(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 111459573 A
(43)申请公布日 2020.07.28

(21)申请号 202010250544.9

(22)申请日 2020.04.01

(71)申请人 济南浪潮高新科技投资发展有限公司
地址 250100 山东省济南市高新区浪潮路1036号S05楼北六楼

(72)发明人 谭强 孙善宝 徐驰 金长新

(74)专利代理机构 北京君慧知识产权代理事务所(普通合伙) 11716
代理人 董延丽

(51)Int.Cl.
G06F 9/445(2018.01)

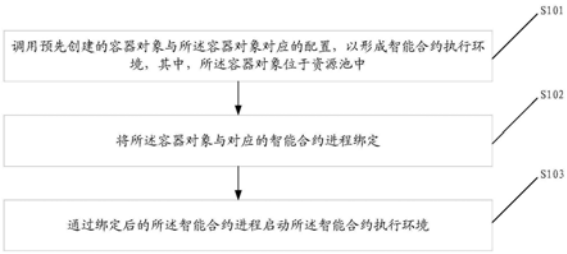
权利要求书1页 说明书8页 附图2页

(54)发明名称

一种智能合约执行环境的启动方法以及装置

(57)摘要

本申请公开了一种智能合约执行环境的启动方法，通过预先创建容器对象与容器对象对应的配置，在调用该容器对象与容器对象对应的配置后，形成智能合约执行环境，之后完成容器对象与智能合约进程的绑定，并通过绑定后的智能合约进程启动智能合约执行环境，本申请实施例在上述启动过程中，减少了创建容器对象与容器对象对应的配置的过程，可以减少智能合约执行环境的创建时间，进而可以很大程度上减少智能合约执行环境的启动时间，提升智能合约系统的整体性能。



CN 111459573 A

1.一种智能合约执行环境的启动方法,其特征在于,所述方法包括:

调用预先创建的容器对象与所述容器对象对应的配置,以形成智能合约执行环境,其中,所述容器对象位于资源池中;

将所述容器对象与对应的智能合约进程绑定;

通过绑定后的所述智能合约进程启动所述智能合约执行环境。

2.根据权利要求1所述的智能合约执行环境的启动方法,其特征在于,所述调用预先创建的容器对象与所述容器对象对应的配置前,所述方法还包括:

根据计算节点的机器资源和智能合约执行环境所需的构建文件,创建容器对象,以及所述容器对象对应的配置。

3.根据权利要求2所述的智能合约执行环境的启动方法,其特征在于,所述智能合约执行环境所需的构建文件包括基础镜像、镜像操作指令、以及所述智能合约启动时的执行命令。

4.根据权利要求1所述的智能合约执行环境的启动方法,其特征在于,所述容器对象对应的配置用于访问隔离其他的容器对象,以及对所述智能合约进程资源控制。

5.根据权利要求4所述的智能合约执行环境的启动方法,其特征在于,所述容器对象对应的配置包括命名空间与控制群组。

6.根据权利要求1所述的智能合约执行环境的启动方法,其特征在于,所述将所述容器对象与对应的智能合约进程绑定前,所述方法还包括:

将容器的对象与对应的智能合约进程加锁。

7.根据权利要求1所述的智能合约执行环境的启动方法,其特征在于,所述通过绑定后的所述智能合约进程启动所述智能合约执行环境前,所述方法还包括:

对所述智能合约进程的相关参数和/或所述智能合约执行环境的相关变量进行修改,以便于当前的智能合约执行环境与所述智能合约进程相匹配。

8.根据权利要求1所述的智能合约执行环境的启动方法,其特征在于,所述通过绑定后的所述智能合约进程启动所述智能合约执行环境后,所述方法还包括:

若所述智能合约执行环境的使用频率低于预设阈值时,将所述智能合约进程与所述容器对象解除绑定,以便于销毁该智能合约执行环境。

9.根据权利要求1所述的智能合约执行环境的启动方法,其特征在于,所述容器对象包括执行智能合约的容器与所述容器的接口,所述将所述容器对象与对应的智能合约进程绑定,具体包括:

将所述容器的接口接通对应的智能合约进程,以完成所述容器对象与所述智能合约进程的绑定。

10.一种智能合约执行环境的启动装置,其特征在于,所述装置包括:

