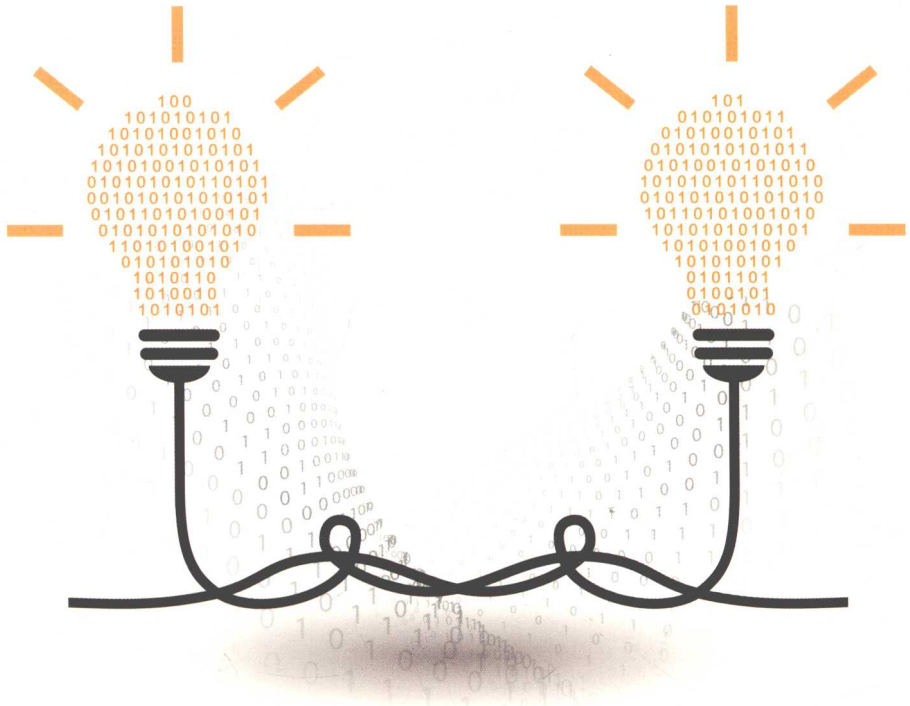
Broadview®
www.broadview.com.cn



Java多线程与Socket
实战微服务框架

庞永华 / 著

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

Java多线程与Socket
实战微服务框架

庞永华 / 著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书从实战角度出发，首先介绍 Java 多线程、Socket、Spring、动态代理、动态字节码、序列化等技术在构建分布式微服务框架中的应用。然后介绍一种微服务框架的架构设计与编程实践，并将这一微服务框架分解为底层 Socket 通信、服务注册与发现、服务暴露与引用、远程方法调用等层面，逐一深入讲解。这里重点介绍作者如何活用相关技术一步步地构建微服务框架的基础 RPC 框架并分享了相应的性能调优经验。最后介绍微服务架构中配套的服务治理系统的设计与实现方案，包括服务的设计、配置、管理与监控。

授人以鱼不如授人以渔，作者并不希望读者通过抄写代码来编写出一模一样的东西，而是希望读者通过这一学习过程，深刻掌握 Java 多线程、Socket、动态代理等相关技术，最终能够做到举一反三，灵活地运用它们，从而提升自身的 Java 编程水平，并为进一步学习和研究 Java 分布式技术与微服务框架打下基础。因此，本书从相关的基础知识入手，通过剖析现有框架，讲解这些基础知识在实践中的应用，逐步将读者带入 Java 分布式与微服务技术领域。本书适合有一定 Java 基础的初中级开发人员阅读。

