计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2020,29(2):233−237 [doi: 10.15888/j.cnki.csa.007285]

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

基于微服务架构 B/S 系统的性能分析①

陈建海[1], 陈　淼[1,2], 浦云明[1]

[1](集美大学 计算机工程学院, 厦门 361021)
[2](北京邮电大学 网络技术研究院, 北京 100876)
通讯作者: 浦云明, E-mail: yunmingpu@163.com

摘　要: 传统的单体应用架构系统, 随着用户需求和系统功能的变动, 出现了单体应用功能模块边界模糊、部署效率低、扩展困难、技术更迭代价高等缺点, 尤其是单个模块修改部署效率低的问题. 因此, 微服务技术得到关注和应用, 微服务架构的业务边界确定服务边界, 具有高内聚性, 易于开发与维护、局部修改部署、技术选择不受限等优势. 本文研究微服务应用系统的架构优势, 设计了一 B/S 应用系统进行测试分析. 实验设计测试指标为线程响应时间、吞吐量以及部署时间的实验方案, 并使用 Jmeter 性能测试工具进行测试, 分析了 20 个和 50 个并发用户的测试数据. 实验结果表明微服务在响应时间、吞吐量等指标有明显的效率和性能优势.

关键词: 微服务; 系统性能; 软件架构; Spring Cloud

引用格式: 陈建海,陈淼,浦云明.基于微服务架构 B/S 系统的性能分析.计算机系统应用,2020,29(2):233–237. http://www.c-s-a.org.cn/1003-3254/7285.html

B/S System Performance Analysis Based on Microservices Architecture

CHEN Jian-Hai[1], CHEN Miao[1,2], PU Yun-Ming[1]

[1](Computer Engineering College, Jimei University, Xiamen 361021, China)
[2](Institute of Network technology, Beijing University of Posts and Telecommunications, Beijing 100876, China)

Abstract: Existing systems are generally based on traditional single-application architectures. With the continuous improvement of user requirements and the continuous improvement of website functions, there are shortcomings such as monotonous application function module obfuscation, low deployment efficiency, difficulty in extension functions, and high cost of technology and iterative methods. Therefore, the microservices technology gets attentions and adopted in project development. The microservices architecture has the advantages of easy development and maintenance, partial modification and deployment, easy extension functions, and unlimited technical options. In this work, the advantages of microservices application was studied, and setup an B/S system for performance testing. An experimental scheme for thread response time, throughput, and deployment time was designed and tested using Jmeter testing tools. The test data of 20 and 50 concurrent users were analyzed. Experimental results show that microservices has greater performance advantages and higher efficiency.

Key words: microservices; system performance; software architecture; Spring Cloud

常用的 B/S 系统开发, 一般是基于单体应用架构, 例如 Java 技术开发的 B/S 系统, 一般选用 SSH (Struct2、Spring、Hibernate) 或者 SSM (SpringMVC、Spring、Mybatis) 框架, 开发出一个 war 包后将其部署到 Tomcat

① 基金项目: 福建省科技计划 (2019H0021); 厦门市科技计划 (3502Z20173028, 3502Z20173027)
Foundation item: Science and Technology Program of Fujian Province (2019H0021); Science and Technology Program of Xiamen Municipality (3502Z20173028, 3502Z20173027)
收稿时间: 2019-07-18; 修改时间: 2019-08-22; 采用时间: 2019-08-27; csa 在线出版时间: 2020-01-16

中发布. 单体应用架构的开发、部署、测试较为容易, 但随着需求的不断增加, 每一次系统的更新, 都需要将 war 包重新部署, 并且 war 包如同滚雪球一般越滚越大, 系统的可维护性、可靠性、灵活性逐渐降低, 维护成本越来越高, 任何一个 bug 都会导致系统崩溃. 随着时间的推移, 整个程序的代码量变得越来越大, 使得已有的系统设计和代码变得难以维护, 系统的构建和部署时间也不断增加. 单体应用中每次功能变更、bug 修复都会导致整个项目需要进行重新部署, 增加了项目部署时间、成本和风险.

