从0到1搭建微服务系统

从1到0实现微服务的拆分

通过引入Spring Cloud技术栈，实现对微服务的治理

Spring Cloud 微服务架构开发实战

柳伟卫◎著

北京大学出版社
PEKING UNIVERSITY PRESS

Spring Cloud 微服务架构开发实战

柳伟卫◎著

北京大学出版社
PEKING UNIVERSITY PRESS

内 容 提 要

众所周知，Spring Cloud 是开发微服务架构系统的利器，企业对 Spring Cloud 方面的开发需求也非常旺盛。然而，虽然市面上介绍 Spring Cloud 的概念及基础入门的书籍较多，但这些书籍中的案例往往只是停留在简单的“Hello World”级别，缺乏可真正用于实战落地的指导。

本书与其他书籍不同，其中一个最大的特色是真正从实战角度出发，运用 Spring Cloud 技术来构建一个完整的微服务架构的系统。本书全面介绍 Spring Cloud 的概念、产生的背景，以及围绕 Spring Cloud 在开发微服务架构系统过程中所面临的问题时应当考虑的设计原则和解决方案。特别是在设计微服务架构系统时所面临的系统分层、服务测试、服务拆分、服务通信、服务注册、服务发现、服务消费、集中配置、日志管理、容器部署、安全防护、自动扩展等方面，给出了作者自己独特的见解。本书不仅介绍了微服务架构系统的原理、基础理论，还以一个真实的天气预报系统实例为主线，集成市面上主流的最新的实现技术框架，手把手地教读者如何来应用这些技术，创建一个完整的微服务架构系统。这样读者可以理论联系实践，从而让 Spring Cloud 真正地落地。

此外，本书不仅可以令读者了解微服务架构系统开发的完整流程，而且通过实战结合技术点的归纳，令读者知其然且知其所以然。本书所涉及的技术符合当前主流，并富有一定的前瞻性，可以有效提高读者在市场中的核心竞争力。

本书主要面向以 Spring 为核心的 Java EE 开发者，以及对 Spring Cloud 和微服务开发感兴趣的读者。

图书在版编目(CIP)数据

Spring Cloud 微服务架构开发实战 / 柳伟卫著. — 北京 : 北京大学出版社, 2018.6
ISBN 978-7-301-29456-7

Ⅰ. ①S… Ⅱ. ①柳… 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(2018)第079071号

书　　名　Spring Cloud 微服务架构开发实战
SPRING CLOUD WEI FUWU JIAGOU KAIFA SHIZHAN
著作责任者　柳伟卫　著
责任编辑　吴晓月
标准书号　ISBN 978-7-301-29456-7
出版发行　北京大学出版社
地　　址　北京市海淀区成府路205 号　100871
网　　址　http://www.pup.cn　　新浪微博: @ 北京大学出版社
电子信箱　pup7@ pup.cn
电　　话　邮购部 62752015　发行部 62750672　编辑部 62570390
印 刷 者　三河市博文印刷有限公司
经 销 者　新华书店
787毫米×1092毫米　16开本　23.75印张　552千字
2018年6月第1版　2018年6月第1次印刷
印　　数　1—4000册
定　　价　79.00 元

本书献给我的大女儿菀汐，愿她永远健康快乐！

前言

Preface

写作背景

对于微服务知识的整理归纳，最早是在笔者的第一本书《分布式系统常用技术及案例分析》的微服务章节中，作为微服务的基础理论来展开的。由于篇幅限制，当时书中所涉及的案例深度和广度也比较有限。其后，笔者又在 GitHub 上，以开源方式撰写了《Spring Boot 教程》《Spring Cloud 教程》系列教程，为网友们提供了更加丰富的使用案例。在 2017 年，笔者应邀给慕课网做了一个关于 Spring Boot、Spring Cloud 实战的系列视频课程。视频课程上线后受到广大 Java 技术爱好者的关注，课程的内容也引起了热烈的反响。很多学员通过学习该课程，不但可以学会 Spring Boot 及 Spring Cloud 最新的周边技术栈，掌握运用上述技术进行整合、搭建框架的能力，熟悉单体架构及微服务架构的特点，而且最终实现掌握构建微服务架构的实战能力。最为重要的是提升了学员自身在市场上的价值。①

