计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(9):224−231 [doi: 10.15888/j.cnki.csa.005962]

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

面向工作流的 Gitlab 服务化设计①

张宇光[1,2], 王俊杰[1], 胡渊喆[1], 王　青[1,2,3]

[1](中国科学院软件研究所 互联网软件技术实验室, 北京 100190)
[2](中国科学院大学, 北京 100049)
[3](中国科学院软件研究所 计算机科学国家重点实验室, 北京 100190)

摘　要: 软件制品间使用服务化的方法进行交互, 能够将制品内部产生的事件和数据以服务的形式产出. 工作流是一种常见的用于处理消息流和事件流的方式, 使得制品产生的事件和数据能够在不同制品间流转. Gitlab 作为开源的、被广泛使用的代码与文档管理工具, 针对其在工作流环境中进行对外服务能力的研究是很有意义的. 在研究中发现, Gitlab 的服务化能力有灵活性差, 拓展能力弱, 服务粒度粗等问题. 基于这样的问题, 本文提出 GITService 服务化方法, 重新定义了 Gitlab 的服务流程和服务标准, 在服务实现中, 使用消息队列和异步机制, 设计实现了新的监听服务和执行服务方案, 并针对该解决方案进行了实验分析. 实验表明, 在保证 Gitlab 运行时间几乎不受影响的情况下, GITService 拥有灵活性好、扩展性强、细粒度的服务化能力. 本文所提供的方法, 能够为其他场景下的服务化设计和实现提供有益参考.

关键词: Gitlab; 工作流; 服务化; RESTful; 服务粒度

引用格式: 张宇光,王俊杰,胡渊喆,王青.面向工作流的 Gitlab 服务化设计.计算机系统应用,2017,26(9):224–231. http://www.c-s-a.org.cn/1003-3254/5962.html

Design of Workflow Oriented Gitlab as a Service

ZHANG Yu-Guang[1,2], WANG Jun-Jie[1], HU Yuan-Zhe[1], WANG Qing[1,2,3]

[1](Laboratory for Internet Software Technologies, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
[2](University of Chinese Academy of Sciences, Beijing 100049, China)
[3](State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

Abstract: Web service could make events and data generated in software products as service by which software products interact with each other. Workflow is a very popular way to deal with message flow and event flow which could deliver events and data among software products. It is very meaningful to study Gitlab which is known as a widely used open source code and document management tool. However, poor flexibility, weak expandability and raw service granularity are found in Gitlab service during the study. To tackle the problems in Gitlab, this paper proposes a new service solution to Gitlab which redefines the service process, service standard and service implement. To implement the services, we design and implement new solutions for listening services and execution services which involve the message queue and asynchronous mechanism. According to experimental analysis to this solution, GITService has high flexibility, strong expandability and intensive service granularity with little cost of time. The solution we provide in this paper is of significance to design and implementation of service in other situations.

Key words: Gitlab; workflow; software as a service; RESTful; service granularity

① 基金项目: 国家自然科学基金(61432001, 61602450)
收稿时间: 2016-12-29; 采用时间: 2017-01-20

1 引言

工作流系统的作用是“使在多个参与者之间按照某种预定义的规则传递文档、信息或任务的过程自动进行, 从而实现某个预期的业务目标, 或者促使此目标的实现”[1]. 为使信息能够在工作流运行的过程中流转, 就需要由工作流串联起来的各个工具之间能够满足工作流传输数据的标准, 满足工作流对于实际业务的需求.

在软件开发过程中, 为了实现复杂的开发和管理流程, 通常需要多个开发工具之间的事件交互和消息交互, 在实践中, 经常选择工作流平台来实现软件开发工具间的事件和消息的流转. 在自动化的软件开发与管理过程中, 会有多个软件制品协同对外提供服务, 由于软件开发与管理的复杂性, 其所需要的服务种类多种多样, 这就对各个制品的对外服务能力提出很高的要求. 代码与文档管理作为软件开发过程的核心部分, 其对外服务能力直接影响到软件开发与管理的质量. Gitlab 作为软件开发过程最重要的文档及代码管理工具, 在接入工作流平台后发挥了重要的作用, 在工作流平台中被频繁使用.