为解决单体应用的缺陷, Fowler M 提出了微服务架构[1], 它完全不同于单体应用架构, 将应用程序逻辑拆分、设计、开发为一组小型服务, 这些小型服务只关注自身所负责的功能, 不关心其他服务及其内部实现. 这些服务可以独立部署在平台即服务 PaaS (Platform as a Service) 上, 或者运行在自己的进程中, 进程与进程之间相互隔离, 降低各服务的耦合性, 服务间通信采用轻量级通信机制 REST 风格, REST 是资源表现层状态转换 (REpresentational State Transfer), 所有独立的服务构成了一个完整的软件系统. 这些服务由于部署在各自独立的进程中, 各服务间内聚性大, 耦合性小, 可以采用不同的编程语言、不同的数据存储技术实现系统功能. 微服务架构由于将服务分割专注化, 因此不会像传统的单体应用程序一样, 修改一个 bug 或增加一个功能就要重新进行部署, 只需要将修改的服务重新部署, 不会影响其他服务的运行.

本文的测试对象是基于微服务架构的 B/S 应用系统, 后端使用 Spring Cloud 技术, 前端使用 JQuery、Bootstrap 以及 Thymeleaf 模板, 数据库使用 MySQL, 并采用非关系内存数据库 Redis 进行 Session 模拟和部分基础功能的实现, 使用 Intellij IDEA 集成开发环境, 由于 Spring Cloud 内部集成了 Tomcat, 所以只需要运行启动类, 通过相应地址就可以访问相关服务. 系统使用了 Jmeter 测试工具, 对单体应用和微服务架构进行不同级别的测试, 性能指标上微服务系统架构体现出明显优势.

1 微服务技术

系统前端技术采用开源的 Bootstrap 和 JQuery 框架, 用户输入字符验证采用 JQuery validate 框架, Bootstrap 已经处于 github 上星级项目 (starred project) 前列, 利于技术人员编写用户体验良好的前端组件和动作[2].

后端采用 Spring Cloud 框架实现微服务的基本框架搭建, 数据连接与操作采用 Mybatis, 用户密码采用 Shiro 的 MD5 加密, 防止被非法人员侵入数据库后得到用户密码后进行非法活动. 系统后端与前端之间的数据交互采用 json 字符串格式, 方便前端解析后端传递的内容. 数据方面, 选用 MySQL 来存储用户以及系统文章、评论等的基本数据, 由于各微服务运行于各自隔离的进程中, 无法将 HTTP Session 交于统一的 Servlet 容器, 因此采用内存数据库 Redis 模拟实现 Session. 系统选用 Maven 进行 Java 的依赖包管理和项目的搭建, 并使用 Git 进行项目的版本控制.

1) Spring Cloud 是在 Java 快速开发框架 Spring Boot 基础上构建的一个开发框架. 它在 Spring Boot 便利性的基础上很好地降低了微服务系统实现的门槛, 如实现微服务的注册与发现, 实现负载均衡, 实现 REST 通信, 构建微服务网关等一系列功能, 都可以使用 Spring Cloud 通过最简单的配置或者几行编码就完成实现与部署[3].

2) Redis 是一个非关系数据库, 它可以存储键与其他五种不同类型的值之间的映射关系. 因为 Redis 数据库本身是基于内存存储的, 所以 redis 的处理与运行速度相比于传统的数据库快速高效. Redis 还可以通过简单的设置就将存储在内存的数据持久化到硬盘中, 使之下次读取的时候就可以直接从硬盘中获取数据. 因为 Redis 不使用关系表结构来进行数据的存储, 所以 Redis 的数据库不会强制要求用户对 Redis 存储不同的数据进行相应的关联[4]. 使用 Redis 使得用户要求进行数据持久化时, 才将这些数据存储在硬盘中, 从而提高整个系统代码的运行效率, 给用户提供更好的运行体验.

3) REST 是一种软件架构风格, 并不是一种软件设计的标准, REST 提供了一组设计原则和约束条件, 以寻求降低开发的复杂性, 提高系统的可伸缩性的目的[5].

4) Mybatis 封装了系统与数据库的连接、校验、操作实现等底层代码的实现, 使得用户可以使用 XML 配置或者 Mybatis 注解完成数据库的连接, 操作, 关闭数据连接池等基本操作[6]. 相比于 JDBC、Hibernate 操作数据库, Mybatis 代码更具易读性优势.

5) Git 是目前软件开发领域中最好的分布式版本控制工具. 是 Linux 之父为了帮助管理 Linux 内核开发

所制作的一个开源版本控制软件[7].

