微服务框架的设计与实现①

张 晶[1], 黄小锋[2], 李春阳[3]

[1](北京中电普华信息技术有限公司, 北京 100192)
[2](中国电建集团国际工程有限公司, 北京 100048)
[3](国网信息通信产业集团有限公司, 北京 100031)

摘 要: 相对于传统单块架构, 微服务框架具有技术选型灵活, 独立部署, 按需独立扩展等优点, 更适合当前互联网时代需求. 但微服务架构的使用引入了新的问题, 如服务注册发现、服务容错等. 对微服务框架引入的问题进行分析, 并给出了微服务框架的一种实现方案, 在框架层面解决服务注册发现、服务容错等共性问题, 使业务系统开发人员专注于业务逻辑实现, 简化系统开发的难度, 提高开发效率.

关键词: 微服务框架; 服务注册; 服务发现; 服务容错

Design and Implementation of Microservice Architecture

ZHANG Jing[1], HUANG Xiao-Feng[2], LI Chun-Yang[3]

[1](Beijing China Power Information Technology Co. Ltd., Beijing 100192, China)
[2](PowerChina International Group Limited, Beijing 100048, China)
[3](State Grid Information & Telecommunication Industry Group Co. Ltd., Beijing 100031, China)

Abstract: Compared with traditional single block architecture, microservice architecture has many advantages, such as flexible technology selection, independent deployment, and independent scalability more suitability for the current needs of the internet age, etc. But microservice architecture also introduces new problems such as service registration, service discovery, service fault tolerance. On the basis of the analysis for problems mentioned above, this paper proposes one implementation of microservice framework, which can solve service registration, service discovery, service fault tolerance and other common problems. Based on this, developers only need to focus on the development of business functions, so that it can simplify the difficulty of system development and improve development effectiveness.

Key words: microservice architecture; service registration; service discover; fault tolerance

传统信息化系统的典型架构是单块架构(Monolithic Architecture), 即将应用程序的所有功能都打包成一个应用, 每个应用是最小的交付和部署单元, 应用部署后运行在同一进程中. 单块架构应用具有 IDE 友好、易于测试和部署等优势, 但是, 随着互联网的迅速发展, 单块架构临着越来越多的挑战, 主要表现在维护成本高、持续交付周期长、可伸缩性差等方面[1].

微服务架构(Microservices)的出现以及在国内外的成功应用, 成为系统架构的一种新选择. 很多大型宝等都已经从传统单块架构迁移到微服务架构[2]. 微服务架构提倡将单块架构的应用划分成一组小的服务, 互联网公司如 Twitter、Netflix、Amazon 、eBay、淘服务之间互相协调、互相配合, 为用户提供最终价值.

1 微服务架构

微服务架构是一种架构模式, 采用一组服务的方式来构建一个应用, 服务独立部署在不同的进程中, 不同服务通过一些轻量级交互机制来通信, 例如 RPC、HTTP 等, 服务可独立扩展伸缩, 每个服务定义了明确的边界, 不同的服务甚至可以采用不同的编程语言来实现, 由独立的团队来维护[3].

相对于传统的单体应用架构, 微服务架构具有单个服务易于开发、理解和维护; 复杂度可控; 技术选

① 收稿时间:2016-09-18;收到修改稿时间:2016-11-03 [doi: 10.15888/j.cnki.csa.005796]

型灵活; 独立部署; 按需独立扩展等优点[4]. 但是, 采用微服务架构也会引入新的问题, 本文对微服务架构引入的问题进行分析, 提出了一种微服务框架的实现方式, 在框架层面解决服务注册发现、服务容错等共性问题, 使业务系统开发人员专注于业务逻辑实现, 降低开发难度, 提升开发效率.

