课程报告

Linux NFC管理机制与驱动开发

作 者 姓 名： 焦 珊

学科、 专业： 软件工程

学 号： 31817046

指 导 教 师： 吴国伟

完 成 日 期： 2019.5

大连理工大学

Dalian University of Technology

目录

**[1 NFC的起源与发展](#_Toc23701_WPSOffice_Level1)** **[3](#_Toc23701_WPSOffice_Level1)**

[1.1 起源](#_Toc11075_WPSOffice_Level2) [3](#_Toc11075_WPSOffice_Level2)

**[2 NFC的工作原理](#_Toc11075_WPSOffice_Level1)** **[4](#_Toc11075_WPSOffice_Level1)**

[2.1 工作原理结构图](#_Toc167_WPSOffice_Level2) [4](#_Toc167_WPSOffice_Level2)

[2.2 NFC三种工作模式结构图](#_Toc32561_WPSOffice_Level2) [5](#_Toc32561_WPSOffice_Level2)

**[3 NFC数据格式](#_Toc167_WPSOffice_Level1)** **[6](#_Toc167_WPSOffice_Level1)**

[3.1 NFC数据格式解析图](#_Toc3040_WPSOffice_Level2) [6](#_Toc3040_WPSOffice_Level2)

**[4 NFC协议栈](#_Toc32561_WPSOffice_Level1)** **[8](#_Toc32561_WPSOffice_Level1)**

[4.1 NFC协议栈结构图](#_Toc32668_WPSOffice_Level2) [8](#_Toc32668_WPSOffice_Level2)

**[5 NFC工作模式之源码分析](#_Toc3040_WPSOffice_Level1)** **[11](#_Toc3040_WPSOffice_Level1)**

[5.1 NFC源码分析之R/W工作模式](#_Toc32508_WPSOffice_Level2) [11](#_Toc32508_WPSOffice_Level2)

**[6 NFC读写程序的设计](#_Toc32668_WPSOffice_Level1)** **[17](#_Toc32668_WPSOffice_Level1)**

[6.1 NFC读程序设计](#_Toc14672_WPSOffice_Level2) [17](#_Toc14672_WPSOffice_Level2)

[6.2 NFC写程序设计](#_Toc18540_WPSOffice_Level2) [18](#_Toc18540_WPSOffice_Level2)

**[7 NFC的应用](#_Toc32508_WPSOffice_Level1)** **[20](#_Toc32508_WPSOffice_Level1)**

[7.1 支付的应用](#_Toc27266_WPSOffice_Level2) [20](#_Toc27266_WPSOffice_Level2)

[7.2 安防的应用](#_Toc14572_WPSOffice_Level2) [21](#_Toc14572_WPSOffice_Level2)

[7.3 标签的应用](#_Toc1756_WPSOffice_Level2) [21](#_Toc1756_WPSOffice_Level2)

**[8 NFC与其他技术的比较](#_Toc14672_WPSOffice_Level1)** **[21](#_Toc14672_WPSOffice_Level1)**

[8.1 NFC与RFID的比较](#_Toc11256_WPSOffice_Level2) [21](#_Toc11256_WPSOffice_Level2)

[8.2 NFC与蓝牙的比较](#_Toc8334_WPSOffice_Level2) [22](#_Toc8334_WPSOffice_Level2)

[8.3 NFC与红外的比较](#_Toc9467_WPSOffice_Level2) [22](#_Toc9467_WPSOffice_Level2)

**[9 实验过程](#_Toc18540_WPSOffice_Level1)** **[22](#_Toc18540_WPSOffice_Level1)**

# NFC的起源与发展

## 1.1 起源

根据有关资料显示，大约在 2003 年的时候，索尼公司和当时的飞利浦半导体( 现恩智浦 NXP 半导体) 进行合作，计划基于非接触式射频卡技术研发一种更加安全快捷的并且能与之兼容的无线通讯技术。经过几个月的研发后，双方联合对外发布了一种兼容 ISO14443 非接触式卡协议的无线通讯技术，取名为 NFC( Near Field Communication) ，具体通信规范称作 NFCIP-1 规范。在发布 NFC 技术没多久，双方向欧洲电脑制造商协会( ECMA) 提交标准草案，申请成为近场通信标准并很快被认可为 ECMA-340 标准，紧接着借助 ECMA 向 ISO/IEC 提交了标准申请并最终被认可为 ISO/IEC18092 标准。

NFC 称为近距离无线通讯技术, 运行于10cm距离内。其传输速度有106kbit/s, 212kbit/s, 424kbit/s三种。在非接触式射频识别( RFID) 技术的基础上，结合无线互联技术研发而成，它为我们日常生活中越来越普及的各种电子产品提供了一种十分安全快捷的通信方式。NFC 中文名称中的“近场”是指临近电磁场的无线电波。因为无线电波实际上就是电磁波，所以它遵循麦克斯韦方程，电场和磁场在从发射天线传播到接收天线的过程会一直交替进行能量转换，并在进行转换时相互增强，例如我们的手机所使用的无线电信号就是利用这种原理进行传播的，这种方法称作远场通信。而在电磁波 10 个波长以内，电场和磁场是相互独立的，这时的电场没有多大意义，但磁场却可以用于短距离通讯，我们称之为近场通信。

