文章编号：2095-2163(2020)05-0158-06 中图分类号：TP3 文献标志码：A

微服务架构在校园智能安全接送系统中的应用

万 燕，朱 翔
（东华大学 计算机科学与技术学院，上海 201600）

摘 要：针对学生上下学的安全问题，校园智能安全接送系统能够自动快速甄别幼儿园、小学接送人员身份，保护学生安全，且不影响通行速度。在实际使用过程中，发现使用单体架构的系统对于高并发的承载性能不足，且存在过多的请求失败、过长的请求响应时间等问题，为了解决这个问题，引入微服务架构对整个系统进行重构。本文从微服务的由来和设计理念出发，介绍了相比单体架构，微服务架构所具有的优势，并根据单体架构系统已有的业务模块、存在的问题以及实际业务情况，选用 Spring Cloud 框架，对系统进行了架构设计并实现。功能和性能测试表明，微服务架构下的校园智能安全接送系统在可用性、响应速度、并发承载量的表现均超过原系统，且提升了系统的可拓展性和稳定性。

关键词：微服务；单体架构；系统设计；Spring Cloud

Application of Micro-service Architecture in Campus Intelligent Security Pick-up System

WAN Yan, ZHU Xiang
（College of Computer Science and Technology, DONG HUA University, Shanghai 201600, China）

【Abstract】 Aiming at the safety issues of students going to and from school, the campus intelligent security pick-up system can automatically and quickly identify the identities of the kindergartens and elementary school shuttle staff to protect the safety of students without affecting the speed of traffic. However, in the actual use process, it was found that the system using a single architecture was insufficient for high concurrent bearing performance, and there were problems such as excessive request failures and long request response times. In order to solve this problem, the micro-service architecture is introduced to refactor the entire system. Based on the origin of micro-services and its design concept, this article introduces the advantages of micro-services architecture compared to monolithic architectures. Based on the existing business modules, existing problems and actual business conditions of monolithic architecture systems, this article chooses Spring Cloud framework, designs and implements the system architecture. Functional and performance tests show that the campus intelligent security pick-up system under the micro-services architecture surpasses the original system in terms of availability, response speed, and concurrent carrying capacity, and improves the scalability and stability of the system.

【Key words】 Micro-services; Monolithic architectures; System design; Spring Cloud

0 引 言

学生上学接送问题作为校园安全中的重要一环，涉及到学校和家长两方面的配合，现代社会快速发展，大多数父母工作繁忙无法亲自接送孩子，很多时候需要家中老人接送孩子或者孩子独自一人上下学，但是老人毕竟行动不便，而孩子自身缺乏安全意识。父母会担心孩子是否安全上下学，但又无法实时联系确认。

校园智能安全接送系统是针对这一现实情况所设计的系统，它能够快速甄别幼儿园、小学接送人员的身份，保护学生安全，且不影响通行速度。而在此系统实际使用过程中发现，在工作日的早上送孩子入校和晚上接孩子放学两个时间段会产生大量的、单一功能性的请求，在普通的时间段请求量相对较少。基于这种情况，采用传统的单体架构式应用无法对于某些功能单独进行高并发配置，只能对整个系统进行配置，无法根据时间段灵活切换，例如无法单独增加或减少某一服务以达到负载均衡，只能增加或减少整个系统的部署数量，造成部分资源的浪费。此外，单体架构本身也存在开发、维护、扩展困难的问题。而新兴的微服务架构则可以很好的满足这种任务的需要。

1 微服务架构简介

微服务最早由 Martin Fowler 与 James Lewis 于 2014 年共同提出，其目的是有效的拆分应用，实现敏捷开发和部署。微服务架构的设计理念是使用多