1.1 微服务框架

目前, 业界流行的微服务框架主要有两个: Spring Boot 和 Dropwizard. SpringBoot 继承了原有 Spring 框架的优秀基因, 简化了开发、配置、部署和监控过程, 帮助开发者快速启动一个 Web 容器. Dropwizard 从前端网页、核心服务、资料库存取到资源监控, 提供了一个轻量级的开发架构, 帮助开发者快速的打造一个 Rest 风格的后台服务, 同时集成 hibernate4、log4j、slf4j、jackson 等开源组件.

2 微服务框架设计

基于微服务架构的应用是分布式系统, 增加了系统设计和实现的难度, 主要体现在以下几个方面:

① 在运行环境中, 微服务实例的网络地址是动态分配的, 微服务运行的实例数量也是动态变化的, 因此, 需要一套服务发现机制, 使服务调用者可以获取正确的服务地址, 同时服务提供者一般以集群方式提供服务, 也引入了负载均衡和健康检查问题.

② 在微服务架构中, 服务之间存在着复杂的依赖关系, 在高并发访问下, 依赖的稳定性影响系统的可用性及性能. 因此需要提供服务容错机制, 当依赖的服务发送故障时, 不会使调用方线程被长时间占用不释放, 避免故障在系统中蔓延.

③ 每个服务独立部署运行, 服务之间的通信是进程间通信, 需要有一个高效的进程间通信机制支撑微服务之间的交互.

④ 每个微服务可以有多个不同的实例, 这使得服务按需动态伸缩成为可能, 在高并发时可以启动同一服务的多个实例, 以提高响应速度. 但服务实例的启停及流量的调度和分发需要一套负载监控和负载均衡组件来管理.

⑤ 一个微服务应用可能由上百个服务构成, 服务可以采用不同语言和框架. 每个服务都有自己的部署、资源、扩展和监控需求. 因此需要提供版本管理和部署组件, 实现微服务的动态部署和无缝升级.

2.1 服务发现

目前服务发现的方式主要有两种: 服务端发现 (server-side discovery) 和客户端发现 (client-side discovery). 这两种方式都通过注册中心方式提供服务注册信息的分布式管理. 当服务上线时, 服务提供者将服务信息注册到注册中心, 并通过心跳维持长链接. 服务调用者通过注册中心寻址, 根据负载均衡算法找到服务. 当服务下线时, 注册中心会发通知给服务客户端. 如图 1 和图 2 所示.

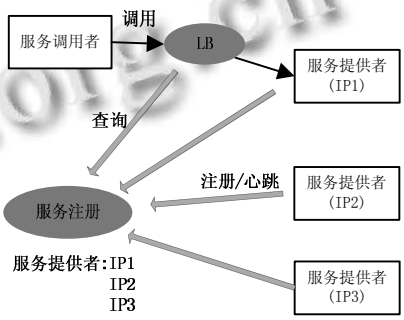

图 1　服务端发现

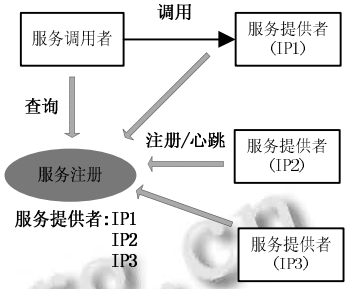

图 2　客户端发现

服务端方式, 所有服务对于服务调用者透明, 但系统中需要维护一个高可用的负载均衡组件, 负载均衡组件在服务调用者和服务提供者之间增加了一跳, 有一定性能开销. 客户端方式架构简单, 扩展灵活, 同时服务消费方和服务提供方之间是直接调用, 没有额外开销, 性能比较好.

2.2 服务容错

在微服务架构中, 服务之间会有错综复杂的依赖关系. 在实际生产环境中, 由于网络连接缓慢、资源繁忙、脱机等原因会造成服务出错或者服务延迟, 如果微服务不能对其依赖的服务的故障进行容错和隔离, 那么该服务本身就处在被拖垮的风险中. 因此在微服务框架设计时需要加入容错措施, 确保某一服务出问题不会影响系统整体可用性. 可用的措施包括[5]:

① 断路器

当某个微服务发生故障后, 通过断路器的故障监控, 向调用方返回一个错误响应, 调用方主动熔断, 以防止线程因调用故障服务被长时间占用不释放, 避免了故障在分布式系统中的蔓延. 如果故障恢复, 电路又能自动恢复.

