基于微服务的工作流技术在云管平台的应用

罗钦凯[1,2],倪成章[1]
(1. 华中科技大学,湖北 武汉 430074;
2. 武汉邮电科学研究院,湖北 武汉 430074)

摘 要:针对软件即服务(SaaS)层应用采用单体架构方案时开发难度高、耦合性强、扩展性和可维护性差的问题,设计了一种由工作流引擎驱动业务流程的基于微服务架构的 SaaS 层云管平台(CMP)。基于工作流核心设计模型和微服务开发框架 Spring Cloud,设计了基于微服务的工作流组件结构模型,由组件内工作流引擎(Activiti)驱动 CMP 业务流程;根据工作流引擎 API 封装模式提出面向业务流程的微服务组件间 REST API 设计方案、流程节点自由跳转算法以及命令查询职责分离(CQRS)模式数据操作方案。以面向 OpenStack 的云资源调度为具体应用场景,每个微服务组件的业务功能均可被独立设计开发,工作流组件将解耦的业务组件串联,驱动完成完整的业务流程,验证了 REST API 与流程节点自由跳转,以及 CQRS 模式数据操作的可行性。

关键词:工作流;微服务;云管平台;持续集成;可扩展性;敏捷开发
中图分类号:TP311.5 **文献标识码**:A **文章编号**:1673-629X(2019)09-0122-06
doi:10.3969/j.issn.1673-629X.2019.09.024

Application of Workflow Technology Based on Micro-service in Cloud Management Platform

LUO Qin-kai[1,2], NI Cheng-zhang[1]
(1. Huazhong University of Science and Technology, Wuhan 430074, China;
2. Wuhan Research Institute of Posts and Telecommunications, Wuhan 430074, China)

Abstract: For software-as-a-service (SaaS) layer applications, it is difficult to develop with high coupling, poor scalability and poor maintainability when adopting a single architecture scheme. A SaaS layer cloud management platform (CMP) application based on micro-service architecture is designed, which is driven by workflow engine. Based on the workflow core design model and micro-service development frame (Spring Cloud), the workflow component structure model based on the micro-service is designed, and the CMP business process is driven by the in-component workflow engine (Activiti). According to the encapsulation mode of workflow engine API, the REST API design scheme among micro-service components for business processes, the free jump algorithm of process nodes and the data operation scheme of command query responsibility segregation (CQRS) mode are proposed. Taking OpenStack oriented cloud resource scheduling as a specific application scenario, the business functions of each micro-service component can be independently designed and developed. Workflow components connect decoupled business components in series and drive the complete business process to verify the feasibility of REST API and process node free jump as well as CQRS mode data operation.

Key words: workflow; micro-service; cloud management platform; continuous integration; expandability; agile development

0 引 言

全行业大规模的“上云”需求,和以 OpenStack 为代表的云计算资源类型的多样化,极大扩展了云管平台(cloud management platform, CMP)的服务领域,亟待实现一种架构层面直观反映复杂业务逻辑的、便于敏捷开发与持续集成的 CMP 开发方案。单体架构已不能满足软件即服务(software-as-a-service, SaaS)层应用在高扩展性和高可用性等方面的需要,亟待一种新的设计架构能最大限度地实现高扩展性。

单体架构应用开发成型迅速,但随着线上业务需求复杂多样化,新需求调研阶段的沟通成本显著,业务逻辑开发困难,无法满足应用对扩展性和维护性的要

收稿日期:2018-07-13 **修回日期**:2018-11-15 **网络出版时间**:2019-04-24
基金项目:中央高校基本科研业务费资助:HUST(2018JYCXJJ052)
作者简介:罗钦凯(1994-),男,硕士研究生,研究方向为计算机应用、云计算。
网络出版地址:http://kns.cnki.net/kcms/detail/61.1450.TP.20190424.1004.002.html

求[1],也将导致需求调研阶段需求沟通的成本显著和上线维护阶段可维护性差。从开发实现业务逻辑的痛点出发,通过解构工作流核心设计模型,设计出工作流微服务组件结构模型,并基于 Spring Cloud 微服务开发框架给出组件间 REST API 与流程节点自由跳转算法,以及 CQRS 模式数据操作等方案。

