(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 109731334 B
(45)授权公告日 2020.08.11

(21)申请号 201811399616.5

(22)申请日 2018.11.22

(65)同一申请的已公布的文献号
申请公布号 CN 109731334 A

(43)申请公布日 2019.05.10

(73)专利权人 腾讯科技(深圳)有限公司
地址 518000 广东省深圳市南山区高新区科技中一路腾讯大厦35层

(72)发明人 陈明建

(74)专利代理机构 北京康信知识产权代理有限责任公司 11240
代理人 周婷婷 江舟

(51)Int.Cl.
A63F 13/60(2014.01)
A63F 13/822(2014.01)
G06F 8/71(2018.01)

(56)对比文件
CN 101957751 A,2011.01.26
CN 108650217 A,2018.10.12
CN 105656688 A,2016.06.08
US 8147339 B1,2012.04.03
陈明建.状态机的一种实现jc::fsm介绍(1):状态机的模型.《https://gameinstitute.qq.com/community/detail/101956》.腾讯游戏学院,2015,全文.
陈明建.状态机的一种实现jc::fsm介绍(2):简单应用及图形化工具介绍.《https://gameinstitute.qq.com/community/detail/101957》.2015,全文.

审查员 张静

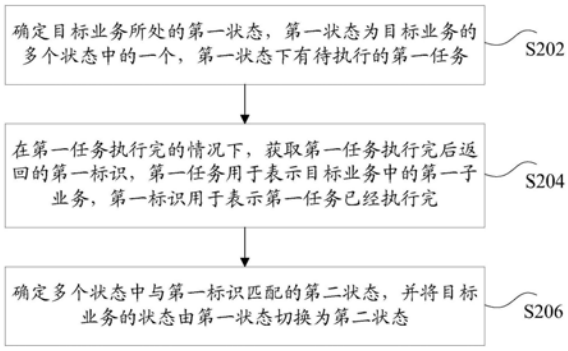
权利要求书3页 说明书18页 附图8页

(54)发明名称

状态的切换方法和装置、存储介质、电子装置

(57)摘要

本发明公开了一种状态的切换方法和装置、存储介质、电子装置。其中,该方法包括:确定目标业务所处的第一状态,其中,第一状态为目标业务的多个状态中的一个,第一状态下有待执行的第一任务;在第一任务执行完的情况下,获取第一任务执行完后返回的第一标识,其中,第一任务用于表示目标业务中的第一子业务,第一标识用于表示第一任务已经执行完;确定多个状态中与第一标识匹配的第二状态,并将目标业务的状态由第一状态切换为第二状态。本发明解决了相关技术中应用内状态机的状态流转的故障率较高的技术问题。



CN 109731334 B

1.一种状态的切换方法,其特征在于,包括:

确定目标业务所处的第一状态,其中,所述第一状态为所述目标业务的多个状态中的一个,所述第一状态下有待执行的第一任务,所述第一任务为多个;

在多个所述第一任务为并行执行的任务的情况下,在多个所述第一任务执行完的情况下,将多个所述第一任务中,最后一个任务执行完后返回的标识作为第一标识,其中,所述第一任务用于表示所述目标业务中的第一子业务,所述第一标识用于表示多个所述第一任务已经执行完;

确定所述多个状态中与所述第一标识匹配的第二状态,并将所述目标业务的状态由所述第一状态切换为所述第二状态。

2.根据权利要求1所述的方法,其特征在于,所述多个状态中的每个状态对应有一个用于触发进入该状态的触发标识,其中,确定所述多个状态中与所述第一标识匹配的第二状态,并将所述目标业务的状态由所述第一状态切换为所述第二状态包括:

从所述多个状态中获取触发标识与所述第一标识匹配的所述第二状态;

将所述目标业务的状态从所述第一状态切换为所述第二状态。

3.根据权利要求1或2所述的方法,其特征在于,在将所述目标业务的状态从所述第一状态切换为所述第二状态之前,所述方法还包括:

将第二标识作为用于触发进入所述第一状态的触发标识,并生成所述第一状态下的所述第一任务。

4.根据权利要求3所述的方法,其特征在于,将第二标识作为用于触发进入所述第一状态的触发标识,并生成所述第一状态下的所述第一任务包括:

获取配置文件,其中,所述配置文件是采用第一计算机语言编写的文件;

按照所述配置文件的指示将所述第二标识作为用于触发进入所述第一状态的触发标识,并生成所述第一状态下的所述第一任务。

5.根据权利要求4所述的方法,其特征在于,按照所述配置文件的指示将所述第二标识作为用于触发进入所述第一状态的触发标识,并生成所述第一状态下的所述第一任务包括:

利用所述配置文件生成任务文件,其中,所述任务文件是采用第二计算机语言编写的记录有所述第一状态下的所述第一任务和所述第一状态的所述第二标识的文件,所述第二计算机语言不同于所述第一计算机语言。

6.根据权利要求5所述的方法,其特征在于,利用所述配置文件生成任务文件包括:

在所述任务文件中将所述第一状态的名字创建为所述配置文件所指示的名字,并将所述第二标识作为用于触发进入所述第一状态的触发标识;

在所述任务文件中创建以所述第一状态为源状态、并以所述第二状态为目的状态的所述第一任务,其中,在所述任务文件中的所述第一任务为利用所述第二计算机语言编写的任务模板创建的;

在所述任务文件中为每个所述第一任务分配标识,其中,为所述第一任务分配的标识不同于任意一个已生成任务的标识。

7.根据权利要求1或2所述的方法,其特征在于,确定目标业务所处的第一状态包括:

基于驱动类型确定所述目标业务所处的所述第一状态,其中,所述驱动类型用于指示

是否存在任务执行完后返回的标识。

8.根据权利要求7所述的方法，其特征在于，基于驱动类型确定所述目标业务所处的所述第一状态包括：

根据所述目标业务的所述驱动类型确定切换后所在的所述第一状态，其中，所述驱动类型包括外部事件驱动和内部返回码驱动，所述目标业务由安装在终端上的业务应用提供，所述外部事件驱动用于指示在所述终端上对所述业务应用的操作，所述内部返回码驱动用于指示任务执行完后返回有标识。

9.根据权利要求8所述的方法，其特征在于，根据所述目标业务的所述驱动类型确定切换后所在的所述第一状态包括：

将所述多个状态中触发标识与所述外部事件驱动的标识匹配的状态作为所述第一状态;或，

获取第二任务执行完后返回的第三标识，将所述多个状态中触发标识与所述第三标识匹配的状态作为所述第一状态，其中，所述第二任务用于表示所述目标业务中的第二子业务，所述第三标识用于表示所述第二任务已经执行完。

10.根据权利要求7所述的方法，其特征在于，在确定目标业务所处的第一状态之后，所述方法还包括采用以下方式将所述目标业务的状态切换至所述第一状态：

在第三状态与所述第一状态为同一状态的情况下，保持所述目标业务所处的状态不变，并执行所述第一状态下的所述第一任务，其中，所述第三状态为切换为所述第一状态前所述目标业务所处的状态；

在所述第三状态与所述第一状态不同的情况下，通过执行对所述第三状态的退出任务和对所述第一状态的进入任务来将所述目标业务的状态由所述第三状态切换为所述第一状态，或通过执行对所述第三状态的退出任务和对所述第一状态的进入任务来将所述目标业务的状态由所述第三状态切换为所述第一状态，并执行所述第一状态下的所述第一任务。

11.一种状态的切换装置，其特征在于，包括：

确定单元，用于确定目标业务所处的第一状态，其中，所述第一状态为所述目标业务的多个状态中的一个，所述第一状态下有待执行的第一任务，所述第一任务为多个；

获取单元，用于在多个所述第一任务为并行执行的任务的情况下，在多个所述第一任务执行完的情况下，将多个所述第一任务中，最后一个任务执行完后返回的标识作为第一标识，其中，所述第一任务用于表示所述目标业务中的第一子业务，所述第一标识用于表示多个所述第一任务已经执行完；

切换单元，用于确定所述多个状态中与所述第一标识匹配的第二状态，并将所述目标业务的状态由所述第一状态切换为第二状态。

12.根据权利要求11所述的装置，其特征在于，所述多个状态中的每个状态对应有一个用于触发进入该状态的触发标识，其中，所述切换单元包括：

获取模块，用于从所述多个状态中获取所述触发标识与所述第一标识匹配的所述第二状态；

切换模块，用于将所述目标业务的状态从所述第一状态切换为所述第二状态。

13.一种存储介质，其特征在于，所述存储介质包括存储的程序，其中，所述程序运行时

执行上述权利要求1至10任一项中所述的方法。

14.一种电子装置，包括存储器、处理器及存储在所述存储器上并可在所述处理器上运行的计算机程序，其特征在于，所述处理器通过所述计算机程序执行上述权利要求1至10任一项中所述的方法。

状态的切换方法和装置、存储介质、电子装置

技术领域

[0001] 本发明涉及互联网领域，具体而言，涉及一种状态的切换方法和装置、存储介质、电子装置。

背景技术

[0002] 随着网络技术的飞速发展，应用产品的种类越来越多，常见的应用产品有游戏、购物、生活等类型，其中游戏类型的应用产品包括网络游戏和单机游戏等。在游戏中，游戏引擎直接控制剧情、关卡、美工、音乐、操作等内容，它扮演着发动机的角色，把游戏中的所有元素捆绑在一起，在后台指挥它们有序地工作，游戏引擎可控制所有游戏功能的主程序，从计算碰撞、物理系统和物体的相对位置，到接受玩家的输入，以及按照正确的音量输出声音等等。

[0003] 以控制剧情为例，可将虚拟角色的行为描述为：在行走时不可以攻击，在不行走时可以攻击。为了实现对该角色是否可以发起攻击这一动作的限定，通常的做法是采用基于堆栈嵌套的状态机实现，但是采用堆栈嵌套的机制会引起状态流转出现故障错误，从而影响了游戏逻辑的稳定性，类似地，在购物、生活等类型的应用中，也存在类似问题。

[0004] 针对上述的问题，目前尚未提出有效的解决方案。

发明内容

[0005] 本发明实施例提供了一种状态的切换方法和装置、存储介质、电子装置，以至少解决相关技术中应用内状态机的状态流转的故障率较高的技术问题。

[0006] 根据本发明实施例的一个方面，提供了一种状态的切换方法，包括：确定目标业务所处的第一状态，其中，第一状态为目标业务的多个状态中的一个，第一状态下有待执行的第一任务；在第一任务执行完的情况下，获取第一任务执行完后返回的第一标识，其中，第一任务用于表示目标业务中的第一子业务，第一标识用于表示第一任务已经执行完；确定多个状态中与第一标识匹配的第二状态，并将目标业务的状态由第一状态切换为第二状态。

[0007] 根据本发明实施例的另一方面，还提供了一种状态的切换装置，包括：确定单元，用于确定目标业务所处的第一状态，其中，第一状态为目标业务的多个状态中的一个，第一状态下有待执行的第一任务；获取单元，用于在第一任务执行完的情况下，获取第一任务执行完后返回的第一标识，其中，第一任务用于表示目标业务中的第一子业务，第一标识用于表示第一任务已经执行完；切换单元，用于确定多个状态中与第一标识匹配的第二状态，并将目标业务的状态由第一状态切换为第二状态。

[0008] 根据本发明实施例的另一方面，还提供了一种存储介质，该存储介质包括存储的程序，程序运行时执行上述的方法。

[0009] 根据本发明实施例的另一方面，还提供了一种电子装置，包括存储器、处理器及存储在存储器上并可在处理器上运行的计算机程序，处理器通过计算机程序执行上述的方

法。

[0010] 在本发明实施例中，确定目标业务所处的第一状态，其中，第一状态为目标业务的多个状态中的一个，第一状态下有待执行的第一任务；在第一任务执行完的情况下，获取第一任务执行完后返回的第一标识，其中，第一任务用于表示目标业务中的第一子业务，第一标识用于表示第一任务已经执行完；确定多个状态中与第一标识匹配的第二状态，并将目标业务的状态由第一状态切换为第二状态，可以解决相关技术中应用内状态机的状态流转的故障率较高的技术问题，进而达到提高应用内状态机的状态流转的准确率的技术效果。

