南昌航空大學

毕业设计（论文）

题　　目： 下一代的反应式编程框架研究与实现

学　　院： 软件学院

专业名称： 软件工程

班级学号： 15201229

学生姓名： 张凤育

指导教师： 李智慧 阮威 储珺

2019 年 6 月

摘要

随着用户需求和用户量的日益增进以及 5G 时代的到来，如何让大型软件系统满足高并发、高负载的访问压力，以及如何快速响应、保持服务 100%可用、弹性适应各种并发环境成为了当今的热点问题。因此具有消息驱动、即时响应、弹性以及回弹性的反应式系统成为开发者讨论的新一轮话题。RxJava 以及 Reactor 都是反应式编程框架的先驱者，他们利用观察者模式，基于发布-订阅的事件响应方式实现了各自的反应式系统。这两者基于函数式编程方式实现，降低了代码的可阅读性，同时也加大了学习成本；其次代码间的耦合度较高，不利于现代系统的分布式部署。因此，本论文基于 Akka Actor 模型构建了新一代反应式编程框架—Flower，将消息作为一等公民，应用天然符合反应式理念的 Actor 模型，将开发者服务封装在 Actor 内部，形成一个完全由消息驱动的反应式编程框架。

本论文主要完成 Flower 框架服务到 Actor 模型的封装、消息处理模式的实现、HTTP 请求支持以及系统测试四个方面工作。服务是最小的执行单元，Flower 框架为每一个用户服务绑定一个或多个 Actor 对象形成服务封装。服务之间没有关联，通过传递消息进行通信；流程定义联系多个服务形成服务通道，消息在服务通道中被一对一的传递、一对多的分发或者多对一的聚合，驱动各个服务的执行，完成用户定义的业务逻辑。在服务封装与消息处理的基础上，整合 Spring 框架的 Web 模块，将 Spring 处理后的请求参数封装成流程中第一个服务的消息，实现 Flower 框架对 HTTP 请求的支持。最后对系统框架功能的完整性以及系统性能进行了测试。

经 JMeter 压力测试表明，使用 Flower 框架能够极大地提升系统在面对大流量、高并发请求时的处理能力，更适合现代容量大、响应快速的网络环境。同时，Flower 框架应用流式计算的思想，通过额外的流程定义，使完全没有代码耦合的 Service 共同为某项业务服务；极大简化反应式编程难度，使开发者以较低的开发成本，快速得到一个符合现代高并发网络环境的反应式系统。

关键字：反应式编程框架；Akka Actor；流式计算；Flower 框架

Abstract

With the increasing demand of users and the arrival of 5G era, how to make large-scale software systems meet the high concurrent and high load access pressure, fast response, 100% availability of services, flexibility to adapt to a variety of concurrent environments has become a hot issue. Reactive Systems with Message-Driven, Responsive, Elastic and Resilient are emerging and rapidly becoming a new topic for developers to discuss. RxJava and Reactor are pioneers of Reactive Programming Frameworks. They use the observer model to implement their respective reactive systems based on publish-subscribe event response. These two methods are based on Functional Programming, which reduces the readability of the code and increases the cost of learning. Secondly, the coupling degree of the code is high, which is not conducive to the distributed deployment of modern systems. Therefore, this paper builds a new generation of Reactive Programming Framework, named Flower, based on Akka Actor. To achieve a completely Message-Driven Reactive Programming Framework, Flower Framework regards message as a first-class citizen, and encapsulates developer services with Actor model, which naturally conforms to reactive concept.

This paper mainly completes Flower Framework from four aspects: encapsulation of Framework Services to Actor model, implementation of message processing mode, HTTP request support and system testing. Flower Service is the smallest execution unit in Flower Framework, which binds one or more Actor objects. There is no connection between services, but communicating with Message. Process definition connects multiple services to a service channel. Messages are transmitted one-to-one, distributed one-to-many or aggregated many-to-one in the service channel, which drives the execution of each service and completes user-defined business logic. To support HTTP requests, Flower framework integrate the Web module of Spring Framework. The request parameters processed by Spring are encapsulated into the message of the first service in the Flower Service channel. Finally, the integrity and performance of the system framework are tested.

The JMeter stress test report shows that the Flower Framework can greatly improve the system's processing ability in the face of large traffic and high concurrent requests, and

is more suitable for modern network environment with large capacity and fast response. At the same time, Flower Framework applies the idea of flow computing, through additional process definition, makes Service without code coupling serve a certain business together; greatly simplifies the difficulty of reactive programming, and enables developers to quickly get a Reactive System in line with the modern high concurrent network environment at a lower development cost.

Keywords: Reactive Programming Framework; Akka Actor; Flow Computing; Flower Framework

目录

第1章 引言

1.1 课题的来源、意义和目标

近年来，随着互联网应用越来越成熟，大数据和物联网技术越来越普及，对应用程序的需求日益增进，系统开发模式的变化也随之而来。仅在几年前，一个大型应用程序往往需要使用数十台服务器作为负载，请求的响应时间才勉强达到秒级，数据量仅维持在 GB 级；由于庞大的服务器体系，每一次的维护时间以小时计数。而今，用户期望着毫秒级的响应时间、随时可用的服务、以 PB 计量的数据；应用程序被部署到了形态各异的载体上，从移动设备到运行着数以千计的多核心处理器的云端集群。昔日的软件架构已经无法满足当今的需求，具有即时响应性（Responsive）、回弹性（Resilient）、弹性（Elastic）以及消息驱动（Message Driven）四大特质的反应式系统的出现正是顺应了时代发展的潮流。反应式系统将更加适用于当今高并发下的互联网环境，也更符合现代社会的交互模式。

反应式编程（Reactive Programming，RP）是时代发展的思维产物，它是一套异步编程方案。为了适应高并发下的服务器编程，最早由微软公司提出并引入.NET 平台，形成了著名的 Rx.NET 异步开发框架。在反应式系统中，异步操作是实现即时响应的极其重要的方式，使用反应式编程方式构建反应式系统会让系统更加灵活，降低组件之间的耦合，提高系统伸缩性。

