

新工科建设之路·计算机类规划教材



微服务分布式架构基础与实战

——基于 Spring Boot + Spring Cloud



张方兴 编著

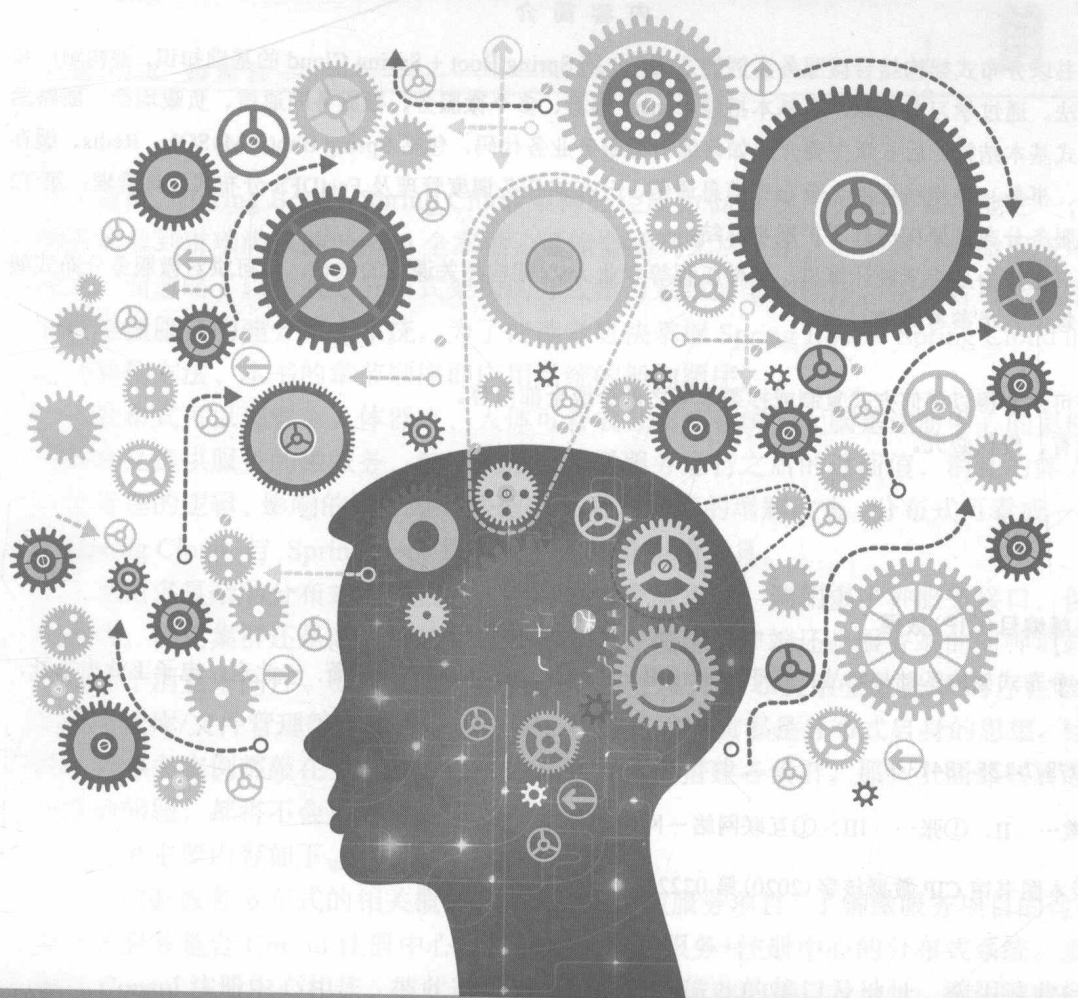


中国工信出版集团



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
<http://www.phei.com.cn>

新工科建设之路·计算机类规划教材



微服务分布式架构基础与实战

——基于 Spring Boot + Spring Cloud

张方兴 编著

电子工业出版社
Publishing House of Electronics Industry
北京·BEIJING

内 容 简 介

本书以分布式架构结合微服务实例的方式,介绍 Spring Boot + Spring Cloud 的基础知识、架构顺序和操作方法。通过学习前 5 章,可基本搭建 Consul 集群、多个微服务、微服务间通信、负载均衡、断路器的分布式基本结构;后 6 章主要介绍如何编写微服务业务代码,包括 Spring Boot、MySQL、Redis、缓存一致性、事务、异步线程池、分布式消息通信、分布式任务调度管理及 FastDFS 分布式文件管理;第 12 章对微服务分布式架构进行了扩展与总结。

本书可作为高等学校计算机、软件工程等专业 Java 架构相关课程的教材,也可供对微服务分布式架构感兴趣的人员参考阅读。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有,侵权必究。

图书在版编目(CIP)数据

微服务分布式架构基础与实战:基于 Spring Boot+Spring Cloud/张方兴编著. —北京:电子工业出版社, 2020.3

ISBN 978-7-121-38413-4

I. ①微… II. ①张… III. ①互联网络—网络服务器 IV. ①TP368.5

中国版本图书馆 CIP 数据核字(2020)第 022234 号

责任编辑:章海涛

文字编辑:张鑫

印刷:三河市华成印务有限公司

装订:三河市华成印务有限公司

出版发行:电子工业出版社

北京市海淀区万寿路 173 信箱

邮编:100036

开本:787×1092 1/16 印张:17

字数:436 千字

版次:2020 年 3 月第 1 版

印次:2020 年 3 月第 1 次印刷

定 价:59.00 元

凡所购买电子工业出版社图书有缺损问题,请向购买书店调换。若书店售缺,请与本社发行部联系,联系及邮购电话:(010)88254888, 88258888。

质量投诉请发邮件至 zltz@phei.com.cn, 盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式:192910558(QQ 群)。

