(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 104243743 B
(45)授权公告日 2017.12.05

(21)申请号 201410251275.2

(22)申请日 2014.06.06

(65)同一申请的已公布的文献号
申请公布号 CN 104243743 A

(43)申请公布日 2014.12.24

(30)优先权数据
2013-120103 2013.06.06 JP

(73)专利权人 佳能株式会社
地址 日本东京都大田区下丸子3-30-2

(72)发明人 藤泽邦匡 长田守

(74)专利代理机构 北京怡丰知识产权代理有限公司 11293
代理人 迟军

(51)Int.Cl.
H04N 1/00(2006.01)
G06F 11/34(2006.01)

(56)对比文件
CN 102693188 A,2012.09.26,
CN 102693188 A,2012.09.26,
CN 1534467 A,2004.10.06,
US 2001001328 A1,2001.05.17,

审查员 奚惠宁

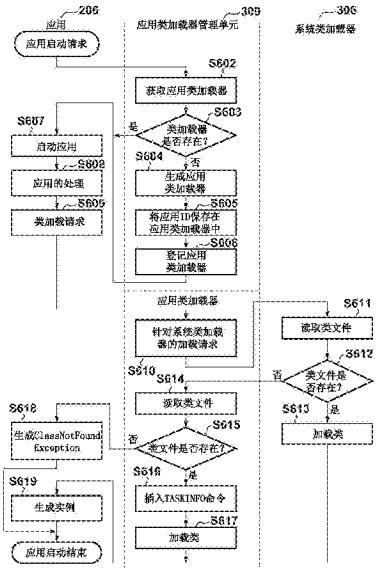
权利要求书2页 说明书15页 附图24页

(54)发明名称

图像处理装置及其控制方法

(57)摘要

本发明提供一种图像处理装置及其控制方法。该图像处理装置响应于应用的启动请求，读取所述应用的类的类文件，在所读取的类文件中包括的方法的开头将用于记录表示所述应用的应用信息的代码添加到线程，并加载所述类。此外，所述图像处理装置在执行所读取的类文件中包括的所述方法期间，分配要用于待生成的对象的内存，并将所述线程记录中的应用信息记录到所分配的内存或文件大小，同时生成所述对象并与内存大小或盘使用量相关联地管理所生成的对象的应用信息。



CN 104243743 B

1.一种至少包括应用控制单元和系统控制单元的图像处理装置，

所述应用控制单元包括：

控制单元，其被配置为响应于应用的启动请求，读取所述应用的类的类文件，并加载所述类，

所述系统控制单元包括：

对象生成单元，其被配置为加载所述类，分配要用于待生成的对象的内存，并将所述对象生成到所分配的内存，

所述应用控制单元还包括：

内存管理单元，其被配置为将启动请求被接受的所述应用与用于启动所述应用的内存相关联，并管理所关联的应用和内存，

其中，所述系统控制单元将多个应用作为一个应用来处理，并在存在由所述对象生成单元生成的多个对象的情况下，将各对象与由所述内存管理单元使用的相应应用相关联。

2.根据权利要求1所述的图像处理装置，其中，所述控制单元还在所读取的类文件中包括的方法中添加用于将表示所述应用的应用信息记录到线程的代码，并在所读取的类文件中包括的所述方法的末尾，添加用于删除所述线程中记录的应用信息的代码。

3.根据权利要求2所述的图像处理装置，其中，在多个应用协同操作的情况下，所述线程中记录的所述应用信息具有栈结构。

4.根据权利要求2所述的图像处理装置，所述图像处理装置还包括：

表，其中所述应用信息与所述应用可用的内存大小被有联系地定义；

确定单元，其被配置为在所述对象生成单元分配了用于生成对象的内存的情况下，确定与由所述内存管理单元管理的所述应用信息相对应的应用正在使用的内存大小与所分配的内存大小的总和是否超过所述相应的可用内存大小；以及

限制单元，其被配置为在所述确定单元确定所述总和超过所述可用内存大小的情况下，生成内存错误。

5.一种至少包括应用控制单元和系统控制单元的图像处理装置，

所述应用控制单元包括：

控制单元，其被配置为响应于应用的启动请求，读取所述应用的类的类文件，并加载所述类，

所述系统控制单元包括：

对象生成单元，其被配置为加载所述类，分配要用于待生成的对象的文件大小，并且使用所分配的文件大小作为盘使用量来生成所述对象，

所述应用控制单元还包括：

盘管理单元，其被配置为将启动请求被接受的所述应用与用于启动所述应用的盘相关联，并管理所关联的应用和盘，

其中，所述系统控制单元将多个应用作为一个应用来处理，并在存在由所述对象生成单元生成的多个对象的情况下，将各对象与由所述盘管理单元使用的相应应用相关联。

6.根据权利要求5所述的图像处理装置，其中，所述控制单元还在所读取的类文件中包括的方法中添加用于将表示所述应用的应用信息记录到线程的代码，并在所读取的类文件中包括的所述方法的末尾，添加用于删除所述线程中记录的应用信息的代码。

7.根据权利要求6所述的图像处理装置，其中，在多个应用协同操作的情况下，所述线程中记录的所述应用信息具有栈结构。

8.根据权利要求6所述的图像处理装置，所述图像处理装置还包括：

表，其中所述应用信息与所述应用可用的盘使用量被有联系地定义；

确定单元，其被配置为在所述对象生成单元分配了用于生成对象的文件大小的情况下，确定与由所述盘管理单元管理的所述应用信息相对应的应用正在使用的盘使用量与所分配的文件大小的总和是否超过所述相应的可用盘使用量；以及

限制单元，其被配置为在所述确定单元确定所述总和超过所述可用盘使用量的情况下，生成盘错误。

9.一种至少包括应用控制单元和系统控制单元的图像处理装置的控制方法，所述控制方法包括如下步骤：

在所述应用控制单元中，响应于应用的启动请求，读取所述应用的类的类文件，并加载所述类；

在所述系统控制单元中，加载所述类，分配要用于待生成的对象的内存，并将所述对象生成到所分配的内存；以及

在所述应用控制单元中，将启动请求被接受的所述应用与用于启动所述应用的内存相关联，并管理所关联的应用和内存，

其中，所述系统控制单元将多个应用作为一个应用来处理，并在存在由所述系统控制单元生成的多个对象的情况下，将各对象与在所述应用控制单元中使用的相应应用相关联。

10.一种至少包括应用控制单元和系统控制单元的图像处理装置的控制方法，所述控制方法包括如下步骤：

在所述应用控制单元中，响应于应用的启动请求，读取所述应用的类的类文件，并加载所述类；

在所述系统控制单元中，加载所述类，分配要用于待生成的对象的文件大小，并且使用所分配的文件大小作为盘使用量来生成所述对象；以及

在所述应用控制单元中，将启动请求被接受的所述应用与用于启动所述应用的盘相关联，并管理所关联的应用和盘，

其中，所述系统控制单元将多个应用作为一个应用来处理，并在存在由所述系统控制单元生成的多个对象的情况下，将各对象与在所述应用控制单元中使用的相应应用相关联。

图像处理装置及其控制方法

技术领域

[0001] 本说明书涉及一种操作多个应用的图像处理装置及其控制方法。

背景技术

[0002] 当前越来越多的多功能外围设备(MFP)具有执行除了内置在MFP中功能(例如文档的复印、扫描和打印)以外的应用的功能。许多MFP具有Java执行环境作为应用执行环境,并且能够执行以Java(注册商标)记述的应用。示例性应用执行环境包括佳能(注册商标)开发的MEAP(注册商标)。

[0003] 虽然在PC上的Java的情况下,按照每个应用一个处理的方式执行应用,但是许多MFP由于CPU或者内存限制,通过使用OSGi框架等,经由一个Java处理执行多个应用。因此,当MFP上正在执行的应用中的一个应用的错误(bug)导致内存溢出时,可能发生OutOfMemoryError(内存溢出错误),从而造成所有应用停止。此外,在当应用请求内存时没有要分配的内存的情况下,发生OutOfMemoryError,因此在正常运行的应用的执行期间,也可能发生OutOfMemoryError。因此,难以指定造成内存溢出的应用。

[0004] 日本特开第2005-269439号公报提出用于逐线程测量内存的技术。然而,在例如如稍后讨论的、如图13所示的一个线程执行多个应用的代码的情况下,无法测量各应用使用的内存。

[0005] 目前,为了发现内存溢出,可想到以下两种方法。一种方法涉及使用称为分析器(profiler)的工具来监视由应用生成的对象的状态。另一方法涉及将由Java VM使用的堆内存的内容转存(dumping)(下文中,“堆转存(heap dumping)”)并分析由应用生成的对象。

[0006] 在这些方法中,由于使用分析器监视对象的状态极大地降低了应用的执行速度,因此在CPU或内存非常有限的MFP上的应用是成问题的。因此,使用了执行堆转存并分析由应用生成的对象的技术。同样,在一些情况下,预先确定用作应用执行环境的可用盘容量。在盘满(disk full)的情况下,与内存溢出不同,即使当重启MFP时,状况也不会被自动恢复。因此,在“iR-ADV手册”(佳能,“安装应用”页,“使用应用”页(2013年5月17日检索),互联网<URL:http://cweb.canon.jp/manual/ir-adv/>)中公开了如下技术:应用预先声明使用量并且实施安装限制以不超过该使用量。

[0007] 然而,在上述传统技术中存在以下问题。使用传统堆转存,从设备中提取所获取的堆转存信息并对其进行分析以定位造成内存溢出的应用,并且对由应用使用的内存量进行累计。因此,无法实时获知各应用的内存使用量。另一方面,由于应用的执行速度极大地降低,因此实际上,使用分析器难以实时测量各应用使用的内存量。

[0008] 即使在应用预先声明盘使用量并且实施安装限制以不超过该使用量的情况下,MFP上正在执行的应用中的一个也可能具有错误并且整个应用执行环境可能处于盘满状态。在该情况下,即使在无错误的应用中也发生写入错误,并且不再能够进行正常操作。

发明内容

[0009] 本发明使得能够实现在维持性能的同时实时测量各应用使用的内存或者盘的使用量的机制。

[0010] 本发明的一个方面提供一种用于执行多个应用的图像处理装置，该图像处理装置包括：控制单元，其被配置为响应于应用的启动请求，读取所述应用的类的类文件，在所读取的类文件中包括的方法中添加用于将表示所述应用的应用信息记录到线程的代码，并加载所述类；对象生成单元，其被配置为在执行所读取的类文件中包括的所述方法期间，分配要用于待生成的对象的内存，并将所述线程记录中的应用信息记录到所分配的内存，并生成所述对象；以及内存管理单元，其被配置为与内存大小相关联地管理由所述对象生成单元生成的所述对象的应用信息。

[0011] 本发明的另一方面提供一种用于执行多个应用的图像处理装置，该图像处理装置包括：控制单元，其被配置为响应于应用的启动请求，读取所述应用的类的类文件，在所读取的类文件中包括的方法中添加用于将表示所述应用的应用信息记录到线程的代码，并加载所述类；对象生成单元，其被配置为在执行所读取的类文件中包括的方法期间，分配要用于待生成的对象的文件大小，并且使用所分配的文件大小作为盘使用量来记录所述线程中记录的应用信息，并生成所述对象；以及盘管理单元，其被配置为将由所述对象生成单元生成的所述对象的应用信息与盘使用量被有联系地管理。

[0012] 本发明的又一方面提供一种图像处理装置的控制方法，该图像处理装置执行多个应用，所述控制方法包括如下步骤：响应于应用的启动请求，通过控制单元读取所述应用的类的类文件，在所读取的类文件中包括的方法中添加用于将表示所述应用的应用信息添加到线程的代码，并加载所述类；在执行所读取的类文件中包括的所述方法期间，通过对象生成单元，分配要用于待生成的对象的内存，将所述线程中记录的应用信息记录到所分配的内存，并生成所述对象；以及通过内存管理单元，与内存大小相关联地管理由所述对象生成单元生成的所述对象的应用信息。

[0013] 本发明的再一方面提供一种图像处理装置的控制方法，该图像处理装置执行多个应用，所述控制方法包括如下步骤：响应于应用的启动请求，通过控制单元读取所述应用的类的类文件，在所读取的类文件中包括的方法中添加用于将表示所述应用的应用信息添加到线程的代码，并加载所述类；在执行所读取的类文件中包括的所述方法期间，通过对象生成单元，分配要用于待生成的对象的文件大小，并且使用所分配的文件大小作为盘使用量来记录所述线程中记录的应用信息，并生成所述对象；以及通过盘管理单元，与盘使用量相关联地管理由所述对象生成单元生成的所述对象的应用信息。

[0014] 通过以下参照附图对示例性实施例的描述，本发明的其他特征将变得清楚。

附图说明

[0015] 图1是示出应用管理装置的配置的图。

[0016] 图2是示出应用管理装置的软件配置的图。

[0017] 图3是示出应用管理装置的模块配置的图。

[0018] 图4是清单文件。

[0019] 图5是app(应用)类加载器表以及使用内存表。

[0020] 图6是类加载处理的流程图。

[0021] 图7是示出添加了TASKINFO类方法的示例性代码的图。

[0022] 图8是根据第一实施例的线程结构的图。

[0023] 图9是根据第一实施例的执行插入了TASKINFO命令的方法时的流程图。

[0024] 图10是示出对象结构以及堆内存的图。

[0025] 图11是示出根据第一实施例的对象生成处理的流程图。

[0026] 图12是GC处理的流程图。

[0027] 图13是示出多个应用之间的示例性调用处理的图。

[0028] 图14是根据第二实施例的线程结构的图。

[0029] 图15是根据第二实施例的执行插入了TASKINFO命令的方法时的流程图。

[0030] 图16是示出根据第二实施例的对象生成处理的流程图。

[0031] 图17是示出根据第三实施例的应用管理装置的配置的图。

[0032] 图18是使用内存量声明表。

[0033] 图19是示出根据第三实施例的对象生成处理的流程图。

[0034] 图20是示出根据第四实施例的应用管理装置的配置的图。

[0035] 图21是示出使用盘表2002以及使用盘量声明表2003的配置的图。

[0036] 图22是根据第四实施例的清单文件的图。

[0037] 图23是根据第四实施例的启动时的流程图。

[0038] 图24是FileOutputStream对象生成时的流程图。

[0039] 图25是执行FileOutputStream对象write方法时的流程图。

[0040] 图26是执行RandomAccessFile对象setLength方法时的流程图。

[0041] 图27是执行RandomAccessFile对象write方法时的流程图。

[0042] 图28是执行File对象delete方法时的流程图。

具体实施方式

[0043] 现在，参照附图对本发明的实施例进行详细描述。应当指出，除非另外具体说明，这些实施例中描述的部件、数字表示以及数值的相对布置不限制本发明的范围。

[0044] 第一实施例

[0045] 应用管理装置的配置

[0046] 下文中，参照图1至图12描述本发明的第一实施例。首先，参照图1描述应用管理装置100的硬件配置。应用管理装置100是图像处理装置的示例。应用管理装置100配设有控制器120、扫描器111、打印机112以及操作面板113，并且还配设有可拆卸的IC读卡器116。控制器120配设有CPU101、ROM102、RAM103、外部存储设备104、USBH I/F控制器105、扫描器I/F控制器106、打印机I/F控制器107、NVRAM108、面板控制器109以及网络I/F控制器110。

[0047] CPU101执行应用管理装置100的软件程序并进行装置的整体控制。ROM102为只读存储器并且存储装置的引导程序、固定参数等。RAM103是随机存取存储器，并且在CPU101控制装置时用于临时数据存储等。外部存储装置104用于各种数据的存储，例如所安装的应用、应用数据以及打印数据的存储。

[0048] USBH I/F控制器105用于控制USB主机接口，并且控制与各种USB设备的通信。扫描器I/F控制器106是控制扫描器111的装置。打印机I/F控制器107是控制打印机112的装置。

NVRAM108是非易失性存储器，并且其中保存应用管理装置100的各种设定值。

[0049] 面板控制器109用于控制操作面板113、显示各种信息以及接收来自用户的指令的输入。网络I/F控制器110控制网络115的数据发送和接收。CPU101、ROM102、RAM103、外部存储设备104、USBH I/F控制器105、扫描器I/F控制器106、打印机I/F控制器107、NVRAM108、面板控制器109以及网络I/F控制器110与总线114连接。此外，总线114是经由其发送和接收来自CPU101的控制信号以及不同装置之间的数据信号的系统总线。IC读卡器116是用于进行验证的USB设备。

[0050] 软件配置

[0051] 接下来，参照图2描述应用管理装置100的软件200的示例。硬件201执行应用管理装置的软件。OS202执行处理的管理、内存管理以及输入输出管理。本地应用203是实现设备的基本功能(例如复印等)的程序。

[0052] 应用管理装置100的软件200由Java VM204和应用平台205构成。Java VM204是执行Java程序的虚拟机。应用平台205是管理应用的生命周期(即至少一个或者多个应用程序在单个Java VM上的启动、停止、安装以及卸载)的程序。app A206和app B207是在应用平台205上操作的应用程序。Java VM204将应用平台205、app A206和app B207作为单个Java应用208来处理。

[0053] 模块配置

[0054] 接下来，参照图3描述构成本发明的应用管理装置100的软件200的模块的示例。Java VM204配设有字节代码执行单元301、内存管理单元302、堆内存305以及系统类加载器306。

[0055] 字节代码执行单元301解释和执行作为Java程序代码的字节代码。堆内存305是由Java VM204管理的内存区域，并且保持通过Java程序的执行而生成的Java对象。内存管理单元302管理应用使用的内存。内存管理单元302由对象生成单元303和GC执行单元304构成。对象生成单元303根据由字节代码执行单元301执行的程序代码的指令来生成Java对象。GC执行单元304执行用于从保存在堆内存305中的Java对象中删除不再使用的Java对象的垃圾收集。系统类加载器306加载由JavaVM204管理的类。通常，类加载器根据需要(即，当执行处理时)首先加载Java类。

