(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 104376256 B
(45)授权公告日 2017.04.05

(21)申请号 201410724739.7

(22)申请日 2014.12.02

(65)同一申请的已公布的文献号
申请公布号 CN 104376256 A

(43)申请公布日 2015.02.25

(73)专利权人 北京奇虎科技有限公司
地址 100088 北京市西城区新街口外大街28号D座112室(德胜园区)
专利权人 奇智软件(北京)有限公司

(72)发明人 曹阳 杨威

(74)专利代理机构 北京市立方律师事务所 11330
代理人 王增鑫

(51)Int.Cl.
G06F 21/53(2013.01)

(56)对比文件
CN 103514007 A,2014.01.15,
CN 104050001 A,2014.09.17,
李彬.基于Android沙箱的软件行为分析系统的设计与实现.《中国优秀硕士学位论文全文数据库》.2013,全文.

审查员 刘义乐

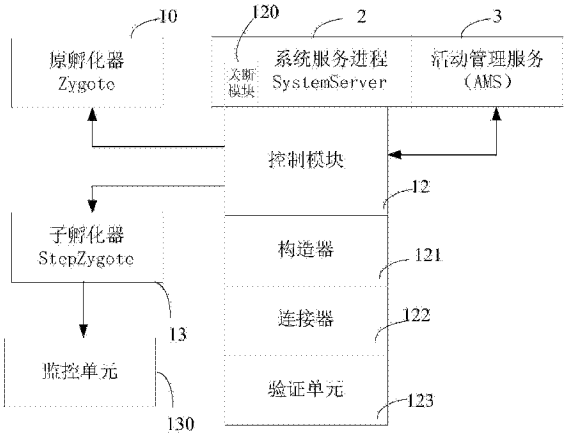
权利要求书3页 说明书25页 附图5页

(54)发明名称

应用程序进程孵化控制方法及装置

(57)摘要

本发明涉及一种应用程序进程孵化控制方法,包括以下步骤:运行控制模块以向系统服务进程注入用于关闭活动管理服务与原孵化器的连接的关断模块;由所述控制模块利用原孵化器构造子孵化器;由所述控制模块接收在系统服务进程注册的活动管理服务的为运行应用程序而发起的请求,并将其传递给所述子孵化器;由所述子孵化器响应于该请求而孵化新进程以运行所述应用程序。本发明还提供了一种用于执行该方法的相应的装置。本发明使得目标应用程序能够运行于沙箱运行环境中,能确保系统的安全。



CN 104376256 B

1.一种应用程序进程孵化控制方法,其特征在于,包括以下步骤:

运行控制模块以向系统服务进程注入用于关闭活动管理服务与原孵化器的连接的关断模块;

由所述控制模块利用原孵化器构造子孵化器;

由所述控制模块接收在系统服务进程注册的活动管理服务的为运行应用程序而发起的请求,并从该请求中提取应用程序特征信息,基于该特征信息从本地或远程已知设置数据中检验该特征信息所对应的应用程序是否应被限制或被禁止,当其为应被限制或应被禁止的应用程序时,将其传递给所述子孵化器;

由所述子孵化器响应于该请求而孵化新进程以运行所述应用程序。

2.根据权利要求1所述的应用程序进程孵化控制方法,其特征在于,所述控制模块的执行过程中,获取活动管理服务的请求的先导过程,包括如下步骤:

利用原孵化器的套接口建立控制模块与原孵化器的连接;

利用所述关断模块关闭活动管理服务基于所述原孵化器套接口所维持的连接;

利用所述原孵化器的套接口监听所述活动管理服务的为运行应用程序而发起的请求。

3.根据权利要求2所述的应用程序进程孵化控制方法,其特征在于,所述控制模块利用原孵化器构造子孵化器的步骤,在控制模块与原孵化器建立连接之后执行。

4.根据权利要求2所述的应用程序进程孵化控制方法,其特征在于,所述运行控制模块以向系统服务进程注入关断模块的步骤,在所述利用原孵化器的套接口建立控制模块与原孵化器的连接的步骤之后,且即执行该关断模块以关闭活动管理服务基于所述原孵化器套接口所维持的连接。

5.根据权利要求2所述的应用程序进程孵化控制方法,其特征在于,所述子孵化器建立有相应的套接口,所述控制模块通过子孵化器套接口建立与该子孵化器的连接,以向该子孵化器传递所述的请求。

6.根据权利要求5所述的应用程序进程孵化控制方法,其特征在于,所述控制模块依据预设置数据确定将所述请求传递给子孵化器或原孵化器。

7.根据权利要求6所述的应用程序进程孵化控制方法,其特征在于,所述预设置数据,经由用户界面基于待运行应用程序接收用户对所述原孵化器和子孵化器的选定结果而生成。

8.根据权利要求5所述的应用程序进程孵化控制方法,其特征在于,所述子孵化器从原孵化器孵化之后,即构建出其所述的套接口,该套接口对应的数据存储于相应的文件中。

9.根据权利要求8所述的应用程序进程孵化控制方法,其特征在于,所述子孵化器的套接口文件存储于本地目录中。

10.根据权利要求9所述的应用程序进程孵化控制方法,其特征在于,所述子孵化器的套接口文件存储于系统目录/dev/socket中。

11.根据权利要求9所述的应用程序进程孵化控制方法,其特征在于,所述子孵化器的套接口文件的名称与该子孵化器的进程名称相同。

12.根据权利要求1所述的应用程序进程孵化控制方法,其特征在于,所述控制模块利用原孵化器构造子孵化器的过程包括如下步骤:

复制原孵化器的可执行代码;

向该可执行代码中插入用于实现外部调用的调用指令；

运行修改后的可执行代码以实现所述子孵化器的构造。

13.根据权利要求12所述的应用程序进程孵化控制方法，其特征在于，所述实现外部调用的调用指令，用于调用外部监控单元，以实现对当前子孵化器所构造的进程空间所发生的事件行为的监控。

14.根据权利要求12所述的应用程序进程孵化控制方法，其特征在于，所述控制模块利用原孵化器构造子孵化器的过程中，还包括向该可执行代码中插入用于实现子孵化器自校验的代码。

15.根据权利要求1至14中任意一项所述的应用程序进程孵化控制方法，其特征在于，该方法还包括如下步骤：

由控制模块以与构造子孵化器相同的方法构造新的子孵化器；

针对待运行应用程序，由控制模块依据预设置数据将活动管理服务的请求传递给相关的原孵化器和多个子孵化器中的一个，以为该应用程序的运行选定相应的孵化器。

16.根据权利要求1至14中任意一项所述的应用程序进程孵化控制方法，其特征在于，所述子孵化器利用fork函数孵化出用于运行所述应用程序的新进程。

17.根据权利要求1至14中任意一项所述的应用程序进程孵化控制方法，其特征在于，向系统服务进程注入的所述关断模块，其用于实现其至少部分功能的函数包含于共享库文件中。

18.根据权利要求1至14中任意一项所述的应用程序进程孵化控制方法，其特征在于，所述系统服务进程为SystemServer进程，所述原孵化器为Zygote进程，所述活动管理服务进程为ActivityManagerService进程。

19.根据权利要求1至14中任意一项所述的应用程序进程孵化控制方法，其特征在于，所述控制模块依据已知设置数据确定是否满足所述活动管理服务所发起的请求。

20.一种应用程序进程孵化控制装置，其特征在于，包括：

控制模块，被配置为：

用于向系统服务进程注入用于关闭活动管理服务与原孵化器的连接的关断模块；

用于利用原孵化器构造子孵化器；

用于接收在系统服务进程注册的活动管理服务的为运行应用程序而发起的请求，并从该请求中提取应用程序特征信息，基于该特征信息从本地或远程已知设置数据中检验该特征信息所对应的应用程序是否应被限制或被禁止，当其为应被限制或应被禁止的应用程序时，将其传递给所述子孵化器；

所述的子孵化器，用于响应于该请求而孵化新进程以运行所述应用程序。

21.根据权利要求20所述的应用程序进程孵化控制装置，其特征在于，

所述控制模块包括连接器，其被配置为执行如下功能：

利用原孵化器的套接口建立控制模块与原孵化器的连接；

利用所述关断模块关闭活动管理服务基于所述原孵化器套接口所维持的连接，

所述控制模块利用所述原孵化器的套接口监听所述活动管理服务的为运行应用程序而发起的请求。

22.根据权利要求21所述的应用程序进程孵化控制装置，其特征在于，所述连接器还被

配置为负责向系统服务进程注入用于关闭活动管理服务与原孵化器的连接的关断模块。

23.根据权利要求21所述的应用程序进程孵化控制装置，其特征在于，所述连接器还被配置为执行如下功能：通过子孵化器所具有的套接口建立所述控制模块与该子孵化器的连接，以向该子孵化器传递所述的请求。

24.根据权利要求23所述的应用程序进程孵化控制装置，其特征在于，所述控制模块依据预设置数据确定将所述请求传递给子孵化器或原孵化器。

25.根据权利要求24所述的应用程序进程孵化控制装置，其特征在于，所述预设置数据，经由用户界面基于待运行应用程序接收用户对所述原孵化器和子孵化器的选定结果而生成。

26.根据权利要求23所述的应用程序进程孵化控制装置，其特征在于，所述子孵化器的套接口对应的数据存储于相应的文件中。

27.根据权利要求26所述的应用程序进程孵化控制装置，其特征在于，所述子孵化器的套接口文件存储于本地目录中。

28.根据权利要求27所述的应用程序进程孵化控制装置，其特征在于，所述子孵化器的套接口文件存储于系统目录/dev/socket中。

29.根据权利要求27所述的应用程序进程孵化控制装置，其特征在于，所述子孵化器的套接口文件的名称与该子孵化器的进程名称相同。

30.根据权利要求29所述的应用程序进程孵化控制装置，其特征在于，所述控制模块包括构造器，用于利用原孵化器构造子孵化器，该构造器被配置为执行如下功能：

复制原孵化器的可执行代码；

向该可执行代码中插入用于实现外部调用的调用指令；

运行修改后的可执行代码以实现所述子孵化器的构造。

31.根据权利要求30所述的应用程序进程孵化控制装置，其特征在于，所述实现外部调用的调用指令，用于调用外部监控单元，以实现对当前子孵化器所构造的进程空间所发生的事件行为的监控。

32.根据权利要求30所述的应用程序进程孵化控制装置，其特征在于，所述构造器还被配置为用于执行如下功能：向该可执行代码中插入用于实现子孵化器自校验的代码。

33.根据权利要求20至32中任意一项所述的应用程序进程孵化控制装置，其特征在于，所述关断模块配置有共享库文件，该共享库文件包含有用于实现所述关断模块的至少部分功能的函数。

34.根据权利要求20至32中任意一项所述的应用程序进程孵化控制装置，其特征在于，所述系统服务进程为SystemServer进程，所述原孵化器为Zygote进程，所述活动管理服务进程为ActivityManagerService进程。

应用程序进程孵化控制方法及装置

技术领域

[0001] 本发明涉及计算机软件安全技术领域，尤其涉及一种应用程序进程孵化控制方法及相应的装置。

背景技术

[0002] 沙箱是一种按照安全策略限制程序行为的执行环境，目前已经广泛实用于各种操作系统中。以Android为例，一些应用程序，出于实现应用程序固有功能需要之外的目的，特别是商业目的，随意申请系统权限，获取用户隐私数据、执行网络访问、保持设备活动、发送短信行为等。轻则可能导致用户隐私数据泄露，或者占用系统资源，重则可能通过恶意扣费、植入广告、消耗资费、欺诈诱骗等，使用户遭受损失。因此，通过沙箱技术提供的执行环境，由沙箱对系统的资源、权限进行管理，让应用程序于该沙箱中运行，应用程序的访问先经沙箱按安全策略进行审查，由此，形成一种相对于系统本身的隔离运行效果，可以有效地保护系统的安全。对于沙箱中所用到的安全策略，适应各种不同的操作系统有不同的细节考虑，这些有关技术实现的基本知识，均已为本领域技术人员所掌握，恕不赘述。

[0003] 目前有多种实例来实现沙箱技术。这些实例中，一方面，沙箱技术为了兼容市面的多种应用，一般仅仅通过限定沙箱的安全策略，控制该应用的可执行资源而实现。然而，在安全领域，攻防双方的技术水平此消彼长，传统的仅仅通过限制安全策略的沙箱，有时难以确保能够达到所期望的目的，必须借助于更富技术含量的新方案。另一方面，沙箱技术往往涉及系统底层操作，而在诸如以Android为代表的Unix系的操作系统中，本身有着严格的权限管理，这样，便导致在未获得Root授权的前提下，难以应用沙箱技术去构造沙箱。可以独辟蹊径，去实现免Root环境下的沙箱环境，然而，在这种情况下，往往会引起多方面的一些技术障碍，其防护效果依然有限，这些障碍依沙箱的具体实现方式而定。

[0004] 导致这些沙箱实例难以取得更佳的效果的原因是由操作系统的既定的逻辑决定的。具体而言，恶意应用程序的开发人员知晓操作系统的实现原理及各种函数接口，对于Android而言，甚至熟知其开放的源代码。在此基础上，开发人员得以依据这些原理，针对系统所提供的各种功能模块、函数接口等进行非法利用，绕开系统的安全原旨，达到自身的非法目的。

