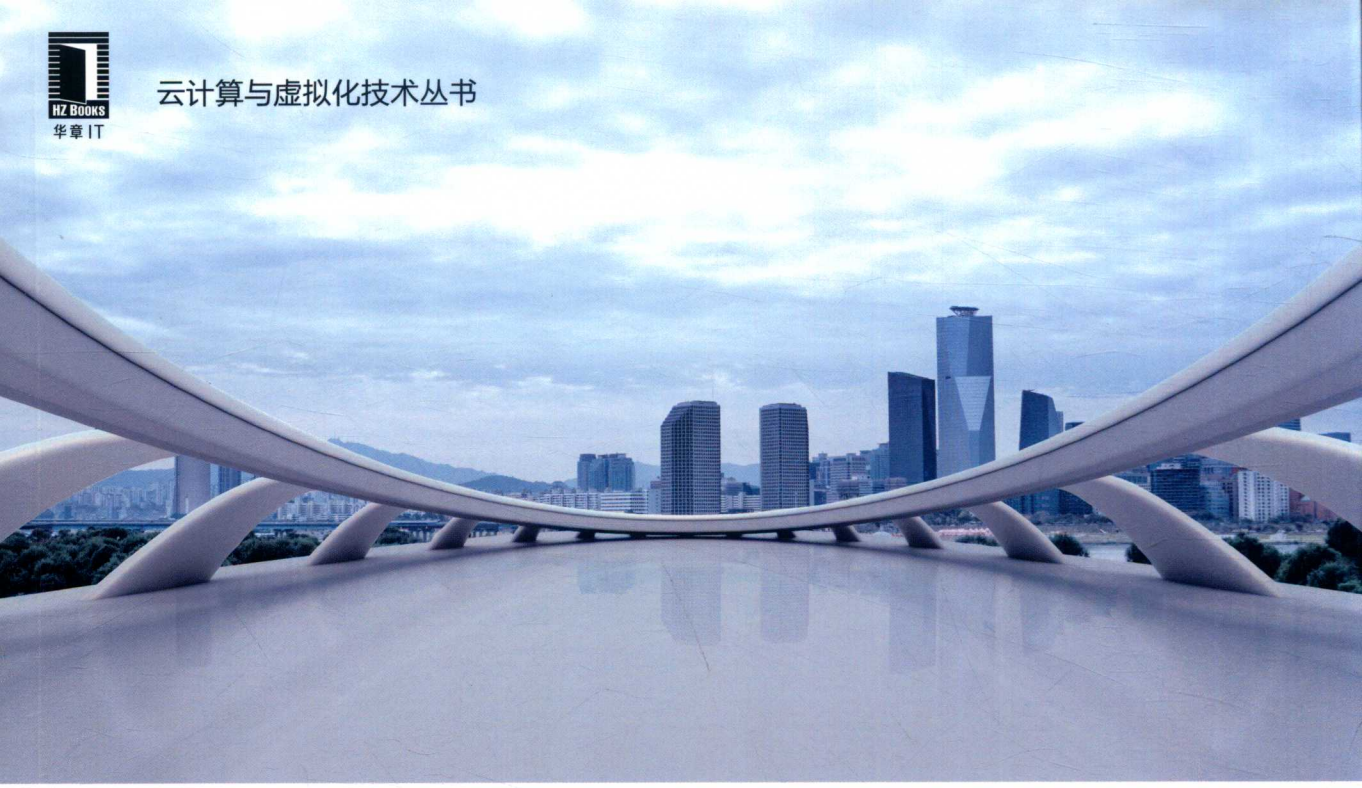
HZ BOOKS
华章IT

云计算与虚拟化技术丛书



An in-Depth Analysis of Spring Cloud

Spring Cloud 微服务架构进阶

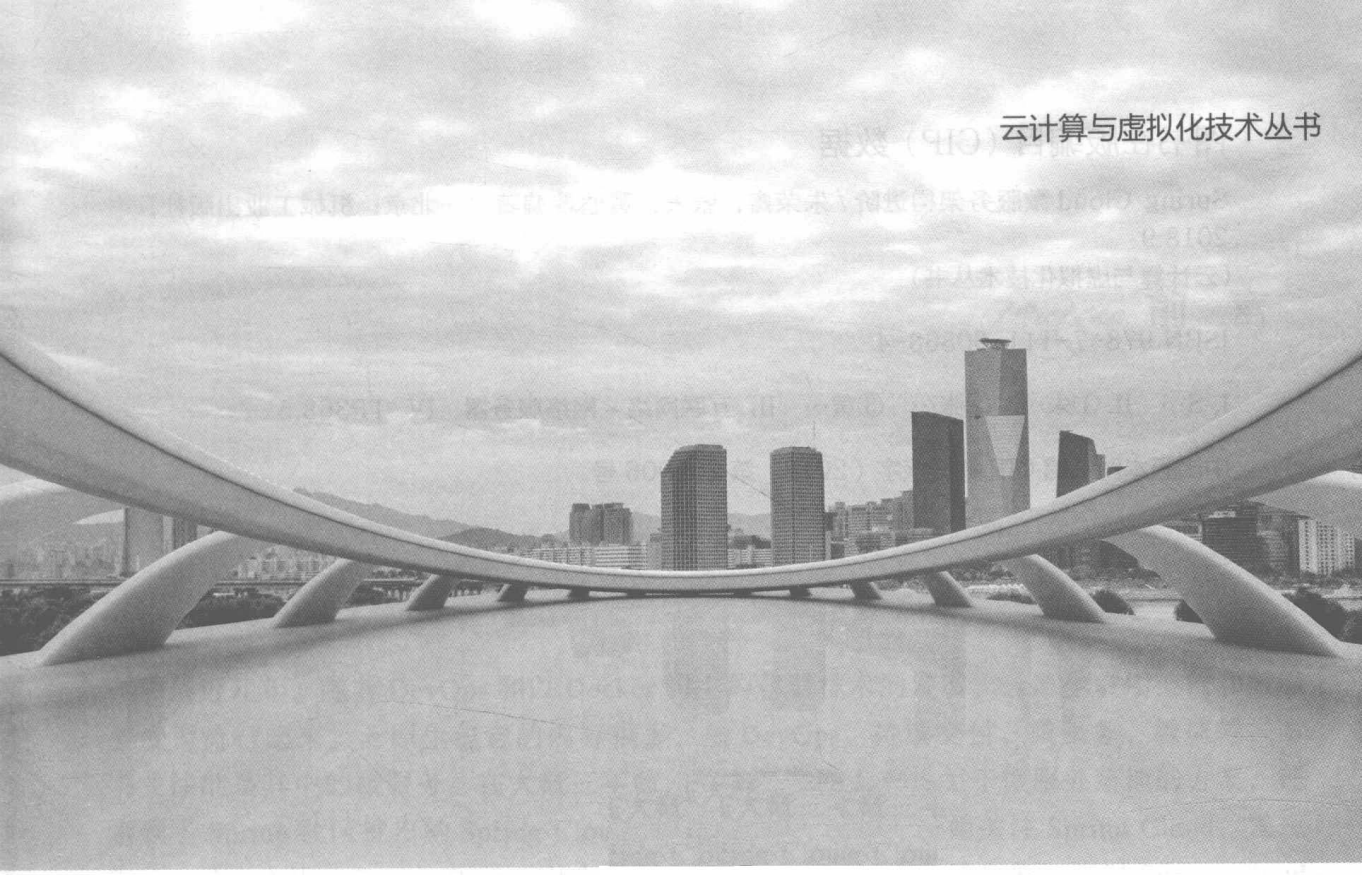
朱荣鑫 张天 黄迪璇 编著

全面系统地介绍Spring Cloud及其在微服务架构中的应用。

从基础应用到源码分析，再到进阶应用，提供了大量案例，可帮助读者快速进入实战。

机械工业出版社
China Machine Press

云计算与虚拟化技术丛书



An in-Depth Analysis of Spring Cloud

Spring Cloud微服务架构进阶

朱荣鑫 张天 黄迪璇 编著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

Spring Cloud 微服务架构进阶 / 朱荣鑫，张天，黄迪璇编著 . —北京：机械工业出版社，2018.9
（云计算与虚拟化技术丛书）

ISBN 978-7-111-60868-4

I. S…　II. ①朱…　②张…　③黄…　III. 互联网络 - 网络服务器　IV. TP368.5

中国版本图书馆 CIP 数据核字（2018）第 209906 号

Spring Cloud 微服务架构进阶

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码：100037）
责任编辑：吴　怡　　责任校对：张惠兰
印　　刷：中国电影出版社印刷厂　　版　　次：2018 年 10 月第 1 版第 1 次印刷
开　　本：186mm×240mm　1/16　　印　　张：26.75
书　　号：ISBN 978-7-111-60868-4　　定　　价：89.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88379426　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzit@hzbook.com

Preface 前 言

最近几年，随着 DevOps 和以 Docker 为主的容器技术的发展，云原生应用架构和微服务变得流行起来。云原生包含的内容很多，如 DevOps、持续交付、微服务、敏捷等，本书关注的是其中的微服务。在大概三年前，我在互联网上查找关于微服务落地的方案，搜索到了 Spring 社区推出的 Spring Cloud 项目，在那个时候就开始关注 Spring Cloud，发现 Spring Cloud 基于 Spring Boot，引入依赖后开箱即用，使用非常方便。当时 Spring Cloud 中的组件数量和成熟度远不如现今，Spring Cloud 的版本为 Brixton。后来我在项目中尝试使用 Spring Cloud，主要用了 Spring Cloud Config 和 Spring Cloud Stream，使用过程中发现这两个组件在易用性、功能性等各方面都令人满意，慢慢地便在项目中铺开使用。

