计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2019,28(7):44−50 [doi: 10.15888/j.cnki.csa.006991]

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

基于微服务架构的国土档案系统①

章仕锋, 潘善亮

(宁波大学 信息科学与工程学院, 宁波 315211)
通讯作者: 潘善亮, E-mail: panshanliang@nbu.edu.cn

摘　要: 针对传统 SOA 架构设计的国土档案系统面临的服务安全、负载均衡和伸缩性等问题, 设计实现了基于微服务架构的分布式档案系统. 根据微服务架构思想, 将档案系统功能划分为细粒度的微服务组件, 微服务之间设计鉴权服务模块来实现微服务的安全访问控制, 通过服务注册中心、服务网关和 SpringCloud 体系框架来解决系统的软负载问题, 利用 Docker 微服务集群完成微服务组件的独立部署运行和业务功能伸缩. 对档案数据文件建立了倒排索引, 提高了档案数据查询时的速度和准确度.

关键词: SOA 架构; 档案系统; 微服务架构; docker; 倒排索引

引用格式: 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):44–50. http://www.c-s-a.org.cn/1003-3254/6991.html

Land Archives System Based on Micro Service Architecture

ZHANG Shi-Feng, PAN Shan-Liang

(Faculty of Electrical Engineering and Computer Science, Ningbo University, Ningbo 315211, China)

Abstract: Aiming at the problems of service security, load balancing and scalability faced by the land archives system designed by traditional SOA architecture, a distributed archival system based on microservice architecture is designed and implemented. According to the idea of micro-service architecture, the function of archives system is divided into fine-grained micro-service components, and the Authentication Service module between microservices is designed to realize the secure access control of microservices, and the soft load problem of the system is solved by service registry, service Gateway and SpringCloud system framework. The Docker microservice cluster is used to complete the independent deployment run and business function scaling of the microservice components. The inverted index of file data file is established, which improves the speed and accuracy of file data query.

Key words: Service-Oriented Architecture (SOA); archive system; microservice architecture; docker; inverted index

随着软件体系架构的发展, 选择合适的软件架构成为软件开发过程中的关键要点, 国土档案系统是不动产登记信息化工作中的重要工具, 传统的国土档案系统, 一般采用单体架构设计开发, 单体架构将整个系统功能和数据作为一个整体, 统一地进行设计、开发和部署, 单体架构设计的档案系统有着明显的缺陷, 首先, 单体架构设计的档案系统内部紧密耦合, 难以根据业务需求变化进行灵活调整, 其次, 单体架构设计的系统所有部分采用同一技术路线, 无法与其他信息系统之间进行协同与共享从而产生“信息孤岛”现象.

针对单体架构存在的问题, 面向服务的架构 (Service-Oriented Architecture, SOA)[1]随之出现, 不同于单体架构, SOA 架构将业务服务作为整个系统设计的核心, 设计时可以根据具体的业务需求为每个服务

① 基金项目: 浙江省公益性技术应用研究计划 (2017C33001)
Foundation item: Technology Application Plan for Public Welfare of Zhejiang Province (2017C33001)
收稿时间: 2019-01-13; 修改时间: 2019-02-19; 采用时间: 2019-02-26; csa 在线出版时间: 2019-07-01

选择特定的技术路线, 通过 Web 服务统一标准协议, 将定义封装好的服务接口进行发布, 各个服务接口组件之间通过轻量级通信协议进行交互, 从而实现一个灵活互通松耦合的信息系统.

虽然 SOA 架构实现国土档案系统跨平台开发及松耦合度, 但随着国土档案系统业务需求日趋多样化和复杂化, SOA 架构设计的系统局限性开始显现, 主要面临的问题如下:

(1) 安全性不足: 国土档案数据作为国土业务工作的核心, 数据安全和保密性要求较高, SOA 架构在异构平台下进行 Web 服务的调用, 由于异构环境下各平台技术安全标准的不统一, 缺乏调用时应有加密保护措施来进行对调用者的权限验证, 导致服务可能被恶意调用, 系统数据的安全性不足.

