天津滨海职业学院

RFID 应用技术综合实训

韩少男

2015 年 12 月

前 言

作为物联网专业的核心课程之一，RFID 应用技术课程的最终目标是使学生能够对此项技术有清晰的了解并能够进行实际运用。普遍的 RFID 实训教学设备所提供的开发环境大多建立在业务层之上，在接触不到底层通讯过程的情况下，整个课程的内容与信息系统开发无异，而一些完全建立在通讯、硬件之上的实训设备又不完全符合计算机专业学生的学习内容。所以，为本课程所专门开发的一套校园一卡通系统开发框架（以下简称为一卡通系统）本着简洁、复用的设计思想，并配以本教材作为讲解。

本课程要求学生在完成前四部分的学习后能够完成一项 RFID 应用系统，如，公交一卡通系统、地铁卡系统、门禁系统等。所以课程所要解决的核心问题可以归纳为：学生如何从一个仅了解 RFID 通讯原理的学习者转换成为一个 RFID 应用的开发者。

要解决这一问题并不是一件易事，因为这一过程中会出现许多商业化的设施：硬件平台、卡片与阅读器的结构、通讯协议、开源（第三方）库源码等，同时涉及到一些设计模式等不易理解的内容。基于此，我们设计了一个能够满足各种 RFID 应用场景的开发框架（包括简单的数据表示及使用方法），并完成了一套校园消费一卡通全部功能作为示例程序，以支撑课程内容。框架选取 Arduino 与 MFRC500 系列作为硬件平台，选取 C/C++作为开发语言，尽可能的少引入其他商业（或开源）设施，从而将注意力集中在如何应用 RFID 技术本身，作为学生只需有基本的 C 语言开发基础即可看懂示例程序并按照框架着手自己的开发项目。

课程本身为一个完整、能够实际应用的项目，并将项目本身拆解为四个部分，循序渐进的进行讲解，其中第四部分则是立足于业务层，讲解如何利用一卡通系统提供的库资源进行业务描述。第五部分则进入课程设计阶段，对课程设计内容，成果要求作出规定，并配与选题指导帮助学生完成设计。

每章内容将分为学习内容、实训项目、参考资料 3 个部分。其中参考资料部分即作为对学习内容的补充讲解，也作为扩展阅读。教材还配与 4 章附录，请注意，这些附录内容是开发中必不可少的参考。所以，本书既作为教材，同时也作为开发手册使用。

目 录

第一部分 校园一卡通管理系统

从本章开始，将对本课程所要求掌握的主要学习内容进行介绍与梳理。首先引入的校园一卡通系统将从展示的角度介绍一个完整 RFID 应具有哪些基本功能。第二、三小节将对本课程的学习目标提出具体的要求，标注了需要掌握的内容。最后，以简介的方式解释本课程所需的开发环境，以及搭建开发环境所要做的准备工作。

学习内容

1.1 基于 Arduino 与 MFRC500 的校园一卡通管理系统

校园一卡通系统（以下简称一卡通系统）是一套软硬件系统的统称。课程最终的考核方式即为参考一卡通系统的设计、利用一卡通系统提供的函数库资源，完成一个可实际应用的 RFID 项目。所以，应该关注并思考的内容是：

1. 一卡通系统具有哪些功能？
2. 一卡通系统是如何完成这样的功能的？
3. 我的项目如何完成这样的功能？

系统的硬件设施包括 4 个部分：Arduino UNO、MFRC522、Joystick、卡片。Arduino UNO 是 Arduino 开发板家族成员之一，Arduino 本身即为一单片机系统以及开发 IDE 的统称，可以认为一卡通系统是存放并且运行在这个单片机系统中的。MFRC522 是一块连接在 Arduino UNO 开发板上的芯片，它作为在 RFID 系统读写模型中“阅读器”的角色，它工作在 13.56MHz 频段。JoyStick 是一个简易的控制摇杆，它能够测量四个方向的输入并将其以模拟信号的形式传送给 Arduino UNO。在本系统中，JoyStick 扮演了用户输入控制器的角色，一卡通系统的所有功能选项都以上、下、左、右四项菜单方式提供给用户使用，比使用传统的键盘控制更加直观，有良好的用户体验。卡片是系统操作的对象，本课程所使用的卡片为当前广泛应用于公交、地铁、门禁、饭卡等各类充值控制卡片-----NXP 公司的 Mifare One 系列卡片。卡片的存储结构与工作过程是今后要讨论的一个极为重要的话题。

系统的软件设施为一套函数库文件以及工程文件，工程的编译工作须由 Arduino IDE 完成，编译完成后同样由 Arduino IDE 将二进制文件上传至 Arduino UNO 开发板中。上传完毕后，项目即开始运行，直到 UNO 断电停止运行。在运行的过程中，一卡通系统将一直提供服务，接收用户的控制指令并作出响应。

系统的输出将以串口信息的方式呈现，通过观察 Arduino IDE 提供的串口来获取系统响应。下面将从一卡通系统运行及输入响应的角度来介绍系统的功能。

一个一卡通系统应该具有基本的充值、消费、查询功能，同时，还要考虑到对卡片的管理，即开卡与销毁卡操作。开卡的涵义可以理解为：将一张新的 Mifare One 卡片纳入本系统进行管理，此卡片将用于本系统，且仅用于本系统。销毁卡的功能则与开卡相反，及从本系统中删除本卡片的所有信息，卡片今后还可以使用于其他系统（当然这有一定的条件）。既然系统具有记录卡片信息的功能，所以还应该具有一些列管理数据的功能，如查询已有数据、清除数据、统计等等。

那么，总结一卡通系统应该具有的功能如表 1-1：

	一级功能	二级功能
1	Management	Registration
		Destory
2	Consumption	Query
		Deposit
		Consume
3	DataBase Check	Flush Database
		Print DataBase
		Count Record
4	Return	

表 1-1 一卡通系统功能

系统运行后，将首先显示主功能菜单，每一项主菜单下都有一个二级菜单。所有的功能菜单将以 4 项为一组的方式显示，分别对应 JoyStick 的上下左右操作，如图 1-1 所示。

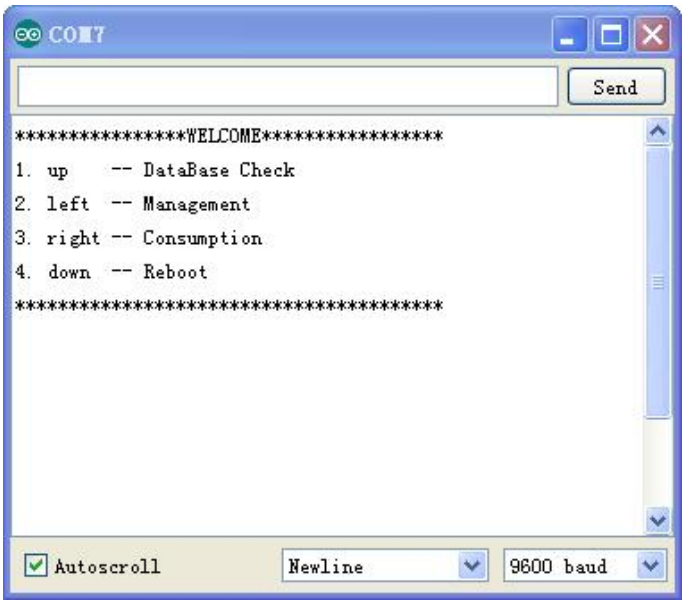


图 1-1 一卡通系统主功能菜单

拨动 JoyStick 摇杆可以选择二级功能。每一个二级功能菜单都设置 down 操作（向下）为返回一级菜单。二级菜单 Management 如图 1-2 所示。

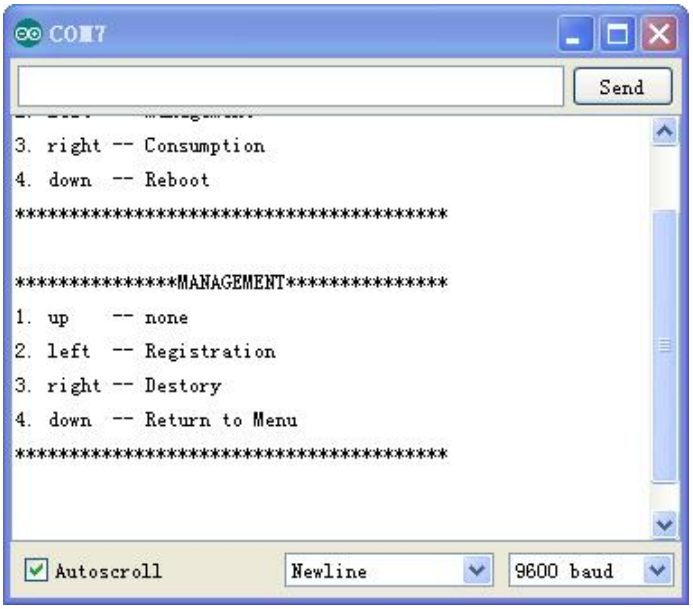


图 1-2 二级菜单：Management

为了简洁，所有二级菜单的内容以文本的形式进行介绍，如表 1-2 所示。

二级菜单：Management	**********MANAGEMENT********** 1. up -- none 2. left -- Registration 3. right -- Destory 4. down -- Return to Menu **
二级菜单：Consumption	**********CONSUMPTION********** 1. up -- Query 2. left -- Deposit 3. right -- Consume 4. down -- Return to Menu **
二级菜单：DataBase	**********DATABASE********** 1. up -- Flush Database 2. left -- Print DataBase 3. right -- Count Record 4. down -- Return to Menu **

表 1-2 二级菜单功能

有了对系统基本功能的了解，下面通过一个例子来为讲解一卡通系统的使用方法：

拟对一张新卡片（新卡片被定义为未记录于系统中的卡片）进行开卡，开卡后查询卡片信息，并向卡片中预存 500.99 元余额；操作卡片进行消费，金额为 327.25 元，观察消费后卡片余额；再次操作卡片消费 400 元，观察操作结果；查询数据库，显示统计信息；销毁卡片。为了描述操作与观察结果的简洁，所有操作方法及操作结果以表格方式展示，如表 1-3 所示。注意，JoyStick 的方向定义如下：拿起 JoyStick 摇杆指向使用者自己，有排针的一侧定义为“上”，其余依此类推。

	流程	操作	结果
1	1. 系统上电，打开串口监视器； 2. 将卡片静置于阅读器上。	-	**********WELCOME********** 1. up -- DataBase Check 2. left -- Management 3. right -- Consumption 4. down -- Reboot **
2	开卡： 1. 选择 Management； 2. 选择 Registration。	Left	**********MANAGEMENT********** 1. up -- none 2. left -- Registration 3. right -- Destory 4. down -- Return to Menu **
		Left	Registration SUCCESS! **********QUERY********** 1. Card serNum is: 23B7679A69 2. Card index is: 1 3. Reserve info is: 4. Card status is: 1 5. Card balance is: 0.00 **
3	返回主菜单	Down	略
4	充值： 1. 选择 Consumption； 2. 选择 Deposit 3. 在串口控制台输入金额，回车确认	Right	**********CONSUMPTION********** 1. up -- Query 2. left -- Deposit 3. right -- Consume 4. down -- Return to Menu **
		Left	**********DEPOSIT********** HINT: Please attach the card to the Reader. Card recognized. Card balance is: 0.00
		输入金额确认（500.99）	**********DEPOSIT********** HINT: Please attach the card to the Reader. Card recognized. Card balance is: 0.00

			Deposit Success! Current balance is:　500.99 **
5	消费： 1. 选择 Consume； 2. 输入金额并确认	Right	**********CONSUME********** HINT: Please attach the card to the Reader. Card recognized.Card balance is:　500.99
		输入金额确认（327.25）	**********CONSUME********** HINT: Please attach the card to the Reader. Card recognized.Card balance is:　500.99 Consume Success! Current balance is:　173.74 **
6	再次消费： 1. 选择 Consume，输入金额并确认	输入金额确认（400）	**********CONSUME********** HINT: Please attach the card to the Reader. Card recognized.Card balance is:　173.74 **********No enough money!********** **
7	查询余额	Up	**********QUERY********** 1. Card serNum is: 23B7679A69 2. Card index is: 1 3. Reserve info is: 4. Card status is:　1 5. Card balance is:　173.74 **
8	查询数据库（Database 菜单）	Right	The record number in DataBase is: 1
9	销毁卡片（Management 菜单）	Right	Destory card SUCCESS! Clear record in DataBase.

表 1-3　一卡通操作示例

一卡通系统同样应该考虑任何可能的错误情况，例如，在表 1-3 的流程 6：再次消费中，由于卡片余额不足，则无法消费，余额不发生改变并给予提示。

类似的情况有很多，例如：输入的金额为数字与字母，系统会如何处理？如果对已经开卡的卡片再次进行开卡，系统会如何处理？如果系统已经开卡卡 A，如果此时清空数据库，在对卡 A 进行销毁操作，系统会如何处理？请注意，这类问题必须考虑周全，因为这些问题一定程度上决定了系统的数据结构与设计模式。

1.2 课程学习目标与内容归纳

分析计算机行业中 RFID 的相关岗位可以发现，能够进行“针对 RFID 设备的开发”这一要求是出现次数最多的。“能够进行 RFID 系统的相关开发”其实并不是一件易学易懂的事情，这其中既涉及到系统设计与开发能力，也涉及到对支撑平台的了解。而这两部分内容恰恰是所有计算机课程都会讲授但却不易掌握的内容，它需要的是------积累。

本课程最终要求的成果为一项由学生自行设计并编码实现的 RFID 应用系统。所以，本课程所讲授与解决的核心问题是：如何从一个了解 RFID 技术原理的学生到能够完成系统设计、系统实现的项目开发者。

如上所述，这并不是一件易事。在这一角色转变的过程中会涉及到多项支撑技术与相关的商业化产品，关于这些内容如果讲的过少则不易于理解，不适于后续的系统开发要求；如果讲的过多则容易偏离 RFID 技术本身。为了保证在学习本课程后能够对 RFID 开发有一个基本的了解，我们选取了一个能够完成系统的“最小集”作为样例展示给大家-----校园一卡通系统（一卡通系统）的设计与实现。

通过对一卡通系统的硬件组成、软件资源库、系统设计模式的学习，可以对 RFID 开发有一个清晰的了解。这些内容将分别对应本教材后续章节。所以在这里，将今后会出现的各项知识与技术归纳成表格，可以对今后的学习起到参考作用，如表 1-4 所示。

原理、知识类	ISO/IEC 144443 规程
	Mifare One 卡片存储结构与工作过程
	Mifare One 卡片命令字
	Mifare One 卡片工作过程
	一卡通系统的设计模式与实现方法
商业技术类	Arduino IDE 与资源库
	Arduino UNO 开发板的基本结构与规格参数
	MFRC522 阅读器的寄存器

表 1-4 内容归纳

1.3 开发环境与准备工作

本课程中将会使用到的开发硬件软件环境如表 1-5 所示，其中校园一卡通系统函数库将用于自行设计与开发的支撑平台，由信息工程系教师自行开发，以后简称为“一卡通系统函数库”。

类别	设备与环境	描述
硬件平台	Arduino UNO × 1	一卡通系统的硬件平台，最终的项目开发也要求在本硬件平台上完成
	MFRC522 × 1	
	JoyStick 摇杆 × 1	
	Mifare One S50 卡片 × 1	
软件平台	Arduino IDE v1.05 r2 （含 Arduino 资源库）	注意版本如果过高，可能编译后出现运行不稳定
	其他开发编译工具	可选项，根据自己的习惯自行选择，甚至可以使用纯编辑器
	一卡通系统函数库（/RFID）	完成课程设计必须的库资源

表 1-5 准备工作

实训项目

1-A 搭建开发环境

一、连接硬件平台

如表所示连接 Arduino 与 MFRC522 及 JoyStick。

Arduino	MFRC522
10	NSS
13	SCK
11	MOSI
12	MISO
GND	GND
5	RST
3.3v	VCC

Arduino	JoyStick
GND	GND
5v	5v
VRx	A0
VRy	A1

二、安装 Arduino IDE v1.05 r2

三、安装 Arduino 串口驱动程序

四、复制一卡通系统函数库（RFID 目录）到$(Arduino)\libraries\下

五、安装其他开发用编译器或编辑器（可选）

在完成以上步骤后，启动 Arduino IDE，并将 Arduino UNO 通过 USB 连接线接入主机。请注意，Arduino UNO 接入主机后将被作为一个串口设备被识别，记录此串口编号，如 COM1，COM7，并在 Arduino IDE 中选择菜单 Tools ---Serial Port 来选中对应的串口端口。

1-B 运行一卡通系统

一、打开目录$(Arduino)\libraries\RFID\projectFINAL，启动工程文件 projectFINAL.ino。

二、将工程编译并上传至 Arduino UNO 开发板。

三、打开串口监视器，运行一卡通系统，并完成以下问题：

1. 利用一卡通系统获取你的卡片序列号。
2. 操作卡片开卡后，打印数据库的内容，找出数据库内容与卡号之间的联系。
3. 对卡片进行充值操作，然后进行查询操作。
4. 对卡片进行消费操作，要求消费金额超出卡片余额。
5. 再次进行充值操作，输入的金额为：537abc.3#6，观察系统响应。
6. 再次对卡片进行开卡操作。
7. 销毁卡操作成功的前提是什么？

8. 销毁卡操作成功后，打印数据库内容。

参考资料

I Arduino Plugin for Visual Studio

Visual.Micro.Arduino.Studio.vsix

http://www.visualmicro.com/page/User-Guide.aspx?doc=index

第二部分 商业设施及读写的原理与实现

本章将从硬件角度入手，阐述 Mifare One 卡片的存储结构与工作原理。其中，卡片的存储结构是日后项目开发中设计数据结构的依据，同时也是理解卡片读写机制基础。卡片的工作原理则是另一项重要内容。例如，一张卡片一直停留在信号耦合区，阅读器会对其进行多次相同业务的读写么？要回答这样的问题，则必须了解卡片工作流程以及卡片的状态定义。

对于阅读器，需要学习的是：阅读器是如何受控于 Arduino UNO 的。通信数据是怎样从 Arduino 传递到阅读器再发送给卡片等等。

有了对卡片与阅读器的基本了解，我们可以介绍本章最重要的内容，RFID 这项通信技术是如何在阅读器上实现的，或者说是怎样的一些列代码可以使阅读器这样来工作。我们会深入解析 MFRC522 阅读器的代码来了解 RFID 的读写机制。

最后，将介绍 Arduino 的资源库的使用与参考方法。我们会以 Serial 类为例进行介绍，并且引入在后续内容中十分重要的 EEPROM 资源库。

本章内容后附有很多参考文献，它们由于篇幅的限制无法作为正文出现在本书中，但是对于系统开发有及其重要的作用。参考文献的部分有些是将其与课程紧密相关的部分提取出来作为扩展阅读，有些则给出出处的链接，便于课下自行学习。

学习内容

2.1 Mifare One S50 的结构与工作过程

一、RFID 卡片的国际标准

RFID 的典型应用场景有很多种，例如公交地铁系统，大型购物中心自动结算，不停车收费（ETC）等等。这些不同的应用场景对 RFID 作用距离的要求是不同的，使用的频段也不一样。在本课程中所使用的 RFID 标准为近耦合型 RFID（ISO/IEC 14443），国际标准化组织 ISO 编制了三种应用于不同场景的 RFID 卡片标准，如表 2-1 所示。

标准	类型	作用距离
ISO/IEC 10536	密耦合 CICC (Close Coupled IC Card)	0 ~ 1cm
ISO/IEC 14443	近耦合 PICC (Proximity Coupled IC Card)	0 ~ 10cm
ISO/IEC 15693	疏耦合 VICC (Vicinity Coupled IC Card)	0 ~ 1m

表 2-1 RFID 卡片 ISO 标准

在校园一卡通系统中所使用的为近耦合型卡片，它适用于电子钱包快捷支付领域，日常生活中使用的公交地铁卡片都属于此类卡片。对于近耦合 RFID 卡片，在 RFID 工作模型中称为 PICC，相应的阅读器被称为 PCD (Proximity Coupling Device)。卡片（PICC）与阅读器（PCD）之间的数据传输过程由 ISO/IEC 14443 标准定义，一共定义了两种通信模式：Type A 与 Type B。Type A 与 Type B 所使用的射频载波都为 13.56MHz，主要区别在于载波的调制与编码方式。Type A 技术的主要代表厂商是 Philips，Type B 技术的主要代表厂商是 Freescale（Motorola）。目前市场上占据绝大数份额的近耦合卡片为 Philips NXP 的 Mifare One 系列卡片，所以 Mifare One 卡片的结构是本课程学习的重点。Mifare One 系列卡片有 S50 及 S70 两种类型。两种卡片的物理特性及工作工程一致，区别在于 S70 的存储容量比 S50 大。

