




# RFID实验一总结

原创

龙云尧  于 2017-04-20 06:04:40 发布  8767  收藏 9

分类专栏: [rfid](#) 文章标签: [Java-Card](#) [博客](#) [Applet](#) [RFID](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: <https://blog.csdn.net/Michael753951/article/details/70254340>

版权



[rfid](#) 专栏收录该内容

3 篇文章 0 订阅

订阅专栏

龙云尧个人博客, 转载请注明出处。

CSDN地址: <http://blog.csdn.net/michael753951/article/details/70254340>

个人blog地址: <http://yaoyl.cn/rfidshi-yan-yi-zong-jie-2/>

刚接到这个课程项目的时候, 是一脸懵逼的。毕竟是第一次接触JavaCard编程 (其实就是自己没认真听课)。不过在围观各路大佬的博客之后, 总算对整个项目有了较为深入的了解。

在实验过程中, 需要不断翻阅实验课PPT之《01 Java智能卡之概述》, 《02 电子钱包的文件系统》, 《实验2文档》, 以及CSDN大佬吕浪的[课程总代码](#), 以及他相关博客的[Java card开发系列文章](#)。

然后再自己不断重写代码, 理解整个实现过程, 才能对这个课程实验有较为深入的了解。

代码在未征得本人同意之前, 请勿直接Ctrl+C, Ctrl+V, 谢谢。

## 正式实验

### 实验分析

首先我们要知道本次实验中需要修改哪些函数, 实现那哪些功能。

首先我们在PPT最后知道本次实验的主要目的是:

- 创建文件
- 写密钥
- 读写二进制文件

再看详细内容, 我们大概可以捋清如下关系:

- 创建文件
  - 卡片收到命令并且开始解析
  - 所谓的解析就是判断是何种文件，然后再进行创建
  - 异常处理
- 写密钥
  - 密钥消息是一条一条接受的，每次只会写入一条密钥
  - 卡片收到命令以后，取出数据，然后写入密钥文件
- 读写二进制文件
  - 写指令只需要一条
  - 根据指令内容获得需要的参数，然后将其写入持卡人文件或者应用文件中
  - 注意：写入之前需要检查数据时候超过限定大小
  - 读取和写类似

在有了大概思路以后，我们开始阅读源代码。经过简单寻找，我们发现本次实验涉及的代码大多集中在Purse.java中。

```
//APDU Object
private Papdu papdu;

//文件系统
private KeyFile keyfile;           //密钥文件
private BinaryFile cardfile;       //应用基本文件
private BinaryFile personfile;     //持卡人基本文件
private EPFile EPfile;             //电子钱包文件
```

上面四个元素指明了我们将要操作的几个对象。

```
public void process(APDU apdu) {
    if (selectingApplet()) {
        return;
    }
    //步骤1:取APDU缓冲区数组引用并将之赋给新建数组
    //步骤2:取APDU缓冲区中数据放到变量papdu
    //步骤3:判断命令APDU是否包含数据段，有数据则获取数据长度，并对le赋值
    boolean rc = handleEvent();
    //步骤4:判断是否需要返回数据，并设置apdu缓冲区
}
```

这个部分似乎用处不明。但是根据让我们填写的部分，可以看出这一块是卡片读取指令的地方，将其缓冲到papdu中，让我们得以进行后续分析。

```
/*
 * 功能：对命令的分析和处理
 * 参数：无
 * 返回：是否成功处理了命令
 */
private boolean handleEvent(){
    switch(papdu.ins){
        case condef.INS_CREATE_FILE: return create_file();
        //todo: 完成写二进制命令，读二进制命令，写密钥命令
    }
    ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
    return false;
}
```

