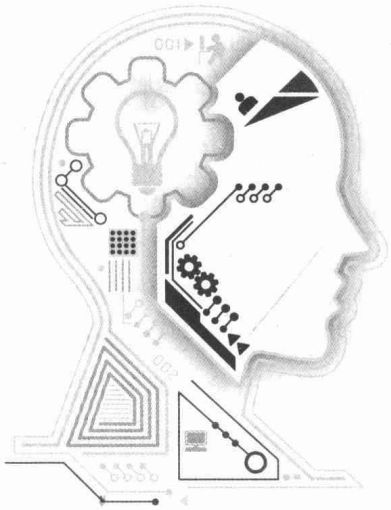
Broadview®
www.broadview.com.cn

Spring响应式微服务

Spring Boot 2+Spring 5+Spring Cloud实战

郑天民◎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



Spring响应式微服务

Spring Boot 2+Spring 5+Spring Cloud实战

郑天民◎著

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

内 容 简 介

本书主要包含构建响应式微服务架构过程中所应具备的技术体系和工程实践。围绕响应式编程和微服务架构的整合，我们将讨论如何使用 Reactor 响应式编程框架、如何构建响应式 RESTful 服务、如何构建响应式数据访问组件、如何构建响应式消息通信组件、如何构建响应式微服务架构，以及如何测试响应式微服务架构等核心主题，并基于这些核心主题给出具体的案例分析。

本书面向立志于成为微服务架构师（尤其是响应式微服务架构师）的后端服务开发人员，读者不需要有很高的技术水平，也不限于特定的开发语言，但熟悉 Java EE 常见技术并掌握一定异步编程模型和分布式架构的基本概念有助于更好地理解书中的内容。同时，本书也可以供具备不同技术体系的架构师同行参考，希望能给日常研发和管理工作带来启发和帮助。

图书在版编目（CIP）数据

Spring 响应式微服务 : Spring Boot 2+Spring 5+Spring Cloud 实战 / 郑天民著. —北京：电子工业出版社，2019.6

ISBN 978-7-121-36383-2

Ⅰ.①S… Ⅱ.①郑… Ⅲ.①JAVA 语言－程序设计②互联网络－网络服务器 Ⅳ.①TP312.8②TP368.5

中国版本图书馆 CIP 数据核字（2019）第 076225 号

策划编辑：张春雨
责任编辑：李利健
印　　刷：山东华立印务有限公司
装　　订：山东华立印务有限公司
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：17.25　　字数：385 千字
版　　次：2019 年 6 月第 1 版
印　　次：2019 年 6 月第 1 次印刷
定　　价：75.00 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

当下互联网行业飞速发展，快速的业务更新和产品迭代也给系统开发过程和模式带来新的挑战。在这个时代背景下，以 Spring Cloud 为代表的微服务架构实现技术应运而生。微服务架构是一种分布式系统，在业务、技术和组织等方面具备相应优势的同时，也不得不面临分布式系统所固有的问题。确保微服务系统的即时响应性和服务弹性是我们构建微服务架构的一大挑战。幸运的是，Spring 框架的开发人员已经创建了一个崭新的、支持响应式的项目版本，用来支持响应式微服务架构的设计和开发。通过构建响应式微服务架构，我们将在传统微服务架构的基础上提供即时响应性和服务弹性。

本书从响应式编程和微服务架构的基本概念开始并逐步展开。你将了解响应式的基本原理，以及 Spring 5 框架所集成的 Project Reactor 响应式开发框架。同时，你将进一步了解如何构建响应式 RESTful 服务、如何构建响应式数据访问组件、如何构建响应式消息通信组件、如何构建响应式微服务架构，以及如何测试响应式微服务架构等核心主题。所有这些内容都将应用于一个简明而又完整的示例项目，确保你能够将所学到的技能付诸于实践。

本书主要包含响应式微服务架构实现过程中所应具备的技术体系和工程实践，在组织结构上分如下 8 章内容。

第 1 章“直面响应式微服务架构”，作为全书的开篇，围绕响应式微服务架构的概念和构建方式展开讨论。通过对比传统的编程方法和响应式编程方法引出响应式编程的核心概念，并引用响应式宣言来阐述响应式系统所应具备的基本系统特性和维度。同时，本章在介绍传统微服务架构的基础上，分析了响应式微服务架构的设计原则，然后对响应式编程和微服务架构进行了整合。