附图说明

[0011] 此处所说明的附图用来提供对本发明的进一步理解，构成本申请的一部分，本发明的示意性实施例及其说明用于解释本发明，并不构成对本发明的不当限定。在附图中：

[0012] 图1是根据本发明实施例的状态的切换方法的硬件环境的示意图；

[0013] 图2是根据本发明实施例的一种可选的状态的切换方法的流程图；

[0014] 图3是根据本发明实施例的一种可选的状态机结构的示意图；

[0015] 图4是根据本发明实施例的一种可选的状态机结构的示意图；

[0016] 图5是根据本发明实施例的一种可选的状态机结构的示意图；

[0017] 图6是根据本发明实施例的一种可选的状态机结构的示意图；

[0018] 图7是根据本发明实施例的一种可选的状态机结构的示意图；

[0019] 图8是根据本发明实施例的一种可选的状态机结构的示意图；

[0020] 图9是根据本发明实施例的一种可选的状态机结构的示意图；

[0021] 图10是根据本发明实施例的一种可选的状态机结构的示意图；

[0022] 图11是根据本发明实施例的一种可选的状态的切换装置的示意图；

[0023] 以及

[0024] 图12是根据本发明实施例的一种终端的结构框图。

具体实施方式

[0025] 为了使本技术领域的人员更好地理解本发明方案，下面将结合本发明实施例中的附图，对本发明实施例中的技术方案进行清楚、完整地描述，显然，所描述的实施例仅仅是本发明一部分的实施例，而不是全部的实施例。基于本发明中的实施例，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，都应当属于本发明保护的范围。

[0026] 需要说明的是，本发明的说明书和权利要求书及上述附图中的术语“第一”、“第二”等是用于区别类似的对象，而不必用于描述特定的顺序或先后次序。应该理解这样使用的数据在适当情况下可以互换，以便这里描述的本发明的实施例能够以除了在这里图示或描述的那些以外的顺序实施。此外，术语“包括”和“具有”以及他们的任何变形，意图在于覆盖不排他的包含，例如，包含了一系列步骤或单元的过程、方法、系统、产品或设备不必限于清楚地列出的那些步骤或单元，而是可包括没有清楚地列出的或对于这些过程、方法、产品或设备固有的其它步骤或单元。

[0027] 首先，在对本发明实施例进行描述的过程中出现的部分名词或者术语适用于如下

解释：

[0028] 可扩展标记语言XML，外文名为Extensible Markup Language，是标准通用标记语言的子集，是一种用于标记电子文件使其具有结构性的标记语言。

[0029] 根据本发明实施例的一方面，提供了一种状态的切换方法的方法实施例。

[0030] 可选地，在本实施例中，上述状态的切换方法可以应用于如图1所示的由服务器101和/或用户终端103所构成的硬件环境中。如图1所示，服务器101通过网络与终端103进行连接，可用于为终端或终端上安装的客户端提供服务(如游戏服务、应用服务等)，可在服务器上或独立于服务器设置数据库105，用于为服务器101提供数据存储服务，上述网络包括但不限于：广域网、城域网或局域网，终端103并不限定于PC、手机、平板电脑等。

[0031] 本发明实施例的状态的切换方法可以由服务器101来执行，也可以由终端103来执行，还可以是由服务器101和终端103共同执行。其中，终端103执行本发明实施例的状态的切换方法也可以是由安装在其上的客户端来执行。

[0032] 图2是根据本发明实施例的一种可选的状态的切换方法的流程图，如图2所示，该方法可以包括以下步骤：

[0033] 步骤S202，服务器确定目标业务所处的第一状态，第一状态为目标业务的多个状态中的一个，第一状态下有待执行的第一任务。

[0034] 上述的目标业务为用于为用户提供某种服务的业务，如提供游戏服务、购物服务、地图服务、外卖服务等服务的业务，该目标业务可承载在业务应用上，换言之，用户可以通过该业务应用来获取目标业务，如通过游戏业务应用来获取游戏服务、通过外卖业务应用点外卖、通过地图业务服务应用来进行定位等。

[0035] 对于目标业务而言，可以在逻辑上将该业务逻辑分为若干个子逻辑(如该子逻辑可以是业务逻辑内的运算模块、目标业务中的子业务等)，每个子逻辑可以对应于一个状态，上述的多个状态即与多个子逻辑对应的状态，每个状态可以包括三个部分，用于进入该状态的进入动作(或称进入任务)、用于退出该状态的退出动作(或称退出任务)以及用于表示该状态下的子逻辑的逻辑动作(或称逻辑任务，如前述第一任务、下述第二任务等)。

[0036] 步骤S204，在第一任务执行完的情况下，服务器获取第一任务执行完后返回的第一标识，第一任务用于表示目标业务中的第一子业务，第一标识用于表示第一任务已经执行完。

[0037] 标识(包括上述第一标识、下述第二标识等)可为该任务的返回码，该返回码可用数字、英文字母、中文字符等表示，每个任务可对应有一个返回码，任意两个不同任务对应的返回码可不同或相同，并行执行的相同任务的返回码可不同；子业务相当于前述的子逻辑。

[0038] 步骤S206，服务器确定多个状态中与第一标识匹配的第二状态，并将目标业务的状态由第一状态切换为第二状态。相当于是根据状态机内部的返回码来进行状态转换，即内部返回码驱动(retcode_driven)，状态机中根据某个或一系列动作(action)的返回码，进行状态机流转的方式，例如，在任务为不可复用且每个任务的返回码不同的情况下，那么可以直接根据返回码确定下一个流转至的状态，在任务为可复用的情况下，可以从当前状态的可流转至的状态中查找与返回码匹配(如相同)的那个状态。

[0039] 相关技术中，参见图3，在通用状态机模型中，由于状态机只能通过外部事件驱动，

因此存在如下情况:在状态S1中,执行动作action1()函数(相当于一个任务,或称事件处理函数)的过程中,如果需要根据action1的执行情况进行迁移到状态S2,需要在action1()函数内调用类似fire_event()的函数接口触发事件event1,触发事件event1后,状态机会马上迁移到状态s2,并执行action2()动作(相当于另一个任务),但这时action1()还没有退出,导致状态机的action1()和action2()都处于执行状态,并且action2()函数是在action1()函数中执行,也就是action堆栈嵌套了,在这种情况下,当action2()函数退出时,会返回到action1()中,当返回到action1()时,这时状态机的状态已经是S2了,并且其他内部参数可能已经被改变,这时继续在action1()中执行代码就会带来严重的逻辑错误。

[0040] 而在本申请的技术方案中,在状态机中引入了内部返回码,图3的流程就变成图4所示,当在状态S1中,执行动作action1()的过程中,如果需要根据action1()的执行情况进行迁移到状态S2,则应当在action1()结束时,返回一个内部返回码retcode1,并退出action1(),状态机发现该action返回了retcode1,则通过内部的返回码驱动进行状态迁移到S2,然后执行action2()动作,action2()执行完毕后,如果不需要继续进行状态迁移,则返回到状态机的外部循环,继续等待外部事件的触发。也就是说,在状态机中,不允许在action执行过程中向状态机发送事件,而是通过action的返回码来告知状态机下一步动作。从而可以解决上述嵌套的问题。

[0041] 上述实施例以本发明实施例的状态的切换方法由服务器101来执行为例进行说明,本发明实施例的状态的切换方法也可以由终端103来执行,本发明实施例的状态的切换方法还可以是由服务器101和终端103共同执行,服务器执行步骤S202至步骤S206中的一个或两个步骤(如步骤S202至步骤S204),终端执行剩余步骤(如步骤S206)。其中,终端103执行本发明实施例的状态的切换方法也可以是由安装在其上的客户端(如上述业务应用的客户端)来执行。

[0042] 通过上述步骤S202至步骤S206,确定目标业务所处的第一状态,其中,第一状态为目标业务的多个状态中的一个,第一状态下有待执行的第一任务;在第一任务执行完的情况下,获取第一任务执行完后返回的第一标识,其中,第一任务用于表示目标业务中的第一子业务,第一标识用于表示第一任务已经执行完;确定多个状态中与第一标识匹配的第二状态,并将目标业务的状态由第一状态切换为第二状态,可以解决相关技术中应用内状态机的状态流转的故障率较高的技术问题,进而达到提高应用内状态机的状态流转的准确率的技术效果。

[0043] 在执行本申请的技术方案之前,可以将第二标识作为用于触发进入第一状态的触发标识,并生成第一状态下的第一任务。

[0044] 可选地,将第二标识作为用于触发进入第一状态的触发标识,并生成第一状态下的第一任务包括:获取配置文件,配置文件是采用第一计算机语言编写的文件;按照配置文件的指示将第二标识作为用于触发进入第一状态的触发标识,并生成第一状态下的第一任务。

[0045] 在上述实施例中,按照配置文件的指示将第二标识作为用于触发进入第一状态的触发标识,并生成第一状态下的第一任务可包括:利用配置文件生成任务文件,任务文件是采用第二计算机语言记录有第一状态下的第一任务和第一状态的第二标识的文件,第二计

算机语言不同于第一计算机语言。

[0046] 规则定义文件(配置文件)的第一计算机语言可采用xml,不过只要是基于规则定义元素就可以实现,也可以采用其他语言来定义,比如json来定义,本状态机系统中,状态机模型是本申请的最重要部分之一,可以利用第二计算机语言实现,如利用xml实现的第二计算机语言C++代码的自动生成,但是根据需要也可以根据规则定义文件生成其他类型的第二计算机语言的状态机代码,比如python、C#、C等。

[0047] 可选地,利用配置文件生成任务文件包括:

[0048] 1)在任务文件中将第一状态的名字创建为配置文件所指示的名字,并将第二标识作为用于触发进入第一状态的触发标识,例如,在xml文件中利用status name＝“s_init”为第一状态命名,在状态机的c++源文件中第一状态机的名字为“s_init”;

[0049] 2)在任务文件中创建以第一状态为源状态、并以第二状态为目的状态的第一任务,其中,在任务文件中的第一任务为利用第二计算机语言编写的任务模板创建的,例如,在xml文件利用rule from＝“s_init”指定源状态为“s_init”,利用sub_rule event＝“EVT_DROP”to＝“s_carried”来指定在事件“EVT_DROP”的驱动下,会从源状态“s_init”(即第一状态)转换为目的状态“s_carried”(即第二状态),利用action＝“on_pickup”来指定第一任务“on_pickup”(也可称为动作);

[0050] 3)在任务文件中为每个第一任务分配标识,其中,为第一任务分配的标识不同于任意一个已生成任务的标识,在xml文件可利用args＝“arg1＝value1”来为第一任务分配的任务标识arg1分配相应的任务标识的取值value1。

[0051] 在上述实施例中,通过配置文件提供了一套基于xml的状态机HFsm的配置规则,并根据该配置规则实现HFsm的图形可视化展示;实现了状态机逻辑代码的自动代码生成,根据HFsm配置规则自动生成C++代码;上述的配置文件可动态加载,保证了服务的不中断,同时也可实现服务的更新,实现了动态加载器,可动态加载HFsm配置文件,动态运行;在本申请的状态机运行的过程中,可以通过状态机中的日志模块实时记载状态机的工作日志,以便于在出错时进行纠错,从而实现了一套基于C++代码的HFsm日志系统,用于HFsm运行时日志分析。

[0052] 在步骤S202提供的技术方案中,服务器确定目标业务所处的第一状态,第一状态为目标业务的多个状态中的一个,第一状态下有待执行的第一任务。

[0053] 可选地,在确定目标业务所处的第一状态时,可基于驱动类型确定目标业务所处的第一状态,驱动类型用于指示是否存在任务执行完后返回的标识,若是则为内部返回码驱动,若不是,则是外部事件驱动。

