计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(8):81−88 [doi: 10.15888/j.cnki.csa.008049]

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

# 基于微服务架构的企业 ERP 设计与应用①

桂　俊, 沈迎春

(武汉数字工程研究所, 武汉 430205)
通讯作者: 桂　俊, E-mail: xxh@cssc709.net

**摘　要**: 随着云计算、大数据的发展, 以及企业应用的规模、复杂度增加和产品需求不断扩展, 传统单体式架构 ERP 系统中存在可扩展性差、灵活性低等弊端. 本文提出了使用微服务架构来构造企业应用, 首先分析了微服务架构的特点, 针对微服务架构的服务独立、低耦合、可扩展等优势, 设计了基于微服务的企业 ERP 系统架构, 解决了 ERP 开发中接口协作问题, 提出基于微服务的实现技术 Spring Cloud 来重构应用, 最后详细论述了在开源环境下微服务应用开发过程. 包括 Spring Boot 子系统构建、服务注册中心搭建、负载均衡架构设计、网关设计等, 并完成了系统的接口和性能测试, 论证了基于微服务架构系统的易维护、扩展等优势.

**关键词**: 微服务架构; 企业 ERP; Spring Cloud; 单体架构; 负载均衡

引用格式: 桂俊,沈迎春.基于微服务架构的企业 ERP 设计与应用.计算机系统应用,2021,30(8):81–88. http://www.c-s-a.org.cn/1003-3254/8049.html

## Design and Application of Enterprise ERP Based on Micro-Service Architecture

GUI Jun, SHEN Ying-Chun

(Wuhan Digital Engineering Institute, Wuhan 430205, China)

**Abstract**: Amid the progress in cloud computing and big data, as well as the increasing scale and complexity of enterprise applications and the expanding demand for products, the traditional separate-architecture ERP system exposes disadvantages including poor scalability and low flexibility. In this study, we propose to use micro-service architecture to construct enterprise applications. Firstly, we analyze the characteristics of micro-service architecture. In light of the advantages of micro-service architecture, such as independent service, low coupling, and great scalability, we design the enterprise ERP system architecture based on micro-service and solve the problems of interface cooperation in ERP development. Then, we introduce the implementation technology, Spring Cloud, based on micro-service to reconstruct the application. Finally, we elaborate the development process of micro-service in an open source environment, including construction of Spring Boot subsystems and the service registration center, design of load balancing architecture and gateways. The system interface and performance testing are completed, demonstrating the advantages of the micro-service architecture based system, such as easy maintenance and scalability.

**Key words**: micro-service architecture; enterprise ERP; Spring Cloud; separate architecture; load balancing

传统 ERP 体系结构往往基于单体的三层架构, 伴随着业务功能的扩展, 组件之间依赖越来复杂, 业务数据的大量积累等导致系统的运行速度和用户体验越来越差, 如何解决上面困境, 以满足高可用、高并发的要求, 迫切需要一种新的软件架构模式来解决上述问题. 文献 [1] 指出传统单体架构设计的系统内部紧密耦合, 灵活性差, 以及基于面向服务架构 SOA 的应用系统的安全性不足、负载均衡等局限性. 文献 [2] 采用新型微

① 收稿时间: 2020-11-22; 修改时间: 2020-12-22; 采用时间: 2021-01-08; csa 在线出版时间: 2021-07-31

服务架构的应用系统可以有效解决 ERP 系统暴露出的软件复杂性问题, 各个微服务能够单独部署, 每个独立的微服务模块负责传统 ERP 系统中一个或多个关联的业务系统. 为了适应应用系统性能要求、需求快速变化, 传统单体架构由于耦合度强、维护复杂, 不利于新功能扩展、快速交付, 随着开源技术的成熟, 进化出满足开发敏捷性、持续交付、可伸缩、最终一致的新兴系统架构即微服务架构[3]. 本文首先研究了微服务架构的理论和技术基础, 分析了基于微服务构造应用的优势, 从基础服务、业务服务、公共服务、接入服务等几个维度来描叙基于微服务的应用架构, 针对 ERP 的具体需求设计了基于微服务架构的企业 ERP 架构, 详细介绍了微服务的实现技术 Spring Boot、Spring Cloud、负载均衡、网关架构设计、微服务间调用 Feign, 以订单子系统为例, 详细论述了微服务的开发过程.