图书在版编目（CIP）数据

Java 多线程与 Socket：实战微服务框架 / 庞永华著. —北京：电子工业出版社，2019.3
ISBN 978-7-121-36035-0

Ⅰ. ①J… Ⅱ. ①庞… Ⅲ. ①JAVA 语言－程序设计 Ⅳ. ①TP312.8

中国版本图书馆 CIP 数据核字（2019）第 026420 号

策划编辑：陈晓猛
责任编辑：李云静
印　　刷：三河市良远印务有限公司
装　　订：三河市良远印务有限公司
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：787×980　1/16　　印张：28.25　　字数：646 千字
版　　次：2019 年 3 月第 1 版
印　　次：2019 年 3 月第 1 次印刷
定　　价：99.00 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 zlts@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：010-51260888-819，faq@phei.com.cn。

前言

本书适合有一定 Java 基础且有志成为架构师的开发人员阅读。一个优秀的架构师必须要有扎实的编程功底和丰富的理论知识，不光要能完成架构设计，更要有能力将设计转换为实际的产品。不会写代码、纸上谈兵的“架构师”设计出来的“架构”是靠不住的。因此，本书将从相关的基础知识讲起，通过剖析一个小巧精练的微服务框架的核心，介绍这些基础知识是如何在实践中被灵活、适当地运用的。

本书不是关于微服务的理论书籍，也不是某个微服务框架的使用手册。微服务涉及的范围很广，我们很难在一本书里讲清楚微服务的方方面面。

目前相关的主流框架是 Spring Cloud 和 Dubbo。虽然 Spring Cloud 和 Dubbo 都很强大，但这个世界上从来没有最好，只有更好，因此我们永远不要停下自己创新的脚步。我们依然可以有自己的想法，并勇于尝试、付诸实践。事实上 Spring Cloud 和 Dubbo 都存在各自的问题和不足。我们通过对它们的学习和研究，站在巨人的肩膀上，吸取它们的优点再加以创新，是完全有可能做到青出于蓝而胜于蓝的。

在正式开始之前，让我们先了解一下微服务的发展背景。

面向服务的架构

1996 年，Gartner 公司首次提出了面向服务的架构（Service-Oriented Architecture，SOA）这一软件设计思想。其核心理念是将一个个的业务功能包装成一个个的标准服务，为这些服务定义良好的接口和服务契约，以便在需要的时候可以重用和水平伸缩。通过将这些服务进行组合和编排，可以创建新的业务流程，或者灵活地修改现有流程，以适应不断变化的业务需求，让我们的系统功能更丰富、结构更灵活、更易于扩展。同时，让系统规模能够根据需要弹性伸缩，最大限度地利用现有资产，提高效率，降低成本。总之，要使我们的系统能更灵活、更快地响应不断变化的需求。

不过，受到当时计算机水平的限制，面向服务的架构思想在诞生之初，并没有得到广泛的关注和发展。随着软件系统的规模越来越大、越来越复杂，软件系统的架构也在不断地演进，

面向服务的架构开始受到人们的关注和认可。目前，大型软件系统的服务端架构多数都是面向服务的架构，或者正在朝这一架构迁移。

要明确的是，面向服务的架构中的“服务”虽然也包括系统对外提供的服务，但更多的是指系统内部的各个“模块”或“组件”的“服务化”，以及模块（服务）之间的相互调用与协同。它是“分布式”与“服务化”两个技术发展趋势合流的产物。

分布式系统

所谓的“分布式”是相对于“集中式”的一种应用系统内部组织结构。相对于传统的集中式系统（单机应用系统和集群式应用系统都属于集中式系统），分布式系统将原本集中在一个服务端应用中的功能模块拆分出来，分为多个系统组件或应用，分散部署在多个服务器上，并通过网络将它们连接起来协同工作。而客户端系统感觉不到服务端系统内部的这种变化，仍然和原来调用集中式系统一样。

分布式设计使得原本集中在单个服务器上进行的计算或存储需求被分散到了多个服务器上，从而降低了我们对单一服务器的性能要求。这样就让我们能用相对廉价的 PC 服务器代替昂贵的传统服务器，并通过水平扩容的方式继续提升系统的处理能力。