[0056] 应用平台205由应用管理单元308、应用内存管理单元307以及app类加载器管理单元309构成。应用内存管理单元307具有用于管理各app使用的内存的使用内存表313，并针对各应用管理要使用的内存。应用管理单元308管理应用的生命周期，例如应用的安装、启动、停止以及卸载。app类加载器管理单元309具有app类加载表310，并且针对各应用生成和管理app类加载器。app类加载器从保存在外部存储设备104中的应用的程序文件312中加载类。

[0057] 清单文件

[0058] 接下来，将参照图4描述应用的清单文件的内容。清单文件是应用的Jar文件中包括的文件。在清单文件中记载了如下信息。

[0059] Bundle-Name401是应用的名称。Application-Id402是作为用于识别应用的唯一标识符的应用ID。MaximumMemoryUsage403是应用可用的最大内存使用量。这些设定项目由应用平台205规定。请注意，上述设定项目和设定值是示例并不旨在不限制本发明，并且除

了上述项目和设定值以外，在清单文件中还可以记载各种设定项目和设定值。

[0060] 表

[0061] 接下来，将参照图5描述由应用平台205管理的app类加载器表310和使用内存表313。app类加载器表310是将作为应用的标识符的应用ID501和应用的app类加载器502有联系(tie together)地管理的表。使用内存表313是将作为应用的标识符的应用ID503和应用当前正在使用的内存量504有联系地管理的表。

[0062] 加载处理

[0063] 接下来，将参照图6描述根据本实施例的应用平台205加载应用的类时的处理过程。通过CPU101将存储在ROM102或者外部存储设备104中的控制程序读出到RAM103并执行该控制程序来实现下文中描述的处理。

[0064] 当执行应用的启动请求时，CPU101向app类加载器管理单元309发出app类加载器请求。在S602中，app类加载器管理单元309开始获取app类加载器。首先，在S603中，app类加载器管理单元309确定在类加载器表310中是否存在与针对其进行了类加载器请求的应用的应用ID相对应的类加载器。如果存在相应的类加载器，则处理进入S607，而如果不存在相应的类加载器，则处理进入S604。

[0065] 在S604中，app类加载器管理单元309生成app类加载器。然后，在S605中，app类加载器管理单元309针对在S604生成的app类加载器保存应用ID。此外，在S606中，app类加载器管理单元309将在S604中生成的app类加载器以及针对其进行了类加载器请求的应用的应用ID登记在app类加载器表310中。之后，处理进入S607，并返回到应用。

[0066] 如果在S603中确定存在相应的类加载器或者在S606的处理之后，CPU101在S607中使用从app类加载器管理单元309获取的app类加载器开始应用206的启动。然后，在S608中，启动的应用206开始生成应用对象。此外，为了加载应用的类，应用206在S609中向从app类加载器管理单元309获取的app类加载器发出针对应用类的类加载请求。

[0067] 在S610中，应用类加载器在从应用206接收到app类的加载请求时，首先请求Java VM中内置的系统类加载器306加载类。在S611中，系统类加载器306在接收到类加载请求时，读取所请求的类的类文件的字节代码。然后，在S612中，系统类加载器306确定类文件的读取是否完成，并且如果完成，则进入S613。另一方面，如果类文件的读取尚未完成，则由于正在加载的应用的固有的类，处理返回到app类加载器并进入S614。在S613中，系统类加载器306基于在S611中读取的字节代码来加载类，进入S619，并向应用206返回类对象。

[0068] 另一方面，在S614中，app类加载器从app的Jar文件312读取所请求的类的字节代码。然后，在S615中，app类加载器确定是否类文件存在并且读取成功，如果成功则进入S616，如果不成功则进入S618。

[0069] 在S616中，app类加载器向读取的字节代码中插入TASKINFO命令，并在S617中，基于插入了TASKINFO命令的字节代码来加载类。稍后将使用图7详细讨论TASKINFO命令。之后，处理进入S619，并且类对象被返回到应用206。

[0070] 另一方面，如果在S615中确定类文件不存在，则应用206在S618中生成ClassNotFoundException，并结束应用的启动。在S619中，应用206使用在S617中加载的类，来生成应用对象的实例，并结束app的启动。

[0071] TASKINFO命令

[0072] 接下来，将参照图7描述根据本发明的在S616中在字节代码中插入的TASKINFO命令。附图标记701表示插入TASKINFO命令前的代码。附图标记702表示插入TASKINFO命令后的代码。

[0073] 如附图标记703所示，在方法的开头插入“TASKINFO.set(应用ID);”。这里，给出“11-1111-1111”作为示例性应用ID。在S605中保存在类加载器中的值被转换为字符串并被插入作为应用ID。此外，如附图标记704所示，在方法的末尾，插入“TASKINFO.remove();”。稍后详细讨论TASKINFO.set()方法和TASKINFO.remove()方法。

[0074] 线程结构

[0075] 接下来，将参照图8描述根据本发明的第一实施例的线程的结构。线程的结构801具有用于存储应用ID的字段802。TASKINFO.set()方法和TASKINFO.remove()方法使用字段802。具体地说，TASKINFO.set()方法设定应用ID，TASKINFO.remove()方法去除应用ID。

[0076] 方法的处理

[0077] 接下来，将参照图9描述根据本发明在字节代码执行单元301中执行插入了应用的程序中包括的TASKINFO命令的方法时的处理过程。通过CPU101将存储在ROM102或者外部存储设备104中的控制程序读出到RAM103并执行该控制程序来实现下文中描述的处理。

[0078] 在开始执行应用的方法时，应用206调用位于方法的头(head)的“TASKINFO.set(应用ID)”。此时，应用ID被添加作为参数。之后，处理转变到TASKINFO类的set方法。

[0079] 在S903中，TASKINFO类的set方法首先获取正在执行处理的当前线程。然后，set方法将作为set方法的参数传送的应用ID的值保存到在S904中获取的线程结构的应用ID字段802中，并将处理返回到应用206。

[0080] 在S905中，应用206执行app的方法中的处理，在S906中，调用位于方法的末尾的“TASKINFO.remove();”。之后，处理转变到TASKINFO类的remove方法。

[0081] 在S907中，TASKINFO类的remove方法首先获取正在执行处理的当前线程。在S908中，remove方法将在S907中获取的线程的线程结构的字段802中设定的应用ID的值去除，并将处理返回到应用。应用结束方法的处理。由此在方法的执行期间记录与线程相对应的应用信息。使用该信息，根据本实施例的图像处理装置能够实时查明内存使用量。

[0082] 对象结构以及堆内存

[0083] 接下来，将参照图10描述本发明的对象的结构以及堆内存。附图标记1001表示用作比较示例的Java VM的堆内存。根据本实施例的对象的结构1002与堆内存1001的不同之处在于：除了对象固有的信息以外，还配设用于保存应用ID(应用信息)的字段1003。因此，根据本实施例的Java VM204的堆内存305如附图标记1004所示。例如，如图10所示，堆内存305分配对象A至E的结构的区域。分配的区域还分别配设有用于保存结构中包括的应用ID的区域。因此，能够容易地确认与任意对象相关的应用。在图10中，未使用的内存区域被表示为空余内存。

[0084] 对象生成处理

[0085] 接下来，参照图11描述根据第一实施例的对象生成处理的处理过程。在上述图9的S905的处理期间执行对象生成处理。通过CPU101将存储在ROM102或者外部存储设备104中的控制程序读出到RAM103并执行该控制程序来实现下文中描述的处理。

[0086] 在字节代码执行单元301正在执行进行对象生成的应用的代码时，开始对象生成

处理，并且Java VM204的对象生成单元303被调用。在S1102中，对象生成单元303首先获取当前线程。然后，在S1103中，对象生成单元303读取在S1102中获取的线程的结构的应用ID字段802，并获取应用ID。

[0087] 在S1104中，对象生成单元303获取用于对象生成的内存，并生成对象。此外，在S1105中，对象生成单元303将在S1103中获取的应用ID记录在对象的结构的应用ID字段1003中。在S1106中，为了记录内存增加量，对象生成单元303使用在S1105中记录的应用ID以及在S1104中生成的对象的大小作为参数，来调用应用内存管理单元307。

[0088] 在S1107中，应用内存管理单元307使用在S1106中被指定为参数的应用ID，搜索使用内存表313的应用ID字段503。这里，如果存在应用ID，则处理进入S1109，而如果不存在应用ID，则进入S1108。

[0089] 在S1108中，应用管理单元308生成使用内存大小为0的新记录，将所生成的记录登记在使用内存表313中，然后进入S1109。在S1109中，应用管理单元308更新与使用内存表313中的应用ID相对应的使用内存的值504，并将处理返回到对象生成单元303。在S1110中，对象生成单元303将所生成的对象返回到应用。应用206接收所生成的对象并结束对象生成处理。以这种方式，根据本实施例的应用管理装置在生成对象时，通过应用内存管理单元307来管理相应的应用ID和分配的内存大小。

[0090] GC处理

[0091] 接下来，将参照图12描述根据本发明的第一实施例的GC处理的处理过程。当对象生成没有足够的空余内存时，Java VM204通过释放不再需要的对象来进行用于增加空余内存的处理。该处理被称为垃圾收集(GC)。通过CPU101将存储在ROM102或者外部存储设备104中的控制程序读出到RAM103并执行该控制程序来实现下文中描述的处理。

[0092] 当对象生成单元303的对象生成没有足够空余内存时，GC执行单元304开始GC处理。在S1202中，GC执行单元304在堆内存305中搜索作为GC目标的、任何人未在参照的任意对象。在S1203中，GC执行单元304确定是否找到目标对象。这里，如果存在目标对象，则处理进入S1204，而如果不存在目标对象，则处理结束。

[0093] 在S1204中，GC执行单元304从堆内存305中获取目标对象的内容。然后，GC执行单元304获取在S1204中获取的对象的应用ID字段的值。此外，在S1206，为了记录内存减少量，GC执行单元304使用在S1205中记录的应用ID以及在S1204中获取的对象的大小作为参数，来调用应用内存管理单元307。

[0094] 在S1207中，应用内存管理单元307更新与在S1206中被指定为参数的应用ID相对应的使用内存表313中的使用内存的值504，并将处理返回到GC执行单元304。当处理从应用内存管理单元307返回时，GC执行单元304在S1208中释放对象，并进入S1202以找到作为GC目标的下一对象。

[0095] 如上所述，根据本实施例的图像处理装置响应于应用的启动请求，读取应用的类的类文件，在所读取的类文件中包括的方法的开头添加用于将表示应用的应用信息记录到线程的代码，并加载该类。此外，该图像处理装置在方法的执行期间，分配要用于待生成的对象的内存，将线程中记录的应用信息记录到分配的内存，同时生成对象并将生成的对象的应用信息与内存大小有联系地管理。因此，通过该图像处理装置，能够在维持性能的同时实时测量各应用的内存使用量或者盘使用量。

[0096] 第二实施例

[0097] 以下，参照图13至图16描述本发明的第二实施例。请注意，将省略与第一实施例类似的配置和控制的描述。与第一实施例类似地执行类加载处理。首先，将参照图13描述多个应用在应用平台205上协同操作的序列。

[0098] 用作本发明的图像处理装置的应用管理装置100以及由网络连接的设备操作浏览器1301。设备是PC、智能手机或者平板。HTTP服务器app1302是在应用平台205上操作的HTTP服务器程序，并且具有应用ID“44-4444-4444”。servlet app1303是与HTTP服务器app1302协同操作的servlet型应用，并且具有应用ID“11-1111-1111”。

[0099] 在S1304中，浏览器1301通过网络向HTTP服务器app1302发送HTTP请求。然后，在S1305中，HTTP服务器app1302在接收到HTTP请求时，生成用于进行HTTP请求响应处理的一个线程，并进行HTTP请求分析处理。

[0100] 如果HTTP请求针对servlet app1303，则HTTP服务器app1302在S1306中将信息附在HTTP请求上并调用servlet app1303。在S1307中，servlet app1303执行由servlet实施的固有处理，并且在S1308中，将servlet的处理结果返回到HTTP服务器app1302。

[0101] 在S1309中，HTTP服务器app1302基于接收到的servlet的处理结果来生成HTTP响应，并且在S1310中，将HTTP响应通过网络返回到浏览器。此时，通过一个线程处理S1305至S1309。

[0102] 在应用平台205上如图所示的多个应用协同操作是常见的。通过HTTP请求处理(S1305)、servlet处理(S1307)或者HTTP响应处理(S1309)或者上述全部处理来进行对象生成。因此，需要区分对象是由HTTP服务器app1302生成还是servlet app1303生成。

[0103] 线程结构

[0104] 接下来，将参照图14描述根据本发明的第二实施例的线程的结构。线程结构1401具有应用ID栈1402作为字段。当在图13的处理中正在执行作为servlet固有处理的S1307时，正在执行的应用的应用ID被堆栈。在该情况下，servlet app1303的应用ID1403和HTTP服务器app1302的应用ID1404从顶部依次堆栈在应用ID栈1402中。

[0105] 方法的处理

[0106] 接下来，将参照图15描述根据本发明的第二实施例的在字节代码执行单元301中执行插入了应用的程序中包括的TASKINFO命令的方法时的处理过程。通过CPU101将存储在ROM102或者外部存储设备104中的控制程序读出到RAM103并执行该控制程序来实现下文中描述的处理。

[0107] 当开始执行应用的方法时，应用206调用位于方法的头的“TASKINFO.set(应用ID);”703，处理进入TASKINFO类的set方法。在S903中，TASKINFO类的set方法首先获取正在执行处理的当前线程。然后，在S1501中，set方法将作为set方法的参数传送的应用ID的值添加到获取的线程结构的应用ID栈1402的头，并将处理返回到应用。

[0108] 在S905中，应用206执行应用的方法中的处理，调用位于方法的末尾的“TASKINFO.remove();”704，并且处理进入TASKINFO类的remove方法。在S907中，TASKINFO类的remove方法首先获取正在执行处理的当前线程。然后，在S1502中，remove方法删除在S907中获取的线程的线程结构的应用ID栈的头的应用ID，并将处理返回到应用。之后，应用206结束方法的处理。

[0109] 对象生成处理

[0110] 接下来，将参照图16描述根据本发明的第二实施例的对象生成处理。通过CPU101将存储在ROM102或者外部存储设备104中的控制程序读出到RAM103并执行该控制程序来实现下文中描述的处理。在字节代码执行单元301正在执行进行对象生成的应用的代码时，开始对象生成处理，并且Java VM204的对象生成单元303被调用。

[0111] 在S1102中，对象生成单元303首先获取当前线程。然后，在S1601中，对象生成单元303获取位于在S1102中获取的线程的结构的应用ID栈的头的应用ID。

[0112] 在S1104中，对象生成单元303获取用于对象生成的内存，并生成对象。然后，在S1105中，对象生成单元303将在S1601中获取的应用ID记录在对象的结构的应用ID字段1003中。此外，在S1106，为了记录内存增加量，对象生成单元303使用在S1105中记录的应用ID以及在S1104中生成的对象的大小作为参数，来调用应用内存管理单元307。

[0113] 在S1107中，应用内存管理单元308使用应用ID搜索使用内存表313的应用ID字段503，并确定是否存在应用ID。如果存在应用ID，则处理进入S1109，而如果不存在应用ID，则进入S1108。在S1108中，应用管理单元308生成使用内存大小为0的新记录，将所生成的记录登记在使用内存表313中，并使处理进入S1109。在S1109中，应用管理单元308更新与使用内存表313的应用ID相对应的使用内存的值504，并将处理返回到对象生成单元303。

[0114] 在S1110中，对象生成单元303将所生成的对象返回到应用。应用206接收所生成的对象并结束对象生成处理。由于GC处理与第一实施例类似，因此省略其描述。

[0115] 如上所述，根据本实施例，在多个应用协同操作的情况下，在线程中记录的应用信息将具有栈结构。由此，即使在如图13所示的情况下，与第一实施例类似，也能够在维持性能的同时测量各应用的内存使用量或者盘使用量。

[0116] 第三实施例

[0117] 下文中，将参照图17至图19描述本发明的第三实施例。请注意，下文中，将省略与第一实施例和第二实施例类似的配置和控制的描述。首先，参照图17描述本实施例的应用管理装置的软件200的模块的示例性配置。

[0118] 如图17所示，除了图3所示的配置以外，应用内存管理单元307还配设有使用内存量声明表1701。与第一实施例类似地执行类加载处理。与第一实施例类似地执行插入了TASKINFO命令的方法执行时的处理。

[0119] 使用内存量声明表

[0120] 接下来，将参照图18描述使用内存量声明表1701。使用内存量声明表1701是将应用ID1801与示出由应用ID识别的应用使用的最大内存量的最大使用内存1802有联系地管理的表。

[0121] 在图4所示的应用的清单文件的MaximumMemoryUsage设定403中记载应用的最大内存使用量。当安装应用时，应用管理单元308从应用的Jar文件读取Application-Id设定402和MaximumMemoryUsage设定403。此外，应用管理单元308将所读取的值添加到使用内存量声明表1701中。

[0122] 对象生成处理

[0123] 接下来，将参照图19描述根据本实施例的对象生成处理的处理过程。通过CPU101将存储在ROM102或者外部存储设备104中的控制程序读出到RAM103并执行该控制程序来实