1　研究背景

工作流技术顺应了信息时代“异构化、松耦合”的发展趋势[2]。工作流相关领域的专家学者对此做了许多研究。例如,于乐等[3]设计了云工作流在商业智能业务场景下的 SaaS 层应用;邓丽等[4]设计的空间科学任务协同论证平台论证了复杂任务下的协同性与一致性;李金艳等[5]实现了一种面向协作的基于角色的柔性工作流访问控制机制;陈儒等[6]研究了基于事务规则驱动的动态工作流模型;张型龙等[7]设计并实现了基于服务集成的工作流模型。针对工作流技术应用于传统单体架构时功能与业务逻辑耦合度较高所带来的扩展性差、可维护性差等问题,文中提出采用基于微服务的工作流技术,将面向分布式服务的工作流[8]应用于 CMP,以期实现对分布式云计算资源的调度管理。

1.1　工作流

工作流技术沿着以数据为中心的流程建模与柔性分布式设计的方向发展[9]。业界提出旨在提供易于理解和易于标识的业务流程建模标记(business process modeling notation,BPMN)。衍生的 BPMN2.0 规范可以直观和准确地描述业务流程的细节,降低在整个产品生命周期中按需开发的沟通成本;有利于业务逻辑与功能逻辑进一步解耦,降低工作流技术在复杂业务场景下持续集成新功能的实施成本[10];它更注重语法定义的通用性和交换格式的标准化,XML 格式规范便于移植在所有遵从 BPMN 规范的工作流引擎中解析利用。

1.2　微服务

微服务架构本质上体现的是解耦再封装[11]。由多个团队自由选择合适技术独立开发每个服务,只要遵守统一的系统内 HTTP 型无状态通信规范 API 即可,降低复杂系统的设计门槛,适合团队敏捷开发;每个服务均能被独立部署和验证,让持续部署成为可能,可以显著提高部署效率;只需要在特定的微服务组件中增加所需功能,而不影响整体业务和其他功能,整个微服务系统的可扩展性得到提升。微服务架构有别于面向服务的架构(service-oriented architecture,SOA),微服务架构各个服务间基于端到端的订阅方式,业务流程发生完全按照业务逻辑,当前事件的触发只需关注前置事件即可,需求变化带来的改变也只面向业务上下文而非功能,架构设计相当灵活。

2　工作流核心设计模型

主流的工作流引擎是 jBPM5 与 Activiti。通用性方面,Activiti 所采用宽松的 Apache License 2.0 开源协议不会在二次开发商用过程中引起不必要的著作权纠纷。易用性方面,微服务架构分布式的调用方式使 Spring Cloud 开发框架成为主流,Activiti 架构继承自 jBPM4,在整合已有 jBPM4 项目上具有原生优势,这也意味着它对 Spring 框架的原生支持[12],可以轻松集成基于 Spring 代理的事务管理以及基于业务逻辑的面向切面编程。

2.1　流程推进模型

使工作流流程推进模型首先要以满足以下定义作为前提:

定义 1(状态转换):从一个给定状态出发,只能进入有限状态机中其他状态的一个子集;服务随流程实例的演进,完成调用其他服务或是某些需要手动确认触发的业务流程,流程状态从当前节点转移到下一节点等待被触发。

定义 2(事务控制):下一事件只能在 BPMN 流程定义模型描述的前置事件完成之后完成,不能跳过任何事务步骤,否则不满足事务性。

定义 3(定量描述):BPMN 流程定义模型对流程的定性描述还需通过对各类流程数据持久化处理使其定量化。

BPMN 流程定义模型结合 Petri-Net[13-14] 和 UML 两种建模思路实现 Activiti 工作流的流程推进模型。由起点(StartEvent)、终点(EndEvent)、任务节点(Activity)与连线(Transition)组成的有向无环图(directed acyclic graph,DAG)即为流程定义(ProcessDefination),实例化的流程定义即流程实例(ProcessInstance),实例中流程执行体(ProcessExecution)相当于指针。每个实例被加载进工作流引擎都会首先触发 activityBehaviour()事件找到起点,紧接着会触发监听器(executionListener())执行起点的 end()事件并找到出方向连线事件 outgoingTransition();同样,触发监听器执行出方向连线的 take()事件找到第一个任务节点 Activity;触发监听器执行当前节点的 start()事件,为当前节点设置属性、指派任务或调用 API 后,触发监听器执行当前节点的 end()事件完成当前任务节点,并提交事务,将当前的状态保存到数据库里。该流程实例会暂停推进,直到外部请求再次触发监听器执行出方向连线的 take()事件找到下一任务节点,流程会继续向下一步骤推进。图 1 是流程推进模型分镜操作时序图,每次流程推进都由一次或多次上述步骤组成。

