Java智能卡解析优化方法

张大伟 丁文锐

(北京交通大学 计算机与信息技术学院, 北京 100044) (北京航空航天大学 无人驾驶飞行器设计研究所, 北京 100191)

摘 要: 针对复杂的传统 Java智能卡指令解析问题,结合 Applet的部署特点,提出了指令预解析的概念;给出了转换后的 Applet文件 (CAP, Converted Applet file)中导入组件、常数池和导出组件的预解析优化存储结构及包注册表结构,使用定长存储结构替代原有的变长存储结构以加快查找过程,引入了导出组件辅助索引表以加快外包类的解析过程;设计了基于优化后组件的静态域、静态方法和类访问字节码指令的快速解析流程,将解析算法的时间复杂度降为常数阶;在 Applet下载过程中通过预解析来生成优化后的组件存储结构,在执行过程中依据快速解析流程来完成静态域、静态方法和类对象的快速访问,从而缩短了 Applet运行时的解析时间. 测试结果表明,在不改变标准 Applet下载、运行流程的情况下,该方法有效地减少了卡内存储器的访问次数,缩短了解析算法的执行时间,提高了 Applet执行的整体性能.

关 键 词: Java编程语言; 智能卡; 优化

中图分类号: TP 302

文献标识码: A **文 章 编 号**: 1001-5965(2009)01-0078-04

Optimization of resolution on Java card

Zhang Dawei

(School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China)

Ding Wenrui

(Research Institute of Unmanned Aerial Vehicle, Beijing University of Aeronautics and Astronautics, Beijing 100191, China)

Abstract: The pre-resolution was proposed based on the process of Applets deployment in order to optimize the resolution on Java card. At first, The optimized structure of import, constant pool and export component in CAP (converted Applet file) and package registration table were given. The fixed length structure instead of variable length structure in optimized components was used so as to speed up the search of resolution objects. Furthermore, the index of export component was designed to improve the resolution of classes in external packages. Secondly, the fast runtime resolution algorithms of static fields, static methods and classes were designed based on this optimized structure. The time complexity of those algorithms is $O(1)$. Thirdly, the optimized component structures were generated by pre-resolution during Applet download. The fast resolution of static fields, static methods and classes was implemented with runtime resolution based on those optimized structures. Test results show that this optimization algorithm reduces the times of memory access, speeds up the execution of resolution and improves the runtime performance of Applets on Java card.

Key words: Java programming language; smart cards; optimization

Java智能卡技术是 Java和智能卡技术的结合. Java智能卡是在智能卡硬件系统的基础之上在片内通过软件构造的一个支持 Java程序下载、安装并运行的软、硬件系统. 它在有限资源的智能卡环境中支持 Java语言的一个子集,是 Java嵌入到智能卡中的一种新应用[1]. 在标准仿真平台上

收稿日期: 2008-02-29

基金项目: 国家 863基金资助项目 (2007AA01Z410)

作者简介: 张大伟 (1974 -),男,辽宁朝阳人,讲师, dwzhang@bjtu.edu.cn.

开发的智能卡应用程序 Applet,通过下载工具可下载到不同厂家的 Java智能卡上正确运行. 实现了 Java设计者所宣称的:“(应用程序)一次写完,到处运行”的理念[2].

Java智能卡 Applet经过编译转换后下载到智能卡中的文件称为转换后的 Applet文件 (CAP, Converted Applet file),它包括导入、导出组件、类组件、方法组件和常数池组件等[3]. Java智能卡虚拟机在这些组件中使用标号 token来标识对包、类等的引用[4],而在虚拟机的执行过程中必须将这些 token转换为实际对象的引用 (物理地址),这一过程称为解析.

