电子钱包的文件系统

有梦想的咸鱼四

3 / 23 / 2017

理论课内容

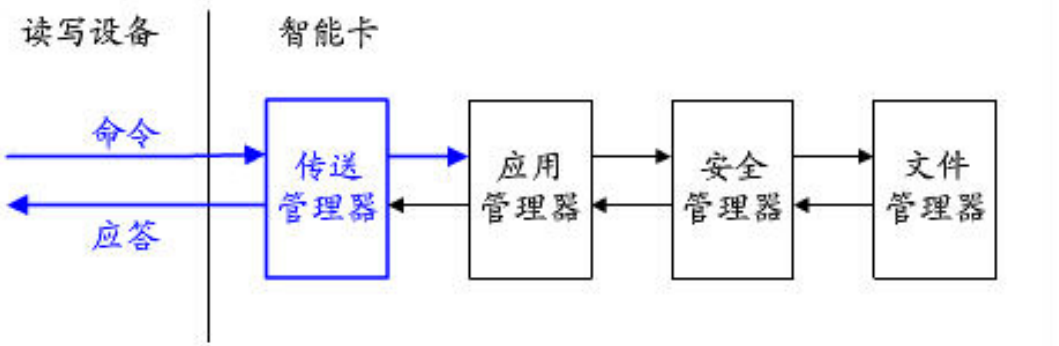
- 操作系统COS
- 文件管理
- 智能卡的个人化
- 智能卡的文件系统

操作系统COS

- COS的全称是Chip Operating System(卡内操作系统）
- COS的四个核心部分
  - 传送管理器
  - 安全管理器
  - 应用管理器
  - 文件管理器