目前，Spring Boot + Spring Cloud 架构已经成为 Java 程序员的必备技能之一，刚开始学习时看到琳琅满目的 Spring 全家桶，可能会感到无从下手。如果只了解微服务中的各知识点，而忽略了以微服务分布式架构的方式学习系统的架构顺序，初学者可能就不知道如何使用微服务构建分布式系统。为了使读者更快掌握 Spring Boot + Spring Cloud 的基础知识与架构方法，本书的章节顺序即应用系统的架构顺序。

分布式可以理解为人体的器官，人体可看成分布式系统，大脑是注册中心的集群，四肢与器官是提供服务的微服务，前进的距离是微服务运行之后的返回值，消耗的体力是微服务中处理的逻辑，影响的记忆是某些微服务对数据库的增删改查。分布式可看成一种思想，而 Spring Cloud 与 Spring Boot 是实现了这种思想的工具。

无论多复杂的分布式应用程序，整合多少个服务器，调用多少种服务接口，使用多少种协议，使用集群还是高可用的何种架构方式，使用客户端还是服务端的何种负载均衡，使用哪个消息中间件、哪个数据库集群，使用搜索引擎/非关系型数据库/时序性数据库/关系型数据库/文件管理等多少种存储介质，其分布式本质都是分布式自身的思想。建议读者动手将本书实例都敲在 IDE 中，在 Linux 服务器上搭建各集群，那么上面那些看起来颇具难度的问题，都将不会是难题。

本书主要内容如下。

介绍微服务分布式的相关概念，搭建第一个微服务项目，了解微服务项目的运行过程。通过微服务整合 Consul 注册中心，搭建第一个微服务+注册中心的分布式系统。多个微服务与 Consul 注册中心相连，彼此通信，微服务获得彼此的接口及地址，调用彼此的接口与服务。然后无可避免地需要处理彼此通信时报错的情况，以免单一微服务无法正常提供服务，导致整个分布式系统的瘫痪。此时采用 Ribbon 客户端负载均衡方案，依靠多台服务器部署多个相同微服务项目，以提高系统的性能。在分布式通信不足以解决全部报错问题时，可选择 Hystrix 进行更精细划分，保证在任何一台微服务出现问题时，系统整体仍然能正常运行。

至此初步搭建了分布式系统，然后开始增加微服务的增删改查等业务功能。在系统处理增删改查过程的同时，还需要事务功能的支撑与管理。在初步增删改查后，依靠微服务的缓存增加微服务的性能，同时需要确保 Redis 与 MySQL 之间的增删改查一致性。另外，如果有特殊业务需求，可由分布式消息通信彼此协作、沟通、处理。

在处理基本业务后，还会有定时任务等业务需求，此时需要微服务的任务调度进行处理。而单节点任务调度的微服务可能会有宕机或重复执行等相关问题，只有使多台微服务同时进行任务调度，且彼此协同的情况下，才能解决此类问题。这时需要 Quartz 分布式任务调度解决多个任务调度间彼此协同、相互管理等问题。若有文件上传、下载等相关需求，需要使用微服务的文件上传管理。单节点文件上传可能存在磁盘空间不足

或不易管理等问题，需要 FastDFS 分布式文件管理以解决多台文件管理服务器彼此协同、磁盘扩容等问题。

在对整体分布式微服务编程后，需要进行部署，第 12 章介绍如何部署微服务分布式项目，以及架构方案、设计方式、编程框架、网关等。

本书适合作为高等学校计算机、软件工程等相关专业 Java 架构课程的教材，也可供对微服务分布式架构感兴趣的人员参考阅读。

本书由张方兴编著。本书即将付梓，还要感谢张鑫编辑持之以恒的支持与帮助。编者微信公众号为北漂程序员的吐槽人生（微信号：beipiaochengxuyuan），读者有任何意见与建议，可随时沟通。

