(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 104395881 B
(45)授权公告日 2017.11.28

(21)申请号 201380032776.5
(22)申请日 2013.03.14
(65)同一申请的已公布的文献号
申请公布号 CN 104395881 A
(43)申请公布日 2015.03.04
(30)优先权数据
13/487,090 2012.06.01 US
(85)PCT国际申请进入国家阶段日
2014.12.19
(86)PCT国际申请的申请数据
PCT/US2013/031335 2013.03.14
(87)PCT国际申请的公布数据
WO2013/180808 EN 2013.12.05
(73)专利权人 谷歌公司
地址 美国加利福尼亚
(72)发明人 丹尼尔·肯尼思·克利福德
雅各布·马丁·鲁迪格·库梅罗
扬·郭
(74)专利代理机构 中原信达知识产权代理有限责任公司 11219
代理人 周亚荣 安翔
(51)Int.Cl.
G06F 9/45(2006.01)
(56)对比文件
CN 1518693 A,2004.08.04,
US 7080354 B2,2006.07.18,
Hunlock.Mastering Javascript Array.《http://www.hulock.com/blogs/Mastering_Javascript_Arrays》.2007,第1-13页.
Xavier Leroy.The effctiveness of type-based unboxing.《http://pauilac.inria,fr/~xleroy/public/unboxing-tic97.pdf》.1997,第1-6页.
审查员 谢萍

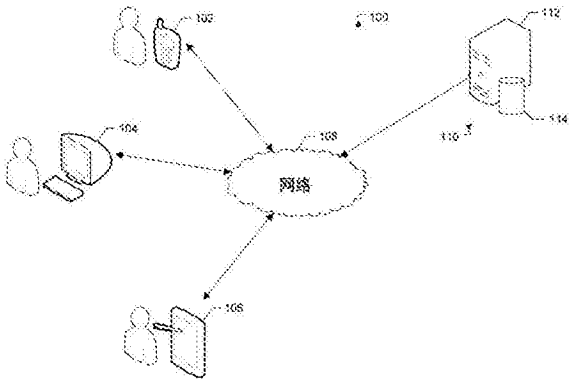
权利要求书3页 说明书12页 附图4页

(54)发明名称

动态型阵列的表达和变换

(57)摘要

本发明提供了一种运行时的动态型程序代码中阵列的表达。程序代码在运行时被访问。用于添加、更新或删除阵列的一个或多个元素的指令在代码中被检测。阵列与包括第一表示和打包形式的第一类元素相关。基于指令或一个或多个元素中的至少一种来确定第二类元素,第二类元素包括第二表达或打包形式中的至少一种。基于第一表达形式或打包形式与第二表达形式或打包形式之间的不一致性做出第一类元素和第二类元素之间不一致的确定。变换阵列,使得第一类元素和第二类元素一致。基于指令,添加、更新或删除一个或多个阵列元素。



CN 104395881 B

1.一种用于表达运行时的动态型程序代码中的阵列的机器实现的方法，所述方法包括：

在运行时访问程序代码，所述程序代码对应动态型编程语言；

在所述程序代码内，检测用于添加、更新或删除阵列的一个或多个元素的指令，其中所述阵列与第一类元素相关，所述第一类元素包括第一表达形式和第一打包形式；

基于指令或一个或多个元素中的至少一种确定第二类元素，所述第二类元素包括第二表达形式和第二打包形式中的至少一个；

基于所述第一表达形式和所述第二表达形式或所述第一打包形式和所述第二打包形式之间的不一致性，确定所述第一类元素与所述第二类元素不一致；

变换所述阵列，使得所述第一类元素与所述第二类元素一致；

以及基于所述指令添加、更新或删除变换的阵列的一个或多个元素，

其中所述第一打包形式和所述第二打包形式中的每一个与指示符值相对应，所述指示符值指示所述阵列是已打包形式还是非打包形式，在所述已打包形式下所述阵列不具有被删除的或未初始化的元素，在所述非打包形式下所述阵列具有至少一个被删除的或未初始化的元素，所述第一打包形式与所述第二打包形式不同。

2.根据权利要求1所述的方法，其中所述第一表达形式和所述第二表达形式中的每个是小整型、双精度值型或标记值型中的一种。

3.根据权利要求2所述的方法，其中所述标记值型对应于对象指针、小整数或堆数。

4.根据权利要求3所述的方法，其中所述指令用于添加或更新所述阵列的一个或多个元素，且其中所述第二表达形式基于所述一个或多个元素的数据类型。

5.根据权利要求4所述的方法，其中所述变换包括变换所述阵列，使得所述第一表达形式从小整型变为双精度值型，从小整型变为标记值型，或从双精度值型变为标记值型。

6.根据权利要求1所述的方法，其中所述指令用于删除所述阵列的一个或多个元素，或用于在所述阵列结尾处之后添加一个或多个元素，且其中所述第二打包形式是非打包形式。

7.根据权利要求6所述的方法，其中所述变换包括将所述第一打包形式从已打包形式变为非打包形式，以标记所述阵列，使得访问所述阵列的指令预先将被删除的或未初始化的元素作为可能的元素。

8.根据权利要求1所述的方法，其中所述阵列包括隐藏类数值，所述隐藏类数值表示与所述阵列相关的所述第一类元素。

9.根据权利要求1所述的方法，进一步包括：

确定所述变换与所述程序代码中的循环相关；以及

提供将所述变换吊出所述循环，使得与所述循环关联的变换不会重复发生。

10.根据权利要求1所述的方法，进一步包括：

访问关于与所述阵列相关的元素类别的信息，所述信息对应所述程序代码之前的运行；以及

减少基于被访问的信息执行确定或变换中的至少一个的次数。

11.根据权利要求1所述的方法，进一步包括：

对应于所述阵列中的空位在所述阵列中存储第一位模式的值；以及

在所述阵列中存储第二位模式的值，所述第二位模式的值是非数字的(NaN)，其中所述第一位模式不同于所述第二位模式。

12.根据权利要求1所述的方法，其中所述指令与第三类元素相关，所述第三类元素与第三表达形式相关，所述第三表达形式基于用于添加或更新一个或多个阵列的一个或多个元素的之前的指令，且其中基于所述第三类元素确定所述第二类元素。

13.一种用于表达运行时的动态型程序代码中的阵列的系统，所述系统包括：

一个或多个处理器；

以及机器可读介质，其包括存储在其中的指令，当由所述处理器执行时，所述指令促使处理器执行操作，所述操作包括：

在运行时访问程序代码，所述程序代码对应动态型编程语言；

在所述程序代码中，检测用于添加或更新阵列的一个或多个元素的指令，其中所述阵列与第一类元素相关，所述第一类元素包括第一表达形式和第一打包形式；

基于指令或一个或多个元素中的至少一种来确定第二类元素，所述第二类元素包括第二表达形式，所述第二表达形式基于所述一个或多个元素的数据类型和第二打包形式；

基于所述第一表达形式和所述第二表达形式之间或者所述第一打包形式和所述第二打包形式之间的不一致性，确定所述第一类元素与所述第二类元素不一致；

变换所述阵列，使得所述第一类元素与所述第二类元素一致；

以及基于所述指令，添加或更新变换的阵列的一个或多个元素，

其中所述第一打包形式和所述第二打包形式中的每一个与指示符值相对应，所述指示符值指示所述阵列是已打包形式还是非打包形式，在所述已打包形式下所述阵列不具有被删除的或未初始化的元素，在所述非打包形式下所述阵列具有至少一个被删除的或未初始化的元素，所述第一打包形式与所述第二打包形式不同。