二、Mifare One S50 卡片的结构

MiFare One S50 卡片内部包括一块微型芯片和一个天线。在工作时，卡片与阅读器之间通过 13.56MHz 频段进行通信。由于卡片本身无源，卡片在工作时通过空间耦合的方式从阅读器所发出的电磁波中获取能量，在读写距离最大 10cm 的范围内支持 106Kbps 的数据率。微型芯片内部包含数字控制单元、射频接口，以及一块 EEPROM，如图 2-1 所示。

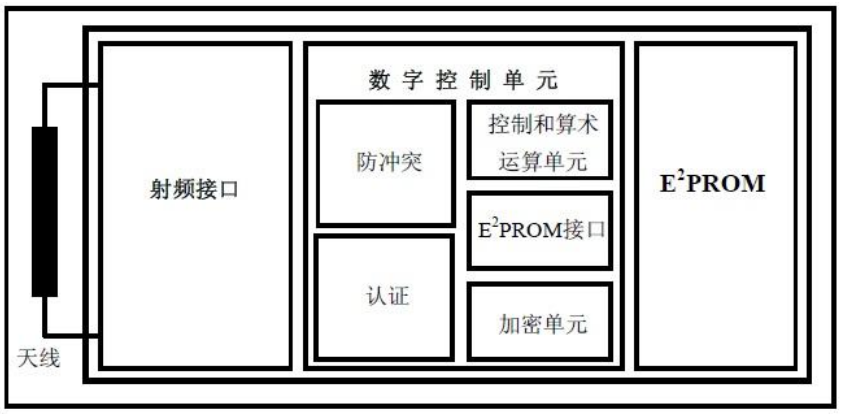


图 2-1 MiFare One 卡片结构

1. 卡片的数据存储

卡片的微型芯片内建了 EEPROM 存储器，存储容量为 8Kbit。整个存储空间被划分为 16 个扇区，记为扇区 0-15，每个扇区有 4 个块，记为块 0、1、2、3，每个块的大小为 16 个字节，对卡片的读写以数据块为单位进行，即每次阅读器与卡片通信，所传输的数据都是 16 字节的倍数，这一点对后续理解阅读器的实现非常重要。每个扇区的尾块为本扇区的密钥块，扇区其余三个块称为数据块，数据块保存卡片的所需记录的内容，例如公交卡的余额信息就存放在数据块中。数据块的信息大多情况下是不允许用户私自修改的，所以对扇区的任意数

据块进行访问、读写都要先经过本扇区密钥块的认证。每张卡片具有唯一的序列号，序列号及制造商信息被保存在 0 扇区第 0 块中，此数据块被设置为只读，如图 2-2 所示。

扇区号	块号	描述
15	63	本扇区密钥块
	62	数据块
	61	数据块
	60	数据块
…	…	…
1	7	本扇区密钥块
	6	数据块
	5	数据块
	4	数据块
0	3	本扇区密钥块
	2	数据块
	1	数据块
	0	卡片序列号及厂商标识

图 2-2 卡片的存储组织

近耦合卡片通常的应用场景是快捷支付，所以 Mifare One 卡片专门设计了“电子钱包”功能：卡片允许将数据块转换成专门存放表示金额的数值块，并允许阅读器对卡片发出调整此数值的指令----增值、减值操作。所以，Mifare One 卡片可以认为具有三类块：

- 数据块：用于存放任意类型数据信息，数据以字节单位直接存放，支持读、写操作
- 数值块：可由数据块转换而成，以特定格式存储一定范围浮点数，支持读、写、增值、减值操作
- 密钥块：存放本扇区密钥，支持读、写操作

2. 卡片的密钥与权限

扇区密钥块与数据块的大小是一样的，同为 16 字节，密钥块设置在每一个扇区的最末，如图 2-2，所以密钥块也称为尾块。所以，一张 S50 卡片具有 16 个密钥块，每一个密钥块负责本扇区内数据块的读写控制。例如，如果要读写卡片的第 5 数据块，它在第 1 号扇区，则必须通过本扇区密钥块----卡片的第 7 块----的验证。

每一个密钥块的结构都是相同的，被分为三个部分，其中第 0~5 字节称为密钥 A（KeyA），中间第 6~9 字节为权限信息位，最后 10~15 字节称为密钥 B（KeyB），如图 2-3 所示。密钥 A 与密钥 B 是不可以读出的，对扇区数据块的操作要通过密钥 A 或 B 的验证。这个验证过程由卡片内部逻辑电路认证模块完成（而不是阅读器），保证了卡片数据加密的统一性与安全性。通过了密钥验证后并不代表可以对本扇区内的所有 4 个块（包括密钥块自身）可以随意操作，具体对那个块，做哪一种操作，还需要依据权限信息位。

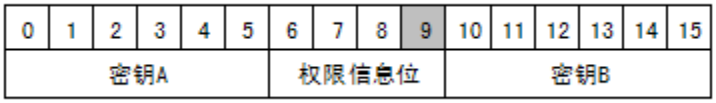


图 2-3 密钥块的结构

权限信息位由 4 个字节组成（6~9），但 ISO/IEC 14443 仅对其中前 3 个字节（6~8）进行了定义，最后一个字节（9）作为保留，用于用户数据。为了能够对本扇区 4 个块可能出现的读、写、增值、减值操作进行清晰的描述，ISO 对这些可能出现的操作可以使用三位二进制来进行表示，这三比特被标识为 C_1、C_2、C_3。对于数据块（数值块）来说，三个比特位用于标识是否能够读、写、增值、减值；而对于密钥块来说，三个比特位用于标识是否能够读写密钥块的密钥 A、密钥 B 或权限位，这些情况如图 2-4 与图 2-5 所示。

Access bits			**Access condition for**				**Application**
C1	C2	C3	read	write	increment	decrement, transfer, restore	
0	0	0	key A|B	key A|B	key A|B	key A|B	transport configuration[1]
0	1	0	key A|B	never	never	never	read/write block[1]
1	0	0	key A|B	key B	never	never	read/write block[1]
1	1	0	key A|B	key B	key B	key A|B	value block[1]
0	0	1	key A|B	never	never	key A|B	value block[1]
0	1	1	key B	key B	never	never	read/write block[1]
1	0	1	key B	never	never	never	read/write block[1]
1	1	1	never	never	never	never	read/write block

图 2-4 数据块的访问权限

Access bits			**Access condition for**						**Remark**
			KEYA		**Access bits**		**KEYB**		
C1	C2	C3	read	write	read	write	read	write	
0	0	0	never	key A	key A	never	key A	key A	Key B may be read[1]
0	1	0	never	never	key A	never	key A	never	Key B may be read[1]
1	0	0	never	key B	key A|B	never	never	key B	
1	1	0	never	never	key A|B	never	never	never	
0	0	1	never	key A	key A	key A	key A	key A	Key B may be read, transport configuration[1]
0	1	1	never	key B	key A|B	key B	never	key B	
1	0	1	never	never	key A|B	key B	never	never	
1	1	1	never	never	key A|B	never	never	never	

图 2-5 密钥块的访问权限

- Key A: 验证密钥 A 即可
- Key B: 验证密钥 B 即可

- Key A|B: 验证密钥 A 或 B 都可
- Never: 验证那个密钥都不可

对于数据块的权限，我们举几个例子来进行说明，例如：

$C_1C_2C_3 = 000$：最宽松，验证密码 A 或密码 B 后可以进行任何操作；

$C_1C_2C_3 = 111$：无论验证哪个密码都不能进行任何操作，相当于把对应的块冻结了；

$C_1C_2C_3 = 010/101$：都是只读，数据块如果保存的是智能看但不能改的基本信息，可以设为这两种模式；

$C_1C_2C_3 = 001$：只能读和减值，电子钱包一般设为这种模式，例如公交卡，用户只能查询或扣钱，不能加钱，充值的时候需要在总控端读卡器先改变控制位使卡片可以充值，充完值再改回来。

图 2-4 与 2-5 中表示的 8 种权限是对一个数据块来讲的，而一个扇区中有 4 个数据块，所以在标准中，有 4 组 C_1、C_2、C_3，用于分别表示 4 个块的权限。定义一个扇区编号从低到高分别表示本扇区的 0、1、2、3 扇区，其中 3 扇区为密钥块，那么权限位 C_1、C_2、C_3 就有 C_{10}、C_{20}、C_{30} 到 C_{13}、C_{23}、C_{33} 共四组，共计 12 个比特，如图 2-6 所示。

Access Bits	Valid Commands		Block	Description
$C1_3$, $C2_3$, $C3_3$	read, write	→	3	sector trailer
$C1_2$, $C2_2$, $C3_2$	read, write, increment, decrement, transfer, restore	→	2	data block
$C1_1$, $C2_1$, $C3_1$	read, write, increment, decrement, transfer, restore	→	1	data block
$C1_0$,$C2_0$, $C3_0$	read, write, increment, decrement, transfer, restore	→	0	data block

图 2-6 四个块的四组控制位

最后，这 4 组权限位共计 12 比特按照如图 2-7 的方式分布在密钥块的第 6~8 字节中。ISO/IEC 14443 标准考虑到权限位的重要性，又对这 12 比特按照反码再次存储了一遍，共计 24 比特，刚好由第 6~8 这 3 个字节覆盖。

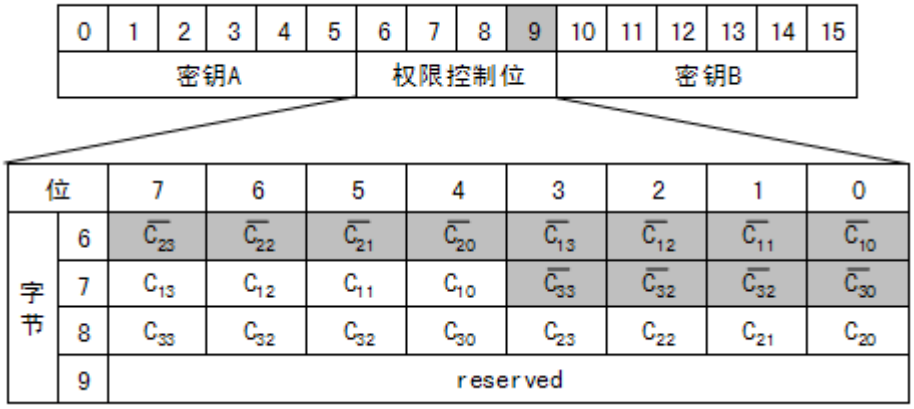


图 2-7 四个块的四组控制位

了解了密钥块结构之后，我们来观察一下一张卡片在出厂时（空卡）密钥块的默认状态，在空卡状态下，每一个扇区的密钥块都按照以下模式设定：

密钥 A：0xFFFFFFFFFFFF

密钥 B：0xFFFFFFFFFFFF

权限位：0XFF078069

对于更多的关于 S50 存储结构标准的描述，我们推荐大家阅读本章参考资料 II Mifare S50 Product Data Sheet 中的第 8.7 节进行进一步的了解。

三. Mifare One S50 卡片的命令字

卡片是一种具有特定行为、功能的设备，卡片能够执行的行为是预先被定义好的，一旦接受到阅读器相应的指令，就开始执行，我们称这样的指令为卡片的命令字。由 Mifare 与 ISO/IEC 14443 共同定义了卡片的命令字有以下几种：

Request：在阅读器天线的有效工作范围内寻找卡片，卡片在接到此命令后会发送自身的卡号。

Anticollision：当一张以上的卡片出现在天线工作范围内，为了保证只选中一张卡片，卡片会进行此操作防止数据碰撞。通常此操作可以在进行 *Select* 的过程中同时完成。

Select：选取卡片操作，即在一张卡片与阅读器之间建立通信，以卡号作为标识。卡片能完成此操作则认为自己可与阅读器进行数据传输，否则即便卡片在阅读器范围内被 *Request*，也必须等待。

Authentication：要求卡片使用密钥 A 或 B 对进行读写权限认证。

MIFARE Read：要求卡片将进行读操作。

MIFARE Write：要求卡片将进行写操作。

MIFARE Increment：要求卡片进行增值操作。

MIFARE Decrement：要求卡片进行减值操作。

Halt：要求卡片进入空闲状态。

以上这些命令字表征了卡片能够执行的操作，这些操作的方法由卡片内部芯片完成，所以这些命令的实现无需由自定义实现，开发者只需要通过阅读器将这些命令字发送给卡片即可。Mifare S50 的命令字的编码如图 2-8 所示。

Command	ISO/IEC 14443	Command code (hexadecimal)
Request	REQA	26h (7 bit)
Wake-up	WUPA	52h (7 bit)
Anticollision CL1	Anticollision CL1	93h 20h
Select CL1	Select CL1	93h 70h
Anticollision CL2	Anticollision CL2	95h 20h
Select CL2	Select CL2	95h 70h
Halt	Halt	50h 00h
Authentication with Key A	-	60h
Authentication with Key B	-	61h
Personalize UID Usage	-	40h
SET_MOD_TYPE	-	43h
MIFARE Read	-	30h
MIFARE Write	-	A0h
MIFARE Decrement	-	C0h
MIFARE Increment	-	C1h
MIFARE Restore	-	C2h
MIFARE Transfer	-	B0h

图 2-8 命令字编码

三、Mifare One S50 卡片的工作过程

在阅读器第一次识别卡片时，阅读器需要通过寻卡来得到卡片的状态（是否空闲）、类型（Type A 或 Type B）、以及卡号，并作防止冲突检测，然后进行选卡，此后就可以按照需要对卡片进行读写，读写前需要对读写权限进行认证。简化的工作流程如图 2-9 所示。

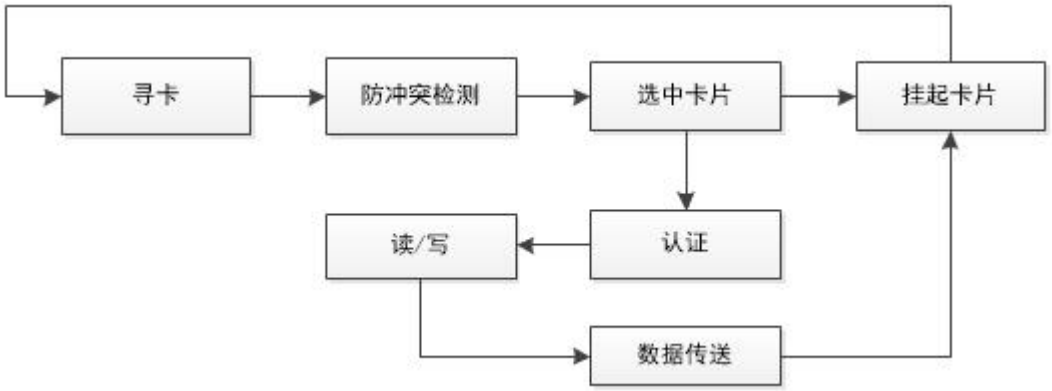


图 2-9 卡片的工作流程

2.2 阅读器 MFRC522

校园一卡通系统选取 MFRC522 作为系统的阅读器，它处于 Arduino 与卡片之间，可以看做数据传输的媒介，上位机所发送的所有数据都要通过阅读器传送给卡片，同样，卡片返回的数据也要由阅读器来接受并转交给上位机。

MFRC522 同样是 NXP 公司的产品，它工作在 13.56MHz，支持 ISO 14443 Type A/Mifare 模式，搭配使用的卡片为 Mifare Class 系列，即 2.1 小节所介绍的 S50。注意，MFRC522 仅支持 Type A，所以对于图 2-9 的寻卡阶段，在程序设计与开发的过程中我们无需考虑 Type B 的情形，这一定程度上简化了开发规模。

对于计算机专业来讲，MFRC522 的结构细节不必过多深究，所以我们以下对开发过程将遇到的三个方面问题进行讲解：阅读器的结构、阅读器的命令字、寄存器。

一、MFRC522 阅读器的结构

MFRC522 的结构简图如图 2-10 所示。

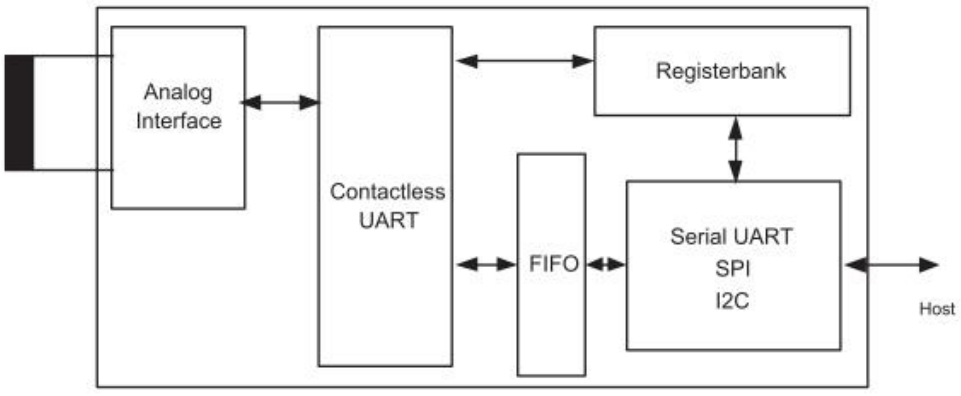


图 2-10 MFRC522 的结构

我们通过结构图来观察MFRC522能够做什么。Analog Interface模块连接阅读器的天线，它的主要功能是将对从/到天线的模拟信号进行调制解调；Contactless UART 负责处理所有与协议相关的通信操作，即协议中的所有编码、组帧都是由此模块来完成，所以它必须直接与FIFO 缓冲区[1]相连，也同时要与寄存器（组）相连；FIFO，数据缓冲区，上位机与卡片之间所交互的数据都会通过此缓冲区，大小为 64 字节；Registerbank，寄存器组，几乎 MFRC522 阅读器能够执行的所有操作都与寄存器组相关，其中包括命令寄存器、状态寄存器等，这是后续进行开发时需要关注的重要内容；结构模块，MFRC522 提供了三类结构与主机相连，分别为 Serial URT、SPI、I^2C，在本一卡通系统中，我们采用 SPI 接口与 Arduino 相连。

二、MFRC522 阅读器命令字

同 Mifare 卡片一样，操作阅读器的方法即向阅读器发送命令字，命令字定义了阅读器能够执行的所有动作，如图 2-11 所示。需要注意，MFRC522 的所有命令字必须发送给命令寄存器才能够有效。

Command	Command code	Action
Idle	0000	No action; cancels current command execution.
Mem	0001	Stores 25 byte into the internal buffer
Generate RandomID	0010	Generates a 10 byte random ID number
CalcCRC	0011	Activates the CRC co-processor or performs a selftest.
Transmit	0100	Transmits data from the FIFO buffer.
NoCmd Change	0111	No command change. This command can be used to modify different bits in the command register without touching the command. E.g. Power-down.
Receive	1000	Activates the receiver circuitry.
Transceive	1100	Transmits data from FIFO buffer to the antenna and activates automatically the receiver after transmission.
-	1101	Reserved for further use
MFAuthent	1110	Performs the MIFARE® standard authentication as a reader
Soft Reset	1111	Resets the MFRC522

图 2-11 MFRC522 命令字及编码

三、MFRC522 阅读器的寄存器

[1] FIFO 缓冲区：为了方便理解，本教材中的所有关于 FIFO 结构的数据存储空间一律都将用缓冲区这一词进行讲解。

图 2-10 所示结构中，寄存器组与非接触串口模块之间是直接相连的，其实，操作 MFRC522 阅读器的方式即向寄存器组中的 CommandReg 寄存器发送相应的命令字，同时，阅读器处理在执行某命令的过程中所涉及的一系列状态信息、中断、校验也都要通过寄存器组的其他相应寄存器来完成。

寄存器的定义与状态信息非常重要，与我们的开发直接相关。由于篇幅原因，在开发过程中，请查看本教材的参考资料 III MFRC522 Contactless Reader IC Product Data Sheet 的第 9 节，里面详细的介绍了每一个寄存器的定义与状态位。

