JCVM 中栈与帧的研究和设计*

何利明,李代平,徐宏宁,谢晶晶,马海峰
(广东工业大学 计算机学院,广东 广州 510006)

摘 要:介绍了 Java Card 虚拟机的相关知识,并就其存储资源有限的特点,提出了一套有效的资源管理策略和一个符合规范且可行的栈与帧的结构设计方案,详细说明了该方案中栈与帧的执行过程。

关键词:Java 卡;Java 卡虚拟机;栈;帧;存储空间
中图分类号:TP391 **文献标识码**:A **文章编号**:1674-7720(2011)07-0004-03

The study and designment about stack and frame in JCVM

He Liming, Li Daiping, Xu Hongning, Xie Jingjing, Ma Haifeng
(College of Computer, Guangdong University of Technology, Guangzhou 510006, China)

Abstract: The article put emphasis on introducing relevant knowledge about JCVM and put forward a series of effective resource management strategies and a standard and practical physical design project about stack and frame, according to its limited storage resources. In addition, it illustrated the executing process of stack and frame in the project in detail.

Key words: Java card; Java card virtual machine; stack; frame; memory space

近年来,智能卡已经越来越广泛地应用于社会生产、生活的各个领域,相关技术也得到了迅猛发展。Java 技术具有安全、简单、即时编译和跨平台的众多优点,将 Java 技术运用于智能卡的开发领域,已经成为智能卡技术发展的热点。Java 卡是一种能运行 Java 应用程序的智能卡,支持部分 Java 编程语言,是 Java 技术和智能卡的一个成功结合。Java 卡技术的核心是 Java 卡虚拟机 JCVM (Java Card Virtual Machine)。不同于一般的 Java 虚拟机,由于受到智能卡存储与处理能力的限制,JCVM 的可用资源非常有限。通常 JCVM 对于内部资源的管理和使用非常严格。

1 JCVM 和相关技术简介

由于智能卡的可用资源非常有限,不可能在智能卡内实现 JCVM 的全部功能,所以,通常将 JCVM 分为卡外虚拟机和卡内虚拟机两部分。卡外虚拟机可运行于一般的 PC 或 SUN 工作站上,主要工作是将由 Java 语言编写的 class 文件转换成为字节码形式的 CAP 文件。而卡内虚拟机在智能卡内部实现,负责装载、执行字节码和支持 Java 语言。下面所说的 JCVM 主要是指卡内虚拟机部分。Java 卡应用程序的开发和其他 Java 应用程序的开发在最初阶段基本相同,开发者编写若干 Java 类源代码,利用 Java 编译器编译成类文。然后,将相应的类文件用 Java 卡转换器转换成为 CAP 文件,CAP 文件相比起类文件更加紧凑、短小,将其下载到卡内虚拟机。卡内虚拟机执行 CAP 文件中的代码,完成应用的安装,使应用处于能够被卡内虚拟机执行的状态。

对于 JCVM 的载体,Java 卡的系统结构如图 1 所示。

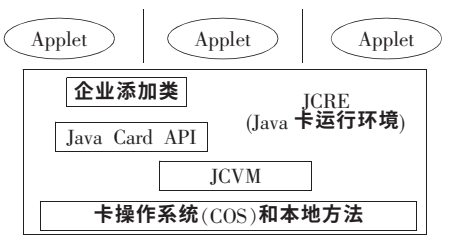

图 1 Java 卡的系统结构图

Java 卡系统主要由三个部分组成:COS 和本地方法层、JCRE、Java 卡应用程序。其中,JCRE 包括 JCVM、Java Card API 和企业添加类。

在系统结构图 1 中,位于最下层的 COS 和本地方法层主要用于对智能卡的硬件进行管理和操作。JCRE 是 Java 卡的核心部分,主要工作是维护 Java 卡系统运行时的环境,同时也负责应用的执行和安全。JCVM 是完成

* 基金项目:广东省广州市自然科技基金项目(2008-GX-015)

