(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 103218551 B
(45)授权公告日 2016.04.06

(21)申请号 201310159747.7
(22)申请日 2013.05.03
(73)专利权人 飞天诚信科技股份有限公司
地址 100085 北京市海淀区学清路 9 号汇智大厦 B 楼 17 层
(72)发明人 陆舟 于华章
(51)Int.Cl.
G06F 21/14(2013.01)

(56)对比文件
CN 102214281 A,2011.10.12,
WO 2006/063876 A1,2006.06.22,
CN 102043932 A,2011.05.04,
张洪娜等 . Java Class 文件的结构分析及其解析执行 . 《计算机应用与软件》. 2011, 第 28 卷(第 7 期), 第 180-182 页 .

审查员 崔成东

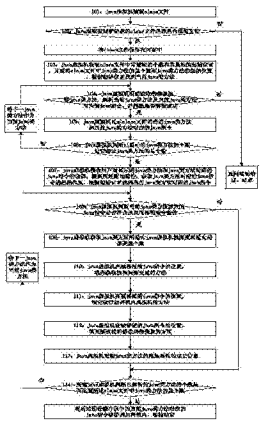
权利要求书3页 说明书15页 附图2页

(54)发明名称

一种保护 java 程序的方法

(57)摘要

本发明公开一种保护 java 程序的方法，属于软件保护领域。所述方法包括：java 虚拟机装载 class 文件，当 class 文件中的 java 类方法对应的 java 指令符合预设可移植指令条件时，将 java 类方法对应的 java 指令移植到预设缓冲区中，获取 java 类方法的最大栈深度和最大局部变量个数，并用获取栈和局部变量的 java 类方法、执行加密锁内虚拟机的 java 类方法、修改栈和局部变量的 java 类方法填充 class 文件中被移植的 java 指令的位置。采用本发明的技术方案，将部分 java 程序移植到加密锁中，修改了被保护的 java 程序，使得程序逻辑不完整，无法还原原始 java 程序，提高了 java 程序的安全性。



CN 103218551 B

1.一种保护 java 程序的方法,其特征在于,包括 :

步骤 A :java 虚拟机装载 class 文件,并判断所述 class 文件是否是合法的文件,如果是,则执行步骤 B,否则返回错误信息,结束 ;

步骤 B :所述 java 虚拟机获取所述 class 文件中 java 类方法的总个数和 java 类方法的起始位置,根据所述起始位置找到当前 java 类方法 ;

步骤 C :所述 java 虚拟机解析当前 java 类方法对应的 java 指令,并判断是否解析成功,如果是,则执行步骤 D,否则返回错误信息,结束 ;

步骤 D :所述 java 虚拟机判断所述当前 java 类方法对应的 java 指令是否符合预设可移植指令条件,如果是,则将所述当前 java 类方法对应的 java 指令移植并保存到预设缓冲区中,执行步骤 E,否则执行步骤 G ;

步骤 E :所述 java 虚拟机获取当前 java 类方法的最大 java 虚拟机栈深度和最大局部变量个数 ;

步骤 F :所述 java 虚拟机根据所述最大 java 虚拟机栈深度和最大局部变量个数,用获取栈和局部变量的 java 类方法、执行加密锁内虚拟机的 java 类方法与修改栈和局部变量的 java 类方法填充所述 class 文件中被移植的 java 指令的位置,执行步骤 G ;

步骤 G :所述 java 虚拟机判断已解析的 java 类方法的个数是否达到所述 class 文件中 java 类方法的总个数,如果是,则将所述预设缓冲区中的当前 java 类方法对应的 java 指令移植到加密锁内,结束,否则继续获取下一 java 类方法作为当前 java 类方法,返回执行步骤 C ;

所述步骤 C 与所述步骤 D 之间包括 :

步骤 A-1 :所述 java 虚拟机显示 class 文件的当前 java 类方法和当前 java 类方法对应的 java 指令 ;

步骤 B-1 :所述 java 虚拟机判断已显示的 java 类方法的个数是否超过 java 类方法的总个数,如果是,则执行步骤 C-1,否则将下一 java 类方法作为当前 java 类方法,返回执行步骤 C ;

步骤 C-1 :所述 java 虚拟机接收用户对显示的 java 类方法和 java 类方法对应的 java 指令的选择,获取当前 java 类方法对应的 java 指令,执行步骤 D。

2.根据权利要求 1 所述的方法,其特征在于,所述步骤 A 中,所述判断所述 class 文件是否是合法的文件,具体为 :获取并判断 class 文件的文件头是否为预设字符串,如果是,则表示所述 class 文件是合法的文件,否则表示所述 class 文件是不合法的文件。

3.根据权利要求 1 所述的方法,其特征在于,所述步骤 C 中,所述解析当前 java 类方法对应的 java 指令,具体为 :

步骤 C-1 :所述 java 虚拟机获取所述 class 文件中常量池的个数和常量池的起始位置 ;

步骤 C-2 :所述 java 虚拟机根据当前 java 类方法的第一成员变量和第二成员变量,从所述常量池中获取 java 类方法名称和 java 类方法描述符,并判断是否获取成功,如果是,则执行步骤 C-3,否则返回错误信息,结束 ;

步骤 C-3 :所述 java 虚拟机根据当前 java 类方法的第三成员变量和第四成员变量,获取当前 java 类方法的附加属性,并判断是否获取成功,如果是,则执行步骤 D,否则返回错

误信息，结束。

4. 根据权利要求 3 所述的方法，其特征在于，所述当前 java 类方法的第一成员变量是对所述常量池中的 java 类方法名称的索引；所述当前 java 类方法的第二成员变量是对所述常量池中 java 类方法描述符的索引；所述当前 java 类方法的第三成员变量为当前 java 类方法的附加属性的个数；所述当前 java 类方法的第四成员变量为当前 java 类方法的附加属性的起始位置。

5. 根据权利要求 3 或 4 所述的方法，其特征在于，所述附加属性包括 code 属性，code 属性中包括当前 java 类方法的 java 指令和当前 java 类方法的 java 指令的长度。