2.3 Arduino UNO 开发环境与开发规程

Arduino UNO 在一卡通系统中作为主机的角色，通过 SPI 接口与 MFRC522 相连，它是一个完整的主机平台，并搭配有自己的编译环境（本系统采用 Arduino IDE v1.05 r2），我们所有的项目开发都是针对此平台进行的。

一、Arduino UNO

Arduino UNO 是一块搭载了 ATmega328P 单片机的 8 位主机开发板，具有 14 脚数字 IO 接口及 6 脚模拟 IO 接口，时钟频率 16MHz，可以通过 5v 电源或者 USB 接口供电。以下参数（ATmega328P）则是需要关注的，请务必牢记：

- Flash Memory：32KB
- SRAM：2KB
- EEPROM：512B/1KB

其他参数如果有兴趣关注可以参阅 *https://www.arduino.cc/en/Main/ArduinoBoardUno*

二、Arduino IDE v1.05 r2

Arduino IDE 允许采用 C/C++语言进行编程开发[2]，并配以控制接口、数据存储的库提供给开发者。这一部分内容不再赘述，提供以下参考资料，在需要时进行查阅即可。

https://www.arduino.cc/en/Reference/HomePage

https://www.arduino.cc/en/Reference/Libraries

[2] 采用 C/C++语言进行编程开发: 严格来讲 Arduino 采用的语言是自定义的一种语言，但此语言与 C 语言完全兼容，并支持大部分 C++语言特性，有兴趣的同学可以登录 https://www.arduino.cc/ 与 ISO/IEC 14882:2014 自行比对。

在后续的开发中，我们主要会使用到的库是 EEPROM，它允许我们向 Arduino UNO 中存储最多 1KB 的数据，按并字节编址。后续我们在讲解一卡通系统存储结构的设计时主要就是针对如何使用好这 1KB 的空间进行讨论。一卡通库源码 Storage.h 大量的使用了 EEPROM 库。为了大家能够快速阅读并掌握一卡通系统的库源码，请仔细阅读参考资料 IV Arduino libraries: EEPROM，并上机实验示例程序。

2.4 控制摇杆 JoyStick

本系统选取了在课程开发套件中赠送的一块控制摇杆 JoyStick 用于用户与系统的交互控制。此控制摇杆为一个单纯的电子部件，JoyStick 的功能非常之简单：根据摇杆对中心的倾斜位置产生 X 轴与 Y 轴的模拟信号并直接输出到 X 与 Y 管脚。在系统中，它由 Arduino UNO 进行供电，并连接 2 个模拟管脚来检测当前摇杆的位置。

JoyStick 所输出的模拟信号在 Arduino 中将被量化为 0~999 的数值，其中 500 表示摇杆在当前轴的中间位置，那么我们可以根据 X、Y 两个轴的数值来判定当前摇杆摇向上、下、左、右（自定义）中的哪一个方向。参考本章 2-C 题目 2。

这样，我们就可以将四个方向当做四个按键，来设计我们的界面功能，比如上可以用于表示卡片管理菜单，左表示消费管理菜单等等。关于如何一个可以完成响应数据输入的 JoyStick 的进行在第三部分详细介绍。

2.5 校园一卡通系统的开发

在了解了卡片、阅读器、主机三部分之后，我们来简要描述一下整个系统的工作过程。作为开发者，会针对阅读器的功能与命令字编写阅读器功能代码，比如寻卡、验证密钥、校验、读写卡片、停止卡片等，同时还要完成一卡通系统应用环境的业务代码，比如数据存储、管理卡片、开通卡片、注销卡片、充值、消费等。所有这些代码全部运行在主机 Arduino UNO 上，Arduino UNO 通过 SPI 接口与阅读器进行命令与数据的收发，收发的过程数据会通过阅读器的 FIFO 缓冲区。数据在阅读器中不会停滞，它要么流向卡片，要么流向 Arduino，从而完成了主机与卡片之间的数据交互。用户通过连接在 Arduino 上的控制摇杆，根据 Arduino IDE 串口监视器上的信息控制整个系统。

在本系统的开发过程中，对于阅读器功能的实现参考了某开源代码，2.5 节将对其实现进行解释。而其他所有内容则完全需要自行开发。根据本课程的要求，笔者开发了一套可以

完全模拟校园一卡通管理系统的完整代码，并通过库形式提供给学生。在完成课程设计时，同学们可以直接引用其中的代码，但要求业务层（可以只编写一个类）必须自行完成。

2.6 读写机制的实现

本节将着重解释第三方开源库 *RFID*，借此展示如何对阅读器进行操作，进而控制卡片的行为。本库是针对 MFRC522 开发的，结构很简单，仅有一个名为 *RFID* 的类，被定义在 *RFID.h* 与 *RFID.cpp* 两个源文件中，完整源代码参见附录 B。本节中将大量引用附录 B 中的内容，请同学们务必参照附录 B 来阅读本节的内容。

RFID 的主要功能可以归结为两个问题：1. 如何控制阅读器发起各种操作，或者说阅读器是如何完成一次卡片的工作流程的；2. 卡片如何理解阅读器发出的指令进而完成指定操作。在这个过程中，我们还可以看到前文提到过的命令字、组帧、缓存的问题的处理方法。

一、*RFID.h*

文件的开始引入了两个文件，*Arduino.h* 与 *SPI.h*，这两个文件是我们在 Arduino 上进行开发必备的两个库，后续的一卡通系统框架也通过 *RFID.h* 也引入了这两个库。其中 *Arduino.h* 作为在 Arduino 上进行系统开发必备引入，而引入 *SPI.h* 是由于阅读器与 Arduino UNO 之间是通过 SPI 接口进行连接的。

RFID.h：11 行定义了 MAX_LEN 为 16，类中所有函数所开的数组都以 MAX_LEN 为长度。这些数组全部为 unsigned char 类型，都被用于存储阅读器与卡片之间所要传输的数据。所以定义 MAX_LEN 为 16 是因为阅读器一次传送数据最长为 16 字节，即 Mifare 卡片一个块的大小。

RFID.h：13~35 行定义了卡片的命令字与阅读器的命令字，这是我们做具体功能开发时必须用到的。这些命令字的编码定义来自与 2.1 节与 2.2 节所给出的命令字表。43~110 行定义了 MFRC522 阅读器的全部寄存器地址，参考 MFRC522 Contactless Reader IC Product Data Sheet 的第 9 节，其中需要关心的是命令寄存器 CommandReg，它的地址为 0x01，另一个是缓冲区寄存器 FIFODataReg，它的地址是 0x09。

RFID.h：112~139 行为类定义，以下我们观察 RFID 类中做了哪些定义。对于卡片来讲，卡号是其唯一标识，开辟一个 `serNum[]`来保存卡号，长度为 5，其中 4 字节为卡号，1 字节为校验字节。同时定义阅读器连接管脚`_chipSelectPin`，`_NRSTPD`，这两个管脚需要在创建对象时告知其与 Arduino 的管脚的对应关系。这里可以注意到 `serNum[]`为一个 `public`

成员，原因是卡号是需要给暴露给上层应用函数来使用的，因为后续其他类或结构使用serNum[]的频率非常高，为了方便将其封在类外。

RFID 构造函数负责建立管脚的对应关系。由于私有成员为管脚，不需要特意设置析构函数。以下我们来观察其他成员函数的定义，具体函数的实现我们会在 RFID.cpp 中讲解。

为了方便教学，我们有意的将 setBitMask ()，clearBitMask ()，writeMFRC522()，readMFRC522()这 4 个函数设为内联，一方面是这 4 个函数的使用频率非常高，更重要的是 writeMFRC522()，readMFRC522()这一对函数专门负责向阅读器的寄存器中写值，因为此前我们介绍过，控制阅读器的方法即通过向阅读器的寄存器中写入固定格式的值来完成。几乎所有的函数都会使用它们。另外，antennaOn()，antennaOff()两个函数直接控制阅读器开天线、关天线，无需深究；init()函数用于初始化阅读器各个寄存器的状态，在阅读器第一次上电时进行；reset()函数则直接传送 PCD_RESETPHASE 命令字复位阅读器。

其他的成员函数则全部根据与卡片的工作流程进行定义：isCard()，readCardSerial()，selectTag()，anticoll()，calculateCRC()，auth()，read()，write()，halt()。这些函数与卡片工作流程全完对应，其中，这些函数都会有一些共同的操作，如写入阅读器命令字，组织传送数据，轮询传输传输完成（中断），所以设置 MFRC522ToCard()函数来完成这些共通性的操作，供以上几个工作流程函数使用。

二、*RFID.cpp*

- RFID(int chipSelectPin, int NRSTPD)

  如上文所述，构造函数负责建立 Arduino 与 MFRC522 阅读器之间的对应关系，在校园一卡通系统的硬件连接中，阅读器通过 SPI 接口与主机连接，其中阅读器的 NSS 与 Arduino 10 数字口相连接，RST 与 Arduino 5 数字口相连接，所以在创建对象时，需要：

  RFID(10, 5);

  建立好管脚的对应关系，这两个管脚如果位置发生变化，则注意修正参数，否则阅读器与 Arduino 之间无法建立通信。

- inline void writeMFRC522(unsigned char addr, unsigned char val)

  writeMFRC522()及 readMFRC522()可以说是类中完成操控阅读器最实质的函数，它的作用是直接向阅读器的某个寄存器写入数据（值）。参数 addr 表示要写入寄

存器的地址，这些地址在 *RFID.h* 中已经定义；参数 val 表示写入的值，这个值一般都是某个命令字，同样也已经定义。观察一下它的实现（*RFID.cpp* ：103~110）：

```
void RFID::writeMFRC522(unsigned char addr, unsigned char val)
{
  digitalWrite(_chipSelectPin, LOW);
  SPI.transfer((addr<<1)&0x7E);
  SPI.transfer(val);
  digitalWrite(_chipSelectPin, HIGH);
}
```

可以看出，能够通过 Arduino 控制阅读器其实是通过 SPI 库函数 `transfer()`,它的参数为地址及值，其中地址格式必须为：0XXXXXX0 的形式。由于地址与命令字的格式并不一致所以要做左移及掩码。使用 `transfer()`的的方法是首先传入地址，然后再传入值。在 SPI 进行通信时，需要使用 Arduino 库将`_chipSelectPin` 管脚电平拉低。对于 `readMFRC522()`原理相同则不再赘述。

- `void init()`

`init()`用于第一次上电时初始化阅读器，所做的所有操作全部都依照参考资料 III 中的定义。需要注意 `init()`函数最后会打开阅读器天线。

- `void reset()`

复位函数会向阅读器发送复位指令，我们观察一下它的用法（*RFID.cpp* ：92~95）：

```
void RFID::reset()
{
  writeMFRC522(CommandReg, PCD_RESETPHASE);
}
```

可以看出，这个函数很简单，它会直接调用 `writeMFRC522()`向阅读器写入命令，这里传入命令为复位命令 PCD_RESETPHASE，它被发至命令寄存器 CommandReg，命令寄存器接到命令后会开始执行复位指令。通过这个例子也可以看到阅读器的操控用法，即控制阅读器的方法就是通过 `writeMFRC522()`发送阅读器命令字。

- `bool isCard()`

`isCard()`函数对应卡片工作流程的第一步：寻卡。寻卡的作用是阅读器发现天线区内是否有卡片的存在。（*RFID.cpp* ：30~40）可以看到 `isCard()`函数并没有参

数，返回 bool，在本函数中其中并没有直接向阅读器发送寻卡命令字，整个函数也只有寥寥几行。这样设计有两个原因：第一，寻卡的操作不是阅读器自身能够完成的，还需要卡片的配合，或者可以说，寻卡操作要求向阅读器发送阅读器的寻卡命令字，而阅读器也必须向卡片发送卡片的寻卡命令字；第二，这样的操作在所有的卡片流程中都会出现，可以把他统一为一个操作供所有的卡片流程函数使用。所以 isCard()实质是调用 MFRC522Request()，而自身只判断调用是否成功执行。

- unsigned char MFRC522Request(unsigned char reqMode, unsigned char *TagType)

  MFRC522Request()被 isCard()调用，完成实质的寻卡操作。我们通过以下一个描述来感性认识这个过程。首先，阅读器必须知道要做的工作为寻卡，并且为寻卡结果做好准备，因为根据卡片工作流程，寻卡的结果应该至少返回卡片是否在天线区存在及卡片类型这两个信息，所以会设置 status；其次，寻卡的过程需要阅读器完成发送数据给卡片这一客观存在的数据传输，这个过程包括发送阅读器以命令字及通过阅读器发送卡片命令字，并且，还必须要做好发送数据及接受数据的工作。对于第一点，由 MFRC522Request()自身来准备，并负责处理结果；对于第二点，MFRC522Request()通过调用 MFRC522ToCard()来完成。通过名字直接可以看出，MFRC522ToCard()是真正完成阅读器与卡片之间数据传输的函数，这个函数也是整个 RFID 类的核心。

- unsigned char MFRC522ToCard(unsigned char command, unsigned char *sendData, unsigned char sendLen, unsigned char *backData, unsigned int *backLen)

  MFRC522ToCard()函数可以认为是学习读写机制的重头戏。

  首先我们考虑一下它是如何设计的，根据上文，卡片工作流程中都会设计一个共同的步骤，即通过阅读器向卡片发送命令或数据，那么由于流程不同，阅读器的命令字不同，进而卡片命令字也可能不同，例如寻卡，阅读器与卡片都有各自对应的命令字，而对于验证，阅读器与卡片也有各自对应的命令，而且卡片命令字必须经由阅读器发送，所以我们准备设计一个通用的，能够在阅读器与卡片之间传送各种不同命令及数据的函数，这就是 MFRC522ToCard()的设计意图。

  观察它的参数：

command：这里填入阅读器的命令字，指导阅读器的动作；可以看到函数中定义了可能出现的两种阅读器动作：PCD_AUTHENT，PCD_TRANSCEIVE，其中的PCD_TRANSCEIVE 的功能相当于读与写。

sendData：所发送的数据，我们让它指向要发送数据的地址，从后文可以看到，sendData 总是指向一个长度为 16 字节的字节数组。所发送的数据无论是什么都放入 sendData 的位置，那么卡片的命令字呢？它被要求放在数组的第一个字节。观察之前的 MFRC522Request()函数，它在调用 MFRC522ToCard()时将 sendData 指向了 TagType 数组，并将第一字节的位置填入了卡片命令字：（*RFID.cpp* ：324~325）

```
TagType[0] = reqMode;
status = MFRC522ToCard(PCD_TRANSCEIVE, TagType, 1, TagType,
&backBits);
```

其实在 MFRC522Request()中 reqMode 与 TagType 仍然为形参，那么我们再向上观察其调用者 isCard()：（*RFID.cpp* ：35）

```
status = MFRC522Request(PICC_REQIDL, str);
```

终于看到，卡片的寻卡命令字 PICC_REQIDL 及要传输的数据 str[]早早就由 isCard()填好，最后层层传入 MFRC522ToCard()。

sendLen：阅读器向卡片发送的数据长度。要发送的 16 字节数据存放在哪里呢？回想在 2.2 节图 2-10 我们介绍了在阅读器中有一个 FIFO 缓冲区，数据是这样进入缓冲区的：（*RFID.cpp* ：255~256）

```
for (i=0; i<sendLen; i++)
    writeMFRC522(FIFODataReg, sendData[i]);
```

这里的 FIFODataReg 是缓冲区寄存器，sendData 实际指向的是 isCard()中的 str[]，数据就是这样进入了阅读器的缓冲区，一旦填装完毕，阅读器一声令下，它就会通过无线信道送至卡片。

backData：与 sendData 相同，它也指向一个数组，用于接收返回数据，这个数组是由 MFRC522Request()准备的，细心的同学会发现它仍指向了 str[]。通过这一点，可以看出，我们在发送前将数据准备在一个 str[]中，然后交由 isCard()处理，传输完毕后，卡片返回的结果也被保存在 str[]中。

backLen:返回数据的长度，这个值一般用于观察传输中是否出现错误。但注意这它是一个指针，指向存放长度值的位置。

理解了参数的作用，MFRC522ToCard()的功能基本就确定了。但是还有一个问题没有明朗：阅读器与卡片之间数据传送的过程是怎么开始怎么结束的？

判定开始很简单，观察（*RFID.cpp* ：258~260），函数向阅读器发送了 PCD_AUTHENT，PCD_TRANSCEIVE 命令字开启了数据的传输。

而判定结束就没那么容易了，阅读器如何知道卡片接受到了数据并且返回了应答？在计算机科学中，这类问题都可以归结为一种称为同步的问题。这类问题的核心在于，阅读器是永远无法知晓卡片侧发生了什么，任何事物都没有上帝视角。所以我们采取的方法是：等待。

阅读器等待某一个自身信号来告诉它：卡片的数据已经回来了。这个信号如果没有出现，阅读器就一直等待下去，直到超时。这种过程我们称之为：中断。

所以我们在函数中会看到一段忙等代码：（*RFID.cpp* ： 262~271）

```
i = 2000;
do{
    n = readMFRC522(CommIrqReg);
    i--;
} while ((i!=0) && !(n&0x01) && !(n&waitIRq));
```

首先设置一个计时器，它会在循环中倒计时至结束，那么说明出现超时，计时器是根据 Arduino 的处理器速度与协议进行权衡得到的。在这一段时间里，它会不停的轮询 CommIrqReg 的值，同学们可以参照参考资料 III 得知这个寄存器为中断寄存器。当这个寄存器中某些位被置 1，表明某一类中断发生。有一类中断则为收到数据后产生的中断，这也是我们希望看到的，这就是 waitIRq 的作用。

在 273 行我们看到：

```
if (i != 0){
//…
}
```

请务必养成这个习惯，跳出中断后再次判断一下是否发生的是期望的中断，而不是倒计时结束（那是我们不希望看到的，说明超时了）。

- bool readCardSerial()

readCardSerial()的功能是获得卡片的卡号，然后这个卡号会被复制到 serNum[]中（类中的公有成员），供上层函数使用。读卡号的过程其实也是借助另一个函数 anticoll()完成的。

- unsigned char anticoll(unsigned char *serNum)

  anticoll()调用了 MFRC522ToCard()，完成了真正的数据传输，获得了卡号，这个过程不再赘述。

- unsigned char selectTag(unsigned char *serNum)

  通过传入 PICC_SElECTTAG 卡片命令字，selectTag()完成了选卡。注意参数 serNum,要求给出卡片的序列号。所以 selectTag()一定是在 readCardSerial()成功获取卡号后才能进行的：readCardSerial()成功执行，获取卡号保存在成员 serNum[]中，执行 selectTag()时将 serNum[]传给 serNum。

- unsigned char auth(unsigned char authMode, unsigned char BlockAddr, unsigned char *Sectorkey, unsigned char *serNum)

  这里需要注意的是，认证操作不能被简单认为是对密钥块的“读”“写”，认证是单独的一个命令字。认证的过程需要传入的密钥被定义在 *SectorKey.h* 中（*SectorKey.h*：13~20）。这个文件中保存的密钥为出厂密钥，可以直接读写出厂卡。作为教学，我们尽量避免随意定义密码，一旦定义的密钥被忘掉了，被加密的扇区中的数据永远无法再使用，卡片就损毁了，因为 Mifare 卡片是无法暴力破解的。

  所以，要求学生在征得老师同意下才可以自行增加 *SectorKey.h* 中的密钥并使用。

- unsigned char read(unsigned char blockAddr, unsigned char *recvData)

  通过 MFRC522ToCard()传入 PICC_READ 命令字，卡片完成读操作。

- unsigned char write(unsigned char blockAddr, unsigned char *writeData)

  通过 MFRC522ToCard()传入 PICC_WRITE 命令字，卡片完成写操作。

最后提示一点，本教材中没有介绍如何进行增值、减值操作，校园一卡通系统的实现是通过将数值（金额）作为字节流来统一处理的。有兴趣的同学可以参照相关文献自行考虑 INCREMENT/DECREMENT 的实现。

2.7 RFID 类的使用

以下我们通过一个最精简的例子来演示如何使用 RFID 类来完成卡片的读写及验证操作。我们必须根据卡片的工作流程来一步步使用 RFID 类中的函数，但在这之前，我们还需做一些列准备工作：1. 建立阅读器与 Arduino 的连接关系；2. 初始化阅读器。所以我们每次进行一次卡片的读/写操作，都可以总结为以下流程：