14.根据权利要求13所述的系统，其中所述第一表达形式和所述第二表达形式中的每个是小整型、双精度值型或标记值型中的一种。

15.根据权利要求14所述的系统，其中所述标记值型对应对象指针、小整数或堆数。

16.根据权利要求15所述的系统，其中所述变换包括变换所述阵列，使得所述第一表达形式从所述小整型变为所述双精度值型，从所述小整型变为所述标记值型，或从所述双精度值型变为所述标记值型。

17.根据权利要求13所述的系统，其中所述指令与第三类元素相关，所述第三类元素与第三表达形式相关，所述第三表达形式基于用于添加或更新一个或多个阵列的一个或多个元素的之前的指令，且其中确定所述第二类元素基于所述第三类元素。

18.一种包括在其中存储指令的机器可读介质，当由系统执行时，所述指令使得所述系统执行操作，所述操作包括：

在运行时访问程序代码，所述程序代码对应动态编语言；

在所述程序代码中，检测用于删除阵列的一个或多个元素，或用于添加一个或多个元素至所述阵列结尾之后的指令，其中所述阵列与第一类元素相关，所述第一类元素包括第一打包形式；

基于所述指令或所述一个或多个元素中的至少一个来确定第二类元素，所述第二类元素包括第二打包形式；

基于所述第一打包形式和所述第二打包形式之间的不一致性，确定所述第一类元素与所述第二类元素不一致；

变换所述阵列，使得所述第一类元素与所述第二类元素一致；以及

基于所述指令，删除变换的阵列的一个或多个元素，或添加一个或多个元素至变换的阵列结尾之后，其中所述第一打包形式和所述第二打包形式中的每个与指示符值相对应，所述指示符值指示所述阵列是已打包形式还是非打包形式，在所述已打包形式下所述阵列不具有被删除的或未初始化的元素，在所述非打包形式下所述阵列具有至少一个被删除的或未初始化的元素，所述第一打包形式与所述第二打包形式不同。

19.根据权利要求18所述的机器可读介质，其中所述第二打包形式是所述非打包形式。

20.根据权利要求19所述的机器可读介质，其中所述变换包括将所述第一打包形式从所述已打包形式变为所述非打包形式，以标记所述阵列，使得访问所述阵列的指令预先将空位作为可能的元素。

动态型阵列的表达和变换

技术领域

[0001] 本主题技术总体涉及动态型输入程序代码，并具体涉及表达运行时的动态型程序代码的阵列。

背景技术

[0002] 许多静态型语言，如C语言和C++语言在存储器中以打包和有效的“原始”形式存储双精度浮点值(或“双精度值”)，例如使用每个占据64位存储器的双精度值(例如，使用IEEE 754 64位形式)。因为当程序在这些语言中被编译时，值的类型和因此它们的表达是已知的，编译器可生成在存储器中载入、存储并操作双精度值的有效的代码。在动态型语言中(例如JavaScript)，值的类型通常直到运行时才知道。这导致对于语言实现，难以为双精度值使用简单且有效的“原始”存储形式。

发明内容

[0003] 本公开主题涉及表达运行时的动态类型程序代码中的阵列的机器实现方法。该方法包括在运行时访问程序代码，该程序代码对应于动态型程序语言，并在程序代码内检测用于添加、更新或删除阵列的一个或多个元素的指令。该阵列与第一类元素相关，该第一类元素包括第一表达形式和第一打包形式。该方法进一步包括基于指令或一个或多个元素中的至少一种确定第二类元素，该第二类元素包括第二表达形式或第二打包形式中的至少一种，并且基于第一和第二表达形式或第一和第二打包形式之间的不一致性，确定第一类元素与第二类元素不一致。另外，该方法包括变换阵列，使得第一类元素与第二类元素一致，并且根据指令，添加、更新或删除变换的阵列的一个或多个元素。

[0004] 所述公开主题进一步涉及用于表达运行时的动态型程序代码中阵列的系统。该系统包括一个或多个处理器和机器可读介质，其包含存储于其中的指令，该指令在由处理器执行时促使处理器执行操作，操作包括在运行时访问程序代码，该程序代码对应动态型程序语言，并在程序代码内检测用于增加或更新阵列的一个或多个元素的指令。阵列与第一类元素相关，该第一类元素包括第一表达形式。操作进一步包括基于指令或一个或多个元素中的至少一种来确定第二类元素，第二类元素包括第二表达形式，其基于一个或多个元素的数据类型，并且基于第一和第二表达形式之间的不一致性，确定第一类元素与第二类元素不一致。另外，操作包括变换阵列，使得第一类元素与第二类元素一致，并且基于指令添加或更新变换的阵列的一个或多个元素。

[0005] 公开的主题还涉及机器可读介质，其包括储存在其中的指令，该指令在由系统执行时促使系统执行操作，该操作包括在运行时访问程序代码，该程序代码对应动态型编程语言，并在程序代码内检测用于删除阵列的一个或多个元素，或在阵列结尾之后处添加一个或多个元素的指令。该阵列与第一类元素相关，该第一类元素包括第一打包形式。操作还包括基于指令或一个或多个元素中的至少一种确定第二类元素，该第二类元素包括第二打包形式，以及基于第一和第二打包形式的不一致性确定第一类元素与第二类元素不一致。

另外，操作包括变换阵列，使得第一类元素与第二类元素一致。操作还包括删除变换阵列的一个或多个元素，或基于指令，添加一个或多个元素到变换阵列的结尾之后。

[0006] 应当理解，从下述具体描述中，本主题技术的其他配置对本领域技术人员应是显而易见的，其中通过说明性的方式示出并描述了主题技术的多种配置。应当意识到，本主题技术可具有其他不同配置，且在完全没有偏离本主题技术范围的情况下，可对各种细节在各种其他方面进行修改。相应地，附图及详细描述应被看作是说明性的而非限制性的。

附图说明

[0007] 主题技术的特定技术特征在随附的权利要求中加以阐述。但出于解释的目的，本主题技术的一些实施例在下列附图中进行阐述。

[0008] 图1示出了用于处理程序代码的分布式网络环境的实例。

[0009] 图2是示出了在阵列更新时影响元素类型转变的框图。

[0010] 图3是示出了在动态程序代码运行时表达阵列的流程图。

[0011] 图4概念性地示出了电子系统，使用该电子系统实现一些主题技术的实现。

具体实施方式

[0012] 下面阐述的具体描述旨在作为本主题技术不同配置的描述，且并不旨在代表可实践本主题技术的唯一形式。附图在此引入并构成具体描述的一部分。具体表述包括用于提供本主体技术充分理解的具体细节。但本领域技术人员应该清楚，所述主题技术并不限定于本文中阐述的具体细节，且可以在没有这些具体细节的情况下被实践。一些实例中，已知的结构和元件以框图的形式示出以避免与本主题技术混淆。

[0013] 许多静态型语言，如C和C++，在存储器中以打包并有效的“原始”形式存储双精度值(例如，每个双精度数值占据64位存储器)。因为当程序以这些语言编译时，值的类型和因此他们的表达是已知的。在动态型语言中，值的类型经常直到运行时才可知。这使得对语言实现来讲很难对双精度值使用简单并有效的“原始”存储形式。动态型语言包括但不限于APL、Erlang、Groovy、JavaScript、Lisp、Lua、MATLAB、GNU Octave、Perl(对用户定义类型，但并不是内嵌类型)、PHP、Pick BASIC、Prolog、Python、Ruby、Smalltalk和Tcl。