调用单元,调用预先创建的容器对象与所述容器对象对应的配置,以形成智能合约执行环境,其中,所述容器对象位于资源池中;

绑定单元,用于将所述容器对象与对应的智能合约进程绑定;

启动单元,用于通过绑定后的所述智能合约进程启动所述智能合约执行环境。

一种智能合约执行环境的启动方法以及装置

技术领域

[0001] 本申请涉及计算机技术领域，尤其涉及一种智能合约执行环境的启动方法以及装置。

背景技术

[0002] 智能合约是一种旨在以信息化方式传播、验证或执行合同的计算机协议。智能合约允许在没有第三方的情况下进行可信交易，这些交易可追踪且不可逆转。智能合约执行环境可以在容器中进行，容器的作用是在一个沙箱中执行合约代码，并对合约所使用的资源进行隔离和限制。容器通常是指借助容器引擎，让开发者可以打包其应用，也可以实现虚拟化。

[0003] 在现有技术中，智能合约执行时存在启动时间较长，对整个智能合约系统影响较大。

发明内容

[0004] 有鉴于此，本申请实施例提供了一种智能合约执行环境的启动方法以及装置，用于解决现有技术中的智能合约在执行时存在启动时间较长的问题。

[0005] 本申请实施例采用下述技术方案：

[0006] 本申请实施例提供一种智能合约执行环境的启动方法，所述方法包括：

[0007] 调用预先创建的容器对象与所述容器对象对应的配置，以形成智能合约执行环境，其中，所述容器对象位于资源池中；

[0008] 将所述容器对象与对应的智能合约进程绑定；

[0009] 通过绑定后的所述智能合约进程启动所述智能合约执行环境。

[0010] 进一步的，所述调用预先创建的容器对象与所述容器对象对应的配置前，所述方法还包括：

[0011] 根据计算节点的机器资源和智能合约执行环境所需的构建文件，创建容器对象，以及所述容器对象对应的配置。

[0012] 进一步的，所述智能合约执行环境所需的构建文件包括基础镜像、镜像操作指令、以及所述智能合约启动时的执行命令。

[0013] 进一步的，所述容器对象对应的配置用于访问隔离其他的容器对象，以及对所述智能合约进程资源控制。

[0014] 进一步的，所述容器对象对应的配置包括命名空间与控制群组。

[0015] 进一步的，所述将所述容器对象与对应的智能合约进程绑定前，所述方法还包括：

[0016] 将容器的对象与对应的智能合约进程加锁。

[0017] 进一步的，所述通过绑定后的所述智能合约进程启动所述智能合约执行环境前，所述方法还包括：

[0018] 对所述智能合约进程的相关参数和/或所述智能合约执行环境的相关变量进行修

改,以便于当前的智能合约执行环境与所述智能合约进程相匹配。

[0019] 进一步的,所述通过绑定后的所述智能合约进程启动所述智能合约执行环境后,所述方法还包括:

[0020] 若所述智能合约执行环境的使用频率低于预设阈值时,将所述智能合约进程与所述容器对象解除绑定,以便于销毁该智能合约执行环境。

[0021] 进一步的,所述容器对象包括执行智能合约的容器与所述容器的接口,所述将所述容器对象与对应的智能合约进程绑定,具体包括:

[0022] 将所述容器的接口接通对应的智能合约进程,以完成所述容器对象与所述智能合约进程的绑定。

[0023] 本申请实施例还提供一种智能合约执行环境的启动装置,其特征在于,所述装置包括:

[0024] 调用单元,调用预先创建的容器对象与所述容器对象对应的配置,以形成智能合约执行环境,其中,所述容器对象位于资源池中;

[0025] 绑定单元,用于将所述容器对象与对应的智能合约进程绑定;

[0026] 启动单元,用于通过绑定后的所述智能合约进程启动所述智能合约执行环境。

[0027] 本申请实施例还提供一种计算机可读介质,其上存储有计算机可读指令,所述计算机可读指令可被处理器执行以实现下述方法:

[0028] 调用预先创建的容器对象与所述容器对象对应的配置,以形成智能合约执行环境,其中,所述容器对象位于资源池中;