6) Maven 是一个项目管理工具. 开发团队可以通过 Maven 自动完成项目的基础工具建设, Maven 使用标准的目录结构和默认构建生命周期[8]. 基于 Maven 的 Java 项目中, 其项目的依赖包是统一管理的, 有效避免 Java 项目的依赖包因为版本原因而产生冲突.

7) Zuul 是微服务网关组件. 微服务网关是介于客户端和服务器端的中间层, 用户提交的所有外部请求都会先经过微服务网关的处理和过滤, 可以实现用户身份认证与安全、审查与监控、动态路由、压力测试、负载分配、静态响应处理等功能. 使用 Zuul 微服务网关后, 实际上封装了系统内部的所有服务, 用户只需要和微服务网关交互, 不必直接调用微服务的相关接口[9].

8) Eureka 是用于实现微服务架构中的服务注册与发现的组件. 服务提供者在服务启动时, 将自身以及 URL 等一些信息注册到注册组件中, 而服务注册组件会存储各个服务提供者的这些基本信息. 各个微服务与服务发现组件之间通过一定机制进行通信, 例如心跳机制, 即各个微服务每隔一定的时间向服务发现组件发送信息, 表示自己还在运行中, 可以被调用, 若持续一段时间未向服务发现组件提供信息, 则服务发现组件会认为该服务出现故障或者已被关闭, 则从注册表中注销该服务[10,11].

2 实验环境和应用场景

2.1 实验环境

1) 操作系统: Windows 10 企业版 64 位.

2) 数据库: MySQL 5.7; Redis.

3) 软件包: Java 8; Spring Cloud Camden.SR7; Spring Boot 1.5.3.RELEASE.

4) 测试包: Junit4, Jmeter.

5) 开发工具: Intellij IDEA; Maven 3.3.9; Git.

2.2 应用场景

测试系统分为前台和后台两部分, 前台为基本页面, 用户可以对前台进行查看和操作, 后台是系统管理页面, 系统管理员需要通过验证后进行相关操作.

前台页面完成以下场景: (1) 一般用户访问主页面, 根据类别或者文章题目进行文章查看; (2) 用户登录后, 可以查看个人信息, 并对部分个人信息进行修改;用户可以根据自己的需求发布文章; (3) 用户登录后, 可以对所有文章进行评论, 也可以对他人的评论进行回复, 但一旦回复均无法删除.

后台页面完成以下场景: (1) 对非法用户进行删除; (2) 对违法文章和之下的所有评论进行清理; (3) 增加文章类别和发布文章; (4) 将表现良好的用户赋予管理员头衔.

系统整体采用微服务架构, 如图 1 所示. 每一个服务采用 MVC架构并拥有自己独立数据源, 每个服务不需要其他服务的支持就可以独立运行. 同时这些服务都注册到 Eureka 组件中, 相互之间使用 REST 进行通信, 充分降低了各服务之间的耦合度, 增加了系统的内聚性[12].

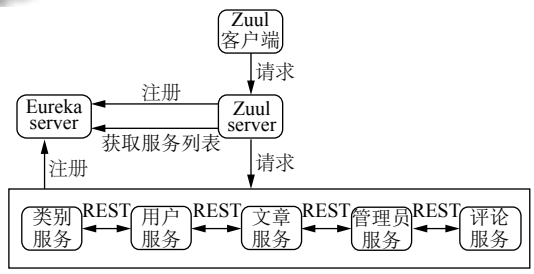

图 1 微服务架构图

3 微服务性能分析

3.1 雪崩效应处理机制

微服务之间是使用轻量级通信机制进行通信, 当某一个服务提供者因为网络原因无法被调用时, 其后的服务消费者都会出现“级联故障”, 即雪崩效应, 如图 2.

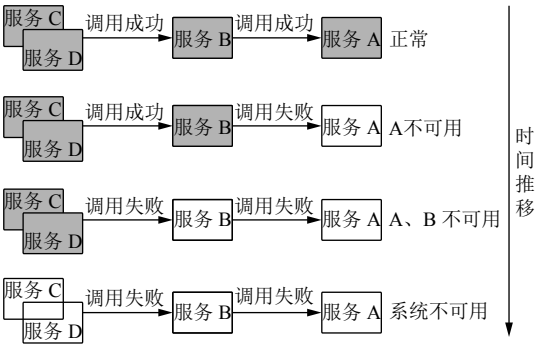

图 2 雪崩效应

使用 Spring Cloud 的 Hystrix 提供的熔断机制, 一旦服务提供者出现错误导致服务消费者无法调用, 系统会立即根据编码人员的设置, 对请求失败、超时执

行回退代码, 防止雪崩效应, 从而提升整个系统的可用性.

