计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(9):283−287 [doi: 10.15888/j.cnki.csa.006434]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

高性能云工作流状态机的原理与实现①

程　旭, 张　斌, 刘一田

(南瑞集团有限公司 (国网电力科学研究院有限公司), 南京 210003)
通讯作者: 程　旭, E-mail: ceinx@qq.com

摘　要: 传统工作流程通过设计活动和迁移线等元素来实现流程的基本流转, 但随之而来的问题是当流程异常复杂, 例如存在多达几十个活动且活动之间需要不断跳转交互的情况下, 不仅开发复杂度成倍增加, 而且运行时性能也会持续降低. 为解决此问题, 本文基于有限状态机的原理, 结合云计算技术, 提出了云工作流状态机, 能够充分利用代码逻辑开发的便捷性, 简化流程的活动和迁移, 最终达到运行时高性能的架构目标. 文中详细阐述了工作流状态机的实现原理, 运行机制, 以及云计算下的状态机服务框架, 最后介绍了基于云状态机的业务应用开发方式, 并给出容器下的压力测试结果: 流程在双节点流转的单步耗时非常短, 运行稳定. 实践证明, 基于容器的云架构在保证可扩展性的同时亦能满足高性能的设计目标.

关键词: 工作流; 状态机; 云计算; 容器

引用格式: 程旭,张斌,刘一田.高性能云工作流状态机的原理与实现.计算机系统应用,2018,27(9):283–287. http://www.c-s-a.org.cn/1003-3254/6434.html

Mechanism and Implementation of High Performance Workflow State Machine Cloud

CHENG Xu, ZHANG Bin, LIU Yi-Tian

(NARI Group Corporation (State Grid Electric Power Research Institute), Nanjing 210003, China)

Abstract: Traditional business processes are usually designed with process elements—activities and transitions. However, when the process is becoming complicated, such as dozens of activities which could jump to each other in one complex workflow, the process modeling is becoming harder, and the system will probably turn to poor performance. In order to solve this problem, this paper presents a system of workflow state machine cloud based on cloud computing and the mechanism of finite state machine, which can simplify the activities and transitions by coding the flow process to achieve high performance, and elaborates the mechanism of the state machine framework in cloud. Finally, the developing method of business application based on the workflow state machine cloud is introduced, and the performance results tested in container are given: the time consuming of one-step flow between activities running in two containers is very short. That means the design based on Docker can achieve high performance as well as maintaining scalability.

Key words: workflow; state machine; cloud computing; container

在当今信息化社会中, 工作流程已得到广泛和深入的使用[1]. 在传统流程的建模过程中, 流程开发者通过设计流程图, 构建活动节点, 来规划流程的走向, 实现流程的功能[2]. 虽然传统的流程建模方便, 易用性高, 但是, 当流程非常复杂时, 流程图中的活动和迁移线也会错综复杂, 甚至还需要在流程脚本中实现相当多的代码, 成倍增加设计的复杂度.

传统流程流转的本质是通过代码创建活动实例和

① 收稿时间: 2017-10-20; 修改时间: 2017-11-14; 采用时间: 2017-12-15; csa 在线出版时间: 2018-08-16

工作项等流程的基本状态信息, 而使用工作流状态机开发流程, 就是将其本质抽象出来, 通过状态中的代码控制流程的流转, 省去繁杂的活动和迁移线. 将活动节点抽象成状态, 通过工作流状态机实现状态之间的迁移, 可以对传统流程中的活动进行简化. 云计算在业界早已具有成熟的应用案例, 工作流状态机依托现有的云计算技术, 实现高性能、可扩展的云状态机.

1 国内外研究进展

自工作流管理联盟 (WfMC)1993 年成立以来, 工作流技术已日趋成熟稳定[3]. 业界提出了众多流程规范, 包括流程设计规范《业务流程模型和符号》2.0 版本 (BPMN 2.0) 等[4], 这些都为工作流产品提供了统一的标准, 当前主流商业产品如 IBM 业务流程管理, 开源产品如 Activiti 等均支持 BPMN 规范. 通常流程引擎采用的核心调度算法包括有限状态机 (FSM) 和基于令牌的 Petri 网 (PetriNet) 等.