在应用 Spring Cloud 的过程中，我见证了它的不断完善和丰富。在其间也遇到了一些“坑”，通过源码分析才解决了一些问题。Spring Cloud 并没有重复造轮子，这些组件有些是 Spring Cloud 的全新项目，如 Spring Cloud Gateway、Spring Cloud Config 等，还有很多是基于业界现有的开源组件，如 Netflix 的合集 Netflix Ribbon 等。

在 2017 年下半年的时候，我开始对每个组件进行梳理，深入到每个组件的实现原理和源码。毕竟 Spring Cloud 中包含了众多组件，我断断续续花了半年时间把各个组件大概梳理了一遍，没想到这些积累成为了本书的写作基础。

本书详细介绍 Spring Cloud 相关组件及其在微服务架构中的应用。全书共 13 章，第 1 章介绍微服务架构相关的基本概念；第 2 章介绍 Spring Cloud 中包含的组件以及 Spring Cloud 约定的上下文；第 3 章介绍 Spring Cloud 的基础 Spring Boot，包括如何构建一个 Spring Boot 服务、Spring Boot 的配置等；第 4～13 章详细讲解 Spring Cloud 组件，包括 Eureka（服务注册与发现）、OpenFeign（声明式 RESTful 客户端）、Hystrix（断路器）、Ribbon（客户端负载均衡器）、Gateway（API 网关）、Config（配置中心）、Stream（消息驱动）、Bus（消息总线）、Security（认证与授权）、Sleuth（服务链路追踪）。本书的目标是深入到 Spring Cloud 组件实现的技术内幕，并介绍了进阶应用的思路，为读者提供使用 Spring Cloud 进行微服务架构实践的参考。

本书在介绍 Spring Cloud 中的重要组件时，从基础应用的案例着手，尽可能将这类组件的设计思路和实现原理讲清楚，以帮助读者加深理解，并结合源码讲解组件的实现原理，最后还介绍了组件的进阶功能与应用。本书适合具有一些 Java 基础的开发人员，特别适合正在尝试微服务实践并想要深入了解 Spring Cloud 各个组件原理的开发人员和架构师。书中的很多案例都提供了源代码，可以随时下载，下载地址为：

- github 地址：https://github.com/Advanced-SpringCloud/cloud-book
- gitee 地址：https://gitee.com/Advanced-SpringCloud/cloud-book

本书最终由三个人共同完成，具体分工如下：第 1、8、9、11、13 章由朱荣鑫编写，第 2、5、7、10 章由张天编写，第 3、4、6、12 章由黄迪璇编写，全书由朱荣鑫统稿。能够完成本书需要感谢很多人，丁二玉老师在本书的撰写过程中提供了很多内容组织方面的建议，花了很多休息时间帮助审稿，非常感谢丁老师的大力帮助；感谢笔者所在的公司，一个年轻而富有活力的公司，为我们提供了很好的平台，从而积累了很多微服务架构实践的经验；感谢机械工业出版社的吴怡编辑及其他工作人员，为本书投入了很多精力。由于时间有限，书中难免存在一些问题，请读者不吝赐教。

朱荣鑫

2018 年 5 月

Contents 目 录

第 1 章 Chapter 1

微服务架构介绍

近年来，微服务架构一直是互联网技术圈的热点之一，越来越多的互联网应用都采用了微服务架构作为系统构建的基础，很多新技术和理念如 Docker、Kubernetes、DevOps、持续交付、Service Mesh 等也都在关注、支持和跟随微服务架构的发展。

本章将会概要性地介绍微服务架构：包括微服务架构是如何演进的，微服务架构的主要流派，当前主流的云原生应用与微服务之间的关系等。

1.1 微服务架构的出现

从单体应用架构发展到 SOA 架构，再到微服务架构，应用架构经历了多年的不断演进。微服务架构不是凭空产生的，而是技术发展的必然结果，分布式云平台的应用环境使得微服务代替单体应用成为互联网大型系统的架构选择。目前，虽然微服务架构还没有公认的技术标准和规范草案，但业界已经有了一些很有影响力的开源微服务架构解决方案，在进行微服务化开发或改造时可以进行相应的参考。