现下文中描述的处理。

[0124] 在字节代码执行单元301正在执行进行对象生成的应用的代码时开始对象生成处理，并且Java VM204的对象生成单元303被调用。在S1102中，对象生成单元303首先获取当前线程。然后，在S1601中，对象生成单元303读取在S1102中获取的线程的结构的应用ID字段802，并获取应用ID。

[0125] 在S1104中，对象生成单元303获取用于对象生成的内存，并生成对象。然后，在S1105中，对象生成单元303将在S1103中获取的应用ID记录在对象结构的应用ID字段1003中。此外，在S1106，为了记录内存增加量，对象生成单元303使用在S1105中记录的应用ID以及在S1104中生成的对象的大小作为参数，来调用应用内存管理单元307。

[0126] 在S1107中，应用内存管理单元307使用应用ID搜索使用内存表313的应用ID字段503，并确定是否存在应用ID。如果存在应用ID，则处理进入S1901，而如果不存在应用ID，则进入S1108。在S1108中，应用内存管理单元307生成使用内存大小为0的新记录，将所生成的记录登记在使用内存表313中，并使处理进入S1901。

[0127] 在S1901中，应用内存管理单元307使用应用ID搜索使用内存表313的应用ID字段503，并获取当前使用内存大小。

[0128] 接下来，在S1902中，应用内存管理单元307通过应用ID搜索使用内存量声明表1701，并获取与应用ID相对应的应用的使用内存声明量。在S1903中，应用内存管理单元307确定在S1106中从对象生成单元303传送的对象大小与在S1901中获取的当前使用内存大小的总和是否小于或等于在S1902中获取的最大内存使用量。如果总和小于或等于最大内存使用量，则处理进入S1109，如果总和超过最大内存使用量，则进入S1904。

[0129] 在S1109中，应用内存管理单元307更新与使用内存表313的应用ID相对应的使用内存的值504，并将处理返回到对象生成单元303。在S1110中，对象生成单元303将所生成的对象返回到应用。应用接收所生成的对象并结束对象生成处理。

[0130] 另一方面，当在S1903中确定总和超过最大内存使用量时，应用内存管理单元307在S1904中产生内存错误。然后，在S1905中，应用内存管理单元307请求应用管理单元308进行用于停止与应用ID相对应的应用的处理，并将处理返回到应用。应用结束对象生成处理。

[0131] 如上所述，根据本实施例，除了第一实施例和第二实施例中的至少一个的配置以外，还配设有定义各应用的最大内存使用量的表。此外，图像处理装置能够根据对象生成时分配的存储器大小来进行控制，以在当前已经正在使用的内存使用量与分配的内存大小的总和超过最大内存使用量的情况下，停止对象的生成。

[0132] 第四实施例

[0133] 下文中，参照图20至图28描述本发明的第四实施例。在本实施例中，将描述与内存类似地使用第一至第三实施例的机构限制外部存储设备104的盘使用量的实施例。因此，本实施例能够分别与第一至第三实施例组合。请注意，下文中，将省略与第一至第三实施例类似的配置和控制的描述。与第一实施例类似地执行类加载处理。与第一实施例类似地执行插入了TASKINFO命令的方法执行时的处理。

[0134] 在本实施例中，由应用管理单元308安装的应用被存储在外部存储设备104的预定目录中。此外，假定应用被限制为仅能够使用安装了应用自身的目录下的容量。

[0135] 模块配置

[0136] 首先，将参照图20描述根据本实施例的应用管理装置100的软件200的模块的示例性配置。除了图17的配置以外，应用平台205配设有应用盘管理单元2001，Java VM204配设有文件访问管理单元2004。

[0137] 应用盘管理单元2001由稍后描述的使用盘表2002和使用盘量声明表2003构成。文件访问管理单元2004由访问控制器2005构成，并且通过应用206控制对外部存储设备104的文件访问。

[0138] 使用盘表及使用盘量声明表

[0139] 图21示出了使用盘表2002以及使用盘量声明表2003的配置。在使用盘表2002中，将应用ID2101与相应的应用的当前盘使用量2102有联系地管理。在使用盘量声明表2003中，将应用ID2201与相应的应用的最大使用量2202有联系地管理。使用盘表2002和使用盘量声明表2003可以作为单个表来管理，并且不限于本实施例的配置，只要管理等价信息即可。

[0140] 图22示出了本实施例的应用的清单文件的内容。表示应用的最大盘使用量的MaximumFilespaceUsage2301已经被添加到图4所示的内容。请注意，这里，还包括MaximumMemoryUsage设定403，但是本发明不限于此，例如可以采用仅包括MaximumFilespaceUsage2301的配置。

[0141] 初始化处理

[0142] 接下来，将参照图23描述当本实施例的应用平台205启动时应用管理单元308的初始化处理。通过CPU101将存储在ROM102或者外部存储设备104中的控制程序读出到RAM103并执行该控制程序来实现下文中描述的处理。

[0143] 当应用平台205启动时，应用管理单元308在S2401中获取安装的应用的数量。应用管理单元308在S2402中将变量i初始化为0，在S2403中将变量i与应用的数量j进行比较，如果i<j，则进入S2404，并结束处理，如果不是这种情况，则判定针对所有应用进行了稍后讨论的处理。在S2404中，应用管理单元308获取应用ID，并且在S2405中获取由应用声明的盘使用量。然后，调用应用盘管理单元2001。

[0144] 在2406中，应用盘管理单元2001确定在使用盘量声明表中是否登记了相应的应用ID，并且如果未登记，则在S2407中添加记录，并进入S2408。另一方面，如果已登记，则处理直接进入S2408。在S2408中，在将处理返回到应用管理单元308之前，应用盘管理单元2001将应用ID2201的数据与最大使用量2202联系起来，并将该数据添加到使用盘量声明表2003。

[0145] 接下来，在S2409中，应用管理单元308获取安装应用的文件夹下的容量，并再次调用应用盘管理单元2001。在2410中，应用盘管理单元2001确定在使用盘表中是否登记了相应的应用ID，并且如果未登记，则在S2411中添加记录，并进入S2412。另一方面，如果已登记，则处理直接进入S2412。在S2412中，在将处理返回到应用管理单元308之前，应用盘管理单元2001将应用ID2101的数据与最大使用量2102联系起来，并将该数据添加到使用盘表2002。在S2413中，应用管理单元308将变量i递增1，并返回到S2403的确定。

[0146] 请注意，当新安装应用时，应用管理单元308将应用安装在外部存储设备104的预定文件夹中。显然，之后，进行与该流程图类似的处理，将该信息添加到使用盘量声明表2003和使用盘表2002。此外，当卸载应用时，应用管理单元308删除安装了应用的文件夹。显

然，此时从使用盘量声明表2003和使用盘表2002中删除与应用相关的信息。

[0147] 本流程图的处理使得在启动用作图像处理装置的应用管理装置100时，能够管理由应用管理单元308管理的应用的启动初始状态下的盘使用量。

[0148] 对象生成处理

[0149] 图24是生成用于进行文件的生成和写入的FileOutputStream对象的流程图。通过CPU101将存储在ROM102或者外部存储设备104中的控制程序读出到RAM103并执行该控制程序来实现下文中描述的处理。这里，描述与第一实施例中描述的图11的控制不同的部分。

[0150] 当在S1103中获取应用ID时，对象生成单元303在S2501中确定要生成的对象是否为FileOutputStream。如果是FileOutputStream，则处理进入S2502，而在生成其他对象的情况下，处理进入S1104。

[0151] 在S2502中，对象生成单元303获取用于生成FileOutputStream对象的参数，并且在S2503中，根据获取的参数获取操作目标的文件大小。然后，在2504中，对象生成单元303调用用于生成原始FileOutputStream对象的处理。然后，在S2505中，对象生成单元303基于在S2502中获取的参数或者生成对象的方法确定是否正在以Append模式生成FileOutputStream对象。Java中的FileOutputStream对象的规范是：如果为Append模式则操作以将数据添加到现有文件的末尾，而如果不是这种情况，则操作以将文件大小返回至0并进行重写。如果确定结果表示Append模式，则处理进入S1110。

[0152] 另一方面，如果不是Append模式，则判定操作目标的文件大小为0，处理进入S2506。在S2506中，应用盘管理单元2001从使用盘表2002中的当前应用的使用量2102中减去在S2503中获取的大小，将处理返回到对象生成单元303并进入S1110。

[0153] 在S1110中，对象生成单元303将处理返回到应用，应用结束对象生成处理。

[0154] FileOutputStream对象的write方法

[0155] 接下来，将参照图25描述应用206调用FileOutputStream对象的write方法时的处理过程。在图9和图15中的S905的处理期间执行下文中描述的处理。此外，通过CPU101将存储在ROM102或者外部存储设备104中的控制程序读出到RAM103并执行该控制程序来实现下文中描述的处理。在执行时，字节代码执行单元301调用文件访问管理单元2004的访问控制器2005的处理。

[0156] 在S2601中，字节代码执行单元301确定方法是否是FileOutputStream对象的write方法。如果方法是FileOutputStream对象的write方法，则处理进入S2602，而如果不是这种情况，则在S2611中调用其他方法的处理，并结束当前处理。

[0157] 在执行write方法时，文件访问管理单元2004在S2602中读取当前线程的线程结构的应用ID字段802。然后，在S2603中，文件访问管理单元2004获取被传送到write方法的写入数据的大小(=a)，并将处理传送到应用盘管理单元2001。

[0158] 接下来，在S2604中，应用盘管理单元2001使用在S2602中获取的应用ID，并从使用盘表2002中获取应用的使用量2012(=X)。此外，在S2605中，应用盘管理单元2001从使用盘量声明表2003中获取最大使用量2202(=Y)，并将处理返回到文件访问管理单元2004。

[0159] 在S2606中，文件访问管理单元2004确定通过将应用的使用量X与写入数据大小a相加而获得的大小是否超过最大使用量Y。如果超过最大使用量Y，则处理进入S2608，文件访问管理单元2004判定要进行的写入超过由应用声明的最大使用量2202，通知预定错误

(盘错误)，并结束当前处理。

[0160] 如果未超过最大使用量Y，则处理进入S2607，文件访问管理单元2004调用原始write方法，并执行文件的写入。之后，在S2609中，文件访问管理单元2004确定文件的写入是否失败。如果文件的写入失败，则处理结束，并将处理传送到应用盘管理单元2001，而如果文件的写入成功，则处理进入S2610。在S2610中，应用盘管理单元2001将写入数据大小(=a)与使用盘表2002中的应用的使用量2102相加，并结束处理。

[0161] Java具有与FileOutputStream类似的FileWriter类。由于在使用的情况下通过采用与FileOutputStream相同的配置能够支持FileWriter类，因此在本实施例中省略其描述。

[0162] RandomAccessFile对象的setLength方法

[0163] 接下来，将参照图26描述应用206调用RandomAccessFile对象的setLength方法时的处理过程。在图9和图15中的S905的处理期间执行下文中描述的处理。此外，通过CPU101将存储在ROM102或者外部存储设备104中的控制程序读出到RAM103并执行该控制程序来实现下文中描述的处理。在执行时，字节代码执行单元301调用文件访问管理单元2004的访问控制器2005的处理。Java中的RandomAccessFile对象的setLength规范是：操作以将操作目标的文件大小改变为由setLength指定的大小。

[0164] 在S2701中，字节代码执行单元301确定方法是否是RandomAccessFile对象的setLength方法。如果方法是RandomAccessFile对象的setLength方法，则处理进入S2702，而如果不是这种情况，则处理进入S2716并调用其他方法的处理，之后结束当前处理。

[0165] 在执行setLength方法时，文件访问管理单元2004在S2702中读取当前线程的线程结构的应用ID字段802。然后，在S2703中，文件访问管理单元2004获取setLength方法的目标文件的大小(=a)，并在S2704中获取在setLength方法中指定的大小(=b)。之后，在S2705中，文件访问管理单元2004比较a和b，并且如果b较大，即如果文件大小较大，则进入S2709。如果不是这种情况，则文件访问管理单元2004进入S2706，调用原始setLength方法，并在S2707中确定setLength方法的执行结果。如果成功，则处理进入S2708，如果未成功，则处理结束。在S2708中，应用盘管理单元2001从使用盘表2002中的应用的使用量2102中减去通过a-b得到的值，并结束处理。

[0166] 另一方面，如果在S2705中确定文件大小较大，则应用盘管理单元2001在S2709中获取在S2701中获取的应用ID的盘使用量2102(=X)。然后，在S2710中，应用盘管理单元2001获取最大使用量2202(=Y)。然后，在S2711中，文件访问管理单元2004确定当前使用量X中由setLength方法产生的增加是否超过最大使用量Y(X+(b-a)>Y)。

[0167] 如果超过最大使用量Y，则处理进入S2715，并且文件访问管理单元2004判定要进行的写入超过由应用206声明的最大使用量2202，通知预定错误，并结束当前处理。另一方面，如果判定未超过最大使用量Y，则处理进入S2712，文件访问管理单元2004调用原始setLength方法。然后，在S2713中，文件访问管理单元2004确定setLength方法的执行结果，并且如果成功，则进入S2714。在S2714中，应用盘管理单元2001将使用盘表2002中的应用的使用量2102与通过b-a得到的值相加，并结束处理。另一方面，如果在S2713中确定执行结果错误，则结束处理。

[0168] RandomAccessFile对象的write方法

[0169] 接着，将参照图27描述应用206调用RandomAccessFile对象的write方法时的处理过程。在图9和图15中的S905的处理期间执行下文中描述的处理。此外，通过CPU101将存储在ROM102或者外部存储设备104中的控制程序读出到RAM103并执行该控制程序来实现在下文中描述的处理。在执行时，字节代码执行单元301调用文件访问管理单元2004的访问控制器2005的处理。Java中的RandomAccessFile对象的write规范是：操作以从作为操作目标的文件的文件指针的位置写入数据。此时，write规范是：在写入大小超过文件大小的情况下操作，使得文件大小增加。

[0170] 在S2801中，字节代码执行单元301确定方法是否是RandomAccessFile对象的write方法。如果方法是RandomAccessFile对象的write方法，则处理进入S2802，而如果不是这种情况，则在S2815中调用其他方法的处理，并结束当前处理。

[0171] 在执行write方法时，文件访问管理单元2004在S2802中读取当前线程的线程结构的应用ID字段802。然后，在S2803中，文件访问管理单元2004获取被传送到write方法的写入数据的大小(=a)，并且在S2804中获取文件指针的当前位置(=b)。此外，在S2805中，文件访问管理单元2004获取操作目标文件的大小(=c)。之后，在S2806中，文件访问管理单元2004将文件指针的位置b与写入大小a相加，并确定文件大小是否增加(a+b>c)。如果文件大小增加，则处理进入S2808，而如果不是这种情况，则在S2807中调用原始write方法，由于文件大小未改变，因此当前处理结束。

[0172] 在S2808中，应用管理单元2001获取在S2801中获取的应用ID的盘使用量2102(=X)，在S2809中获取最大使用量(=Y)，并进入S2810。在S2810中，文件访问管理单元2004确定当前使用量X中由write方法产生的增加是否超过最大使用量Y((a+b)-c+X>Y)。如果超过最大使用量Y，则处理进入S2814，并且文件访问管理单元2004判定要进行的写入超过由应用声明的最大使用量2202，通知预定错误，并结束当前处理。

[0173] 另一方面，如果判定未超过最大使用量Y，则处理进入S2811，并且文件访问管理单元2004调用原始write方法。之后，在S2812中，文件访问管理单元2004确定write方法的执行结果。如果成功，则处理进入S2813，应用盘管理单元2001将通过a+b-c得到的值与使用盘表2002中的应用的使用量2102相加，并结束处理。另一方面，如果在S2811中确定write方法未被成功执行，则结束该处理。

[0174] File对象的delete方法

[0175] 接下来，将参照图28描述当应用206调用File对象的delete方法时的处理过程。在图9和图15中的S905的处理期间执行下文中描述的处理。此外，通过CPU101将存储在ROM102或者外部存储设备104中的控制程序读出到RAM103并执行该控制程序来实现下文中描述的处理。

[0176] 在S2901中，字节代码执行单元301确定方法是否是File对象的delete方法。如果方法是File对象的delete方法，则处理进入S2902，而如果不是这种情况，则处理进入S2907，并调用其他方法的处理，之后结束当前处理。

[0177] 在S2902中，文件访问管理单元2004在执行delete方法时，读取当前线程的线程结构的应用ID字段802。然后，在S2903中，文件访问管理单元2004获取delete方法的目标文件的大小(=a)，并在S2904中调用原始delete方法。

[0178] 在S2905中，文件访问管理单元2004确定delete方法的执行结果，并且如果确定

delete方法未被成功执行，则结束当前处理。另一方面，如果确定delete方法被成功执行，则处理进入S2906。在S2906中，应用盘管理单元2001从使用盘表2002中的应用的使用量2102中减去操作目标的文件大小，并结束处理。

[0179] 请注意，在本实施例中，如果超过应用206生成的最大使用量，则通知预定错误(S2608、S2715、S2814)。应用管理单元308还可以被构成为在检测到该通知的情况下停止相应的应用。

[0180] 如上所述，根据本实施例的图像处理装置响应于应用启动请求，读取应用的类的类文件，在所读取的类文件中包括的方法的开头添加用于记录表示应用的应用信息的代码，并加载类。此外，该图像处理装置在方法的执行期间，分配要用于待生成的对象的文件大小，并且使用分配的文件大小作为盘使用量，记录在线程中记录的应用信息，同时生成对象，并将所生成的对象的应用信息与盘使用量有联系地管理。由此，在本实施例中，关于盘使用量也能够获得与第一至第三实施例类似的效果。