被接入工作流后, Gitlab 现有的服务化方案展现出了诸多问题, 在灵活性方面, Gitlab 的服务化参数是固化的, 不能通过解析参数的方法灵活地选取服务; 在拓展性方面, Gitlab 只能提供固定的服务, 当前架构难以对服务拓展; 在服务粒度方面, Gitlab 只能在 project 和 group 的维度提供服务, 不能提供更细粒度的服务. 对工具的服务化而言, 文献[2,3]研究了通用领域的服务选择和组合方法. 文献[4]指出, 由于 Web 应用的广泛性, 在诸多种类 Web 中进行 Web 服务化的相关研究时, 只有考虑 Web 服务所应用的领域才有意义, 本文基于 Gitlab 在软件开发领域的重要作用, 研究发现 Gitlab 现有的服务化方案存在的三类问题, 并提出了针对该专用领域的解决方案.

针对在与工作流协同工作时, Gitlab 所展现出的灵活性差, 拓展性差和服务粒度粗等情况, 本文对 Gitlab 进行了深入研究, 在兼容原有对外服务体系的基础上, 提出了更灵活, 拓展性更强, 细粒度的服务化解决方案 GITService, 实验表明 GITService 提升了 Gitlab 的服务化能力.

2 相关工作

罗海滨等[5]结合不同研究者的定义, 将工作流定义为, 工作流是通过计算机软件进行定义执行并监控的经营过程, 而这种计算软件就是工作流管理系统. 文中对工作流技术进行了全面而详尽的综述.

文献[6,7]提到目前有很多研究基于 Web service 展开, Web 服务化有诸多优点. 张卫等[8]结合 HLA 仿真与 Web 的优点, 通过服务化的方法实现了分布式的 HLA 仿真系统. 将 Web 服务的优势附加给 HLA, 实现了 HLA Web 服务化的三个层次. 文章借助 Web 服务化的优势, 对软件结合 Web 实现服务化能力提出了一种实践方案.

文献[9-12]提到 RESTful Web 服务的简便性、轻量性、可扩展性和安全性, RESTful 架构已经成为 Web 服务的主流技术. 它基于 RESTful Web 服务的特点, 提出了有服务发现与识别、服务搜索和服务接入三个模块的 RESTful Web 服务开放平台的实现方案. 在 Web 服务化的相关研究中, 由于 RESTful Web 的诸多优势, 针对 RESTful Web 服务化的研究越来越多, 针对 RESTful Web 的实现案例也越来越多, 因此, 针对 RESTful Web 的研究是很有意义的. 在需求越来越多与场景变化多样的情况下, 能否提供高质量的服务显得十分关键, 针对 Web 服务化存在诸多挑战, 文献[13,14]研究了如何提供更好的服务化方法. 本文提出的基于 Gitlab 的服务化解决方案就是基于 RESTful Web 实现的.

3 研究介绍

本章节面向工作流, 基于 Gitlab 的服务化能力无法满足工作流对服务集成的需求的背景, 首先介绍了工作流和 Gitlab 相关的背景知识; 然后指出 Gitlab 服务化的现状和不足; 最后, 着重从服务流程、服务标准和服务实现这三方面介绍 GITService 服务化模型.

3.1 背景介绍

3.1.1 工作流

工作流系统的作用是“使在多个参与者之间按照某种预定义的规则传递文档、信息或任务的过程自动进行, 从而实现某个预期的业务目标, 或者促使此目标的实现”[1]. 本文基于软件开发过程中的工作流平台开展研究, 该平台是以 JBPM 为核心开发, 集任务管理, 代码管理, 缺陷管理等工作为一体的自动化软件开发任务流程管理平台, 任务管理、代码管理和缺陷管理等工具通过工作流平台被串联起来, 开发任务在工作流平台发布后自动在任务管理工具间流转.

在工作流中存在包含子流程的情况. 子流程中涉及当前服务与前置服务, 当前服务是指当前的工作流节点所提供的执行或监听服务, 前置服务是子流程最开始的节点所对应的执行或监听服务, 前置服务与当前服务相关联, 若当前服务为子流程中最后一个节点, 如果此时判断最后一个节点的状态满足子流程结束条件, 则终止子流程, 若不满足, 则重置当前服务节点与前置服务节点状态, 以前置节点为起始点, 重新执行子流程, 直至子流程满足终止条件.