6. 根据权利要求 3 所述的方法，其特征在于，所述步骤 C 与所述步骤 D 之间具体包括：

步骤 A-1：所述 java 虚拟机显示 class 文件的当前 java 类方法和当前 java 类方法对应的 java 指令；

步骤 B-1：所述 java 虚拟机判断已解析的 java 类方法的个数是否超过所述 class 文件中 java 类方法的总个数，如果是，则执行步骤 C-1，否则将下一 java 类方法作为当前 java 类方法，返回执行步骤 C；

步骤 C-1：所述 java 虚拟机接收用户对显示的 java 类方法和 java 类方法对应的 java 指令的选择，根据所述附加属性，获取用户选择的 java 类方法对应的 java 指令的起始位置，根据所述起始位置，找到当前 java 类方法对应的 java 指令，执行步骤 D。

7. 根据权利要求 3 所述的方法，其特征在于，所述步骤 D 具体为：

步骤 A-2：所述 java 虚拟机判断当前 java 类方法对应的 java 指令是否符合预设可移植指令条件，如果是，则执行步骤 B-2，否则执行步骤 G；

步骤 B-2：所述 java 虚拟机显示 java 文件的当前 java 类方法和当前 java 类方法对应的 java 指令；

步骤 C-2：所述 java 虚拟机判断已解析的 java 类方法的个数是否超过 class 文件中 java 类方法的总个数，如果是，则执行步骤 D-2，否则将下一 java 类方法作为当前 java 类方法，返回执行步骤 C；

步骤 D-2：所述 java 虚拟机接收用户对显示的 java 类方法和 java 类方法对应的 java 指令的选择，根据附加属性，获取被选择的 java 指令的起始位置，将起始位置的 java 指令作为当前 java 指令，将所述当前 java 指令移植并保存到预设缓冲区中，执行步骤 E。

8. 根据权利要求 1 或 6 或 7 中任意一项所述的方法，其特征在于，所述判断当前 java 类方法对应的 java 指令是否符合预设可移植指令条件，具体为：

步骤 D-1：所述 java 虚拟机判断当前 java 类方法对应的 java 指令的类型是否存在于预设可移植 java 指令类型集合中，如果是，则执行步骤 D-2，否则表示当前指令不可被移植，执行步骤 G；

步骤 D-2：所述 java 虚拟机判断当前 java 类方法对应的 java 指令是否存在于预设引用常量池指令集合中，如果是，则表示当前指令不可被移植，执行步骤 G，否则执行步骤 D-3；

步骤 D-3：所述 java 虚拟机判断当前 java 类方法对应的 java 指令是否涉及内存操作，如果是，则表示当前指令不可被移植，执行步骤 G，否则执行步骤 D-4；

步骤 D-4：所述 java 虚拟机判断当前 java 类方法对应的 java 指令是否对 class 文件

结构体的成员变量进行操作，如果是，则表示当前指令不可被移植，执行步骤 G，否则执行步骤 D-5 ；

步骤 D-5 ：所述 java 虚拟机判断当前 java 类方法对应的 java 指令是否调用外部 java 类方法，如果是，则表示当前指令不可被移植，执行步骤 G，否则执行步骤 E。

9. 根据权利要求 8 所述的方法，其特征在于，所述预设可移植 java 指令类型集合，包括栈操作指令类型、逻辑运算指令类型、位操作指令类型、算数运算指令类型。

10. 根据权利要求 1 所述的方法，其特征在于，所述步骤 F 中，所述用获取栈和局部变量的 java 类方法填充所述 class 文件中被移植的 java 指令的位置，具体为 ：

步骤 F-11 ：所述 java 虚拟机定位到被移植的当前 java 类方法的位置 ；

步骤 F-12 ：所述 java 虚拟机创建当前 java 类方法的常量池的索引 ；

步骤 F-13 ：所述 java 虚拟机将获取栈和局部变量的 java 类方法与所述常量池的索引填充到被移植的当前 java 类方法的位置。

11. 根据权利要求 1 所述的方法，其特征在于，所述步骤 F 中，所述用执行加密锁内虚拟机的 java 类方法填充所述 class 文件中被移植的 java 指令的位置，具体为 ：

步骤 F-21 ：所述 java 虚拟机定位到被移植的当前 java 类方法的位置 ；

步骤 F-22 ：所述 java 虚拟机创建当前 java 类方法的常量池的索引 ；

步骤 F-23 ：所述 java 虚拟机将执行加密锁内虚拟机的 java 类方法与所述常量池的索引填充到被移植的当前 java 类方法的位置。

12. 根据权利要求 1 所述的方法，其特征在于，所述步骤 F 中，所述用修改栈和局部变量的 java 类方法填充所述 class 文件中被移植的 java 指令的位置，具体为 ：

步骤 F-31 ：所述 java 虚拟机定位到被移植的当前 java 类方法的位置 ；

步骤 F-32 ：所述 java 虚拟机创建当前 java 类方法的常量池的索引 ；

步骤 F-33 ：所述 java 虚拟机将修改栈和局部变量的 java 类方法与所述常量池的索引填充到所述 class 文件中被移植的当前 java 类方法的位置。

一种保护 java 程序的方法

技术领域

[0001] 本发明涉及软件保护领域，尤其涉及一种保护 java 程序的方法。

背景技术

[0002] Java 语言是一种解释型语言。Java 源代码经过编译，生成 java 指令文件，所生成的 java 指令文件在 Java 虚拟机 (JVM) 中解释执行。这种解释执行的运行机制导致 Java 源代码很容易被反编译。目前这种反编译工具很多，并且反编译的效果也很好，使得任何使用 Java 软件的用户都可以很容易地反编译和重构产品的源代码，从而使所有基于 Java 语言的授权认证许可变得毫无意义。

[0003] 因此，保证 Java 软件程序不被反编译和重构或者提高反编译和重构的难度，是基于 Java 语言的软件保护领域需要解决的一个重要问题。

发明内容

[0004] 本发明的目的是为了克服现有技术的不足，提供了一种保护 java 程序的方法。

[0005] 本发明采用的技术方案是：一种保护 java 程序的方法，包括：

[0006] 步骤 A：java 虚拟机装载 class 文件，并判断所述 class 文件是否是合法的文件，如果是，则执行步骤 B，否则返回错误信息，结束；

[0007] 步骤 B：所述 java 虚拟机获取所述 class 文件中 java 类方法的总个数和 java 类方法的起始位置，根据所述起始位置找到当前 java 类方法；

