(19)国家知识产权局

(12)发明专利

(10)授权公告号 CN 112639745 B
(45)授权公告日 2025.02.25

(21)申请号 201980055768.X
(22)申请日 2019.08.23
(65)同一申请的已公布的文献号
申请公布号 CN 112639745 A
(43)申请公布日 2021.04.09
(30)优先权数据
16/112,220 2018.08.24 US
(85)PCT国际申请进入国家阶段日
2021.02.24
(86)PCT国际申请的申请数据
PCT/US2019/048037 2019.08.23
(87)PCT国际申请的公布数据
WO2020/041777 EN 2020.02.27

(73)专利权人 甲骨文国际公司
地址 美国加利福尼亚
(72)发明人 J·G·苏伊斯 R·J·奥多诺格胡 N·J·阿伦
(74)专利代理机构 中国贸促会专利商标事务所有限公司 11038
专利代理师 罗亚男

(51)Int.Cl.
G06F 11/3604(2025.01)
G06F 8/41(2018.01)
(56)对比文件
CN 101017458 A,2007.08.15
US 6658635 B1,2003.12.02
审查员 邹孝杰

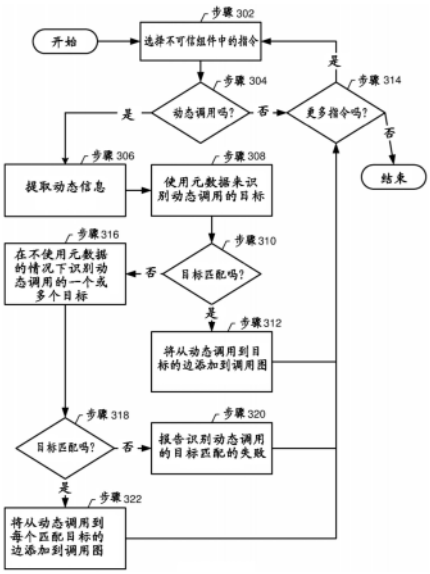
权利要求书3页 说明书10页 附图9页

(54)发明名称

动态应用的可缩放预分析

(57)摘要

一种方法可以包括将代码划分为可信组件和不可信组件,并且在代码的第一组件中识别动态调用。第一组件可以是不可信组件。该方法还可以包括从动态调用中提取动态信息,并且使用动态信息和描述代码的动态行为的元数据来识别动态调用的目标。目标可以与代码的第二组件对应。该方法还可以包括确定目标与动态调用匹配,并且响应于确定目标与动态调用匹配而将从动态调用到目标的边添加到从代码生成的调用图。



CN 112639745 B

1.一种由计算机实现的用于分析具有动态行为的代码的方法,包括:

将源代码划分为可信组件和不可信组件;

通过执行对源代码的静态分析而不执行源代码,在源代码的第一组件中识别第一动态调用,其中第一组件是不可信组件之一,其中第一动态调用对应于源代码的体系架构规范;

通过所述静态分析以及从第一动态调用中提取第一签名,第一签名包括一系列各自具有类型的参数;

通过所述静态分析并且使用第一签名和包括源代码的第二组件的标识符的第一元数据来识别第一动态调用的第一目标,其中第一目标被源代码的第二组件包括;

通过所述静态分析,通过确定第一目标实现该体系架构规范,确定第一目标包括与第一签名匹配的第二签名;以及

响应于确定第一目标包括与第一签名匹配的第二签名,通过所述静态分析将从第一动态调用到第一目标的边添加到从源代码静态生成的调用图。

2.如权利要求1所述的由计算机实现的用于分析具有动态行为的代码的方法,还包括:

用存根替换从第一组件到可信组件之一的调用。

3.如权利要求1所述的由计算机实现的用于分析具有动态行为的代码的方法,还包括:

在源代码的第三组件中识别第二动态调用;

从第二动态调用中提取第二动态信息;

使用第二动态信息和描述源代码的动态行为的第二元数据未能识别出第二动态调用的目标;

响应于未能识别出目标,使用第二动态信息和环境变量来识别第二动态调用的一个或多个目标;

确定所述一个或多个目标与第二动态调用匹配;以及

响应于确定所述一个或多个目标与第二动态调用匹配,将从第二动态调用到所述一个或多个目标中的每个目标的边添加到调用图。

4.如权利要求1所述的由计算机实现的用于分析具有动态行为的代码的方法,还包括:

在源代码的第三组件中识别第二动态调用;

从第二动态调用中提取第二动态信息;

使用第二动态信息和描述源代码的动态行为的第二元数据未能识别出第二动态调用的目标;以及

响应于未能识别,报告识别第二动态调用的目标的失败。

5.如权利要求1所述的由计算机实现的用于分析具有动态行为的代码的方法,其中将源代码划分为可信组件和不可信组件包括:

使用用户设置来识别计算机系统的文件系统中与可信组件对应的第一位置和文件系统中与不可信组件对应的第二位置。

6.如权利要求1所述的由计算机实现的用于分析具有动态行为的代码的方法,还包括:

使用调用图对源代码执行安全性分析;以及

通过所述安全性分析在源代码中识别与从第一动态调用到第一目标的边对应的安全漏洞。

7.一种由计算机实现的用于分析具有动态行为的代码的系统,包括:

储存库,被配置为存储源代码、包括源代码的第二组件的标识符的第一元数据和从源代码静态生成的调用图;

存储器,耦合到处理器;

代码拆分器,在处理器上执行并使用存储器,被配置为将源代码划分为可信组件和不可信组件;以及

静态重写器,在处理器上执行并使用存储器,被配置为:

通过执行对源代码的静态分析而不执行源代码,在源代码的第一组件中识别第一动态调用,其中第一组件是不可信组件之一,其中第一动态调用对应于源代码的体系架构规范;

通过所述静态分析以及从第一动态调用中提取第一签名,第一签名包括一系列各自具有类型的参数;

通过所述静态分析并且使用第一签名和第一元数据来识别第一动态调用的第一目标,其中第一目标被源代码的第二组件包括;

通过所述静态分析,通过确定第一目标实现该体系架构规范,确定第一目标包括与第一签名匹配的第二签名;以及

响应于确定第一目标包括与第一签名匹配的第二签名,通过所述静态分析将从第一动态调用到第一目标的边添加到调用图。

8.如权利要求7所述的由计算机实现的用于分析具有动态行为的代码的系统,还包括翻译器,该翻译器被配置为用存根替换从第一组件到可信组件之一的调用。

9.如权利要求7所述的由计算机实现的用于分析具有动态行为的代码的系统,其中静态重写器还被配置为:

在源代码的第三组件中识别第二动态调用;

从第二动态调用中提取第二动态信息;

使用第二动态信息和描述源代码的动态行为的第二元数据未能识别出第二动态调用的目标;

响应于未能识别出目标,使用第二动态信息和环境变量来识别第二动态调用的一个或多个目标;

确定所述一个或多个目标与第二动态调用匹配;以及

响应于确定所述一个或多个目标与第二动态调用匹配,将从第二动态调用到所述一个或多个目标中的每个目标的边添加到调用图。

10.如权利要求7所述的由计算机实现的用于分析具有动态行为的代码的系统,其中静态重写器还被配置为:

在源代码的第三组件中识别第二动态调用;

从第二动态调用中提取第二动态信息;

使用第二动态信息和描述源代码的动态行为的第二元数据未能识别出第二动态调用的目标;以及

响应于未能识别,报告识别第二动态调用的目标的失败。

11.如权利要求7所述的由计算机实现的用于分析具有动态行为的代码的系统,其中存储器还包括文件系统,并且其中代码拆分器还被配置为通过使用用户设置识别文件系统中与可信组件对应的第一位置和文件系统中与不可信组件对应的第二位置来将源代码划分

为可信组件和不可信组件。

12.一种非暂态计算机可读介质,包括指令,所述指令在由处理器执行时执行:

将源代码划分为可信组件和不可信组件;

通过执行对源代码的静态分析而不执行源代码,在源代码的第一组件中识别第一动态调用,其中第一组件是不可信组件之一,其中第一动态调用对应于源代码的体系架构规范;

通过所述静态分析以及从第一动态调用中提取第一签名,第一签名包括一系列各自具有类型的参数;

通过所述静态分析并且使用第一签名和包括源代码的第二组件的标识符的第一元数据来识别第一动态调用的第一目标,其中第一目标被源代码的第二组件包括;

通过所述静态分析,通过确定第一目标实现该体系架构规范,确定第一目标包括与第一签名匹配的第二签名;以及

响应于确定第一目标包括与第一签名匹配的第二签名,通过所述静态分析将从第一动态调用到第一目标的边添加到从源代码静态生成的调用图。

13.如权利要求12所述的非暂态计算机可读介质,还包括执行以下操作的指令:

用存根替换从第一组件到可信组件之一的调用。

14.如权利要求12所述的非暂态计算机可读介质,还包括执行以下操作的指令:

在源代码的第三组件中识别第二动态调用;

从第二动态调用中提取第二动态信息;

使用第二动态信息和描述源代码的动态行为的第二元数据未能识别出第二动态调用的目标;

响应于未能识别出目标,使用第二动态信息和环境变量来识别第二动态调用的一个或多个目标;

确定所述一个或多个目标与第二动态调用匹配;以及

响应于确定所述一个或多个目标与第二动态调用匹配,将从第二动态调用到所述一个或多个目标中的每个目标的边添加到调用图。

15.如权利要求12所述的非暂态计算机可读介质,其中将源代码划分为可信组件和不可信组件包括:

使用用户设置来识别计算机系统的文件系统中与可信组件对应的第一位置和文件系统中与不可信组件对应的第二位置。

动态应用的可缩放预分析

背景技术

[0001] 大规模软件应用常常包含动态生成的内容或软件组件之间的链接。例如,可以将Servlet、Java Bean和Java Server Page(JSP)组合在一起以创建动态生成的HTML页面,其中底层程序元素可以通过调用时传递的参数来公开。在对此类应用执行静态分析时,动态特点提出了挑战。例如,在不了解应用的动态行为的情况下,无法完全生成对通过应用的执行路径进行建模的调用图(例如,当执行静态分析时)。此外,不完整的调用图会干扰安全漏洞的识别。在存在动态行为的情况下对执行路径进行建模的常规方法包括静态分析工具的自定义分析和手动配置,这会导致大量的附加工作和复杂性。

