工作流自动伸缩模型, 结合流程负荷自动伸缩流程的容器数目以提升流程引擎服务性能, 但不太适用于顺序工作流. 因此, 主流工作流框架在柔性适配流程开发和运维方面还面临着一些问题和挑战, 主要表现在: (1) 业务活动的简单变更需要修改流程模板、流程表单及流程迁移事件, 但修改无法即时生效, 增加开发运维成本并影响生产效率. (2) 流程活动参与者范围筛选计算复杂, 影响业务开发效率. (3) 顺序工作流引擎的运行时扩展柔性不足, 云计算结合容器技术虽然让资源扩缩变得方便, 但依然需要人工手动操作, 智能和自动化程度较低.

针对上述问题, 本文提出一种容器环境下基于领域驱动设计的工作流开发框架, 将企业应用根据领域模型细分后, 拆分为分工明确的核心子域微服务、边界上下文和业务微服务, 通过容器作为微服务的载体, 灵活集成企业工具链, 在提高服务质量、可扩展性和可靠性的同时, 降低了企业运行成本, 具备以下特征: (1) 支持基于领域模型驱动的在线流程表单零代码开发、运行模型变更和属性状态权限控制; (2) 支持动态的业务组织关键点设计, 柔性适配活动参与者, 支持流程活动脚本的运行时更新. (3) 结合监控微服务的容器服务柔性伸缩策略及控制.

2 容器化柔性微服务工作流开发框架设计

本文设计的容器化柔性微服务工作流开发框架结构如图 1 所示. 通用服务和业务服务都以 Docker 镜像方式发布, 并推送到容器镜像仓库, 通过 K8S 容器管理组件从容器镜像仓库拉取镜像并部署到容器池中. 数据库不做容器化部署. 框架的结构说明如下.

(1) 所有微服务统一走网关路由, 微服务网关采用集群方式部署, 支持高可用路由和基于安全策略的最小权限访问控制.

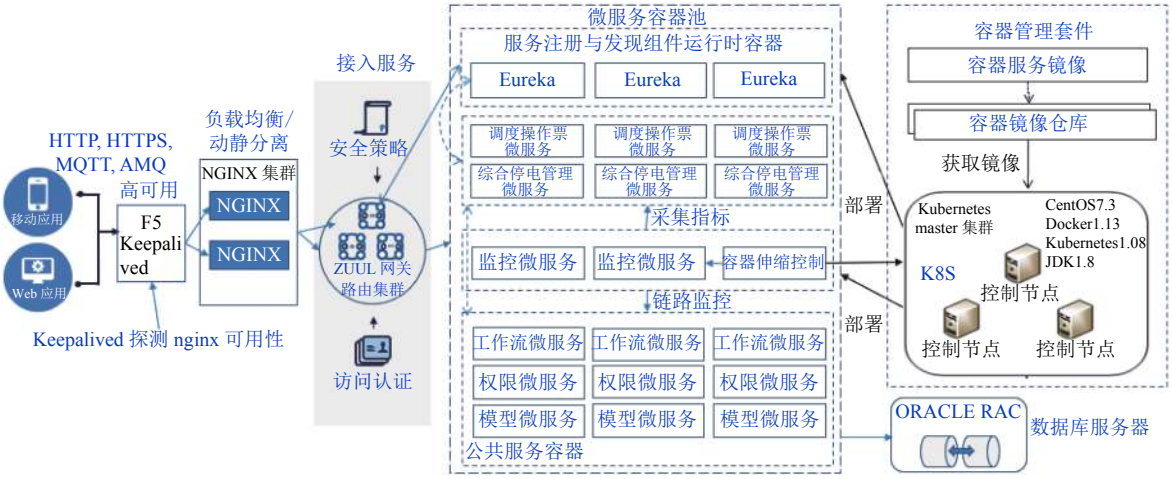


图 1 容器化微服务流程开发框架结构

(2) 所有服务以容器化方式启动运行并注册到 Spring Cloud Netflix Eureka[7]中, 接入监控代理, 监控代理将容器及服务的监控数据发送给监控服务, 实现 Docker 容器及微服务的服务水平质量监控.

(3) 权限微服务提供基于 RBAC 的访问控制, 并支持业务组织管理、用户管理、角色管理、菜单管理、功能管理等.

