文章编号：2096-1472(2020)-04-29-04
DOI:10.19644/j.cnki.issn2096-1472.2020.04.008

# 基于流水线理念持续集成与持续交付平台的设计与实现

王 俊，牛亚运

(上海理工大学机械学院，上海 200082)
✉961037987@qq.com;993868993@qq.com

**摘 要**：DevOps是近年来兴起的软件开发理念，能够有效提高开发效率与版本质量，持续集成与持续交付作为DevOps理念的最佳实践，应用广泛。本文结合流水线的理念，设计并实现了持续集成与持续交付平台，通过触发器匹配机制启动流水线，自动执行流水线中各阶段定义的任务，有效地支撑项目的快速迭代，提高了开发效率，保证软件稳定、持续的构建和发布，实现了持续集成持续交付的全流程应用，为后续DevOps理念的研究及实践提供了参考依据。

**关键词**：DevOps；流水线；持续集成；持续交付
**中图分类号**：TP31 **文献标识码**：A

# Continuous Integration and Continuous Delivery Platform Based on the Pipeline Concept

WANG Jun,NIU Yayun

( *School of Mechanical Engineering,University of Shanghai for Science and Technology,Shanghai* 200082,*China*)
✉961037987@qq.com;993868993@qq.com

**Abstract:**DevOps is a software development concept that has emerged in recent years.It can effectively improve development efficiency and version quality.Continuous integration and continuous delivery are the best practices of the DevOps concept,which are widely used.This paper combines the concept of the pipeline to design and implement continuous integration and continuous delivery platform.The trigger matching mechanism is used to start the pipeline and automatically execute the tasks defined in each stage of the pipeline.This design effectively supports the rapid iteration of the project,improves the development efficiency,ensures the stable and continuous construction and release of the software,and realizes the continuous integration and continuous delivery of the whole process application,which provides a reference for the research and practice of the subsequent DevOps concepts.

**Keywords:**DevOps;pipeline;continuous integration;continuous delivery

## 1 引言(Introduction)

DevOps是一组过程、方法与系统的统称，用于促进软件开发、技术运营和质量保障(QA)部门之间的沟通、协作与整合[1]，其核心是持续集成与持续交付。在持续集成中，团队成员频繁集成工作成果，每人每天至少集成一次也可多次，每次集成经过自动构建的检验，以求在最短的时间内发现集成错误[2,3]。持续交付的核心思想则是通过创建可重复、可靠的操作过程，将软件从概念变为现实[4]。持续交付流水线的搭建使得持续交付本身成为可能，将软件交付的过程分成不同阶段，每个阶段旨在从不同的角度验证新特性的质量以确认新功能，防止可能发生的失误给用户造成的不良影响[5]。持续集成与持续交付继承了优秀的DevOps开发理念，成为未来软件工程重要的组成部分。

## 2 持续集成与持续交付平台需求分析(Demand analysis of continuous integration and continuous delivery platform)

软件项目的目标是开发出可运行、客户满意的软件系统。传统开发存在Bug总是在最后才出现、越到项目后期问题越难解决、软件交付时机无法保障、程序经常需要变更等问题，影响项目的管理和开发效率，为解决上述问题，持续集成与持续交付平台需要实现在开发人员提交代码后，开发人员相互评审，评审通过后，服务器自动更新代码，编译，运行单元测试、功能测试、集成测试，进行部署。软件开发人员通过观察流水线，清楚的了解开发中每个阶段的任务状态，快速发现并解决问题(编译失败、测试失败等)，减小系统发生错误、不能在用户环境中运行、系统集成时涌现大量

问题的风险。从而使整个的项目进度完全掌握，提高项目质量，保证项目的持续稳定交付。

## 3 持续集成与持续交付平台设计概要(Design of continuous integration and continuous delivery platform)

### 3.1 持续集成持续交付平台的核心功能设计

为构建需求分析中完整的软件开发流水线，实现持续集成、自动化测试、包管理等流水线软件产品的交付过程，持续集成与持续交付平台设计包含四大模块：(1)入口配置模块；(2)流水线配置模块；(3)触发器匹配模块；(4)流程编排模块。图1为触发器的核心功能模块示意图。