自谷歌十年前提出云计算概念以来, 云技术经历了长足的发展, 目前已经形成了通信即服务 (CaaS)、基础设施即服务 (IaaS)、监测即服务 (MaaS)、平台即服务 (PaaS) 和软件即服务 (SaaS) 等多种基于互联网的云计算服务[5]. 工作流结合云计算也已具有广泛应用, 与传统的工作流中间件相比, 优势明显: 云工作流与应用系统的架构无关; 基于网络协议的任何系统都可以消费云工作流 API; 可以被独立部署, 支持分布式和集群; 单一的云流程服务器可以满足多个系统嵌入工作流的需要; 使用云工作流易于大数据下的流程节点横向扩展.

微软最早在 2005 年 Windows Workflow Foundation (WWF) 产品中成功实践了状态机工作流的概念[6], 其自身 Activity 机制能绘制出清晰的状态图. 但是在复杂场景下, 例如存在多次交互跳转的情况下, 其使用上仍然不如直接开发代码简洁易用. 因此, 本文基于有限状态机的原理提出了云工作流状态机, 不对流程逻辑代码做出任何束缚, 充分发挥代码开发的优势, 利用云环境下的特性, 创建出高效、易用和可扩展的工作流程.

2 工作流状态机的设计原理

工作流状态机是有限状态机的理论在业务流程中的应用, 将传统流程中活动的迁移表示为状态之间的变换, 以实现流程的流转.

2.1 工作流状态机的数学模型

有限状态机有三个特征: 状态总数是有限的; 任一时刻, 只处在一种状态之中; 某种条件下, 会从一种状态转变到另一种状态. 工作流状态机是一种有限状态机[7], 是表示有限多个状态以及在这些状态之间转移和状态内动作的数学模型. 有限状态机是五元组 $(\Sigma, S, s_0, \delta, F)$, 其中, Σ是输入字母表 (符号的非空有限集合); S是状态的非空有限集合; s_0是初始状态, 它是 S的元素; δ是状态转移函数: $\delta: S \times \Sigma \to S$; F是最终状态的集合, S的 (可能为空) 子集.

以工作流状态机为用例, 状态机的状态总数为S, 其中启动流程的初始状态为s_0, 进入状态时的输入信息为 Σ, 每一个状态 (即S中的每一个元素) 内部包含状态转移函数 δ, 输出状态为F. 启动流程时, 外界输入信息 Σ到状态s_0, s_0状态调用内部状态转移函数δ, 生成下一步的状态F; 发送流程时, 将F作为本次状态转移的初始状态s_0, 再进行状态迁移, 生成下一步状态F; 结束流程时, 生成的下一步状态F为空, 不再进行状态转移. 可见, 工作流状态机的下一步状态F是由当前状态 s_0中的状态转移函数 δ和输入信息 Σ共同决定的.

2.2 工作流状态机的运行模式

一个工作流状态机由若干个状态组成, 每个状态提供“进入 (Enter)”和“流转 (Resume)”方法. 状态机进入某个状态 (如状态 1) 后产生一个标志 (Bookmark), 将标志返回给调用者后就开始等待. 直到外界再次通过标志触发流转方法后, 该状态继续流转, 根据输入值和流转的内部逻辑, 决定下一个状态, 并进入到下一个状态的进入方法, 执行完毕会返回一个标志并开始等待. 如此循环直到状态返回“结束”时, 工作流状态机不再进入新的状态, 而是结束当前状态并结束状态机, 实现流程的结束功能. 其基本运行模式如图 1 所示.

