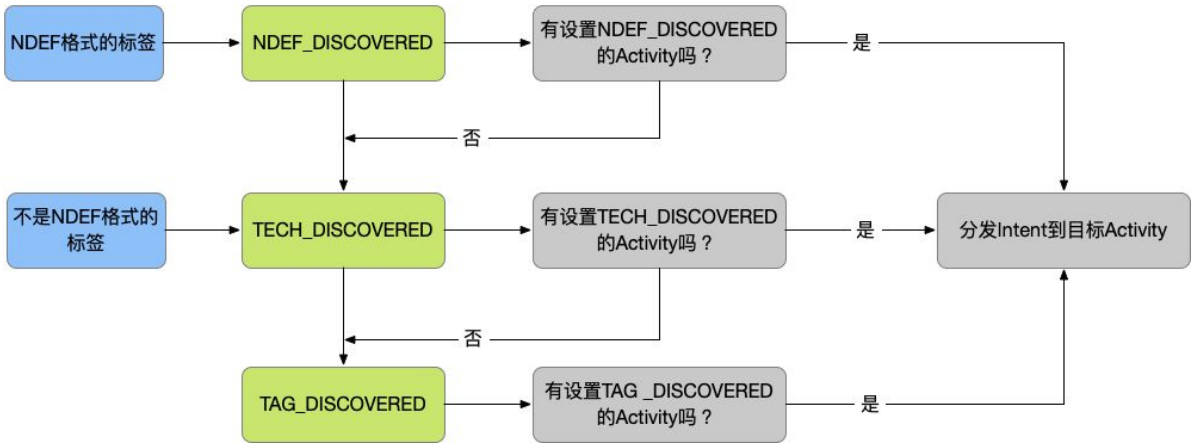
如果没有登记要处理 `ACTION_NDEF_DISCOVERED` Intent 的 Activity，则标签调度系统会尝试使用此 Intent 来启动应用。此外，如果扫描到的标签包含无法映射到 MIME 类型或 URI 的 NDEF 数据，或者该标签不包含 NDEF 数据，但它使用了已知的标签技术，那么也会直接启动此 Intent（无需先启动 `ACTION_NDEF_DISCOVERED`）。

3. **ACTION_TAG_DISCOVERED**

   如果没有处理 `ACTION_NDEF_DISCOVERED` 或者 `ACTION_TECH_DISCOVERED` Intent 的 Activity，则使用此 Intent 启动 Activity。

标签调度系统的基本工作方式如下：

1. 在解析 NFC 标签（`ACTION_NDEF_DISCOVERED` 或 `ACTION_TECH_DISCOVERED`）时，尝试使用由标签调度系统创建的 Intent 启动 Activity。
2. 如果不存在过滤该 Intent 的 Activity，则尝试使用下一优先级的 Intent（`ACTION_TECH_DISCOVERED` 或 `ACTION_TAG_DISCOVERED`）启动 Activity，直到应用过滤该 Intent 或者直到标签调度系统试完所有可能的 Intent。
3. 如果没有应用过滤任何 Intent，则不执行任何操作。



如果想处理所有的NFC标签，上面3个可以在清单文件中都进行设置。

###

4..3.4 过滤 NFC Intent

要在扫描到您打算处理的 NFC 标签时启动您的应用，您的应用可以在 Android 清单中过滤一个、两个或所有三个 NFC Intent。不过，您通常需要过滤 `ACTION_NDEF_DISCOVERED` Intent，以最有力地控制应用在何时启动。如果没有应用过滤 `ACTION_NDEF_DISCOVERED`，或者负载不是 NDEF，`ACTION_TECH_DISCOVERED` Intent 会取代 `ACTION_NDEF_DISCOVERED`。`ACTION_TAG_DISCOVERED` 通常因过于笼统而不适合过滤。

以下介绍了如何过滤各类 Intent：

ACTION_NDEF_DISCOVERED

要过滤 `ACTION_NDEF_DISCOVERED` Intent，请声明 Intent 过滤器以及要过滤的数据类型。

以下示例展示了如何过滤 MIME 类型为 `text/plain` 的 `ACTION_NDEF_DISCOVERED` Intent：

```
    <intent-filter>
        <action android:name="android.nfc.action.NDEF_DISCOVERED"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain" />
    </intent-filter>
```