3.1.2 Gitlab

Gitlab 是目前被广泛使用的基于 git 的开源代码管理平台, 基于 Ruby on Rails 构建, 主要针对软件开发过程中产生的代码和文档进行管理, Gitlab 主要针对 group 和 project 两个维度进行代码和文档管理, 其中 group 是群组, project 是工程项目, 一个 group 可以管理多个 project, 可以理解为一个群组中有多项软件开发任务, 而一个 project 中可能包含多个 branch, 意为每个项目中有多个分支, 分支间相互独立, 不同分支可以进行归并. 如表 1 所示, 在 Gitlab 中, 除 push 操作外, 其他操作都可以通过 Web 的形式进行交互.

Gitlab 有两种服务化能力, 监听服务和执行服务. 监听服务是指, 监听某一任务执行, 执行后, 将结果返回给服务调用方; 执行服务是指, 直接执行所调用的服务, 并将结果返回给服务调用方. 两种服务化能力均以 RESTful Web 服务化体系为基础.

表 1 Gitlab 基本操作介绍

名称	意义
push	将代码从本地提交到project的对应分支
merge request	代码归并请求
close merge request	关闭/拒绝代码归并
accept merge request	接受代码归并
issue	在project或group提交问题
note	对project或merge评论

3.2 Gitlab 服务化现状与不足

在与工作流协同工作时, Gitlab 的服务化能力难以满足工作流的需求, Gitlab 的服务化主要是针对具体资源进行的, 在 Gitlab 的 26 种资源服务中, 当前仅提供针对 project, group, merge 三种资源在 project 和 group 维度上的服务化能力. 如表 2 所示, 通过对有代表性的几种服务分析后发现, Gitlab 现存服务化能力难以满足工作流的需求, 我们将问题归类为灵活性差、拓展性弱以及服务粒度粗等三个方面.

表 2 Gitlab 固有服务化存在的问题

资源类型	服务名称	功能	工作流的需求
project	push events	监听push事件	需要监听具体的人对某一分支的push
	issues events	监听issue提交事件	需要监听具体的人对project提交包含特定关键字的issue
	note events	监听评论事件	需要区别对project和merge request的评论
group	push events	监听群组的push事件	区分group和project的push事件
merge	merge request events	监听merge request	需要监听特定人或merge信息中包含特定关键字的merge request
	note events	监听评论事件	需要区别对project和merge request的评论
tag	Tag push	只能针对主分支提交tag	能够选择相应分支提交tag, 对不同分支进行版本控制

在灵活性方面, Gitlab 的服务化标准难以应对多种参数解析和配置的需求, 使得服务化能力不灵活; 在拓展性方面, 在 Gitlab 服务化流程中, 增加资源监听只能以修改数据标准和关键部分为代价添加, 在应对复杂监听需求的情况下, 展现出了较弱的拓展性; 在服务粒度方面, Gitlab 的服务化只能在粗粒度的资源上提供事件监听服务, 却无法以定制化的方法针对资源下发生的事件提供更加细粒度的服务. 代码管理作为整个软件开发过程中最重要的环节, 为了实现对整个开发任务的管理, Gitlab 需要有更强的服务化能力.

3.3 GITService 服务化模型

3.3.1 服务流程

基于 Gitlab 的服务化问题, 我们提出了基于 Gitlab 的 GITService 服务化模型, 新的服务化方法依然基于 RESTful API 构建, 兼容了固有的 Gitlab 服务化方法, 提供了更加灵活、拓展性更强、可定制的细粒度的对外服务. 从流程上讲, 如图 1 所示, 根据所请求的服务种类的不同, 使用不同的解析方式对参数解析, 并继续执行对应的服务化流程. 在 GITService 服务化流程中, 当前服务完成后, 首先判断当前服务是否与前置服务

有关联, 如果判断为真, 说明当前服务与前置服务在同一个子流程中, 需要判断子流程是否满足终止条件, 如果不满足, 则继续执行子流程, 若当前服务为子流程最后一个节点, 则将当前服务状态更新为未完成, 以前置服务为起点, 重新执行整个子流程, 直至流程中最后的服务节点的执行状态满足子流程终止条件.