[0014] 本主题公开用于追踪阵列的表达形式(例如，小整形、双精度型或标签)，且不论阵列是否打包。标记值类型可对应对象指针、小整型或堆数。如在本文中使用的，“已打包”阵列是指不含被删除元素的阵列，而“非打包”阵列指具有被删除或否则是空元素的阵列。用于追踪阵列类型的附加信息被存储为阵列内部类型的一部分，例如，作为阵列的“隐藏类”值。

[0015] 更具体地，本主题被公开以用于表达运行时的动态程序的阵列。在运行时访问程序代码，该程序代码对应动态型编程语言。用于添加、更新或删除阵列的一个或多个元素的指令在程序代码中被检测，其中阵列与第一类元素相关，该第一类元素包括第一表达形式和第一打包形式。基于指令或一个或多个元素中的至少一种确定第二类元素，该第二类元素包括第二表达形式或第二打包形式中的至少一种。基于第一与第二表达形式或第一和第二打包形式之间的不一致性确定第一类元素与第二类元素不一致。变换阵列使得第一类元素与第二类元素一致。基于指令，添加、更新或删除变换阵列的一个或多个元素。

[0016] 图1示出了示例分布式网络环境，其可用于处理程序代码。网络环境100包括若干电子设备102-106，其通过网络108与服务器110可通信地连接。服务器110包括处理设备112和数据存储装置114。例如，处理设备112执行存储在数据存储装置114中的计算机指令以作为应用(例如网站)的主机。用户可使用电子设备102-106中的任一设备通过网络108与应用互动。

[0017] 在实例方面，可在运行时访问程序代码，其中程序代码对应动态编程语言。例如，程序代码可被任一电子设备102-106访问。用于添加、更新或删除阵列的一个或多个元素的指令在程序代码中(例如，在电子设备中)被检测，其中阵列与第一类元素相关，第一类元素包括第一表达形式和第一打包形式。基于指令或一个或多个元素中的至少一种确定第二类元素，第二类元素包括第二表达形式或第二打包形式中的至少一个。基于第一与第二表达形式或第一和第二打包形式之间的不一致性确定第一类元素与第二类元素不一致。变换阵列使得第一类元素与第二类元素一致(例如，在电子设备中)。基于指令，添加、更新或删除变换阵列的一个或多个元素。

[0018] 电子设备102-106可以是如笔记本计算机或台式计算机、智能手机、个人数字助理、便携式媒体播放器、平板计算机的计算设备或其他适合的可被使用的计算设备，例如，可使用该计算设备访问网络应用。在图1所示的实例中，电子设备102被描绘为智能手机，电子设备104被描绘为台式计算机，且电子设备106被描绘为个人数字助理。

[0019] 在一些实例方面，服务器110可以是单独的计算设备，如计算机服务器。在另一些实施例中，服务器110可代表共同工作以实现服务器计算机的行为的多个计算设备(例如，云计算)。可被用于实现服务器110的计算设备的实例包括，但不限于，网络服务器、应用服务器、代理服务器、网络式服务器或以服务器群中的计算设备组。

[0020] 在任意电子设备102-106与服务器110之间的通信可通过网络(例如，网络108)辅助进行。网络108可以是公共通信网络(例如，互联网、蜂窝数据网络、电话网络上的拨号调制解调器)或专用通信网络(例如，专用局域网，租用线路)。在任意电子设备102-106与服务器110之间的通信可通过通信协议辅助进行，如超文本传输协议(HTTP)。其他通信协议可被辅助用于任意电子设备102-106与服务器110之间的部分或全部通信，包括例如，可扩展信息和在场协议(XMPP)通信。

[0021] 图2是示出了在阵列更新时影响元素类型转变的实例的框图。如上所述，在动态型语言中，值的类型通常直到运行时才知道。这可使得对语言实现来讲难以对双精度值使用简单并有效的“原始”存储形式(如64位)。

[0022] 一种解决方法是，分配小对象以保留双精度值，并称为堆数。如本文所述，在对象中包裹原始值是指“封装”。封装的优点是其允许语言运行时间像处理其他对象一样(例如JavaScript对象)内部处理双精度值。即便当值的类型在运行时改变，程序代码引擎可生成编译函数的单独版本(例如，JavaScipt函数)其处理的数据可是对象或双精度类型。

[0023] 本方法的一个缺点是，在被存储为封装的堆数对象时，双精度值消耗RAM中的更多存储器。对于封装双精度值的大型阵列，该较高的存储器消耗是可观的。这种情况下，封装值的阵列通常使用比非封装形式的同样的阵列多两倍的存储器。

[0024] 第二个缺点是性能。在这方面，封装的双精度值的算术操作要求额外的存储器间接块以访问堆数对象的“有效载荷”(例如，64位IEEE-754有效载荷)。当新的双精度值被建

立(例如由于算数操作),必须分配新的堆数以存放结果值。该分配可能比简单存储64位双精度值慢很多。

[0025] 在主题技术的实例方面,双精度阵列的自动解封装可被用于降低双精度值大阵列的上述缺点。通过内部追踪包含在封装值阵列内的值类型,可以识别并利用一些情况,这样当双精度阵列自动解封装时可降低存储器消耗并提高性能。在这些情况中,通过追踪在运行时存储在阵列中的值,可将阵列变换为优化的存储形式(例如,小整型、双精度值或标签阵列)。

[0026] 在不改变语言的语义的情况下,该变换降低了由双精度阵列消耗的存储器,并允许编译器(例如,实时(JIT)编译器)产生有效的特殊编码,其直接操作没有封装值的阵列。

[0027] 本主题公开提供了追踪阵列元素表达形式的类型(例如,小整型、双精度值或标记值)和阵列的打包形式的类型(例如,已打包和未打包)。该附加信息存储在阵列的隐藏类值中。如本文中使用的,用于追踪阵列类型和阵列形式的附加信息是指阵列的“元素类型”。应该注意,隐藏类值不限于存储阵列的元素类型,且可包含附加信息。在这一方面,隐藏类的检查可访问“元素类型”值或隐藏类数值包含的任意其他信息。

[0028] 小整型值(SMI)可在不带有附加分配的情况下被存储在对象值阵列中,这是因为SMI在他们最低有效位通常不同于指向对象的指针。具体地,对于SMI,较低位是清空的,且对于对象指针(例如,包括堆数指针),较低位是被设置的。因为许多阵列初始只包含SMI(例如,阵列初始只包含零)且随后可被双精度值占据,只含SMI的阵列被使用元素类型值追踪,使得他们可以在双精度值存储至其中时被优化。

[0029] 在实例方面,参考图2,元素类型值被设置为下述数值中的一个:

[0030] SMI_Elements_Packed 206

[0031] SMI_Elements_Nonpacked 208

[0032] Double_Elements_Packed 210

[0033] Double_Elements_Nonpacked 212

[0034] Tagged Elements Packed 214

[0035] Tagged Elements Nonpacked 216

[0036] 在这方面,具有元素类型SMI_Elements_Packed 206的阵列必须只包括小整型数值(SMI)。具有元素类型SMI_Elements_Nonpacked 208的阵列必须只包括SMI和“空位”标记值,其表示删除的或未初始化的阵列值。例如,值得使用空位追踪阵列的至少一个原因是通过优化含有空位的阵列编译器产生的代码通常比不含空位的阵列代码执行更多工作。

[0037] 具有元素类型Double_Elements_Packed 210的阵列只包含双精度值。具有元素类型Double_Elements_Nonpacked 212的阵列只包含双精度值和“空位”标记值。