[0005] Android自身出于安全的考虑，已经利用虚拟机原理加以实现，以最大程度降低可能的侵入可能。虚拟机用于进一步运行应用程序进程。虚拟机的启动源于系统的Zygote(业内称之为孵化器)模块，Zygote由Linux底层实现的init函数加载。Zygote被加载后，便通过自身的孵化函数fork()来复制自身，新进程被命名为SystemServer，SystemServer便是Zygote孵化的第一个成功运行的进程，为理解的便利本发明称其为长子进程。继而，由SystemServer进程去实现系统服务的一系列初始化功能，包括对Native层的服务进行初始化、对Java层的服务进行初始化，最终进入Binder通信系统监听请求，给应用层和系统提供各种服务请求。在这个过程中，ActivityManagerService(AMS)和PackageManagerService(PMS)在内的一系列的Java层的服务被陆续加载，而Zygote则退居后台继续监听是否有新

的孵化请求。一旦AMS为运行应用程序而向Zygote发起孵化请求时，Zygote便会继续孵化自身，然后通过新的Zygote进程加载虚拟机，使该应用程序运行于该虚拟机中。

[0006] Android希望利用这一机制来实现更为安全的进程保护效果，一方面希望确保单个的虚拟机的崩溃不会影响到其它虚拟机的正常运行，另一方面，希望每个应用程序进程都能被以虚拟机为单位进行管理。从这个角度来看，虚拟机便天然具有沙箱的特质，只不过这种特质对程序开发人员而言的公开透明的。于是，现实中，不少恶意应用正是利用了Android进程加载原理的这些特质，在获得系统Root权限的前提下，利用各种公知的病毒手段或黑客手段，深入到Android的底层，包括Zygote、SystemServer均可能被非法利用，从而达到非法目的。

[0007] 诚然，Android的权限管理较为严格，在未获得Root的情况下，恶意程序的很多常规侵犯手段可以被一般的安全应用软件拒绝。然而，随着Android越来越开放，以及用户出于自定义预安装应用的需要，越来越多的移动终端设备被永久Root，于是，这些移动终端的安全问题就越来越突出，完善已Root设备的安全防护机制，实现更为具体有效的沙箱实例，是业内悬而未决的问题。

[0008] 以上对现有技术的剖析可以大致概括为两方面，或者说是两个层次的问题，一是操作系统如何在已Root环境下避免受底层攻击从而实现其安全目的；二是操作系统如何在已Root环境下实现更为安全的沙箱实例。这两方面虽相辅相成，也有其相对独立性。

发明内容

[0009] 本发明的第一目的是为了克服前述的至少部分问题而提供一种应用程序进程孵化控制方法。

[0010] 本发明的第二目在于提供一种适于构造第一目的所述的方法的应用程序进程孵化控制装置。

[0011] 为实现本发明的目的，本发明采取如下技术方案：

[0012] 本发明提供的一种应用程序进程孵化控制方法，包括以下步骤：

[0013] 运行控制模块以向系统服务进程注入用于关闭活动管理服务与原孵化器的连接的关断模块；

[0014] 由所述控制模块利用原孵化器构造子孵化器；

[0015] 由所述控制模块接收在系统服务进程注册的活动管理服务的为运行应用程序而发起的请求，并将其传递给所述子孵化器；

[0016] 由所述子孵化器响应于该请求而孵化新进程以运行所述应用程序。

[0017] 进一步，所述控制模块的执行过程中，获取活动管理服务的请求的先导过程，包括如下步骤：

[0018] 利用原孵化器的套接口建立控制模块与原孵化器的连接；

[0019] 利用所述关断模块关闭活动管理服务基于所述原孵化器套接口所维持的连接；

[0020] 利用所述原孵化器的套接口监听所述活动管理服务的为运行应用程序而发起的请求。

[0021] 较佳的，所述控制模块利用原孵化器构造子孵化器的步骤，在控制模块与原孵化器建立连接之后执行。

[0022] 根据本发明的一个变例，所述运行控制模块以向系统服务进程注入关断模块的步骤，在所述利用原孵化器的套接口建立控制模块与原孵化器的连接的步骤之后，且即执行该关断模块以关闭活动管理服务基于所述原孵化器套接口所维持的连接。

[0023] 进一步，所述子孵化器建立有相应的套接口，所述控制模块通过子孵化器套接口建立与该子孵化器的连接，以向该子孵化器传递所述的请求。

[0024] 进一步，所述控制模块依据预设置数据确定将所述请求传递给子孵化器或原孵化器。

[0025] 较佳的，所述预设置数据，经由用户界面基于待运行应用程序接收用户对所述原孵化器和子孵化器的选定结果而生成。

[0026] 进一步，所述子孵化器从原孵化器孵化之后，即构建出其所述的套接口，该套接口对应的数据存储于相应的文件中。所述子孵化器的套接口文件存储于本地目录中。较佳的，所述子孵化器的套接口文件存储于系统目录/dev/socket中。所述子孵化器的套接口文件的名称与该子孵化器的进程名称相同。

[0027] 进一步，控制模块利用原孵化器构造子孵化器的过程包括如下步骤：

[0028] 复制原孵化器的可执行代码；

[0029] 向该可执行代码中插入用于实现外部调用的调用指令；

[0030] 运行修改后的可执行代码以实现所述子孵化器的构造。

[0031] 较佳的，所述实现外部调用的调用指令，用于调用外部监控单元，以实现对当前子孵化器所构造的进程空间所发生的事件行为的监控。

[0032] 此外，所述控制模块利用原孵化器构造子孵化器的过程中，还包括向该可执行代码中插入用于实现子孵化器自校验的代码。

[0033] 进一步，本发明的所述方法还包括如下步骤：

[0034] 由控制模块以与构造子孵化器相同的方法构造新的子孵化器；

[0035] 针对待运行应用程序，由控制模块依据预设置数据将活动管理服务的请求传递给相关的原孵化器和多个子孵化器中的一个，以为该应用程序的运行选定相应的孵化器。

[0036] 具体的，所述子孵化器利用fork函数孵化出用于运行所述应用程序的新进程。

[0037] 较佳的，向系统服务进程注入的所述关断模块，其用于实现其至少部分功能的函数包含于共享库文件中。

[0038] 较佳的，所述系统服务进程为SystemServer进程，所述原孵化器为Zygote进程，所述活动管理服务进程为ActivityManagerService进程。

[0039] 较佳的，所述控制模块依据已知设置数据确定是否满足所述活动管理服务所发起的请求。具体而言，所述控制模块在接收所述的请求之后，从该请求中提取应用程序特征信息，基于该特征信息从本地或远程已知设置数据中检验该特征信息所对应的应用程序是否应被限制或被禁止，当其为应被限制或应被禁止的应用程序时，控制模块将该请求中转至所述原孵化器之外的其余子孵化器或不中转该请求至任何孵化器。

[0040] 本发明提供的一种应用程序进程孵化控制装置，包括：

[0041] 所述的控制模块，被配置为：

[0042] 用于向系统服务进程注入用于关闭活动管理服务与原孵化器的连接的关断模块；

[0043] 用于利用原孵化器构造子孵化器；

[0044] 用于接收在系统服务进程注册的活动管理服务的为运行应用程序而发起的请求，并将其传递给所述子孵化器；

[0045] 所述的子孵化器，用于响应于该请求而孵化新进程以运行所述应用程序。

[0046] 具体的，所述控制模块包括连接器，其被配置为执行如下功能：

[0047] 利用原孵化器的套接口建立控制模块与原孵化器的连接；

[0048] 利用所述关断模块关闭活动管理服务基于所述原孵化器套接口所维持的连接，

[0049] 所述控制模块利用所述原孵化器的套接口监听所述活动管理服务的为运行应用程序而发起的请求。

[0050] 所述连接器还被配置为负责向系统服务进程注入用于关闭活动管理服务与原孵化器的连接的关断模块。

[0051] 进一步，所述连接器还被配置为执行如下功能：通过子孵化器所具有的套接口建立所述控制模块与该子孵化器的连接，以向该子孵化器传递所述的请求。

[0052] 较佳的，所述控制模块依据预设置数据确定将所述请求传递给子孵化器或原孵化器。所述预设置数据，经由用户界面基于待运行应用程序接收用户对所述原孵化器和子孵化器的选定结果而生成。

[0053] 具体的，所述子孵化器的套接口对应的数据存储于相应的文件中。所述子孵化器的套接口文件存储于本地目录中。较佳的，所述子孵化器的套接口文件存储于系统目录/dev/socket中。所述子孵化器的套接口文件的名称与该子孵化器的进程名称相同。

[0054] 进一步，所述控制模块包括构造器，用于利用原孵化器构造子孵化器，该构造器被配置为执行如下功能：

[0055] 复制原孵化器的可执行代码；

[0056] 向该可执行代码中插入用于实现外部调用的调用指令；

[0057] 运行修改后的可执行代码以实现所述子孵化器的构造。

[0058] 较佳的，所述实现外部调用的调用指令，用于调用外部监控单元，以实现对当前子孵化器所构造的进程空间所发生的事件行为的监控。

[0059] 此外，所述构造器还被配置为执行如下功能：向该可执行代码中插入用于实现子孵化器自校验的代码。

[0060] 具体的，所述关断模块配置有共享库文件，该共享库文件包含有用于实现所述关断模块的至少部分功能的函数。

[0061] 较佳的，所述系统服务进程为SystemServer进程，所述原孵化器为Zygote进程，所述活动管理服务进程为ActivityManagerService进程。

[0062] 进一步，所述控制模块包括验证单元，用于在接收所述的请求之后，从该请求中提取应用程序特征信息，基于该特征信息从本地或远程已知设置数据中检验该特征信息所对应的应用程序是否应被限制或被禁止，当其为应被限制或应被禁止的应用程序时，控制模块将该请求中转至所述原孵化器之外的其余子孵化器或不中转该请求至任何孵化器。

[0063] 相较于现有技术，本发明至少具有如下优点：

[0064] 1、本发明利用Android系统固有的原孵化器Zygote构造出新的子孵化器，来使子孵化器独立于原孵化器，然后通过控制活动管理服务的请求的转向，而实现应用程序在由本发明构造的子孵化器中运行。一般的非法侵入是基于系统已知的机制而实现的，由于子

孵化器相对于原孵化器而独立，恶意程序由于不能识别子孵化器的内部机制，因而，即使恶意程序在系统已Root的情况下企图深入系统底层对Zygote进行破坏，或者企图通过诸如ELF文件感染的方式实现病毒传播，这些企图均可能对子孵化器失效，由子孵化器衍生进程加载的应用程序的运行也就更为安全。

[0065] 2、构造出本发明的子孵化器，并且由本发明的控制模块实现了活动管理服务所发起的请求的管理，其本质即控制了应用程序的运行进程的源头，而由于子孵化器有相对的独立性，因此，由子孵化器孵化出来的进程空间，在加载了应用程序之后，便成为一个沙箱。辅以对应用程序的事件行为实施监控的监控单元之后，自然可以起到更为卓越的沙箱监控效果。

[0066] 3、本发明进而通过在子孵化器构造过程中植入外部调用指令，通过该外部调用指令可以实现对监控单元的加载，使加载的监控单元先于应用程序而启动，从而确保事件行为监控效果。由于子孵化器实质上是系统原孵化器的副本，因此适用对fork()函数的调用，因此子孵化器才能够用于孵化适于应用程序运行的新进程空间。本发明的子孵化器在构造过程中便已被植入外部调用指令，通过该外部调用指令加载的模块，包括所述监控单元在内，均可以随同子孵化器为响应请求所进行的孵化而被复制，因此而确保监控单元在每个由子孵化器产生的新进程中起作用，可以获得良好的运行可靠性。

[0067] 4、本发明可以借助其方法构造出多个彼此相互独立的子孵化器，这些子孵化器与原孵化器均与本发明的控制模块通过相应的套接口建立连接，因此能被本发明的控制模块有效地维护。控制模块甚至可以增加相应的回收机制，在必要时通过杀灭个别子孵化器来回收系统内存。更为重要地，多个子孵化器可以用于实现不同类型应用程序的运行管理，让一个子孵化器对应于一类应用程序，而让另一子孵化器对应于另一类程序，同理有利于改善系统安全。

[0068] 由此可见，本发明所取得的优点是系统性，不仅具有从底层到应用层均进行安全防护的效果，而且具有对进程进行灵活中转和控制的管理功能。

[0069] 本发明附加的方面和优点将在下面的描述中部分给出，这些将从下面的描述中变得明显，或通过本发明的实践了解到。

附图说明

[0070] 本发明上述的和/或附加的方面和优点从下面结合附图对实施例的描述中将变得明显和容易理解，其中：

[0071] 图1是本发明的整体构思而提供的一种系统的原理框图；

[0072] 图2是本发明的应用程序进程孵化控制方法的流程示意图；

[0073] 图3是本发明的步骤S12的具体步骤流程示意图；

[0074] 图4是本发明的步骤S13的具体步骤流程示意图；

[0075] 图5是本发明的安全沙箱构造方法的流程示意图；

[0076] 图6是本发明的步骤S31的具体步骤流程示意图；

[0077] 图7是本发明的步骤S312的具体细分步骤流程示意图。

具体实施方式

[0078] 下面详细描述本发明的实施例，所述实施例的示例在附图中示出，其中自始至终相同或类似的标号表示相同或类似的元件或具有相同或类似功能的元件。下面通过参考附图描述的实施例是示例性的，仅用于解释本发明，而不能解释为对本发明的限制。