众所周知，Spring Cloud 是开发微服务架构系统的利器，企业对 Spring Cloud 方面的开发需求也非常旺盛。然而，虽然市面上介绍 Spring Cloud 的概念及基础入门的书籍较多，但这些书籍中的案例往往只是停留在简单的“Hello World”级别，缺乏可真正用于实战落地的指导。于是，笔者打算写一本可以完整呈现 Spring Cloud 实战的书籍。

笔者将以往系列课程中的技术做了总结和归纳，采用目前最新的 Spring Boot 及 Spring Cloud 技术栈（本书案例基于最新的 Spring Boot 2.0.0.M4 及 Spring Cloud Finchley.M2 编写）来重写了整个教学案例，并整理成书，希望能够弥补 Spring Cloud 在实战方面的空白，从而使广大 Spring Cloud 爱好者都能受益。

内容介绍

本书围绕如何整合以 Spring Cloud 为核心的技术栈，来实现一个完整的微服务架构的系统而展开。全书大致分为三部分。

第一部分（第 1 至 4 章）：从 Spring Boot 入手，从 0 到 1 快速搭建了具备高并发能力、界面友好的天气预报系统。

第二部分（第 5 至 7 章）：首先剖析单块架构的利弊，从而引入微服务架构的概念，并从 1 到 0 实现了微服务的拆分。

第三部分（第 8 至 16 章）：通过引入 Spring Cloud 技术栈来实现对微服务的治理，重点讲解服务注册与发现、服务通信、服务消费、负载均衡、API 网关、集中化配置、容器部署、日志管理、服务熔断、自动扩展等方面的话题。

---

① 有关笔者的课程和教程介绍，可见 https://waylau.com/books/。

源代码

本书提供源代码下载，下载地址为 https://github.com/waylau/spring-cloud-microservices-development。

本书所涉及的技术及相关版本

技术版本是非常重要的，特别是对实战内容而言。因为不同的版本之间存在兼容性问题，而且不同的版本，软件所对应的功能也是不同的。本书所列出的技术，版本上相对比较新，都是经过笔者自己大量实际测试的。这样，读者在自行搭建微服务架构系统时，可以参考本书所列出的版本，从而避免很多因为版本兼容性差异所产生的问题。建议读者将相关开发环境设置成与本书所采用的开发环境一致，或者不低于本书所列的配置。详细的版本配置，可以参阅本书“附录 A”的内容。

本书示例采用 Eclipse 编写，但示例源码与具体的 IDE 无关，读者可以选择适合自己的 IDE，如 IntelliJ IDEA、NetBeans 等。

勘误和交流

本书如有勘误，会在 https://github.com/waylau/spring-cloud-microservices-development 上进行发布。由于笔者能力有限，加上时间仓促，错漏之处在所难免，欢迎读者批评指正。

您也可以通过以下方式直接联系：

博客：https://waylau.com

邮箱：waylau521@gmail.com

微博：http://weibo.com/waylau521

开源：https://github.com/waylau

致谢

感谢北京大学出版社的各位工作人员为本书出版所做出的努力。

感谢我的父母、妻子和两个女儿。由于撰写本书，牺牲了很多陪伴家人的时间，在此感谢他们对我工作的理解和支持。

最后，感谢 Spring Cloud 团队为 Java 社区提供了这么优秀的框架。由衷地希望 Spring Cloud 框架发展得越来越好！

柳伟卫

目录

Contents

第1章

微服务概述

1.1 传统软件行业面临的挑战

自 20 世纪 50 年代计算机诞生以来，软件行业也迎来了蓬勃的发展。软件从最初的手工作坊式的交付方式，演变成为职业化开发、团队化开发，进而定制了软件行业的相关规范，形成了软件产业。

但在早期的计算机系统中，软件往往围绕着硬件来开发。硬件之间能够通用，而软件程序却往往是为一个特定的硬件中的某个目标功能而编写，软件的通用性非常有限。

早期的软件，大多数是由使用该软件的个人或机构开发的，所以软件往往带有非常强烈的个人色彩。早期的软件开发没有什么系统的方法论可以遵循，也不存在所谓的软件设计，纯粹就是某个人的头脑中的思想表达。而且，由于在软件开发过程中不需要与他人协作，所以，软件除了源代码外，往往没有软件设计、使用说明书等文档。这样，就造成了软件行业缺乏经验的传承。