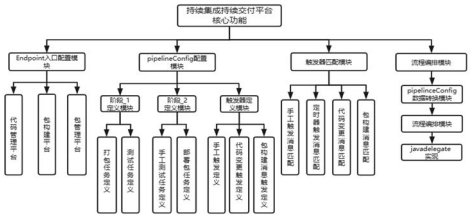


图1 核心功能模块示意图

Fig.1 The schematic of core function module

### 3.2 持续集成与持续交付平台运行过程

持续集成与持续交付平台采用微服务的架构方式，包含三个服务，分别是主服务、流程处理服务和通知服务，其中主服务中包含入口配置模块和流水线配置模块，流程处理服务包含触发器匹配模块和流程编排模块，服务之间采用集成RestTemplate的方式互相调用对外暴露的接口，通过集成RabbitMQ消息队列的方式进行通信，保证了流水线运行期间各个阶段中任务的有序执行；通知服务作为辅助服务实现了从流水线开始执行到结束执行各个阶段执行状态的通知功能。用户首先根据入口配置规则填写入口验证信息，验证是否能与第三平台进行可靠对接，之后配置流水线信息，流水线配置信息完成后，点击触发按钮，此时主服务通过postEvent(Event event)方法向流程处理服务发送启动事件，流程处理服务接收到HTTP请求之后，对传入的流水线配置信息进行数据转换，转换为流程编排模块可以处理的数据格式，流程处理服务中的启动流程监听器监听到启动消息后，流程编排引擎开始执行流水线中定义的任务，最后任务执行结束后，结束流程监听器监听到消息后将会结束流水线执行。如图2所示为持续集成与持续交付平台运行时序图。

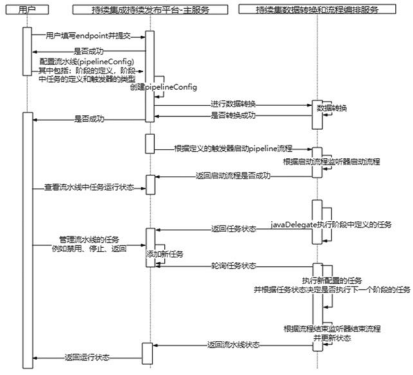


图2 系统运行时序图

Fig.2 The sequence diagram of system operation

## 4 持续集成与持续交付平台核心功能实现(Core function implementation)

### 4.1 入口配置模块的实现

持续集成与持续交付平台具备完成流水线完整功能的能力，必须要具备代码管理与审阅功能、构建和测试功能，以及包管理等功能，因此平台需要支持无缝对接第三方平台，为此持续集成持续交付平台设计了入口配置模块。选用Gitlab作为代码管理与评审平台，GitLab是由GitLabInc.开发，使用MIT许可证的基于网络的Git仓库管理工具，使用Gitlab可以完美的实现不同操作人员协同开发、互相评审的功能。选用Jenkins作为包构建和测试工具，Jenkins是一个开源的、提供友好操作界面的持续集成(CI)工具，主要用于持续、自动的构建/测试软件项目、监控外部任务的运行。具有易用性高、易配置支持插件等特点。选用Nexus作为包管理平台，Nexus是一个强大的maven仓库管理器，它极大地简化了本地内部仓库的维护和外部仓库的访问,利用私服代理外部仓库，消除了对外的重复构件下载，降低带宽的压力，是一套开箱即用的系统，如表1所示，入口配置模块中提供了对接GitLab、Jenkins、Nexus入口并进行验证的方式。