[0079] 本技术领域技术人员可以理解，除非特意声明，这里使用的单数形式“一”、“一个”、“所述”和“该”也可包括复数形式。应该进一步理解的是，本发明的说明书中使用的措辞“包括”是指存在所述特征、整数、步骤、操作、元件和/或组件，但是并不排除存在或添加一个或多个其他特征、整数、步骤、操作、元件、组件和/或它们的组。应该理解，当我们称元件被“连接”或“耦接”到另一元件时，它可以直接连接或耦接到其他元件，或者也可以存在中间元件。此外，这里使用的“连接”或“耦接”可以包括无线连接或无线耦接。这里使用的措辞“和/或”包括一个或更多个相关联的列出项的全部或任一单元和全部组合。

[0080] 本技术领域技术人员可以理解，除非另外定义，这里使用的所有术语(包括技术术语和科学术语)，具有与本发明所属领域中的普通技术人员的一般理解相同的意义。还应该理解的是，诸如通用字典中定义的那些术语，应该被理解为具有与现有技术的上下文中的意义一致的意义，并且除非像这里一样被特定定义，否则不会用理想化或过于正式的含义来解释。

[0081] 本技术领域技术人员可以理解，这里所使用的“终端”、“终端设备”既包括无线信号接收器的设备，其仅具备无发射能力的无线信号接收器的设备，又包括接收和发射硬件的设备，其具有能够在双向通信链路上，执行双向通信的接收和发射硬件的设备。这种设备可以包括：蜂窝或其他通信设备，其具有单线路显示器或多线路显示器或没有多线路显示器的蜂窝或其他通信设备；PCS(Personal Communications Service，个人通信系统)，其可以组合语音、数据处理、传真和/或数据通信能力；PDA(Personal Digital Assistant，个人数字助理)，其可以包括射频接收器、寻呼机、互联网/内联网访问、网络浏览器、记事本、日历和/或GPS(Global Positioning System，全球定位系统)接收器；常规膝上型和/或掌上型计算机或其他设备，其具有和/或包括射频接收器的常规膝上型和/或掌上型计算机或其他设备。这里所使用的“终端”、“终端设备”可以是便携式、可运输、安装在交通工具(航空、海运和/或陆地)中的，或者适合于和/或配置为在本地运行，和/或以分布形式，运行在地球和/或空间的任何其他位置运行。这里所使用的“终端”、“终端设备”还可以是通信终端、上网终端、音乐/视频播放终端，例如可以是PDA、MID(Mobile Internet Device，移动互联网设备)和/或具有音乐/视频播放功能的移动电话，也可以是智能电视、机顶盒等设备。

[0082] 本技术领域技术人员可以理解，这里所使用的服务器、云端、远端网络设备等概念，具有等同效果，其包括但不限于计算机、网络主机、单个网络服务器、多个网络服务器集或多个服务器构成的云。在此，云由基于云计算(Cloud Computing)的大量计算机或网络服务器构成，其中，云计算是分布式计算的一种，由一群松散耦合的计算机集组成的一个超级虚拟计算机。本发明的实施例中，远端网络设备、终端设备与WNS服务器之间可通过任何通信方式实现通信，包括但不限于，基于3GPP、LTE、WIMAX的移动通信、基于TCP/IP、UDP协议的计算机网络通信以及基于蓝牙、红外传输标准的近距无线传输方式。

[0083] 本领域技术人员应当理解，本发明所称的“应用”、“应用程序”、“应用软件”以及类似表述的概念，是业内技术人员所公知的相同概念，是指由一系列计算机指令及相关数据资源有机构造的适于电子运行的计算机软件。除非特别指定，这种命名本身不受编程语言

种类、级别，也不受其赖以运行的操作系统或平台所限制。理所当然地，此类概念也不受任何形式的终端所限制。

[0084] 本发明以下即将描述的方法和装置所实施的应用场景，是安装在移动终端上的基于Android操作系统的运行环境。

[0085] 本领域技术人员应当可以预见，由于本发明所揭示的技术涉及到对Android系统级别资源的调用，因而，在实施本发明前需要为以本发明实例化的应用程序的运行获取Root权限，但获取Root权限本身属于现有且公知的先决技术，现实中移动终端用户已经具备自行获取Root权限的操作能力和自觉意识。此外，部分开明的移动终端在其机器出厂时已经为用户开放了系统的Root权限，或者故意为获取Root权限提供了便利手段。因此，不应将其视为影响本发明实施的必要构件。

[0086] 众所周知，Root权限是指Unix类操作系统(包括Linux、Android)的系统管理员权限，类似于Windows(视窗)系统中的Administrator(管理员)权限；Root权限可以访问和修改用户的移动设备中几乎所有的文件(Android系统文件及用户文件，不包括ROM)。但是，由于目前移动终端系统对于Root权限的管理依然严格，通常情况下多数应用或程序都不具备Root权限，因此对于某些需要具备Root权限的操作就无法执行，例如安装或卸载应用等操作，又如实施本发明的方法和装置。基于此，本发明推荐通过如下方式获取Root权限：通过调用系统内置的SU(Super User，超级用户)命令获取Root权限，或者通过获取具有Root权限的shell获取Root权限并在shell中启动进程，然后在获取所述系统的Root权限授权后，即可使后续其他调用进程需执行相关操作时无需重复申请Root权限；具体Root权限获取过程可参照现有技术的Root权限调用函数，因关于Root提权的实现纯属现有技术范畴，本发明在此不再赘述。获取Root权限之后，也就可以对系统实施底层操作，包括本发明中对Zygote的ELF感染接触、让控制模块作为底层服务而运行、甚至由此建立的基于Binder机制的通信等，均基于此而实现。

[0087] 本发明的实现依赖于Android操作系统的固有原理，因而，同理，有必要先介绍以下内容：

[0088] 一、Zygote启动过程：

[0089] Android系统在启动时首先会启动Linux基础系统，然后引导加载Linux Kernel并启动初始化进程(Init)。接着启动Linux守护进程。在启动Linux守护进程的同时还需要启动Zygote进程。

[0090] Zygote在业内被形象地称为孵化器，Zygote进程启动后，首先初始化一个Dalvik VM(虚拟机)实例，然后为它加载资源与系统共享库，并开启Socket监听服务，当收到创建Dalvik VM实例请求时，会通过COW(copy on write)技术最大程度地复用自己，生成一个新的Dalvik VM实例。Dalvik VM实例的创建方法基于Linux系统的fork原理。Zygote进程在系统运行期间，通过Socket监听端口接收到创建虚拟机请求时，通过调用fork函数，从自身孵化出Dalvik VM实例，可以将其理解为孵化出了用于运行目标应用程序的进程空间。

[0091] 在Zygote进程启动完成之后，Init进程会启动Runtime进程。Runtime进程首先初始化服务管理器(Service Manager)，并把它注册为绑定服务(Binder services)的默认上下文管理器，负责绑定服务的注册与查找。然后Runtime进程会向Zygote进程发送启动系统服务组件(System Server)的请求，Zygote进程收到请求后，会“孵化”出一个新的Dalvik

VM实例并启动系统服务进程。

[0092] SystemServer会首先启动两个本地服务(由C或C++编写的native服务),Surface Flinger和Audio Flinger,这两个本地系统服务向服务管理器注册成为IPC服务对象,以便在需要它们的时候很容易查找到。然后SystemServer会启动一些Android系统管理服务,包括硬件服务和系统框架核心平台服务,其中也包括活动管理服务ActivityManagerService(AMS),并将它们注册为IPC服务对象。

[0093] 当SystemServer加载了所有的系统服务后就意味着系统就准备好了,它会向所有服务发送一个系统准备完毕(systemReady)广播。当需要启动一个Android应用程序时,ActivityManagerService会通过Socket进程间通信机制,发送请求通知Zygote进程为这个应用程序创建一个新的进程。

[0094] 二、AMS响应应用程序启动过程:

[0095] Android应用程序框架层中,是由ActivityManagerService组件负责为Android应用程序创建新的进程的,它本来也是运行在一个独立的进程之中,不过这个进程是在系统启动的过程中创建的。ActivityManagerService组件一般会在以下情况下为应用程序创建一个新的进程:当系统决定要在一个新的进程中启动一个Activity或者Service时,AMS就会试图去创建一个新进程,然后在这个新的进程中启动这个Activity或者Service。

[0096] 当ActivityManagerService启动一个应用程序的时候,就会通过Socket与Zygote进程进行通信,请求它fork一个子进程出来作为这个即将要启动的应用程序的进程。在前面的介绍中可以看到,系统中的两个重要服务PackageManagerService和ActivityManagerService,都是由SystemServer进程来负责启动的,而SystemServer进程本身是Zygote进程在启动的过程中fork出来的。

[0097] 可以看出,Zygote与AMS之间,是基于socket套接口实现通信的。Zygote在启动之前由init创建socket套接口文件,存储于系统目录/dev/socket之下,并且通常所创建的套接口文件,其文件名与Zygote进程名称是相同的,因而,通过这一机制,在上述系统目录处查看套接口文件,即可验证系统是否创建了新的孵化器。该文件存储关于该socket套接口的设置数据。AMS正是通过读取一个这样的套接口文件来建立其与Zygote的直接通信机制的。后续本发明即将揭示的基于socket的通信机制,均与此处同理实现。

[0098] 三、向系统服务进程SystemServer注入功能模块的参考技术:

[0099] 如前所述,Zygote启动后,第一件事便是从自身fork出SystemServer,使其成为系统服务进程,通过该系统服务进程而加载AMS、PMS等服务进程。因此,现有技术中广泛使用注入技术来将需要实现特定功能的功能函数注入到SystemServer中,使之得以执行,实现目的。

[0100] 例如现有技术中的一种实现系统服务进程代码注入的过程为:

[0101] 步骤1:查找Android系统中com.android.phone,system_server,/system/bin/meidaserver三个进程的进程号Process ID,即PID;

[0102] 步骤2:根据所述的PID分别对所述的三个进程的运行状态进行修改,执行加载监视器模块指令,开辟内存空间并将用来加载监视器模块的指令写入其中;

[0103] 步骤3:分别更改所述的三个进程的寄存器状态,使CPU跳转执行所述的指令;

[0104] 步骤4:根据所述的指令,加载监视器模块到注入器模块的内存空间中,所述的监

视器模块开始初始化操作；

[0105] 步骤5：监视器模块在初始化结束后，查找当前进程的libbinder.so的初始地址，并定位ioctl函数在libbinder.so的全局对象列表Global Ojects Table中对应的表项的地址，即GOT中对应的表项的地址；

[0106] 步骤6：修改ioctl对应的GOT表项的内容，使用钩子函数hooked_ioctl的地址进行替换；

[0107] 步骤7：软件执行敏感行为时，会通过ioctl与com.android.phone，system_server，/system/bin/mediaserver三个进程的一个或者多个进行通信和数据交换，钩子函数hooked_ioctl读取并解析软件的敏感行为类型；

[0108] 步骤8：所述的监视器模块写入敏感行为的发起者和时间到日志文件中，得到软件敏感行为监控记录；

[0109] 步骤9：所述的监视器模块监控到敏感行为时，发送消息给用户，同时使敏感行为的操作暂停；

[0110] 步骤10：所述的用户决定是否运行敏感行为的执行，返回同意或者拒绝命令给所述的监视器模块；

[0111] 步骤11：所述的监视器模块获取所述的用户选择的结果，若用户选择同意则使敏感行为继续执行；若用户选择拒绝则终止敏感行为的继续执行。

[0112] 现有技术中能够实现类似上述的注入的方案不胜枚举，可以参照上述关于监视器模块的方法来为本发明后续揭示的控制模块向系统服务进程SystemServer的注入提供已知方案。但为使本发明后续的揭示更有清楚，有必要知悉，本发明即将揭示的关断模块，用于实现关闭AMS与Zygote之间Socket连接的相关函数，可被实现于如前一示例所揭示的共享库文件libbinder.so中，在这种情况下，通过对该动态库文件进行反向，可以至少部分地了解实现本发明的函数的实现细节。

[0113] 四、基于Linux可执行文件ELF的感染接触原理

[0114] ELF(Excutable Linking Format)文件是Linux的可执行文件，用于存放可执行代码。ELF感染接触原理是一种现有技术，通过复制程序的可执行代码，向其中插入实现某种企图的新增代码，然后执行修改后的可执行代码，从而实现对程序进行修改的目的。本发明以下的揭示，即将利用这一原理，而对系统原孵化器Zygote做出修改，从而构造出子孵化器，通过子孵化器实现本发明的方法、装置以及沙箱实例。

[0115] 在了解了上述系统原理和相关知识之后，便于进一步理解本发明的实施例。

[0116] 需要说明的是，本发明试图结合计算机程序的静态和动态两个方面进行描述，所谓静态方面，是指程序安装包、文件、数据库等存储于媒介的存储对象；所谓动态方面，是指被调入内存中执行的动态对象，包括但不局限于进程、线程、所用到的数据等。鉴于计算机软件技术的这些特点，不应将本发明所述及的各个方法、步骤、子步骤、装置、单元、模块等，孤立地理解为仅静态或仅动态的方面，本领域技术人员对此应当知晓。故而，本领域技术人员应当能够依据本发明有关静态的表述而将其对应到动态的进程活动，或者依据本发明有关动态的进程活动对应到其静态的表现形式，建立起静态与动态两方面的必然性关联，以此为基础来理解本发明。

[0117] 此外，本发明结合沙箱原理而提出，故而，本领域技术人员得以结合公知的沙箱实

现原理来理解本发明的实施。沙箱的作用是为目标应用程序提供相对封闭的运行环境，使应用程序对系统的资源访问，借助沙箱安全策略的应用，而被限制在规定的范围之内。因而，本发明后续将揭示其实质的一个方面在于提供一种沙箱实例。