# NFC的工作原理

## 2.1 工作原理结构图

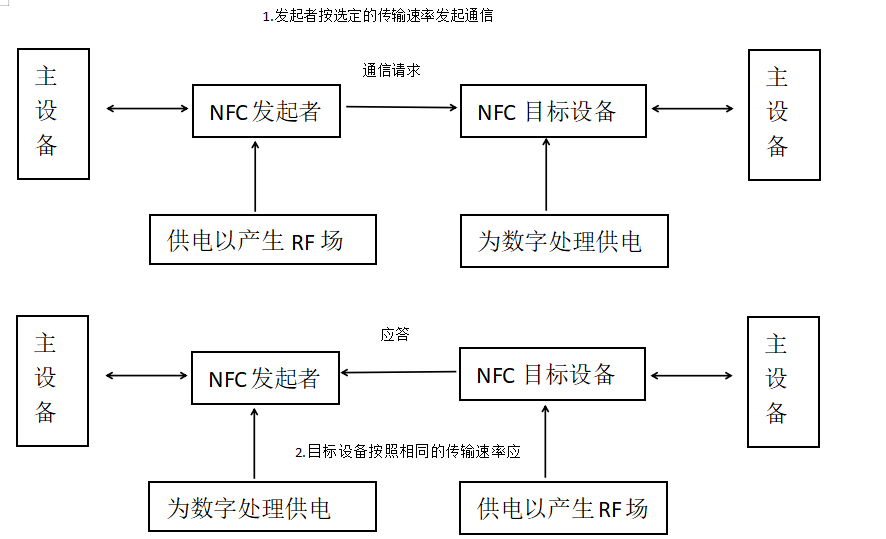


图 1 Linux NFC 工作原理之主动模式结构图

Linux NFC( Near Field Communication) 是一种短距高频的无线电技术，NFCIP-1 标准规定 NFC 的通信距离为 10 厘米以内，运行频率为 13．56MHz，传输速度有 106 Kbit/s、212 Kbit/s 或者 424Kbit / s 三种。NFCIP-1 标准详细规定 NFC 设备的传输速度、编解码方法、调制方案以及射频接口的帧格式，此标准中还定义了 NFC的传输协议，其中包括启动协议和数据交换方法等。NFC 工作模式分为被动模式和主动模式,如上图1所示主动模式。

NFC工作模式之主动模式，发起设备和目标设备在向对方发送数据时，都必须主动产生射频场，所以称为主动模式，它们都需要供电设备来提供产生射频场的能量。这种通信模式是对等网络通信的标准模式，可以获得非常快速的连接速率。

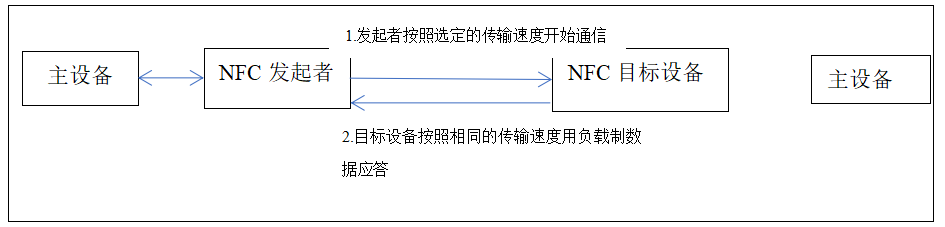


图 2 Linux NFC 工作原理之被动模式结构图

NFC工作模式之被动模式，NFC 发起设备( 也称为主设备) 需要供电设备，主设备利用供电设备的能量来提供射频场，并将数据发送到 NFC 目标设备( 也称作从设备) ，传输速率需在 106kbps、212kbps 或 424kbps中选择其中一种。从设备不产生射频场，所以可以不需要供电设备，而是利用主设备产生的射频场转换为电能，为从设备的电路供电，接收主设备发送的数据，并且利用负载调制( loadmodulation) 技术，以相同的速度将从设备数据传回主设备。因为此工作模式下从设备不产生射频场，而是被动接收主设备产生的射频场，所以被称作被动模式，在此模式下，NFC 主设备可以检测非接触式卡或 NFC 目标设备，与之建立连接。

NFC 通信的关键构件主要有两部分，即发起设备、目标设备，如果 NFC 设备配有专门的供电功能，那么可以任意选择充当发起设备，还是目标设备，传输数字信号时主要依靠电磁场，利用频移/幅移键控的方法调制通信过程中的载波；在实施通信时，发起设备首先利用无线射频磁场对通信过程进行初始化操作，之后目标设备对发生变化的磁场进行检测，最后依靠自身或者负载调制形成速率与之相同的射频电磁场，以此对发起设备所传输的命令进行响应，在此过程中，如果目标设备对发起命令的响应依靠自身射频磁场完成，那么此种通信方式即为主动通信；如果目标设备对发起命令的响应依靠负载调制技术完成，那么此种通信方式即为被动通信。