1.1.1 单体应用架构

与微服务架构对比的是传统的单体应用。Web 应用程序发展的早期，大部分 Web 工程是将所有的功能模块打包到一起部署和运行，例如 Java 应用程序打包为一个 war 包，其他语言（Ruby、Python 或者 C++）编写的应用程序也有类似的做法。单体应用的实现架构类似于图 1-1 中的电影售票系统。

这个电影售票系统采用分层架构，按照调用顺序，从上到下为表示层、业务层、数据访问（DAO）层、DB 层。表示层负责用户体验；业务层负责业务逻辑，包括电影、订单和

用户三个模块；数据访问层负责 DB 层的数据存取，实现增删改查的功能。业务层定义了应用的业务逻辑，是整个应用的核心。在单体应用中，所有这些模块都集成在一起，这样的系统架构就叫做单体应用架构，或称为巨石型应用架构。单体应用是最早的应用形态，开发和部署都很简单。在中小型项目中使用单体应用架构，能体现出其优势，且单体应用的整体性能主要依赖于硬件资源和逻辑代码实现，应用架构自身不需要特别关注。

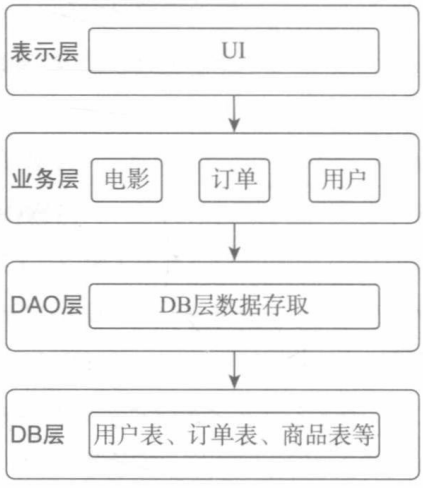


图 1-1 电影售票系统的架构图

单体应用的集成非常简洁，IDE 集成开发环境（如 Eclipse）和其他工具都擅长开发一个简单应用；单体应用易于调试，由于一个应用包含所有功能，所以只需要简单运行此应用即可进行开发测试；单体应用易于部署，只需要把应用打包，拷贝到服务器端即可；通过负载均衡器，运行多个服务实例，单体应用可以轻松实现应用扩展。

但是，随着应用项目变得复杂、开发团队不断扩张之后，单体应用的不足和弊端将会变得很明显，主要有如下不足：

- **可靠性：**每个 bug 都可能会影响到整个应用的可靠性。因为所有模块都运行在一个进程中，任何一个模块中的一个 bug，比如内存泄露，都可能拖垮整个进程。
- **复杂性高：**单体应用巨大的代码库可能会令人望而生畏，特别是对团队新成员来说。应用难以理解和迭代，进而导致开发速度减慢。由于没有清晰的模块边界，模块化会逐渐消失。
- **持续部署困难：**巨大的单体应用本身就是频繁部署的一大障碍。为了更新一个组件，必须重新部署整个应用。这会中断那些可能与更改无关的后台任务（例如 Java 应用中的 Quartz 任务），同时可能引发其他问题。另外，未被更新的组件有可能无法正常启动。重新部署会增加风险，进而阻碍频繁更新。
- **扩展能力受限：**单体架构只能进行一维扩展。一方面，它可以通过运行多个应用服务实例来增加业务容量，实现扩展。但另一方面，不同的应用组件有不同的资源需求：有的是 CPU 密集型的，有的是内存密集型的。单体架构无法单独扩展每个组件。
- **阻碍技术创新：**单体应用往往使用统一的技术平台或方案解决所有问题，团队的每个成员都必须使用相同的开发语言和架构，想要引入新的框架或技术平台非常困难。单体架构迫使团队长期使用在开发初期选定的技术栈，比如选择了 JVM 的语言，此时以非 JVM 语言编写的组件就无法在该单体架构的应用中使用。

1.1.2 SOA 架构

面向服务的架构（SOA）是 Gartner 于 20 世纪 90 年代中期提出的。2002 年 12 月，Gartner 提出“面向服务的架构”是现代应用开发领域最重要的课题之一。

SOA 的核心主体是服务，其目标是通过服务的流程化来实现业务的灵活性。服务就像一堆“元器件”，这些元器件通过封装形成标准服务，它们有相同的接口和语义表达规则。但服务要组装成一个流程和应用，还需要有效的“管理”，包括如何注册服务、如何发现服务、如何包装服务的安全性和可靠性，这些就是 SOA 治理。SOA 治理是将 SOA 的一堆元器件进行有效组装。这是形成一个“产品”的关键，否则那些永远是一堆元器件，而无法形成一个有机整体。

