轻量级 C++微服务框架的设计

唐 稳，刘艳辉

（华北计算技术研究所 应用支撑技术研发部，北京 100083）

摘 要：为满足高实时、低延迟软件系统的上云要求，设计轻量级 C++微服务框架。基于插件技术，设计服务治理插件堆栈、运行支撑中间件插件集、微服务插件动态组装等技术机制，设计 C++微服务之间的 RPC 交互接口，支持微服务治理能力的灵活裁剪和增加，支持运行支撑中间件的动态接入。从 RPC 吞吐率、响应时间等方面，与业界主流微服务框架进行性能对比测试与分析，该方法表现出明显的性能优势。

关键词：微服务框架；PaaS 云平台；远程过程调用；插件；服务治理；运行支撑中间件

中图法分类号：TP311.1 **文献标识号**：A **文章编号**：1000-7024（2019）08-2396-06

doi：10.16208/j.issn1000-7024.2019.08.049

Design of lightweight C++ microservices framework

TANG Wen, LIU Yan-hui

(Research and Development Department of Applied Support Technology, North China Institute of Computing Technology, Beijing 100083, China)

Abstract: To meet the requirements of cloud computing in high real-time and low latency software systems, lightweight C++ micro service framework was designed. Based on plug-in technology, service management plug-in stack, running support middleware plug-in set, microservices plug-in dynamic assembly and other technical mechanism were designed, and RPC interaction interface between C++ microservices was also designed. Flexible cutting and increasing of micro service governance capability was supported. Dynamic accession of runtime support middleware was supported. The performance comparison test and analysis were carried out with the mainstream microservices framework of the industry from the perspectives of RPC throughput, response time, and so on. The results show obvious performance advantages.

Key words: microservices framework; PaaS cloud platform; RPC; plug-in; service management; running support middleware

0 引 言

作为微服务的支撑平台，微服务框架主要解决微服务开发与测试、微服务 RPC 交互、微服务治理及微服务运行支撑等方面的问题，可极大降低微服务应用的开发、设计、测试和运维管理成本，例如 Spring Cloud[1] 就是一个完整的微服务框架。

由于微服务架构的复杂性，很多微服务框架并没有提供完整的支撑平台。例如阿里的 Dubbo[2] 和华为的 CSE (cloud service engine)[3]。此外，还有一些微服务框架仅提供了微服务在一个进程内的开发与集成框架，一般只支持一种开发语言，例如 NutzBoot、RedKale 等开源项目支持 Java 语言，Linkerd 开源项目支持 Scala 语言[4]。这类框架旨在降低微服务的开发成本，并提供了与 PaaS 云平台的其它组件集成的机制和方法，目前对此类框架没有明确定义，在本文中我们称之为轻量级微服务框架。

上述轻量级微服务框架主要面向互联网应用和企业应用，一般不适合用在仿真训练等领域。在这些领域，业界大多采用 C++语言进行开发，这是因为作为一种编译型语言，C++比 Java、Scala 等解释型语言具有更好的运行效率，可以提供更高的性能，从而满足高实时、低延迟等特性。目前业界尚无支持 C++语言的轻量级微服务框架，所以，为了支持此类系统的微服务集成，本文提出了一种基于 C++语言的轻量级微服务框架设计方案。

1 研究背景

1.1 微服务架构

微服务架构是相对于单块（Monolithic）[5] 架构提出的

收稿日期：2018-08-14；修订日期：2018-09-17

作者简介：唐稳（1974-），男，湖南长沙人，硕士，高级工程师，研究方向为云平台、软件中间件；刘艳辉（1972-），女，河北秦皇岛人，硕士，高级工程师，研究方向为软件中间件。E-mail：wentang9880@sina.cn

新架构模式。其定义请参照 Martin Fowler 的文献［6］。微服务架构能够解决单体式应用维护升级困难、设计开发成本高、软件耦合度高等一系列问题，是分解复杂应用的一种全新架构。