[0181] 其他实施例

[0182] 本发明的实施例还能够由读出并执行记录在存储介质(例如非暂时性计算机可读存储介质)上的用于执行本发明的上述实施例的一个或者多个功能的计算机可执行指令的系统或装置的计算机来实现，以及通过由系统或装置的计算机例如读出并执行来自存储介质的用于执行上述实施例的一个或者多个功能的计算机可执行指令来执行的方法来实现。计算机可以包括中央处理单元(CPU)、微处理单元(MPU)或者其他电路中的一个或者多个，并且可以包括独立的计算机或者独立的计算机处理器的网络。计算机可执行指令可以从例如网络或者存储介质提供给计算机。存储介质可以包括例如硬盘、随机存取存储器(RAM)、只读存储器(ROM)、分布式计算机系统的存储、光盘(例如光盘(CD)、数字通用盘(DVD)或者蓝光盘(BD)™)、闪存设备、存储器卡等中的一个或者多个。

[0183] 虽然参照示例性实施例描述了本发明，但是应当理解，本发明不限于所公开的示例性实施例。应对所附权利要求的范围给予最宽的解释，以使其覆盖所有变型例以及等同结构和功能。

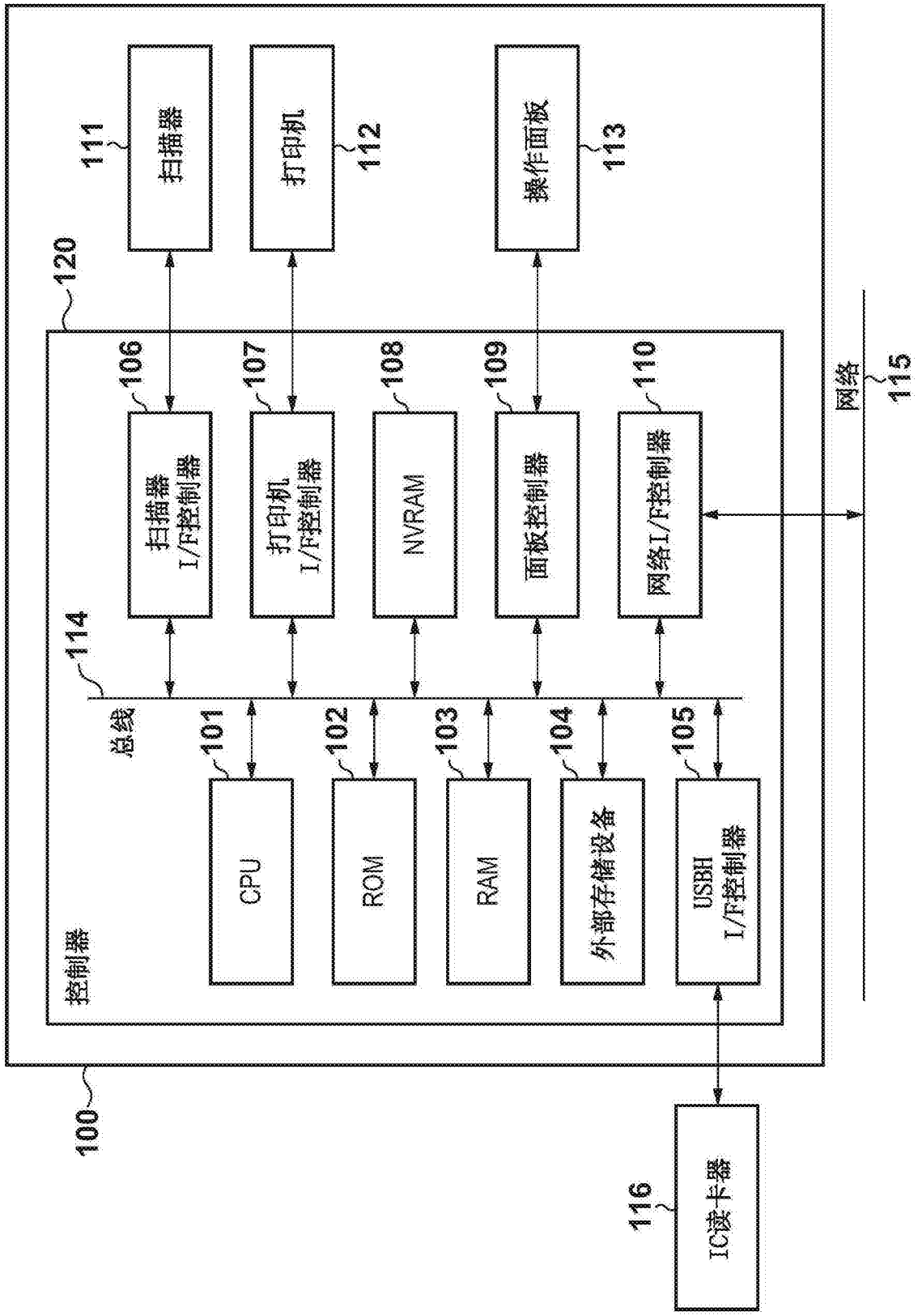

100
120
111
扫描器
112
打印机
113
操作面板
106
扫描器 I/F控制器
107
打印机 I/F控制器
108
NVRAM
109
面板控制器
110
网络I/F控制器
115
网络
114
总线
101
CPU
102
ROM
103
RAM
104
外部存储设备
105
USBH I/F控制器
控制器
116
IC读卡器

图1

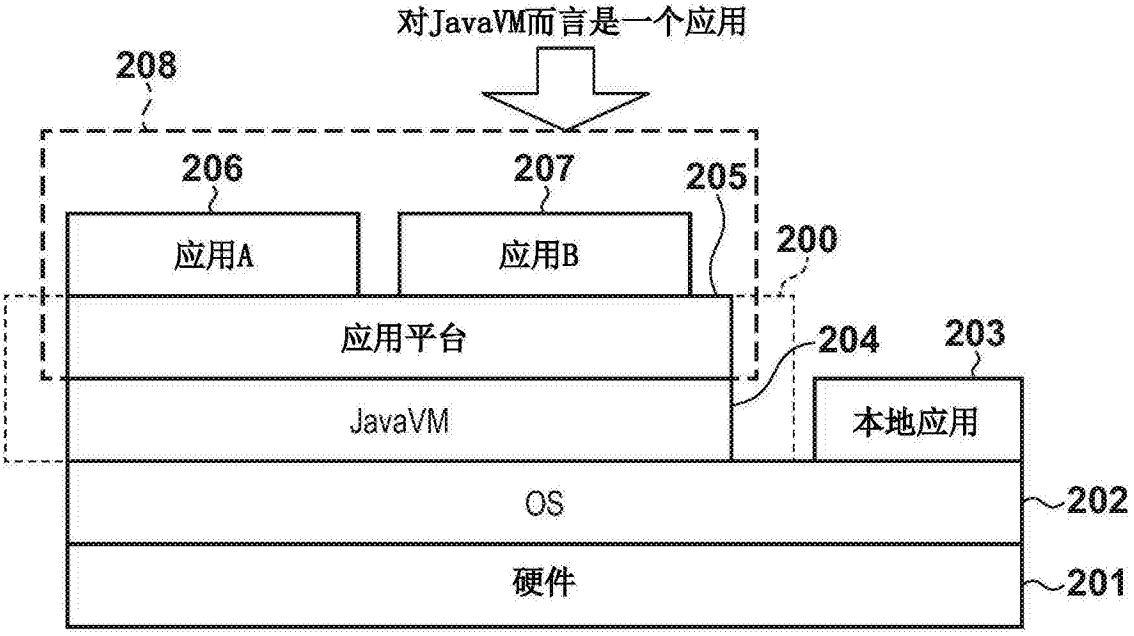

对JavaVM而言是一个应用
208
206
207
205
200
应用A
应用B
应用平台
204
203
JavaVM
本地应用
OS
202
硬件
201

图2

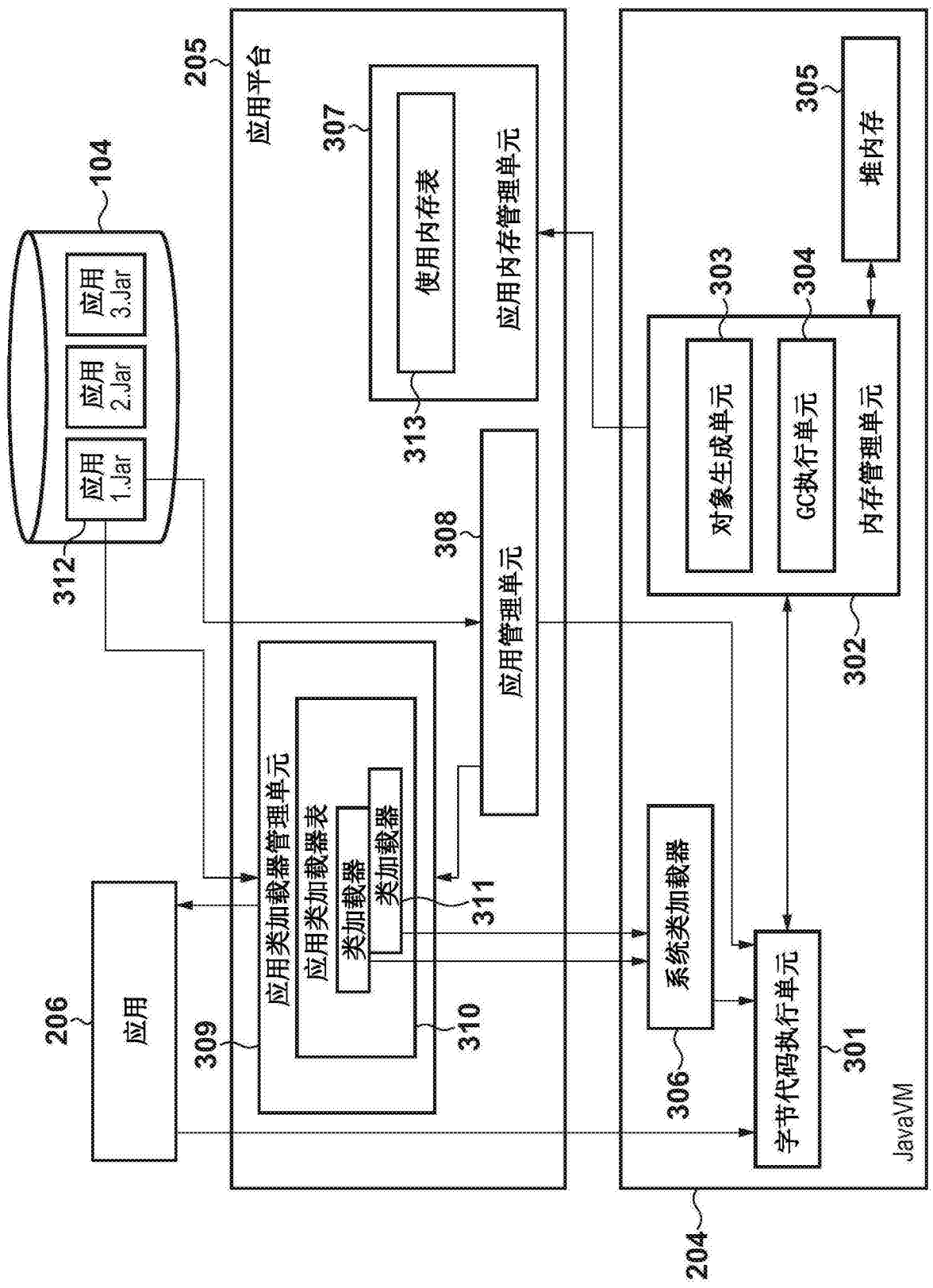

205
应用平台
307
使用内存表
应用内存管理单元
313
305
堆内存
104
应用 3.Jar
应用 2.Jar
应用 1.Jar
312
308
303
304
对象生成单元
GC执行单元
内存管理单元
302
应用管理单元
应用类加载器管理单元
应用类加载器表
类加载器
类加载器
311
206
应用
309
310
系统类加载器
306
字节代码执行单元
301
JavaVM
204

图3

Bundle-Name : Application A ~401
Application-Id : 11-1111-1111 ~402
MaximumMemoryUsage : 2000KB ~402

图4

501 502

应用ID	类加载器
11-1111-1111	App1Calssloader
22-2222-2222	App2Calssloader
33-3333-3333	App3Calssloader

应用类加载器表
310

503 504

应用ID	使用内存
11-1111-1111	500KB
22-2222-2222	100KB
33-3333-3333	2000KB

使用内存表
313

图5

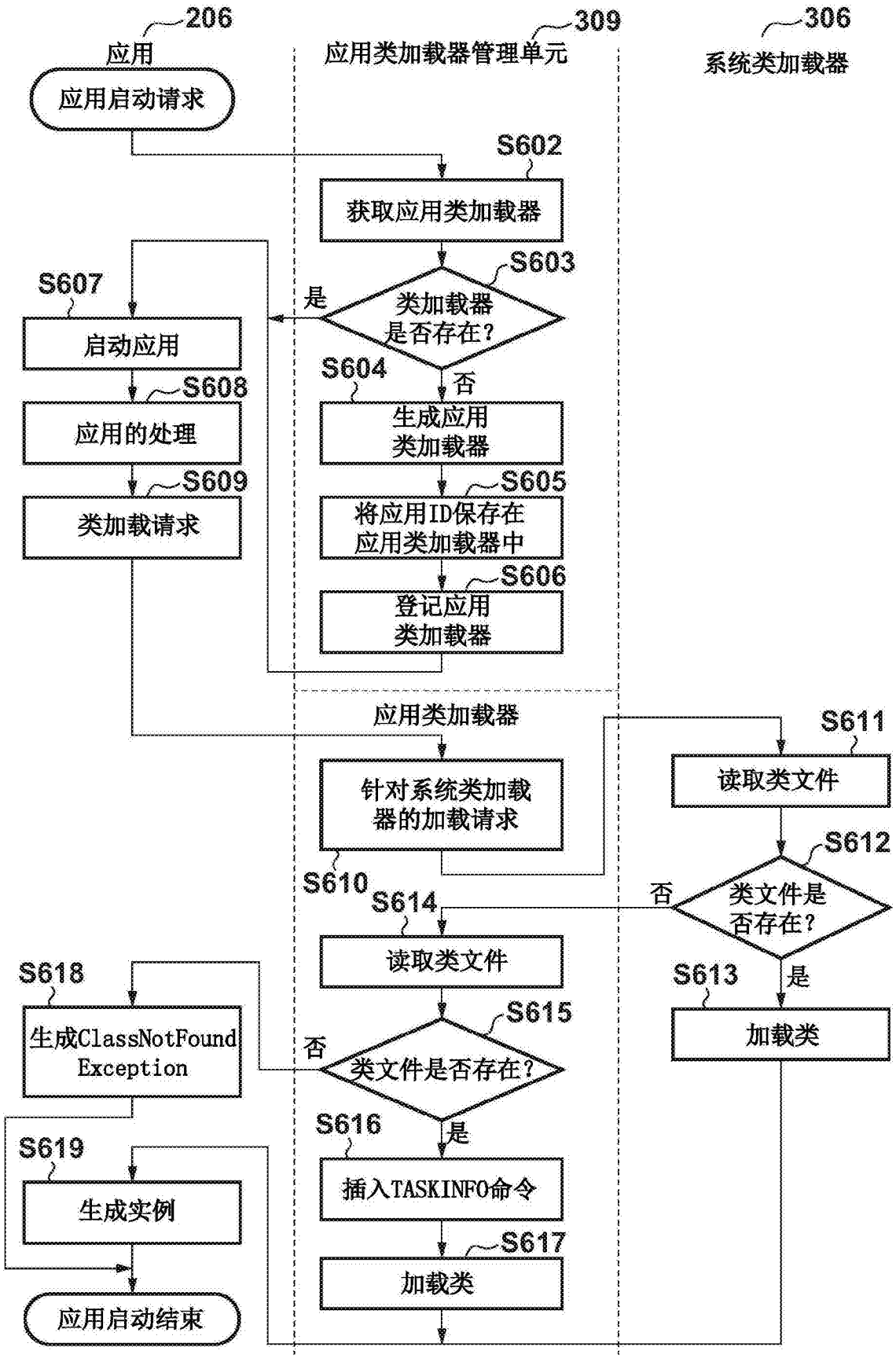

206
应用
应用启动请求
309
应用类加载器管理单元
306
系统类加载器
S602
获取应用类加载器
S603
类加载器是否存在?
是
否
S607
启动应用
S608
应用的处理
S609
类加载请求
S604
生成应用类加载器
S605
将应用ID保存在应用类加载器中
S606
登记应用类加载器
应用类加载器
针对系统类加载器的加载请求
S610
S611
读取类文件
S612
类文件是否存在?
否
是
S613
加载类
S614
读取类文件
S615
类文件是否存在?
否
是
S616
插入TASKINFO命令
S617
加载类
S618
生成ClassNotFoundException
S619
生成实例
应用启动结束

图6

701

```
public Class App1_Class {
    public void app_method(type arg) {
        方法的处理
        String  a = new String("AAA");
        Hoge hoge = new Hoge();
        Button button = new Button("OK");
    }
}
```

702

```
public Class App1_Class {
    public void app_method(type arg) {
        TASKINFO.set("11-1111-1111");   703
        方法的处理
        String  a = new String("AAA");
        Hoge hoge = new Hoge();
        Button button = new Button("OK");
        TASKINFO.remove();   704
    }
}
```