完整的 SOA 架构由五大部分组成：基础设施服务、企业服务总线、关键服务组件、开发工具、管理工具等。

- **基础设施：**为整个 SOA 组件和框架提供一个可靠的运行环境，以及服务组件容器，它的核心组件是应用服务器等基础软件支撑设施，提供运行期完整、可靠的软件支撑。
- **企业服务总线：**提供可靠消息传输、服务接入、协议转换、数据格式转换、基于内容的路由等功能，屏蔽了服务的物理位置、协议和数据格式。
- **关键服务组件：**SOA 在各种业务服务组件的分类。
- **开发工具和管理工具：**提供完善的、可视化的服务开发和流程编排工具，包括服务的设计、开发、配置、部署、监控、重构等完整的 SOA 项目开发生命周期。

具体来说，就是在分布式的环境中，将各种功能都以服务的形式提供给最终用户或者其他服务。企业级应用的开发多采用面向服务的体系架构来满足灵活多变、可重用性的需求。它将应用程序的不同功能单元（称为服务）通过这些服务之间定义良好的接口和契约联系起来。接口采用中立的方式进行定义，它应该独立于实现服务的硬件平台、操作系统和编程语言。这使得构建在各种各样的系统中的服务可以以一种统一和通用的方式进行交互。

SOA 是在企业计算领域中提出的，目的是要将紧耦合的系统，划分为面向业务的、粗粒度、松耦合、无状态的服务。服务发布出来供其他服务调用，一组互相依赖的服务就构成了 SOA 架构的系统。基于这些基础的服务，可以将业务过程用类似 BPEL（业务流程执行语言）流程的方式编排起来，BPEL 反映的是业务处理的过程，这些过程对于业务人员更为直观。企业还需要一些服务治理的工具，比如服务注册库、监控管理等。在企业计算领域，如果不是交易系统的话，并发量都不是很大，所以大多数情况下，一台服务器就可以容纳许多的服务，这些服务采用统一的基础设施，可能都运行在一个应用服务器的进程中。虽然说是 SOA 架构，但还可能是单一的系统。

1.1.3　微服务架构

微服务最早是由 Martin Fowler 与 James Lewis 于 2014 年共同提出，需要了解细节的读者可以阅览 https://martinfowler.com/articles/microservices.html。其实 Martin 先生并没有给出明确的微服务定义，根据其描述，微服务的定义可以概括如下：微服务架构是一种使用一系列粒度较小的服务来开发单个应用的方式；每个服务运行在自己的进程中；服务间采

用轻量级的方式进行通信（通常是 HTTP API）；这些服务是基于业务逻辑和范围，通过自动化部署的机制来独立部署的，并且服务的集中管理应该是最低限度的，即每个服务可以采用不同的编程语言编写，使用不同的数据存储技术。

如今，微服务架构已经不是一个新概念了，很多业界前沿互联网公司的实践表明，微服务是一种渐进式的演进架构，是企业应对业务复杂性，支持大规模持续创新行之有效的架构手段。

1. 组成

微服务架构是一种比较复杂、内涵丰富的架构模式，它包含很多支撑“微”服务的具体组件和概念，其中一些常用的组件及其概念如下：

- **服务注册与发现：**服务提供方将己方调用地址注册到服务注册中心，让服务调用方能够方便地找到自己；服务调用方从服务注册中心找到自己需要调用的服务的地址。
- **负载均衡：**服务提供方一般以多实例的形式提供服务，负载均衡功能能够让服务调用方连接到合适的服务节点。并且，服务节点选择的过程对服务调用方来说是透明的。
- **服务网关：**服务网关是服务调用的唯一入口，可以在这个组件中实现用户鉴权、动态路由、灰度发布、A/B 测试、负载限流等功能。
- **配置中心：**将本地化的配置信息（Properties、XML、YAML 等形式）注册到配置中心，实现程序包在开发、测试、生产环境中的无差别性，方便程序包的迁移。
- **集成框架：**微服务组件都以职责单一的程序包对外提供服务，集成框架以配置的形式将所有微服务组件（特别是管理端组件）集成到统一的界面框架下，让用户能够在统一的界面中使用系统。
- **调用链监控：**记录完成一次请求的先后衔接和调用关系，并将这种串行或并行的调用关系展示出来。在系统出错时，可以方便地找到出错点。
- **支撑平台：**系统微服务化后，各个业务模块经过拆分变得更加细化，系统的部署、运维、监控等都比单体应用架构更加复杂，这就需要将大部分的工作自动化。现在，Docker 等工具可以给微服务架构的部署带来较多的便利，例如持续集成、蓝绿发布、健康检查、性能健康等等。如果没有合适的支撑平台或工具，微服务架构就无法发挥它最大的功效。