一、告知 Arduino 与阅读器的连接关系，这一点通过构造函数完成。

二、调用初始化函数使阅读器完成初始化并打开天线。

三、按照卡片工作流程依次完成：寻卡、读卡号、选卡、验证读写块密钥。

四、根据需求进行块读写。

五、读写完毕，停止卡片，准备下一次的寻卡操作。

我们通过以下例子来演示这一系列过程。这个例子演示了读取两个块 INDEXBLOCK 与 BALANCEBLOCK 的方式。

```
#include <RFID.h>
#include <SectorKey.h>
#define SECTORKEY    7
#define INDEXBLOCK     4
#define BALANCEBLOCK  5

RFID rfid(10, 5);

rfid.init();
rfid.isCard();
rfid.readCardSerial();
rfid.selectTag(rfid.serNum);
if (rfid.auth(PICC_AUTHENT1A, SECTORKEY, sectorKeyA[1], rfid.serNum) ==
MI_OK) {
  rfid.read(INDEXBLOCK, indexArray);
  rfid.read(BALANCEBLOCK, balanceArray);
}
rfid.halt();
```

实训项目

2-A 习题

1. 2.1 节中介绍一张空卡在出厂时密钥块的默认状态为：0xFFFFFFFFFFFF FF078069 FFFFFFFFFFFF，那么以此密钥读出此密钥块的密码 A 部分，读出的数据是什么？权限位设置为：FF078069，本扇区 3 个数据块及 1 个密钥块分别拥有表示怎样的权限？

2. 请查阅附录 B 中 *SectorKey.h* 文件，解释 sectorKeyA[][]的涵义。

3. 卡片的命令字 *Authentication* 与阅读器的命令字 *MFAuthent* 有什么区别？

4. 在第一部分的实训 1-B 中，我们给出了使用菜单 Consumption 对卡片进行充值的操作方法，请描述数据是如何从 Arduino IDE 监视器的输入框传入卡片扇区的。

2-B RFID.cpp

1. 阅读附录 B RFID.cpp 源代码，并画出函数调用关系图。

2. 请解释 `write()`函数与 `writeMFRC522()`函数各自的功能是什么。

3. 根据参考资料 MFRC522 Data Sheet，解释附录 B *RFID.cpp* 文件第 262 行~第 271 行涵义。

4. 请解释寻卡过程中，`isCard()`函数中 `str[]`的作用；描述寻卡的卡片的命令字 `PICC_REQIDL` 是如何到达卡片的。

2-C 程序设计

1. 根据本章内容，请设计两个项目：

项目一：

根据卡片的工作流程及 RFID 类，编写一个运行在 Arduino 上的程序，完成以下功能。

在卡片的第 3 号扇区内，存入项目组所有人的姓名拼音缩写（大写字母），每一个成员占用一个数据块，例如：

成员：韩少男　　拼音缩写：HSN　　数据存放：0 号数据块

成员：…　　拼音缩写：…　　数据存放：1 号数据块

项目二

根据卡片的工作流程及 RFID 类，编写一个运行在 Arduino 上的程序，完成以下功能。

设计一个程序，能够在 Arduino IDE 串口监视器上显示项目一所存入的数据，例如：

刷卡，监视器显示：

****
No.3 Sector
Data from block 0~2 is:
HSN
...
...
****

2. 控制摇杆 JoyStick 实验

按照第一部分实训项目 1-A 中的说明将摇杆元件连接到 Arduino 上，打开 Arduino IDE，输入以下程序，编译并下载代码至 Arduino UNO，启动串口监视器。

摆动摇杆，观察 X 轴 Y 轴的数值变化。

```
int value = 0;
void setup() {
    Serial.begin(9600);
}
void loop() {
    value = analogRead(0);
    Serial.print("X:");
    Serial.print(value, DEC);
    value = analogRead(1);
    Serial.print(" | Y:");
    Serial.print(value, DEC);
    delay(100);
}
```

参考资料

I ISO14443

10373-6(Test methods).pdf，14443-1(Physical characteristics).pdf，14443-2(Radio frequency power and signal interface).pdf，14443-3(Initialization and anticollision).pdf，14443-4(Transmission protocol).pdf

II Mifare Classic S50 Product Data Sheet

MF1S50YYX_V1.pdf

AN10833.pdf

III MFRC522 Contactless Reader IC Product Data Sheet

RC522.pdf

IV Arduino libraries: EEPROM

The microcontroller on the Arduino AVR based board has EEPROM: memory whose values are kept when the board is turned off (like a tiny hard drive). This library enables you to read and write those bytes.
The supported micro-controllers on the various Arduino boards have different amounts of EEPROM: 1024 bytes on the ATmega328, 512 bytes on the ATmega168 and ATmega8, 4 KB (4096 bytes) on the ATmega1280 and ATmega2560.

Functions

```
read()
write()
update()
get()
put()
```

Examples: read()

Description
Reads a byte from the EEPROM. Locations that have never been written to have the value of 255.

Syntax
`EEPROM.read(address)`

Parameters

address: the location to read from, starting from 0 (int)

Returns
the value stored in that location (byte)

```
#include <EEPROM.h>
int a = 0;
int value;
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  value = EEPROM.read(a);
  Serial.print(a);
  Serial.print("\t");
  Serial.print(value);
  Serial.println();
  a = a + 1;
  if (a == 512)
    a = 0;
  delay(500);
}
```

Examples: write()

Description
Write a byte to the EEPROM.

Syntax
`EEPROM.write(address, value)`

Parameters
address: the location to write to, starting from 0 (int)
value: the value to write, from 0 to 255 (byte)

Returns
none

Note
An EEPROM write takes 3.3 ms to complete. The EEPROM memory has a specified life of 100,000 write/erase cycles, so you may need to be careful about how often you write to it.

```
#include <EEPROM.h>
void setup()
{
  for (int i = 0; i < 255; i++)
    EEPROM.write(i, i);
}
void loop()
{
}
```

第三部分 开发框架的设计与实现

在第二部分的内容中，我们介绍了如何利用商业化的卡片、阅读器及主机平台完成ISO/IEC 14443 协议的实现 RFID 的完整访问流程。在本章节，我们讨论如何设计并搭建一套整合卡片管理功能的系统框架。我们对于整套系统框架的有两个设计原则：

第一，系统必须具有非常强的复用性以满足不同的 RFID 应用场景。也就是说这套框架不是仅仅为了校园一卡通这一种应用场景来设计，还应该可以满足公交卡系统、地铁计费卡系统、门禁系统等不同场景的需要。所以，要求系统中的各个模块具有清晰的接口，尽可能全面的功能，以及简明易用的整合方法以提高整个框架的复用性。

第二，系统必须较少的引入其他商业设施，以方便教学。在第二部分我们介绍了 Mifare 卡片、MFRC522 阅读器，这是 RFID 不可缺少的设施，除此之外，我们在希望构建系统时尽量少的引入其他的内容，从而将教学核心放在 RFID 技术本身。所以，系统框架中的控制主机的功能，例如数据存储，仅考虑在 Arduino UNO 上完成，同时，用户控制也仅考虑使用 JoyStick 的四个方向作为按键，从而完全脱离对开发端 PC 机的依赖。也就是说，我们希望系统可以在接通一个 5V 电源的情况下即开始正常工作，PC 机仅仅起到一个串口输出监视器的作用。

学习内容

3.1 系统需求

在第一部分中，我们以校园一卡通管理系统为例演示了一套简单的 RFID 系统正常运行的方法。但是，对于学习 RFID 系统来讲，不能够仅考虑校园一卡通这一种应用场景。我们希望能够建立起一套适用于各种场景的系统框架。所以，在考虑系统需求的时候，总是从通用性的角度进行考量。

对于一套 RFID 管理系统框架来讲，管理功能可以分为三大部分：

- 对卡片的管理；
- 对数据的管理；
- 对用户交互控制的设计。

卡片的管理在这里定义为：如何让管理系统能够知晓某张卡片是属于自己的管理范畴，并且支持它的一些特定的读写要求。那么首先就是无论什么样的应用场景，一定会有注册卡

片与注销卡片这两个功能。数据的管理是系统框架的核心，有什么样的数据描述决定了操作的方式。所以，如何保存系统中的卡片数据？如何访问、修改这些数据？这必须要求我们考虑两点：1. 数据存储的位置；2.数据的存储结构。也就是说，设计一种记录格式来保存系统中卡片数据是数据管理的核心。然后，根据数据存储结构的特性，我们必须设计一套支持数据管理与访问的接口。这两个问题我们将通过 3.4 节进行详细介绍。最后，我们希望系统具有一种简洁的交互方式，利用 JoyStick 的特性，直接控制系统。

3.2 功能定义

根据需求确定系统设计，以下我们对 3.1 节中讨论的三部分功能做如图 3-1 的定义。

	功能	描述
卡片管理	卡片注册	在系统内记录一张卡片（卡号）
	卡片注销	将系统内已有的一张卡片（卡号）从系统中删除
	（根据应用场景定义的其他功能）	-
数据管理	在系统中创建一条记录	将卡片（卡号）与记录进行绑定，填充所有初始化字段
	读取系统中某条记录	读取某条记录的所有字段
	修改系统中某条记录	修改某条检录的所有字段
	删除系统中某条记录	解除卡片与记录的绑定关系，使记录无效化
	清空所有记录	解除所有记录的绑定关系
	统计记录条目	统计有效记录的数量
	打印记录	打印所有记录
	清空数据库	清空数据存储区中的一切数据
交互控制	JoyStick 方向键	使系统得到一次 JoyStick 键入，并返回确定方向
	系统菜单打印	打印统一格式的系统菜单

图 3-1 功能定义

3.3 数据存储结构

项目设计首先要考虑数据，数据的存储结构设计是系统框架的最核心问题。讨论这一个问题要权衡诸多的内容，所以设计的出发点应该基于两个基本原则：第一，系统必须具有非常强的复用性以满足不同的 RFID 应用场景。第二，必须较少的使用其他技术确保系统简洁。

一、系统数据与卡片数据

哪些数据要存放在系统中，哪些数据应该存放于卡片自身之中，这是首先要确定的问题。例如考虑一种场景，一张饭卡，饭卡的余额信息在每次刷卡的时候都能够显示。这个金额信

息是保存在刷卡的阅读器一端的整套系统中，还是保存在卡片自身呢？从技术上来说，让金额保存在阅读器中是可以的，这样就必须要求整套 RFID 系统的所有阅读器实时联网，来获取这一信息；如果将金额存储于卡片自身中，那么就必须要求卡片本身具有加密能力；如果同时存储于系统及卡片上，那么就要维护系统端与卡片的数据一致性。在这里我们给出结论：通常，将金额信息仅保存在卡片上，由加密系统保证这一金额不被随意改动。

对于一些初始化状态的信息，例如卡片基本信息，例如发卡单位，卡片状态等等，则需要保存在系统中。还有一些不经常改动的信息，例如持卡人信息等，则既可以保存在卡片中，也可以保存在系统中。

最后，我们考虑如何在卡片与系统之间建立绑定关系，就是让系统能够知道某张卡片是属于本系统的。由于卡片具有一个唯一属性----卡号，那么我们就利用卡号让系统来识别并绑定卡片，即卡片注册操作。我们让每一条记录都维护一个卡号字段，以表示当前记录所对应的卡片。

二、数据的存放位置

为了尽可能不适用其他的存储技术的情况下，我们采用 Arduino UNO 自身的一块 EEPROM 作为数据存储区域，它的大小为 512B，本文后续简称这一区域为数据库。利用这一块 EEPROM 的是处于这样的考量： EEPROM 是 Arduino UNO 上唯一的非易失存储空间，如果采用其他存储位置，则势必要借助 PC 机，那么我们就要遭遇操作系统调用与文件系统这两个话题，这无疑增加了课程难度。利用这 512 字节的存储空间，通过合理设计每一条记录的存储结构，可以保存若干条卡片信息。

EEPROM 空间被 Arduino 划分为块为单位进行读写，每一块的大小为 1 字节，块号从 0 开始编址，即块 0，块 1，…，块 511，这样我们可以方便对存储空间中的任意一字节进行读写。Arduino 为程序员提供了一个名为 EEPROM 的库来完成读写操作，它通过块号进行访问，这个库被定义在 Arduino Library 的 *EEPROM.h*，*EEPROM.cpp* 中，所以在项目中我们要引入这两个文件。库中所提供的操作方法已经在 2.3 节介绍过，同时我们给出了操作范例，详见第二部分的参考资料 IV：Arduino libraries:EEPROM。

三、数据记录的结构设计

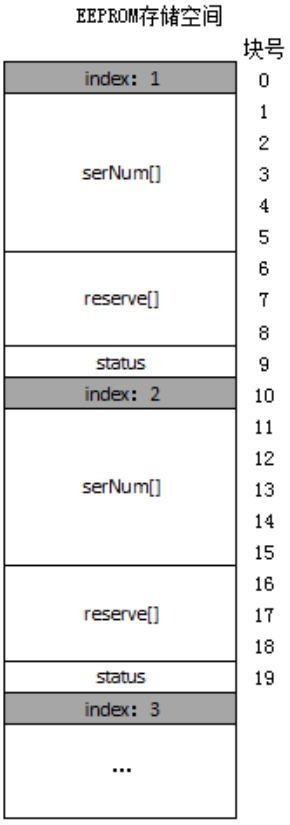
- 系统记录结构

利用 EEPROM 保存系统记录，系统记录表如图 3-2 所示。这里有必要解释 index 字段的设计：首先，index 字段用于记录的索引，虽然卡号也具有唯一性，但是如果利用 5 字节的卡号进行索引，效率较为低下；第二，index 字段同时兼具记录有效性标识，我们要求 index 编号一定是一个大于 0 的数，表示“第 XX 条记录”，一旦 index 值为 0，则表示此条记录已经从数据库中删去，所以在清除记录时，我们可以通过将该记录的 index 置为 0 来完成。

字段	类型	大小	描述
index	unsigned char	1	记录索引号
serNum	unsigned char[5]	5	卡号
reserve	unsigned char[3]	3	保留字段，由框架使用者自行定义。三个字节可以整体使用，也可部分使用，用于对不同应用场景提供数据存储
status	unsigned char	1	卡片的状态信息，由框架使用者自行定义

图 3-2 系统记录表

根据图 3-2，可以看到每一条记录要求 10 字节存储空间，一块 EEPROM 有 512 字节，则可以最多存放 51 条记录，即一个具有 51 张卡片的系统，这对于一个教学系统来讲够用了。这些记录按照 EEPROM 块号的顺序分布，在每一条记录内部，同样按照图 3-2 的由上到下顺序排列各个字段，其中，index 作为索引号，处于记录块的开始位置，如右图所示。



- 卡片记录结构

卡片需要保存的数据由应用场景而定，在这里，我们以校园一卡通系统为例，需要在卡片中保存金额数据，即电子钱包功能；同时，在卡片中我们保存一个系统中定义的 index 号码，这样则方便我们在日后的实现中简化查找记录操作。

那么在卡片中我们可以使用两个数据块来分别保存索引号及金额，比如定义这使用第二扇区的第 4、5 号数据块分别为保存金额与索引号，这些完全可以由程序员自行决定。

为了方便使用，我们可以在程序文件中添加如下内容：

```
#define    INDEXBLOCK     4
#define    BALANCEBLOCK   5
```

四、数据存储结构的实现

我们可以按照以下方式建立存储结构，这一结构定义在 Storage.h 文件中（*Storage.h*：12~15）：

```
typedef struct Data {
    unsigned char index;
    unsigned char serNum[5];
    unsigned char reserve[3];
    unsigned char status;
}Data;
```

那么，系统需要向 EEPROM 中写入数据时，可以直接装入以上结构。考虑到整体的设计及初始化等相关问题，我们一不作二不休，进一步为数据添加基本初始化操作，如下（*Storage.h*：11~30）：

```
typedef struct Data {
    unsigned char index;
    unsigned char serNum[5];
    unsigned char reserve[3];
    unsigned char status;
    Data(){
        index = 255;
        for(int i = 0; i < 5; i++)   serNum[i] = 0;
        for(int i = 0; i < 3; i++)   reserve[i] = 0;
        status = 0;
    }
    Data(unsigned char num):index(num){
        for(int i = 0; i < 5; i++)   serNum[i] = 0;
        for(int i = 0; i < 3; i++)   reserve[i] = 0;
        status = 0;
    }
    void setIndex(unsigned char num){
        index = num;
    }
```

```
}Data;
```

这里重载构造函数 Data(unsigned char num)的目的是为了方便对一个已存在系统中的记录进行修改、删除等操作。

五、数据存储结构的使用

为什么我们要设计这样的一个结构，而不将它直接封装成类呢？这样的考量有一个深层原因：Data 是连接卡片数据与系统数据连接的桥梁。不将它设计为类是为了将它的所有成员暴露在外，便于卡片操作函数使用。假设有一条金额数据从卡片发到系统，此时，我们创建一个 Data 对象来接收这一数据，可以方便的直接使用暴露的成员，随后系统可以通过 Data 对象向 EEPROM 中进行相关操作。另一方面，将所有数据封装在结构中，对于参数传递具有重要的意义，这大大简化了系统数据管理函数的接口设计，我们直接将 Data 的引用传递给所有系统数据操作函数，每一个函数可以自行考虑使用哪一字段而没有任何的传参开销，在后续章节可以看到，我们是通过用存储操作类组合 Data 对象的引用来实现的。

可以这样做一个比喻：在卡片与系统 EEPROM 两片数据存储区域之间有一条鸿沟，它们之间每次进行数据交互时，都会使用 Data 搭建一座桥梁，在数据交互完毕后桥梁 Data 自行消亡。

3.4 数据管理功能的实现

在 *Storage.h* 中我们将实现对数据存储的所有操作，通过定义类：STORAGE。我们设想的 STORAGE 类应该具有这样的特性：1、它负责一切保存关于 Data 的数据到 EEPROM 中；2、为了简明、复用，所有函数尽可能少的传递参数。所以我们设置成员：

```
unsigned char ptr;
Data& data;
```

ptr 的作用看做一个指向 EEPROM 记录块首的指针，即 ptr * 10 总是指向 index 块所在的位置，这样，由于 EEPROM 中所有记录块的结构相同，在访问特定的记录块内地址时，可以采用

ptr * 10 + 块内偏移量 （0 ≤ ptr ≤ 50）

这样的固定公式来完成。看以看出 index 与 ptr 之间始终保持着

index = ptr + 1

的数学关系。

data 作为对 Data 类的引用，将要处理的数据传给操作者 STORAGE 类。

- void ptr2Zero()

  用于将 ptr 复位。因为我们的设想是：采用从存储空间起始位置开始的顺序查找法。对于存储空间很小的 EEPROM 来讲，这种方法虽然效率不高，但实现最为简单。

- bool read()

  将 data 的内容根据记录块进行更新。

- bool write()

  将记录块的内容根据 data 进行更新。

- bool writeIndex()

  write()只能更新除了 index 之外的所有字段，所以我们设计一个可以更新 index 字段的 writeIndex()作为补充。

- unsigned char create()

  create()函数在数据块中查找一个空闲记录块，对记录块做初始化工作，建立卡片与记录的绑定关系，并返回索引号。这个函数主要用于卡片注册。考虑这样一种情况：设卡片 A 已在系统内注册并开始正常消费使用，是否允许卡片 A 再次进行注册？答案是不允许，那么在 create()中我们必须做对卡号的查重监测，即不允许出现一张卡片创建两次以上的记录块，保证卡片与记录的一一对应，我们通过这样的方法完成对卡号的查重（*Storage.h*：80~96）：

```
for (; tempPtr <= 50; tempPtr++){
    if (!EEPROM.read(tempPtr * 10)) continue;
    else {
        for(int i = 0; i < 5; i++)
            validate[i] = EEPROM.read(tempPtr * 10 + 1 + i);
        unsigned char* p = data.serNum;
        unsigned char* q = validate;
        unsigned char counter = 0;
        for (; counter <= 5 && *p++ == *q++; counter++);
        if (counter == 5) break;
    }
}
```

- `static unsigned char countDataBase()`：统计数据库 EEPROM 中有效记录条数。
- `static void printDataBase()`：从 0~511 逐块打印数据。
- `static void flushDataBase()`：清空数据库，将所有数据块置 0。