[0054] 上述实施例中,基于驱动类型确定目标业务所处的第一状态包括:根据目标业务的驱动类型确定切换后所在的第一状态,驱动类型包括外部事件驱动和内部返回码驱动,目标业务由安装在终端上的业务应用提供,外部事件驱动为当在终端上存在对业务应用的操作这一外部事件的情况下的驱动类型,内部返回码驱动为获取到前一任务执行完后返回的标识的情况下的驱动类型。

[0055] 可选地,根据目标业务的驱动类型确定切换后所在的第一状态包括:将源状态的可流转至的多个状态中触发标识与外部事件驱动的标识匹配的状态作为第一状态;获取第二任务执行完后返回的第三标识,将多个状态中触发标识与第三标识匹配的状态作为第一

状态,第二任务用于表示目标业务中的第二子业务,第三标识用于表示第二任务已经执行完。

[0056] 无论是对于外部事件驱动还是内部返回码驱动,在确定目标业务所处的第一状态之后,可采用以下方式之一将目标业务的状态切换至第一状态:在第三状态与第一状态为同一状态的情况下,即ReEnter方式,保持目标业务所处的状态不变,并执行第一状态下的第一任务,第三状态为切换为第一状态前目标业务所处的状态;在第三状态与第一状态不同的情况下,即Non-ReEnter方式,通过执行对第三状态的退出任务和对第一状态的进入任务来将目标业务的状态由第三状态切换为第一状态;在第三状态与第一状态不同的情况下,通过执行对第三状态的退出任务和对第一状态的进入任务来将目标业务的状态由第三状态切换为第一状态,并执行第一状态下的第一任务。

[0057] 在上述方案中,扩展了状态机模型,设计了内部返回码驱动(retcode_driven)方式,实现了HFsm的自驱动(self_driven),并解决了传统状态机模型仅依赖外部事件驱动带来的状态机堆栈嵌套问题。

[0058] 在步骤S204提供的技术方案中,在第一任务执行完的情况下,服务器获取第一任务执行完后返回的第一标识,第一任务用于表示目标业务中的第一子业务,第一标识用于表示第一任务已经执行完。

[0059] 上述的第一任务可以为一个或多个,若第一任务为多个,则在获取第一任务执行完后返回的第一标识时,可在多个第一任务为并行执行的任务的情况下,获取多个第一任务中位于目标位置的第一任务返回的标识为第一标识,如位置为最后一个第一任务返回的标识,该目标位置还可以为第一位、中间等位置,具体可以根据实际情况确认。从而实现了并行动作序列(parallel-action),以实现游戏玩法中的并行操作需求。

[0060] 在步骤S206提供的技术方案中,服务器确定多个状态中与第一标识匹配的第二状态,并将目标业务的状态由第一状态切换为第二状态。

[0061] 可选地,多个状态中的每个状态对应有一个用于触发进入该状态的触发标识,其中,确定多个状态中与第一标识匹配的第二状态,并将目标业务的状态由第一状态切换为第二状态包括:从多个状态中获取触发标识与第一标识匹配(如触发标识与第一标识相同)的第二状态;将目标业务的状态从第一状态切换为第二状态。

[0062] 本状态机系统(HFsm)主要是为了解决相关技术中状态机系统存在的不足而设计的,目标是让其在后台分布式系统开发以及游戏引擎核心模式玩法开发中得到有效应用。HFsm主要在如下方面进行了改进:扩展了传统状态机模型,设计了内部返回码驱动(retcode_driven)方式,实现了HFsm的自驱动(self_driven),并解决了相关技术中状态机模型仅依赖外部事件驱动带来的状态机堆栈嵌套问题;实现了并行动作序列(parallel-action),以实现游戏玩法中的并行操作需求;实现了基于状态(status)和动作(action)的上下文(context)机制,用于支持action局部参数的动态配置;设计并实现了一套基于xml的HFsm配置规则,并根据该配置规则实现HFsm的图形可视化展示;实现了自动代码生成器,根据HFsm配置规则自动生成C++代码;实现了动态加载器,可动态加载HFsm配置文件,动态运行;实现了一套基于C++代码的HFsm日志系统,用于HFsm运行时日志分析。

[0063] 作为一种可选的实施例,下面以利用本申请的技术方案实现状态机为例详述本申请的技术方案,后续以HFsm作为本有限状态机系统的名称,以下涉及本状态机的系统可用

HFsm来指代。

[0064] 在一个可选的技术方案中，提供了一种通用有限状态机，一般可以用如下表1所示的状态转移表来描述：

[0065] 表1

[0066]

状态 / 条件	状态 A	状态 B	状态 C
条件 1	…	…	…
条件 2	…	状态 C	…
条件 3	…	…	…

[0067] 状态(Status)：状态节点，包含进入动作和退出动作。

[0068] 动作(action)：包括输入动作和转移动作两种类型，输入动作依赖当前状态和输入条件，转移动作在状态转移时进行。

[0069] 该模型没有对输入条件进行细分，在代码实现上容易出现堆栈嵌套，从而引发一些bug；该模型中没有解决并行action节点的问题，也没有提供action参数配置的解决方案；该模型动作分为输入动作和转移动作，导致在实际应用时，较复杂。

[0070] 在又一个可选的技术方案中，提供了一种采用本申请技术方案实现的状态机HFsm，HFsm对输入条件进行了细分，定义了外部事件驱动和内部返回码驱动方式，让状态机有了自驱动的能力，解决了相关技术中模型实现时存在的堆栈嵌套问题；引入动作并行运行parallel-action，满足了并行处理的需求；重新定义了动作action和状态转移的关系，简化了动作action的应用情形；同时提供了参数配置的扩展能力，便于在开发中动态调整状态机参数；HFsm还通过xml定义了状态机配置规则，提供了自动生成C++代码和动态加载xml文件执行状态机的能力。

[0071] 在游戏中，可大量地使用HFsm来实现游戏内核心玩法的逻辑，比如在爆破模式中，炸药包的配置文件“SDBombFsm.xml”部分配置如下：

```
<status_set start = "s_init" end = "s_removed">
    <status name = "s_init"         entry = ""  exit = "" />
    <status name = "s_dropped"      entry = "entry_dropped"     exit = "" />
    <status name = "s_carried"      entry = ""  exit = "" />
    <status name = "s_planting"     entry = ""  exit = "exit_planting" />
    <status name = "s_plantted"     entry = ""  exit = "" />
    <status name = "s_defusing"     entry = ""  exit = "exit_defusing" />
    <status name = "s_defused"      entry = ""  exit = "" />
    <status name = "s_exploded"     entry = "entry_exploded"    exit = "" />
    <status name = "s_removed"      entry = "entry_remove"  exit = "" />
</status_set>

<rule_set>
    <rule from = "s_init">
        <sub_rule event = "EVT_DROP" to = "s_dropped" />
    </rule>

    <rule from = "s_dropped">
        <sub_rule event = "EVT_PICKUP" to = "s_carried" action = "on_pickup" />
    </rule>

    <rule from = "s_carried">
        <sub_rule event = "EVT_DROP" to = "s_dropped" />
        <sub_rule event = "EVT_PLANT" to = "s_planting" action = "on_start_plant"/>
    </rule>

    <rule from = "s_planting">
        <sub_rule event = "EVT_DROP" to = "s_dropped" />
        <sub_rule event = "EVT_INTERRUPT" to = "s_carried" action = "on_plant_interrupt"/>
        <sub_rule event = "EVT_TIMEOUT" to = "s_plantted" action = "on_plantted" />
    </rule>

    <rule from = "s_plantted">
        <sub_rule event = "EVT_DEFUSE" to = "s_defusing" action = "on_start_defuse" />
        <sub_rule event = "EVT_EXPLODE" to = "s_exploded" />
    </rule>

    <rule from = "s_defusing">
        <sub_rule event = "EVT_INTERRUPT" to = "s_plantted" action = "on_defuse_interrupt" ,
        <sub_rule event = "EVT_EXPLODE" to = "s_exploded" />
        <sub_rule event = "EVT_TIMEOUT" to = "s_defused" action = "on_defused" />
    </rule>

    <rule event = "EVT_REMOVE" to = "s_removed" >
        <sub_rule from = "*" />
    </rule>

</rule_set>
```

[0072]

[0073] 通过该配置文件,生成的可视化状态机SDBombFsm如图5所示(仅仅示出了SDBombFsm的部分),状态机包含五个基本元素:状态节点Status(如“s_init”、“s_dropped”、“s_removed”等),外部事件Event(如“EVT_DROP”、“EVT_REMOVE”等),内部返回码RetCode,动作Action(如“on_pickup”等),参数args以及由这五元素共同组成的转移规则(Rule)。

[0074] 如rule from=“s_init”表示状态节点s_init的转移规则,sub_rule event=“EVT_DROP”to“s_dropped”表示在外部事件EVT_DROP的触发下转移至状态s_dropped,其余状态节点的转移方式与此类似,不再赘述。

[0075] 下面结合图6所示的状态机HFsmExample来说明HFsm中涉及的技术细节的实现方案(如HFsm的状态模型定义)。图6所示的状态机HFsmExample可以利用如下所示的配置文件生成。

[0076]

```
<status_set start = "s_s0" end = "s_s3">
    <status name = "s_s0"          entry = "entry_s0"      exit = "exit_s0" />
    <status name = "s_s1"          entry = "entry_s1"      exit = "exit_s1" />
    <status name = "s_s2"          entry = "entry_s2"      exit = "exit_s2" />
    <status name = "s_s3"          entry = "entry_s3"      exit = "exit_s3" />
</status_set>

<rule_set>
  <rule from = "s_s0">
    <sub_rule event = "hfsm::EVT_REQ0" to = "s_s0" action = "do_action1" args = "key1=value1"/>
    <sub_rule event = "hfsm::EVT_REQ1" to = "s_s1" action = "do_action2;do_action3" args = "key1=value1"/>
    <sub_rule retcode = "do_action1.hfsm::RET_CODE2" to = "s_s2" args = "key3=value3"/>
  </rule>

  <rule from = "s_s1">
    <sub_rule event = "hfsm::EVT_REQ2" to = "s_s3"/>
    <sub_rule retcode = "do_action2;do_action3.hfsm::RET_CODE3" to = "s_s3" args = "key1=value2"/>
  </rule>

  <rule from = "*">
    <sub_rule retcode = "*.hfsm::RET_FATAL" to = "s_s3" action = "do_action4" args = "reason=ret_fatal" />
  </rule>
</rule_set>
```

[0077] 如图6所示的HFsmExample的状态节点s_s0：

[0078] s_s0:状态的名称,也是规则定义文件中的状态名；

[0079] entry_s0():非ReEnter方式进入到s_s0状态时,执行的进入动作：

[0080] exit_s0():非ReEenter方式退出s_s0状态时,执行的退出动作；

[0081] do_action2()、do_action3():并行动作序列,可以最后一个动作序列的返回码作为动作序列的返回码,该动作序列可在状态迁移完后执行,并且HFsm提供类似get_retcodes()的接口,让每个action可以获得当前序列中已执行action的返回值。

[0082] Args:根据状态和当前action共同定义的一组参数值,可以在action中通过参数名获取。

[0083] 图6中红色带箭头线(即图6中箭头401)表示外部事件驱动,“hfsm::EVT_*”(“*”表示任意字符)表示事件名称;图6中蓝色带箭头(即图6中箭头403)先表示内部返回码驱动,“hfsm::RET_*”表示返回码名称;图6中红色边框的(即图6中边框405)状态节点,其中的状态名为“*”,状态节点“*”代表了状态机中任意的节点;图6中绿色的菱形块(即图6中菱形块407)表示action序列,每个action序列节点可用一条虚线跟某个状态节点status相连,表示该action序列节点是属于该status节点。

[0084] hfsm_tool根据配置文件,生成状态机类HFsmExample,并生成规则类THFsmExampleRule,和action接口类IHFsmExampleActions,并定义变量THFsmExampleRules_rule,保存状态机的规则；