非常感谢阅读本书，学无止境，与君共勉。

由于编者水平有限，加之编写时间仓促，书中难免出现错误与不足之处，欢迎读者批评指正。

编者

2020 年 1 月

目 录

CONTENTS

第 1 章 微服务分布式架构设计原理	1
1.1 Java Web 应用程序的发展历史	1
1.2 微服务分布式	2
1.2.1 Spring Boot 微服务的定义和特点	3
1.2.2 Spring Boot 的职场导读	3
1.2.3 Spring 部分内容	4
1.2.4 微服务的拆分	6
1.3 【实例】微服务工程 Hello World	7
1.3.1 实例背景	7
1.3.2 创建 Maven Project	7
1.3.3 使用空 Maven Project 模板	7
1.3.4 编辑 Maven 坐标定位及工程名	8
1.3.5 检查 Maven 目录结构	9
1.3.6 编写 Pom 文件	10
1.3.7 Spring Boot 依赖包的导入	12
1.3.8 编写 Spring Boot 启动类	14
1.3.9 编写 Spring Boot 接口	14
1.3.10 当前项目结构	14
1.3.11 启动工程	15
1.3.12 Spring Boot 初始化启动后	16
1.3.13 实例易错点	16
1.4 Spring Boot 启动类扫描 Bean	18
1.4.1 @SpringBootApplication 注解	18
1.4.2 @ComponentScan 注解	20
1.4.3 Spring Boot 扫描其他包下文件	20
1.5 【实例】将端口号改成 9090	21
1.5.1 实例背景	21
1.5.2 创建 application.properties 资源配置文件	22
1.5.3 增加资源配置文件中的配置信息	23
1.5.4 运行结果	23
1.5.5 实例易错点	23
1.6 YAML 文件	24
1.6.1 YAML 文件简介	25
1.6.2 YAML 文件的书写格式	25
1.7 【实例】使用 YAML 配置文件	25
1.7.1 实例背景	25
1.7.2 原 properties 文件	25
1.7.3 转换格式后的 YAML 文件	26
1.7.4 实例易错点	26
1.8 【实例】通过单配置文件让工程适应多应用场景	27
1.8.1 实例背景	27
1.8.2 更改 application.yml 文件	27
1.8.3 更改启动类	27
1.8.4 输入启动参数	29
1.8.5 运行结果	30
1.8.6 实例易错点	30
1.9 【实例】通过多配置文件使工程适应多应用场景	31
1.9.1 实例背景	31
1.9.2 新建 SIT 和 UAT 环境所需资源配置文件	31
1.9.3 新建系统资源配置文件	31
1.9.4 编写启动类	31
1.9.5 当前项目结构	32
1.9.6 运行结果	32
1.10 微服务配置权重	32
1.10.1 资源配置信息类型的权重	32
1.10.2 资源配置文件类型的权重	33
1.10.3 资源配置文件存在位置与权重解读	33
1.11 本章小结	34
1.12 习题	34
第 2 章 分布式的注册中心	35
2.1 注册中心	35

2.1.1 Eureka 与 Consul 的区别	35	相关 K/V 存储	56
2.1.2 Consul 的相关术语	37	2.5.6 调用 MyConfig.java 中的参数	57
2.1.3 Consul 的安装	37	2.5.7 在启动类引用相关配置	57
2.2 Consul 的常用命令	37	2.5.8 当前项目结构	58
2.2.1 consul agent -dev	38	2.5.9 运行结果	58
2.2.2 consul -members	39	2.5.10 实例易错点	60
2.2.3 consul leave	40	2.6 本章小结	60
2.2.4 agent 命令的常用配置参数	40	2.7 习题	60
2.2.5 HTTP API	41	第 3 章 分布式的通信	61
2.3 【实例】创建第一个微服务分布式项目	42	3.1 分布式通信	61
2.3.1 实例背景	42	3.1.1 Spring Cloud Feign	61
2.3.2 搭建 Consul 集群	42	3.1.2 Swagger	61
2.3.3 创建微服务工程编写相应依赖文件	45	3.2 【实例】微服务集成 Swagger	62
2.3.4 Spring Cloud 和 Spring Boot 的版本对应关系	46	3.2.1 实例背景	62
2.3.5 编写微服务 YAML 资源配置文件	46	3.2.2 编写 Swagger 依赖	62
2.3.6 编写微服务启动类注册到 Consul 上	48	3.2.3 编写 Swagger 配置	63
2.3.7 当前项目结构	48	3.2.4 编写接口与接口处的 Swagger 配置	64
2.3.8 运行结果	49	3.2.5 当前项目结构	66
2.3.9 实例易错点	50	3.2.6 运行效果	66
2.4 【实例】通过代码获取 Consul 中的服务信息	51	3.2.7 实例易错点	70
2.4.1 实例背景	51	3.3 【实例】Feign 调用微服务接口	72
2.4.2 编写获得其他注册服务的代码	52	3.3.1 实例背景	72
2.4.3 运行结果	53	3.3.2 引入相关配置信息	73
2.4.4 实例易错点	53	3.3.3 编写 Feign 客户端	73
2.5 【实例】Spring Cloud 操作 Consul 的 K/V 存储	54	3.3.4 编写调用	75
2.5.1 实例背景	54	3.3.5 编写启动类	76
2.5.2 添加依赖	54	3.3.6 当前项目结构	76
2.5.3 利用 Consul 的 UI 界面添加 K/V 存储	54	3.3.7 运行结果	77
2.5.4 编写 YAML 资源配置文件对应 K/V 存储	55	3.3.8 实例易错点	77
2.5.5 编写 MyConfig.java 文件对应		3.4 【实例】Feign 的拦截器	78
		3.4.1 实例背景	78
		3.4.2 在 cloud-admin-8084 工程中增加拦截器	78
		3.4.3 当前项目结构	79
		3.4.4 运行结果	79
		3.4.5 实例易错点	80