非常荣幸能够在校外实习期间进入同程艺龙公司并且参与到李智慧老师发起的反应式框架—Flower（以下简称 Flower 框架）项目研发中。李老师在第一次提出研发一个反应式框架的时候，给我的印象是震撼。从开始学习面向过程的 C 语言编程到后来学习使用面向对象的 Java 开发，似乎从来没有想过开发出的系统与用户的交互是否真的符合实际，每一次请求是否都是必须或者只需要等待一个响应？仔细思考之后，发现每个人其实就是一个反应式的系统，人与人的交流也是反应式的。这也是我第一次接触到反应式的概念，它不仅仅是异步操作，同时也是对现有系统的开发、运作模式的创新。

Flower 框架旨在将反应式编程的门槛降低，把反应式系统的概念融汇于框架之中，完全隐藏反应式编程的细节，使用流式编程的手段，让开发者只需要针对每一个细粒度的业务功能开发相应的 Service 服务，将这些服务进行业务流程的编排，就能

够在几乎零框架学习成本的情况下完成一个反应式系统的开发。Flower 框架在反应式的基础上，还将提供分布式微服务的架构设计，使用流式计算的设计理念，以一种轻量、易于开发、消息驱动、弱依赖、反应式并发的技术特点实现微服务系统。

Flower 框架是对现有编程方式以及系统开发模式的一次创新探索，其流式编程的设计思维领先于业内的其他开发模式，在简化开发者工作的同时提升应用系统的性能。本论文的主要工作围绕 Service 核心封装、消息驱动模式、Web 应用开发支持以及系统测试四个方面展开。

1.2 国内外研究现状

提及反应式编程，RxJava 是一项里程碑式的创造，它的出现极大的推动了反应式系统登上现代舞台[1]。RxJava 来自于 Rx.Net 库，起初由微软公司设计与开发，2012 年开始迁移至 JVM 平台并更名为 RxJava[2]。在反应式较为平淡的时期，国内外纷纷表示了对其的兴趣，在开源社区引起了一波热潮。发展至 2016 年，java.util.concurrent 包[3]、Akka streams 库[4]、CompletableFuture 库以及著名的 Netty 框架[5]等都对其进行了引用，以提供反应式组件。RxJava 基本的消息响应的方式为发布-订阅，是一种扩展的观察者模式[6]。基于 RxJava 提供的四个基本概念：Observable(可观察者，即被观察者)、Observer(观察者)、subscribe(订阅)、Event(事件)，以及 RxJava 的提供线程调度器，开发者便可通过调度器手动管理线程调度，实现一个反应式系统。

RxJava 严格意义上来说并非是反应式编程，而是函数反应式编程（Functional Reactive Programming，FRP）的产物。反应式编程是异步编程的一个子集，由信息的时效性进行推动；而函数反应式编程则是由一个线程去推动（A thread-of-execution）控制流，当然，这些方式都有助于创建一个反应式系统[7]。从 RxJava 的线程调度器就可以很明显的感知到其函数反应式编程思想，由线程来推动控制流，将异步控制交到开发者手中[8]。

随着研究的深入以及影响范围的扩张，在 RxJava 基础之上，规避了众多不直观的底层实现，逐渐形成了 Reactive Streams（反应式流）规范，定义了一种为非阻塞背压提供异步流处理的标准，并且在 JDK8 开始支持[9]。反应式流规范真正意义上定义了反应式编程的规范，基于反应式流规范，在开源社区兴起了第三代反应式库——RxJava2。RxJava2 对反应式流规范进行了实现，并且兼容了 RxJava 的某些特性，深得开发者的青睐，使用 RxJava2 将更容易实现一个反应式系统。

开源社区似乎并不满足于 RxJava2，兼容过 RxJava 的新一代反应式库在底层实现上并不是真正符合反应式流规范，具有一定局限性。Project Reactor 库顺应而生。Project Reactor 被称为第四代反应式库，简称 Reactor，它的设计完全基于 Reactive Streams 规范。Spring Framework 5 中的 WebFlux 组件正是依赖于 Project Reactor 核心进行运作，实现非阻塞调用以及基于网络 HTTP/TCP/UDP 等传输协议的背压处理。除此之外，天然符合反应式设计的 Actor 开发模型也名声大噪[10]。Actor 模型由 Carl Hewitt 博士在 1973 年提出，但受限于当时的处理器水平无法支撑并行系统，未能将这项理论付诸实践；在 2009 年，Karmani 等学者发布的一篇论文中讨论了 Actor 模型在 JVM 平台的实现；直到近期反应式的流行，这项超前的理论才得以重新浮出水面[11]。

近年来，在各个开发者论坛以及开源社区对反应式编程的讨论愈加激烈，对各种支持反应式的技术也都持有各自的见解[12][13]，尝试着将这些技术应用到现代系统的各个层面[14]。在摸索中探寻真理，不难发现反应式系统的兴起成为了一个必然的发展趋势。国内外各大公司的架构师都在为大型网站系统的高并发、大流量、高可用以及稳定具有韧性的分布部署而绞尽脑汁[15]。无论是 RxJava 还是新一代的 Reactor，还是反应式微服务，都是为了适应这些日益壮大的需求，增加系统的防灾能力而顺应产生[16]。仅仅依赖本地的事件驱动将难以满足多样化的云应用部署。2014 年 9 月 16 日发布的《反应式宣言》提出了更健壮、更具回弹性、更灵活，也能更好地满足现代化需求反应式系统架构，对于开发者而言，如何得到一个正确的、反应式的、能够承载新时代需求的系统成为主流问题[17]。

RxJava 以及 Reactor 在反应式编程框架的渐进式发展中都给出了各自的实现，但是仍存在使用上的弊端。首先基于函数式编程手段实现的 RxJava 以及 Reactor 在使用上会产生不直观的代码片断，开发者往往需要进行大量的函数逻辑块的梳理，降低了代码的可阅读性。其次，在函数式编程中，将函数作为一等公民，函数的调用存在天然的内存共享，这将不利于系统之间的解耦，即使通过 RPC 等远程调用的手段来解耦系统，也很难保证系统的即时响应性；RPC 等通过轮询或阻塞方式实现的远程调用方式在一定程度上占用系统资源，加速系统资源的消耗。