发明内容

[0002] 提供本发明内容以引入概念的选择,这些概念将在下面的详细描述中进一步描述。本发明内容不旨在识别所要求保护的主题的关键或必要特征,也不旨在用于帮助限制所要求保护的主题的范围。

[0003] 一般而言,在一个方面,一个或多个实施例涉及一种方法,该方法包括将代码划分为可信组件和不可信组件,并且在代码的第一组件中识别动态调用。第一组件是不可信组件之一。该方法还包括从动态调用中提取动态信息,并且使用动态信息和描述代码的动态行为的元数据来识别动态调用的目标。目标与代码的第二组件对应。该方法还包括确定目标与动态调用匹配,并且响应于确定目标与动态调用匹配而将从动态调用到目标的边添加到从代码生成的调用图。

[0004] 总的来说,在一个方面,一个或多个实施例涉及一种系统,该系统包括被配置为存储代码、描述代码的动态行为的元数据和从代码生成的调用图的储存库,耦合到处理器的存储器,以及在处理器上执行并使用存储器的代码拆分器,该代码拆分器被配置为将代码划分为可信组件和不可信组件。该系统还包括静态重写器,该静态重写器在处理器上执行并使用存储器,该静态重写器被配置为识别代码的第一组件中的动态调用。第一组件是不可信组件之一。静态重写器还被配置为从动态调用中提取动态信息,并且使用动态信息和元数据来识别动态调用的目标。目标与代码的第二组件对应。静态重写器还被配置为确定目标与动态调用匹配,并且响应于确定目标与动态调用匹配而将从动态调用到目标的边添加到从代码生成的调用图。

[0005] 一般而言,在一个方面,一个或多个实施例涉及一种非暂态计算机可读介质,其包括指令,所述指令在由处理器执行时执行:将代码划分为可信组件和不可信组件,并且在代码的第一组件中识别动态调用。第一组件是不可信组件之一。指令还执行:从动态调用中提取动态信息,并且使用动态信息和描述代码的动态行为的元数据来识别动态调用的目标。目标与代码的第二组件对应。指令还执行:确定目标与动态调用匹配,并且响应于确定目标与动态调用匹配而将从动态调用到目标的边添加到从代码生成的调用图。

[0006] 根据以下描述和所附权利要求,本发明的其它方面将变得显而易见。

附图说明

[0007] 图1示出了根据本发明的一个或多个实施例的系统。

[0008] 图2和图3示出了根据本发明的一个或多个实施例的流程图。

[0009] 图4A、图4B、图4C、图4D、图4E和图4F示出了根据本发明的一个或多个实施例的示例。

[0010] 图5A和图5B示出了根据本发明的一个或多个实施例的计算系统。

具体实施方式

[0011] 现在将参考附图详细描述本发明的具体实施例。为了一致性,各个附图中的相同元件由相同的标号表示。

[0012] 在以下对本发明实施例的详细描述中,阐述了各种具体细节,以便提供对本发明更透彻的理解。但是,对本领域普通技术人员将显而易见的是,本发明没有这些具体细节也可以实践。在其它情况下,众所周知的特征没有详细描述,以避免不必要地使描述复杂化。

[0013] 贯穿整个申请,序数(例如,第一、第二、第三等等)可以被用作元素(即,申请中的任何名词)的形容词。序数的使用并不意味着或创建任何特定的元素排序,也不是将任何元素限制到仅单个元素,除非明确地公开,诸如通过使用术语“之前”、“之后”、“单个”及其它此类术语。相反,序数的使用是为了区分元素。作为示例,第一元素不同于第二元素,并且第一元素可以涵盖多于一个元素并且在元素的排序中在第二元素之后(或之前)。

[0014] 一般而言,本发明的实施例针对用于分析具有动态行为的代码的方法、系统和计算机可读介质。可以分析代码中的安全漏洞,包括SQL注入或跨站点脚本攻击的漏洞。在一个或多个实施例中,代码被划分为可信组件和不可信组件。分析可以集中在不可信组件上,从而改善可伸缩性。通过用存根(stub)替换对可信组件的调用,可伸缩性可以得到进一步改善。在一个或多个实施例中,分析使用在代码外部的配置代码的动态行为的元数据将动态调用与目标(例如,方法或函数)进行匹配。在一个或多个实施例中,动态调用是其运行时行为取决于代码外部的信息的语句。例如,动态调用可以与体系架构规范(例如,接口或包)对应,而目标可以实现体系架构规范。动态调用的其它示例包括反射调用(例如,Java反射工具的方法invoke)。元数据可以命名当实现体系架构规范时要使用的代码的组件(例如,类)。

[0015] 从动态调用到匹配的目标的边可以被添加到从代码生成的调用图。因此,在不执行代码的情况下,使用元数据来解析对目标的动态调用,可以使调用图更加精确和完整。如果缺少元数据或无法匹配动态调用,那么通过利用环境变量(例如,Java Classpath变量)来搜索匹配的目标,可以通过添加从动态调用到多个可能的目标的边来过度逼近调用图。

[0016] 图1示出了根据本发明的一个或多个实施例的计算机系统(100)。如图1中所示,计算机系统(100)包括储存库(102)、代码拆分器(104)、翻译器(106)、处理器(108)和静态重写器(110)。在一个或多个实施例中,计算机系统(100)采用相对于图5A和以下所附描述所描述的计算系统(500)的形式,或者采用相对于图5B所描述的客户端设备(526)的形式。在一个或多个实施例中,处理器(108)采用相对于图5A和以下所附描述所描述的处理器(502)的形式。

[0017] 在一个或多个实施例中,储存库(102)可以是用于存储数据的任何类型的存储单

元和/或设备(例如,文件系统、数据库、表的集合或任何其它存储机制)。另外,储存库(102)可以包括多个不同的存储单元和/或设备。多个不同的存储单元和/或设备可以是或可以不是相同的类型或位于相同的物理站点。

[0018] 在一个或多个实施例中,储存库(102)包括代码(112)。在一个或多个实施例中,代码(112)可以是包括各种软件组件的源代码的任何集合。即,代码(112)可以是以人类可读的编程语言编写的计算机指令(例如,语句)的任何集合。可以由编译器将代码(112)变换成二进制机器代码。然后,可以由处理器(例如,包含在计算机系统(100)中)执行经编译的机器代码,以便执行从代码(112)生成的软件组件。在一个或多个实施例中,代码(112)可以是目标代码的任何集合(例如,由编译器生成)或另一种形式的代码(112)。例如,代码(112)可以是打包经编译的组件的存档文件。

[0019] 在一个或多个实施例中,代码(112)包括可信组件(114)和不可信组件(116)。在一个或多个实施例中,可信组件(114)是代码(112)的被认为是安全的部分。库和框架可以是可信组件(114)的示例。在一个或多个实施例中,对代码(112)的静态分析可以省略对可信组件(114)的分析。在一个或多个实施例中,可信组件(114)是免除对代码(112)的分析的代码(112)的任何部分。

[0020] 在一个或多个实施例中,不可信组件(116)是代码(112)的被认为潜在不安全的部分。例如,不可信组件(116)可以包括用户定义的代码。在一个或多个实施例中,不可信组件(116)包括代码(112)的从外部源接收输入和/或向外部源发送输出的部分(例如,可能容易受到外部攻击的代码)。例如,不可信组件(116)可以包括动态生成诸如Java服务器页面(JSP)之类的网页的代码。不可信组件(116)可以是代码(112)的不属于可信组件(114)的部分的任何部分。

[0021] 在一个或多个实施例中,基于用户设置(126)来定义可信组件(114)和/或不可信组件(116)。例如,用户设置(126)可以在计算机系统(100)的文件系统中指定包括可信组件(114)的一个目录,并且在文件系统中指定包括不可信组件(116)的另一个目录。

[0022] 在一个或多个实施例中,可信组件(114)和不可信组件(116)包括动态调用(118)。在一个或多个实施例中,动态调用(118)是其运行时行为取决于代码(112)外部的信息的语句。动态调用(118)可以是反射调用,诸如由反射能力支持的方法调用,该反射能力使代码能够在运行时修改其结构和/或行为。例如,Java反射工具提供了反射调用,诸如方法invoke。在一个或多个实施例中,动态调用(118)的运行时行为取决于元数据(122)和/或环境变量(124)。在一个或多个实施例中,可以使用元数据(122)和/或环境变量(124)来解析用于动态调用(118)的目标。动态调用(118)的目标可以是方法、函数、过程等。

[0023] 在一个或多个实施例中,动态调用(118)可以与代码(112)的体系架构规范对应。体系架构规范可以指定行为,而不指定如何实现行为。在一个或多个实施例中,可以使用代码(112)的不同组件以不同方式来实现体系架构规范的行为。体系架构规范的一个示例是接口,如下面的代码片段所示:

[0024] 公共接口A{public String getMsg();}

[0025] 上面示出的接口A没有指定抽象方法getMsg的实施方式,只是指定通过任何具体方法实现getMsg都会生成字符串输出。代码中可以有不同的组件以不同的方式实现接口A。例如,不同的类可以包括实现抽象方法getMsg的不同具体方法,以生成不同的字符串输出。

实现抽象方法的具体方法可以被称为目标方法。在运行时实际实现接口A的类可以取决于外部信息(例如,元数据(122)和/或环境变量(124))。体系架构规范的另一个示例是不定义其抽象方法的实施方式的抽象类。

[0026] 在一个或多个实施例中,调用图(120)包括各自与代码(112)中的语句对应的节点。语句之间的控制流可以经由节点之间的边指示。在一个或多个实施例中,调用图(120)可以是被编译成可执行机器代码的代码(112)的中间表示的一部分。在一个或多个实施例中,可以对代码(112)的中间表示执行静态分析(例如,安全性分析)。

[0027] 在一个或多个实施例中,调用图(120)包括在每个动态调用(118)与动态调用(118)的零个或多个潜在目标之间的零个或多个边。例如,取决于诸如元数据(122)和/或环境变量(124)之类的外部信息,作为调用动态调用(118)的结果,可以在运行时调用潜在目标。

