计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2019,28(4):52−60 [doi: 10.15888/j.cnki.csa.006843]

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

基于微服务的维修资金管理系统①

刘从军[1,2], 刘　毅[1]

[1](江苏科技大学 计算机学院, 镇江 212003)
[2](江苏科大汇峰科技有限公司, 镇江 212003)
通讯作者: 刘　毅, E-mail: 381200225@qq.com

摘　要: 为了解决传统住宅维修资金管理系统功能模块复杂, 开发维护过程繁琐, 并且新增需求难以及时处理, 提出了基于微服务架构的系统设计开发. 本文设计了基于微服务架构的住宅维修资金管理系统, 将系统划分为几个微服务模块分别实现. 系统使用 Spring Cloud 来搭建微服务架构, 并且实现了服务注册和发现、负载均衡、路由网关以及容错处理等技术, 最后实现了各个微服务功能. 采用该微服务框架, 降低了系统的耦合性, 加快了系统开发周期以及使系统的部署和维护更为便捷.

关键词: 微服务架构; Spring Cloud; 维修资金管理

引用格式: 刘从军,刘毅.基于微服务的维修资金管理系统.计算机系统应用,2019,28(4):52–60. http://www.c-s-a.org.cn/1003-3254/6843.html

Maintenance Fund Management System Based on Micro Service

LIU Cong-Jun[1,2], LIU Yi[1]

[1](School of Computer Science, Jiangsu University of Science and Technology, Zhenjiang 212003, China)
[2](Jiangsu Keda Huifeng Technology Co. Ltd., Zhenjiang 212003, China)

Abstract: In order to solve the issues such that function module of the traditional housing maintenance fund management system is complicated, the process of development and maintenance is tedious, and the new demand is difficult to deal with in time, the system design and development based on the micro service architecture is proposed. This study designs a residential maintenance fund management system based on micro service architecture, and divides the system into several micro service modules. The system uses Spring Cloud to build a micro service architecture, and implements the technology of service registration and discovery, load balancing, routing gateway, and fault tolerance processing, and finally implements various micro service functions. The micro service framework reduces the coupling of the system, speeds up the development cycle of the system, and makes the deployment and maintenance of the system more convenient.

Key words: micro service architecture; Spring Cloud; maintenance fund management

“微服务”的概念, 虽然诞生时间不长, 却迅速的成为了软件架构领域争相讨论的热点. 在微服务架构理论不断完善的同时, 具体实施微服务架构的相关技术框架也正在不断的涌现出来.

国内最出名的微服务框架是阿里巴巴旗下服务化治理框架 Dubbo, 该框架目前除了应用于阿里巴巴集团的各成员站点, 也被国内很多互联网公司使用.

国外的微服务框架 Spring Cloud 最为突出, Spring Cloud 是 Spring 家族的产品, 它包含了开发分布式系统所必需的多个组件, 覆盖了微服务架构开发的方方面面, 在微服务开发中相对比较成熟. 微服务框架在各个系统中的应用也越来越广泛.

① 收稿时间: 2018-10-15; 修改时间: 2018-10-31; 采用时间: 2018-11-05; csa 在线出版时间: 2019-03-28

相对于微服务架构, 传统的单体式架构系统往往存在以下的问题:

(1) 开发效率低: 代码的编写在同一个项目中进行, 开发人员提交代码时冲突严重.

(2) 代码维护难: 项目中包含太多模块, 模块之间耦合性高, 当某一功能需要修改时, 可能会关联或影响到其他功能的使用[1].

(3) 部署效率低: 单体式架构需要整体部署, 修改系统中一个小功能都需要重新部署应用. 当项目功能复杂、代码量大的时候, 构建和部署项目需要花费大量时间, 并且当修改其中一个功能时, 虽然其他功能没有问题, 但依然不能正常使用[2].

(4) 可靠性不高: 所有模块运行在同一进程. 任何一个模块出现故障, 可能会导致整个系统崩溃.

(5) 扩展能力低: 因为个模块之间耦合度高, 需要添加新的功能时, 往往需要大规模的改动项目.