标准的解析流程是复杂且费时的. 为了解决这一问题,相关的研究工作给出了一些解析优化的方法[5-6]. 但存在如下一些问题:

1) 文献[5-6]的方法均需在下载前在卡外对 CAP文件进行预处理,这标准流程之外增加了操作,不利于应用推广和被终端用户接受;

2) 文献[5-6]均未考虑到存储管理开销对于解析过程的影响,随着智能卡存储容量的不断加大,取代简单的线性管理方法的是基于扇区、分配表等存储管理技术,因此存储管理开销在解析优化过程中不容忽视;

3) 文献[6]在优化处理过程中需多次改写字节码指令内容,由此带来的时间开销过大问题使得这一过程无法在卡内完成.

本文采用了保留字节码指令内容不变只优化 CAP结构的方法,从而降低了时间开销,使得优化过程可全部在卡内完成.

基于上述分析,本文在对标准 CAP文件的卡外处理流程不做更改的前提下,给出了卡内解析优化的设计与实现方案.

1　解析优化的总体原则

首先,本文的解析优化全部在 Java智能卡内来完成,不必在卡外对标准 CAP文件进行处理.

其次,基于对标准 CAP文件结构和解析流程的分析,将一部分解析工作交由下载管理器在下载过程中来完成,并据此来组织优化后的 CAP组件存储结构优化后的 Applet文件 (OAP, Optimized Applet file),这一过程称为预解析. 预解析的相关内容在第 2节中会详细介绍.

Applet在卡内运行时,解释器对于字节码执行过程中的解析执行完全基于优化后的 OAP文件进行,有效地降低了内存访问次数、缩短了解析时间. 优化后的运行时解析流程在第 3节中会详细介绍. 优化后的解析流程由预解析和运行时解析 2部分组成,如图 1所示.

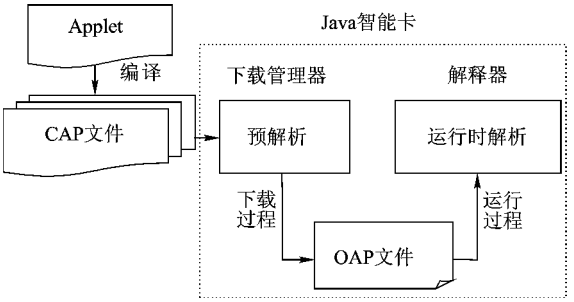

图 1　优化后的解析流程

2　优化后的 OAP组件结构

传统的解析流程主要存在的不足有: ①标准组件中使用 token进行链接,如果解析过程全部在运行时完成,必然影响速度; ②多个标准组件 (如导入组件,导出组件)均使用了变长结构,这使得基于 token(为组件结构项的索引)的查找过程是线性阶 $O(n)$,解析过程中的多次内存访问带来了存储管理开销的增大,增加了解析算法的复杂度.

因此在下载过程的预解析中,考虑到处理时间和内存容量的限制,在基本不改动原有组件大小和产生信息丢失的前提下,本文将完成如下工作:

1) 为了便于组件的管理,在卡内设计了注册表,每一个包 (package)对应一个注册表项,其中包含包中各个组件的存储地址;

2) 完成 token的部分或全部解析工作,即使用解析对象的偏移量或绝对地址重写 token项;

3) 改进组件结构,采用定长结构存储,将基于 token的查找过程的时间复杂度降为常数阶 $O(1)$,使得解析时间不依赖于 token值.

预解析的处理过程涉及到导入组件、常数池组件和导出组件,预解析优化后的结构如下:

导入组件:

导入组件包含了本包类所导入的其他包. 其中使用包 token进行索引的包信息 package_info结构如下:

```
package_info {
    u1 minor_version
    u1 major_version
    u1 AID_length
    u1 AID[AID_length]
}
```

其中, ux表示 x字节的无符号数. 导入组件

预解析方案将在注册表中根据应用标识符(AID, Application Identifier)查找相应包所对应的项,使用注册表项的首地址 package_address(扇区编号和偏移量)重写 packages[]数组中 AID_length和AID域.而 pacakge_info中的其他信息保持不变.从而在完成导入包地址解析的同时将变长结构改进为定长结构.