以下示例展示了如何过滤采用 `https://developer.android.com/index.html` 形式的 URI：

```
    <intent-filter>
        <action android:name="android.nfc.action.NDEF_DISCOVERED"/>
        <category android:name="android.intent.category.DEFAULT"/>
       <data android:scheme="http"
                  android:host="developer.android.com"
                  android:pathPrefix="/index.html" />
    </intent-filter>
```

ACTION_TECH_DISCOVERED

如果您的 Activity 过滤 `ACTION_TECH_DISCOVERED` Intent，您必须创建一个 XML 资源文件，用它在 `tech-list` 集内指定您的 Activity 所支持的技术。如果 `tech-list` 集是标签所支持的技术（可通过调用 `getTechList()` 来获取）的子集，则您的 Activity 会被视为一个匹配项。

例如，如果扫描到的标签支持 MifareClassic、NdefFormatable 和 NfcA，为了使它们与您的 Activity 匹配，您的 `tech-list` 集必须指定所有这三种技术，或者其中的两种或一种技术。

以下示例定义了所有技术。您可以移除自己不需要的技术。将此文件（你可以随便命名）保存到 `<project-root>/res/xml` 文件夹中。

```
    <resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
        <tech-list>
            <tech>android.nfc.tech.IsoDep</tech>
            <tech>android.nfc.tech.NfcA</tech>
            <tech>android.nfc.tech.NfcB</tech>
            <tech>android.nfc.tech.NfcF</tech>
            <tech>android.nfc.tech.NfcV</tech>
            <tech>android.nfc.tech.Ndef</tech>
            <tech>android.nfc.tech.NdefFormatable</tech>
            <tech>android.nfc.tech.MifareClassic</tech>
            <tech>android.nfc.tech.MifareUltralight</tech>
        </tech-list>
    </resources>
```

您还可以指定多个 `tech-list` 集。每个 `tech-list` 集都是独立的；如果任意一个 `tech-list` 集是由 `getTechList()` 返回的技术的子集，则您的 Activity 会被视为一个匹配项。这为匹配技术提供了 `AND` 和 `OR` 语义。以下示例展示了如何与支持 NfcA 和 Ndef 技术的标签或者支持 NfcB 和 Ndef 技术的标签相匹配：

```
    <resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
        <tech-list>
            <tech>android.nfc.tech.NfcA</tech>
            <tech>android.nfc.tech.Ndef</tech>
        </tech-list>
        <tech-list>
            <tech>android.nfc.tech.NfcB</tech>
            <tech>android.nfc.tech.Ndef</tech>
        </tech-list>
    </resources>
```

在您的 `AndroidManifest.xml` 文件中，在 `<activity>` 元素的 `<meta-data>` 元素中指定您刚刚创建的资源文件，如以下示例所示：

```
    <activity>
    ...
    <intent-filter>
        <action android:name="android.nfc.action.TECH_DISCOVERED"/>
    </intent-filter>

    <meta-data android:name="android.nfc.action.TECH_DISCOVERED"
        android:resource="@xml/nfc_tech_filter" />
    ...
    </activity>
```

ACTION_TAG_DISCOVERED

要过滤 `ACTION_TAG_DISCOVERED`，请使用以下 Intent 过滤器：

```
<intent-filter>
        <action android:name="android.nfc.action.TAG_DISCOVERED"/>
    </intent-filter>
```

4.3.5 从 Intent 中获取信息

如果某个 Activity 由于 NFC Intent 而启动，您可以从该 Intent 中获取有关扫描到的 NFC 标签的信息。Intent 可以包含以下 extra，具体取决于扫描到的标签：

- `EXTRA_TAG`（必需）：一个 `Tag` 对象，表示扫描到的标签。
- `EXTRA_NDEF_MESSAGES`（可选）：从标签中解析出的一组 NDEF 消息。此 extra 对于 `ACTION_NDEF_DISCOVERED` Intent 而言是必需的。
- `EXTRA_ID`（可选）：标签的低级别 ID。