注释已经说明了，在《01 Java智能卡之概述》P30也有讲到该部分的作用，这里就是我们分析指令的地方，通过这个地方我们可以知道卡片当前接收的指令目的是什么。

将PPT上的操作内容先填进去，具体实现我们待会再说。

```
/**
 * 功能：对命令的分析和处理
 * 参数：无
 * 返回：是否成功处理了命令
 * 《01 Java智能卡之概述》P30
 */
private boolean handleEvent(){
    switch(papdu.ins){
        case condef.INS_CREATE_FILE: return create_file(); // E0 文件创建
        //todo: 完成写二进制命令，读二进制命令，写密钥命令
        case condef.INS_WRITE_KEY: return write_key(); // D4 写密钥
        case condef.INS_WRITE_BIN: return write_read_bin((byte)'U'); // D6 写二进制文件
        case condef.INS_READ_BIN: return write_read_bin((byte)'R'); // B0 读二进制文件
    }
    ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED); // 0x6D00 表示 CLA错误
    return false;
}

package purse;
/**
 * 已给出部分常数值，其他根据需要自行添加
 */
public class condef {
    //----- INS Byte -----
    final static byte INS_CREATE_FILE = (byte)0xE0; //文件建立命令的INS值
    final static byte INS_WRITE_KEY = (byte)0xD4; //写入密钥命令的INS值
    final static byte INS_WRITE_BIN = (byte)0xD6; //写入二进制命令的INS值
    final static byte INS_READ_BIN = (byte)0xB0; //读取二进制命令的INS值
    final static byte INS_NIIT_TRANS = (byte)0x50; //初始化圈存和初始化消费命令的INS值
    final static byte INS_LOAD = (byte)0x52; //圈存命令的INS值

    //----- FILE TYPE Byte -----
    final static byte KEY_FILE = (byte)0x3F; //密钥文件的文件类型
    final static byte CARD_FILE = (byte)0x38; //应用基本文件的文件类型
    final static byte PERSON_FILE = (byte)0x39; //持卡人基本文件的文件类型
    final static byte EP_FILE = (byte)0x2F; //电子钱包文件的文件类型

    //----- SW -----
    final static short SW_LOAD_FULL = (short)0x9501; //圈存超额
}
```

我们发现这里还需要添加读二进制的常量。

再往下读，就是创建文件部分。

```

/*
 * 功能：创建文件
 */
private boolean create_file() {
    switch(papdu.pdata[0]){
        case condef.EP_FILE: return EP_file();
        //todo:完成创建密钥文件，持卡人基本文件和应用基本文件
        default:
            ISOException.throwIt(ISO7816.SW_FUNC_NOT_SUPPORTED);
    }
    return true;
}

/*
 * 功能：创建电子钱包文件
 */
private boolean EP_file() {
    if(papdu.cla != (byte)0x80)
        ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);

    if(papdu.lc != (byte)0x07)
        ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);

    if(EPfile != null)
        ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);

    EPfile = new EPFile(keyfile);

    return true;
}

```

注释和代码里已经说明，将会有4种文件可以被创建

- 电子钱包文件(EP\_file，这个部分已经给出)
- 秘钥文件
- 持卡人基本文件
- 应用基本文件

剩下的就是圈存指令获取，初始化，以及一些其他操作，不是本次试验需要关心的部分。

## 开始打码

前面的分析中，我们已经对本次实验有了大致的了解，接下来就是开始打码的过程了。

我们在PPT《01 Java智能卡之概述》中已经知道了cAPDU的结构。

# Applet通信基础

- C-APDU (APDU命令, command APDU) 结构:

命令头必写部分				命令选择部分		
CLA	INS	P1	P2	LC	数据段	LE

CLA: 指令的类

INS: 指令编码

P1、P2: 参数1和参数2

LC: 数据段的长度

data: 发送的数据

LE: 所期待的响应字节数

➤ 例子: select命令的结构是

CLA	INS	P1	P2	LC	Data	LE
00	A4	04	00	05 - 10	AID	00

24

我们也在各种文件中知道了每个参数的意义:

- CLA, 即class, 指令的类, 占2Bytes
- INS, instructions, 指令编码, 占2Bytes
- P1 - 第一个指令参数, 2Bytes
- P2 - 第二个指令参数 (P1, P2 根据INS不同, 也有不同的含义), 2Bytes
- LC, 数据段的长度 (多少个字节), 2Bytes
- Data部分就是真正的数据了, 最长可以是255个字节
- LE字节表示的是期望卡片发回来的字节长度, 注意不包括9000等响应