[0029] 将所述容器对象与对应的智能合约进程绑定;

[0030] 通过绑定后的所述智能合约进程启动所述智能合约执行环境。

[0031] 本申请实施例采用的上述至少一个技术方案能够达到以下有益效果:本申请实施例通过预先创建容器对象与容器对象对应的配置,在调用该容器对象与容器对象对应的配置后,形成智能合约执行环境,之后完成容器对象与智能合约进程的绑定,并通过绑定后的智能合约进程启动智能合约执行环境,本申请实施例在上述启动过程中,减少了创建容器对象与容器对象对应的配置的过程,可以减少智能合约执行环境的创建时间,进而可以很大程度上减少智能合约执行环境的启动时间,提升智能合约系统的整体性能。

附图说明

[0032] 此处所说明的附图用来提供对本申请的进一步理解,构成本申请的一部分,本申请的示意性实施例及其说明用于解释本申请,并不构成对本申请的不当限定。在附图中:

[0033] 图1为本说明书实施例一提供的一种智能合约执行环境的启动方法的流程示意图;

[0034] 图2为本说明书实施例二提供的应用场景示意图;

[0035] 图3为本说明书实施例三提供的一种智能合约执行环境的启动装置的结构示意图。

具体实施方式

[0036] 为使本申请的目的、技术方案和优点更加清楚,下面将结合本申请具体实施例及

相应的附图对本申请技术方案进行清楚、完整地描述。显然,所描述的实施例仅是本申请一部分实施例,而不是全部的实施例。基于本申请中的实施例,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,都属于本申请保护的范围。

[0037] 智能合约是一种旨在以信息化方式传播、验证或执行合同的计算机协议。智能合约允许在没有第三方的情况下进行可信交易,这些交易可追踪且不可逆转。目前主流的智能合约执行环境的设计主要分为两种:虚拟机和容器(Docker)。无论是虚拟机还是容器,它们的作用都是在一个沙箱中执行合约代码,并对合约所使用的资源进行隔离和限制。容器通常是指借助容器引擎,让开发者可以打包其应用以及依赖包到一个可移植的容器中,也可以实现虚拟化。Docker本身没有采用虚拟化技术,程序是直接运行在底层操作系统上,代码执行的效率很高,但与轻量级虚拟机相比,其过于庞大的架构,使得部署和启动Docker本身需要消耗大量的时间和计算资源。智能合约本质上是区块链上可执行的代码,那么在智能合约的执行过程中,我们需要关注两个问题,即指令的执行速度和智能合约运行环境的启动速度。对于智能合约而言,运行环境的启动速度比指令的执行速度更加重要。这是因为,针对轻量化的虚拟机或容器,智能合约的代码中很少会涉及到IO相关的指令,所以这些指令代码易于优化。而智能合约的每次调用,都必须在一个新的虚拟机或容器中进行,因此智能合约执行环境启动时间对整个智能合约系统影响较大。

[0038] 以下结合附图,详细说明本申请各实施例提供的技术方案。

[0039] 图1为本说明书实施例一提供的一种智能合约执行环境的启动方法的流程示意图,本申请实施例的执行主体可以为智能合约系统,具体包括:

[0040] 步骤S101,调用预先创建的容器对象与所述容器对象对应的配置,以形成智能合约执行环境,其中,所述容器对象位于资源池中。

[0041] 步骤S102,将所述容器对象与对应的智能合约进程绑定。

[0042] 步骤S103,通过绑定后的所述智能合约进程启动所述智能合约执行环境。

[0043] 本申请实施例通过预先创建容器对象与容器对象对应的配置,在调用该容器对象与容器对象对应的配置后,形成智能合约执行环境,之后完成容器对象与智能合约进程的绑定,并将智能合约进程的信息传入容器后,在容器中开始执行智能合约进程,进而实现智能合约执行环境的启动过程,本申请实施例在上述启动过程中,减少了创建容器对象与容器对象对应的配置的过程,可以减少智能合约执行环境的创建时间,进而可以很大程度上减少智能合约执行环境的启动时间,提升智能合约系统的整体性能。

[0044] 与上述实施例一对应的,本说明书实施例二提供的一种输电线路检测设备的调整方法,具体包括:

[0045] 步骤S201,调用预先创建的容器对象与所述容器对象对应的配置,以形成智能合约执行环境,其中,容器对象位于资源池中。

[0046] 在本说明书实施例的步骤S201中,在智能合约系统执行调用预先创建的容器对象与所述容器对象对应的配置步骤前,本说明书实施例还需要执行步骤包括:

[0047] 根据计算节点的机器资源和智能合约执行环境所需的构建文件,创建容器对象,以及容器对象对应的配置。容器对象对应的配置可以位于资源池中,也可以位于其他位置,本申请实施例对比不做限定。

[0048] 其中,计算节点的机器资源包括计算节点的CPU信息与计算节点的内存空间大小,

计算节点可以为物理机。

[0049] 智能合约执行环境所需的构建文件包括基础镜像、镜像操作指令、软件包信息操作、以及所述智能合约启动时的执行命令，具体参见表1。

[0050] 容器对象对应的配置用于访问隔离其他的容器对象，以及对所述智能合约进程资源控制。

[0051] 进一步的，容器对象对应的配置包括命名空间与控制群组。

[0052] 命名空间(Namespace)用于访问隔离其他的容器对象，命名空间是将内核的全局资源做封装，使得每个命名空间都有一份独立的资源，因此不同的进程在各自的命名空间内对同一种资源的使用互不干扰。比如，执行sethostname这个系统调用会改变主机名，这个主机名就是全局资源，内核可以通过UTS Namespace可以将不同的进程分隔在不同的UTS Namespace中，在某个Namespace修改主机名时，另一个Namespace的主机名保持不变。

[0053] 控制群组用于对所述智能合约进程进行资源控制，原理是将一组进程放在放在一个控制组里，通过给这个控制组分配指定的可用资源，达到控制这一组进程可用资源的目的。

[0054]

FROM	指定基础镜像
MAINTRAINER	指定维护者信息
RUN	构建时需要运行的 Shell 命令
ADD	Copy 文件（自动解压）或目录到镜像中
WORKDIR	设置当前工作目录为 RUN、CMD、ENTRYPOINT 以及 COPY 和 AND
VOLUME	指定容器挂载点到宿主机自动生成的目录或其他容器
EXPOSE	声明容器运行的服务端口
CMD	启动容器时执行的 Shell 命令
ENV	设置环境内环境变量

[0055] 表1

[0056] 步骤S202，将所述容器对象与对应的智能合约进程绑定。

[0057] 在本说明书实施例的步骤S202中，每个智能合约进程可以包含有对应的标识，并预先设置智能合约执行环境需要绑定的智能合约进程，或者，预先设置容器对象需要绑定的智能合约进程。智能合约系统可以根据智能合约进程的标识，将容器对象与对应的智能合约进程。

[0058] 在本说明书实施例的步骤S202中，在智能合约系统执行将所述容器对象与对应的智能合约进程绑定的步骤前，还需要执行步骤包括：

[0059] 将容器的对象与对应的智能合约进程加锁，以便于更好的完成容器对象与智能合约进程的绑定。在智能合约系统运行时，会同时包含多个容器对象与多个进程，为了保证每个容器可以与对应的进程进行绑定，需要将容器的对象与对象的智能合约进程加锁，或者，将智能合约执行环境与智能合约进行加锁，防止出现容器对象与多个智能合约进程绑定，

或者多个容器对象与一个智能合约进程绑定。

[0060] 步骤S203,通过绑定后的所述智能合约进程启动所述智能合约执行环境。

[0061] 在本说明书实施例的步骤S203中,在智能合约系统执行通过绑定后的所述智能合约进程启动所述智能合约执行环境前,还需要执行的步骤包括:对所述智能合约进程的相关参数和/或所述智能合约执行环境的相关变量进行修改,以便于当前的智能合约执行环境与所述智能合约进程相匹配。

