Design and Implementation of Intelligent Medical System based on Microservices

Hai Bai*[1], Xiaoyan Liu[2]
email address[1]: 3305965101@qq.com, email address[2]: 463480242@qq.com

School of Information Engineering and Automation, Kunming University of Science and Technology, Kunming 650500, China

Abstract. It aims at the shortcomings of the current traditional monolithic medical information management system such as stability, scalability, coordination between development and testing and operation and maintenance personnel, and long software development cycle. The Spring Cloud framework of microservices was used to redesign the traditional monolithic system to build an intelligent medicine management system (IMM) system. The response speed and scalability of the reconstructed IMM system are greatly improved, and it is convenient for O&M personnel to better manage it, which solves the complexity of the traditional monolithic architecture extension function. The results show that the average delay of the reconstructed IMM system in calling services is about 370ms, the submission delay of data services is about 240ms, and the load capacity is also increased by 42%, which is a huge improvement compared with the traditional monolithic system.

Keywords: Microservice; Spring Cloud; Service Latency; Load capacity

1 Introduction

At present, most of the domestic medical information management systems are based on the traditional monolithic architecture to develop software systems [1-3], due to the continuous development of computer technology, the quality requirements for the current software have begun to become higher and higher. However, the system developed using the traditional monolithic architecture involves more modules at start up, and the entire system startup cycle is long. Functions can only be rounded when expanding systems are rescaled and individual functional modules cannot be expanded. Poor system isolation, any module error will lead to the entire system downtime, these drawbacks lead to great problems in the face of multi-user, high-demand situations. Therefore, this paper designs and implements a medical information management system based on microservice architecture, and the use of microservice architecture [4] can effectively solve these problems. The current industry trend shows that large companies at home and abroad are transforming from service-oriented architecture (SOA) to microservice architecture, the most well-known of which are Alibaba, JD.com, Tencent, Twitter, Amazon, etc., and microservice architecture has been widely used. Microservice architecture splits the system into several relative, independent, autonomous microservice nodes [5] by function, each microservice runs in its own process, and communicates between each microservice through communication protocols such as HTTP and lightweight APIs, when one microservice fails, the impact on other microservices is small, so each microservice can be developed, deployed, and run independently [6]. This makes the system more convenient in the

MSIEID 2022, December 09-11, Chongqing, People's Republic of China
Copyright © 2023 EAI
DOI 10.4108/eai.9-12-2022.2327629

expansion of functions, with high cohesion and low coupling characteristics, in line system into multiple microservice nodes with different functions with software quality requirements.

2 Related work

Microservice is a software architecture that can disassemble the traditional monolithic operation and maintenance of the entire according to functions, and each microservice node communicates with each other through communication protocols such as HTTP and lightweight APIs. The reason for the current widespread adoption of microservice architecture: each microservice is an independent project, can be deployed and developed independently, if you need to change the function, you can modify it for a single microservice that needs to be changed, do not depend on other microservices, and the degree of coupling is low. The split microservices can be started quickly. Microservices can be dynamically scaled on demand. Through these characteristics, each microservice node can manage its own functions and its own database in its own process, making the development, testing, and system more convenient [7].

In order to improve the load capacity of the software, there are usually two solutions: Configure enough hardware to ensure that the server configuration is enough to bear the needs of users, that is, hardware load balancing. However, this method is costly, and it may not use so much computer resources when the number of users is insufficient, resulting in a waste of resources. The software load balancing method is adopted to solve the problem of load imbalance of microservice clusters. Servers in a microservices architecture are usually deployed in clusters to increase the number of visits and complete tasks that a single server cannot complete. However, when building a service cluster, there is often a large gap between the system resource utilization of each server, resulting in a significant reduction in the performance of the cluster. In order to solve this problem, an efficient load balancing mechanism must be used to distribute requests reasonably to the back-end servers. This realizes load balancing of microservice clusters and improves the load capacity and resource utilization of the system. Common microservice load balancing algorithms are divided into static load balancing algorithms and dynamic load balancing algorithms. Inthis article, the polling selection algorithm in the static load balancing algorithm is used to implement the load balancing of the system. The algorithm sends user requests to the servers of the microservice cluster in turn, which has the advantages of balance, efficiency, simple implementation, and easy to scale out. Microservice frameworks include: Spring Boot, Dubbo, Service Fabric, Spring Cloud, of which Spring Cloud has become the standard framework for microservices.