表1 对接验证GitLab、Jenkins、Nexus入口的配置方式表

Tab.1 The configuration mode table for GitLab,Jenkins and Nexus entry verification

| 平台名称 | 用户名 | 密码/token | Gitlab地址 |
|---|---|---|---|
| Gitlab | name | password | Gitlab服务器地址：端口号 |
| Gitlab | 无 | token | Gitlab服务器地址：端口号 |
| Jenkins | name | password | Jenkins服务器地址：端口号 |
| Jenkins | name(可缺省) | token | Jenkins服务器地址：端口号 |
| Nexus | name | password | Nexus服务器地址：端口号 |
| Nexus | 无 | Token(企业版) | Nexus服务器地址：端口号 |

以对接Nexus包管理平台为例，用户在前端页面填写入口配置信息后，信息被保存到了入口配置模型(Endpoint)中，后台控制层接收到前端发送的POST请求后，调用NexusConnectTest(Endpoint endpoint)方法，方法内部利用Spring框架提供的RestTemplate将入口配置模型中保存的用户名、密码、Nexus仓库地址等信息封装成一个HTTP请求并发送到Nexus服务器，Nexus服务器接收到HTTP请求后进行解析验证是否认证成功，根据认证结果返回不同的响应码，进而根据响应码来判断对接Nexus包管理平台是否成功。如下为对接验证Nexus包管理平台的部分代码实现：

```
1. public EndpointConnectionStatus nexusConnectionTest(Endpoint endpoint) {
2.     RestTemplate restTemplate=buildClient(endpoint);
3.     ResponseEntity<List>responseEntity;
```

```
4.    try {
5.    responseEntity=restTemplate.getForEntity
(NEXUS_REPOSITORY_URL, List.class);
6.        } catch (HttpClientErrorException e) {
7.            if (e.getStatusCode().equals(HttpStatus.
UNAUTHORIZED)) {
8.                return EndpointConnectionStatus.
AUTHENTICATION_ERROR;    }
9.                return EndpointConnectionStatus.
IINVALID_ENDPOINT_ADDRESS;
10.       } catch (Exception e) {
11.               return EndpointConnectionStatus.
IINVALID_ENDPOINT_ADDRESS;    }
12.       HttpStatus httpStatus=responseEntity.
getStatusCode();
13.       if (httpStatus.equals(HttpStatus.OK)
&& responseEntity.getBody().size() !=0) {  return
EndpointConnectionStatus.SUCCESS;    }
14.       return EndpointConnectionStatus.
AUTHENTICATION_ERROR;
15. }
```

### 4.2　流水线配置模块

持续集成持续交付平台的流水线配置模块中提供了定义完整流水线的功能，其定义的内容包括主要包含四部分：(1)流水线配置信息，包括流水线的名称、阶段、触发器等。一条流水线相当于一次构建任务，里面可以包含多个流程，比如自动构建、自动进行单元测试、自动进行代码检查等流程等。(2)阶段配置信息，包括每个阶段的名称、标识、阶段的构建时间、执行时间、状态等。阶段就是上面提到的流程，可以在一条流水线中定义多个阶段，所有阶段会按照顺序运行，即当一个阶段完成后，下一个阶段才会开始，只有当所有阶段成功完成后，该构建任务(流水线)才算执行成功,如果任何一个阶段失败，后面的就不会执行，该构建任务(流水线)失败。(3)任务配置信息，包括任务名称、任务所属的阶段、执行时间、上下文环境等。任务表示某个阶段里面执行的工作，一个阶段里面可以定义多个任务，在相同阶段中的任务可以并行执行也可以串行执行，执行相同阶段中的任务都执行成功时，该阶段才会成功，如果任何一个任务失败，那么该阶段就失败，即该条流水线失败。(4)触发器相关信息，包括触发器的类型和名称。流水线中可以定义多个触发器并且支持多种触发类型，通过触发匹配机制接收触发消息启动流水线。为了遵循数据库设计的三范式要求，在保证三范式要求的前提下尽量保证松耦合，流水线配置模块中定义的信息分为流水线配置信息表，如表2所示；阶段配置信息表，如表3所示；任务配置信息表，如表4所示。