3.2 实验性能测试

使用 Jmeter 测试工具, 在近似相同环境下对基于微服务架构系统与基于单体应用的系统进行测试, 为了尽可能保持测试数据的客观性, 两个系统的业务功能逻辑代码实现基本相同. 微服务系统测试结构如图 3 所示.

使用 Jmeter 测试软件, 对微服务与单体应用两个系统设置 20 个用户和 50 个用户, 进行 100 000 个样本测试, 50 个用户的单体应用测试结果参见图 4.

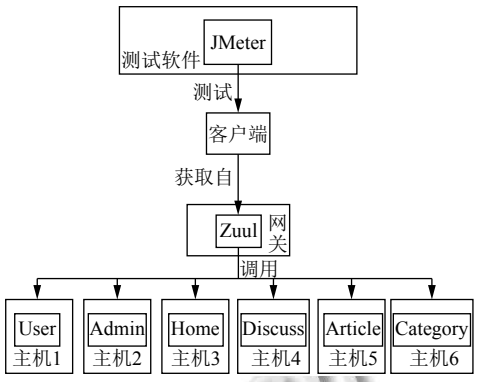

图 3 微服务测试结构图

图 4 单体应用 (50 个线程)

以图 4 为例, 在 100 000 个样本中, 平均响应时间为 21 ms, 50% 的响应时间在 2 ms 之内, 90% 的响应时间不超过 80 ms, 最小响应时间为 1 ms, 最大响应时间为 489 ms, 出错率为 0, 吞吐量为每秒 2222 次, 接收数据量为 25 583.25 KB/s, 发送数据量为 253.88 KB/s. 各测试数据汇总后, 性能数据如表 1、表 2.

表 1 20 个用户性能比较表

功能	平均响应时间 (ms)	50% 响应时间 (ms)	90% 响应时间 (ms)	最小响应 (ms)	最大响应 (ms)	出错率 (%)	吞吐量 (次/s)
微服务	8	3	8	0	2502	0.02	2418.8
单体应用	11	6	19	1	1296	0.02	1755.9

表 2 50 个用户性能比较表

功能	平均响应时间 (ms)	50% 响应时间 (ms)	90% 响应时间 (ms)	最小响应 (ms)	最大响应 (ms)	出错率 (%)	吞吐量 (次/s)
微服务	16	3	35	0	2721	0	2813.7
单体应用	21	2	80	1	489	0	2222.0

由两表对比可知, 在样本数量近似相同的情况下, 微服务架构的论坛系统的响应时间低于单体应用的响应时间, 吞吐量则比单体应用的论坛系统高. 原因是微服务架构的论坛系统具有容错机制, 一旦基本服务发生无法访问, 系统也将立即终止访问, 从而返回用户一个编码时设定的缺省值, 之后会再重试调用出错的服务, 所以无论系统是否出现错误, 用户都会得要一个友好界面和内容, 保证了用户的体验感.

3.3 部署时间分析

微服务架构相较于单体应用的最大优势就是部署效率较高, 传统单体应用每修改一个功能或者缺陷就必须关闭服务器重新部署整个项目, 随着需求的不断增大, 项目代码量不断增多, 重新部署耗费更多时间. 微服务由于采用领域驱动设计, 每个微服务之间相互隔离, 低耦合、高内聚性使得微服务每修改一个功能或者缺陷只需要重新部署相对应的微服务, 其他服务可以继续运行不必停止.

实验 1. 分别对单体应用和微服务架构系统进行部署, 记录不同服务修改的部署平均时间, 每个服务修改一个功能, 部署时间见表 3.

