(19)国家知识产权局

(12)发明专利

(10)授权公告号 CN 111651169 B
(45)授权公告日 2023.07.04

(21)申请号 202010425267.0

(22)申请日 2020.05.19

(65)同一申请的已公布的文献号
申请公布号 CN 111651169 A

(43)申请公布日 2020.09.11

(73)专利权人 鼎链数字科技(深圳)有限公司
地址 518000 广东省深圳市坪山区坪山街道六联社区坪山大道2007号创新广场B1703-5

(72)发明人 苏年乐

(74)专利代理机构 广州市华学知识产权代理有限公司 44245
专利代理师 郑浦娟

(51)Int.Cl.
G06F 8/60(2018.01)
G06F 8/20(2018.01)

(56)对比文件
CN 107526624 A,2017.12.29
WO 2020011288 A2,2020.01.16
CN 110288307 A,2019.09.27
CN 107562513 A,2018.01.09

审查员 万林青

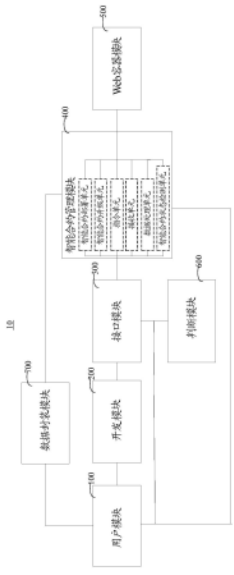
权利要求书2页 说明书8页 附图6页

(54)发明名称

基于web容器的区块链智能合约运行方法及系统

(57)摘要

本发明公开一种基于web容器的区块链智能合约运行方法及系统,包括用户模块、开发模块、智能合约管理模块、接口模块及Web容器模块;通过将Web容器模块作为智能合约运行容器,进而使得智能合约与链平台代码完全隔离,降低了对链平台的安全风险;本申请中能够将不同的智能合约部署在不同的Web容器模块内,从而使得智能合约之间具备隔离性;此外,通过设置用户模块及开发模块,使得用户可对智能合约的完整生命周期进行管理;同时,Web容器模块依赖于JVM运行环境,常用的服务器系统中自带该运行环境,不需安装额外的运行程序,从而提高了智能合约容器安装的便利性和智能合约容器部署的兼容性。



CN 111651169 B

1.基于web容器的区块链智能合约运行方法,其特征在于,

所述基于web容器的区块链智能合约运行方法基于基于web容器的区块链智能合约运行系统;

所述基于web容器的区块链智能合约运行系统包括:

用户模块;

开发模块,所述开发模块与所述用户模块连接,所述开发模块用于提供调用区块链节点的接口;

智能合约管理模块,所述智能合约管理模块与所述开发模块连接,所述智能合约管理模块用于对智能合约生命周期进行管理,所述智能合约管理模块包括部署单元及智能合约升级单元、指令单元及接收单元、数据处理单元及检测单元,所述部署单元与所述智能合约升级单元连接,所述部署单元及所述智能合约升级单元还分别与所述用户模块连接,所述指令单元与所述智能合约升级单元连接,所述接收单元与所述指令单元连接,所述数据处理单元与所述接收单元连接,所述数据处理单元与所述检测单元连接;

接口模块,所述接口模块与所述智能合约管理模块连接,所述接口模块用于提供访问智能合约的接口;及

Web容器模块,所述Web容器模块与所述智能合约管理模块连接,所述Web容器模块用于部署智能合约;

判断模块,所述判断模块与所述接口模块连接;

数据封装模块,所述数据封装模块与所述用户模块连接;

所述基于web容器的区块链智能合约运行方法包括如下步骤:

用户模块发送开发智能合约指令至开发模块,以进行智能合约编写,进而生成智能合约,同时将所述智能合约转换成War包,分别发送至智能合约管理模块及Web容器模块中;

所述用户模块发送调用智能合约部署指令至所述智能合约管理模块,以进行智能合约部署,同时,所述智能合约管理模块对所述War包进行读取操作,并将其转换为字节数组保存至区块链账本;

所述Web容器模块接收到所述War包后,生成读取指令,以调用所述智能合约管理模块,同时,所述智能合约管理模块生成第一HTTP请求,并判断通过调用所述智能合约管理模块通过所述第一HTTP请求的方式对智能合约是否能进行访问,若是,则生成部署成功信息,并将所述部署成功信息发送至所述智能合约管理模块,所述智能合约管理模块将部署成功信息发送至所述用户模块。

2.根据权利要求1所述的基于web容器的区块链智能合约运行方法,其特征在于,在所述步骤所述用户模块发送调用智能合约部署指令至所述智能合约管理模块,以进行智能合约部署,同时,所述智能合约管理模块对所述War包进行读取操作,并将其转换为字节数组保存至区块链账本,具体包括如下步骤:

所述用户模块生成预处理指令,并发送所述预处理指令;

执行所述预处理指令,并生成提案信息,将所述提案信息发送至区块链的背书节点。

3.根据权利要求1所述的基于web容器的区块链智能合约运行方法,其特征在于,在所述步骤所述用户模块发送调用智能合约部署指令至所述智能合约管理模块,以进行智能合约部署,同时,所述智能合约管理模块对所述War包进行读取操作,并将其转换为字节数组

保存至区块链账本后,还包括如下步骤:

所述用户模块发送调用智能合约指令至区块链节点;

所述用户模块接收来自所述区块链节点发送的智能合约待处理指令,并发送至接口模块;

所述接口模块将所述智能合约待处理指令发送至所述智能合约管理模块中,同时,所述智能合约管理模块生成第二HTTP请求,以通过所述第二HTTP请求的方式判断智能合约是否能够被访问,若否,则生成错误反馈信息,并将所述错误反馈信息发送至所述智能合约管理模块,由所述智能合约管理模块将所述错误反馈信息发送至所述用户模块中;若是,则将第二HTTP请求发送至所述接口模块中,以调用智能合约。

基于web容器的区块链智能合约运行方法及系统

技术领域

[0001] 本发明涉及区块链技术领域，特别是涉及一种基于web容器的区块链智能合约运行方法及系统。

背景技术

[0002] 区块链是一种融合数学、密码学、互联网和计算机编程等领域技术，构造的一种具有去中心化、不可篡改、可追溯、集体维护等特性的以分布式账本形式存储链式数据的基础设施，而智能合约是基于区块链的应用与区块链进行数据交互的中间件，在其中可以编写逻辑代码，并输出与输入相对应的执行结果。

[0003] 目前使用的智能合约执行环境主要有四类，分别为采用EVM容器部署智能合约、采用Docker容器部署智能合约、采用引入智能合约Jar包的方式部署以及采用内置智能合约的方式部署。然而，采用EVM容器不支持主流编程语言对智能合约进行编写，同时，与区块链平台的兼容性较低；而采用Docker容器部署智能合约在运行时需要占用较多的计算机资源；进一步地，采用引入智能合约Jar包的方式部署，由于智能合约Jar包与区块链平台自身的Jar包共享内存，没有隔离性，进而对链平台造成安全风险；再进一步地，采用内置智能合约部署的方式无法对智能合约完整生命周期进行管理，即无法对智能合约进行修改升级。

发明内容

[0004] 本发明的目的是克服现有技术中的不足之处，提供一种能够支持主流编程语言对智能合约进行编写、能够减少智能合约在运行时的占用资源、能够降低链平台安全风险的、以及能够对智能合约完整生命周期进行管理的基于web容器的区块链智能合约运行方法及系统。

[0005] 本发明的目的是通过以下技术方案来实现的：

[0006] 一种基于web容器的区块链智能合约运行系统，包括：用户模块、开发模块、智能合约管理模块、接口模块及Web容器模块，所述用户模块与所述开发模块连接，所述开发模块与所述智能合约管理模块连接，所述智能合约管理模块与所述接口模块连接，所述Web容器模块与所述智能合约管理模块连接；

