Computer Science and Application 计算机科学与应用, 2013, 3, 102-109
http://dx.doi.org/10.12677/csa.2013.32018 Published Online April 2013 (http://www.hanspub.org/journal/csa.html)

An EEPROM Defragmentation Algorithm on Java Smart Card

Wen Xiang, Chong Lin, Chuanguang Xiong

Engineering Research Center for Software and Embedded Systems, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan
Email: xw@whty.com.cn, linch_1988@163.com, xcg@whty.com.cn

Received: Feb. 17th, 2013; revised: Mar. 1st, 2013; accepted: Mar. 19th, 2013

Abstract: Java smart card is widely used in the highly security fields such as bank, social security etc. Because the hardware resources of the smart card are limited, and especially the multi-application smart card is popular, the card's storage resources have been paid more and more attention. The implementations and related mechanisms of Java Card virtual machine are also seriously affecting the efficient of the card. This paper analyzes the management methods that Sun Inc. has given of the Java Card's EEPROM, finds the disadvantages of the existing methods, redesigns the EEPROM structure, and at last designs a new management algorithm that include EEPROM's defragmentation.

Keywords: Java Smart Card; EEPROM; Storage Management; Defragmentation

Java 智能卡 EEPROM 碎片整理算法

向　文，林　冲，熊传光

华中科技大学计算机科学与技术学院软件与嵌入式系统工程研究中心，武汉
Email: xw@whty.com.cn, linch_1988@163.com, xcg@whty.com.cn

收稿日期：2013 年 2 月 17 日；修回日期：2013 年 3 月 1 日；录用日期：2013 年 3 月 19 日

摘　要：Java 智能卡已经被广泛应用于金融、通信等安全性要求很高的领域。但是受限于当前智能卡的硬件资源，特别是多应用智能卡的流行，卡片的存储资源越来越受到重视，Java 卡虚拟机的具体实现方式和相关机制也严重影响了卡片的执行效率。通过分析 Sun 公司 Java 卡规范算法，总结出现有算法的一些缺陷，对 EEPROM 的结构重新设计，提出一种带碎片整理的 EEPROM 存储管理算法。

关键词：Java 智能卡；EEPROM；存储管理；碎片整理

1. 引言

Java 智能卡技术是 Java 技术在智能卡领域的扩展，它给传统的智能卡应用带来变革，提高了发卡商或服务提供商选择智能卡的自主性。虽然 Java 智能卡具备众多优点并且取得了广泛的应用，但目前的 Java 智能卡技术还存在一些需要改进的地方，特别是其较低的执行效率。造成这种低效的主要原因如下：其一智能卡有限的硬件资源，但是随着半导体技术飞快发展，卡片上承载的硬件性能也越来越高，因此卡片硬件资源的影响会越来越不明显；其二 Java 卡智能卡上字节码的解释执行效率，Java 卡片的虚拟机都是 Java 语言编写的，但是 Java 字节码的执行速度比 C 语言是慢一些的；其三 Java 卡虚拟机自己本身的一些实现机制不完善，对 Java 智能卡的执行效率也有较大的影

响[1]。

2. Java 卡的 EEPROM 结构

Java Card 系统中，ROM 主要用于存储 JCRE 代码和 Applet 代码。RAM 作为临时存储器来使用，RAM 主要用于存储堆数据。还有另外一些数据也是存放在 RAM 上的，如一些 Native 方法的中间结果。在 EEPROM 中存放的是永久数据和下载来的 Applet 类[2]。

在 Java Card 设备上，内存是极为稀缺的资源，因此传统的 Java 卡将所有用 new 操作符创建的应用实例对象和相关的数据都存储在 EEPROM 中。在如今的 Java Card 技术中，RAM 用于方法的调用和存储临时数据[3]，另一方面，断电后仍需保存的永久数据需要保存在 EEPROM 中[4]。