图7

图8

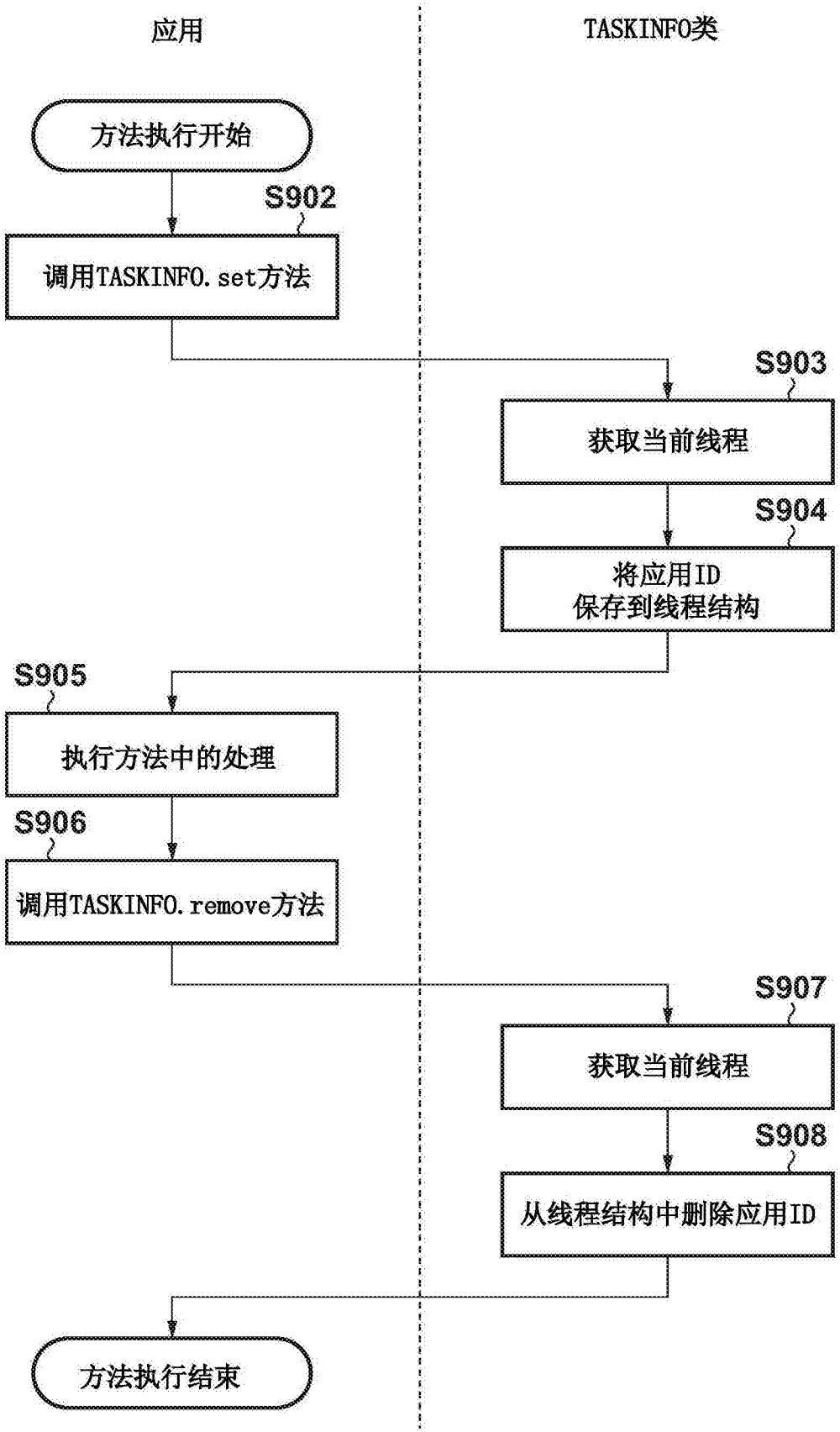

应用
TASKINFO类
方法执行开始
S902
调用TASKINFO.set方法
S903
获取当前线程
S904
将应用ID
保存到线程结构
S905
执行方法中的处理
S906
调用TASKINFO.remove方法
S907
获取当前线程
S908
从线程结构中删除应用ID
方法执行结束

图9

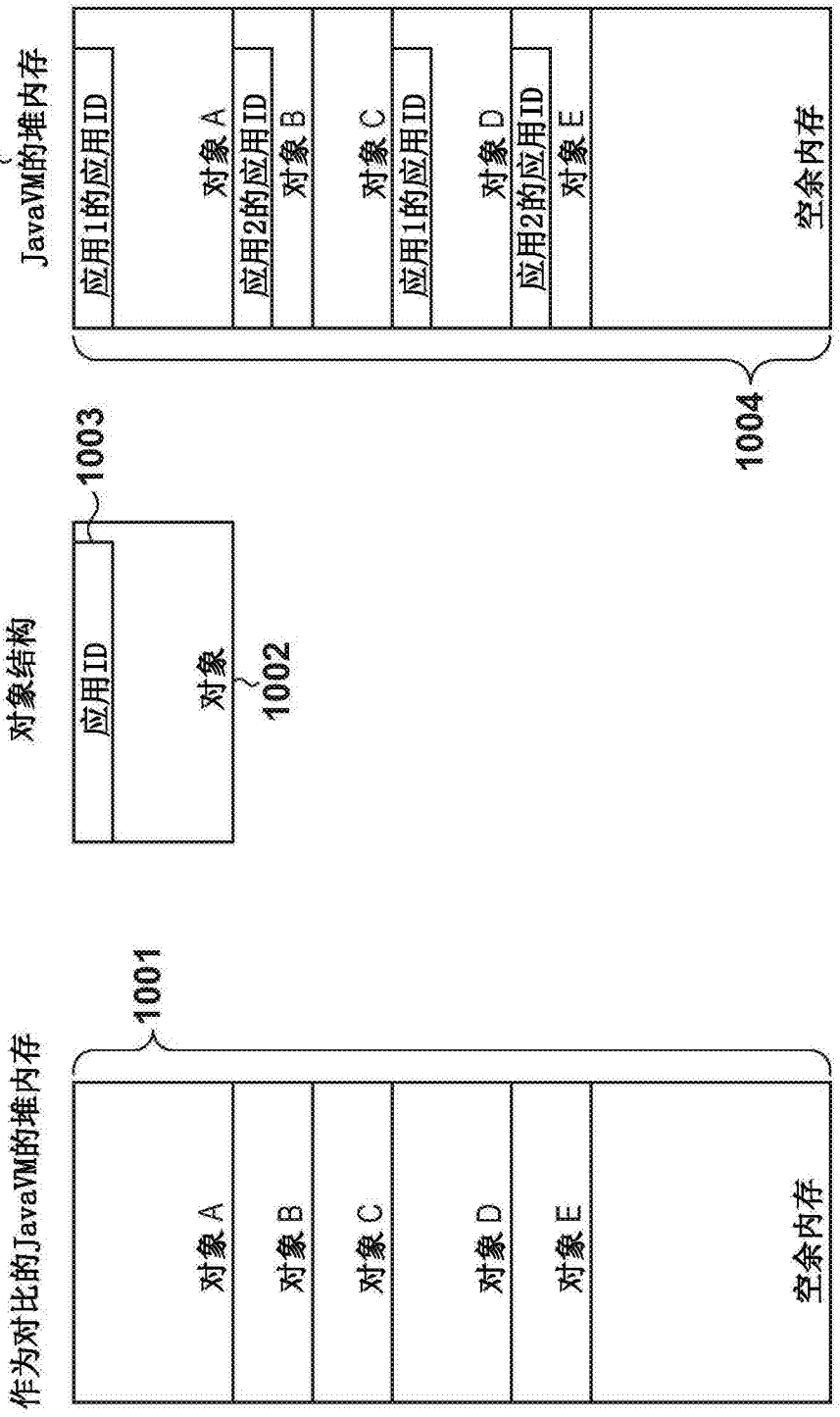

305
JavaVM的堆内存
应用1的应用ID
对象A
应用2的应用ID
对象B
对象C
应用1的应用ID
对象D
应用2的应用ID
对象E
空余内存
1004
1003
对象结构
应用ID
对象
1002
1001
作为对比的JavaVM的堆内存
对象A
对象B
对象C
对象D
对象E
空余内存

图10

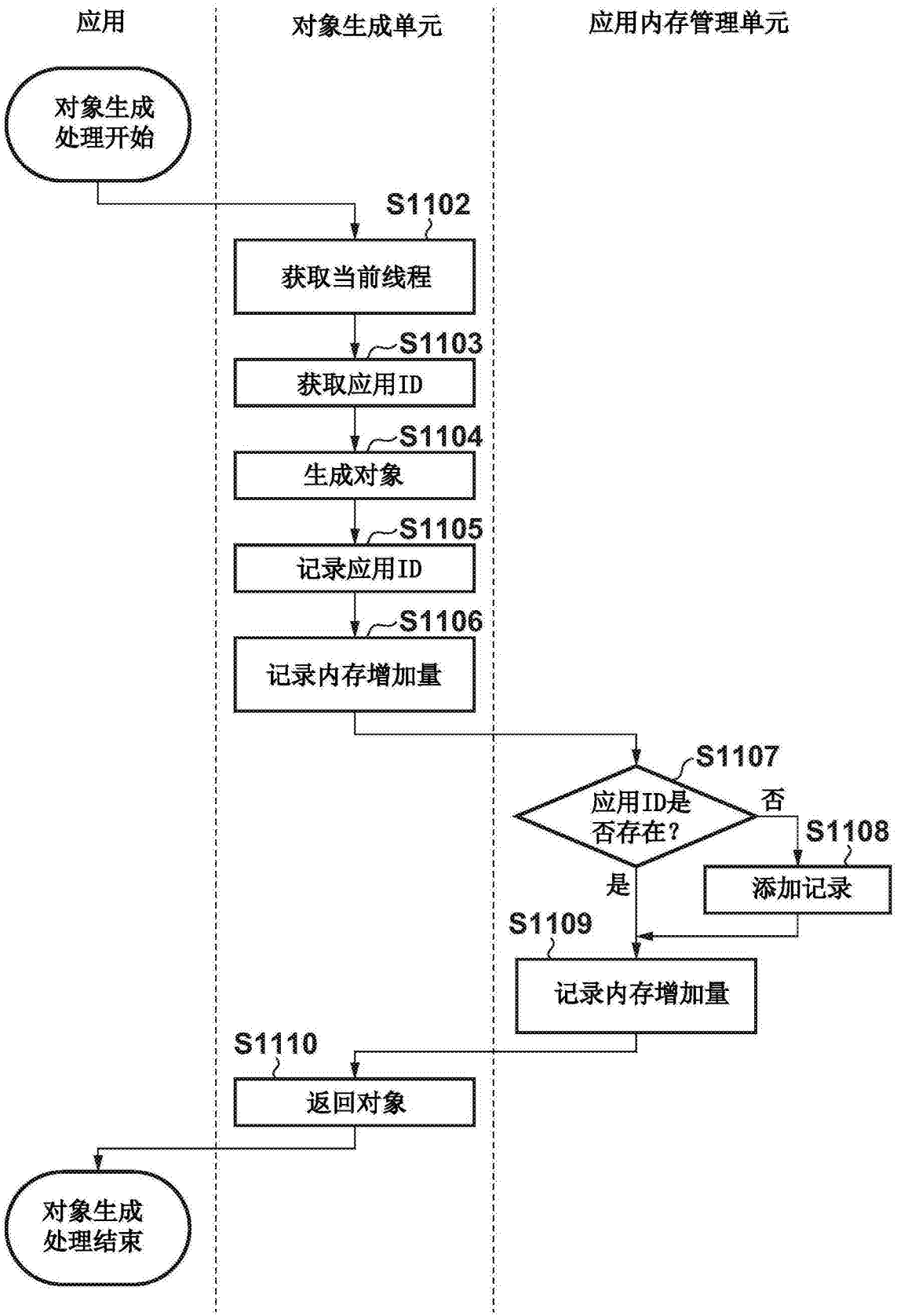

应用
对象生成单元
应用内存管理单元
对象生成处理开始
S1102
获取当前线程
S1103
获取应用ID
S1104
生成对象
S1105
记录应用ID
S1106
记录内存增加量
S1107
应用ID是否存在?
否
S1108
添加记录
是
S1109
记录内存增加量
S1110
返回对象
对象生成处理结束

图11

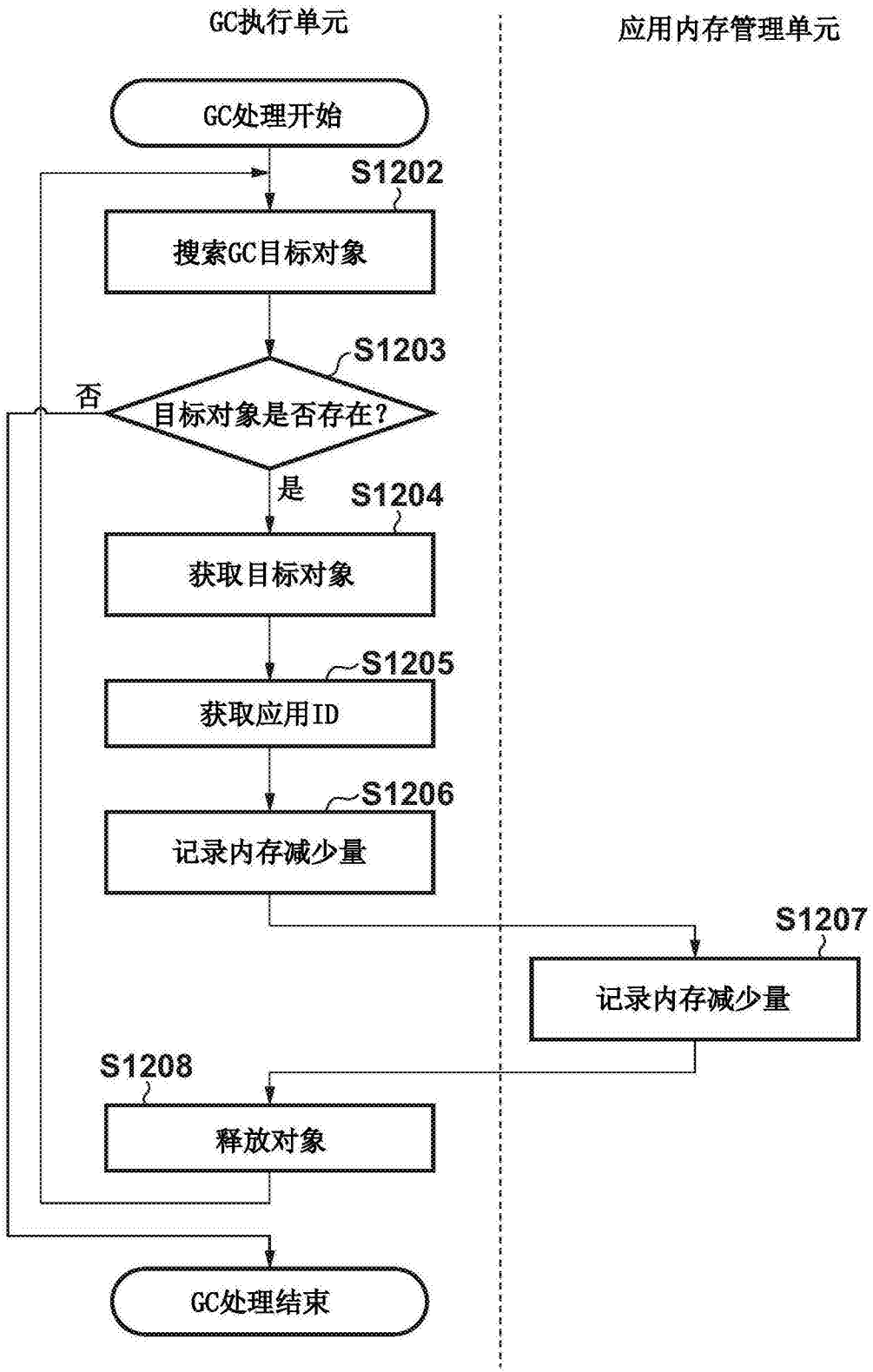

GC执行单元
应用内存管理单元
GC处理开始
S1202
搜索GC目标对象
S1203
否
目标对象是否存在?
是
S1204
获取目标对象
S1205
获取应用ID
S1206
记录内存减少量
S1207
记录内存减少量
S1208
释放对象
GC处理结束