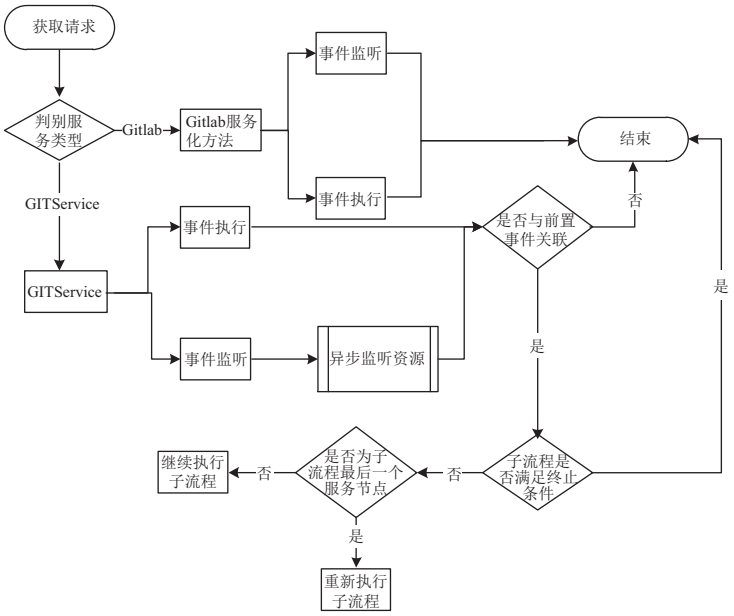

图 1 GitService 服务化流程图

3.3.2 服务标准

通过分析 Gitlab 现有服务的数据标准, 发现这个标准难以对不同参数灵活处理, 难以拓展新的服务也难以提供更细粒度的服务.

因此, 为了提供更灵活、拓展性更强、更细粒度的服务, 我们需要设计新的服务标准. 该服务标准能够对参数灵活解析; 能够以参数化的方法进行新服务的配置, 从而实现对服务的拓展; 能够利用服务标准, 通过解析服务需求的方法, 对资源进行细粒度的服务. 新的服务数据标准如表 3 所示. 通过 iscas_type 表示要执行的对外服务编号, 添加新服务时, 在配置文件中添加新服务和对应的编号即可, 该方法以可配置的形式实现了较强的服务拓展性. 在 content 字段中, 以 json 格式记录关键参数信息, 在执行服务时将字段灵活解析, 借助 json 解析速度快的优势, 实现了灵活的服务能力且不会对于服务性能造成过多影响; 根据解析得到的参数, 解析其具体含义, 可以定制任意细粒度的服务. 使用 issue_id 作为触发服务验证的标记, 如果事件中包含 issue_id 关键字内容, 则验证当前事件是否为服务所监听的事件. 使用 status 字段和 target_id 字段, 实现前置服务与当前服务的关联, 借助 status 标记子流程完成情况的状态信息, 决定整个循环服务的结果; 也可以使用 flag 为触发服务验证的标记, 如果事件中包含 flag 信息, 则验证当前事件是否为服务所监听的事件.

3.3.3 服务实现

依据 Gitlab 服务化能力类型, 将 GITService 分为监听服务和执行服务两部分实现.

3.3.3.1 监听服务

监听服务, 主要包含两个部分, 监听服务信息存储和监听反馈. 监听服务信息存储是指, 将请求监听服务时传入的参数信息、状态信息、回调地址等信息解析, 将信息以服务标准格式化, 并存储起来. 监听反馈是指在实际的代码与开发中, 获取用户与 Gitlab 的交互信息, 异步的判断发生事件是否为所需要监听的事件, 若判断为真, 则将事件发生的情况反馈给回调地址.

监听服务信息存储主要包括“三步验证一步判断”. 如图 2 所示, Gitlab 获取服务请求后, 首先验证请求中的权限信息(验证 1), 判断其是否被允许调用 Gitlab 的服务, 随后根据请求的地址和解析的字段信息判断所请求的是 Gitlab 事件监听服务还是 GITService 事件监听服务(判断 1), 在 GITService 监听中, 需要再次对重要参数进行解析, 验证请求是否有权限对相应资源监听(验证 2), 将信息以服务标准格式化, 对参数合法性

进行验证(验证 3), 验证成功后需要对重要参数以 json 的形式组合并存储在 content 中, 存储格式化后的信息供监听反馈阶段使用.