## 2.2 NFC三种工作模式结构图

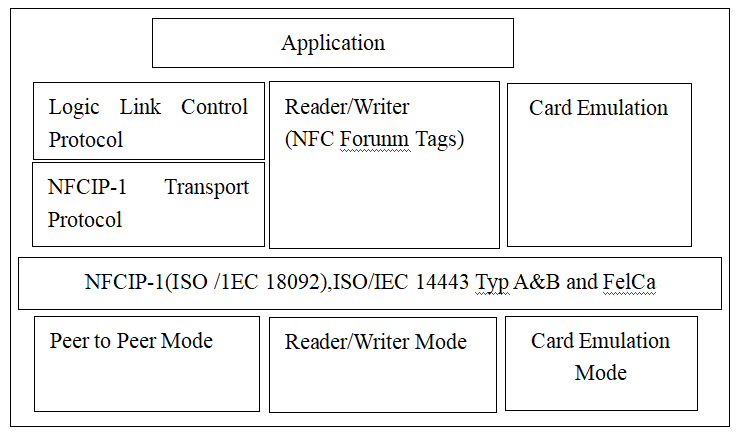


图 3 Linux NFC 三种工作模式结构图

点对点模式：可在两个具备 NFC 功能的设备上进行数据交换，如交换音乐、传送文件等。使用此模式需要两台设备均具备 NFC 功能，NFC 文件传输基本属于被遗弃的传输方式，因为 NFC 传输速度非常慢，通常只用来在手机间传递单个联系人，网址等信息量很小的数据，所以日常中使用场景较少。

读写模式：读写模式也称作读卡器模式，可作为非接触读卡器使用。能够从电子标签上读取或写入信息，但由于市场上 NFC 电子标签未能大面积使用，所以消费者更多是通过此模式给实体公交卡进行充值，但是随着虚拟公交卡功能的逐步实现，人们对实体公交卡的依赖会越来越弱，集成到手机里的虚拟公交卡服务会更加方便。

卡模式：其实就是相当于一张采用 RFID 技术的 IC 卡。可以用配备了 NFC 功能的手机替代大量的实体 IC 卡（公交卡、信用卡等）使用场景，实现移动支付功能。

# NFC数据格式

## 3.1 NFC数据格式解析图

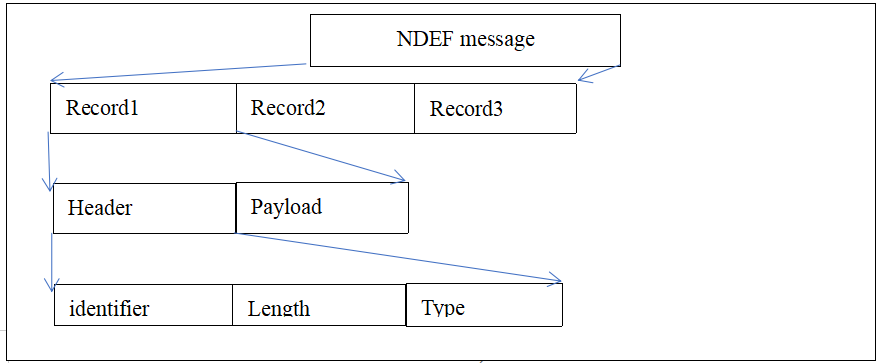


图 4 Linux NFC 数据结构

为了实现标签和NFC设备，以及NFC设备之间的交互通信，(NFC Froum)定义了称为NFC数据交换格式(NDEF)的通用数据格式。

NDEF是轻量级的紧凑的二进制格式，可带有URL，vCard和NFC定义的各种数据类型。NDEF使得NFC的各种功能能容易使用各种支持的标签类型传输数据，因为NDEF标签类型封装了标签的种类细节信息，使得应用不用关心与何种标签在通信。NDEF交换的信息由一系列记录组成。每条记录包含一条有效载荷。使用NFC定义的数据类型，载荷内容必须被定义在一个NFC记录类型定义(RTD)文件中记录数据的类型和大小，由记录载荷的头部注明。类型域的值由类型名字格式指定。

# NFC协议栈

## 4.1 NFC协议栈结构图

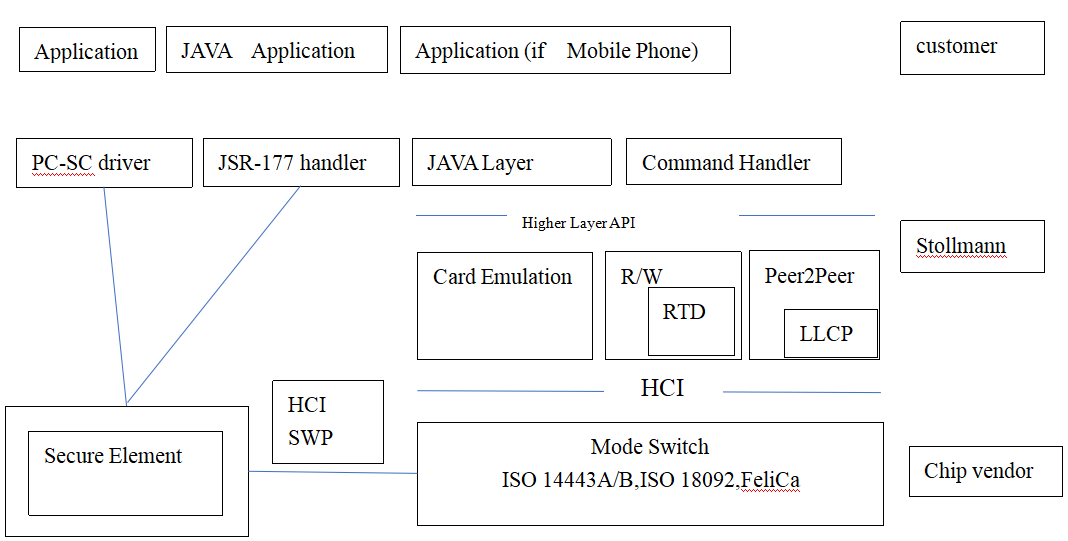


图 5 Linux NFC 协议栈结构图

NFC 论坛规定 NFC 协议栈的通用架构在 NFC 协议栈中，其主要构件包括驱动、操作系统、硬件等抽象层、主机、逻辑链路等控制层，以及论坛参考实现层等，下面对上述构件进行详细的分析和介绍：