(4) 工作流微服务提供基于 Web 的可视化流程建模和发布、监控管理. 其中工作流模型的机构模型主要描述使用者的角色、权限分配、活动状态以及组织机构关系等, 并基于权限微服务提供的组织、角色和用户进行配置和解析.

(5) 模型微服务提供基于 Web 在线运行时领域对象模型建模功能, 支持数据源、业务字典、编辑器模板、业务对象模型、展现模型、应用控制模型等模型的建模, 其中, 业务对象模型是业务表的抽象, 并增加属性编辑器模板的绑定等约束, 属性编辑器模板和业务字典绑定, 支持运行时表单编辑, 应用控制模型提供业务展现场景视图基于状态的对象和属性访问控制配置.

(6) 监控微服务提供对包含自身在内的所有微服务的 URL 链路服务质量监控, 容器 CPU、内存、线程等资源的实时监控; 基于监控指标的容器伸缩策略配置, 并根据服务质量状态伸缩策略进行实时告警推送和容器伸缩控制.

3 容器化柔性微服务工作流开发框架实现

一个完整的工作流模型由 3 个子模型组成: 其中组织模型描述工作流执行中的组织实体, 框架中基于通用权限服务配置业务组织实体以及角色、用户等实体; 信息模型描述工作流执行中的信息项, 框架中基于领域建模实现; 过程模型描述工作流的功能, 框架中基于可视化流程建模配置及解析实现. 微服务工作流子模型之间的实体关系形式化描述如图 2 所示.

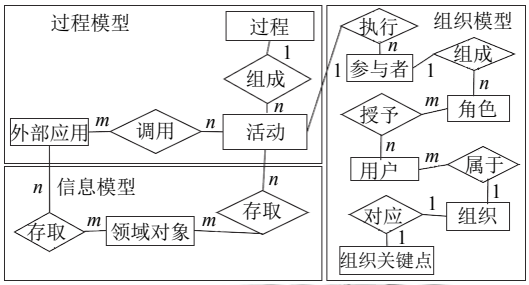


图 2 微服务工作流子模型的实体关系描述

结合微服务工作流子模型实体关系, 容器化微服务工作流开发框架的开发流程如下:

(1) 通过模型驱动微服务提供的建模视图按业务领域建模, 构建领域对象模型, 并基于领域对象模型构建业务表单模型. (2) 通过工作流微服务提供的流程设计器构建业务流程, 设置流程启动者和参与者, 限定参与者为指定组织用户, 在流程活动模板中绑定表单模型, 并允许在表单中设置流程活动状态的表单属性访问控制. (3) 通过可视化界面设计器[8]设计界面, 支持最终用户的业务流程操作流转.

根据上述开发流程, 需要实现 3 个主要关键技术组件: (1) 基于模型驱动微服务的零代码流程开发套件; (2) 基于通用权限微服务的流程活动组织关键点策略和流程脚本的热更新机制; (3) 结合监控微服务的容器弹性伸缩策略和伸缩控制.

3.1 基于领域驱动微服务流程零代码开发

模型微服务基于多租户模式, 实现了基于领域模型驱动的在线业务对象模型建模、流程表单模型建模、和应用控制属性状态管理, 模型关系如图 3 所示. 领域对象模型具体构建方式为: (1) 首先基于分层架构将领域层划分出来, 领域专家划分领域为合适的子域、边界上下文、实体、值对象和服务; (2) 研发人员基于子域分包建模, 通过可视化建模视图构建领域包、数据源、业务对象类型和类型属性、类型关联. 表单建模是根据业务需求构建应用控制和表单模型, 应用控制模型设置对业务类型和基于状态的类型属性的访问控制, 表单模型引用应用控制模型, 在表单模型中设计表单属性的编辑器模板, 添加业务字典并和编辑器绑定以支持界面交互. 最后创建流程模型, 在流程活动模板中绑定表单模型并发布流程, 通过界面设计器设计界面并绑定对象类型和流程参数完成零码开发.