作者简介：万 燕(1970-)，女，博士，教授，主要研究方向：计算机图像、人工智能；朱 翔(1996-)，男，硕士研究生，主要研究方向：计算机应用、微服务架构。

通讯作者：朱 翔 Email：2181781@mail.dhu.edu.cn

收稿日期：2020-03-06

个小型服务组合来开发单个应用的方式，每个服务运行在自己的进程中，服务之间使用轻量级机制进行通信，通常使用 HTTP 协议下的应用程序接口，即 HTTP Application Programming Interface（HTTP API）。服务基于业务能力构建，能够独立部署，满足分布式部署的需求。此外，各个服务可以使用不同的编程语言、不同数据存储技术，并保持最低限度的集中式管理，实现单个服务的高内聚、服务之间低耦合的效果。

微服务的关键不仅仅是微服务本身，系统本身也要提供一套基础的架构，使得微服务可以独立的部署、运行、升级。不仅如此，整个系统架构需要使微服务与微服务之间在结构上"松耦合"，在功能上则为一个统一的整体。这种所谓的"统一的整体"的表现方式是统一风格的界面，统一的权限管理，统一的安全策略，统一的上线过程，统一的日志和审计方法，统一的调度方式，统一的访问入口等等[1]。

对比传统的单体架构模式，微服务架构的优势是显而易见的[2-3]：

（1）按业务功能划分使得每一个微服务功能较整个单体架构而言简单很多，且功能之间都具有一定的逻辑性，服务边界明确，更加具有目标性，易于开发和维护。

（2）每一个微服务都可以使用不同的开发语言，基于不同的平台实现，可以根据技术栈自由选择任何合适的技术进行开发，仅仅需要提供统一的对外接口即可，使得每一个微服务的灵活性大大提高。

（3）每一个微服务都可以独立部署，拥有独立的进程。当某一个微服务进行了修改，仅需对其服务进行重新编译、部署。而当某一个微服务出现了问题，也仅影响到某些与其有关联的微服务，对整个应用的影响较小。

（4）微服务架构属于分布式架构系统，各个微服务之间的耦合度较低。随着业务的改变，可以随时增加新的业务微服务或者删除旧的业务微服务，具有强大的横向扩展性。并且可以根据实际需要，对某些微服务进行集群部署，以解决随着用户数量增加而带来的并发问题。

2　微服务架构在校园智能安全接送系统中的应用

当前开发和应用的校园智能安全接送系统，在实际使用过程中暴露出诸多问题，例如高峰时间段并发请求响应时间过长、新增部署服务器操作步骤繁琐、在开发过程中未对代码进行有效管理、单体架构本身的复杂度高、服务间耦合度高导致新增功能或修改旧功能十分困难等严重问题。现引入微服务架构对该系统进行重构，提升系统的稳定性、可用性、高并发性、以及横向扩展性[4]。

2.1　系统主要功能模块

在微服务架构中，每一个业务逻辑服务仅包含功能单一、独立、简单的业务，以便于不同开发者基于自己的技术栈进行开发，从而保证系统的快速迭代和升级[5]。校园智能安全接送系统的主要功能模块如图 1。

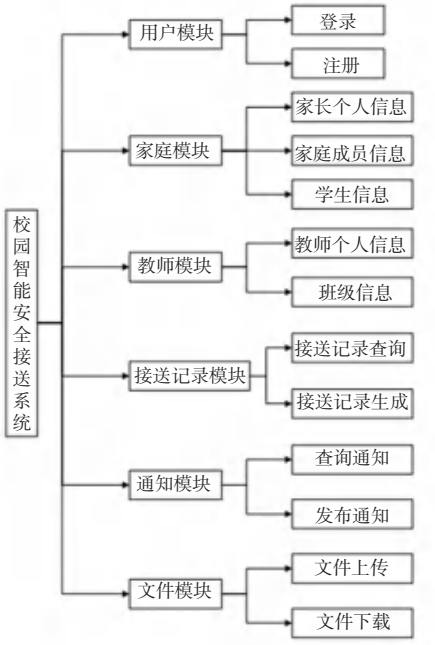

图 1　校园智能安全接送系统主要功能模块

Fig. 1　The main functional modules of the Campus Intelligent Security Pick-up System