[0118] 首先请参阅图1来理解应用了本发明的一种应用程序进程孵化控制装置而实现的一个系统原理框图。该控制装置包括控制模块12、子孵化器13，以及系统固有的原孵化器10与系统服务进程2中活动管理服务进程模块3(ActivityManagerService，以下简称AMS)。以下详细揭示各个模块所实现的功能以及工作机理：

[0119] 所述的控制模块12，作为一个基于系统底层的独立进程，可以建立基于Binder机制的通信，可以通过用户的点击指令触发界面程序而被该程序进程所调用；也可以通过adb shell命令登录到系统去执行。控制模块负责建立进程运行环境，为此建立各种相关连接。可以看出，基于接入Android系统底层的需要，控制模块具有相对复杂的功能，以下列出有助于实现本发明的几个方面：一方面，该控制模块12需要在运行过程中利用原孵化器10(Zygote)构造出一个或多个子孵化器13(StepZygote)，另一方面，需要负责向系统服务进程注入关断模块而切断活动管理服务与原孵化器10之间的Socket连接，再一方面，控制模块12还需要负责监控并接收在系统服务进程2注册的AMS进程所发起的请求，该请求是AMS为了加载运行应用程序而基于原孵化器10的套接口而建立的连接而提出的，控制模块12需要将该请求转发给所述的子孵化器13，以便确保AMS的请求得到有效的响应。AMS发起的请求，通常包括程序UID、包名之类的参数，而正常情况下，孵化器将为相应的请求返回进程PID，AMS由此获得其程序入口地址而得以加载应用程序。但由于本发明的控制模块12的控制作用，这种机制将通过控制模块12来中转实现。

[0120] 可以看出，图1所示的控制模块12中，进一步还包括连接器122和构造器121，以及验证单元123。

[0121] 所述的连接器122负责执行如下的功能：

[0122] 1、利用原孵化器10的套接口建立控制模块12与原孵化器10的连接。这一功能的实现，是在控制模块12得以运行之后执行的。控制模块12运行之后，通过读取原孵化器10位于/dev/socket/目录下的zygote套接口文件，该套接口文件存储有相应的套接口数据，来建立控制模块12到原孵化器10之间的连接，以便后续通过构造器121构造子孵化器13以及实现对原孵化器10的其他控制。

[0123] 2、向系统服务进程2注入一关断模块120以使之执行，以关闭AMS与Zygote之间的Socket。具体而言，先连接并注入到SystemServer进程，获得android.os.process类的sZygoteSocket成员，调用该成员的close方法，例如socket.close()，实现AMS与Zygote之间的Socket连接的关断，由此，活动管理服务3基于所述原孵化器10套接口所维持的连接被关闭。也就是说，前面提到AMS通过socket与Zygote直接通信，但这一通信机制在此处被关断，从而使AMS不能直接与原孵化器10进行通信。需要注意的是，向系统服务进程注入用于关闭活动管理服务与原孵化器的连接的关断模块120这一功能，虽然在本实施例中由连接器来执行，但本领域技术人员应当理解，该功能也可由较之连接器122更上位的一个模块来执行。

[0124] 3、通过子孵化器13所具有的套接口建立所述控制模块12与该子孵化器13的连接，以向该子孵化器13(StepZygote)传递所述的请求。在本发明的构造器121构造了新的孵化

器之后，依据Zygote固有的功能，将在本地目录下产生一个与子孵化器13相应的套接口文名。这个套接口文件可以与子孵化器13的进程名称(StepZygote)同名以利识别，当然也可不同名。同理，尽管这个套接口文件可以存储于本地的其它目录，但推荐存储于/dev/socket目录以利识别。由此，该连接器122便读取该套接口文件，以该套接口与该子孵化器13建立基于socket机制的通信。

[0125] 所述连接器122通过执行上述的功能，使得控制模块12可以利用所述原孵化器10的套接口监听所述活动管理服务3进程(AMS)为运行应用程序而发起的请求，并且控制着原孵化器10和子孵化器13的连接端口。一旦AMS尝试建立与默认的原孵化器10(Zygote)的连接，控制模块12便可以收到该请求，并将其按需传递给原孵化器10或者子孵化器13，由相应的孵化器向其返回所需的进程PID。

[0126] 所述的构造器121，参照前述关于ELF感染接触技术的介绍，可以知晓，该构造器121所执行的功能中，先是复制原孵化器10的可执行代码，然后在执行代码中插入用于实现外部调用的调用指令，必要时加入实现子孵化器13的程序自校验的代码，最后运行修改的可执行代码以实现所述子孵化器13的构造，由此便可实现利用原孵化器10构造子孵化器13的功能。

[0127] 这里所称的调用指令，可由本领域技术人员灵活实现，例如，调用一个外部函数，也即调用本发明后续即将详细揭示的监控单元130，来辅以实现一个沙箱实例，具体而言，实现对当前子孵化器13所构造的进程空间所发生的事件行为的监控。应当知晓，子孵化器13调用fork函数复制自身之后，这些外部调用及自校验代码均会被复制，也就是说，不仅子孵化器13进程自身，而且由其孵化的进程也能够加载所述的监控单元130，从而子孵化器13孵化一个新进程，即意味着为相应的目标应用程序提供了一个沙箱环境，也就实现了本发明的沙箱实例。

[0128] 所述控制模块12可以依照相同的逻辑，按需调用构造器121去实现多个子孵化器13，使多个子孵化器13分别用于响应不同的应用程序分类的请求，从而为Android系统创建多个安全的孵化器。外部程序对原孵化器10的攻击，理论上将不会影响到由所述控制模块12构建的子孵化器13。

[0129] 所述验证单元123，是为了便于系统操作从而实现交互而设置的可选组件，该验证单元123用于在控制模块12接收所述AMS的请求之后，从该请求中提取应用程序特征信息，例如前文所述的UID、包名等信息，基于该些特征信息从本地或远程已知设置数据(例如存储于本地的或远程的数据库中)中检验该特征信息所对应的应用程序是否应被限制或被禁止，当其为应被限制或应被禁止的应用程序时，由控制模块12负责将该请求中转至所述原孵化器10之外的其余子孵化器13或不中转该请求至任何孵化器。该已知设置数据可以存储各个UID到子孵化器13之间的映射关系，以及是否被限制或被禁止的信息。显然，设置验证单元123能够提供更安全的控制效果，通过配置所述的已知设置数据，尤其在该已知设置数据能够得到及时的远程的更新时，可以借助这一数据来及时控制移动终端的恶意应用程序的运行。

[0130] 有鉴于该验证单元123，本发明的控制模块12在建立好与其他部件的socket连接之后，实际上发挥着路由器的作用。因此，控制模块12需要一张类似路由表的预设置数据，这一预设置数据既可以与前述验证单元123中的已知设置数据一致，也可以是独立的，这种

情况下验证单元123成为控制模块12实现其路由功能的依据，特别适用于具有多个子孵化器13的情况。而在一些简单的实例中，例如，仅仅通过由一前台终端提供的用户界面收集的用户指令，由用户选定将待运行的目标应用程序置于沙箱中运行，即意味着将该应用程序置于子孵化器13所孵化的进程空间中运行，这种情况下，所述的预设置数据可以理解为针对该待运行目标应用而设置的参数，控制模块12通过获取该参数，便可确定将与该待运行目标应用程序相对应的请求中转给子孵化器13而非原孵化器10，从而由该子孵化器13构造该目标应用程序的进程空间，加载该目标应用程序使其运行。同理，如果用户通过前台模块提供的用户界面选定非沙箱运行，则控制模块12可将其理解为应中转给原孵化器10，从而由原孵化器10运行该目标应用程序。

[0131] 通过上述的分析可以看出，一个或多个子孵化器13与原孵化器10如何利用，既可取决于程序提供的安全策略，也可取决于用户的按需选定；既可以利用数据库技术来实现，也可以利用参数的形式来实现；既可以只实现单个子孵化器13与原孵化器10之间的调度实例，也可以实现多个子孵化器13与原孵化器10并存的调度实例。不管依据如何，本发明的控制模块12均能实现有效的进程孵化控制和调度效果。

[0132] 如前所述，本发明的子孵化器13，由于本发明是采用ELF感染接触原理去复制Zygote而构造子孵化器13的，这种情况下，Zygote自身公知且固有的运行机制未被改变，因此，控制模块12控制之下产生的子孵化器13，其依然按照原孵化器10的实现机理，用于响应于控制模块12中转的请求，而孵化出新进程，并以进程PID应答相应的请求。AMS获得该进程PID了，即将待运行的目标应用程序加载到该相应的进程空间中，使目标应用程序得以运行。可以看出，一个子孵化器13崩溃，或者一个由子孵化器13孵化的进程死亡，不会对原孵化器10及其相关进程产生影响，反之亦然。

[0133] 由此可见，本发明的安全进程孵化控制装置可以实现更为安全的进程孵化机制。

[0134] 为了说明本发明的应用程序进程孵化控制装置的详细实现过程，请参阅图2，本发明结合其运行机制而提供一种应用程序进程孵化控制方法。该方法包括如下步骤：

[0135] S11、向系统服务进程2注入关断模块120。

[0136] 由该方法所实现的程序在运行后，基于已获Root权限为前提，利用公知的注入方法，向系统服务进程SystemServer 2注入本发明的关断模块120，这一部分，既包括前述的共享库文件libbinder.so，也包括对诸如地址表的修改等一切便于实现成功注入的操作，以便借此实现关闭AMS与Zygote之间的Socket连接。

[0137] S12、由所述控制模块12利用原孵化器10构造子孵化器13。

[0138] 需要注意的是，这一步骤的实现，与步骤S13中部分子步骤可以根据实际情况调整顺序执行。

[0139] 本步骤在实质上是前述装置中的构造器121所实现的功能的程序化执行过程，用于实现按需获得子孵化器13，而这种需求将由控制模块12依据前述的已知设置数据和/或预设置数据来实现，或者依照默认规则强制自动构造并应用。结合所述构造器121的功能，参阅图3，本步骤可细分为如下步骤：

[0140] S121、复制原孵化器Zygote 10的可执行代码；

[0141] S122、向该可执行代码中插入用于实现外部调用的调用指令，根据不同实施例的需要，通过该调用指令调用用于实现本发明的沙箱实例的监控单元130，采用该监控单元

130来实现对由构造后的子孵化器13所孵化的进程空间的事件行为的监控；

[0142] S123、向该可执行代码插入可用于实现子孵化器StepZygote 13的程序自校验的代码，使得子孵化器13不易被攻击；

[0143] S124、运行修改后的可执行代码，从而使所述子孵化器13得以成功构造，待命用于为活动管理服务ActivityManagerService 3的请求孵化新进程。

[0144] 本领域技术人员应当理解，本发明中，可以通过本步骤S12在原孵化器10的基础上构造出子孵化器13，但子孵化器13的数量不应被限制为一个，理论上子孵化器13的可扩展个数仅受内存空间的限制而复制出多个具有相同性质的子孵化器13，只要如本发明所揭示，在控制模块12中实现对多个这样的子孵化器13的有效调度即可。

[0145] 需要强调的是，上述步骤S122与S123可以更换顺序实现。

[0146] S13、由所述控制模块12接收在系统服务进程2注册的活动管理服务3(进程)为运行应用程序而发起的请求，并将其传递给所述子孵化器13。

[0147] 参照关于前述关于装置的描述可知，本发明实现向系统服务进程SystemServer 2的注入之后，控制模块12即可搭建起新的系统架构，参阅图4，其搭建过程详见如下步骤：

[0148] S131、利用原孵化器10的套接口建立控制模块12与原孵化器10的连接。

[0149] 前文述及，原孵化器Zygote 10的套接口文件/dev/socket/zygote是建立套接口连接的基础。在本步骤中，控制模块12读取该套接口文件，建立与原孵化器10的连接。因此，可以优先考虑在本子步骤之后执行前述步骤S12，将更符合程序实现逻辑。

[0150] S132、执行关断模块120以关闭活动管理服务3(进程)基于所述原孵化器10套接口所维持的连接。

[0151] 结合前文可知，AMS进程维持着其与原孵化器Zygote 10的socket连接，本步骤通过注入到systemserver进程的关断模块120，获得android.os.process类的sZygoteSocket成员，调用该成员的close方法例如socket.close()关闭AMS与Zygote的socket连接，使得原孵化器10不能直接响应于AMS的孵化进程的请求，以便进一步获取更大的控制权。可以看出，前述步骤S11应于步骤S132之前实施，既可以将S11置于S131与S132之间实施，也可以将步骤S132置于步骤S11之后紧接实施。本领域技术人员可以据此变化出本发明的多个实施例。

[0152] S133、利用所述原孵化器10的套接口监听所述活动管理服务3的为运行应用进程而发起的请求。

[0153] 在前述的步骤中，在控制模块12的控制下，已经实现了所述子孵化器13的构造，因此，依照init函数创建Zygote的逻辑，子孵化器StepZygote 13将创建/dev/socket/stepzygote套接口文件，当然该套接口文件可以存储于本地他处，也未必需与子孵化器10的进程名称相同。在本步骤中，控制模块12通过该套接口建立与该子孵化器13的socket连接，与此同时，开始监听原孵化器10的套接口，以便监听AMS发起的孵化进程的请求，以便在后续由控制模块12向原孵化器10或子孵化器13中转该请求。

[0154] 在本步骤中，控制模块12以上述子步骤实现了新的系统架构的搭建，为了进一步实现更为智能化的管理，控制模块12还实现了类似路由的功能。具体而言，当控制模块12接收到AMS发起的所述的请求时，需要对其做出传递给何种孵化器的判断，这一判断的逻辑依据可以有多种实施方式。在前述有关本发明的孵化控制装置的揭示中已经给出的预设置数