表 3 GITService 服务标准

名称	含义
url	服务执行结果的反馈地址
target_id	循环前置事件编号
iscas_type	服务类型
task_id	前置服务编号
content	Json格式的关键参数
issue_id	服务编号
status	子流程服务执行状态
flag	触发服务的标记信息

监听反馈主要包括“五步判断”. 由于监听反馈过程需要对 Gitlab 的任何事件进行监听, 再删选出需要监听的事件, 但如果在相同模块同步监听, 由于增加了判断与验证机制, 会使 Gitlab 的交互响应时间大大延长. 而使用消息队列机制捕捉当前事件的关键信息, 使用异步机制进行判断与验证, 能够从最大程度上减少监听反馈对于 Gitlab 的交互响应时间的影响, 所以本文采用消息队列与异步机制进行监听反馈. 如图 3 所示, 当用户与 Gitlab 交互时, 使用字符串匹配的方法, 判断交互信息中是否包含如 issue_id 或 flag 的标记字段(判断 1), 如果判断为真, 证明其有可能是所需监听的事件, 异步的执行监听任务和事件, 在事件异步执行的过程中, 通过 Ruby on Rails 框架提供的消息队列与异步机制, 从消息队列获取所监听事件的关键信息, 结合上一步存储的监听服务信息, 异步判断当前发生的事件是否为所需要监听的事件(判断 2), 判断为真的情况下, 封装当前事件的监听信息, 以规定的格式反馈给回调地址, 继续判断当前监听服务是否与前置服务有关联(判断 3), 若判断为假则结束当前监听服务, 若判断为真, 说明当前服务存在于子流程中, 继续判断子流程是否满足终止条件(判断 4), 若为真, 说明已经执行完子流程, 结束子流程, 若未完成, 则继续进行子流程, 判断当前服务节点是否为子流程的最后一个节点(判断 5), 若判定为真, 则重置前置服务状态, 将前置服务状态更新为未完成, 以前置服务为起点, 重新执行整个子流程, 直至子流程满足终止条件.

3.3.3.2 执行服务

执行服务是指在完成指定任务的执行后, 将执行结果的相关信息反馈给回调地址. 执行服务中主要涉及“两步验证三步判断”. 如图 4 所示, Gitlab 获取请求后, 验证请求是否有权限调用该执行服务(验证 1), 在通过权限验证之后, 解析请求字段, 分析判断所请求的是 GITService 执行服务还是 Gitlab 执行服务(判断 1). 对于 GITService 执行服务而言, 需要针对请求中的重要字段解析, 使用服务标准格式化并存储重要信息, 通过验证服务成功判断请求对所需执行资源有权限后(验证 2), 即在 Gitlab 执行对应的任务, 执行结束后, 与监听服务类似, 若判断子流程结束(判断 2), 则将执行结果以固定格式反馈给回调地址, 若不满足子流程结束条件, 判断当前服务节点是否为子流程的最后一个服务节点, 若判断为假则继续执行子流程, 若判断为真, 则重新执行子流程.

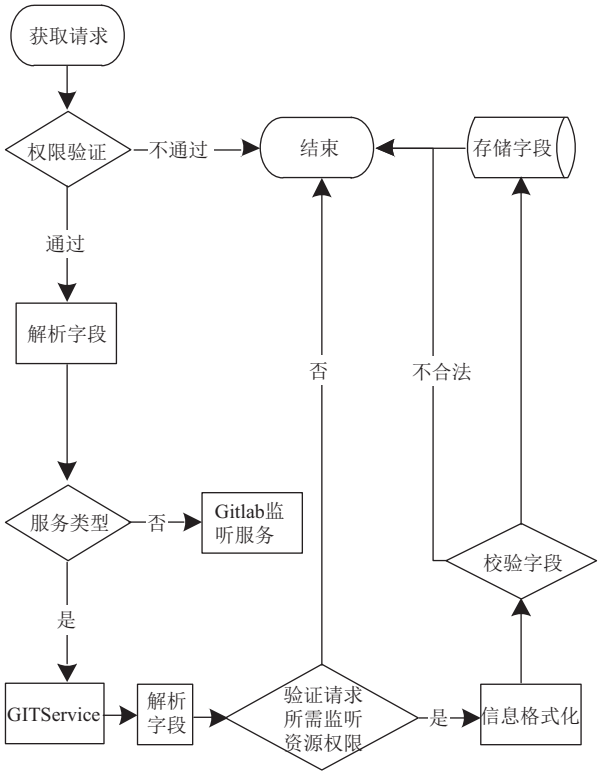

图 2 监听服务信息存储流程图

4 实验分析

基于 Gitlab 服务化存在的三类问题, 本章首先针对 GITService 的服务化能力进行评价, 然后针对 GITService 可能给 Gitlab 的正常业务处理和响应时间带来的潜在负面影响, 进行响应时间的对比分析.