3.5 Feign 的配置	81	第 5 章 分布式的断路器	99
3.5.1 传输数据压缩配置	81	5.1 断路器	99
3.5.2 日志配置	82	5.1.1 为什么需要断路器	99
3.5.3 超时配置	83	5.1.2 Hystrix	99
3.6 【实例】Feign 的降级回退处理		5.1.3 Hystrix 解决的问题	100
——Feign 的 Fallback 类	84	5.1.4 Hystrix 如何解决问题	100
3.6.1 实例背景	84	5.2 【实例】Hystrix 断路器的降级	
3.6.2 在资源配置文件中开启 Feign 内置		回退	101
的 Hystrix 权限	84	5.2.1 实例背景	101
3.6.3 编写 Fallback 降级类	84	5.2.2 编写相关 Pom 文件	101
3.6.4 Service 整合 Fallback 降级类	84	5.2.3 编写 application 资源配置文件	101
3.6.5 当前项目结构	85	5.2.4 编写 Ribbon 配置类	102
3.6.6 运行结果	85	5.2.5 编写启动类	102
3.7 【实例】Feign 的降级回退处理		5.2.6 编写 Service 类	103
——Feign 的 Fallback 工厂	86	5.2.7 编写 Controller 类	103
3.7.1 实例背景	86	5.2.8 当前项目结构	104
3.7.2 编写 Fallback 降级工厂	86	5.2.9 运行结果	105
3.7.3 整合 Fallback 降级工厂	87	5.2.10 实例易错点	106
3.7.4 实例易错点	87	5.3 Hystrix 线程池	108
3.8 本章小结	88	5.3.1 Hystrix 断路器注解式的命令	
3.9 习题	88	配置	109
第 4 章 分布式的客户端负载均衡	89	5.3.2 Hystrix 断路器的注解式线程池	
4.1 负载均衡	89	配置	111
4.1.1 传统服务器端负载均衡	89	5.3.3 Hystrix 断路器注解式的整体	
4.1.2 Ribbon 客户端负载均衡	89	定制配置	112
4.2 【实例】Feign 整合 Ribbon 分发		5.3.4 Hystrix 断路器资源配置式的	
请求	90	整体定制配置	113
4.2.1 实例背景	90	5.4 【实例】Hystrix 断路器的请求	
4.2.2 编写 cloud-book-8086 启动类与		缓存	114
配置类支持 Ribbon	91	5.4.1 实例背景	114
4.2.3 Service 和 Controller	92	5.4.2 通过 Filter 初始化 Hystrix	
4.2.4 当前项目结构	94	上下文	114
4.2.5 运行效果	95	5.4.3 让启动类扫描 Filter 过滤器	116
4.2.6 实例易错点	96	5.4.4 编写 Controller 的 Helper 类	116
4.3 Ribbon 的负载均衡策略配置	97	5.4.5 编写 Controller 类	118
4.4 本章小结	98	5.4.6 当前项目结构	118
4.5 习题	98	5.4.7 运行结果	119
		5.4.8 销毁 Hystrix 的请求缓存	121

5.4.9 实例易错点	121	6.3.2 增加新的服务接口	141
5.5 【实例】Hystrix 的请求合并	123	6.3.3 增加新的服务实现	141
5.5.1 实例背景	123	6.3.4 增加新的调用	142
5.5.2 增加@HystrixCollapser 请求合并 修饰的函数	124	6.3.5 当前项目结构	142
5.5.3 Controller 中调用请求合并函数	126	6.3.6 运行程序查看异步线程池效果	142
5.5.4 当前项目结构	126	6.3.7 实例易错点	143
5.5.5 运行结果	127	6.4 【实例】优化异步线程池	143
5.5.6 实例易错点	128	6.4.1 实例背景	143
5.6 【实例】Hystrix 的可视化监控	129	6.4.2 创建初始化线程池配置类	143
5.6.1 实例背景	129	6.4.3 更改无返回值的异步线程池 Service 实现类	145
5.6.2 Hystrix 可视化监控的依赖	129	6.4.4 运行程序查看异步线程池效果	145
5.6.3 Hystrix 可视化监控的启动类	129	6.4.5 实例易错点	146
5.6.4 被监控的微服务增加响应地址	130	6.5 【实例】优雅停止异步线程池	146
5.6.5 当前项目结构	131	6.5.1 实例背景	146
5.6.6 运行结果	132	6.5.2 何为“优雅”	146
5.6.7 实例易错点	134	6.5.3 修改原 Config 配置类	147
5.7 本章小结	135	6.5.4 修改原 Controller 控制层	148
5.8 习题	135	6.5.5 当前项目结构	149
第 6 章 微服务的异步线程池	136	6.5.6 优雅停止异步线程池的执行 效果	150
6.1 异步线程池	136	6.5.7 实例易错点	152
6.1.1 异步线程池特点	136	6.6 @Enable*注解	152
6.1.2 常见的线程池	136	6.7 本章小结	152
6.2 【实例】创建无返回值异步线 程池	137	6.8 习题	153
6.2.1 实例背景	137	第 7 章 微服务整合持久化数据源	154
6.2.2 编写 Pom 文件	137	7.1 spring-data	154
6.2.3 编写 Spring Boot 启动类	138	7.1.1 ORM 规范	154
6.2.4 编写异步线程池任务接口与 实现	138	7.1.2 JPA、Hibernate、spring-data-jpa 之间的关系	155
6.2.5 编写外部可调用接口	139	7.1.3 安装 MySQL	155
6.2.6 当前项目结构	140	7.2 【实例】Spring Boot 整合 MyBaits 注解式编程	156
6.2.7 运行程序查看异步线程池效果	140	7.2.1 实例背景	156
6.2.8 实例易错点	141	7.2.2 添加 Pom 文件	156
6.3 【实例】创建有返回值异步 线程池	141	7.2.3 编写 application 资源配置文件	157
6.3.1 实例背景	141	7.2.4 编写 dao 层	157