(2) 负载均衡问题: SOA 架构设计的档案系统将一些业务功能做成服务的形式发布, 最大化的提高服务的重用性, 然而业务量的拓展和使用人数的增加将导致对特定服务负载压力过大, 当某一服务发生错误就会导致整个系统受到影响, 系统无法根据当前状态进行负载上的调整, 系统稳定性将会下降.

(3) 伸缩拓展性不足: 在国土业务部门大数据、云平台的功能需求背景下, 应用系统对计算机资源的使用日益灵活, 业务高峰时需要增加应用部署、分配存储资源等操作来满足使用需求, 业务低谷时则应对相应的资源进行回收再利用, 传统 SOA 档案系统紧密耦合难以根据需要进行灵活的伸缩, 运维部署难度大.

本文提出一种基于微服务架构的分布式档案系统设计方案, 通过微服务架构解决了传统 SOA 架构中存在的服务易崩溃、负载不均衡和服务调用不安全等问题, 引入 Docker 容器级虚拟化技术, 提高了系统运维部署的可伸缩性. 微服务架构化后的国土档案系统, 显著提升了系统的可靠性、稳定性, 满足了国土档案系统在现今和未来海量业务数据功能的需求.

1 总体架构

系统整体架构如图 1 所示, 系统架构具体可分为数据层、索引层、Docker 微服务群、业务层和网关层, 系统各个部分按照电子政务统一平台建设标准规范进行设计开发, 作为电子政务云平台组成部分可以与其他政务系统进行衔接和共享.

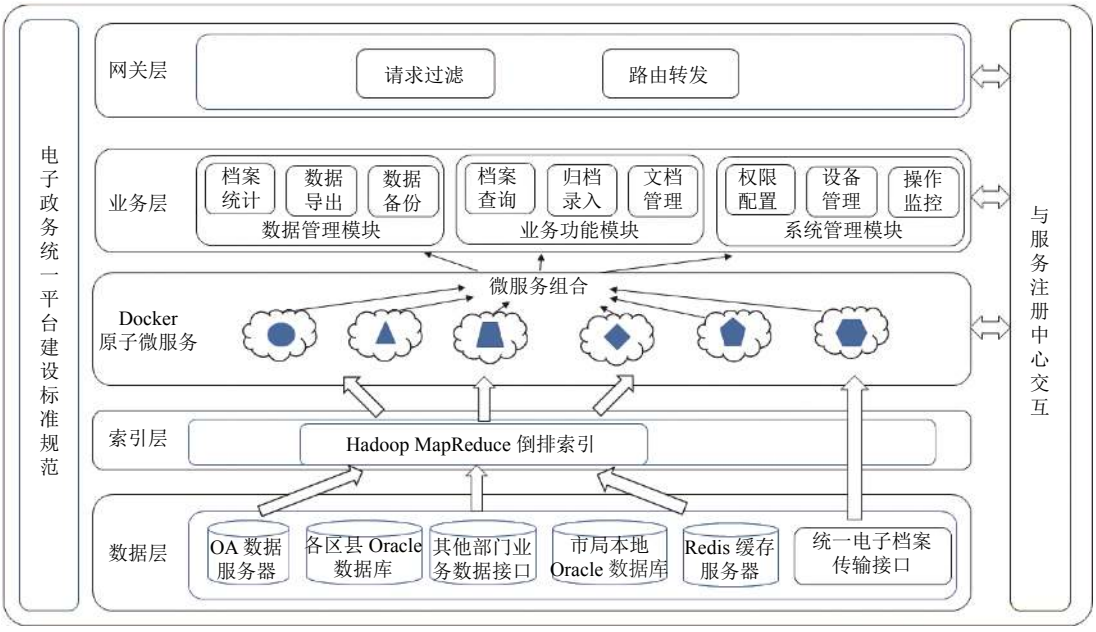

图 1 系统总体架构图

数据层将各个区县的数据库通过国土局专用内网连接整合, 同时结合市局 OA 系统, 对上层提供统一数据接口, 采用 Redis 数据库保存缓存数据, 根据需要利用缓存来直接对上提供数据来提高查询效率. 索引层是对接收的数据信息建立倒排索引机制, 上层服务进行关键字查询时, 利用倒排索引提供的关键字文档, 能