4.1.1 驱动抽象层（DAL）

该层的主要作用和功能是实现物理连接驱动，进而和 NFC 芯片通信，它在很大程度上依赖于具体的平台，如果 DAL 层具有不同的接口、平台，那么在移植过中，需要综合考虑其实际操作系统再移植；

4.1.2 逻辑链路控制层(LLC）

逻辑链路控制层(LLC）能够和 MAC 层、基底层完成逻辑链路控制、数据传输等，MAC 层功能的实现在很大程度上依赖于 LLCP 协议的映射，在本质上看，LLC 是上层 OSI的构件.

4.1.3 主机控制接口（HCI）

主机控制接口（HCI）完成对设备接口的定义，如果有设备需要和 NFC 主控设备实现连接时，通常配有专门的星形网络，主机设备正是以 HCI 层为控制接口，其结构共计三层，分别是协议消息机制，命令交换、响应和事件的通道，路由协议（可实现消息的随时分割）

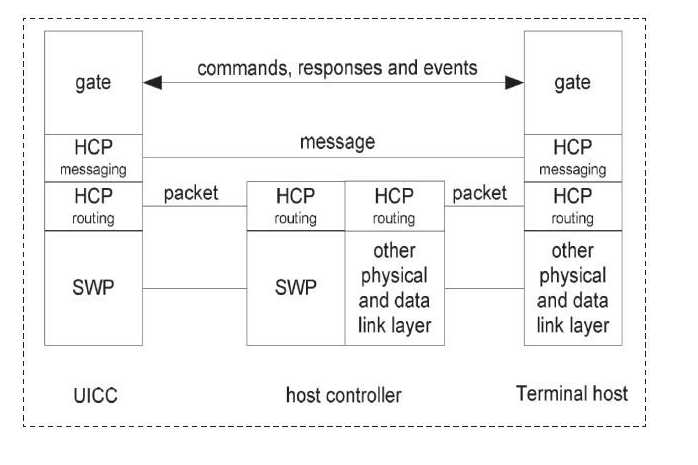


图 5 Linux NFC HCI层结构图

4.1.4 硬件抽象层(HAL）

硬件抽象层(HAL）可以重新封装不同硬件平台的接口，保证较为统一的接口，使其完全不受硬件平台的影响，实现两者的无关性。

4.1.5 操作系统抽象层(OSAL）

对各种操作系统的资源实施重新封装，完成封装后，对协议栈的内存管理、消息处理等接口进行统一，保证平台系统不会影响硬件抽象层、论坛参考实现层，保证两者的无关性。

4.1.6 论坛参考实现（FRI）

将 SDK 提供给 OSAL，实现和 NFC 芯片的通信，有利于开发者在开发软件过程中对 API 的调用。作为一种运行库，NFC 协议栈属于单线程，在用户所构建的进程中可以正常工作，该栈的结构共有六层，可以利用消息和回调机制实现通信。由于 NFC协议栈并不拥有专属的消息队列，所以，主要利用平台的相关操作实现回调机制。煤层具有基本相同的运作原理，基于上层的命令、消息对下层的回调和接口函数进行调用，在完成所分配的任务后，下层会向上层返回相应的信息，基于此类信息，上层实施处理和操作。NFC 共有读卡、点对点、卡模拟三种不同的工作模式，每种模式所对应的协议架构必然存在不同和差异，NFC 协议栈支持三种常见通讯接口：UART、I2C 和 SPI .

UART,UART是一种通用串行[数据总线](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E6%80%BB%E7%BA%BF" \t "https://baike.baidu.com/item/_blank)，用于[异步通信](https://baike.baidu.com/item/%E5%BC%82%E6%AD%A5%E9%80%9A%E4%BF%A1" \t "https://baike.baidu.com/item/_blank)。该总线双向通信，可以实现[全双工传输](https://baike.baidu.com/item/%E5%85%A8%E5%8F%8C%E5%B7%A5%E4%BC%A0%E8%BE%93" \t "https://baike.baidu.com/item/_blank)和接收。在嵌入式设计中，UART用于主机与辅助设备通信，如汽车音响与外接AP之间的通信，与PC机通信包括与监控调试器和其它器件，如[EEPROM](https://baike.baidu.com/item/EEPROM" \t "https://baike.baidu.com/item/_blank)通信。

I2C总线是由[Philips](https://baike.baidu.com/item/Philips" \t "https://baike.baidu.com/item/I2C%E6%80%BB%E7%BA%BF/_blank)公司开发的一种简单、双向二线制同步串行总线。它只需要两根线即可在连接于总线上的器件之间传送信息。

主器件用于启动总线传送数据，并产生时钟以开放传送的器件，此时任何被寻址的器件均被认为是从器件．在总线上主和从、发和收的关系不是恒定的，而取决于此时数据传送方向。如果主机要发送数据给从器件，则主机首先寻址从器件，然后主动发送数据至从器件，最后由主机终止数据传送；如果主机要接收从器件的数据，首先由主器件寻址从器件．然后主机接收从器件发送的数据，最后由主机终止接收过程。在这种情况下．主机负责产生定时时钟和终止数据传送。

