计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2022,31(3):351−355 [doi: 10.15888/j.cnki.csa.008394]

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

基于 Nginx-F5 的双架构应用并行及流量切换方案①

黄　晨, 柏路平

(中移信息技术有限公司, 深圳 518048)
通信作者: 黄　晨, E-mail: huangchenit@163.com

摘　要: 随着业务量和功能需求的不断增长, 各大业务系统的应用正逐步实现 Spring Boot 架构到 Spring Cloud 微服务架构的升级. 由于版本的变更度较大, 需要经过充分的内部测试与对外联调才能正式上线. 在现有联调环境存在 DMZ 域机器资源有限, 以及尽量减少公网暴露面等要求下, 文章提出了基于 Nginx-F5 的双架构应用并行及流量切换方案, 使测试系统同时具备 Boot 和 Cloud 两种架构应用的对外联调测试功能. 该方案将外部商户的请求按照商户编码、业务类型或省代码等属性标识及一定的百分比进行拆分并转发至微服务应用系统, 使微服务版本的应用获得充分的调测, 为需要进行大版本并行测试的系统扩展和建设提供参考.

关键词: 微服务架构; Nginx; F5; 应用并行; 流量切换

引用格式: 黄晨,柏路平.基于 Nginx-F5 的双架构应用并行及流量切换方案.计算机系统应用,2022,31(3):351–355. http://www.c-s-a.org.cn/1003-3254/8394.html

Dual-architecture Application Paralleling and Flow Switching Scheme Based on Nginx-F5

HUANG Chen, BAI Lu-Ping

(China Mobile Information Technology Co. Ltd., Shenzhen 518048, China)

Abstract: With the continuous growth of business volume and functional requirements, applications of major business systems are gradually upgrading from the Spring Boot architecture to the SpringCloud micro-service architecture. Due to the significant version change, internal testing and external joint testing need to be conducted sufficiently before the applications officially go online. Under the constraints of limited machine resources in the DMZ domain and minimizing the exposure to the public network of the existing joint environment, this study proposes a dual-architecture application paralleling and flow switching scheme based on the Nginx-F5, which provides the test system with both the external joint testing functions of Boot and Cloud architecture applications. The scheme splits the requests of external merchants according to attributes such as merchant code, business type, or province code and a certain percentage and forwards them to the micro-service application system. In this way, the application in the micro-service version can be fully tested, which provides a reference for system expansion and construction that needs large version parallel testing.

Key words: micro-service architecture; Nginx; F5; application paralleling; flow switching

1　引言

Spring Boot 框架[1,2] 因其简洁的配置、内嵌的容器、自动的依赖管理、组件化的功能配置, 能够让开发者快速地搭建项目、编码与新增功能, 加快整个系统开发流程. 以统一支付系统应用为背景, 系统开发使用了 Spring Boot 框架, 快速支撑了支付业务上线. 然而随着业务的发展, 接入统一支付系统的外部商户数量不断增加、业务种类、业务吞吐量、需求功能都在

① 收稿时间: 2021-05-25; 修改时间: 2021-07-01; 采用时间: 2021-07-13; csa 在线出版时间: 2022-01-24

不断的增长, 现有的开发部署架构在应用频繁迭代上线、故障定位方面支持度较低, 也比较难以适应业务量月初、节假日等波动性增长.

微服务[3,4] 是将一个系统拆分成许多较小的、松散耦合的组件和服务的架构, 具有组件共享、故障可控、弹性扩展、易于部署等优点, 近年来, 随着云平台技术与容器管理技术的成熟, 基于微服务框架来构建大型应用系统成为互联网行业的趋势. 为解决支付系统 Spring Boot 架构现存的问题, 开发人员同步开发了 Spring Cloud 微服务版本并准备上线. 变更如此大的版本上线前必然要经过充分的调试, 并且之前的 Spring Boot 架构的支付系统也还要存留一段时间以备出现意外情况时可以回退, 因此联调环境需同时使用这两种架构的系统应用供对外商户调用并进行联调验证.