4.1 服务化能力评价

我们基于常用服务, 对 Gitlab 与 GITService 的服

务化能力比较, 在灵活性、可拓展性、服务粒度等三个指标上评价 GITService 服务化能力. 如表 4 所示, GITService 在所有常用服务中, 均表现出更优的灵活性、能够提供更细粒度的服务; 在扩展性方面, 在部分项目上具有更强的可拓展性, 在其他项目上保持原有的可拓展性.

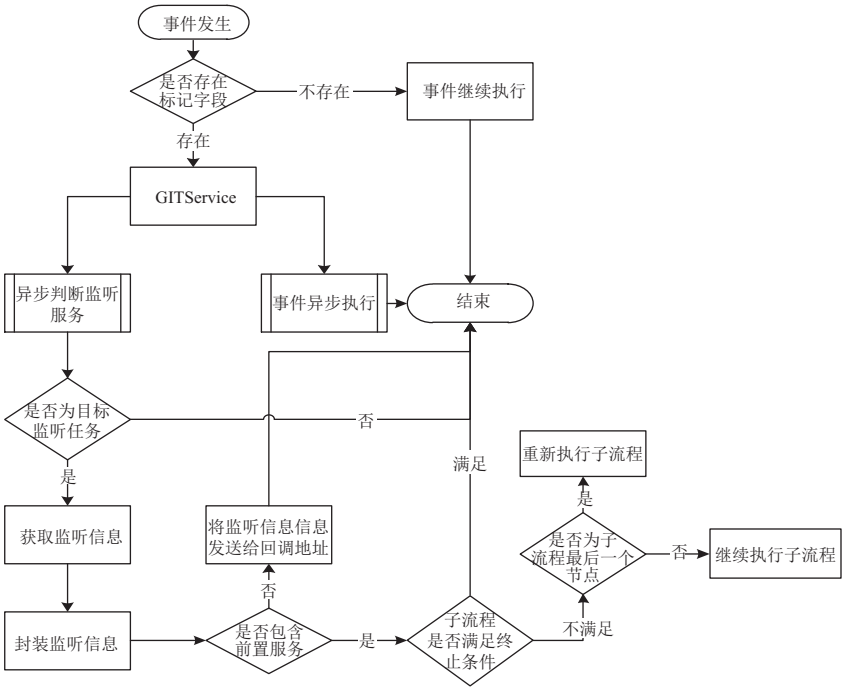

图 3 监听反馈流程图

4.2 时间分析

由于 GITService 服务化提供了更多更细粒度的监听服务, 在实现时, 在不同资源请求操作中需要调用监听服务模块, 虽然在设计上采用了异步的形式执行监听服务, 但是在执行监听服务之前需要针对事件进行针对 flag 或 issue_id 的判断, 需要在 Gitlab 处理自身业务时, 判断当前处理的业务是否为目标监听事件, 存在对 Gitlab 的 Web 业务交互处理时间的潜在负面影响.

我们从时间上分析了 GITService 是否存在对于 Gitlab 的 Web 业务交互的负面影响. 我们在相同机器、相同网络环境下进行试验, 针对 Gitlab 常用的几类资源, 对比 Gitlab 引入 GITService 服务体系前后的平均响应时间. 如表五所示, GITService 对于 Gitlab 在常用资源的 Web 业务处理与响应时间的影响很小, 根据[15]的观点, 响应时间在 2 秒以内即可以达到较好的用户体验, 在实验中发现, 响应时间在 10 毫秒级别的影响在 3% 以内, 可以认为 GITService 几乎不存在对 Gitlab 处理 Web 交互业务与响应时间的负面影响.