## 1 微服务架构设计

微服务架构是一种软件架构模式, 它将一个完整的应用从数据存储到业务逻辑的开发垂直切分多个不同的服务, 服务运行于独立的进程之中, 具有自己独立的生命周期, 服务之间采用统一的轻量级通信协议相互通讯, 具有独立开发、维护、部署和扩展的优势[4]. 一个基于微服务架构的应用平台通常由业务服务、接入服务、公共服务、基础服务等 4 部分组成, 微服务应用架构设计如图 1 所示.

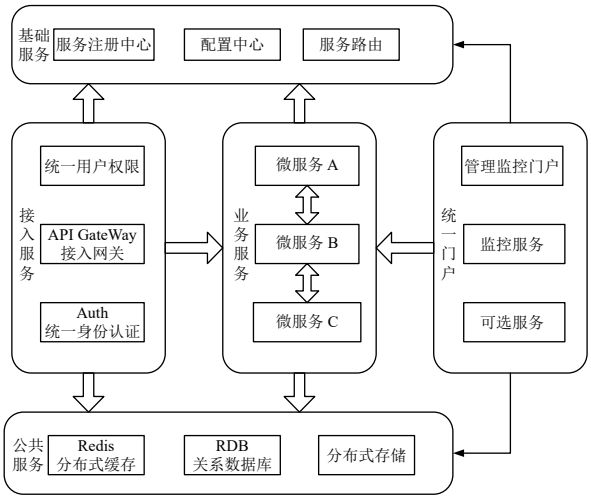


图 1 微服务应用架构设计

其中业务服务负责实现各个服务, 同时提供运维监控手段对服务进行监控, 业务服务层将各个业务系统提供的服务接口进行集成, 形成统一的服务标准为接入层提供服务. 接入服务层通过服务路由、负载均衡等手段负责服务的分发, 基础服务包括使用标准的中间件进行服务的注册, 公共服务包括基础数据存储服务 MongoDB、关系型 RDB、缓存服务 Redis, 服务接口之间采用轻量级 RESTful 格式消息交互机制. 统一门户负责管理监控业务服务.

## 2 微服务架构企业 ERP 设计

### 2.1 ERP 的总体需求

ERP 系统的功能之一是实现企业业务流程的自动化. 当企业客户下达订单之后, 销售部门结合企业的库存和产能审核客户提交的订单是否能被满足, 如果库存物料和生产产能同时能够满足订单需求, 对客户订单予以确认, 然后根据订单上所列产品, 向仓储部提出产品出库申请, 若库存产品能够满足订单需求, 直接出库由配送人员将产品交付给客户. 如果库存不满足情况下, 由生产部根据订单确认生产, 生产出产品提交仓储部进行产品入库, 当生产部申领的物料, 当前的库存无法满足时, 需要向采购部提出采购申请, 采购申请经过审批确认之后, 由采购人员选择合适的供应商进行物料的采购, 并将采购回来的物料提交给仓储部进行物料入库, 入库之后的物料由仓储部交付给申领物料的生产人员. 同时采购人员将采购单提交给财务部, 由财务部进行付款确认, 进行财务结算, 打款给供应商. 客户签收之后, 销售人员将订单提交给财务部由财务部进行收款确认, 财务部再进行财务结算, 最后整个业务流程结束.

通过对业务流程的分析, 接下来需要解决微服务中的领域划分及微服务粒度划分问题, 遵循面向服务的领域驱动设计原则进行领域建模, 通过业务领域拆分出一组领域模块微服务, 每个微服务必须是高度内聚, 符合开闭原则、自治等, 仅聚集自己的业务、领域服务间通过接口交互达到松散耦合. 通过领域分解识别出若干个业务功能子域, 每个业务功能域对应一个子系统, 通过子系统再次分解最终产生子流程再以迭代的方式逐步分解细化, 最后达到与用户交互的级别的子流程称为原子流程. 分析上述 ERP 销售订单业务流程后, 通过领域分解可以识别出销售订单、库存、生产、采购、财务等几个子域, 每个子域将其设计为一个领域服务, 每个领域服务对应一个功能集合, 比如订单微服务包括订单创建、订单查询、订单审核等操作原子微服务, 而每个原子微服务又可以被其他微服

务共享, 每个微服务可以看做是多个原子微服务的聚合服务. 接下来需要定义微服务的接口, 每个接口里封装了若干操作, 包含接口名称、请求 url、request 参数、封装返回结果等.