② 限流

增加限流机制, 限定对微服务的并发访问, 如限制单位时间内的并发数, 对超过这个限制的请求拒绝, 防止在突发流量或被攻击时被击垮.

③ 回退

回退是系统的弹性恢复能力, 是指在熔断或者限流发生的时候, 系统采用何种处理逻辑. 常见的处理策略有直接抛出异常, 也称快速失败; 返回空值或缺省值; 返回备份数据.

2.3 总体架构

通过对微服务框架进行分析, 确定了微服务框架的总体架构, 在框架层面解决负载均衡、服务发现、服务容错、监控报警、进程通信、自动化弹性部署等问题. 架构图如图 3 所示.

图 3 总体架构图

微服务框架包括基础开发框架、服务运行管理和部署监控工具三部分.

基础开发框架: 提供微服务开发的基础组件, 包括展现组件、IOC 组件、持久化组件、序列化组件、服务通信组件、服务监控组件、日志管理组件、缓存组件、嵌入式组件和权限管理组件, 实现微服务开发的通用功能, 整合开源成熟框架, 屏蔽底层技术细节. 其中, 日志管理记录重要的框架层日志、度量和调用链数据, 同时将日志、度量等接口暴露出来, 让业务层能根据需要记录业务日志数据; 服务通信组件提供基于 REST/RPC 的同步请求响应模式和基于消息的异步通信模式.

服务运行管理: 提供微服务运行时服务注册、服务发现、服务路由及限流容错功能. 其中, 服务路由实现请求验证、请求动态路由和静态响应处理等功能; 限流和容错组件, 能够在运行时自动限流和容错, 保护服务. 同时和动态配置相结合, 实现动态限流和熔断.

部署监控工具包括动态配置、部署管理、中间件监控、服务监控、调用链分析功能. 其中, 动态配置组件支持文件方式配置, 集成动态运行时配置, 能够在运行时针对不同环境动态调整服务的参数和配置; 部署管理实现一键部署灰度发布功能.

3 框架实现

3.1 技术架构

微服务框架实现时采用的技术架构如图 4 所示.

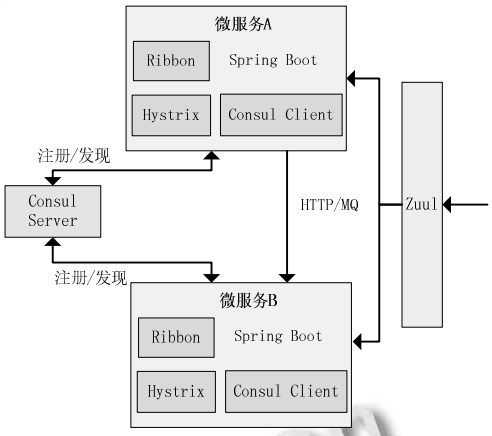

图 4 技术架构图

在微服务框架实现时, 选择了业界开源成熟的框架, 通过框架整合实现微服务开发和运行管理. 其中 Spring Boot 作为微服务的基础开发框架, 提供展现、依赖注入、持久化、嵌入式容器、日志、缓存等基础功能. 集成 Consul 组件实现服务注册发现[6]; Ribbon 组件实现客户端负载均衡; Hystrix 组件实现服务延迟和容错[7]; Zuul 组件提供动态路由, 监控, 弹性, 安全等边缘服务.

3.2 基础架构选型

目前开源的微服务基础框架主要有 Dropwizard 和 SpringBoot. 两个框架在实现技术上存在一定差异[8], 如表 1 所示.