如图 1 所示为 Java 卡 EEPROM 内部结构，其主要由六部分组成：

1) Applet Area 存储下载至 EEPROM 中的 Applet 类和相关信息；

2) 静态字段区(Static Field Area)存储每个 Java package 中类的静态字段，它们在 Applet 编写时声明和定义；

3) 持久对象表区，表中存储的指针值指向持久对象的指针值，这些指针指向的对象都存储在 EEPROM 上；

4) 堆区(Heap Area)存储 Java 卡对象，包括持久对象和暂态对象引用；

5) 暂态对象表区，表中存储的指针值为暂态对象的指针值，这写指针指向的对象都存储在 RAM 上；

6) 事务缓冲区，用来临时存放在 EEPROM 写操作过程中被更新的原始数据，以实现中断时的原始数据回滚，保证 EEPROM 所存储的数据的原子性。

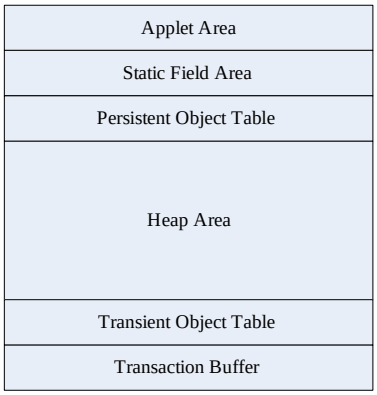

Figure 1. The internal structure of EEPROM
图 1. EEPROM 内部结构

在 Java 智能卡的存储管理中，EEPROM 内部采用的是分页机制，其页面大小是由卡片开发商决定的，一般为每页 128~256 字节，通常采用字节流的写入方式。

3. Sun 公司 Java 卡参考实现 EEPROM 管理算法分析

Sun 公司 2006 年给出一种 Java 卡规范，它提供了一种具体的 Java 卡实施方案。该方案构架中位于底层的 COS 主要负责对智能卡 RAM、EEPROM、串口及中断等硬件资源的管理；JCRE 位于中间层，主要用于维护 Java 卡系统运行时的环境；JCVM 则主要完成 Java 语言字节码的解析及执行任务[5]。在参考实现中的 EEPROM 管理算法主要包括 EEPROM 的空间分配和空间释放。

Java 卡创建对象时，要为其分配 EEPROM 空间。Sun 公司的 Java 卡参考实现中，EEPROM 的空间分配采用的是最佳匹配原则，即是在分配的过程中，对所有的空闲空间进行遍历，找到存储空间最合适，甚至是完全匹配的空间分配，这样使得产生的碎片最小化。为了寻求最佳的匹配块，在每次空间分配的时候都需要对所有的空闲块进行遍历，在找到最佳匹配块后，需要更新空闲链表中的记录情况，EEPROM 空间分配算法在时间开销上要求是比较高的。

Java 卡中一块空间不再被使用时，需要被释放使得被占用的空间能够被重新利用，提高存储效率。Sun 公司给出的 EEPROM 空间回收方式是：待释放空间的数据被清除后，需要将此空间与前后相邻的空闲块整合在一起，成为一块更大的内存空间。当确定一块 EEPROM 空间需要被释放时，JCVM 会删除此空间的对象数据，然后 EEPROM 管理器会将此空间的信息写入到相应的段表中。

从 Sun 公司 Java 卡规范提供的 EEPROM 管理算法可知：在空间分配的过程中，采用了空间最佳匹配原则，也就是需要分配与需求空间大小最接近甚至是完全相等的空间，但每次分配的空间都相等是完全不可能的，因此会导致存储碎片产生。并且在空间分配的时候，由于每次都对空闲存储空间进行遍历，因此

对存储的性能也会产生比较大的影响。

在 EEPROM 的释放过程中，Sun 公司 Java 卡规范算法也考虑相邻空闲空间的合并，但是 EEPROM 中还是可能存在一些永久的数据将这些空间分割得七零八碎，从而导致虽然 EEPROM 总空间满足分配的需求，但无法提供足够的连续存储空间。这是现有 EEPROM 管理算法的一个重大缺陷，给只拥有有限资源的 JAVA 卡片设备带来了一个严重的效率瓶颈。

Java 智能卡是一种存储资源稀缺的设备，随着智能卡的广泛应用，以及一卡多用的现实需求[6]，虽然卡片内允许的存储空间越来越大，但还是不能满足需求的增长，因此如何在稀缺的卡片资源中更加高效、更加合理的使用存储空间成为了一个很重要的考虑方向。

4. Java 卡 EEPROM 碎片整理算法

Java 卡在 EEPROM 最初的结构设计上没有涉及到其碎片整理，为了能够更加高效、合理的对碎片进行整理，本文对 EEPROM 的结构进行重新设计，对 EEPROM 的管理方法也进行了改进。