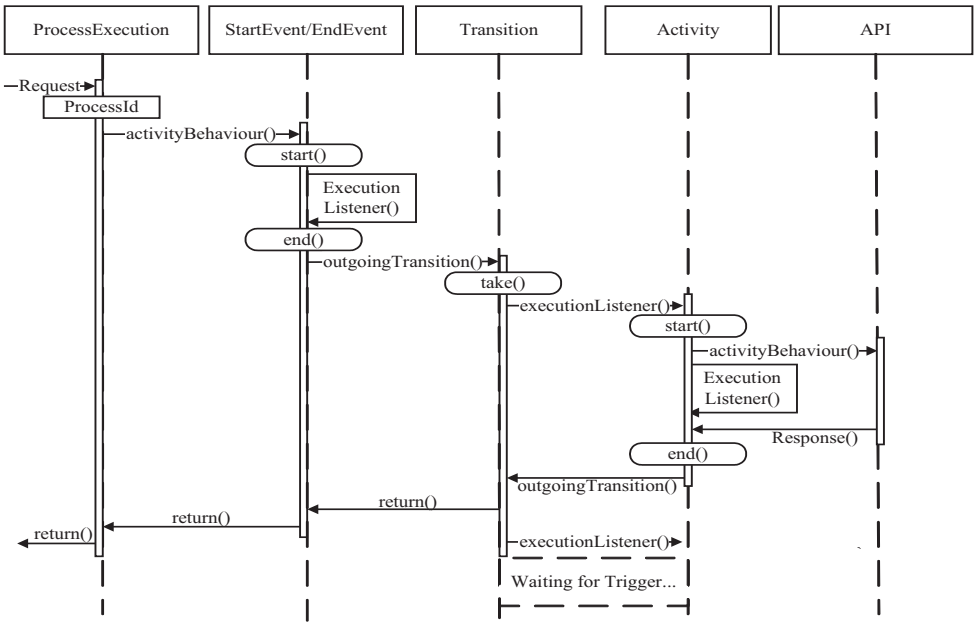

图 1 流程推进模型分镜操作时序

2.2 面向 API 调用

面向 API 调用是 Activiti Service 的设计思路。工作流引擎通过对外暴露 Service API(见表 1),依据流程定义部署流程实例、配置流程参数、指派任务执行人或代理人,实现审批、回退、驳回、创建等业务功能,完成工作流的内在功能逻辑。

工作流引擎未直接暴露的底层服务 API 也可以通过 ProcessEngine API 调用并借此完成节点自由跳转等特性功能。

由此衍生出的概念层面三种不同粒度的 API 封装模式如图 2 所示。

表 1 工作流引擎 Service API

名称	详情
RepositoryService	管理已部署的流程资源
RuntimeService	管理运行中的流程实例
HistoryService	管理流程实例历史
TaskService	会签、指派、查询任务
IdentityService	管理和查询用户、组间关系
ManagementService	引擎、数据库等配置项
ProcessEngine	获取流程引擎底层服务

