UDC [004.72+654]-027.31

INFORMATION SYSTEM DESIGN BASED ON MICROSERVICE ARCHITECTURE

HE RUNHAI, LI BOYI, ZHOU QUANHUA, ZHONG WU, ZHANG HENGRUI

Belarusian State University of Informatics and Radioelectronics, Republic of Belarus

Received March 15, 2023

Abstract. In recent years, microservice architecture has become the mainstream approach to modern information system design, and the central idea of this architecture is to improve the scalability, reliability and maintainability of the system by splitting it into small, independent service units. This paper introduces the main ideas and practices of information system design based on microservice architecture, including service splitting, service communication, service governance and service deployment. In addition, this paper analyzes the advantages and disadvantages of the current microservice architecture, and illustrates how to design and implement information systems based on microservice architecture in practical applications with real cases.

Keywords: microservices, architecture design, service communication, service governance, service deployment.

Introduction

As the internet technology advances, information systems have become a vital part of various industries. However, traditional monolithic applications are no longer sufficient for modern applications. This has led to the emergence of microservice architecture as a new approach to software architecture, which is gaining increasing attention. Microservice architecture splits a system into small, independent service units, which enhances system scalability, reliability, and maintainability [1]. This paper aims to introduce the main ideas and practices of designing information systems based on microservice architecture. We will also analyze the advantages and disadvantages of microservice architecture and provide real-life examples of how to design and implement information systems based on microservice architecture in practical applications.

Service Splitting

In microservice architecture, splitting the entire system into multiple small, independent service units is a very critical step. The purpose of service unbundling is to break down a large, complex system into multiple, small, easily managed and maintained parts. The principle of unbundling is usually to divide parts with clear responsibilities and functions into separate service units [2]. For example, an online shopping system can be split into multiple services, such as catalog service, cart service, order service, payment service, etc.; the division is show in the Figure 1.

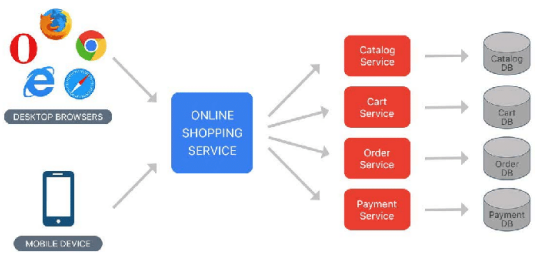

Figure 1. Service Splitting

Service Communication

In a microservice architecture, communication between services is achieved through the network. There are usually two ways of communication between services: synchronous and asynchronous. In synchronous communication, services need to wait for each other to respond, while in asynchronous communication, services do not need to wait for each other to respond. The differences between these two ways are shown in the Figure 2. The advantage of synchronous communication is that it is simple and straightforward, easy to understand and debug, but it can cause performance bottlenecks in large-scale systems. Therefore, asynchronous communication is usually used in large-scale systems, such as message queues, event-driven, etc. With asynchronous communication methods, services can hand over requests to other services for processing and respond when they get the results after the processing is completed. Service communication also needs to consider the protocol, format, interface and security aspects between services, so it is necessary to know about network communication, API design, security and data format.

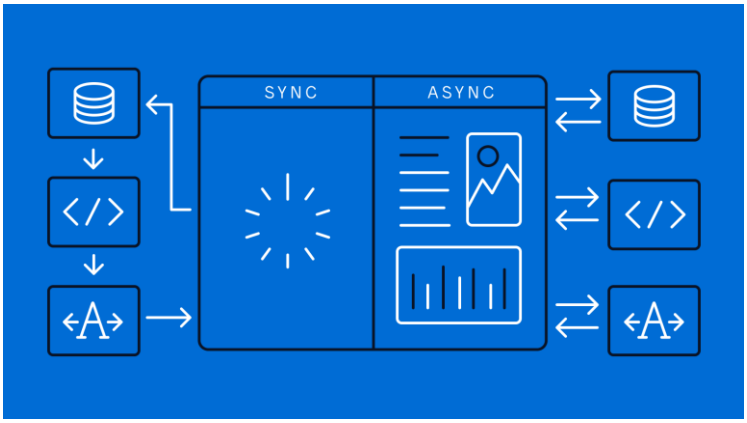

Figure 2. The difference between synchronous and asynchronous

Service Governance

Services in microservice architectures are often decentralized, so the governance and management of services is a very important issue. Service governance includes aspects such as service registration, service discovery, load balancing, failover and monitoring [3].

Service registration and discovery refers to registering services to service centers so that other services can discover them and communicate, there are some common SpringBoot combined with microservice architecture service discovery tools such as Eureka, Consul, Zookeeper, etc. These service discovery tools and frameworks provide APIs and UI to manage service registration and discovery, making it easy for developers to use them to build microservice architecture.