4.1. EEPROM 存储结构设计

为了利于 EEPROM 管理算法的实现，在 Sun 公司 Java 卡规范算法基础上对 EEPROM 的结构设计做了相应的改进，使得其更利于带有碎片整理的 EEPROM 管理。在 EEPROM 的结构中，如图 2 所示设置了一些标志位，例如补丁的起始位置标志、事务完成标志、回收防拔存储位等。

EEP_RESERVED：COS 的预留区，从 EEPROM 的首部开始。其中有一块 C_EEP_PATCH_BEGIN(3B) 在 COS 预留区的底部，为补丁的起始地址，如果没有补丁，此处为 EEPROM 尾地址+1。最下面有个 C_EEP_APP_END(1B)，即把 COS 预留区设置为事务完成标志。

EEP_STATE_TABLE：是 EEPROM 状态表的存储区域。eep_state_table 表示页内 16 字节块空闲情况，某一位为 0：对应 16 字节块空闲，某一位为 1：对应 16 字节块已使用。

EEP_STATUS：是 EEPROM 的使用状况。

EEP_POWERDOWN：是 EEPROM 的 EEPROM 回收防拔位存储区域，0x55 表示防拔开始；0x0 表示防拔完成。EEP_USER 为 EEPROM 的用户数据存储区域。

4.2. EEPROM 空间分配算法

在 EEPROM 空间分配开始之前，可能并不知道

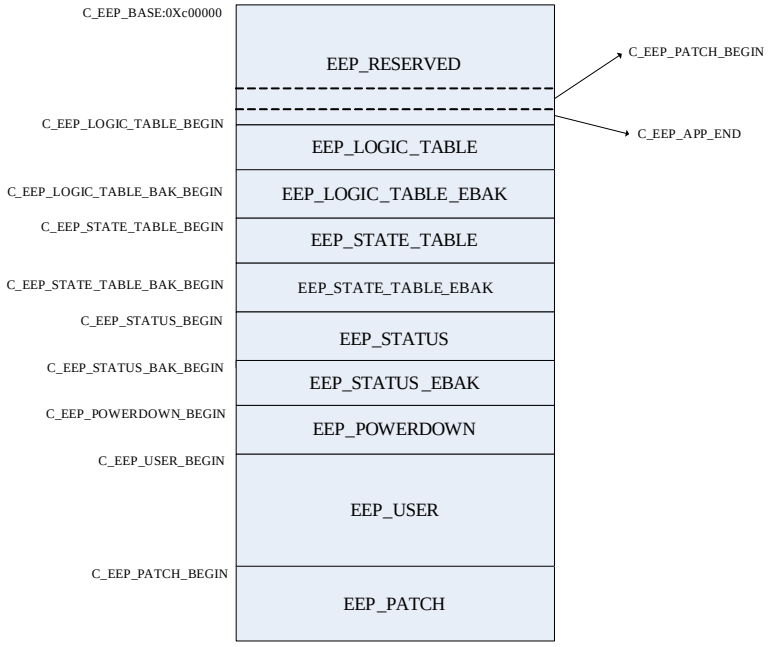

Figure 2. The storage structure of EEPROM
图 2. EEPROM 存储结构设计图

对象所需存储空间的大小，在未知对象长度的情况下，只需要给其就近分配一块存储空间即可，即先分配预留空间。在已知对象长度的情况下，若对象长度小于页大小，则优先分配碎片。若对象长度大于页大小，则需要找到满足需要的整页空闲页，且后续空间能够满足整页存储后的零碎空间。

EEPROM 空间分配采用就近分配法则，在EEPROM 管理算法中有碎片整理的功能，采用就近分配方法产生的存储碎片不会对 EEPROM 的性能产生很大的影响，相反，就近分配法能很好的缩短了空间分配的时间，提高分配效率。EEPROM 的空间分配算法主要分成两种情况，未知对象长度的情况，该情况常见在 APP 安装的时候(也即首次分配空间的时候)，其分配空间需要找到一个空闲块即可，不需要考虑块大小是否合适。当 APP 运行时会产生堆栈数据，这时对象长度就是已知的，此时 EEPROM 分配空间算法如下：