（1）用户模块。主要包括登陆和注册功能，用于为家长和教师提供进入系统的接口。

（2）家庭模块。由家庭、家长以及学生 3 个功能模块组成，用于家长用户对于自己家庭的管理。

（3）教师模块。包含教师和班级两个功能模块，用于维护和查看教师个人信息以及其管理的班级的信息。

（4）接送记录模块。根据人脸识别返回的结果生成接送记录，为家长和教师用户提供学生的接送记录查询功能。

（5）通知服务。用于接收校内的通知类消息，教师也可以通过该模块发布通知，通知家长和其他教师相关的校内事务。

（6）文件模块。主要功能是接收上传的图片，并将图片存入文件存储服务器，提供前端的人脸识别进行人脸对比识别，确定学生以及其接送人的身份，保证学生的安全。

2.2 服务划分

根据微服务架构原理,结合当前系统的实际情况,对校园智能安全接送系统进行拆分,划分出系统运行服务和业务逻辑服务。系统运行服务提供整个系统运行所需要的功能;业务逻辑服务提供系统用户所需要的功能。系统划分出的微服务如表 1 所示。

表 1 系统微服务列表

Tab. 1 System micro-services list

序号	服务名称	分类
1	服务注册中心	
2	服务配置中心	系统运行服务
3	路由网关服务	
4	链路监控服务	
5	登陆注册服务	
6	家长服务	
7	学生服务	
8	家庭服务	
9	教师服务	业务逻辑服务
10	班级服务	
11	接送记录服务	
12	通知服务	
13	文件服务	

各个微服务对其它微服务暴露独立的服务接口,用于对外或对内调用,每一个微服务只关注单一独立的业务功能,方便后期扩展或修改。

2.3 架构技术选型

整体微服务平台基于 Spring Cloud 进行搭建,Spring Cloud 是一系列框架的有序结合,利用 Spring Boot 的开发便利性简化了分布式系统的开发,包含了微服务的方方面面,具有无配置文件、组件轻量且优秀,开发简便、灵活等特点[6]。

Spring-boot 作为微服务的基础开发框架,提供包括依赖注入、嵌入式容器、持久化开发等功能。各个微服务的自动化注册和发现由 Eureka 提供。Spring Cloud Config 提供集中管理式的微服务配置。Nginx 以及 Zuul 相互配合提供网关、路由选择、监控以及负载均衡。各个微服务之间的调用通过引入 Fegin 实现。系统整体的链路状态通过 ZipKin 监控。