表 3 不同服务数修改一个功能部署平均时间 (ms)

功能	1 个服务	2 个服务	3 个服务
微服务	2234	4227	6346
单体应用	5951	5827	6023

实验 2. 分别修改单体应用和微服务架构系统中的同个服务中的多个功能, 平均部署时间数据见表 4 所示. 修改并部署较少服务的时候, 相比于单体应用架构的系统, 微服务架构在部署时间上花费更少, 节省了约 60% 的部署时间.

表 4 同个服务中修改不同功能后部署平均时间 (ms)

功能	1 个功能	2 个功能	3 个功能
微服务	2222.67	2139.67	2088.67
单体应用	5971.67	5929	5868.3

基于微服务架构的测试系统共由 8 个不同服务构成, 由表 3 数据可得, 当修改服务数不超过两个时, 即修改服务数占总系统服务数的 20% 左右时, 微服务部署时间少于单体应用架构的部署时间. 实验结果也符合软件故障 80/20 原则. 依据表 4 可知, 当所有修改的功能模块是位于同个服务中时, 微服务架构的部署的时间相比于单体应用架构明显加快, 原因在于微服务修改功能模块都在同一个服务中, 只要部署该服务而不必重新部署整个系统, 所以避免花费许多不必要的部署时间和资源, 相反, 单体应用架构的系统, 无论修改的功能是否在同一个模块中, 都得重新部署整个系统, 大大浪费了部署资源和时间. 因此, 微服务架构对于软件系统的维护与部署有着很好的性能优势.

4 结论和下一步工作

微服务是一个细粒度的 SOA (Service-Oriented Architecture, 面向服务架构), 服务的划分基于领域驱动设计, 每个微服务只专注自己的职责, 符合软件设计高内聚、低耦合原则. 微服务单独部署, 服务之间使用 REST 风格通信机制, 各个微服务部署在不同主机并采用分布式管理机制.

传统单体应用程序在项目变得越来越庞大时, 任意一个 bug 将导致整个应用系统重新部署. 微服务架构只需要部署更新的微服务, 任何一个功能修改, 只需要停止对应的微服务, 不需要暂停整个系统, 解决了 bug 修复和系统更新需要停止整个系统访问的问题. 从实验结果看, 系统的性能在微服务架构系统上具有明显优势.

未来将使用容器引擎 Docker 更快地将微服务进行打包、测试以及部署. 基于进程隔离技术的 Docker, 将缩短从编码到部署运行的周期.

参考文献

1 周立. Spring Cloud 与 Docker 微服务架构实战. 北京: 电子工业出版社, 2017.

2 舒后, 熊一帆, 葛雪娇. 基于 Bootstrap 框架的响应式网页设计与实现. 北京印刷学院学报, 2016, 24(2): 47–52. [doi: 10.3969/j.issn.1004-8626.2016.02.013]

3 Götz B, Schel D, Bauer D, *et al*. Challenges of production microservices. Procedia CIRP, 2018, 67: 167–172. [doi: 10.1016/j.procir.2017.12.194]

4 Gade AN, Larsen TS, Nissen SB, *et al*. REDIS: A value-based decision support tool for renovation of building portfolios. Building and Environment, 2018, 142: 107–118. [doi: 10.1016/j.buildenv.2018.06.016]

5 冯新扬, 沈建京. REST 和 RPC: 两种 Web 服务架构风格比较分析. 小型微型计算机系统, 2010, 31(7): 1393–1395.

6 杨开振. 深入浅出 MyBatis 技术原理与实战. 北京: 电子工业出版社, 2016.

7 Helleland C, Hervik S. Wick rotations and real GIT. Journal of Geometry and Physics, 2018, 123: 343–361. [doi: 10.1016/j.geomphys.2017.09.009]

8 许晓斌. Maven 实战. 北京: 机械工业出版社, 2014.

9 杨恩雄. 疯狂 Spring Cloud 微服务架构实战. 北京: 电子工业出版社, 2018.

10 Suryotrisongko H, Jayanto DP, Tjahyanto A. Design and development of backend application for public complaint systems using microservice spring boot. Procedia Computer Science, 2017, 124: 736–743. [doi: 10.1016/j.procs.2017.12.212]

11 Jander K, Braubach L, Pokahr A. Defense-in-depth and role authentication for microservice systems. Procedia Computer Science, 2018, 130: 456–463. [doi: 10.1016/j.procs.2018.04.047]

12 Sharma S. Mastering Microservices with Java. Birmingham: Packt Publishing, 2016.