SPI,SPI是串行外设接口（Serial Peripheral Interface）的缩写。SPI，是一种高速的，全双工，同步的通信总线，并且在芯片的管脚上只占用四根线，节约了芯片的管脚，同时为PCB的布局上节省空间，提供方便，正是出于这种简单易用的特性，如今越来越多的芯片集成了这种通信协议，比如AT91RM9200。

NFC 协议栈的多数功能都是依靠 Libandroid、libnfc 实现，通常由芯片厂商提供，对 NFC 芯片进行操作。平台厂商库、JNI 层主要运行于 nfc 进程，按照系统的标准调用方式对 NFC 驱动程序（Linux 内核空间）实施调用。在对 NFC 协议栈架构进行分析后发现，在对协议栈驱动抽象层（DAL）进行修改后方可移植 NFC 协议栈，把 URAT 驱动（NFC 芯片）编译至 Linux 内核。

# NFC工作模式之源码分析

## 5.1 NFC源码分析之R/W工作模式

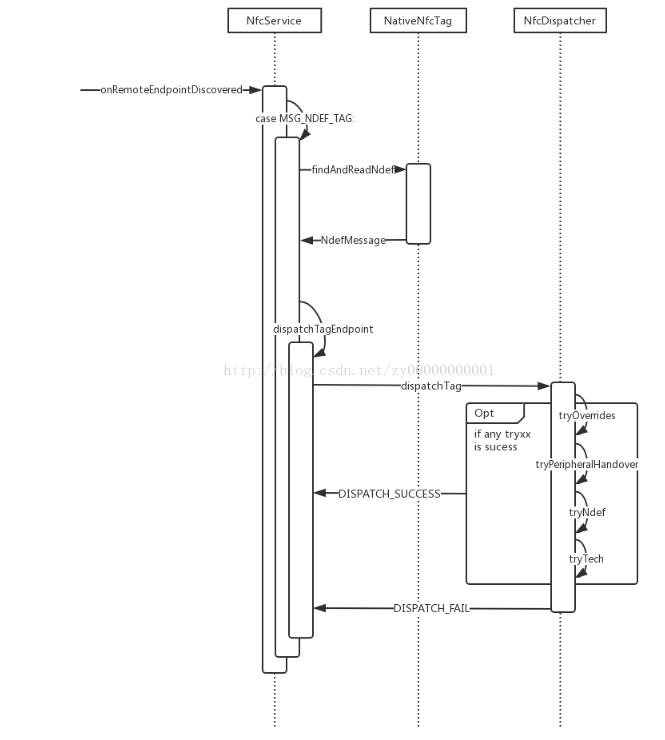


图 6 NFC读写工作模式结构图

如上图所示，NFC读取和写入Tag的流程，NFCService启动完成后，会通过NFCService中的applyRouting方法设置对应的Discovery，根据设置的命令参数来决定设备是处于listen模式，还是polling模式。当处于polling模式下会检测具体是哪一个Tag进入到了自己的射频厂，进而产生反应。与其相对的是listen模式，Tag就相当于是listen模式，监听谁往外polling，产生响应.

部分源码解析：

1.applyRouting函数。参数force为true的时候，基本上就会执行一enableDiscovery.

**void applyRouting(boolean force) {**

**synchronized (this) {**

**//mInProvisionMode：用来标记当前系统是否处于开机向导界面.**

**//对应于android开机时候的引导provision apk，是否出在引导模式**

**if (mInProvisionMode) {**

**mInProvisionMode = Settings.Secure.getInt(mContentResolver,**

**Settings.Global.DEVICE\_PROVISIONED, 0) == 0;**

**//当运行到这里的时候就代表，这是刷完系统，开机引导完，第一次走到这里，**

**//并且现在开机引导已经完毕，此时下面可能已经是false**

**if (!mInProvisionMode) {**

**//就是设置NfcDispatcher中的mProvisioningOnly = false;**

**mNfcDispatcher.disableProvisioningMode();**

**//调用native方法doSetProvisionMode，去通知native层，当前不是处于ProvisionMode**

**mDeviceHost.doSetProvisionMode(mInProvisionMode);**

**}**

**}**

**// 当此时正在和一个Tag在通信的时候，延迟re-configuration.**

**if (mScreenState == ScreenStateHelper.SCREEN\_STATE\_ON\_UNLOCKED && isTagPresent()) {**

**Log.d(TAG, "Not updating discovery parameters, tag connected.");**

**mHandler.sendMessageDelayed(mHandler.obtainMessage(MSG\_RESUME\_POLLING),**

**APPLY\_ROUTING\_RETRY\_TIMEOUT\_MS);**

**return;**

**}**

**try {**

**//此处更新于NFC RF的Discovery参数.**

**NfcDiscoveryParameters newParams = computeDiscoveryParameters(mScreenState);**

**//传入参数为true的时候或者参数有所变化.**

**if (force || !newParams.equals(mCurrentDiscoveryParameters)) {**

**//只要routingtable变过，或者有新的tech的改变。**

**if (newParams.shouldEnableDiscovery()) {**

**boolean shouldRestart =**

**mCurrentDiscoveryParameters.shouldEnableDiscovery();**

**//最重要的就是调用到了这里，mDeviceHost的实现类是NativeNfcManager**

**//最终调用到内部的如下：**

**// private native void doEnableDiscovery(int techMask,**

**//                 boolean enableLowPowerPolling,**

**//                 boolean enableReaderMode,**

**//                 boolean enableP2p,**

**//                 boolean restart);**

**//去native层，初始化RF的相关参数。**

**mDeviceHost.enableDiscovery(newParams, shouldRestart);**

**} else {**

**mDeviceHost.disableDiscovery();**

**}**

**mCurrentDiscoveryParameters = newParams;**

**} else {**

**Log.d(TAG, "Discovery configuration equal, not updating.");**

**}**

**} finally {**

**......**

**}**

**}**

**}**

1. computeDiscoveryParameters这个函数，它返回NFCDiscoveryParameters，这个类描述了用于enable NFC的tag discovery、polling、host card emulation的各个参数，配置的不同最终初始化的NFC功能不同.以上是关于NFC工作的RF状态的描述分析，下面开始分析，当NFC检测到remote tag的时候的回调。最终native层通过NativeNfcManager.cpp中的