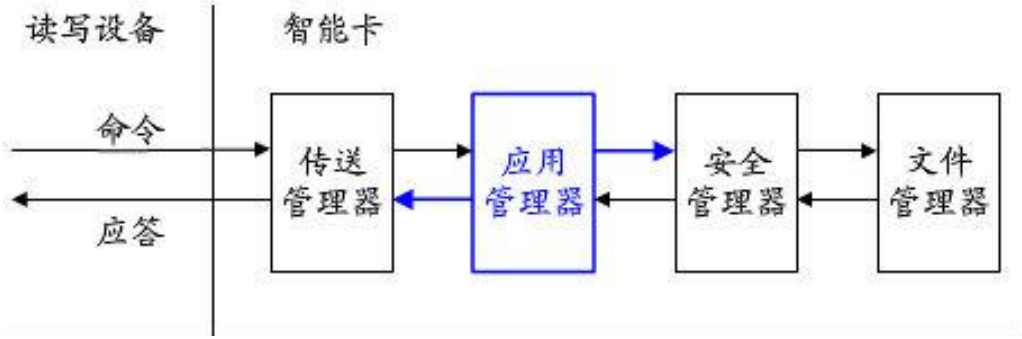
操作系统COS

- 传送管理器主要工作
  - 接收终端传送给卡片的命令
  - 发送卡片的应答数据



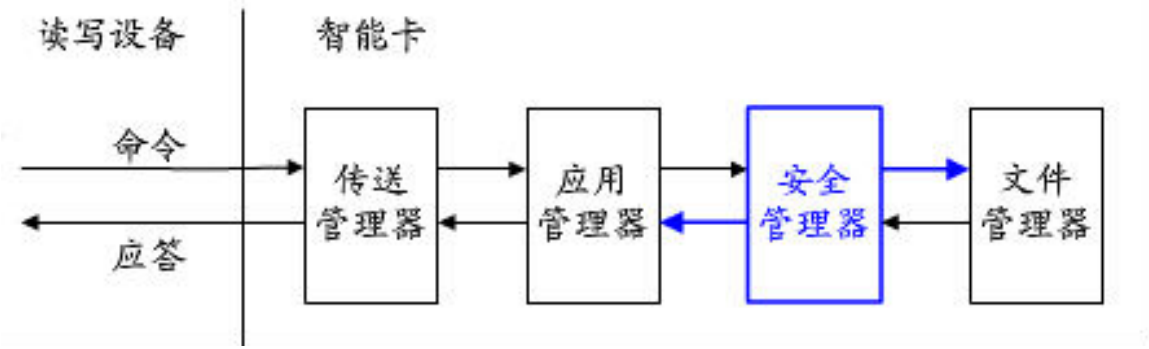
操作系统COS

- 应用管理器主要工作
  - 对接收到的每条命令中的参数进行正确性的分析和检查



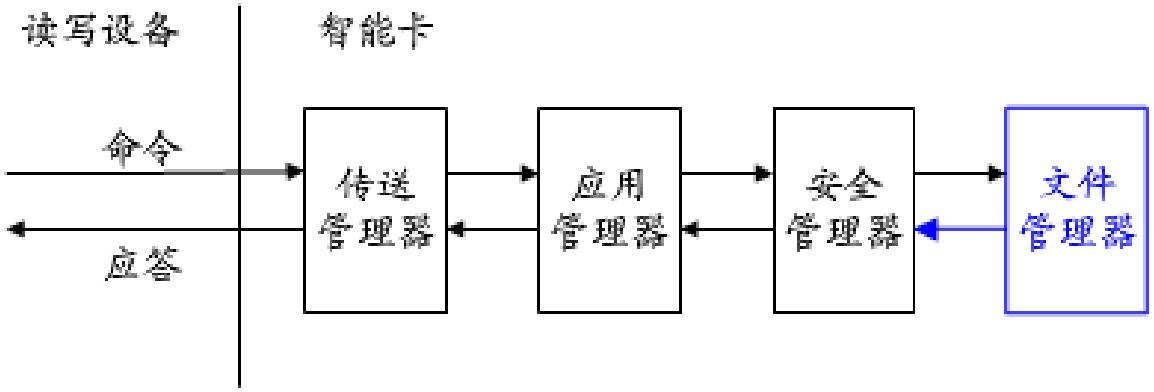
操作系统COS

- 安全管理器主要工作
  - 对传送来的信息进行安全性的检验或处理。主要完成MAC的生成和验证，其涉及到随机数的产生和数据加解密运算



操作系统COS

- 文件管理器主要工作
  - 实现对文件的操作和访问



文件管理

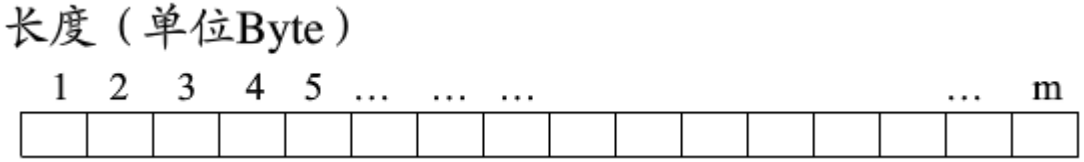
- 在电子钱包应用中，根据文件用途进行分类，可分为密钥文件、持卡人基本文件以及应用基本文件。
- 电子钱包文件系统如下所示