[0038] 具有元素类型Tagged Elements Packed 214的阵列可包含任意对象指针,包括SMI、堆数和对象。具有元素类型Tagged Elements Nonpacked 216的阵列可包含除原始双精度值以外的任意类型元素,并且包括“空位”标记值。

[0039] 图2的实例中,用于生成或分配阵列的指令202可与任意元素类型值相关联。在这方面,当可以从特定的对阵列构造器(例如,JavaScript中的“Array()”)的注释中确定优化的阵列形式时,在由阵列构造器或阵列标示符分配时确定阵列的初始元素类型。否则其被设置为Tagged_Elements_Nonpacked 216。空阵列起始具有包含SMI_Elements_Packed

206元素类型的隐藏类数值。程序代码实例(例如,JavaScript)如下:

[0040] var a=new Array();//has SMI_Elements_Packed 206

[0041] var b=new Array(5);//has SMI_Elements_Nonpacked 208

[0042] var c=[2.5,0.5,10.2,3.5];//has Double_Elements_Packed 210

[0043] var d=[2.5,,,3.5];//has Double_Elements_Nonpacked 212

[0044] var e=new Array(65,new Object(),3.5);//has Tagged_Elements_Packed 214

[0045] var f=[2.5,,,new ObjectQ];//has Tagged_Elements_Nonpacked 216

[0046] 如图2所示,元素类型的转变段形成点阵:

[0047] SMI_Elements_Packed 206→SMI_Elements_Nonpacked 208

[0048] SMI_Elements_Packed 206→Double_Elements_Packed 210

[0049] SMI_Elements_Packed 206→Tagged Elements Packed 214

[0050] SMI_Elements_Nonpacked 208→Double Elements Nonpacked 212

[0051] SMI_Elements_Nonpacked 208→TaggedJElements Nonpacked 216

[0052] Double_Elements_Packed 210→Tagged Elements Packed 214

[0053] Double_Elements Packed 210→Double_Elements_Nonpacked 212

[0054] Double_Elements_Nonpacked 212→Tagged Elements Nonpacked 216

[0055] Tagged_Elements Packed 214→Tagged_Elements_Nonpacked 216

[0056] 随着在运行中将元素添加至阵列(例如,通过指令204),如果添加的表达形式与阵列当前元素类型不一致,阵列的元素类型被转变(例如,被转变)为较不具体的元素类型。因为直接的转变段是非周期的,在到达定点元素类型Tagged Elements Nonpacked 216之前,存在有限数量的可能的阵列转变。

[0057] 当从SMI_Elements_Nonpacked 208或SMI_Elements_Nonpacked 206转变阵列至元素类型Double_Elements_Nonpacked 212或Double_Elements_Packed 210时,其元素被转变为它们的压缩非封装双精度表达。当阵列从Double_Elements_Nonpacked 212或Double_Elements_packed 210转变至Tagged_Elements Nonpacked 216或Tagged_Elements_Packed 214时,阵列元素转变(回)封装形式,包括对所有其包含的双精度值分配堆数。程序编码的实例(例如,JavaScript)如下:

```
var a = new Array();      // has SMI_Elements_Packed 206
a[0] = 1;                 // still SMI_Elements_Packed 206
a.push(5);                // still SMI_Elements_Packed 206
a[2] = 2.5;               // transition to Double_Elements_Packed 210, including
                          // conversion to unboxed double format
a[25] = 4;                // transition to Double_Elements_Nonpacked 212 because
                          // array is no longer packed
a[4] = new ObjectQ;       // transition to Tagged_Elements_Nonpacked 216.
                          // Unboxed format converted to boxed format again,
                          // including allocating heap numbers for all doubles in the
                          // array.
```

[0058]

[0059] 在实例方面，为了使用元素类型信息以产生更快的编码，可以使用两个编译器（例如，两个实时编译器），包括非优化编译器和优化的编译器。例如，程序代码（例如，JavaScript代码）初始时被编译成未优化的具体结构的汇编码，且关键的性能代码随后被优化编译器重新编译，非优化编码包含额外的检查，检查是否元素类型转变在每个阵列中的存储都是必要的。非优化编码还在载入或存储至其中时记录阵列的元素类型。在这方面，非优化编译器在运行时执行（例如，作为快速编译器），并与运行非优化编码过程中收集的反馈类型相适应。该反馈类型使优化变为可能。

[0060] 当具有阵列访问码的函数（例如，JavaScript函数）被优化时，优化编译器基于在运行非优化代码时收集的元素类别信息来释放特定编译码。例如，该特殊编码通过隐藏类数值检查被禁止以确保阵列具有期望的元素类型。对具有Double_Elements_Nonpacked 212和Double_Elements Packed 210元素类型的阵列，优化编码直接在双精度值的非封装表达中访问双精度值。对已知具有打包的元素类型的阵列（例如，SMI_Elements_Packed 206，Double_Elements_Packed 210，Tagged_Elements Packed 214），优化编译器可省略对阵列的空位检查。优化编译器只为要求非优化码中转变的阵列释放阵列元素转变码以存储。由于转变可能是不常见的，普通情况下，生成的具有Double_Elements_Packed 210元素类型的阵列优化码是打包的并接近使用C和C++编写的双精度阵列代码的效率。

[0061] 尽管不常见，但阵列元素在运行时的转变可能计算起来是昂贵的。因此，阻止在关键性能代码中的阵列转变是有利的。在第一实例中，元素类型转变可被吊出循环，这取决于元素类型转变只在第一次执行时改变阵列的内容且对后续执行没有影响。编程代码的实例（例如，JavaScript）如下：

[0062]

```
var a = new Array(); // ← a has SMI_Elements_Packed 206
a[0] = 0; a[1] = 0; a[2] = 0; a[3] = 0; a[4] = 0; // Still SMI_Elements_Packed 206
for(i=0;i<5;i++) {
  a[i] = i + .5; // a transitions to Double_Elements_Packed 210, but
                 // hidden-class check and transition are hoisted out of loop.
}
```

[0063] 在第二实例中，如果可能，一旦收集了被指定至阵列的元素的信息，阵列可通过使用正确的元素类型注释阵列的分配地点而被“预转变”，并防止进一步的元素类型在生成的代码中的转变。

[0064] 在实例中的方面，当前指令(例如，指令202)可基于之前的阵列指令与元素类型相关联，特别地，对于之前添加或更新阵列元素的指令。例如，如果之前在程序代码中的指令导致第一阵列的转变(例如，从SMI形式转变到双精度形式)，当前的第二阵列指令可与双精度关联，而无论存储当前指令的数值的实际数据类型(例如，SMI)。这样，即使添加或更新的数值是SMI值，第二阵列也可转变(例如，从SMI形式变为双精度形式)。

[0065] 在另一个实例中，双精度值在其“原本的”64位IEEE编码中被表示。

[0066] 在可选的系统中，对象指针由双精度非数字(NaN)数值表示，其中指针地址以未定义的较低52位NaN数值被编码。所有阵列被表示为64位元素矢量。尽管该系统可以在存储值至阵列时避免双精度值的堆数的存储器分配，但其通常要求值类型的运行检查，以确定是否将值作为双精度或另外的类型处理。另外，在32位指令结构上，为阵列分配不包含双精度值的64位元素可能是浪费的，其消耗两倍于上述方法的存储器。

[0067] 图3是示出了在动态程序代码运行时表达阵列的实例的流程图。跟随起始框300，程序代码在运行时在302被访问。该程序代码对应动态编程语言。

[0068] 在步骤304，用于添加、更新或删除一个或多个阵列元素的指令在程序代码中被检测。阵列与第一类元素(例如，第一元素类型)相关，第一类元素包括第一表达形式和第一打包形式。阵列可包括隐藏类数值，其表示了与阵列关联的第一类元素。