首先我们完善process。

```
public void process(APDU apdu) {
    // 如果Applet为空, 退出
    if (selectingApplet()) {
        return;
    }
    // 步骤1: 取APDU缓冲区数组引用并将之赋给新建数组
    byte apdu_buffer[] = apdu.getBuffer();
    // 步骤2: 取APDU缓冲区中数据放到变量papdu
    // 将apdu读取到卡片缓冲区当中并返回data段的长度
    short lc = apdu.setIncomingAndReceive();
    papdu.cla = apdu_buffer[0];
    papdu.ins = apdu_buffer[1];
    papdu.p1 = apdu_buffer[2];
    papdu.p2 = apdu_buffer[3];
    Util.arrayCopyNonAtomic(apdu_buffer, (short)5, papdu.pdata, (short)0, lc);
}
```

按照已知的结构, 将缓冲中的数据取出来, 并且依次按照对应的位置放进papdu中。

```

papdu.p2 = apdu_buffer[3];
Util.arrayCopyNonAtomic(apdu_buffer, (short)5, papdu.pdata, (short)0, 1c);
/*
 * 步骤3: 判断命令APDU是否包含数据段, 有数据则获取数据长度, 并对le赋值
 * 否则, 即不需要lc和data, 则获取缓冲区原本lc实际上是le
 * 获取le的方法, 因为不确定papdu有le部分, 所以IOS7816下标可选项并没有le而是放在数据块中的.
 * 如果有数据块, 那le就是buffer[ISO7816.OFFSET_CDATA+1c]
 * 调用papdu函数判断, 不能直接通过lc判断, 因为没lc只有le也会把le赋给lc
 *
 * 若papdu命令包含数据块, 则需要更新le
 * 否则, 将data的长度直接赋值给le
 * 注: LE字节表示的是期望卡片发回来的字节长度
 */
if(papdu.APDUContainData()) {
    papdu.le = apdu_buffer[ISO7816.OFFSET_CDATA+1c];
    papdu.lc = apdu_buffer[ISO7816.OFFSET_LC];
} else {
    papdu.le = apdu_buffer[ISO7816.OFFSET_LC];
    papdu.lc = 0;
}
// rc获取返回数据, 判断操作是否成功
boolean rc = handleEvent();
//步骤4:判断是否需要返回数据, 并设置apdu缓冲区
//if(papdu.le != 0)
// 如果成功, 则返回数据, 并且设置apdu缓冲区
if( rc ) {
    Util.arrayCopyNonAtomic(papdu.pdata, (short)0, apdu_buffer, (short)5, (short)papdu.pdata;
    apdu.setOutgoingAndSend((short)5, papdu.le); //把缓冲区的数据返回给终端
}
}
}

```

因为新加入了数据, 所以我们需要更新卡片当前的LE, 以便和终端进行确认。

handleEvent函数我们已经按照PPT改好了。接下来就是将它们一个一个实现。

```

/*
 * 功能: 创建文件
 */
private boolean create_file() {
    // 判断DATA域文件控制信息(AEF)
    switch(papdu.pdata[0]){
        case condef.EP_FILE:      return EP_file();      // 0x2F电子钱包文件
        //todo:完成创建密钥文件, 持卡人基本文件和应用基本文件
        case condef.PERSON_FILE:  return Person_file(); // 0x39持卡人基本文件
        case condef.CARD_FILE:    return Card_file();    // 0x38应用基本文件
        case condef.KEY_FILE:     return Key_file();     // 0x3F密钥文件
        default:
            ISOException.throwIt(ISO7816.SW_FUNC_NOT_SUPPORTED);
    }
    return true;
}
}

```

模仿提示的case, 根据todo, 我们可以列出其他3中情况, 分别指向创建密钥文件, 持卡人基本文件和应用基本文件。具体的对应条件在实验2文档中有详细说明。

表· DATA 域文件控制信息 (AEF) ◊

◊	命令报文数据域 ◊					
文件类型 ◊	BYTE-1 ◊	BYTE-2-3 ◊	BYTE-4 ◊	BYTE-5 ◊	BYTE-6 ◊	BYTE-7 ◊
持卡人基本文件 ◊	39 ◊	0037 ◊	FF ◊	FF ◊	FF ◊	FF ◊
应用基本文件 ◊	38 ◊	001e ◊	FF ◊	FF ◊	FF ◊	FF ◊
电子钱包文件 ◊	2F ◊	FFFF ◊	FF ◊	FF ◊	FF ◊	FF ◊
密钥文件 ◊	3F ◊	FFFF ◊	FF ◊	FF ◊	FF ◊	FF ◊