Dropwizard 简单轻量, 但第三方集成力度不足, 社区支持薄弱, 很多关键特性如依赖注入需要开发者自己实现; Spring boot 聚焦于 Spring 应用, 可以借力 Spring 家族体系的其它成员, 完成通信、数据访问等

功能, 体系强大, 社区活跃. Spring Boot 带来了一系列的生态圈的集成, 为微服务开发提供了基础. 因此选择 Spring Boot 作为基础框架.

表 1 Dropwizard 和 SpringBoot 对比

框架	HTTP	REST	JSON	Official integration	Service type
Spring Boot	Tomcat Jetty	Spring	Jackson GSON Json-simple	40+ Official Starter POMs for any purpose	HTTP/REST JMS Message Queuing SOAP
Dropwizard	Jetty	Jersey	Jackson	Hibernate Validator、Guava、Apache HttpClient、Jersey、JDBI、Freemarker、Joda time、Mustache	HTTP/REST

3.3 运行管理框架

服务运行管理引入了四个主流的开源框架, 每个框架相互独立, 实现时需要与基础框架 Spring Boot 集成. 为了实现框架整合, 同时简化开发难度, 引入 Spring Cloud 组件. Spring Cloud 是一个基于 Spring Boot 实现的云应用开发工具, 它为基于 JVM 的云应用开发中的配置管理、服务发现、断路器、智能路由、分布式会话和集群状态管理等操作提供了一种简单的开发方式[9]. 以 Hystrix 组件整合为例, Spring Cloud 提供了 spring-cloud-starter-hystrix 模块, 通过注解的方式轻松使用 Hystrix, 如注解@HystrixCommand 将断路器机制加入到业务处理方法中, 代码如表 2 所示.

表 2 @HystrixCommand 示例

```
.指定 getProducerFallback 为备用方法. 当断路器处于断开状态时,
getProducerFallback 方法将替代 getValue 接受调用.
@HystrixCommand(fallbackMethod = "getProducerFallback")
public ProducerResponse getValue() {
    return restTemplate.getForObject("http:.producer",
            ProducerResponse.class);
}
private ProducerResponse getProducerFallback() {
    return new ProducerResponse(42);
}
```

4 结语

微服务框架已经在国家电网统一应用开发平台 V2.9.0 版本中实现, 该版本经过第三方安全、性能测试, 目前正在项目组试点应用.

本文针对目前流行的微服务框架进行了介绍, 对微服务框架实现时需要解决的问题进行了分析, 设计了微服务的总体架构、实现架构, 重点分析了微服务架构所带来的服务注册发现、服务容错问题. 该微服务架构全面的解决了微服务开发的通用问题, 基于该微服务框架进行业务系统开发, 开发人员只需要关注微服务内部业务功能的开发, 可以降低系统的开发难度, 提高开发效率.

参考文献

1 王磊.解析微服务架构(一)单块架构系统以及其面临的挑战 .http://www.infoq.com/cn/articles/analysis-the-architecture-of-microservice-part-01.[2015-05-11].

2 姜斌.2016 技术将继续颠覆现实 发力引擎之“微服务”.http://www.csdn.net/article/a/2016-05-11/15838064.[2016-05-11].

3 Fowler M, Lewis J. Microservices. http://martinfowler.com/articles/microservices.html.[2014-03-25].

4 肖勤.微服务架构实践经验分享.http:.www.csdn.net/article/2015-08-07/2825412.[2015-08-12].

5 杨波.实施微服务,我们需要哪些基础框架？ http://www.infoq.com/cn/articles/basis-frameworkto-implement-micro-service/.[2015-12-01].

6 HashiCorp. Introduction to Consul. https://www.consul.io/intro/index.html.

7 Netflix. Hystrix: Latency and Fault Tolerance for Distributed Systems. https://github.com/Netflix/Hystrix.

8 Ullah R. Dropwizard vs Spring Boot—A Comparison Matrix. https://dzone.com/articles/dropwizard-vs-spring-boot?utm_source=tuicool&utm_medium=referral.[2015-02-02].

9 Pivotal Software. Spring Cloud. http://projects.spring.io/spring-cloud/.2016.