Java 程序字节码解析和执行的主要部分。最上面的是 Java 卡应用程序。Java 卡允许一卡多用的存在，当卡中有多个应用时，应用需要通过被选择，才能处于工作状态，其程序字节码才能通过 JCVM 进行解析和执行。

从图 1 所示可以明确得到 JCVM 在 Java 卡内所处的层次关系。JCVM 本身是 JCRE 的一部分。JCVM 通过调用 COS 提供的命令和一些本地方法，控制硬件的运算、存储等操作。JCVM 被各个 Applet 通过企业添加类和 API 调用，执行应用程序的字节码。

2 JCVM 实现难点

对于 JCVM 栈与帧结构的实现，主要难点可以归结为以下几点：

(1)智能卡因为其受到存储空间的限制，需要一套科学有效的管理和利用方案，以保证存储空间的高效使用。

(2)JCVM 中栈与帧的结构设计，需要做到既能高效地利用有限的空间，同时又能保证 Java 应用程序正常执行。

(3)在栈中，如何实现帧的创建和销毁，这并非只是简单地申请空间或释放空间的操作，而且还包括整个程序执行过程的相关动态链接、全局控制等数据的处理。

(4)如何符合规范地进行帧内部数据的处理。

下面将就这些实现难点进行分析，提出设计思路和解决方案。

3 JCVM 中存储空间的划分

因为受到智能卡平台的资源限制，JCVM 中的存储空间非常有限。为了能够科学节约地使用和管理这些空间，将 JCVM 的存储空间虚拟划分为以下部分：

(1)应用代码区：用于存储 CAP 文件通过安装器安装以后的中间代码，主要是方法字节码等。

(2)静态变量区：用于存储非易失性应用数据内容，一般是应用的域、应用处理的最终结果、方法调用过程中 Token 和方法地址的转换表、常量池等重要信息。应用可以通过相关的指令读写静态空间存储的各种内容。

(3)信息共享区：属于易失性存储区域，主要用于公共信息内容的交换，由 OS 负责提供，应用和终端均可访问 OS 中的内容。其主要存储 APDU 的内容，也存储栈运行过程中需要用到的一些临时变量。

(4)方法执行区：用于虚拟机执行过程中栈操作的空间，保存运行的执行数据、中间结果等。

应用代码区和静态变量区可以通过文件系统进行存储，保存在智能卡的 Flash 中。信息共享区和方法执行区一般通过物理地址读写，保存在智能卡的 RAM 中。

4 JCVM 中栈与帧的执行对象

由于具有方便移植、安全和程序代码小等优点，JCVM 中执行的程序都是以字节码的形式存在，而栈与帧的主要执行对象就是存放在应用代码区的 Java 卡应用程序方法字节码。方法字节码由 2 个或 4 个字节的方法头和之后若干长度的方法执行字节码组成。方法头中主要包含了 max_local，max_stack，narg 等信息，其中，max_local 指出了该方法需要申请用于局部变量区数组的空间大小，max_stack 指出了该方法需要申请用于操作数栈执行的空间空间大小，narg 则说明了需要传递的参数个数。这些信息为之后创建相应的帧提供了重要的信息。而方法执行字节码则是一串符合 JCVM 规范并得以实现该方法的 16 字节操作码。

5 JCVM 中栈与帧的结构设计

通过对 JCVM 中存储空间的划分，可以得到一片预留的区域(即方法执行区)用于专门实现 JCVM 中栈的虚拟。给这片预留的区域制定一个类似于栈先进后出的操作规则，即为虚拟的出栈。而入栈的基本单位，则为帧(Frame)。帧和方法具有一一对应的映射关系，每调用一个方法，就需要创建一个帧，并且入栈，而当方法执行完并返回值之后，相应的帧也将出栈并销毁。