随着房地产市场的飞速发展, 房屋的维修资金的交存使用以及监管问题成为了广大业主以及政府相关部门所关心的重要问题[3]. 目前存在的住宅维修资金管理系统, 大多属于基于单体式架构的应用系统, 这些系统也存在着这些问题: 1) 系统每一个功能的新增或者修改都需要重新部署整个系统, 届时其他功能也无法使用. 如新增维修资金查询的图表分析的功能时, 虽然只涉及到查询部分的功能, 但是资金交存以及资金的使用办理等功能依然无法使用. 2) 系统资金交存办理时, 如果同一时刻交存人数过多, 系统处理特别慢, 系统性能不足以支撑高并发访问. 3) 系统中如果某一功能出现故障, 会导致整个系统不可用. 比如系统中的查询模块出现问题时, 整个系统都会崩溃不能运行, 此时资金交存、使用等功能虽然没有问题仍会不可使用.

为解决现阶段维修资金管理系统建设中存在的问题, 本文提出了基于 Spring Cloud 的微服务架构, 将系统中高度耦合的功能分解到各个离散的微服务中以实现对应用系统的解耦[4]. 将系统拆分为几个微服务同时进行开发, 加快系统开发周期, 降低了系统部署的难度. 每个服务之间通过 RESTful API 接口进行通信, 以业务为中心, 构建起自动化运行机制, 实现了集中式管理[5].

1 微服务架构

1.1 微服务架构介绍

微服务架构 (Microservice Architect) 是近年来软件开发领域兴起的一种新型软件架构, 它提倡将单块架构的应用划分成一组小的服务, 服务之间互相协调、互相配合, 为用户提供最终价值[6]. 每个服务运行在其独立的进程中, 服务与服务间采用轻量级的通信机制互相沟通[7]. 每个服务都围绕着具体业务进行构建, 并且能够被独立的部署到生产环境、类生产环境等. 微服务架构旨在通过将功能分解到各个离散的服务中以实现对解决方案的解耦[8].

微服务架构如图 1 所示.

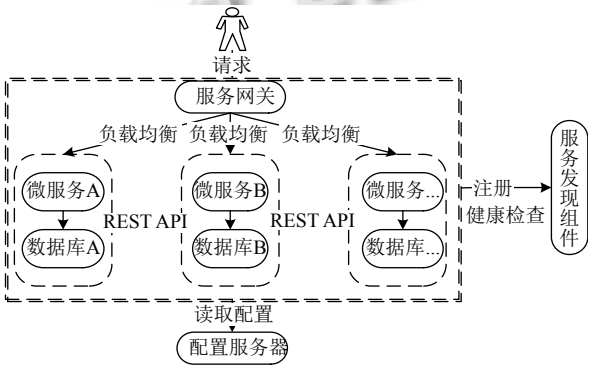

图 1 微服务架构图

微服务架构的优点[9]可以总结如下:

(1) 微服务将巨大单体式应用分解为多个小服务, 每个服务业务清晰、功能明确简单、代码量小, 开发和维护单个微服务相对简单. 解决了系统开发和维护的复杂性问题.

(2) 每个微服务可以有不同的人员进行开发, 并且开发技术栈不受限制, 开发人员可以采取不同的技术进行开发.

(3) 每个微服务可以独立的部署, 相对于单体应用来说, 微服务架构下若某一功能需求发生变更, 只需对这一单独的服务重新编码部署, 不影响其他服务的使用.

(4) 每个服务独立扩展, 可以根据新的需求, 实现细粒度的扩展.

1.2 Spring Cloud 微服务框架

Spring Cloud 是一个微服务框架, 主要为了简化分布式系统的开发. 利用 Spring Boot 一键启动、部署的特点对云应用开发中的服务注册发现、API Gateway、断路器、服务配置治理、负载均衡等操作都提供了简单的开发方式.

由于 Spring Cloud 微服务框架相对于其他微服务

框架更为成熟, Spring 社区也更为活跃, 故选用 Spring Cloud 作为系统微服务的开发框架.