RxJava以及Reactor在反应式的内部实现上都对观察者模式进行了拟合，实现了数据流的发布与订阅。通过这种发布与订阅的推送手段确实能够实现反应式编程的消息驱动，但实际上并不符合反应式编程提出的消息驱动理念，其本质还是事件驱动。在

反应式编程中，消息是包含着事件，是事件的载体。在消息驱动的系统中更多的注重消息的接收者，发送者无需在逻辑层面关注接收者信息，而接收者只需要休眠的等待消息，在消息接收成功之后对消息做出响应；事件驱动的系统则更多的关注消息产生源（即发送者），监听事件的产生，根据产生的事件调用相应的事件回调。事件不等于消息，事件的消耗链条是一在系列回调函数中被开发者设定好的，在代码上存在高度耦合，不利于反应式系统弹性的处理事件请求。

1.3 论文结构

论文在前四章对 Flower 框架进行阐述，最后一章是对 Flower 框架的总体概述，展望其未来发展，并提出了几点应用 Flower 框架的开发建议，这将有利于更好的使用 Flower 框架，获得最大化的效益；也有利于 Flower 框架的发展与提升。

第一章：引言部分，介绍了反应式编程的发展趋势以及 Flower 框架的由来、短期的开发目标以及开发的意义；同时对国内外同类框架进行了分析，描述了其实现原理以及发展历程。

第二章：可行性分析及总体方案选择。分析了 Flower 框架在反应式方向上实现方案的选择及其可行性，重点阐述了 Akka 在 Flower 中的封装，以及 Actor 模型对 Flower 设计理念的支撑。

第三章：框架设计与实现。阐述了框架的设计理念，分别详细描述了 Flower 框架主要功能的实现过程及其原理。

第四章：展示了 Flower 框架的测试过程。涵盖了框架功能的单元测试以及系统的性能测试，在这一章也能更为直观的感受到 Flower 框架带来的性能提升，同时也是 Flower 框架的使用示例。

第2章　可行性分析及总体方案选择

2.1 底层反应式组件选择

Flower 框架基于 Akka Actor 构建。反应式编程贯彻整个 Flower 框架，我们期望通过框架的手段得到一个反应式的系统；其次是能够满足流式的，异步非阻塞的消息驱动。Akka Actor 正是这样的一个反应式组件。Actor 模型起源于并发计算的数学模型，是一种无锁的代理对象，执行它们的线程能够保持持续活跃，不会被阻塞。

在 Actor 模型中，系统由 Actor 对象以及消息组成，每个 Actor 具有一个唯一的，不可变的名称[18]。Actor 对象通过发送异步的消息进行通信，这就让 Actor 对象执行的操作能够高度并发，使加入 Actor 的系统对并行问题的解决能够得心应手。图 2.1 展示了 Actor 系统中 Actor 对象就的通信模式。

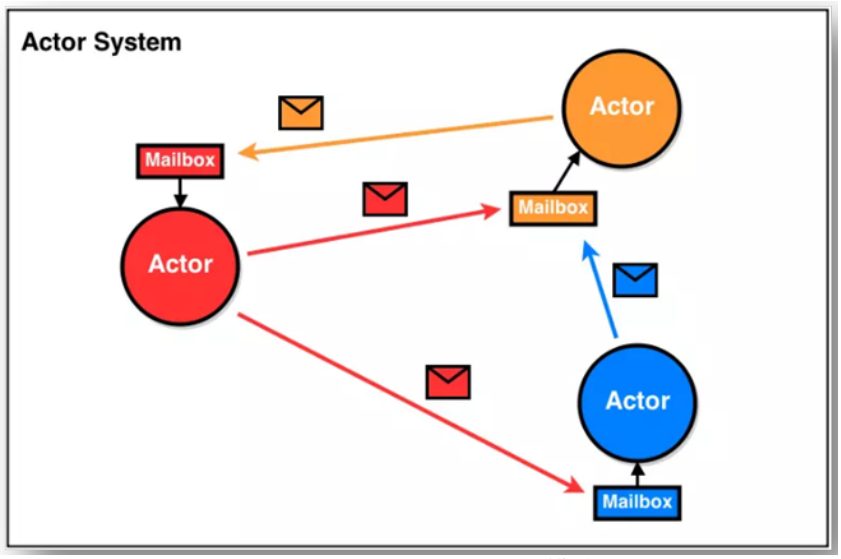

图2.1　Actor System 通信模式

每一个 Actor 对象都有其独自的消息缓存单元，称之为邮箱（Mailbox）,通过邮箱机制，实现异步的消息通信。这个模式与 Flower 框架的设计理念不谋而合。我们一样期望 Flower 提供的 Service 不需要和其他 Service 共享状态，唯一的交互手段就是消息，通过消息来驱动业务流转。

Actor 对象无需锁策略。Actor 对象在单个原子步骤中只会对一条消息产生响应，在需要的时候对消息进行回应。由于消息的发送和接收都是异步执行的，Actor 不会拥有系统资源。而在内存共享模型中，支持锁策略的轮询和阻塞机制将持续占用 CPU 资源。Actor 能够将多核 CPU 从各种锁策略中解放出来，更集中精力于增加吞吐量，

让所有的反应式线程都不会被阻塞，白白消耗系统资源。这也是选用 Actor 的原因之一，Web 应用程序天然的在一个并发环境下，编写一个 Web 服务同样需要考虑锁的问题，采用 Actor，即保证了并发量又抛开了同步系统资源的烦恼。

Actor 具有并行性。并行与并发是不同的概念，并发指多个操作同时出现，并行处理则是以并发的方式完成单独的一个目标。Actor 的并行性可以将复杂问题拆分，分配给多个 Actor 加快处理速度[19]。Flower 框架也需要这样的并行处理能力，以加速系统中的耗时的操作。Actor 是轻量级的对象，默认 Actor 对象大小仅仅 16KB,创建多个 Actor 对象来改善响应时间，不仅不会消耗过多的系统资源，反而是加快了对有限资源的释放。然而如果是开启一个新线程来完成这项操作，系统资源很快就会被耗尽。因此将 Service 封装成 Actor 是可行，也是有效、受推荐的方式。