[0007] 所述接口模块用于提供访问智能合约的接口，所述智能合约管理模块用于对智能合约生命周期进行管理，所述开发模块用于提供调用区块链节点的接口。

[0008] 在其中一个实施例中，所述智能合约管理模块包括部署单元及智能合约升级单元，所述部署单元与所述智能合约升级单元连接，所述部署单元及所述智能合约升级单元还分别与所述用户模块连接。

[0009] 在其中一个实施例中，所述智能合约管理模块还包括指令单元及接收单元，所述指令单元与所述智能合约升级单元连接，所述接收单元与所述指令单元连接。

[0010] 在其中一个实施例中，所述智能合约管理模块还包括数据处理单元及检测单元，所述数据处理单元与所述接收单元连接，所述数据处理单元与所述检测单元连接。

[0011] 在其中一个实施例中，还包括判断模块，所述判断模块与所述接口模块连接。

[0012] 在其中一个实施例中，还包括数据封装模块，所述数据封装模块与所述用户模块连接。

[0013] 基于上述实施例中任一项的基于web容器的区块链智能合约运行系统的基于web容器的区块链智能合约运行方法，包括如下步骤：

[0014] S101、用户模块发送开发智能合约指令至开发模块，以进行智能合约编写，进而生成智能合约，同时将所述智能合约转换成War包，分别发送至智能合约管理模块及Web容器模块中；

[0015] S102、所述用户模块发送调用智能合约部署指令至所述智能合约管理模块，以进行智能合约部署，同时，所述智能合约管理模块对所述War包进行读取操作，并将其转换为字节数组保存至区块链账本；

[0016] S103、所述Web容器模块接收到所述War包后，生成读取指令，以调用所述智能合约管理模块，同时，所述智能合约管理模块生成第一HTTP请求，并判断通过调用所述智能合约管理模块通过所述第一HTTP请求的方式对智能合约是否能进行访问，若是，则生成部署成功信息，并将所述部署成功信息发送至所述智能合约管理模块，所述智能合约管理模块将部署成功信息发送至所述用户模块。

[0017] 在其中一个实施例中，在所述步骤所述用户模块发送调用智能合约部署指令至所述智能合约管理模块，以进行智能合约部署，同时，所述智能合约管理模块对所述War包进行读取操作，并将其转换为字节数组保存至区块链账本，具体包括如下步骤：

[0018] S102a、所述用户模块生成预处理指令，并发送所述预处理指令；

[0019] S102b、执行所述预处理指令，并生成提案信息，将所述提案信息发送至区块链的背书节点。

[0020] 在其中一个实施例中，在所述步骤所述用户模块发送调用智能合约部署指令至所述智能合约管理模块，以进行智能合约部署，同时，所述智能合约管理模块对所述War包进行读取操作，并将其转换为字节数组保存至区块链账本后，还包括如下步骤：

[0021] S102c、所述用户模块发送调用智能合约指令至区块链节点；

[0022] S102d、所述用户模块接收来自所述区块链节点发送的智能合约待处理指令，并发送至接口模块；

[0023] S102e、所述接口模块将所述智能合约待处理指令发送至所述智能合约管理模块中，同时，所述智能合约管理模块生成第二HTTP请求，以通过所述第二HTTP请求的方式判断智能合约是否能够被访问，若否，则生成错误反馈信息，并将所述错误反馈信息发送至所述智能合约管理模块，由所述智能合约管理模块将所述错误反馈信息发送至所述用户模块中；若是，则将第二HTTP请求发送至所述接口模块中，以调用智能合约。

[0024] 本发明相比于现有技术的优点及有益效果如下：

[0025] 本发明为一种基于web容器的区块链智能合约运行方法及系统，通过将Web容器模块作为智能合约运行容器，使得智能合约与链平台代码完全隔离，降低了对链平台的安全风险；由于一个Web容器模块中可部署多个智能合约，进而能够有效减少智能合约占用的资源；进一步地，本申请中能够将不同的智能合约部署在不同的Web容器模块内，进而使得智能合约之间具备隔离性；再进一步地，通过设置用户模块及开发模块，使得用户可对智能合

约进行完整生命周期进行管理;同时,本申请的Web容器模块只需运行环境支持Java语言即可,不需安装额外的运行程序,由于主流服务器的操作系统支持Java,从而增强了智能合约容器部署的便利性和跨平台部署的能力。

附图说明

[0026] 为了更清楚地说明本发明实施例的技术方案,下面将对实施例中所需要使用的附图作简单地介绍,应当理解,以下附图仅示出了本发明的某些实施例,因此不应被看作是对范围的限定,对于本领域普通技术人员来讲,在不付出创造性劳动的前提下,还可以根据这些附图获得其他相关的附图。

[0027] 图1为本发明一实施方式的基于web容器的区块链智能合约运行系统的功能模块图;

[0028] 图2为本发明一实施方式的基于web容器的区块链智能合约运行方法的步骤流程图;

[0029] 图3为本发明一实施方式的智能合约开发和调用基本流程示意图;

[0030] 图4为本发明一实施方式的区块链节点调用智能合约的类图;

[0031] 图5为本发明一实施方式的Web容器模块基础类图;

[0032] 图6为本发明一实施方式的智能合约的生命周期状态机图;

[0033] 图7为本发明一实施方式的智能合约部署流程图;

[0034] 图8为本发明一实施方式的智能合约部署时序图;

[0035] 图9为本发明一实施方式的智能合约调用流程图;

[0036] 图10为本发明一实施方式的智能合约调用时序图。

具体实施方式

[0037] 为了便于理解本发明,下面将参照相关附图对本发明进行更全面的描述。附图中给出了本发明的较佳实施方式。但是,本发明可以以许多不同的形式来实现,并不限于本文所描述的实施方式。相反地,提供这些实施方式的目的是使对本发明的公开内容理解的更加透彻全面。

[0038] 需要说明的是,当元件被称为“固定于”另一个元件,它可以直接在另一个元件上或者也可以存在居中的元件。当一个元件被认为是“连接”另一个元件,它可以是直接连接到另一个元件或者可能同时存在居中元件。本文所使用的术语“垂直的”、“水平的”、“左”、“右”以及类似的表述只是为了说明的目的,并不表示是唯一的实施方式。

[0039] 除非另有定义,本文所使用的所有的技术和科学术语与属于本发明的技术领域的技术人员通常理解的含义相同。本文中在本发明的说明书中所使用的术语只是为了描述具体的实施方式的目的,不是旨在于限制本发明。本文所使用的术语“及/或”包括一个或多个相关的所列项目的任意的和所有的组合。

[0040] 请参阅图1,一种基于web容器的区块链智能合约运行系统10包括:用户模块100、开发模块200、智能合约管理模块400、接口模块300及Web容器模块500,用户模块100与开发模块200连接,开发模块200与智能合约管理模块400连接,智能合约管理模块400与接口模块300连接,Web容器模块500与智能合约管理模块400连接;接口模块300用于提供访问智能

合约的接口，智能合约管理模块400用于对智能合约生命周期进行管理，开发模块200用于提供调用区块链节点的接口。

[0041] 需要说明的是，本发明将智能合约作为Web应用程序通过主流编程语言JAVA进行开发，Web容器模块500用于部署智能合约，且一个Web容器模块500可以部署多个智能合约。进一步地，智能合约管理模块400有以下功能：1、负责智能合约生命周期管理，其中包括部署、升级智能合约；2、负责向区块链节点发起调用请求；3、负责接收、处理和转发智能合约执行结果。其中，部署功能包括对智能合约War包的上链存储、Web容器模块500内部署以及状态检测等操作；而升级功能包括对智能合约版本号的检测、智能合约War包上链存储、Web容器模块500内部署以及状态检测等操作；而向区块链节点发起调用请求是指向Web容器模块500发起HTTP请求，通过URL方式访问智能合约；接收智能合约执行结果为在前发起HTTP请求后，等待并接收请求结果；处理智能合约执行结果指将请求结果进行数据解析和重新封装，以满足用户模块所需数据格式，最后的转发智能合约执行结果用于将重新封装后的请求结果发送至用户模块。更进一步地，接口模块用于提供访问智能合约的接口，使得智能合约在在Web容器模块500能够被外部应用通过HTTP的方式进行访问；再进一步地，开发模块200为智能合约开发基础包，其为用户模块提供了开发智能合约的基础框架，用户只需要编写核心代码即可完成智能合约的开发工作，同时，智能合约开发基础包内提供调用区块链节点的必要接口，进而使得智能合约在执行过程中能够通过RPC方式获取区块链上的数据，从而能够向区块链写入数据。