1.2　平台即服务（platform-as-a-service，PaaS）

PaaS 指的是将软件研发和运行的平台作为一种基础服务提交用户，可以看成是组件和中间件层面的服务封装。其优势在文献［7］中有详细介绍。

1.3　微服务框架与 PaaS 云平台

为了支持微服务架构，PaaS 云平台通常集成了微服务框架[8]，例如 CloudwareHub[9]。总的来看，在 PaaS 云平台中，微服务框架具有如下能力。

（1）持续集成和持续交付支撑，实现自动构建、部署、配置、测试和运行微服务，通常需要提供自动化运行的脚本，并与终端集成开发环境结合。

（2）远程过程调用（remote procedure call，RPC）支持[10]，包含同步、异步的 RPC 调用与提供，包括 HTTP、TCP 等多种传输协议的支持。

（3）服务治理，包含服务注册发现、服务负载均衡、服务弹性伸缩、服务故障切换、服务路由、服务流程编排、服务安全、服务监控、服务调用链跟踪等方面[11]。

（4）运行支撑，包含日志中间件、消息中间件、分布式缓存中间件、关系型数据库/NoSQL 数据库访问中间件等中间件服务。

1.4　轻量级 C++微服务框架

本文设计的轻量级 C++微服务框架是一个功能可扩展、可裁剪的微服务运行本地进程容器，可被很多 PaaS 云平台集成，主要包括如下能力。

（1）为微服务之间的交互提供适配多种传输协议的、多种请求模式共存的一体化 C++RPC 调用框架。

（2）在进行 RPC 交互时，可以按需动态接入 PaaS 云平台中服务治理的相关服务，尽量以透明的方式提供给上层 C++微服务。

（3）可以按需动态接入 PaaS 云平台中的运行支撑中间件，以 C++API 接口的方式提供给上层微服务调用。

（4）可以按需在一个进程内集成一个或多个 C++微服务，实现 C++微服务的动态柔性组装。

2　框架设计

如图 1 所示，轻量级 C++微服务框架采用了基于插件技术的设计方法，由 4 个部分组成。其中插件框架提供了 C++插件集成基础环境。服务治理代理堆栈包括一组代理插件，当微服务之间进行 RPC 交互时，代理堆栈可透明地提供访问 PaaS 云平台的服务治理相关服务的能力。运行支撑中间件插件集包括一系列代理插件，以 C++ API 接口的方式为 C++微服务提供了访问 PaaS 云平台的日志、消息、分布式缓存等中间件服务的能力。

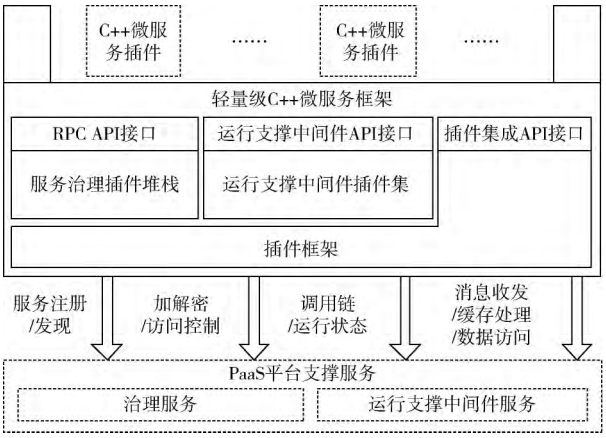

图 1　轻量级 C++微服务框架设计

2.1　插件框架设计

插件技术的核心在于对程序的框架不进行改变，却能实现对其功能的增加与删除。使用公开的插件接口规范，无论是企业还是个人都能实现所需的功能插件进行事务处理和管理，即能提供真正意义上的开放式和简易式使用[12]。

C++微服务框架完全基于插件技术进行设计，将服务治理、微服务 RPC 调用、运行支撑中间件等方面的能力，以及业务微服务均以插件的方式进行封装，从而支持整个微服务程序的动态灵活组装。

插件框架是微服务框架的核心模块，其设计参照了 OSGi 规范，对插件的概念模型进行了规定，对插件集成 API 接口、插件生命周期模型、插件描述文件格式以及插件目录结构进行了规范。插件框架完全基于 C++语言实现，并提供了 C++API 接口。

（1）插件概念模型。如图 2 所示，插件框架定义了一个 3 层概念模型来描述一个插件对外可见的部分：插件、服务和接口，其中插件表示插件模块本身，一般是一个 C++动态连接库或者可执行程序。服务表示了插件在运行过程中动态产生的一个 C++对象，同一个插件可以创建多个服务。接口表示了其它插件访问服务的入口，通常是一个 C++抽象类指针，一个服务可以提供多个接口。在插件框架下，服务提供者通常基于名称注册自己的服务接口，服务消费者发现某个名称的服务接口，并调用该接口访问服务。