[0008] 步骤 C：所述 java 虚拟机解析当前 java 类方法对应的 java 指令，并判断是否解析成功，如果是，则执行步骤 D，否则返回错误信息，结束；

[0009] 步骤 D：所述 java 虚拟机判断所述当前 java 类方法对应的 java 指令是否符合预设可移植指令条件，如果是，则将所述当前 java 类方法对应的 java 指令移植并保存到预设缓冲区中，执行步骤 E，否则执行步骤 G；

[0010] 步骤 E：所述 java 虚拟机获取当前 java 类方法的最大 java 虚拟机栈深度和最大局部变量个数；

[0011] 步骤 F：所述 java 虚拟机根据所述最大 java 虚拟机栈深度和最大局部变量个数，用获取栈和局部变量的 java 类方法、执行加密锁内虚拟机的 java 类方法与修改栈和局部变量的 java 类方法填充所述 class 文件中被移植的 java 指令的位置，执行步骤 G；

[0012] 步骤 G：所述 java 虚拟机判断已解析的 java 类方法的个数是否达到所述 class 文件中 java 类方法的总个数，如果是，则将所述预设缓冲区中的当前 java 类方法对应的 java 指令移植到加密锁内，结束，否则继续获取下一 java 类方法作为当前 java 类方法，返回执行步骤 C；

[0013] 所述步骤 A 中，所述判断所述 class 文件是否是合法的文件，具体为：获取并判断 class 文件的文件头是否为预设字符串，如果是，则表示所述 class 文件是合法的文件，否则表示所述 class 文件是不合法的文件；

[0014] 所述步骤 C 中，所述解析当前 java 类方法对应的 java 指令，具体为：

[0015] 步骤 C-1：所述 java 虚拟机获取所述 class 文件中常量池的个数和常量池的起始位置；

[0016] 步骤 C-2：所述 java 虚拟机根据当前 java 类方法的第一成员变量和第二成员变量，从所述常量池中获取 java 类方法名称和 java 类方法描述符，并判断是否获取成功，如果是，则执行步骤 C-3，否则返回错误信息，结束；

[0017] 步骤 C-3：所述 java 虚拟机根据当前 java 类方法的第三成员变量和第四成员变量，获取当前 java 类方法的附加属性，并判断是否获取成功，如果是，则 执行步骤 D，否则返回错误信息，结束；

[0018] 所述当前 java 类方法的第一成员变量是对所述常量池中的 java 类方法名称的索引；所述当前 java 类方法的第二成员变量是对所述常量池中 java 类方法描述符的索引；所述当前 java 类方法的第三成员变量为当前 java 类方法的附加属性的个数；所述当前 java 类方法的第四成员变量为当前 java 类方法的附加属性的起始位置；

[0019] 所述附加属性包括 code 属性，code 属性中包括当前 java 类方法的 java 指令和当前 java 类方法的 java 指令的长度；

[0020] 所述步骤 C 与所述步骤 D 之间包括：

[0021] 步骤 A-1：所述 java 虚拟机显示 class 文件的当前 java 类方法和当前 java 类方法对应的 java 指令；

[0022] 步骤 B-1：所述 java 虚拟机判断已解析的 java 类方法的个数是否超过所述 class 文件中 java 类方法的总个数，如果是，则执行步骤 C-1，否则将下一 java 类方法作为当前 java 类方法，返回执行步骤 C；

[0023] 步骤 C-1：所述 java 虚拟机接收用户对显示的 java 类方法和 java 类方法对应的 java 指令的选择，根据所述附加属性，获取用户选择的 java 类方法对应的 java 指令的起始位置，根据所述起始位置，找到当前 java 类方法对应的 java 指令，执行步骤 D；

[0024] 所述步骤 D 具体为：

[0025] 步骤 A-2：所述 java 虚拟机判断当前 java 类方法对应的 java 指令是否符合预设可移植指令条件，如果是，则执行步骤 B-2，否则执行步骤 G；

[0026] 步骤 B-2：所述 java 虚拟机显示 java 文件的当前 java 类方法和当前 java 类方法对应的 java 指令；

[0027] 步骤 C-2：：所述 java 虚拟机判断已解析的 java 类方法的个数是否超过 class 文件中 java 类方法的总个数，如果是，则执行步骤 D-2，否则将下一 java 类方法作为当前 java 类方法，返回执行步骤 C；

[0028] 步骤 D-2：所述 java 虚拟机接收用户对显示的 java 类方法和 java 类方法对 应的 java 指令的选择，根据附加属性，获取被选择的 java 指令的起始位置，将起始位置的 java 指令作为当前 java 指令，将所述当前 java 指令移植并保存到预设缓冲区中，执行步骤 E；

[0029] 所述判断当前 java 类方法对应的 java 指令是否符合预设可移植指令条件，具体为：

[0030] 步骤 D-1：所述 java 虚拟机判断当前 java 类方法对应的 java 指令的类型是否存在于预设可移植 java 指令类型集合中，如果是，则执行步骤 D-2，否则表示当前指令不可被

移植，执行步骤 G ；

[0031] 步骤 D-2 ：所述 java 虚拟机判断当前 java 类方法对应的 java 指令是否存在于预设引用常量池指令集合中，如果是，则表示当前指令不可被移植，执行步骤 G，否则执行步骤 D-3 ；

[0032] 步骤 D-3 ：所述 java 虚拟机判断当前 java 类方法对应的 java 指令是否涉及内存操作，如果是，则表示当前指令不可被移植，执行步骤 G，否则执行步骤 D-4 ；

[0033] 步骤 D-4 ：所述 java 虚拟机判断当前 java 类方法对应的 java 指令是否对 class 文件结构体的成员变量进行操作，如果是，则表示当前指令不可被移植，执行步骤 G，否则执行步骤 D-5 ；

[0034] 步骤 D-5 ：所述 java 虚拟机判断当前 java 类方法对应的 java 指令是否调用外部 java 类方法，如果是，则表示当前指令不可被移植，执行步骤 G，否则执行步骤 E ；

