Spring Cloud 微服务架构浅谈

王齐

（杭州师范大学 信息科学与技术学院, 杭州 311121）

摘 要：微服务在近几年不断流行起来，并已经成为当下最优秀的系统架构。与传统的单体式架构不同，微服务架构将复杂应用拆分成多个独立自治的服务，服务与服务之间通过松耦合的形式进行交互。作为微服务的代表 Spring Cloud 是目前市场上主流的开源微服务框架之一。

关键词: 微服务；系统架构；Spring Cloud

A Brief Talk on Spring Cloud Microservice Architecture

WANG Qi

(*School of Information Science and Technology, Hangzhou Normal University, Hangzhou 311121, China*)

Abstract: Microservices have become popular in recent years and have become the best system architecture at the moment. Different from the traditional monolithic architecture, the microservice architecture splits complex applications into multiple independent and autonomous services, and services and services interact in the form of loose coupling. As a representative of microservices, Spring Cloud is one of the mainstream open source microservice frameworks on the market.

Key words: microservice; system structure; Spring Cloud

0 引言

微服务架构基于 SOA 架构演进而来，它根据业务功能设计出多个独立的服务，并以全自动的方式进行部署，各个服务之间可以相互通信。与传统的单体架构相比，微服务架构具有服务组件化、独立部署化、高扩展性、技术栈灵活等特点[1]，但同时也带来开发和运维的复杂性。

1 概述

传统的 web 开发方式大都是一个 WAR 包整合所有的功能，包含 DO/DAO 层、Service 层、UI 等所有的业务逻辑和实现，然后部署在一个容器（常用的容器有 Tomcat、Jboss、WebLogic 等)里运行。它的主要优点是开发简单，集中式管理；功能都在本地，没有分布式的管理和调用消耗等，但

是也存在比如效率低、维护难、稳定性差、扩展性不够等问题。随着用户要求的提高，对于大型的系统来说传统架构已经很难满足市场需求，于是衍生出了分布式架构，又从分布式衍生出了 SOA 架构，在这个过程中，功能的颗粒度被拆解得越来越小，这就发展成为了微服务架构[2]。

在微服务架构出现之前，传统的架构有 SOA 架构和单体架构。单体架构在规模较小的情况下能够较好地满足业务需求，随着系统规模的不断扩大，单体架构应用的不足也越来越突出，主要有以下几个方面：

(1) 企业的信息化和 IT 系统建设发展到一定程度后，自然会从 IT 系统的规划和建设期发展到后期的 IT 系统管控治理和运维期。到了后期不再有大量的新系统规划建设,而更多的是为了业务流程优化进行的 IT 系统需求变更、优化和功能改造。采用传统单体架构的应用，面对需求变更或者功能变更时,往往需要很长时间的发布周期,对业务的快慢变化响应较慢。

(2) 企业人员流动是所有企业面临的最正常的事情，由于疏于代码质量的自我管束，离职员工可能在功能实现中遗留有不少的隐形缺陷,由于单体应用项目代码量非常之大,遗留的问题很难被识别和纠正，在这种情况下，增加一个新的功能反而可能会导致多个老功能出现问题,技术债务越滚越大。

(3) 传统单体应用从需求到开发到测试到发布整个过程自动化程度低。自动化程度低加之代码量大、规模庞大，导致有些项目部署所花费的时间越来越多,研发人员在启动项目和调试上花费的时间越来越长,极大增加研发成本，也降低了研发人员的满意度感受。

(4) 传统单体应用无法做到按需伸缩。应用各业务模块对资源的需求不同,有的业务模块属于 CPU 密集型,有的业务模块属于 IO 密集型,但是由于同在一个单体应用内部,所有的模块都在一个架构下，因此在扩展某一个模块的性能时，不得不考虑其他模块的因素,无法做到真正的按需进行伸缩。

2 微服务架构

2.1 定义

2014 年 Martin Fowler 与 James Lewis 共同提出了微服务的概念，定义了微服务是由以单一应用程序构成的小服务，自己拥有自己的进程与轻量化处理，服务依业务功能设计，以全自的方式部署，与其他服务使用 HTTP API 通信。同时服务会使用最小的规模的集中管理(例如 Docker)能力，服务可

以用不同的编程语言与数据库等组件实现。

微服务架构解决的关键问题可以概括为两个：

(1) 如何快速响应业务需求变化并发布系统?

(2） 如何以最小的代价和影响来发布系统?