从 20 世纪 60 年代中期到 70 年代中期是计算机系统发展的第二个时期，在这一时期软件开始被当作一种产品而广泛使用。所谓产品，就是可以提供给不同的人使用，从而提高了软件的重用率，降低了软件开发的成本。例如，以前，一套软件只能专门提供给某个人使用，现在同一套软件可以批量地卖给不同的人，显然，就分摊到相同软件上的开发成本而言，卖得越多，成本自然就越低。这个时期，出现了类似“软件作坊”的专职替别人开发软件的团体。虽然是团体协作，但软件开发的方法基本上仍然沿用早期的个体化软件开发方式，这样导致的问题是软件的数量急剧膨胀，软件需求日趋复杂，软件的维护难度也就越来越大，开发成本变得越来越高，从而导致软件项目频频遭遇失败。这就演变成了“软件危机”。

1.1.1 软件危机概述

“软件危机”迫使人们开始思考软件的开发方式，使人们开始对软件及其特性进行了更加深入的研究，人们对待软件的观念也在悄然地发生改变。在早期，由于计算机的数量很少，只有少数军方或科研机构才有机会接触到计算机，这就让大多数人认为，软件开发人员都是稀少且优秀的（一开始确实也是如此）。由于软件开发的技能只能被少数人所掌握，所以大多数人对于“什么是好的软件”缺乏共识。实际上，早期那些被认为是优秀的程序常常很难被别人看懂，里面充斥着各种程序技巧。加之当时的硬件资源比较紧缺，迫使开发人员在编程时，往往需要考虑更少地占用机器资源，从而会采用不易阅读的“精简”方式来开发，这更加加重了软件的个性化。而现在人们普遍认为，优秀的程序除了功能正确和性能优良外，还应该更加让人容易看懂、容易使用、容易修改和扩充。这就是软件可维护性的要求。

1968 年 NATO（北大西洋公约组织）会议上首次提出“软件危机”（Software Crisis）这个名词，同时，提出了期望通过“软件工程（Software Engineering）”来解决“软件危机”。“软件工程”的目的就是要把软件开发从“艺术”和“个体行为”向“工程”和“群体协同工作”进行转化，从

而解决“软件危机”。

“软件工程”包含以下两个方面的问题：

（1）如何开发软件，以满足不断增长、日趋复杂的需求；

（2）如何维护数量不断增长的软件产品。

在软件的可行性分析方面，首先对软件进行可行性分析，可以有效地规避软件失败的风险，提高软件开发的成功率。

在需求方面，软件行业的规范是，需要制定相应的软件规格说明书、软件需求说明书，从而让开发工作有了依据，划清了开发边界，并在一定程度上减少了“需求蔓延”情况的发生。

在架构设计方面，需制定软件架构说明书，划分系统之间的界限，约定系统间的通信接口，并将系统分为多个模块。这样更容易将任务分解，从而降低系统的复杂性。

1.1.2 软件架构的发展

软件工程的出现，在一定程度上减少了软件危机的发生。但随着软件行业的高速发展，人们对于软件的要求也越来越高。在 30 年前，软件也许只是满足数据计算。而今，软件不只在娱乐、金融、节能、医疗等各个行业发挥作用，同时，软件架构也不断向着分布式、高并发、高可用、可扩展等方面演进。

早期的软件往往采用集中式的部署方式。这种部署所需要的主机要有非常高的可靠性，因为这样的主机一旦宕机，那么，部署在该主机上的软件就不可用了。同时，这类主机又需要有比较高的配置，能够承载业务的升级需要。

依据摩尔定律，计算机芯片每 18 个月集成度翻番，价格减半。传统的晶体管是由硅制成的，然而 2011 年来硅晶体管已接近原子等级，达到物理极限，由于这种物质的自然属性，硅晶体管的运行速度和性能难有突破性发展。正是由于硬件上的限制，使得主机的性能不可能无限地提升。单机的部署方式自然会受到限制，所以近些年来，分布式部署的方式越来越普及。