Akka Actor 还对 Actor 模型进行了拓展，阐述了其位置透明性。Actor 的命名不受实际位置的影响，也就是说，只需要得到一个 Actor 的抽象引用，就能给这个 Actor 发送消息，并且不受 Actor 系统的限制。相互通信的 Actor 可能位于同一个核心、同一个网络中的不同节点。而一个分布式系统包括了若干台通过网络互联的计算机[20]；Flower 框架将这些计算机拟合成多个互联的 Actor 系统，通过 Actor 的位置透明性支撑分布式部署。框架内部对 Service 进行 Actor 封装，享受 Actor 对象不受地域限制的访问，这将简化分布式的部署方案。位置透明性让 Actor 对象具有了可移动性，开发者无需为系统定义静态的调用逻辑，只需要关注运行时服务的状态变化，实现运行实例与引用的解耦。

Akka Actor 提供了 Actor 对象的监察模式。父 Actor 对象具有创建子 Actor 的权利，这样父 Actor 就成为了监察者，监管着所有被它创建的 Actor 对象，管理子 Actor 的运行，重启或停止。多层级的 Actor 监管者形成任务处理层级，在 Flower 中，父级 Actor 管理着子 Service 的生命周期、行为方式。

2.2 Flower 框架性能提升方案

首先需要提及的是 Actor 系统本身的性能特性。为了让特定的 Actor 对象能够回应由并发方式收到的消息，系统的每个 CPU 核心都被充分利用起来。Akka 调度器会将线程池中的线程分配给邮箱中有消息的 Actor 对象，在繁忙的系统中，大量的消息不断的进入 Actor 对象的邮箱，Akka 调度器持续为线程分配任务；Actor 系统正是通

过这样无阻塞的方式来获取理想、高效的并发系统。通过此模式，增加每台服务器的资源利用效率，减少集群中服务器的数量，节约成本。

此外，Flower 在整体的代码运行逻辑上也有不一样的设计。

传统的 Web 应用程序线程模型如图 2.2 所示。

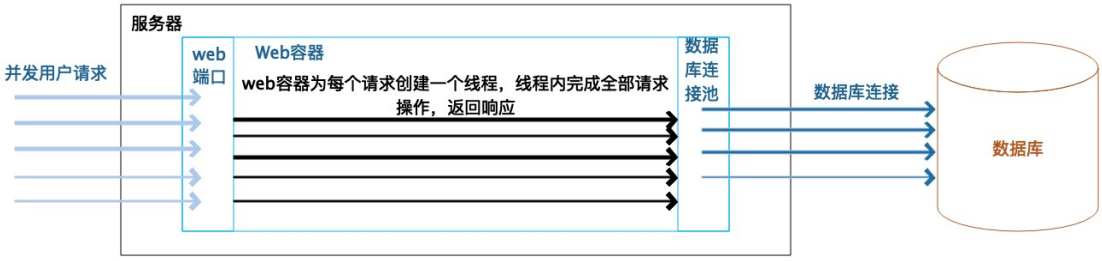

图2.2 传统的 Web 应用程序线程模型

当并发用户访问 Web 容器时，例如 Tomcat，服务器为每一个用户请求创建一个线程，线程内完成全部业务操作之后返回响应。占用线程时间长，如果需要访问数据库，则还需分配数据库连接。当容器默认的 200 个线程分配完毕，第 201 个到来的请求就需要被阻塞等待，当更多的用户段时间内访问系统时，线程资源被耗尽，系统只能重启。

Flower 框架提供的 Web 容器在很大程度上提升了资源耗尽的并发上限，其线程模型如图 2.3 所示。

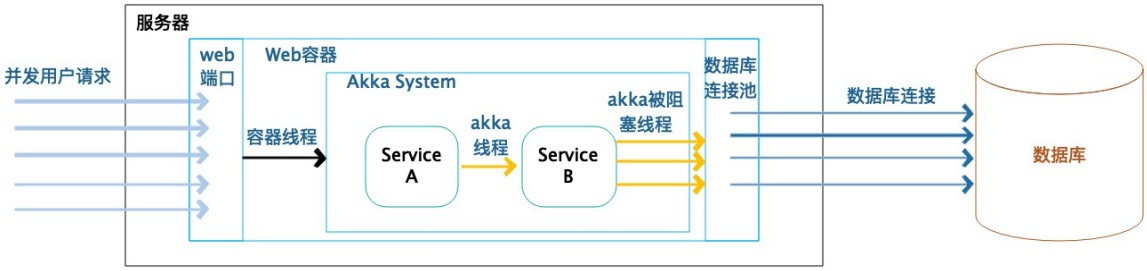

图2.3 Flower 框架 Web 应用程序线程模型

如图 2.3 所示，容器线程是无阻塞的线程，并发用户请求不会被阻塞等待。由于 Actor 邮箱的存在，Service 之间的 Akka 线程也是异步非阻塞线程，请求被快速异步传递。并发的用户在一定范围内可以保持在较高的水平。

即便 Flower 已经将程序内部的线程模型进行了改进，在进行数据库操作时，依旧存在被阻塞的可能。数据库连接的获取是一项耗时较长的操作，普遍的应用程序为了加快数据库连接的获取，都会使用数据库连接池来维持一定数量的数据库连接，并

且复用连接；通过这种方式来加快程序获取数据库连接的过程。然而数据库连接池能够维持的连接数量有限，当大量用户请求数据库时，即使是数据库连接轻微的阻塞，也可能将数据库连接池迅速耗尽，从而引起请求的阻塞。为了解决此问题，Flower 框架已经有了大概思路，构建纯异步非阻塞的线程模型，如图 2.4 所示。