文件管理

- 文件的数据结构分类

① 二进制文件

透明二进制文件是卡片内所有文件的基础，卡片将数据作为一个字节流来进行处理

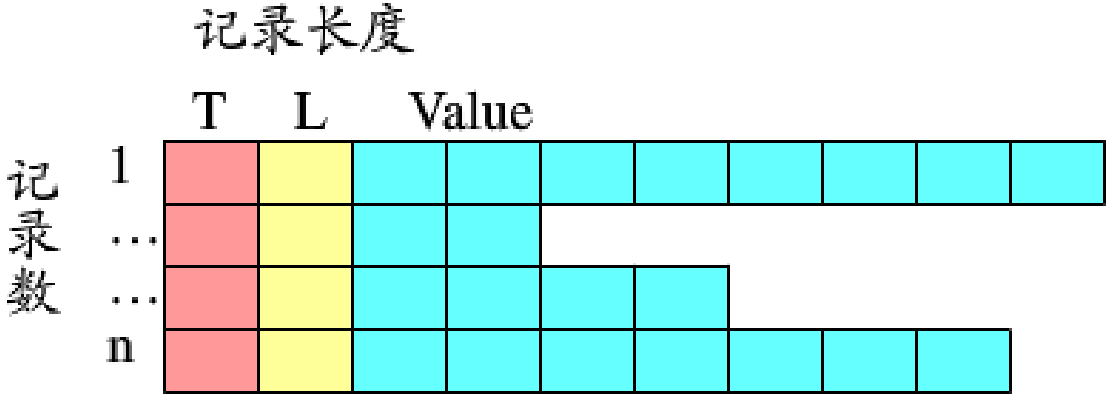


文件管理

- 文件的数据结构分类

② 变长记录文件

数据以记录为单位进行存储，但每个记录段的长度都不相同，通常变长记录以TLV（Tag-Length-Value）格式存在。在进行记录读取的过程中，以记录号或记录标识符来进行选择。



智能卡的文件系统

1. 密钥文件

- 密钥文件是存放密钥的文件，在任何情况下均不可由外界读出
- 每个应用只能有一个KEY文件，且必须最先建立
- KEY文件通常是一变长记录文件，每一条记录对应一条密钥
- 密钥＝密钥头（5个字节）+密钥值

文件体（密钥）							
标识	长度	密钥类型	使用权	更改权	密钥版本	算法标识	密钥值
06	0x15	34（TAC密钥）	F0	F0	90	00	CEB726EDC01B793BC37DC09E2F768534
标识	长度	密钥类型	使用权	更改权	密钥版本	算法标识	密钥值
07	0x15	3e（消费密钥）	F0	F0	01	00	09F4ACB09131420B8FE1B4CC007AC52B
标识	长度	密钥类型	使用权	更改权	密钥版本	算法标识	密钥值
08	0x15	3f（圈存密钥）	F0	F0	01	00	EB9BC6DCDF74FF4E4B43F2E34A6727B6

智能卡的文件系统

2. 持卡人基本文件

- 其文件中存储着持卡人的基本信息，如持卡人姓名、证件号码等
- 在二进制文件中，当数据的位数没有占满时，数据元左靠齐且右补十六进制”0”

文件体			
字节	数据元	长度	值
1	卡类型标识	1	00
2	本行职工标识	1	00
3-22	持卡人姓名	20	SAMPLE.CARD.ADF1
23-54	持卡人证件号码	32	11010298121800101101029812180010
55	持卡人证件类型	1	05

智能卡的文件系统

3. 应用基本文件

- 其文件中存储着应用的基本信息，如应用启用日期、有效日期等

文件体			
字节	数据元	长度	值
1-8	发卡方标识	8	6264002233330001
9	应用类型标识	1	03
10	发卡方应用版本	1	01
11-20	应用序列号	10	00012001081700000001
21-24	应用启用日期	4	20010101
25-28	应用有效日期	4	20011231
29-30	发卡方自定义FCI数据	2	5566

实验课内容

- Applet Demo简介
- Applet 基本方法
- TASK:智能卡的个人化
- API基本用法简介

Applet基本方法

- Java卡运行时环境(JCRE)调用Applet的五个基本方法来完成Applet的建立和执行。

① 安装 install()

② 注册 register()