[0028] 在一个或多个实施例中,元数据(122)包括用于配置代码(112)的动态(例如,运行时)行为的任何信息。可以存在与每条元数据(122)相关联的具体过程,该具体过程被用于解析代码(112)的动态行为。在一个或多个实施例中,元数据(122)可以被用于识别动态调用(118)的目标。例如,元数据(122)可以命名在实现代码(112)的体系架构规范时要使用的组件(例如,类)。在一个或多个实施例中,元数据(122)在代码(112)的外部。例如,元数据(122)可以存储在与代码(112)分开的文件(例如,可扩展标记语言(XML)文件)中。当评估在运行时使用以实现体系架构规范的潜在目标时,元数据(122)可以与框架相关的注释(诸如Spring框架的@Autowired和@Component注释)匹配。

[0029] 在一或多个实施例中,环境变量(124)是计算机系统(100)的参数。例如,环境变量(124)可以是搜索路径(例如,Java Classpath变量),其指定计算机系统(100)的文件系统中的一个或多个位置。当尝试将类名解析为实现对应类的目标时,可以使用搜索路径。更一般而言,环境变量(124)可以指示当解析代码(112)中引用的名称时要使用的上下文。例如,环境变量(124)可以指示当解析代码(112)中引用的类名、方法名或变量名时要使用的命名空间。

[0030] 在一个或多个实施例中,用户设置(126)是可由用户配置的计算机系统(100)的参数。例如,用户设置(126)可以指定计算机系统(100)的文件系统中的一个或多个位置,以在识别可信组件(114)时使用。类似地,用户设置(126)可以指定计算机系统(100)的文件系统中的一个或多个位置,以在识别不可信组件(116)时使用。例如,用户设置(126)可以指定库和框架都是不可信组件(116)(例如,在具体静态分析的上下文中)。

[0031] 继续图1,在一个或多个实施例中,代码拆分器(104)可以以硬件(例如,电路系统)、软件、固件和/或其任何组合来实现。在一个或多个实施例中,代码拆分器(104)可以包括将代码(112)划分为可信组件(114)和不可信组件(116)的功能。代码拆分器(104)可以包括以适于进一步分析的格式(例如,类文件)表示可信组件(114)和不可信组件(116)的功能。

[0032] 在一个或多个实施例中,翻译器(106)可以以硬件(例如,电路系统),软件,固件和/或其任何组合来实现。在一个或多个实施例中,翻译器(106)可以包括将代码(112)变换成机器代码的编译器。编译器可以是计算机程序,其被设计为将以编程语言编写的源代码或中间表示变换成能够在虚拟机中执行的机器代码。在一个或多个实施例中,编译器包括

将程序的中间表示翻译成虚拟机被配置为执行的机器代码的功能。例如,编译器可以包括创建机器代码的功能,该机器代码在代替中间表示的直接执行而被执行时提高代码(112)的执行速度。在本发明的一个或多个实施例中,编译器包括在程序正在执行时执行操作或函数的动态编译的功能(例如,即时(JIT)编译)。

[0033] 在一个或多个实施例中,翻译器(106)可以包括将一个或多个不可信组件(116)转换(例如,编译)成中间表示(例如,包括调用图(120)的中间表示)的功能。在一个或多个实施例中,翻译器可以将与不可信组件(116)对应的类文件翻译成中间表示。

[0034] 继续图1,在一个或多个实施例中,处理器(108)包括执行代码(112)的功能。在一个或多个实施例中,处理器(108)包括执行代码拆分器(104)、翻译器(106)和/或静态重写器(110)的功能。

[0035] 在一个或多个实施例中,静态重写器(110)可以以硬件(例如,电路系统)、软件、固件和/或其任何组合来实现。在一个或多个实施例中,静态重写器(110)可以包括识别和分析动态调用(118)的功能。静态重写器(110)可以包括基于将一个或多个目标与动态调用(118)进行匹配而将一个或多个边添加到调用图(120)的功能。

[0036] 在一个或多个实施例中,静态重写器(110)可以包括生成错误报告(128)的功能。报告(128)可以包括没有为其找到匹配的目标的动态调用(118)的日志。报告(128)可以被存储为能够由感兴趣的实体(例如,程序员)访问的文档。

[0037] 虽然图1示出了组件的配置,但是在不背离本发明的范围的情况下可以使用其它配置。例如,各种组件可以被组合,以创建单个组件。作为另一个示例,由单个组件执行的功能可以由两个或更多个组件执行。

[0038] 图2示出了根据本发明的一个或多个实施例的流程图。该流程图描绘了用于分析代码的过程。图2中的一个或多个步骤可以由以上参考图1进行了讨论的计算机系统(100)的组件(例如,代码拆分器(104)、翻译器(106)或静态重写器(110))执行。在本发明的一个或多个实施例中,图2中所示的步骤中的一个或多个可以被省略、重复和/或并行执行,或者以与图2所示的次序不同的次序执行。因而,本发明的范围不应当被认为限于图2中所示的步骤的具体布置。

[0039] 最初,在步骤200中,将代码划分为可信组件和不可信组件。在一个或多个实施例中,代码拆分器将可信组件放置在计算机系统的文件系统中由用户设置指示的位置(例如,目录)。类似地,代码拆分器可以将不可信组件放置在文件系统中由另一个用户设置指示的位置。

[0040] 在一个或多个实施例中,翻译器可以通过用存根替换从不可信组件到可信组件的一个或多个调用来减小不可信组件的尺寸。例如,由于假定服务器内部是安全的(即,可信组件),因此翻译器可以用存根替换不可信组件中对支持Java服务器页面(JSP)的web服务器内部的调用。消除对可信组件的调用的进一步分析可以改善分析不可信组件的性能,而不会降低分析的精度或完整性。

[0041] 在步骤202中,识别代码的组件中的动态调用。在一个或多个实施例中,该组件是不可信组件。即,为了改善性能,静态重写器将其焦点限制在不可信组件中的动态调用上。

[0042] 在一个或多个实施例中,动态调用可以是反射调用,诸如Java中的方法invoke。可替代地,可以经由体系架构规范来指示动态调用,诸如Java中的接口或包。此外,可以经由

与框架相关的注释(诸如Spring框架中的@Autowired注释)来指示动态调用。例如,可以使用已知的反射调用和体系架构规范(例如,特定于编程语言)的列表以及已知的注释关键字(例如,特定于框架)的列表来识别动态调用。

[0043] 在步骤204中,从动态调用中提取动态信息。在一个或多个实施例中,动态信息包括标识符。例如,标识符可以与动态调用的潜在目标的名称对应。在一个或多个实施例中,动态信息包括签名,该签名包括一系列各自具有类型的参数。如果不能从动态调用中提取动态信息,那么静态重写器可以报告错误。

[0044] 在步骤206中,使用动态信息和描述代码的动态行为的元数据来识别动态调用的目标。目标可以与代码的组件对应。在一个或多个实施例中,元数据包括与在上面的步骤204中从动态调用中提取出的标识符匹配的代码的组件的标识符。例如,目标可以是用元数据中包括的标识符注释的类的方法或函数。

[0045] 在一个或多个实施例中,为了提高处理元数据的效率,首先对元数据进行预处理以移除与代码的动态行为不相关的信息。

[0046] 在步骤208中,确定目标与动态调用匹配。在一个或多个实施例中,当目标在动态调用的范围内潜在地可达并且目标具有与在上面步骤204中提取出的动态信息的签名匹配的签名时,目标与动态调用匹配。例如,当目标包括在实现动态调用的代码的组件中时,目标在动态调用的范围内潜在地可达。在一个或多个实施例中,当目标的参数的数量、位置和类型与动态信息的参数的数量、位置和类型匹配时,目标的签名与动态信息匹配。例如,动态信息和匹配目标都可以指定3个参数,前两个参数是整数类型,并且第三个参数是字符串类型。

[0047] 在步骤210中,响应于确定目标与动态调用匹配,将从动态调用到目标的边添加到调用图。翻译器可以从代码生成调用图。因此,静态重写器可以静态地确定(即,不执行代码)动态调用与目标之间的调用图边。

[0048] 本文所述的过程被用于在运行静态安全性分析之前对大规模应用进行预处理。添加到调用图的边启用静态分析,以避免分析无关的执行路径。此外,静态分析能够在添加到调用图的边处识别代码中的安全漏洞,因此,有可能将静态分析应用于之前由于缺乏可伸缩性和精确的分析工具而还没有被分析的应用。结果是识别出几个其它手段未检测到的安全缺陷,其误报率低。

[0049] 图3示出了根据本发明的一个或多个实施例的流程图。该流程图描绘了用于分析代码的过程。图3中的一个或多个步骤可以由以上参考图1进行了讨论的计算机系统(100)的组件(例如,代码拆分器(104)、翻译器(106)或静态重写器(110))执行。在本发明的一个或多个实施例中,图3中所示的步骤中的一个或多个可以被省略、重复和/或并行执行,或者以与图3中所示的次序不同的次序执行。因而,本发明的范围不应当被认为限于图3中所示的步骤的具体布置。

[0050] 最初,在步骤302中,选择不可信组件之一中的指令(参见上面步骤200的描述)。在一个或多个实施例中,在步骤302的相继迭代中选择不可信组件中的每个指令。在一个或多个实施例中,在步骤302中选择的指令限于包括在已知与动态调用对应的关键字列表(例如,特定于编程语言和/或框架的列表)中的条目的指令。

[0051] 如果在步骤304中确定所选择的指令是动态调用,那么执行下面的步骤306。否则,

如果步骤304确定所选择的指令不是动态调用,那么执行下面的步骤314。

[0052] 在步骤306中,从动态调用中提取动态信息(参见上面的步骤204的描述)。

[0053] 在步骤308中,使用元数据来识别动态调用的目标(参见上面的步骤206的描述)。

[0054] 如果在步骤310中确定目标与动态调用匹配(参见上面的步骤208的描述),那么执行下面的步骤312。否则,如果步骤310确定目标与动态调用不匹配,那么执行下面的步骤316。例如,当目标在动态调用范围内不可达或者目标具有与上面的步骤306中提取的动态信息不匹配的签名时,目标与动态调用不匹配。

[0055] 在步骤312中,从动态调用到目标的边被添加到调用图(参见上面的步骤210的描述)。

[0056] 在步骤314中,如果确定在不可信组件中还有尚未被选择的附加指令,那么再次执行步骤302以选择另一个指令。

[0057] 在步骤316中,在不使用元数据的情况下识别动态调用的一个或多个目标。即,在未能使用元数据识别出与动态调用匹配的唯一目标之后,静态重写器可以搜索多个匹配目标而不使用元数据。有关确定目标何时与动态调用匹配的准则,参见上面的步骤208的描述。