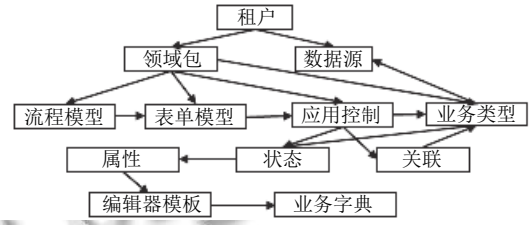
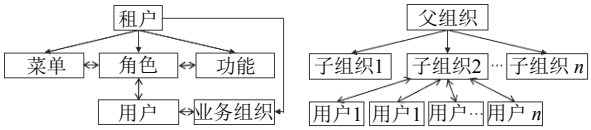


图 3 领域模型构建

3.2 基于通用权限微服务的流程活动关键点设置和活动脚本动态更新

通用权限微服务提供了多租户的权限管理功能, 并支持基于网关和安全策略的统一权限认证, 实现了基于租户的业务组织、菜单、功能等资源管理, 并通过角色和资源及用户的绑定实现权限解耦和授权, 如图 4(a) 所示. 服务之间的认证采用轻量级 JWT 认证[9].



(a) 资源授权 (b) 组织关键点

图 4 通用权限服务的资源授权及组织关键点

工作流引擎微服务基于开源引擎 Activiti 实现, Activiti 提供了工作流存储、运行时、任务、身份、管理、历史、表单等 7 类流程服务接口, 框架对其身份接口进行扩展, 集成框架通用权限微服务, 允许操作者将流程活动参与者设置为和流程启动者或上一个活动的发送者相同的业务组织, 实现了基于通用权限业务组织的活动关键点功能: 给出业务组织列表, 允许操作者将指定的业务组织设置为活动关键点, 如图 4(b) 所示. 当发送流程时, 当发送者隶属于某已设置的关键点组织时, 则将发送范围限定为该关键点内具有可接受任务角色的用户, 解决了活动参与者设置业务的复杂计算等问题. 框架采用 JSON 描述流程定义, 用关键字描述了图 2 中子模型之间的关系, 其中流程、活动、迁移线都抽象为资源对象, 表单作为活动资源对象的子对象, 通过迁移事件支持二次开发. 如图 5 所示. 其

中 user 表示活动参与者, 以角色描述, formtemplate 表示表单模型, overlay 表示组织关键点, stateId 表示应用控制模型状态. event 表示事件, action 表示事件方法.


```
{"resourceId": "", "properties": { "process_id": "", "name":
"流程名"},"childShapes": [{"resourceId": "",
"properties": {"user": [{"roleID": "", "roleName": ""}]
}, "overlay": {"key": "", "keytext": "."}, "event": [{
"event": "start","action": ""}], "name": "", "formtemplate":
[{"name": "","id": "","fields": {"stateId": ""}}]}]}
```


图 5 微服务流程的形式化定义

框架提供流程脚本管理界面, 支持将流程脚本 jar 包上传到流程微服务内存及数据库中, 由于流程活动脚本相对独立, 利用 JVM classloader 的动态代理机制, 实现了流程活动脚本运行时即时热更新, 避免了需求变更升级脚本时对运行时生产流程的影响.

3.3 结合监控微服务的容器弹性伸缩策略

资源使用率通常都有峰谷期, 传统工作流框架无法根据任务量或任务到达速率进行动态伸缩来提高集群的整体资源利用率和服务质量, 本文设计的框架是通过基于监控策略的监控框架对微服务容器和微服务进行监控[10], 全自动的收集 Docker 容器实例、微服务实例及服务接口的信息.图 6 给出了结合监控微服务的容器弹性伸缩策略控制过程. 其中容器弹性伸缩策略支持配置两种策略算法: (1) 基于阈值的响应式伸缩算法实现: 定义容器伸缩配置 yaml 文件, 作为 K8S 创建容器的配置, 周期性地采集微服务宿主 Docker 容器的 CPU、内存、网络、IO 等指标占用率, 以及服务平均响应时间等, 通过与设定阈值进行比较从而动态改变集群数量, 为了避免瞬时资源消耗峰谷造成的频繁伸缩, 在策略的配置上增加了伸缩延迟时间配置. (2) 基于预测的伸缩算法实现: 主要采用时间序列分析和机器学习两类, 其中时间序列分析主要提供了简单移动平均法和指数平滑法两种算法; 机器学习法采用了 BP 神经网络预测模型[11], 将正确的结果和产生的结果进行比较, 根据误差逆推对神经网络中的权重进行反馈修正, 从而来完成学习的过程.

4 案例分析与评估

为了验证本文提出的容器化微服务工作流开发框架, 在一个真实的环境中进行了案例的部署和研究. 案例环境由 3 台 8 核 32 GB 内存的浪潮服务器 (SA5248L) 组成, 其中数据库采用虚拟机部署, 网关微服务、服务注册微服务、工作流微服务、权限微服务、模型驱动微服务、监控微服务采用容器部署, 容器副本数均默认为 3. 数据库和应用服务采用某电力公司综合停电管理系统的测试数据库和经过领域划分的微服务, 微服务实例以 K8S 部署的 Docker 容器形式运行.