表2 流水线配置信息表

Tab.2 Pipeline configuration information table

| 字段名称 | 类型 | 长度 | 说明 |
|---|---|---|---|
| groupName | String | 128 | 流水线所在项目名称 |
| groupId | String | 128 | 流水线所在项目id |
| name | String | 128 | 流水线配置名称 |
| description | String | 512 | 流水线配置的描述 |
| List<Stage> stages | List<Map> | 动态扩展 | 触发器配置 |
| lastBuildTime | Date | 默认 | 上次运行时间 |
| lastExecutionId | String | 128 | 上次运行id |
| List<Map> notify | List<Map> | 动态扩展 | 通知配置 |

表3 阶段配置信息表

Tab.3 Stage configuration information table

| 字段名称 | 类型 | 长度 | 说明 |
|---|---|---|---|
| refId | String | 128 | 唯一标识一个阶段 |
| name | String | 128 | 阶段名称 |
| List<Task>task | List<Map> | 动态扩展 | 阶段中任务配置 |
| Map<K,V>output | Map | 动态扩展 | 阶段的输出信息 |
| startTime | Date | 默认 | 开始时间 |
| endTime | Date | 默认 | 结束时间 |
| context | Map | 动态扩展 | 上下文环境 |
| executeMod | ENUM | 128 | 模式：串/并行 |

表4 任务配置信息表

Tab.4 Task configuration information table

| 字段名称 | 类型 | 长度 | 说明 |
|---|---|---|---|
| refId | String | 128 | 标识一个任务 |
| name | String | 128 | 任务名称 |
| type | String | 64 | 任务类型 |
| comments | String | 512 | 任务说明 |
| Map<K,V> outputs | List<Map> | 动态扩展 | 任务输出 |
| Map<K,V> context | Map | 动态扩展 | 任务上下文 |
| Map<K,V> metadata; | Map | 动态扩展 | 任务输入 |
| List<Map> notify | List<Map> | 动态扩展 | 任务通知 |

### 4.3　触发器匹配模块

持续集成与持续交付平台支持手工触发、定时触发、事件驱动三种触发方式，手工触发通过点击前端展示的触发按钮调用startPipeline(String id, PipelineTrigger trigger)方法启动流水线；定时触发通过定时任务线程轮询触发任务状态并根据上下文信息启动流水线；事件触发包括Gitlab代码变更消息触发、Jenkins构建消息触发、Nexus包管理平台包更新

触发。不同的触发方式设置了不同的触发匹配机制。以事件触发为例，Jenkins构建消息触发流水线执行，当Jenkins服务器上发生一次构建的时候，持续集成与持续交付平台主服务中的轮询任务接收到构建信息后，通过rabbitMQ消息队列将该消息发送到流水线处理服务中的指定队列中，队列接收消息后执行onJenkinsBuildComplete(String message)方法，该方法根据消息中包含的Endpoint入口信息和指定的job调用matchJenkinsTrigger(endPointID,jobName)方法匹配触发器，最后进入流程编排模块完成流水线中定义的各个阶段任务。如下为接收Jenkins构建消息触发的代码实现：

```
1. public void onJenkisBuildComplete(String message){
2.     logger.info("receive message,jenkins job build complete");
3.     try {
4.         if (StringUtils.isEmpty(message)){ return; }
5.         Map<String,String>params=Utils.fromJsonByJackson(message, Map.class);
6.         String endpointId=params.get("endpoint");
7.         String jobName=params.get("jobName");
8.         List<PipelineTrigger>triggers=triggerService.matchJenKinsTrigger(endpointId, jobName);
9.         if (CollectionUtils.isEmpty(triggers)){
10.            logger.info("no trigger matched, ignore it");
11.            return; }
12.        for (PipelineTrigger trigger:triggers) {
13.    pipelineConfigService.startPipeline(trigger.getPipelineConfig().getId(),trigger);
14.    }
15. } catch (IOException e) {e.printStackTrace();
16.    }
17. }
```