优化后的 package_info结构如下:

```
package_info {
    u1 minor_version
    u1 major_version
    u4 package_address
}
```

常数池组件:

常数池组件的处理主要包括静态域、静态方法和类引用项.其中外包解析(包之间的互相访问)流程较为复杂,涉及到导入组件、注册表、导出组件和解析对象组件,解析流程如图 2所示.

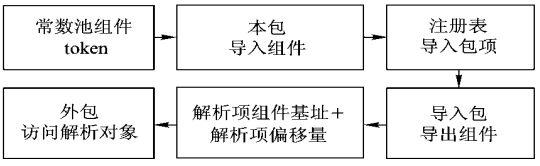

图 2 外包解析项的解析流程

因此在常数池项的解析预处理中,重点为外包项的处理,考虑计算资源的限制,本文将完成全部(绝对地址)或部分(偏移量)解析流程.

静态域项:

根据已下载的静态域组件以及表项中的偏移量或 token值计算出其中静态域的绝对物理地址,写入静态域项中.优化后的静态域项结构如下:

```
CONSTANT_StaticFieldref_info {
    u4 static_field_address
}
```

静态方法项:

根据已下载的方法组件以及表项中的偏移量或 token值计算出静态方法的偏移地址,以及此方法所属的上下文 context(以用于运行时解析)写入静态方法项中.

优化后静态方法项结构如下:

```
CONSTANT_StaticMethodref_info {
    u1 tag
    u1 context
    u2 static_method_offset
}
```

类引用项:

根据已下载的类组件以及表项中的偏移量或 token值计算出其中类信息的偏移地址,以及此类所属的上下文 context(以用于运行时解析)写入类引用项中.优化后的类引用项结构如下:

```
CONSTANT_Classref_info {
    u1 tag
    u1 context
    u2 class_offset
}
```

导出包组件:

导出组件包含本包所导出的公共类和接口信息.由于结构信息的组织采用变长结构,因此添加了导出组件辅助索引表(定长结构),表中包含每一个导出类在导出组件中的地址 class_address,以加快外包对本包导出类的解析速度.索引表的定义如下:

```
Public_classref_component    {
    u1    count;
    u4    class_address[count];
}
```

其中 class_address数组索引为相应类的 token.

3 优化后的解析流程

基于优化后的 OAP组件结构,参考标准解析流程[7-8],本文设计了运行时静态域、静态方法和类的解析算法,它们具有常量阶时间复杂度 $O(1)$,即解析时间不依赖于解析对象的 token.

静态域指令解析流程如图 3所示.

1) 访问常数池组件,获得静态域项 CONSTANT_StaticFieldref_info;

2) 根据 static_field_address访问静态域.

静态方法解析流程如图 4所示.

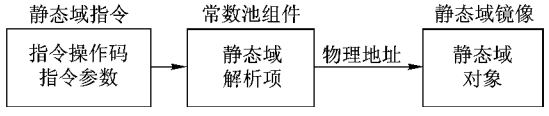

图 3 静态域解析流程

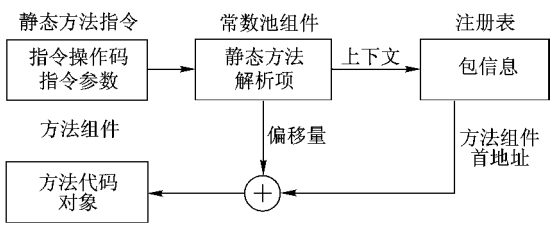

图 4 静态方法解析流程

1) 访问常数池组件,获得静态方法项 CONSTANT_StaticMethod_info;

2) 根据 context访问注册表获得方法组件首地址;

3) 根据组件首地址和 static_method_offset访问静态方法.

类解析流程如图 5所示.

1) 访问常数池组件,获得静态方法项 CONSTANT_Classref_info;

2) 根据 context访问注册表获得类组件首地址;

3) 根据首地址和 Class_offset访问类信息.