据与已知设置数据及其相关辅助逻辑，便是本方法的执行基础。

[0155] 为了实现这种类路由的控制功能，所述控制模块12将按照如下过程来处理其所监听到的AMS请求：

[0156] 所述控制模块12，在控制模块12接收所述AMS的请求之后，从该请求中提取应用程序特征信息，例如前文所述的UID、包名等信息，基于该些特征信息从本地或远程已知设置数据(例如存储于本地的或远程的数据库中)中检验该特征信息所对应的应用程序是否应被限制或被禁止，当其为应被限制或应被禁止的应用程序时，由控制模块12负责将该请求中转至所述原孵化器10之外的其余子孵化器13或不中转该请求至任何孵化器。该已知设置数据可以存储各个UID到子孵化器13之间的映射关系，以及是否被限制或被禁止的信息。显然，经控制模块12的这一处理过程，能够提供更安全有效的控制管理效果，通过配置所述的已知设置数据，尤其在该已知设置数据能够得到及时的远程的更新时，可以借助这类数据来及时控制移动终端的恶意应用程序的运行。

[0157] 本发明的控制模块12在建立好与其他部件的socket连接之后，实际上已经发挥着路由器的作用。因此，控制模块12需要一张类似路由表的预设置数据，这一预设置数据既可以与前述的已知设置数据一致，甚至合二为一，也可以是独立的，这种情况下相关数据便成为控制模块12实现其路由功能的依据，特别适用于具有多个子孵化器13的情况。在一些简单的实例中，例如，仅仅通过由一前台模块提供的用户界面收集的用户指令，由用户选定将待运行的目标应用程序置于沙箱中运行，即意味着将该应用程序置于子孵化器13所孵化的进程空间中运行，这种情况下，所述的预设置数据可以将该用户指令表征为针对该待运行目标应用而设置的参数，控制模块12通过获取该参数，便可确定将与该待运行目标应用程序相对应的请求中转给子孵化器13而非原孵化器10，从而由该子孵化器13构造该目标应用程序的进程空间，加载该目标应用程序使其运行。同理，如果用户通过前台终端提供的用户界面选定非沙箱运行，则控制模块12可将其理解为应中转给原孵化器10，从而由原孵化器10运行该目标应用程序。

[0158] 通过上述的分析可以看出，一个或多个子孵化器13与原孵化器10如何利用，既可取决于程序提供的安全策略，也可取决于用户的按需选定；既可以利用数据库技术来实现，也可以利用参数的形式来实现；既可以只实现单个子孵化器13与原孵化器10之间的调度实例，也可以实现多个子孵化器13与原孵化器10并存的调度实例。不管依据如何，本发明的控制模块12均能实现有效的进程孵化控制和调度效果。

[0159] S14、由所述子孵化器13响应于该请求而孵化新进程以运行所述应用程序。

[0160] 如前所述，经过所述控制模块12的控制，当控制模块12将AMS的请求进行中转后，其中转目标是确定的，如果不是中转给系统的原孵化器10，则转给由本发明构造的一个子孵化器13。需要注意的是，这里所称的中转，不仅包括形式与构造上与AMS发起的原请求一致的指令，还包括经过所述控制模块12按照与子孵化器13协议(例如通过在构造子孵化器13时向可执行代码插入相关代码)的既定规则进行加工转换、能被所述子孵化器13依照该协议而读取的指令。

[0161] 所述子孵化器13接收到控制模块12中转过来的源自AMS的请求后，按照其继承自原孵化器Zygote 10的固有机制，利用其fork()函数复制出新进程，将进程PID返回给AMS。新进程负责加载在构造子孵化器13时预置了接口的监控单元130，还负责构造用于运行发

起所述请求的目标应用程序的虚拟机实例。

[0162] AMS获得进程入口之后，便将所述目标应用程序加载到所述新进程的进程空间中，使所述目标应用程序成功运行。当该目标应用程序结束运行时，由系统依其机制回收虚拟机空间即可。

[0163] 如上所述，本发明提供的应用程序进程孵化控制方法能够对应用程序而言起到更为安全的防护效果。

[0164] 可以看出，前文对本发明的应用程序进程孵化控制方法和装置的揭示，着重在系统层面的实现。进一步，本发明将通过其他实例突出揭示本发明在应用层的实施例。应当理解，基于同一发明构思的事实，前文所揭示的方法和装置中所采用的思路，也将同样适用于后文的方法和装置中。

[0165] 请再次参阅图1，本发明进一步提供的一种安全沙箱构造装置，包括控制模块12、子孵化器13，以及监控单元130。

[0166] 所述的控制模块12，参考图1，具体包括连接器122、构造器121，进一步还可以包括一验证单元123。所述连接器122用于维护控制模块12基于套接口进行的连接，以实现控制模块12分别与原孵化器10和子孵化器13之间的连接；所述构造器121用于以原孵化器10为基础构造所述子孵化器13；所述验证单元123用于依据已知设置数据确定是否满足所述活动管理服务3(进程)发起的请求。在仅实现一个子孵化器13并且默认由该子孵化器13为所有应用程序孵化新进程的实例中，所述验证单元123显然可以直接省略。

[0167] 所述的控制模块12通过前文所揭示的方法向系统服务进程SystemServer 2注入关断模块120，该关断模块120的注入可以由控制模块12的连接器122来实施并负责调用。所述的控制模块12，作为一个基于系统底层的独立进程，可以建立基于Binder机制的通信，可以通过用户的点击指令触发界面程序而被该程序进程所调用；也可以通过adb shell命令登录到系统去执行。控制模块负责建立进程运行环境，为此建立各种相关连接。可以看出，基于接入Android系统底层的需要，控制模块具有相对复杂的功能，以下列出有助于实现本发明的几个方面：一方面，需要在运行过程中利用原孵化器10(Zygote)构造出一个或多个子孵化器13(StepZygote)，另一方面，需要负责向系统服务进程注入关断模块而切断活动管理服务与原孵化器10之间的Socket连接，再一方面，控制模块12还需要负责监控并接收在系统服务进程2注册的AMS进程所发起的请求，该请求是AMS为了加载运行应用程序而基于原孵化器10的套接口而建立的连接而提出的，控制模块12需要将该请求转发给所述的子孵化器13，以便确保AMS的请求得到有效的响应。AMS发起的请求，通常包括程序UID、包名之类的参数，而正常情况下，孵化器将为相应的请求返回进程PID，AMS由此获得其程序入口地址而得以加载应用程序。但由于本发明的控制模块12的控制作用，这种机制将通过控制模块12来中转实现。

[0168] 所述的连接器122负责执行如下的功能：

[0169] 1、利用原孵化器10的套接口建立控制模块12与原孵化器10的连接。这一功能的实现，是在控制模块12得以运行之后执行的。控制模块12运行之后，通过读取原孵化器10位于/dev/socket/目录下的zygote套接口文件，该套接口文件存储有相应的套接口数据，来建立控制模块12到原孵化器10之间的连接，以便后续通过构造器121构造子孵化器13以及实现对原孵化器10的其他控制。

[0170] 2、向系统服务进程2注入一关断模块120以使之执行，以关闭AMS与Zygote之间的Socket。具体而言，先连接并注入到SystemServer进程，获得android.os.process类的sZygoteSocket成员，调用该成员的close方法，例如socket.close()，实现AMS与Zygote之间的Socket连接的关断，由此，活动管理服务3基于所述原孵化器10套接口所维持的连接被关闭。也就是说，前面提到AMS通过socket与Zygote直接通信，但这一通信机制在此处被关断，从而使AMS不能直接与原孵化器10进行通信。

[0171] 3、通过子孵化器13所具有的套接口建立所述控制模块12与该子孵化器13的连接，以向该子孵化器13(StepZygote)传递所述的请求。在本发明的构造器121构造了新的孵化器之后，依据Zygote固有的功能，将在本地目录下产生一个与子孵化器13相应的套接口文名。这个套接口文件可以与子孵化器13的进程名称(StepZygote)同名以利识别，当然也可不同名。同理，尽管这个套接口文件可以存储于本地的其它目录，但推荐存储于/dev/socket目录以利识别。，由此，该连接器122便读取该套接口文件，以该套接口与该子孵化器13建立基于socket机制的通信。

[0172] 所述连接器122通过执行上述的功能，使得控制模块12可以利用所述原孵化器10的套接口监听所述活动管理服务3进程(AMS)为运行应用程序而发起的请求，并且控制着原孵化器10和子孵化器13的连接端口。一旦AMS尝试建立与默认的原孵化器10(Zygote)的连接，控制模块12便可以收到该请求，并将其按需传递给原孵化器10或者子孵化器13，由相应的孵化器向其返回所需的进程PID。

[0173] 所述的构造器121，参照前述关于ELF感染接触技术的介绍，可以知晓，该构造器121所执行的功能中，先是复制原孵化器10的可执行代码，然后在执行代码中插入用于实现外部调用的调用指令，必要时加入实现子孵化器13的程序自校验的代码，最后运行修改的可执行代码以实现所述子孵化器13的构造，由此便可实现利用原孵化器10构造子孵化器13的功能。

[0174] 这里所称的调用指令，可由本领域技术人员灵活实现，例如，调用一个外部函数，也即调用本装置的监控单元130，来辅以实现一个沙箱实例，具体而言，实现对当前子孵化器13所构造的进程空间所发生的事件行为的监控。应当知晓，子孵化器13调用fork函数复制自身之后，这些外部调用及自校验代码均会被复制，也就是说，不仅子孵化器13进程自身，而且由其孵化的进程也能够加载所述的监控单元130，从而子孵化器13孵化一个新进程，即意味着为相应的目标应用程序提供了一个沙箱运行环境。

[0175] 理论上，所述控制模块12可以依照相同的逻辑，按需调用构造器121去实现多个子孵化器13，使多个子孵化器13分别用于响应不同的应用程序分类的请求，从而为Android系统创建多个安全的孵化器。外部程序对原孵化器10的攻击，理论上将不会影响到由所述控制模块12构建的子孵化器13。

[0176] 所述验证单元123，是为了便于系统操作从而实现交互而设置的可选组件，该验证单元123用于在控制模块12接收所述AMS的请求之后，从该请求中提取应用程序特征信息，例如前文所述的UID、包名等信息，基于该些特征信息从本地或远程已知设置数据(例如存储于本地的或远程的数据库中)中检验该特征信息所对应的应用程序是否应被限制或被禁止，当其为应被限制或应被禁止的应用程序时，由控制模块12负责将该请求中转至所述原孵化器10之外的其余子孵化器13或不中转该请求至任何孵化器。该已知设置数据可以存储

各个UID到子孵化器13之间的映射关系，以及是否被限制或被禁止的信息。显然，设置验证单元123能够提供更安全的控制效果，通过配置所述的已知设置数据，尤其在该已知设置数据能够得到及时的远程的更新时，可以借助这一数据来及时控制移动终端的恶意应用程序的运行。

[0177] 有鉴于该验证单元123，本发明的控制模块12在建立好与其他部件的socket连接之后，实际上发挥着路由器的作用。因此，控制模块12需要一张类似路由表的预设置数据，这一预设置数据既可以与前述验证单元123中的已知设置数据一致，也可以是独立的，这种情况下验证单元123成为控制模块12实现其路由功能的依据，特别适用于具有多个子孵化器13的情况。而在一些简单的实例中，例如，仅仅通过由一前台终端提供的用户界面收集的用户指令，由用户选定将待运行的目标应用程序置于沙箱中运行，即意味着将该应用程序置于子孵化器13所孵化的进程空间中运行，这种情况下，所述的预设置数据可以理解为针对该待运行目标应用而设置的参数，控制模块12通过获取该参数，便可确定将与该待运行目标应用程序相对应的请求中转给子孵化器13而非原孵化器10，从而由该子孵化器13构造该目标应用程序的进程空间，加载该目标应用程序使其运行。同理，如果用户通过前台模块提供的用户界面选定非沙箱运行，则控制模块12可将其理解为应中转给原孵化器10，从而由原孵化器10运行该目标应用程序。

[0178] 通过上述的分析可以看出，一个或多个子孵化器13与原孵化器10如何利用，既可取决于程序提供的安全策略，也可取决于用户的按需选定；既可以利用数据库技术来实现，也可以利用参数的形式来实现；既可以只实现单个子孵化器13与原孵化器10之间的调度实例，也可以实现多个子孵化器13与原孵化器10并存的调度实例。不管依据如何，本发明的控制模块12均能实现有效的进程孵化控制和调度效果。

[0179] 所述的子孵化器13，用于通过自身孵化出进程环境并在该进程环境中加载监控单元130及待运行程序。子孵化器13由控制模块12加以构造，控制模块12采用ELF感染接触原理去复制Zygote而构造子孵化器13的，这种情况下，Zygote自身公知且固有的运行机制未被改变，因此，控制模块12控制之下产生的子孵化器13，其依然按照原孵化器10的实现机理，用于响应于控制模块12中转的请求，而孵化出新进程，并以进程PID应答相应的请求。AMS获得该进程PID了，即将待运行的目标应用程序加载到该相应的进程空间中，使目标应用程序得以运行。可以看出，一个子孵化器13崩溃，或者一个由子孵化器13孵化的进程死亡，不会对原孵化器10及其相关进程产生影响，反之亦然。而在目标应用程序得以运行之前，由于控制模块12已经在子孵化器13内部插入了调用所述监控单元130的指令，因此，这种情况下，监控单元130将被提前加载，并且开始监视其所在的进程所发生的事件行为，也就实现了对后续运行的目标应用程序的事件行为的监控。