TEP1：EEPROM 空间分配算法启动，首先备份 eep_logic_table、eep_state_table、eep_status 几个表；

TEP2：需要分配的空间与 PAGE_SIZE 比较，如果小于 FAGE_SIZE 的，直接找碎片，找碎片的算法接下来会详述。如果大于 PAGE_SIZE，即是需要整页分配，就计算需要的整页数和零碎的空间，然后从下而上查找第一个空闲整页；

TEP3：判断此页是否大于 eep_maxpages；

TEP4：判断整页数是否足够，如果不够则继续判断连续的物理页面是够为整空闲页，直到整页足够未知，整页足够之后再判断连续的物理空闲空间是否满足零碎的空间，其中有任意一处的连续物理空间不满足条件，此处的空间都不满足，跳转至 TEP3。如若连续的空间够分配，查找分配空间成功；

TEP5：分配当前页，保存逻辑页表，记录页内偏移，分配空间成功。

从上述的分配算法中可知：在需要分配的空间小于 PAGE_SIZE 时，优先分配碎片，碎片的分配也是采用最快分配原则。查到空闲区后，继续看有没有连续的空闲区，用连续的空闲区的大小与需要分配的空间比较，如果能够满足，就直接分配此空间，如果不满足，继续往下查找，全部找过一边后，如果没有找到合适的分配空间，分配就会失败，返回 NULL。成功查找到满足的空闲区后，就分配空间，并修改逻辑页表，记录页内偏移。

由于采用就近分配原则，只要空闲空间大小能满足即分配，没有考虑最匹配问题，因此会更容易产生碎片。但通过碎片整理后，可以有效的规避这个缺陷，也因此提高了空间分配的效率。在需要分配的空间小于 PAGE_SIZE 的时候，算法中也是优先分配碎片，可以有效的减少碎片占用存储空间。在上述些分配法则的有效结合下，可以得到一种高效的 EEPROM 分配算法。

4.3. EEPROM 空间释放算法

Java 卡的存储空间非常有限，如果没有有效的存储空间释放机制，空间会很快消耗完，EEPROM 也不例外。有效的 EEPROM 空间释放算法会给后续的操作带来很多方便。

EEPROM 的空间释放是指定对象，指定长度的空间释放，必须知道待释放对象的逻辑页表序号 logic_index、页内偏移 offset 及对象长度。指定的对象所在物理页可能在一页，也可能出现跨页的情况。

图 4(a)所示是对象仅在一个物理页内，但不在页的首部且有页内偏移，只需要找到指定的空间，修改 eep_stats_table 当前页的使用状态。图 4(b)所示是所需要释放的空间跨页且有页内偏移，需要找到指定的空间，修改 eep_stats_table 此页的使用状态，并将当前页的指针指到下一页，然后再修改当前页。图 4(c)所示是待释放的空间在页的首部，并且需要整页释放，整页释放的时候，需要在 eep_logic_table 中擦除当前的逻辑页。