然而，对于集中式的部署而言，分布式部署软件的方式也不是没有缺点。在设计分布式系统时，经常需要考虑如下的挑战。

- 异构性：分布式系统由于基于不同的网络、操作系统、计算机硬件和编程语言来构造，必须要考虑一种通用的网络通信协议来屏蔽异构系统之间的差异。一般交由中间件来处理这些差异。
- 缺乏全球时钟：在程序需要协作时，它们通过交换消息来协调它们的动作。紧密的协调经常依赖于对程序动作发生时间的共识，但是，实际上网络上计算机同步时钟的准确性受到极大的限制，即没有一个正确时间的全局概念。这是通过网络发送消息作为唯一的通信方式这一事实带来的直接结果。
- 一致性：数据被分散或复制到不同的机器上，如何保证各台主机之间数据的一致性将成为一个难点。

- 故障的独立性：任何计算机都有可能发生故障，且各种故障不尽相同。它们之间出现故障的时机也是相互独立的。一般分布式系统要设计成被允许出现部分故障，而不影响整个系统的正常使用。
- 并发：分布式系统的目的是更好地共享资源。那么系统中的每个资源都必须被设计成在并发环境中是安全的。
- 透明性：分布式系统中任何组件的故障，或者主机的升级和迁移对于用户来说都是透明的，不可见的。
- 开放性：分布式系统由不同的程序员来编写不同的组件，组件最终要集成为一个系统，那么组件所发布的接口必须遵守一定的规范且能够被互相理解。
- 安全性：加密用于给共享资源提供适当的保护，在网络上所有传递的敏感信息，都需要进行加密。拒绝服务攻击仍然是一个有待解决的问题。
- 可扩展性：系统要设计成随着业务量的增加，相应的系统也必须要能扩展来提供对应的服务。有关常见分布式系统架构的内容，会在下一章节进行讨论。

1.1.3 人的因素

不管软件行业如何发展，始终围绕的一个话题是，人在软件开发过程中产生的影响。这里的人特指软件开发人员。

软件行业的人，相比其他行业而言，有其独特性。软件开发的本质是一个艺术创作的过程，即很难通过相同的技能传授来使每个开发人员具备相同的技术经验。软件开发受制于开发人员的思考，即便是相同的开发人员，在不同的环境或不同的心情下，所产生的软件质量也参差不齐。而且，软件开发的工作量也存在比较难的量化过程，很难通过代码行数或工时来评定开发人员的工作量。这就是艺术的特点，具备太多的不确定性，需要有创作思维。

Frederick P. Brooks 在《人月神话》（*The Mythical Man-Month*）一书中阐述道：人力（人）和时间（月）之间的平衡远不是线性关系，在项目的进展过程中会存在各种不可预知的问题，增加人员反而导致项目进度越来越慢。自该书出版以来，40 多年过去了，书中的观点也仍然符合现状，这表明软件开发总是非常困难的，不会有特定的技术和方法，可以使软件项目开发的能力有突破性的进展，甚至可以说，好的开发人员是项目成功的关键。一个成功的项目里面肯定会有好的开发人员，但好的开发人员并不一定能造就项目的成功。

软件行业相比其他行业，还是一个比较新的行业，所以，经验不足是这个行业的通病。有数据表明，从 1950 年以来，每 5 年世界上的程序员的数量就增加一倍。各地软件开发的培训班如雨后春笋一般。很多培训班往往只是提供一些速成的方法，帮助学员来应付面试。这就是为什么很多用人单位发现，应聘者面试时能说会道，但真正用起来却是捉襟见肘。也就是说，这个行业虽然看上去从业人员很多，但大部分都是缺乏开发经验者，缺乏实际解决问题的能力，所以只能从事比较低

级的、相对不需要太多“创作”的工作。

在传统的项目中，往往会较少关注人的因素。其实，人的心理决定了做事的质量。一个斗志昂扬、怀抱愿景的开发人员，肯定要比态度消极、浑浑噩噩的开发人员的效率要高很多。

近些年，不少企业也关注到了这个问题。这也是为什么很多企业极力宣传企业价值观，甚至专门设置了提高开发人员幸福感的“程序员鼓励师”职位的原因。

1.1.4 技术的迭代创新

软件行业的一个最大的特点就是技术更新快。以 Java 语言为例，Sun 公司最初设计 Java 是准备用于智能家电类（如机顶盒）的程序开发。然而，由于当时机顶盒的项目并没有成功拿下，于是 Java 被阴差阳错地应用于万维网。搭着互联网的快车，Java 在移动终端、桌面程序及企业级应用中都占据了很大的市场份额。