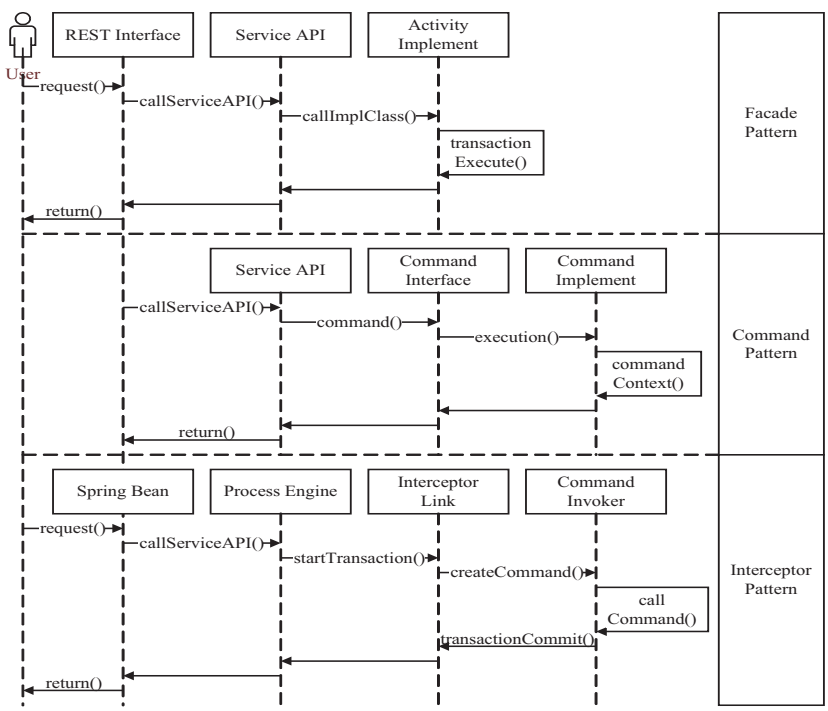

图 2 不同粒度的 API 封装模式

外观模式(Facade Pattern)下 Service API 是对功能逻辑的抽象定义,通过调用对外已经暴露的 ServiceAPI 或重写的自定义 ServiceAPI 实现功能逻辑;命令模式(Command Pattern)实质上是将不同命令(Command)交由命令执行器(CommandExecutor)统一执行;拦截器模式(Interceptor Pattern)下,命令模式执行当前某个命令的过程,对应于在当前命令执行前后分别执行若干前置或后置拦截器的过程,旨在实现最细粒度的事务控制。ProcessEngine API 在外观模式层面将工作流引擎内部作透明处理,屏蔽底层架构实现层面上复杂的技术细节,使得调用服务不必关心引擎事务规则和流程定义等的具体实现。

3　工作流微服务组件设计方案

微服务架构(见图 3)的设计理念在于,有编排功能的基于业务逻辑的架构可以减少耦合,在此基础上所有的微服务组件可以订阅与其业务相关的其他服务所发生的事件,进而对事件做出响应,最终通过 REST 服务完成功能。采用 API 网关机制聚合系统外部请求和映射系统内部服务,内部每个微服务组件提供 REST API。请求基于 HTTP 协议通过 API 网关分发到目标服务,可以降低外部请求与具体服务的耦合度,减少通信频次。采用去中心化的服务注册机制,每个微服务组件的服务注册节点之间没有主次之分,针对注册节点宕机问题提供自动检测及周期恢复功能,所有组件通过引用服务发现客户端,在每个组件入口类上添加服务发现注解后即可完成服务注册。