（2）插件 API 接口。包括两个部分，插件需要实现的通用 C++ API 接口，包括了插件初始化和结束化接口。插件框架实现的插件集成 C++API 接口，包括了插件查找、插件解析、插件状态获取、服务注册、服务查找、服务注销、内存申请、内存释放等接口。

（3）插件生命周期模型。包括了插件安装、解析、加

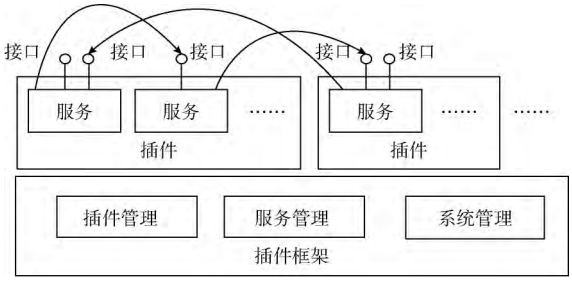

图 2 插件框架设计

载、激活、运行、关闭、卸载等阶段，插件框架基于该生命周期模型对插件进行精细化管理。

（4）插件描述文件格式。基于键值对模型，对插件的名称、厂商、版本、依赖插件等信息进行描述，便于插件框架对插件进行管理，对插件信息进行解析。见表 1。

表 1 插件描述文件示意

Manifest-Version：1.0
Bundle-Name：xxService
Bundle-SymbolicName：xx.xx.xxService
Bundle-Version：1.0.0
Bundle-Vendor：xx
Bundle-Copyright：**版权所有**(c) 2018-2019，xx **公司**
Bundle-Activator：xx_Bundle_Activator；library＝xxSo
Bundle-RunLevel：5
Bundle-LazyStart：false
Require-Bundle：yy.yy.yyService，zz.zz.zzService

（5）插件目录结构。插件框架规定，一个插件通常包含了 bin、log、res、config、manifest 子文件夹，分别存放插件二进制文件及其依赖库、插件日志文件、插件资源文件、插件配置文件以及插件描述文件。

2.2 服务治理插件堆栈设计

微服务治理能力包括了服务注册发现、服务负载均衡、服务路由、服务安全、服务监控等功能。当微服务之间进行 RPC 交互时，这些功能中的一部分需要被调用，以支持相关的治理行为。为了降低 C＋＋微服务的开发成本，这些功能由框架自动调用，对微服务透明。不同微服务对治理能力的需要可能不同，比如有的微服务可能需要增加对私有传输协议的支持，需要增加服务配置文件解析能力，有的微服务需要去掉数据加解密能力等，框架需要支持治理能力的灵活裁剪和增加。

本文设计了服务治理插件堆栈，由一系列治理插件和一个数据传输插件集组成，每个治理插件均提供某一方面的治理能力，例如服务发现插件、负载均衡插件、服务监控插件、数据加解密插件、传输调度插件等。为了以一致的方式管理，所有治理插件均实现 RPC C＋＋API 接口，框架可以在不修改的前提下，根据配置灵活地增加或者去掉治理能力，确保了整个框架良好的可扩展性和可维护性。

数据传输插件集由一组数据传输插件组成，包括 HTTP 插件、TCP 插件、UDP 插件以及私有协议传输插件，应用可以根据传输质量、传输性能、服务间互操作性等方面的要求确定需使用的传输插件。