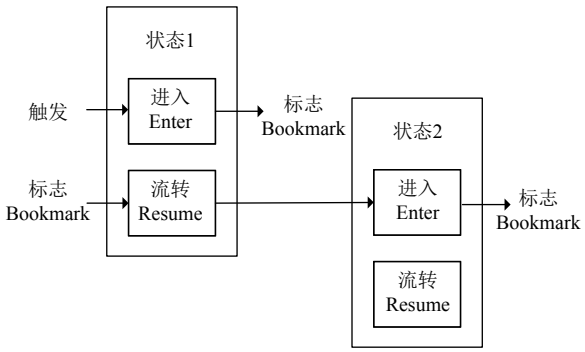

图 1 工作流状态机的基本运行模式

工作流状态机对外暴露“启动 (Start)”、“流转 (Resume)”和“终止 (Abort)”三个基本的方法. 一个工作流状态机相当于一个流程实例, 可以由外界通过这三个方法控制流程实例的流转. 在所有的状态之中必须具有一个开始状态, 用于接受启动方法调用, 当执行启动方法时, 工作流状态机进入到开始状态的“进入”方法, 并返回标志 (Bookmark) 将流程悬挂, 等待下一次的调用. 当外界调用流转方法时, 将标志 (Bookmark) 传入, 根据当前状态的返回值决定并进入下一个要进入的状态. 在终止方法被调用时, 系统自动设置标志 (Bookmark) 的状态和工作流状态机的状态为失效, 用于标识流程的不可用状态. 工作流状态机通过每一个状态的“流转”方法返回的结果实现了不同状态的多次跳转, 轻松实现流程的追回、回退以及自由流的功能, 从而脱离了原先通过迁移线固化流程运行轨迹的流程设计方式, 取而代之的是通过代码操控流程的流向——这种流程设计的优势在于流程的设计更加灵活, 不用局限于原有的活动和迁移线的概念, 当流程运行场景非常复杂时, 可通过代码来简化原先流程的活动节点. 工作流状态机的流转机制如图 2 所示.

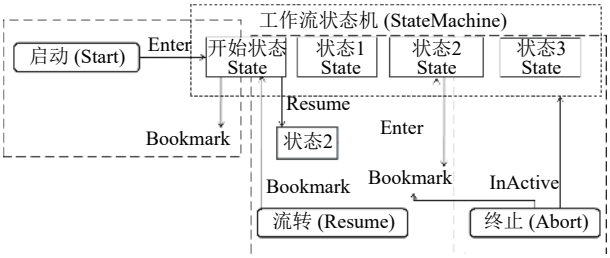

图 2 工作流状态机的流转机制

流程通过状态的返回结果来控制流程的流向, 在流程正常流转到下一个状态的情况下, 系统会将当前的标志 (Bookmark) 置为失效, 并创建一个基于新状态的标志 (Bookmark); 当状态返回“结束”标志时, 流程终结, 不再进入后续状态, 系统自动结束当前的工作流状态机和标志 (Bookmark); 当状态返回为当前本身的状态时, 即下一个状态与当前状态相同, 那么在重新进入自身状态执行时, 当前状态的标志 (Bookmark) 不失效, 且不重新生成新的标志 (Bookmark), 此种场景多用于活动的多实例会签. 三种场景的比较如图 3 所示.