第 2 章“响应式编程模型与 Reactor 框架”，本章全面介绍响应式编程模型并引出了响应式流规范，Reactor 框架为我们提供了一整套实现该规范的具体实现。我们在介绍 Reactor 框架中 Mono 和 Flux 这两个核心组件的基础上，进一步提供了一系列强大的操作符来操作这些组件。本章最后还对 Reactor 框架中的背压机制做了简单介绍，Reactor 框架提供了 4 种背压处理策略以满足不同场景的需求。

第 3 章“构建响应式 RESTful 服务”，要想构建响应式微服务架构，首先需要构建单个响应式微服务。在 Spring 5 中引入了全新的响应式服务构建框架 Spring WebFlux，支持使用注解编程模型和函数式编程模型两种方式来构建响应式 RESTful 服务。本章基于 Spring Boot，对 Spring WebFlux 框架做了全面介绍。

第 4 章“构建响应式数据访问组件”，对于响应式微服务架构而言，数据访问也是构建全栈响应式系统的重要一环。为此，Spring Data 框架也专门提供了 Spring Reactive Data 组件用来创建响应式数据访问层组件。在本章中，我们重点就 MongoDB 和 Redis 这两个支持响应式特性的 NoSQL 数据库分别给出了如何使用 Spring Reactive Data 来实现响应式数据访问的基本步骤和代码示例。

第 5 章“构建响应式消息通信组件”，本章内容围绕构建响应式微服务架构的另一个重要主题展开讨论，即响应式消息通信。我们使用 Reactive Spring Cloud Stream 框架来实现响应式消息通信组件。本章先从事件驱动架构和模型出发，引出了 Spring Cloud 家族中实现消息通信的 Spring Cloud Stream 框架。然后对 Spring Cloud Stream 进行升级，结合响应式编程模型全面介绍 Reactive Spring Cloud Stream 框架的使用方法。

第 6 章“构建响应式微服务架构”，本章是全书的重点章节，我们通过使用 Spring Cloud 框架来实现响应式微服务架构。我们从服务治理、负载均衡、服务容错、服务网关、服务配置和服务监控共 6 大主题出发全面讨论了响应式微服务架构的核心组件及其实现方案。对于每个组件的介绍，我们都包含了使用该组件的具体方法以及相应的代码示例。同时，我们还专门使用一节内容来介绍 WebClient 这一响应式服务调用的实现工具。

第 7 章“测试响应式微服务架构”，本章首先介绍初始化测试环境的准备工作，然后分别给出了测试响应式微服务架构中一系列独立层组件的方法和示例，即从数据流层出发，分别对基于响应式 MongoDB 的 Repository 层、Service 层以及 Controller 层进行测试。

第 8 章“响应式微服务架构演进案例分析”，本章作为全书的最后一章，通过一个完整的案例分析全面介绍了构建一个响应式微服务系统的各个方面。在介绍该案例时，首先采用了传统的微服务架构来实现该案例。然后，在传统微服务架构构建完毕的基础上，重点对如何向响应式微服务架构演进的方法和过程做了具体展开。一方面，我们需要更新基础设施类服务，另一方面，需要完成对数据访问方式、事件通信方式、服务调用方式的全面升级。这里涉及响应式 WebFlux、响应式 MongoDB 和 Redis、响应式 Spring Cloud Stream 等响应式组件的使用方式和最佳实践。

本书面向想成为微服务架构师尤其是响应式微服务架构师的服务开发人员，读者不需要有很深的技术水平，也不限于特定的开发语言，但如果你熟悉 Java EE 常见技术并掌握一定异步编程模型和分布式架构的基本概念有助于更好地理解书中的内容。通过对本书的系统学习，读者将对响应式微服务架构的技术体系和实现方式有全面且深入的了解，为后续的工作和学习做好准备。

在撰写本书的过程中，感谢我的家人，特别是我的妻子章兰婷女士在我占用大量晚上和周末时间的情况下，能够给予极大的支持和理解。感谢以往以及现在公司的同事们，身处在

业界领先的公司和团队中，让我得到很多学习和成长的机会，没有大家的帮助，不可能有这本书的诞生。最后，特别感谢电子工业出版社的张春雨编辑，这本书能够顺利出版，离不开他的帮助。