够快速找到相关的文件信息并定位到文件的存储位置, 最大化的提高查询速率和准确性. Docker 原子微服务将业务功能划分为细粒度的微小服务, 每个 Docker 容器承载一个原子微服务, 利用集群部署的方式来提高系统可用性, 每个 Docker 服务都会在服务注册中心进行注册, 通过心跳包的方式维持与服务注册中心的联系. 业务层将 Docker 原子微服务提供的服务进行组合封装, 形成具体业务功能的粗粒度服务, 这些粗粒度的服务同样利用 Docker 容器进行部署运行, 提供相应的服务接口进行服务调用, 服务注册中心实时监控各个接口状态信息, 根据服务调用情况进行负载限流调整. 服务网关控制前后端路由转发, 通过对请求进行权限验证来决定是否转发, 丢弃验证不通过的数据请求.

2 系统设计

2.1 数据层设计

档案系统的核心数据主要包括档案文件条目及对应的电子文件, 地籍类档案数据来源各区县 Oracle 数据库和市局已有 Oracle 数据库, 市局中部署 OA 系统服务器和 Redis 缓存服务器, OA 系统保存并提供各类日常办公数据, 如会计、文书等, Redis 缓存服务器保存数据层被频繁访问的数据, 提高系统查询速率, 同时, 业务数据接口汇总采集相关业务部门数据, 集成如住建、测绘、海洋等数据, 数据层完成对数据的综合利用, 为上层形成利用具体的档案提供数据支持. 数据层动作主要分为数据采集和数据处理两个部分, 完成对数据的预处理和封装.

2.1.1 数据采集

数据采集是指将物理上分布于各个区县的异构来源的数据资源进行集成, 各区县原有档案文件条目是建立在不同技术路线的信息系统之上, 针对各区县数据库条目字段差异, 在市局数据库建立统一条目查询视图, 查询视图将分布式环境下各个数据库条目进行整合, 对上层提供统一的数据库查询接口.

对于档案电子文件, 建立统一电子档案传输接口服务, 各个区县档案业务部门进行业务归档后, 统一电子档案传输接口服务通过 ftp 协议将分散在各地的电子文档发送到市局服务器中保存.

2.1.2 数据处理

数据层完成数据采集后将对数据资源进行数据处理, 将对于档案数据条目进行数据清洗[2], 从各个区县采集到的数据条目存在着各种系统冗余和错误数据, 因此数据清洗时主要包括检查数据一致性[3]、处理档案条目中存在的无效和缺失记录, 处理后的档案数据条目为上层索引提供统一的资源利用视图, 从上层来看, 各数据不再是分布于各个不同物理平台之中, 而是存在于数据层封装好的虚拟视图之中.

从各区县采集的电子档案文件, 在存储格式上存在各种差异, 通过在线格式转换模块, 将不同格式来源的电子档案文件统一转换成 PDF 格式文档, 转换完成后的电子档案文件进行统一的备份保存.

2.2 索引层设计

索引层的作用在于提高系统对电子档案进行关键字查询时的速度和准确度, 索引层数据来自于数据层整理集成后的数据库档案条目, 每个档案条目对应各自的电子档案文件, 档案条目中包含电子档案信息的关键字条目, 对关键字条目建立数据文档作为索引数据来源, 索引层将数据层获得的数据利用 Hadoop MapReduce 技术建立倒排索引[4], 根据档案各个案卷文件中的关键字出现频率及位置, 能够快速为查询服务提供关键字匹配率由高到低的所有结果.

在 Hadoop 平台上利用 MapReduce 编程模型, 首先对档案中各案卷进行 Map 过程, 形成<key, value>形式键值对, 其中 key 表示案卷中包含的各个字段的信息以及案卷对应的 URL, value 表示案卷中各个关键词出现的频率, 然后进行 Combine 过程, 将 Map 过程输出结果作为自己的输入, 把同一案卷中的同一关键字作累加统计运算, 得到各个关键词出现的案卷位置及频率结果, 最后进行 Reduce 过程, 把 Combine 过程输出的结果作为输入, 形成以关键词对应的案卷位置和在关键词在各个案卷中出现的频率为格式的倒排索引文档, 档案系统在进行关键字查询时, 查询服务优先查询倒排索引文档记录, 关键字匹配时直接返回查询结果来提高查询速率. 案卷索引建立过程如图 2 所示.