7.2.5	编写访问接口	159	只读	175
7.2.6	当前项目结构	160	8.1.6	@Transactional 声明式事务指定异常
7.2.7	运行效果	160		176
7.2.8	实例易错点	160	8.2	【实例】Spring Boot 整合声明式事务
7.3	@Mapper 注解详解	162		176
7.3.1	@Mapper 和 XML 形式的对应关系	162	8.2.1	实例背景
7.3.2	MyBatis 的注解式编程多表查询	162		176
7.3.3	MyBatis 的注解式编程分页查询	163	8.2.2	整合 @Transactional 的 Service 层编写
7.3.4	注册 DataSource 数据源	165		177
7.4	【实例】Spring Boot 整合 spring-data-jpa	166	8.2.3	整合 @Transactional 的 Controller 层编写
7.4.1	实例背景	166		178
7.4.2	添加 Pom 文件	166	8.2.4	当前项目结构
7.4.3	添加资源配置文件中的相关信息	166		179
7.4.4	添加实体类映射	167	8.2.5	运行结果
7.4.5	添加 JPA 的 dao 层	168		179
7.4.6	添加 Controller 控制层查询 JPA 的 dao 层	169	8.2.6	实例易错点
7.4.7	当前项目结构	170		180
7.4.8	运行结果	170	8.3	本章小结
7.4.9	实例易错点	170		181
7.5	本章小结	171	8.4	习题
7.6	习题	171		181
第 8 章	微服务事务	172	第 9 章	微服务的缓存与分布式的消息通信
8.1	@Transactional 注解	172		182
8.1.1	@Transactional 声明式事务的传播行为	173	9.1	Redis
8.1.2	脏读、不可重复读与幻读	173		182
8.1.3	@Transactional 声明式事务的隔离级别	174	9.1.1	BSD 协议
8.1.4	@Transactional 声明式事务的超时时间	175		182
8.1.5	@Transactional 声明式事务的	175	9.1.2	Java 与 Redis 的历史
				183
			9.1.3	Spring Data Redis
				183
			9.2	【实例】微服务整合 Spring Data Redis 增删改查
				184
			9.2.1	实例背景
				184
			9.2.2	编写 application.properties 资源配置文件
				184
			9.2.3	配置 RedisTemplate 模板
				185
			9.2.4	编写操作 Redis 的工具类
				186
			9.2.5	编写实体类及接口调用
				188
			9.2.6	当前项目结构
				189
			9.2.7	运行结果
				189
			9.2.8	实例易错点
				190
			9.3	【实例】分布式使用 Redis 实现消息通信
				190
			9.3.1	消息通信应用场景
				190
			9.3.2	Redis 与 MQ 一系列消息队列的区别
				191
			9.3.3	实例背景
				191

9.3.4	在 send 微服务中配置模板	192	10.2.2	每个字段允许值	207
9.3.5	在 send 微服务中定时向队列发布数据	192	10.2.3	cron 特殊字符意义	208
9.3.6	在 listener 微服务中编写订阅渠道的配置信息	193	10.2.4	常用 cron 表达式	208
9.3.7	在 listener 微服务中编写监听实现类	195	10.3	任务调度的分布式	209
9.3.8	当前项目结构	195	10.3.1	任务调度的分布式解决方案	209
9.3.9	send 微服务与 listener 微服务运行结果	196	10.3.2	任务调度的分布式实现原理	210
9.3.10	实例易错点	196	10.4	【实例】微服务整合任务调度分布式	210
9.4	Spring Cache 与 Spring Data Redis 的区别	196	10.4.1	实例背景	210
9.5	【实例】保持 MySQL 与 Redis 数据一致性	197	10.4.2	增加 Quartz 依赖	210
9.5.1	实例背景	197	10.4.3	在数据库中增加 Quartz 分布式的管理表	210
9.5.2	编写资源配置文件	198	10.4.4	编写资源配置文件	211
9.5.3	编写实体类 Java Bean	198	10.4.5	创建任务调度管理 Java Bean	212
9.5.4	编写 JPA 仓库	199	10.4.6	创建所需执行的任务	213
9.5.5	编写 Service 接口及实现类	199	10.4.7	创建执行任务的操作类	213
9.5.6	编写 Controller 接口进行测试	202	10.4.8	增加控制层	216
9.5.7	当前项目结构	202	10.4.9	当前项目结构	218
9.5.8	运行结果	202	10.4.10	运行效果	218
9.5.9	实例易错点	203	10.4.11	实例易错点	219
9.6	本章小结	204	10.5	本章小结	220
9.7	习题	204	10.6	习题	220
第 10 章	微服务的任务调度与分布式的任务调度	205	第 11 章	微服务的文件上传与分布式文件管理	221
10.1	【实例】微服务整合任务调度	205	11.1	文件上传/下载原理	221
10.1.1	实例背景	205	11.1.1	SpringMVC 文件上传原理	223
10.1.2	编写任务调度实现类	205	11.1.2	文件下载原理	225
10.1.3	编写资源配置文件	206	11.2	【实例】微服务的单文件和多文件上传	226
10.1.4	当前项目结构	206	11.2.1	实例背景	226
10.1.5	运行效果	206	11.2.2	编写 application.properties 资源配置文件	227
10.1.6	实例易错点	206	11.2.3	编写相关接口	228
10.2	@Scheduled 注解详解	207	11.2.4	编写前台页面	230
10.2.1	cron 表达式	207	11.2.5	当前项目结构	231
			11.2.6	运行结果	232
			11.2.7	实例易错点	233