2.4 系统整体框架

基于微服务的校园智能安全接送系统的架构如图 2 所示。

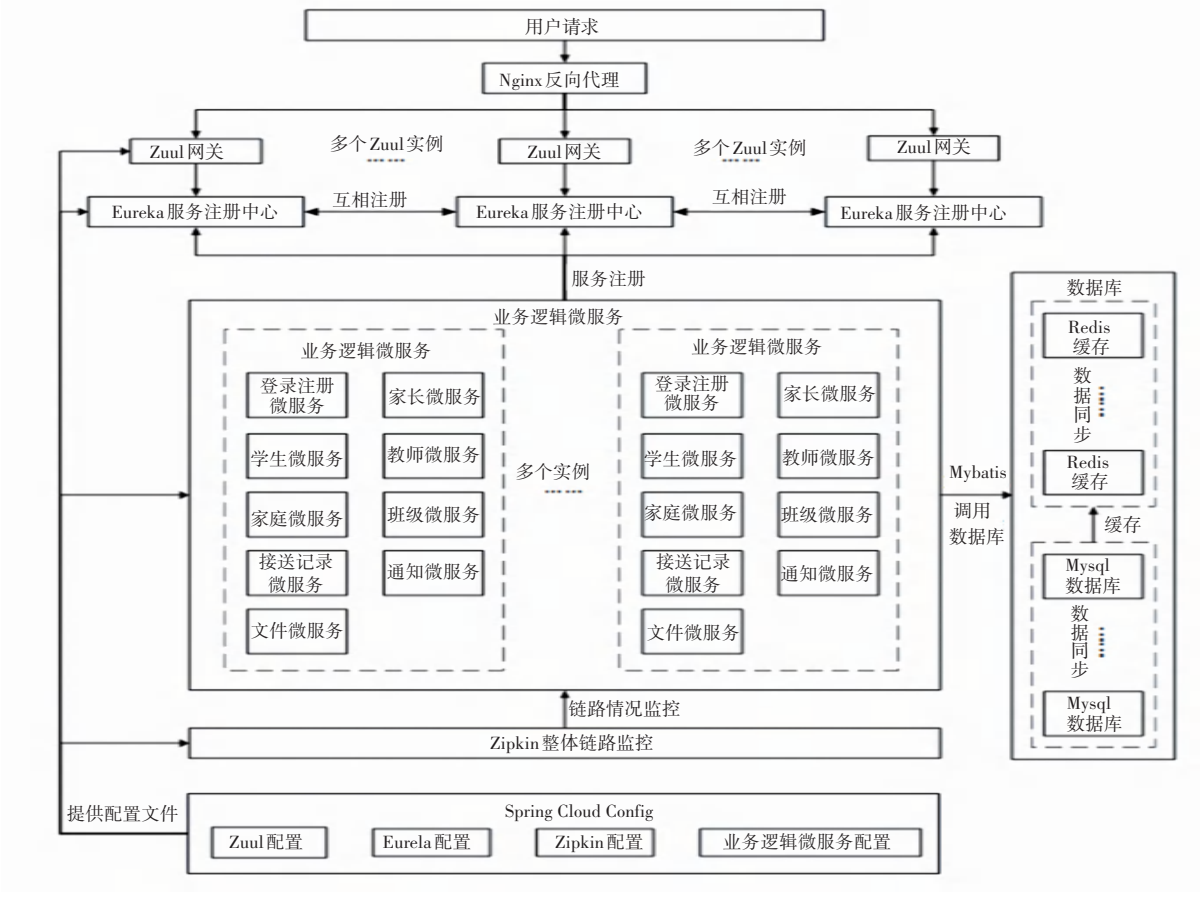

图 2 校园智能安全接送系统整体架构

Fig. 2 The overall architecture of the Campus Intelligent Security Pick-up System

基于微服务的“高内聚、松耦合”的设计思想，为了实现系统高可用、高并发、高性能的三个目标，保持系统的开放性、可扩展性，对现有系统提供的业务进行重构，使之解耦成相互独立且功能专一的服务，服务之间通过 Spring Cloud 的 Feign 组件进行通信以及负载均衡，通过各个服务互相协作构建供外部访问的应用系统[7]。

校园智能安全接送系统框架主要分为 4 个模块：

(1)外部访问模块。Nginx 作为反向代理服务器对外提供接口地址，Zuul 作为微服务网关向 Nginx 提供微服务内部访问的接口，通过 Nginx 和多个 Zuul 组合成为系统的统一对外接口，用户仅需访问 Nginx 提供的对外接口，即可使用系统相对应的功能[8]。

(2)服务配置及治理模块。该模块由 Spring Cloud Config 以及 Spring Cloud Eureka 组成，所有微服务的配置文件由 Config 微服务统一管理，方便修改和部署。Eureka 负责服务发现和注册，将微服务信息提供给 Zuul 和 Feign 用于查询各个微服务的地址以及进行负载均衡。

(3)业务逻辑与服务模块。由重构、拆分出来的多个相互独立、可拓展的微服务构成，包括业务逻辑相关的微服务与基础服务。业务逻辑微服务包括用户微服务、家庭微服务、班级微服务等。基础服务则是一些通用微服务，包括 redis 微服务、文件微服务以及工具类微服务。

(4)数据库模块。由 redis 和 Mysql 构成的数据库模块，基于微服务的理念，对于每一个需要数据库的微服务均构建了一个单独的数据库，这样虽然会造成部分数据冗余，但是对于单个服务的开发、管理、扩展而言有很大的益处。同时，采用了 redis 作为数据库的二级缓存，提高在高请求量、高并发环境下的数据读取速度。