[0085] 针对status_set中的状态定义,通过HFsm的接口add_status生成状态代码,比如生成如下代码：

[0086] s_rule.add_status(″s_s0″,″entry_s0″,″exit_sC″)；

[0087] s_rule.add_status(″s_s1″,″entry_s1″,″exit_s1″)；

[0088] s_rule.add_status(″s_s2″,″entry_s2″,″exit_s2″)；

[0089] s_rule.add_status(″s_s3″,″ertry_s3″,″exit_s3″)；

[0090] hfsm_tool根据配置文件解析出所有action函数,通过add_action往状态机中添加action信息,生成如下代码：

[0091] s_rule.add_action(NULL,″NULL″)；

[0092] s_rule.add_action(&IHFsmExampleActions::entry_s0,″entry_s0″)；

[0093] s_rule.add_action(&IHFsmExampleActions::exit_s0,"exit_s0");

[0094] s_rule.add_action(&IHFsmExampleActions::entry_s1,"entry_s1");

[0095] s_rule.add_action(&IHFsmExampleActions::exit_s1,"exit_s1");

[0096] s_rule.add_action(&IHFsmExampleActions::entry_s2,"entry_s2");

[0097] s_rule.add_action(&IHFsmExampleActions::exit_s2,"exit_s2");

[0098] s_rule.add_action(&IHFsmExampleActions::entry_s3,"entry_s3");

[0099] s_rule,add_action(&IHFsmExampleActions::exit_s3,"exit_s3");

[0100] s_rule.add_action(&IHFsmExampleActions::do_action1,"do_action1");

[0101] s_rule.add_action(&IHFsmExampleActions::do_action2,"do_action2");

[0102] s_rule.add_action(&IHFsmExampleActions::do_action3,"do_action3");

[0103] s_rule.add_action(&IHFsmExampleActions::do_action4,"do_action4");

[0104] hfsm_tool根据配置文件解析出事件驱动规则,通过event_add接口添加事件驱动代码,比如配置文件中的EVT_REQ0和EVT_REQ1,生成如下代码:

[0105] s_rule.event_add("s_s0","s_s0",hfsm::EVT_REQ0,"do_action1","hfsm::EVT_REQ0");

[0106] s_rule.event_add("s_s0","s_s1",hfsm::EVT_REQ1,"do_aotion2;do_action3","hfsm::EVT_REQ1");

[0107] hfsm_tool根据配置文件解析出内部状态码驱动规则,通过code_add接口添加内部返回码驱动代码,比如配置文件中的RET_CODE2和RET_CODE3,生成如下代码:

[0108] s_rule.code_add("s_s0","s_s2",hfsm::RET_CODE2,"NULL","hfsm::RET_CODE2");

[0109] s_rule.code_add("s_s1","s_s3",hfsm::RET_CODE3,"NULL","hfsm::RET_CODE3");

[0110] hfsm_tool根据配置文件解析出状态机参数规则,通过set_action_arg接口添加参数代码,以上配置文件生成如下代码:

[0111] s_rule.set_action_arg("s_s2","s_s3","do_action4","reason","ret_fatal");

[0112] s_rule.set_action_arg("s_s3","s_s3","do_action4","reason","ret_fatal");

[0113] s_rule.set_action_arg("s_s0","s_s1","do_action3","key1","value1");

[0114] s_rule.set_action_arg("s_s0","s_s0","do_action1","key1","value1");

[0115] s_rule.set_action_arg("s_s0","s_s1","do_action2","key1","value1");

[0116] s_rule.set_action_arg("s_s0","s_s3","do_action4","reason","ret_fatal");

[0117] s_rule.set_action_arg("s_s1","s_s3","do_action4","reason","ret_fatal");

[0118] 这些代码就完成HFsmExample的核心实现。

[0119] 下面结合图7所示的状态机“HFsmExampleEvent”和图8所示的状态机“HFsmExampleRetCode”详述状态迁移方案的实现,包括外部事件驱动方式event_driven和

内部返回码驱动方式retcode_driven。

[0120] 外部事件驱动方式event_driven,可分两种情况,ReEnter和Non-ReEnter两种方式,如图7所示,由外部事件hfsm::EVT_REQ0驱动的状态迁移是ReEnter方式,这种方式下,HFsm的内部实现认为当前状态由s_s0以ReEnter方式迁移到s_s0中,这时entry_s0()和exit_s0()两个动作都不被执行,仅执行do_action1()动作,从HFsm模型来看,相当于当前状态不做迁移,仅执行do_action1()动作。

[0121] 由外部事件hfsm::EVT_REQ1驱动的状态迁移是Non-ReEnter方式,这种方式下,HFsm的执行流程是:hfsm::EVT_REQ1事件被触发时,s_s0的退出动作exit_s0()先被执行,然后状态迁移到s_s1中,并执行s_s1的进入动作entry_s1(),然后执行do_action2()、do_action3()动作序列。

[0122] 由外部事件hfsm::EVT_REQ2驱动的状态迁移也是Non-ReEnter方式,这种方式下,HFsm的执行流程是:hfsm::EVT_REQ2事件被触发时,s_s1的退出动作exit_s1()先被执行,然后状态迁移到s_s2中,并执行s_s2的进入动作entry_s2(),然后HFsm停留在s_s2状态,等待下一个外部事件。

[0123] 内部返回码驱动方式retcode_driven是为了实现执行action后,根据执行结果进行流程选择的需求,理论上来说,在某个action处理的结尾在向状态机发送外部事件,也可以达到这种目的,但是这种方式把外部驱动和内部驱动混为一谈,同时这种方式使用不当时,比如在action中间出发事件,将导致状态机调用栈嵌套,状态流转出现故障问题,因此HFsm中引入了retcode_driven方式。

[0124] retcode_driven理论上也可分两种情况ReEnter和Non-ReEnter两种方式,不过在实际使用HFsm时,一般主要使用Non-ReEnter方式,如图8所示,该状态机由外部事件hfsm::EVT_REQ1驱动执行do_action2()、do_action3()序列后,如果该序列的返回码为hfsm::RET_CODE3,则HFsm执行s_s0的退出动作exit_s0(),然后状态迁移到s_s3,并执行其进入动作entry_s3();如果该序列的返回码为hfsm::RET_CODE4,则HFsm执行s_s0的退出动作exit_s0(),然后状态迁移到s_s2,并执行其进入动作entry_s2(),然后再执行do_action4()动作;如果该序列的返回码为其他值(图8中未示出),那么HFsm将保持在s_s0状态,不进行状态迁移,也不执行s_s0的进入和退出动作,而从HFsm内部实现来说,这种情况属于retcode_driven的ReEnter方式。

[0125] 下面结合如图9所示状态机“HFsmExampleAny”详述任意状态驱动的处理。在HFsm的规则定义文件(即配置文件)中,可以“*”来表示任意状态,图9中红框(图9中以701示出)即表示任意状态。图9中红色箭头线(图9中以703示出)表示,在任意状态下,如果有外部事件hfsm::EVT_QUIT到来,则执行原状态的退出动作,然后迁移到s_s1状态,并执行s_s1的进入动作entry_s1(),然后执行do_action1动作。

[0126] 图9中蓝色箭头线(图9中以705示出)表示,在任意状态下,如果某个action序列的返回码是Hfsm::RET_ERROR,则执行原状态的退出动作,然后迁移到s_s2状态,并执行s_s2的进入动作entry_s2(),然后执行do_action2动作。

[0127] 下面结合图10所示的状态机“HFsmExampleAction”详述并行动作序列(parallel-action)的实现方案。

[0128] 在实际的项目需求中,在某些触发条件下,需要并行执行多个action,为了复用已

有action,并满足这类需求,HFsm引入了动作序列,如图10所示的hfsm::EVT_REQ0事件触发后,执行了s_s1下的do_action1()、do_action2()、do_action3()动作序列,HFsm中提供了接口get_retcodes(),该接口返回了当前action序列中已执行action的返回码,每个action中可以调用该接口获得返回码列表,并做相应的动作,在action序列中,最后一个action的返回码可作为该序列的最终返回码,HFsm把这个返回码作为retcode_driven的返回码。

[0129] 下面结合具体配置文件详述基于状态和动的上下文机制的实现方案,为了满足在规则定义中配置外部参数的需求,HFsm实现了基于status-action的context机制,规则定义文件配置方式如下:

[0130]
```
<rule from = "s_s0">
  <sub_rule event = "hfsm::EVT_REQ0" to = "s_s1"
            action = "do_action1;do_action2;do_action3;"
            args = "arg1=value1;arg2=value2"/>

  <sub_rule event = "hfsm::EVT_REQ1" to = "s_s2"
            action = "do_action1"
            args = "arg1=value3"/>
```

[0131] 在每条sub_rule中,可通过args=“key1=value1;key2=value2”的方式配置指定sub_rule的内容context,不同的sub_rule间相同key可以有不同的value,如所示,在同一个do_action1()中,当不同的sub_rule生效时,取得的key1值将不一样。

[0132] 实现方案中,context以源状态(status_from)、目的状态(status_to),动作(action),参数名(arg)为key建立关联表,并在自动生成代码时,生成该关联表的创建代码。在HFsm运行过程中,在某个action中需要获得某个参数(arg)的值,通过类似get_action_arg(arg_name)接口获取对应的arg的值,这个接口仅需要提供参数名(arg_name),不需要提供其他几个key(status_from,status_to和action),这几个key由HFsm根据当前运行状态来生成,简化了context的使用。

[0133] 下面结合具体配置文件详述HFsm规则定义的实现方案,以HFsmExample的规则定义文件为例:

[0134]
```
<status_set start = "s_s0" end = "s_s3">
    <status name = "s_s0"          entry = "entry_s0"      exit = "exit_s0" />
    <status name = "s_s1"          entry = "entry_s1"      exit = "exit_s1" />
    <status name = "s_s2"          entry = "entry_s2"      exit = "exit_s2" />
    <status name = "s_s3"          entry = "entry_s3"      exit = "exit_s3" />
</status_set>

<rule_set>
  <rule from = "s_s0">
    <sub_rule event = "hfsm::EVT_REQ0" to = "s_s0" action = "do_action1" args = "key1=value1"/>
    <sub_rule event = "hfsm::EVT_REQ1" to = "s_s1" action = "do_action2;do_action3" args = "key1=value1"/>
    <sub_rule retcode = "do_action1.hfsm::RET_CODE2" to = "s_s2" args = "key3=value3"/>
  </rule>

  <rule from = "s_s1">
    <sub_rule event = "hfsm::EVT_REQ2" to = "s_s3"/>
    <sub_rule retcode = "do_action2;do_action3.hfsm::RET_CODE3" to = "s_s3" args = "key1=value2"/>
  </rule>

  <rule from = "*">
    <sub_rule retcode = "*.hfsm::RET_FATAL" to = "s_s3" action = "do_action4" args = "reason=ret_fatal" />
  </rule>
</rule_set>
```

[0135] HFsm的规则定义文件可用来定义HFsm规则,并作为后续自动生成代码和可视化图形的基础。以HFsmExmaple为例,HFsm的规则定义文件内容主要由desc_info、status_set、rule_set三部分组成。

[0136] 其中desc_info中定义了HFsm生成代码对应的C++类名class_name和名字空间namespace,同时说明了该HFsm中使用到的符号文件symbol_files;status_set定义了HFsm中涉及到的所有状态节点信息,每个状态节点信息需要定义name,entry和exit,其中name是必须的,entry和exit是可选的,如果没有entry和exit,则配置为空字符串;rule_set定义了HFsm的状态迁移规则、action序列,以及context参数信息,rule_set可由多个rule组成,每个rule由多个sub_rule组成,在HFsm中,每个rule由源状态from,目标状态to,动作action,参数args以及事件event或返回码retcode(二者中的一个)构成,所有rule可以根据这个五个元素进行组合。

[0137] 关于定义状态,可以用通配符“*”表示任意状态,以此简化某些对所有状态均生效的规则。在HFsm代码的自动生成方案中,HFsm可自动生成的代码是C++代码,代码生成器可由python编写,通过模板代码和规则定义配置相结合生成最终代码,HFsm的C++实现中组要包含ihfsm、hfsm_context、hfsm_rule三个类。