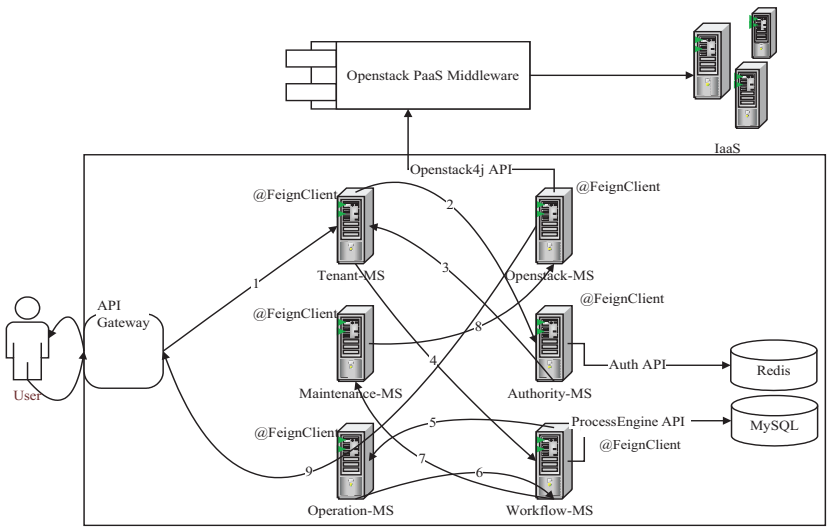

图 3　CMP 微服务架构示意

3.1　微服务框架与工作流结构模型

面向领域模型(domain-oriented model,DOM)可以准确反映业务语言,真正实现以业务逻辑实体为核心的灵活拓展;基于“Spring+Hibernate”传统架构的 J2EE 应用关注功能逻辑过于业务逻辑,而 Spring Cloud 依据领域划分微服务,以领域模型替代“功能服务+数据表”模型,以并发的业务事件驱动替代传统消息总线数据驱动。基于微服务的工作流微服务组件设计延续了 Spring Cloud“约定优于配置”的契约式编程思想,bootstrap. yml 配置文件依据开发环境在 spring. profiles. active 配置项激活目标环境;Java 持久层 API(Java persistence API,JPA)将数据表名和字段名按照约定映射到方法类,构造面向对象的查询语句,以非侵入式设计灵活操作流程数据。

架构层面实现不同服务间功能解耦,工作流组件只需暴露 API 被其他服务调用,解决复杂业务逻辑的同时,在后续持续集成需求开发新业务功能时,可以在不影响业务逻辑和 API 原有设计前提下对工作流组件重新设计开发。工作流组件如表 2 所示。

表 2　工作流组件概览

流程管理层	事务规则引擎层	服务接口层	持久层
流程管理器	事务规则库 解释器 组合器	Service API	Mysql
过程模型库	执行器		

工作流组件运行的具体步骤如图 4 所示。

(1)流程管理器访问 BPMN 定义模型库,读取当前流程定义进入工作流引擎;

(2)由事务规则引擎驱动过程模型的启动和执行,事务规则引擎负责依据 XML 设计规范解析执行过程模型文件;依据事务规则库中的事务规则子模块对业务流转实现事务控制;

(3) Service API 完成当前步骤并持久化流程数据,并且通过暴露的 REST API 被其他微服务调用;

(4) 当前步骤执行完成后,工作流引擎处于等待状态直到流程监听器相应请求被触发。

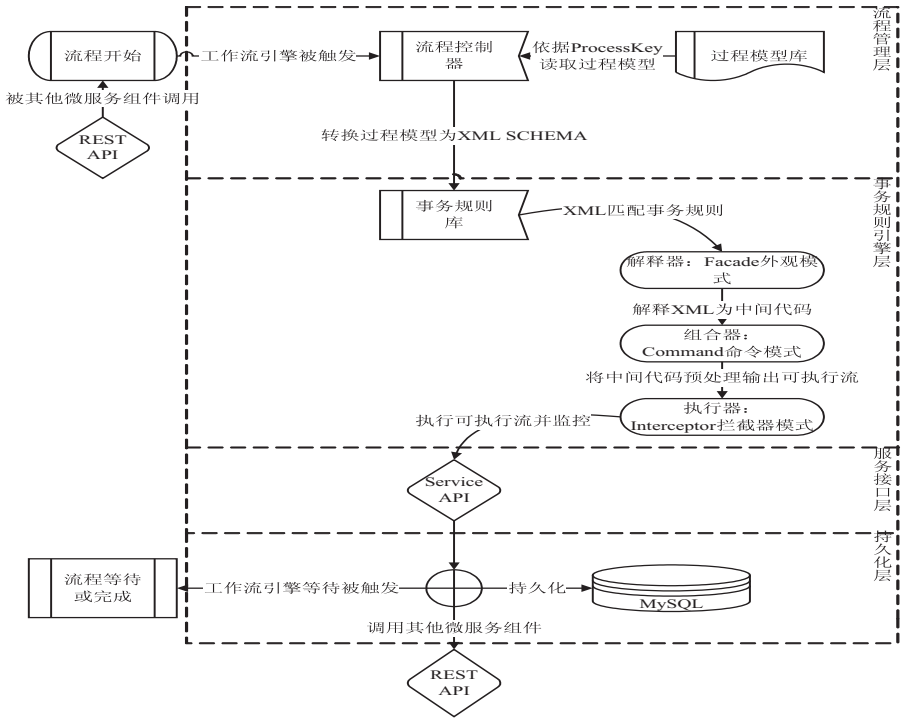

图 4 工作流组件运行示意

3.2 REST API 与流程节点自由跳转设计

CMP 中每个微服务可能是某个事件的生产者、消费者或二者兼有。当租户客户端提交云资源订单申请,租户服务端微服务组件作为当前事件的消费者向工作流组件 REST API 发送 HTTP 请求传入流程参数;工作流组件作为当前事件的生产者,调用业务层 nextStep( )方法类,完成持久化和参数传递驱动当前流程向下推进直到完成当前事件;工作流又作为当前事件的消费者,完成当前事件过程中向 OpenStack-MS 微服务组件 REST API 发送 HTTP 请求;OpenStack 组件以生产者身份参与当前事件,依据 API 参数分配云硬盘、云主机等云资源。