Spring Cloud [8], a microservice framework based on Spring Boot, simplifies the development of distributed systems through the characteristics of Spring Boot, so that the basic components of related microservices, such as service discovery and registration, gateways, load balancing of microservice clusters, circuit breakers, etc., can be started and deployed with one click. The component framework call relationship of Spring Cloud is shown in Figure 1.

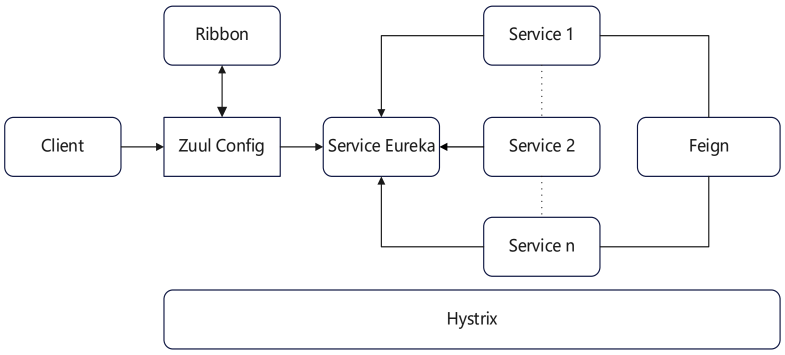

Figure 1 Spring Cloud framework

Spring Cloud is a one-stop solution for distributed microservices architecture, which provides an easy-to-use programming model that allows us to easily build microservices systems on top of Spring Boot. Spring Cloud is known as the "family bucket" for building distributed microservices systems, which is not a single technology, but an ordered collection of microservices solutions or frameworks. It integrates the mature and proven microservices frameworks on the market, reencapsulates them with Spring Boot thinking, masks complex configuration and implementation principles, and ultimately provides developers with a distributed systems development kit that is easy to understand, easy to deploy, and easy to maintain. Spring Cloud includes nearly 20 sub-projects such as spring-cloud-config and spring-cloud-bus, providing solutions in the fields of service governance, service gateway, intelligent routing, load balancing, circuit breakers, monitoring and tracing, distributed message queuing, and configuration management. Spring Cloud itself is not a ready-to-use framework, it is a set of microservices specifications with two generations of implementations. Spring Cloud Netflix is the first generation implementation of Spring Cloud and consists mainly of components such as Eureka, Ribbon, Feign, Hystrix, and others. Spring Cloud Alibaba is the second generation implementation of Spring Cloud, which mainly consists of components suchas Nacos, Sentinel, and Seata. The Spring Cloud framework provides efficient and available microservices infrastructure components [9], including:

Zuul is a microservice gateway of a system in the middle layer between the front and back ends. A total of 4 standar filter types are defined [10].

1.PRE, implement authentication.

2.ROUTING, build the device sent to the microservice node.

3.POST, add a standard HTTP header for the response.

4.ERROR, execute when an error occurs.

When a request from a front-end to the backend is called, all requests must first pass through the Zuul gateway to access the internal service, locate to the specific service node, and the ribbon decides which service the request is assigned to.

The task of Ribbon [11] is mainly to load balance the microservice cluster and determine the performance and stability of the entire microservice cluster. Its essence is a client component that is load balanced by software. In the case of high concurrent access usage of the system, the previously configured load balancing algorithm is used to distribute its request trafficbalance to multiple servers to achieve the purpose of expanding server bandwidth, enhancing data processing capabilities, increasing throughput, and improving network availability and flexibility. Commonly used load balancing algorithms are divided into static load balancing: round-robin, random, weighted round-robin, etc. and dynamic load balancing: minimum number of connections, maximum response speed method.

Service Eureka [12] is mainly responsible for registering and discovering each microservice node in the system, including server and client. Eureka Server provides service registration function, when the microservice starts, it will register its own service to the server, the server maintains a list of available services, stores the information of all available services registered to the server, these available services can be seen intuitively in the management interface of the server. Eureka Client usually refers to the various microservices in the microservice system, which are mainly used to interact with the server.