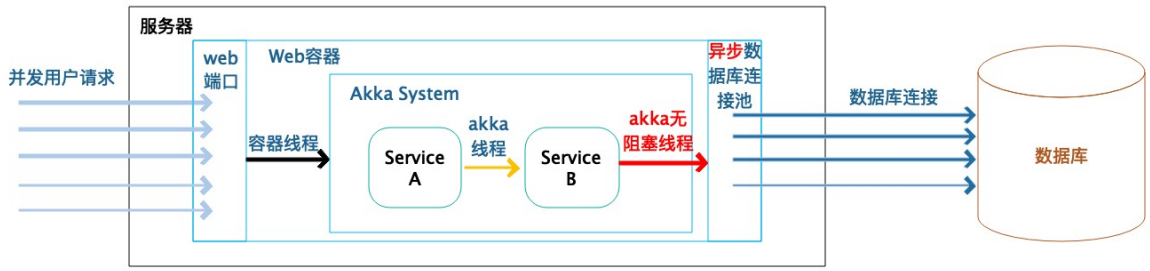

图2.4　Flower 框架 Web 应用程序纯异步无阻塞线程模型

将数据库连接池替换成异步的数据库连接，即使某一类数据库连接被阻塞，也不影响 Web 容器的响应性。Web 容器无需关心数据库连接池是否被耗尽，只需要发起数据库请求即可；异步数据库连接池缓存该请求，并且在数据库连接可用时执行。在异步数据库访问模式中，面对大量用户的请求也能够从容应对，引起请求阻塞的原因不再是数据库连接的数量上限，而是缓存的上限；相比于增加数据库连接，提高应用程序的可使用内存则更简单、有效。

第3章 框架设计与实现

3.1 框架功能模块设计

Flower 框架提供 common 核心模块，Web 模块，分布式注册中心三个主要功能模块，以支持流式计算的反应式系统开发。框架功能模块组成结构图如图 3.1 所示：

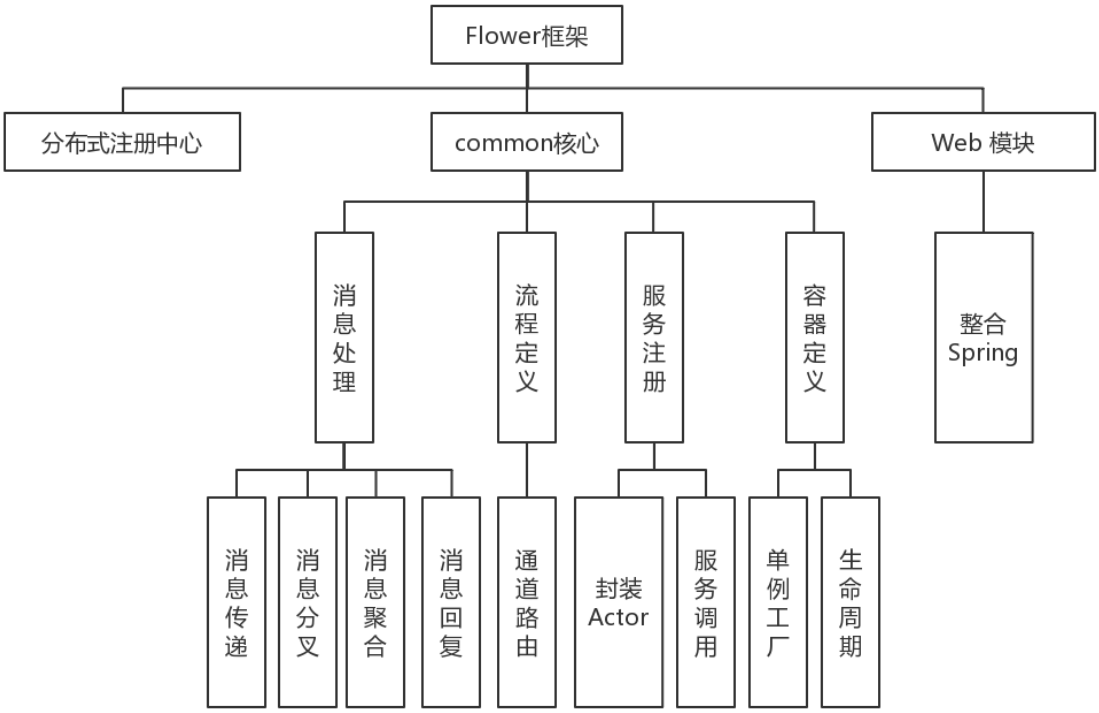

图3.1 Flower 框架功能模块图

3.1.1 common 核心模块

common 核心模块是 Flower 框架的基础设施，其他功能模块的开发均依赖于它，也是构建反应式系统的核心。

(1) 消息处理

消息是 Flower 框架中的一等公民、服务之间的唯一耦合、服务的输入和输出，消息经过一个或多个服务的处理以完成相应的系统功能。消息处理的灵活程度在一定程度上决定了 Flower 框架整体的响应性以及其使用的业务场景。在 Flower 框架中，消息的处理模式有：简单消息传递、消息分叉、消息聚合、消息回复。

简单消息传递，即一对一的将先序服务的返回消息传递到后继服务，是 Flower 框架中最为基础的消息处理模式。

消息分叉，即将先序服务的返回消息一对多的分发传递到多个后继服务。收到消息的后继服务并行执行。Flower 框架提供可限制分发条件的消息分叉模式。在业务的处理过程中，多个后继服务并非需要完全执行，利用条件分发的消息分叉模式，可并行执行满足条件的后继服务。

消息聚合，即将多个先序服务的返回消息聚合到一个后继服务，是多对一的消息传递。消息聚合可用于收集多个并行服务产生的消息。

消息回复，即阻塞的等待消息处理结果，消息回复模式可以让业务流程的调用者获得处理结果。与以上三种消息处理模式不同的是，消息回复模式是同步调用，让 Flower 框架能够适应传统的命令式编程方式，但是对于一个反应式系统来说，应尽量减少消息回复模式的使用。

(2) 流程定义。

Flower 框架通过流程定义将多个服务串联，从而构成的一条完整的业务处理逻辑。被定义的流程由 Flower 框架加载，构建成流程通道实例，一个通道实例显示的为一类消息提供业务处理能力。

Flower 框架可为一个流程定义创建多个通道实例，通道路由器为相同定义的流程通道实例提供消息路由，多个相同的通道实例在通道路由器的帮助下形成负载，以加强系统的并发处理能力，提升系统的弹性。

(3) 服务（Service）注册

服务是 Flower 框架中处理业务逻辑的最小单位，服务的粒度可大可小，取决于开发者自身的需求，但是粒度太大的服务不被 Flower 框架推荐，失去了反应式的优势。服务的粒度应尽可能的满足单一职责原则，将整个业务流程切分成各自独立的处理单元。在设计理念上，服务置身于流程之中，具有一定的先后顺序，分别称为先序服务、后继服务。服务是可以被单独执行的，可以被复用于不同的流程；Flower 框架同时会提供部分框架内置服务，用于完成框架内部的一些操作，通常用来实现消息模式。