[0069] 在步骤306，基于指令或一个或多个元素中的至少一个，确定第二类元素(例如，第二元素类型)，第二类元素包括至少第二表达形式或第二打包形式中的一个。第一和第二表达形式中的每个可以是小整型、双精度值型或标记值型中的一个。该标记值类型可对应于对象指针、小整数或堆数。第一和第二打包形式中的每个可以是对应于不含空位的阵列的打包形式，或对应于具有至少一个空位的阵列的非打包形式。在这方面，具有非打包形式且恰好不包含任何空位的阵列是可能的。但是，对于具有非打包形式的阵列，访问代码必须检查空位(然后，具有打包形式的阵列是保证没有空位的，且没有实施检查的)。

[0070] 指令可以是添加或更新一个或多个阵列元素的，且第二表达形式可以是基于一个或多个元素的数据类型。另外，指令可以用于删除一个或多个阵列元素，或用于添加一个或多个元素至超出阵列结尾处，且第二打包形式可以是非打包形式。

[0071] 在步骤308，确定第一类元素是否与第二类元素相关。该确定是基于第一和第二表达形式之间的一致性的，且基于第一与第二打包形式之间的一致性。如果第一类元素与第二类元素不一致，那么变换阵列，使得第一类元素与第二类元素在步骤310处一致。

[0072] 在实例方面，如果指令用于添加或更新，变换可包括变换阵列使得第一表达形式从小整型变为双精度型，从小整型变为标签型，或从双精度型变为标签型。在另一实例方面，如果指令用于删除，那么变换可包括将第一打包形式从打包形式变为非打包形式以标记阵列，使得访问它的指令期待空位作为可能的元素。

[0073] 应当注意将表达形式和打包形式转变可同时发生。不仅在删除时，而且在添加元素至前面阵列结尾处之外，使得在其和现存元素间具有空位时，打包形式可转变为非打包形式。编程代码(例如，JavaScript)的实例如下：

[0074] var a=[1,2,3];//SMI_Elements_Packed

[0075] a[25]=2.5;//transition to Double_Elements_Nonpacked

[0076] 在步骤312,基于指令,为变换的阵列添加、更新或删除一个或多个元素。该处理随后在方框314处结束。在实例方面,可以确定变换与程序代码中的循环相关,且变化可被吊出循环,使得与循环相关的变化不会反复出现。在另一个实例方面,与和阵列关联的元素类型相关的信息可以被评估,信息对应前面运行的程序代码,且至少执行确定或变换中的一个的次数可以基于评估信息而减少。

[0077] 但是,在另一个实例方面,第一位模式可被存储用于阵列中对应阵列中的空位的值,且第二位模式可被存储用于阵列中为非数字(NaN)的值,其中第一位模式不同与第二位模式。例如,根据IEEE-754,对于NaN存在多种的位模式。第一位模式是如NaN(并不是其他任意位模式)。因此,通过仅使用单独的NaN位模式(第二位模式)来表示NaN,可以避免空位与实际NaN之间的混淆。

[0078] 但在另一实例方面,指令与元素的第三类相关,元素的第三类与第三表达形式相关。第三表达形式基于之前的用于添加或更新一个或多个阵列中的一个或多个元素的指令。基于元素的第三类确定第二类元素。编程代码的实例(例如,JavaScript)如下:

```
function store_value(x, val) {
x[0] = val; // The first call to this function has a double value in
            // parameter val and converts the array x to
            // Double Elements Packed, furthermore marking the store
            // instruction to always convert SMI_Elements_Packed
            // arrays to Double_Elements_Packed, even if val is an
            // SMI. The third call to this function has an array x with
            // ElementsKind Object Elements Packed and marks
            // the store instruction to always covert arrays with
            // ElementsKind SMI_Elements_Packed or
            // Double Elements Packed to
            // Object_Elements_Packed, regardless of the type of
            // parameter val or the ElementsKind of x.
}
var x = [0];          // x is SMI_Elements_Packed
store_value(x, 2.5); // x is converted to Double_Elements_Packed inside the
                      // function call due to the element value being of type
                      // double.
x = [0];              // x is SMIJElements_Packed
store_value(x, 10);  // x is converted to Double_Elements_Packed, even though
                      // the stored element is an SMI.
x = [new Object()]; // x is Object Elements Packed
store_value(x, 10);
x = [0];              // x is SMI_Elements_Packed
store_value(x, 10);  // x is converted to Object_Elements_Packed
```

[0079]

[0080] 许多上述特征与应用作为软件处理实现，软件处理被定义为一系列记录在计算机可读存储介质(也指计算机可读介质)上的指令。当这些指令由一个或多个处理单元(例如，一个或多个处理器，处理器的核，或其余处理单元)执行时，它们促使处理单元执行指令指明的操作。计算机可读介质的实例包括，但不限于光盘只读存储器、闪存、读写存储器、硬盘、可擦可编程只读存储器等。计算机可读介质不包括无线或通过有线传播的载波和电子信号。

[0081] 在本说明书中，术语“软件”旨在包括位于只读存储器的固件或存于磁存储器中的应用，其可被读入存储器由处理器中处理。另外，在一些实现中，本主题公开的多软件方面可被作为大型程序的子部分实现而保留本主题公开的明确的软件方面。在一些实现中，多软件方面可作为分开的程序来实现。最后，任意分开的程序的组合，其共同实现本文描述的软件方面包括在本主题公开的范围内。在一些实现中，软件程序，当被安装以执行一个或多

个电子系统时，定义了一个或多个具体的机器实现，其执行并实现软件程序的操作。

[0082] 计算机程序(也称作程序、软件、软件应用、脚本或代码)可以任何编程语言编写，包括编译语言或翻译语言，陈述性或过程化语言，且其可以被以任何形式采用，包括作为独立的程序或作为模块、元素、子程序、对象或其它适于在计算环境下使用的单元。计算机程序可以，但不必须对应于文件系统中的文件。程序可以被存储在文件的一部分中，其保留了其它程序或数据(例如，一个或多个数据存储器在标记语言文档中)在单独的用于相关程序的文件中，或在多个相互关联的文件中(例如，存储一个或多个模块的文件、子程序或部分代码)。计算机程序可用于在一台或多台计算机上被执行，这些计算机位于同一地点或分布在多个地点并由通信网络连接。

[0083] 图4概念性地示出了电子系统，使用该电子系统实现了本主题技术的一些实现。电子系统400可以是计算机、电话、个人数字助理或任何其他种类电子设备。这样的电子设备包括不同种类的计算机可读介质和用于各种不同种类的计算机可读介质的接口。电子系统400包括总线408，处理单元412，系统存储器404，只读存储器(ROM)410，永久存储设备402，输入设备接口414，输出设备接口406和网络接口416。

[0084] 总线408集体代表整个系统、外围，且芯片组以通信方式连接电子系统400的多个内部设备。例如，总线408以通信方式连接处理单元412与ROM 410、系统存储器404和永久性存储设备402。

[0085] 从这些不同的存储器单元，处理单元412取回执行指令和处理数据，以执行本主题公开的过程。在不同实现中，处理单元可以是单独的处理器或多核处理器。

[0086] ROM 410存储静态数据和指令，其被处理单元412和电子系统的其它模块所需要。另一方面，永久性存储设备402是读写存储器设备。该设备是非易失性存储器单元，其存储指令和数据，甚至当电子系统关闭时。本主题公开的一些实现使用大容量存储器(例如，磁盘或光盘及其对应的盘驱动)作为永久存储设备402。