[0062] 需要说明的是,执行步骤S203的可以为智能合约系统的控制模块,该控制模块将各个智能合约进程的信息传入对应的容器,若该当前的智能合约执行环境与所述智能合约进程相不匹配,可以通过该控制模块修改智能合约进程的相关参数和/或所述智能合约执行环境的相关变量,其中,该控制模块的功能涵盖了命名空间的使用、控制群组的管理、根文件系统(Rootfs)的配置启动、以及进程运行的变量配置,根文件系统挂载在容器根目录上,用来为智能合约进程提供隔离后执行环境的文件系统。

[0063] 需要说明的是,本申请实施例的控制模块可以为Libcontainer。Libcontainer可以通过接口的方式定义了一系列容器管理的操作,包括处理容器的创建(Factory)、容器生命周期管理(Container)、进程生命周期管理(Process)等一系列接口。

[0064] 进一步的,在智能合约系统执行通过绑定后的所述智能合约进程启动所述智能合约执行环境步骤后,智能合约系统还需要执行的步骤包括:

[0065] 若智能合约执行环境的使用频率低于预设阈值时,将智能合约进程与容器对象解除绑定,以便于销毁该智能合约执行环境,但此处的销毁并不是删除对应的容器对象与智能合约进程,而是将对应的容器对象与智能合约进程解除绑定关系,预先创建的容器对象还是处在资源池中,可以再次应用在其他智能合约执行环境中。

[0066] 进一步的,容器对象包括执行智能合约的容器与所述容器的接口,所述将所述容器对象与对应的智能合约进程绑定,具体包括:

[0067] 将所述容器的接口接通对应的智能合约进程,以完成所述容器对象与所述智能合约进程的绑定。

[0068] 需要说明的,本申请实施例可以应用在部署多个智能合约执行环境与进程组时,每个智能合约执行环境的创建与销毁可以根据当前的应用频率,若某个智能合约执行环境使用的频率小于预设阈值时,则可以将智能合约进程与容器的接口解除绑定,为其他智能合约执行环境提供资源。

[0069] 需要说明的是,图2为本申请实施例二提供的场景示意图,在LinuxKernel中,包括多组容器对象(图中的容器)与容器对象对应的配置(图中的Namespace与Cgroups),并且,多组容器对象处于资源池中,容器对象的上层为进程组(图中的队列)。

[0070] 需要说明的是,本申请实施例将智能合约执行环境的隔离机制视为资源池中的资源,在智能合约执行环境启动时将资源池中的容器对象与智能合约进程进行动态绑定,销毁智能合约执行环境时解除与智能合约进程的绑定,并将容器对象返还资源池。此方法能有效避免智能合约执行环境高并发启动时带来的资源竞争,从而提高了智能合约执行环境启动性能。

[0071] 本申请实施例通过预先创建容器对象与容器对象对应的配置,在调用该容器对象与容器对象对应的配置后,形成智能合约执行环境,之后完成容器对象与智能合约进程的

绑定,并将智能合约进程的信息传入容器后,在容器中开始执行智能合约进程,进而实现智能合约执行环境的启动过程,本申请实施例在上述启动过程中,减少了创建容器对象与容器对象对应的配置的过程,可以减少智能合约执行环境的创建时间,进而可以很大程度上减少智能合约执行环境的启动时间,提升智能合约系统的整体性能。

[0072] 与上述实施例二对应的,图3为本说明书实施例三提供的一种智能合约执行环境的启动装置的结构示意图,包括:调用单元1、绑定单元2以及启动单元3。

[0073] 调用单元1用于调用预先创建的容器对象与所述容器对象对应的配置,以形成智能合约执行环境,其中,所述容器对象位于资源池中。

[0074] 绑定单元2用于将所述容器对象与对应的智能合约进程绑定。

[0075] 启动单元3用于通过绑定后的所述智能合约进程启动所述智能合约执行环境。

[0076] 本申请实施例通过预先创建容器对象与容器对象对应的配置,在调用该容器对象与容器对象对应的配置后,形成智能合约执行环境,之后完成容器对象与智能合约进程的绑定,并将智能合约进程的信息传入容器后,在容器中开始执行智能合约进程,进而实现智能合约执行环境的启动过程,本申请实施例在上述启动过程中,减少了创建容器对象与容器对象对应的配置的过程,可以减少智能合约执行环境的创建时间,进而可以很大程度上减少智能合约执行环境的启动时间,提升智能合约系统的整体性能。