3.5 金额处理

由于在 RFID 类中，我们没有实现 ISO/IEC 14443 中的 Increment 与 Decrement 这两种操作，所以我们不得不将金额信息存放在普通的数据块中。在数据块中，数据被逐字节识别，这对消费与充值操作这样加减运算造成了一定的麻烦，我们必须自行实现字符到数值的转换。这种转换涉及这样几个问题：

1. 来自 PC 机串口监视器输入的字符串必须转换为数值；
2. 来自卡片的字节数据必须转换为数值；
3. 经过加减运算的数值必须转换回字节数据发送给卡片。

在 Money 类中，我们实现了三个函数来完成这样的转换（*Money.h*：16~36）：

- `double atof(unsigned char s[])`

  `atof()`接收来自串口监视器的字符串，这个字符串包含有效的数值信息，允许其达到 4 字节浮点数的精度，返回这个数值。根据 Arduino UNO 处理器的规定，double 类型的长度为 4 字节，按照浮点数在主机的表示方式计算可知，其取值范围为：-3.4E38 ~ 3.4E38，这样，我们定义的缓存字符串的长度就不能够超过 41 位。这里还需考虑到的一个问题就是 Arduino UNO 串口缓冲区的最大字节数是否能够满足 41 字节，根据 www.arduino.cc 中的文档可知，Arduino UNO 串口缓冲区为 64 字节。

  那么，我们在函数中开辟一个 `unsigned char serialBuffer[]`，长度为 34，作为串口缓存是合适的。（其实，对于金额来讲，货币单位为圆角分，最多精确到小数点后 2 位，34 位过长了，这是笔者将项目实现完才想到的）

  在 `atof()`，我们使用技术手段保证了输入的金额仅允许数字 0~9、负号、小数点这三种字符，一旦出现其他字符，则直接截断，使后续字符无效。为此，我们用一个宏来完成是否为数字的判别：

  ```
  #define isdigit(char) ((char) >= '0' && (char) <= '9')
  ```

- `inline double byte2Double(unsigned char s[])`

byte2Double()将来自卡片的 16 字节数据进行截取，仅保存前 4 位，将其从字节型转换为数值型：

```
inline double byte2Double(unsigned char s[]){
    unsigned char c[] = {s[0], s[1], s[2], s[3]};
    return *((double*)c);
}
```

- inline unsigned char double2Byte(double d, int index)

  与上一个函数类似，double2Byte()用于在数值向卡片发送之前，将其从数值型转换为字节型：

```
inline unsigned char double2Byte(double d, int index){
    return ((unsigned char*)&d)[index];
}
```

这种数值的转换并不完美，因为他并不是返回一个字符串，而是，允许将数值拆分为 4 个字符来对待。为此，我们设计另一个函数 getResultBuffer()来做组合操作，返回一个装着“数值的字符串”（*Money.h*：60~66）：

```
unsigned char* getResultBuffer(){
    unsigned char tempbuffer[4];
    unsigned char* p = tempbuffer;
    for (int i = 0; i < 4; i++)
        tempbuffer[i] = double2Byte(balance, i);
    return p;
}
```

有了这些技术上的准备，我们可以着手设计 MONEY 类了，观察类的成员，其中 cardBuffer[]用于保存卡片数值块的数据，只保留其前 4 字节；serialBuffer[]用于保存串口监视器输入的字符串，而 balance 则作为真正做加减运算的金额数值：

```
unsigned char cardBuffer[4];
unsigned char serialBuffer[34];
double balance;
```

对金额数值可以完成取值、加值、减值运算，对于减值运算，我们没有做异常处理，例如消费金额超过卡片余额应该不允许进行减值，否则会出现 balance 小于 0 的情况，但这些我们认为应该由上层业务来进行保证。

3.6 菜单的设计与实现

在这里我们考虑对菜单的设计，通过校园一卡通管理系统的展示可以看出，所有的界面最多显示 4 个功能，分别对应控制摇杆的 4 个方向。我们希望系统能有统一的界面，所以我们将所有的界面用长度为 40 个字符的“ *”装饰在界面的最上与最下，其中最上行中要在中央位置放置功能名的大写英文。

我们创建 CreateUI 类，它的功能分别为打印最上行装饰、打印最下行装饰、打印其中的菜单这三项，分别对应以下函数：

- `static void titleLine(String s)`

  打印一行带功能名的装饰行，参数为显示在最上杭装饰的功能名字符串。

- `static void endLine()`

  直接打印一行装饰行。

- `static void menu(String up, String left, String right, String down)`

输出 4 行对应 4 个参数的方向键的菜单，参数为 4 个方向功能键的功能名字符串。其中最上杭装饰中“ *”的个数需要根据功能名字母个数的不同而自行调整。

这里必须提及的一点是，将打印菜单这个操作封装为一个类进行操作并不是仅仅为了方便，而是因为一个非常重要的问题：内存。通过 Arduino UNO 参数可知其只有 2KB 内存空间，如果不使用 CreateUI 类统一打印而是频繁调用 `Serial.println()`直接打印 40 个“ *”，那么运行进程的全局静态区会被大量的“ *”这样的字面常量占满，从而导致运行的不稳定，这才是创建 CreateUI 类的原因，如果使用一块 Arduino MEGA 来完成本框架，则不需要定义 CreateUI 类。

3.7 用户交互功能的实现

用户交互是本系统的另一个重要的话题。根据 2.4 节的介绍及实训 2-C 可以观察到这样一个事实：JoyStick 能做仅仅是在 2 个模拟管脚给 Arduino 一个信号。这与对交互功能来说远远不够。所以我们需要专门定义一个类：OPTION，来对这两个模拟管脚适当的处理。这个类必须具有以下特性：

- 能够得知 JoyStick 输入值所对应的方向；
- 能够将用户输入过的值保存下来，提供给系统使用；
- 能够使系统需要用户输入时停下来等待用户摇动 JoyStick。

这里需要解释第三点，为什么要系统停下来等待？程序永远是按照模块及调用关系顺序执行的，对于用户输入这样的事件，我们不能在需要用户输入时直接检测输入逻辑，因为很可能用户还没来得及输入就已经检测完毕，这样会得到一个无任何输入的结果，计算机的速度与人的物理动作的速度差距是巨大的。我们一贯采用的方法是：使系统停下，等待用户输入，设置一个标识，系统不停检测这个标识，当标识表示用户输入数据已得到并保存，继续运行。有兴趣的同学可以学习操作系统原理课程中的进程同步问题（IPC 问题）来了解这一类思想。

实现这样的方法很简单，例如标识为 valid，valid 的初值设置为 0，当 valid 被置 1 则认为用户输入完毕。在设函数 isValid(),当 valid 为 1 返回 true，为 0 返回 false。那么可以通过以下方法进行忙等来使系统停下来：

```
while(!isValid());
```

我们下面来观察一个 OPTION 类的最简单的实现方法：

```
enum oren {middle, left, right, up, down};
class OPTION{
    oren opt;
    bool joystickInput(){
        int xvalue, yvalue;
        xvalue = analogRead(0); yvalue = analogRead(1);
        if (xvalue < 250) opt = up;
        else if (xvalue > 750) opt = down;
        else if (yvalue < 250) opt = right;
        else if (yvalue > 750) opt = left;
        else opt = middle;
        if(opt) return 1;
        else return 0;
    }
public:
```

```
    OPTION():opt(middle) {}
    oren getOpt(){
        while(!joystickInput());
        return opt;
    }
};
```

我们在类外定义了一个 oren，方便对方向进行描述，这是我们日后使用 Option 类时需要的。analogRead()所读取的 0、1 为 JoyStick 与 Arduino 连接的两个 X/Y 轴信号管脚 A0 与 A1。根据 2.4 节我们可知 JoyStick 每一轴的取值为 0 ~ 999，初始位置（即摇杆在中心时）X 轴 Y 轴的值都为 500，那么我们定义如果摇动使这个值的偏移量超过一半时，即 250，则认为摇杆方向输入有效。我们对 JoyStick 的方向定义如下：拿起 JoyStick 摇杆指向使用者自己，有排针的一侧定义为“上”，这可以根据程序员的喜好自行规定。那么我们可以将 enum 的值与 X/Y 轴的值进行关系绑定，从而完成了方向键入功能。

在这里，我们看到了，OPTION 类只提供了一个 getOpt()函数，它实现了让系统停下等待用户，指导用户返回一个有效方向为止。getOpt()是通过不停检测 joystickInput()实现的。所以，细心的同学可以看出这里的 opt 相当于 valid，而 joystickInput()相当于 isValid()。

以上是一个最简单的 OPTION 类，在校园一卡通系统中如果使用这个类，会出现一个致命的 BUG，所以真正在项目中使用的 OPTION 类如下：

```
class OPTION{
    oren opt;
    bool bounced;
    bool isBounce(){
        int xvalue, yvalue;
        xvalue = analogRead(0); yvalue = analogRead(1);
        if(opt){
            if(abs(xvalue - 500) < 250 && abs(yvalue - 500) < 250) {
                opt = middle;
                bounced = 1;
                return 1;
```

```
            }else{
                bounced = 0;
                return 0;
            }
        }
        else return 1;
    }
    bool joystickInput(){
        int xvalue, yvalue;
        if(isBounce()){
            xvalue = analogRead(0); yvalue = analogRead(1);
            if (xvalue < 250) opt = up;
            else if (xvalue > 750) opt = down;
            else if (yvalue < 250) opt = right;
            else if (yvalue > 750) opt = left;
            else opt = middle;
        }
        if(opt && bounced) return 1;
        else return 0;
    }
public:
    OPTION():opt(middle), bounced(1){}
    oren getOpt(){
        while(!joystickInput());
        return opt;
    }
};
```

之前的 OPTION 类的 BUG 是什么？它会对校园一卡通系统产生什么影响？而项目中的 OPTION 类是通过什么原理修正这个 BUG 的？我们将这几个问题作为实训项目留给小组讨论并回答。

实训项目

3-A 习题

1. 请逐句注释 Option.h 文件。并回答：

   1) 3.7 节中所提出的 BUG 是什么，它会在什么情况下出现？

   2) 这个 BUG 如何进行修正？

2. 在 MONEY 类中，解释 `double atof(unsigned char s[])`函数的实现方法。

3. 在 MONEY 类中，为什么重载构造函数，他们分别用于什么场合？

4. 在 MONEY 类的构造函数 `MONEY(unsigned char cb[], unsigned char sb[])`中，以下两句的作用是什么？

```
for (int i = 0; i < 4; i++)  cardBuffer[i] = cb[i];
for (int i = 0; sb[i]; i++)  serialBuffer[i] = sb[i];
```

5. 在 STORAGE 类中，ptr 的作用是什么？

6. 在 STORAGE 类中，如果将成员 data 的类型从 Data&改为 Data 是否可行，为什么？

7. 在 STORAGE 类中，`write()`函数与 `writeIndex(unsigned char n)`函数分别有什么作用，请举出一个例子说明 `writeIndex(unsigned char n)`的使用场景。

8. *SectorKey.h* 文件中的如下定义的涵义是什么？

```
#define SECTORKEY     7
#define INDEXBLOCK    4
#define BALANCEBLOCK  5
```

第四部分 校园一卡通管理系统业务的实现

学习内容

在第二部分中，我们介绍了如何实现卡片的数据操作；在第三部分中，我们介绍了如何实现系统数据、输入输出、交互控制等主要功能。这两部分所包含的所有文件构成了系统框架，利用这个框架，我们可以着手开发一系列的 RFID 应用，包括一卡通消费卡系统、公交卡系统、地铁卡系统、门禁系统等等。那么我们在本章通过实现一种一卡通消费卡系统：校园一卡通管理系统，来讲解如何使用这个框架。

4.1 业务描述

我们定义校园一卡通系统的业务功能如图 4-1 所示。

	一级功能	二级功能	描述
1	Management	Registration	注册卡片。 注册成功提示："Registration SUCCESS!" 并给出注册后卡片在数据库中的查询结果。 重复注册提示："Registration FAILED, make sure it's a NEW card." 返回 Management 菜单。
		Destory	注销卡片。 注销成功提示："Destory card SUCCESS! Clear record in DataBase." 注销失败提示："Destory FAILED, no record in DataBase." 返回 Management 菜单。
2	Consumption	Query	查询卡片在数据库中的所有信息及卡片金额信息，按照以下格式显示。 **********QUERY********** 1. Card serNum is: 23B7679A69 2. Card index is: 1 3. Reserve info is: 4. Card status is: 1 5. Card balance is: 0.00 ********** 若查询失败则显示： *********No record in DataBase. ********** ********** 返回 Consumption 菜单。

		Deposit	卡片充值。 显示提示信息： **********DEPOSIT********** HINT: Please attach the card to the Reader. Card recognized. Card balance is: 0.00 等待用户在串口监视器输入数值并确认后，开始向卡片进行充值。 充值成功后显示： Deposit Success! Current balance is: 500.99 ********** 充值失败（系统原因）显示： *********No record in DataBase. ********** ********** 充值失败（卡片原因）显示： "WARNING:Communication Failed!" 返回 Consumption 菜单。
		Consume	卡片消费。 显示提示信息： **********CONSUME********** HINT: Please attach the card to the Reader. Card recognized.Card balance is: 500.99 等待用户在串口监视器输入数值并确认后，开始向卡片进行消费。 充值成功后显示： Consume Success! Current balance is: 173.74 ********** 充值失败（系统原因）显示： *********No record in DataBase. ********** ********** 充值失败（卡片原因 1）显示： "WARNING:Communication Failed!" 充值失败（卡片原因 2）显示： "No enough money!" 返回 Consumption 菜单。
3	DataBase Check	Flush Database	清空数据库。 清空成功显示： **********Data Flushed. ********** 返回 DataBase Check 菜单。
		Print DataBase	打印数据库。 全部打印完毕返回 DataBase Check 菜单。
		Count Record	打印有效记录条数。 显示：

			The record number in DataBase is: 1
4	-	Return	返回上一级菜单。
5	Reboot	-	使 loop()进入下一轮。

图 4-1 校园一卡通系统业务功能定义

4.2 业务类的实现

我们通过构建 BUSINESS 类（*Business.h*）来实现上述二级功能，通过构建 UI 类（*MenuList.h*）来完成所有二级功能菜单的组织，最后，构建 Arduino 项目来完成一级功能菜单的组织，至此，项目完成。在这里，BUSINESS 类的构建是重点。

通过附录 C *Business.h* 可以看到这个类中不再包含任何数据成员，而且所有成员函数允许直接使用，所以可以说，BUSINESS 类其实仅仅是一个业务流程函数的集合。

一、*Business.h*

- `static bool query()`

  我们首先设置 `status` 标识，用于判别卡片流程是否成功，同时开辟两个缓存区 `indexArray` 与 `balanceArray`，负责存放从卡片读得的数据块。接下来，根据 2.7 节所介绍的卡片业务流程最精简示例完成一次完整的卡片请求。在 `query()`业务中，我们不仅需要得到卡片的 `index`，也需要得到卡片的金额 `balance`。

  之后就是系统的工作了，构建 `Data` 对象，并用其创建 `STORAGE` 对象。

  ```
  Data data(indexArray[0]);
  STORAGE storage(data);
  ```

  我们总是先使用 `storage.read()`来查看卡片是否已在系统注册，如果成功，则可以开始考虑 `query()`函数的功能了。`query()`需要打印出指定卡片的金额以及卡片在数据库中存储的各个字段，则我们可以按照定义的格式追条打印数据库的各个字段；金额信息不是存放在数据库中的，所以我们需要利用 `MONEY` 类将来自卡片的的金额数据块 `balanceArray` 转换为一个数值。

  `query()`函数是所有流程函数中最简单的一个，通过这个函数，可以看到我们对系统框架的使用方法，后续函数即是业务较为复杂，也是遵循这个方法来完成的。

- `static bool registration()`

  `registration()`在卡片请求的过程中有一些变化（*Business.h*：31~58），这是由于 `registration()`是对一张新卡片进行写入操作，这需要：

1. 首先寻卡得到卡片序列号；
2. 在系统段为卡片寻找空闲记录块进行绑定；
3. 将返回的索引号 index 写回卡片；
4. 停止卡片。

所以 registration()是在一次请求过程中完成以上所有操作的，这其中需要做好卡片序列号的查重逻辑。

- static void destory()

destroy()函数仅需要获取卡片的索引号即可，通过索引号我们可以在数据库中找到记录快并将其无效化，从而将卡片从系统中注销。对于卡片来讲，也应该消除 INDEXBLOCK 的数据，使其无效化，这里面我们将其写为字符“@”，这个字符的值不在 index 的合法取值范围内。

- static void consume()

consume()需要考虑的问题较多，首先是完成用户金额输入这一操作。我们是通过串口监视器向 Arduino 中输入数据的，这个数据暂存在 serialArray 中。整个函数分为两次卡片请求来完成，第一次卡片请求是获取卡片的 INDEXBLOCK 与 BALANCEBLOCK，第二次卡片请求是将消费后的金额写回 BALANCEBLOCK。

为什么不在一次请求中同时完成读取与写入操作呢？

这是由于写入操作需要根据用户的键入数据进行运算，等待用户键入这一漫长的过程中如果卡片一直处于未关闭状态在协议上是不允许的，所以我们设计在两次卡片请求之间来完成用户键入、运算、存储等操作。

等待用户在串口监视器键入数据我们这样来完成：

```
while (!Serial.available());
```

并设立指针 p 来负责将数据从缓冲区搬运到 serialArray，其中 2 毫秒的延迟是根据 Arduino UNO 的硬件读写速度而确定的：

```
unsigned char* p = serialArray;
while (Serial.available() > 0){
    *p++ = (unsigned char)Serial.read();
    delay(2);
}
```

得到用户输入后可以开始进行运算，可以直接利用 MONEY 类完成：

```
MONEY moneyCal(balanceArray, serialArray);
balance = moneyCal.minus();
result = moneyCal.getResultBuffer();
for (int i = 0; i < 4; i++, result++) balanceArray[i] = *result;
```

在这里面我们还需要考虑到这样一个问题：消费金额超出余额的情况。所以我们在进行 moneyCal.minus()之后要判别：

```
if(balance < 0) {…}
```

如果成立则打印“No Enough Money”的提示并直接 return 来终止函数。

- static void deposit()

  deposit()的工作与 consume()类似，不再赘述。

- static void flushDB()

  调用 STORAGE::flushDataBase()。

- static void printDB()

  调用 STORAGE::printDataBase()。

- static void counter()

  调用 STORAGE::countDataBase()。

二、*MenuList.h*

UI 类中将对二级菜单界面及操作方法进行定义，包括 Management 菜单，Consumption 菜单，Database 菜单，由于涉及到用户操作，需要引入：

```
#include <Option.h>
```

对于 OPTION 类的使用方法，我们的设想是这样的：使用者负责维护一个 oren 类型的变量来保存上一次的输入结果。在需要获取用户输入时，创建 OPTION 的对象并使用 getOption()获取输入。

```
OPTION option;
oren o;
o = option.getOpt();
```

UI 类的结构同样为一个简单的函数集合，它根据用户输入来使用 BUSINESS 类。为了使每一项功能执行后能够继续显示菜单并允许用户键入，即设计一种状态机，我们需要人为设置一个循环流程，这里以 Consumption 菜单为例来介绍 UI 类的实现（*MenuList.h*：

58~94）。在 Consumption()中，我们使函数进行无休止循环，直到用户键入 return 方向键为止。在每一次循环中，我们都首先获取用户键入，根据键入的 up/down/left/right 的不同情况来调用相应的业务。设置标识 loop 来确定循环的终止函数的退出，当用户选择 return 是，loop 无效，函数终止，返回 Arduino 项目程序的 loop()中。这样的设计思路允许我们进行多级菜单的设计，请务必掌握这种方法。

三、创建 Arduino 项目

至此，所有的项目内容都已经实现，我们可以很容易创建 Arduino 项目，负责显示一级菜单。

```
#include <MenuList.h>
#include <SPI.h>
#include <Option.h>
OPTION option;
oren o;
void setup(){
  Serial.begin(9600);
  SPI.begin();
}
void loop(){
  CreateUI::titleLine("WELCOME");
  CreateUI::menu("DataBase Check", "Management", "Consumption", "Reboot");
  CreateUI::endLine();
  Serial.println();
  o = option.getOpt();
  switch(o){
    case up: UI::Database(); break;
    case left: UI::Management(); break;
    case right: UI::Consumption(); break;
    case down:
      CreateUI::titleLine("Restarting System...");
```

```
      Serial.println();
      break;
  }
  delay(500);
}
```