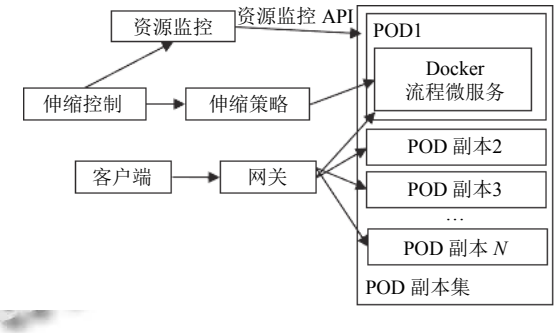


图 6 基于监控微服务的容器弹性伸缩

部署完成后, 由两名业务配置人员经过培训后基于停电检修流程所涉及的领域对象、表单和流程建模, 并根据流程活动和应用控制状态设置流程表单对象的不同字段的动态访问控制权限. 其中检修流程活动包括方式专责初审、通信处和调度处等各部门会签、方式处审批、网公司审批、调度执行、归档等活动环节, 检修流程定义的形式化描述如图 7 所示.


```
{"resourceId": "e8ae0e9f42...","properties": {  "process_id":
"DDJXSQDLC","name": "地调检修申请单流程},
   "childShapes": [{"resourceId": "sid-C8292D15-*",
      "properties": {"name": "开始", "id": "StartNoneEvent"}}, {
       "resourceId": "sid-38E52D9D-CB92-*",
      "properties": {"user": [{" roleID": "bf1bd3afa828...",
     "roleName": "方式专责"}], "overlay": {"type": "starter",
"key": "b2b420c....,3926ebc2d...,...", "keytext": "潮州供电局,
梅州供电局,..."},  "name": "方式专责初审",
     "formtemplate": [{"name": "检修申请单表单",
     "id": "3d5fb90e0...", "fields": {"stateId": "e16028df..."}}]}},
 { "resourceId": "sid-3925FC7A-*",
       "properties": {  "name": "网公司审核"}, {
        "resourceId": "sid-890313DA-*",
        "properties": {"user": [{"roleID": "bf1bd3afa8...",
        "roleName " : "方式专责"}],  "name": "方式审批"}
 }, {"resourceId": "sid-F4772913-*", "properties": {"user":
[{"roleID ": "d540ebc2...", "roleName": "通信专责"}]...}},
{"resourceId": "sid-5DF99270-*", "properties": {"user": [{"
roleID ":"cc73a7d0a...", "roleName": "调度专责"}], "name": "调
度执行"}}, {"resourceId": "sid-58EE-*", "properties": {"name": "
结束"}}, {"resourceId": "sid-2C201182-*", "properties":
{"name": "归档","servicetaskclass":
"aadd1dc...*!com.nariit.bpm.AutoChangeStateForDDJXRWD"}}]}
```


图 7 停电检修流程的形式化定义

在检修活动表单授权时, 每个检修流程活动对应于一个检修单状态, 状态是类型属性和属性权限的属性安全域, 并基于角色设置属性的访问控制, 状态和流

程活动匹配, 以支持运行时获取表单及属性访问控制. 如图 8 所示. 将基于模型的表单和流程活动挂接, 设置流程活动参与者及关联组织关键点, 组织关键点的定义和选择是以省内地市供电局和电厂为关键点, 这样地市供电局和电厂的用户在处理流程时, 流程活动的参与者被限定为当前供电局或电厂. 经评估, 相同人工下较之前代码开发方式提升开发效率 35% 以上.

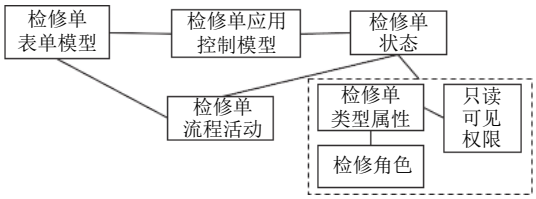


图 8 检修流程表单类型属性权限授权模型

在运行时工作流等微服务性能弹性伸缩方面, 通过 loadrunner 模拟并发 300 用户的登录、检修流程创建、保存、发送、审批等流程活动, 通过监控策略和容器弹性伸缩策略, 即监控每 3 分钟内服务容器 CPU 和内存占用率, 15 分钟后取监控周期内 CPU 和内存占用平均值, 当 CPU 或内存平均占用率达到 80% 时, 启动容器扩容操作, 自动增加 1 个指定服务的容器, 当 CPU 或内存平均占用率低于 30% 时, 删除指定服务的 1 个容器服务实例, 以避免因为监控数据抖动而引起频繁的扩容或缩容操作, 观察容器化微服务流程开发框架的自动弹性伸缩能力和服务稳定性, 结果见表 1.