如图 3 所示，以两个微服务之间的 RPC 交互为例，说明微服务治理插件堆栈的工作原理。

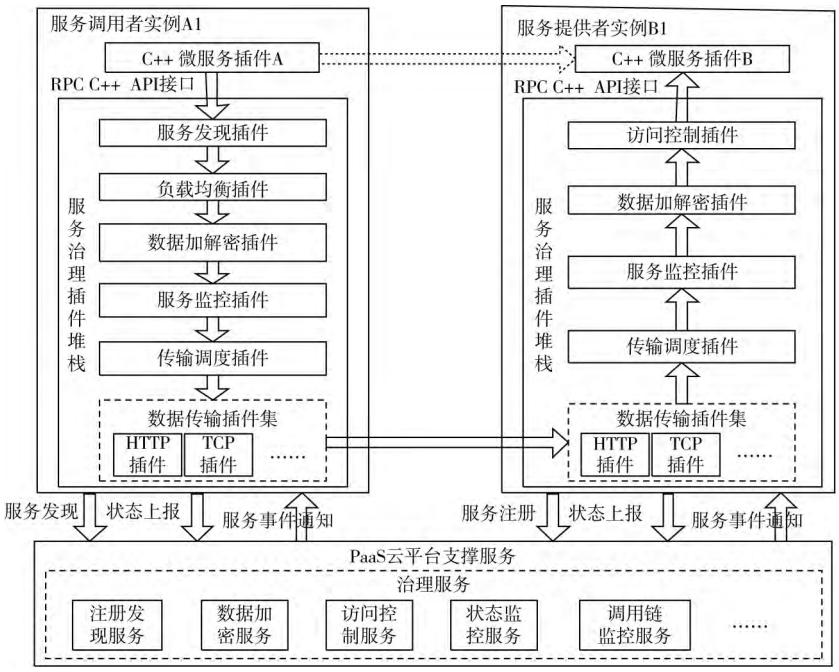

图 3 服务治理插件堆栈工作原理

（1）在服务调用方，C++微服务实例 A1 的插件 A 通过 RPC 接口发起一次向微服务 B 的基于 TCP 协议的 RPC 调用，发出调用请求 R。

（2）A1 的服务发现插件拦截到请求 R，插件与 PaaS 云平台支撑服务的注册发现服务联系，获取到微服务 B 集群的所有实例信息以及负载均衡策略信息，将这些信息补充到请求 R 中。

（3）A1 的负载均衡插件拦截到请求 R，根据微服务集群 B 注册的实例信息以及策略，选择了服务实例 B1 作为请求的接收处理方，将微服务实例 B1 的连接方式补充到请求 R 中。

（4）A1 的数据加密插件拦截到请求 R，对 RPC 请求进行加密，并将加密的密文替换掉 R 中的原始 RPC 请求明文。

（5）A1 的服务监控插件拦截到请求 R，在 R 中添加相应的服务调用链信息，同时将调用链信息上报 PaaS 云平台调用链监控服务。

（6）A1 的传输调度插件拦截到请求 R，从中获取到 B1 的连接方式，选择数据传输插件集中的 TCP 插件进行请求 R 的发送。

（7）A1 的 TCP 插件与 B1 的 TCP 插件取得联系，将请求 R 发送到服务提供者 B1。

（8）B1 的 TCP 插件接收到请求 R，并交给传输调度插件。

（9）B1 的传输调度插件确认接收到了一个完整的 RPC 请求。

（10）B1 的服务监控插件拦截到请求 R，在 R 中添加相应的服务调用链信息，同时将调用链信息上报 PaaS 云平台调用链监控服务。

（11）B1 的数据加解密插件拦截到请求 R，对 RPC 请求进行解密，并将解密的明文替换掉 R 中的 RPC 请求密文。

（12）B1 的访问控制插件拦截到请求 R，基于发起调用的用户名、服务 A 的名称以及 RPC 调用的信息进行权限判定，确认该请求 R 是合法的请求。

（13）B1 的微服务插件 B 最终接收到请求 R，并进行处理。

（14）B1 对 R 的处理结果以相同的方式通过服务治理插件堆栈，最终到达实例 A1 的微服务插件 A。

可见，调用请求和应答在微服务框架的服务治理插件堆栈中穿行，完成了服务发现、负载均衡、数据加解密、服务监控、访问控制等一系列治理任务，这些治理任务由微服务框架自动调用相关的插件完成。在整个过程中，微服务插件仅调用 RPC C++API 即可，实现了治理服务对上层微服务的透明化支持。

2.3 RPC C++API 接口设计

RPCC++API 接口是插件堆栈提供给微服务进行 RPC 访问的 API 接口，为了保证服务治理插件栈的动态组装，要求所有的治理插件都需要实现该接口。RPC C++API 接口在设计时遵循了两个原则。