分布式系统并没有约定其内部的各个组件或应用之间应采用什么样的形式来实现彼此通信。早期人们使用 DCOM、CORBA 等技术实现组件的暴露和远程访问。这便是分布式系统的早期形态。但是这些技术比较复杂且十分笨重，只在大型系统和企业级应用中被使用。尽管之后又出现了 COM+、RMI、EJB 等技术，但仍然比较笨重和难用。

服务化

早期的系统都是相互独立的。受限于计算机的处理能力，其规模都比较小。比如，财务系统和人力资源管理系统分别是两个不同的系统。它们可能由不同的软件厂商，采用不同的开发语言和技术开发，并运行在各式各样的操作系统和硬件设备上。随着企业发展的需要，人们试着在两个系统间建立通信，尝试让它们彼此协同。双方的通信方式和协议由厂商之间彼此协商来制定，开发起来成本很高。各个厂商都试图制定自己的通信技术标准和协议，并力争成为业内标准。显然，如果所有厂商都遵循同一套技术标准和通信协议，就能大大地降低开发成本，让各个系统彼此更容易地互联互通。

随着基于 Web 的应用的普及，以及 XML 技术的出现和成熟，出现了基于 HTTP、XML 的服务暴露与远程访问方式，这就是 Web Services。但 Web Services 的协议和实现方式很多，技术标准也多种多样。企业内部各系统之间的互联互通仍然比较麻烦。于是，企业服务总线

（Enterprise Service Bus，ESB）系统被设计来实现各系统之间的服务接口适配和管理，以便各系统能够用自己熟悉的技术、标准和规范来相互调用彼此的服务。随着时间的推移，简单对象访问协议（Simple Object Access Protocol，SOAP）、Web 服务描述语言（Web Services Description Language，WSDL），以及通用描述、发现与集成服务（Universal Description，Discovery and Integration，UDDI）协议逐渐成为主流的 Web Services 标准和规范。

后来，随着互联网技术的发展，基于 HTTP RESTful 的轻量级 Web Services 逐渐取代了基于 SOAP 的传统 Web Services 技术成为主流。由于 HTTP RESTful 服务暴露和调用开销比 SOAP 小很多，且速度更快，这就使得我们可在大型系统内部也大量使用，以作为各子系统之间，尤其是异构子系统之间最佳的通信形式。

面向服务

当我们将系统中的模块或组件服务化，代替 COM+、RMI、EJB 等分布式领域的组件通信技术后，系统架构就转变为面向服务的架构。当然，分布式系统中的很多组件是难以服务化的。不是所有的模块和组件都适合服务化。比如，有些模块的调用频率很高，接口复杂，转变成服务后，访问性能可能无法满足设计要求。又比如，有些模块与模块之间的耦合度较高，如果不进行重构来解耦，那么是无法单独作为服务暴露的。还有些模块的重用度不高或调用频率过低，没有服务化的必要。在对系统进行 SOA 改造时，一定要分析清楚。

综上所述，面向服务的架构是分布式架构中的一种。面向服务的架构一定是分布式架构，但分布式架构不一定面向服务。能够对外提供服务的系统并不一定是面向服务的架构，面向服务架构的系统也不一定对外暴露服务。

什么是微服务

顾名思义，微服务就是指粒度较小的服务。这意味着我们要对现有的分布式系统进行进一步的拆分，将其划分为更多、更小的服务来进行设计或重构。“微服务”的概念源于 2014 年 3 月 Martin Fowler 所写的一篇文章 *Microservices*。它扩大了面向服务架构中“服务”的概念。不再局限于系统与系统之间的接口调用，也不局限于某种具体的服务形式。系统中凡是可被复用的功能模块都可以被“服务化”，转变为“服务”。这些服务可以对外暴露，也可能仅限于在系统内部使用。这对面向服务架构提出了更高的设计要求。由于服务数量更多、粒度更小，因此管控难度会更大，对性能的要求也更高。