由于作者水平和经验有限，书中难免有疏漏和错误之处，恳请读者批评、指正。

郑天民

2019 年 3 月于杭州钱江世纪城

目 录

第 1 章 直面响应式微服务架构 1

1.1 响应式系统核心概念 1

1.1.1 从传统编程方法到响应式编程方法 1

1.1.2 响应式宣言与响应式系统 4

1.2 剖析微服务架构 6

1.2.1 分布式系统与微服务架构 6

1.2.2 服务拆分与集成 8

1.2.3 微服务架构的核心组件 11

1.2.4 微服务架构技术体系 13

1.3 构建响应式微服务架构 15

1.3.1 响应式微服务架构设计原则 15

1.3.2 整合响应式编程与微服务架构 18

1.4 全书架构 19

1.5 本章小结 20

第 2 章 响应式编程模型与 Reactor 框架 21

2.1 响应式编程模型 21

2.1.1 流 22

2.1.2 背压 24

2.1.3 响应式流 25

2.2 Reactor 框架 28

2.2.1 响应式编程实现技术概述 28

2.2.2 引入 Reactor 框架 31

2.3 创建 Flux 和 Mono 34

2.3.1 创建 Flux 34

2.3.2 创建 Mono 37

2.4 Flux 和 Mono 操作符 39

2.4.1 转换操作符 39

2.4.2 过滤操作符 43

2.4.3 组合操作符 46

2.4.4 条件操作符 49
2.4.5 数学操作符 52
2.4.6 Observable 工具操作符 54
2.4.7 日志和调试操作符 56
2.5 Reactor 框架中的背压机制 58
2.6 本章小结 60
第 3 章 构建响应式 RESTful 服务 61
3.1 使用 Spring Boot 2.0 构建微服务 61
3.1.1 Spring Boot 基本特性 61
3.1.2 基于 Spring Boot 的第一个 RESTful 服务 63
3.1.3 使用 Actuator 组件强化服务 67
3.2 使用 Spring WebFlux 构建响应式服务 80
3.2.1 使用 Spring Initializer 初始化响应式 Web 应用 80
3.2.2 对比响应式 Spring WebFlux 与传统 Spring WebMvc 82
3.2.3 使用注解编程模型创建响应式 RESTful 服务 84
3.2.4 使用函数式编程模型创建响应式 RESTful 服务 88
3.3 本章小结 93
第 4 章 构建响应式数据访问组件 94
4.1 Spring Data 数据访问模型 94
4.1.1 Spring Data 抽象 95
4.1.2 集成 Spring Data JPA 98
4.1.3 集成 Spring Data Redis 100
4.1.4 集成 Spring Data Mongodb 103
4.2 响应式数据访问模型 104
4.2.1 Spring Reactive Data 抽象 105
4.2.2 创建响应式数据访问层组件 107
4.3 响应式 MongoDB 108
4.3.1 初始化 Reactive Mongodb 运行环境 109
4.3.2 创建 Reactive Mongodb Repository 112
4.3.3 使用 CommandLineRunner 初始化 MongoDB 数据 112
4.3.4 在 Service 层中调用 Reactive Repository 114
4.4 响应式 Redis 117
4.4.1 初始化 Reactive Redis 运行环境 117
4.4.2 创建 Reactive Redis Repository 120

4.4.3 在 Service 层中调用 Reactive Repository 122
4.5 本章小结 123
第 5 章 构建响应式消息通信组件 124
5.1 消息通信系统简介 125
5.2 使用 Spring Cloud Stream 构建消息通信系统 126
5.2.1 Spring Cloud Stream 基本架构 126
5.2.2 Spring Cloud Stream 中的 Binder 组件 130
5.2.3 使用 Source 组件实现消息发布者 135
5.2.4 使用@StreamListener 注解实现消息消费者 137
5.3 引入 Reactive Spring Cloud Stream 实现响应式消息通信系统 139
5.3.1 Reactive Spring Cloud Stream 组件 139
5.3.2 Reactive Spring Cloud Stream 示例 141
5.4 本章小结 147
第 6 章 构建响应式微服务架构 148
6.1 使用 Spring Cloud 创建响应式微服务架构 148
6.1.1 服务治理 149
6.1.2 负载均衡 154
6.1.3 服务容错 161
6.1.4 服务网关 166
6.1.5 服务配置 173
6.1.6 服务监控 177
6.2 使用 WebClient 实现响应式服务调用 182
6.2.1 创建和配置 WebClient 182
6.2.2 使用 WebClient 访问服务 183
6.3 本章小结 187
第 7 章 测试响应式微服务架构 188
7.1 初始化测试环境 189
7.1.1 引入 spring-boot-starter-test 组件 189
7.1.2 解析基础类测试注解 190
7.1.3 编写第一个测试用例 191
7.2 测试 Reactor 组件 192
7.3 测试响应式 Repository 层组件 194
7.3.1 测试内嵌式 MongoDB 194
7.3.2 测试真实的 MongoDB 197