### 2.2 基于微服务的 ERP 系统架构设计

上面详细分析了整个 ERP 的业务流程, 大致梳理出 ERP 的业务功能, 从业务领域中识别出若干微服务模块, 包括库存管理、生产管理、销售管理及财务管理等, 每个模块继续以迭代的方式分解为更细粒度的业务服务. 基于微服务架构的 ERP 系统架构如图 2 所示, 整个体系结构自底向上分为 4 层, 基设施层提供以数据存储为主的基础服务, 包括内存数据库 Redis 提供缓存服务及关系型数据库资源, 微服务层包括所有业务微服务模块的基础业务服务及由基础服务组合而成的聚合微服务, 基础微服务通过操作业务数据集来实现单一的业务规则, 而聚合微服务往往实现跨业务模块的复杂业务规则. 各个微服务在注册中心组件完成注册部署, 微服务之间通过 Feign 方式交互, 并向上层提供接口服务. 服务网关提供外部访问的统一入口, 外部通过网关接入微服务, 同时提供动态路由、授权安全、调度、监控等的服务网关功能及 Nginx 反向代理实现服务器的负载均衡. 应用交互层包括 Web 页面、APP 页面及调用的第三方系统, 往往采用前后端分离技术, 基于 RESTful 风格交互, 后端提供 Rest 接口, 前后端基于 HTTP 协议通信、JSON 格式数据传递. 接下来根据系统架构来进行实现框架的选型, Spring Cloud 是 J2EE 环境下最流行、先进的微服务实现框架, 它是一序列微服务开发工具集, 包括微服务的分布式配置、服务发现、路由、负载均衡、断路器、服务网关、消息传递等应用组件的提供. 微服务架构是一序列单体微服务应用的集合, Spring Boot 框架通过简化配置快速简单开发 Spring 应用, 可以利用 Spring Boot 专注于单个应用的快速构建. 采购、生产、库存各个子系统都是独立的微服务, 通过 Eureka 进行服务治理, 各个子系统将提供的服务注册到 Eureka Server 中, 作为服务提供端, 同时各个子系统作为服务消费端使用 Rest 接口对服务提供端进行调用. API 网关为客户端提供统一接口, 服务网关 Gateway 是一种基于 MVC 模式的响应式 Web 框架, 它简化前端调用逻辑, 根据请求代理到不同的服务, 通过转发将前端请求路由到后端服务中, 调用单个服务或通过 API 组合调用多个微服务返回结果[5]. 网关使用 Nginx 做路由, 能够将前端的请求分流分量的发送给后端服务, 实现负载均衡, 减少后端服务压力.

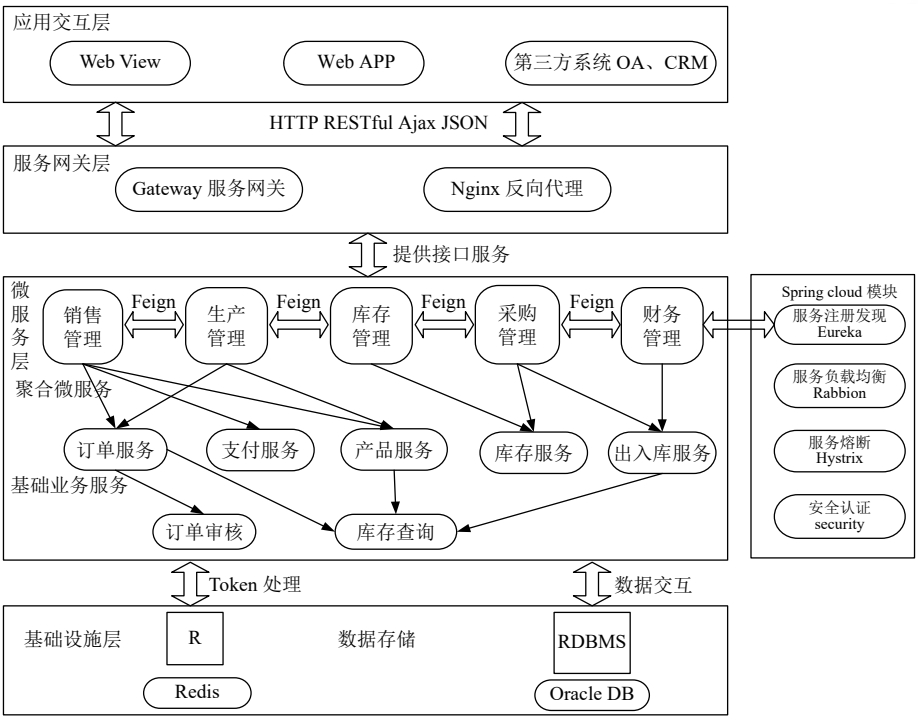


图 2 基于微服务的 ERP 系统架构图

### 2.3 协作接口设计

定义协作接口是规定团队合作的契约, 保证开发不同限界上下文的特性团队能够并行开发. 本文设计的 ERP 系统涉及到订单、客户、员工、文件等实体, 同时还要与第三方 OA 系统的集成工作. 订单的上下文映射如图 3 所示.