[0035] 所述预设可移植 java 指令类型集合，包括栈操作指令类型、逻辑运算指令类型、位操作指令类型、算数运算指令类型 ；

[0036] 所述步骤 F 中，所述用获取栈和局部变量的 java 类方法填充所述 class 文件中被移植的 java 指令的位置，具体为 ：

[0037] 步骤 F-11 ：所述 java 虚拟机定位到被移植的当前 java 类方法的位置 ；

[0038] 步骤 F-12 ：所述 java 虚拟机创建当前 java 类方法的常量池的索引 ；

[0039] 步骤 F-13 ：所述 java 虚拟机将获取栈和局部变量的 java 类方法与所述常量 池的索引填充到被移植的当前 java 类方法的位置 ；

[0040] 所述步骤 F 中，所述用执行加密锁内虚拟机的 java 类方法填充所述 class 文件中被移植的 java 指令的位置，具体为 ：

[0041] 步骤 F-21 ：所述 java 虚拟机定位到被移植的当前 java 类方法的位置 ；

[0042] 步骤 F-22 ：所述 java 虚拟机创建当前 java 类方法的常量池的索引 ；

[0043] 步骤 F-23 ：所述 java 虚拟机将执行加密锁内虚拟机的 java 类方法与所述常量池的索引填充到被移植的当前 java 类方法的位置 ；

[0044] 所述步骤 F 中，所述用修改栈和局部变量的 java 类方法填充所述 class 文化中被移植的 java 指令的位置，具体为 ：

[0045] 步骤 F-31 ：所述 java 虚拟机定位到被移植的当前 java 类方法的位置 ；

[0046] 步骤 F-32 ：所述 java 虚拟机创建当前 java 类方法的常量池的索引 ；

[0047] 步骤 F-33 ：所述 java 虚拟机将修改栈和局部变量的 java 类方法与所述常量池的索引填充到所述 class 文化中被移植的当前 java 类方法的位置 ；

[0048] 本发明取得的有益效果是 ：采用本发明的技术方案，将部分 java 指令移植到加密锁中，修改了被保护的 java 程序，使得程序逻辑不完整，无法还原出原始的软件代码，提高了 java 程序的安全性。

附图说明

[0049] 图 1 是本发明实施例 1 提供的一种保护 java 程序的方法流程图 ；

[0050] 图 2 是本发明实施例 1 中步骤 108 的细化图。

具体实施方式

[0051] 下面将结合本发明实施例中的附图，对本发明实施例中的技术方案进行清楚、完整地描述，显然，所描述的实施例仅仅是本发明一部分实施例，而不是 全部的实施例。基于本发明中的实施例，本领域普通技术人员在没有做出创造性劳动前提下所获得的所有其他实施例，都属于本发明保护的范围。

[0052] 实施例 1 ：

[0053] 本发明的实施例 1 提供了一种保护 java 程序的方法，如图 1 所示，包括 ：

[0054] 步骤 101 ：java 虚拟机装载 class 文件 ；

[0055] 步骤 102 ：java 虚拟机判断装载的 class 文件是否是合法的文件，如果是，则将 class 文件保存在内存中，执行步骤 103，否则表示所述 class 文件是不合法的文件，返回错误信息，结束 ；

[0056] 在本实施例中，所述 class 文件的文件头为 classfile 结构体 ；

[0057] 所述 classfile 结构体为 ：

[0058]

```
Classfile{
u4    magic;
u2    magic_version;
u2    magic_version;
u2    constant_pool_count;
cp_infoconstant_pool[constant_pool_cpunt-1];
u2    access_flags;
u2    super_class;
u2    interfaces_count;
```

[0059]

```
u2    fields_count;
field_info    fields[fields_count];
u2    methods_count;
method_info    methods[methods_count];
u2    attributes_count;
attribute_info    attributes[attributes_count];
}
```

[0060] 优选的，判断 classfile 结构体中的成员变量 magic 是否为 0xCAFEBABE，如果是，则表示所述 class 文件是合法的文件，否则表示所述 class 文件是不合法的文件；

[0061] 步骤 103：java 虚拟机获取 class 文件中常量池的个数和常量池的起始位置，并获得 class 文件中 java 类方法的总个数和 java 类方法的起始位置，根据起始位置找到当前 java 类方法；

[0062] 所述常量池中包括 java 类方法名称和 java 类方法描述符；

[0063] 优选的，获取 Classfile 结构体中的成员变量 constant_pool_count 的值，即为 class 文件中常量池的个数，根据 Classfile 结构体中的成员变量 Cp_infoconstant_pool[constant_pool_cpunt-1] 可知常量池的起始位置；

[0064] 获取 Classfile 结构体中的成员变量 methods_count 的值，即为 class 文件中 java 类方法的总个数；根据 Classfile 结构体中的成员变量 Method_info methods[methods_count] 可知 java 类方法的起始位置；

[0065] 所述当前 java 类方法为：

[0066]

```
public class CRC32 {
    public static int getCRC32(String _source){
        intcrc = 0xFFFFFFFF;
        int poly = 0xEDB88320;
        byte[] bytes = _source.getBytes();
        for (byte b : bytes) {
        int temp = (crc ^ b) & 0xff;
        for (inti = 0; i< 8; i++)
            { if ((temp & 1) == 1)
             temp = (temp >>> 1) ^ poly;
             else
             temp = (temp >>> 1);
              }
             crc = (crc>>> 8 ^ temp);
          }
        crc = crc ^ 0xffffffff;
        returncrc;
      }
}
```

[0067] 步骤 104 :java 虚拟机根据获取的常量池和 java 类方法,解析当前 java 类方法及当前 java 类方法对应的 java 指令,并判断是否解析成功,如果是,则执行步骤 105,否则返回错误信息,结束 ;

[0068] 所述解析当前方法,具体为 :

[0069] 步骤 104-1 :java 虚拟机根据当前 java 类方法的第一成员变量和第二成员变量,从常量池中获取 java 类方法名称和 java 类方法描述符,并判断是否获取成功,如果是,则执行步骤 104-2,否则返回错误信息,结束 ;

[0070] 所述方法的结构体为 :

[0071]

```
Method_info{
u2    access_flags;
u2    name_index;
u2    descriptor_index;
u2    attributes_count;
Attribute_info    attributes[attributes_count];
}
```