图12

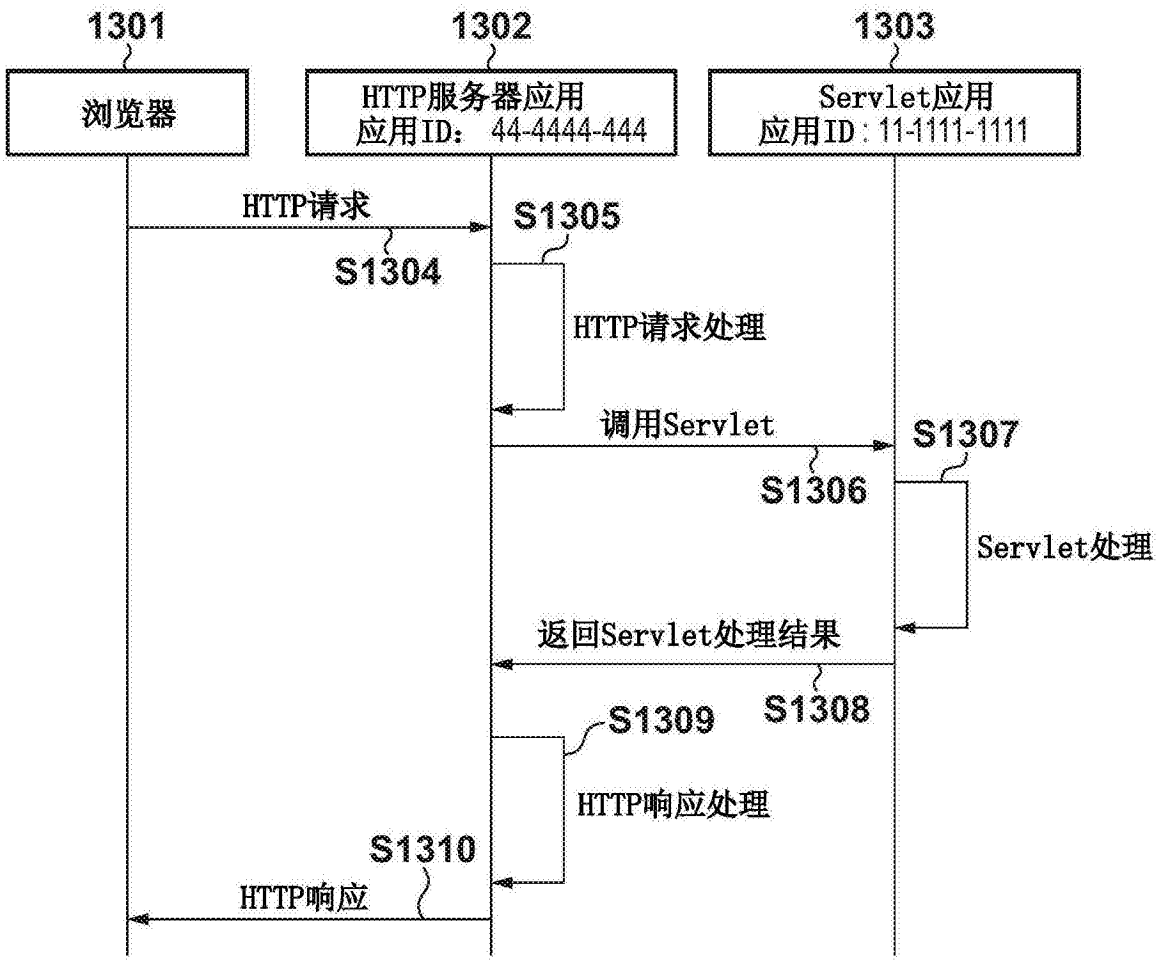

1301
1302
1303
浏览器
HTTP服务器应用
应用ID： 44-4444-444
Servlet应用
应用ID：11-1111-1111
HTTP请求
S1304
S1305
HTTP请求处理
调用Servlet
S1306
S1307
Servlet处理
返回Servlet处理结果
S1308
S1309
HTTP响应处理
S1310
HTTP响应

图13

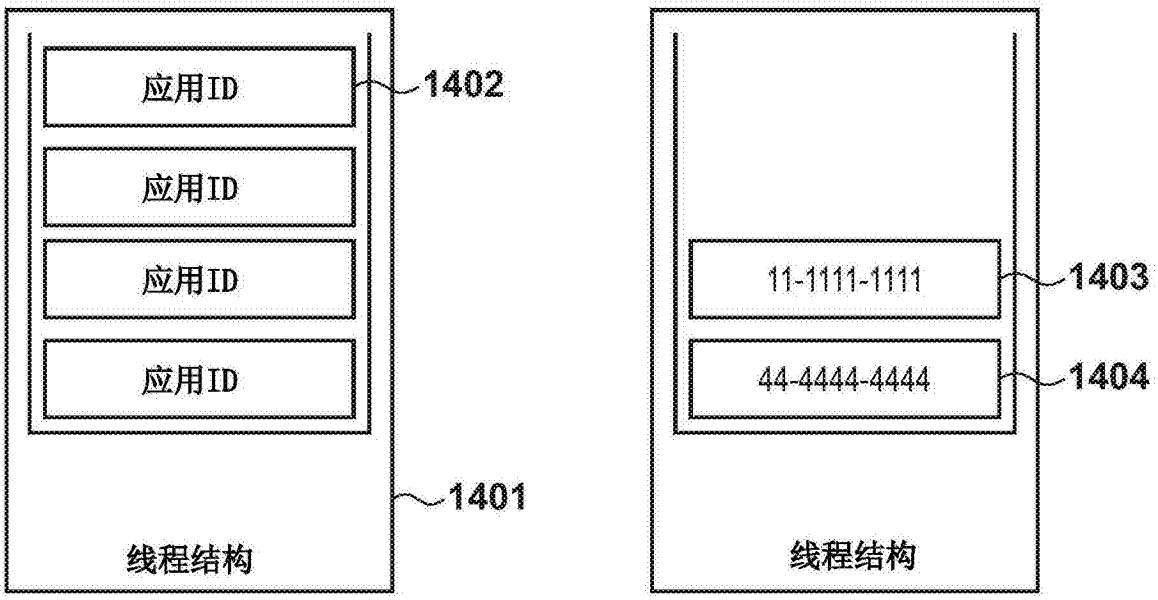

应用ID
应用ID
应用ID
应用ID
1402
1401
线程结构
11-1111-1111
44-4444-4444
1403
1404
线程结构

图14

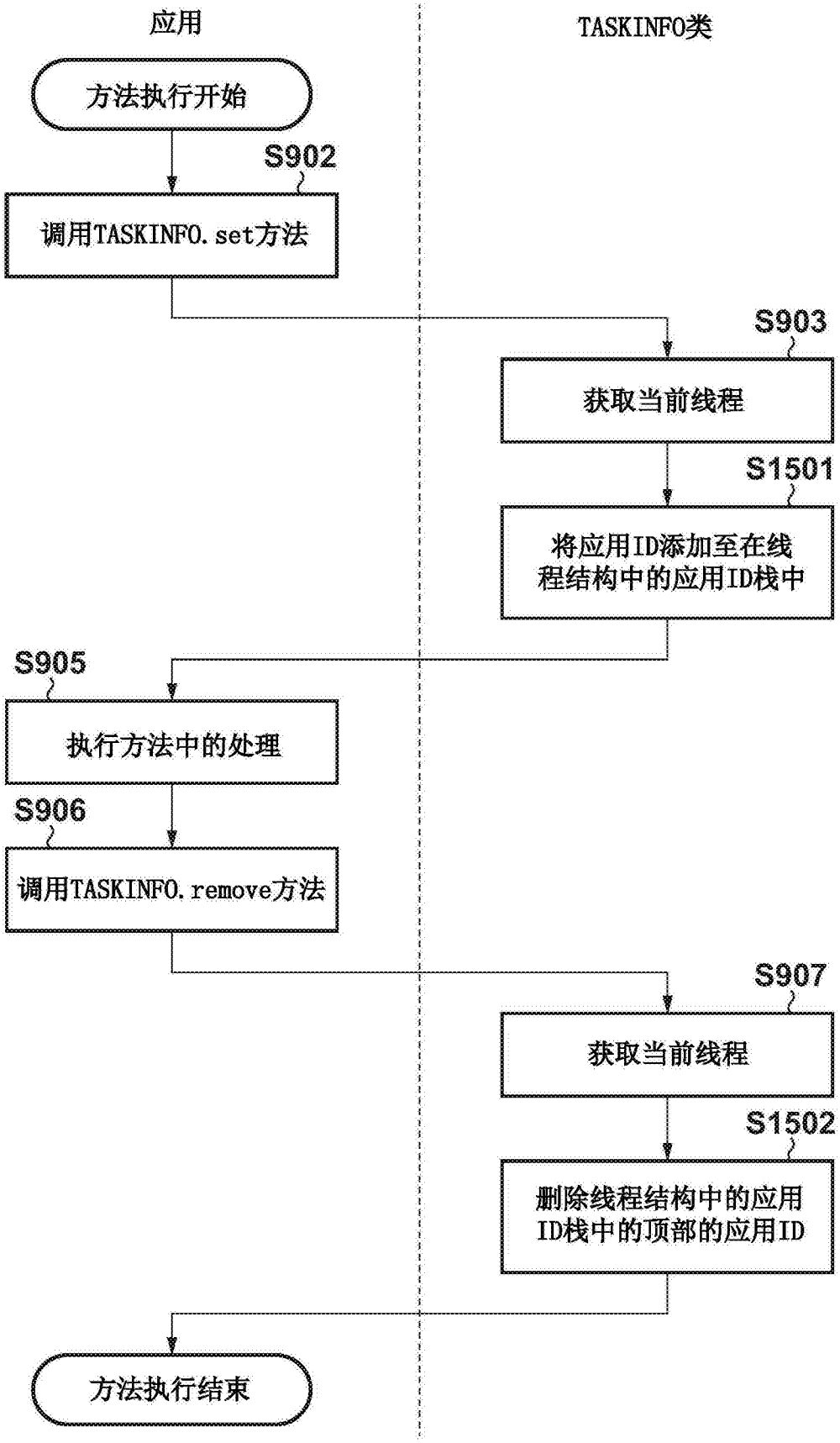

应用
TASKINFO类
方法执行开始
S902
调用TASKINFO.set方法
S903
获取当前线程
S1501
将应用ID添加至在线程结构中的应用ID栈中
S905
执行方法中的处理
S906
调用TASKINFO.remove方法
S907
获取当前线程
S1502
删除线程结构中的应用ID栈中的顶部的应用ID
方法执行结束

图15

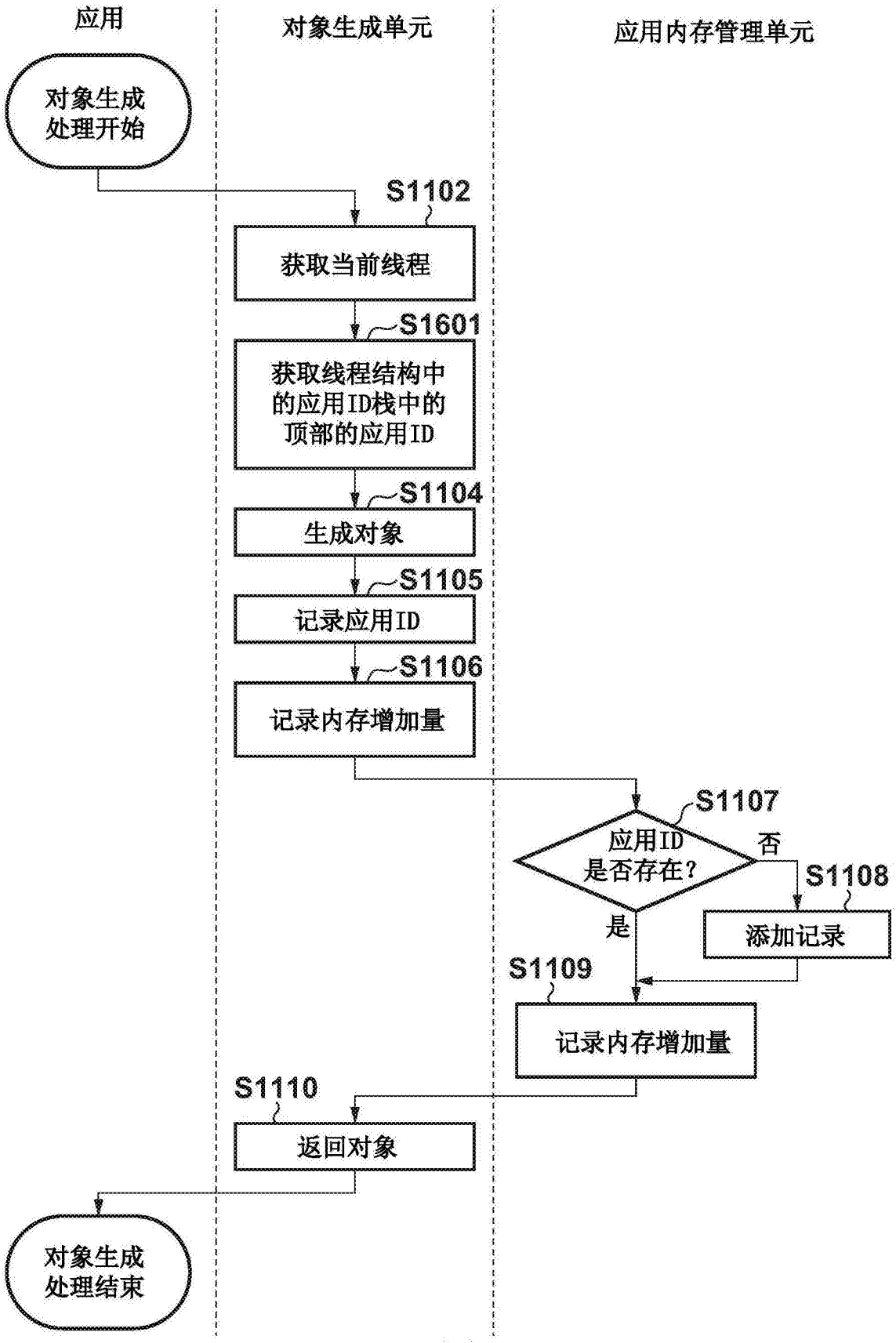

应用
对象生成单元
应用内存管理单元
对象生成处理开始
S1102
获取当前线程
S1601
获取线程结构中的应用ID栈中的顶部的应用ID
S1104
生成对象
S1105
记录应用ID
S1106
记录内存增加量
S1107
应用ID是否存在？
否
S1108
添加记录
是
S1109
记录内存增加量
S1110
返回对象
对象生成处理结束

图16

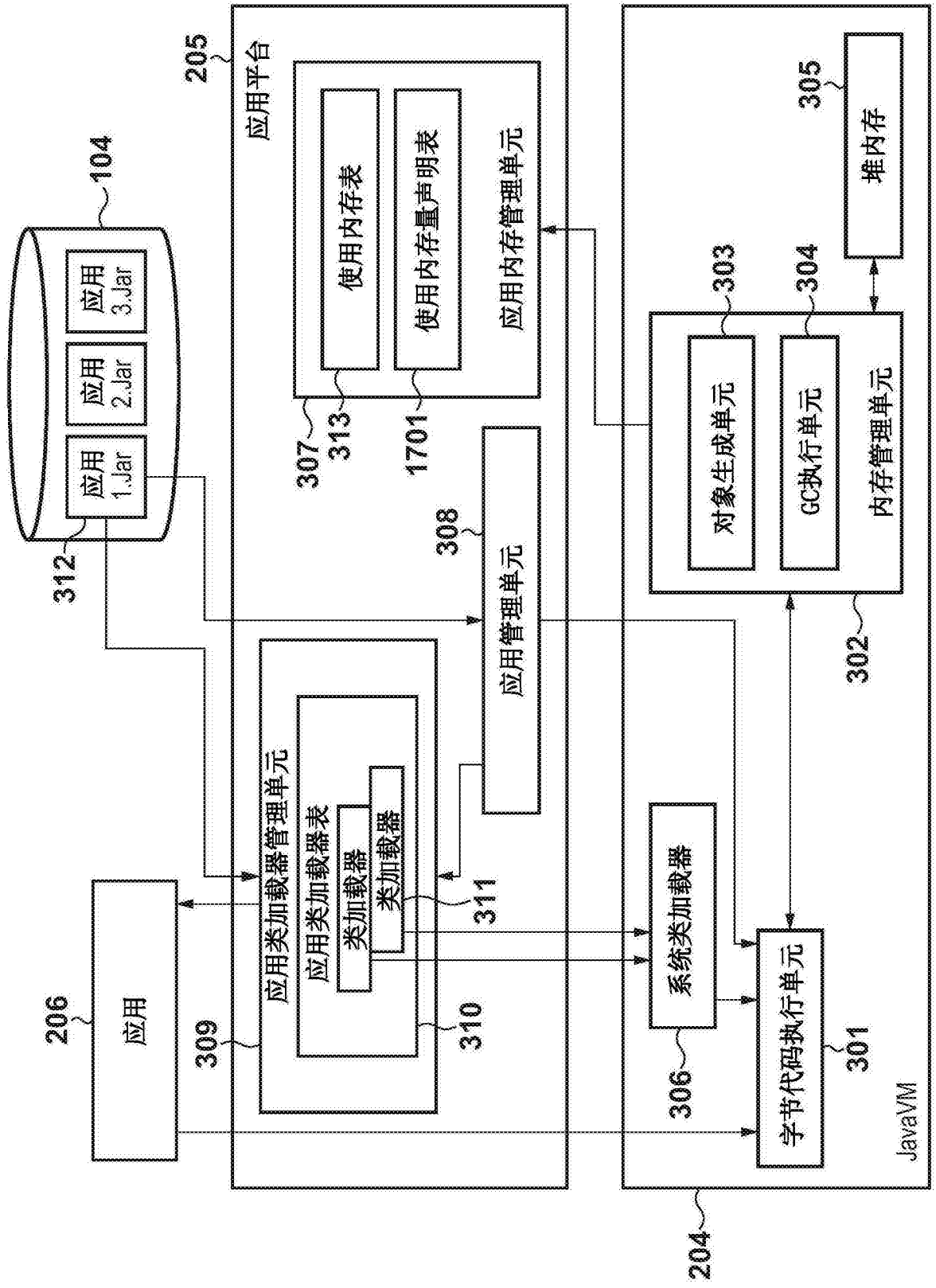

205
应用平台
104
应用 3.Jar
应用 2.Jar
应用 1.Jar
312
使用内存表
使用内存量声明表
应用内存管理单元
307
313
1701
308
应用管理单元
应用类加载器管理单元
应用类加载器表
类加载器
类加载器
311
310
309
206
应用
系统类加载器
306
字节代码执行单元
301
204
JavaVM
305
堆内存
303
304
对象生成单元
GC执行单元
内存管理单元
302