[0058] 可以相对于由一个或多个环境变量指定的计算机系统的文件系统中的位置来进行对匹配目标的搜索。例如,Java Classpath环境变量可以指定可以被用于搜索可以包括候选目标的类文件的目录。

[0059] 如果在步骤318中确定在上面的步骤316中识别出的一个或多个目标与动态调用匹配,那么执行下面的步骤322。否则,如果在步骤318中确定没有一个目标与动态调用匹配,那么执行步骤320。

[0060] 在步骤320中,报告识别动态调用的目标匹配的失败。静态重写器可以生成包括动态调用和元数据的错误报告。在一个或多个实施例中,匹配失败可以是由于要求在运行时提供附加的软件组件。例如,附加的软件组件可以包括动态调用的预期目标。在一个或多个实施例中,响应于匹配失败,静态重写器可以请求用户显式地指定用于动态调用的预期目标。然后,静态重写器可以添加在动态调用和用户提供的目标之间的调用图边。

[0061] 然后执行上面的步骤314以继续处理不可信组件中的指令。

[0062] 在步骤322中,从动态调用到每个匹配目标的边被添加到调用图(参见上面的步骤312的描述)。即,在不存在使用元数据识别出的唯一匹配目标的情况下,静态重写器可以通过向调用图添加与上面的步骤316中识别出的与动态调用匹配的每个目标对应的边来保守地过度逼近调用图。然后执行上面的步骤314以继续处理不可信组件中的指令。

[0063] 以下实施例仅用于解释目的,并且不旨在限制本发明的范围。图4A、图4B、图4C和图4D示出了根据本发明的一个或多个实施例的实施方式示例。

[0064] 图4A图示了对存档文件(400)的处理,该存档文件(400)聚集了代码(402)(图1中的(112))、元数据(404)((图1中的(122))和环境变量(406)(图1中的(124))。存档文件(400)包括以Enterprise ARchive(EAR)格式表示的Java Enterprise Edition(JEE)应用(例如,其它可能的格式为Java ARchive(JAR)格式、Web ARchive(WAR)格式等)。代码拆分器(410)(图1中的(104))将代码(402)划分为应用源&类文件(412)、库/框架类文件(414)和应用Java服务器页面(JSP)文件(416)。基于指示用户认为“值得信赖”的组件的用户设置,

代码拆分器(410)将应用源&类文件(412)指派给计算机系统的文件系统的app目录,将库/框架类文件(414)指派给lib目录,并且将应用JSP文件(416)提取到web目录。库/框架类文件(414)是可信组件(图1中的(114))。应用源&类文件(412)和应用JSP文件(416)是不可信组件(图1中的(116))。为了促进应用JSP文件(416)的处理,翻译器A(420a)(图1中的翻译器(106)之一)将应用JSP文件(416)翻译成Java类文件(418)。

[0065] 翻译器B(420b)从应用源&类文件(412)、库/框架类文件(414)和Java类文件(418)生成中间表示(422)。然后,如图4B、图4C和图4D中所示,静态重写器(430)(图1中的(110))使用元数据(404)和环境变量(406)来修改中间表示(422)的调用图。基于用户设置,静态重写器(430)仅分析不可信组件中的动态调用。最后,当执行静态分析(432)时使用中间表示(422)。

[0066] 图4B、图4C、图4D、图4E和图4F图示了基于Oracle应用框架(OAF)的示例,该应用框架是使用Java Enterprise Edition(JEE)技术构建的模型-视图-控制器(MVC)框架。如图4B中所示,元数据(450)识别组件类(452)ExampleAMImpl,静态重写器将使用该组件类来解析动态调用。图4C示出了包括体系架构规范(462)的代码(460),在这种情况下为基类OAApplicationModuleImpl,该基类通过两个不同的组件类(ExampleAMImpl和OtherAMImpl)进行扩展。每个组件类定义实现doSomething方法的目标(464a、464b)。当实例化应用模块时,由元数据(450)指定的组件类(452)被用于选择类ExampleAMImpl来实现基类OAApplicationModuleImpl(即,体系架构规范(462))。在图4D中图示了应用模块实例化点(466)。因此,基于由元数据(450)对组件类(452)ExampleAMImpl的指定,将应用于实例化的应用模块的doSomething方法的动态调用(468)解析为目标A(464a)。

[0067] 图4E图示了元数据(450)与动态调用(468)匹配的场景。静态重写器将连接动态调用(468)和目标A(464a)的边添加到调用图(470)。因此,无需执行代码(460),通过使用元数据(450)解析对目标A(464a)的动态调用(468),使调用图(470)更加精确和完整。

[0068] 图4F图示了元数据(450)丢失或未能与动态调用(468)匹配的替代场景。在该替代场景中,静态重写器试图将边添加到过度逼近的调用图(480)以考虑从动态调用(468)到潜在目标(464a、464b)的所有可能的问执行路径。静态重写器在代码(460)中搜索具有与动态调用(468)的名称和签名匹配的名称和签名的潜在目标(例如,具有字符串自变量和返回空值的布尔自变量的函数)。图4F图示了将连接动态调用(468)和两个目标(464a、464b)的边添加到过度逼近的调用图(480)。

[0069] 本文公开的实施例可以在计算系统上实现。可以使用移动设备、台式机、服务器、路由器、交换机、嵌入式设备或其它类型的硬件的任意组合。例如,如图5A中所示,计算系统(500)可以包括一个或多个计算机处理器(502)、非持久性存储装置(504)(例如,易失性存储器,诸如随机存取存储器(RAM)、高速缓存存储器),持久存储装置(506)(例如,硬盘、诸如光盘(CD)驱动器或数字通用盘(DVD)驱动器之类的光驱闪存等)、通信接口(512)(例如,蓝牙接口、红外接口、网络接口、光学接口等)以及许多其它元件和功能。

[0070] (一个或多个)计算机处理器(502)可以是用于处理指令的集成电路。例如,(一个或多个)计算机处理器可以是处理器的一个或多个核心或微核心。计算系统(500)还可以包括一个或多个输入设备(510),诸如触摸屏、键盘、鼠标、麦克风、触摸板、电子笔或任何其它类型的输入设备。

[0071] 通信接口(512)可以包括用于将计算系统(500)连接到网络(未示出)(例如,局域网(LAN)、诸如互联网之类的广域网(WAN)、移动网络或任何其它类型的网络)和/或另一个设备(诸如另一个计算设备)。

[0072] 另外,计算系统(500)可以包括一个或多个输出设备(508),诸如屏幕(例如,液晶显示器(LCD)、等离子显示器、触摸屏、阴极射线管(CRT)、显示器、投影仪或其它显示设备)、打印机、外部存储装置或任何其它输出设备。输出设备中的一个或多个可以与(一个或多个)输入设备相同或不同。(一个或多个)输入和输出设备可以本地或远程地连接到(一个或多个)计算机处理器(502)、非持久存储装置(504)和持久存储装置(506)。存在许多不同类型的计算系统,并且前述(一个或多个)输入和输出设备可以采取其它形式。

[0073] 执行本文公开的实施例的、计算机可读程序代码形式的软件指令可以全部或部分地,暂时或永久地,存储在非瞬态计算机可读介质上,诸如CD、DVD、存储设备、软盘、带、闪存存储器、物理存储器,或任何其它计算机可读存储介质。具体而言,软件指令可以与计算机可读程序代码对应,当其被(一个或多个)处理器执行时,被配置为执行本文公开的一个或多个实施例。

[0074] 图5A中的计算系统(500)可以连接到网络或者是网络的一部分。例如,如图5B中所示,网络(520)可以包括多个节点(例如,节点X(522)、节点Y(524))。每个节点可以与计算系统(诸如图5A中所示的计算系统)对应,或者组合的一组节点可以与图5A中所示的计算系统对应。举例来说,本文公开的实施例可以在连接到其它节点的分布式系统的节点上实现。通过另一个示例,本文公开的实施例可以在具有多个节点的分布式计算系统上实现,其中本文公开的每个部分可以位于分布式计算系统内的不同节点上。另外,前述计算系统(500)的一个或多个元件可以位于远程位置并通过网络连接到其它元件。

[0075] 虽然在图5B中未示出,但是节点可以与服务器机箱中的刀片对应,该机箱经由底板连接到其它节点。通过另一个示例,节点可以与数据中心中的服务器对应。通过另一个示例,节点可以与具有共享存储器和/或资源的计算机处理器或计算机处理器的微核心对应。

[0076] 网络(520)中的节点(例如,节点X(522)、节点Y(524))可以被配置为为客户端设备(526)提供服务。例如,节点可以是云计算系统的一部分。节点可以包括以下功能:从客户端设备接收请求(526)并向客户端设备(526)传输响应。客户端设备(526)可以是计算系统(诸如图5A中所示的计算系统)。另外,客户端设备(526)可以包括和/或执行本文公开的一个或多个实施例的全部或一部分。

[0077] 图5A和5B中描述的计算系统或计算系统组可以包括执行本文公开的各种操作的功能。例如,(一个或多个)计算系统可以在相同或不同系统上的进程之间执行通信。采用某种形式的主动或被动通信的各种机制可以促进同一设备上进程之间的数据交换。代表这些进程间通信的示例包括但不限于文件、信号、套接字、消息队列、管道、信号灯、共享存储器、消息传递和存储器映射文件的实现。下面提供了与这些非限制性示例中的几个相关的进一步细节。

[0078] 基于客户端-服务器联网模型,套接字可以用作接口或通信信道端点,从而使得能够在同一设备上的进程之间进行双向数据传送。首先,遵循客户端-服务器联网模型,服务器进程(例如,提供数据的进程)可以创建第一套接字对象。接下来,服务器进程绑定第一套接字对象,从而将第一套接字对象与唯一的名称和/或地址相关联。在创建并绑定第一套接

字对象之后,服务器进程然后等待并侦听来自一个或多个客户端进程(例如,寻找数据的进程)的传入的连接请求。此时,当客户端进程希望从服务器进程获得数据时,客户端进程将通过创建第二套接字对象而开始。客户端进程然后继续生成连接请求,该连接请求包括至少第二套接字对象以及与第一套接字对象相关联的唯一名称和/或地址。然后,客户端进程将连接请求传输到服务器进程。取决于可用性,服务器进程可以接受连接请求、与客户端进程建立通信信道,或者忙于处置其它操作的服务器进程可以将连接请求在缓冲区中排队,直到服务器进程准备就绪为止。建立的连接通知客户端进程通信可以开始。作为响应,客户端进程可以生成指定客户端进程希望获得的数据的数据请求。数据请求随后被传输到服务器进程。在接收到数据请求后,服务器进程分析请求并搜集所请求的数据。最后,服务器进程然后生成包括至少所请求的数据的答复并将该答复传输到客户端进程。数据可以更通常地作为数据报或字符流(例如,字节)被传送。