### 4.4 流程编排模块

流程编排模块用于执行流水线阶段中定义的各项任务，是持续集成持续交付流水线的核心所在，本文选用开源的Flowable流程引擎作为实现流程编排模块的实现基础，Flowable是一个使用Jave编写的轻量级业务流程引擎，使用Apache V2 license协议开源，其以JAR形式发布使得Flowable可以轻易加入任何Java环境：Java SE、Tomcat、Jetty或Spring之类的servlet容器；JBoss或WebSphere之类的Java EE服务器等。默认情况下，Flowable引擎的执行文件是bpmn文件，文件中包含事件、顺序流、网关、任务、子流程要素，在Flowable流程引擎执行流水线之前，需要预先将流水线配置模块中定义的流水线信息转换为bpmn文件。Flowable流程引擎对外提供了org.flowable.engine.delegate.JavaDelegate接口及其execute方法，持续集成与持续交付平台通过实现JavaDelegate接口并且为不同任务重写不同的execute方法完成流水线中定义的各种任务。如下所示为重写execute方法执行Jenkins任务的部分代码实现：

```
1. public TaskResult executeTask(TaskInstance task) {
2.     task=taskInstanceService.findOne(task.getId());
3.     JenkinsTaskContext context=parseContext(task.getMetadata());
4.     if (context.inWaitingState()) {
5.         if (context.onBuildStage()) {
6.             BuildWithDetails buildWithDetails=jenkinsService.getBuild(context.buildItemUrl,context.getEndpoint());
7.             if (isCompleted(buildWithDetails.getResult())) {
8.                 if (buildWithDetails.getResult().equals(BuildResult.UNSTABLE) && context.continueExecute) {return TaskResult.ofStatus(ExecutionStatus.TERMINAL);
9.                 }
10.                return TaskResult.ofStatus(ExecutionStatus.SUCCEEDED); }
11.            return TaskResult.ofStatus(ExecutionStatus.RUNNING);
12.        } }
13.    try {
14.        String queueItemUrl=jenkinsService.build(context.getJobName(), context.getJobParameters(), context.getEndpoint());
15.        task.getMetadata().put(KEY_QUEUE_ITEM_URL, queueItemUrl);
16.        taskInstanceService.patch(task.getId(), task);
17.    } catch (Exception e) {
18.     return failTask(task);
19.    }
20. }
```

## 5　结论(Conclusion)

本文基于软件开发的实际环境，设计并实现了持续集成持续交付平台，提供了可视化、可定制、自动触发的持续交付流水线，流水线中支持编译构建、代码检查、自动化测试、部署、流水线控制等多种类型的任务，用户可以根据需求定义各种任务，流水线阶段中的任务支持串行执行与并行执行，支持了多种业务场景，流水线支持手工触发和自动触发，实现了流水线的自动化执行通过配置流水线的方式完成软件开发集成与交付过程，极大地缩短了从开发、集成、测试、部署各个环节的时间，减少了重复性劳动，提高了产品质量，加快了开发效率。本文设计的持续集成与持续交付平台，完整的支撑了生产环境中软件开发的各种需求，为其他用户或者研发团队搭建持续集成与持续交付平台提供了设计参考依据，推动了DevOps理念的落地与实践。

## 参考文献(References)

[1] Haiqin WU,Liangmin WANG,Tao JIANG.Secure and efficient k-nearest neighbor query for location-based services in outsourced environments[J].Science China(Information Sciences),2018,61(03):231-233.

[2] Mengual L,Marbán O,Eibe S.Clustering-based location in wireless networks[J].Expert Systems with Applications,2010,37(9):6165-6175.

[3] Ying Hu.Study on the key algorithms and framework design of intelligent campus education platform based on big data technology[J].Technical Bulletin,2017(15):63-69.