那么，我们为什么要将系统拆分成一个个的微服务呢？

微服务的好处

微服务有如下好处。

1. 方便编排和重用

当我们在开发新功能时，可能需要复用现有的模块。以往我们需要将可复用的代码放到 jar 包中，并在工程中引用。这容易导致依赖关系的复杂和混乱，而且每次都需要重新编译和打包，也很不方便，更不要说对现有的模块进行编排和组合了。将这些模块转变为服务后，我们只需要调用这些服务，而不需要关心服务的具体实现、依赖和提供者。同时还能够使用服务编排工具将现有服务编排到新的流程中，组合成为新的服务，或者修改现有的服务流程。

2. 方便开发和调试

每一个微服务专注于单一功能，并通过定义良好的接口来清晰表述它的边界。服务越小，复杂度越低，开发起来也就越简单、越快，调试和维护也更方便。这就缩短了服务开发和修改的周期，使程序能更快地迭代。

3. 方便部署与更新

微服务一般随应用部署。一个系统中有若干可独立运行的应用，每个应用通常提供了一个或多个微服务。当某个微服务发生变更时，只需编译和部署相应的应用，而不用重新编译和部署整个系统。这使得部署发布更加高效，同时也缩小了每次变更和部署的影响范围，降低了风险。

4. 系统集成更方便

使用不同语言、不同技术栈开发的服务，只要按照统一的协议暴露，比如统一使用 HTTP RESTful 形式暴露为服务，就可以方便地相互调用和协同。这使得我们能够将各类系统轻松集成在一起。

5. 提高容错能力

某一模块或组件发生故障时，容易导致整个系统变得不可用。比如，某个模块的内存泄漏可能导致整个应用因内存不足而崩溃。由于微服务架构中的服务粒度很小，且相互隔离，因此即使某个服务出现问题，处理起来也相对容易，影响范围有限。微服务系统可以通过失败重试、失败转移、服务降级等机制实现容错处理，避免问题扩大，微服务系统甚至能自动从故障中恢复（自愈）。

6. 方便横向扩展

当某个服务出现性能瓶颈时，我们只需要增加此服务所属应用的部署数量，而不用增加整

个系统的部署。而且，由于不同的服务对资源的要求不同，我们可以根据实际情况灵活地对服务进行混合部署，以便更合理地分配服务器资源。

RPC 与微服务

微服务环境下服务的粒度更细，调用频率也更高，一般用于系统内部的面向服务架构，而不是直接对外提供服务。在这种场景下，Web Services 的跨语言、跨平台优势不再那么重要，Web Services 的易用性问题和性能问题反而变得突出起来。

随着 Socket、多线程、序列化等技术的发展，涌现出了很多优秀的 RPC 框架，以代替传统的 RMI 等技术。与 Web Services 偏向对外暴露服务与远程调用不同，RPC 框架专注于细粒度（方法级别）的、系统内部模块或组件之间的相互调用与协同。由于多数 RPC 框架采用 Socket 长连接和二进制协议，因此其性能比基于 HTTP 协议的 Web Services 要高几个数量级。再加上，RPC 可以做到对应用的零侵入，因此其在易用性方面也要强得多。

由于 RPC 在易用性和性能方面优势明显，因此在系统内部，使用远程方法调用（RPC）形式暴露和引用服务要比 Web Services 更合适。对外暴露和引用服务时（跨系统调用）再采用 Web Services 的形式。当然，如果对性能要求不高，也可以一律采用 HTTP RESTful 方式。有些 RPC 框架，如 gRPC、Thrift，支持跨语言调用，也可以被用于跨系统的服务调用，对外提供服务。由于我们总是能轻松地将系统内部的服务以 Web Services 形式对外暴露，因此，建议优先考虑内部服务调用的性能和易用性，选用 RPC 作为微服务的核心和基础。