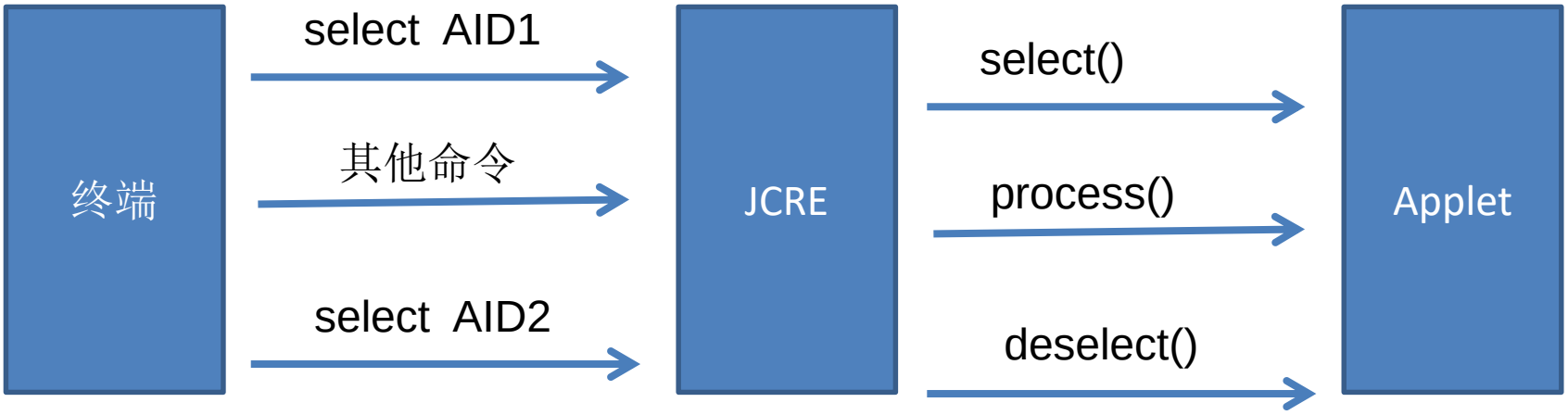
③ 选择 select()

④ 取消选择 deselect()

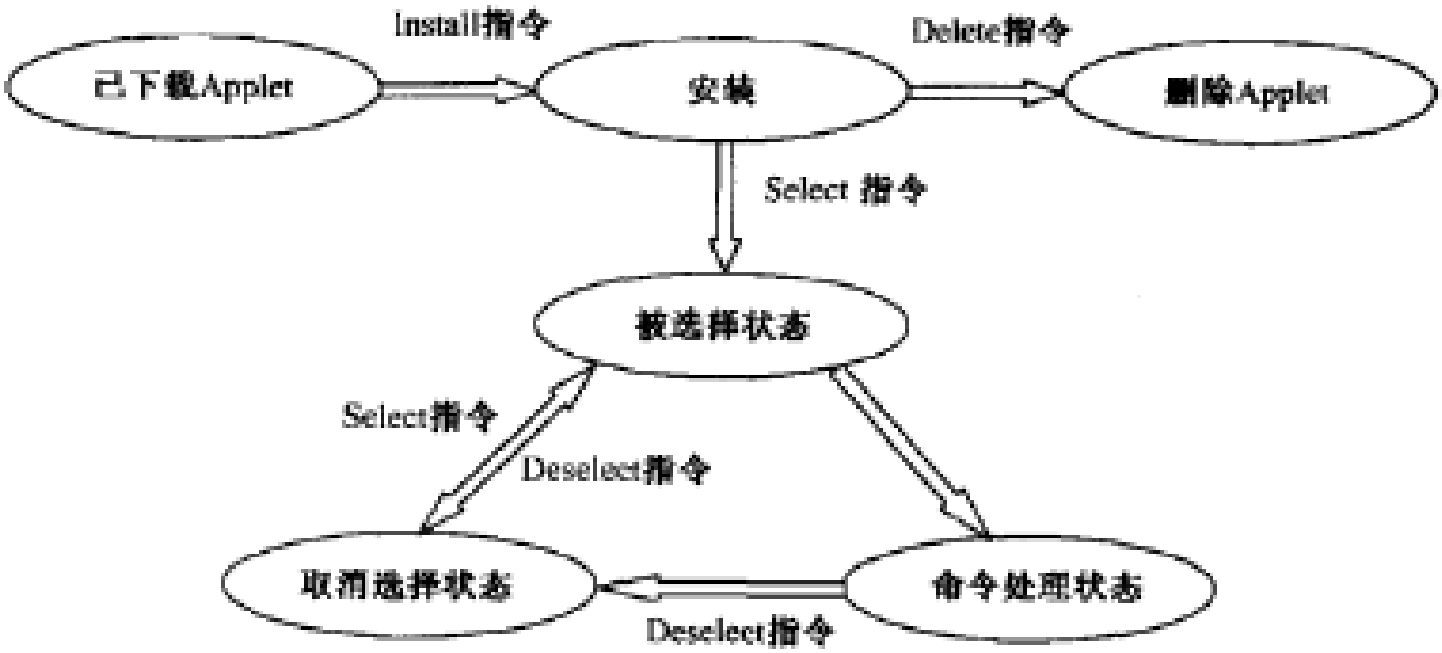
⑤ 执行 process()