[0087] 其它实现使用可删除存储介质(如，软盘、闪存及其对应的盘驱动)作为永久存储设备402。像永久性存储设备402一样，系统存储器404是读写存储器设备。但不像永久性存储设备402，系统存储器404是易失性读写存储器，类似于随机存储器。系统存储器404存储一些处理器在运行时所需的指令和数据。在一些实现中，本主题公开的过程被存储在系统存储器404、永久性存储设备402，和/或ROM 410中。例如，根据一些实现，不同存储器单元包括用于处理程序代码的指令。对这些不同的存储器单元，处理单元412取回执行指令和处理数据以执行一些实现的过程。

[0088] 总线408还连接到输入和输入设备接口414和406。输入设备接口414使得用户通信并选择电子系统命令。使用的具有输入设备接口414的输入设备包括，例如，字母键盘和指针式设备(也称作“光标控制设备”)。输出设备接口406可以例如显示由电子系统400生成的图像。使用的具有输出设备接口406的输出设备包括，例如，打印机和显示设备，如阴极射线管(CRT)或液晶显示器(LCD)。一些实现包括设备，如同时作为输入和输出设备的触摸屏。

[0089] 最后，如图4所示，总线408还通过网络接口416将电子系统400联接至网络。以这种方式，计算机可以是计算机网络的一部分，如局域网(“LAN”)，广域网(“WAN”)或内联网，或众多网络中的网络，如英特网。任意或所有电子系统400的元件可以用于与本主题公开结合。

[0090] 上述这些功能可在数字电子电路、计算机软件、固件或硬件中实现。使用一个或多个计算机程序产品可实现该技术。可编程处理器和计算机可被包括在移动设备中或作为移动设备被打包。该过程和逻辑流可由一个或多个可编程处理器或由一个或多个可编程逻辑电路实现。普通与特殊目的的计算设备和存储设备可通过通信网络互连。

[0091] 一些实现包括电子元件，例如微处理器、存储器和存储器，其将计算机程序指令存储在机器可读或计算机可读介质中(可选的，指计算机可读存储介质、计算机可读介质或机器可读存储介质)。这样的计算机可读介质的实例包括RAM、ROM、只读打包磁盘(CD-ROM)、可记录打包磁盘(CD-R)、可重写打包磁盘(CD-RW)、只读数字易失性磁盘(例如，DVD-ROM，双面DVD-ROM)、各种可记录/可重写DVDs(例如，DVD-RAM，DVD-RW，DVD+RW等)、闪存(如SD卡、小型SD卡、微型SD卡等)、磁性和/或固态硬盘、只读和可记录蓝光®盘、高密度光盘、任何其他光介质或磁介质和软盘。计算机可读介质可储存计算机程序，其可由至少一个处理单元执行，并包括用于实现各种操作的指令群。计算机程序或计算机代码的实例包括如由编译器产生的机器代码和包括由计算机执行的较高级代码的文件，电子元件或使用翻译器的微处理器。

[0092] 尽管上面讨论主要涉及微处理器或多核处理器，执行软件，一些由一个或多个集成电路执行的实现，如特定用途集成电路(ASIC)或现场可编程门阵列(FPGA)。在一些实现中，这样的集成电路执行存储在电路本身的指令。

[0093] 如在本申请说明书中和任意权利要求中使用的，术语“计算机”、“服务器”、“处理器”和“存储器”全部指电子或其他科技设备。这些术语不包括人或人群。为了本说明书的目的，术语“显示”意为显示在电子设备上。如在说明书和任意权利要求中所使用的，术语“计算机可读介质”完全局限于有形的、物理对象，其以计算机可读形式存储信息。这些术语不包括无线信号、有限下载信号和任意其他瞬时信号。

[0094] 为提供与用户的交互，本说明书中描述的主题的实现可在具有用来给用户显示信息的显示设备，和具有提供输入的键盘和指针设备的计算机上实现。显示设备如CRT(阴极射线管)或LCD(液晶显示)器，输入设备如鼠标和轨迹球。其他种类设备可被用于提供与用户的交互；例如，提供给用户的反馈可以是任意形式的传感反馈，例如，视觉反馈、听觉反馈或触觉反馈；且来自用户的输入可以任意形式接收，包括声学、语言或触觉输入。另外，计算机可与用户通过发文件给用户和从用户使用的设备处接收文件的方式进行交互；例如，通过发送网页至用户客户端设备上的网页浏览器以响应从网页浏览器接收到的请求。

[0095] 发明书中描述的主题的实施例可在计算机系统中被实现，该计算机系统包括后端元件，例如，数据服务器，或其包括中间设备元件，例如，应用服务器，或其包括前端元件，例如，具有图形用户接口或网络浏览器的客户端计算机，通过该图形用户接口或网络浏览器用户可与说明书中描述的主题或与任意一个或多个这样的后端、中端或前端元件实现交互。系统的元件可以通过例如通信网络的数字数据通信的任意形式或介质互连。通信网络的实例包括局域网(“LAN”)和广域网(“WAN”)，跨网络(例如，英特网)和对等的网络(例如，ad hoc对等的网络)。

[0096] 计算系统可包括客户端和服务器。客户端和服务器通常彼此相聚较远，且通常通过通信网络互动。客户端与服务器间的关系借助在分别的计算机上运行的并彼此具有客户端-服务器关系的计算机程序而发生。在一些实施例中，服务器传输数据(例如，HTML页)至

客户端设备(例如,用于显示数据至给用户,和从与客户端设备交互的用户处接收用户输入)。产生在客户端设备的数据(例如,用户交互的结果)可从位于服务器的客户端设备处被接收。

[0097] 应该理解,公开的任何特定的过程中的步骤的顺序或层次结构是实例性方法的表示。基于设计优选,应该将特定过程中的步骤的顺序或层次结构理解为是可以重新排列的,或所有所述步骤是可以执行的。一些步骤可以同时执行。例如,在特定环境中,多任务和并行处理可能是有利的。此外,上述实施例中的不同系统元件的分开不应理解为在所有实施例中都要求这样的分开,且应该理解描述的程序元素和系统可通常集成在单独的软件产品或打包至多个软件产品。

[0098] 提供前面的描述使得本领域技术人员可以实践本文的各个方面。这些方面不同的变体对本领域技术人员是非常显而易见的,且本文中定义的总原则可适用于其他方面。因此,权利要求不旨在限制本文中的方面,而应被赋予与语言权利要求一致的整个范围,其中,除非特别说明,涉及到单数的元素并不意为“一个且仅有一个”,而是“一个或多个”。除非特别说明,术语“一些”是指一个或多个。阳性代词(例如,他的)包括女性和中性(例如,她的和它的)且反之亦然。标题和副标题只为方便起见而使用,并不限制主题的公开。

[0099] 短语如“方面”并不指该方面对主题技术很关键,或该方面适用于所有形式,或一个或多个形式。如“方面”这样的术语可指一个或多个方面,反之亦然。术语如“形式”并不指该形式对主题技术很关键,或该形式适用于所有主题技术的形式。涉及到形式的公开可适用于所有形式,或一个或多个形式。短语,如“形式”可指一个或多个形式,反之亦然。