[0079] 共享存储器是指虚拟存储器空间的分配,以便证实一种机制,数据可以由多个进程进行通信和/或访问。在实现共享存储器时,初始化过程首先在持久或非持久存储装置中创建可共享的片段。在创建后,初始化过程安装该可共享的片段,然后将该可共享的片段映射到与初始化进程相关联的地址空间中。在安装之后,初始化进程继续识别并向一个或多个被授权的进程授予访问许可,这些进程也可以向可共享的片段写入数据或从可共享的片段读取数据。由一个进程对可共享的片段中的数据所做的改变可以立即影响也链接到该可共享的片段的其它进程。另外,当被授权的进程之一访问可共享的片段时,该可共享的片段映射到那个被授权的进程的地址空间。常常在任何给定时间除了初始化进程外都只有一个被授权的进程可以安装该可共享的片段。

[0080] 在不脱离本发明的范围的情况下,可以使用其它技术在进程之间共享数据(诸如本申请中描述的各种数据)。这些进程可以是相同或不同应用的一部分,并且可以在相同或不同的计算系统上执行。

[0081] 图5A中的计算系统可以实现和/或连接到数据储存库。例如,一种类型的数据储存库是数据库。数据库是为简化数据检索、修改、重组和删除而配置的信息的集合。数据库管理系统(DBMS)是为用户提供定义、创建、查询、更新或管理数据库的接口的软件应用。

[0082] 用户或软件应用可以向DBMS提交语句或查询。然后,DBMS解释语句。语句可以是请求信息的select语句、update语句、create语句、delete语句等。而且,语句可以包括指定数据或数据容器(数据库、表、记录、列、视图等)的参数、(一个或多个)标识符、条件(比较运算符)、函数(例如,联接、完全联接、计数、平均值等)、排序(例如,升序、降序)或其它。DBMS可以执行语句。例如,DBMS可以访问存储器缓冲区、对文件的引用或索引以进行读取,写入、删除或其任何组合,以响应该语句。DBMS可以从持久或非持久存储装置中加载数据并执行计算以响应查询。DBMS可以将(一个或多个)结果返回给用户或软件应用。

[0083] 以上对功能的描述仅呈现了由图5A的计算系统以及图5B中的节点和/或客户端设备执行的功能的几个示例。可以使用本文公开的一个或多个实施例来执行其它功能。

[0084] 虽然本发明已经关于有限数量的实施例进行了描述,但是受益于本公开的本领域技术人员将认识到,可以设计出不背离如本文公开的本发明的范围的其它实施例。因而,本发明的范围应当只由所附权利要求来限定。

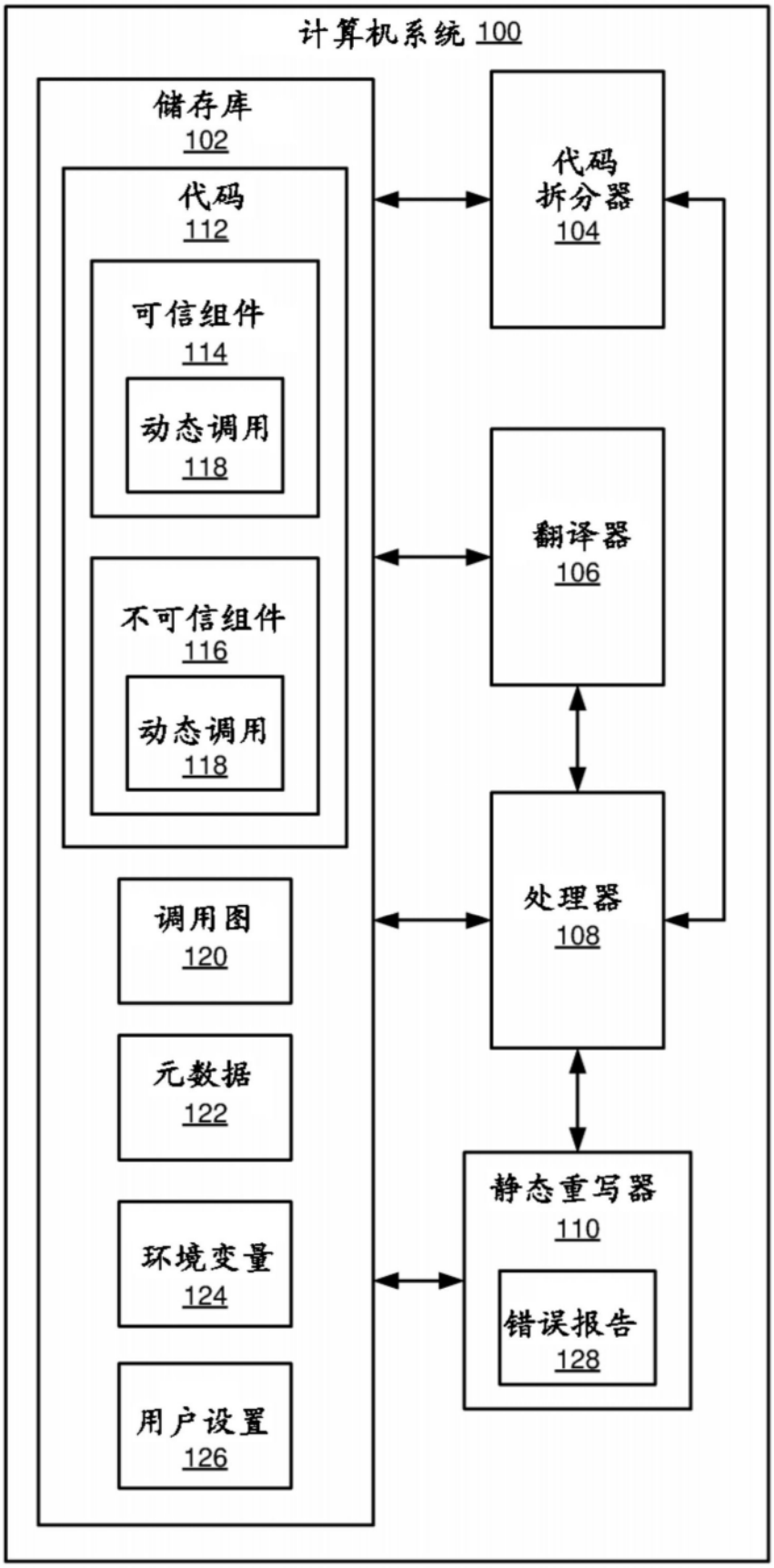

计算机系统 100
储存库
102
代码
112
可信组件
114
动态调用
118
不可信组件
116
动态调用
118
调用图
120
元数据
122
环境变量
124
用户设置
126
代码
拆分器
104
翻译器
106
处理器
108
静态重写器
110
错误报告
128

图1

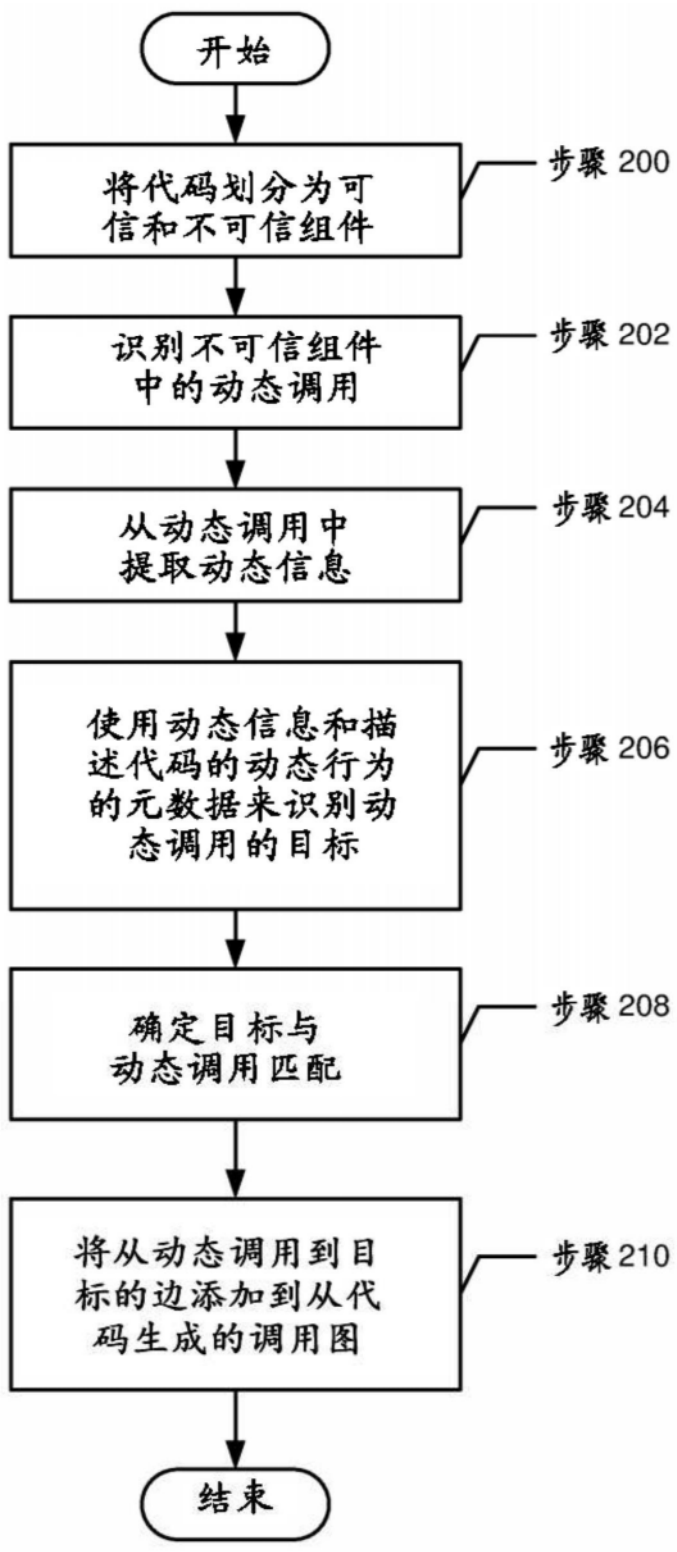

开始
将代码划分为可信和不可信组件
步骤 200
识别不可信组件中的动态调用
步骤 202
从动态调用中提取动态信息
步骤 204
使用动态信息和描述代码的动态行为的元数据来识别动态调用的目标
步骤 206
确定目标与动态调用匹配
步骤 208
将从动态调用到目标的边添加到从代码生成的调用图
步骤 210
结束