[4] Jia Yanmei.Construction and application of intelligent campus in colleges and Uuniversities under the background of big data[C].Advances in Intelligent Systems and Computing,2019:866-872.

[5] 陈敏,杨阳.基于Docker下DevOps系统的设计与实现研究[J].信息与电脑(理论版),2019(11):97-98;101.

## 作者简介：

王　俊(1993-)，男，硕士生.研究领域：智能制造，软件开发.

牛亚运(1991-)，男，硕士生.研究领域：软件开发.

(上接第36页)

### 3.5　APP架构

APP总体架构分为用户UI交互层、Service层、Database层和Model层四层。构造性良好的架构之间是完全解耦的，便于系统后期升级更新与维护[6]。多层架构的特点是为了使整个层次实现“高内聚，低耦合”的思想，利于降低层与层之间的依赖性[7]，增强APP模块中的可重用性和可移植性。

UI交互层主要用于用户日常使用界面，支持用户的输入、输出显示，UI层提供用户视觉层面的基础界面，良好的易懂可得性保障了用户使用的便利。

Service接受UI层传入的数据，进行运算处理，实现用户要求功能，同时Service连接后台Database层，可进行数据访问获取。Service层起到连接用户使用的UI层和底层数据的作用，是四层架构的核心部分。

Model层和Database起到底层模板搭建和数据收集的作用，作为整个程序的支撑框架。详见图6。

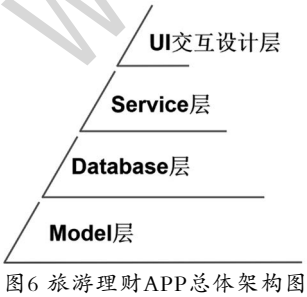


图6 旅游理财APP总体架构图

Fig.6 The overall architecture of the travel finance APP

## 4　结论(Conclusion)

随着大数据时代的到来，大数据技术与互联网金融的深度整合成为互联网金融快速发展的助推器[8]。本文针对传统商行受互联网金融冲击后的转型需求和大学生理财休闲娱乐需求为目标设计了一款以银行作为发行者的大学生旅游理财APP，旨在解决双方困境。该APP开发采用Android作为开发环境，MYSQL作为数据库，Python作为开发语言，SAE作为云服务器，并内嵌IM Plus即时通信渠道，实现用户储蓄理财，旅游套餐管理与订购，账单分析，用户分享等多样化功能，实现银行、用户、旅行社三方合作的关系，为大学生理财—娱乐提供一站式服务，实现“玩+理”结合，方便新颖。

## 参考文献(References)

[1] 贺小荣,陈雪洁.中国文化旅游70年:发展历程、主要经验与未来方向[J].南京社会科学,2019(11):1-9.

[2] 丁勇,程璐,蒋翠清.基于二部图的P2P网络借贷投资组合决策方法[J].数据分析与知识发现,2019(12):76-83.

[3] 郭霖.第一行代码:Android(第2版)[M].北京:人民邮电出版社,2016.

[4] 张茗芳.动态语言Python探讨与比较[J].企业科技与发展,2012(13):57-60.

[5] 周亮,刘剑锋.基于SAE平台下的学生测试与咨询系统设计[J].自动化与仪器仪表,2019(07):106-112.

[6] 李秀红,徐介新,吕兰兰,等.基于Android的大学生掌上理财管理系统的分析与设计[J].软件工程,2018,21(12):25-28.

[7] CSDN-专业IT技术社区.三层架构的优点缺点有哪些？[EB/OL].https://blog.csdn.net/LiKang0825/article/details/80575951,2018-06-05.

[8] 夏德虎.基于互联网金融的大数据驱动模式[J].电子技术与软件工程,2019(24):144-145.

## 作者简介：

王青青(1999-),女，本科生.研究领域：经济金融，金融工程.

潘东亮(1999-),男，本科生.研究领域：经济金融，金融工程.