仔细分析EP\_file函数，这是一个钱包文件创建的样例，仔细阅读便可以对指令的解析过程有一个深入的了解。

表· 文件创建 (CREATE-FILE) 命令报文 ◊

代码 ◊	值 ◊
CLA ◊	80 ◊
INS ◊	E0 ◊
P1 ◊	00 ◊
P2 ◊	00 ◊
Lc ◊	文件信息长度 07 ◊
Data ◊	文件控制信息 ◊

根据上图，我们对EP\_file稍作修改，添加了ins校验，keyfile校验，p1p2校验。校验过程中，我们对cAPDU中所有参数进行全部确认一遍，然后确认EPfile目前为空，KEY文件已经建立，我们才能开始进行文件的读写，以免该函数在其他地方被误调用。

```
private boolean EP_file() {
    // CLA: 识别电子卡
    if(papdu.cla != (byte)0x80)
        ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);

    //EP的ins必须为0xE0，校验以免误调用
    if(papdu.ins != condef.INS_CREATE_FILE)
        ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);

    //p1和p2应该为0x00
    if(papdu.p1 != (byte)0x00 || papdu.p2 != (byte)0x00)
        ISOException.throwIt(ISO7816.SW_WRONG_P1P2);

    // LC: Data Field之长度
    // 文件创建时文件信息长度为0x07
    if(papdu.lc != (byte)0x07)
        ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);

    // 如果已经被分配
    if(EPfile != null)
        ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);

    //每个应用只能有一个KEY文件，且必须最先建立
    //《02 电子钱包的文件系统》P19
    if(keyfile == null)
        ISOException.throwIt(ISO7816.SW_FILE_NOT_FOUND);

    EPfile = new EPFile(keyfile);

    return true;
}
```

仿照这个模式，我们可以很快吧剩下3个文件建立实现出来。

Person\_file和Card\_file的实现在这里略过，和EP\_file一模一样。Key\_file则需要稍加注意。因为Key是最先创建的，所以这个时候，keyfile肯定还是null，所以和其他判定条件不太一样，其他条件一样。

```

/*
 * Key_file()
 * 密钥文件
 * 仿照EP_file实现
 */
private boolean Key_file() {

    if(papdu.cla != (byte)0x80)
        ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);

    //EP的ins必须为0xE0，校验以免误调用
    if(papdu.ins != condef.INS_CREATE_FILE)
        ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);

    //p1和p2应该为0x00
    if(papdu.p1 != (byte)0x00 || papdu.p2 != (byte)0x00)
        ISOException.throwIt(ISO7816.SW_WRONG_P1P2);

    // LC 应该为0x07，word中说的15有错
    if(papdu.lc != (byte)0x07)
        ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);

    if(keyfile != null)
        ISOException.throwIt(ISO7816.SW_FILE_NOT_FOUND);

    //keyfile在Key_file中创建，故不需要判断是否为null
    keyfile = new KeyFile();

    return true;
}