基于微服务架构的工作流组件将传统工作流系统中基于每项具体业务功能设计的调用 API 设计为“下一步”(nextStep( ))、“回退”(rollback( ))、“开始”(startProcess( ))和“结束”(finishProcess( ))四个 REST API,屏蔽每次请求的具体业务功能细节;基于 Java 泛型设计的动态实体类满足无状态 JSON 格式传入参数并将其代入流程参数传递。如订单审批工作流被调用 startProcess( )方法类的同时,会从 API 参数获取到诸如订单主题、备注、审批需求等订单信息传入当前流程实例的流程参数在流程中传递;针对特性需求,基于 Java 的继承特性,使得子类继承基类,根据事务需求灵活扩展开发。针对流程节点的自由跳转的业务需求,预设流程输出流程不能很好满足,以下伪代码实现了对引擎底层未暴露的 Command API 二次调用。

算法:rollback( )两步回退算法。

(当前节点 CurrAct;当前节点出方向 OutgoingTrans;当前节点出方向列表 OutgoingTransList;当前节点原始出方向 OriOutgoingTrans;当前节点入方向 CurrTrans;当前节点入方向列表 CurrTransList;上一步节点 PreAct;上一步节点入方向 PreTrans;上一步节点入方向列表 PreTransList;上两步节点 MultiPreAct;当前节点新出方向 NewTrans;当前节点新出方向列表 NewTransList;)

```
for(OutgoingTrans:OutgoingTransList){
OriOutgoingTrans=OutgoingTrans;}
OutgoingTransList.clear();
for(CurrTrans:CurrTransList){
PreAct=CurrAct;
for(PreTrans:PreTransList){
MultiPreAct=PreAct;
NewTrans.setDestination(MultiPreAct);
NewTransList.add(NewTrans);
taskService.complete();
historyService.deleteHistoricTaskInstance;}}
NewTransList.clear();
OutgoingTransList.add(OriOutgoingTrans);
```

3.3 命令查询职责分离模式数据操作

传统单体架构应用采用 CRUD 设计模式(Create/Retrieve/Update/Delete,“增加/读取/更新/删除”),持久层操作和查询一种类型的数据通过相同的实体类。微服务架构系统要求业务逻辑的实现从数据驱动

(Data-Driven)转向任务驱动(Task-Driven)和事件驱动(Event-Driven)。工作流微服务组件采用“命令查询职责分离”(command query responsibility segregation,CQRS)设计模式,为了使数据操作 Command(会修改系统状态的增删改操作)和数据查询 Query(不会修改系统状态的查询操作)分离。图 5 展示了 CQRS 与 CRUD 模式的异同。

涉及的 CQRS 中所有涉及到对数据库的改变均通过发送 Command 异步触发对应事务性操作,避免了 CRUD 中操作和查询同时进行可能导致的事务冲突;此外面向切面编程思想,通过添加“@ Transactional”注解实现每个方法类粒度上的事务管理,从而最大限度保证对数据库事务操作的原子性和一致性;针对 CMP 资源订单的实时流程状态和历史流程状态进行列表分页展示的需求,基于 CQRS 设计模式,针对实时流程数据和历史流程数据设计独立的服务调用 REST API,被请求时根据请求参数诸如租户 ID、流程 ID、执行步骤、关键词等列表分页展示条件查询或精确查询结果。