Flower 框架基于 Akka Actor 构建，每一个 Service 都会被 Flower 框架隐式的加载成 Actor 模型，而开发者无需感知内部细节。服务的调用是一次异步、非阻塞的操作，这也符合反应式理念。结合 Flower 框架的异步处理优势，极快的响应每一个服务处理单元，加快整个业务流程的处理效率。

(4) 容器定义

Flower 框架提供容器实例，每一个应用系统实例都建立在容器之上，系统的业务部署也在容器中启动。单例容器不仅仅给分布式部署提供便利，同时也简化应用系统的创建过程。

Flower 框架提供生命周期管理，容器的初始化、加载、重启，单例工厂的创建与结束都与生命周期息息相关。

3.1.2 Web 模块

Web 模块依赖于 common 模块，是 Flower 框架为开发 web 应用程序提供的帮助模块。Flower 框架本身并不提供 HTTP 请求的解析能力，而 Spring 的 HTTP 解析模式一直都是被称赞优秀的范例，Flower 框架通过 Web 模块来集成这一优势，简化工作量的同时，让框架有了插入现有 Spring 系统，改造局部反应式的能力。集成当今主流的 Spring 开发框架，也扩展了 Flower 的使用场景。

3.1.3 分布式注册中心

分布式注册中心是使用 Flower 框架实现的应用程序，用于 Flower 容器实例注册服务信息以及流程信息，提供容器分布式部署的能力。Flower 框架的分布式部署是轻量级、简单易操作的，注册中心直接打包成一个可执行的应用程序，开发者只需要启动注册中心，开发自己的业务容器，编排服务流程即可。

3.2 框架整体架构设计

3.2.1 底层核心类设计

Flower 框架底层的核心类设计理念如图 3.2 所示。核心的设计是最初的设计理念，是 Flower 框架最基础的能力，也是 Flower 框架未使用容器封装时的外貌。

Flower 框架对开发者暴露 Service<P,R>接口，用于执行开发者提供的业务逻辑代码。具体的开发者服务逻辑类需要实现 Service<P,R>接口，这是开发者服务被 Flower 框架加载的必要条件之一；泛型 P 标识该 Service 接收的消息类型，泛型 R 则标识 Service 返回的消息类型。在这里消息类型 P、R 符合里氏替换原则(Liskov Substitution Principle),此特性也拓展了 Service 的消息接受范围，开发者在实现 Service 接口进行业

务代码设计时，只需要考虑为某一类消息提供单一指责的 Service，增加 Service 复用的可能性。

FlowerService 接口是 Service<P,R>的上层接口，用于 Flower 框架标识 Service 类型。Flower 框架通过面向接口的编程方式，依赖与抽象的 FlowerService 接口，而不依赖于具体的用户实现代码，将用户实现的 Service 隐式的加载为能被框架接受的 FlowerService，在概念上统一所有的 Service 实现。

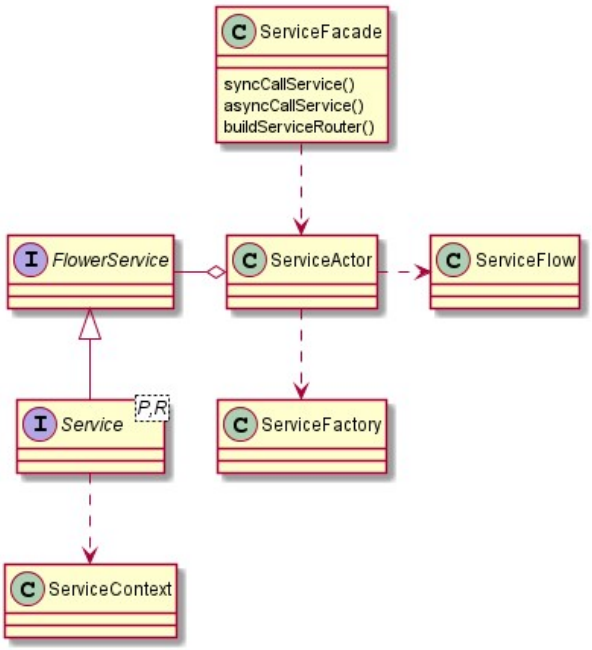

图3.2 核心类设计图

ServiceContext 是框架内置的服务上下文，用于保存当前所属处理流程的信息以及当前处理的服务节点信息；如果是 web 请求，ServiceContext 将保存客户端的端点位置，以便随时向客户端传输有效的返回信息，在这种情况下，请求端点的响应模式不再是一次请求对应一次响应，一次请求将可能对应多次响应。

ServiceFacade 是 Flower 框架服务的门面，提供两种服务调用方式，一种调用方式是异步非阻塞（asyncCallService）的调用，另一种调用方式则是适用于消息回复模式的阻塞调用（syncCallService）。异步非阻塞调用是主要的调用方式，也是反应式即时响应的基础建设。ServiceFacade 还提供服务的路由（buildServiceRouter），路由可以增加服务的负载，可以为同一个服务创建多个处理实例，使单个服务可并发的处理消息。消息经路由收取，经平衡策略决策后，分发至相对空闲的服务实例处理，有效的缓解服务的响应压力。

ServiceFactory 用于 Flower 框架管理用户创建的 Service，例如从*.service 文件中加载并注册服务实例、从注册中心加载服务实例。

ServiceFlow 用于框架管理用户的流程定义，例如从*.flow 文件中加载流程定义，构建流程通道。

ServiceActor 将 FlowerService 封装到反应式组件 Actor 对象，利用 Actor 模型执行用户的业务代码。每一次消息的收取将驱动服务执行一次用户代码，产生一个返回消息。ServiceActor 通过读取 ServiceFlow 加载的流程定义，寻找后继服务，将 FlowerService 产生的消息按照流程定义的逻辑流转到下一个或多个 FlowerService。