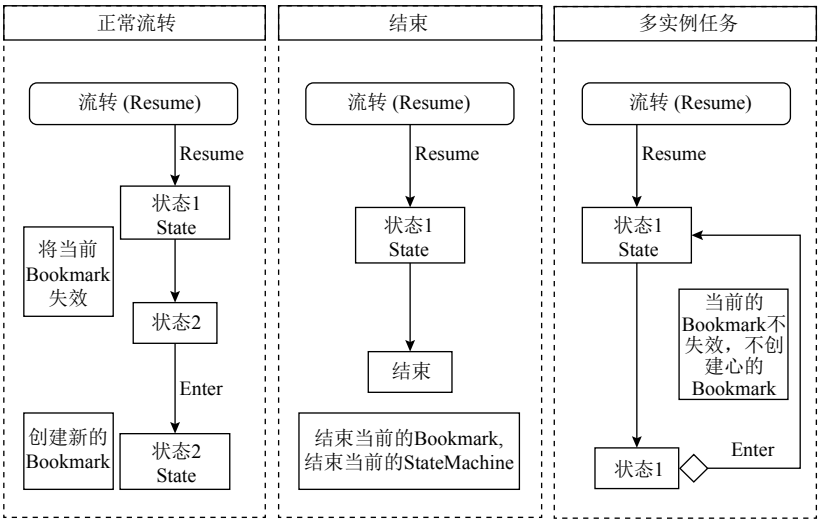

图 3 工作流状态机的三种状态返回场景

3 云工作流状态机的实现

结合云技术与工作流状态机的原理, 采用微服务化的架构设计开发. 传统的工作流服务是一个工作流引擎, 可设计运转多个流程, 而转变为微服务的模式后, 每一个流程就是一个服务, 可部署在容器中, 从而实现流程服务的动态伸缩, 优化利用资源.

3.1 微服务化的云架构

当前主流云架构基本使用 Docker 容器技术搭建[8], 工作流状态机在开发完成后发布成镜像, 可启动多个该镜像的 Docker 容器, 技术上选用 Spring Cloud 解决方案[9], 配合 Eureka 的服务发现机制和 Zookeeper 的分布式协调服务[10], 实现微服务化的状工作流状态机集群.

通过 Eureka 的服务注册发现机制实现负载均衡: 每一个工作流状态机启动时向 Eureka 服务注册, 客户端调用映射出的服务, Eureka 会根据内部算法将请求自动转发到任意一个状态机容器中; 通过 Spring Cloud Config 的中心化配置实现所有容器中配置信息的统一更新. 在启动工作流时, 将状态机 ID 与当前节点信息自动注册到 Zookeeper 服务器中临时性节点, 后续如果有任何与状态机 ID 相关的操作全部转发到此节点中执行, 这样可保证一个流程只在一个节点中执行, 同一个流程的并发操作在流程控制器中排队执行. 将状态机的 Docker 容器加入统一的 Swarm 集群中, 可实现状态机之间的请求转发. 数据库存储表结构比较简单, 无需复杂的 SQL 查询, 因此选用高性能的文档型数据库 MongoDB 存储, 提高插入与查询的效率[11]. 微服务化的云架构图如图 4 所示.

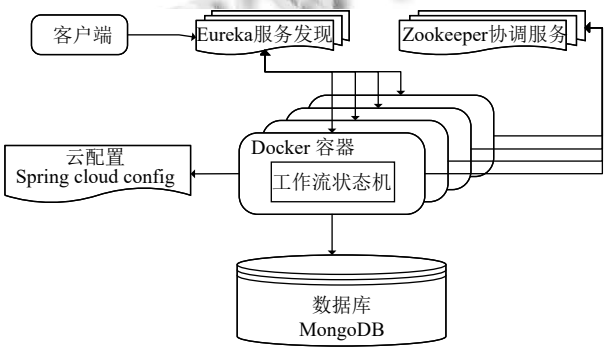

图 4 微服务化的云架构图

3.2 云工作流状态机的处理架构

在云架构下的每个工作流状态机微服务都对外暴露 Restful 接口, 它接收到请求后, 通过分派器查找所请求的流程实例所在的微服务节点, 找到后将请求转发到目标服务所在节点, 目标节点收到请求后由控制器实际处理. 流程开发者继承状态机和状态两个基类, 开发与具体业务相关的流程状态机和流程状态. 控制器中实现了流程状态的跳转, 每一个状态都存在“进入”和“流转”两个方法. 在“进入”方法中, 通过进入上下文创建标识和工作项, 在“流转”方法中, 返回下一个状态的名称. 同时, 还可以设置状态机和状态的变量, 由数据访问层将状态机以及变量数据、标识数据和工作项保存如 MongoDB 数据库中. 云工作流状态机系统结构图如图 5 所示.