实训项目

4-A 程序设计

1. 运行 projectFINAL，观察校园一卡通管理系统的运行。
2. 设计一个业务类（可以继承），具有以下功能：
   - 注册卡片，并默认充值 100 元；
   - 注销卡片；
   - 从卡片扣除固定金额：1.5 元。

   构建一个项目引入业务类，利用 JoyStick“上”操作对卡片进行扣款操作。

第五部分 课程设计

开发一套 RFID 系统所需所有的知识及技术已经在前四部分进行了详细的介绍。本章将利用开发框架完成一套自定义功能 RFID 系统的课程设计，并对项目选题进行简要的指导。

学习内容

5.1 选题指南

（选题指南中所给出项目功能仅供参考，所有功能可以自行定义、变更。）

一、公交卡系统

能够完成注册、注销等卡片管理功能；

1. 能够完成充值功能；
2. 每次消费能够固定扣款，空调车扣款 2，非空调车扣款 1.5，不成功不允许乘车；
3. 能够自定义设置/变更优惠比例：95 折/9 折；
4. 扣款操作为无 JoyStick 干预的一次性划卡操作；
5. 当卡片余额低于 10，划卡后显示：“请充值（Limit balance）！”
6. 设置老年卡类型，非充值，不扣款，划卡后显示：“您好（Greeting）！ ”

二、地铁卡系统

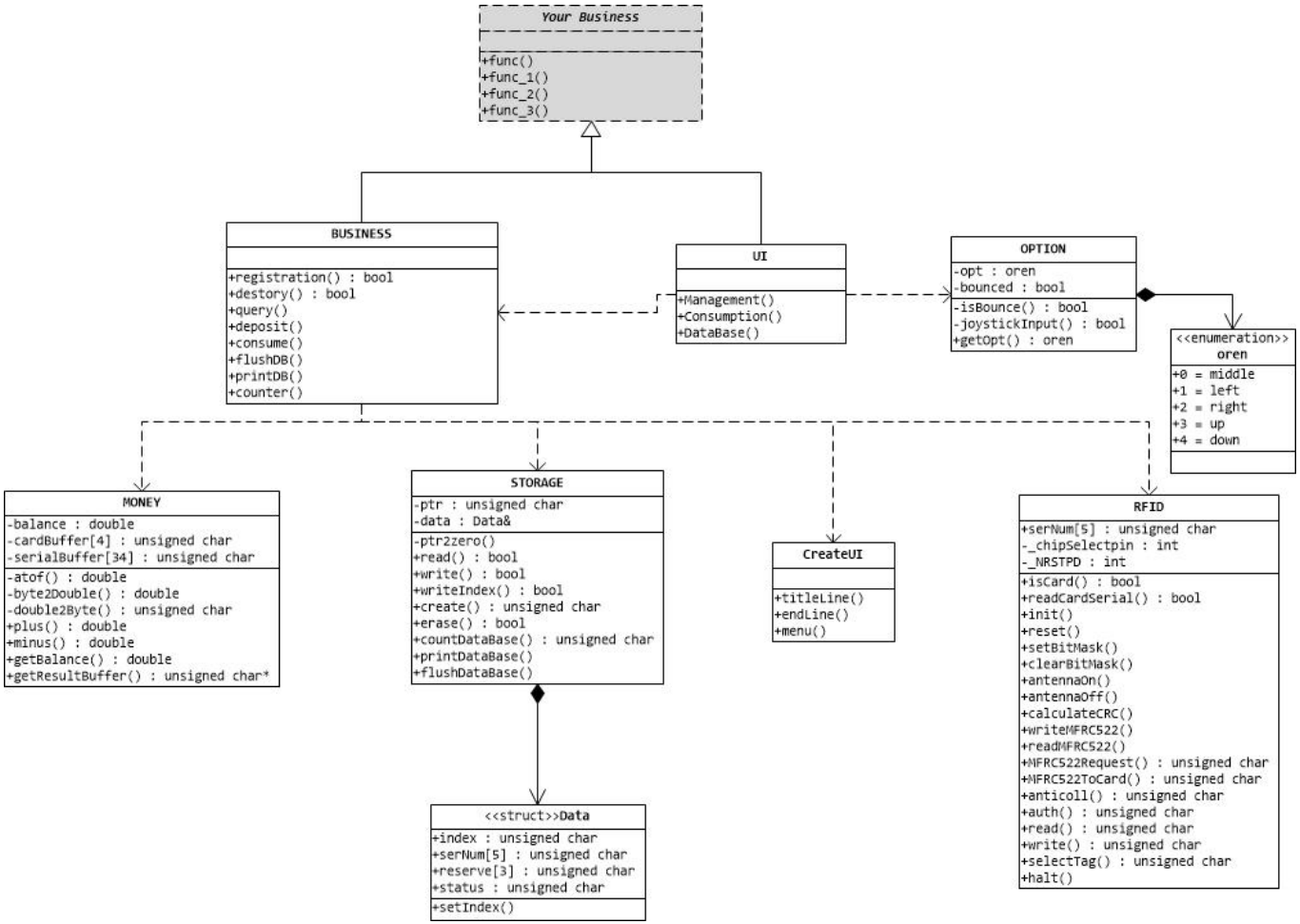
1. 能够完成注册、注销等卡片管理功能；
2. 能够完成充值功能；
3. 设计一种存储 2 条地铁线路（有交叉换成）的数据结构；
4. 能够计算入口闸机及出口闸机的最短路径，在出口完成本次乘车扣款，扣款最小值为 2，最大值不超过 10；
5. 具备卡片挂失能力，挂失后可恢复系统最后一次所记录的卡片金额；
6. 能够自定义设置/变更优惠比例：95 折/9 折；

三、门禁系统

1. 能够完成注册、注销等卡片管理功能；
2. 记录持卡人信息，设计部门字段、权限字段，允许查看持卡人信息；
3. 设计员工组织结构：管理者与员工，员工能够根据各自部门进入房间，并记录进入次数；

4. 管理者能够进入所有部门房间；
5. 具备卡片挂失能力。

附录 A 系统 UML 图（*UML.pdf*）



附录 B 开发框架源代码

1. Storage.h

```
1.  /*************************FILE BEGIN*************************/
2.  /*
3.   * Storage.h - Data structure management, read/write to EEPROM
4.   * Creator: Han.Sketchpink
5.   * Time: 2015.10.1
6.   */
7.  #ifndef STORAGE_H
8.  #define STORAGE_H
9.  #include <EEPROM.h>      //0 ~ 511
10.
11. typedef struct Data {
12.     unsigned char index; //0: Null; other: 1 ~ 51
13.     unsigned char serNum[5];
14.     unsigned char reserve[3];
15.     unsigned char status;
16.     Data(){
17.         index = 255;
18.         for(int i = 0; i < 5; i++)  serNum[i] = 0;
19.         for(int i = 0; i < 3; i++)  reserve[i] = 0;
20.         status = 0;
21.     }
22.     Data(unsigned char num):index(num){
23.         for(int i = 0; i < 5; i++)  serNum[i] = 0;
24.         for(int i = 0; i < 3; i++)  reserve[i] = 0;
25.         status = 0;
26.     }
27.     void setIndex(unsigned char num){
28.         index = num;
29.     }
30. }Data; //Total 10Byte
31.
32. class STORAGE {
33.     unsigned char ptr;       //0 ~ 50
34.     Data& data;
35.     void ptr2Zero() {
36.         ptr = 0;
37.     }
38. public:
```

```
39.     STORAGE(Data& d):data(d), ptr(0){}
40.     bool read(){
41.         ptr2Zero();
42.         for(; ptr <= 50 && EEPROM.read(ptr * 10) != data.index; ptr++);
43.         if(ptr > 50 || !EEPROM.read(ptr * 10)) return 0;
44.         else {
45.             for(int i = 0; i < 5; i++)
46.                 data.serNum[i] = EEPROM.read(ptr * 10 + 1 + i);
47.             for(int i = 0; i < 3; i++)
48.                 data.reserve[i] = EEPROM.read(ptr * 10 + 6 + i);
49.             data.status = EEPROM.read(ptr * 10 + 9);
50.             return 1;
51.         }
52.     }
53.     bool write(){
54.         ptr2Zero();
55.         for(; ptr <= 50 && EEPROM.read(ptr * 10) != data.index; ptr++);
56.         if(ptr > 50 || !EEPROM.read(ptr * 10)) return 0;
57.         else {
58.             for(int i = 0; i < 5; i++)
59.                 EEPROM.write(ptr * 10 + 1 + i, data.serNum[i]);
60.             for(int i = 0; i < 3; i++)
61.                 EEPROM.write(ptr * 10 + 6 + i, data.reserve[i]);
62.             EEPROM.write(ptr * 10 + 9, data.status);
63.             return 1;
64.         }
65.     }
66.     bool writeIndex(unsigned char n){
67.         ptr2Zero();
68.         for(; ptr <= 50 && EEPROM.read(ptr * 10) != data.index; ptr++);
69.         if(ptr > 50 || !EEPROM.read(ptr * 10)) return 0;
70.         else {
71.             EEPROM.write(ptr * 10, n);
72.             return 1;
73.         }
74.     }
75.     unsigned char create(){
76.         unsigned char validate[5] = {0};   //each read from DB
77.         unsigned char tempPtr = 0;      //traverse in this function
78.         unsigned char status;
79.          //locate
80.         for (ptr2Zero(); ptr <= 50 && EEPROM.read(ptr * 10); ptr++);
81.         if (ptr > 50) status = 0; // No enough free block
82.         else {
```

```
83.             for (; tempPtr <= 50; tempPtr++){
84.                 //data invalid
85.                 if (!EEPROM.read(tempPtr * 10)) continue;
86.                 else {     //data exist
87.                     for(int i = 0; i < 5; i++)  validate[i] =
    EEPROM.read(tempPtr * 10 + 1 + i);
88.                     unsigned char* p = data.serNum;
89.                     unsigned char* q = validate;
90.                     unsigned char counter = 0;
91.                     //stopped when not equal
92.                     for (; counter <= 5 && *p++ == *q++; counter++);
93.                     if (counter == 5) break;            //match!
94.
95.                 }
96.             }
97.             if (tempPtr == 51) {                // All checked
98.                 EEPROM.write(ptr * 10, ptr + 1); // index == ptr + 1
99.                 for(int i = 0; i < 5; i++)
100.                    EEPROM.write(ptr * 10 + 1 + i, data.serNum[i]);
101.                for(int i = 0; i < 3; i++)
102.                    EEPROM.write(ptr * 10 + 6 + i, data.reserve[i]);
103.                EEPROM.write(ptr * 10 + 9, data.status);
104.                status = ptr + 1;
105.            }
106.            else status = 0;
107.        }
108.        return status;
109.    }
110.    bool erase(){
111.        ptr = data.index - 1;
112.        if (ptr <= 50) {
113.            EEPROM.write(ptr, 0);
114.            return 1;
115.        }
116.        return 0;
117.    }
118.    static unsigned char countDataBase(){
119.        unsigned char counter = 0;
120.
121.        for (int i = 0; i <= 500; i += 10){
122.            if(EEPROM.read(i)) counter++;
123.        }
124.        return counter;
125.    }
```

```
126.      static void printDataBase(){
127.          for (int i = 0; i < 512; i++) {
128.              Serial.print(EEPROM.read(i), DEC);
129.              Serial.println();
130.              delay(500);
131.          }
132.      }
133.      static void flushDataBase(){
134.          int address = 0;
135.          for (; address < 512; address++) EEPROM.write(address, 0);
136.      }
137.  };
138.  #endif
139.  /************************FILE END************************/
```

2. SectorKey.h

```
1.  /*************************FILE BEGIN**************************/
2.  /*
3.   * SectorKey.h - block definition
4.   * Creator: Han.Sketchpink
5.   * Time: 2015.10.1
6.   */
7.  #ifndef SECTORKEY_H
8.  #define SECTORKEY_H
9.  #define SECTORKEY   7
10. #define INDEXBLOCK    4
11. #define BALANCEBLOCK 5
12.
13. unsigned char sectorKeyA[16][16] = {
14.        {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
15.        {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
16.        {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},};
17. unsigned char sectorNewKeyA[16][16] = {
18.        {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
19.        {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xff,0x07,0x80,0x69, 0xFF,
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
20.        {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xff,0x07,0x80,0x69, 0xFF,
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF},};
21. #endif
22. /*************************FILE END**************************/
```

3. RFID.h

```
1.  /*************************FILE BEGIN**************************/
2.  /*
3.   * RFID.h - RFID : Class Definition
4.   * Editor: Han.Sketchpink
5.   * Time: 2015.10.1
6.   */
7.  #ifndef RFID_h
8.  #define RFID_h
9.  #include <Arduino.h>
10. #include <SPI.h>
11. #define MAX_LEN 16   // 数组最大长度,根据Mifare 1单个数据块长度16字节
12.
13. //MF522命令字PCD,写入MFRC522命令寄存器(CommandReg 0x01)
14. #define PCD_IDLE            0x00              //无动作，取消当前命令
15. #define PCD_AUTHENT         0x0E              //验证密钥
16. #define PCD_RECEIVE         0x08              //接收数据
17. #define PCD_TRANSMIT        0x04              //发送数据
18. #define PCD_TRANSCEIVE      0x0C              //发送并接收数据
19. #define PCD_RESETPHASE      0x0F              //复位
20. #define PCD_CALCCRC         0x03              //CRC计算
21.
22. //Mifare_One卡片命令字PICC，命令Mifare 1完成相应操作
23. #define PICC_REQIDL         0x26            //寻天线区内未进入休眠状态
24. #define PICC_REQALL         0x52              //寻天线区内全部卡
25. #define PICC_ANTICOLL       0x93              //防冲撞
26. #define PICC_SElECTTAG      0x93              //选卡
27. #define PICC_AUTHENT1A      0x60              //验证A密钥
28. #define PICC_AUTHENT1B      0x61              //验证B密钥
29. #define PICC_READ           0x30              //读块
30. #define PICC_WRITE          0xA0              //写块
31. #define PICC_DECREMENT      0xC0              //值增
32. #define PICC_INCREMENT      0xC1              //值减
33. #define PICC_RESTORE        0xC2              //调块数据到缓冲区
34. #define PICC_TRANSFER       0xB0              //保存缓冲区中数据
35. #define PICC_HALT           0x50              //休眠
36.
37. //和MF522通讯时返回的错误代码
38. #define MI_OK               0
39. #define MI_NOTAGERR         1
40. #define MI_ERR              2
```

```
41.
42. //------------------MFRC522寄存器---------------
43. //Page 0:Command and Status
44. #define     Reserved00          0x00
45. #define     CommandReg          0x01
46. #define     CommIEnReg          0x02
47. #define     DivlEnReg           0x03
48. #define     CommIrqReg          0x04
49. #define     DivIrqReg           0x05
50. #define     ErrorReg            0x06
51. #define     Status1Reg          0x07
52. #define     Status2Reg          0x08
53. #define     FIFODataReg         0x09
54. #define     FIFOLevelReg        0x0A
55. #define     WaterLevelReg       0x0B
56. #define     ControlReg          0x0C
57. #define     BitFramingReg       0x0D
58. #define     CollReg             0x0E
59. #define     Reserved01          0x0F
60. //Page 1:Command
61. #define     Reserved10          0x10
62. #define     ModeReg             0x11
63. #define     TxModeReg           0x12
64. #define     RxModeReg           0x13
65. #define     TxControlReg        0x14
66. #define     TxAutoReg           0x15
67. #define     TxSelReg            0x16
68. #define     RxSelReg            0x17
69. #define     RxThresholdReg      0x18
70. #define     DemodReg            0x19
71. #define     Reserved11          0x1A
72. #define     Reserved12          0x1B
73. #define     MifareReg           0x1C
74. #define     Reserved13          0x1D
75. #define     Reserved14          0x1E
76. #define     SerialSpeedReg      0x1F
77. //Page 2:CFG
78. #define     Reserved20          0x20
79. #define     CRCResultRegM       0x21
80. #define     CRCResultRegL       0x22
81. #define     Reserved21          0x23
82. #define     ModWidthReg         0x24
83. #define     Reserved22          0x25
84. #define     RFCfgReg            0x26
```

```
85. #define     GsNReg              0x27
86. #define     CWGsPReg            0x28
87. #define     ModGsPReg           0x29
88. #define     TModeReg            0x2A
89. #define     TPrescalerReg        0x2B
90. #define     TReloadRegH         0x2C
91. #define     TReloadRegL         0x2D
92. #define     TCounterValueRegH    0x2E
93. #define     TCounterValueRegL    0x2F
94. //Page 3:TestRegister
95. #define     Reserved30          0x30
96. #define     TestSel1Reg         0x31
97. #define     TestSel2Reg         0x32
98. #define     TestPinEnReg        0x33
99. #define     TestPinValueReg      0x34
100.    #define     TestBusReg          0x35
101.    #define     AutoTestReg         0x36
102.    #define     VersionReg          0x37
103.    #define     AnalogTestReg        0x38
104.    #define     TestDAC1Reg         0x39
105.    #define     TestDAC2Reg         0x3A
106.    #define     TestADCReg          0x3B
107.    #define     Reserved31          0x3C
108.    #define     Reserved32          0x3D
109.    #define     Reserved33          0x3E
110.    #define     Reserved34          0x3F
111.
112.    class RFID
113.    {
114.      public:
115.        RFID(int chipSelectPin, int NRSTPD);
116.        bool isCard();
117.        bool readCardSerial();
118.        void init();
119.        void reset();
120.        inline void setBitMask(unsigned char reg, unsigned char mask);
121.        inline void clearBitMask(unsigned char reg, unsigned char
    mask);
122.        void antennaOn(void);
123.        void antennaOff(void);
124.        void calculateCRC(unsigned char *pIndata, unsigned char len,
    unsigned char *pOutData);
125.        inline void writeMFRC522(unsigned char addr, unsigned char
    val);
```

```
126.        inline unsigned char readMFRC522(unsigned char addr);
127.        unsigned char MFRC522Request(unsigned char reqMode, unsigned
    char *TagType);
128.        unsigned char MFRC522ToCard(unsigned char command, unsigned
    char *sendData, unsigned char sendLen, unsigned char *backData,
    unsigned int *backLen);
129.        unsigned char anticoll(unsigned char *serNum);
130.        unsigned char auth(unsigned char authMode, unsigned char
    BlockAddr, unsigned char *Sectorkey, unsigned char *serNum);
131.        unsigned char read(unsigned char blockAddr, unsigned char
    *recvData);
132.        unsigned char write(unsigned char blockAddr, unsigned char
    *writeData);
133.        unsigned char selectTag(unsigned char *serNum);
134.        void halt();
135.        unsigned char serNum[5];      //4字节卡序列号，第5字节为校验字节
136.      private:
137.        int _chipSelectPin;
138.        int _NRSTPD;
139.    };
140.    #endif
141.    /*********************FILE END*************************/
```