7.4 测试响应式 Service 层组件 199
7.5 测试响应式 Controller 层组件 201
7.6 本章小结 204
第 8 章 响应式微服务架构演进案例分析 205
8.1 PrescriptionSystem 案例简介 205
8.2 传统微服务架构实现案例 207
8.2.1 构建基础设施类服务 207
8.2.2 构建 Medicine 服务 213
8.2.3 构建 Card 服务 219
8.2.4 构建 Prescription 服务 224
8.3 响应式微服务架构演进案例 237
8.3.1 更新基础设施类服务 237
8.3.2 更新数据访问方式 241
8.3.3 更新事件通信方式 246
8.3.4 更新服务调用方式 251
8.4 本章小结 265
参考文献 266

第1章 直面响应式微服务架构

随着以 Dubbo、Spring Cloud 等框架为代表的分布式服务调用和治理工具的大行其道，以及以 Docker、Kubernetes 等容器技术的日渐成熟，微服务架构（Microservices Architecture）毫无疑问是近年来最热门的一种服务化架构模式。所谓微服务，就是一些具有足够小的粒度、能够相互协作且自治的服务体系。正因为每个微服务都比较简单，仅关注于完成一个业务功能，所以具备技术、业务和组织上的优势[1]。

另外，随着 Spring 5 的正式发布，我们迎来了响应式编程（Reactive Programming）的全新发展时期。Spring 5 中内嵌了响应式 Web 框架、响应式数据访问、响应式消息通信等多种响应式组件，从而极大地简化了响应式应用程序的开发过程和难度。

本章作为全书的开篇，将对微服务架构和响应式系统（Reactive System）的核心概念做简要介绍，同时给出两者之间的整合点，即如何构建响应式微服务架构。在本章最后，我们也会给出全书的组织架构，以便读者能够总览全书。

1.1 响应式系统核心概念

在本节中，我们将带领大家进入响应式系统的世界。为了让大家更好地理解响应式编程和响应式系统的核心概念，我们将先从传统编程方法出发逐步引出响应式编程方法。同时，我们还将通过响应式宣言（Reactive Manifesto）了解响应式系统的基本特性和设计理念。

1.1.1 从传统编程方法到响应式编程方法

在电商系统中，订单查询是一个典型的业务场景。用户可以通过多种维度获取自己已下订单的列表信息和各个订单的明细信息。我们就通过订单查询这一特定场景来分析传统编程

方法和响应式编程方法之间的区别。

1．订单查询场景的传统方法

在典型的三层架构中，图 1-1 展示了基于传统实现方法的订单查询场景时序图。一般用户会使用前端组件所提供的操作入口进行订单查询，然后该操作入口会调用后台系统的服务层，服务层再调用数据访问层，进而访问数据库，数据从数据库中获取之后逐层返回，最后显示在包括前端服务或用户操作界面在内的前端组件上。