[0077] 本申请实施例提供一种计算机可读介质,其上存储有计算机可读指令,所述计算机可读指令可被处理器执行以实现下述方法:

[0078] 调用预先创建的容器对象与所述容器对象对应的配置,以形成智能合约执行环境,其中,所述容器对象位于资源池中;

[0079] 将所述容器对象与对应的智能合约进程绑定;

[0080] 通过绑定后的所述智能合约进程启动所述智能合约执行环境。

[0081] 在20世纪90年代,对于一个技术的改进可以很明显地区分是硬件上的改进(例如,对二极管、晶体管、开关等电路结构的改进)还是软件上的改进(对于方法流程的改进)。然而,随着技术的发展,当今的很多方法流程的改进已经可以视为硬件电路结构的直接改进。设计人员几乎都通过将改进的方法流程编程到硬件电路中来得到相应的硬件电路结构。因此,不能说一个方法流程的改进就不能用硬件实体模块来实现。例如,可编程逻辑器件(Programmable Logic Device,PLD)(例如现场可编程门阵列(Field Programmable Gate Array,FPGA))就是这样一种集成电路,其逻辑功能由用户对器件编程来确定。由设计人员自行编程来把一个数字系统“集成”在一片PLD上,而不需要请芯片制造厂商来设计和制作专用的集成电路芯片。而且,如今,取代手工地制作集成电路芯片,这种编程也多半改用“逻辑编译器(logic compiler)”软件来实现,它与程序开发撰写时所用的软件编译器相类似,而要编译之前的原始代码也得用特定的编程语言来撰写,此称之为硬件描述语言(Hardware Description Language,HDL),而HDL也并非仅有一种,而是有许多种,如ABEL(Advanced Boolean Expression Language)、AHDL(Altera Hardware Description Language)、Confluence、CUPL(Cornell University Programming Language)、HDCal、JHDL(Java Hardware Description Language)、Lava、Lola、MyHDL、PALASM、RHDL(Ruby Hardware Description Language)等,目前最普遍使用的是VHDL(Very-High-Speed Integrated Circuit Hardware Description Language)与Verilog。本领域技术人员也应

该清楚，只需要将方法流程用上述几种硬件描述语言稍作逻辑编程并编程到集成电路中，就可以很容易得到实现该逻辑方法流程的硬件电路。

[0082] 控制器可以按任何适当的方式实现，例如，控制器可以采取例如微处理器或处理器以及存储可由该(微)处理器执行的计算机可读程序代码(例如软件或固件)的计算机可读介质、逻辑门、开关、专用集成电路(Application Specific Integrated Circuit，ASIC)、可编程逻辑控制器和嵌入微控制器的形式，控制器的例子包括但不限于以下微控制器：ARC 625D、Atmel AT91SAM、Microchip PIC18F26K20以及Silicone Labs C8051F320，存储器控制器还可以被实现为存储器的控制逻辑的一部分。本领域技术人员也知道，除了以纯计算机可读程序代码方式实现控制器以外，完全可以通过将方法步骤进行逻辑编程来使得控制器以逻辑门、开关、专用集成电路、可编程逻辑控制器和嵌入微控制器等的形式来实现相同功能。因此这种控制器可以被认为是一种硬件部件，而对其内包括的用于实现各种功能的装置也可以视为硬件部件内的结构。或者甚至，可以将用于实现各种功能的装置视为既可以是实现方法的软件模块又可以是硬件部件内的结构。

[0083] 上述实施例阐明的系统、装置、模块或单元，具体可以由计算机芯片或实体实现，或者由具有某种功能的产品来实现。一种典型的实现设备为计算机。具体的，计算机例如可以为个人计算机、膝上型计算机、蜂窝电话、相机电话、智能电话、个人数字助理、媒体播放器、导航设备、电子邮件设备、游戏控制台、平板计算机、可穿戴设备或者这些设备中的任何设备的组合。