3.2.2 服务初始化及调用

ServiceFacade 根据传入的 Flow 名字以及 Service 名字创建流程通道中的第一个 ServiceActor，ServiceActor 通过 ServiceFactory 装载开发者实现的 Service 实例，并且根据 ServiceFlow 获取流程定义中的后继服务，递归的创建所有被注册的服务实例，形成流程通道。服务初始化时序以及调用时序描述如图 3.3 所示。

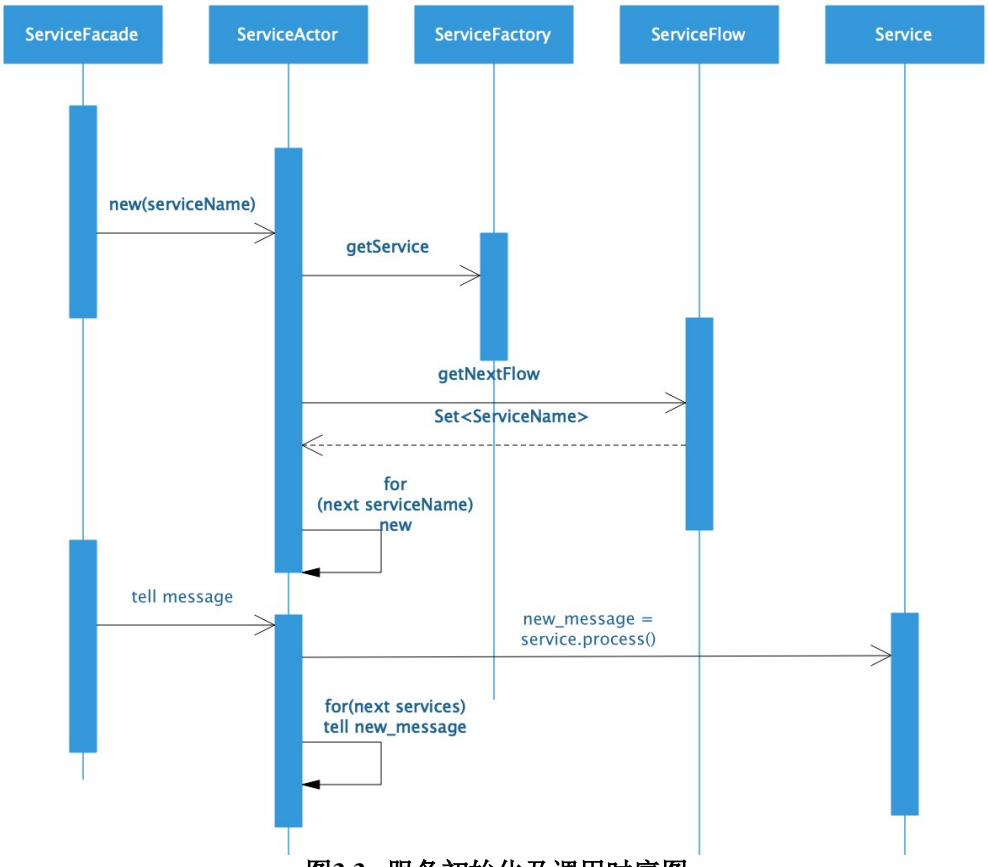

图3.3 服务初始化及调用时序图

开发者通过 ServiceFacade 调用流程，消息被传入流程中的第一个 ServiceActor。ServiceActor 调用用户实例完成逻辑操作，并封装返回值成消息，交给后继的 ServiceActor 继续处理，直到整个流程结束，以完成简单的消息传递模式。

3.2.3 Flower 容器设计

定义 Flower 容器的意义在于形成一个单例的应用实例，增加服务实例缓存，包装其生命周期，封装框架底层的调用逻辑，将与开发者的依赖抽象成接口、注解等形式，简化开发者的开发逻辑，降低框架的学习成本。容器顶层设计如图 3.4 所示。

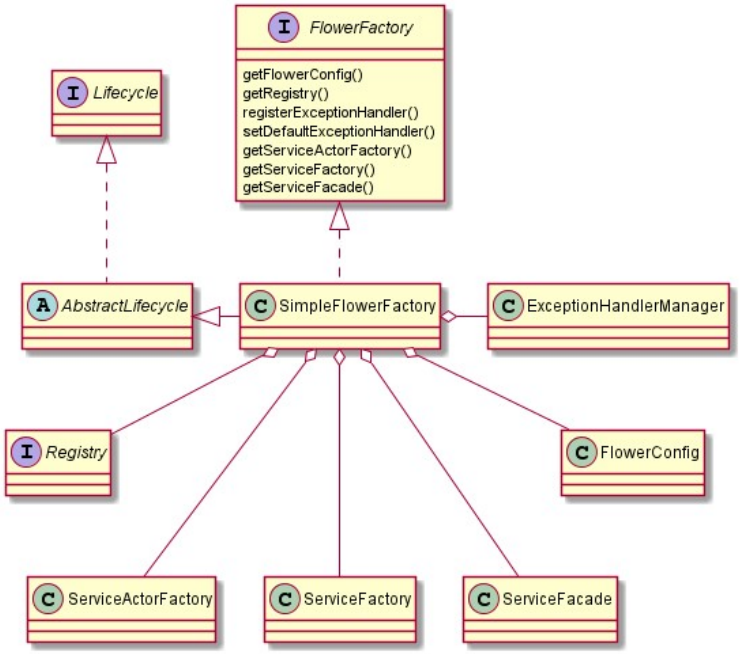

图3.4 Flower 容器顶层设计

接口 FlowerFactory 是框架对外暴露的 Flower 容器接口，类 SimpleFlowerFactory 是 Flower 框架对容器的默认实现，开发者可直接创建该默认实例获取框架的默认容器，默认容器实现了生命周期（Lifecycles）的管理方法，图 3.5 展示了默认容器在 Flower 生命周期中的初始化时序。

默认容器包含异常管理器（ExceptionHandlerManager）、容器配置（FlowerConfig）、服务调用门面（ServiceFacade）、服务工厂（ServiceFactory）、Actor 工厂（ServiceActorFactory）以及注册中心（Registry）。异常管理器用于管理用户代码抛出的异常；容器配置用于加载用户自定义的容器实例配置；服务调用门面则提供给开发者直接调用服务的方法；服务工厂是框架流程内的服务创建、加载、缓存的工厂实