Hystrix [13] as a circuit breaker mechanism, the main role is the system fusing treatment. When the system times out of the request and a single or part of the microservices are unavailable, the Hystrix circuit breaker can provide a downgrade scheme according to the built-in circuit breaker mechanism, that is, provide a designed downgrade scheme after the request fails, and call the method immediately. And use the circuit breaker mechanism to prevent diffusion to other services and ensure the stability of microservices under high concurrency conditions.

Feign is a declarative web service client, which can help call HTTP APIs more conveniently, only need to create an interface and add an annotation to achieve mutual access between various microservices.

3 Overall system design

In order to shorten the system startup cycle, expand the function without affecting other service functions, reduce the coupling degree of the overall system, use the microservice architecture to design the IMM system, refine the entire system into relatively independent microservice nodes according to different functions, and componentize each microservice node. In order to cope with high concurrency access, the system separates the front and back ends to make calls, and divides the entire system into three modules: front-end service, back-end service, and gateway service. When using the IMM system, the user first authenticates the identity through the unified authentication platform in the front-end service, enters the business module after successful authentication, the user selects the function to be used, and the service request sent by the front-end service is sent to the gateway service by the front-end service through the HTTP communication protocol. As an intermediate component that responds to calls between the front-end service and the back-end service, the gateway service is an API gateway and has components such as microservice cluster Ribbon load balancing, Hystrix circuit breaker, and so on. The back-end service has a complete microservice cluster, and all microservice nodes have a unified shared feature, when the back-end service receives a request, it will return the service call to the front-end service through HTTP communication through the gateway service.

The system mainly contains user information: complete the user information management service function. Drug retrieval: Users can query drug keywords and their own related symptoms online, and the system recommends related drugs through symptom information, and can choose to enter the drug encyclopedia function after successful query. Complete collection of drugs: After users query drugs online, the dosage, course of treatment and precautions of drugs are displayed according to the query drugs. Medical masterpieces: Realize users' online access to classic works of related medicine. Common diseases: Users can query the causes, treatment methods and precautions of common contemporary diseases online. Recommended meals: Enableusers to check the nutrients they need and the foods they temporarily avoid online according to their own diseases. User suggestions: Users should propose to the administrator the functions that the system should improve or need to be extended according to their own experience. Data backup: Prevent the loss of relevant information searched by users or some drugs, medical masterpieces and related dietary therapies entered by the administrator when some failures occur in the system. Data recovery: In the event of data loss, the data recovery function recovers its lost data through previous data backups. In this article, each of these functions serves as microservice nodes for the system. The IMM architecture diagram is shown in Figure 2.