[0100] 词语“实例性的”在本文中意为“作为说明实例”。任何本文中描述为“实例性的”方面或设计不必理解为优选的或比其他方面或设计有利的。

[0101] 当前或今后对本领域的技术人员是已知的,本公开中通篇描述的所有结构性和功能性的元素的不同方面的等价体,在本文中作为参考被明确地并入,并旨在被包含于权利要求中。此外,无论是否清晰地在权利要求中详述本公开,本文中公开的内容并不针对公众。

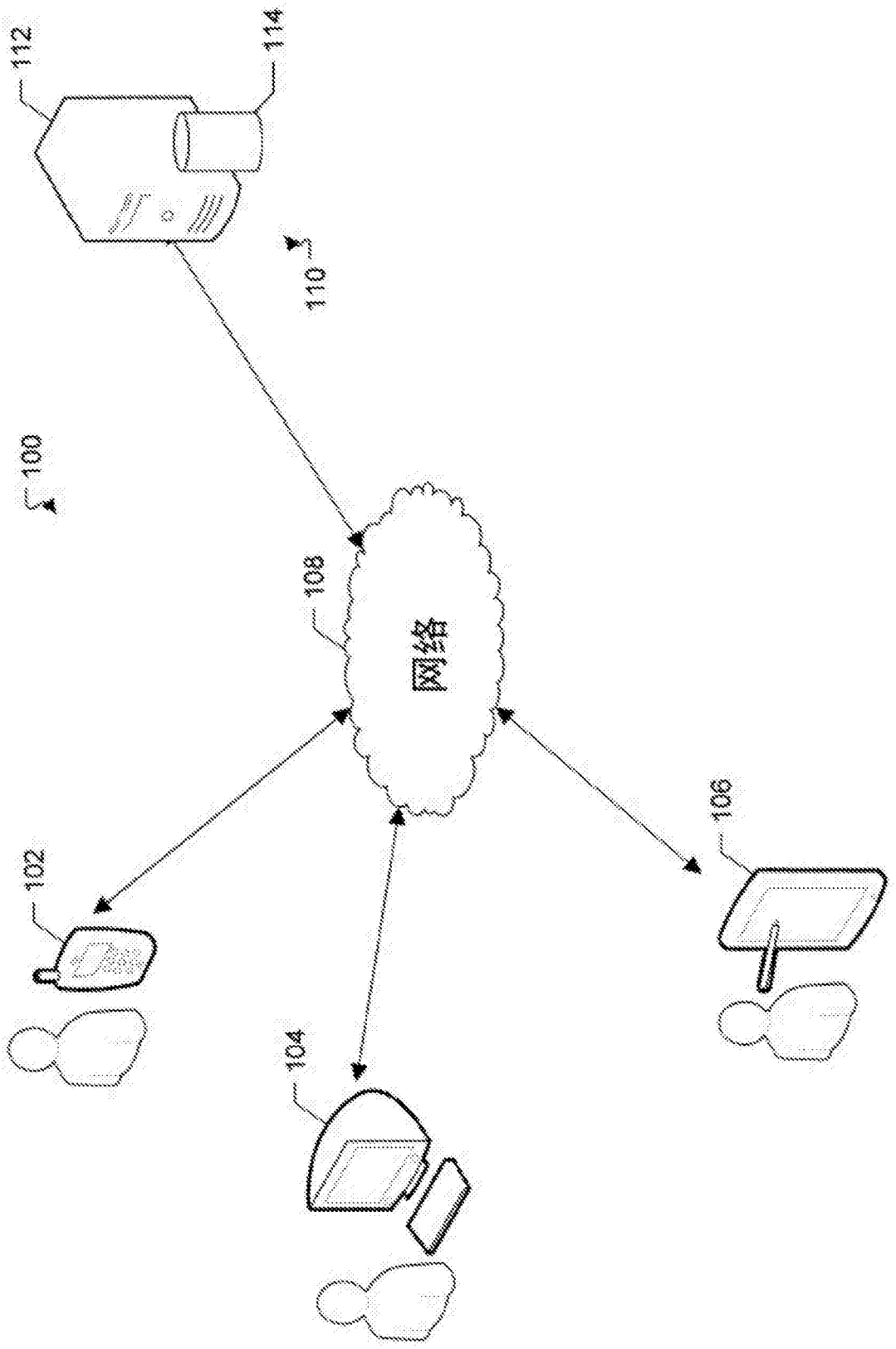


图1

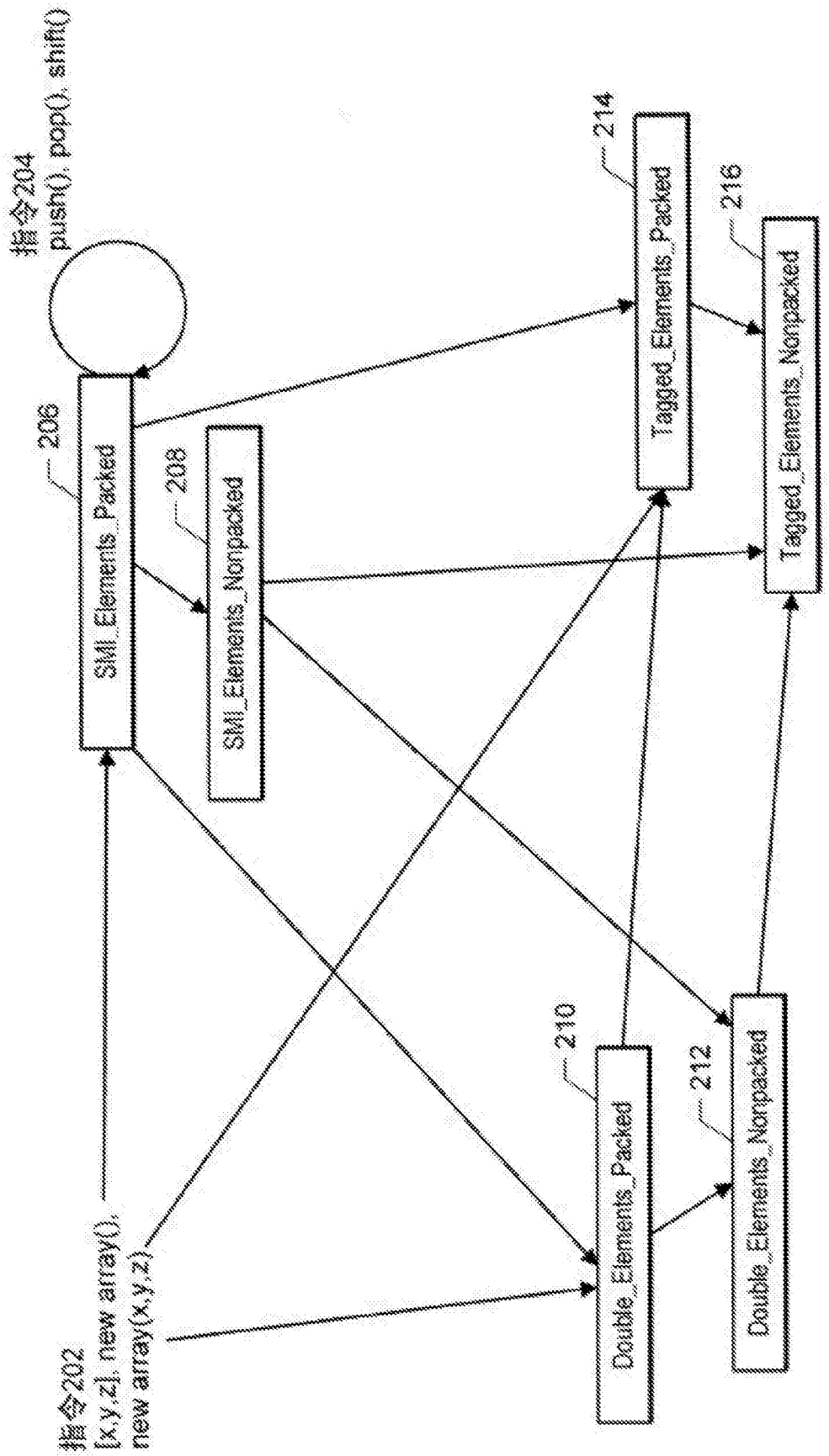

指令204
push(), pop(), shift()
206
SMI_Elements_Packed
208
SMI_Elements_Nonpacked
214
Tagged_Elements_Packed
216
Tagged_Elements_Nonpacked
210
Double_Elements_Packed
212
Double_Elements_Nonpacked
指令202
[x,y,z], new array(),
new array(x,y,z)