**gCachedNfcManagerNotifyNdefMessageListeners = e->GetMethodID(cls.get(),**

**"notifyNdefMessageListeners", "(Lcom/android/nfc/dhimpl/NativeNfcTag;)V");**

回调于NfcTag.cpp中的createNativeNfcTag(..)方法，然后通过JNI调用到NativeNfcManager.java中的

**private void notifyNdefMessageListeners(NativeNfcTag tag) {**

**mListener.onRemoteEndpointDiscovered(tag);**

**}**

mListener的实现就是NfcService，又重新回到Nfcservice中

**@Override**

**public void onRemoteEndpointDiscovered(TagEndpoint tag) {**

**sendMessage(NfcService.MSG\_NDEF\_TAG, tag);**

**}**

**void sendMessage(int what, Object obj) {**

**Message msg = mHandler.obtainMessage();**

**msg.what = what;**

**msg.obj = obj;**

**mHandler.sendMessage(msg);**

**}**

native层对发现的tag进行了一系列的初始化和封装操作，就是按照ndef协议把Tag的消息封装到TagEndpoint当中，TagEndpoint就是代表了一个远端的Nfc tag.然后执行到mHandler中的MSG\_NDEF\_TAG

**case MSG\_NDEF\_TAG:**

**if (DBG) Log.d(TAG, "Tag detected, notifying applications");**

**......**

**//当有别人调用setReaderMode()接口去设置mReaderModeParams的时候.**

**//如第三方app模拟reader.(一般都是在第三方app的reader/write界面，由它们调用)**

**synchronized (NfcService.this) {**

**readerParams = mReaderModeParams;**

**}**

**if (readerParams != null) {**

**presenceCheckDelay = readerParams.presenceCheckDelay;**

**//如果FLAG\_READER\_SKIP\_NDEF\_CHECK标记位不为0，那么直接开始调用dispatchTagEndpoint分发**

**if ((readerParams.flags & NfcAdapter.FLAG\_READER\_SKIP\_NDEF\_CHECK) != 0) {**

**if (DBG) Log.d(TAG, "Skipping NDEF detection in reader mode");**

**tag.startPresenceChecking(presenceCheckDelay, callback);**

**dispatchTagEndpoint(tag, readerParams);**

**break;**

**}**

**}**

**//当是NFC Barcode的时候也是直接分发，看其实意思是解析NFC条形码的时候**

**if (tag.getConnectedTechnology() == TagTechnology.NFC\_BARCODE) {**

**if (DBG) Log.d(TAG, "Skipping NDEF detection for NFC Barcode");**

**tag.startPresenceChecking(presenceCheckDelay, callback);**

**dispatchTagEndpoint(tag, readerParams);**

**break;**

**}**

**//去调用NativeNfcTag的findAndReadNdef，进行相关初始化后把native层的数据按照NDEF**

**//协议封装成java中的类NdefMessage.(此时就是读取NDEF格式的信息了)**

**NdefMessage ndefMsg = tag.findAndReadNdef();**

**//解析的消息无法被实例化成NDEF的时候。**

**if (ndefMsg == null) {**

**// First try to see if this was a bad tag read**

**if (!tag.reconnect()) {**

**tag.disconnect();**

**//当是亮屏的时候弹出提示信息.**

**if (mScreenState == ScreenStateHelper.SCREEN\_STATE\_ON\_UNLOCKED) {**

**//"Read error. Try again"**

**String toastString = mContext.getString(**

**R.string.nfc\_strings\_toast\_prompt\_touch\_again\_txt);**

**if (!ToastMaster.isSameToastShown(toastString)) {**

**ToastMaster.showToast(mContext, toastString);**

**}**

**}**

**break;**

**}**

**}**

**//这个如名字是抖动的tag，就是由于距离没控制好，一直检测到同一个tag的时候。**

**//应该就是这个Tag检测完，没有离开范围，就不会再次检测**

**if (debounceTagUid != null) {**

**// If we're debouncing and the UID or the NDEF message of the tag match,**

**// don't dispatch but drop it.**

**......**

**}**

**mLastReadNdefMessage = ndefMsg;**

**tag.startPresenceChecking(presenceCheckDelay, callback);**

**//解析出问题的ndef消息也要分发出去.**

**dispatchTagEndpoint(tag, readerParams);**

**break;**

1. findAndReadNdef函数，位于NativeNfcTag这个方法中的注释为我们的分析带来很多遍历，此处需要特别注意：NativeNfcTag是Nfc Tag在java层的一个代表，它在native层由NfcTag对象来表示，在NfcTag.cpp中有如下方法用于创建NativeNfcTag对象，并且是实例化相应的参数。