2.3 Docker 原子微服务设计

2.3.1 Docker 容器设计

Docker 容器池主要由 Docker 镜像仓库、基础资源设施、容器调度编排、容器管理运维组成, 大量 Docker 容器部署在相同或者不同宿主服务器中[5], 各个容器之间相互独立, 通过轻量级通信机制进行沟通, 档案系统功能被划分为细粒度的原子功能, 每个原子功能服务部署在 Docker 容器中, 其中重要服务采用多个容器部署的方式来最大化的提高服务可用性.

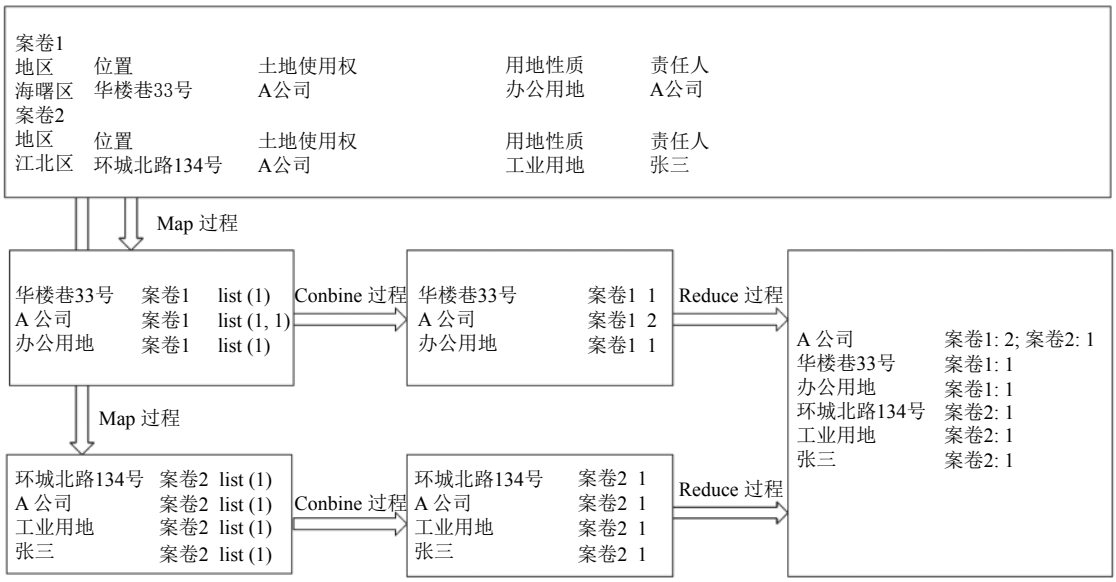

图 2 案卷索引建立过程

(1)Docker 镜像仓库

镜像仓库主要用于存储和扩展业务镜像, 通过从 Docker 服务器端下载初始系统镜像, 然后将微服务程序打包移植到系统镜像上, 形成一个新的独立业务镜像. 将镜像部署到 Docker 容器之中, 容器之间可以通过轻量级的网络协议进行交互.

(2) 基础资源设施

基础资源设施提供微服务程序所需的各种计算机资源, 包括计算机存储、通信网络、服务器资源等.

(3) 容器调度编排

容器调度编排是将部署了微服务应用的 Docker 容器通过 Kubernetes 集群[6]进行调度, Kubernetes 集群主要分为 Master 节点和 Node 节点, Master 节点对外提供控制管理服务, 节点包括管理集群的 API 接口, Node 节点负责运行实际的 Docker 容器, 对节点中的 Docker 容器提供代理交互功能.

(4) 容器管理运维

管理运维模块中包含容器状态监控、容器网络配置、容器版本控制和日志监控等部分, 通过自动化运维工具为容器用户提供容器资源监管控制等功能.