Load balancing is the balanced distribution of requests to multiple service instances to improve system performance and availability, there are some common SpringBoot combined with microservice architecture load balancing are Ribbon, Eureka, Feign, Zuul, etc. Failover is the ability to automatically switch to another available instance when a service instance fails. Monitoring is the ability to improve service availability and performance by monitoring the operational status and performance metrics of the service. The implementation of service governance requires learning about service registration and discovery, load balancing, failover, and monitoring.

Service Deployment

Services in a microservice architecture are usually deployed independently, so the deployment and management of services is also an important issue. The deployment of services needs to consider several aspects, such as runtime environment, configuration management, containerization, etc. Among them, containerization is a common deployment method that packages services and their dependencies into container images for deployment in different environments. Containerization can also improve the

portability, scalability and reliability of services. Service deployment requires knowing about containerization techniques, automated deployment and configuration management.

Advantages and disadvantages analysis

A complete microservice application development and deployment is divided into the process of technical framework building, basic function development, business function development, multi-system integration, production environment deployment and upgrade, production environment problem solving, problem handling, etc. [4]. The development scenarios of directly using the original technology for microservice application development and using the microservice application development platform are different.

Microservices architecture has many advantages, for example, it can improve the scalability, reliability and maintainability of the system, the development scenario of using microservice application development platform shields the underlying complex technology, simplifies the development process, avoids a lot of repetitive development work, and reduces the development difficulty, improves efficiency and quality, increases the productivity of the team. However, microservice architecture also has some disadvantages, for example, it increases the complexity of the system, requires higher technical level and management cost, and may have communication problems between services. Therefore, you need to weigh the advantages and disadvantages when choosing a microservice architecture, and make a choice based on specific business needs and technical strengths.

Practical Case Analysis

Uber is a leading global mobility service platform that provides a wide range of mobility services such as online cars, cabs, carpooling, bike sharing and flying cars. To support the efficient and rapid growth of these services, Uber has adopted a microservices architecture to build its information system. Uber’s microservices architecture consists of thousands of microservices that can be deployed, extended and upgraded independently.

Each microservice has its own specific responsibilities and functions, such as passenger information management, driver routing, payment services, and more. This architecture allows Uber's information systems to be more flexible, scalable and maintainable. In Uber's microservice architecture, each microservice has its own database and API, and the microservices communicate with each other through RESTful APIs, which allows for loose coupling and distributed deployment. In addition, Uber employs distributed message queues to support asynchronous communication between microservices [5]. These technologies allow Uber's microservices to scale horizontally to meet massive concurrency and high load requirements. To support the microservice architecture, Uber also employs containerization technologies and automated deployment tools. Uber uses Docker as the containerization technology to make the deployment and management of microservices easier and more efficient. Also, Uber has developed its own automated deployment tools, such as DeployBot and Cherami, to automate the deployment and management of microservices. Uber also uses a variety of monitoring and troubleshooting tools to ensure high availability and performance of microservices [5]. For example, Uber uses Zipkin to track calls and response times between microservices, Hystrix for circuit breaker patterns and failover, and Prometheus to collect and analyze system metrics. These tools help Uber quickly diagnose and resolve microservice failures and performance issues.

Overall, Uber's microservices architecture is a highly scalable, loosely coupled and distributed architecture that can support rapid iteration and innovation across multiple travel services. At the same time, Uber uses a variety of technologies and tools to ensure high availability and performance of microservices. This architecture has proven to be very effective in practice and can provide useful references and lessons for other enterprises.

Conclusion

Microservice architecture has emerged as a new approach to software architecture that enhances the scalability, reliability, and maintainability of information systems. The splitting of services, communication, governance, and deployment are all critical aspects of designing information systems

based on microservice architecture. Microservices architecture offers many advantages, such as scalability, reliability, and maintainability, but it also has some disadvantages, such as increased complexity and higher management costs. Therefore, the choice to adopt microservice architecture should be based on specific business needs and technical strengths. As exemplified by Uber, microservices architecture has been implemented successfully in practical applications, enabling efficient and rapid growth of services, and improving flexibility, scalability, and maintainability of information systems.

References

1. Fowler, M. Microservices: a definition of this new architectural term, 2014. [Electronic resource]. URL: http://martinfowler.com/articles/microservices.html.
2. Xiao B, Wang YF. / An efficient microservice application development platform architecture design. // China Science and Technology Information. 2023. P. 96–98.
3. Zhao, C., Zhang, J., Zhang, L., C. Research on key technologies of microservice architecture. // Journal of Software. 2017. Vol. 28. P. 1235–1253.
4. Zhang, J., Zhang, C. / A Survey of Microservice Architecture: Principles, Characteristics, and Technologies. // ACM Computing Surveys (CSUR). 2018. Vol. 51. P. 1–35.
5. Uber Engineering. How Uber's microservices platform is built, 2016. [Electronic resource]. URL: https://eng.uber.com/microservice-architecture/.