图2

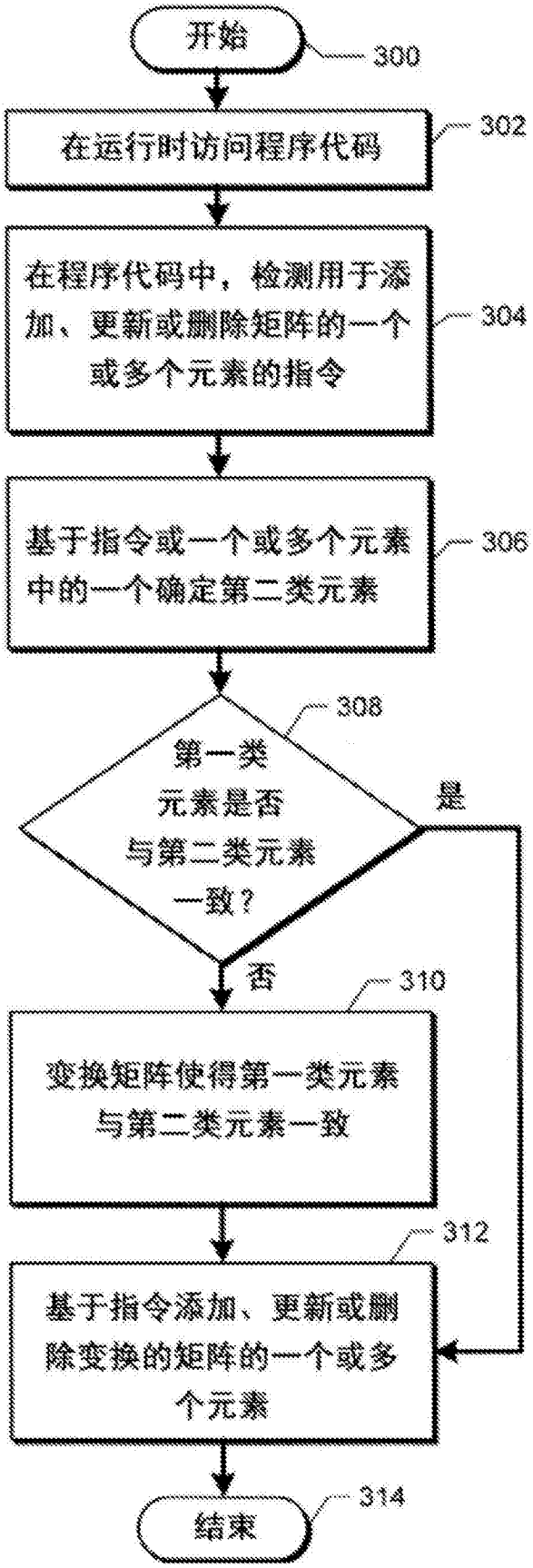

开始
300
在运行时访问程序代码
302
在程序代码中，检测用于添加、更新或删除矩阵的一个或多个元素的指令
304
基于指令或一个或多个元素中的一个确定第二类元素
306
308
第一类元素是否与第二类元素一致？
是
否
310
变换矩阵使得第一类元素与第二类元素一致
312
基于指令添加、更新或删除变换的矩阵的一个或多个元素
结束
314

图3

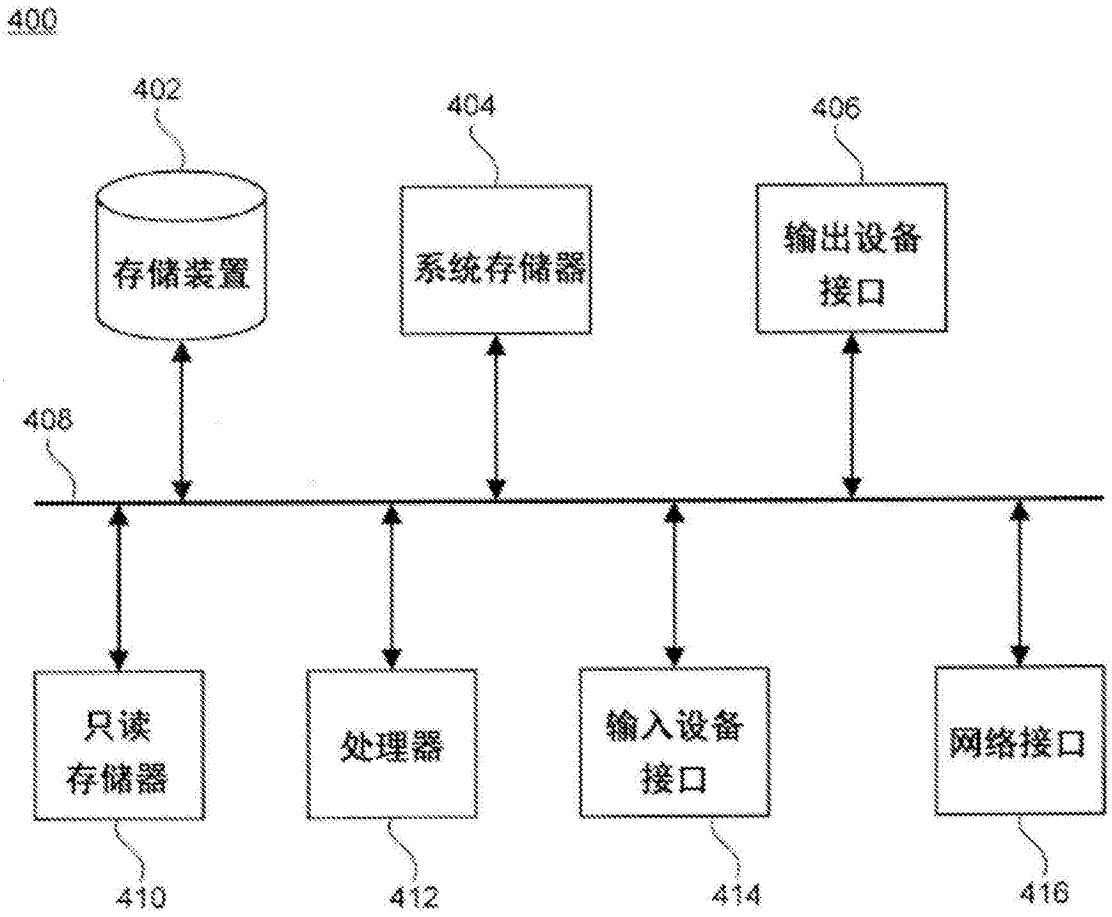

400
402
存储装置
404
系统存储器
406
输出设备
接口
408
只读
存储器
处理器
输入设备
接口
网络接口
410
412
414
416

图4