2.3.2 微服务集群

微服务集群如图 3 所示, 主要分为微服务注册中心、分布式配置中心、原子微服务群、微服务状态监控和错误熔断机制等部分, 采用 Spring Cloud 微服务框架体系[7]来设计实现各个组成部分.

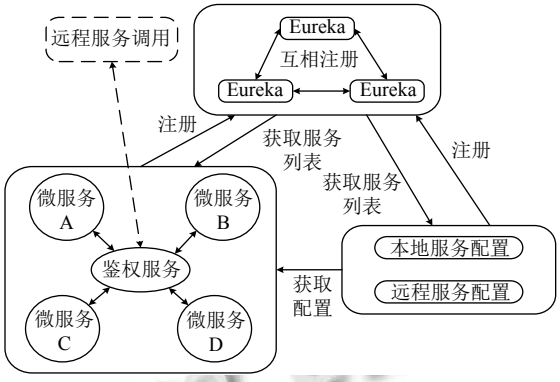

图 3 微服务集群

微服务注册中心负责所有微服务组件的注册和调用, 提供集群负载和微服务可用性的控制, Eureka 高可用的组件来具体实现微服务注册中心的功能, 根据去中心化的思想, 采用多个 Eureka 组件互相注册的方式, 防止单一组件发生崩溃导致整体无法运行, 每个微服务在服务注册中心进行注册, 注册的微服务通过心跳包的方式维持与服务注册中心之间联系.

原子微服务群主要实现微服务粒度和划分和独立部署, 将档案系统功能划分为基本的原子功能, 利用 Docker 容器进行原子服务的部署, 部署后的原子微服务群为业务层的业务组合提供相应的接口.

微服务状态监控包含对单个微服务可用状态和微服务群接口状态及使用频率的监控, 利用 Spring Boot Admin 来实现单个服务可用状态的监控, Hystrix Dashboar 来完成接口状态及使用频率的监控.

分布式配置中心用于统一管理各个微服务配置设置, 能够通过本地或者远程的方式, 对微服务配置进行实时更新. 服务错误熔断机制是指当某个微服务调用发生错误后对其进行隔离, 防止调用出错而造成的网络阻塞, 设计断路器模型利用 Hystrix 组件[8], 对特定微服务的调用不可用达到阈值后, 打开断路器对服务调用返回一个固定值避免连锁反应, 同时利用 Hystrix 组件提供可视化界面来进行数据监控.

2.4 业务层设计

业务层的作用是对整个系统的业务功能进行组合发布, 根据微服务架构思想, 将档案系统的数据管理、业务功能、系统管理三个主要功能模块拆分成细粒度的微服务组件, 对外提供统一的微服务接口, 根据不同接口的负载情况, 对于负载压力大的接口采用集群的方式来发布.

业务层设计的核心思想在于其弱耦合度, 业务层在整个系统架构中有着承上启下的作用, 对下层 Dokcer 原子微服务提供的服务进行组合, 为上层服务网关提供业务接口, 通过对接口的封装, 上层对于下层来说是不可见的, 改变业务层的设计对 Docker 原子微服务的调用不会产生影响.

组合服务时, 服务调用者向服务网关发出具体业务功能接口请求, 服务网关根据微服务注册中心反馈的服务列表向业务层发出微服务组合请求, 业务层根据负载情况将微服务注册中心反馈的可用原子服务集群列表中选择相应的 Docker 服务, 通过对服务集群提供的原子服务进行组合, 封装成档案系统所需的粗粒度业务服务, 业务服务以 Docker 容器的方式部署, 组合后的业务服务将服务接口反馈给服务网关, 服务网关向服务调用者提供调用接口.

业务层还包括负载均衡模块和消息队列模块, 负载均衡模块在服务调用组合过程中, 通过判断比较程序, 设定阈值, 对超出阈值的调用请求选择集群中负载较低的服务处理或者弃用请求, 消息队列模块将一些无需即时返回且耗时的操作提取出来, 进行了异步处理[9].

2.5 服务网关设计

服务网关的主要作用是对前端路由请求进行转发和请求处理过滤, 流程如图 4 所示, 利用 Node.js 技术[10]来实现服务网关功能, 采用 Nginx 作为系统 Web 服务器[11].