[0180] 所述监控单元130，如前所述，先于所述目标应用程序而加载。该监控单元130便是沙箱运行环境的实现者，采用Hook技术，由若干挂钩插件构成，每个挂钩插件均可利用钩子函数对目标应用程序中的相关调用指令的入口点进行监视，截获此一调用指令，转向执行相应的钩子函数，由该钩子函数依据沙箱自身逻辑来应答该调用指令，从而达到监控事件行为的目的。

[0181] 这里需要补充的是：术语“钩子”涵盖了用于通过拦截在软件组件之间传递的函数调用、消息、或事件来改变或增加操作系统、应用程序、或其他软件组件的行为的技术。而处

理这种被拦截的函数调用、事件或消息的代码就被称为钩子hook函数。钩子通常用于各种目标，包括对功能进行调试和对功能进行扩展。其示例可以包括在键盘或鼠标事件传递到应用程序之前拦截它们，或者拦截系统调用(system call)，以监视或修改应用程序或其他组件的功能等等。本实施例即可采用钩子hook函数接管所述应用程序运行时所需的安装自校验操作。

[0182] 为叙述的简洁，涉及利用本发明的监控单元130的具体介绍将在后续集中给出，此处暂缓说明。

[0183] 由于本装置主要在于说明沙箱实例的构造，因此，有必要进一步披露以本发明所实现的程序中的一个前台模块，该前台模块可以通过一个安全软件所提供的活动组件(Activity)来实现，运行该活动组件将提供一用户界面，通过该用户界面可以罗列出所有系统应用和用户应用，当用户点击运行某个应用时，可以进一步弹框让用户选定是否通过沙箱运行此一目标应用，当用户选定通过沙箱运行时，自然地，将驱动调用本发明中控制模块12以此为依据，将AMS因上述的界面操作所产生的请求中转给子孵化器13而非原孵化器10，使该目标应用运行于本发明的监控单元130所实现的沙箱运行环境中。当然，前台模块所提供的用户界面的实现是非常灵活多变的，以上仅仅给出一个示例，本领域技术人员可以灵活变通。如果出于便利考虑，本领域技术人员可以避开这一用户界面提供过程，直接默认为通过子孵化器13启动用户在桌面点击的目标应用程序，则整个过程将变得更为快捷便利，而控制模块12的路由工作以及其验证单元123的工作也将相应简化。

[0184] 可见，本发明的安全沙箱构造装置，通过构建子孵化器13的形式，能够为待运行应用程序实现更为安全、可靠、独立的运行环境。

[0185] 相应的，本发明的安全沙箱构造方法，在于依照机器的处理流程做更高效的组织，通过执行本方法，优化依照本发明所实现的程序的运行效率，以便更高效地为应用程序构造沙箱运行环境。

[0186] 请参阅图5所示，本发明的安全沙箱构造方法，包括如下步骤：

[0187] S31、利用系统原孵化器10构造用于孵化进程环境的子孵化器13。

[0188] 本步骤的目的在于构造出独立于系统原孵化器Zygote 10进程的新的子孵化器13，可以参阅图6所示细分为如下子步骤：

[0189] S311、运行控制模块12。

[0190] 如前所述，用户可以通过前台模块接收用户在桌面或者该前台模块提供的用户界面的点击操作，或者通过adb shell指令，来驱动某个目标应用程序被前台模块识别为需要运行于沙箱中，从而调用本发明的控制模块12使之得以运行。关于控制模块12所实现的功能详见前文，恕不赘述。

[0191] S312、利用所述控制模块12，以原孵化器10为基础构造所述子孵化器13。

[0192] 本步骤采用控制模块12按需获得子孵化器13，而这种需求可由控制模块12依据前述的已知设置数据和/或预设置数据来实现，或者依照默认规则强制自动构造并应用。结合图7，本子步骤可细分为如下由控制模块12执行的子步骤：

[0193] S3121、所述控制模块12利用原孵化器10的套接口建立与原孵化器10的连接。

[0194] 原孵化器Zygote 10的套接口文件/dev/socket/zygote是建立套接口连接的基础。在本步骤中，控制模块12读取该套接口文件，建立与原孵化器10的连接。

[0195] S3122、向系统服务进程注入关断模块使之执行以关闭活动管理服务3基于原孵化器10的套接口所维持的连接。

[0196] 如前所述，由控制模块向系统服务进程2注入一关断模块120以使之执行，以关闭AMS与Zygote之间的Socket。具体而言，先连接并注入到SystemServer进程，获得android.os.process类的sZygoteSocket成员，调用该成员的close方法，例如socket.close()，实现AMS与Zygote之间的Socket连接的关断，由此，使得原孵化器10不能直接响应于AMS的孵化进程的请求，控制模块进一步获取更大的控制权。需要指出的是，参阅前文所述，向系统服务进程2注入关断模块120与利用该关断模块120去执行关断操作可以分为两个细分步骤，且只要保持这种先后关系，两个细分步骤可以分离执行。也就是说，关断模块120并注入后，并不一定要即时实施关断操作，可以是在后续再因为后一细分步骤的需要而被提前调用实施关断。前一细分步骤可在控制模块12一被调用即被执行，然后执行S3121，再执行后一步骤。本领域技术人员应当知晓此一变通，将这种原理作用下的所有变化情况均视为同于本实施例。

[0197] S3123、复制原孵化器10的可执行代码并向其植入用于加载所述监控单元130的调用指令。

[0198] 本子步骤的执行，可以参阅前述关于步骤S121－S124的过程：

[0199] S121、复制原孵化器Zygote 10的可执行代码；

[0200] S122、向该可执行代码中插入用于实现外部调用的调用指令，根据不同实施例的需要，通过该调用指令调用本发明的监控单元130，以便采用该监控单元130来实现对由构造后的子孵化器13所孵化的进程空间的事件行为的监控；

[0201] S123、按需向该可执行代码插入可用于实现子孵化器StepZygote 13的程序自校验的代码，使得子孵化器13不易被攻击，请注意，本子步骤在本发明中是可选的；

[0202] S124、运行修改后的可执行代码，从而使所述子孵化器13得以成功构造，待命用于为活动管理服务ActivityManagerService 3的请求孵化新进程。

[0203] S3124、执行所述子孵化器的代码以构造子孵化器13。一旦该子孵化器13得以运行，便相对于原孵化器Zygote 10而独立存在。

[0204] S313、建立控制模块12与子孵化器13的连接。

[0205] 在控制模块12的控制下，已经实现了所述子孵化器13的构造和运行，因此，依照ini进程创造Zygote的逻辑，子孵化器StepZygote 13将创建/dev/socket/stepzygote套接口文件。在本子步骤中，控制模块12通过该套接口建立与该子孵化器13的socket连接，由此，便可开始监听原孵化器10的套接口，以便监听AMS发起的孵化进程的请求，以便在后续由控制模块12向原孵化器10或子孵化器13中转该请求。

[0206] 本领域技术人员应当理解，本发明中，可以通过本步骤S31在原孵化器10的基础上构造出子孵化器13，但子孵化器13的数量不应被限制为一个，理论上子孵化器13的可扩展个数仅受内存空间的限制而复制出多个具有相同性质的子孵化器13，只要如本发明所揭示，在控制模块12中实现对多个这样的子孵化器13的有效调度即可。

[0207] 需要强调的是，为了简化篇幅，本安全沙箱构造方法中所涉及控制模块12，与本发明的安全沙箱构造装置所描述的控制装置有一一对应性，故控制模块12在该装置中所实现的其它可选性功能，例如关于中转、路由的功能等，也同样适用于本方法中，由适用这些可

选性功能而引起的一系列的变化，也应当随带考虑到本方法中来，恕不赘述。

[0208] S32、通过该子孵化器13进行孵化，以为待运行应用程序建立所述进程环境。

[0209] 到达本步骤时，已经过所述控制模块12的控制，当控制模块12监听到AMS的请求后，将对该请求进行中转，其中转目标是确定的，如果不是中转给系统的原孵化器10，则转给由本发明构造的一个子孵化器13。需要注意的是，这里所称的中转，不仅包括形式与构造上与AMS发起的原请求一致的指令，还包括经过所述控制模块12按照与子孵化器13协议(例如通过在构造子孵化器13时向可执行代码插入相关代码)的既定规则进行加工转换、能被所述子孵化器13依照该协议而读取的指令。

[0210] 所述子孵化器13接收到控制模块12中转过来的源自AMS的请求后，按照其继承自原孵化器Zygote 10的固有机制，利用其fork()函数复制出新进程，将进程PID返回给AMS，以此便为待运行应用程序建立了相应的进程环境。

[0211] S33、利用该子孵化器13孵化而得的进程，将监控单元130及待运行应用程序加载到该进程环境中运行，由所述监控单元130对所述待运行应用程序的事件行为实施监控。

[0212] 新进程负责加载在构造子孵化器13时预置了接口的监控单元130，还负责构造用于运行发起所述请求的目标应用程序的虚拟机实例。AMS获得进程入口之后，便将所述目标应用程序加载到所述新进程的进程空间中，使所述目标应用程序成功运行。当该目标应用程序结束运行时，由系统依其机制回收虚拟机空间即可。

[0213] 以上详细揭示了利用本发明的安全沙箱构造方法为应用程序构造安全的沙箱实例的全过程，借助该方法，可以为每一应用程序建构更为安全可靠的进程运行环境。

[0214] 尽管前文详细披露了本发明的整体构思的多个方面，但仍未尽述。本领域技术人员应当知晓，本发明的应用程序进程孵化控制方法、装置与安全沙箱构造方法、装置之间，是本发明整体构思的两个方面，所采用的技术手段是相互印证的，因此，其中一个方面的说明，同时也将适用于另一个方面中。故而，即使本发明的某个方面的揭示存在疏漏，本领域技术人员也可在另一方面寻找到所需的说明，从而还原该方面的整个方案，而不应以此为据否定该某个方面的充分记载。

[0215] 为了突出本发明的沙箱实例，以下进一步详细揭示本发明前文多处述及的被子孵化器13加载的监控单元130的相关具体实例。

[0216] 利用本发明的监控单元130，可以实现更为强大的沙箱运行环境的构建。所述监控单元130可以从一后台沙箱HOOK框架中获取对应于特定的事件行为的挂钩插件(钩子函数)，利用一个或多个挂钩插件挂钩并监控目标应用的特定事件行为从而实现对目标应用程序进程的活动的监控。所述的后台沙箱HOOK框架的挂钩插件，在云端进行集中管理，向各终端进行分发。其中，云端主要构造有Java挂钩插件库和Native挂钩插件库。监控单元130需要挂钩具体事件行为时，通过远程插件接口向后台沙箱HOOK框架发送请求，获得针对特定事件行为的HOOK函数，即所述的挂钩插件，借此建立对特定事件行为的监控捕获和处理。

[0217] 子孵化器13加载了监控单元130之后，将加载向AMS发起运行请求的所述目标应用程序。由于监控单元130先于目标应用程序被加载，目标应用程序一旦运行，便已被监控单元130利用挂钩插件建立了监控，因此，目标应用程序的一切事件行为均在监控单元130的监控范围之内。目标应用程序的安装包是完整未经修改的，能够通过PackageManagerService的查验，因此，目标应用程序被加载后，能够完全合法、正常地运

行，实现目标应用程序原本能实现的所有功能。

[0218] 由于监控单元130与目标应用程序均处于同一进程空间，因而，运行中的监控单元130即建立了对目标应用程序一切事件行为的监控。目标应用程序运行过程中产生的任何事件行为，其事件消息均会被监控单元130捕获并进行相应的处理。

[0219] 目标应用程序产生的特定事件行为被监控单元130捕获，实质上是触发特定事件行为时，所产生的事件消息被监控单元130中相应的挂钩插件(钩子函数)所捕获。捕获该事件消息，即可知晓该事件的意图，继而可以进行后续的处理。

[0220] 对特定事件行为进行处理，需要获取事件行为处理策略。在这一子步骤中，可以进一步借助系统服务来实现人机交互功能。为了实现人机交互效果，本发明可预先结合安全软件将一交互模块注册为系统服务，通过监控单元130建立的交互接口与该交互模块通信，从而实现对用户指令或预设指令的获取。

[0221] 事件行为策略的获取方式非常灵活多样，可通过构造一策略生成装置来执行，以下列举几种为本发明所择一或任意组合使用的策略：

[0222] (1)监控单元130捕获特定事件行为后，通过该交互接口，向所述交互模块发送请求，由交互模块向安全软件的用户界面弹窗问询用户处理策略，该弹窗界面可以直接告知用户有关事件行为的内容及其风险，由用户选择相应的选项作为处理策略。用户选择相应选项并确定后，交互模块获得针对该特定事件行为的处理策略，将其反馈给监控单元130，监控单元130即可根据该用户指令所产生的处理策略对目标应用程序的相应事件行为进行下一步的处理。

[0223] (2)在某些已被公认为相对低风险的事件行为发生时，例如对联系人的只读操作行为，或者在用户为本发明设置了自行检索针对特定事件行为所应采取的处理策略时，本发明利用一本地策略数据库检索相应的针对特定事件行为的处理策略。也就是说，该本地策略数据库中，建立了特定事件行为与相应的处理策略之间的关联，并且存储了多种事件行为与相应的处理策略之间对应关系的记录数据，可以供本发明检索使用。本发明从本地策略数据库中获取相应的处理策略后，方能对相应事件行为做下一步的处理。