（1）支持多种 RPC 请求模式。包括同步、异步、投递 3 种模式，分别提供请求的高可靠、实时和高实时 Qos（传输服务质量）。其中同步模式意味着 RPC 请求方在发送完请求后阻塞，直到接收到处理结果。异步模式意味着 RPC 请求方在发送完请求后阻塞，一旦对方确认收到该请求即解除阻塞，后面处理结果到达时会以回调的方式通知 RPC 请求方。投递模式意味着 RPC 请求方在将请求发送到本地缓冲区后继续执行，后面处理结果到达时会以回调的方式通知 RPC 请求方。可见，从并发请求支持方面，投递优于异步，异步优于同步。从开发难度方面，同步比异步简单，异步比投递简单。实际开发过程中采用何种模式需要根据开发人员水平和软件运行技术要求综合考虑。

（2）采用字典类参数传递个性化信息。在 RPC 所有 C++API 接口函数中均设计一个字典类参数，由一组键值对组成，并且可以嵌套，键值都可以灵活定义，便于框架向治理插件传递个性化信息。例如框架需要将找到的服务集群信息传递到负载均衡插件，需要将调用链信息传递到传输调度插件等。字典类参数可以用相同的数据结构描述不同类型的信息，特别适合于类型不确定的接口定义，同时还具备很强的可扩展性，确保了框架能以一致的接口集成所有的治理插件。

以异步 RPC 请求为例，其接口定义见表 2。

表 2 异步 RPC 请求接口定义

```
//异步 RPC 请求 C 接口
virtual int _cdecl AsynCall（
        char* serviceName,     //微服务名称
        IMemBufReader* req,    //请求报文，含内容和长度
        IDictionary* callCtxt,    //字典类参数，表示请求上下文
        int timeout = -1,    //超时，单位为毫秒
        int prioLevel = LEVEL_EXCHANGE_GENERAL   //请求优先级
        ) = 0;
```

2.4 运行支撑中间件插件集设计

运行支撑中间件包括了日志记录、状态缓存、消息交换、NoSQL 数据库访问等服务，其特点是提供 C++API 接口，微服务按需调用。

不同的微服务对运行支撑中间件的需求并不完全相同，比如有的需要进行消息交换和状态缓存，有的需要进行日志记录。而且同一类中间件可能存在多个实现，例如消息

中间件就有 Kafka、RocketMQ、ZeroMQ 等项目，其 API 接口都不一定相同。如图 4 所示，C++微服务框架支持对不同种类、不同实现的中间件代理程序进行 C++插件化封装和接入，为不同的微服务提供了一组恰能满足其个性化需要的运行支撑中间件插件集。

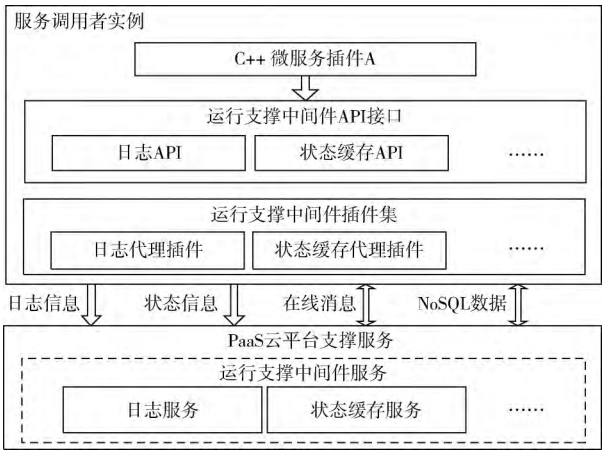

图 4 运行支撑插件集设计

为了支持中间件代理的插件化封装与接入，需要将代理程序与插件框架进行适配。为了支持代理插件的自动启动，需要动态获取中间件代理的初始化运行参数，避免手工配置这些参数，提高运维自动化水平。

（1）与插件框架适配。需要基于中间件代理程序已有 API 接口，按插件框架要求增加插件通用 API 接口的支持。此外，插件的描述文件、生命周期、目录结构等都必须满足插件框架的集成要求。

（2）初始化运行参数动态获取。参数可能包括中间件代理配置文件以及中间件初始化函数的参数。在代理插件启动前，微服务系统需要通过注册发现服务注册这些参数。当代理插件启动时，就可以直接获取到初始化参数了。