然而，技术的发展之快令人侧目，特别是 Android、iOS 系统的普及，让 Java 在 Java ME 领域几乎已经没有市场了。同时，C#、Node.js 等语言的发展，也在“蚕食”着 Java 桌面应用的市场。就目前来说，Java 语言在企业级应用方面还占有一定的地位。

20 世纪末 21 世纪初，在 HTML 平台只能展示简陋的文字排版的时候，Flash 俨然成为令人啧啧称奇的魔法。单调的网页一旦使用了 Flash，面貌往往会焕然一新。Flash 可以说是动态网页的最佳解决方案，被广泛应用于网页游戏、门户网站、广告、视频网站、企业级应用（Flex）等领域，当时 99% 的浏览器都安装了 Flash 插件，成为事实上的标准。

然而，Flash Player 在顶峰时期，并没有完全解决两方面的致命问题，一个是安全漏洞，另一个是耗电。Flash Player 几乎每周都要发布新版本，来解决安全方面的漏洞，但漏洞总是频频出现，引发了一系列的安全问题。同时，在移动端，由于 Flash Player 耗电这一事实并没有多大改变，从而被 Steve Jobs 逐出了 iOS 手机端。同时，由于 HTML 5 的标准出现，HTML 5 开始逐步代替之前 Flash 的应用场景。2012 年 8 月 15 日，Flash 也退出 Android 平台，正式告别移动端。在过去的一年里，包括 Chrome、Edge 和 Safari 在内的各大浏览器都陆续开始屏蔽 Flash。截至 2020 年，Adobe 将正式停止支持 Flash，Flash 终将走到生命的终点。①

1.1.5 数据成为基础设施

大数据时代，数据就是互联网的“水电气煤”。

以前，评价一个网站的价值，很大程度上是靠流量。流量大，意味着这个网站的用户活跃度高，黏性大，甚至有些企业为了“搏出位”，采用购买流量的方式，来提升企业在互联网中的地位。

① 该报道可见 BBC 新闻“*Adobe to kill off Flash plug-in by 2020*”（http://www.bbc.com/news/technology-40716304）。

如今，这一切悄悄发生了转变，大企业如 Google、Amazon 等互联网巨头早就在抢占数据这个新的制高点。Amazon 利用其 20 亿个用户账户的大数据，通过预测分析 140 万台服务器上的 10 亿 GB 的数据来促进销量的增长。Amazon 采用追踪电商网站和 APP 上的一切行为，尽可能多地收集信息。看一下 Amazon.com 的“账户”部分，就能发现其强大的账户管理，这也是为收集用户数据服务的。主页上有不同的部分，如“愿望清单”“为你推荐”“浏览历史”“与你浏览过的相关商品”“购买此商品的用户也买了”，Amazon 保持对用户行为的追踪，为用户提供卓越的个性化购物体验。

尽管 Google 并没有把自己标榜成数据公司，但实际上它的确是数据宝库和处理问题的工具。它已经从一个网页索引发展成为一个实时数据中心枢纽，几乎可以估量任何可以测量的数据，比如天气信息、股票、购物等。当我们进行 Google 搜索时，大数据就会起作用。Google 使用工具来对数据分类和理解，计算程序运行复杂的算法，旨在将输入的查询与所有可用数据相匹配。通过大数据的搜索，它还能推断出你是否正在寻找新闻、事实、人物或统计信息，并从适当的数据库中提取数据。 对于更复杂的操作，如中英文翻译，Google 会调用其他基于大数据的内置算法。Google 的翻译服务研究了数以百万计的翻译文本或演讲稿，旨在为顾客提供最准确的解释。Google 还能通过分析我们浏览的网页，向我们展示可能感兴趣的产品和服务的广告。