2 基于 Nginx-F5 的双架构应用并行方案

2.1 Spring Boot 系统联调部署

图 1 是统一支付系统联调环境部署示意图. 在 DMZ 域[5,6] 部署一台 Nginx, 同时开通 443 和 8080 两个端口的公网接入口, 与统一支付联调系统提供的两大类业务相对应, 通过后端的 http 请求接入 API 业务 (对外 8080 端口), 通过前台的 https 请求接入的 H5 业务 (对外 443 端口), 也有部分商户通过 https 对接 API 业务, 由于 H5 与 API 接口处理支付请求的流程不同, 将它们开发成不同的应用组件, 并以不同的请求 path 加以区分.

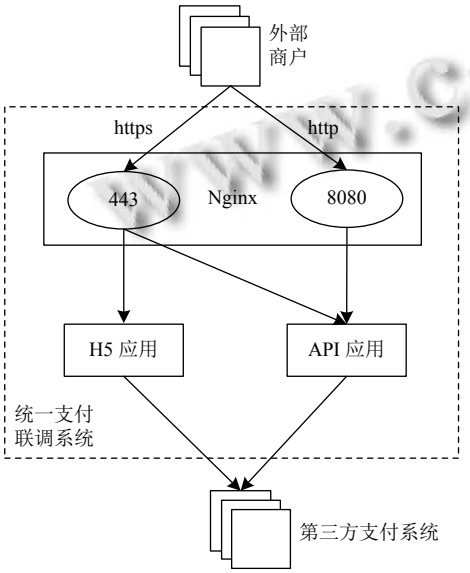

图 1 Spring Boot 版本的统一支付系统联调部署

为了能够保证新开发的 Spring Cloud 架构应用的质量, 有必要将其部署在联调环境, 经过外部商户的测试报文来验证其所有功能. Spring Cloud 架构的系统如何与原来的 Boot 架构的系统在联调环境并存需要考虑如下问题: 首先, Spring Cloud 版本是统一支付系统内部的优化版本, 整个对外联调过程应该对外部商户透明, 即不应让外部商户感知到统一支付系统是调测的哪个架构的应用; 其次为保证内部系统安全, 应尽量减少公网暴露面, 还有 DMZ 域的机器资源使用已经饱和, 新部署的 Cloud 支付系统只能跟其他系统的应用共用机器.

2.2 基于 Nginx-F5 的双架构应用联调并行方案

在 Nginx 作为公网接入口, 并充分考虑上述限制问题的基础上, 本文提出一种基于 Nginx-F5[7–9] 的双架构 (大版本) 应用的并行方案, 既不增加对外联调的端口又可以使两种 Boot 和 Cloud 架构的应用得到高效的调测, 如图 2 所示.

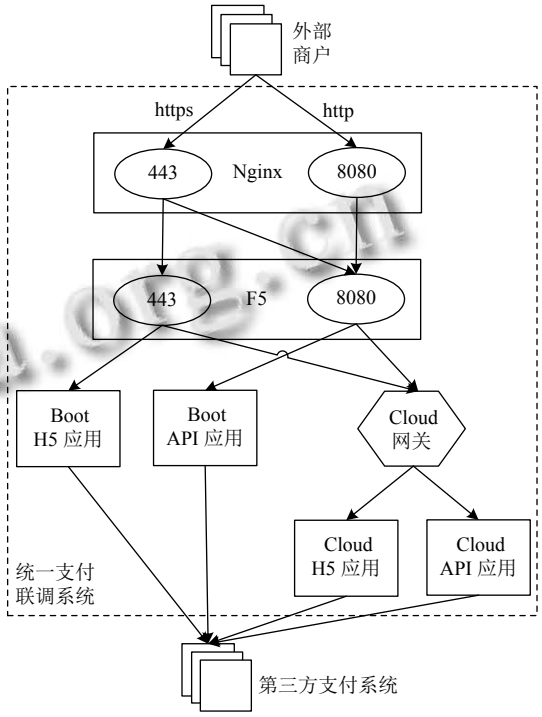

图 2 基于 Nginx-F5 的双架构应用系统并行方案

与之前的部署架构图相比, 除了增加 Cloud 架构的应用外, 还新增了一套 F5 负载均衡器. 该 F5 是一个操作系统模拟器, 需要单独一台虚拟机来安装 F5 的操作系统, 由于 DMZ 域机器资源有限, 原来的 Nginx 宿