5 结语

本文针对面向工作流环境的 Gitlab 服务化所存在的灵活性差、拓展性差和服务粒度粗的问题, 提出了一种在工作流环境中更优的服务化方案——GITService. 通过对比和实验发现, GITService 是一种能够提供灵活性好、拓展性强、细粒度化服务的 Gitlab 服务化解决方案. 同时, GITService 保证了 Gitlab 自身项目结构的完整性, 展现出了较优的兼容性. 最终通过对引入 GITService 前后, 对 Gitlab 交互响应时间的实验对比, 发现 GITService 对 Gitlab 处理 Web 交互相关业务及响应时间的影响很小, 表明 GITService 使用较低的代价, 改善了对于 Gitlab 对外服务体系在工作流环境下的服务能力.

本文仅针对 Gitlab 的服务化进行了研究, 通过提出一种新的 GITService 服务化解决方案, 解决了 Gitlab 服务化中现存的问题. 但是对于 Gitlab 自身而言, 依然有许多需要改进之处, 尤其是对内的权限管理、代码审核等方面依然有较多值得研究的工作. 进一步工作将针对 Gitlab 内部体系、代码审核等方面提出更优的解决方案和验证方法.

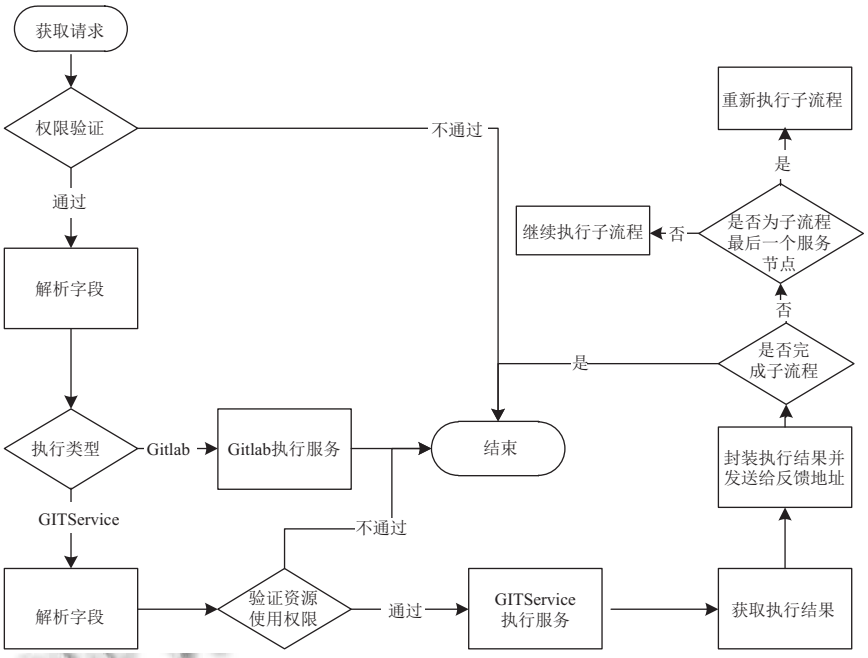

图 4 执行服务流程图

表 4 Gitlab 和 GITService 的服务化能力评价

服务概述	Gitlab服务能力	GTIService服务能力	灵活性	拓展性	服务粒度
创建代码版本	只能对project进行代码版本创建	除了project, 还可以对project的branch创建代码版本	根据参数, 选择对分支版本创建, 更灵活.	拓展性不变	能够对project的branch维度创建版本, 服务粒度更细
监听merge request被接受/关闭	未提供该服务	可以提供服务	根据参数, 选择具体事件的监听, 更灵活	使用属性iscas_type实现了对监听事件的添加, 拓展性更强	针对merge中的接受和关闭事件监听, 服务粒度细
监听merge request	监听project层面的merge动作	监听project和branch的merge动作	根据参数, 选择监听, 更灵活	拓展性不变	可以对project和下面的branch的merge, 服务粒度更细
监听push	监听push事件	监听push事件	根据参数, 验证push的详细信息是否与监听需求一直, 更灵活	拓展性不变	可以通过参数, 对push事件中满足其他组参数信息的push事件监听, 服务粒度更细
监听代码版本创建	未提供该服务	可以提供服务	根据参数, 选择监听project还是某一个branch的版本创建, 更灵活	使用属性iscas_type实现了对监听事件的灵活配置, 拓展性更强	能够对project的branch维度创建版本事件监听, 服务粒度更细
监听评论	对评论类型无区分	根据参数, 对评论类型区分	根据参数, 选择监听merge还是project的评论事件, 更灵活	拓展性不变	可以区分监听的是project还是merge的评论, 服务粒度更细
创建project	创建project	根据参数, 创建一个或多个project	根据参数, 分析所请求服务内容, 判断创建一个或多个project, 更灵活	拓展性不变	可以从数量维度上选择服务的种类, 相当于提供了更强的服务伸缩性, 服务粒度更细
创建merge request	创建某project下的merge事件	根据参数, 创建指定的merge事件	根据参数, 选择创建一个或多个merge事件, 更灵活	拓展性不变	可以从数量维度上选择服务的种类, 相当于提供了更强的服务伸缩性, 服务粒度更细
将project移至其他群组	仅能移动至一个群组	可以根据参数情况移动至多个群组	根据参数, 选择一个或多个project, 并针对具体需求迁移至不同群组, 更灵活	拓展性不变	可以从数量维度上选择服务的种类, 相当于提供了更强的服务伸缩性, 服务粒度更细

表 5 引入 GITService 前后响应时间分析

资源名称	引入GITService前(ms)	引入GITService后(ms)	变化(%)
projects	725	743	2.48
group	245	252	2.86
issue	516	520	0.78
tag	509	521	2.36
merge request	652	671	2.91

参考文献