Applet基本方法

- Java卡运行时环境(JCRE)调用Applet的五个基本方法来完成Applet的建立和执行。



Applet基本方法


Install指令
Delete指令
已下载Applet
安装
删除Applet
Select 指令
被选择状态
Select指令
Deselect指令
取消选择状态
命令处理状态
Deselect指令

安装 install()

```
public static void install(byte[] bArray, short bOffset, byte bLength) {
    new Purse(bArray, bOffset, bLength);
}
```

注册 register()

```
public Purse(byte[] bArray, short bOffset, byte bLength){
    papdu = new Papdu();
    TestCipher = new PenCipher();
    byte aidLen = bArray[bOffset];
    if(aidLen == (byte)0x00)
        register();
    else
        register(bArray, (short)(bOffset + 1), aidLen);
}
```

选择 select()

取消执行 deselect()

执行 process()

```
public void process(APDU apdu) {
    // Good practice: Return 9000 on SELECT
    if (selectingApplet()) {
        return;
    }

    byte[] buf = apdu.getBuffer();
    switch (buf[ISO7816.OFFSET_INS]) {
    case (byte) 0x00:
        break;
    default:
        // good practice: If you don't know the INStruction, say so:
        ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
    }
}
```

Applet通信基础

- C－APDU（APDU命令，command APDU）结构：

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

Applet通信基础

- R－APDU（APDU响应，response APDU）结构

命令选择部分	响应尾必写部分	
数据段	SW1	SW2

数据段：表示卡片返回的数据，其长度由C－APDU的LE所决定；

SW1、SW2：状态字1和状态字2，标志着C－APDU的执行情况，分别占一个字节

- 当返回的SW1,SW2是“90 00”时，意味着正常完成指令的处理，否则是出现了异常情况

Applet通信基础

- APDU有四种情况：

① 终端不向卡片发送数据，卡片也不需要返回数据

命令： CLA INS P1 P2　　响应：SW1 SW2

② 终端不向卡片发送数据，但卡片需要返回数据

命令： CLA INS P1 P2 LE　　响应：DATA SW1 SW2

③ 终端向卡片发送数据，但卡片不需要返回数据

命令： CLA INS P1 P2 LC DATA　响应：SW1 SW2

④ 终端向卡片发送数据，且卡片需要返回数据

命令： CLA INS P1 P2 LC DATA LE　响应：DATA SW1 SW2

Applet Demo架构

1、BinaryFile.java
2、condef.java
3、EPFile.java
4、KeyFile.java
5、Papdu.java
6、PenCipher.java
7、Purse.java
9、Randgenerator.java

Purse.java

COS的传输管理与应用管理部分的实现源码。在此部分中，卡片接收了APDU命令后，正确地对APDU命令进行判断和处理后，返回数据和状态码。

EPFile.java(第三次课内容)

电子钱包文件的实现源码,其中包含的内部数据元:

- EP余额
- EP联机交易序号
- EP脱机交易序号

PenCipher.java(第三次课内容)