[0138] ihfsm定义了一系列的虚接口,hfsm_context用来管理HFsm状态迁移的信息,以及hfsm运行过程的跟踪信息,hfsm_rule定了状态机的规则信息和基于action的参数配置信息。

[0139] 在HFsm调试及日志跟踪时,在HFsm的实现方案中,在hfsm_context中记录了HFsm的关键运行过程,在HFsm执行过程中,在状态迁移、action执行,retcode、event驱动等关键事件发生时,HFsm自动把这些信息记录在hfsm_context中,并在提供了接口供使用者访问这些信息;同时,基于性能的考虑,HFsm提供了接口,允许关闭日志跟踪功能。

[0140] 采用本申请的技术方案,可将本HFsm在CODM手游等应用,降低了手游游戏玩法设计的复杂度,提升了玩法设计效率,本系统提供的代码生成器,减少了代码开发的工作量,同时让开发人员从代码细节中摆脱出来,可以更专注于玩法的设计,动态加载器提供了类似脚本的功能,修改HFsm规则后,无需编译代码,即可运行新的玩法,实现了玩法的在线不停服更新,基于HFsm配置文件的图形可视化展示,提供了直观的展示方式,便于服务器开发人员和前台设计人员的交流,同时结合HFsm日志系统,便于定位问题。

[0141] 需要说明的是,对于前述的各方法实施例,为了简单描述,故将其都表述为一系列的动作组合,但是本领域技术人员应该知悉,本发明并不受所描述的动作顺序的限制,因为依据本发明,某些步骤可以采用其他顺序或者同时进行。其次,本领域技术人员也应该知悉,说明书中所描述的实施例均属于优选实施例,所涉及的动作和模块并不一定是本发明所必须的。

[0142] 通过以上的实施方式的描述,本领域的技术人员可以清楚地了解到根据上述实施例的方法可借助软件加必需的通用硬件平台的方式来实现,当然也可以通过硬件,但很多情况下前者是更佳的实施方式。基于这样的理解,本发明的技术方案本质上或者说对现有技术做出贡献的部分可以以软件产品的形式体现出来,该计算机软件产品存储在一个存储介质(如ROM/RAM、磁碟、光盘)中,包括若干指令用以使得一台终端设备(可以是手机,计算机,服务器,或者网络设备等)执行本发明各个实施例所述的方法。

[0143] 根据本发明实施例的另一个方面,还提供了一种用于实施上述状态的切换方法的状态的切换装置。图11是根据本发明实施例的一种可选的状态的切换装置的示意图,如图11所示,该装置可以包括:确定单元901、获取单元903以及切换单元905。

[0144] 确定单元901，用于确定目标业务所处的第一状态，其中，第一状态为目标业务的多个状态中的一个，第一状态下有待执行的第一任务。

[0145] 获取单元903，用于在第一任务执行完的情况下，获取第一任务执行完后返回的第一标识，其中，第一任务用于表示目标业务中的第一子业务，第一标识用于表示第一任务已经执行完。

[0146] 切换单元905，用于确定多个状态中与第一标识匹配的第二状态，并将目标业务的状态由第一状态切换为第二状态。

[0147] 需要说明的是，该实施例中的确定单元901可以用于执行本申请实施例中的步骤S202，该实施例中的获取单元903可以用于执行本申请实施例中的步骤S204，该实施例中的切换单元905可以用于执行本申请实施例中的步骤S206。

[0148] 此处需要说明的是，上述模块与对应的步骤所实现的示例和应用场景相同，但不限于上述实施例所公开的内容。需要说明的是，上述模块作为装置的一部分可以运行在如图1所示的硬件环境中，可以通过软件实现，也可以通过硬件实现。

[0149] 通过上述模块，确定目标业务所处的第一状态，其中，第一状态为目标业务的多个状态中的一个，第一状态下有待执行的第一任务；在第一任务执行完的情况下，获取第一任务执行完后返回的第一标识，其中，第一任务用于表示目标业务中的第一子业务，第一标识用于表示第一任务已经执行完；确定多个状态中与第一标识匹配的第二状态，并将目标业务的状态由第一状态切换为第二状态，可以解决相关技术中应用内状态机的状态流转的故障率较高的技术问题，进而达到提高应用内状态机的状态流转的准确率的技术效果。

[0150] 可选地，多个状态中的每个状态对应有一个用于触发进入该状态的触发标识，其中，切换单元可包括：获取模块，用于从多个状态中获取触发标识与第一标识匹配的第二状态；切换模块，用于将目标业务的状态从第一状态切换为第二状态。

[0151] 可选地，本申请的装置还可包括：生成单元，用于在将目标业务的状态从第一状态切换为第二状态之前，将第二标识作为用于触发进入第一状态的触发标识，并生成第一状态下的第一任务。

[0152] 上述的生成单元可包括：文件获取模块，用于获取配置文件，其中，配置文件是采用第一计算机语言编写的文件；生成模块，用于按照配置文件的指示将第二标识作为用于触发进入第一状态的触发标识，并生成第一状态下的第一任务。

[0153] 上述的生成模块在按照配置文件的指示将第二标识作为用于触发进入第一状态的触发标识，并生成第一状态下的第一任务时，可利用配置文件生成任务文件，其中，任务文件是采用第二计算机语言记录有第一状态下的第一任务和第一状态的第二标识的文件，第二计算机语言不同于第一计算机语言。

[0154] 上述的生成模块利用配置文件生成任务文件时，可在任务文件中将第一状态的名字创建为配置文件所指示的名字，并将第二标识作为用于触发进入第一状态的触发标识；在任务文件中创建以第一状态为源状态、并以第二状态为目的状态的第一任务，其中，在任务文件中的第一任务为利用第二计算机语言编写的任务模板创建的；在任务文件中为每个第一任务分配标识，其中，为第一任务分配的标识不同于任意一个已生成任务的标识。

[0155] 可选地，第一任务可为多个，其中，获取单元还可用于在多个第一任务为并行执行的任务的情况下，获取多个第一任务中位于目标位置的第一任务返回的标识为第一标识。

[0156] 可选地，确定单元在确定目标业务所处的第一状态时，可基于驱动类型确定目标业务所处的第一状态，其中，驱动类型用于指示是否存在任务执行完后返回的标识。

[0157] 可选地，确定单元在基于驱动类型确定目标业务所处的第一状态时，可根据目标业务的驱动类型确定切换后所在的第一状态，其中，驱动类型包括外部事件驱动和内部返回码驱动，目标业务由安装在终端上的业务应用提供，外部事件驱动用于指示在终端上对业务应用的操作，内部返回码驱动用于指示任务执行完后返回的标识。

[0158] 上述的确定单元在根据目标业务的驱动类型确定切换后所在的第一状态时，可将多个状态中触发标识与外部事件驱动的标识匹配的状态作为第一状态；或，获取第二任务执行完后返回的第三标识，将多个状态中触发标识与第三标识匹配的状态作为第一状态，其中，第二任务用于表示目标业务中的第二子业务，第三标识用于表示第二任务已经执行完。

[0159] 确定单元在确定目标业务所处的第一状态之后，方法还包括采用以下方式之一将目标业务的状态切换至第一状态：在第三状态与第一状态为同一状态的情况下，保持目标业务所处的状态不变，并执行第一状态下的第一任务，其中，第三状态为切换为第一状态前目标业务所处的状态；在第三状态与第一状态不同的情况下，通过执行对第三状态的退出任务和对第一状态的进入任务来将目标业务的状态由第三状态切换为第一状态；在第三状态与第一状态不同的情况下，通过执行对第三状态的退出任务和对第一状态的进入任务来将目标业务的状态由第三状态切换为第一状态，并执行第一状态下的第一任务。

[0160] 本状态机系统(HFsm)主要是为了解决相关技术中状态机系统存在的不足而设计的，目标是让其在后台分布式系统开发以及游戏引擎核心模式玩法开发中得到有效应用。HFsm主要在如下方面进行了改进：扩展了传统状态机模型，设计了内部返回码驱动(retcode_driven)方式，实现了HFsm的自驱动(self_driven)，并解决了相关技术中状态机模型仅依赖外部事件驱动带来的状态机堆栈嵌套问题；实现了并行动作序列(parallel-action)，以实现游戏玩法中的并行操作需求；实现了基于状态(status)和动作(action)的上下文(context)机制，用于支持action局部参数的动态配置；设计并实现了一套基于xml的HFsm配置规则，并根据该配置规则实现HFsm的图形可视化展示；实现了自动代码生成器，根据HFsm配置规则自动生成C++代码；实现了动态加载器，可动态加载HFsm配置文件，动态运行；实现了一套基于C++代码的HFsm日志系统，用于HFsm运行时日志分析。

[0161] 此处需要说明的是，上述模块与对应的步骤所实现的示例和应用场景相同，但不限于上述实施例所公开的内容。需要说明的是，上述模块作为装置的一部分可以运行在如图1所示的硬件环境中，可以通过软件实现，也可以通过硬件实现，其中，硬件环境包括网络环境。

[0162] 根据本发明实施例的另一个方面，还提供了一种用于实施上述状态的切换方法的服务器或终端。

[0163] 图12是根据本发明实施例的一种终端的结构框图，如图12所示，该终端可以包括：一个或多个(图12中仅示出一个)处理器1001、存储器1003、以及传输装置1005，如图12所示，该终端还可以包括输入输出设备1007。

[0164] 其中，存储器1003可用于存储软件程序以及模块，如本发明实施例中的状态的切换方法和装置对应的程序指令/模块，处理器1001通过运行存储在存储器1003内的软件程

序以及模块，从而执行各种功能应用以及数据处理，即实现上述的状态的切换方法。存储器1003可包括高速随机存储器，还可以包括非易失性存储器，如一个或者多个磁性存储装置、闪存、或者其他非易失性固态存储器。在一些实例中，存储器1003可进一步包括相对于处理器1001远程设置的存储器，这些远程存储器可以通过网络连接至终端。上述网络的实例包括但不限于互联网、企业内部网、局域网、移动通信网及其组合。

[0165] 上述的传输装置1005用于经由一个网络接收或者发送数据，还可以用于处理器与存储器之间的数据传输。上述的网络具体实例可包括有线网络及无线网络。在一个实例中，传输装置1005包括一个网络适配器(Network Interface Controller，NIC)，其可通过网线与其他网络设备与路由器相连从而可与互联网或局域网进行通讯。在一个实例中，传输装置1005为射频(Radio Frequency，RF)模块，其用于通过无线方式与互联网进行通讯。

[0166] 其中，具体地，存储器1003用于存储应用程序。

[0167] 处理器1001可以通过传输装置1005调用存储器1003存储的应用程序，以执行下述步骤：

[0168] 确定目标业务所处的第一状态，其中，第一状态为目标业务的多个状态中的一个，第一状态下有待执行的第一任务；

[0169] 在第一任务执行完的情况下，获取第一任务执行完后返回的第一标识，其中，第一任务用于表示目标业务中的第一子业务，第一标识用于表示第一任务已经执行完；

[0170] 确定多个状态中与第一标识匹配的第二状态，并将目标业务的状态由第一状态切换为第二状态。

[0171] 处理器1001还用于执行下述步骤：

[0172] 在第三状态与第一状态为同一状态的情况下，保持目标业务所处的状态不变，并执行第一状态下的第一任务，其中，第三状态为切换为第一状态前目标业务所处的状态；

[0173] 在第三状态与第一状态不同的情况下，通过执行对第三状态的退出任务和对第一状态的进入任务来将目标业务的状态由第三状态切换为第一状态；

[0174] 在第三状态与第一状态不同的情况下，通过执行对第三状态的退出任务和对第一状态的进入任务来将目标业务的状态由第三状态切换为第一状态，并执行第一状态下的第一任务。