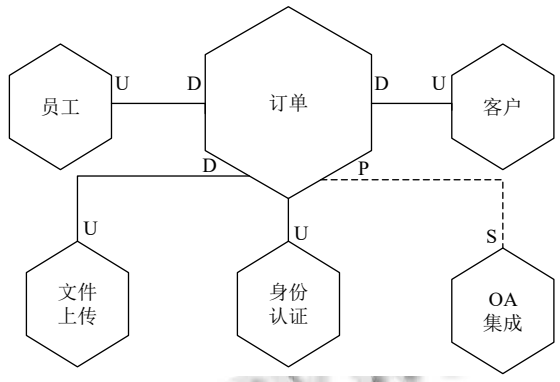


图 3 订单的上下文映射

图 3 中确定除与 OA 集成上下文之间采用“发布者/订阅者”模式, 无需引入防腐层和开放主机服务, 其余限界上下文之间的协作都是“客户方/供应方”模式, 要定义的协作接口其实就是各个限界上下文的应用服务接口. 协作接口采用事件机制, 定义的是限界上下文之间协作的接口, 协作接口完全可以根据之前确定的上下文映射获得, 每个协作关系都意味着一个接口, 不同的上下文映射模式可能会影响到对这些接口的设计. 以生产订单上下文为例, 与前面上下文映射的不同之处是将订单与 OA 集成之间的协作改为了事件机制, 记录与订单上下文相关的协作接口如图 4 所示.

| 生产者 | 消费者 | 模式 | 业务场景 | 服务定义 | 接口功能描述 |
|---|---|---|---|---|---|
| 客户 | 订单 | 客户方/供应方 | 跟踪需求订单 | ClientService::queryById(OrderId):Client | 获取订单的客户信息 |
| 员工 | 订单 | 客户方/供应方 | 跟踪需求订单 | EmployeeService::queryById(OrderId):Employee | 获取订单中需求承担者信息 |
| 文件上传 | 订单 | 客户方/供应方 | 创建市场需求 | UploadFileService::upload():void | 上传需求订单附件 |
| 订单 | OA 集成 | 发布者/订阅者 | 创建需求订单 | OrderCompleted | 通知需求订单承担者 |
| 认证 | 订单 | 客户方/供应方 | 全部场景 | CertificationService::certification(userId):certificationResult | 用户身份认证 |

图 4 记录与订单上下文相关的协作接口

使用生产者 (Producer) 与消费者 (Consumer) 来抽象客户方/供应方模式与发布者/订阅者模式, 多个模式的组合后面的服务定义就应该是遵循 RESTful 服务定义的接口, 开放主机服务, 客户方/供应方与开放主机服务之间的组合. 回顾 OA 集成上下文的上下文映射, 是将事件持有的内容转换为要发送消息通知的内容以及送达的地址, 作为订阅者的 OA 集成上下文在接收到事件, 然后发送消息通知. 订阅的事件应该是相同的, 应该将 Order Completed 修改为 Notification Ready 事件, 处理事件的逻辑完全相同.