[0084] 为了描述的方便，描述以上装置时以功能分为各种单元分别描述。当然，在实施本申请时可以把各单元的功能在同一个或多个软件和/或硬件中实现。

[0085] 本领域内的技术人员应明白，本发明的实施例可提供为方法、系统、或计算机程序产品。因此，本发明可采用完全硬件实施例、完全软件实施例、或结合软件和硬件方面的实施例的形式。而且，本发明可采用在一个或多个其中包含有计算机可用程序代码的计算机可用存储介质(包括但不限于磁盘存储器、CD-ROM、光学存储器等)上实施的计算机程序产品的形式。

[0086] 本发明是参照根据本发明实施例的方法、设备(系统)、和计算机程序产品的流程图和/或方框图来描述的。应理解可由计算机程序指令实现流程图和/或方框图中的每一流程和/或方框、以及流程图和/或方框图中的流程和/或方框的结合。可提供这些计算机程序指令到通用计算机、专用计算机、嵌入式处理机或其他可编程数据处理设备的处理器以产生一个机器，使得通过计算机或其他可编程数据处理设备的处理器执行的指令产生用于实现在流程图一个流程或多个流程和/或方框图一个方框或多个方框中指定的功能的装置。

[0087] 这些计算机程序指令也可存储在能引导计算机或其他可编程数据处理设备以特定方式工作的计算机可读存储器中，使得存储在该计算机可读存储器中的指令产生包括指令装置的制造品，该指令装置实现在流程图一个流程或多个流程和/或方框图一个方框或多个方框中指定的功能。

[0088] 这些计算机程序指令也可装载到计算机或其他可编程数据处理设备上，使得在计算机或其他可编程设备上执行一系列操作步骤以产生计算机实现的处理，从而在计算机或其他可编程设备上执行的指令提供用于实现在流程图一个流程或多个流程和/或方框图一个方框或多个方框中指定的功能的步骤。

[0089] 在一个典型的配置中，计算设备包括一个或多个处理器(CPU)、输入/输出接口、网络接口和内存。

[0090] 内存可能包括计算机可读介质中的非永久性存储器，随机存取存储器(RAM)和/或非易失性内存等形式，如只读存储器(ROM)或闪存(flash RAM)。内存是计算机可读介质的示例。

[0091] 计算机可读介质包括永久性和非永久性、可移动和非可移动媒体可以由任何方法或技术来实现信息存储。信息可以是计算机可读指令、数据结构、程序的模块或其他数据。计算机的存储介质的例子包括，但不限于相变内存(PRAM)、静态随机存取存储器(SRAM)、动态随机存取存储器(DRAM)、其他类型的随机存取存储器(RAM)、只读存储器(ROM)、电可擦除可编程只读存储器(EEPROM)、快闪记忆体或其他内存技术、只读光盘只读存储器(CD-ROM)、数字多功能光盘(DVD)或其他光学存储、磁盒式磁带，磁带式磁盘存储或其他磁性存储设备或任何其他非传输介质，可用于存储可以被计算设备访问的信息。按照本文中的界定，计算机可读介质不包括暂存电脑可读媒体(transitory media)，如调制的数据信号和载波。

[0092] 还需要说明的是，术语“包括”、“包含”或者其任何其他变体意在涵盖非排他性的包含，从而使得包括一系列要素的过程、方法、商品或者设备不仅包括那些要素，而且还包括没有明确列出的其他要素，或者是还包括为这种过程、方法、商品或者设备所固有的要素。在没有更多限制的情况下，由语句“包括一个……”限定的要素，并不排除在包括所述要素的过程、方法、商品或者设备中还存在另外的相同要素。

[0093] 本申请可以在由计算机执行的计算机可执行指令的一般上下文中描述，例如程序模块。一般地，程序模块包括执行特定任务或实现特定抽象数据类型的例程、程序、容器对象、组件、数据结构等等。也可以在分布式计算环境中实践本申请，在这些分布式计算环境中，由通过通信网络而被连接的远程处理设备来执行任务。在分布式计算环境中，程序模块可以位于包括存储设备在内的本地和远程计算机存储介质中。

[0094] 本说明书中的各个实施例均采用递进的方式描述，各个实施例之间相同相似的部分互相参见即可，每个实施例重点说明的都是与其他实施例的不同之处。尤其，对于系统实施例而言，由于其基本相似于方法实施例，所以描述的比较简单，相关之处参见方法实施例的部分说明即可。