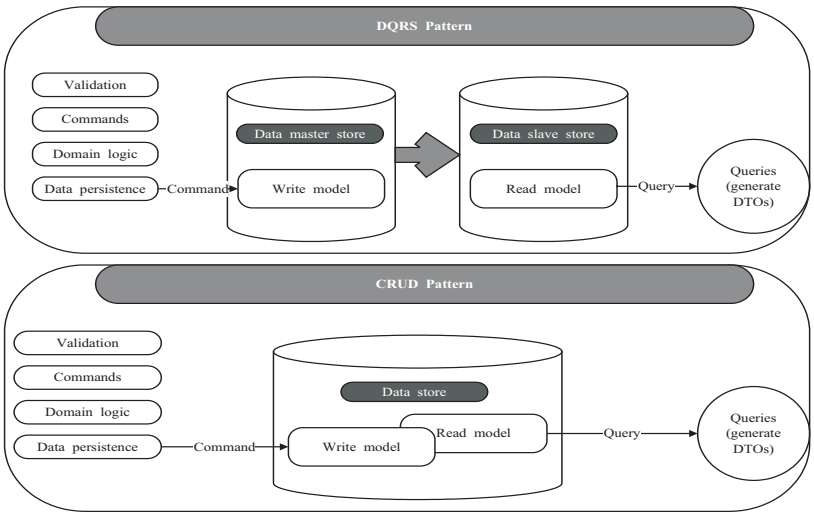

图 5　CQRS 与 CRUD 模式的异同示意

4　结束语

文中设计了 CMP 工作流微服务组件,发挥 Activiti 工作流引擎整合 Spring Cloud 微服务框架的原生优势;系统内 API 调用采用 REST 风格设计,最大限度上保证无状态请求的效率最大化,降低开发难度;流程节点自由跳转有助于有效完成工单订单审批流程在复杂业务场景下的特性需求;CQRS 数据操作方案旨在解决复杂业务场景下功能逻辑的高耦合和低效操作下可能产生的误操作和资源浪费。CMP 在 OpenStack Pike 版本云环境上验证了其在云资源调度管理上的可行性。开发改进方面,可以通过容器部署微服务应用[15],配置文件采取集约管理,以 GitServer 的方式按照不同开发环境依赖要求,实现配置文件集群部署。

参考文献:

[1] 杨　涛,石　琳,宋梦蝶,等. 支持多工具协同的流程管理系统的设计与实现[J]. 计算机应用,2017,37(7):2019-2026.

[2] 柴学智,曹　健. 面向云计算的工作流技术[J]. 小型微型计算机系统,2012,33(1):90-95.

[3] 于　乐,赵　帅,章　洋,等. 云工作流技术在商业智能 SaaS 中的应用[J]. 计算机集成制造系统,2013,19(8):1738-1747.

[4] 邓　丽,韩　潮,曹晋滨,等. 空间科学任务协同设计论证平台[J]. 北京航空航天大学学报,2015,41(4):601-608.

[5] 李金艳,余忠华. 面向协作的柔性工作流访问控制机制[J]. 计算机集成制造系统,2017,23(6):1234-1242.

[6] 陈　儒,肖　刚,张元鸣,等. 基于事务规则的面向服务工作流模型研究[J]. 计算机应用与软件,2014,31(6):5-8.

[7] 张型龙,李松犁,肖俊超. 面向服务集成的工作流模型及其实现方法[J]. 计算机应用,2015,35(7):1993-1998.

[8] 于新征,蒋哲远. 面向服务的云工作流模型与调度研究[J]. 微电子学与计算机,2016,33(7):44-48.

[9] 李竞杰,王维平,杨　峰. 柔性工作流理论方法综述[J]. 计算机集成制造系统,2010,16(8):1569-1577.

[10] 董荣胜. 计算思维的结构[M]. 北京:人民邮电出版社,2017:134-135.

[11] 纽　曼,崔力强,张　骏. 微服务设计[M]. 北京:人民邮电出版社,2016:8.

[12] 闫洪磊. Activiti 实战[M]. 北京:机械工业出版社,2015:3.

[13] 焦合军,张　璟,李军怀,等. 协同设计中基于混合 Petri 网的云工作流表示模型[J]. 应用科学学报,2014,32(6):645-651.

[14] 郑学恩,许承东,范国超,等. 基于面向对象 Petri-net 的 LWF 建模方法[J]. 系统工程与电子技术,2018,40(7):1626-1632.

[15] 崔　蔚,李春阳,刘　迪,等. 面向微服务的统一应用开发平台[J]. 电力信息与通信技术,2016,14(9):12-17.