微服务，关键不仅仅在于微服务本身，而是系统提供一套完整的基础架构,满足微服务独立的部署、运行、升级，使微服务与微服务之间实现结构上的“松耦合”,整体功能上表现为一个统一的整体，包括统一的前端风格、统一的权限认证、统一的安全防护、统一的运维管理以及统一的门户等等。

那么,微服务架构该如何定义?微服务架构是以专注于单一职责的小型功能模块为基础，通过 API 集成相互通信的方式，实现复杂的业务系统搭建的一种设计思想。

2.2 特点

微服务架构主要有以下几个特点：

(1) 系统功能按照业务进行拆分，即“水平拆分”，从技术层面上进行“前后端分离”，即“垂直拆分”，纵横一起切分，就把单一的应用拆分成网格状的小块应用，这样就体现了微服务中的“微”的概念。

(2) 服务和服务都是独立部署、相互隔离的。服务之间无相互影响，体现了“我为人人，人人为我”的设计理念，也体现了微服务中“服务”的思想。

(3) 服务使用轻量级的通讯协议和简单的数据结构，服务之间不需要去关心对方的模型，只需要按照约定的接口来进行数据的流转,体现了“解耦”的思想。

(4) 服务需要高可用。结合PaaS层容器化技术和动态调度技术,单一的服务可以通过多实例的方式进行扩展,多实例的动态扩展保证了服务的高可靠性，随着服务实例的增加，业务需求的高并发需求得以满足，保证了业务的流畅可用。

2.2 优势与挑战

相对于传统架构，微服务架构具有显而易见的优势：

(1) 服务简单，只关注一个业务功能。传统的整体风格的架构在构建部署和扩展伸缩方面有很大的局限性，其服务端应用就像是一块铁板，笨重且不可拆分，系统中任何程序的改变都需要整个应用重新构建和部署新版本。在进行水平扩展时也只能整个系统扩展，而不能针对某一个功能模块进行扩展。而微服务架构将系统以组件化的方式分解为多个服务，服务之

间相对独立且松耦合，单一功能的改变只需要重新构建部署相应的服务即可。

(2) 每个微服务可由不同团队开发。传统的开发模式在分工时往往以技术为单位，比如 UI 团队、服务端团队和数据库团队，这样的分工可能会导致任何功能上的改变都需要跨团队沟通和协调。而微服务则倡导围绕服务来分工，不同的服务可以采用不同的技术来实现，一个团队中应该包含开发所需的所有技能，比如用户体验、数据库、项目管理。

(3) 微服务是松散耦合的。微服务架构抛弃了 ESB 复杂的业务规则编排、消息路由等功能，微服务架构中服务是高内聚的，每个服务都会处理相应的业务，所有的业务逻辑应该尽量在服务内部处理，且服务间的通信尽可能的轻量化，比如使用 Restful 的方式。

(4) 可用不同的编程语言与工具开发。传统的软件开发中经常会使用同一个技术平台来解决所有的问题，而经验表明使用合适的工具做合适的事情会让开发变得事半功倍。微服务架构天生就具有这样的特性，我们可以使用 Node.js 来开发一个简单的报表页面，使用 C++来编写一个实时聊天组件。

在拥有上述优势的同时，微服务架构可能会面临以下的挑战：

(1) 运维开销。更多的服务也就意味着更多的运维，产品团队需要保证所有的相关服务都有完善的监控等基础设施，传统的架构开发者只需要保证一个应用正常运行，而现在却需要保证几十甚至上百道工序高效运转，这是一个艰巨的任务。

(2) DevOps 要求。使用微服务架构后，开发团队需要保证一个 Tomcat 集群可用，保证一个数据库可用，这就意味着团队需要高品质的 DevOps 和自动化技术。而现在，这样的全栈式人才很少。

(3) 隐式接口。服务和服务之间通过接口来“联系”，当某一个服务更改接口格式时，可能涉及到此接口的所有服务都需要做调整。

(4) 重复劳动。在很多服务中可能都会使用到同一个功能，而这一功能点没有足够大到提供一个服务的程度，这个时候可能不同的服务团队都会单独开发这一功能，重复的业务逻辑，这违背了良好的软件工程中的很多原则。

(5) 分布式系统的复杂性。微服务通过 REST API 或消息来将不同的服务联系起来，这在之前可能只是一个简单的远程过程调用。分布式系统也

就意味着开发者需要考虑网络延迟、容错、消息序列化、不可靠的网络、异步、版本控制、负载等，而面对如此多的微服务都需要分布式时，整个产品需要有一整套完整的机制来保证各个服务可以正常运转。

(6) 事务、异步、测试面临挑战。跨进程之间的事务、大量的异步处理、多个微服务之间的整体测试都需要有一整套的解决方案，而现在看起来，这些技术并没有成熟。

3 微服务架构框架——Spring Cloud

一个大型的复杂的系统由多个微服务组成,系统的各个微服务之间都是松散耦合的,并且能很好地解决单一功能问题。Spring Cloud[3]是当前市场上微服务开源的主流框架之一。