图2

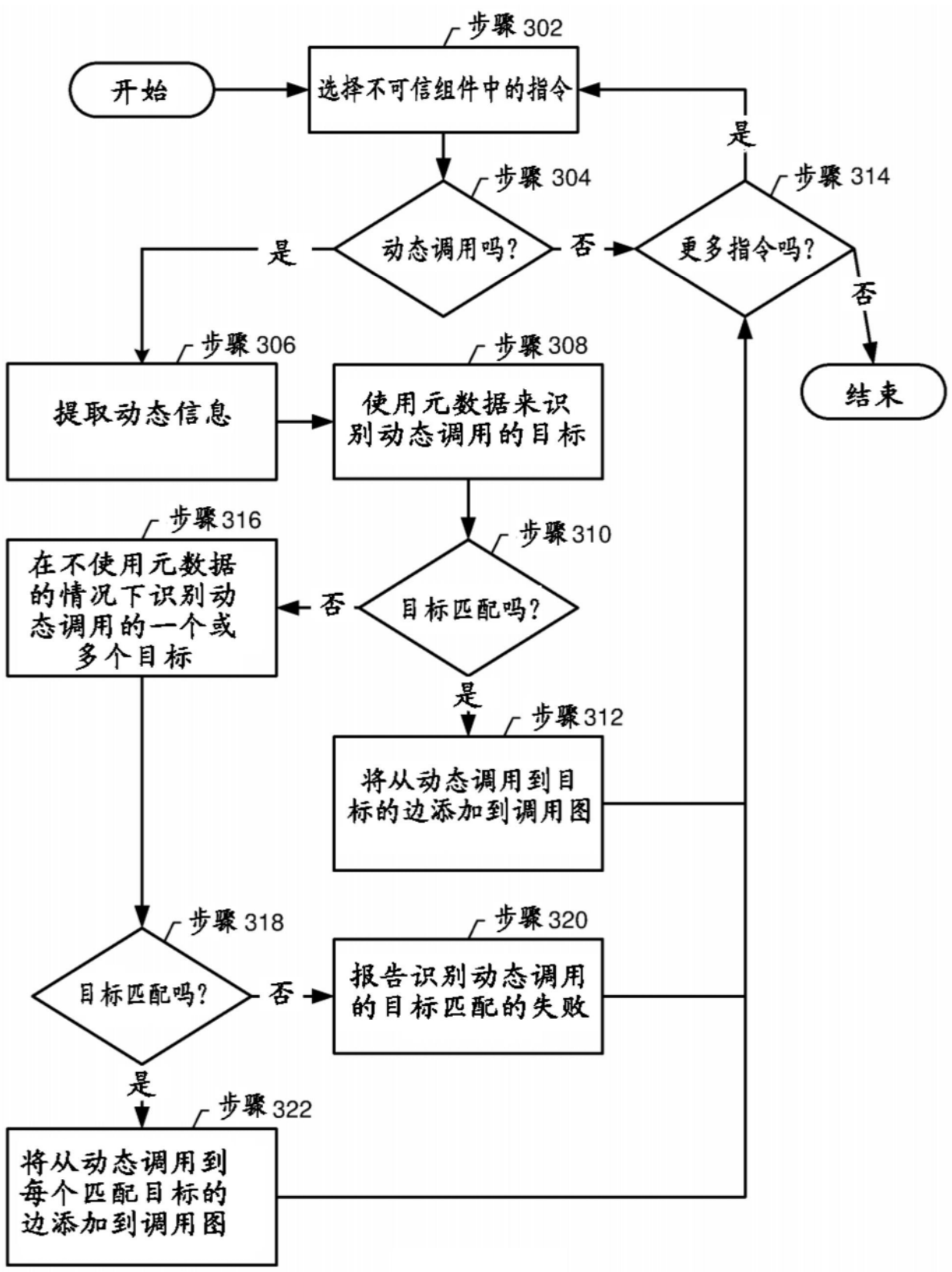

开始
步骤 302
选择不可信组件中的指令
步骤 304
动态调用吗?
是
否
步骤 306
提取动态信息
步骤 308
使用元数据来识别动态调用的目标
步骤 310
目标匹配吗?
否
步骤 316
在不使用元数据的情况下识别动态调用的一个或多个目标
是
步骤 312
将从动态调用到目标的边添加到调用图
步骤 314
更多指令吗?
是
否
结束
步骤 318
目标匹配吗?
否
步骤 320
报告识别动态调用的目标匹配的失败
是
步骤 322
将从动态调用到每个匹配目标的边添加到调用图

图3

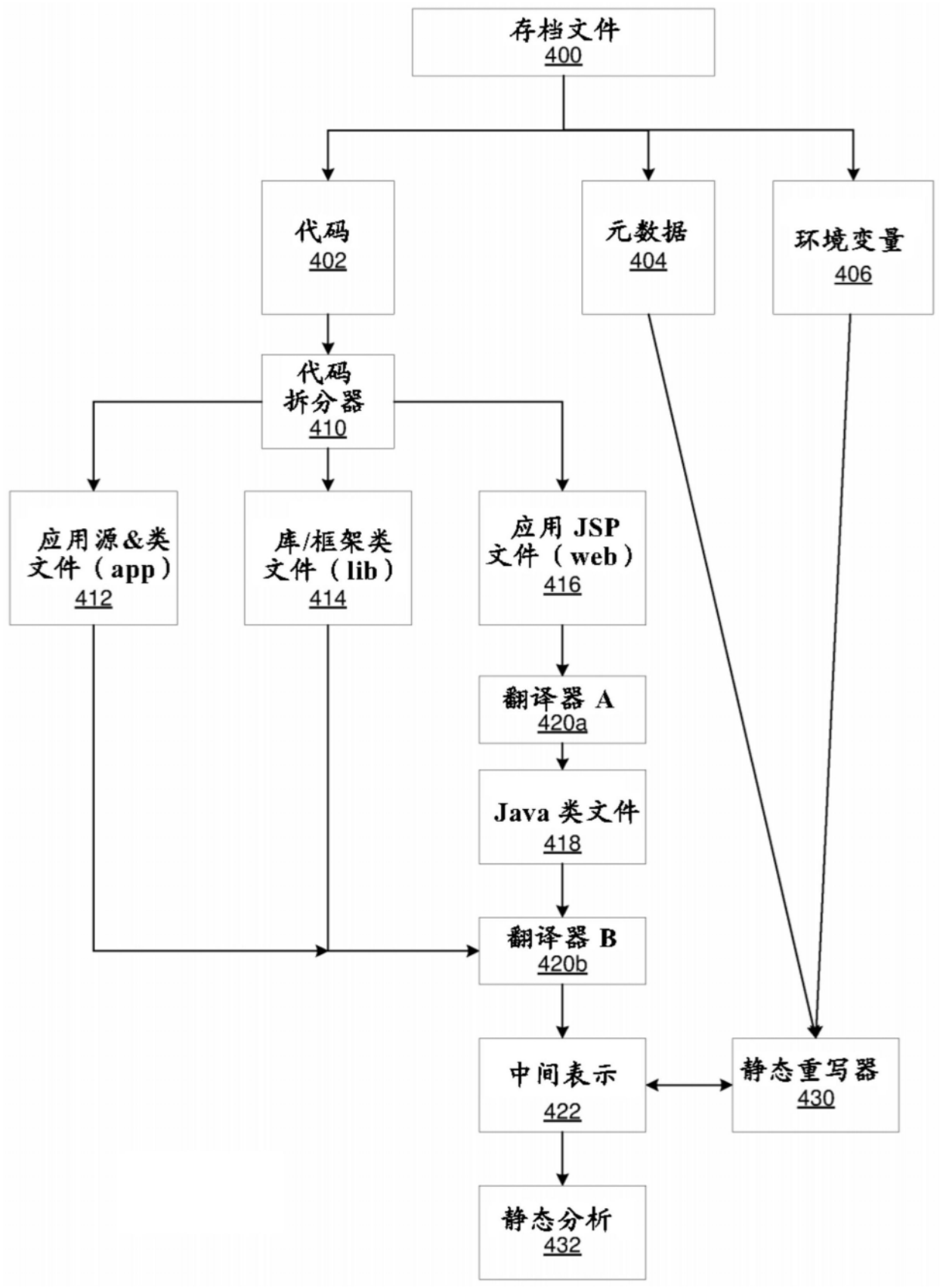

存档文件
400
代码
402
元数据
404
环境变量
406
代码
拆分器
410
应用源&类
文件（app）
412
库/框架类
文件（lib）
414
应用 JSP
文件（web）
416
翻译器 A
420a
Java 类文件
418
翻译器 B
420b
中间表示
422
静态重写器
430
静态分析
432