2 系统分析与设计

2.1 系统需求分析

国务院颁布的《住宅共用部分设施设备维修基金管理办法》中对“维修基金”的定义是: 专项用于住宅共用部位、共用设施设备保修期满后的大修、更新、改造的资金[10].

随着各地房地产业的发展, 维修资金的使用的管理越来越复杂, 这就需要建立科学的、信息化的管理体系, 来实现维修资金的自动化管理[11]. 住宅维修资金管理系统应该满足以下的需求:

(1) 对小区、楼盘以及资金交存的银行和业主等信息进行整合, 建立房屋的信息数据库, 实现对小区、楼盘等信息的规范化管理.

(2) 建立一套规范而且完善的住房维修资金管理业务流程, 实现维修资金交存、使用等管理工作的规范化与自动化.

(3) 实现维修工程、意见征询以及资金拨付等审批的科学化、信息化. 使业务的审核流程变得更为简便快捷.

(4) 实现对维修工程数据以及资金交存、使用等数据的统计分析. 方便业主查询自己的维修资金, 方便领导对维修工程的审查与决策.

根据系统的功能需求, 分析得出系统的业务流程主要分为资金的交存和使用两部分, 系统总体的业务流程图如图 2 所示.

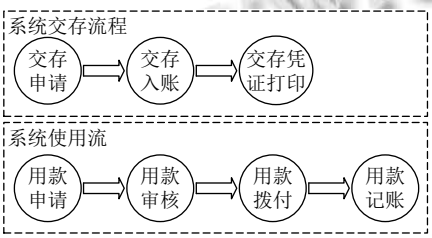

图 2 系统业务流程图

2.2 系统架构设计

2.2.1 微服务拆分

根据系统的需求分析, 将整个系统分为 4 个微服务模块来完成功能开发, 分别是基础数据管理服务、资金交存服务、资金使用服务、查询统计服务. 四个微服务的功能如下:

基础数据管理服务主要包括对楼盘信息、物业区域信息、物业公司信息、维修公司信息、开发企业信息、银行信息和银行账户信息等数据的管理.

资金交存服务主要是指资金交存管理、资金续交、交存调整三个功能.

资金使用服务包括从维修工程开始到资金结付等所有内容, 是住宅专项维修资金管理的重要组成部分. 主要是维修工程申报和审核、维修意见的征询和审核、维修工程的验收和审核以及维修资金的拨付和审批.

查询与统计分析模块包括了维修资金交存记录查询、维修资金使用记录查询、银行账户流水查询、维修工程信息查询以及这些数据的图表统计和报表打印等.

2.2.2 微服务总体架构设计

为了住宅专项维修资金管理系统每个微服务的开发与分工更容易界定, 同时降低开发难度并且提升工作效率; 以及系统后期部署时, 可以实现业务逻辑和应用界面分别部署在不同的服务器中, 提高系统服务的并行性能, 并且系统各服务依赖的数据库也单独部署, 提升系统数据的安全性和可靠性; 为系统复杂的业务流程之间的关系解耦, 使系统达到“高内聚, 低耦合”的设计目标而采用了分层架构的设计模式. 依照以上设计原则, 将系统分为基础支撑层、数据层、服务层、接口层以及应用层进行开发, 系统架构图如图 3 所示. 各层包含内容如下:

(1) 基础支撑层: 主要包括机房、服务器及网络设备、安全组件和光纤宽带接入等组成, 支撑和保障了整个服务平台稳定安全的运行.

(2) 数据层: 数据层主要包资金信息的数据、项目信息的数据、和电子材料数据等. 项目采用统一的数据中心管理模式, 对平台数据进行统一有效的管理, 解决各服务间数据对接和数据共享问题.

(3) 平台服务层: 主要包括基础的业务服务如基础数据管理服务、资金交存服务、资金使用服务和查询统计分析服务. 还包括以 Spring Cloud 为基础的微服务架构服务, 如微服务的基石服务注册和发现组件 Eureka、基于 Ribbon 的负载均衡组件、基于 Hystrix 断路器及容错处理机制以及基于 Spring Cloud Config 的自动化服务等. 以上搭建了微服务架构, 并完成基础业务服务的功能.