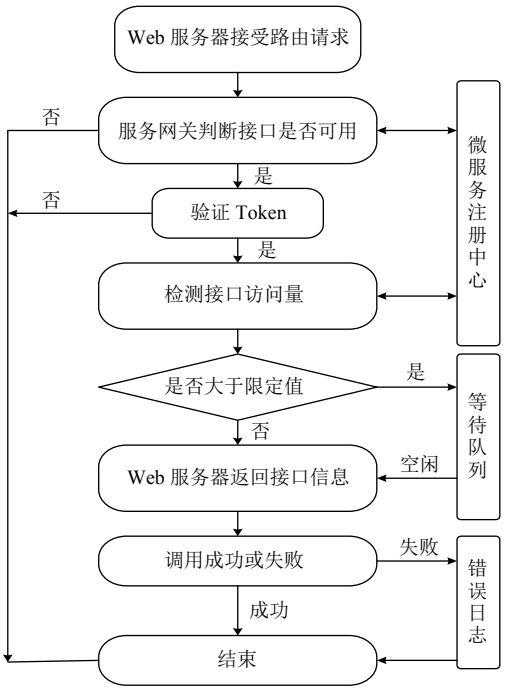

图 4 服务网关流程图

2.5.1 路由转发

服务网关负责接收外界对系统的一切请求, 并将请求转发到业务层上去, 在整个过程中, 服务网关首先从服务注册中心获取业务层接口状态信息, 判断业务层接口是否存在及是否可用, 当业务接口可用时获取接口信息, 通过 Web 服务器执行反向代理过程之后[12], 将业务接口提供给请求者.

2.5.2 请求处理过滤

服务网关作为整个系统与外界的门户, 负责外界对系统请求处理过程进行过滤, 主要实现了权限校验、业务接口监控、访问限流和日志收集等功能, 当外界首次访问时, 进行权限校验, 利用 JWT 技术通过首次登陆时返回一个加密请求 token, 调用服务时进行验证通过后才允许调用, 从系统前端与后台交互时进行限流, 当服务网关检测到前端对业务层接口访问量过大时, 将请求转入等待队列, 待接口空闲后再进行访问. 同时, 监控业务层功能接口状态, 当检测到接口错误时, 将信息反馈到服务注册中心, 形成错误日志保存错误信息.

3 关键模块设计

3.1 鉴权服务模块

微服务集群中设计鉴权服务来保证服务调用的安全性, 鉴权服务就是通过 JWT 技术 (Json Web Token, JWT)[13]

来实现服务安全调用的单独服务模块, JWT 是一种无状态的安全验证机制, JWT 通过自包含的特性让用户无需在服务器内存中存储用户的状态, 减少了查询数据库来验证用户的次数.

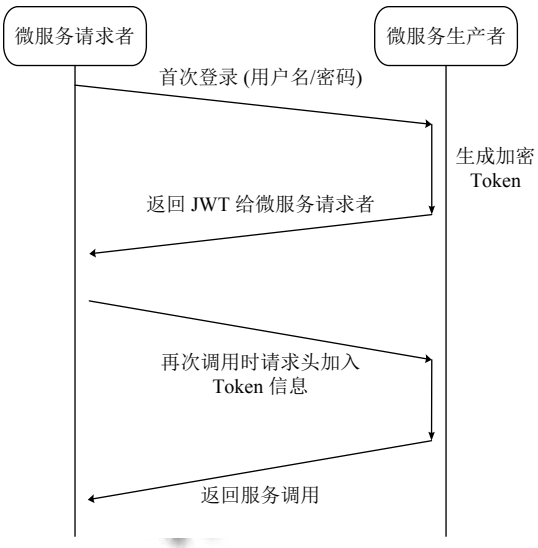

图 5 JWT 验证过程

如图 5 所示, 鉴权服务每次服务调用时验证 Token 来判断调用是否合法. JWT 根据微服务请求者首次登录信息生成相应的加密 Token, 向微服务请求者返回生成的 Token, 服务器不保存任何用户信息, 微服务再次调用时, 在请求头中加入 Token 信息, 鉴权服务验证请求头中 Token 的正确性, 验证通过后返回服务调用给请求者.