[0175] 采用本发明实施例，确定目标业务所处的第一状态，其中，第一状态为目标业务的多个状态中的一个，第一状态下有待执行的第一任务；在第一任务执行完的情况下，获取第一任务执行完后返回的第一标识，其中，第一任务用于表示目标业务中的第一子业务，第一标识用于表示第一任务已经执行完；确定多个状态中与第一标识匹配的第二状态，并将目标业务的状态由第一状态切换为第二状态，可以解决相关技术中应用内状态机的状态流转的故障率较高的技术问题，进而达到提高应用内状态机的状态流转的准确率的技术效果。

[0176] 可选地，本实施例中的具体示例可以参考上述实施例中所描述的示例，本实施例在此不再赘述。

[0177] 本领域普通技术人员可以理解，图12所示的结构仅为示意，终端可以是智能手机(如Android手机、iOS手机等)、平板电脑、掌上电脑以及移动互联网设备(Mobile Internet Devices，MID)、PAD等终端设备。图12其并不对上述电子装置的结构造成限定。例如，终端还可包括比图12中所示更多或者更少的组件(如网络接口、显示装置等)，或者具有与图12所

示不同的配置。

[0178] 本领域普通技术人员可以理解上述实施例的各种方法中的全部或部分步骤是可以通过程序来指令终端设备相关的硬件来完成，该程序可以存储于一计算机可读存储介质中，存储介质可以包括：闪存盘、只读存储器(Read-Only Memory，ROM)、随机存取器(Random Access Memory，RAM)、磁盘或光盘等。

[0179] 本发明的实施例还提供了一种存储介质。可选地，在本实施例中，上述存储介质可以用于执行状态的切换方法的程序代码。

[0180] 可选地，在本实施例中，上述存储介质可以位于上述实施例所示的网络中的多个网络设备中的至少一个网络设备上。

[0181] 可选地，在本实施例中，存储介质被设置为存储用于执行以下步骤的程序代码：

[0182] S12，确定目标业务所处的第一状态，其中，第一状态为目标业务的多个状态中的一个，第一状态下有待执行的第一任务；

[0183] S14，在第一任务执行完的情况下，获取第一任务执行完后返回的第一标识，其中，第一任务用于表示目标业务中的第一子业务，第一标识用于表示第一任务已经执行完；

[0184] S16，确定多个状态中与第一标识匹配的第二状态，并将目标业务的状态由第一状态切换为第二状态。

[0185] 可选地，存储介质还被设置为存储用于执行以下步骤的程序代码：

[0186] S22，在第三状态与第一状态为同一状态的情况下，保持目标业务所处的状态不变，并执行第一状态下的第一任务，其中，第三状态为切换为第一状态前目标业务所处的状态；

[0187] S24，在第三状态与第一状态不同的情况下，通过执行对第三状态的退出任务和对第一状态的进入任务来将目标业务的状态由第三状态切换为第一状态；

[0188] S26，在第三状态与第一状态不同的情况下，通过执行对第三状态的退出任务和对第一状态的进入任务来将目标业务的状态由第三状态切换为第一状态，并执行第一状态下的第一任务。

[0189] 可选地，本实施例中的具体示例可以参考上述实施例中所描述的示例，本实施例在此不再赘述。

[0190] 可选地，在本实施例中，上述存储介质可以包括但不限于：U盘、只读存储器(ROM，Read-Only Memory)、随机存取存储器(RAM，Random Access Memory)、移动硬盘、磁碟或者光盘等各种可以存储程序代码的介质。

[0191] 上述本发明实施例序号仅仅为了描述，不代表实施例的优劣。

[0192] 上述实施例中的集成的单元如果以软件功能单元的形式实现并作为独立的产品销售或使用时，可以存储在上述计算机可读取的存储介质中。基于这样的理解，本发明的技术方案本质上或者说对现有技术做出贡献的部分或者该技术方案的全部或部分可以以软件产品的形式体现出来，该计算机软件产品存储在存储介质中，包括若干指令用以使得一台或多台计算机设备(可为个人计算机、服务器或者网络设备等)执行本发明各个实施例所述方法的全部或部分步骤。

[0193] 在本发明的上述实施例中，对各个实施例的描述都各有侧重，某个实施例中没有详述的部分，可以参见其他实施例的相关描述。

[0194] 在本申请所提供的几个实施例中，应该理解到，所揭露的客户端，可通过其它的方式实现。其中，以上所描述的装置实施例仅仅是示意性的，例如所述单元的划分，仅仅为一种逻辑功能划分，实际实现时可以有另外的划分方式，例如多个单元或组件可以结合或者可以集成到另一个系统，或一些特征可以忽略，或不执行。另一点，所显示或讨论的相互之间的耦合或直接耦合或通信连接可以是通过一些接口，单元或模块的间接耦合或通信连接，可以是电性或其它的形式。

[0195] 所述作为分离部件说明的单元可以是或者也可以不是物理上分开的，作为单元显示的部件可以是或者也可以不是物理单元，即可以位于一个地方，或者也可以分布到多个网络单元上。可以根据实际的需要选择其中的部分或者全部单元来实现本实施例方案的目的。

[0196] 另外，在本发明各个实施例中的各功能单元可以集成在一个处理单元中，也可以是各个单元单独物理存在，也可以两个或两个以上单元集成在一个单元中。上述集成的单元既可以采用硬件的形式实现，也可以采用软件功能单元的形式实现。

[0197] 以上所述仅是本发明的优选实施方式，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。

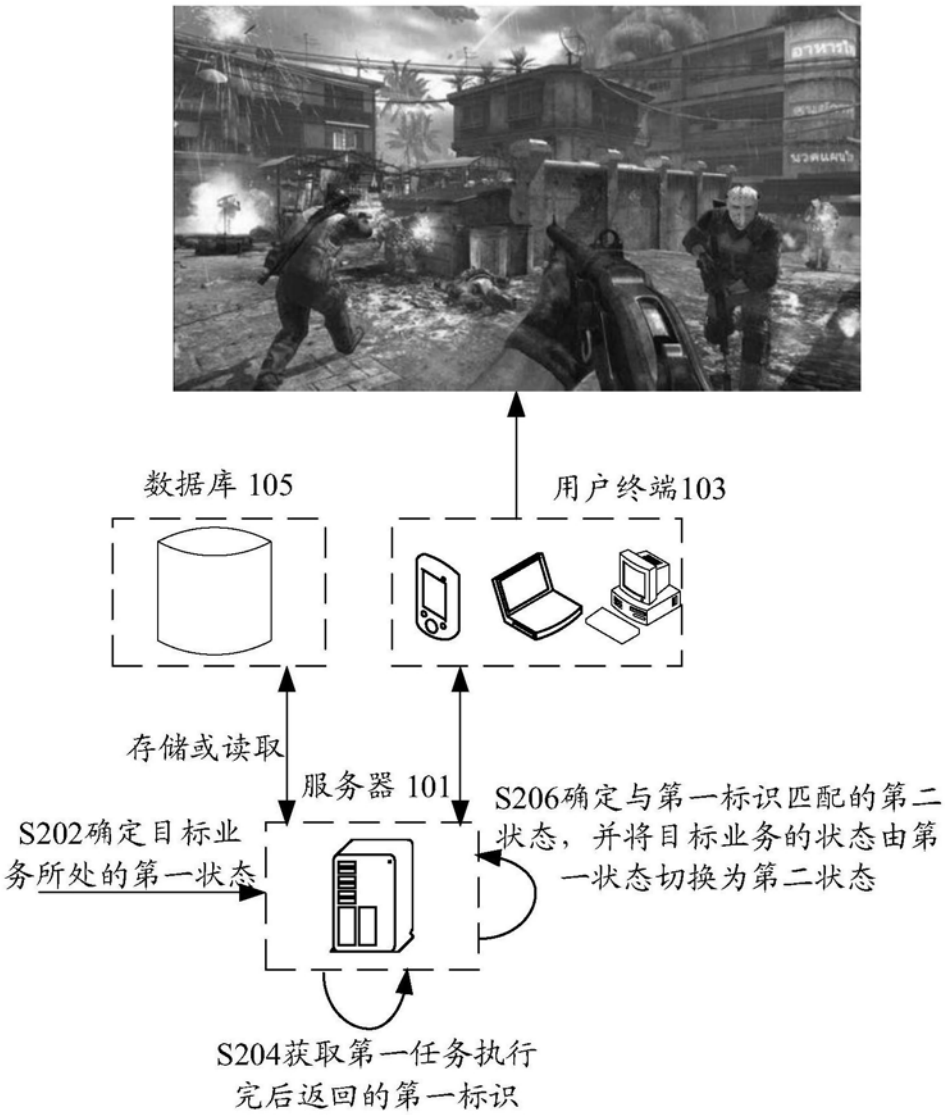

数据库 105
用户终端103
存储或读取
服务器 101
S202确定目标业务所处的第一状态
S206确定与第一标识匹配的第二状态，并将目标业务的状态由第一状态切换为第二状态
S204获取第一任务执行完后返回的第一标识

图1

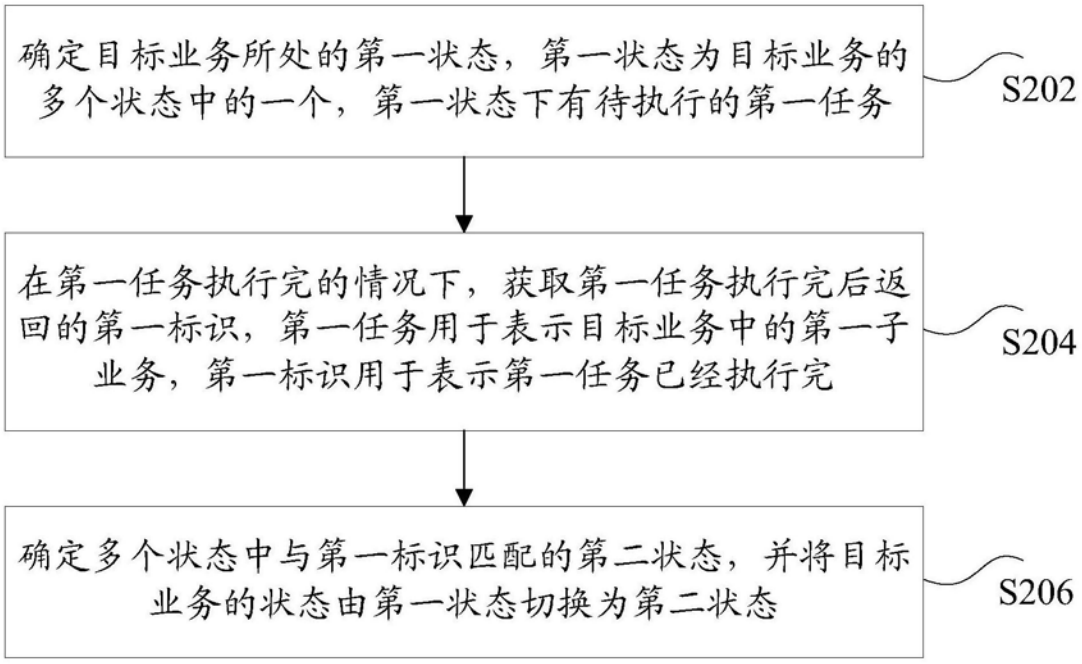

确定目标业务所处的第一状态，第一状态为目标业务的多个状态中的一个，第一状态下有待执行的第一任务
S202
在第一任务执行完的情况下，获取第一任务执行完后返回的第一标识，第一任务用于表示目标业务中的第一子业务，第一标识用于表示第一任务已经执行完
S204
确定多个状态中与第一标识匹配的第二状态，并将目标业务的状态由第一状态切换为第二状态
S206