[0072] 所述第一成员变量为 name_index，是对常量池中的 java 类方法名称的索引，第二成员变量为 u2 descriptor_index，是对常量池中 java 类方法描述符的索引；

[0073] 步骤 104-2：java 虚拟机根据当前 java 类方法的第三成员变量和第四成员变量，获取附加属性，并判断是否获取成功，如果是，则执行步骤 105，否则返回错误信息，结束；

[0074] 所述第三成员变量为 attributes_count，是附加属性的个数，第四成员变量为 attribute_info attributes[attributes_count]，表示附加属性的起始位置；

[0075] 在本实施例中，所述附加属性包括 code 属性和 Exceptions 属性，其中，code 属性用于解释 java 指令的数值，Exceptions 属性用来指出 java 类方法需要检查的可能抛出的异常；

[0076] 所述解析当前 java 类方法对应的 java 指令，具体为：获取当前 java 类方法对应的 java 指令的总长度及其所述当前 java 类方法对应的 java 指令在 code 属性结构体中的偏移；

[0077] 所述 code 属性的结构体为：

[0078]

```
code_attribute {
u2 attribute_name_index;
u4 attribute_length;
u2 max_stack;
u2 max_locals;
u4 code_length;
u1 code[code_length];
u2 exception_table_length;
{ u2 start_pc;
  u2 end_pc;
  u2 handler_pc;
  u2 catch_type;
} exception_table[exception_table_length];
u2 attributes_count;
attribute_info attributes[attributes_count];}
```

[0079] 获取所述 code 属性的结构体的成员变量 code_length 的值，即为当前 java 类方法对应的 java 指令的长度，根据所述 code 属性结构体的成员变量 code[code_length]，获取当前 java 类方法对应的 java 指令的起始位置，依次遍历所述当前 java 类方法对应的 java 指令；

[0080] 步骤 105：java 虚拟机显示 class 文件的当前 java 类方法和当前 java 类方法对应的 java 指令；

[0081] 步骤 106：java 虚拟机判断已显示的 java 类方法的个数是否超过 java 类方法的总个数，如果是，则执行步骤 107，否则将下一 java 类方法作为当前 java 类方法，返回执行步骤 104；

[0082] 步骤 107：java 虚拟机接收用户对显示的 java 类方法和 java 类方法对应的 java 指令的选择，根据所述附加属性，获取 java 类方法对应的 java 指令的起始位置，根据起始位置找到当前 java 类方法对应的 java 指令；

[0083] 根据所述附加属性中的成员变量 code[code_length]，获取当前 java 类方法对应的 java 指令的起始位置，根据起始位置找到当前 java 类方法对应的 java 指令；

[0084] 在本实施例中，当前 java 类方法对应的 java 指令为：

[0085]

```
0:  iconst_m1
1:  istore_1
2:  ldc -306674912
4:  istore_2
5:  aload_0
6:  invokevirtual  java.lang.String.getBytes ()[B
9:  astore_3
10: aload_3
11: astore  %4
13: aload  %4
15: arraylength
16: istore  %5
```

[0086]

```
18:  iconst_0
19:  istore   %6
21:  iload    %6
23:  iload    %5
25:  if_icmpge    #100
28:  aload    %4
30:  iload    %6
32:  baload
33:  istore   %7
35:  iload_1
36:  iload    %7
38:  ixor
39:  sipush   255
42:  iand
43:  istore   %8
45:  iconst_0
46:  istore   %9
48:  iload    %9
50:  bipush   8
52:  if_icmpge    #86
55:  iload    %8
57:  iconst_1
58:  iand
59:  iconst_1
60:  if_icmpne    #74
63:  iload    %8
65:  iconst_1
66:  iushr
```

[0087]

```
67:  iload_2
68:  ixor
69:  istore   %8
71:  goto     #80
74:  iload    %8
76:  iconst_1
77:  iushr
78:  istore   %8
80:  iinc     %9  1
83:  goto     #48
86:  iload_1
87:  bipush   8
89:  iushr
90:  iload    %8
92:  ixor
93:  istore_1
94:  iinc     %6  1
97:  goto     #21
100: iload_1
101: iconst_m1
102: ixor
103: istore_1
104: iload_1
105: ireturn
```

[0088] 步骤 108 :java 虚拟机判断当前 java 类方法对应的 java 指令是否符合预设可移植指令条件,如果是,则执行步骤 109,否则执行步骤 114 ;

[0089] 参见图 2,所述判断当前 java 类方法对应的 java 指令是否符合预设可移植 指令条件,具体为 :

[0090] 步骤 108-1 :判断当前 java 类方法对应的 java 指令的类型是否存在于预设可移植 java 指令类型集合中,如果是,则将所述当前 java 类方法对应的 java 指令保存到预设

缓冲区中，执行步骤 108-2，否则执行步骤 114 ；

[0091] 所述预设可移植的 java 指令类型集合包括 ：栈操作指令类型，逻辑运算指令类型，位操作指令类型，算数运算指令类型 ；

[0092] 优选的，如果当前 java 指令的第一个字节为 0x01，则表示所述 java 指令类型为栈操作指令类型 ；如果当前 java 指令的第一个字节为 0x7e，则表示所述 java 指令类型为逻辑运算指令类型 ；如果当前 java 指令的第一个字节为 0x7a，则表示所述 java 指令类型为位操作指令类型 ；如果当前 java 指令的第一个字节为 0x6c，则表示所述 java 指令类型为算术运算指令类型 ；

[0093] 步骤 108-2 ：判断当前 java 类方法对应的 java 指令是否存在于预设引用常量池指令集合中，如果是，则表示当前 java 类方法对应的 java 指令不可被移植，执行步骤 114，否则执行步骤 108-3 ；

[0094] 例如，当前 java 类方法对应的 java 指令为 ldc 指令时，存在于预设引用常量池指令集合中，所述 ldc 指令用于从常量池中提取数据至栈中，不可被移植 ；