每一个工作流产生的流程实例固定运行在某一个微服务节点上, 所有请求由该节点中的控制器处理. 控制器中的工作原理是一个单线程池, 所有与当前流程实例相关的操作排队进入到控制器中的单线程池中, 一个操作处理完毕后再处理下一个操作, 这样在处理两个同样的请求时, 能够保证只处理一次, 在第一次操作处理完毕后, 数据的状态已经发生变化, 就无法处理第二次请求. 简言之, 通过单线程的排队处理规避了正常情况下使用锁的机制, 以此减少了创建锁和销毁锁的开销, 大幅提升了状态机流转效率. 控制器中通过原子变量累计请求次数, 当控制器中现有多个请求完成后, 即请求次数与完成次数相等时, 工作流服务卸载当前的控制器.

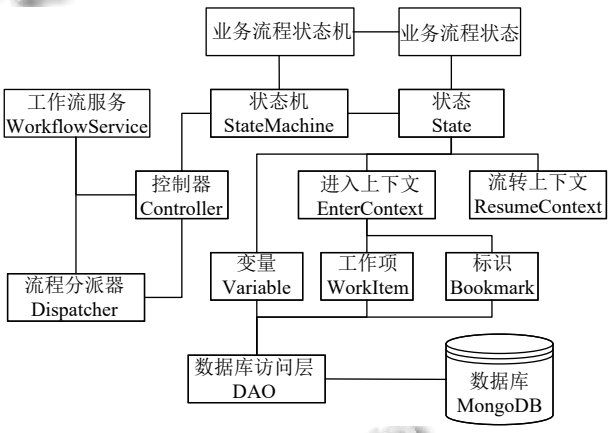

图 5 云工作流状态机系统结构图

3.3 基于云工作流状态机的业务研发

云工作流状态机基于 Spring Cloud 技术提供了云环境下的不同状态之间的迁移功能[12], 从底层保证了状态之间流转的稳定和高效. 业务系统基于状态机研发工作流时不用考虑底层状态流转的性能与并发问题, 只需要编写每一个状态的进入 (Enter) 和流转 (Resume) 方法, 根据需要生成工作项, 并开发客户端, 即可实现一个业务流程. 除此之外, 云状态机不提供统一的流程图和流程日志, 业务研发人员根据需要自行定制流程图, 记录流程日志.

虽然与传统的工作流只需要在界面上拖拽流程元素的开发方式相比, 编写代码要复杂一些, 但遇到极为复杂的流程时, 使用状态机的状态来代替传统流程中的活动能够大幅简化流程图的复杂度. 将多个顺序或分支活动融合为一个状态, 在状态内部通过代码逻辑代替迁移线的判断逻辑, 执行后续状态迁移.

4 实验分析

实验分别对传统工作流程和云工作流状态机进行压力测试, 通过比对分析测试结果而得出结论. 两者的测试环境相同, 硬件环境为多台 8 核 16 GB 内存的 IBM x3650 服务器, 操作系统为 Linux 6.5. 传统工作流程的中间件为 Weblogic 11, 采用双机集群的方式部署在两台服务器上, 数据库为 Oracle 11g, 独立部署在另外一台服务器上. 云状态机在两台服务器上各启动一个 Docker 容器, 加入到国家电网公司智能运检分析管控系统的测试 Swarm 集群中, Eureka、Zookeeper 和 Mongodb 均采用 Swarm 集群中已经配置完成的节点. 使用 LoadRunner 11 编写两个脚本进行压测[13], 每个测试脚本均以简单的报销审批流程为例, 传统流程包含一个开始填单活动和 3 个人工活动, 云状态机包含一个开始填单状态和一个审批状态, 在审批状态中会产生 3 次工作项. 每个压测脚本执行一次启动, 4 次发送操作, 将一个流程发送到结束.