Spring Cloud 是 Spring 体系的微服务解决方案，它在 Spring Boot 基础上集成了包括服务注册与发现、配置中心、网关、服务保护与熔断、分布式配置管理、负载均衡等简单易部署易维护的系统底层开发框架，不需要用户再集成其他的组件即可完成微服务架构的开发以及部署，同时它也支持与其它第三方组件集成开发。其主要组成部分如图 1 所示。

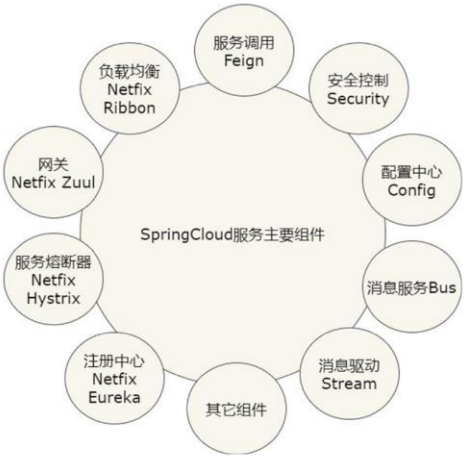

图 1 Spring Cloud 技术体系主要组件

Spring Cloud 常用的 5 个核心组件：

(1) Netflix Eureka —— 注册中心;

(2) Netflix Ribbon —— 客户端负载均衡；

(3) Netflix Hystrix —— 服务熔断器；

(4) Netflix Zuul —— 服务网关；

(5)Spring Cloud Config —— 配置中心。

Spring Boot 框架是 Spring Cloud 的基础，因此它是可以被独立使用来开发应用系统，但是 SpringCloud 却是依赖 Spring Boot 而不能独立使用的，Spring Cloud 对使用 Spring Boot 开发的单体微服务进行组合管理，是有着完整的一个生命周期的微服务框架。

4 关键技术

4.1 负载均衡

负载均衡[4]的目的是为了特定场景下，能够将请求合理地平分到各服

务实例上，以便发挥所有机器的叠加作用。主要考虑的就是不将请求分配到出现故障的机器，性能越好的机器可以分配更多的请求。一般负载均衡可以借助外部工具，硬件负载均衡或软件负载均衡，如 F5/ nginx。当然了，在当前分布式环境遍地开花的情况下，客户端的负载均衡看起来就更轻量级，显得不可或缺。Spring Cloud 提供了几种负载均衡策略:Ran-domLoadBalance（随机负载均衡算法）和 RoundR-obinLoadBalance(轮询负载均衡算法）等策略，可以根据需求进行选择配置。

4.2 通信机制

系统使用微服务架构后，就被拆分成了更细维度的子服务，小到一个功能子模块都可以作为一个微服务，各个微服务之间大都是互不影响的、松耦合的，可以被独立开发以及部署。因此各个服务之间的通讯变得非常重要，RESTful HTTP 协议是微服务架构中最常用的通讯机制。

4.1 自动化部署

在微服务架构中，随着服务越来越多之后，服务的打包部署就会成为一件相当麻烦的事情。比如一个项目中有 10 个微服务需要部署，每次更新之后重新部署都需要手动操作 10 个微服务，这样带来的问题不可预估，有没有什么办法让我们部署一次之后，只要点击执行就可以自动部署呢?我们可以借助 Docker 或者其它工具来完成一个微服务架构中的所有服务的自动化部署工作。

5 总结

随着信息化技术的飞速发展，微服务架构的应用越来越广，在技术选型的时候我们需要根据实际情况选择使用，其中 Spring Cloud 架构的调用方式 Rest API 更符合微服务官方的定义，而且 Spring Cloud 来源于 Spring 家族，它更注重微服务架构生态，是解决微服务架构实施的综合性框架，整合了诸多优秀的组件，并且 Spring Cloud 是 Java 语言中最常用的微服务框架解决方案。Spring Cloud 几乎考虑了服务治理的各个方面，并有 Spring Boot 的支持，开发起来会相对便利和简单，但是具体使用还需要根据具体情况进行研究选择。

参考文献(References)

[1] 汪友杰,王平,李咀庆,王睿昕,张云鹏.浅谈微服务架构[C]//2020 年中国通信学会能源互联网学术报告会论文集.2020:285-289.DOI:10.26914/c.cnkihy.2020.065772.

[2] 马荣彦.Spring Cloud 微服务框架浅析[J].现代电影技术,2021(10):47-50.

[3] 熊益益. 基于微服务架构的电商平台的

研究与实现[D].北京邮电大学,2019.

[4] 冯志勇,徐砚伟,薛霄,陈世展.微服务技术发展的现状与展望[J].计算机研究与发展,2020,57(05):1103-1122.