图4A

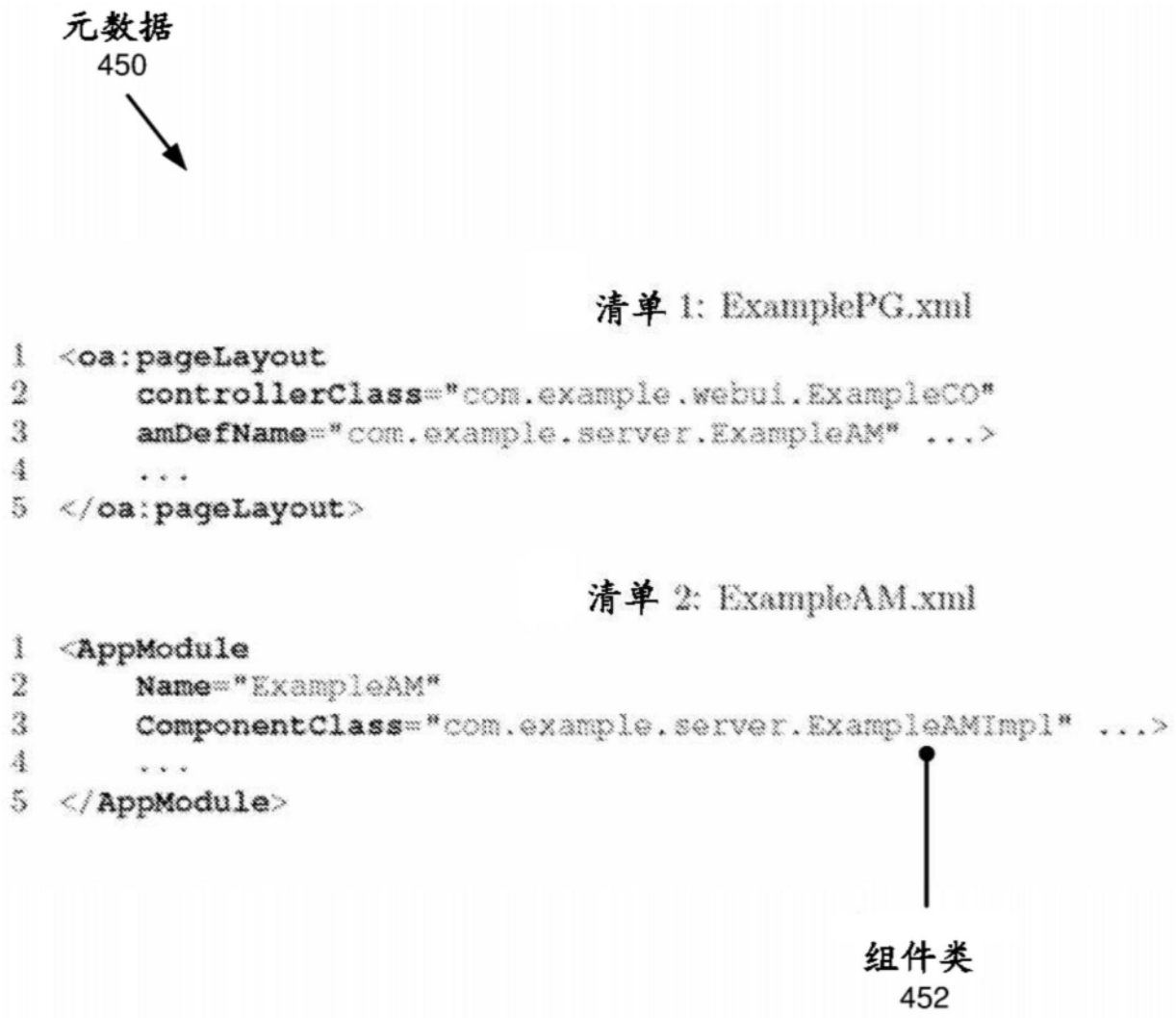

元数据
450
清单 1: ExamplePG.xml
1 <oa:pageLayout
2 controllerClass="com.example.webui.ExampleCO"
3 amDefName="com.example.server.ExampleAM" ...>
4 ...
5 </oa:pageLayout>
清单 2: ExampleAM.xml
1 <AppModule
2 Name="ExampleAM"
3 ComponentClass="com.example.server.ExampleAMImpl" ...>
4 ...
5 </AppModule>
组件类
452

图4B

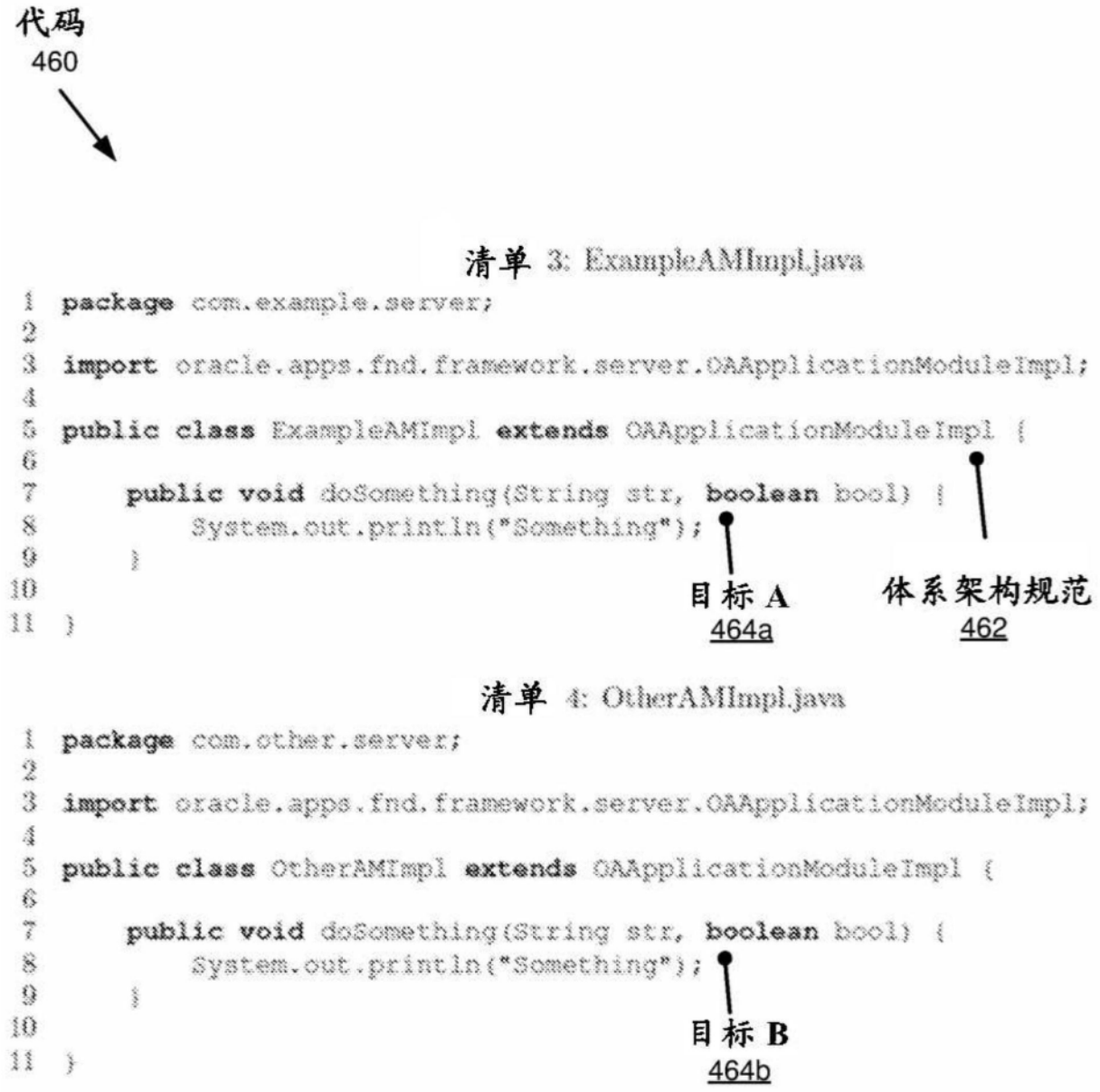

代码
460
清单 3: ExampleAMImpl.java
1 package com.example.server;
2
3 import oracle.apps.fnd.framework.server.OAApplicationModuleImpl;
4
5 public class ExampleAMImpl extends OAApplicationModuleImpl {
6
7 public void doSomething(String str, boolean bool) {
8 System.out.println("Something");
9 }
10
11 }
目标 A
464a
体系架构规范
462
清单 4: OtherAMImpl.java
1 package com.other.server;
2
3 import oracle.apps.fnd.framework.server.OAApplicationModuleImpl;
4
5 public class OtherAMImpl extends OAApplicationModuleImpl {
6
7 public void doSomething(String str, boolean bool) {
8 System.out.println("Something");
9 }
10
11 }
目标 B
464b