[0042] 还需要解释的是，智能合约的访问通过URL方式进行寻址，是通过IP地址+端口号+群组名称+智能合约名称+智能合约版本号的方式进行寻址，采用此方式能够将在管理上更加清晰。进一步地，本发明简化了智能合约开发时的调试过程。由于智能合约规定为Web应用程序，同时设置开发模块，使得智能合约的开发能够在一个限定范围内进行，但并不限定开发人员要将所有的代码都写在一个类中，只需要在相应类里写控制器方法，通过控制器方法调用其他智能合约的代码，而在开发过程中的调试可以使用开发Web应用程序常用的调试方法。再进一步地，本发明分离了智能合约和平台的代码。由于将智能合约作为Web应用程序开发，仅需要引入智能合约开发基础包即可完成智能合约的开发，保证了平台代码对智能合约的不可见性。更进一步地，本发明避免了过多的端口占用，本申请使用URL方式对智能合约进行寻址，避免了使用端口寻址的方式，有效减少了智能合约的端口占用数量；此外，本发明还保障了智能合约的执行效率；由于智能合约在运行过程中使用RPC方式与区块链节点通信，该方式比HTTP方式速度更快，保障了智能合约的执行效率。最后，本发明保证了向智能合约发起的请求具有标准的返回消息。该效果是由于智能合约是一个Web应用程序，使用HTTP方式调用智能合约时，返回的所有信息都是符合Web标准的信息，便于处理。

[0043] 请参阅图1，进一步地，在一实施方式中，智能合约管理模块400包括部署单元及智能合约升级单元，部署单元与智能合约升级单元连接，部署单元及智能合约升级单元还分别与开发模块连接。

[0044] 需要说明的是，部署单元用于对智能合约进行部署操作，智能合约升级单元用于对智能合约进行升级操作。

[0045] 请参阅图1，进一步地，在一实施方式中，智能合约管理模块400还包括指令单元及接收单元，所述指令单元与所述智能合约升级单元连接，所述接收单元与所述指令单元连

接。

[0046] 需要说明的是,指令单元用于生成HTTP请求,接收单元用于接收及转发智能合约的执行结果。

[0047] 请参阅图1,进一步地,在一实施方式中,智能合约管理模块400还包括数据处理单元及检测单元,数据处理单元与接收单元连接,数据处理单元与检测单元连接。

[0048] 请参阅图1,具体地,基于web容器的区块链智能合约运行系统10还包括判断模块600,判断模块600与接口模块300连接。

[0049] 需要说明的是,数据处理单元用于请求结果进行数据解析和重新封装,以满足用户模块所需数据格式;而检测单元及判断模块600结合,用于进行状态检测,即通过检测接口能否通过HTTP请求方式对智能合约进行访问。

[0050] 请参阅图1,进一步地,在一实施方式中,基于web容器的区块链智能合约运行系统10还包括数据封装模块700,数据封装模块700与用户模块100连接。

[0051] 需要说明的是,数据封装模块700用于将智能合约执行结果封装为调用智能合约管理模块400的用户模块100的接收格式。

[0052] 上述基于web容器的区块链智能合约运行系统10,通过将Web容器模块500作为智能合约运行容器,由于一个Web容器模块500中可部署多个智能合约,进而能够有效减少智能合约占用的资源;进一步地,本申请中能够将智能合约部署在不同的Web容器模块500内,进而使得智能合约与链平台代码完全隔离,一方面,使得智能合约之间具备隔离性,另一方面,降低了对链平台的安全风险;再进一步地,通过设置用户模块100及开发模块200,使得用户可对智能合约进行完整生命周期进行管理;同时,本申请的Web容器模块500只需支持JAVA语言即可,不需安装额外的运行程序,从而减少占用计算机的资源。

[0053] 请参阅图2,基于上述实施例中任一项基于web容器的区块链智能合约运行方法,包括:

[0054] S101、用户模块100发送开发智能合约指令至开发模块200,以进行智能合约编写,进而生成智能合约,同时将所述智能合约转换成War包,分别发送至智能合约管理模块400及Web容器模块500中;

[0055] S102、用户模块100发送调用智能合约部署指令至智能合约管理模块400,以进行智能合约部署,同时,智能合约管理模块对War包进行读取操作,并将其转换为字节数组保存至区块链账本。

[0056] S103、Web容器模块500接收到War包后,生成读取指令,以调用智能合约管理模块400,同时,智能合约管理模块400生成第一HTTP请求,并判断通过调用智能合约管理模块400通过第一HTTP请求的方式对智能合约是否能进行访问,若是,则生成部署成功信息,并将部署成功信息发送至智能合约管理模块400,智能合约管理模块400将部署成功信息发送至用户模块100。

[0057] 具体地,在一实施方式中,在步骤用户模块发送调用智能合约部署指令至智能合约管理模块,以进行智能合约部署,同时,智能合约管理模块对War包进行读取操作,并将其转换为字节数组保存至区块链账本,具体包括如下步骤:

[0058] S102a、用户模块100生成预处理指令,并发送所述预处理指令;

[0059] S102b、执行所述预处理指令,并生成提案信息,将所述提案信息发送至区块链的

背书节点。

[0060] 需要说明的是，用户使用开发模块200开发一套智能合约，并将其打包成War包，并将其发送至数据处理单元及Web容器模块500中。接着，用户模块100生成调用智能合约部署指令，并对智能合约部署指令进行处理，生成预处理指令，将预处理指令发送至数据封装模块700中，数据封装模块700执行预处理指令，并生成提案信息，将提案信息发送至区块链的背书节点，最后由背书节点调用部署单元的功能，以对智能合约进行部署操作；同时，数据处理单元对War包进行读取操作，并将其转换为字节数组保存至区块链账本，并且数据处理单元会将War包复制至Web容器模块500的webapps目录下，紧接着，Web容器模块500生成读取指令，以调用检测单元，同时，指令单元生成第一HTTP请求，并发送至判断模块，以判断通过调用检测单元通过第一HTTP请求的方式对智能合约是否能进行访问，若否，则在预设时间阈值内，生成重复判断指令发送至判断模块600，以使判断模块600再次判断通过调用检测单元通过第一HTTP请求的方式对智能合约是否能进行访问，若在预设时间阈值内仍然无法访问，则生成智能合约部署失败信息，并将智能合约部署失败信息发送至接收单元，接收单元将部署失败信息发送至用户模块100；若是，则生成部署成功信息，并将部署成功信息发送至接收单元，接收单元将部署成功信息发送至用户模块100，其中这里的预设时间阈值可以由用户自行设置。上述方法为智能合约开发流程。

[0061] 进一步地，在一实施方式中，在步骤用户模块发送调用智能合约部署指令至智能合约管理模块，以进行智能合约部署，同时，智能合约管理模块对War包进行读取操作，并将其转换为字节数组保存至区块链账本后，还包括如下步骤：