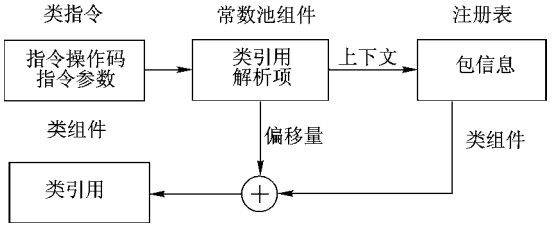

图 5　类解析流程

4　优化后的性能增益分析

由于解析算法的时间复杂度是 $O(n)$,而本文优化算法的的时间复杂度是 $O(1)$,不依赖于解析对象的 token值. 因此,在性能增益的分析中,本文选取了 Sun公司提供的 Java智能卡开发工具中有代表性的几个示例程序,将优化算法的衡量指标同传统算法的衡量指标的平均值进行比较,以分析优化后的性能增益.

本文选取的示例程序及优化指令所占比例如表 1所示.

表 1　优化指令所占比例

程序	类型	总行数	优化指令行数	优化比例/%
HelloWorld	调用密集	25	10	40
Wallet	计算密集	120	32	27
PhotoCard	远程调用	96	22	23
NullApp	普通	16	4	25
总计		161	68	26

优化算法解析过程中的内存访问次数同传统算法的平均内存访问次数的比较如表 2所示.

表 2　优化前后的内存访问次数比较

解析类型	优化前访问次数	优化后访问次数	比例
静态域	5	2	2/5
静态方法	7	3	3/5
类引用	8	3	3/5
总计	20	8	2/5

测试数据表明优化后的内存访问次数减少到传统算法的 2/5.

优化算法的解析执行时间同传统算法的平均执行时间的比较如表 3所示.

表 3　优化前后的执行时间比较

解析类型	优化前执行时间/μs	优化后执行时间/μs	比例/%
静态域	65.2	10.4	16
静态方法	120.6	35.0	29
类引用	152.8	37.5	25
总计	338.6	82.9	24

测试表明优化后执行时间为传统算法的 24%.

5　结 束 语

本文针对复杂的传统 Java智能卡指令解析问题,提出了指令预解析的概念,设计了基于优化后组件的快速字节码指令解析流程. 测试结果表明,优化算法减少了内存访问次数,缩短了指令解析时间,有效地提高了 Applet的执行性能.

参考文献 (References)

[1] Guthery S B. Java Card: internet computing on a smart card [J]. Internet Computing, IEEE, 1997, 1 (1): 57 - 59

[2] Lindholm T, Yellin F. The Java virtual machine specification [M]. New York: Addison Wesley, 1996

[3] Sun. Java card 2.2.2 virtual machine specification [EB/OL]. Santa Clara Califomia: Sun Microsystems, 2006 [2008-04-08]. http://java.sun.com/javacard/specs.html

[4] Denny E. The systhesis of a Java Card tokenization algorithm [C]//Proceedings of the 16th IEEE International Conference on Automated Software Engineering. Washington DC: IEEE Computer Society, 2001: 43 - 52

[5] Beurden J, Knowles G, Bignell G. An extended CAP file to support high performance embedded Java [J]. WSEAS Transactions on Computers, 2004, 3(1): 128 - 135

[6] 靳伟,常青,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 - 1208
Jin Wei, Chang Qing, Li Chunlong, et al. JCVM resolving optimization design and implementation [J]. Journal of Beijing University of Aeronautics and Astronautics, 2004, 30(12): 1204 - 1208 (in Chinese)

[7] Denney E, Jensen T. Correctness of Java Card method lookup via logical relations [J]. Theoretical Computer Science, 2002, 283(2): 305 - 331

[8] Hartel P H, Moreau L. Formalising the safety of Java, the Java virtual machine and Java Card [J]. ACM Computing Surveys, 2001, 33(4): 517 - 558

(责任编辑:吴　琼)