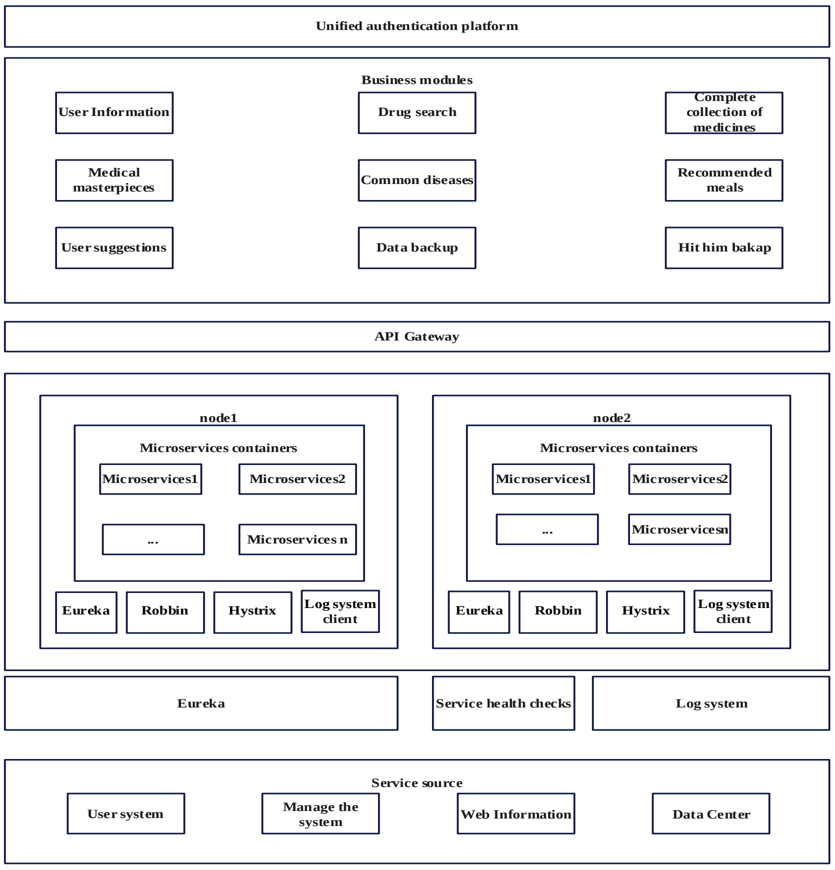

Figure 2 IMM system architecture

4 System test and result analysis

The system is implemented using Spring Cloud's microservice framework, and some components in Spring Cloud are used in the implementation process, such as Eureka for service registration, Zuul for filtering tokens as a gateway, Ribbon to achieve load balancing of microservice clusters, and Hystrix, a circuit breaker to prevent an avalanche effect from the entire system.

4.1 Test Standards

Software testing standards are divided into:

4.1.1 Benchmarking

Understand the resource consumption of the system while it is idle, such as CPU, IO, network bandwidth, network connection.

4.1.2 Single consecutive request test

TPS, response time, server resource consumption in the case of 1 consecutive request, mainly for reference.

4.1.3 Load Testing

Gradually increase the number of concurrent requests, look at TPS, response time, error rate, server resource consumption, mainly used to analyze maximum performance.

4.1.4 Stress Testing

Test maintaining stress at the critical point of maximum performance to see how well the server handles high-stress situations.

4.1.5 Stability test

For xx consecutive days, test under a certain concurrency to see whether the server operation can meet the requirements design.

4.2 Test Scheme

The traditional monolithic system and the IMM system based on microservice architecture are operated in the same operating environment for stability testing and load testing. The system operating environment is: 6-core CPU and 16GB memory, and the running system is Windows 10.

Use the resource monitor that comes with Windows 10 to monitor and collect data for the two systems for 10 consecutive days, and record the response time and immediate delay of calling services when running for 15, 30, 45, 60, 75, 90, and 105 minutes when they are running the same number of concurrent requests every day. After recording, continue to increase the number of concurrent requests, use the j meter tool to simulate high concurrency, view the number of

requests per second of the system, and analyze the load capacity of the two system.

4.3 Test Results

The previous traditional monolithic system has been redesigned to optimize the call delay and submission delay of the entire system. Compared with the traditional monolithic architecture system, Table 1 shows that the delay of calling interfaces in the reconstructed new IMM system within 2hours has been reduced by 56.8%, and Table2 shows that the commit delay of the reconstructed new IMM system has been reduced by 60.7%, and the system has made great progress in response speed. At the same time, in terms of load capacity, the new IMM system realizes the load balancing of microservice clusters due to the round-robin selection algorithm. Figure 3 shows that the new reconstructed IMM system handles the number of requests per second under different request concurrency conditions, and its load capacity is also increased by 42%.

Table 1. Comparison of call delays

Time/min	Average latency of microservices/ms	Monolithic average delay/ms
15	186	548
30	273	648
45	242	628
60	215	607
75	266	670
90	231	564
105	253	580

Table 2. Submit a delay comparison

Time/min	Average latency of microservices/ms	Monolithic average delay/ms
15	405	821
30	345	864
45	326	785
60	376	928
75	355	822
90	402	912
105	400	923