2.5　高并发、高可用处理

在单体架构系统的使用过程中，学生早上上学和下午放学两个时间段是学生出入校的高峰，此时有识别接送人和学生以及生成接送记录这两种功能的大量重复的请求。原有系统对这类情况并没有特殊处理，导致在此种情况下请求会丢失、报错，或是服务响应超时、出错、返回速度慢等问题，严重影响用户的实际体验。引入微服务架构，通过以下 5 点，优化了系统在高并发情况下的可用性。

(1)Nginx 和高可用 Zuul 组合的双层网关。引入 Nginx 后，Zuul 不会直接接收外部的请求，而是通过 Nginx 转发请求，Zuul 会根据高可用策略，启动多个实例。当 Nginx 收到请求后，会根据下辖的 Zuul 实例个数以及承载量，进行负载均衡。不同的 Zuul 网关可以同时承担部分请求转发和微服务查询的负担，进行异步处理，提高并发量和可用性。

(2)针对 Zuul 的配置优化问题。由于 Zuul 整合了 Hystrix，而 Hystrix 的核心功能中包含资源隔离机制，目的是将多个依赖服务的调用分别隔离到各自的资源池内。这是一个很重要的防止服务雪崩的安全措施，但 Zuul 默认使用的信号量隔离机制以及其默认的配置，导致 Zuul 在面对高于 100 的并发量时会直接报错，需要调整信号量大小，或者使用线程隔离机制同时调整线程池的大小，以适应高并发时的请求数量。本文采用调整信号量大小的方式解决高并发请求。

(3)负载均衡。负载均衡是高可用和高并发处理的基础，将同种请求给予相同微服务的不同实例进行处理，减少单个服务器的压力的同时，也进行了异步处理，提高并发处理数量和响应速度。负载均衡不仅在双层网关中应用，在内部的 Feign 中也会被应用，提高微服务内部的并发量和可用性。且由于微服务架构“高内聚，低耦合”的特点，可根据需要，对特定的微服务启动多个实例进行负载均衡，而不需要像单体架构一样需要额外启动所有的服务实例进行负载均衡，从而减少了资源的使用。

Zuul 和 Feign 通过整合的 Ribbon 组件对请求进行负载均衡，默认的负载均衡算法是简单的轮询机制，这种方法适用于部署在配置区别不大的服务器上的实例，而对于部署在配置区别较大的服务器上的实例，适合使用指定权重的方式，增加配置较好的服务器的轮询几率。本文由于所有实例均运行于同一服务器，所以使用 Ribbon 默认的轮询机制进行负载均衡。

(4)Feign 配置。Feign 默认使用基于 Java Development Kit(JDK)提供的 URLConnection 调用 HTTP 接口，其不具备连接池，不能很好的适应高并发的情况，而 Apache HttpClient 和 okhttp 都支持配置连接池功能，可以通过修改配置使用这两者，但需要注意的是 URLConnection 要比 Apache HttpClient 和 okhttp 在调用速度上更优，因此需要根据实际情况来采用不同的 HTTP 请求方式。本文采用 Apache HttpClient 并通过设置连接池以应对高并发请求。

(5)服务重试机制和熔断机制。在原系统的使

用过程中,发现请求过多时,会发生请求报错,但不修改任何错误再次重试却成功,或是请求长时间无反应的情况,针对此种情况,引入了服务重试机制和熔断机制。

①由于服务器卡顿、数据库连接超时或是网络震荡等原因造成的请求报错,通过 Ribbon 的重试机制,对目标微服务再次发送请求,值得注意的是:由于重试机制会再次发送请求,所以当微服务确实出现逻辑问题时,会导致一个请求被循环发送多次,当存在大量类似请求时,容易造成系统内部大量资源被占用,引发整个系统卡顿甚至崩溃,因此重试的次数和时机需要根据实际需要调控,本系统中设定重试 5 次,均失败则返回错误信息。