帧主要用于存储数据和操作结果，返回方法的值。它主要由局部变量区、操作数栈和帧控制信息(FrameCI)组成。局部变量区主要是以 1 个单元，也就是 2 个字节作为其基本单位的局部变量数组 (local variable array)，而存在其中的每一个元素都是属于该方法的一个 local array。其主要用来存储方法传递的参数和相关变量，是数组的结构，在字节码执行过程中，通过数组的索引值进行读写。操作数栈(operand stacks)也是以 1 个单元作为其基本单位，是字节码执行过程中用于临时存储中间数据和操作结构的一片预留区，根据相应方法的方法头信息确定预留空间的大小，通过执行字节码进行出入操作数栈的操作。此外，在调用方法的时候，操作数栈还负责存储传递给该方法的参数值以及存储由该方法返回的返回值。而帧的控制信息主要包括当前方法物理地址 (thismethodP)、当前方法上下文(thiscontext)、调用者帧的地址(invmethodP)、调用者方法 Bytecode 执行进度(invbytecodenum)、调用者方法操作数栈指针(invoperandSP)。这些信息将栈中的每一个帧都动态链接起来，并起到对每个方法执行进度进行记录的作用。当前方法物理地址用于读取当前方法的 Bytecode，上下文的作用相当于防火墙，用来阻止跨界的非法访问，调用者方法物理地址是在当前方法执行完成后读取调用者帧的地址，调用者方法 Bytecode 执行进度和操作数栈指针用于还原调用者帧的在调用前的现场。

栈与帧的结构设计如图 2 所示。

6 方法调用与返回操作的处理

如图 2 所示，每一个运行的方法对应着一个帧的结构。当一个方法需要调用另一个方法时，首先要求将被调用者方法的参数压入当前帧的操作数栈中，然后为该新方法创建一个新的帧，并入栈，将新的帧设置为当前帧。创建新帧的过程，首先是分配一个足够大小的空间给新的帧，这里，通过读取该方法的方法头，可以准确知道局部变量和操作数栈所需要的空间大小；然后初始化帧，将新方法的相关信息传入 FrameCI，并对一些全局变量和指针进行

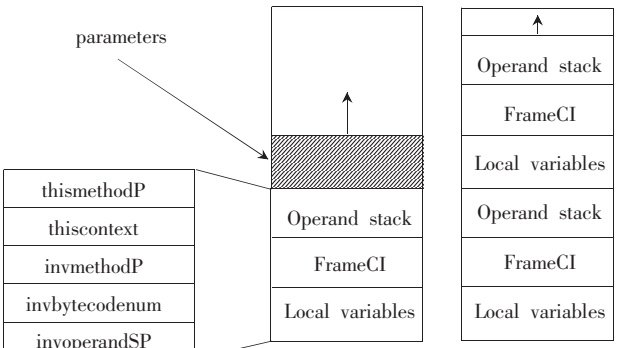

图 2 栈与帧的结构图

修改;接着是参数的传递。在 Java 卡中,新方法所需要调用的参数之前已经被压入调用者方法的帧中,在不考虑叠加技术的情况下,当方法调用执行时,先将参数从调用者方法帧的操作数栈中出栈,然后在顺序进入新方法帧的局部变量区,最后根据新方法的 Bytecode,执行相关操作。

对应方法结束的操作,需要销毁一个帧。这里的方法执行结果分为正常结束执行和不正常结束执行。在正常结束执行的情况下,有可能会有一个返回值给调用者方法,这时,首先将返回结果出该帧的操作数栈,通过 invmethodP 找到调用者帧,将调用者帧设为当前帧,再进调用者帧的操作数栈。然后修改相关全局变量和指针的值。最后回收原方法帧的使用空间,以留给下次帧的创建。若是不正常结束执行,虚拟机内将产生 exception 或因执行到一个抛出指令而抛出 exception,这时的方法就不会有返回值返回给调用者了。

另外,之前提到的叠加技术,就是在实现 JCVM 时,可以将调用者方法帧操作数栈和被调用者方法帧的局部变量区进行叠加的技术,即不需要把之前压入到调用者方法帧操作数栈中的参数出操作数栈再写到被调用者方法帧的局部变量区,而是直接将调用者方法帧操作数栈的参数部分看做被调用者方法帧的局部变量区的一部分,使之实现部分区域重合。采用叠加技术不会对方法的创建和销毁产生任何影响,却能简化方法间参数传递的机制,同时有效节约方法执行区的空间。