## 3 基于微服务的 ERP 系统实现

### 3.1 Spring Boot 子系统构建

开源环境下微服务的开发采用基于 Spring 框架全栈技术, 包括 Web 开发框架 SpringMVC、服务开发框架 Spring Boot、服务治理框架 Spring Cloud 及 ORM 持久框架 Mybatis, 建立统分布式缓存 Redis 集群. 首先我们需要创建 maven 项目, 在 pom.xml 中增加项目中使用到的服务模块 module, 比如配置、服务注册及各种业务服务, 并添加相关依赖, 便可启动服务注册中心. 由于各个微服务可以独立的开发和部署, 但每个服务的数据实体, 控制层、实现层和数据持久层的风格都是一致的, Spring Boot 用来快速构建单个微服务应用, 首先, 在 application.properties 进行配置, 包括数据源及 Mybatis 配置. 然后是业务系统的开发, 这里以订单子系统为例, 订单管理包括产品出库和产品入库两大类, 每种都包括订单的创建、查询、审核及生成收拣货任务, 同时订单子系统与客户子系统及产品子系统产生关联, 首先定义好订单相关接口, 并且接口请求 URL 以 RESTful 风格呈现, 整个订单子系统包括订单实体类、订单控制类、业务服务接口类用以提供多样的方法供控制层调用、数据库映射关系类等 4 种, 用户进入订单查询页面, 输入订单编号、仓库信息等, SpringMVC 控制层接受到查询参数后, 调用订单服务接口中的查询方法, 订单服务实现类封装了查询客户和产品信息的方法, 通过调用 DAO 组件的 Mapper 接口类返回数据, 在服务实现类进行组装再返回给前端用户. 最后启动 Spring 应用主程序, 启动嵌入式的 Tomcat 并初始化 Spring 组件. 基于 Spring Boot 构建订单服务模块如图 5 所示. 搭建 Redis 集群后, 创建主从节点, Spring Boot 添加 Redis 依赖后, 在 application.yml 配置 nodes、poolConfig 等信息, 在 Redis 配置类中修改 Redis 序列化方式, 然后在服务类中注入 RedisTempate 就可以完成数据对象的读写操作, 比如可以将产品类以 JSON 格式存储到 Redis 中, 或者将 Redis 中以 JSON 对象形式返回的产品 list 转换为 JSON 字符串.

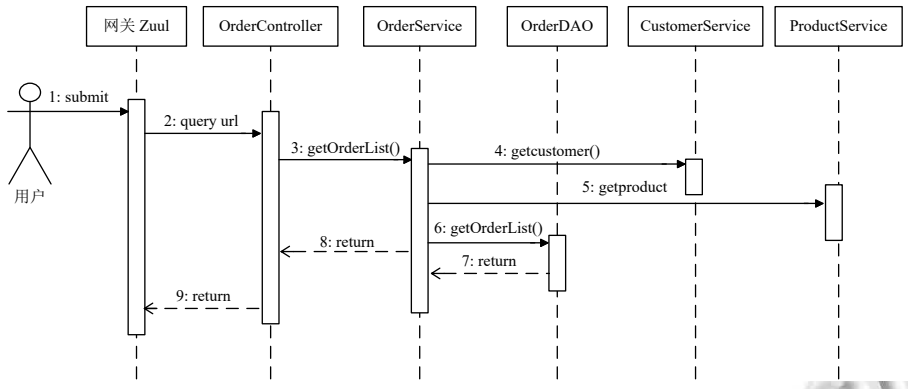


图 5 订单服务时序图

### 3.2 服务注册发现与负载均衡架构

基于 Spring Boot 的服务开发完毕后, 需要利用 Eureka 搭建一个服务注册发现架构, 首先创建服务注册中心, 在 Spring Boot 工程下添加 Eureka-server 依赖及相关配置, 创建产品服务 product-service, 通过在启动类中使用@EnableEurekaServer 注解声明该服务是 Eureka 的服务端, 再添加端口, 修改主机名, 配置服务相关地址, 接下来服务提供者需要将服务注册到服务注册中心供服务消费者订阅, 启动服务端后就可以通过 URL 访问 Eureka 查看注册服务信息. 客户端将自身注册到服务中心, 同时从服务中心获取服务, 服务消费者在 Pom 文件中, 添加起步 Eureka 依赖, 在启动类注解@EnableDiscoverClient, 创建接口来获取产品服务实例, 然后启动网关服务, 客户端统一通过 Zuul 网关访问内部服务, 网关接受发送请求, 从 Eureka 获取可用服务, 由 Ribbon 进行负载均衡, 分发到后端服务实例去调用[6,7]. 负载均衡技术将来自客户端大量访问流量捕获到负载均衡服务器中, 通过调用特定的调度算法向不同服务器分发访问流量. 在微服务架构中, 服务消费者 EurekaClient 向 Rabbion 发起 RestTemplate 请求后会被 LoadBalancer 拦截, 根据 URL 获取服务名, 根据 EurekaClient 中服务状态返回到 Rabbion 的服务注册表中的信息找到匹配的服务. 具体配置如下, 首先添加 Ribbon 的起步依赖, 在 application.yml 中制定端口号及服务注册地址 URL, 通过在 RibbonConfig 类加上@LoadBalanced 注解来注入 RestTemplate 开启负载均衡功能. 服务注册发现与负载均衡架构如图 6 所示, 具体代码如下:

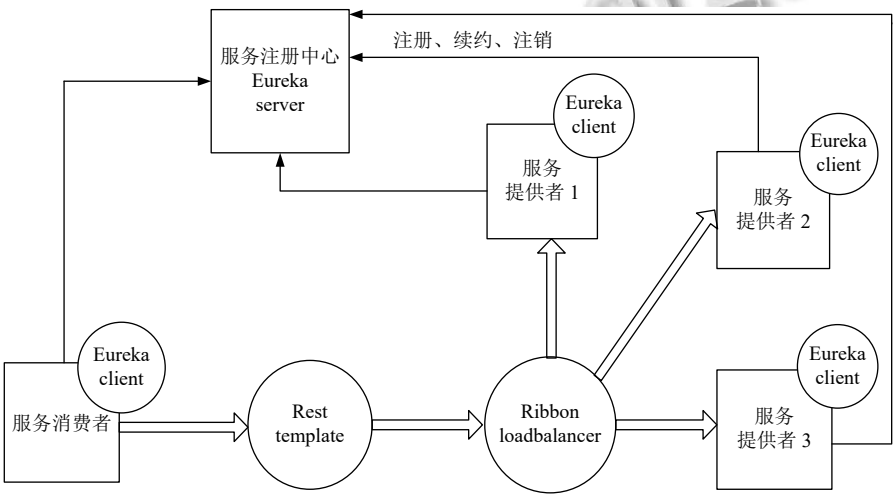


图 6 服务注册发现与负载均衡架构

```
@Configuraton
public class RibbonConfig{
@LoadBalanced
RestTemplate restTemplate ()
```

{ return new RestTemplate(); }

}

定义一个 ProductService 类, 注入 restTemplate 对象, 在该类的 QueryStock() 方法以 Rest 方式调用 Eureka client API 接口, 服务层代码如下:

@Service

public class ProductService {

@Autowired

RestTemplate restTemplate ;

public String QueryStock (String cPdCode) {

Return restTemplate.getForObject(

“http://eureka-client/QueryStock?cPdCode=” +cPdCode, String.class);

}}

在 OrderController 类加上@RestController 注解, 开启 RestController 的功能, 添加 Get 方法的接口, 调用 productService 类的 QueryStock() 方法, 代码如下:

@RestController

public class OrderController {

@Autowired

private ProductService productService;

@GetMapping("/QueryStock")

public String QueryStock(@RequestParam (required = false, defaultValue="83010042") String cPdCode) {

return ribbonService.QueryStock(cPdCode); }

}

### 3.3 网关设计

网关服务作为最上层服务, 提供统一入口, 承担权限身份认证、限流、监控、接口转发工作, 调用下游的基础接口服务, 返回数据呈现给用户. Spring Cloud 中通过响应式的 Spring Webflux 来实现 API Gateway, 执行请求路由到后端服务, 执行 API 组合、协议转换等操作. Spring Cloud 服务网关架构如图 7 所示, 它包括 Main 包、API 包、代理包, Cofiguration 类定义了 Spring beans, 它负责路由与 Order 相关的请求[8–10], OrderHandlers 类实现各种请求处理程序方法, 使用 API 组合获取订单的详细信息. 处理程序使用远程代理调用后端服务. 具体实现代码如下所示:

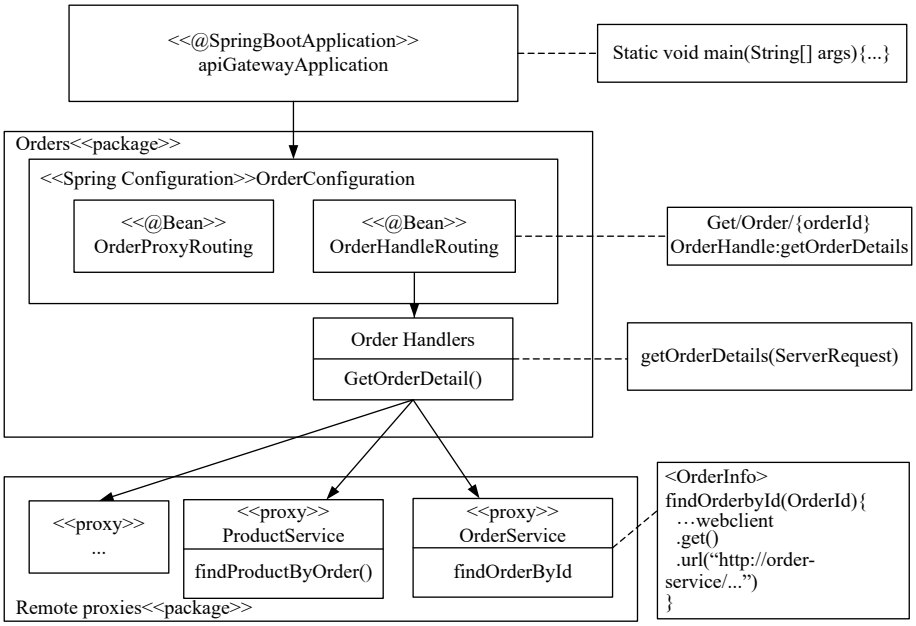


图 7 所示 API Gateway 架构

public class OrderHandlers {

private OrderService orderService;