(4) 接口层: 使用 API Zuul 网关统一所有的服务接口, 连接平台服务与系统应用, 包括与银行的数据对接等, 使用 JWT 实现身份认证等权限的管理.

(5) 应用层: 包括对于业主用户的意见征询和账户查询以及工程查询的功能、对于小区物管的意见征询和工程申报等功能、对于政府相关部门的业务办理以及审核等功能和对于银行系统的资金数据报接功能以及移动端应用的功能.

系统微服务架构由 Spring Cloud 组件构建组成. 通过 Eureka 组件实现服务注册与发现; Feign 组件实现服务的 Ribbon 负载均衡与 Eureka 的整合, 加强网络数据处理能力、提高网络的灵活性和可用性; 将基础数据处理、资金交存等四个业务微服务注册到 Eureka 服务端, 并通过 Fegin 实现的负载均衡完成对外的调用. 通过 Zuul 实现服务网关, 统一向外系统提供 RESTful API; Hystrix 实现微服务的容错处理, 避免在微服务架构中个别服务出现异常时引起的故障蔓延; 数据库使用 SQL Server 数据库, 各个业务服务都有独立数据库, 并且使用 Redis 作为缓存处理.

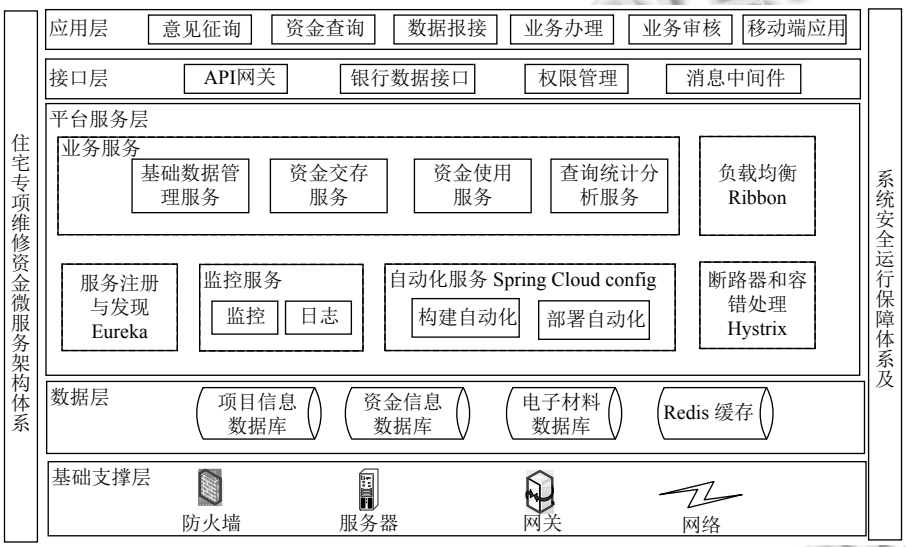

图 3 系统架构图

3 微服务架构实现

3.1 服务注册和发现

服务的注册和发现是微服务架构的基础[12]. Spring Cloud 搭建服务注册中心的核心就是服务发现组件: Eureka. Eureka 一般搭配负载均衡组件 Ribbon 使用, 服务发现的架构图如图 4 所示.

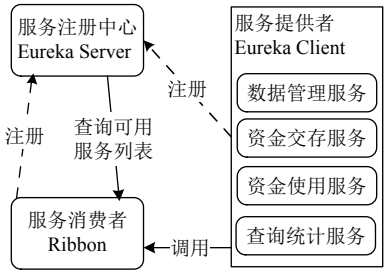

图 4 服务发现架构图

Eureka Server 端实现如下:

(1) pom 文件中添加 spring-cloud-starter-eureka-server 依赖.