图2

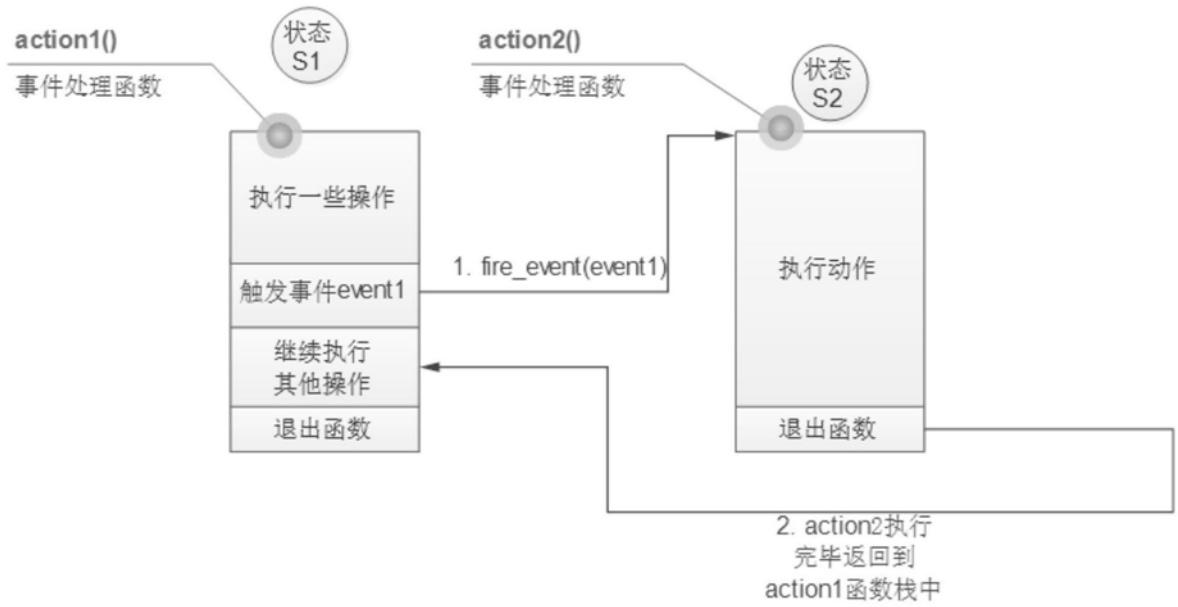

action1()
事件处理函数
状态
S1
action2()
事件处理函数
状态
S2
执行一些操作
触发事件event1
继续执行
其他操作
退出函数
1. fire_event(event1)
执行动作
退出函数
2. action2执行
完毕返回到
action1函数栈中

图3

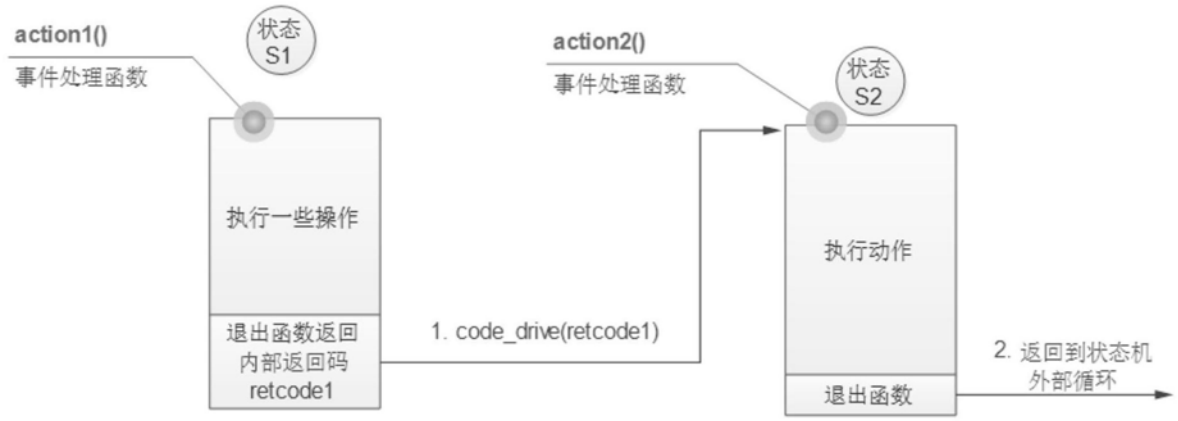

action1()
事件处理函数
状态
S1
执行一些操作
退出函数返回
内部返回码
retcode1
1. code_drive(retcode1)
action2()
事件处理函数
状态
S2
执行动作
退出函数
2. 返回到状态机
外部循环

图4

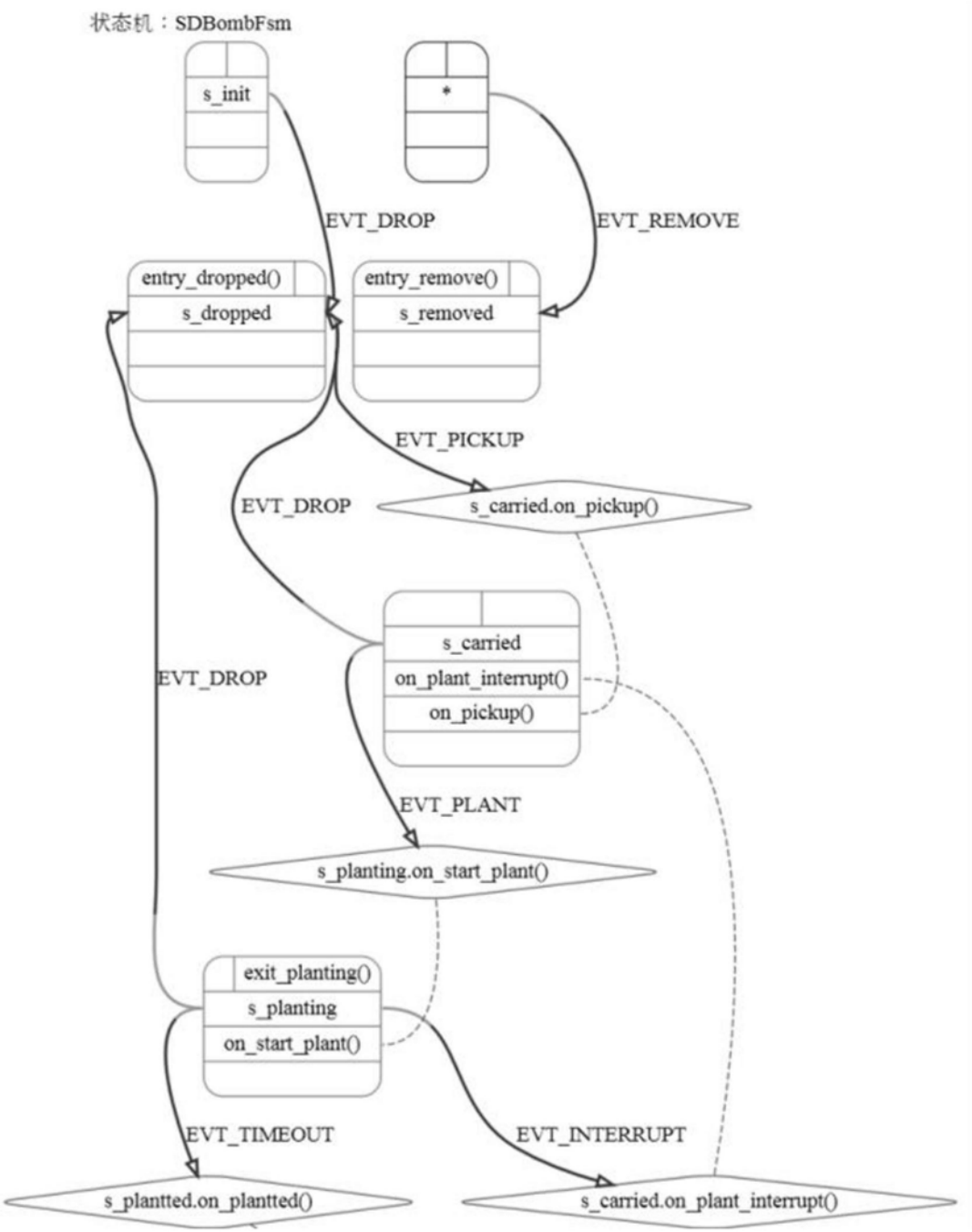

状态机：SDBombFsm
s_init
*
EVT_DROP
EVT_REMOVE
entry_dropped()
s_dropped
entry_remove()
s_removed
EVT_PICKUP
EVT_DROP
s_carried.on_pickup()
s_carried
on_plant_interrupt()
on_pickup()
EVT_DROP
EVT_PLANT
s_planting.on_start_plant()
exit_planting()
s_planting
on_start_plant()
EVT_TIMEOUT
EVT_INTERRUPT
s_plantted.on_plantted()
s_carried.on_plant_interrupt()

图5

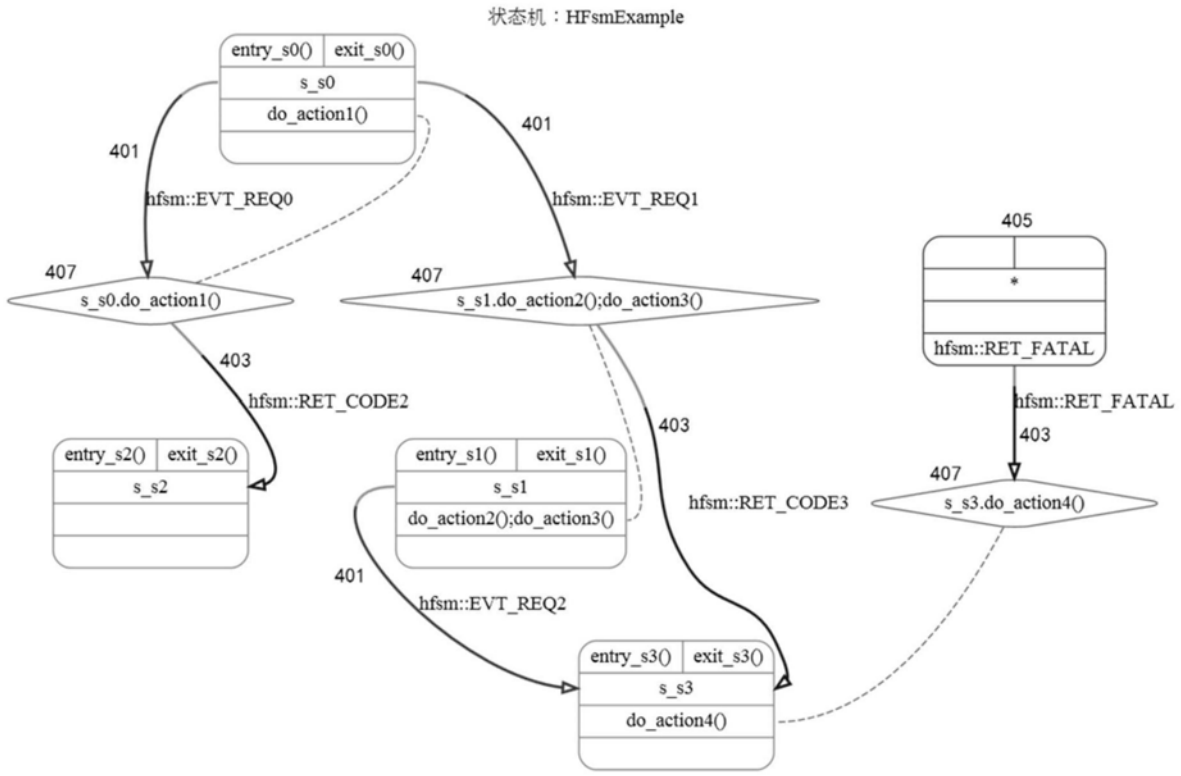

状态机：HFsmExample
entry_s0() exit_s0()
s_s0
do_action1()
401
hfsm::EVT_REQ0
401
hfsm::EVT_REQ1
405
*
hfsm::RET_FATAL
407
s_s0.do_action1()
407
s_s1.do_action2();do_action3()
hfsm::RET_FATAL
403
403
hfsm::RET_CODE2
403
hfsm::RET_CODE3
407
s_s3.do_action4()
entry_s2() exit_s2()
s_s2
entry_s1() exit_s1()
s_s1
do_action2();do_action3()
401
hfsm::EVT_REQ2
entry_s3() exit_s3()
s_s3
do_action4()