[0095] 以上所述仅为本申请的实施例而已，并不用于限制本申请。对于本领域技术人员来说，本申请可以有各种更改和变化。凡在本申请的精神和原理之内所作的任何修改、等同替换、改进等，均应包含在本申请的权利要求范围之内。

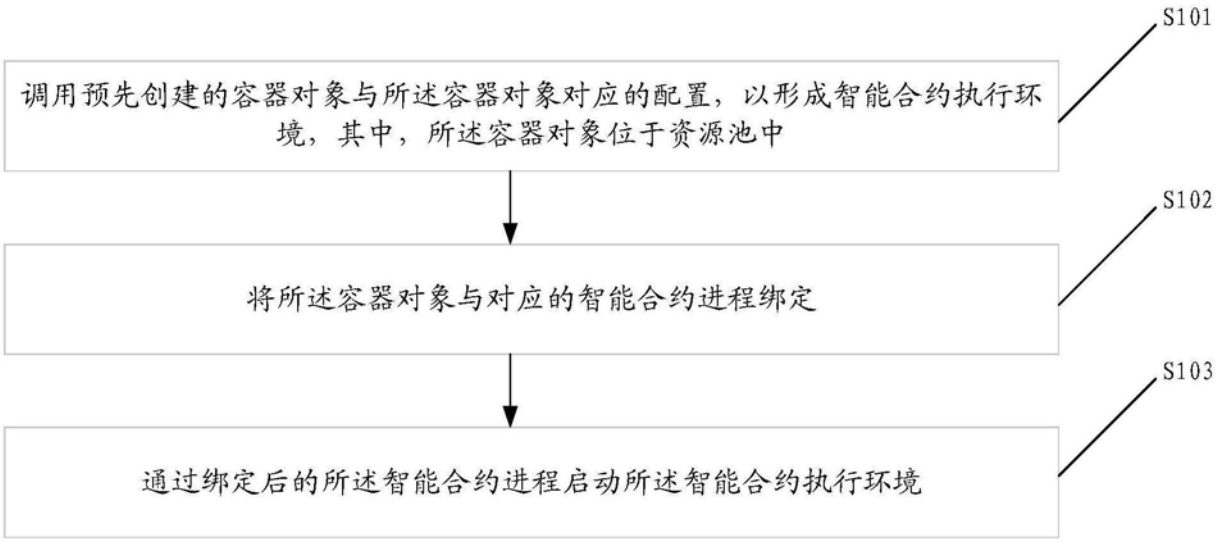

S101
调用预先创建的容器对象与所述容器对象对应的配置，以形成智能合约执行环境，其中，所述容器对象位于资源池中
S102
将所述容器对象与对应的智能合约进程绑定
S103
通过绑定后的所述智能合约进程启动所述智能合约执行环境

图1

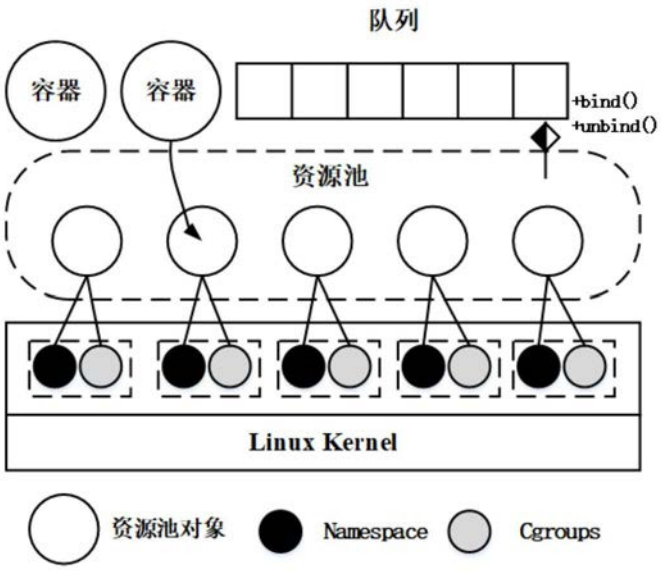

队列
容器
容器
+bind()
+unbind()
资源池
Linux Kernel
资源池对象
Namespace
Cgroups

图2

图3