(2) 在 SpringBoot 主类上添加@EnableEureka Server 注解.

```
@EnableEurekaServer
@SpringBootApplication
public class EurekaServerApplication {
}
```

(3) 在 application.properties 中配置端口号和注册中心的地址, 如下所示:

```
server.port=9003
eureka.client.serviceUrl.defaultZone=_blankhttp://localhost:${server.port}/eureka/
```

启动 EurekaServerApplication 后, 在浏览器中输入

_blankhttp://localhost:9003, 看到如图所示页面, 表明 Eureka 正常启动.

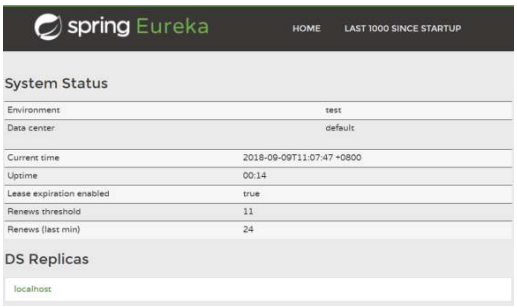

图 5 Eureka 服务页面

Eureka Client 端实现如下:

(1) 在基础数据管理、资金交存、资金使用、查询分析微服务以及 Zuul 路由网关、Hy 断路器等服务中的 pom 文件中添加 spring-cloud-starter-eureka-server 依赖.

(2) 在各自的启动类上添加@EnableEurekaClient 的注解.

(3) 在各服务的 application.properties 配置文件中添加如下配置:

eureka.client.serviceUrl.defaultZone=_blankhttp://localhost:9003/eureka/

之后依次启动各微服务, 在 Eureka Server 页面中可以查看各服务状态, 如图 6 所示.

图 6 服务注册到 Eureka 状态图

各服务状态为 up 表明各服务已经成功注册到 Eureka Server 中了.

3.2 负载均衡

生产环境下, 各个微服务有多个实例, 将消费者的请求分摊到每一个实例中需要使用负载均衡机制. Ribbon 是一款可以控制 HTTP 和 TCP 行为的负载均衡器. Ribbon 可以基于负载均衡算法如轮询算法、随机算法等帮助服务消费者去请求服务提供者的地址[13].

Spring Cloud 中 Feign 组件整合了 Ribbon 和 Eureka 来提供均衡负载的 HTTP 客户端实现. Feign 实现如下:

(1) 在 Pom 文件中添加添加 spring-cloud-starter-feign 依赖.

(2) 创建一个 Feign 接口, 并添加@FeignClient 注解. 以下以查询统计分析微服务为例:

```
@FeignClient(name="QUERY-SERVICE")
public interface FeignClient {
@RequestMapping(value="/listAllWxgc/{gcmc}", method=RequestMethod.GET)
public List<>listAllWxgc(@PathVariable("gcmc") String gcmc);
}
```

其余微服务实现 Fegin 接口与上面类似.

3.3 路由网关

路由网关是为了让所有的微服务对外只有一个接口 API, 我们只需要访问一个网关地址, 即可由路由网关将我们的请求代理到不同的服务中[14]. 如果没有路由网关, 多个服务提供给前端调用地址管理错综复杂, 增加了客户端的复杂性, 认证也相对麻烦, 每个服务都需要编写相同的认证.

图 7 和图 8 分别展示了用户请求多个微服务时使用和不使用路由网关的区别.

Spring Cloud 是通过 Zuul 来实现路由网关的, Zuul 支持自动路由映射到 Eureka Server 上注册的服务. Spring Cloud 提供了注解@EnableZuulProxy 来启动路由代理.

Zuul 组件具体实现代码如下:

(1) pom 中添加 spring-cloud-starter-zuul 依赖.