以下示例展示了如何检查 `ACTION_NDEF_DISCOVERED` Intent 并从 Intent extra 获取 NDEF 消息：

```
    @Override
    protected void onNewIntent(Intent intent) {
        super.onNewIntent(intent);
        ...
        if (NfcAdapter.ACTION_NDEF_DISCOVERED.equals(intent.getAction())) {
            Parcelable[] rawMessages =
```

```
                intent.getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES);
            if (rawMessages != null) {
                NdefMessage[] messages = new NdefMessage[rawMessages.length];
                for (int i = 0; i < rawMessages.length; i++) {
                    messages[i] = (NdefMessage) rawMessages[i];
                }
                // Process the messages array.
                ...
            }
        }
    }
```

或者，您可以从 Intent 中获取 `Tag` 对象，该对象包含负载并允许您枚举标签的技术：

```
    Tag tag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
```

4.4 前台调度系统

NFC 前台调度系统是一种用于在运行对的程序中（前台呈现的Activity）处理 Tag 的技术，即前台调度系统机制允许 Activity 拦截 Intent 对象，并且声明该 Activity 的优先级比其他的处理 Intent 对象 的 Activity 高。

NFC 前台调度系统的作用就是，在你打开一个可以处理NFC标签的Activity时，NFC的消息会优先发送给当前的Activity，不论当前的Activity设置的是哪一个`intent-filter`。

使用前台调度系统的步骤如下：

1、在 Activity 的 `onCreate()` 方法中添加以下代码：

创建一个 `PendingIntent` 对象，这样 Android 系统会使用扫描到的标签的详情对其进行填充。

```
    PendingIntent pendingIntent = PendingIntent.getActivity(
        this, 0, new Intent(this,
getClass()).addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP), 0);
```

声明 Intent 过滤器，以处理您要拦截的 Intent。前台调度系统会对照设备扫描标签时所获得的 Intent 来检查所指定的 Intent 过滤器。

如果匹配，那么应用会处理该 Intent。
如果不匹配，那么前台调度系统会回退到 Intent 调度系统。

指定 Intent 过滤器和技术过滤器的 `null` 数组，以指明要过滤所有回退到 `TAG_DISCOVERED` Intent 的标签。

以下代码段会处理 `NDEF_DISCOVERED` 的所有 MIME 类型。您应只处理需要的内容

```
    IntentFilter ndef = new IntentFilter(NfcAdapter.ACTION_NDEF_DISCOVERED);
        try {
            ndef.addDataType("*/*");    /* Handles all MIME based dispatches.
                                           You should specify only the ones that
you need. */
        }
        catch (MalformedMimeTypeException e) {
            throw new RuntimeException("fail", e);
        }
       intentFiltersArray = new IntentFilter[] {ndef, };
```

设置应用要处理的一组标签技术。调用 `Object.class.getName()` 方法以获取要支持的技术的类。

```
    techListsArray = new String[][] { new String[] { NfcF.class.getName() } };
```

2、替换以下 Activity 生命周期回调，并添加相应逻辑，以分别在 Activity 失去 (`onPause()`) 焦点和重新获得 (`onResume()`) 焦点时启用和停用前台调度。`enableForegroundDispatch()` 必须从主线程调用，并且只能在 Activity 在前台运行时调用（在 `onResume()` 中调用可确保这一点）。您还需要实现 `onNewIntent` 回调以处理扫描到的 NFC 标签中的数据。

```
    public void onPause() {
        super.onPause();
        adapter.disableForegroundDispatch(this);
    }

    public void onResume() {
        super.onResume();
        adapter.enableForegroundDispatch(this, pendingIntent, intentFiltersArray,
techListsArray);
    }

    public void onNewIntent(Intent intent) {
        Tag tagFromIntent = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
        //do something with tagFromIntent
    }
```

4.5 标签技术

将 NFC 标签与 Android 设备结合使用时，用于读取和写入标签数据的主要格式是 NDEF。当设备扫描具有 NDEF 数据的标签时，Android 会尽可能在解析消息和通过 `NdefMessage` 传递该消息方面提供支持。