图17

1801 1802

应用ID	最大使用内存
11-1111-1111	2000KB
22-2222-2222	1000KB
33-3333-3333	3000KB

使用内存量声明表

1701

图18

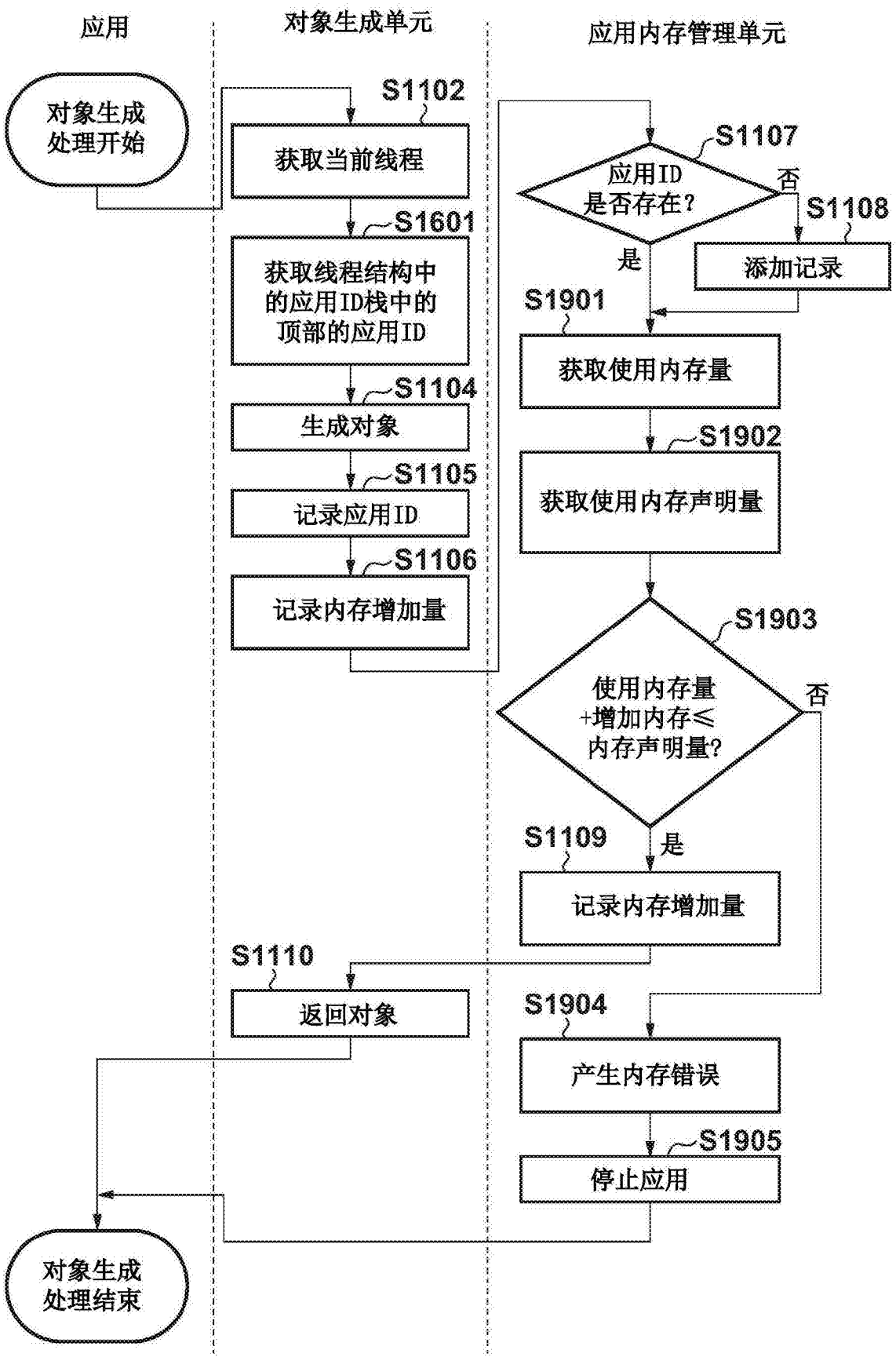

应用
对象生成单元
应用内存管理单元
对象生成处理开始
S1102
获取当前线程
S1601
获取线程结构中的应用ID栈中的顶部的应用ID
S1104
生成对象
S1105
记录应用ID
S1106
记录内存增加量
S1107
应用ID是否存在?
否
S1108
添加记录
是
S1901
获取使用内存量
S1902
获取使用内存声明量
S1903
使用内存量+增加内存≤内存声明量?
否
S1109
是
记录内存增加量
S1110
返回对象
S1904
产生内存错误
S1905
停止应用
对象生成处理结束

图19

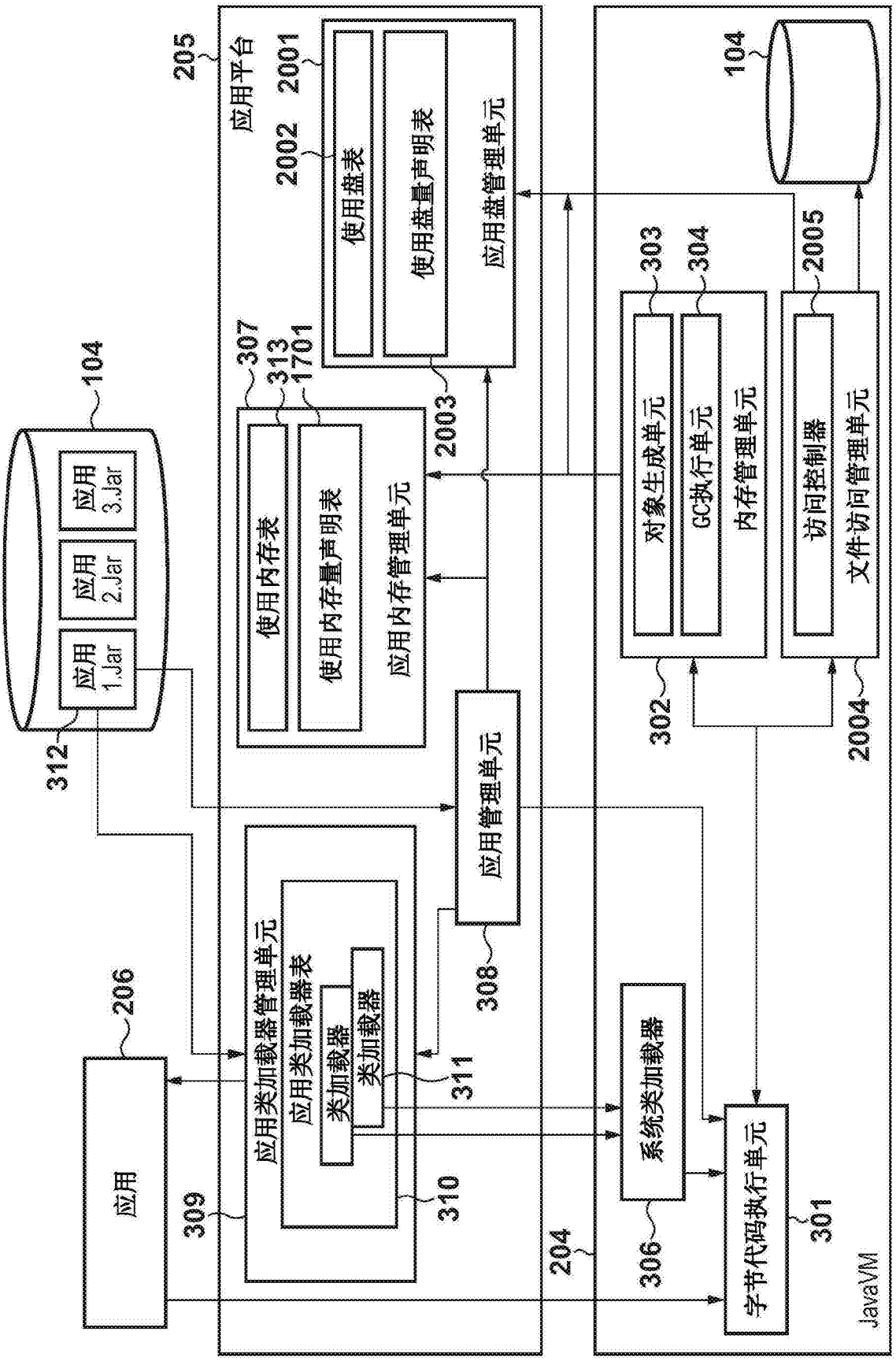

205
应用平台
2001
2002
使用盘表
使用盘量声明表
应用盘管理单元
104
307
313
1701
使用内存表
使用内存量声明表
应用内存管理单元
2003
303
304
2005
对象生成单元
GC执行单元
内存管理单元
访问控制器
文件访问管理单元
104
应用
1.Jar
应用
2.Jar
应用
3.Jar
312
应用管理单元
308
302
2004
206
应用
309
应用类加载器管理单元
应用类加载器表
类加载器
类加载器
311
310
系统类加载器
306
204
字节代码执行单元
301
JavaVM

图20

2101 2102

应用ID	使用量
11-1111-1111	20124KB
22-2222-2222	128KB
33-3333-3333	10968KB

使用盘表

2002

2201 2202

应用ID	最大使用量
11-1111-1111	25000KB
22-2222-2222	512KB
33-3333-3333	12480KB

使用盘量声明表

2003

图21

Bundle-Name : Application A ~401
Application-Id : 11-1111-1111 ~402
MaximumMemoryUsage : 2000KB ~402
MaximumFilespaceUsage : 25000KB ~2301

图22

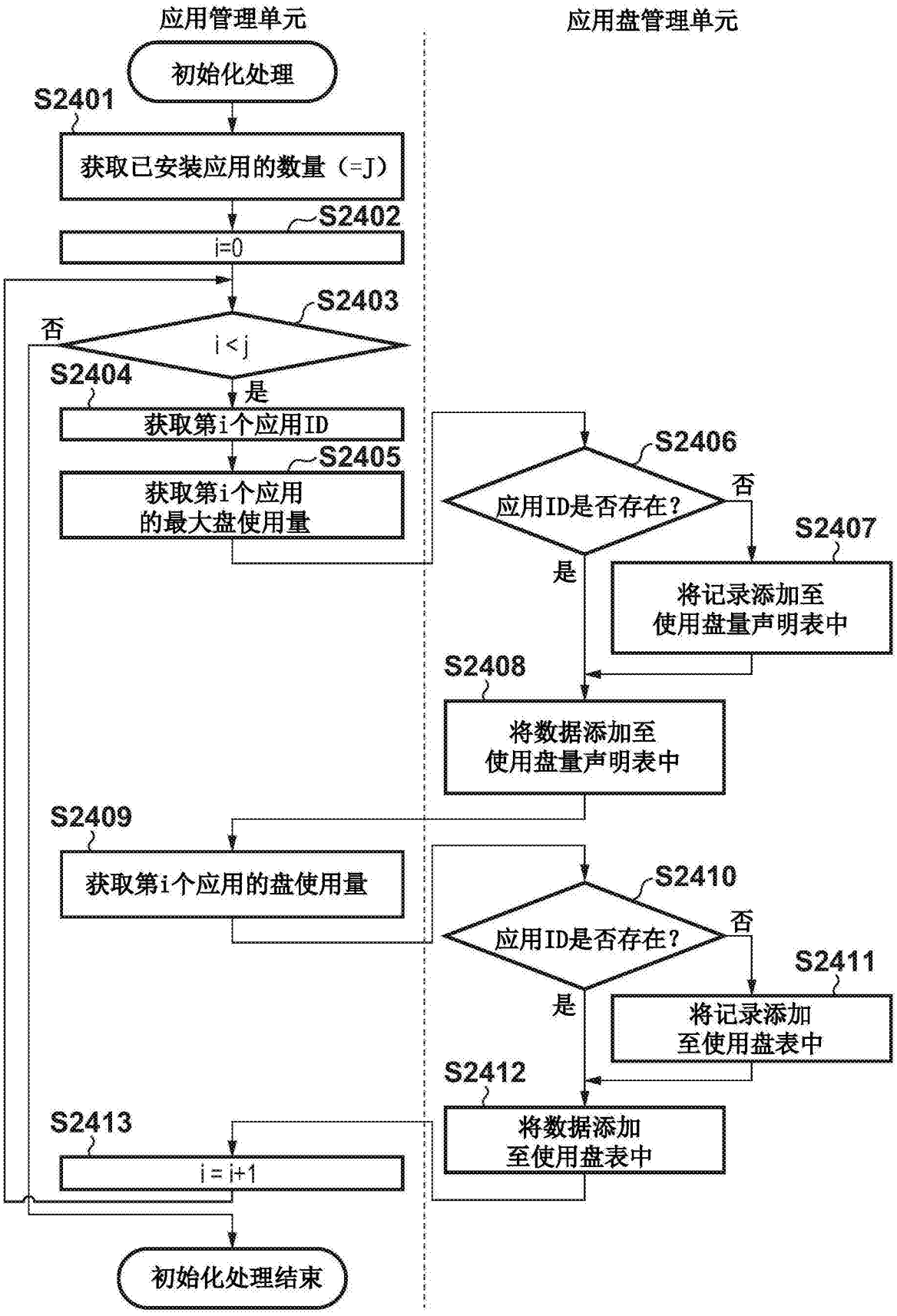

应用管理单元
应用盘管理单元
初始化处理
S2401
获取已安装应用的数量（=J）
S2402
i=0
S2403
i < j
否
是
S2404
获取第i个应用ID
S2405
获取第i个应用
的最大盘使用量
S2406
应用ID是否存在？
否
是
S2407
将记录添加至
使用盘量声明表中
S2408
将数据添加至
使用盘量声明表中
S2409
获取第i个应用的盘使用量
S2410
应用ID是否存在？
否
是
S2411
将记录添加
至使用盘表中
S2412
将数据添加
至使用盘表中
S2413
i = i+1
初始化处理结束

图23

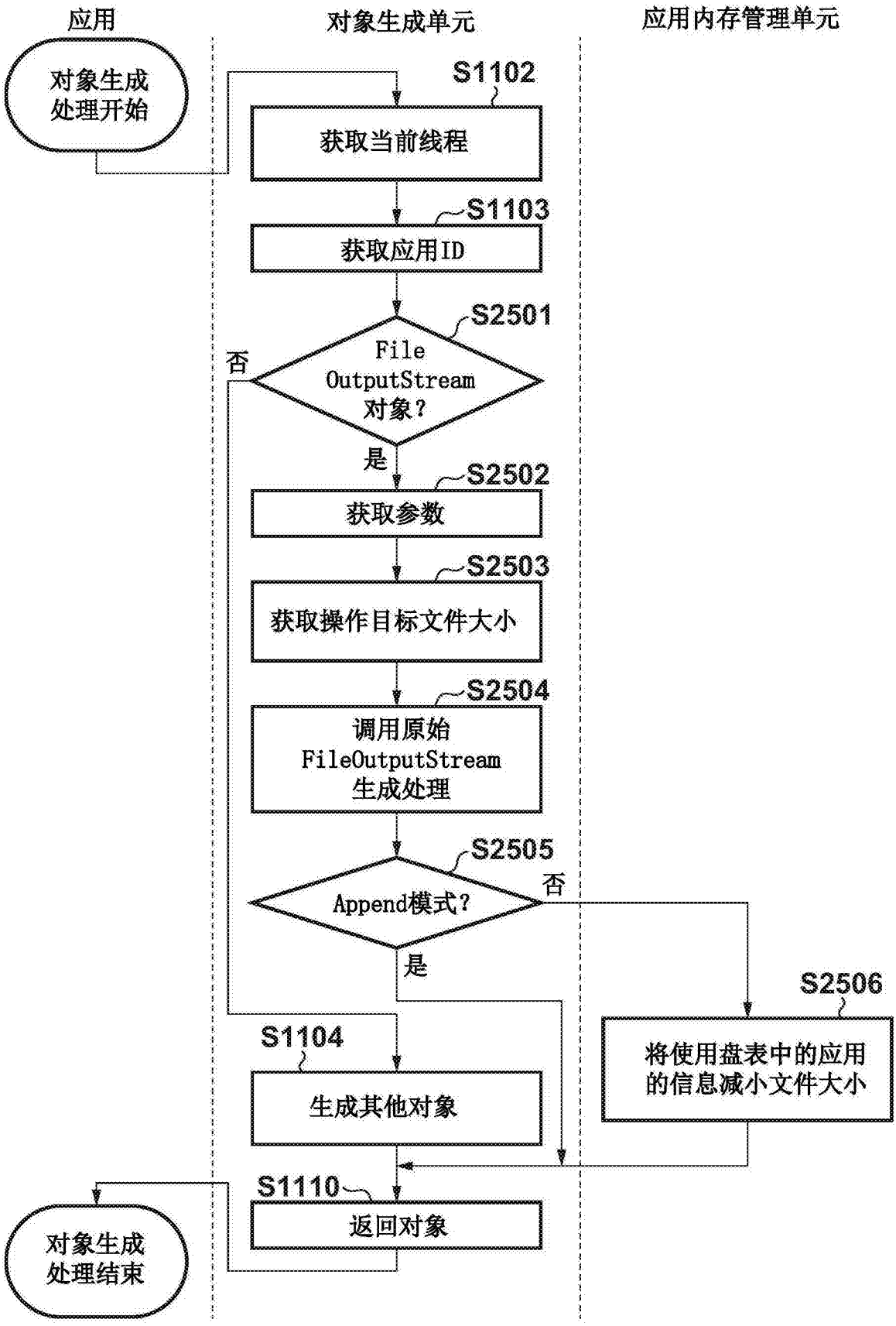

应用
对象生成单元
应用内存管理单元
对象生成处理开始
S1102
获取当前线程
S1103
获取应用ID
S2501
File OutputStream 对象?
否
是
S2502
获取参数
S2503
获取操作目标文件大小
S2504
调用原始 FileOutputStream 生成处理
S2505
Append模式?
否
是
S2506
将使用盘表中的应用的信息减小文件大小
S1104
生成其他对象
S1110
返回对象
对象生成处理结束