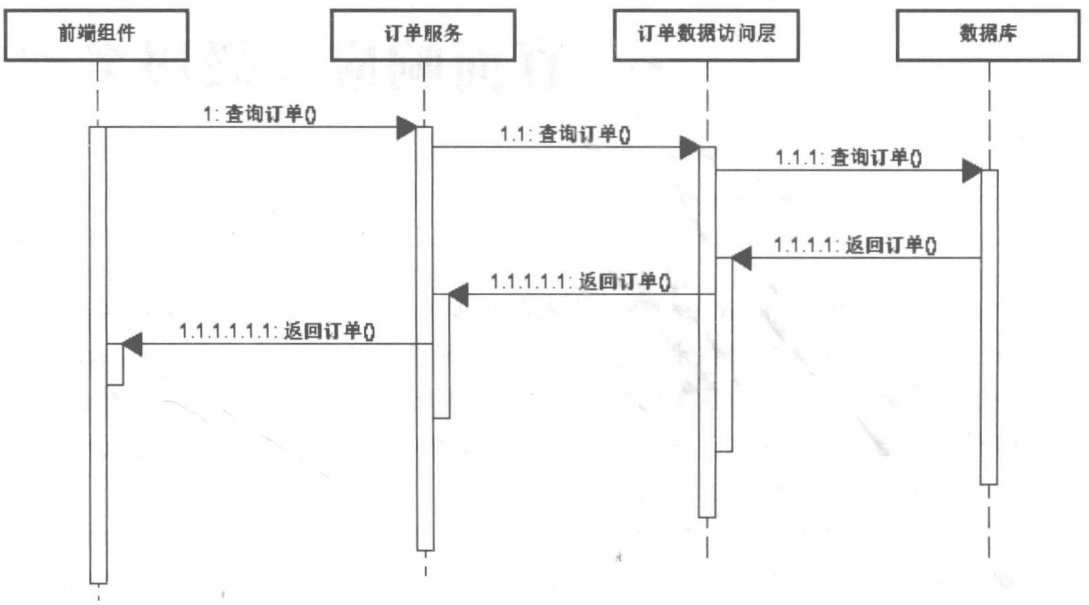


图 1-1　订单查询场景的传统实现方法时序图

显然，在图 1-1 所展示的整个过程中，前端组件通过主动拉取的方式从数据库中获取数据。如果用户不触发前端操作，那么就无法获取数据库中的数据状态。也就是说，前端组件对数据库中的任何数据变更一无所知。

2．订单查询场景的响应式方法

主动拉取数据的方式在某些场景下可以运作得很好，但如果我们希望数据库中的数据一有变化就通知到前端组件，这种方式就不是很合理。这种场景下，我们希望前端组件通过注册机制获取数据变更的事件，图 1-2 展示了这一过程。

在图 1-2 中，我们并不是直接访问数据库来获取数据，而是订阅了 OrderChangedEvent 事件。当订单数据发生任何变化时，系统就会生成这一事件，然后通过一定的方式传播出来。而订阅了该事件的服务就会捕获该事件，从而通过前端组件响应该事件。事件处理的基本步骤涉及对某个特定事件进行订阅，然后等待事件的发生。如果不需要再对该事件做出响应，我们就可以取消对事件的订阅。