不过，在某些情况下，您扫描的标签可能不包含 NDEF 数据，或者 NDEF 数据无法映射为 MIME 类型或 URI。**在这些情况下，您需要直接开启与标签的通信，并使用自己的协议（以原始字节形式）对标签执行读写操作。**

Android 通过 `android.nfc.tech` 软件包对这些用例提供一般性支持，如表 1 所述。您可以使用 `getTechList()` 方法确定标签支持的技术，还可以使用 `android.nfc.tech` 提供的一个类来创建相应的 `TagTechnology` 对象。

表 1. 支持的标签技术

类	说明
`TagTechnology`	这是所有标签技术类都必须实现的接口。
`NfcA`	提供对 NFC-A (ISO 14443-3A) 属性和 I/O 操作的访问权限。
`NfcB`	提供对 NFC-B (ISO 14443-3B) 属性和 I/O 操作的访问权限。
`NfcF`	提供对 NFC-F (JIS 6319-4) 属性和 I/O 操作的访问权限。
`NfcV`	提供对 NFC-V (ISO 15693) 属性和 I/O 操作的访问权限。
`IsoDep`	提供对 ISO-DEP (ISO 14443-4) 属性和 I/O 操作的访问权限。
`Ndef`	提供对 NDEF 格式的 NFC 标签上的 NDEF 数据和操作的访问权限。
`NdefFormatable`	为可设置为 NDEF 格式的标签提供格式化操作。

Android 设备还可以选择支持以下标签技术。

表 2. 可选择支持的标签技术

类	说明
`MifareClassic`	提供对 MIFARE Classic 属性和 I/O 操作的访问权限（如果此 Android 设备支持 MIFARE）。
`MifareUltralight`	提供对 MIFARE Ultralight 属性和 I/O 操作的访问权限（如果此 Android 设备支持 MIFARE）。

当设备扫描包含 NDEF 数据但无法映射为 MIME 或 URI 的标签时，标签调度系统会尝试使用 `ACTION_TECH_DISCOVERED` Intent 来启动 Activity。在扫描到包含非 NDEF 数据的标签时，也会使用 `ACTION_TECH_DISCOVERED`。如果标签调度系统无法解析标签数据，此回退功能可让您直接处理标签数据。

使用标签技术的基本步骤如下：

1. 过滤一个 `ACTION_TECH_DISCOVERED` Intent，指定您要处理的标签技术。

   通常，如果 NDEF 消息无法映射为 MIME 类型或 URI，或者扫描到的标签不包含 NDEF 数据，那么标签调度系统会尝试启动 `ACTION_TECH_DISCOVERED` Intent。

2. 应用收到 Intent 时，会从该 Intent 获取 `Tag` 对象

   ```
   Tag tagFromIntent = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
   ```

3. 通过调用 `android.nfc.tech` 软件包中类的 `get` 工厂方法，获取 `TagTechnology` 的实例。您可以枚举支持的标签技术，只需在调用 `get` 工厂方法之前调用 `getTechList()` 即可。

   例如，要从 `Tag` 获取 `MifareUltralight` 的实例，请执行以下操作：

   ```
   MifareUltralight.get(intent.getParcelableExtra(NfcAdapter.EXTRA_TAG));
   ```

五、Android 中 NFC 开发步骤

5.1 申请权限

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

5.2 添加识别 NFC 标签

当android设备扫描到一个NFC标签时，会自动寻找最适合的Activity来处理这个Tag。在Activity中添加intent-filter标签，当扫描到NFC设备时系统会打开此Activity。配置Activity的launchMode的属性为singleTop，并添加 类型支持信息。

```
<activity
        android:name=".MainActivity"
        android:launchMode="singleTop"
        android:screenOrientation="portrait">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
        <meta-data android:name="android.nfc.action.TECH_DISCOVERED"
                android:resource="@xml/filter_nfc">
        </meta-data>
 </activity>
```