(2) 在启动类上添加注解@EnableZuulProxy, 声明一个 Zuul 代理. 该代理使用 Ribbon 来定位注册在 Eureka Server 中的微服务[15].

```
@SpringBootApplication
@EnableZuulProxy
public class ZuulApplication {
public static void main(String[] args) {
SpringApplication.run(ZuulApplication .class, args);
}
}
```

(3) 编写配置文件 application.yml

server.port: 9090

spring.application.name: eureka-zuul

eureka.client.serviceUrl.defaultZone: http://localhost:9003/eureka/

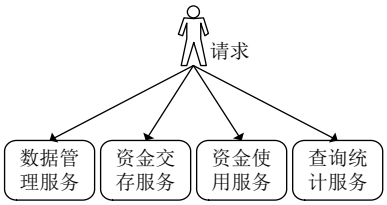

图 7 用户请求多个微服务

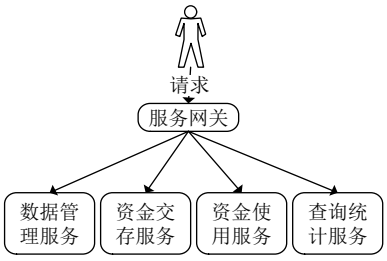

图 8 使用路由网关

3.4 容错处理

当服务消费者请求服务时, 如果此时服务提供者响应时间过长, 那么请求会被等待. 高负载下, 这种问题可能会导致系统崩溃[16]. 采用 Hystrix 组件的容错处理机制可以解决这一问题. 具体实现代码如下:

(1) 在 pom.xml 中添加 spring-cloud-starter-hystrix 依赖.

(2) 在各服务启动类添加注解@EnableHystrix, 为项目启用断路器.

@EnableDiscoveryClient

@EnableHystrix

public class SpringCloudClientDiscoveryApplication {…}

4 系统测试

4.1 微服务接口测试

项目开发采用了前后端分离的形式, 项目首先实现了后端接口功能, 后端各服务接口开发完成后, 需要先对接口进行测试, 本项目中引入 Swagger2 作为接口测试工具.

Swagger 2 是一个开源项目, 用于描述和记录 RESTful API. Swagger 2 是语言无关的, 可扩展到除 HTTP 之外的新技术和协议. Swagger UI 允许其他 API 开发人员或用户与 API 的资源交互, 而没有任何实现逻辑到位. 应该结构化, 使其具有信息性, 简洁性和易于阅读.

Swagger 的实现如下:

(1) 在 pom 文件中添加 springfox-swagger2 和 springfox-swagger-ui 的依赖.

(2) 创建 Swagger2 配置类.

在 Application.java 同级创建 Swagger2 的配置类 Swagger2

@Configuration

@EnableSwagger2

public class Swagger2Config {…}

(3) 在各服务 API 中添加声明.

以下以资金使用服务中的维修工程查询接口为例, 配置 swagger 声明:

@ApiOperation(value="维修工程查询")

@RequestMapping(value="wxgc_list", method= RequestMethod.POST)

public RestAPIResult<Boolean>

pay4(@RequestParam("gcmc") String gcmc,

@RequestParam("kgsj") String kgsj,

@RequestParam("sgdw") BigDecimal sgdw,

String accessToken) {…}

运行各服务, 访问 http://localhost:8080/swagger-ui.html 页面, 如图 9 所示, 可以查看每个微服务的接口.

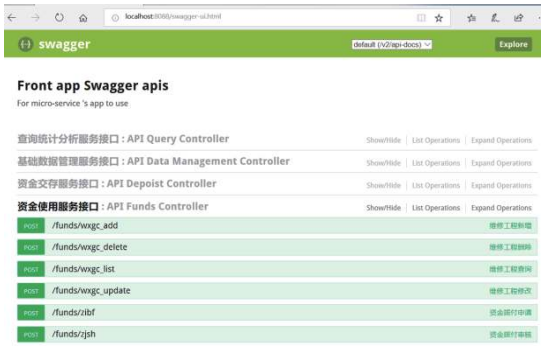

图 9 各服务接口

如图 10 所示, 点击 Expand Operations 按钮可以展开每个接口的详细信息, 包括接口参数, 提示代码等.

图 10 维修工程查询接口详细

输入好接口所需参数, 点击 try it out 按钮可以执行该接口方法, 返回值如图 11 所示.

图 11 维修工程查询接口测试

4.2 系统功能测试

本节以资金使用微服务为例, 对资金使用环节部分功能进行测试.