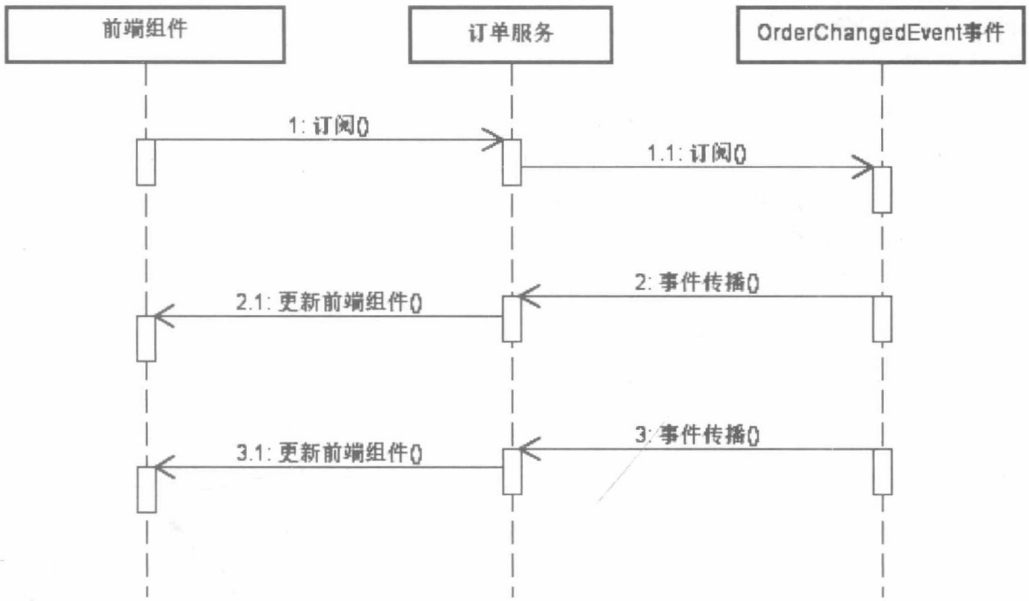


图 1-2　订单查询场景的响应式实现方法时序图

图 1-2 体现的是响应式系统中一种变化传递（Propagation Of Change）思想，即当数据变化之后，会像多米诺骨牌一样，导致直接和间接引用它的其他数据均发生相应变化。一般而言，生产者只负责生成并发出事件，然后消费者来监听并负责定义如何处理事件的变化传递方式。

显然，这些事件连起来会形成一串数据流（Data Stream），如果我们对数据流的每一个事件能够及时做出响应，就会有效提高系统的响应能力。基于数据流是响应式系统的另一个核心特点。

我们再次回到图 1-1，如果从底层数据库驱动，经过数据访问层到服务层，最后到前端组件的这个服务访问链路全部都采用响应式的编程方式，从而搭建一条能够传递变化的管道，这样一旦数据库中的数据有更新，系统的前端组件上就能相应地发生变化。而且，当这种变化发生时，我们不需要通过各种传统调用方式来传递这种变化，而是由搭建好的数据流自动进行传递。

3．传统方法与响应式方法的对比

图 1-1 展示的传统方法和图 1-2 展示的响应式方法具有明显的差异，我们分别从处理过程、线程管理和伸缩性角度做简要对比。

（1）处理过程

传统开发方式下，我们拉取（Pull）数据的变化，这意味着整个过程是一种间歇性、互不相关的处理过程。前端组件不关心数据库中的数据是否有变化。

在响应式开发方式下，一旦对事件进行注册，处理过程只有在数据变化时才会被触发，

类似一种推（Push）的工作方式。

（2）线程管理

在传统开发方式下，线程的生命周期比较长。在线程存活的状态下，该线程所使用的资源都会被锁住。当服务器在同时处理多个线程时，就会存在资源的竞争问题。

在响应式开发方式下，生成事件和消费事件的线程的存活时间都很短，所以资源之间存在较少的竞争关系。

（3）伸缩性

传统开发方式下，系统伸缩性涉及数据库和应用服务器的伸缩，一般我们需要专门采用一些服务器架构和资源来应对伸缩性需求。

在响应式开发方式下，因为线程的生命周期很短，同样的基础设施可以处理更多的用户请求。同时，响应式开发方式同样支持传统开发方式下的各种伸缩性实现机制，并提供了更多的分布式实现选择。图 1-3 展示了事件处理与系统伸缩性之间的关系。

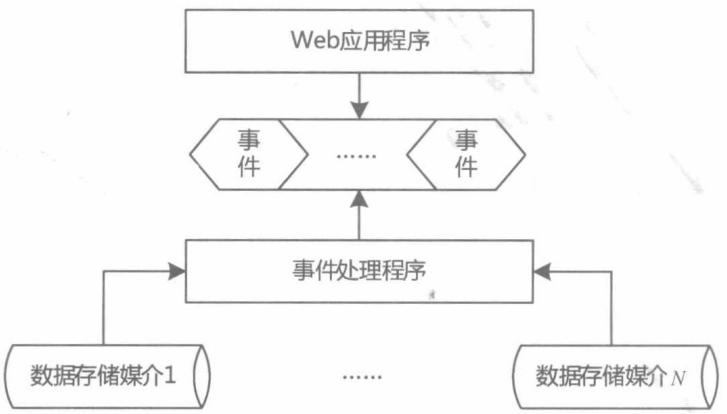


图 1-3　事件处理与系统伸缩性示意图

在图 1-3 中，Web 应用程序和事件处理程序显然可以分别进行伸缩，这为伸缩性实现机制提供了更多的选型余地。

1.1.2　响应式宣言与响应式系统

如同业界的其他宣言一样，响应式宣言是一组设计原则，符合这些原则的系统可以认为是响应式系统。同时，响应式宣言也是一种架构风格，是一种关于分布式环境下系统设计的思考方式，响应式系统也是具备这一架构风格的系统。