5.3 初始化适配器

NFC开发需要使用安卓系统提供的NfcAdapter对象，使用该对象管理NFC设备。

```
        //获取NfcAdapter对象，此方法与获取蓝牙适配器对象类似
        mNfcAdapter = NfcAdapter.getDefaultAdapter(getApplicationContext());
        if (mNfcAdapter == null) {
            Toast.makeText(this, "该设备不支持nfc", Toast.LENGTH_SHORT).show();
            finish();
            return;
        }
        if (!mNfcAdapter.isEnabled()) {
            startActivity(new Intent("android.settings.NFC_SETTINGS"));
            Toast.makeText(this, "设备未开启nfc", Toast.LENGTH_SHORT).show();
        }
```

5.4 启用NFC前台调度

```
 @Override
    protected void onResume() {
        super.onResume();
        //一旦截获NFC消息，就会通过PendingIntent调用窗口
        PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, new
Intent(this,getClass()).addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP), 0);
        IntentFilter[] intentFilters = new IntentFilter[]{};
        //用于打开前台调度（拥有最高的权限），当这个Activity位于前台（前台进程），即可调用这
个方法开启前台调度
        mNfcAdapter.enableForegroundDispatch(this, pendingIntent, intentFilters,
null);
    }
```

5.5 接收数据

当手机端检测接收到NFC设备数据时，在onNewIntent方法中即可接受到数据。

在onNewIntent方法参数intent对象中获取数据：

```
@Override
    protected void onNewIntent(Intent intent) {
        super.onNewIntent(intent);
        if (NfcAdapter.ACTION_NDEF_DISCOVERED.equals(intent.getAction())
                || NfcAdapter.ACTION_TECH_DISCOVERED.equals(intent.getAction())
                || NfcAdapter.ACTION_TAG_DISCOVERED.equals(intent.getAction()))
{
            Tag tag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
            //读取NFC的id
            String id = ByteArrayToHexString(tag.getId());

            //获取tag中的数据信息
            String[] tagTechList = tag.getTechList();
            if (tagTechList != null) {
                for (int i = 0; i < tagTechList.length; i++) {

stringBuilder.append("*").append(tagTechList[i]).append("*").append("\n");
                    stringBuilder.append(readTech(tag, tagTechList[i],intent));
                }
            }
            Parcelable[] rawArray =
intent.getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES);
            if (rawArray != null) {
                //获取NDEF描述信息
                NdefMessage mNdefMsg = (NdefMessage) rawArray[0];
                //获取NDEF记录信息
                NdefRecord mNdefRecord = mNdefMsg.getRecords()[0];
                if (mNdefRecord != null) {
                    String readResult = new String(mNdefRecord.getPayload(),
"UTF-8");
                }
            }
        }
    }
```

5.6 发送数据

```
/**
* 向NFC发送数据
* data要发送的数据
* intent onNewIntent方法回调的Intent对象

public static void writeNFCToTag(String data, Intent intent) throws IOException,
FormatException {
        //获取Tag对象
        Tag tag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
        Ndef ndef = Ndef.get(tag);
        //连接
        ndef.connect();
        NdefRecord ndefRecord = null;
        if (android.os.Build.VERSION.SDK_INT >=
android.os.Build.VERSION_CODES.LOLLIPOP) {
            ndefRecord = NdefRecord.createTextRecord(null, data);
        }
        //数据格式打包
        NdefRecord[] records = {ndefRecord};
        NdefMessage ndefMessage = new NdefMessage(records);
        //发送数据
        ndef.writeNdefMessage(ndefMessage);
}
```

5.7 关闭NFC前台调度

之前在Activity的onResume方法中启用NFC的前台调度，相应的，在onPuse方法中关闭NFC的前台调度。

```
//调用disableForegroundDispatch方法关闭前台调度
if(mNfcAdapter != null){
     mNfcAdapter.disableForegroundDispatch(this);
}
```

过程：初始化 -> 设置系统调度 -> 系统调用 onNewIntent(Intent intent) -> Tag 读写流程 -> 最后取消系统调度

每次检测到 Tag 时，Activity 的生命周期回调：

> onPause() -> onNewIntent() -> onResume()