```

最后就是write\_key和write\_read\_bin的实现了。

观察write\_key表中的参数，以及p2的取值。

表·P2 值

密钥文件	应用基本文件	持卡人基本信息文件	电子钱包文件
00	16	17	18

代码	值
CLA	80
INS	D4
P1	00
P2	密钥标识
Lc	数据域长度
Data	数据

消费子密钥、圈存子密钥、TAC子密钥

CLA	INS	P1	P2	Lc	DATA
80	D4	01	密钥标识	15	3e/3f/34 使用权 更改权 密钥版本号 算法标识 16字节密钥



龙云堯  
明显应该是 0x07，数一下 DATA 中的

我们可以类似于之前创建文件的操作，完成写key操作。



```

private boolean write_key() {
    //每个应用只能有一个KEY文件，且必须最先建立
    //《02 电子钱包的文件系统》P19
    if(keyfile == null)
        ISOException.throwIt(ISO7816.SW_FILE_NOT_FOUND);

    if( papdu.cla != (byte)0x80)
        ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);

    if(papdu.ins != condef.INS_WRITE_KEY)
        ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);

    //密钥文件的p1为0x00， p2
    //write_key中p1必须为0x00， p2必不在0x16到0x18， 否则就是另外三种文件
    if(papdu.p1 != (byte)0x00 || (papdu.p2 >= (byte)0x16 && papdu.p2 <= (byte)0x18))
        ISOException.throwIt(ISO7816.SW_WRONG_P1P2);

    //密钥长度是否正确
    if(papdu.lc != (byte)0x15)
        ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);

    //文件空间已满
    if(keyfile.recNum >= 3)
        ISOException.throwIt(ISO7816.SW_FILE_FULL);

    keyfile.addkey(papdu.p2, papdu.lc, papdu.pdata); //写入一条密钥

    return true;
}

```

再然后就不难了。根据实验2文档，我们可以顺利写完整个代码。

首先是write\_read\_bin()实现。利用switch进行选择。

```

private boolean write_read_bin(byte U_R) {
    switch(U_R) {
        case (byte)'U': return write_binary();
        case (byte)'R': return read_binary();
        default : return false;
    }
}

```

以及二进制的读和写操作函数。不过注意的是，写数据的时候，要注意数据data要在[1, 255]之间进行选择，否则会越界（因为LC不够长）。

```

/**
 * 写二进制文件
 * 写二进制文件只需要一条命令
 * 根据命令的内容获得需要参数
 * 并写入已创建的持卡人基本文件或应用基本文件中；
 * !!! 注意：写入前要先检测写入的数据是否超过文件限定的大小
 * @return
 */
private boolean write_binary() {

    //首先判断cla是否确实为0x00，才能继续执行
    if(papdu.cla != (byte)0x00)
        ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);

    //每个应用只能有一个KEY文件，且必须最先建立
    //《02 电子钱包的文件系统》P19
    if(keyfile == null)
        ISOException.throwIt(ISO7816.SW_FILE_NOT_FOUND);

    //写入前要先检测写入的数据是否超过文件限定的大小
    //写入data长度应该在1-255之间
    if(papdu.lc <= 0x00 && papdu.lc >= 0xFF)
        ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);

    /**
     * 持卡人或者卡不应该为空，如果不为空就可以开始写文件了
     */
    //应用基本文件为0x16，应用文件不应该为空
    if(papdu.p1 == (byte)0x16 && cardfile == null)
        ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
    //持卡人基本文件为0x17，持卡人不应该为null
    if(papdu.p1 == (byte)0x17 && personfile == null)
        ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);

    //应用基本文件为0x16，持卡人基本文件为0x17
    if(papdu.p1 == (byte)0x16){
        cardfile.write_binary(papdu.p2, papdu.lc, papdu.pdata);
    } else if(papdu.p1 == (byte)0x17){
        personfile.write_binary(papdu.p2, papdu.lc, papdu.pdata);
    }

    return true;
}
}

```

```

/**
 * 读取二进制文件
 * 类似
 * @return
 */
private boolean read_binary() {

    //首先判断cla是否确实为0x00，才能继续执行
    if(papdu.cla != (byte)0x00)
        ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);

    if(keyfile == null)
        ISOException.throwIt(ISO7816.SW_FILE_NOT_FOUND);

    //应用基本文件为0x16，应用文件不应该为空
    if(papdu.p1 == (byte)0x16 && cardfile == null)
        ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
    //持卡人基本文件为0x17，持卡人不应该为null
    if(papdu.p1 == (byte)0x17 && personfile == null)
        ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);

    //读取相应的二进文件
    if(papdu.p1 == (byte)0x16){
        this.cardfile.read_binary(papdu.p2, papdu.le, papdu.pdata);
    }
    else if(papdu.p1 == (byte)0x17){
        this.personfile.read_binary(papdu.p2, papdu.le, papdu.pdata);
    }

    return true;
}

```

到这里，整个实验也就结束了。在填写完代码之后整个思绪都变得异常清晰，在操作中使用严格的控制能够良好的避免错误，也能够方便我们整理思路。让我们的撸代码的时候，思路更加清晰。