1．响应式系统特性

响应式宣言给出了响应式系统所应该具备的特性，包括即时响应性（Responsive）、回弹性（Resilient）、弹性（Elastic）以及消息驱动（Message Driven）。具备这些特性的系统可以称为响应式系统。图 1-4 给出了响应式宣言的图形化描述。

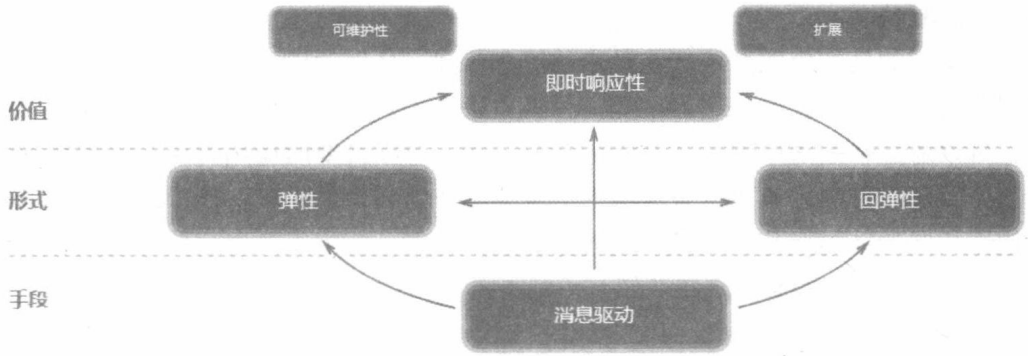


图 1-4　响应式宣言（来自响应式宣言官网）

在图 1-4 中，响应式宣言认为，响应式系统的价值在于提供了即时响应性、可维护性和可扩展性，表现的形式就是回弹性和弹性，而实现的手段则是消息驱动。我们需要对这些名词做一一展开，以下关于响应式系统的各个特性的描述来自响应式宣言中文版，为了在描述上与本书其他内容的统一，部分名称和语句做了调整，读者可访问响应式宣言中文版官方网站获取更多信息。

（1）即时响应性

即时响应性指的是只要有可能，系统就会及时地做出响应。即时响应是可用性和实用性的基石。更加重要的是，即时响应意味着可以快速地检测到问题并有效地对其进行处理。即时响应的系统专注于提供快速、一致的响应，确立可靠的反馈上限，以提供一致的服务质量。这种一致的行为转而将简化错误处理、建立最终用户的信任并促使用户与系统作进一步的互动。

（2）回弹性

回弹性指的是系统在出现失败时依然保持即时响应性。这不仅适用于高可用的任务关键型系统，任何不具备回弹性的系统都将会在发生失败之后丢失即时响应性。回弹性是通过复制、遏制、隔离以及委托来实现的。失败的扩散被遏制在了每个组件内部，与其他组件相互隔离，从而确保系统某部分的失败不会危及整个系统，并能独立恢复。每个组件的恢复都被委托给了另一个内部或外部组件。此外，在必要时可以通过复制来保证高可用性。因此，组件的客户端不再承担组件失败的处理。

（3）弹性

弹性指的是系统在不断变化的工作负载之下依然保持即时响应性。响应式系统可以对输入的速率变化做出反应，比如，通过增加或者减少用于服务这些输入的资源分配。这意味着设计上并没有竞争点和中央瓶颈，系统得以进行组件的分片或者复制，并在它们之间分布输入。通过实现相关的实时性能指标，响应式系统能支持预测式以及响应式的伸缩算法。这些系统可以在常规的硬件以及软件平台上实现高效的弹性。

（4）消息驱动

消息驱动指的是响应式系统依赖异步的消息传递，从而确保松耦合、隔离、位置透明的组件之间有着明确边界。这一边界还提供了将失败作为消息委托出去的手段。使用显式的消息传递，可以通过在系统中塑造并监视消息流队列，并在必要时应用背压，从而实现负载管理、弹性以及流量控制。使用位置透明的消息传递作为通信的手段，使得跨集群或者在单个主机中使用相同的结构成分和语义来管理失败成为可能。非阻塞的通信使得接收者可以只在活动时才消耗资源，从而减少系统开销。