1 Workflow Management Coalition. WFMC-TC-1003: The workflow reference model. Cohasset: Workflow Management Coalition, 1995: 1–55.

2 刘书雷, 刘云翔, 张帆, 等. 一种服务聚合中 QoS 全局最优服务动态选择算法. 软件学报, 2007, 18(3): 646–656.

3 张成文, 苏森, 陈俊亮. 基于遗传算法的 QoS 感知的 Web 服务选择. 计算机学报, 2006, 29(7): 1029–1037.

4 程强. Web Services 服务质量模型与量化算法研究[硕士学位论文]. 成都: 电子科技大学, 2008.

5 罗海滨, 范玉顺, 吴澄. 工作流技术综述. 软件学报, 2000, 11(7): 899–907.

6 Duan Q, Yan YH, Vasilakos AV. A survey on service-oriented network virtualization toward convergence of networking and cloud computing. IEEE Trans. on Network and Service Management, 2012, 9(4): 373–392. [doi: 10.1109/TNSM.2012.113012.120310]

7 Banerjee P, Friedrich R, Bash C, *et al*. Everything as a service: Powering the new information economy. Computer, 2011, 44(3): 36–43. [doi: 10.1109/MC.2011.67]

8 张卫, 张童, 查亚兵. 基于 HLA 分布式仿真的 Web 服务化. 国防科技大学学报, 2008, 30(5): 120–124.

9 周巧俊. RESTful Web 服务开放平台的设计与实现[硕士学位论文]. 杭州: 浙江大学, 2016.

10 Lee S, Jo JY, Kim Y. Method for secure RESTful web service. IEEE/ACIS 14th International Conference on Computer and Information Science (ICIS). Las Vegas, NV, USA. 2015. 77–81.

11 Rathod DM, Dahiya MS, Parikh SM. Towards composition of RESTful web services. 2015 6th International Conference on Computing, Communication and Networking Technologies (ICCCNT). Denton, TX, USA. 2015. 1–6.

12 Selvakumar G, Kaviya BJ. A survey on RESTful web services composition. 2016 International Conference on Computer Communication and Informatics (ICCCI). Coimbatore, India. 2016. 1–4.

13 Mehdi M, Bouguila N, Bentahar J. Trust and reputation of web services through QoS correlation lens. IEEE Trans. on Services Computing, 2016, 9(6): 968–981. [doi: 10.1109/TSC.2015.2426185]

14 Zhong Y, Fan YS, Tan W, *et al*. Web service recommendation with reconstructed profile from mashup descriptions. IEEE Trans. on Automation Science and Engineering, 2016, doi: 10.1109/TASE.2016.2624310.

15 陈能技, 黄志国. 软件测试技术大全: 测试基础 流行工具 项目实战. 3 版. 北京: 人民邮电出版社, 2015.