主机还安装有很多其他的应用, 不能将其铲除, F5 模拟器可以安装在机器资源充足的内网核心域. 将 F5 模拟器部署于原来的 Nginx 之后, 支付联调应用之前. 主要使用 F5 的 7 层转发功能, 细粒度地控制外部商户的模拟报文在 Boot 系统与 Cloud 系统之间切换, 在不影响线上 Boot 架构系统不停迭代新需求的同时, 可以随时验证新的 Cloud 架构系统的基本功能.

2.3 与纯 Nginx 的双架构应用联调并行方案比较

如不使用 F5 模拟器, 直接使用原来的 Nginx 的添加 Lua 脚本模块或者使用 OpenResty 版本的 Nginx, 也能实现根据报文体中的某个属性标识转发请求, 但由于 Nginx 流量转发方式的变更需要修改配置文件或者加载的 Lua 脚本并且重载才能生效, 这个重载机制并不是每次配置变更都会生效, 并且没生效也没有提示. 这时只能强制杀掉 Nginx 的主进程后, 重新启动使变更的配置生效. 而且 Nginx 的健康检查机制是通过将后端服务器成员状态的检查结果打印到日志文件反映, 当有后端成员挂掉也不能及时发现. 而 F5 模拟器具有可视化的 Web 操作界面, 后端服务器成员的启停状态能够很直观地看到, 也很方便将后端成员的状态设置开启与关闭, 其提供的负载规则 irule[10], 可以自定义各种基于报文头和报文体的流量转发策略, 并且各种变更都是实时生效, 且不会使正在活动的连接断开.

因为生产环境的 Boot 架构的应用还在不断的迭代新需求, 而新的 Cloud 架构的应用也需要调测基本的功能, 并且在某个时刻追加 Boot 架构新迭代的全部需求, 本文基于 Nginx-F5 的并行联调架构在下面的 F5 加载 irule 规则的流量切换方法下, 能够在不改变原有的业务入口的条件下, 提高两种架构应用的联调效率, 更好地支撑生产系统的不断升级优化.

3 基于 F5 irule 规则的双架构应用切换方法

图 2 的双架构应用系统并行方案中, F5 模拟器开启 443 和 8080 两个端口的虚拟服务, 分别作为原接入 Nginx 的 443 和 8080 服务的各自的后端服务器. 因为 Nginx 在 443 端口已经卸载了 https 证书, 发送到 F5 的 443 端口的已经是 http 协议的报文, 所以在 F5 的 443 端口的虚拟服务不需要再安装证书. 设置 Nginx 的 443 端口的以 https 服务进来的 API 请求固定地转发到 F5 的 8080 端口, 这样经过 F5 的 443 端口的业务就只有 H5 类型, 在 F5 的 443 端口做 7 层负载分发时就不用考虑 API 类型的业务, 简化了 F5 的流量控制过程.

统一支付系统的外部商户各自的业务特点不尽相同, 因此有必要将某个或者某些商户的业务请求分离出来, 并单独发送至不同后端应用系统.

3.1 F5 基于报文属性值的流量分发规则

因为外部商户的发送的请求都是结构化的报文, 即以 XML 和 JSON 格式存在, XML 中是属性及属性值的集合, 而 JSON 是键/值的集合, 可以用属性值或者键值的具体取值来代表一类商户请求. 因此可以通过 F5 的负载规则 irule 编写处理逻辑来改变请求分发方式.

irule 是 F5 功能强大的扩展组件, 通过 TCL (tool command language) 语言, 可以编写基于事件触发的代码段, 对接收报文的关键字或分发需求进行分析, 以改变进入 F5 的网络请求的默认分发方式, 达到流量按需切换的目的.