图4C

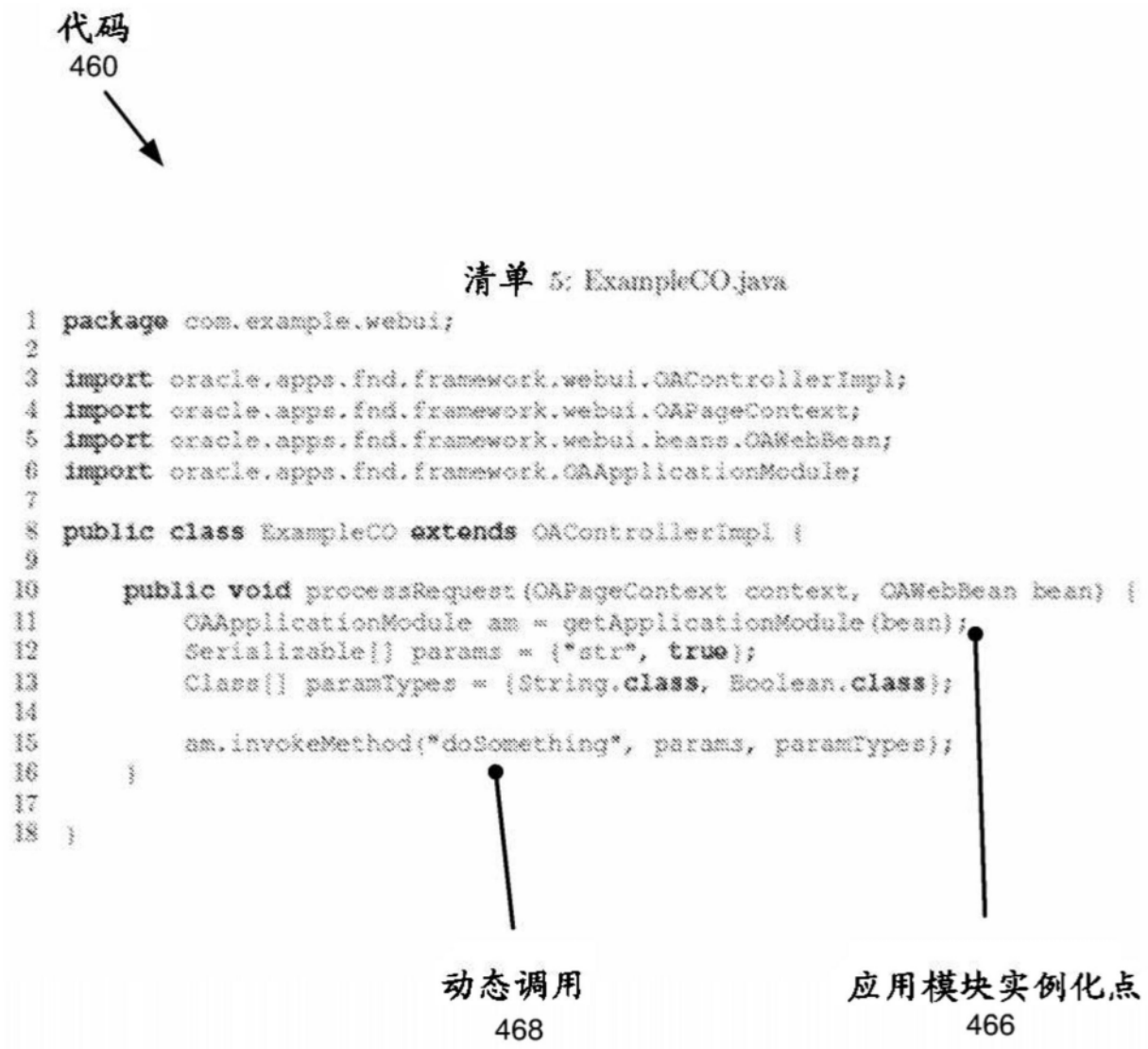

代码
460
清单 5: ExampleCO.java
1 package com.example.webui;
2
3 import oracle.apps.fnd.framework.webui.OAControllerImpl;
4 import oracle.apps.fnd.framework.webui.OAPageContext;
5 import oracle.apps.fnd.framework.webui.beans.OAWebBean;
6 import oracle.apps.fnd.framework.OAApplicationModule;
7
8 public class ExampleCO extends OAControllerImpl {
9
10 public void processRequest(OAPageContext context, OAWebBean bean) {
11 OAApplicationModule am = getApplicationModule(bean);
12 Serializable[] params = {"str", true};
13 Class[] paramTypes = {String.class, Boolean.class};
14
15 am.invokeMethod("doSomething", params, paramTypes);
16 }
17
18 }
动态调用
468
应用模块实例化点
466

图4D

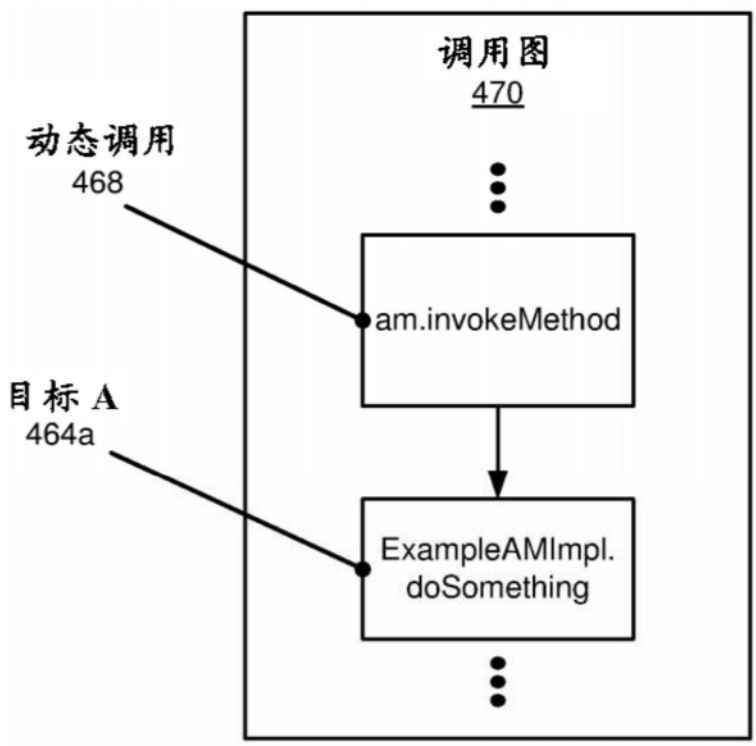

调用图
470
动态调用
468
am.invokeMethod
目标 A
464a
ExampleAMImpl.
doSomething

图4E

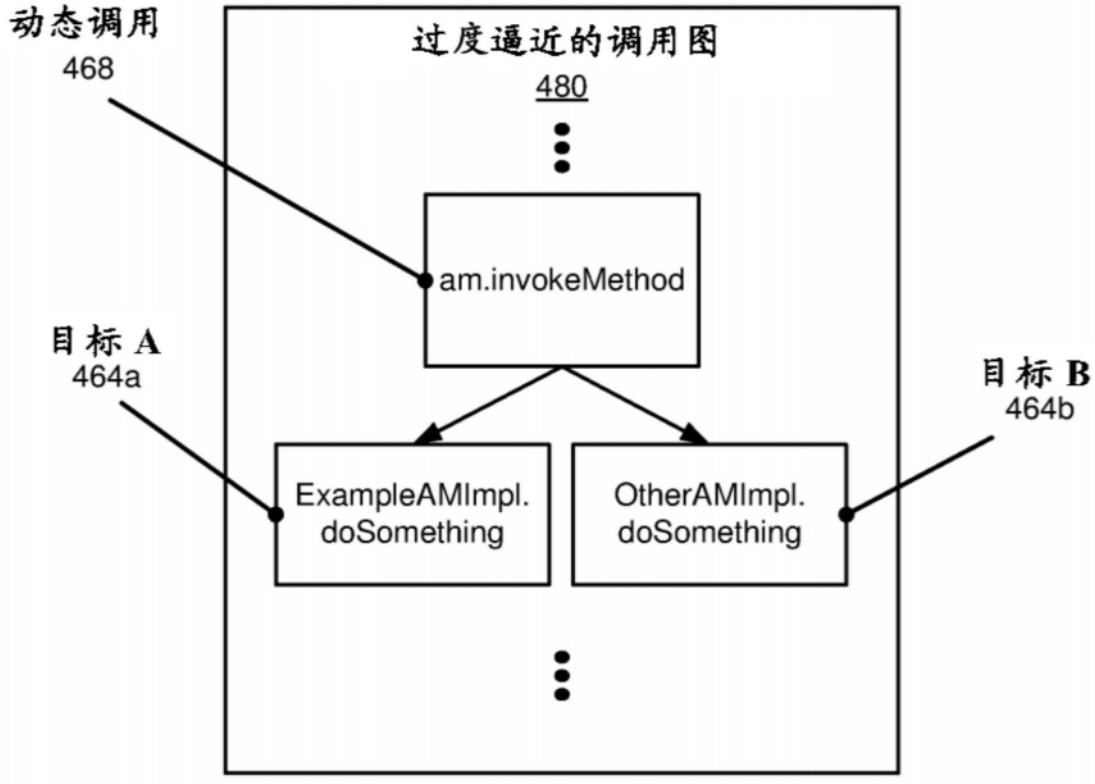

动态调用
468
过度逼近的调用图
480
am.invokeMethod
目标 A
464a
目标 B
464b
ExampleAMImpl.
doSomething
OtherAMImpl.
doSomething

图4F

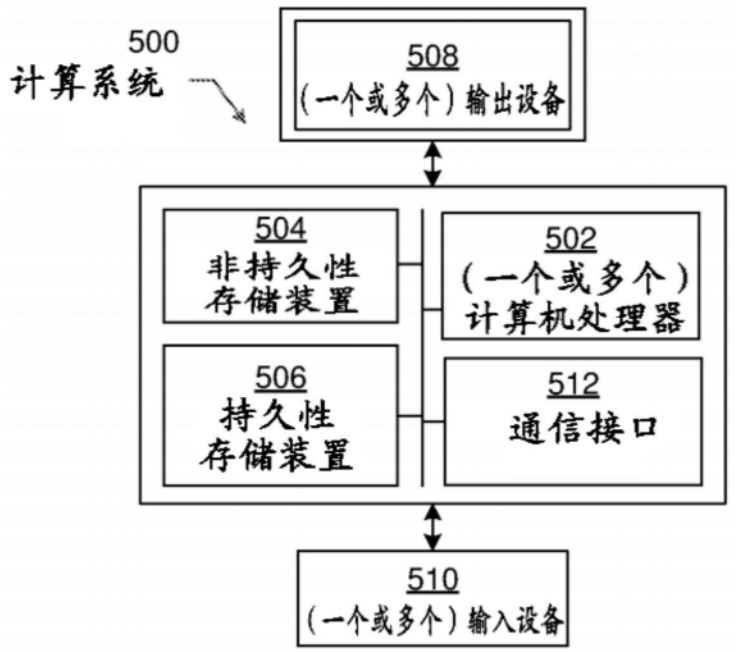

500
计算系统
508
(一个或多个)输出设备
504
非持久性
存储装置
502
(一个或多个)
计算机处理器
506
持久性
存储装置
512
通信接口
510
(一个或多个)输入设备

图5A

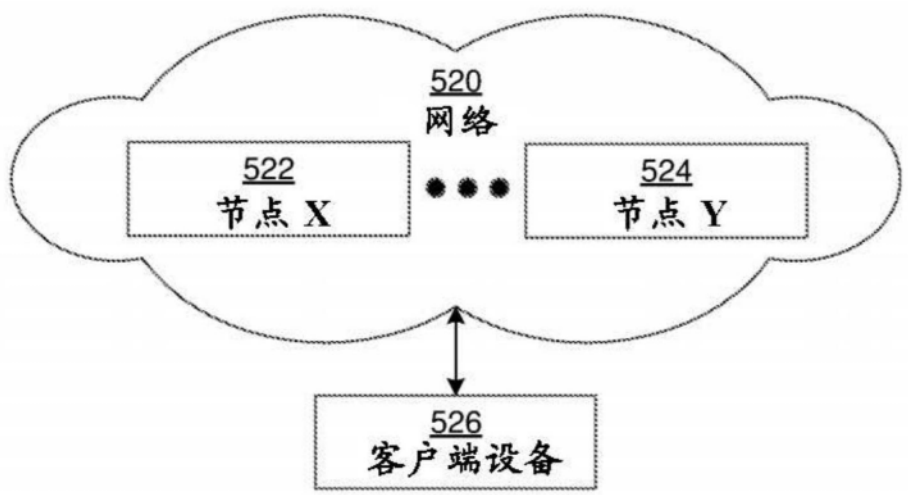

520
网络
522
节点 X
524
节点 Y
526
客户端设备

图5B