private ProductService productService;

private CustomerService customerService;

public OrderHandlers(OrderService orderService,

ProductService productService,

CustomerService customerService) {

this.orderService=orderService;

this.productService=productService;

this.customerService=customerService; }

```
Public Moo<ServerResponse>
getOrderDetails(ServerRequest serverRequest) {
String orderid=serverRequest.pathVarable("orderId");
Mono<Orderinfo>orderinfo=orderService.findOrder
Byid(orderid);
Mono<Optional<Productinfo>>productinfo=
productService.findProdyctByOrderid(orderid);
…
Mono<Tuple4<0rderinfo,productinfo,custoinfo>>
combined=Mono.when(orderinfo,productinfo,cusinfo);
Mono<OrderDetails> orderDetails =
combined.map(OrderDetails :makeOrderDetails);
…
}
```

getOrderDetails() 实现 API 组合, 以获取订单详细信息, 它并行调用 3 个服务, 并将结果组合在一起, 创建一个 OrderDetails 对象转换为 ServerResponse.

OrderService 通过 WebClient 调用 OrderService 远程代理, 将 JSON 响应反序列为 OrderInfo 对象.

```
@Service
public class OrderService {
private OrderIntent orderIntent;
private WebClient client;
Public OrderService(OrderIntent orderIntent,
WebClient client)
{ this.orderIntent=orderIntent;
this.client=client; }
public Mono<OrderInfo>findOrderById(String
orderId){
Mono<ClientResponse>result=client.get()
.uri(orderIntent.orderServiceUrl+"/orders/{order}",
orderId).exchange();
Return result.flatMap(res->res.bodyToMono(OrderInfo
class)).block();}
}
```

### 3.4 微服务调用

微服务之间调用采用 Feign 方式, 这里以订单服务与库存服务交互为例, 定义好 Feign 的起步依赖, 在 application.yml 中配置 eureka-feign-client、端口号、服务注册地址等, 再在启动类中加上注解等. 接下来创建接口使用@FeignClient 注解, 其中 value 表示要远程调用的其他服务名称, 接口中方法 QueryStock 通过 Fegin 来调用 eureka-client 服务中的接口[11–13], 然后在订单服务层 OrderService 中注入 EurekaClientFeign 接口实例, 再创建一个控制器注入服务接口实例, 调用查询库存方法, 就可以实现远程调用 Feign 客户端的服务. 服务间调用代码如下:

```
@FeignClient (value="eureka-client", configuration
= FeignConfig.class)
public interface EurekaClientFeign
{
@GetMapping(value="/ QueryStock")
String QueryStockFromClientEureka
(@RequestParam(value="name") String name) ;
}
```

控制层与服务层代码类似已省略.

## 4 系统测试与运行

### 4.1 接口测试

Swagger 规范用于生成描述文件和接口文档, Spring Boot 使用 swagger 构建 RESTful APIs, 首先在业务服务的 pom 文件中添加 Springfox、Swagger UI 依赖后, 再在控制层和启动类添加注解、层中增加方法及参数, 最后在 UI 配置类中配置接口, 代码如下:

```
@Controller
@RequestMapping("/product")
Public class OrderController
{
@Autowired
private OrderService orderService;
@ApiOperation(value="根据订单 ID 取产品信息")
@RequestMapping(value="/info/{orderId}", method=
RequestMethod.GET, charset="utf-8")
@ResponseBody
Public Product getproductInfo(@ApiParam(name=
"orderId", value="订单 ID") @PathVariable
Long orderId) throws Exception {
Return orderService.getProductInfo(orderId);
}}
```

### 4.2 功能测试

本文以销售微服务为例, 对其订单管理部分功能进行测试, 订单创建界面如图 8 所示, 选择客户、订单日期和商品后即可创建成功.

图 8 订单创建

订单查询界面如图 9 所示, 选择起止日期, 按订单号或客户号输入关键字即可查询订单.