COS安全管理部分的实现源码，其中实现了DES加密运算功能

Randgenerator.java(第三次课内容)

随机数的产生机制，可从中产生4个字节的随机数

condef.java

常数定义文件，主要定义了电子钱包应用所使用到的APDU命令的指令类别INS

```
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_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
}
```

Papdu.java

APDU的实现源码，APDU中除了包含CLA、INS、P1、P2、LC（LE）等必要部分，还包含了255个字节的数据段部分

初始化

```
public Papdu(){
    //apdu的数据段部分最大长度为255字节
    pdata = JCSystem.makeTransientByteArray((short)255, JCSystem.CLEAR_ON_DESELECT);
}
```

```
public boolean APDUContainData(){
    switch(ins){
    case condef.INS_CREATE_FILE:
    case condef.INS_LOAD:
    case condef.INS_NIIT_TRANS:
    case condef.INS_WRITE_KEY:
    case condef.INS_WRITE_BIN:

        return true;
    }
    return false;
}
```

判断是否存在该指令

BinaryFile.java

二进制文件的实现源码，其中实现了二进制文件的写入过程（不需要改动！）

```
/*
 * 功能：写入二进制
 * 参数：off 写入二进制文件 的偏移量； dl  写入的数据长度； data 写入的数据
 * 返回：无
 */
public final void write_bineary(short off, short dl, byte[] data){
    Util.arrayCopyNonAtomic(data, (short)0, binary, off, dl);
}

   /*
    * 功能：读二进制文件
    * 参数：off 二进制读取的偏移量；len 读取的长度； data 二进制数据的缓冲区
    * 返回：二进制数据的字节长度
    */
   public final short read_binary(short off, short len, byte[] data){
       return Util.arrayCopyNonAtomic(binary, off, data, (short)0, len);
   }
```

KeyFile.java

密钥文件的实现源码，其中实现了密钥的写入与密钥的读取（不需要改动！）

智能卡的个人化

“个人化”是将密钥信息、系统应用信息及持卡人个人信息写入卡片。其过程中所涉及到的命令主要包括

① create_file

② write_key

③ write_binary

④ read_binary

智能卡的个人化

- create_file

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

- 文件标识分别是：密钥文件“00”，应用基本信息文件“16”，持卡人信息文件“17”，电子钱包文件“18”
- 文件控制信息见文档“实验2”

智能卡的个人化

- write_key

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

- 密钥是逐条写入的，P2，Lc和Data的内容是根据密钥的不同而不同，具体见文档中密钥文件内容。

智能卡的个人化

- write_binary

代码	值
CLA	00
INS	D6
P1	文件标识符（持卡人基本文件为0x17,应用基本文件为0x16）
P2	欲写文件的偏移量（00）
Lc	数据域长度
Data	数据

- 该指令是用于写入持卡人文件和应用基本文件。

智能卡的个人化

- read_binary

代码	值
CLA	00
INS	B0
P1	文件标识符（持卡人基本文件为0x17,应用基本文件为0x16）
P2	欲读文件的偏移量
Le	数据域长度

- 该指令是用于写入持卡人文件和应用基本文件。

脚本

#//选择电子钱包文件
/select 1235318401

#//建立密钥文件
/send 80e00000073ffffffffffff

#//建立发卡方基本信息文件
/send 80e000160738001effffffff

#//建立持卡人基本信息文件
/send 80e0001707390037ffffffff

#//建立电子钱包文件
/send 80e00018072Fffffffffffff

脚本

#//写入密钥文件
#//增加消费密钥
/send
80d40007153ef0f0010009F4ACB09131420B8FE1B4CC007AC52B

#//增加圈存密钥
/send
80d40008153ff0f00100EB9BC6DCDF74FF4E4B43F2E34A6727B6

#//增加TAC密钥
/send
80d400061534f0f09000CEB726EDC01B793BC37DC09E2F768534

脚本

#//写入发卡方基本信息
/send
00D616001E626400223333000103010001200108170000000120010101200112315566