图24

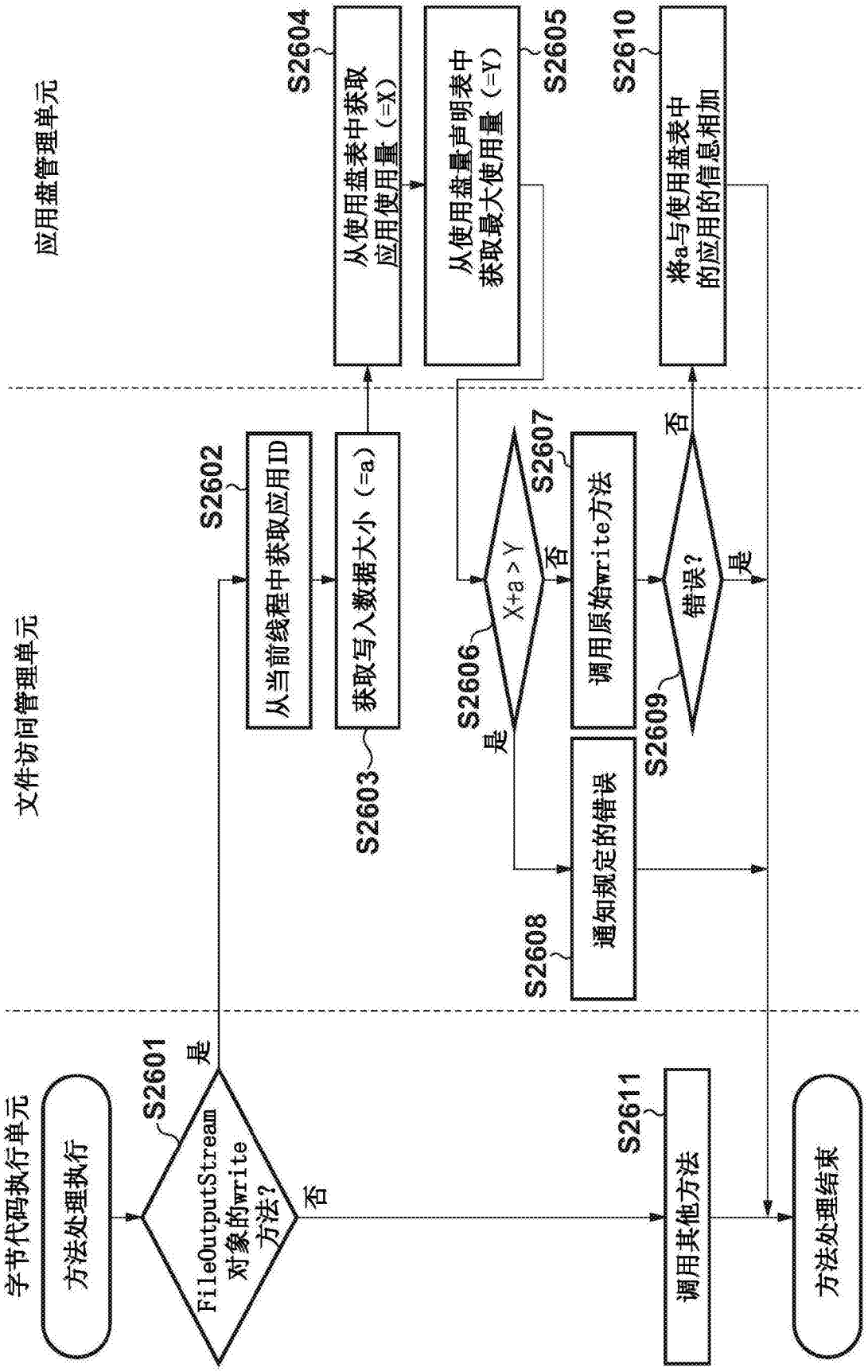

字节代码执行单元
方法处理执行
S2601
FileOutputStream对象的write方法?
是
否
文件访问管理单元
S2602
从当前线程中获取应用ID
S2603
获取写入数据大小（=a）
应用盘管理单元
S2604
从使用盘表中获取应用使用量（=X）
S2605
从使用盘量声明表中获取最大使用量（=Y）
S2606
X+a > Y
是
否
S2607
调用原始write方法
S2608
通知规定的错误
S2609
错误?
是
否
S2610
将a与使用盘表中的应用的信息相加
S2611
调用其他方法
方法处理结束

图25

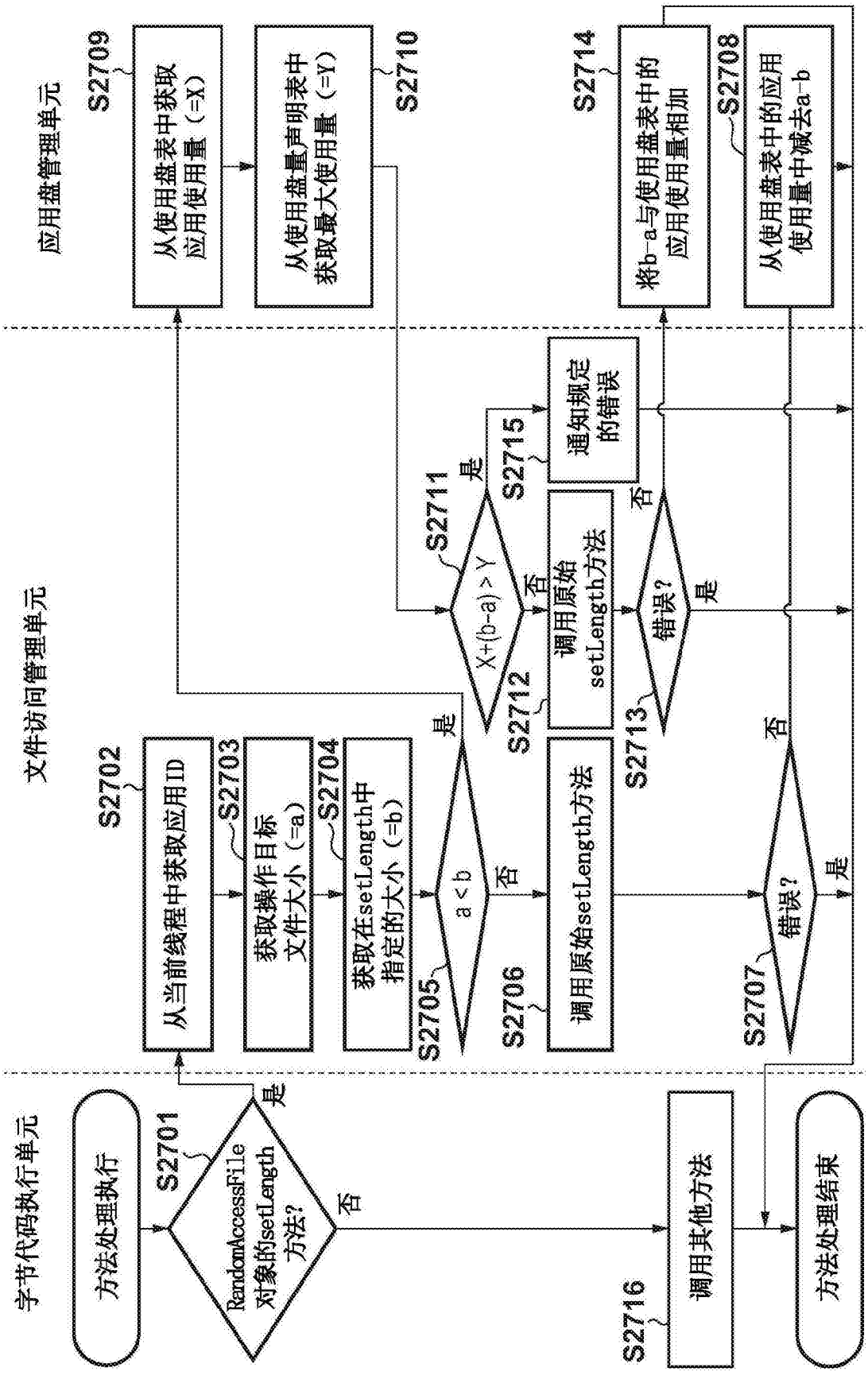

字节代码执行单元
文件访问管理单元
应用盘管理单元
方法处理执行
S2701
RandomAccessFile对象的setLength方法?
是
否
S2702
从当前线程中获取应用ID
S2703
获取操作目标文件大小（=a）
S2704
获取在setLength中指定的大小（=b）
S2705
a < b
是
否
S2706
调用原始setLength方法
S2707
错误?
否
是
S2708
从使用盘表中的应用使用量中减去a-b
S2709
从使用盘表中获取应用使用量（=X）
S2710
从使用盘量声明表中获取最大使用量（=Y）
S2711
X+(b-a) > Y
是
否
S2712
调用原始setLength方法
S2713
错误?
否
是
S2714
将b-a与使用盘表中的应用使用量相加
S2715
通知规定的错误
S2716
调用其他方法
方法处理结束

图26

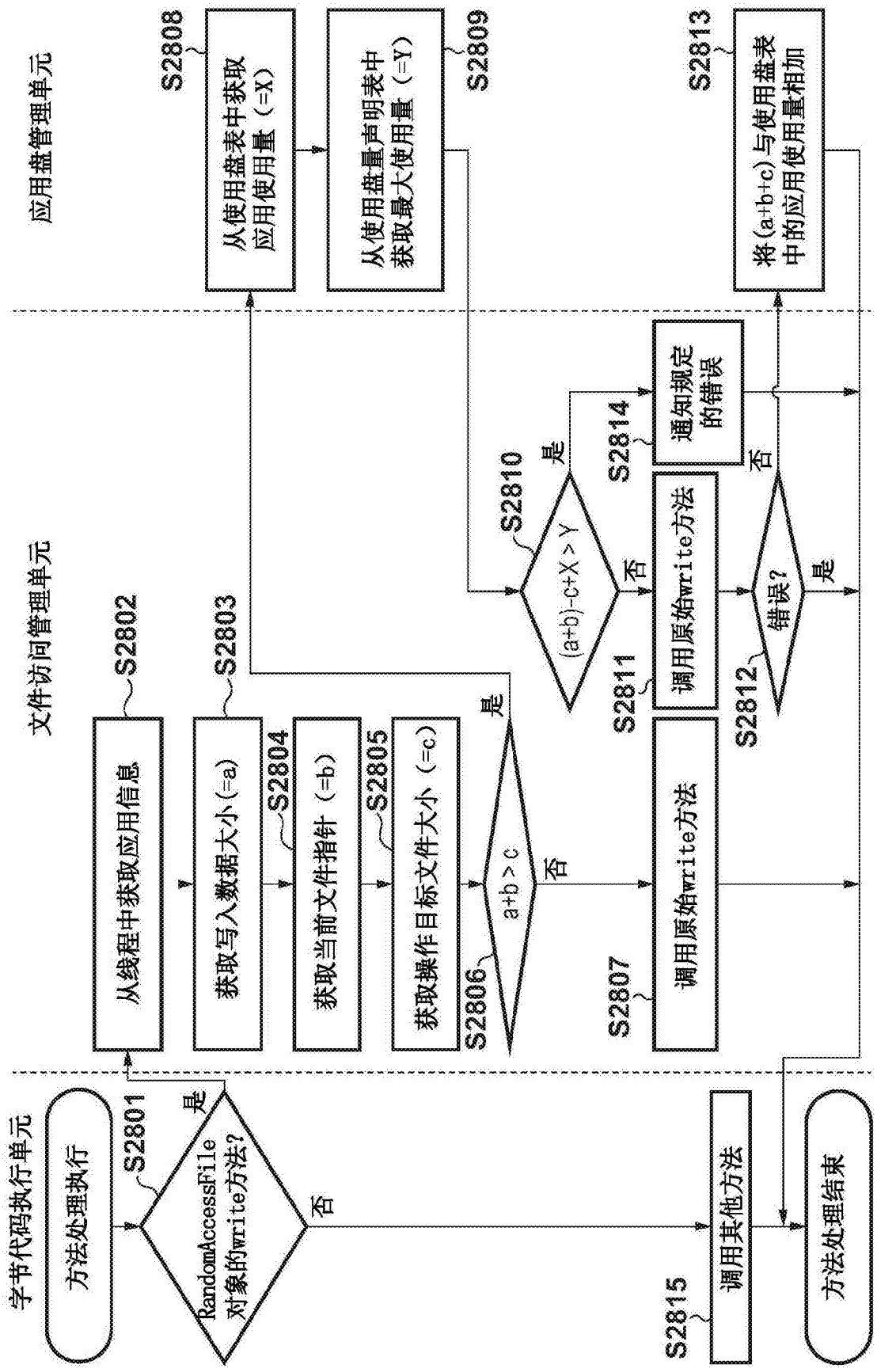

字节代码执行单元
方法处理执行
S2801
RandomAccessFile对象的write方法?
是
否
文件访问管理单元
S2802
从线程中获取应用信息
S2803
获取写入数据大小(=a)
S2804
获取当前文件指针（=b）
S2805
获取操作目标文件大小（=c）
S2806
a+b > c
是
否
S2807
调用原始write方法
S2808
从使用盘表中获取应用使用量（=X）
S2809
从使用盘量声明表中获取最大使用量（=Y）
S2810
(a+b)-c+X > Y
是
否
S2811
调用原始write方法
S2812
错误?
是
否
S2813
将(a+b+c)与使用盘表中的应用使用量相加
S2814
通知规定的错误
应用盘管理单元
S2815
调用其他方法
方法处理结束

图27

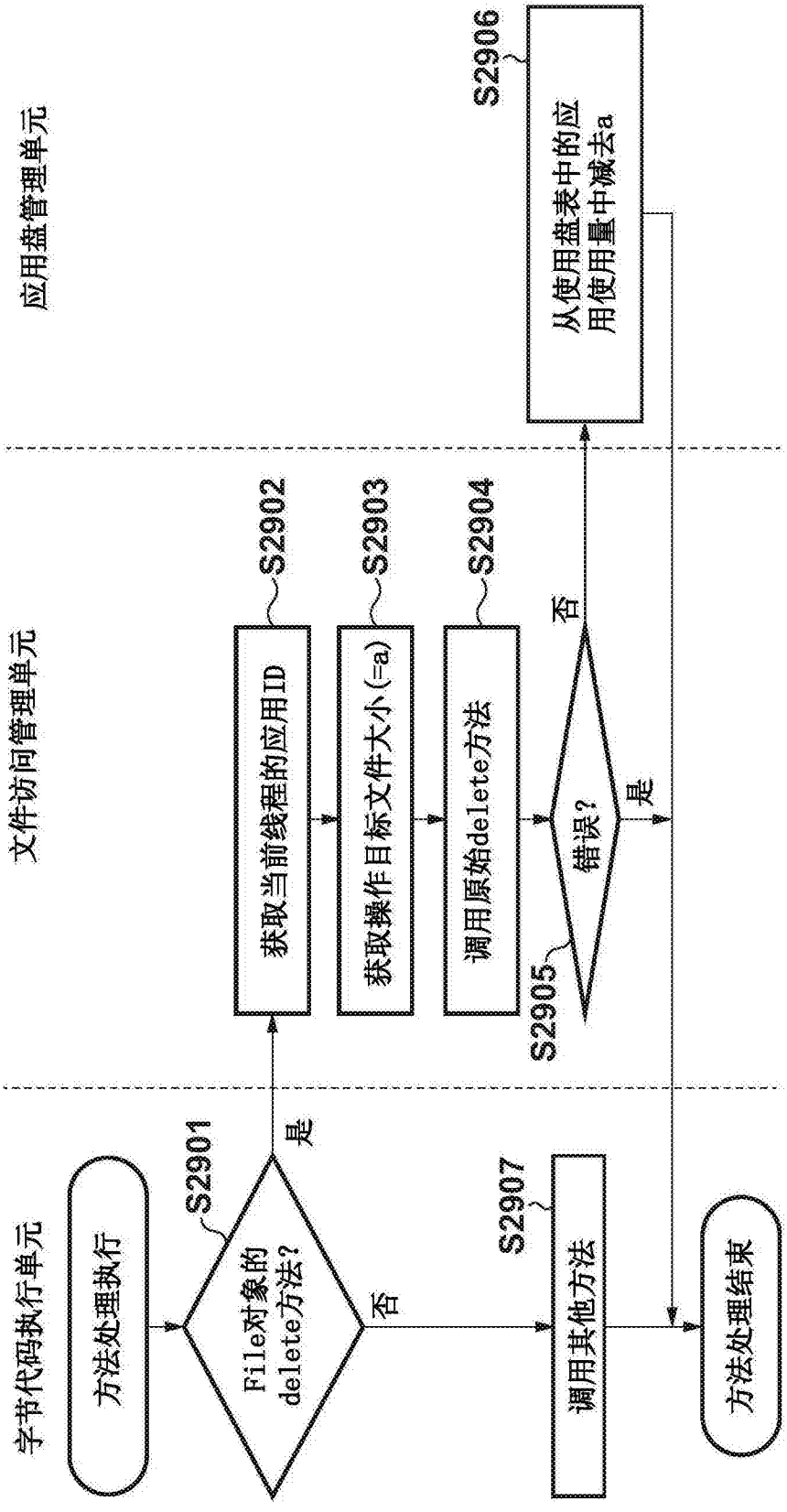

字节代码执行单元
文件访问管理单元
应用盘管理单元
方法处理执行
S2901
File对象的delete方法?
是
否
S2902
获取当前线程的应用ID
S2903
获取操作目标文件大小(=a)
S2904
调用原始delete方法
S2905
错误?
否
是
S2906
从使用盘表中的应用使用量中减去a
S2907
调用其他方法
方法处理结束

图28