针对上述的需要释放的对象的不同情况，设计了如下的 EEPROM 空间释放的算法，EEPROM 空间释放的流程如图 3 所示：

TEP1：在做 EEPROM 空间释放的开始时，需要备份 eep_logic_table、eep_state_table、eep_status 几个存储表；

TEP2：根据待释放对象的逻辑页表序号 logic_index 找到指定的页，释放空间 16 字节对齐，不足 16 字节按 16 字节计算；

TEP3：判断释放的对象所在的空间是否有页内偏移；

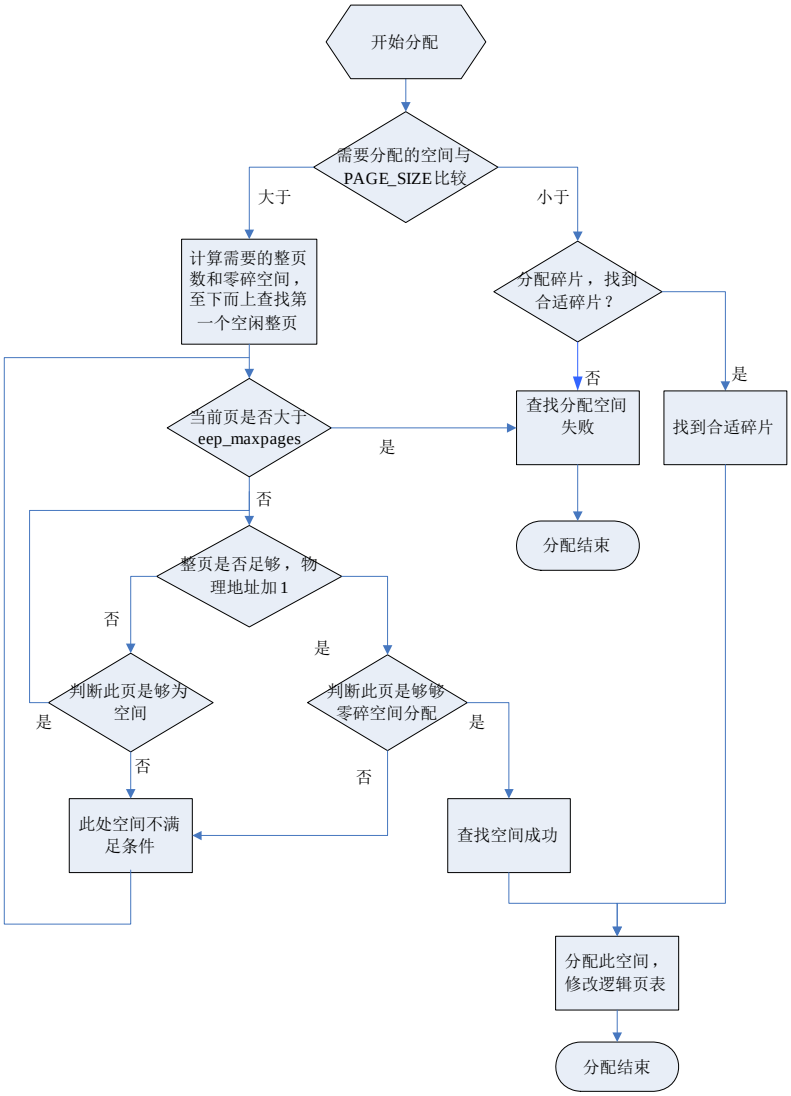

Figure 3. The algorithm flowchart of space allocation
图 3. EEPROM 空间分配算法流程图

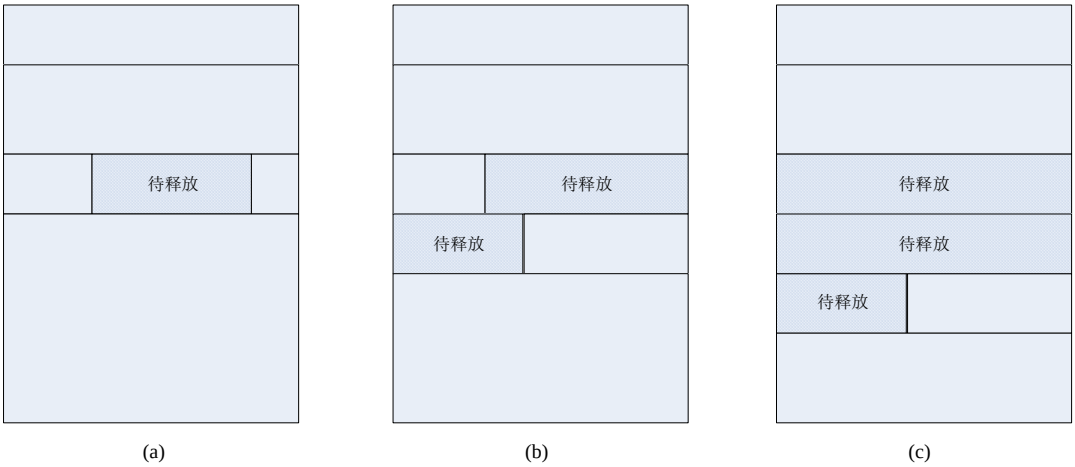

Figure 4. The schematic diagram of space release
图 4. EEPROM 空间释放示意图

TEP4：如若有偏移，判断释放的空间是否跨页，如果没有跨页情况，直接修改 eep_state_table 表中此页的使用情况，空间释放完成。如果有跨页的情况，修改 eep_state_table 表中此页的使用情况，并且当前页的指针指向下一页，进入 TEP5。如果没有偏移，先把前面的整页全部释放直至剩下的空间不大于 PAGE_SIZE，然后进入 TEP5；