#//写入持卡人基本信息
/send
00D617003700053414D504C452E434152442E41444631000000001101029812180010110102981218001000000000000000000000000000000000005

脚本

#//读取发卡方基本信息
/send 00B016001E

#//读取持卡人基本信息
/send 00B0170037

任务

- Task 1：在condef.java和Papdu.java中添加指令INS
- Task 2：完善Purse.java中process()函数
- Task 3：完善Purse.java中handleEvent()函数
- Task 4：完善Purse.java中create_file()函数
- Task 5：完善Purse.java中write_bin()函数
- Task 6：完善Purse.java中write_key()函数
- Task 7：完善Purse.java中read_bin()函数

Task 1：在condef.java中添加指令INS

```
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_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
}
```

检查所有脚本文件中的INS指令是否都添加到了condef.java文件中

代码	值
CLA	00
INS	B0
P1	文件标识符（持卡人基本文件为0x17,应用基本文件为0x16）
P2	欲读文件的偏移量
Le	数据域长度

Task 1：在Papdu.java中添加指令INS

```
public boolean APDUContainData(){
    switch(ins){
    case condef.INS_CREATE_FILE:
    case condef.INS_LOAD:
    case condef.INS_NIIT_TRANS:
    case condef.INS_WRITE_KEY:
    case condef.INS_WRITE_BIN:

        return true;
    }
    return false;
}
```

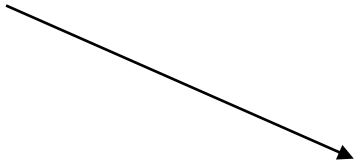
同理添加到switch中

代码	值
CLA	00
INS	B0
P1	文件标识符（持卡人基本文件为0x17,应用基本文件为0x16）
P2	欲读文件的偏移量
Le	数据域长度

Task 2：完善Purse.java中process()函数

步骤一：取APDU缓冲区数组引用并将之赋值给新建数组

Byte[] getBuffer()：取得APDU对象的通信缓冲区数组



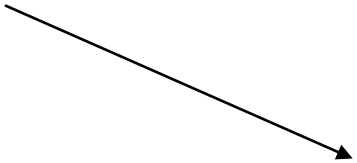
将这个对象保存到一个临时数组中，后面可以分步提取需要用到的APDU信息如：INS、p1、p2

注意这里用的是Byte型数组，我们的指令全是16进制数，所以数组每一个值都可以表示2个16进制数

Task 2：完善Purse.java中process()函数

步骤二：取APDU缓冲区中的数据放到变量papdu

Short setIncomingAndReceive()：基本接收命令，设置通信传输模式为终端到卡片，并在APDU缓冲区数组范围内，接收尽量多的数据并存储到数组中



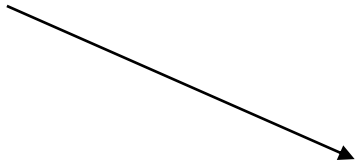
这是一个APDU类中的方法，因此直接对传入的apdu参数调用即可

Task 2：完善Purse.java中process()函数

步骤二：取APDU缓冲区中的数据放到变量papdu

ISO7816 接口

ISO7816.OFFSET_CLA
ISO7816.OFFSET_INS
ISO7816.OFFSET_P1
ISO7816.OFFSET_P2
ISO7816.OFFSET_LC
ISO7816.OFFSET_CDATA



表示对应字段的起始位置，功能类似于数组中的下标，可以直接用于提取之前临时数组中对应字段的数据

我们的指令全是16进制数，所以数组每一个值都可以表示2个16进制数，因此可以直接提取出来出CDATA外的所有字段

Task 2：完善Purse.java中process()函数

步骤二：取APDU缓冲区中的数据放到变量papdu

前面部分样例代码：

```
/*获得APDU缓冲区数组的引用*/
byte buffer[] = apdu.getBuffer();
/*通过引用可以反问缓冲区数据*/
byte ins = buffer[ISO7816.OFFSET_INS];
```