图 9 订单查询

### 4.3 性能测试

此次测试对基于微服务架构的 ERP 重构前后的系统分别使用 Jmeter 工具在相同的并发量下测试其性能. 将模拟并发量提高到 1000, 持续时间 30 s 条件下, 分别对产品信息返回结果的响应时间如图 10 所示.

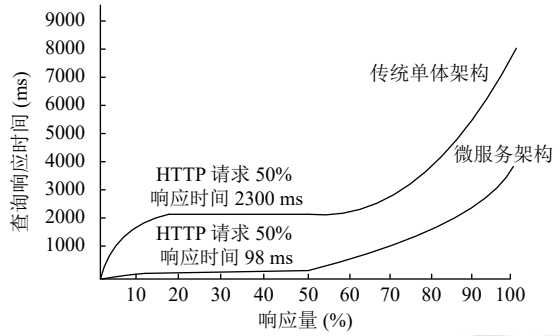


图 10 系统响应时间百分比

图 10 可以看出 ERP 系统在微服务重构前后, 当请求响应量达到 50% 时, 采用单体架构和微服务架构的 响应时间分别为 2.3 s 和 0.098 s, 重构后的系统在高并发情况下平均响应时间更短.

## 5 结语

本文首先分析传统单体架构的缺陷, 提出采用新型的微服务架构的特点和优势. 基于微服务的架构设计因业务模块具有各自的数据库、实体、服务、API 组件等, 可独立或者单独部署多个微服务, 具有灵活、可扩展、去中心化、敏捷性、自治等优势[14]. 本文提出基于微服务架构来构建企业 ERP, 分析微服务的分解模式, 设计了基于微服务的分层架构, 采用基于开源的微服务实现框架 Spring Boot 开发服务、Spring Cloud 来管理服务, 使用 Redis 做分布式数据缓存. 虽然微服务相关技术不断发展创新, 微服务之间如何准确通信, 以满足用户快速响应的需求, 微服务的部署、测试、跨服务实现问题, 如何在通信中保证数据的安全性, 采用什么样身份认证策略、数据加密方法都值得我们未来去探索[15].

## 参考文献


1 章仕锋, 潘善亮. 基于微服务架构的国土档案系统. 计算机系统应用, 2019, 28(7): 44–45. [doi: 10.15888/j.cnki.csa.006991]

2 周文坤, 乔运华, 侯佳佳, 等. 微服务架构的 ERP 应用系统的优势及挑战. 制造业自动化, 2020, 42(6): 123–124, 132. [doi: 10.3969/j.issn.1009-0134.2020.06.029]

3 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.

4 刘旺森. 基于微服务架构的遗留系统重构研究与实践 [ 硕士学位论文 ]. 呼和浩特: 内蒙古大学, 2019.

5 吴化尧, 邓文俊. 面向微服务软件开发方法研究进展. 计算机研究与发展, 2020, 57(3): 525–541. [doi: 10.7544/issn1000-1239.2020.20190624]

6 黄文毅. 分布式微服务架构: 原理与实战. 北京: 清华大学出版社, 2019.

7 吴晓龙. 基于微服务架构的在线学习系统设计与实现 [ 硕士学位论文 ]. 济南: 山东师范大学, 2019.

8 方志朋. 深入理解 Spring Cloud 与微服务构建. 北京: 人民邮电出版社, 2018.

9 翟永超. Spring Cloud 微服务实战. 北京: 电子工业出版社, 2017.

10 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. [doi: 10.15888/j.cnki.csa.006843]

11 于曼, 黄凯, 张翔. 基于微服务架构的 ETC 系统设计. 计算机科学, 2020, 47(S1): 643–647. [doi: 10.11896/jsjkx.190800010]

12 欧阳宏基, 杨铎. 基于微服务架构的学位论文写作辅助平台. 计算机与现代化, 2019, (10): 34–39. [doi: 10.3969/j.issn.1006-2475.2019.10.007]

13 张杰, 司维超, 王丽娜, 等. 一种面向微服务的通用考核系统设计与应用. 计算机与数字工程, 2018, 46(12): 2463–2467, 2533. [doi: 10.3969/j.issn.1672-9722.2018.12.017]

14 闻雷. 基于微服务架构的高校应用集成方案的设计与实现 [ 硕士学位论文 ]. 镇江: 江苏大学, 2018.

15 冯志勇, 徐砚伟, 薛霄, 等. 微服务技术发展的现状与展望. 计算机研究与发展, 2020, 57(5): 1103–1122. [doi: 10.7544/issn1000-1239.2020.20190460]