7 帧内部数据的操作

除了方法调用需要用到的栈操作之外,事实上在 JCVM 中,更多的是帧内部数据的操作。这些操作主要包括对局部变量区的读写操作和对操作数栈的出入栈操作。这些操作连同调用方法的操作一起,完成整个方法的执行。

例如,有这样一个方法 short add(short a,short b),其执行步骤如下:

```
Sload_1      //Load short from local variable 1,then push
Sload_2      //oad short from local variable 2,then push
Sadd         //Pop two shorts,add them,then push the result
Sstore_3     //Pop,then store short into local variable 3
Sreturn //Return short from method,then destroy the Frame
```

当有某方法需要调用这个方法时,首先根据方法头创建帧结构,将局部变量区和操作数栈初始化,控制信息赋值,接着根据方法的执行指令对操作数栈和局部变量区进行操作。具体操作步骤如图 3 所示。

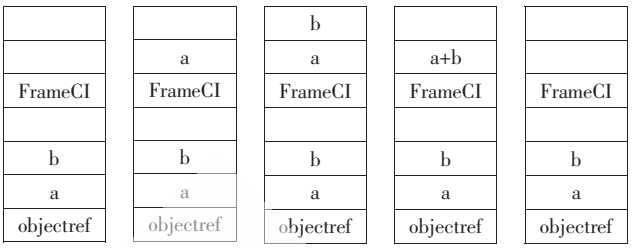

图 3 方法 short add 的演示图

在上例的帧执行演示中,图 3 的初始化步骤是根据方法头进行空间的申请,并将相关数据进行初始化赋值。Objectref 是对象引用,视具体调用方法而定,一般调用中会以参数的形式传给新的方法帧,赋值给 Local variable 0,而相关参数 a、b,也以参数形式在新的方法里分别赋值给 Local variable 1、Local variable 2。而方法字节码的前两个指令,将存储在局部变量区索引为 1、2 的两个数据压入操作数栈,其后 Sadd 指令从操作数栈中弹出这两个数据,进行加法,再将结果压回操作数栈中。然后 Sstore_3 从操作数栈中弹出结果值,存储到局部变量区索引为 3 的位置。最后,Sreturn 将该方法帧销毁,完成该方法的全过程。

通过对 JCVM 开发规范和一些智能卡开发公司需求和测试文档的研究和分析,本论文中所提出的存储资源管理策略,栈与帧结构的设计完全符合要求。通过利用各大公司提供的软件模拟环境和 Applet 应用数据包对栈与帧的设计方案进行测试,证明该方案正确可行。

JCVM 中栈与帧的设计与实现是开发 JCVM 的核心问题。本文提出了符合 JCVM 开发规范的栈与帧的结构设计和执行策略,并对存储空间进行划分管理,优化了有限的智能卡存储空间,并成功使用叠加技术改进了参数传递的机制,很好地完成了 JCVM 中栈与帧的基本功能。本文的研究已经成功运用到华大电子股份有限公司和清华同方公司的芯片上,并已经通过相关部门的软件测试。

参考文献

[1] 接触式智能卡国际规范[S].ISO7816(1-9).
[2] Java Card 2.2.2. Virtual Machine Specification[S]. 2006.
[3] Java Card 2.2.2. Application Programming Interface[S]. 2006.
[4] Java Card 2.2.2. Runtime Environment Specification[S]. 2006.
[5] Java Card Forum. http://www.javacardforum.org/.
[6] 吴东辉,周捷,陈章龙.Java 卡的设计[J].微型电脑应用,2003,19(12).

(收稿日期:2010-11-03)

作者简介:

何利明,男,1984 年生,硕士研究生,主要研究方向:智能卡芯片操作系统。

李代平,男,1955 年生,教授,主要研究方向:智能卡芯片操作系统,网络并行计算。