注意步骤二暂时不需要获取lc字段

Task 2：完善Purse.java中process()函数

步骤三：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

以下称之前读取得到的临时数组为buffer

伪代码：

If （papdu中存在该指令）：
    获取lc字段内容
    使用Util.arrayCopyNonAtomic()函数将buffer中data字段复制到papdu.data中
    if (buffer中的从offset_data起始后的长度 == papdu.lc)
        若一致则表示不需要返回消息所以le为0
        否则le等于buffer中的最后一位

如果不存在该指令
    le字段为buffer中lc的值，lc置为0

Task 2：完善Purse.java中process()函数

Util.arrayCopyNonAtomic()函数介绍：

参数1：byte[] src 复制的源数组
参数2：short offset 复制从原数组的offset位置开始
参数3：byte[] des 存入的数组
参数4：short offset 从offset位置开始存入
参数5：short length 复制的长度

Task 2：完善Purse.java中process()函数

步骤四：判断是否需要返回数据，并设置apdu缓冲区

伪代码：

If （rc为真 && papdu.le不为0）：
使用arrayCopyNonAtomic（）函数将papdu.data复制到一个临时数组中
使用setOutgoingAndSend将apdu置为卡片到终端模式并设置返回偏移量为0和长度为data数据长度

Task 2：完善Purse.java中process()函数

步骤四：判断是否需要返回数据，并设置apdu缓冲区

public void setOutgoingAndSend(short bOff, short len)：设置通信传输模式为卡片到终端，设置响应数据长度len及发送APDU缓冲区中bOff偏移量处len长度。

Task 3：完善Purse.java中handleEvent()函数

```
/*
 * 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;
}
```

case判断指令为：创建文件、写二进制文件、写密钥文件、读二进制文件

分别命名为调用函数：create_file()、write_bin()、write_key()、read_bin()

Task 4：完善Purse.java中create_file()函数

```
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;
}
```

case判断指令为：密钥文件、持卡人文件、应用基本信息

分别命名为调用函数：Key_file()、CARD_file()、PERSON_file()

Key_file()、CARD_file()、PERSON_file() 函数编写

判断cla、p1、lc字段是否符合要求，如果

不符合要求可以弹出如下异常：

```
ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
ISOException.throwIt(ISO7816.SW_WRONG_P1P2);
ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
```

符合要求则进行文件的创建：

进行创建对应的class类型

KeyFile() -> 无参数要求

BinaryFile() -> 参数为pupda.data

Task 5：完善Purse.java中write_bin()函数

判断cla是否符合要求，如果不符合要求可

以弹出如下异常：

```
ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
```

判断p1的值为写入cardfile还是personalfile：

调用：

```
BinaryFile.write_bineary()
```

进行写入文件内容（参数自己查看）

Task 6：完善Purse.java中write_key()函数

判断cla、p1、lc字段是否符合要求，如果

不符合要求可以弹出如下异常：

```
ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
ISOException.throwIt(ISO7816.SW_WRONG_P1P2);
ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
```

判断是否存在KEY文件（是否为NULL）：

不存在可以弹出如下异常

```
ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
```

存在则调用keyfile.addkey函数进行写入（参数自己去看）

Task 7：完善Purse.java中read_bin()函数

判断cla是否符合要求，如果不符合要求可

以弹出如下异常：

```
ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
```

判断p1的值为读出cardfile还是personalfile：

分别调用：

```
BinaryFile.read_binary ()
```

进行读出文件内容（参数自己查看）

输入脚本进行仿真

注意事项：通过脚本仿真时，请一行一行的复制到终端中进行运行，这样可以知道自己哪一段的代码是有问题的从而更好的进行debug

本屁屁踢为**商小四原**创，**未**经**允**许**不得**转载
引用请**注明出**处

小四出品，必属精品