结合外部商户报文的特点, 编写了高效的基于报文属性值的 irule 分发规则, 如规则 1.

规则 1. 基于报文属性值的流量分发规则

1) 设置属性标识列表 iKeyList.
2) 当接收到请求并解析到报文体时, 属性标识列表中的元素按先后顺序分别遍历报文体, 当搜索到属性标识元素 iKey 时就结束遍历.
3) 按照属性值的位数从请求报文中截取 iKey 对应的属性值 iValue.
4) 设置流量切换商户标识列表 sList.
5) 在 sList 搜索报文的属性取值 iValue, 如果搜索到则说明该请求报文是由该商户发出, 就将该笔报文发送到 Cloud 应用系统. 以验证某个商户的微服务版本业务是否正常, 如果没有搜索到则发往默认的 Boot 系统.

当规则 1 的报文属性取某个具有业务意义的值时, 就可以把报文分成两类, 分别发送到不同的系统, 下面以报文属性取 sysNo 的值为例, 描述如何根据外部商户的商户编码将支付请求报文在 Boot 系统和 Cloud 系统之间切换. 图 3 为 irule 按照报文属性值的流量分发的处理流程图.

设置属性标识列表的原因是除了要兼容 XML 与 JSON 格式的请求报文外, 还要考虑特殊字符是否经过 url 编码, 统一支付系统的外部商户数量接近 100 个, 为了管理方便, 每个商户都分配有一个四位数值的编码, 作为一个汇集系统应当兼容不同的报文格式. 对于商户编码 sysNo 这一个属性来说, 属性标识的列表有 4 个元素, 分别是 sysNo>、sysNo%22、sysNo\"、sysNo%3E 四个字符串, 只有搜索到具体是

列表中的哪一个元素后, 才能在报文中按属性值位数截取属性的值. 根据不同测试需求, 修改商户标识列表 sList 的元素取值, 可以实现只把一个商户或者多个商户的流量同时切换到微服务系统, 如 sList={0069, 0075, 0089}表示把 sysNo 取为 0069、0075 与 0089 对应 3 个商户的流量全部切换到微服务系统. 当把该 irule 与 F5 的 443 或者 8080 端口绑定时, 对应的虚拟服务就按照 irule 定义的转发逻辑分发流量, 解除绑定则直接转发到虚拟服务配置的默认 default 服务组.

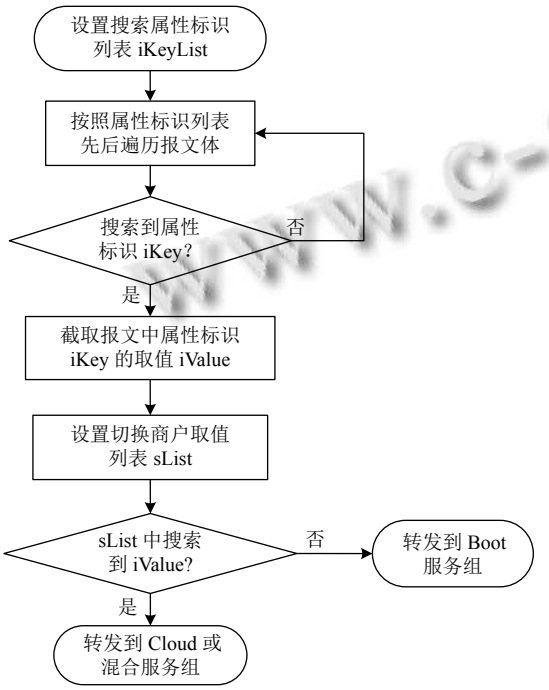

图 3 基于报文属性值的流量分发处理流程

3.2 F5 基于报文属性值的比例分发方法

使用 F5 的 irule 规则除了按照外部商户编码来分发流量之外, 还可以按照流量的比例来进行分发. 很多情况下 Nginx 和 F5 与后端服务器之间是一对多的关系, 即负载均衡器将接收到的请求分发到多个后端服务器, 通常一个虚拟服务具有一个默认服务组, 服务组里面的各个成员默认都是按照轮询的方式分发流量, 各成员接收到的请求数都大致相同. 也可以设置服务组的负载均衡模式为加权轮询, 组内的每个成员都自带有权重值, 每个成员接收到的流量比例为该成员的权重值在组内所有成员权重和的比重.