测试时使用 100 并发压力, 执行时间 30 分钟. 两者事务通过率都为 100%, 测试结果有效, 将 4 次发送耗时取平均值统计如表 1.

表 1 传统工作流程与云工作流性能测试结果对比

100 并发	启动流程 (s)	发送流程 (s)	每秒响应请求数
云状态机	0.02	0.023	120
传统流程	0.074	0.208	118

统计结果表明, 100 并发时云状态机单次请求的平均响应时间约在 20 ms 左右, 远低于传统工作流引擎的响应耗时. 可见, 云工作流状态机内部的系统架构设计以及非关系型数据库的构建行之有效, 保证了流程运行时的高性能. 实际生产环境中当访问压力较大时, 通过启动多个 Docker 容器分散压力, 减少请求响应时间; 当访问量较小时, 可以关闭 Docker 容器, 节约系统资源, 从而实现微服务的弹性伸缩[14].

5 结语

本文设计并实现了微服务化的云工作流状态机, 文中阐述了系统的关键技术与核心架构, 实验结果表明其性能优于传统流程. 云工作流状态机属于国家电网公司网省版智能运检分析管控系统中可扩展框架平台的一部分, 支撑了运检指令等业务流程的开发, 得到了广泛应用. 使用工作流状态机开发业务流程对流程开发人员的水平要求较高, 在进行流程设计前, 需要深度了解业务需求, 规划出若干合理的工作流状态. 后续需要解决的一个问题是, 由于云状态机的性能过高, 导致压力转移到 Zookeeper 上, 引起了单节点 Zookeeper 的不稳定. 另外, 要针对业务需求提供更加丰富的接口, 并研究通用的流程图与流程日志, 减轻业务开发压力.

参考文献

1 Leymann F, Roller D. Production Workflow: Concepts and Techniques. ACM, 2001.

2 Aalst W, Hofstede A, Barotsz K, *et al*. Workflow patterns. Distributed and Parallel Databases, 2003, 14(1): 5–51. [doi: 10.1023/A:1022883727209]

3 Coalition WM. The Workflow Reference Model. Workflow Handbook. John Wiley & Sons, Inc., 1995.

4 Chinosi M, Trombetta A. BPMN: An introduction to the standard. Computer Standards & Interfaces, 2012, 34(1): 124–134.

5 Rittinghouse J, Ransome J. Cloud computing: Implementation, Management, and Security. CRC Press, 2016.

6 Kitta T. Professional Windows Workflow Foundation. Wrox Press Ltd., 2007.

7 Lee D, Yannakakis M. Principles and methods of testing finite state machines-a survey. Proceedings of the IEEE, 2008, 84(8): 1090–1123. [doi: 10.1109/5.533956]

8 Stubbs J, Moreira W, Dooley R. Distributed systems of microservices using docker and serfnode. 7th International Workshop on Science Gateways. Budapest, Hungary: IEEE. 2015. 34–39. [doi: 10.1109/IWSG.2015.16]

9 Gutierrez F. Pro Spring Boot. Apress, 2016.

10 Hunt P, Konar M, Junqueira FP, *et al*. ZooKeeper: Wait-free coordination for internet-scale systems. USENIX Annual Technical Conference. 2010. 653–710.

11 Chodorow K. MongoDB: The Definitive Guide. O'Reilly Media, Inc., 2013.

12 Thönes J. Microservices. IEEE Software, 2015, 32(1): 116–116. [doi: 10.1109/MS.2015.11]

13 伊文斌, 郑剑. 基于 LoadRunner 的 Web 负载测试. 江西理工大学学报, 2008, 29(4): 13–15.

14 苗立尧, 陈莉君. 一种基于 Docker 容器的集群分段伸缩方法. 计算机应用与软件, 2017, 34(1): 34–38. [doi: 10.3969/j.issn.1000-386x.2017.01.006]