②因为服务器积压的请求过多,引起处理速度慢、响应请求时间过长,则通过 Hystrix 的熔断机制:当超过规定时间而服务没有返回任何消息时,断定此服务目前不可用,此时可以通过触发服务重试机制,使得 Ribbon 重试发送请求给其他的实例处理,提高系统的可用性。如果没有其他实例,则返回用户报错信息。需要注意的是:Hystrix 默认的熔断时间仅 1 s,需要根据实际进行调整。本系统根据实际请求响应的时间,设定为 5 s。此外,还需要修改 Hystrix 资源隔离机制,默认是线程隔离机制,需要调整其的线程池大小,本系统中调整为 500 大小。

3 实验与分析

3.1 实验环境

该实验的测试环境为:操作系统 Ubuntu 14.6 版本,32 G 内存,处理器为 Intel 2.6 GHz,JDK 版本为 1.8 的 Java 平台,数据库为 MySql 5.5.27 版本,使用 Tomact 4.0 版本进行部署。单体架构应用和微服务架构中的各个微服务都均只部署一个实例。采用 Jmeter 工具,模拟用户向单体架构系统与重构的微服务架构系统发送请求,主要记录两种架构在调用服务时产生的数据,以及服务器响应时间[9]。

以下对比分析单体架构系统与微服务架构系统在高并发环境下的性能差异,以获取家庭成员信息以及生成接送记录为例。

3.2 实验结果

对比分析单体架构系统与微服务架构系统在高并发环境下的性能差异,以获取家庭成员信息以及生成接送记录。

3.2.1 获取家庭成员信息耗时对比

对两种架构的系统分别模拟每秒 100 个用户同时请求,持续 5 s,请求获取家庭成员信息的接口,结果如图 3 所示。

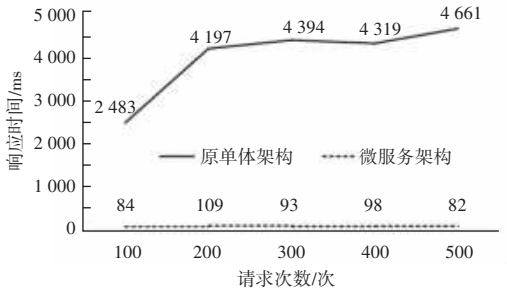

图 3 获取家庭成员信息响应时间对比

Fig. 3 Comparison of response time for obtaining family member information

由图 3 可知,在面对 5 s 的 100 级并发,共 500 次请求的情况下,单体架构的响应时间分别为 2 483 ms、4 187 ms、4 394 ms、4 319 ms、4 661 ms,而微服务架构响应时间大幅度减少,分别为 84 ms、109 ms、93 ms、98 ms、82 ms。此外,单体架构随着请求数量的增加,之前的请求还未响应,导致请求数量堆积愈来愈多,响应速度越来愈慢,而微服务则没有此类的问题。

3.2.2 生成接送记录耗时对比

对两种架构的系统分别模拟每秒 100 个用户同时请求,持续 5 s,请求生成接送记录的接口,结果如图 4 所示。

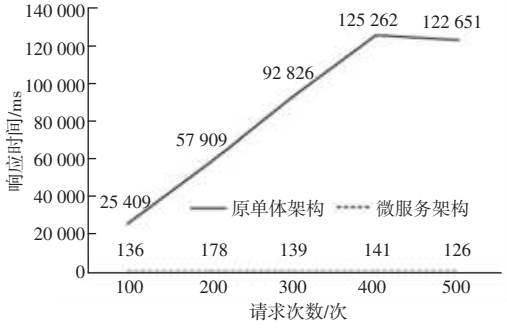

图 4 生成接送记录响应时间对比

Fig. 4 Comparison of response time when generating pick-up records

单体架构由于没有对高并发进行优化,导致在高并发的情况下,响应速度达到平均 84 110 ms,而微服务仅需 144 ms,是一个极大的进步。

经过实验结果分析,重构后使用微服务架构与单体架构相比,极大的减少了在高并发情况下的请求响应时间,此外,该系统还可以通过部署多个服务实例,进一步提高系统性能[10]。

4 结束语

本文介绍了微服务的由来,设计理念。结合传

(下转第 168 页)