毫无疑问，每天互联网产生了数以亿计的数据。这些大数据如何被存储、分类、检索和分析，都将挑战着这个时代的技术极限。

1.1.6 开发方式的转变

早些年，瀑布模型还是标准的软件开发模型。瀑布模型将软件生命周期划分为制订计划、需求分析、软件设计、程序编写、软件测试和运行维护六个基本活动，并且规定了它们自上而下、相互衔接的固定次序，如同瀑布流水，逐级下落。在瀑布模型中，软件开发的各项活动严格按照线性方式进行，当前活动接受上一项活动的工作结果，实施完成所需的工作内容。当前活动的工作结果需要进行验证，如验证通过，则该结果作为下一项活动的输入，继续进行下一项活动，否则返回修改。

瀑布模型的优点是严格遵循预先计划的步骤顺序进行，一切按部就班，整个过程比较严谨。同时，瀑布模型强调文档的作用，并要求每个阶段都要仔细验证文档的内容。但是，这种模型的线性过程太理想化，主要存在以下几个方面的问题。

- 各个阶段的划分完全固定，阶段之间产生大量的文档，极大地增加了工作量。
- 由于开发模型是线性的，用户只有等到整个过程的末期才能见到开发成果，从而增加了开发的风险。
- 早期的错误可能要等到开发后期的测试阶段才能发现，进而带来严重的后果。
- 各个软件生命周期衔接花费的时间较长，团队人员交流成本大。

瀑布式方法在需求不明并且在项目进行过程中可能变化的情况下基本是不可行的，所以瀑布式

方法非常适合需求明确的软件开发。但在如今，时间就是金钱，如何快速抢占市场是每个互联网企业需要考虑的第一要素。所以，快速迭代、频繁发布的原型开发和敏捷开发方式，被越来越多的互联网企业所采用。甚至很多传统企业也在逐步向敏捷、“短平快”的开发方式靠拢，毕竟，谁愿意等待呢?

客户将需求告诉了你，当然是希望越快得到反馈越好，那么，最快的方式莫过于在原有系统的基础上，搭建一个原型提供给客户作为参考。客户拿到原型之后，肯定会反馈他的意见，好或坏的方面都会有。这样，开发人员就能根据客户的反馈对原型进行快速更改，快速发布新的版本，从而实现良好的反馈闭环。

2017 年 8 月 23 日，Martin Fowler 在其博客宣布，他的老板 Roy Singham 将会出售其所在的公司 ThoughtWorks。ThoughtWorks 是世界著名的软件供应商及咨询服务机构，而 Martin Fowler 自 2000 年起，就一直在该公司担任首席科学家，宣导其敏捷开发的工作方式。这次 ThoughtWorks 的交易，也引发了业界的猜想。虽然新东家 Apax Funds 承诺接手后，原有的管理方式不会改变，但这也从侧面反映了传统的软件外包模式和传统的构建软件的方法是存在危机的。

2017 年 8 月，Oracle 在其官方博客上宣称，会将 Java EE 开源。这意味着，传统闭源的开发方式不一定是最佳的方案，即便是像 Oracle 这样的大企业，也无法靠一己之力来运营像 Java EE 这样的企业级产品。Oracle 希望 Java EE 能够更快、更好地为企业服务，那么将 Java EE 开源，无疑是更加开放、更加贴近开源社区。通过开源社区更多的力量来共同促进 Java EE 的发展，无论是对于 Oracle 还是社区，都是双赢的选择。未来软件行业的发展，开源的软件开发方式也会占有一定的市场。

总之，软件开发天生就没有银弹！开发人员唯有不断地学习技术，做好创新，才能不断地满足这个时代对于技术的要求。

1.2 常见分布式系统架构

复杂的大型软件系统，倾向于使用分布式系统架构。就像 Warren Buffett 有个关于投资的名言，就是“不要把鸡蛋放在一个篮子里”。对于系统而言也是如此。厂商的机器不可能保证永远不坏，也无法保证黑客不会来对系统搞破坏，最为关键的是，我们无法保证自己的程序不会出现 Bug。问题无法避免，错误也不可避免。我们只能把鸡蛋分散到不同的篮子里，来减少“一锅端”的风险。这就是需要分布式系统的一个重要原因。

使用分布式系统的另外一个理由是可扩展性。毕竟任何主机（哪怕是小型机、超级计算机）都会有性能的极限。而分布式系统可以通过不断扩张主机的数量以实现横向水平性能的扩展。

本章将会介绍市面上常见的分布式系统架构，并对这些架构做优缺点的比较。本章大部分内容