维修申报列表页面如图 12 所示. 由小区物业管理单位 (或社区) 进行维修工程申报, 写申请维修原因、申请维修费用、维修方案等信息.

工程申报之后可以选择分摊户进行分摊, 分摊之后可以提交住建部门领导审核. 维修工程提交之后由物管中心进行审核, 主要对维修工程的维修内容、施工公司信息以及资金预算信息等进行审核. 审核通过之后才可进行维修意见征询流程. 审核页面如图 13.

图 12 维修工程申报

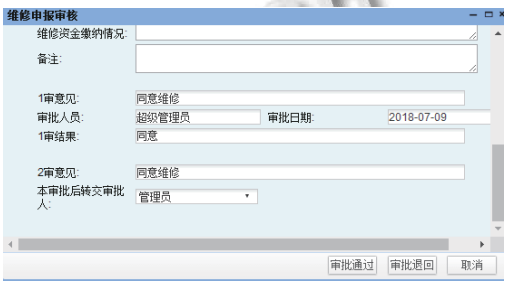

图 13 维修工程审核

维修工程经市物业管理中心审核通过后, 由小区物管中心打印维修意见征询表找与此维修工程申报相关的受益业主进行签字确认. 由小区物管单位根据受益业主签字确认情况, 在系统中确认已同意维修的受益业主. 意见征询页面如图 14 所示.

图 14 维修工程意见征询

维修工程意见征询通过后, 项目便开始实施. 当项目竣工后, 施工单位组织工程验收并填写维修工程验收表, 经业主委员会、物业公司签字确认. 根据业主委员会、物业公司在维修工程验收表中的签字意见, 在系统中进行维修验收登记, 界面如图 15 所示.

维修工程验收通过之后, 维修公司选择需要付款的结算单向市物业管业中心提交付款申请 (分为首期款和尾款). 资金拨付申请页面如图 16 所示.

4.3 系统性能测试

本文使用 Apache Bench 模拟 100 个用户对系统中不同业务同时发出访问请求, 在高并发请求访问环

境下, 对微服务架构改进下的系统以及传统的 Java Web 架构下的两种维修资金管理系统处理不同业务平均响应时间进行比较. 记录数据图如表 1 所示.

图 15 维修工程验收登记

图 16 资金拨付申请

表 1 本系统与传统住宅专项维修资金管理系统部分业务处理高并发访问平均响应时间对比数据分析 (单位: ms)

业务名称	传统系统	改进系统
资金交存办理	89	46
维修工程申报	78	39
业主账户查询	60	33
资金使用统计	96	53

从表 1 中的对比可以看出, 在模拟 100 位用户并发访问的情况下, 使用微服务架构的系统比传统系统响应时间明显下降, 时间少了大约 1/2, 这说明在处理高并发业务时, 采用微服务架开发的住宅专项维修资金管理系统性能明显优于原系统, 用户体验更加友好.

图 17 展示了维修系统部分业务实例在高并发场景随时间内存的消耗情况. 上图中横轴表示时间, 单位为秒, 纵轴表示内存使用大小, 单位为 MB, 上方红色线条表示传统系统测试实例的内存消耗总和, 下方蓝色线条表示基于微服务架构系统的测试实例的内存消耗总和. 从图中可以看出, 在高并发场景下, 基于微服务架构的系统内存消耗小于传统单体式架构的内存消耗, 并且内存使用相对平稳.

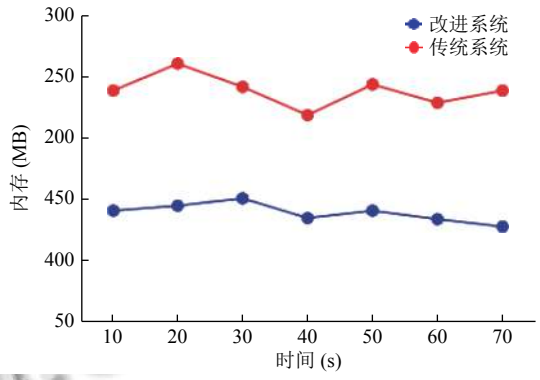

图 17 内存消耗测试

基于微服务架构的住宅专项维修资金管理系统采用了分布式架构, 将系统整体功能拆分为各个服务, 分别部署在不同的服务节点, 可以有效避免因为单节点服务压力过大引起系统宕机的风险. 系统微服务框架搭建采用了基于 Ribbon 的负载均衡组件, 使得对于系统的高并发访问请求进行分摊, 实现系统的高可用、对网络压力有了缓解, 解决了传统系统内存压力过大而导致系统资源加载过慢的情况, 更好的提升了用户的体验.