3 性能实验

为了说明轻量级 C++微服务框架在性能上优于业界主流微服务框架，并且满足特定领域的高实时、低延迟交互与处理需求，本文基于同步模式与业界主流微服务框架的 RPC 调用性能进行了对比。

本文搭建了性能测试环境，安装了 Dubbo 微服务框架、SpringCloud 微服务框架、轻量级 C++微服务框架及相关的模拟测试微服务。为了保证测试结果的公平性，3 个框架及微服务均部署在同样配置计算机的 Docker 容器中。

针对每个框架均设计一组功能相同的模拟测试微服务，每组微服务均包含 RPC 同步请求和 RPC 请求处理两个微服务实例。其中 Dubbo 和 SpringCloud 框架对应的微服务实例基于 Java 语言编写，而 C++微服务框架对应的微服务实例基于 C++语言编写。

一次 RPC 同步请求的过程是：RPC 同步请求微服务将一个 1KB 字节的字符串传递给 RPC 请求处理微服务，RPC 请求处理微服务将该字符串作为处理结果发回给 RPC 同步请求微服务，RPC 同步请求微服务继续下一次请求。3 组微服务均在单线程场景下进行循环测试，经过多轮测试，统计吞吐率（每秒 RPC 处理次数）和请求处理响应时间（每次 RPC 处理所消耗的时间），取平均值，结果见表 3。

表 3 单线程测试

微服务框架	微服务开发语言	吞吐率/tps	请求处理响应时间/ms
Dubbo	Java	1456	0.687
SpringCloud	Java	506	1.976
轻量级 C++微服务框架（同步模式）	C++	2849	0.351

可见，基于同等条件，轻量级 C++微服务框架在请求吞吐率和响应时间指标两个方面均优于另外两个框架，主要原因在于作为编译型语言，C++比基于 JVM 虚拟机的解释型语言 Java 的运行效率更高。

对本测试中请求处理响应时间做进一步分析，由于 RPC 请求处理微服务仅仅将请求作为结果进行了返回，请求处理时间几乎为 0，所以可以认为本测试的请求处理响应时间仅仅包括了数据传输和路由分派的时间。

一般仿真训练、指挥控制系统所能接受的请求响应时间在几毫秒到几十毫秒之间，而导弹模拟、军事预警系统能接受的时间甚至更短。基于本实验的结果，假设一个请求需要由 6 个微服务进行处理，每个微服务对请求处理时间约为 1 ms。预计 Dubbo 的请求处理响应时间约为 6*（1+0.687）=10.12 ms/次，单个服务吞吐率约为 1000/10.12=99 次/s。预计 SpringCloud 的请求处理响应时间约为 6*（1+1.976）=17.86 ms/次，单个服务吞吐率约为 1000/17.86=56 次/s。预计 C++微服务框架的请求处理响应时间约为 6*（1+0.351）=8.11 ms/次，单个服务吞吐率约为 1000/8.11=123 次/s。可见，轻量级 C++微服务框架比另外两个框架更适合于高实时、低延迟应用。

4 结束语

本文设计了基于 C++语言的轻量级微服务框架，提出了服务治理插件堆栈、运行支撑中间件插件集、微服务插件动态组装等设计思想，对微服务框架的 C++API 接口设计进行了讨论，为 C++微服务的开发与集成提供了完整的插件式解决方案，可以应对微服务系统在服务治理、运行支撑等方面的复杂需求。对性能的测试结果表明，在 RPC 处理的吞吐率、响应时间等方面性能良好，明显优于业界主流的微服务框架。可以为基于 C++语言的模拟仿

真、指挥控制等领域应用的高实时、低延迟等交互与处理需求提供支撑，具有较好的实用价值与应用前景。后续可以对微服务组播交互支持、C++微服务云开发与测试环境等方面进一步展开研究。

参考文献：

[1] Pivotal Software Inc. Spring cloud [EB/OL]. https://cloud.spring.io, 2018.

[2] The Apache Software Foundation. Apache Dubbo [EB/OL]. http://dubbo.apache.org/, 2018.

[3] HUAWEI Software Technology Co. Ltd. Cloud container engine [EB/OL]. http://www.huaweicloud.com/product/cce.html, 2018 (in Chinese). [华为软件技术有限公司. 云容器引擎 [EB/OL]. http://www.huaweicloud.com/product/cce.html, 2018.]