[0062] S102c、用户模块100发送调用智能合约指令至区块链节点；

[0063] S102d、用户模块100接收来自所述区块链节点发送的智能合约待处理指令，并发送至所述接口模块300；

[0064] S102e、接口模块300将智能合约待处理指令发送至智能合约管理模块400中，同时，智能合约管理模块400生成第二HTTP请求，以通过第二HTTP请求的方式判断智能合约是否能够被访问，若否，则生成错误反馈信息，并将错误反馈信息发送至智能合约管理模块400，由智能合约管理模块400将错误反馈信息发送至用户模块100中；若是，则将第二HTTP请求发送至接口模块300中，以调用智能合约。

[0065] 需要说明的是，上述方法为用户或者调用方调用智能合约的过程。这里需要补充的是，在智能合约执行过程中，若是需要对区块链账本进行操作，则会调用开发模块200中的相应方法，通过RPC方式与区块链进行数据交互，在智能合约代码执行完成后，将会把结果原路返回至数据封装模块700，由数据封装模将智能合约执行结果封装为调用智能合约的调用方或用户的接收格式，并发送至调用方或用户模块100。

[0066] 为了更好的理解上述智能合约的开发、升级及调用的流程，下面列举一具体实施例。

[0067] 请参阅图3，图3为智能合约开发和调用基本流程示意图，用户具体调试流程如下：1、智能合约以Web应用的形式部署在Web容器中；2、用户通过浏览器或者Web调试工具访问智能合约对外提供的接口；3、智能合约通过RPC与区块链节点进行通信。而用户具体调用流程：1、智能合约以Web应用的形式部署在Web容器中；2、区块链节点通过HTTP访问智能合约对外提供的接口；3、智能合约通过RPC与区块链节点进行通信。

[0068] 进一步地，为了更好地解释上述基于web容器的区块链智能合约运行方法请参阅图4，图4为区块链节点调用智能合约的类图，由ISmartContractSupport继承一个用于web容器的类RestSmartContractSupport，用以进行智能合约的载入和调用。在SmartContractExecutor调用support之后，会进行判断，若判断结果为launch，support调用RestContainerServer中的deploy()对智能合约的War包进行处理，先将其拷贝至容器的webapps目录，并重命名，接着将智能合约War包上链保存，并返回部署结果。若判断结果为execute，则support调用RestContainerServer中isExecutable()检测智能合约是否可用，若不可用，返回错误信息；若可用，执行invoke()与Web容器模块的controller进行通信。下面对图2中部分名词进行解释，destroy为删除智能合约功能；upgrade为升级智能合约功能；verifySc指校验智能合约是否与链上所存相同；uploadSc2Chain至上传智能合约到链上；downloadScFromChain指从链上下载只能合约。

[0069] 请参阅图5，图5为Web容器模块基础类图，在Web容器模块中，包含一个基础包Base，用于提供与区块链节点进行交互的接口，以及Web应用的基本环境。其中，ScGrpcClient负责将getState、putState等操作向Node发起请求，由Node端的RPC服务端对请求进行处理。在apps包中，存放所有智能合约的代码，每一个智能合约需要有一个类继承SmartContractBase，覆写init和invoke方法，调用Base提供的getState、putState对账本进行操作。智能合约通过继承自Base的init和invoke方法接收HTTP请求，对HTTP请求进行个性化处理，调用SmartContractBase提供的方法，通过ScGrpcClient执行getState、putState方法，将处理结果用HTTP响应返回给请求方，这里的请求方指用户模块。

[0070] 请参阅图6，图6 为智能合约的生命周期状态机图，在用户使用SmartContractBase将智能合约打包为War包，并放在服务器上时，视为已就绪状态，在调用launch()方法将智能合约War包上链、拷贝到Web容器的指定目录下、将其存放于智能合约map中，并处于可访问状态时，视为已部署状态。在智能合约部署后，对智能合约进行升级操作，即部署一份版本号不同的同名智能合约，在新部署智能合约实现已部署状态时，视为对原有智能合约完成升级，原有智能合约的状态变为已升级状态，将后续对该名称智能合约的调用指向新部署智能合约。而对于已经部署的智能合约，将其War包删除，并从智能合约Map中将其移除，视为已删除状态。最后，当用户模块向智能合约发起调用指令，执行智能合约内部代码，视为被调用状态，智能合约将在状态结束时返回智能合约内部程序运行结果。

[0071] 为了更好解释智能合约的部署功能，请参阅图7，图7为智能合约部署流程图，SmartContractExecutor将智能合约路径传入Support；Support调用RestContainerServer将智能合约拷贝至容器的webapps目录下，并使用《智能合约名称+智能合约版本号》对智能合约War包进行重命名；RestContainerServer将智能合约进行上链保存；在容器中，构建SmartContractMap单例对象，将智能合约名称作为Key，智能合约War包名称作为value保存到对象的scMap中；若是以上操作没有报错，返回部署成功，否则，返回错误信息。进一步地，请参阅图8，图8为智能合约部署时序图，SmartContractExecutor在调用Support时，传入智能合约路径scPath、调用指令launch command，Support调用RestContainerServer提供的isExecutable()方法检测智能合约是否是已部署状态，若是返回false，则智能合约未部署，发送智能合约路径scPath，调用RestContainerServer提供的deploy()方法。RestContainerServer通过智能合约路径获取智能合约War包，将其拷贝至Web容器的

webapps目录下,并对其进行重命名,使用isExecutable()方法测试智能合约是否已部署。BaseScController调用SmartContractMap单例对象,将智能合约相关信息放入对象中进行保存,返回智能合约已部署状态。逐级返回操作结果至SmartContractExecutor。还需要说明的是,智能合约升级流程与智能合约部署流程相同,只需额外增加一步校验版本号是否重复即可。

[0072] 为了更好的解释智能合约调用流程,请参阅图9,图9为智能合约调用流程图,RestContainerServer访问对应的智能合约的URL,传入智能合约调用参数ScCallArgs;智能合约从传入参数中读取调用参数,执行智能合约;若是需要调用getState()、putState()等操作,则使用Base类中提供的getState()、putState()方法,Base类中调用ScServiceGrpc中实现的getState()、putState()方法与区块链节点建立RPC连接,使用远程调用方式调用Node端的getState()、putState()方法,执行完智能合约内部程序后,执行下一步;若是不需要调用getState()、putState()方法,则在智能合约内部程序执行完毕后,执行下一步;智能合约将执行结果返回至RestContainerServer。进一步地,请参阅图10,图10为智能合约调用时序图,首先,SmartContractExecutor发送要调用的智能合约名称和调用参数至Support;其次,Support调用RestContainerServer提供的isExecutable()方法检测智能合约状态是否是已部署;若是已部署,则发送智能合约名称和调用指令至RestContainerServer,RestContainerServer对指令进行解析,封装为JSON对象,使用HTTP1的方式向智能合约容器发起请求,并携带该JSON对象;然后,BaseScController对请求进行解析,将请求参数发送至相应的智能合约,待智能合约执行完内部程序后,返回执行结果;最后执行结果逐级回传。

[0073] 与现有技术相比,本发明具有以下优点:

[0074] 上述基于web容器的区块链智能合约运行方法及系统,通过将Web容器模块500作为智能合约运行容器,使得智能合约与链平台代码完全隔离,降低了对链平台的安全风险;由于一个Web容器模块500中可部署多个智能合约,进而能够有效减少智能合约占用的资源;进一步地,本申请中能够将不同的智能合约部署在不同的Web容器模块500内,进而使得智能合约之间具备隔离性;再进一步地,通过设置用户模块100及开发模块200,使得用户可对智能合约进行完整生命周期进行管理;同时,本申请的Web容器模块500只需运行环境支持Java语言即可,不需安装额外的运行程序,由于主流服务器的操作系统支持Java,从而增强了智能合约容器部署的便利性和跨平台部署的能力。

[0075] 以上所述实施方式仅表达了本发明的几种实施方式,其描述较为具体和详细,但并不能因此而理解为对本发明专利范围的限制。应当指出的是,对于本领域的普通技术人员来说,在不脱离本发明构思的前提下,还可以做出若干变形和改进,这些都属于本发明的保护范围。因此,本发明专利的保护范围应以所附权利要求为准。