结合 F5 的加权轮询负载模式, 以及自定义 irule 脚本筛选出的特定属性报文, 可以实现两种不同粒度的比例分发, 如图 4 所示. 将 Boot 应用和 Cloud 应用添加到同一个服务组, 称作混合服务组, 这个服务组的负载均衡类型按成员权重分发, 通过修改 Boot 与 Cloud 应用的权重则可控制转发到微服务应用的流量比例在 0–100% 之间变化. 一种方式是直接将混合服务组添加到 F5 虚拟服务的默认服务组, 则经过该虚拟服务的所有请求都将按照比例分发, 属于较粗的比例控制.另一种是将混合服务组与请求报文中具体的属性标识相结合, 在规则 irule 中将外部商户编码属于 sList 集合的报文发往混合服务组 (对应图 3), 即可将经过该虚拟服务的某个商户或某几个商户的流量按照一定的比例分发到微服务应用, 这种流量分发控制更为精细.

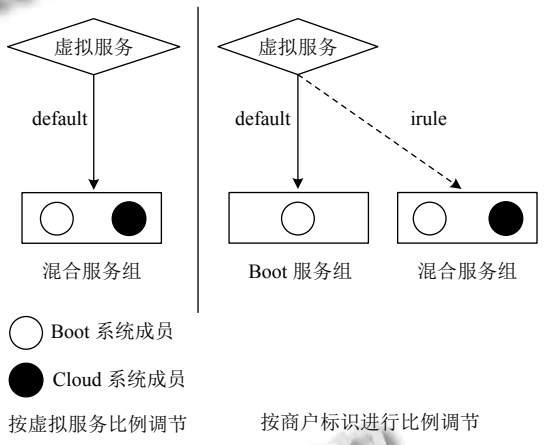

图 4 两种粒度的流量比例分发控制

4 实验结果验证

为验证在本文架构下流量切换方法的正确性, 采用的方法是模拟各外部商户系统发送测试报文到联调系统的 Nginx 入口, 查看报文经过 F5 的 7 层负载后, 最终是分发到了 Boot 系统还是 Cloud 系统, 是否与 F5 虚拟服务加载的 irule 定义的分发规则一致.

实验 1: 验证规则 1 基于报文属性值流量分发对于单个属性取值的应用效果, 将 sList 设置为{0230}, 即只将 0230 商户的请求切换到 Cloud 系统, 并将该 irule 规则与 F5 的 8080 虚拟服务绑定, 然后模拟 0230, 0280, 0069 三个商户分别发送 1 000 笔支付请求报文, 通过 F5 提供的日志以及查看应用日志统计确认, 0280, 0069 发送的 2 000 笔报文都发送到了 F5 的 API 虚拟服务的默认 Boot 应用系统, 而 0230 商户的这 1 000 笔报文都 Cloud 应用系统, 说明该规则对单个商户的切

换有效, 正确率达到了 100%.

实验 2: 验证该分发规则对于多个属性取值时的应用效果, 将 sList 的取值设置为{0230 0280 0069}, 模拟这 3 个商户各发送 1 000 笔报文, 经确认这 3 000 笔报文都发送到了 irule 指定的 Cloud 服务组, 并且无其他商户的报文发送到该服务组, 说明该规则对多个商户的同时切换也同样有效.