11.3 分布式文件管理.....	235	12.1.6 应用层与数据层分离	251
11.3.1 分布式文件管理特性.....	235	12.1.7 CDN 加速.....	251
11.3.2 分布式文件管理解决的问题.....	235	12.1.8 异步架构.....	251
11.3.3 分布式文件管理解决方案.....	235	12.1.9 响应式编程.....	251
11.4 FastDFS 解决方案.....	235	12.1.10 冗余化管理.....	252
11.4.1 FastDFS 的存储策略	236	12.1.11 灰度发布	252
11.4.2 FastDFS 的文件上传过程.....	236	12.1.12 页面静态化.....	252
11.4.3 FastDFS 的文件同步过程.....	236	12.1.13 服务端主动推送	253
11.4.4 FastDFS 的文件下载过程.....	237	12.2 微服务扩展	253
11.5 FastDFS 的安装部署.....	237	12.2.1 微服务整合日志	253
11.5.1 安装 LibFastCommon	237	12.2.2 微服务整合单元测试	253
11.5.2 安装 FastDFS	237	12.2.3 微服务整合全局异常	253
11.5.3 配置 FastDFS 的跟踪服务器.....	238	12.2.4 微服务整合 JSR-303 验证机制.....	254
11.5.4 配置 FastDFS 的数据存储 服务器	239	12.2.5 微服务整合国际化.....	254
11.5.5 配置 FastDFS 的客户端 并测试	240	12.2.6 微服务整合安全与认证.....	254
11.5.6 安装 Nginx 部署 FastDFS.....	240	12.2.7 微服务整合 WebSocket 协议	254
11.6 【实例】分布式微服务整合 FastDFS	243	12.2.8 微服务整合 HTTPS.....	255
11.6.1 实例背景	243	12.2.9 微服务整合批处理.....	255
11.6.2 编写 FastDFS 核心配置类.....	244	12.2.10 微服务整合 lombok.....	255
11.6.3 编写 FastDFS 工具类.....	244	12.2.11 微服务整合异步消息驱动	255
11.6.4 编写测试接口.....	245	12.2.12 分布式链路监控	255
11.6.5 当前项目结构.....	246	12.2.13 分布式单点登录	256
11.6.6 运行结果	246	12.3 【实例】分布式网关的初步 测试.....	256
11.6.7 实例易错点	247	12.3.1 实例背景	256
11.7 本章小结	248	12.3.2 使用资源配置文件的方式配置 分布式网关.....	256
11.8 习题	248	12.3.3 使用注册 Bean 的方式配置 分布式网关.....	257
第 12 章 扩展与部署.....	249	12.3.4 运行结果	258
12.1 微服务分布式架构相关方案 总结.....	249	12.4 微服务打包	258
12.1.1 解决方案与目标.....	249	12.4.1 Jar 包	258
12.1.2 分布式部分技术细节扩展.....	250	12.4.2 War 包.....	259
12.1.3 动静分离	250	12.5 本章小结.....	259
12.1.4 前后端分离	250	12.6 习题.....	259
12.1.5 数据库读写分离与主从分离	251	参考文献	260

第 1 章 微服务分布式架构设计原理

本章按照微服务分布式架构的顺序对其进行讲解，并以实例形式介绍如何编写分布式微服务的代码。每个微服务的架构都先从微服务的注册、微服务间的通信开始，然后编写每个微服务的持久化、缓存等内容。

1.1 Java Web 应用程序的发展历史

Java Web 应用程序的发展历史经历了以下几个过程。

1. Servlet 类项目

在 web.xml 中对 Servlet 容器方式的项目架构进行配置。截至目前依然还有部分老项目用 Servlet 方式进行运行。

2. EJB 类项目

EJB 将业务逻辑从客户端软件中抽取出来，封装在一个组件中。EJB 组件运行在一个独立的服务器上，客户端软件通过网络调用组件提供的服务以实现业务逻辑，而客户端软件的功能是只负责发送调用请求和显示处理结果。在 Java 中，能够实现远程对象调用的技术是 RMI，而 EJB 技术的基础正是 RMI。通过 RMI 技术，J2EE 将 EJB 组件创建为远程对象，客户端可以通过网络调用 EJB 对象。

3. REST+JBoss 平台类项目

REST+JBoss 的设计风格很独特。JBoss 提供了一个编程平台，以及一系列拦截器和 Web 类工具，在编写代码时拦截并过滤模块、向外提供服务，构建 REST 轻量级开发，只需编写很少的配置文件即可引用相关拦截器。在 REST+JBoss 平台时期，国内很多公司也在研发自己的 Java 平台，平台中会集成多种框架和工具，如线程池、持久化、缓存、异步、数据模型等，方便公司内部员工编程。

4. SSM 类项目（Spring+Struts/Struts2+Hibernate/MyBatis）

Java 逐渐开始研发更完善的轻量级开发架构，减少配置文件等压力，将程序专注于业务本身。在 EJB 时期每个项目至少都有几十个配置文件，相比较下 SSM 减轻了依赖很多配置文件的压力，因为 SSM 类项目的架构快速便捷，大部分功能由 Spring 进行集中管理。由于 SSM 架构具备此优点，所以国内部分公司开始减少重复开发的行为，小规模公司则很少再开发公司内部的编程平台。

5. SOA 类项目

在单个 SSM 无法解决大批量高并发需求时,人们开始将一些大型程序分成多个独立运行的模块进行处理,将每个模块部署在不同的服务器上,模块之间用 WebService 协议进行通信。虽然在 SOA 初期便逐渐有了微服务分布式的苗头,但在 SOA 阶段如果大型 Java 项目写得精细一些,每个 WebService 下就都会有相关的管理工具、调度页面、性能监控、业务监控、前置系统和黑白名单,整个大型项目的代码十分复杂且数量庞大。在 SOA 模式下写出来的项目十分安全、稳定、性能优良。因此,时至今日安防类、金融类项目仍然会采取 SOA 架构模式来架构系统,并且在不同的公司之间通信通常也会采用 WebService 协议。