TEP5：判断需要释放的当前指向的页是否为整页，如果为整页，需要修改 eep_state_table，并删除逻辑页表中的当前页。没有整页的时候，修改 eep_state_table 表中此页的使用情况，空间释放完成；

TEP6：最后需要保存修改的逻辑页表，从下向上找到第一个非空闲页，将非空闲页所对应的逻辑页表表项设置为当前页，如无非空闲页(EEP 空)，则将第一页设置为当前页。

从上述的空间释放的流程可以看出将会产生空间碎片，并且在此处没有对这些碎片做任何的处理。所以在 EEPROM 空间之后，分配之前，必须先调用碎片整理功能。

4.4. EEPROM 碎片整理算法

EEPROM 碎片整理的主要目的是将存储器的空闲空间集中到存储器的尾部，也可以称为空间紧凑，从而保证 EEPROM 提供最大的连续空闲块，EEPROM 的碎片整理算法流程如下：

TEP1：从上到下找到第一个空闲页(reclaim_new_physical_index)，如搜索到底部仍无空闲页则退出；

TEP2：从其下一页开始找到第一个非空闲页，如搜索到底部仍无非空闲页则退出；

TEP3：查找非空闲页所对应的逻辑页表表项(reclaim_logic_index)，将非空闲页复制到空闲页；

TEP4：修改 logic_table：将 g_eep_logic_table 对应页复制到 cache，修改 cache，用 cache 更新 g_eep_logic_table_bak 对应页；

TEP5：修改 state_table:state_table[free_page]对应页复制到 cache，修改 cache，用 cache 更新 state_table_bak[free_page]对应页，state_table[used_page]对应页复制到 cache，修改 cache，用 cache 更新 state_table_bak[used_page]对应页，如 state_table[free_page]和 state_table[used_page]在一页，则 6.4~6.6 合并到 6.1~6.3；

TEP6：记录防拨信息 reclaim_logic_index、reclaim_new_physical_index、reclaim_middle；

TEP7：将 g_eep_logic_table_bak 对应页复制到 g_eep_logic_table 对应页，将 state_table_bak[free_page]对应页复制到 state_table[free_page]对应页，将 state_table_bak[used_page]对应页复制到 state_table[used_page]对应页，如在一页，则合并完成；

TEP8：防拔完成，记录防拔信息：reclaim_end；

TEP9：记录指针 eep_reclaim_flag_index+1，如 eep_reclaim_flag_index==EEP_RECLAIM_MAX_INDEX，则将 eep_status 复制到 eep_status_bak(++index)，擦除 eep_status，eep_reclaim_index = 0；

TEP10：重复 TEP1~9 步，直至退出；

TEP11：从下向上找到第一个非空闲页，查找非空闲页所对应的逻辑页表表项，将其设置为当前页，如无非空闲页(EEP 空)，则将第一页设置为当前页。

根据上述的碎片整理流程，对于图 5A 碎片整理前的存储情况，首先找到第一个空闲页 P1，然后找到第一个非空闲页 P3，将 P3 页复制到 P1 页，并修改逻辑表中的 L1 序列，将其中的物理页面记录改为 P1，修改状态表，至此整理其中的第一页完成，如图 5B 所示。按此步骤一直整理，直至所有的存储碎片整理完毕，整理完成后的情况如图 5C 所示。从上图中可知：EEPROM 碎片整理后，空闲的大块空间全部集中到了存储器的底部，可以达到碎片整理的最终目标。

4.5. EEPROM 写操作掉电处理

由于 Java 智能卡的应用一般不针对单个的卡片系统，往往会涉及到一个大量的卡片用户群体，卡片数据的存储一致性对整个系统的影响也是不言而喻的。在对卡片做写操作的时候，掉电会对卡片的操作有很严重的后果，轻者丢失文件，严重时甚至使整个文件系统遭到破坏，造成系统崩溃。但是在卡片的操作过程中，掉电又是不可避免的情况，所以对卡片的存储操作过程做好掉电应急处理是一项重要并且必不可少的工作。