[0095] 步骤 108-3 ：判断当前 java 类方法对应的 java 指令是否涉及内存操作，如果是，则执行步骤 114，否则执行步骤 108-4 ；

[0096] 例如，当前 java 类方法对应的 java 指令为 new 指令时，用于创建一个对象，涉及内存操作，不可被移植 ；

[0097] 步骤 108-4 ：判断当前 java 类方法对应的 java 指令是否对 classfile 的成员变量进行操作，如果是，则执行步骤 114，否则执行步骤 108-5 ；

[0098] 例如，当前 java 类方法对应的 java 指令为 getfield 指令时，用于获取 classfile 中的成员变量，涉及到对成员变量的操作，不可被移植 ；

[0099] 步骤 108-5 ：判断当前 java 类方法对应的 java 指令是否调用 java 虚拟机外部方法，如果是，则执行步骤 114，否则执行步骤 109 ；

[0100] 例如，当前 java 类方法对应的 java 指令为 invokevirtual 指令时，用于调用实例方法，需要调用外部方法，不可被移植 ；

[0101] 所述步骤 105 至步骤 108 还可以为 ：

[0102] 步骤 A-1 ：java 虚拟机判断当前 java 类方法对应的 java 指令是否符合预设可移植指令条件，如果是，则执行步骤 B-1，否则执行步骤 114 ；

[0103] 步骤 B-1 ：java 虚拟机显示 java 文件的当前 java 类方法和当前 java 类方法对应的 java 指令 ；

[0104] 步骤 C-1 ：java 虚拟机判断已解析的 java 类方法的个数是否超过 java 类方法的总个数，如果是，则执行步骤 D-1，否则将下一 java 类方法作为当前 java 类方法，返回执行步骤 104 ；

[0105] 步骤 D-1 ：java 虚拟机接收用户对显示的 java 类方法和 java 类方法对应的 java 指令的选择，根据所述附加属性，获取被选择的 java 类方法对应的 java 指令的选择的起始位置，根据起始位置找到当前 java 类方法对应的 java 指令，将所述当前 java 类方法对应的 java 指令移植保存到预设缓冲区中，执行步骤 109 ；

[0106] 所述步骤 105 至步骤 108 还可以为 ：

[0107] 步骤 A-2 ：java 虚拟机判断当前 java 类方法对应的 java 指令是否符合预设可移

植指令条件，如果是，则执行步骤 B-2，否则执行步骤 114 ；

[0108] 步骤 B-2 ：java 虚拟机将所述当前 java 类方法对应的 java 指令移植保存到预设缓冲区中，执行步骤 109 ；

[0109] 步骤 109 ：java 虚拟机获取 java 类方法的最大 java 虚拟机栈深度和最大局部变量个数 ；

[0110] Java 虚拟机根据所述 code 属性结构体中的成员 max_stack 获取 java 类方法的最大栈深度，根据 code 属性结构体中的成员 max_locals 获取 java 类方法的最大局部变量个数 ；

[0111] 步骤 110 ：java 虚拟机用获取栈和局部变量的 java 类方法填充所述 class 文件中被移植的 java 指令的位置 ；

[0112] 具体为 ：

[0113] 步骤 110-1 ：java 虚拟机定位到被移植的当前 java 类方法的位置 ；

[0114] 步骤 110-2 ：java 虚拟机创建当前 java 类方法的常量池的索引 ；

[0115] 优选的，所述常量池的索引，指向存放的数据结构 CONSTANT_Methodref_info，用于描述当前 java 类方法，所述常量池的索引为两个字节，例如，所述常量池的索引为 0001，指向的是常量池中的 java 类方法为 javaanalyzer.R7JavaRuntime.GetLocalData (B)[B ；

[0116] 所述数据结构 CONSTANT_Methodref_info 为 ：

[0117]

```
CONSTANT_InterfaceMethodref_info {

u1 tag;

u2 class_index;

u2 name_and_type_index; };
```

[0118] 其中 class_index 表示类 r7javaanalyzer.R7JavaRuntime，name_and_type_index 表示方法描述符 GetStackData (B)[B ；

[0119] 所述常量池的索引为 0002，则指向的常量池的 java 类方法为 ：r7javaanalyzer.R7JavaRuntime.GetStackData (B)[B ；

[0120] 步骤 110-3 ：java 虚拟机将获取栈和局部变量的 java 类方法与所述常量池的索引填充到被移植的当前 java 类方法的位置 ；

[0121] 所述获取栈与局部变量的 java 类方法为 invokestatic 方法，该方法的数值为 0xb8，它需要两个字节的操作符，这两个字节的操作符即为所述常量池的索引 ；

[0122] 在本实施例中，在被移植的 java 指令的位置插入的内容为 ：Invokestatic r7 javaanalyzer.R7JavaRuntime.GetLocalData (B)[B，对应的数值为 0xb8 00 01 ；invokestaticr7javaanalyzer.R7JavaRuntime.GetStackData (B)[B，对应的数值为 0xb8 00 02 ；

[0123] 步骤 111 ：java 虚拟机用执行加密锁内虚拟机的 java 类方法填充所述 class 文件中被移植的 java 指令的位置 ；

[0124] 具体为：

[0125] 步骤 111-1：java 虚拟机定位到被移植的当前 java 类方法的位置；

[0126] 步骤 111-2：java 虚拟机创建当前 java 类方法的常量池的索引；

[0127] 优选的，所述常量池的索引为两个字节，例如，所述常量池的索引为 0003，指向的是常量池中的方法为 r7javaanalyzer.R7JavaRuntime.R7Java_RunJava (SS)V；

[0128] 步骤 111-3：java 虚拟机将执行加密锁内虚拟机的 java 类方法与所述常量池的索引填充到被移植的当前 java 类方法的位置；

[0129] 所述执行加密锁内虚拟机的 java 指令为 invokevirtual 指令，该指令的数值为 0xC7，它需要两个字节的操作符，这两个字节的操作符即为所述常量池的索 引；

[0130] 在本实施例中，在被移植的 java 指令的位置插入的内容为：invokevirtualr7javaanalyzer.R7JavaRuntime.R7Java_RunJava (SS)V，对应的数值为 0xb8 00 01；