[4] OSChina.NET. Microservices framework open source software library [EB/OL]. https://www.oschina.net/project/tag/461/microservice, 2018 (in Chinese). [开源中国. 微服务框架开源软件库 [EB/OL]. https://www.oschina.net/project/tag/461/microservice, 2018.]

[5] ZHANG Jing, WANG Yanjie, HUANG Xiaofeng. The implement of application framework based on microservice [J]. Computer Systems & Applications, 2017, 26 (4): 82-86 (in Chinese). [张晶, 王琰洁, 黄小锋. 一种微服务框架的实现 [J]. 计算机系统应用, 2017, 26 (4): 82-86.]

[6] WANG Lei. Archtecture and practise of microservice [M]. Beijing: Publishing House of Electronics Industry, 2016: 14-15 (in Chinese). [王磊. 微服务架构与实践 [M]. 北京: 电子工业出版社, 2016: 14-15.]

[7] ZHANG Jian, XIE Tianjun. Research of platform as a service architecture based on the Docker [J]. Information Technology and Informatization, 2014 (10): 131-132 (in Chinese). [张建, 谢天钧. 基于 Docker 的平台即服务架构研究 [J]. 信息技术与信息化, 2014 (10): 131-132.]

[8] ZHANG Jing, HUANG Xiaofeng. Application framework based on microservice [J]. Computer Systems & Applications, 2016, 25 (9): 266-267 (in Chinese). [张晶, 黄小锋. 一种基于微服务的应用框架 [J]. 计算机系统应用, 2016, 25 (9): 266-267.]

[9] GUO Dong, WANG Wei, ZENG Guosun. A new cloudware PaaS platform based on microservices architecture [J]. Netinfo Security, 2015 (11): 15-20 (in Chinese). [郭栋, 王伟, 曾国荪. 一种基于微服务架构的新型云件 PaaS 平台 [J]. 信息网络安全, 2015 (11): 15-20.]

[10] LI Chunyang, LIU Di. Unified application development platform based on micro-service architecture [J]. Computer Systems & Applications, 2017, 26 (4): 44-45 (in Chinese). [李春阳, 刘迪. 基于微服务架构的统一应用开发平台 [J]. 计算机系统应用, 2017, 26 (4): 44-45.]

[11] TAN Yimin. Design and development of platformization service framework based on microservice architecture [D]. Beijing: Beijing Jiaotong University, 2017: 32-35 (in Chinese). [谭一鸣. 基于微服务架构的平台化服务框架的设计与实现 [D]. 北京: 北京交通大学, 2017: 32-35.]

[12] WANG Meisu. Research and implementation of snmp feture plug-in [D]. Beijing: Beijing University of Posts and Telecommunications, 2016: 25-26 (in Chinese). [王美苏. 功能插件化的研究与实现 [D]. 北京: 北京邮电大学, 2016: 25-26.]

（上接第 2311 页）

[10] YANG Jiahui, LIU Fang'ai. Collaborative filtering algorithm based on pap coefficient and Jaccard coefficient [J]. Computer Application, 2016, 36 (7): 2016-2010 (in Chinese). [杨家慧, 刘方爱. 基于巴氏系数和 Jaccard 系数的协同过滤算法 [J]. 计算机应用, 2016, 36 (7): 2006-2010.]

[11] SUN Yiqi, WU Aiguo, DONG Na, et al. Human hand tracking algorithm based on particle filter and improved GVF Snake [J]. Journal of Shanghai Jiaotong University, 2018, 52 (7): 801-807 (in Chinese). [孙一奇, 吴爱国, 董娜, 等. 基于粒子滤波与改进 GVF Snake 的人手跟踪算法 [J]. 上海交通大学学报, 2018, 52 (7): 801-807.]

[12] ZHANG Mingjie, KANG Baosheng. A particle filter tracking method based on graph model [J]. Computer Application Research, 2016, 33 (2): 591-593 (in Chinese). [张明杰, 康宝生. 一种基于图模型的粒子滤波跟踪方法 [J]. 计算机应用研究, 2016, 33 (2): 591-593.]

[13] Weidong Min, Yu Zhang, Jing Li. Recognition of pedestrian activity based on dropped-object detection [J]. Signal Processing, 2017, 9 (2): 238-252.