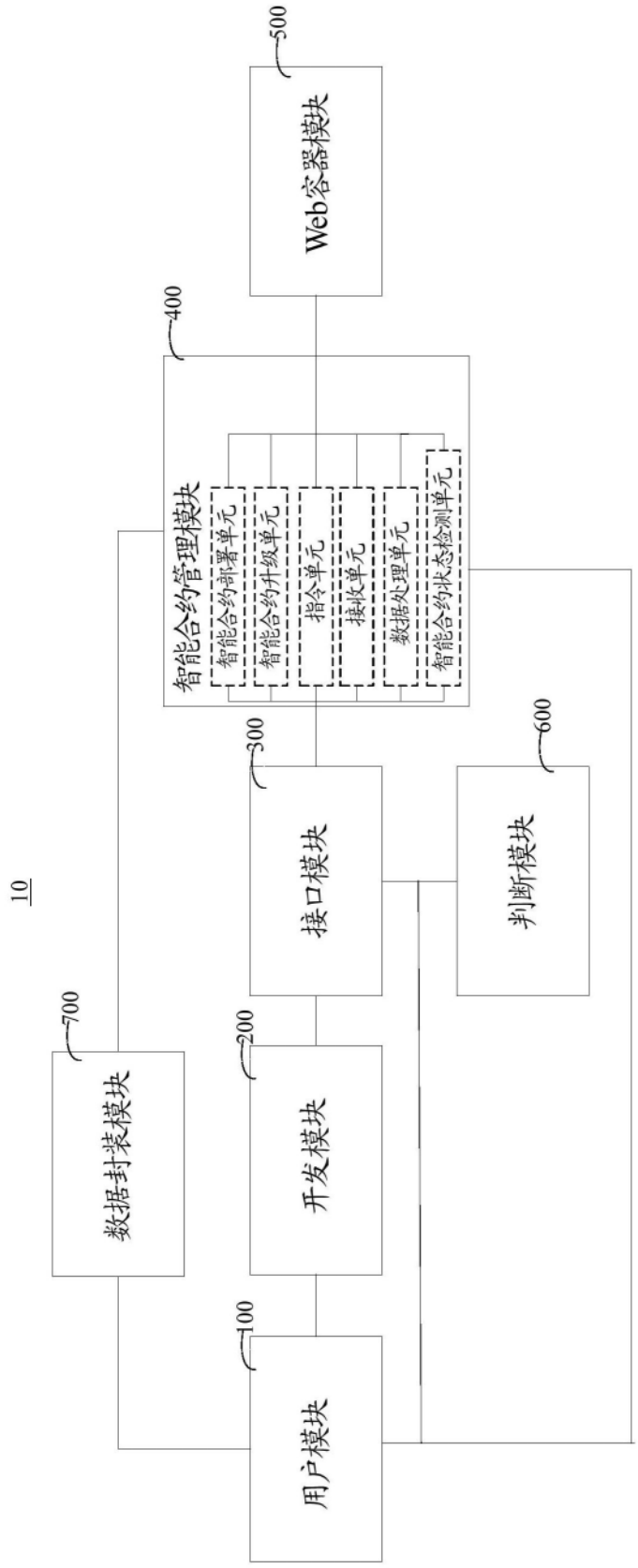

10
100
用户模块
200
开发模块
300
接口模块
400
智能合约管理模块
智能合约部署单元
智能合约升级单元
指令单元
接收单元
数据处理单元
智能合约状态检测单元
500
Web容器模块
600
判断模块
700
数据封装模块

图1

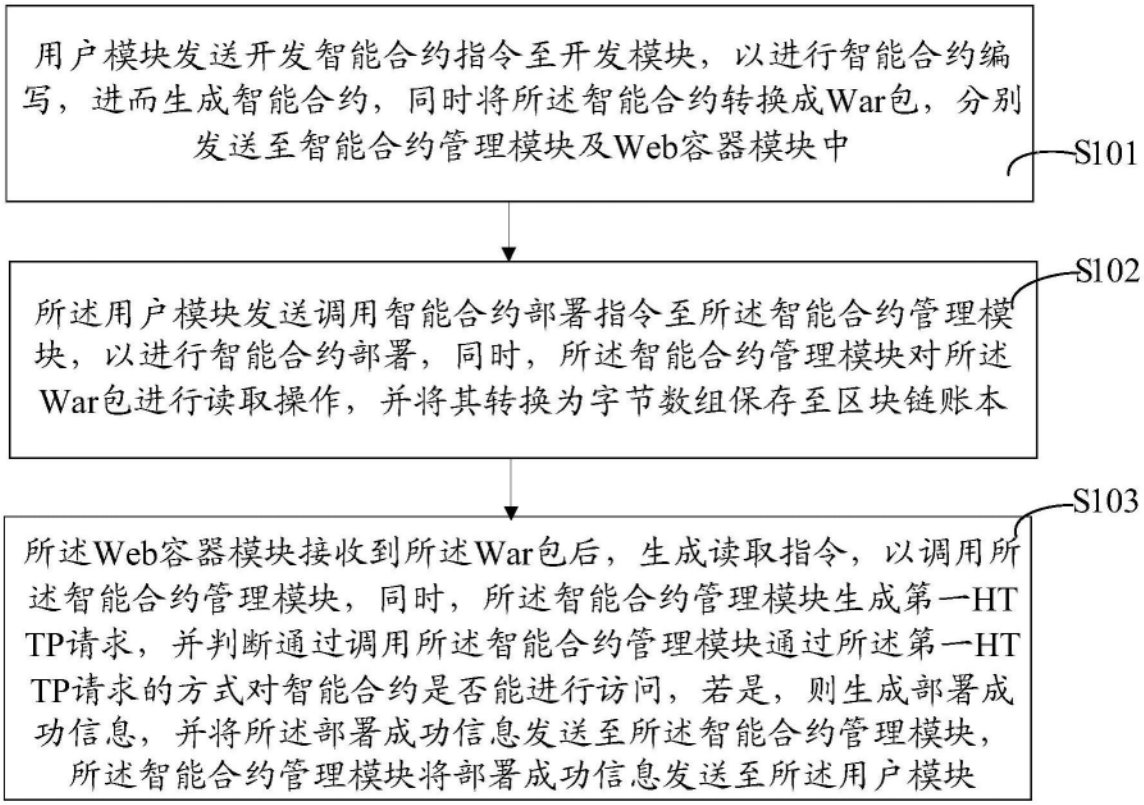

用户模块发送开发智能合约指令至开发模块，以进行智能合约编写，进而生成智能合约，同时将所述智能合约转换成War包，分别发送至智能合约管理模块及Web容器模块中
S101
所述用户模块发送调用智能合约部署指令至所述智能合约管理模块，以进行智能合约部署，同时，所述智能合约管理模块对所述War包进行读取操作，并将其转换为字节数组保存至区块链账本
S102
所述Web容器模块接收到所述War包后，生成读取指令，以调用所述智能合约管理模块，同时，所述智能合约管理模块生成第一HTTP请求，并判断通过调用所述智能合约管理模块通过所述第一HTTP请求的方式对智能合约是否能进行访问，若是，则生成部署成功信息，并将所述部署成功信息发送至所述智能合约管理模块，所述智能合约管理模块将部署成功信息发送至所述用户模块
S103