5 结束语

本文设计并实现了基于微服务架构的住宅维修资金管理系统. 相对于传统的资金管理系统, 本系统将功能划分为基础数据管理、维修资金交存、维修资金使用和查询统计分析四项微服务. 每个微服务可以单独开发和部署, 其中一个服务出现问题时, 可以单独修改不会影响到其他服务的使用, 采微服务架构, 系统处理高并发请求时, 效率更高. 本文采用了 Spring Cloud 框架及其相关组件来搭建微服务架构, 同时实现了服务注册和发现、负载均衡、路由网关以及容错处理机制. 通过该服务框架可以对住宅维修资金管理系统进行快速开发, 实现系统的组件化、独立部署、维护风险降低等优势.

参考文献

1 洪华军, 吴建波, 冷文浩. 一种基于微服务架构的业务系统设计与实现. 计算机与数字工程, 2018, 46(1): 149–154. [doi: 10.

3969/j.issn.1672-9722.2018.01.032]
2 靳磊. 微服务在铁路调度管理系统改造中的应用. 铁路计算机应用, 2017, 28(4): 43–47. [doi: 10.3969/j.issn.1005-8451.2017.04.011]
3 陈燕, 廖烈谊. 物业专项维修资金管理系统设计. 无线互联科技, 2017, (3): 61–62. [doi: 10.3969/j.issn.1672-6944.2017.03.026]
4 李春阳, 刘迪, 崔蔚, 等. 基于微服务架构的统一应用开发平台. 计算机系统应用, 2017, 26(4): 43–48. [doi: 10.15888/j.cnki.csa.005757]
5 黄林, 杨军, 徐亮亮. 基于微服务构建模型的应用系统增量更新算法. 计算机与现代化, 2018, (2): 39–43, 88. [doi: 10.3969/j.issn.1006-2475.2018.02.009]
6 Newman S. 微服务设计. 崔力强, 张骏, 译. 北京: 人民邮电出版社, 2016.
7 蒋勇. 基于微服务架构的基础设施设计. 软件, 2016, 37(5): 93–97. [doi: 10.3969/j.issn.1003-6970.2016.05.024]
8 黄嘉诚, 董晶. 基于微服务的智能档案服务系统设计与实现. 电子设计工程, 2018, 26(2): 26–30. [doi: 10.3969/j.issn.1674-6236.2018.02.007]
9 周建丁. 七牛技术总监肖勤: 微服务架构实践经验分享. http://www.csdn.net/article/2015-08-07/2825412. [2015-08-12]
10 渠天龙. 我国物业小区住宅专项维修资金法律制度研究[硕士学位论文]. 太原: 山西大学, 2010.
11 吴敏. 住房维修资金管理系统的设计与实现[硕士学位论文]. 哈尔滨: 哈尔滨工程大学, 2011.
12 张晶, 黄小锋. 一种基于微服务的应用框架. 计算机系统应用, 2016, 25(9): 265–270. [doi: 10.15888/j.cnki.csa.005347]
13 马雄. 基于微服务架构的系统设计与开发[硕士学位论文]. 南京: 南京邮电大学, 2017.
14 郑彬彬. 基于微服务的 OJ 系统重构与优化[硕士学位论文]. 上海: 东华大学, 2017.
15 龚鹏. 微服务分布式构架开发实战. 北京: 人民邮电出版社, 2018.
16 陆文虎. 基于微服务架构的制造执行系统设计与实现[硕士学位论文]. 杭州: 浙江大学, 2018.