3.2 负载均衡模块

负载均衡模块主要通过 Spring Cloud 微服务体系框架中 Zuul 和 Ribbon 框架[14], 来实现对服务网关及微服务客户端软负载, Ribbon 框架内置多种负载均衡规则, 主要包括轮询、服务器忽略、权值判断和随机选择等方式, Zuul 服务网关主要通过限制客户端请求速率的方式来避免大量服务调用使系统阻塞, 利用服务注册中心配合服务网关 Zuul 和 Ribbon 框架能够较好满足系统负载均衡性能需求.

如图 6 所示关键业务功能对应的原子微服务通过在多个 Docker 容器中部署的方式来提高服务的可用性, Ribbon 客户端通过轮询的方式从服务注册中心获取可用服务列表, 选择负载较低的微服务组件来进行组合, Zuul 网关对客户端的请求进行转发和过滤, 根据服务调用情况进行流量上的控制.

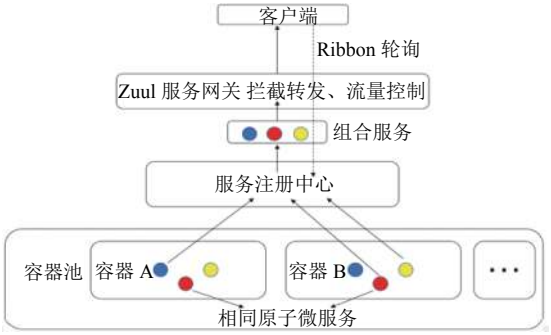

图 6 通过 Zuul、Ribbon 来完成负载流量控制

3.3 容器伸缩模块

容器资源池中设计容器伸缩模块来满足系统应对业务高峰低谷时的需求, 通过响应式混合伸缩对业务需求已使用的容器数进行检测, 建立伸缩策略, 当检测到数值超过或低于设定的阈值时, 向系统运维发送提醒, 运维人员可以根据设定的步长进行手动或自动的增加或减少容器数量, 同时向 Kubernetes 集群发出指令, 调整相应的节点数量.

响应式混合伸缩过程策略:

(1) 根据业务运行性能情况为系统设定伸缩调整检测阈值范围, 设置伸缩步长.

(2) 垂直伸缩阶段: 当检测到的业务负载偏离阈值范围时, 对容器进行垂直伸缩, 调整分配给相应容器的资源配额, 增加或减少内存、CPU 核数等资源.

(3) 水平伸缩阶段: 水平伸缩阶段主要利用 Kubernetes 集群 Master 节点来完成对节点数量的控制, 用户通过向 Master 节点发送相应的 API 指令, 增加或减少容器实例数来满足系统的性能需求.

4 系统运行效果测试

4.1 查询响应时间

查询响应时间是国土档案系统的重要性能指标, 完成系统部署后, 对国土局原 SOA 架构档案系统和微服务架构档案系统进行实验, 比较系统间在不同档案条目数量下, 查询响应时间的变化情况.

实验选取国土局虚拟档案室原档案系统 (SOA) 和微服务架构设计的档案系统进行比较, 数据库选择统一的 Oracle10g 版本, 服务器硬件配置为 16 GB 内存, 4 核 i7 处理器, 服务器运行环境为 Windows Server 2008.

测试实验结果图 7 表明, 在档案条目数量较少时, 系统之间查询效率相差不大, 但随着档案条目数量不断增加, 微服务架构设计的档案系统查询响应时间增

量要明显低于原 SOA 架构设计的档案系统, 显示了微服务档案系统查询效率要高于原 SOA 系统, 从结果说明, 相对于直接利用 Oracle 数据库查询, 微服务系统架构设计中的倒排索引及 Redis 缓存能够较有效的提高系统关键字查询时的速率.