2. 优点

微服务架构模式有很多优势可以有效解决单体应用扩大之后出现的大部分问题。首先，通过将巨大单体式应用分解为多个服务的方法解决了复杂性问题。在功能不变的情况下，应用分解为多个可管理的模块或服务。每个服务都有一个用 RPC 或者消息驱动 API 定义清楚的边界。微服务架构模式为采用单体式编码方式很难实现的功能提供了模块化的解决方案。由此，单个服务变得很容易开发、理解和维护。

其次，微服务架构模式使得团队并行开发得以推进，每个服务都可以由专门开发团队

来开发。不同团队的开发者可以自由选择开发技术，提供 API 服务。这种自由意味着开发者不需要被迫使用之前采用的过时技术，他们可以选择最新的技术。甚至于，因为服务都是相对简单的，即使用新技术重写以前的代码也不是很困难的事情。

再次，微服务架构模式中每个微服务独立都是部署的。理想情况下，开发者不需要协调其他服务部署对本服务的影响。这种改变可以加快部署速度。UI 团队可以采用 AB 测试，快速地部署变化。微服务架构模式使得持续化部署成为可能。

最后，微服务架构模式使得每个服务易于独立扩展。

3. 挑战

微服务的一些想法是好的，但在实践中也会呈现出其复杂性，具体如下：

- **运维要求较高**。更多的服务意味着需要更多的运维投入。在单体架构中只需要保证一个应用的正常运行即可；而在微服务中，需要保证几十甚至几百个服务的正常运行与协作，这带来了巨大的挑战。
- **分布式固有的复杂性**。使用微服务构建的是分布式系统。对于一个分布式系统来说，系统容错、网络延迟、分布式事务等都会带来巨大的挑战。
- **接口调整成本高**。微服务之间通过接口进行通信。如果修改某个微服务的 API，可能所有使用了该接口的微服务都需要做调整。
- **重复劳动**。很多服务可能都会使用到相同的功能，而这个功能并没有达到分解为一个微服务的程度，这个时候，可能各个服务都会开发这一功能，导致代码重复。
- **可测试性的挑战**。在动态环境下，服务间的交互会产生非常微妙的行为，难以进行可视化及全面测试。

1.2　微服务架构的流派

常见的微服务架构方案有四种，分别是 ZeroC IceGrid、基于消息队列、Docker Swarm 和 Spring Cloud。下面分别介绍这四种方案。

1. ZeroC IceGrid

ZeroC IceGrid 是基于 RPC 框架 Ice 发展而来的一种微服务架构，Ice 不仅仅是一个 RPC 框架，它还为网络应用程序提供了一些补充服务。Ice 是一个全面的 RPC 框架，支持 C++、C#、Java、JavaScript、Python 等语言。IceGrid 具有定位、部署和管理 Ice 服务器的功能，具有良好的性能与分布式能力，下面具体介绍 IceGrid 的功能。

Ice 的 DNS。DNS 用于将域名信息映射到具体的 IP 地址，通过域名得到该域名对应的 IP 地址的过程叫做域名解析。IceGrid 为 Ice 提供了类似的服务：它允许 Ice 客户端通过简单的名称来查找 Ice 对象。Ice 客户端可以通过提供此对象的完整寻址信息来访问服务器中的 Ice 对象，例如 chatRoom1 ： ssl -h demo.zeroc.com -p 10000。这样的硬编码虽然很简单，

但缺乏灵活性。因为需要为 Ice 服务器（本例中端口为 10000）选择一个固定的端口号，因此将 Ice 服务器移到不同的主机上需要更新其客户端。

IceGrid 提供了对这种寻址信息使用符号名称的选项，例如 chatRoom1@chatRoomHost。当 Ice 客户端尝试访问 chatRoomHost 中的对象时，它会要求 IceGrid 提供与此符号名称关联的实际地址。例如，IceGrid 返回 -h demo.zeroc.com -p 65431，客户端就可以直接并透明地连接到服务器。IceGrid 架构如图 1-2 所示。