## 验证实验

首先将脚本.txt文件转换成.jcsh文件，然后转码成UTF-8，不转码就会出现乱码，eclipse就无法正常读取脚本。

然后在eclipse的控制台输入 `/set-var path "D:\Eclipse_cpp_workspace\purse"`，注意文件路径填写的是你的.jcsh脚本文件所在的位置，然后输入你的脚本文件名就行了，如 `purse_script`。（不需要加后缀）

```
cm> purse_script
/select 1235318401
=> 00 A4 04 00 05 12 35 31 84 01 00
(287371 nsec)
<= 90 00
Status: No Error
/send 80e00000073fffffffffffffff
=> 80 E0 00 00 07 3F FF FF FF FF FF FF
(458996 nsec)
<= 90 00
Status: No Error
/send 80e000160738001efffffffffff
=> 80 E0 00 16 07 38 00 1E FF FF FF FF
(513162 nsec)
<= 90 00
Status: No Error
/send 80e0001707390037fffffffffff
=> 80 E0 00 17 07 39 00 37 FF FF FF FF
(362635 nsec)
<= 90 00
Status: No Error
/send 80e00018072Ffffffffffffffff
=> 80 E0 00 18 07 2F FF FF FF FF FF FF
(1119 usec)
<= 90 00
Status: No Error
/send 80d40007153ef0f0010009F4ACB09131420B8FE1
=> 80 D4 00 07 15 3E F0 F0 01 00 09 F4 AC B0
42 0B 8F E1 B4 CC 00 7A C5 2B
(527418 nsec)
<= 90 00
Status: No Error
/send 80d40008153ff0f00100EB9BC6DCDF74FF4E4B4
=> 80 D4 00 08 15 3F F0 F0 01 00 EB 9B C6 DC
FF 4E 4B 43 F2 E3 4A 67 27 B6
(347810 nsec)
<= 90 00
Status: No Error
/send 80d400061534f0f09000CEB726EDC01B793BC37I
=> 80 D4 00 06 15 34 F0 F0 90 00 CE B7 26 ED
79 3B C3 7D C0 9E 2F 76 85 34
(511452 nsec)
<= 90 00
Status: No Error
/send 00D616001E62640022333300010301000120010
=> 00 D6 16 00 1E 62 64 00 22 33 33 00 01 03
01 20 01 08 17 00 00 00 01 20 01 01 01 20
31 55 66
(328424 nsec)
<= 90 00
Status: No Error
```

```
/send 00D6170037000053414D504C452E434152442E41
=> 00 D6 17 00 37 00 00 53 41 4D 50 4C 45 2E
    52 44 2E 41 44 46 31 00 00 00 00 11 01 02
    18 00 10 11 01 02 98 12 18 00 10 00 00 00
    00 00 00 00 00 00 00 00 00 00 00 05
(371757 nsec)
<= 90 00
Status: No Error
/send 00B016001E
=> 00 B0 16 00 1E
(383732 nsec)
<= 62 64 00 22 33 33 00 01 03 01 00 01 20 01
    00 00 00 01 20 01 01 01 20 01 12 31 55 66
Status: No Error
/send 00B0170037
=> 00 B0 17 00 37
(298205 nsec)
<= 00 00 53 41 4D 50 4C 45 2E 43 41 52 44 2E
    46 31 00 00 00 00 11 01 02 98 12 18 00 10
    02 98 12 18 00 10 00 00 00 00 00 00 00 00
    00 00 00 00 00 00 05 90 00
Status: No Error
```

这个时候我们发现，所有指令已经能够 `no error` 了。实验一成功了。

2017/4/25 更新

基于验收过程中，跑真机过程中出现0x6B00报错的情况。注释掉了create\_file中关于P1参数的检验。注释掉了ins检验（ins检验其实应该没问题）。可能原因是因为真机中运行时指令和之前给的测试样本中的指令不完全一致，或者在create\_file中p1有某种特殊的意义而不一定比为0x00（这部分没有在ppt和word中找到）。

另外，基于这个问题，在Purse.java（Java卡的入口类）编写的时候，不推荐按照我的实现方法进行实现。