Java 智能卡 EEPROM 的 COS 预留区最后一个字节为事务完成标志，在对 EEPROM 的操作过程中，这个标志位就相当于对掉电起到了报警作用。在本 EEPROM 存储管理中，防拔掉电处理与此标志有很紧

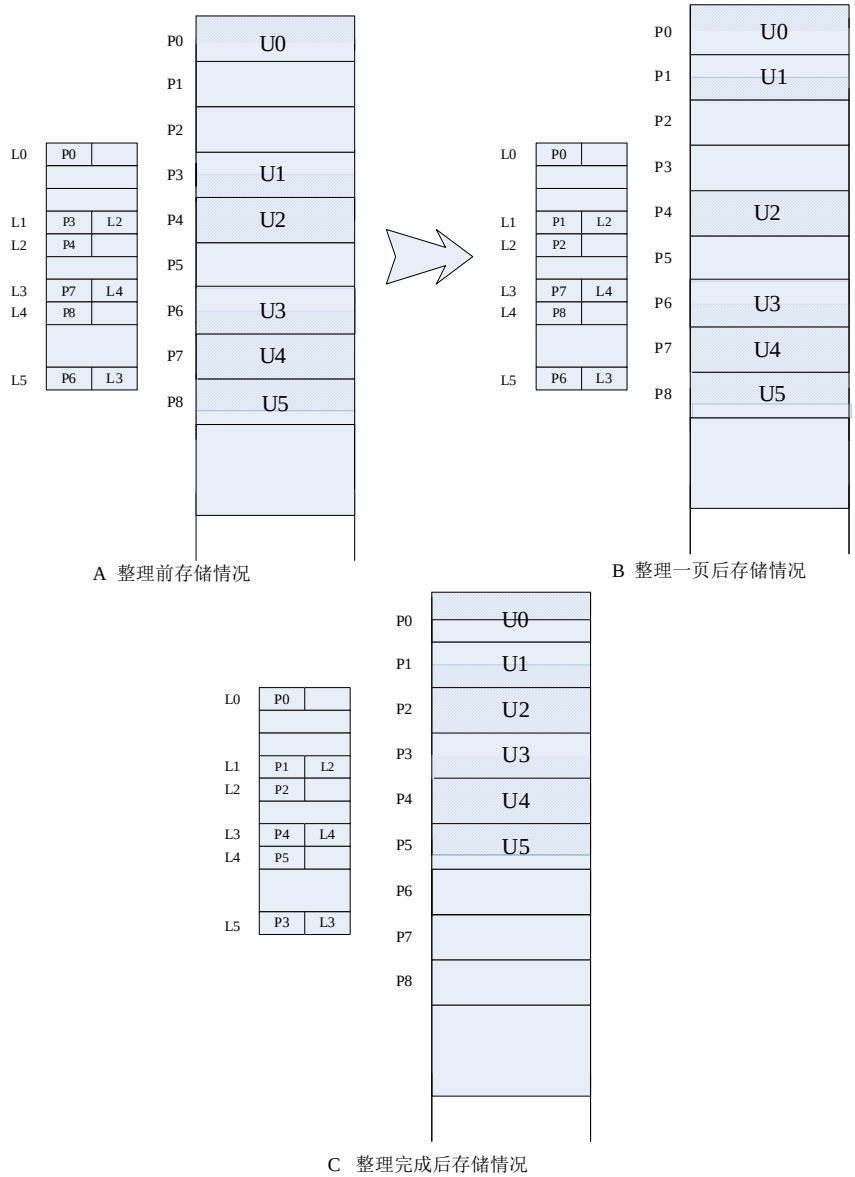

Figure 5. The schematic diagram of defragmentation
图 5. 碎片整理示意图

密的联系，在每次上电的时候，EEPROM 都会对 C_EEP_APP_END 这个标志检查。

5. EEPROM 碎片整理算法的实现及验证

本文主要研究的是 Java 卡 EEPROM 的管理算法，其中重点是在 EEPROM 的管理算法中加入了碎片整理，并且所有的操作都做了防拔掉电处理。但是由于 EEPROM 结构设计上的变化，EEPROM 在空间分配和释放等操作上也需要重新测试其性能，并且 EEPROM 各种操作在被拔掉电处理上的测试也是必不可少的。