图2

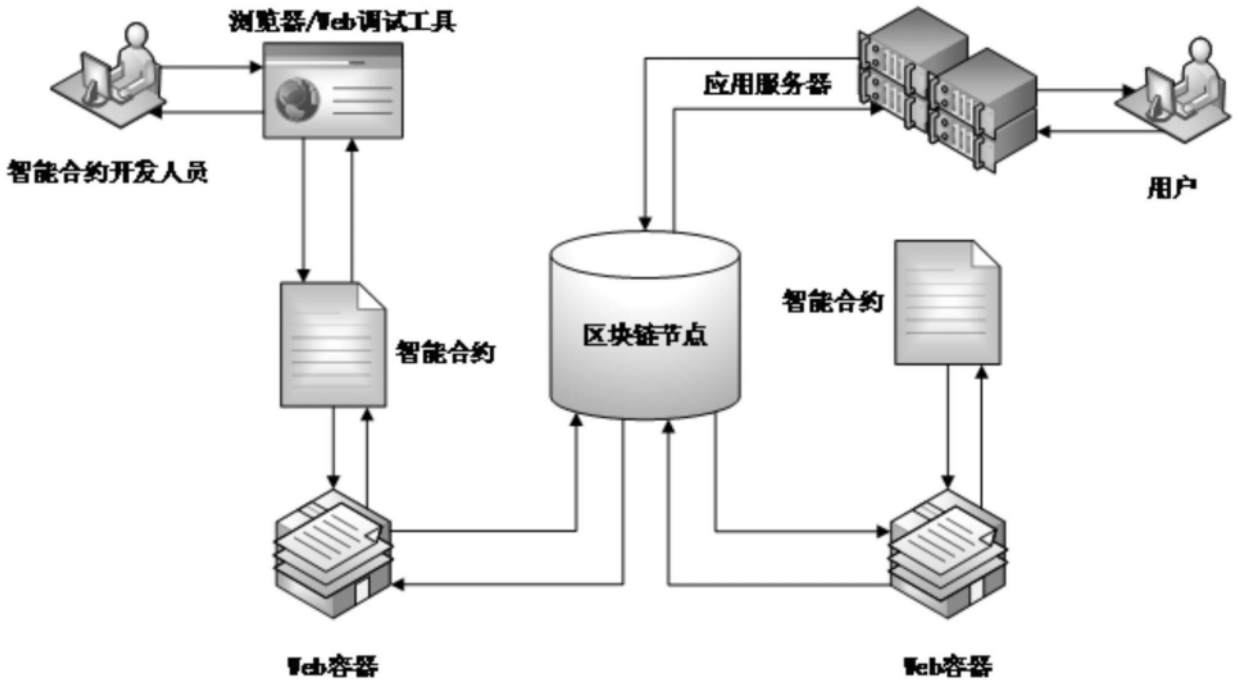

浏览器/Web调试工具
智能合约开发人员
应用服务器
用户
智能合约
区块链节点
智能合约
Web容器
Web容器

图3

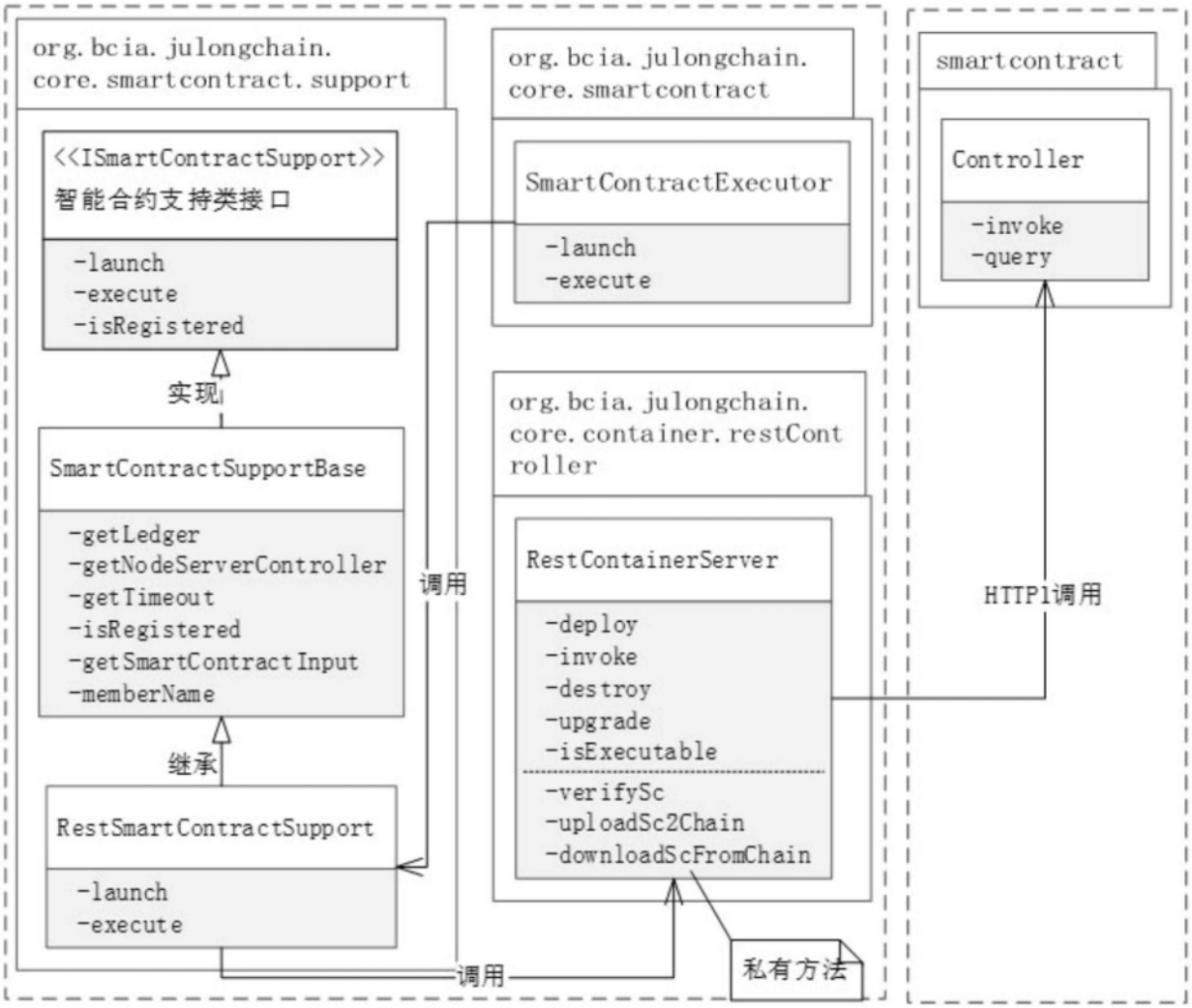

org.bcia.julongchain.
core.smartcontract.support
org.bcia.julongchain.
core.smartcontract
smartcontract
<<ISmartContractSupport>>
智能合约支持类接口
-launch
-execute
-isRegistered
SmartContractExecutor
-launch
-execute
Controller
-invoke
-query
实现
SmartContractSupportBase
-getLedger
-getNodeServerController
-getTimeout
-isRegistered
-getSmartContractInput
-memberName
调用
org.bcia.julongchain.
core.container.restCont
roller
RestContainerServer
-deploy
-invoke
-destroy
-upgrade
-isExecutable
-verifySc
-uploadSc2Chain
-downloadScFromChain
HTTP1调用
继承
RestSmartContractSupport
-launch
-execute
调用
私有方法