4. RFID.cpp

```
1. /*************************FILE BEGIN****************************/
2. /*
3.  * RFID.cpp - RFID : Class Implement
4.  * Editor: Han.Sketchpink
5.  * Time: 2015.10.1
6.  */
7. #include <Arduino.h>
8. #include <RFID.h>
9. /*****************************************************************
10.  * 构造 RFID
11.  * int chipSelectPin RFID /ENABLE pin
12. *****************************************************************/
13. RFID::RFID(int chipSelectPin, int NRSTPD)
14. {
15.   _chipSelectPin = chipSelectPin;
16.   _NRSTPD = NRSTPD;
17.   pinMode(_chipSelectPin,OUTPUT);     // 设置管脚_chipSelectPin为输出
    并连接到模块使能口
18.   digitalWrite(_chipSelectPin, LOW);
19.   pinMode(_NRSTPD,OUTPUT);         // 设置管脚NRSTPD为输出，非重置或掉电
20.   digitalWrite(_NRSTPD, HIGH);
21.   for(int i = 0; i < 5; i++)serNum[i] = 0;
22. }
23.
24. /*****************************************************************
25.  * 函 数 名：isCard
26.  * 功能描述：寻卡：在阅读器功率范围内找到效卡片
27.  * 输入参数：无
28.  * 返 回 值：成功返回ture 失败返回false
29. *****************************************************************/
30. bool RFID::isCard()
31. {
32.   unsigned char status;
33.   unsigned char str[MAX_LEN];
34.   //str位置获取了若干字节返回数据
35.   status = MFRC522Request(PICC_REQIDL, str);
36.   if (status == MI_OK)
37.     return true;
38.   else
39.     return false;
```

```
}//str内容并未使用，退栈/析构

/*****************************************************************
 * 函 数 名：readCardSerial
 * 功能描述：返回卡的序列号 4字节，调用防冲突完成取序列号
 * 输入参数：无
 * 返 回 值：成功返回ture 失败返回false
 *****************************************************************/
bool RFID::readCardSerial(){
  unsigned char status;
  unsigned char str[MAX_LEN];
  // 防冲撞，返回卡的序列号 4字节，存入serNum中
  status = anticoll(str);       //防冲突检测包含取序列号操作
  if (status == MI_OK){
    memcpy(serNum, str, 5);  //str最终被退栈析构，公有成员serNum保存数据
    return true;
  }
  else
    return false;
}

/*****************************************************************
 * 函 数 名：init
 * 功能描述：初始化RC522
 * 输入参数：无
 * 返 回 值：无
 *****************************************************************/
void RFID::init()
{
  digitalWrite(_NRSTPD,HIGH);
  reset();
  //Timer: TPrescaler*TreloadVal/6.78MHz = 24ms
  //Tauto=1; f(Timer) = 6.78MHz/TPreScaler
  writeMFRC522(TModeReg, 0x8D);
  //TModeReg[3..0] + TPrescalerReg
  writeMFRC522(TPrescalerReg, 0x3E);
  writeMFRC522(TReloadRegL, 30);
  writeMFRC522(TReloadRegH, 0);
  writeMFRC522(TxAutoReg, 0x40);    //100%ASK
  writeMFRC522(ModeReg, 0x3D);    // CRC valor inicial de 0x6363
  //ClearBitMask(Status2Reg, 0x08); //MFCrypto1On=0
  //writeMFRC522(RxSelReg, 0x86);   //RxWait = RxSelReg[5..0]
  //writeMFRC522(RFCfgReg, 0x7F);   //RxGain = 48dB
  antennaOn();    //打开天线
```

```
84. }
85.
86. /*********************************************************************
87.  * 函 数 名：reset
88.  * 功能描述：复位RC522
89.  * 输入参数：无
90.  * 返 回 值：无
91.  *********************************************************************/
92. void RFID::reset()
93. {
94.   writeMFRC522(CommandReg, PCD_RESETPHASE);
95. }
96.
97.   /*********************************************************************
98.     * 函 数 名：writeMFRC522
99.     * 功能描述：向MFRC522的某一寄存器写一个字节数据
100.    * 输入参数：addr--寄存器地址；val--要写入的值
101.    * 返 回 值：无
102.   *********************************************************************/
103.   void RFID::writeMFRC522(unsigned char addr, unsigned char val)
104.   {
105.     digitalWrite(_chipSelectPin, LOW);
106.     //地址格式：0XXXXXX0
107.     SPI.transfer((addr<<1)&0x7E);
108.     SPI.transfer(val);
109.     digitalWrite(_chipSelectPin, HIGH);
110.   }
111.
112.   /*********************************************************************
113.    * 函 数 名：readMFRC522
114.    * 功能描述：从MFRC522的某一寄存器读一个字节数据
115.    * 输入参数：addr--寄存器地址
116.    * 返 回 值：返回读取到的一个字节数据
117.   *********************************************************************/
118.   unsigned char RFID::readMFRC522(unsigned char addr)
119.   {
120.     unsigned char val;
121.     digitalWrite(_chipSelectPin, LOW);
122.     SPI.transfer(((addr<<1)&0x7E) | 0x80);
123.     val =SPI.transfer(0x00);
124.     digitalWrite(_chipSelectPin, HIGH);
125.     return val;
126.   }
127.
```

```
128.  /*************************************************************
129.   * 函 数 名：setBitMask
130.   * 功能描述：置RC522寄存器位
131.   * 输入参数：reg--寄存器地址;mask--置位值
132.   * 返 回 值：无
133.   *************************************************************/
134.  void RFID::setBitMask(unsigned char reg, unsigned char mask)
135.  {
136.    unsigned char tmp;
137.    tmp = readMFRC522(reg);           //取寄存器原值并置掩码
138.    writeMFRC522(reg, tmp | mask);    //set bit mask, 掩码处置1
139.  }
140.
141.  /*************************************************************
142.   * 函 数 名：clearBitMask
143.   * 功能描述：清RC522寄存器位
144.   * 输入参数：reg--寄存器地址;mask--清位值
145.   * 返 回 值：无
146.  *************************************************************/
147.  void RFID::clearBitMask(unsigned char reg, unsigned char mask)
148.  {
149.    unsigned char tmp;
150.    tmp = readMFRC522(reg);            //取寄存器原值并置掩码
151.    writeMFRC522(reg, tmp & (~mask));  // clear bit mask, 掩码处清0
152.  }
153.
154.  /*************************************************************
155.   * 函 数 名：antennaOn
156.   * 功能描述：开启天线,每次启动或关闭天险发射之间应至少有1ms的间隔
157.   * 输入参数：无
158.   * 返 回 值：无
159.  *************************************************************/
160.  void RFID::antennaOn(void)
161.  {
162.    unsigned char temp;
163.    temp = readMFRC522(TxControlReg);
164.    if (!(temp & 0x03))
165.    {
166.      setBitMask(TxControlReg, 0x03);
167.    }
168.  }
169.
170.  /*************************************************************
171.   * 函 数 名：antennaOff
```

```
172.     * 功能描述：关闭天线,每次启动或关闭天险发射之间应至少有1ms的间隔
173.     * 输入参数：无
174.     * 返 回 值：无
175.     *************************************************************/
176.    void RFID::antennaOff(void)
177.    {
178.      unsigned char temp;
179.      temp = readMFRC522(TxControlReg);
180.      if (!(temp & 0x03))
181.      {
182.        clearBitMask(TxControlReg, 0x03);
183.      }
184.    }
185.
186.    /*************************************************************
187.     * 函 数 名：calculateCRC
188.     * 功能描述：用MF522计算CRC
189.     * 输入参数：pIndata--要读数CRC的数据，len--数据长度，pOutData--计算的
    CRC结果
190.     * 返 回 值：无
191.     *************************************************************/
192.    void RFID::calculateCRC(unsigned char *pIndata, unsigned char len,
    unsigned char *pOutData)
193.    {
194.      unsigned char i, n;
195.      clearBitMask(DivIrqReg, 0x04);      //CRCIrq = 0
196.      setBitMask(FIFOLevelReg, 0x80);     //清FIFO指针
197.      //Write_MFRC522(CommandReg, PCD_IDLE);
198.      //向FIFO中写入数据
199.      for (i=0; i<len; i++)
200.        writeMFRC522(FIFODataReg, *(pIndata+i));
201.      writeMFRC522(CommandReg, PCD_CALCCRC);
202.      //等待CRC计算完成
203.      i = 0xFF;        //255倒计时
204.      do
205.      {
206.        n = readMFRC522(DivIrqReg);
207.        i--;
208.      }
209.      while ((i!=0) && !(n&0x04));      //CRCIrq = 1
210.      //读取CRC计算结果
211.      pOutData[0] = readMFRC522(CRCResultRegL);
212.      pOutData[1] = readMFRC522(CRCResultRegM);
213.    }
```

```
214.
215.   /*************************************************************
216.    * 函 数 名：MFRC522ToCard
217.    * 功能描述：RC522和ISO14443卡通讯
218.    * 输入参数：command--MF522命令字，
219.    *           sendData--通过RC522发送到卡片的数据，
220.    *                     sendLen--发送的数据长度
221.    *                     backData--接收到的卡片返回数据，
222.    *                     backLen--返回数据的位长度
223.    * 返 回 值：成功返回MI_OK
224.   *************************************************************/
225.   unsigned char RFID::MFRC522ToCard(unsigned char command, unsigned
   char *sendData, unsigned char sendLen, unsigned char *backData,
   unsigned int *backLen)
226.   {
227.     unsigned char status = MI_ERR;
228.     unsigned char irqEn = 0x00;
229.     unsigned char waitIRq = 0x00;
230.     unsigned char lastBits;
231.     unsigned char n;
232.     unsigned int i;
233.     switch (command)
234.     {
235.       case PCD_AUTHENT:   //认证卡密
236.       {
237.         irqEn = 0x12;       // 0001 0010
238.         waitIRq = 0x10;     // 0001 0000
239.         break;
240.       }
241.       case PCD_TRANSCEIVE:  //发送FIFO中数据
242.       {
243.         irqEn = 0x77;       // 0111 0111  为什么不写0xF7?
244.         waitIRq = 0x30;     // 0011 0000
245.         break;
246.       }
247.       default:
248.         break;
249.     }
250.     writeMFRC522(CommIEnReg, irqEn|0x80); //允许中断请求 1000 0000
251.     clearBitMask(CommIrqReg, 0x80);       //清除所有中断请求位
252.     setBitMask(FIFOLevelReg, 0x80);     //FlushBuffer=1, FIFO初始化
253.     writeMFRC522(CommandReg, PCD_IDLE);   //无动作，取消当前命令
254.     //内存向MFRC522 FIFO中写入数据,不超过16字节,其中第一个字节为卡片命令
   字PICC_READ、PICC_WRITE,第二字节为块地址
```

```
255.     for (i=0; i<sendLen; i++)
256.       writeMFRC522(FIFODataReg, sendData[i]);
257.     //执行命令
258.     writeMFRC522(CommandReg, command);    //传入命令字：准备交互数据或
   认证
259.     if (command == PCD_TRANSCEIVE)
260.       setBitMask(BitFramingReg, 0x80);
   //StartSend=1,transmission of data starts  开始交互数据
261.     //等待接收数据完成,等待中断信号
262.     i = 2000;           //i根据时钟频率调整，操作M1卡最大等待时间25ms
263.     do
264.     {
265.       //CommIrqReg[7..0]
266.       //Set1 TxIRq RxIRq IdleIRq HiAlerIRq LoAlertIRq ErrIRq
   TimerIRq       7类中断
267.       n = readMFRC522(CommIrqReg);    //轮询CommIrqReg,任意一类中断
   发生，n非0
268.       i--;
269.     }
270.     //i倒计时结束、TimerValueReg超时结束、5 4位发生中断结束(期望)
271.     while ((i!=0) && !(n&0x01) && !(n&waitIRq));
272.     clearBitMask(BitFramingReg, 0x80);      //StartSend=0
273.     if (i != 0)     //检查是否由于超时结束，保证非第一类结束
274.     {
275.       if(!(readMFRC522(ErrorReg) & 0x1B)) //BufferOvfl Collerr
   CRCErr ProtecolErr 非此若干种类错误，否则MI_ERR
276.       {
277.         status = MI_OK;                //成功!
278.         if (n & irqEn & 0x01)
279.           status = MI_NOTAGERR;
280.        //若做的是数据交互，则做数据保存的善后处理
281.         if (command == PCD_TRANSCEIVE)
282.         {
283.           n = readMFRC522(FIFOLevelReg);  //字节数
284.           lastBits = readMFRC522(ControlReg) & 0x07;
285.           if (lastBits)  //计数，位
286.             *backLen = (n-1)*8 + lastBits;//末尾若干bit不足1字节
287.           else
288.             *backLen = n*8;  //字节整数倍
289.           if (n == 0)        //两种越界情况
290.             n = 1;
291.           if (n > MAX_LEN)
292.             n = MAX_LEN;
293.       //读取FIFO中接收到的数据，FIFO是真正的交换区，其他变量为临时交换区
```

```
294.          for (i=0; i<n; i++)
295.            backData[i] = readMFRC522(FIFODataReg);
296.        }
297.       }
298.       else
299.         status = MI_ERR;
300.     }
301.     //SetBitMask(ControlReg,0x80);          //timer stops
302.     //Write_MFRC522(CommandReg, PCD_IDLE);
303.     return status;
304.   }
305.
306.   /*************************************************************
307.    * 函 数 名：MFRC522Request
308.    * 功能描述：寻卡，读取卡类型号
309.    * 输入参数：reqMode--寻卡方式，
310.    *          TagType--返回卡片类型
311.    *                 0x4400 = Mifare_UltraLight
312.    *                 0x0400 = Mifare_One(S50)
313.    *                 0x0200 = Mifare_One(S70)
314.    *                 0x0800 = Mifare_Pro(X)
315.    *                 0x4403 = Mifare_DESFire
316.    * 返 回 值：成功返回MI_OK
317.   *************************************************************/
318.   unsigned char RFID::MFRC522Request(unsigned char reqMode,
   unsigned char *TagType)
319.   {
320.     unsigned char status;
321.     unsigned int backBits;      //接收到的数据bit数
322.     writeMFRC522(BitFramingReg, 0x07);    //TxLastBists =
   BitFramingReg[2..0] ???
323.     //TagType放置发送数据(1Byte)，同时TagType位置保存返回数据(backbits
   bit),此区域可看做数据交换区
324.     TagType[0] = reqMode;
325.     status = MFRC522ToCard(PCD_TRANSCEIVE, TagType, 1, TagType,
   &backBits);
326.     //检查状态，backbits不超过16位?
327.     if ((status != MI_OK) || (backBits != 0x10))
328.       status = MI_ERR;
329.     return status;
330.   }
331.
332.   /*************************************************************
333.    * 函 数 名：anticoll
```

```
334.     * 功能描述：防冲突检测：多张卡片同时出现在广播方位内时，只选取一张卡片
335.     * 输入参数：serNum--返回4字节卡序列号,第5字节为校验字节
336.     * 返 回 值：成功返回MI_OK
337.     *****************************************************/
338.    unsigned char RFID::anticoll(unsigned char *serNum)
339.    {
340.      unsigned char status;
341.      unsigned char i;
342.      unsigned char serNumCheck = 0;       // 0000 0000
343.      unsigned int unLen;
344.      //ClearBitMask(Status2Reg, 0x08);   //TempSensclear
345.      //ClearBitMask(CollReg,0x80);    //ValuesAfterColl
346.      writeMFRC522(BitFramingReg, 0x00);    //TxLastBists =
   BitFramingReg[2..0]
347.      serNum[0] = PICC_ANTICOLL;
348.      serNum[1] = 0x20;
349.      //防冲突的核心内容为完整一次卡片通信，
350.      status = MFRC522ToCard(PCD_TRANSCEIVE, serNum, 2, serNum,
   &unLen);
351.      if (status == MI_OK)
352.      {
353.        //校验卡序列号
354.        for (i=0; i<4; i++)              //本卡4字节序列号，1字节校验
355.          serNumCheck ^= serNum[i];
356.        if (serNumCheck != serNum[i])   //5个字节全部做异或校验
   (serNumCheck),结果与serNum[4]校验字节应一致
357.          status = MI_ERR;
358.      }
359.      //SetBitMask(CollReg, 0x80);    //ValuesAfterColl=1
360.      return status;      //serNum成员保存返回
361.    }
362.
363.    /*****************************************************************
364.     * 函 数 名：auth
365.     * 功能描述：验证卡片密码
366.     * 输入参数：authMode--密码验证模式
367.     *                   0x60 = 验证A密钥
368.     *                   0x61 = 验证B密钥
369.     *          BlockAddr--块地址
370.     *          Sectorkey--扇区密码
371.     *          serNum--卡片序列号，4字节
372.     * 返 回 值：成功返回MI_OK
373.     *****************************************************/
374.    unsigned char RFID::auth(unsigned char authMode, unsigned char
```

```
   BlockAddr, unsigned char *Sectorkey, unsigned char *serNum)
375.   {
376.     unsigned char status;
377.     unsigned int recvBits;
378.     unsigned char i;
379.     unsigned char buff[12];
380.     //验证指令+块地址＋扇区密码＋卡序列号
381.     buff[0] = authMode;
382.     buff[1] = BlockAddr;
383.     for (i=0; i<6; i++)
384.       buff[i+2] = *(Sectorkey+i);
385.     for (i=0; i<4; i++)
386.       buff[i+8] = *(serNum+i);
387.     status = MFRC522ToCard(PCD_AUTHENT, buff, 12, buff, &recvBits);
388.     if ((status != MI_OK) || (!(readMFRC522(Status2Reg) & 0x08)))
389.       status = MI_ERR;
390.     return status;
391.   }
392.
393.   /*************************************************************
394.    * 函 数 名：read
395.    * 功能描述：读块数据
396.    * 输入参数：blockAddr--块地址;recvData--读出的块数据
397.    * 返 回 值：成功返回MI_OK
398.    *************************************************************/
399.   unsigned char RFID::read(unsigned char blockAddr, unsigned char
   *recvData)
400.   {
401.     unsigned char status;
402.     unsigned int unLen;
403.     recvData[0] = PICC_READ;      //第一字节为卡片命令字
404.     recvData[1] = blockAddr;      //第二字节为块地址
405.     calculateCRC(recvData,2, &recvData[2]);
406.     status = MFRC522ToCard(PCD_TRANSCEIVE, recvData, 4, recvData,
   &unLen);
407.     if ((status != MI_OK) || (unLen != 0x90))
408.       status = MI_ERR;
409.     return status;
410.   }
411.
412.   /*************************************************************
413.    * 函 数 名：write
414.    * 功能描述：写块数据
415.    * 输入参数：blockAddr--块地址;writeData--向块写16字节数据
```

```
416.     * 返 回 值：成功返回MI_OK
417.     *************************************************************/
418.    unsigned char RFID::write(unsigned char blockAddr, unsigned char
    *writeData)
419.    {
420.      unsigned char status;
421.      unsigned int recvBits;
422.      unsigned char i;
423.      unsigned char buff[18];       //16 + 2
424.      buff[0] = PICC_WRITE;         //第一字节为卡片命令字
425.      buff[1] = blockAddr;          //第二字节为块地址
426.      calculateCRC(buff, 2, &buff[2]);
427.      status = MFRC522ToCard(PCD_TRANSCEIVE, buff, 4, buff,
    &recvBits);
428.      if ((status != MI_OK) || (recvBits != 4) || ((buff[0] & 0x0F) !=
    0x0A))
429.        status = MI_ERR;
430.      if (status == MI_OK)   //第一次toCard确认写操作，第二次toCard完成
    数据写入
431.      {
432.        for (i=0; i<16; i++)    //?FIFO?16Byte?? Datos a la FIFO 16Byte
    escribir
433.          buff[i] = *(writeData+i);  //buff加载数据，传给toCard
434.        calculateCRC(buff, 16, &buff[16]);
435.        status = MFRC522ToCard(PCD_TRANSCEIVE, buff, 18, buff,
    &recvBits);    //数据写入
436.        if ((status != MI_OK) || (recvBits != 4) || ((buff[0] & 0x0F) !=
    0x0A))
437.          status = MI_ERR;
438.      }
439.      return status;
440.    }
441.
442.    /*************************************************************
443.     * 函 数 名：selectTag
444.     * 功能描述：选卡，读取卡存储器容量，锁定卡片，防止多次读写
445.     * 输入参数：serNum--传入卡序列号
446.     * 返 回 值：成功返回卡容量
447.    *************************************************************/
448.    unsigned char RFID::selectTag(unsigned char *serNum)
449.    {
450.      unsigned char i;
451.      unsigned char status;
452.      unsigned char size;
```

```
453.     unsigned int recvBits;
454.     unsigned char buffer[9];
455.     //ClearBitMask(Status2Reg, 0x08); //MFCrypto1On=0
456.     buffer[0] = PICC_SElECTTAG;
457.     buffer[1] = 0x70;
458.     for (i=0; i<5; i++)
459.       buffer[i+2] = *(serNum+i);
460.     calculateCRC(buffer, 7, &buffer[7]);
461.     status = MFRC522ToCard(PCD_TRANSCEIVE, buffer, 9, buffer,
   &recvBits);
462.     if ((status == MI_OK) && (recvBits == 0x18))
463.       size = buffer[0];
464.     else
465.       size = 0;
466.     return size;
467.   }
468.
469.   /*************************************************************
470.    * 函 数 名：Halt
471.    * 功能描述：命令卡片进入休眠状态
472.    * 输入参数：无
473.    * 返 回 值：无
474.   *************************************************************/
475.   void RFID::halt()
476.   {
477.     unsigned char status;
478.     unsigned int unLen;
479.     unsigned char buff[4];
480.     buff[0] = PICC_HALT;
481.     buff[1] = 0;
482.     calculateCRC(buff, 2, &buff[2]);
483.     status = MFRC522ToCard(PCD_TRANSCEIVE, buff, 4, buff,&unLen);
484.   }
485.   /************************FILE END************************/
```