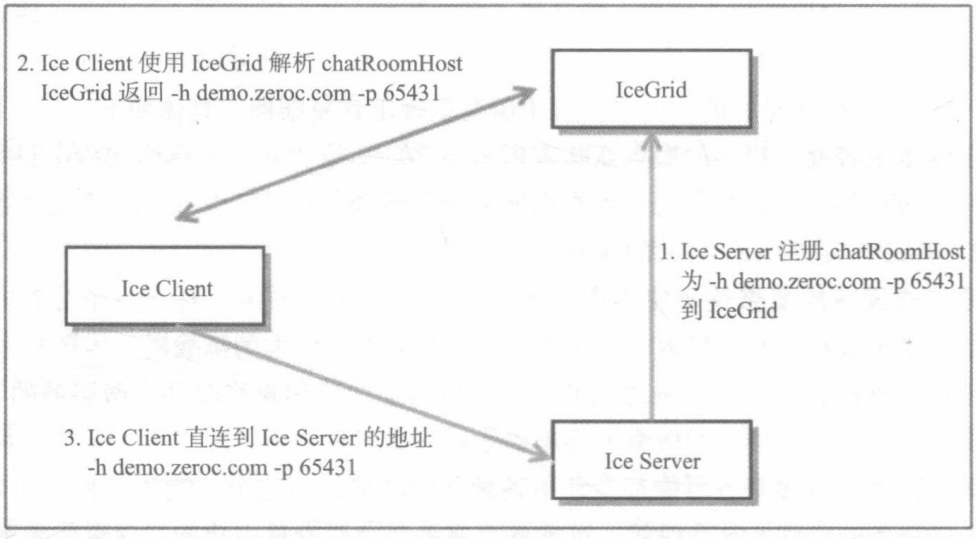


图 1-2　IceGrid 架构图

服务器部署。若直接通过 IP 地址 + 端口号的方式，当 Ice 客户端尝试连接到未运行的服务器时，连接将会失败。通过符号或间接寻址，IceGrid 有机会检查目标服务器是否正在运行，并在服务器未运行时启动或重新启动服务器。可以配置 IceGrid 以各种方式启动服务器：手动（通过管理命令）、按需（无论何时请求服务器）和 IceGrid 始终保持服务器运行。

服务器的复制。IceGrid 允许部署同一服务器的多个副本，并可以配置 IceGrid 将符号名称解析到此服务器副本的策略。可以在所有副本中对客户端进行负载平衡，或者使用主备配置，客户端只要保持可用状态，就使用主备配置。

管理和监控。IceGrid 的管理工具可以完全控制已部署的应用程序。诸如启动服务器或修改配置设置等活动只需单击鼠标即可。图 1-3 为 IceGrid 管理工具的界面。

IceGrid 当前最新的版本为 3.7.1，在 3.6 版本之后增加了容器化的运行方式。总的来说，IceGrid 作为微服务架构早期的实践方案，其流行时间并不是很久，当前国内选用这种微服务架构方案的公司非常少。

2. 基于消息队列

在微服务架构的定义中讲到，各个微服务之间使用“轻量级”的通信机制。所谓轻量级，是指通信协议与语言无关、与平台无关。微服务之间的通信方式有两种：同步和异步。

同步方式有 RPC，REST 等；除了标准的基于同步通信方式的微服务架构外，还有基于消息队列异步方式通信的微服务架构。

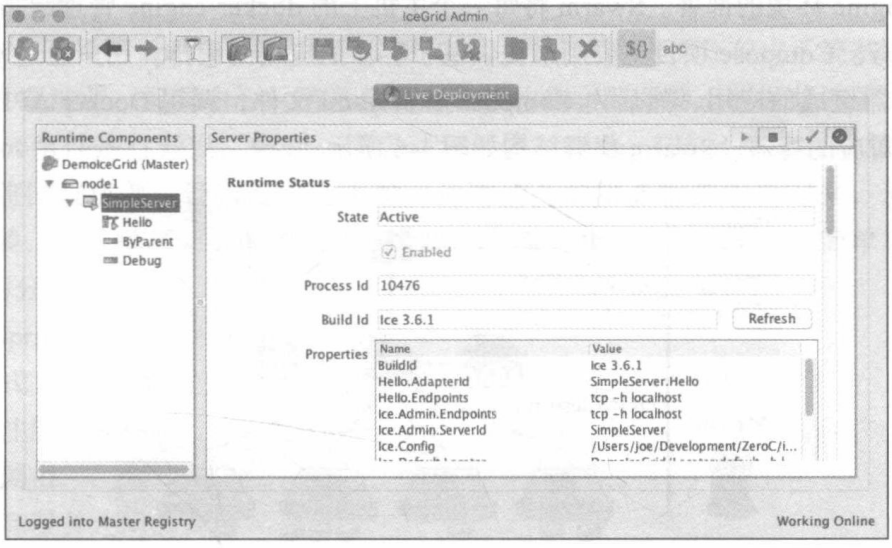


图 1-3　IceGrid Admin

在基于消息队列的微服务架构方式中，微服务之间采用发布消息与监听消息的方式来实现服务之间的交互。图 1-4 是一个简单的电商系统中商品服务、用户服务、订单服务和库存服务之间的交互示意图，可以看到消息中间件（MQ）是关键，它负责连通各个微服务，承担了整个系统互联互通的重任。