6. Dubbo+Zookeeper 类项目

Dubbo+Zookeeper 分布式服务架构的项目也流行了一段时间,Dubbo 是阿里巴巴公司于 2011 年末开源的一个高性能服务框架,使用高性能 RPC 实现服务的输入和输出,属于分布式项目。在分布式的思想上与 Spring Cloud + Spring Boot 并无任何区别,只是因为 Dubbo 还没有推广起来,阿里巴巴公司就在 2013 年停止了 Dubbo 的更新维护,所以这个服务框架渐渐地被埋没了,虽然现在阿里巴巴公司正在重启 Dubbo 项目,但目前 Spring Cloud 过于火热,Dubbo 一直处于不温不火的状态。

7. Spring Cloud+Spring Boot 类项目

微服务的分布式架构利用 Eureka 或 Consul 平台进行注册,将每个业务分成一个个小的 Spring Boot 微服务包,让每个微服务包独立运行后在注册平台进行注册,通过 Spring Cloud Feign 进行相互通信,也是目前主流的架构。

1.2 微服务分布式

微服务工程将众多框架通过一个入口集成到自身的程序中,让编程人员不需关注框架原理即可直接进行调用。当初的 SSM 类项目也基于相同的思想,但当 SSM 类项目集成的框架过多后,配置文件数目与日俱增且难以管理,而此时 Spring Boot 微服务赋予了它们所需基本配置参数的大部分框架和工具,将初始配置参数的压力再次缩减。

分布式架构将不同的业务模块通过不同的服务器运行,曾经的多个 WebService 组建的 SOA 架构的项目也基于相同的思想,而此时的 Spring Cloud 把过去的架构变得更加简便、轻松。

从 SOA 类项目开始将大型应用程序拆分成多个独立运行的小型应用程序,多个应用程序之间由 WebService 协议进行通信。因为每个 WebService 服务还要集成 SSM,甚至每个服务还要做相关的管理页面和各种监控,所以每个 WebService 服务开发的过程都比较冗长。

微服务架构下的每个微服务和 WebService 一样都是独立运行的,将复杂的业务以模块的形式拆开,方便开发人员编写。微服务通常使用 Spring Boot 进行快捷开发,大部分基础信息在 Spring Boot 的底层之中都已经被配置好了,不像 SSM 一样需要编写大量 XML 程序,因此每个微服务在编程的过程中都会十分快速。

1.2.1 Spring Boot 微服务的定义和特点

Spring Boot 微服务可以轻松地创建独立的产品级应用程序。Spring 平台和第三方库采取自行管理的方式,可更加便捷地开始编写应用程序。Spring Boot 通过 pom.xml 文件,依赖 Spring Boot 的 Starter,将所有需要的 Jar 包集中在一起。Spring Boot 微服务替代了原本的 Spring + SpringMVC/Struts2 + MyBatis/Hibernate 结构。

Spring Boot 做了很多配置文件的管理。原本需要写三四个 Application.xml 文件,对数据库资源池、事务、服务、视图、静态资源等进行配置,现在则完全不需要,而事实上与以前做的传统项目大部分的配置文件也没有较大的区别。

打包变成了直接运行的 Jar 包,因为包变小了,包内所提供的内容更加清晰明确,所以称为微服务,特点如下。

- 创建独立的 Spring 应用程序。
- 直接嵌入 Tomcat、Jetty 或 Undertow (不需要部署 war 文件)。
- 提供被约定好的 Starter 依赖内容,以简化构建配置。
- 尽可能自动配置 Spring 和第三方库。
- 提供产品在生产过程中所需要的如健康检查和外部化配置等。
- 没有绝对的代码生成,也不需要 XML 配置。

Spring Boot 的部署简单,只要服务器有 Java 环境便可运行,即使再多的服务器也能快速且稳定运行。

Spring Boot 的设计初衷是约定大于配置,过多的重复性配置压力在 SSM 里让人深有体会,每个 SSM 项目中的 XML 大部分内容都是类似的,对于资深程序员来讲,任何一个新项目的编写都是一个复制粘贴的过程,而 Spring Boot 省略了初始化配置和重复性配置的过程。

1.2.2 Spring Boot 的职场导读

如今 Spring Boot+Spring Cloud 的微服务分布式项目架构,如同当年的 SSM 类架构一样,属于 1~3 年工作经验 Java Web 研发程序员找工作的必备技能之一。通常初级 Java Web 研发职位的技能要求如下:

- (1) 熟练掌握 Java 后台编码基础技术,如 JVM、语法、GC、多线程、IO、网络编程、反射等;
- (2) 熟练掌握 Spring Cloud 与 Spring Boot 架构开发;
- (3) 熟悉 Spring、SpringMVC、MyBatis 等开源框架,最好了解其中部分源码;
- (4) 熟悉 Tomcat、JBoss、Weblogic 等主流应用服务器;
- (5) 熟悉 HTML、CSS、JavaScript、jQuery 等前段基础知识;
- (6) 熟悉 Oracle 或 MySQL 数据库开发,如 SQL 与存储过程;
- (7) 熟练使用 Linux 操作系统,掌握基本 Shell 命令;
- (8) 熟练掌握 Redis、MongoDB;
- (9) 熟练使用 Webservice 主流框架,如 CXF、Axis 等;
- (10) 熟练使用 Maven、Git、SVN 等相关版本控制工具。