# NFC读写程序的设计

## 6.1 NFC读程序设计

NFC读在写程序里加入“writer tag”控件，将按钮控件的事件列表项设置为写入活动。选择数据记录类型，可以使程序简单化，便于查看写程序的设计中，需要确定通知是否ACTION\_NDEF\_DISCOVERED。需要在适配器 Adapter 中调用 Get Action()方法。如果结果返回 true，则使用标签对象来创建标签实例。获取 Parcelable Extra 方法将数据封装在标签中，以便数据可以存储在标签中。

先处理输入数据并把数据写进 tag 中。根据输入的数据传递参数消息，并调用相应的函数实现其功能。比如，输入数据为 TNF\_ABSOLUTE\_URI，找到该数据存储的变量传送 NEX 消息给 URI。 Ndef 消息将 URI 值存放于数组中，并编写成对应的 ASCII 编码值，该数组会自动生成一个函数来初始化 Ndef Message。

在将预处理数据写入标签之前，调用 Ndef 的 get 方法获取标签的内容并保存。要确定 ndef 对象是否为空，如果标签内容不为 NDEF 格式为空，则需要使用 Ndef Formatable 类来处理数据; 不为 null，我们使用 Ndef 类来处理。调用这种连接方法来建立与标签的接，以便读写 I/O 操作的标签。然后确定是否可以写标签，如果标签是只读的，将打印消息并返回 fasle 这表示写操作失败; 如果需要发送数据长度超过标签提供的空间，写操作将被取消。满足上述条件后，调用写入 Ndef Message 方法将预处理的 NDEF 消息写入标签，最后关闭与标签的连接。该函数返回 true，表示写操作成功。调用 Ndef 的 get 方法核心代码如下所示。

**NDef ndef = NDef.get(detected Tag);**

**If(!ndef.writable()){**

**Toast.make.Text(“(Tag is read only ! Toast.LENGTH\_SHORT).show()”);**

**Return false;**

**}**

**If(ndef.getMaxSize()<size){**

**Toast.make.Text(this.“(The data cannot written to tag. Tag capacity is”+ndef.getMaxSize()+”bytes,message is”+size+”bytes .”,Toast.LENGTH\_SHORT).show();**

**return false);**

**}**

**ndef.write.Ndef Message(message);**

**ndef.close();**

**return true;**

为了捕获异常，易于代码调试，上面的代码主要放在 try-catch 块中。

## 6.2 NFC写程序设计

在读操作程序里需要加入“read tag”控件，并设置相应的控件按钮，以便监视标签界面读标签活动。当 NFC 智能手表感应到标签时，点击控件按钮获取标签中的数据，因为 NDEF 消息可能包含多个 NFC 记录，实际消息被封装在 NFC记录中。因此，根据 NFC 记录数据（有效载荷）的具体格式进行相应的数据分析和数据预处理，最终打印标签读取信息。 读标签和写标签应用的设计思想基本类似。

在读标签操作中，是用 Ndef Message 类来得到 tag 中的 NDEF 消息。首先要完成对 Ndef Message 类型的数组构造，接着获取 tag 中的 NDFF 数据，最后将get Ndef Messages 函数获得的数据存储在数组中。其核心代码如下

**private String read(Tag tag) throws IOException, Format Exception {**

**if (tag != null) {**

**//解析 Tag 获取到 NDEF 实例**

**Ndef ndef = Ndef.get(tag);**

**//打开连接**

**ndef.connect();**

**//获取 NDEF 消息**

**Ndef Message message = ndef.get Ndef Message();**

**//将消息转换成字节数组**

**byte[] data = message.to Byte Array();**

**//将字节数组转换成字符串**

**String str = new String( Charset="UTF-8");**

**//关闭连接**

**ndef.close();**

**return str;**

**}**

**else {**

**Toast.make Text(Nfc Demo Activity.this, "设备与 nfc 卡连接断开，请重新连接...",**

**Toast.LENGTH\_SHORT).show();**

**}**

**return null;**

**}**

在取得 NDEF 消息后，需要根据不同的记录类型处理数据。在本文中，每种类型的数据处理被封装成一个函数。 获取 NDEF 消息后，将作出判断，然后根据记录类型选择相应的处理功能，这里以 TNF\_ABSOLUTE\_URI 记录为例。 Pro NdefMessage URI 函数接收参数 message，然后使用两个 for 循环来获取记录的各个部分的信息，并将它们存储在 my Text 变量中，核心代码如下所示.

**for(int i = 0;i<message.length;i++){**

**My Text.append(“Message”+(i+1)+”:\n”);**

**For(int j = 0;j<messages[0.getRecords().length;j++]){**

**Ndef Record record = messages[i].get Records()[i];**

**My Text.append(j+1)+”th. Record type:”+new String(record.getTYpe()+”:\n”)**

**My Text.append(j+1)+”th. Record id:”+new String(record.getId()+”:\n”)**

**payLoad = new String(record.get Payload());**

**My Test.append(j+1)+”th.Record payload:”+payload+”\n”);**

**}**

**}**

# NFC的应用

NFC 作为一种近场通信技术，其应用十分广泛。NFC 应用可以分为四个基本类型，下面针对这三种基本应用类型进行介绍和分析。

## 7.1 支付的应用

NFC 支付主要是指带有 NFC 功能的手机虚拟成银行卡、一卡通等的应用。NFC 虚拟成银行卡的应用，称为开环应用。理想状态下是带有 NFC 功能的手机可以作为一张银行卡在超市、商场的 POS机上进行刷手机消费，但目前在国内还无法实现。主要原因是作为开环应用下的 NFC 支付有着繁冗的产业链，背后的卡商、方案商的利益和产业格局博弈十分复杂。就目前国内 NFC 开环应用的大环境来说，由于各方面利益的博弈，NFC 开环支付应用已经错过了在支付宝和微信支付等移动支付普及之前的最佳时机，NFC 开环支付已经不可能再单独发展起来。NFC 开环支付以后的发展只有寻求和支付宝和微信支付进行衔接和捆绑，作为支付宝和微信支付的身份认证手段，才有可能在未来的移动支付中占有一席之地。