上面所述的 eep_manager 的操作已经全部用 C 代码实现，下面给出其中主要数据结构和接口实现：

logic_table 数据结构如下：

```
typedef struct
{
    u16 physical_index;//逻辑页所对应的物理页号(page0，page1，...，)
```

```
                                    // 为了区分是否已使用，将最高位置"0"表示已使用
    u16 next_logic_index; // 链接逻辑页表项(逻辑页表序号)
}eep_logic_table_t;
state_table 数据结构如下：
typedef struct
{
    u16   free16; // 页内 16 字节块空闲标记，某一位为 1：对应 16 字节块空闲，某一位为 0：对应 16 字
                  节块已使用
}eep_state_table_t;
reclaim_flag 数据结构如下：
typedef struct
{
    u16 reclaim_logic_index; // 正在做回收处理的逻辑页表表项
    u16 reclaim_new_physical_index; // 正在做回收处理的新物理页页号
    u16 reclaim_used_physical_index; // 正在做回收处理的原物理页号
    u16 reclaim_middle; // 0xA55A 表示已完成物理页 copy 及 logic_table_bak 和 state_table_bak 修改
    u16 reclaim_end; // 回收操作防拔标志，0x5AA5 表示防拔完成
}eep_reclaim_flag_t;
```

本文分别对 EEPROM 空间分配、空间释放，碎片整理，碎片整理防拨做了相应的测试。由于测试项目繁多，下面仅就碎片整理后的 EEPROM 存储情况进行说明。

图 6 中是 EEPROM 在一系列空间的分配和释放后的使用情况表记录，图 7 中是经过碎片整理后的使用情况表记录，从上面可以看出在 EEPROM 的尾部的碎片被紧凑在一起。

6. 结束语

本文研究 Java 智能卡的 EEPROM 空间存储管理。提出了一种带碎片整理的 EEPROM 管理算法。主要从以下几个方面展开：

1) 改进 Java 卡 EEPROM 空间管理的数据结构，设置了相关的标志位，支持 EEPROM 使用情况记录表的备份存储，并预留补丁存储空间；

2) 基于改进的 EEPROM 结构设计了一种高效的存储空间管理算法，包括空间的分配和释放；

3) 设计并实现了一种高效的带防拔处理的 Java 卡 EEPROM 碎片整理算法。将 EEPROM 的空闲空间块整理紧凑到存储器的尾部，提供了最大化的连续存储空间，并能对被拔掉电做有效的处理。

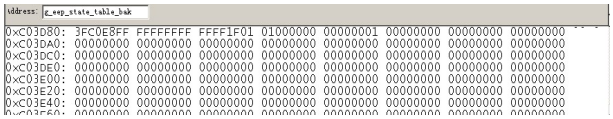

Figure 6. The state_table after space allocation and release
图 6. EEPROM 经过空间分配释放后的使用情况表

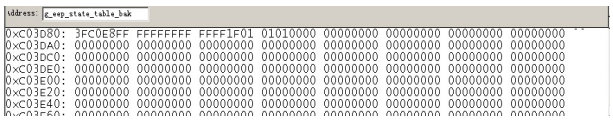

Figure 7. The state_table after defragmentation
图 7. EEPROM 经过碎片整理后的使用情况表

参考文献 (References)

[1] 常青, 靳伟, 李春龙等. JCVM 解析优化设计与实现[J]. 北京航空航天大学学报, 2004, 30(12): 1204-1207.

[2] Z. Q. Chen. Java card technology for smart cards: Architecture and programmer's guide. Boston: Addison-Wesley, 2000.

[3] M. Oestreicher, K. Ksheeradbhi. Object lifetimes in java card. Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology, Berkeley: USENIX Association, 1999: 129-137.

[4] M.-S. Jin, M.-S. Jung. A study on fast JCVM by moving object from EEPROM to RAM. Proceeding of the 11th IEEE International Conference on RTCSA, Hongkong, 17-19 August 2005: 502-507.

[5] Sun Microsystems, Inc. Virtual machine specification java card™ platform. Version 2.2.2. Santa Clara: Sun Microsystems, Inc., 2006: 23-28, 89-251.

[6] 曹计昌, 吴垣辉. 多应用智能卡自由存储区管理的研究[J]. 计算机工程与科学, 2007, 29(12): 147-150.