目前,主流的架构通常用 Spring Boot + Spring Cloud 作为程序后台,用 VUE 或 AngularJS 作为程序前台,达到 VUE+Spring Boot+Spring Cloud 前后台分离的架构模式,Java 程序员再不需要关心 JSP 等前台相关内容,这些都由前端人员自行管理,减轻了沟通成本,也减轻不少编程和维护上的压力。

微服务类框架除了 Spring Boot,还有 JFinal、JBoot 等国产微服务框架。

1.2.3 Spring 部分内容

如表 1-1 所示, Spring 的常用框架有很多种,而且 Spring 定制了大多分布式架构所需的工具与框架。Spring 已经替编程人员完成了分布式的架构,而分布式编程人员在其中挑选一些适合项目的工具并参照操作文档进行集成即可。

表 1-1 Spring 的常用框架

主项目	子项目	作用
Spring Framework		Spring 框架核心 IOC/AOP
Spring Boot		微服务
Spring Data	Spring Data JDBC	通过模板的方式操作持久化数据库,但是不提供缓存、延迟加载等
	Spring Data JDBC Extensions	Spring Data JDBC 的扩展框架,提供了 Oracle 数据库的一些高级功能,包括故障转移、高级队列、高级数据类型、数据源预置器等
	Spring Data JPA	通过定义函数名或注解的形式使用 JPA 仓库的一种操作持久化数据库的 ORM 类方案
	Spring Data LDAP	使用 Spring 应用程序操控轻量级目录访问协议 (LDAP) 的实现方案
	Spring Data MongoDB	通过模板方式操作非关系型数据库 MongoDB 的方案,编写方式类似于 Spring Data JDBC
	Spring Data Redis	通过模板方式操作非关系型数据库 Redis 的方案,编写方式类似于 Spring Data JDBC
	Spring Data R2DBC	JDBC 驱动程序的反应变体称为 R2DBC,它允许数据异步流式传输到已订阅它的任何端点,结合使用 R2DBC 等反应式驱动程序;类似于异步访问持久化的一种方式,与 Spring WebFlux 能够编写一个完整的响应式应用程序来异步进行数据的接收和发送
	Spring Data REST	使在 Spring 数据存储库上构建超媒体驱动的 REST Web 服务变得更加容易
	Spring Data for Apache Cassandra	最初由 Facebook 公司开发的 Cassandra 是一个开源分布式 NoSQL 数据库系统
	Spring Data for Apache Geode	ApacheGeode 项目的主要目标是使用 ApacheGeode 为分布式数据管理构建高可伸缩性的 Spring 支持的应用程序
Spring Data for Apache Solr、Spring Data for Pivotal Gemfire、Spring Data Couchbase、Spring Data Elasticsearch、Spring Data Evers、Spring Data Neo4J、Spring for Apache Hadoop 为 Spring Data 子项目里操作不同的数据源,有兴趣可自行查阅相关资料		

续表

主项目	子项目	作用
Spring Cloud	Spring Cloud Config	在所有环境中为应用程序管理外部属性,每个微服务不需要重新定义其环境变量, Spring Cloud Config 允许统一在 GIT、SVN 中进行管理
	Spring Cloud Stream	用于构建与共享消息系统相连接的高度可伸缩的事件驱动微服务
	Spring Cloud for Amazon Web Services	简化了与托管的 Amazon Web 服务的集成,提供了一种使用众所周知的 Spring 习惯用法和 API (如消息传递或缓存 API) 与 AWS 提供的服务交互方法,开发人员围绕托管服务构建应用程序,而不必关心基础设施或维护; Amazon Web Services 即 AWS,是 Amazon 公司的云计算 IaaS 和 PaaS 平台服务
	Spring Cloud Bus	将分布式系统的节点与轻量级消息代理连接起来,然后用于广播状态更改(如配置更改)或其他管理指令,目前唯一的实现是使用 AMQP 代理作为传输,但其他传输路线图上有着相同的基本功能集; AMQP 即 Advanced Message Queuing Protocol,一个提供统一消息服务的应用层标准高级消息队列协议,是应用层协议的一个开放标准,为面向消息的中间件设计
	Spring Cloud Consul	类似于 Spring Eureka 的注册中心
	Spring Cloud Eureka	分布式的注册中心,每个微服务会在 Eureka 中进行注册,方便微服务之间互相调用,但是 Eureka 2.0 以后已经停止更新
	Spring Cloud Gateway	一种简单而有效的方法使路由到 API,通常给外部请求(非分布式微服务下项目)进行路由
	Spring Cloud Feign	分布式微服务之间互相调用的工具
	Spring Cloud Ribbon	分布式微服务的客户端负载均衡,通过内部进行分发请求,提供了一些负载均衡的策略算法
	Spring Cloud Zookeeper	分布式微服务绑定 Zookeeper 的工具
	Spring Cloud Kubernetes	分布式微服务整合 K8S
	Spring Cloud Task	提供云端管理任务调度
	Spring Cloud Security	分布式权限
	Spring Cloud Zuul	为微服务集群提供代理、过滤、集群的工具
	此处省略了众多不常用的子项目,有兴趣可自行查阅相关资料	
Spring Cloud Data Flow		实时数据处理的工具
Spring Security	Spring Security SAML	身份验证与授权
	Spring Security Oauth	单点登录 SSO
	Spring Security Kerberos	对接 Kerberos 协议
		Spring Security 是 Spring 负责安全部分的框架,其下也有 Spring Security Messaging 为 WebSocket 提供安全、Spring Security Webflux 为异步 Web 提供安全等相关模块,有兴趣可自行查阅相关资料