[0131] 步骤 112：java 虚拟机用修改栈和局部变量的 java 类方法填充所述 class 文化中被移植的 java 指令的位置；

[0132] 具体为：

[0133] 步骤 112-1：java 虚拟机定位到被移植的当前 java 类方法的位置；

[0134] 步骤 112-2：java 虚拟机创建当前 java 类方法的常量池的索引；

[0135] 优选的，所述常量池的索引为两个字节，例如，所述常量池的索引为 0005，指向的是常量池中的方法为 r7javaanalyzer.R7JavaRuntime.SetLocalData (B)[B；

[0136] 所述常量池的索引为 0006，指向的常量池的方法为：invokestaticr7javaanalyzer.R7JavaRuntime.SetStackData (B)[B；

[0137] 步骤 112-3：java 虚拟机将执行修改栈和局部变量的 java 类方法与所述常量池的索引填充到被移植的当前 java 类方法的位置；

[0138] 所述修改栈和局部变量的 java 指令为 invokestatic 指令，该指令的数值为 0xD8，它需要两个字节的操作符，这两个字节的操作符即为所述常量池的索引；

[0139] 在本实施例中，在被移植的 java 指令的位置插入的内容为：invokestatic r7javaanalyzer.R7JavaRuntime.SetLocalData (B)[B，对应的数值为 0xD8 00 05；invokestatic r7javaanalyzer.R7JavaRuntime. SetStackData (B)[B，对应的数值为 0xD8 00 06；

[0140] 步骤 113：java 虚拟机更新 java 类方法的附加属性的成员信息；

[0141] 例如，由于填充的 java 指令的内容增多，因此需要修改附加属性 code 属性中的长度；

[0142] 步骤 114：所述 java 虚拟机判断已解析的 java 类方法的个数是否达到所述 class 文件中 java 类方法的总个数，如果是，则将所述预设缓冲区中的当前 java 类方法对应的 java 指令移植到加密锁内，移植结束，否则将下一 java 类方法作为当前 java 类方法，返回执行步骤 108；

[0143] 在本实施例中，填充的 java 类方法为：

```
invokestatic  r7javaanalyzer.R7JavaRuntime.GetLocalData (B)[B
invokestatic  r7javaanalyzer.R7JavaRuntime.GetStackData (B)[B
sipush        1    dirID
sipush        1    filedID
invokevirtual r7javaanalyzer.R7JavaRuntime.R7Java_RunJava (SS)V
invokestatic  r7javaanalyzer.R7JavaRuntime.SetLocalData (B)[B
invokestatic  r7javaanalyzer.R7JavaRuntime.SetStackData (B)[B
```