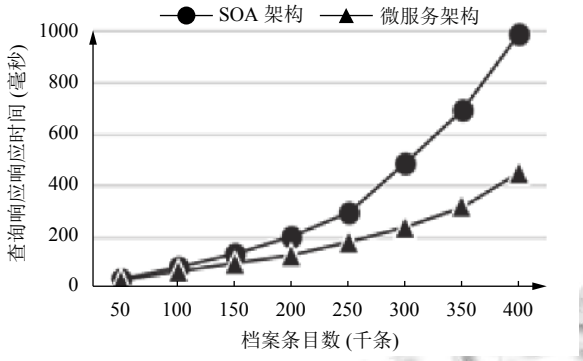

图 7 查询响应时间比较

4.2 系统界面

本系统设计满足了国土档案业务部门基本的需求, 主要是全市档案的查询、显示、各类档案的归档、后台权限管理等业务需求, 系统运行效果如图 8 所示, 当前系统运行良好, 不仅可以在国土档案业务部门使用, 也可满足其他业务部门档案管理需求, 较好的解决了档案数据的集成问题.

5 结论与展望

随着信息系统规模和复杂度的不断提升, 微服务架构成了越来越多应用系统架构的首选, 许多原有系统也纷纷进行微服务化改造. 本文介绍了微服务架构的优势和特点, 将 Docker 技术与微服务架构结合起来, 设计实现了一套基于微服务架构的分布式档案系统, 相较传统 SOA 架构开发的系统, 主要解决了传统系统中面临的安全性低、负载不均衡和伸缩性不足等问题, 新系统的稳定性、易拓展性和可维护性等方面大大加强, 有效降低了原先系统在运行部署和后期维护上的复杂度, 未来将进一步在系统查询效率优化、系统流畅性等方面进行改进.

图 8 系统查询界面

参考文献

1 赵莉娟. 基于 SOA 架构的高校数字档案信息资源整合的研究与设计[硕士学位论文]. 武汉: 武汉理工大学, 2012.
2 封富君, 姚俊萍, 李新社, 等. 大数据环境下的数据清洗框架研究. 软件, 2017, 38(12): 193–196. [doi: 10.3969/j.issn.1003-6970.2017.12.037]
3 魏恒峰. 分布数据一致性技术研究[博士学位论文]. 南京: 南京大学, 2016.
4 谌超, 强保华, 石龙. 基于 Hadoop MapReduce 的大规模数据索引构建与集群性能分析. 桂林电子科技大学学报, 2012, 32(4): 307–312. [doi: 10.3969/j.issn.1673-808X.2012.04.012]
5 肖伟民, 邓浩江, 胡琳琳, 等. 基于容器的 Web 运行环境轻量级虚拟化方法. 计算机应用研究, 2018, 35(6): 1768–1772. [doi: 10.3969/j.issn.1001-3695.2018.06.037]
6 王骏翔, 郭磊. 基于 Kubernetes 和 Docker 技术的企业级容器云平台解决方案. 上海船舶运输科学研究所学报, 2018, 41(3): 51–57. [doi: 10.3969/j.issn.1674-5949.2018.03.009]
7 张树新, 吴海斌, 蒙辉, 等. 基于 SpringCloud 的航运 EDI 平台 IT 生态环境设计. 中国储运, 2018, (2): 100–103. [doi: 10.3969/j.issn.1005-0434.2018.02.033]
8 翟永超. Spring Cloud 微服务实战. 北京: 电子工业出版社, 2017. 130-196.
9 裴鹏飞. 支持事务的分布式消息队列中间件的设计与实现[硕士学位论文]. 济南: 山东大学, 2018.
10 王壮. 基于 Nodejs 框架的企业人事管理系统的设计与实现[硕士学位论文]. 长春: 吉林大学, 2017.
11 张尧. 基于 Nginx 高并发 Web 服务器的改进与实现[硕士学位论文]. 长春: 吉林大学, 2016.
12 冯贵兰, 李正楠. Nginx 反向代理在高校网站系统中的应用研究. 网络安全技术与应用, 2017, (6): 111, 120.
13 范展源, 罗福强. JWT 认证技术及其在 WEB 中的应用. 数字技术与应用, 2016, 34(2): 114.
14 毛可嘉. 客户端负载均衡算法研究及在即时通讯架构中的应用与实现[硕士学位论文]. 成都: 电子科技大学, 2018.