NFC 虚拟成一卡通卡的应用，称为闭环应用。目前 NFC的闭环应用在国内的发展也不太理想。虽然在有些城市的公交系统已经开放了手机的 NFC 功能，但并没有得到普及。根本原因以卡为载体的一卡通系统有一个发卡的获利，系统集成商和运营商。发卡上可以获得丰厚的利润。所以目前小米和华为都在一些城市试点开通手机的 NFC 公交卡功能，但目前都还需要开通服务费。但是随着 NFC 手机的普及技术的不断成熟，一卡通系统会逐渐支持 NFC 手机的应用，前景是乐观的，但过程注定是曲折的。

## 7.2 安防的应用

NFC安防的应用主要是将手机虚拟成门禁卡，电子门票等。NFC虚拟门禁开就是将现有的门禁卡数据写入手机的NFC，这样无需使用智能卡，使用手机就可以实现门禁功能，这样不仅是门禁的配置、监控和修改等都十分方便，而且可以实现远程修改和配置，例如在需要时临时分发凭证卡等。NFC 虚拟电子门票的应用就是在用户购票后，售票系统将门票信息发送给手机，带有 NFC 功能的手机可以把门票信息虚拟成电子门票，在检票是直接刷手机即可。

NFC 在安防系统的应用是今后 NFC 应用的重要领域，前景十分广阔。因为在这个领域可以直接为该技术使用者带来经济利益，让他们更有动力进行现有设备和技术的升级。因为使用手机虚拟卡，可以减少门禁卡或者磁卡式门票的使用，直接降低使用成本，另外还可以适当提高自动化程度，降低人员成本和提升效率。

## 7.3 标签的应用

NFC 标签的应用就是把一些信息写入一个 NFC 标签内，用户只需用 NFC 手机在 NFC 标签上挥一挥就可以立即获得相关的信息。例如商家可以把含有海报、促销信息、广告的 NFC标签放在店门口，用户可以根据自己的需求用 NFC 手机获取相关的信息，并可以登录社交网络、和朋友分享细节或好东西。虽然 NFC 标签在应用上十分便捷，成本也很低，但在目前移动网络的普及和二维码的逐渐流行，NFC 标签的应用前景不容乐观。

因为和 NFC 标签相比，二维码只需要生成和印刷成一个小图像，可以说几乎是零成本，提供的信息和 NFC 一样很丰富，很容易就会替代 NFC 标签的应用。

# NFC与其他技术的比较

目前的近距离无线通信技术包括了 RFID、蓝牙、红外等，下面针对各通信技术的特点与 NFC 技术进行分析与比较。

## 8.1 NFC与RFID的比较

第一，工作模式不同。NFC 是将点对点通信功能，读写器功能和非接触卡功能集成进一颗芯片，而 RFID 则有阅读器和标签两部分组成。NFC 技术既可以读取也可以写入，而 RFID只能实现信息的读取以及判定。

第二，传输距离不同。NFC 传输距离比 RFID 小的多，NFC的传输距离只有 10 厘米，RFID 的传输距离可以达到几米、甚至几十米。NFC 是一种近距离的私密通信方式，相对于 RFID来说 NFC 具有距离近、带宽高、能耗低、安全性高等特点。

第三，应用领域不同。NFC 更多的应用于消费类电子设领域，在门禁、公交、手机支付等领域发挥着巨大的作用; RFID 则更擅长于长距离识别，更多的被应用在生产物流、跟踪、资产管理上。

## 8.2 NFC与蓝牙的比较

NFC 和蓝牙都是短程通信技术，相对于蓝牙很早就被集成到移动电话中并已经被普及，NFC 最近几年才开始被集成进移动电话中，并且到目前为止只集成在少数移动电话中。

第一，建立时间不同，NFC 通信设置程序简单，通信建立时间很短，仅需 0．1s 左右; 而蓝牙通信设置程序相对复杂，通信建立时间较长，大概需要 6s。

第二，传输距离不同，NFC 传输距离只有 10cm，而蓝牙传输距离可达 10m。但 NFC 在传输功耗和安全性方面略优于蓝牙。

第三，传输速度和工作频率不同，NFC 工作频率为 13. 56MHz，传输速度最大 424 Kbit/s，而蓝牙工作频率为 2．4GHz，传输速度可达 2．1 Mbit/s。

## 8.3 NFC与红外的比较

NFC 和红外传输相比，传输距离相当，但比红外传输速度更快，NFC 传输速度最大可达 424 Kbit/s，而红外传输速度大概100Kbit / s。建立时间 NFC 比红外略快，NFC 建立时间为 0.1s，红外传输建立时间为 0．5s。红外传输必须严格的对齐才能传输数据，且中间不能有障碍物，而 NFC 则没有这种限制; 另外NFC 比红外更安全可靠。

# 实验过程