[0224] (3)如果用户为本发明设置了远程获取处理策略的选项，或者默认在本地策略数据库检索不到特定事件行为的具体策略时可以远程获取，又或通过前述第(1)种情况进行交互而在规定时限内得不到用户对弹窗的响应，诸如此类的情况，安全软件均可通过其内建的远程策略接口，向预架构的云端发送请求，获得对应于该特定事件行为的相应的处理策略，并用于后续的处理。

[0225] 需要指出的是，有关以上三种获取处理策略的方式，可以交叉配合使用，例如，一旦交互模块接收到监控单元130传递的事件消息的特征，即可依照默认设置，参照第(2)种方式先行检索本地策略数据库，获得系统推荐的处理策略(如果不能从本地策略数据库中获得，甚至可以进一步按第(3)种方式从云端策略数据库中获取)。继而，参照第(1)种方式，在弹窗界面设置系统推荐的处理策略为默认选项。如果用户未在规定时限内确认该默认选项，则以系统推荐的处理策略为准执行后续指令；如果用户将之改变为新的默认选项，则向监控单元130返回用户设置的处理策略。可见，人机交互过程是可以更为灵活自由地实现的。

[0226] 所述的本地策略数据库，可以是云端策略数据库的一个复件，因此，本发明中，设

置一个更新步骤，用于下载云端策略数据库用于更新本地策略数据库。

[0227] 一般情况下，针对特定事件行为的策略可以设置为“拒绝”、“运行”、“询问”三个常见选项，其表征的具体意向为：

[0228] 拒绝：针对该特定事件行为，向目标应用程序发送事件行为已经执行完毕的虚假消息，以禁止该事件行为实际发生；

[0229] 运行：针对该特定事件行为不做任何改变，将相应的事件消息直接转送给系统消息机制，允许目标应用程序继续其事件行为；

[0230] 询问：独立或依附于前述两个选项任意之一，针对该特定事件行为，标记其状态为未知状态，后续重复发生该行为时，需要再行弹窗询问用户。

[0231] 实际应用中，选项“询问”可被忽略，仅需考虑是否拒绝或允许当前事件行为发生即可。

[0232] 所述的事件行为，多种多样，具体包括如下几大类型：

[0233] (1)终端、联网有关的操作：

[0234] 获取运营商信息：目标应用程序例如通过getSimOperatorName()函数可以获得移动终端的IMSI，由此可进一步判断运营商的名称，进一步可以向运营商发送约定指令，实现扣费之类的非法目的。监控平台通过挂钩与此相关的消息，便可以对事件行为的捕获。

[0235] 切换APN操作：同理，目标应用程序通过与APN切换有关的函数实现ANP切换控制的操作，也可被监控单元130通过调用相应的挂钩插件进行监控。

[0236] 类似的操作，还包括获取手机识别码IME的操作，也与上述同理。

[0237] (2)通知栏广告操作：通知栏广告是最易被恶意程序利用的手段，监控单元130通过调用相应的挂钩插件对notify函数产生的事件消息进行监控，也可对其实施监控。

[0238] (3)通信操作：

[0239] 如电话拨打操作，通过startActivity()函数可以监控调用系统拨号界面的事件行为，利用相应的挂钩插件可以对拔打电话操作建立事件行为监控。

[0240] 短信操作，对应于sendTextMessage()之类的函数，同理，可以借助挂钩插件对这类函数建立事件行为监控。

[0241] 联系人操作：一般对应于query()、insert()函数，监控单元130利用挂钩插件挂钩此类函数可以实现对此类事件行为的监控捕获。

[0242] (4)命令操作：

[0243] 如SU提权操作或执行命令操作，均需用到Execve()函数，监控单元130通过监控此函数的返回消息，便可实现该类事件行为的监控。

[0244] (5)界面及访问操作：

[0245] 如创造快捷方式的事件行为，则对应于sentBroacast()函数。同理，对于隐藏程序图标的操作，也可对应特定函数监控之。

[0246] 如HTTP网络访问操作，则对应于sentTo()、write()等函数。

[0247] (6)程序操作：

[0248] 如应用加载操作，指当前目标应用程序加载相关应用的操作，通过对dexClassloader()、loadLibrary()等函数进行挂钩监控，可以实现对此类事件行为的捕获。

[0249] 又如安装子包，则对应于installPackage()函数。

[0250] (7)其它危险操作：

[0251] 例如，子进程侵入操作、衍生物操作、激活设备管理器操作等。

[0252] 其中，子进程是指目标应用程序建立的子进程，在目标应用程序创建子进程时，该子进程的进程空间同样由子孵化器13构造产生，因此，子进程也难逃监控单元130监控。因而，无论是目标应用程序的自身进程，还是其创建的子进程，它们直接或间接所触发的事件行为，均能被本发明的监控单元130所监控，实现较佳的主动防御效果更佳。

[0253] 而所述衍生物，是指目标应用程序自行创建的文件，或者远程下载的文件，通常是指敏感的衍生物，例如安装包。通过挂钩fClose()函数可以捕获该事件。需要指出的是，当监控单元130捕获该事件行为后，可以按照前述的方法，进一步利用远程规则库接口发送请求到云端，由云端利用其黑、白、灰的安全等级行为规则判断该衍生物的安全等级，本发明通过远程规则库接口获得云端判定结果后，进一步弹窗询问用户是否建立对该敏感衍生物的主动防御，由此便可进一步巩固主动防御的效果。

[0254] 上述的事件行为仅为摘录之用，不能理解为对本发明监控的事件行为的限制。

[0255] 依据上述的处理策略和上述关于事件行为的说明，本发明的主动防御方法便可对各种事件行为进行相应的处理。以下列举几种典型的应用实例：

[0256] (1)对目标应用程序的精细拦截的应用：

[0257] 部分恶意程序被安装后，在相当长的一段时间内处于正常使用的状态，麻痹用户的安全意识。但是，运行一段长时间之后，该目标应用程序尝试从后台插入一短信引起用户的关注，达到广告和诈骗的效果。对该目标应用程序应用本发明的沙箱实例之后，通过监控单元130中相应的挂钩插件对短信操作函数的监控，一旦目标应用程序产生短信操作的事件行为，便可捕获这一事件行为，继而，监控单元130通过其交互接口通知作为系统服务运行的交互模块，由交互模块向用户界面弹窗示警。用户点选“拒绝”的处理策略后，被逆反馈给监控单元130，其中相应的挂钩插件便能阻该事件行为的实际发生，达到防范风险的目的。

[0258] (2)对目标应用程序释放恶意文件的应用。

[0259] 目标应用程序为一游戏软件，通过检查更新的方式下载并释放恶意子包，并且调用系统功能安装该子包。本发明对该目标应用程序建立了主动防御的沙箱运行环境之后，可以监控到其下载完文件而产生的事件行为，据此通过交互模块弹窗告警。用户指令拒绝之后，监控单元130中相应的挂钩插件便可直接删除该文件，或者仅仅拒绝该文件的安装行为。

[0260] 本发明中，对于诸如此类的恶意子包，视为敏感衍生物，对衍生物是否存在恶意的判断，可以通过利用预先确定的安全等级进行远程判断。具体而言，当检测到产生衍生物时，将相应的文件或者其签名之类的特征信息通过远程规则库接口发送给云端，并从云端获得其安全等级，如果为黑、灰应用，则在弹窗中建议用户拒绝安装；如果为白应用，则可允许其通行。通过这种方法，便可实现对敏感衍生物的安全防御。如果云端检测不到该衍生物的相关记录，可以要求本方法为其上传该文件，并由云端标示为未知应用，相应的，以灰应用予以标记，以备后用。

[0261] (3)对子进程侵入的应用。

[0262] 被监控的目标应用程序在运行过程中创建子进程，而子进程进一步释放恶意事件行为。监控单元130监控到目标应用程序创建子进程时，即获得子进程的入口，理论上即可以内联钩子的方式加载到该子进程中对该子进程的事件行为的监控。然而，子进程由于也是由子孵化器13孵化的，因此，子孵化器13所孵化的新进程将先于该子进程而加载所述监控单元130，不必利用内联钩子也可以实现对所述子进程的监控。可以看出，无论是由目标应用程序进程直接触发的事件行为，还是由目标应用程序进程所创建的子进程所触发的间接事件行为，均能被监控单元130成功监控。

[0263] 由上述的分析可见，本发明所建构的沙箱运行环境，具有更为高效的可行性。

[0264] 为便于本领域技术人员进一步实现本发明，以下进一步揭示云端服务器与终端设备如何相互配合实现安装包安全等级判断的相关内容：

[0265] 如前所述，由客户端通过远程规则库接口发送到云端服务器的特征信息，包括：Android安装包的包名，和/或，版本号，和/或，数字签名，和/或，Android组件receiver的特征，和/或，Android组件service的特征，和/或，Android组件activity的特征，和/或，可执行文件中的指令或字符串，和/或，Android安装包目录下各文件的MD5值(签名)。

[0266] 实现了本发明的方法或装置的客户端，将指定的特征信息上传到服务器(云端)，在服务器预置的规则库中查找与指定的单个特征信息或其组合相匹配的特征记录；其中，所述服务器预置的规则库中包含特征记录及特征记录对应的安全级别，每条特征记录中包含单个特征信息或特征信息的组合；

[0267] 服务器端规则库中预置了数千条特征记录，其中，第一条特征记录中列出了某种病毒的Android安装包包名，第二条特征记录中列出了某个正常应用的Android安装包版本号及其数字签名的MD5值，第三条特征记录中列出了某个正常应用的Android安装包包名及其receiver特征，第四条特征记录中列出了某种木马的Android安装包包名、版本号及其ELF文件中的特定字符串，等等。

[0268] 关于安全等级的标识，即黑，白(安全)或者灰(未知，可疑)三种标识，可以进一步地表示为：

[0269] 安全：该应用是一个正常的应用，没有任何威胁用户手机安全的行为；

[0270] 危险：该应用存在安全风险，有可能该应用本身就是恶意软件；也有可能该应用本来是正规公司发布的正常软件，但是因为存在安全漏洞，导致用户的隐私、手机安全受到威胁；

[0271] 谨慎：该应用是一个正常的应用，但是存在一些问题，例如会让用户不小心被扣费，或者有不友好的广告遭到投诉等；当发现这类应用之后，会提示用户谨慎使用并告知该应用可能的行为，但是由用户自行决定是否清除该应用；

[0272] 木马：该应用是病毒、木马或者其他恶意软件，此处为了简单统称为木马，但并不表示该应用仅仅是木马。

[0273] 应当理解，云端与客户端之间的配合，可以由本领域技术人员根据本发明所揭示的内容进一步扩充、变换、增删而改善。因而，以上揭示的内容不应理解为实现本发明的方法和装置的限制。

[0274] 经过测试，本发明相对于现有技术有了较宽广的应用范围和应用效果，以下略加阐述：

[0275] 由于本发明已经将HOOK框架做成了服务平台，以挂钩插件的方式为终端配置监控单元130，因此，其加载仅需依赖于相应的配置文件，管理高效且易于实现，对技术人员而言，一些简单的函数调用仅需编写配置文件即可实现挂钩插件的配置，HOOK重入、并发性能高。

[0276] 采用宿主应用程序先后实现对监控单元130和目标应用程序的加载，继而借助监控单元130对目标应用程序的事件行为建立监控，可以实现对Java函数、Native函数的挂钩。

[0277] 综上所述，本发明使得目标应用程序能够运行于更为安全的沙箱运行环境中。

[0278] 以上所述仅是本发明的部分实施方式，应当指出，对于本技术领域的普通技术人员来说，在不脱离本发明原理的前提下，还可以做出若干改进和润饰，这些改进和润饰也应视为本发明的保护范围。

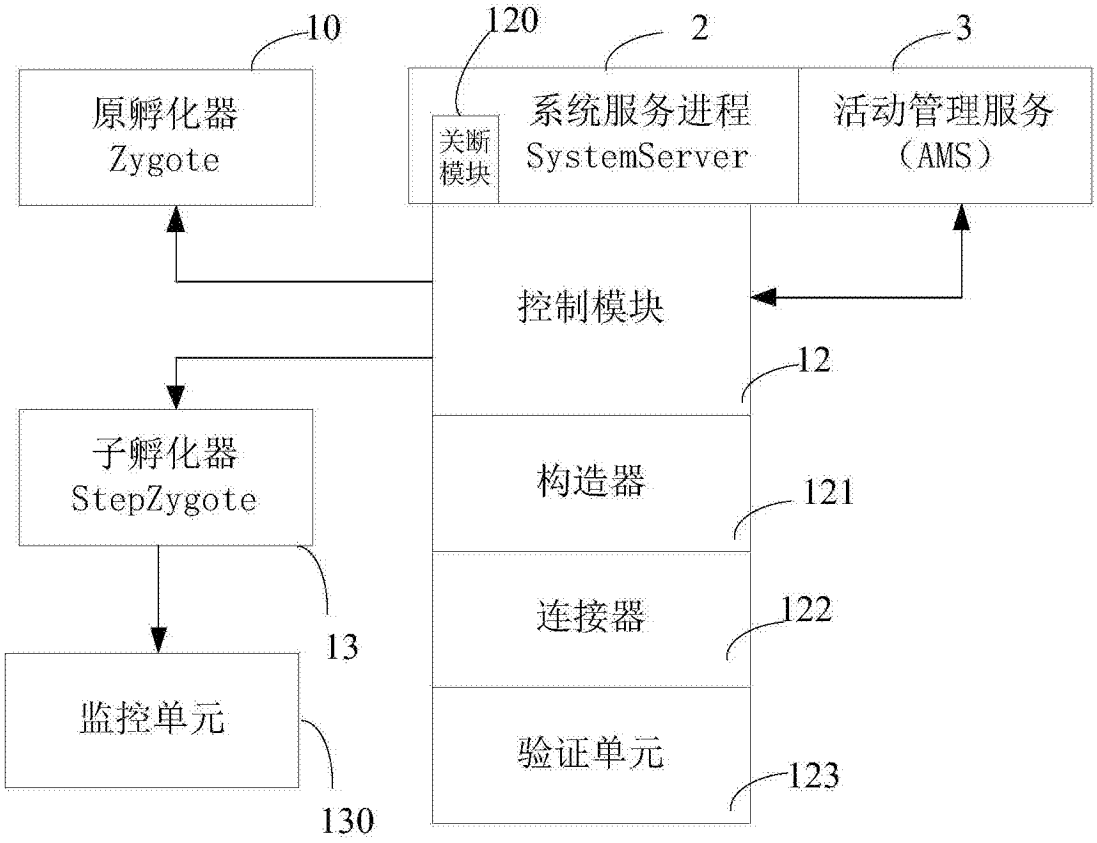

10
120
2
3
原孵化器
Zygote
关断
模块
系统服务进程
SystemServer
活动管理服务
（AMS）
控制模块
12
子孵化器
StepZygote
构造器
121
连接器
122
13
监控单元
验证单元
123
130