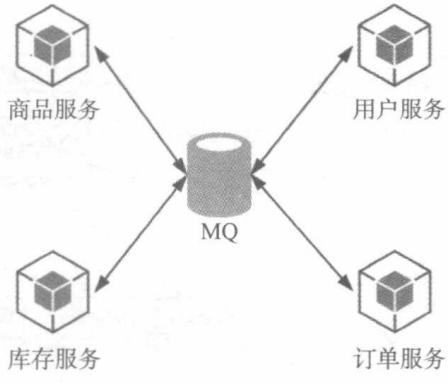


图 1-4　基于消息队列的微服务架构

基于消息队列的微服务架构是全异步通信模式的一种设计，各个组件之间没有直接的耦合关系，也不存在服务接口与服务调用的说法，服务之间通过消息来实现彼此的通信与业务流程的驱动。基于消息队列的微服务架构应用的案例并不多，更多地体现为一种与业务相关的设计经验，每个公司都有不同的实现方式，缺乏公认的设计思路与参考架构，也没有形成一个知名的开源平台。因此，如要实施这种微服务架构，需要项目组自己从零开始去设计实现一个微服务架构基础平台，这可能会造成项目的成本较高且风险较大，决策之前需要进行全盘思考与客观评价。

3. Docker Swarm

Swarm 项目是 Docker 公司发布的三剑客中的一员，用来提供容器集群服务，目的是更好地帮助用户管理多个 Docker Engine，方便用户使用。通过把多个 Docker Engine 聚集在一起，形成一个大的 Docker Engine，对外提供容器的集群服务。同时这个集群对外提供 Swarm API，用户可以像使用 Docker Engine 一样使用 Docker 集群。

如图 1-5 所示 Docker 三剑客包括：Machine、Compose 和 Swarm。通过 Machine 可以在不同云平台上创建包含 docker-engine 的主机。Machine 通过 driver 机制，支持多个平台的 docker-engine 环境的部署。Swarm 将每一个主机上的 docker-engine 管理起来，对外提供容器集群服务。Compose 项目主要用来提供基于容器的应用的编排。用户通过 yaml 文件描述由多个容器组成的应用，然后由 Compose 解析 yaml 文件，调用 Docker API，在 Swarm 集群上创建对应的容器。Swarm 集群结构如图 1-6 所示。

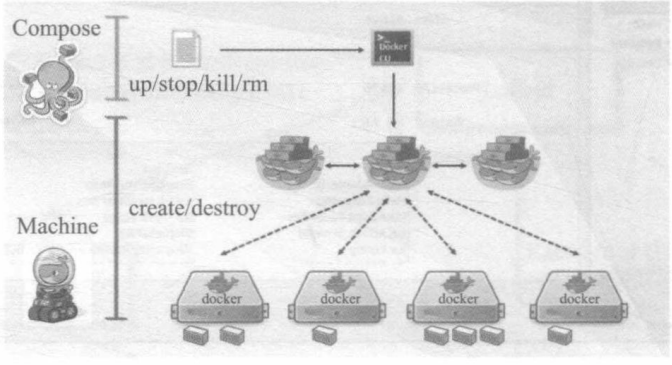


图 1-5　Docker 三剑客

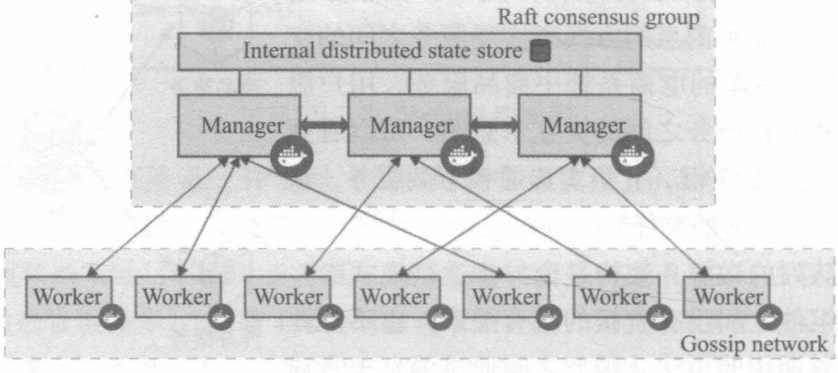


图 1-6　Docker Swarm 结构

从图 1-6 中我们看到一个 Swarm 集群中有两种角色的节点：

- Manager：负责集群的管理、集群状态的维持及将任务（Task）调度到工作节点上等。
- Worker：承载运行在 Swarm 集群中的容器实例，每个节点主动汇报其上运行的任务并维持同步状态。

Docker Swarm 对外提供 Docker API，自身轻量，学习成本、二次开发成本都比较低，是一个插件式框架。从功能上讲，Swarm 是类似于 Google 开源的 Kubernetes 微服务架构平台的一个产品。

4. Spring Cloud

Spring Cloud 是一个基于 Spring Boot 实现的云应用开发工具，是一系列框架的集合，