图6

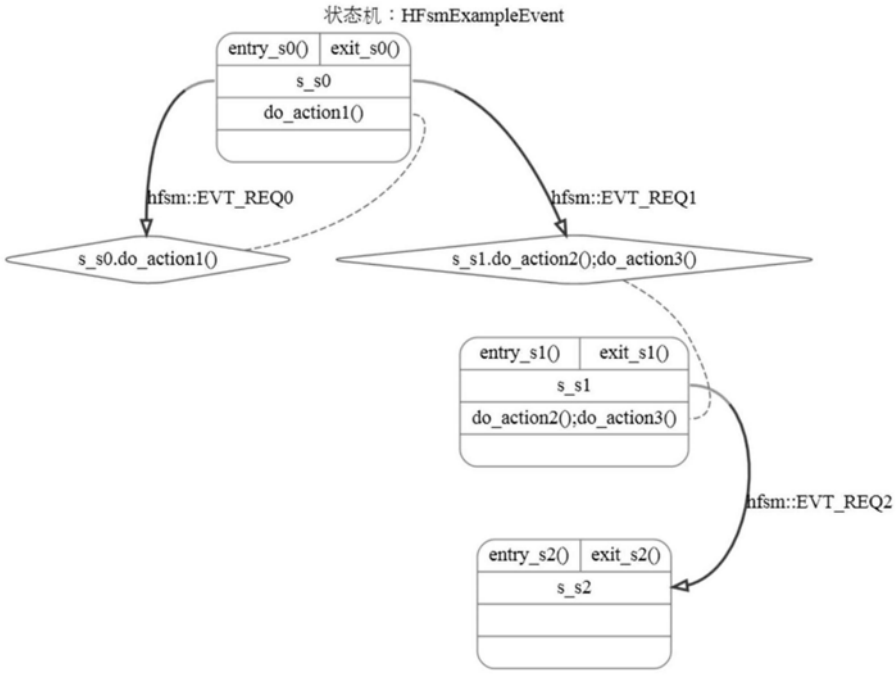

状态机：HFsmExampleEvent
entry_s0() exit_s0()
s_s0
do_action1()
hfsm::EVT_REQ0
hfsm::EVT_REQ1
s_s0.do_action1()
s_s1.do_action2();do_action3()
entry_s1() exit_s1()
s_s1
do_action2();do_action3()
hfsm::EVT_REQ2
entry_s2() exit_s2()
s_s2

图7

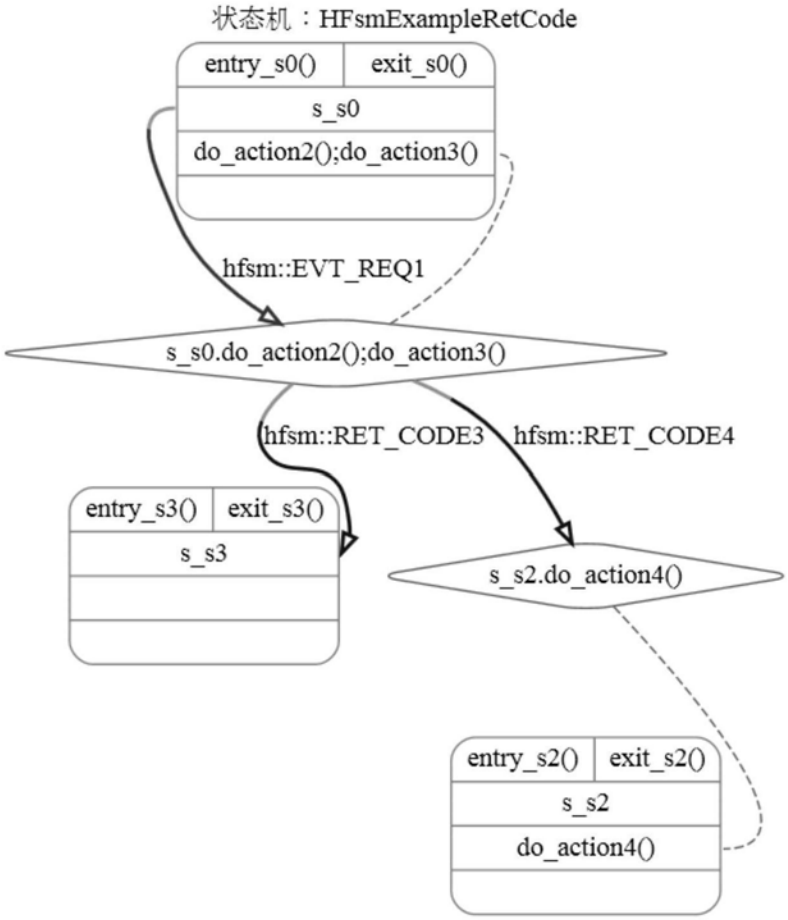

状态机：HFsmExampleRetCode
entry_s0()
exit_s0()
s_s0
do_action2();do_action3()
hfsm::EVT_REQ1
s_s0.do_action2();do_action3()
hfsm::RET_CODE3
hfsm::RET_CODE4
entry_s3()
exit_s3()
s_s3
s_s2.do_action4()
entry_s2()
exit_s2()
s_s2
do_action4()

图8

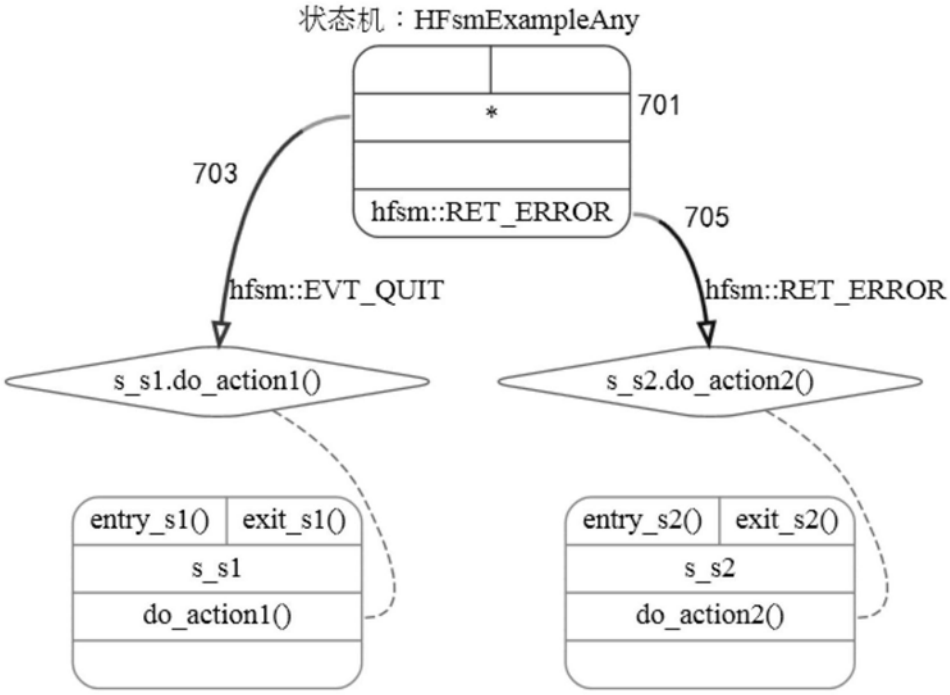

状态机：HFsmExampleAny
*
701
703
hfsm::RET_ERROR
705
hfsm::EVT_QUIT
hfsm::RET_ERROR
s_s1.do_action1()
s_s2.do_action2()
entry_s1()
exit_s1()
s_s1
do_action1()
entry_s2()
exit_s2()
s_s2
do_action2()

图9

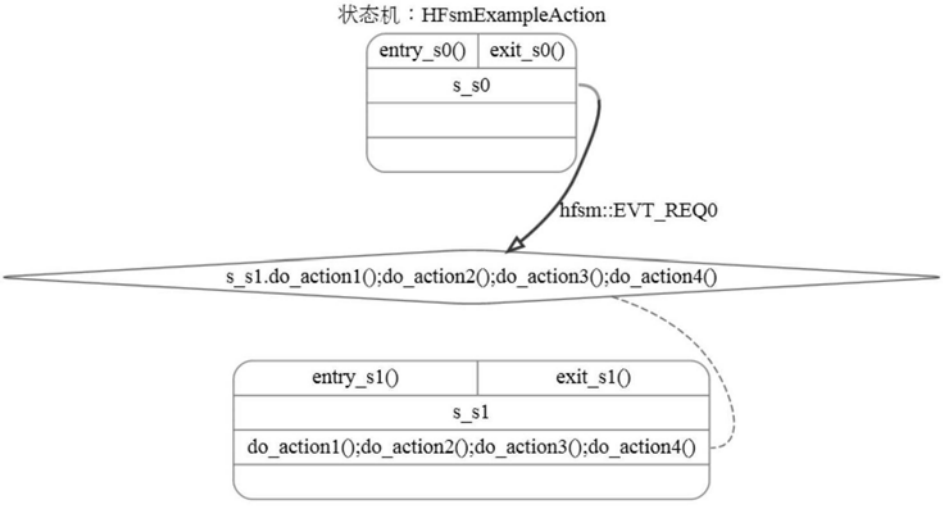

状态机：HFsmExampleAction
entry_s0()
exit_s0()
s_s0
hfsm::EVT_REQ0
s_s1.do_action1();do_action2();do_action3();do_action4()
entry_s1()
exit_s1()
s_s1
do_action1();do_action2();do_action3();do_action4()

图10

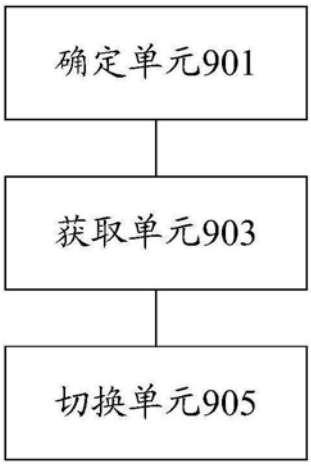


图11

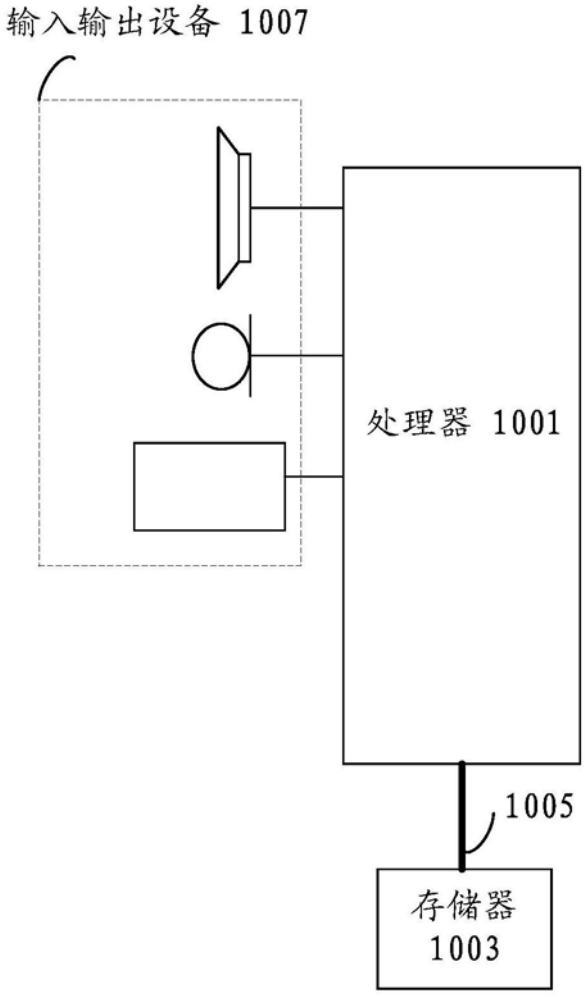


图12