图1

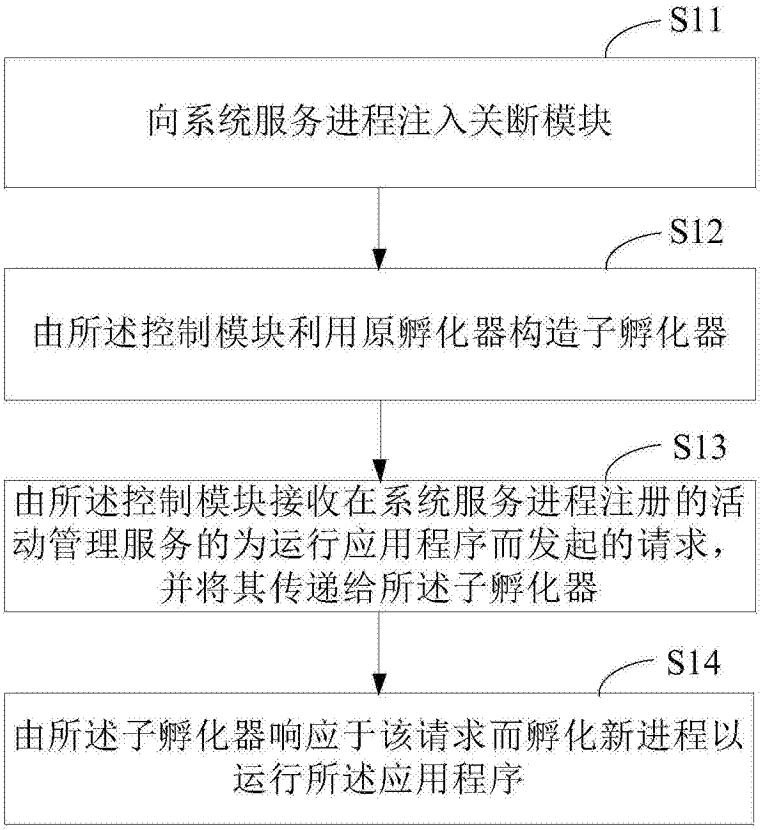

S11
向系统服务进程注入关断模块
S12
由所述控制模块利用原孵化器构造子孵化器
S13
由所述控制模块接收在系统服务进程注册的活动管理服务的为运行应用程序而发起的请求，并将其传递给所述子孵化器
S14
由所述子孵化器响应于该请求而孵化新进程以运行所述应用程序

图2

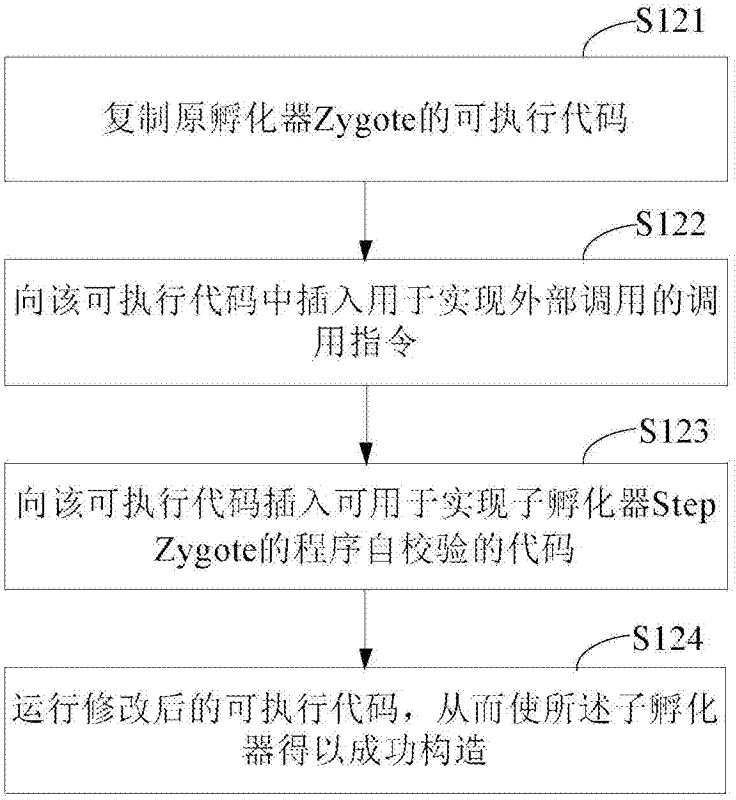

S121
复制原孵化器Zygote的可执行代码
S122
向该可执行代码中插入用于实现外部调用的调用指令
S123
向该可执行代码插入可用于实现子孵化器Step Zygote的程序自校验的代码
S124
运行修改后的可执行代码，从而使所述子孵化器得以成功构造

图3

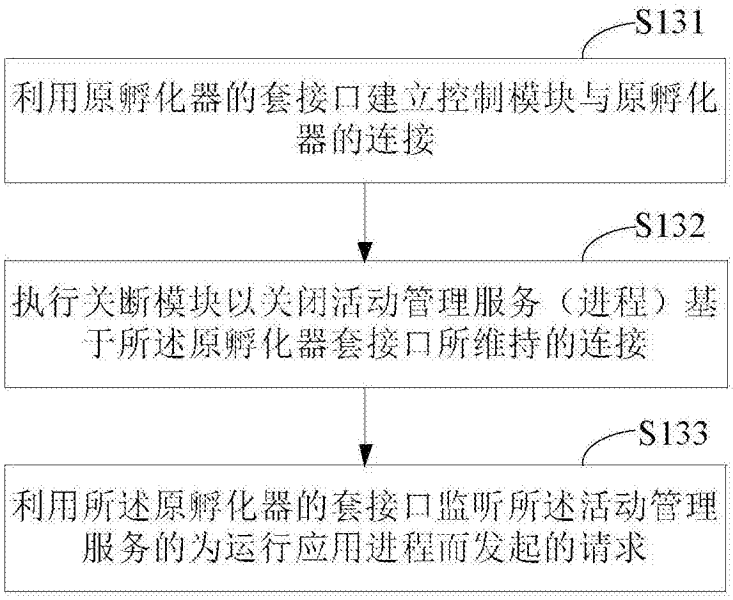

S131
利用原孵化器的套接口建立控制模块与原孵化器的连接
S132
执行关断模块以关闭活动管理服务（进程）基于所述原孵化器套接口所维持的连接
S133
利用所述原孵化器的套接口监听所述活动管理服务的为运行应用进程而发起的请求

图4

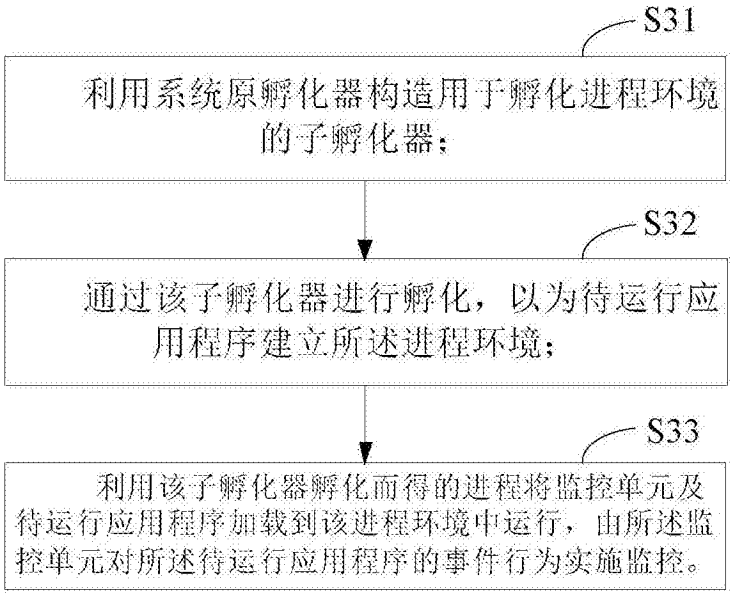

S31
利用系统原孵化器构造用于孵化进程环境的子孵化器；
S32
通过该子孵化器进行孵化，以为待运行应用程序建立所述进程环境；
S33
利用该子孵化器孵化而得的进程将监控单元及待运行应用程序加载到该进程环境中运行，由所述监控单元对所述待运行应用程序的事件行为实施监控。

图5

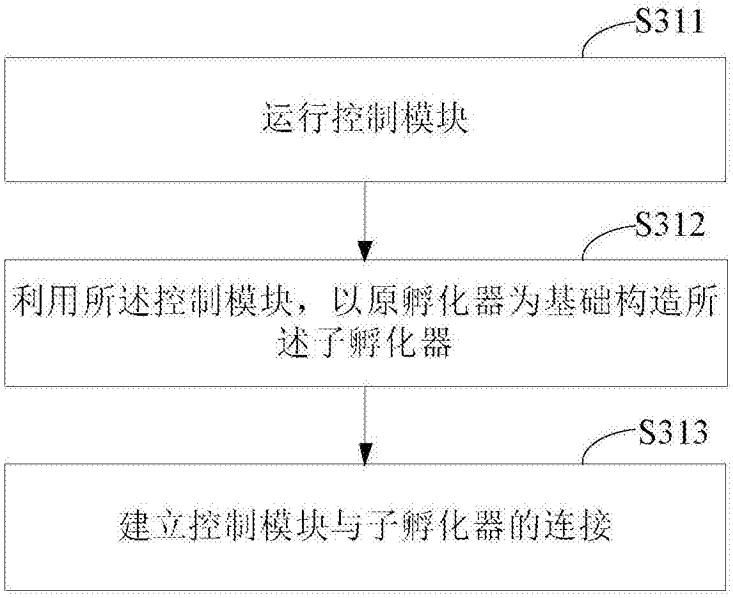

S311
运行控制模块
S312
利用所述控制模块，以原孵化器为基础构造所述子孵化器
S313
建立控制模块与子孵化器的连接

图6

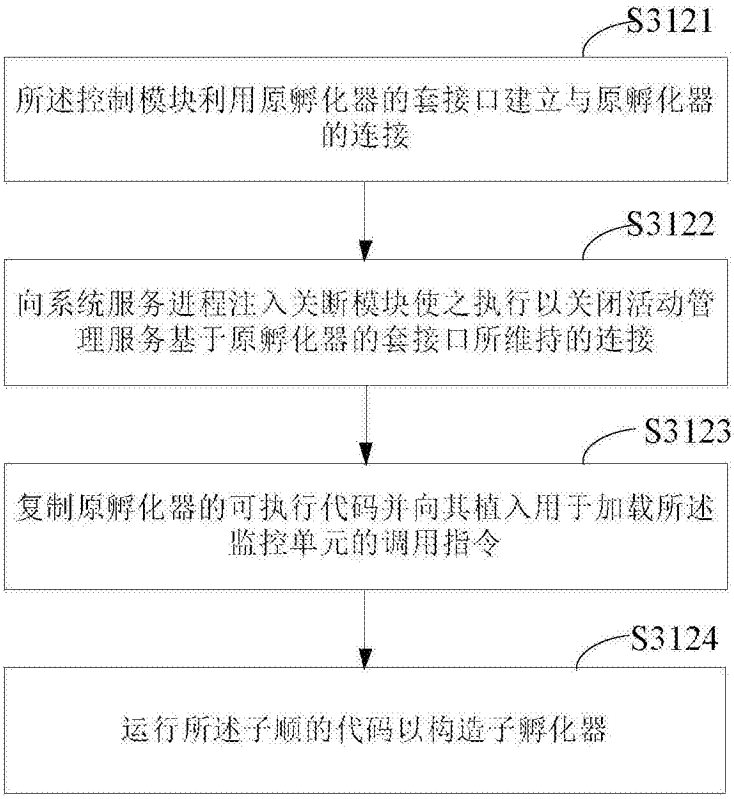

S3121
所述控制模块利用原孵化器的套接口建立与原孵化器的连接
S3122
向系统服务进程注入关断模块使之执行以关闭活动管理服务基于原孵化器的套接口所维持的连接
S3123
复制原孵化器的可执行代码并向其植入用于加载所述监控单元的调用指令
S3124
运行所述子顺的代码以构造子孵化器

图7