[0144] 。

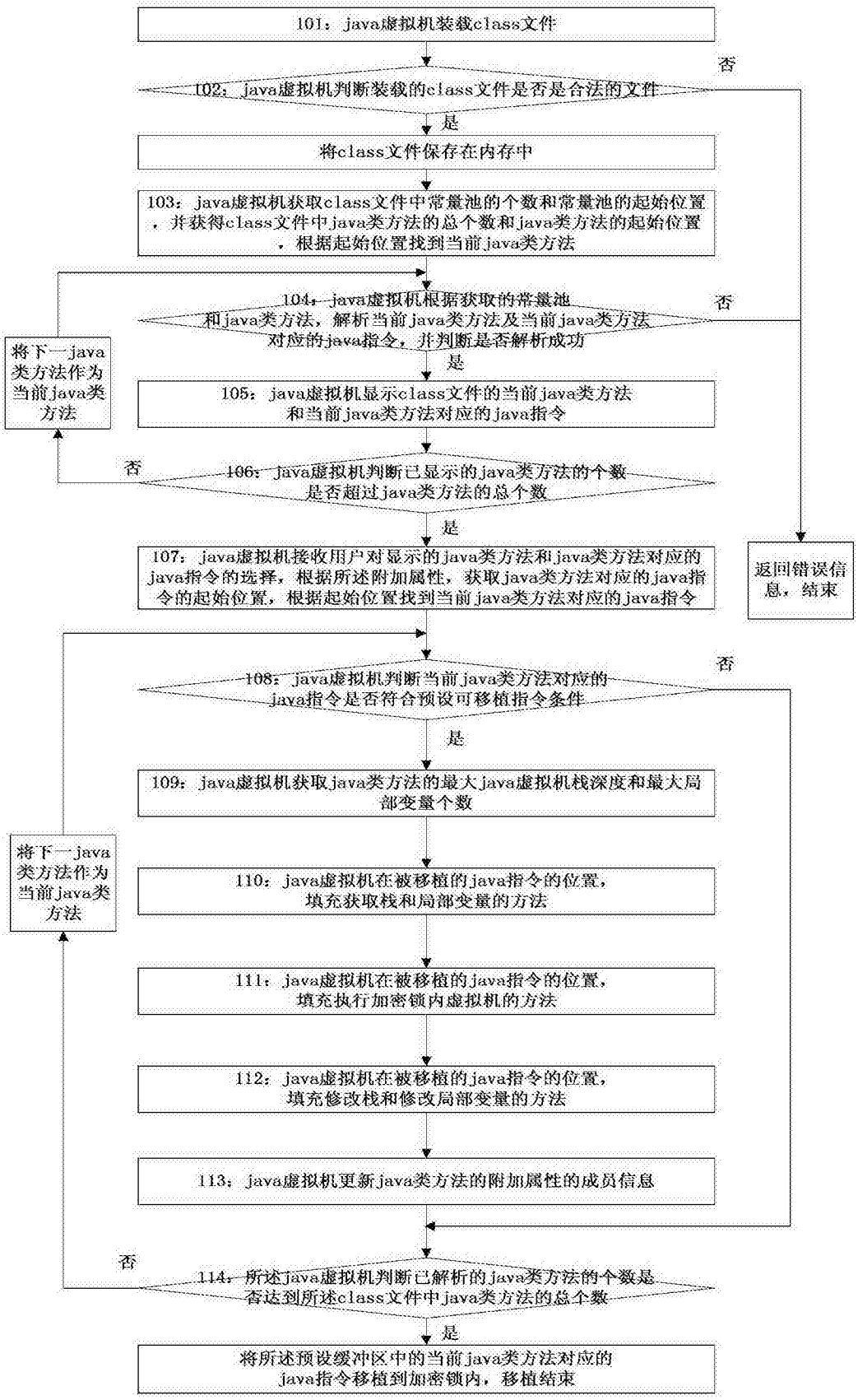

101：java虚拟机装载class文件
102：java虚拟机判断装载的class文件是否是合法的文件
否
是
将class文件保存在内存中
103：java虚拟机获取class文件中常量池的个数和常量池的起始位置，并获得class文件中java类方法的总个数和java类方法的起始位置，根据起始位置找到当前java类方法
104：java虚拟机根据获取的常量池和java类方法，解析当前java类方法及当前java类方法对应的java指令，并判断是否解析成功
否
是
将下一java类方法作为当前java类方法
105：java虚拟机显示class文件的当前java类方法和当前java类方法对应的java指令
106：java虚拟机判断已显示的java类方法的个数是否超过java类方法的总个数
否
是
107：java虚拟机接收用户对显示的java类方法和java类方法对应的java指令的选择，根据所述附加属性，获取java类方法对应的java指令的起始位置，根据起始位置找到当前java类方法对应的java指令
返回错误信息，结束
108：java虚拟机判断当前java类方法对应的java指令是否符合预设可移植指令条件
否
是
109：java虚拟机获取java类方法的最大java虚拟机栈深度和最大局部变量个数
将下一java类方法作为当前java类方法
110：java虚拟机在被移植的java指令的位置，填充获取栈和局部变量的方法
111：java虚拟机在被移植的java指令的位置，填充执行加密锁内虚拟机的方法
112：java虚拟机在被移植的java指令的位置，填充修改栈和修改局部变量的方法
113：java虚拟机更新java类方法的附加属性的成员信息
否
114：所述java虚拟机判断已解析的java类方法的个数是否达到所述class文件中java类方法的总个数
是
将所述预设缓冲区中的当前java类方法对应的java指令移植到加密锁内，移植结束

图 1

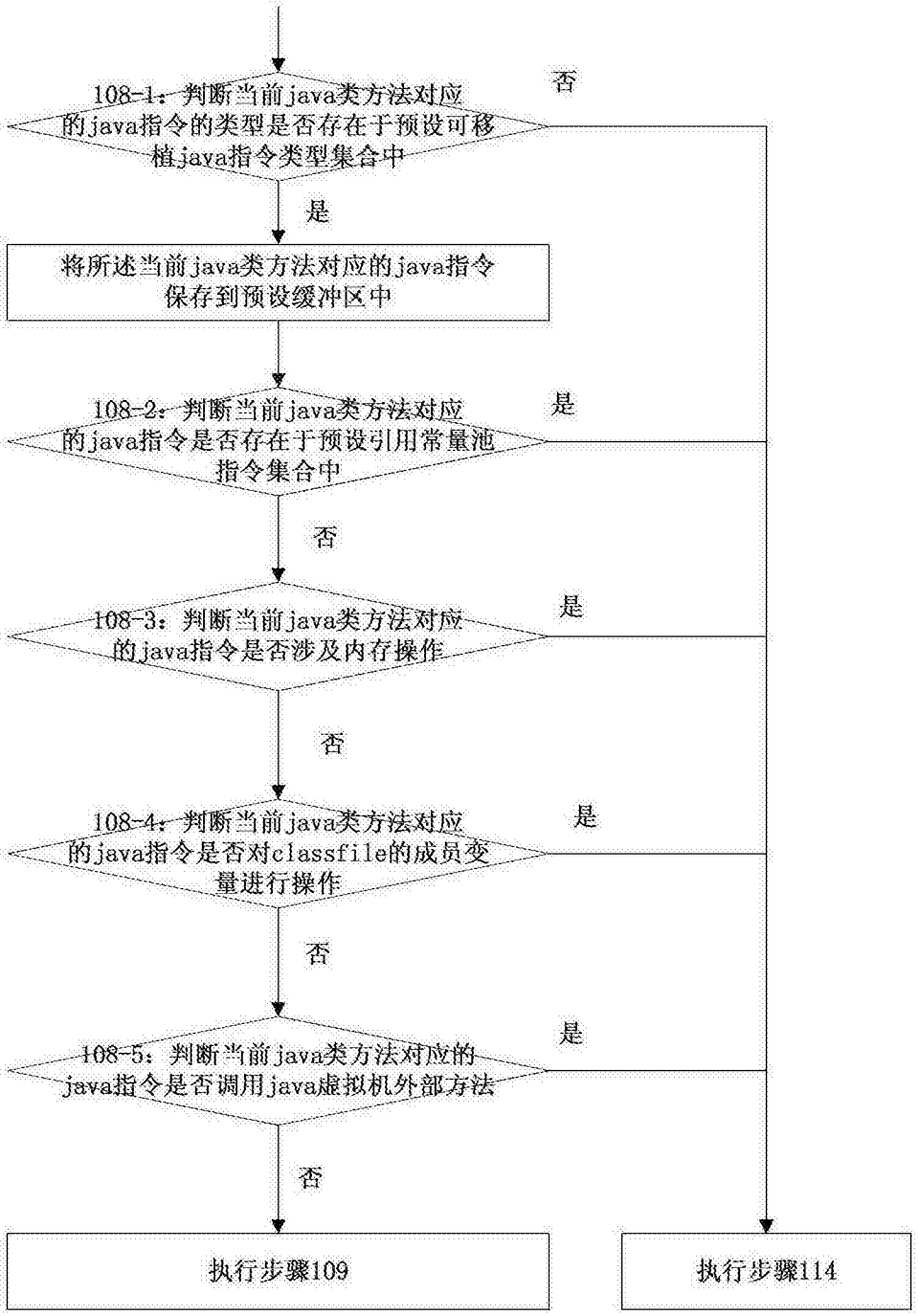

108-1：判断当前java类方法对应的java指令的类型是否存在于预设可移植java指令类型集合中
否
是
将所述当前java类方法对应的java指令保存到预设缓冲区中
108-2：判断当前java类方法对应的java指令是否存在于预设引用常量池指令集合中
是
否
108-3：判断当前java类方法对应的java指令是否涉及内存操作
是
否
108-4：判断当前java类方法对应的java指令是否对classfile的成员变量进行操作
是
否
108-5：判断当前java类方法对应的java指令是否调用java虚拟机外部方法
是
否
执行步骤109
执行步骤114

图 2