图4

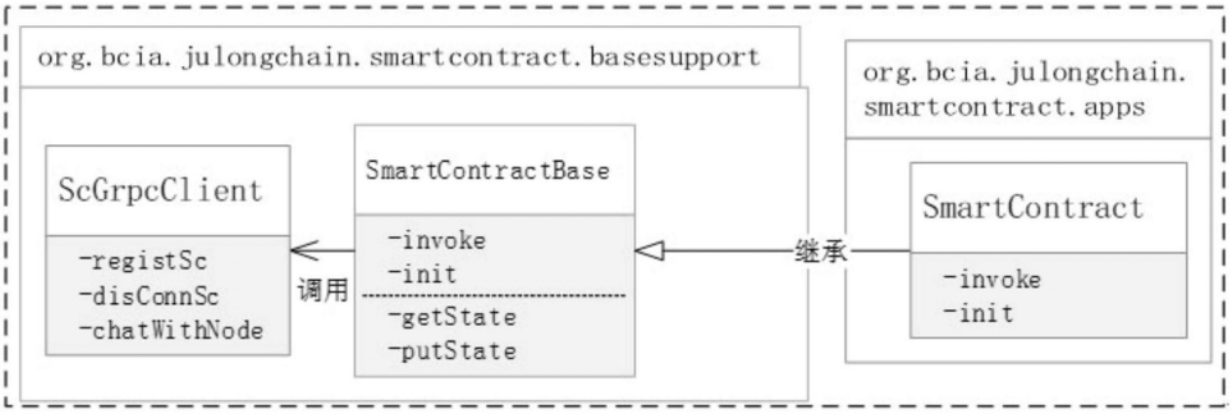

org.bcia.julongchain.smartcontract.basesupport
org.bcia.julongchain.
smartcontract.apps
ScGrpcClient
-registSc
-disConnSc
-chatWithNode
调用
SmartContractBase
-invoke
-init
-getState
-putState
继承
SmartContract
-invoke
-init

图5

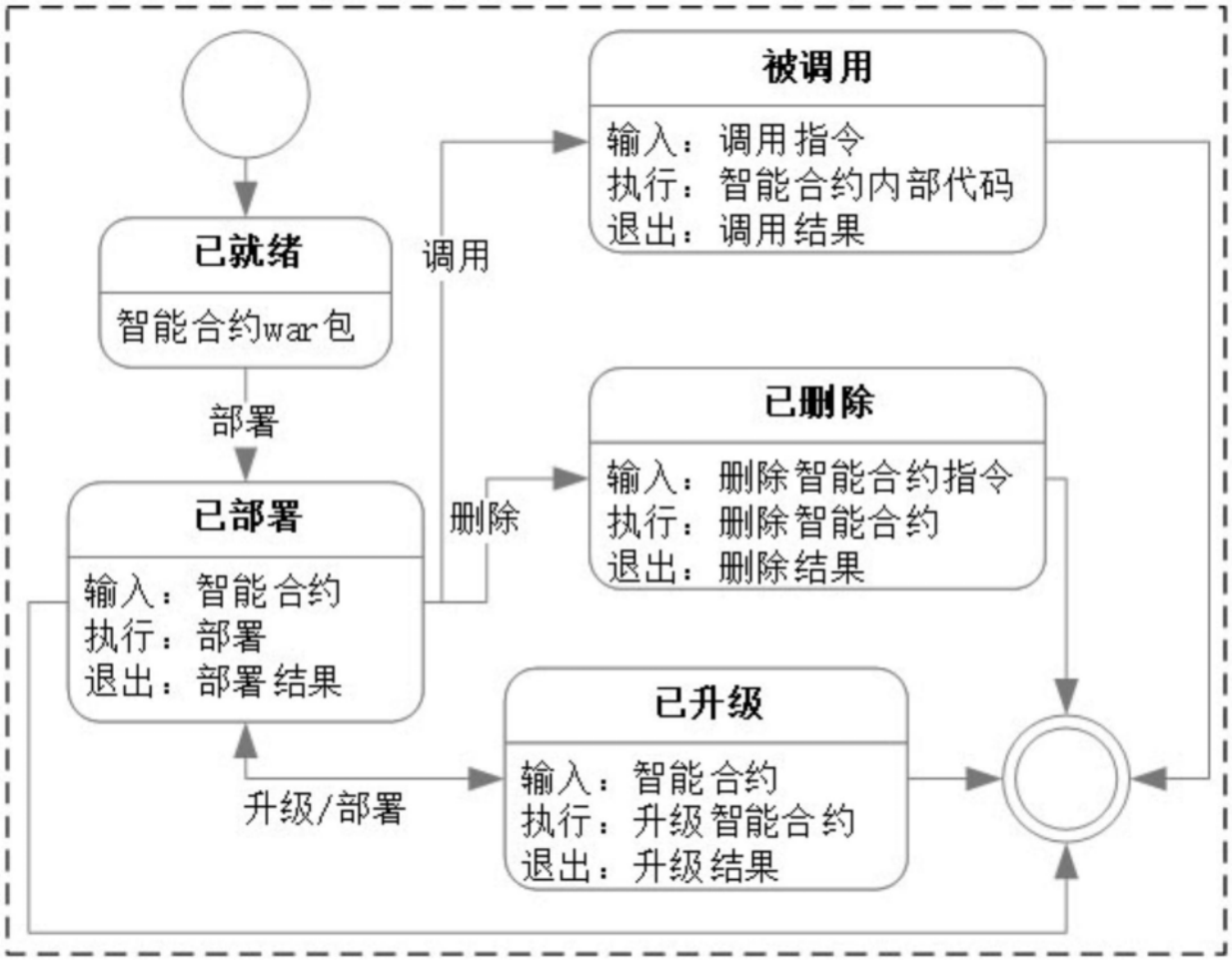

被调用
输入：调用指令
执行：智能合约内部代码
退出：调用结果
已就绪
智能合约war包
调用
部署
已删除
输入：删除智能合约指令
执行：删除智能合约
退出：删除结果
已部署
输入：智能合约
执行：部署
退出：部署结果
删除
已升级
输入：智能合约
执行：升级智能合约
退出：升级结果
升级/部署

图6

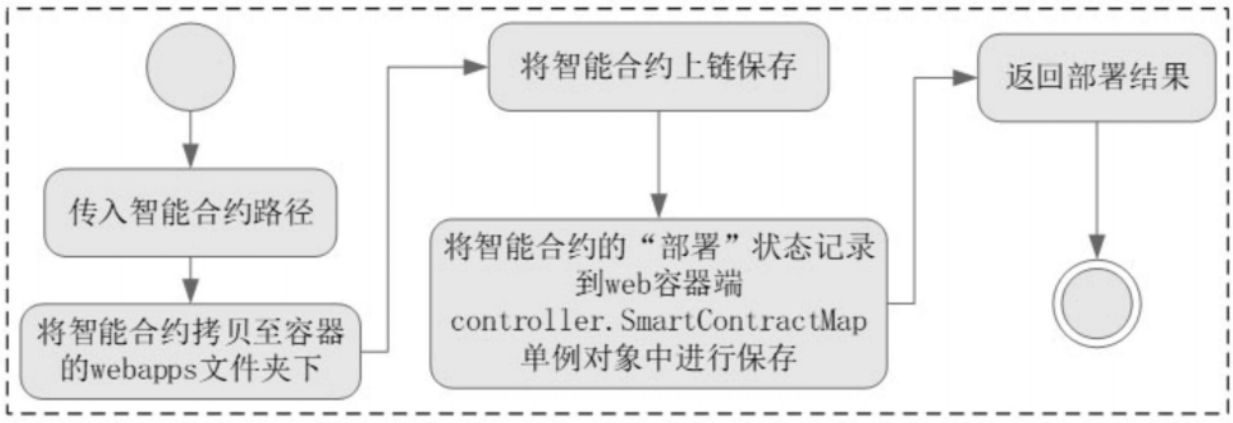

将智能合约上链保存
返回部署结果
传入智能合约路径
将智能合约的“部署”状态记录到web容器端controller.SmartContractMap单例对象中进行保存
将智能合约拷贝至容器的webapps文件夹下