5. Option.h

```
1. /************************FILE BEGIN***************************/
2. /*
3.  * OPTION.h - Initialize joystick as input
4.  * Creator: Han.Sketchpink
5.  * Time: 2015.10.1
6.  */
7. #ifndef OPTION_H
8. #define OPTION_H
9.
10. enum oren {middle, left, right, up, down};
11.
12. class OPTION{
13.     oren opt;
14.     bool bounced;
15.     bool isBounce(){
16.         int xvalue, yvalue;
17.         xvalue = analogRead(0);
18.         yvalue = analogRead(1);
19.         if(opt){
20.             if(abs(xvalue - 500) < 250 && abs(yvalue - 500) < 250){
21.                 opt = middle;
22.                 bounced = 1;
23.                 return 1;
24.             }else{
25.                 bounced = 0;
26.                 return 0;
27.             }
28.         }
29.         else return 1;
30.     }
31.     bool joystickInput(){
32.         int xvalue, yvalue;
33.         if(isBounce()){
34.             xvalue = analogRead(0);
35.             yvalue = analogRead(1);
36.             if (xvalue < 250) opt = up;
37.             else if (xvalue > 750) opt = down;
38.             else if (yvalue < 250) opt = right;
39.             else if (yvalue > 750) opt = left;
40.             else opt = middle;
```

```
41.         }
42.         if(opt && bounced) return 1;
43.         else return 0;
44.     }
45. public:
46.     OPTION():opt(middle), bounced(1) {}
47.     oren getOpt(){
48.         while(!joystickInput());
49.         return opt;
50.     }
51. };
52. #endif
53. /*************************FILE END**************************/
```

6. Money.h

```
1.  /************************FILE BEGIN***************************/
2.  /*
3.   * Money.h - balance calculate
4.   * Creator: Han.Sketchpink
5.   * Time: 2015.10.1
6.   */
7.  #ifndef MONEY_H
8.  #define MONEY_H
9.  #define isdigit(char) ((char) >= '0' && (char) <= '9')
10.
11. class MONEY {
12. private:
13.     unsigned char cardBuffer[4];
14.     unsigned char serialBuffer[34];     //Arduino serial IN buffer is
    64 byte, but 4 bytes double variable needs 34 char at most
15.     double balance;
16.     double atof(unsigned char s[]){
17.         double val, power;
18.         int i;
19.         for (i = 0; !isdigit(s[i]); i++);
20.         for (val = 0.0; isdigit(s[i]); i++)
21.             val = 10.0 * val + (s[i] - '0');
22.         if (s[i] == '.') i++;
23.         for (power = 1.0; isdigit(s[i]); i++){
24.             val = 10.0 * val + (s[i] - '0');
25.             power *= 10.0;
26.         }
27.         return val / power;
28.     }
29.
30.     inline double byte2Double(unsigned char s[]){
31.         unsigned char c[] = {s[0], s[1], s[2], s[3]};
32.         return *((double*)c);
33.     }
34.     inline unsigned char double2Byte(double d, int index){
35.         return ((unsigned char*)&d)[index];
36.     }
37. public:
38.     MONEY(unsigned char cb[], unsigned char sb[]){
39.         for (int i = 0; i < 4; i++) cardBuffer[i] = cb[i];
```

```
40.         for (int i = 0; sb[i]; i++) serialBuffer[i] = sb[i];
41.         //initialize with card money
42.         balance = byte2Double(cardBuffer);
43.     }
44.     MONEY(unsigned char cb[]){
45.         for (int i = 0; i < 4; i++) cardBuffer[i] = cb[i];
46.         balance = byte2Double(cardBuffer);
47.     }
48.     double plus(){
49.         balance += atof(serialBuffer);          //plus
50.         return balance;
51.     }
52.     double minus(){
53.         balance -= atof(serialBuffer);
54.         //BUSINESS should control if balance was below 0
55.         return balance;
56.     }
57.     double getBalance(){
58.         return balance;
59.     }
60.     unsigned char* getResultBuffer(){
61.         unsigned char tempbuffer[4];
62.         unsigned char* p = tempbuffer;
63.         for (int i = 0; i < 4; i++)
64.            tempbuffer[i] = double2Byte(balance, i);
65.         return p;
66.     }
67. };
68. #endif
69. /************************FILE END************************/
```

7. CreateUI.h

```
1.  /************************FILE BEGIN**************************/
2.  /*
3.   * CreateUI.h - createUI by function
4.   * Creator: Han.Sketchpink
5.   * Time: 2015.10.1
6.   */
7.
8.  #ifndef CreateUI_H
9.  #define CreateUI_H
10.
11. #define LEN 40
12.
13. class CreateUI{
14. public:
15.     static void titleLine(String s){
16.         unsigned char half = (LEN - s.length()) / 2;
17.
18.         for (int i = 1; i <= half; i++) Serial.print('*');
19.         Serial.print(s);
20.         if(s.length() % 2){
21.             for (int i = 1; i <= half + 1; i++) Serial.print('*');
22.         } else {
23.             for (int i = 1; i <= half; i++) Serial.print('*');
24.         }
25.         Serial.println();
26.     }
27.     static void endLine(){
28.         for (int i = 1; i <= LEN; i++)  Serial.print('*');
29.         Serial.println();
30.     }
31.     static void menu(String up, String left, String right, String
    down){
32.         Serial.println((String)"1. up    -- " + up);
33.         Serial.println((String)"2. left  -- " + left);
34.         Serial.println((String)"3. right -- " + right);
35.         Serial.println((String)"4. down  -- " + down);
36.     }
37. };
38. #endif
39. /************************FILE END**************************/
```

附录 C 校园一卡通系统业务源代码

1. Business.h

```
1.  /************************FILE BEGIN***************************/
2.  /*
3.   * Business.h - registration, destory, consume, deposit, query, flush,
    printDB, count
4.   * Creator: Han.Sketchpink
5.   * Time: 2015.10.1
6.   */
7.  #ifndef BUSINESS_H
8.  #define BUSINESS_H
9.
10. #include <RFID.h>
11. #include <SectorKey.h>
12. #include <Storage.h>
13. #include <Money.h>
14. #include <CreateUI.h>
15.
16. class BUSINESS{
17.
18. public:
19.     static bool registration(){
20.         bool status;
21.         unsigned char indexArray[16] = {0, 'X', 'X', 'G', 'C', 'X'};
22.         unsigned char balanceArray[16] = {0};
23.
24.         RFID rfid(10, 5);
25.
26.         rfid.init();
27.         rfid.isCard();
28.         status = rfid.readCardSerial();
29.         rfid.selectTag(rfid.serNum);
30.
31.         if (status) {
32.             Data data;
33.             for(int i = 0; i < 5; i++){
34.                 data.serNum[i] = rfid.serNum[i];
35.             }
36.             data.status = 1;
37.
```

```
38.           STORAGE storage(data);
39.           data.index = storage.create();
40.
41.           if(data.index) {
42.
43.               indexArray[0] = data.index;
44.               if (rfid.auth(PICC_AUTHENT1A, SECTORKEY,
    sectorKeyA[1], rfid.serNum) == MI_OK) {
45.                   //16 byte all write in
46.                   rfid.write(INDEXBLOCK, indexArray);
47.                   rfid.write(BALANCEBLOCK, balanceArray);
48.               }
49.               rfid.halt();
50.               return 1;
51.           } else {
52.               rfid.halt();
53.               return 0;
54.           }
55.       } else {
56.           rfid.halt();
57.           return 0;
58.       }
59.   }
60.   static bool destory(){
61.       bool status;
62.       unsigned char indexArray[16] = {0, 'X', 'X', 'G', 'C', 'X'};
63.
64.       RFID rfid(10, 5);
65.
66.       rfid.init();
67.       rfid.isCard();
68.       status = rfid.readCardSerial();
69.       rfid.selectTag(rfid.serNum);
70.       if (rfid.auth(PICC_AUTHENT1A, SECTORKEY, sectorKeyA[1],
    rfid.serNum) == MI_OK) {
71.           rfid.read(INDEXBLOCK, indexArray);
72.       }
73.       if (status) {
74.           Data data(indexArray[0]);   //construct with index
75.           STORAGE storage(data);
76.           if (!storage.read()) {
77.               rfid.halt();
78.               return 0;
79.           } else {
```

```
80.                 storage.writeIndex(0);
81.             //clear the card index, 64 > 51, as a clear mark, ASCII '@'
82.                 indexArray[0] = 64;
83.                 if (rfid.auth(PICC_AUTHENT1A, SECTORKEY,
    sectorKeyA[1], rfid.serNum) == MI_OK) {
84.                     rfid.write(INDEXBLOCK, indexArray);
85.                 }
86.                 rfid.halt();
87.                 return 1;
88.             }
89.         } else {
90.             rfid.halt();
91.             return 0;
92.         }
93.     }
94.     static void query(){
95.         bool status;
96.         unsigned char indexArray[16] = {0, 'X', 'X', 'G', 'C', 'X'};
97.         unsigned char balanceArray[16] = {0};
98.
99.         RFID rfid(10, 5);
100.        CreateUI::titleLine("QUERY");
101.        rfid.init();
102.        rfid.isCard();
103.        rfid.readCardSerial();
104.        rfid.selectTag(rfid.serNum);
105.        if (rfid.auth(PICC_AUTHENT1A, SECTORKEY, sectorKeyA[1],
    rfid.serNum) == MI_OK) {
106.            rfid.read(INDEXBLOCK, indexArray);
107.            rfid.read(BALANCEBLOCK, balanceArray);
108.
109.        }
110.        rfid.halt();
111.        Data data(indexArray[0]);
112.        STORAGE storage(data);
113.
114.        if (!storage.read()) {
115.            CreateUI::titleLine("No record in DataBase.");
116.            CreateUI::endLine();
117.            Serial.println();
118.            return;
119.        }
120.        MONEY money(balanceArray);
121.        Serial.print("1. Card serNum is: ");
```

```
122.        //include validate byte
123.        for(int i = 0; i < 5; i++) Serial.print(data.serNum[i], HEX);
124.        Serial.println("");
125.        Serial.print("2. Card index is: ");
126.        Serial.println(data.index);
127.        Serial.print("3. Reserve info is: ");
128.        Serial.print((char*)data.reserve);
129.        Serial.println("");
130.        Serial.print("4. Card status is:  ");
131.        Serial.println(data.status);
132.        Serial.print("5. Card balance is:  ");
133.        Serial.println(money.getBalance());
134.        CreateUI::endLine();
135.        Serial.println();
136.        Serial.flush();
137.    }
138.    static void deposit(){
139.        bool status;         //not for return value
140.        unsigned char indexArray[16] = {0, 'X', 'X', 'G', 'C', 'X'};
141.        unsigned char balanceArray[16] = {0};
142.        unsigned char serialArray[34] = {0};   //Arduino serial IN
    buffer is 64 byte, but 4 bytes double variable needs 34 char at most
143.        double balance;
144.        unsigned char* result;
145.        CreateUI::titleLine("DEPOSIT");
146.        Serial.println("HINT: Please attach the card to the Reader.");
147.
148.        RFID rfid(10, 5);
149.
150.        rfid.init();  //for MFRC522, not for card procedure
151.        rfid.isCard();
152.        rfid.readCardSerial();
153.        rfid.selectTag(rfid.serNum);
154.        if (rfid.auth(PICC_AUTHENT1A, SECTORKEY, sectorKeyA[1],
    rfid.serNum) == MI_OK) {
155.            rfid.read(INDEXBLOCK, indexArray);
156.            rfid.read(BALANCEBLOCK, balanceArray);
157.        }
158.        rfid.halt();
159.
160.        Data data(indexArray[0]);
161.        STORAGE storage(data);
162.
163.        if (!storage.read()) {
```

```
164.          CreateUI::titleLine("No record in DataBase.");
165.          CreateUI::endLine();
166.          Serial.println();
167.          return;
168.      }
169.      //moneyQuery for query balance
170.      MONEY moneyQuery(balanceArray);
171.
172.      Serial.print("Card recognized. Card balance is:  ");
173.      Serial.println(moneyQuery.getBalance());
174.      Serial.println();
175.      Serial.flush();
176.
177.      while (!Serial.available());
178.      unsigned char* p = serialArray;
179.      while (Serial.available() > 0){
180.          *p++ = (unsigned char)Serial.read();
181.          delay(2);
182.      }
183.      MONEY moneyCal(balanceArray, serialArray);
184.      //return in float, for read, possible
185.      balance = moneyCal.plus();
186.      //return in Array, for transform
187.      result = moneyCal.getResultBuffer();
188.
189.      for (int i = 0; i < 4; i++, result++) balanceArray[i] = *result;
190.
191.      rfid.init();
192.      rfid.isCard();       //AGAIN!
193.      status = rfid.readCardSerial();
194.      rfid.selectTag(rfid.serNum);
195.      if (rfid.auth(PICC_AUTHENT1A, SECTORKEY, sectorKeyA[1],
    rfid.serNum) == MI_OK) {
196.          rfid.write(BALANCEBLOCK, balanceArray);
197.      }
198.
199.      if(status){
200.          Serial.print("Deposit Success! Current balance is:  ");
201.          Serial.println(balance);
202.          CreateUI::endLine();
203.          Serial.println();
204.
205.          rfid.halt();
206.
```

```
207.        } else {
208.            Serial.println("WARNING:Communication Failed!");
209.            CreateUI::endLine();
210.            Serial.println();
211.            rfid.halt();
212.        }
213.
214.    }
215.    static void consume(){
216.        bool status;         //not for return value
217.        unsigned char indexArray[16] = {0, 'X', 'X', 'G', 'C', 'X'};
218.        unsigned char balanceArray[16] = {0};
219.        //Arduino's IN serial buffer is 64 Byte
220.        unsigned char serialArray[64] = {0};
221.        double balance;
222.        unsigned char* result;
223.
224.        CreateUI::titleLine("CONSUME");
225.        Serial.println("HINT: Please attach the card to the Reader.");
226.
227.        RFID rfid(10, 5);
228.
229.        rfid.init();      //for MFRC522, not for card procedure
230.        rfid.isCard();
231.        rfid.readCardSerial();
232.        rfid.selectTag(rfid.serNum);
233.        if (rfid.auth(PICC_AUTHENT1A, SECTORKEY, sectorKeyA[1],
    rfid.serNum) == MI_OK) {
234.            rfid.read(INDEXBLOCK, indexArray);
235.            rfid.read(BALANCEBLOCK, balanceArray);
236.        }
237.        rfid.halt();
238.
239.        Data data(indexArray[0]);
240.        STORAGE storage(data);
241.
242.        if (!storage.read()) {
243.            CreateUI::titleLine("No record in DataBase.");
244.            CreateUI::endLine();
245.            Serial.println();
246.            return;
247.        }
248.        //moneyQuery for query balance
249.        MONEY moneyQuery(balanceArray);
```

```
250.        Serial.print("Card recognized.Card balance is:  ");
251.        Serial.println(moneyQuery.getBalance());
252.        Serial.println();
253.        Serial.flush();
254.
255.        while (!Serial.available());
256.        unsigned char* p = serialArray;
257.        while (Serial.available() > 0){
258.            *p++ = (unsigned char)Serial.read();
259.            delay(2);
260.        }
261.        MONEY moneyCal(balanceArray, serialArray);
262.        balance = moneyCal.minus();
263.        if(balance < 0) {
264.            CreateUI::titleLine("No enough money!");
265.            CreateUI::endLine();
266.            Serial.println();
267.            return;
268.        }
269.        //return in Array, for transform
270.        result = moneyCal.getResultBuffer();
271.
272.        for (int i = 0; i < 4; i++, result++) balanceArray[i] = *result;
273.
274.        rfid.init();
275.        rfid.isCard();                              //AGAIN!
276.        status = rfid.readCardSerial();
277.        rfid.selectTag(rfid.serNum);
278.        if (rfid.auth(PICC_AUTHENT1A, SECTORKEY, sectorKeyA[1],
    rfid.serNum) == MI_OK) {
279.            rfid.write(BALANCEBLOCK, balanceArray);
280.
281.        }
282.        if(status){
283.            Serial.print("Consume Success! Current balance is:  ");
284.            Serial.println(balance);
285.            CreateUI::endLine();
286.            Serial.println();
287.            rfid.halt();
288.        } else {
289.            Serial.println("WARNING:Communication Failed!");
290.
291.            CreateUI::endLine();
292.            Serial.println();
```

```
293.          rfid.halt();
294.
295.      }
296.
297.  }
298.  static void flushDB(){
299.      STORAGE::flushDataBase();
300.      CreateUI::titleLine("Data Flushed.");
301.      Serial.println();
302.  }
303.  static void printDB(){
304.      CreateUI::titleLine("DATABASE CONTENT");
305.      STORAGE::printDataBase();
306.  }
307.  static void counter(){
308.      Serial.print("The record number in DataBase is: ");
309.      Serial.println(STORAGE::countDataBase());
310.      Serial.println();
311.  }
312.
313.  };
314.  #endif
315.  /************************FILE END************************/
```

2. MenuList.h

```
1.  /************************FILE BEGIN***************************/
2.  /*
3.   * MenuList.h - MenuList
4.   * Creator: Han.Sketchpink
5.   * Time: 2015.10.1
6.   */
7.
8.  #ifndef MenuList_H
9.  #define MenuList_H
10.
11. #include <Business.h>
12. #include <Option.h>
13. // extern from ino project file, not included cpp files
14. extern OPTION option;
15. extern oren o;
```

```
16.
17. class UI{
18. public:
19.     static void Management(){
20.         bool loop = 1;
21.
22.         while(loop){
23.             CreateUI::titleLine("MANAGEMENT");
24.             CreateUI::menu("none", "Registration", "Destory",
    "Return to Menu");
25.             CreateUI::endLine();
26.             Serial.println();
27.             Serial.flush();
28.
29.             o = option.getOpt();
30.             switch(o){
31.                 case up:
32.                     loop = 0;
33.                     break;
34.
35.                 case left:
36.                     if(BUSINESS::registration()) {
37.                         Serial.println("Registration SUCCESS!");
38.                         BUSINESS::query();
39.                     }
40.                     else Serial.println("Registration FAILED, make
    sure it's a NEW card.");
41.                     Serial.println();
42.                     break;
43.
44.                 case right:
45.                     if(BUSINESS::destory()) {
46.                         Serial.println("Destory card SUCCESS! Clear
    record in DataBase.");
47.                     }
48.                     else Serial.println("Destory FAILED, no record in
    DataBase.");
49.                     Serial.println();
50.                     break;
51.
52.                 case down:
53.                     loop = 0;
54.                     break;
55.             }
```

```
56.         }
57.     }
58.     static void Consumption(){
59.         bool loop = 1;
60.         while(loop){
61.             CreateUI::titleLine("CONSUMPTION");
62.             CreateUI::menu("Query", "Deposit", "Consume", "Return to
    Menu");
63.             CreateUI::endLine();
64.             Serial.println();
65.             Serial.flush();
66.
67.             o = option.getOpt();
68.             switch(o){
69.                 case up: BUSINESS::query(); break;
70.                 case left: BUSINESS::deposit(); break;
71.                 case right: BUSINESS::consume(); break;
72.                 case down: loop = 0; break;
73.             }
74.         }
75.     }
76.     static void Database(){
77.         bool loop = 1;
78.         while(loop){
79.             CreateUI::titleLine("DATABASE");
80.             CreateUI::menu("Flush Database", "Print DataBase", "Count
    Record", "Return to Menu");
81.             CreateUI::endLine();
82.             Serial.println();
83.             Serial.flush();
84.
85.             o = option.getOpt();
86.             switch(o){
87.                 case up: BUSINESS::flushDB(); break;
88.                 case left: BUSINESS::printDB(); break;
89.                 case right: BUSINESS::counter(); break;
90.                 case down: loop = 0; break;
91.             }
92.         }
93.     }
94. };
95. #endif
96. /************************FILE END************************/
```