实验 3: 验证基于报文属性值的比例分发方法, 保持 sList 的取值设置为{0230 0280 0069}不变, 修改 irule 规则, 将查找到切换商户取值列表中的元素后转发的服务组更换为混合服务组, 该服务组里面既有 Boot 应用成员, 又有 Cloud 应用的成员, 将 Boot 应用成员的权重值设置为 9, Cloud 应用成员的权重值设置为 1, 这样 Cloud 应用的占比为 10%, 模拟这 3 个商户各发送 1 000 笔报文, 发现 Cloud 应用接收到 302 笔报文, 并且全部来自 0230, 0280, 0069 这 3 个商户, Boot 应用收到 2 698 笔来自这 3 个商户的报文, Cloud 应用接收到的报文比例大约 10%, 说明该规则与加权轮询的负载均衡算法结合使用能够达到更细粒度的流量比例控制.

表 1 是以上 3 个实验的流量分发数据表, 实验结果说明本文提出的 Nginx-F5 架构下的流量切换方法, 在规则属性设置为单个和多个值的情况下, 都能够将流量正确地分发到报文属性值对应 Boot 系统和 Cloud 系统, 能够很好地支撑双架构应用系统的并行联调.

表 1 3 种实验的不同商户报文接收情况

实验序号	sList 取值	请求商户号	Boot系统接收量	Cloud系统接收量
1	0230	0230	0	1 000
		0280	1 000	0
		0069	1 000	0
2	0230	0230	0	1 000
	0280	0280	0	1 000
	0069	0069	0	1 000
3	0230	0230	900	100
	0280	0280	899	101
	0069	0069	899	101

5 总结

为使统一支付系统能够透明地对众多外部商户同时提供 Spring Boot 架构的应用和 Spring Cloud 架构应用的联调, 本文提出了基于 Nginx-F5 的 Spring Boot 和 Spring Cloud 两种架构应用的并行方案, 以及一种基于报文属性值的流量分发规则, 能够将接入 F5 虚拟服务满足某个属性值的请求报文发送到 Cloud 应用系统, 而不是默认的 Boot 应用系统. 将该流量分发规则与加权轮询的服务组联合使用, 可实现更细粒度的百分比例分发, 如只将某一个或某几个外部商户的约 5% 的流量转发到 Cloud 系统. 采用这种双架构应用的并行联调方案, 能够做到既不影响生产 Boot 系统的不断需求迭代更新, 又能使新的 Cloud 系统得到充分完整的对外调测, 有效地支撑生产系统不断升级优化.

本文中的 Boot 及 Cloud 两种系统可理解为同一系统的不同版本, 本文所提出的并行架构及流量切换方法也可用于其他任何系统多版本并行及联调. 流量分发规则中的报文属性可变更为其他应用系统有意义的分类标识, 如用户归属省、业务类型等, 可推广到其他系统的建设或者优化演进.

参考文献

1 吕宇琛. Spring Boot 框架在 Web 应用开发中的探讨. 科技创新导报, 2018, 15(8): 168, 173.

2 王永和, 张劲松, 邓安明, 等. Spring Boot 研究和应用. 信息通信, 2016, (10): 91–94.

3 顾芒芒, 吴铭程. 基于 Spring Cloud 实现任务调度微服务化的设计与实现. 工业控制计算机, 2021, 34(3): 117–119.

4 冯志勇, 徐砚伟, 薛霄, 等. 微服务技术发展的现状与展望. 计算机研究与发展, 2020, 57(5): 1103–1122.

5 檀斌, 张洁, 何琳, 等. 多网段下视频会议互联互通的研究. 电脑编程技巧与维护, 2020, (7): 148–150.

6 刘景林. 基于 GNS3 模拟器的 Cisco ASA 防火墙技术应用仿真实现. 河北软件职业技术学院学报, 2018, 20(3): 12–16.

7 赵凯. 一种基于 Nginx 反向代理机制的微服务负载均衡方法分析. 无线互联科技, 2020, 17(16): 140–141.

8 石浩波. Nginx 工作实践. 电脑知识与技术, 2015, 11(14): 239–240.

9 赵中英, 李斌, 王敏. F5 负载均衡综述. 现代信息科技, 2019, 3(2): 60–61.

10 姚峰. 利用 Load Balance 对流媒体系统进行优化和扩展 [硕士学位论文]. 上海: 复旦大学, 2006. [doi: 10.7666/d.y1022186]