图7

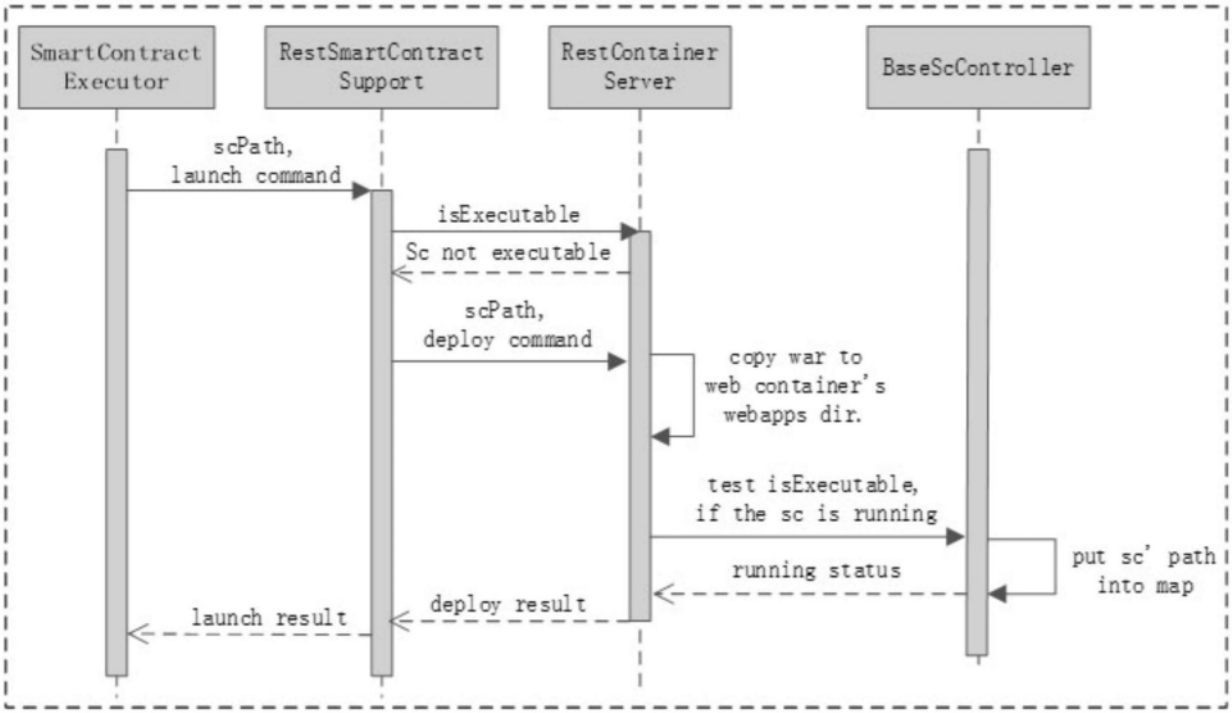

SmartContract Executor
RestSmartContract Support
RestContainer Server
BaseScController
scPath, launch command
isExecutable
Sc not executable
scPath, deploy command
copy war to web container's webapps dir.
test isExecutable, if the sc is running
running status
put sc' path into map
launch result
deploy result

图8

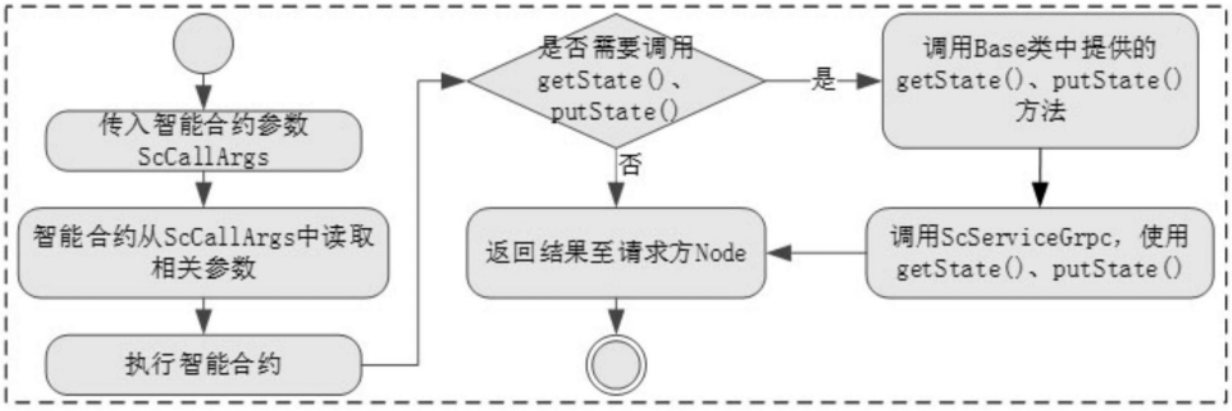

传入智能合约参数 ScCallArgs
智能合约从ScCallArgs中读取相关参数
执行智能合约
是否需要调用 getState()、putState()
是
否
调用Base类中提供的 getState()、putState()方法
调用ScServiceGrpc，使用 getState()、putState()
返回结果至请求方Node

图9

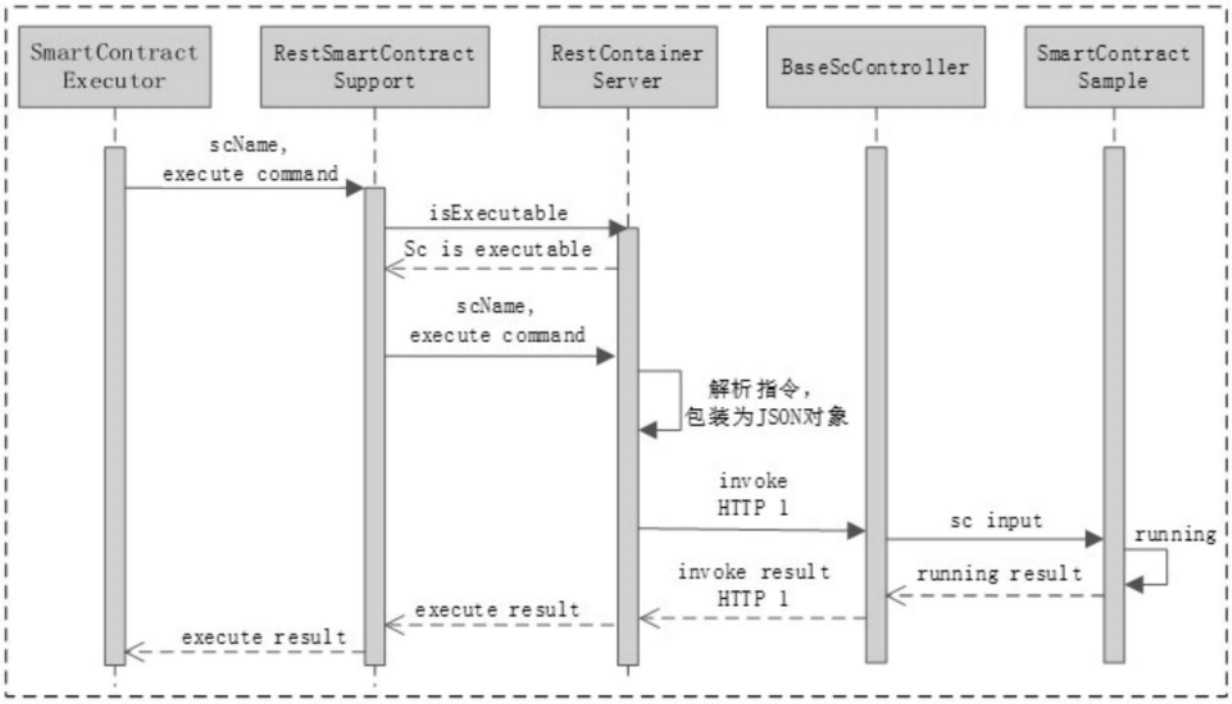

SmartContract Executor
RestSmartContract Support
RestContainer Server
BaseScController
SmartContract Sample
scName, execute command
isExecutable
Sc is executable
scName, execute command
解析指令，包装为JSON对象
invoke HTTP 1
sc input
running
invoke result HTTP 1
running result
execute result
execute result

图10