表 1 检修流程性能压力测试

300 用户并发执行检修流程持续 30 分钟平均值					
Docker 容器		业务性能			
CPU%	MEM%	吞吐率 (M/s)	响应时间 (s)	每秒事务数	事务失败率
39.8	59.5	2.06	1.15	5.329	0

验证结果表明, 柔性微服务工作流开发框架能有效根据资源使用率的峰谷进行容器服务伸缩, 保证了工作流等微服务的运行时性能、扩展能力和高可用, 但也发现了一些问题, 针对小规模的应用现有框架的部署架构略显复杂, 增加了运维难度, 尚有提升空间. 目前, 柔性微服务工作流开发框架已在南网某省综合停电管理系统中实现并应用, 取得了较好的应用效果.

5 结束语

本文研究了传统工作流框架建模、领域驱动设计、容器化微服务运行时自动伸缩等技术, 在此基础上, 设计了容器化柔性微服务流程开发框架, 给出了基于领域驱动设计的微服务的零代码流程开发套件、基于通用权限微服务的流程活动组织关键点适配和流程脚本热更新机制、结合监控微服务的容器弹性伸缩策略及伸缩控制等创新点, 阐述了该框架的架构设计及关键实现技术. 最后, 以南方电网某省综合停电管理系统的检修单应用案例为背景, 给出了柔性微服务工作流开发框架的应用验证评估, 验证结果、效率评估及生产运行实践表明, 该框架提升了分布式环境下微服务工作流应用开发和管理的灵活度、效率, 实现了基于容器的工作流等微服务运行时柔性伸缩能力, 提高了电网应用信息系统的服务水平. 后续将针对遗留问题持续改进优化该框架.

参考文献

1 Bernstein D. Containers and cloud: From LXC to docker to kubernetes. IEEE Cloud Computing, 2014, 1(3): 81–84. [doi: 10.1109/MCC.2014.51]
2 Deelman E, Singh G, Su MH, *et al*. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. Scientific Programming, 2005, 13(3): 219–237. [doi: 10.1155/2005/128026]
3 Guan ZJ, Hernandez F, Bangalore P, *et al*. Grid-flow: A Grid-enabled scientific workflow system with a Petri-net-based interface. Concurrency and Computation: Practice and Experience, 2006, 18(10): 1115–1140. [doi: 10.1002/cpe.988]
4 Netto HV, Lung LC, Correia M, *et al*. State machine replication in containers managed by Kubernetes. Journal of Systems Architecture, 2017, 73: 53–59. [doi: 10.1016/j.sysarc.2016.12.007]
5 尚世锋, 姜进磊, 郑纬民. CWFlow: 支持资源自适应使用的云工作流框架. 清华大学学报 (自然科学版), 2013, 53(3): 415–420.
6 刘彪, 王宝生, 邓文平. 云环境下支持弹性伸缩的容器化工作流框架. 计算机工程, 2019, 45(3): 7–13.
7 Spring cloud Netflix. https://spring.io/projects/spring-cloud-netflix.
8 刘一田, 刘士进. 可视化 Web 设计器. 计算机系统应用, 2015, 24(10): 80–84. [doi: 10.3969/j.issn.1003-3254.2015.10.013]
9 刘一田, 林亭君, 刘士进. 柔性微服务安全访问控制框架. 计算机系统应用, 2018, 27(10): 70–74. [doi: 10.15888/j.cnki.csa.006568]
10 刘一田, 刘士进, 郭伟, 等. 柔性微服务监控框架. 计算机系统应用, 2017, 26(10): 139–143. [doi: 10.15888/j.cnki.csa.006006]
11 张景阳, 潘光友. 多元线性回归与 BP 神经网络预测模型对比与运用研究. 昆明理工大学学报 (自然科学版), 2013, 38(6): 61–67. [doi: 10.3969/j.issn.1007-855x.2013.06.010]