微服务框架

在微服务领域，比较流行的几个关键词是 Spring Cloud、Dubbo、Docker、Kubernetes、Service Mesh。其中 Spring Cloud 是目前最接近完整微服务框架的产品。它是一个基于 Java 语言的微服务开发框架，面向的是有 Spring 开发经验的 Java 语言开发者。但它真的只是一个“框架”性的东西，还需要集成一系列的第三方组件才能发挥作用。Netflix 公司为其开发了一套组件，并成为 Spring Cloud 的推荐或默认实现。

Dubbo 是阿里巴巴开源的分布式服务框架。其本质上是一个高性能二进制 RPC 框架，致力于提供高性能和透明化的 RPC 远程服务调用方案，以及 SOA 服务治理方案。其功能主要包括高性能 NIO 通信及多协议集成、服务动态寻址与路由、软负载均衡与容错、依赖分析与服务降级等。可以看出 Dubbo 有着自己的多层架构体系，涵盖了 Spring Cloud 中的一部分内容。它虽然算不上一个完整的微服务框架，但却比较实用，未来可能会支持与 Spring Cloud 的集成，融入 Spring Cloud 生态圈。

Docker 是一个应用容器引擎，允许我们将要部署的应用和运行时环境打包成一个镜像文件，

部署到 Docker 容器中。Rocket 是与之类似的另一款应用容器引擎。

Kubernetes 是 Google 开源的一个自动化容器操作平台，简称 K8S。它可以编排并自动执行容器（如 Docker、Rocket）的部署、复制等操作，随时扩展或收缩容器规模，并提供容器间的负载均衡，监控容器的状态，自动升级或替换容器。由此可见，Kubernetes 不是面向开发者的平台，而是面向 IT 基础设施运维人员的。

Spring Cloud 和 Dubbo 同属于 PaaS（Platform as a Service，平台即服务），而 Docker 和 Kubernetes 同属于 IaaS（Infrastructure as a Service，基础设施即服务）。由于微服务平台中服务的载体是应用，而环境中要部署的应用实例众多，因此使用 Docker 和 Kubernetes，通过整合 Spring Cloud、Docker 和 Kubernetes，可以构建更加完整和强大的微服务架构程序。

不过，应用的部署并不一定需要 Docker 这样的容器，Linux 自身就支持进程间的资源隔离。通过一个简单应用代理服务（Agent）加 Shell 脚本即可实现应用的部署。本书所介绍的示例微服务系统正是使用了这一方式，效果也很好。使用这种方式可以实现应用的部署、启停、更新、JVM 监控、资源监控等功能，扩展也更方便。

Service Mesh 则还是一个比较新的概念。它可将微服务间的调用、限流、熔断和监控等功能需求提炼为一个通用的中间层基础服务，甚至下沉到基础设施层。Spring Cloud、Kubernetes 和 Istio 似乎都正在朝这一方向努力。

本书作为示例所介绍的 mac-rpc 和 Dubbo 非常相似，其本质是一个内置服务注册与发现功能的高性能 RPC 框架，是微服务系统的核心和基础。其特点是全异步和高性能，小巧精练，但开放、灵活，可自由定制和扩展。mac-rpc 目前已经较为成熟和稳定，其最新版本是 1.0.3。它与服务治理系统、流程引擎、自动化部署、消息、缓存等子系统和组件集成，可以构建起完整的微服务系统。读者若想了解更多关于 mac 与 boar 系列的开源项目，请访问 http://www.boarsoft.com 和 https://www.gitee.com/Mac_J/projects。

读者服务

轻松注册成为博文视点社区用户（www.broadview.com.cn），扫码直达本书页面。

- **下载资源**：本书如提供示例代码及资源文件，均可在下载资源处下载。
- **提交勘误**：您对书中内容的修改意见可在 提交勘误 处提交，若被采纳，将获赠博文视点社区积分（在您购买电子书时，积分可用来抵扣相应金额）。
- **交流互动**：在页面下方 读者评论 处留下您的疑问或观点，与我们和其他读者一同学习交流。

页面入口：http://www.broadview.com.cn/36035

目录