2．响应式的维度

响应式的概念还体现在不同维度上，包含响应事件、响应压力、响应错误和响应用户。

（1）响应事件

基于消息驱动机制，响应式系统可以对事件做出快速响应。

（2）响应压力

响应式系统可以在不同的系统压力下进行灵活响应。当压力较大时，使用更多的资源；当压力变小时，则释放不需要的资源。

（3）响应错误

响应式系统可以优雅地处理错误，监控组件的可用性，并在必要时冗余组件。

（4）响应用户

响应式系统的确能够积极响应用户请求，但当消费者没有订阅事件时，就不会浪费资源进行不必要的处理。

1.2　剖析微服务架构

目前，微服务架构已经成为一种主流的软件开发方法论，它把一种特定的软件应用设计方法描述为能够独立部署的服务套件。本节将对微服务设计原理与架构做精简而全面的介绍。

1.2.1　分布式系统与微服务架构

微服务架构首先表现为一种分布式系统（Distributed System），而分布式系统是传统单块系统（Monolith System）的一种演进。

1．单块系统

在软件技术发展过程的很长一段时间内，软件系统都表现为一种单块系统。时至今日，很多单块系统仍然在一些行业和组织中得到开发和维护。所谓单块系统，简单地讲就是把一个系统所涉及的各个组件都打包成一个一体化结构并进行部署和运行。在 Java EE 领域，这种一体化结构很多时候就体现为一个 WAR 包，而部署和运行的环境就是以 Tomcat 为代表的各种应用服务器。

单块系统有其存在和发展的固有优势。当团队规模并不是太大的时候，一个单块应用可以由一个开发者团队进行独立维护。该团队的成员能够对单块应用进行快速学习、理解和修改，因为其结构非常简单。同时，因为单块系统的表现形式就是一个独立的 WAR 包，想要对它进行集成、部署以及实现无状态集群，相对也比较简单，通常只要采用负载均衡机制并运行该单块系统的多个实例，就能达到系统伸缩性要求。

但在另一方面，随着公司或者组织业务的不断扩张、业务结构的不断变化以及用户量的不断增加，单块系统的优势已无法适应互联网时代的快速发展，面临着越来越多的挑战，例如，如何处理业务复杂度、如何防止代码腐化、如何处理团队协作问题以及如何应对系统伸缩性问题[1]。针对以上集中式单块系统所普遍存在的问题，基本的解决方案就要依赖于分布式系统的合理构建。

2．分布式系统

所谓分布式系统，是指硬件或软件组件分布在不同的网络计算机上，彼此通过一定的通信机制进行交互和协调的系统。我们从这个定义中可以看出分布式系统包含两个区别于单块系统的本质特征：一个是网络，分布式系统的所有组件都位于网络中，对互联网应用而言，则位于更为复杂的互联网环境中；另一个是通信和协调，与单块系统不同，位于分布式系统中的各个组件只有通过约定、高效且可靠的通信机制进行相关协作，才能完成某项业务功能。这是我们在设计和实现分布式系统时首先需要考虑的两个方面。

分布式系统相较于集中式系统而言具备优势的同时，也存在一些我们不得不考虑的特性，包括但不限于网络传输的三态性、系统之间的异构性、数据一致性、服务的可用性等[1]。以上问题是分布式系统的基本特性，我们无法避免，只能想办法进行利用和管理，这就给我们设计和实现分布式系统提出了挑战。微服务架构本质上也是一种分布式系统，但在遵循通用分布式特性的基础上，微服务架构还表现出一定的特殊性。接下来将围绕微服务架构的这些特性展开讨论。

3．微服务架构

Martin Fowler 指出[2]，微服务架构具有以下特点。

（1）服务组件化

组件（Component）是一种可独立替换和升级的软件单元。在日常开发过程中，我们可能会设计和使用很多组件，这些组件可能服务于系统内部，也可能存在于系统所运行的进程之外。而服务就是一种进程外组件，服务之间利用诸如 RPC（Remote Procedure Call，远程过程调用）的通信机制完成交互。服务组件化的主要目的是服务可以独立部署。如果你的应用程序由一个运行在独立进程中的很多组件组成，那么对任何一个组件的改变都将导致必须重新部署整个应用程序。但是如果你把应用程序拆分成很多服务，显然，通常情况下，你只需要重新部署那个改变的服务。在微服务架构中，每个服务运行在其独立的进程中，服务与服务之间采用轻量级通信机制互相沟通。