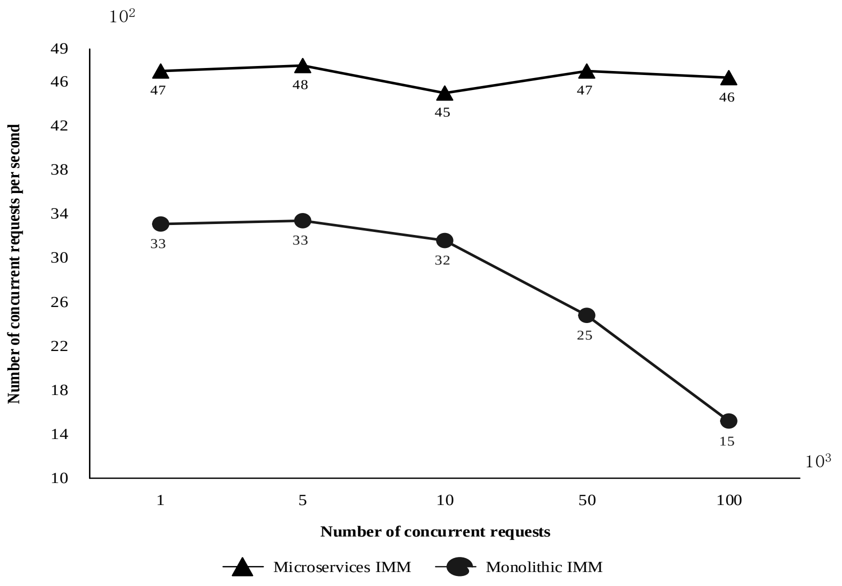

Figure 3 Comparison of operating efficiency

4 Conclusion

In view of the shortcomings of the traditional monolithic medical information management system, the concept of microservice architecture is used to redesign it. Using the components of Spring Cloud, the design and implementation of the intelligent medical system based on the microservice architecture are realized, and the response speed and load capacity of the IMM system have been greatly improved after the reconstruction. The system is implemented using the Java language and the Intellij IDEA development platform.

References

[1] HUANG Xianshun, CHEN Jia-liang. Design and implementation of medical information management system[J]. Examination Weekly,2018(54):12+14.)

[2] Ma Jie. Design and implementation of information management system of Jinan pharmaceutical company[D]. Shandong University,2014.

[3] ZHOU Xiangrong. Development and design of medical consulting service system[J]. ElectronicWorld,2012(24):151-153.

[4] de Almeida Murilo Góes, Canedo Edna Dias. Authentication and Authorization in Microservices Architecture: A Systematic Literature Review[J]. Applied Sciences,2022,12(6).

[5] Joseph Christina Terese, Chandrasekaran K.. IntMA: Dynamic Interaction-Aware Resource Allocation for Containerized Microservices in Cloud environments[J]. Journal of Systems Architecture,2020,111 (prepublish).

[6] Ruiqi Zeng, Yue Zhao, Hong Su, Xiaoyu Guo. ANovel Construction Technology of Enterprise Business Deployment Architecture Based on Containerized Microservices[C]//.

Proceedings of the 5th International Conference on Communication, Image and Signal Processing(CCISP2020).,2020:274-281.DOI:10.26914/c.cnkihy.2020.032043.
[7] Yang Tianyi. On the advantages and disadvantages of microservice architecture[C]//. Proceedings of the 36th China (Tianjin) 2022' IT, Network, Information Technology, Electronics, Instrumentation Innovation Conference. [Publisher unknown], 2022:294-297.DOI:10.26914/c.cnkihy.2022.01504
[8] Li Na. Application of Spring Cloud microservice architecture[J].Electronic Technology and SoftwareEngineering,2019(12):142.)
[9] HU Shaoxuan. Design and implementation of academic affairs management system based on SpringCloud [D].Jilin University,2022.DOI:10.27162/d.cnki.gjlin.2022.005019.)
[10] GE Meng, LI Chuangnan, OUYANG Hongji. Application and implementation of microservice architecture based on Spring Cloud[J].Modern Information Technology,2021,5(19):23-26.DOI:10.19850/j.cnki.2096-4706.2021.19.005.
[11] Original Ming. Research on load balancing mechanism based on microservice architecture [D].Beijing Institute of Graphic Communication,2022.)
[12] WU Xiongjin. Design and development of service registry in microservice framework[J]. Industrial Control Computer,2021,34(08):130-132.)
[13] Wang Zhuo. Optimization analysis and research based on Hystrix service circuit breaker degradation strategy[J].Software,2022,43(08):125-127.