例；Actor 工厂是 Actor 系统的实例，用于创建、管理 Actor 对象，将 Actor 对象绑定到框架内部服务；注册中心是服务实例在框架内部的注册管理器，同时也能够通过配置来加载远程的注册中心，将服务进行分布式部署。

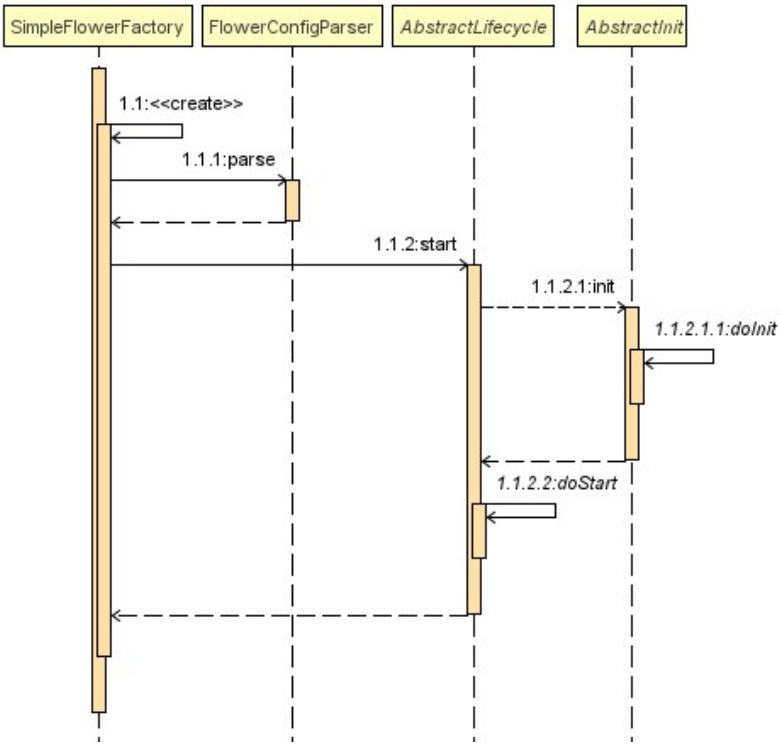

图3.5 Flower 容器初始化时序图

FlowerConfigParser 从自定义的 flower.yml 文件中加载容器配置（FlowerConfig），简化了配置的复杂性。yml 文件是一种易使用，易理解的配置文件，可配置多种数据结构；对开发者而言，也是一种容易接受的方式。

AbstractLifecycle 是生命周期接口的抽象实现，同时继承了抽象的初始化定义（AbstractInit），共同管理容器的初始化、启动与关闭。在设计上，依赖抽象接口，不与用户代码形成直接依赖。

3.2.4 Web 模块设计

Web 模块对 Spring 进行整合，将 Flower 容器实例的加载成为 Spring Bean，因此可以在任何一个使用 Spring 框架的系统中集成 Flower；

针对 HTTP 请求，Flower 框架本身并没有设计相关直接解析 HTTP 请求的方案，而是利用了 Spring，获取 Spring 解析后的 HTTP 请求参数封装成为流程通道中第一个服务的接收消息，后续的消息流转则由 Flower 完成。设计思路如图 3.6 所示。

InitializingBean 接口是由 Spring 框架提供，可自定义初始化 Bean。SpringFlowerFactory 是整合 Spring 的 Flower 容器实现，Spring 容器启动之后，自动加载 Flower 容器实例；同理，抽象类 FlowerController 封装了流程定义，Flower 容器将在 Spring 容器启动之后自动加载流程通道。

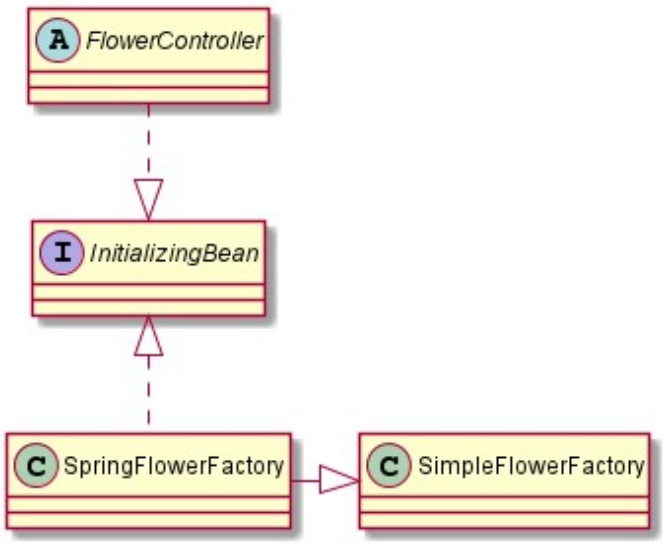

图3.6 整合 Spring 实例设计思路

实质是将 Flower 容器、FlowerService 加载成为 Spring 中的 Bean 对象，Flower 的流程定义加载为 Spring 的 Controller，形成流程路由，通过 Controller 获取 HTTP 请求的参数，传递到 Flower 容器完成后续处理。

3.2.5 分布式流式微服务设计

正如之前章节所提及到的，Flower 框架不仅仅是一个反应式编程框架，同时也是一个支持分布式部署的流式微服务框架。Flower 的分布式微服务架构与传统的，通过远程调用（RPC）等方式来实现系统解耦以及分布式部署架构体系不一样，利用 Akka Actor 模型的位置透明性，我们将本地运行的服务注册到远程的注册中心，用户可以无感知的混合编排本地服务和远程服务，定义流程通道。服务之间依旧不存在直接的代码耦合，通过消息驱动，Flower 框架的特性依旧适用于分布式部署的系统。

在 Web 开发的领域，Web 请求就是天然的消息，而消息就是领域中的模型，只要设计好领域模型，无需再去让领域模型驱动详细的开发设计过程，而是让模型流动起来就完成了系统的构建，模型被流动到不同的服务，在不同的服务中被计算、填充、完善，最后整个流程完成就得到了需要的结果，是真正的面向模型设计。

Flower 分布式部署模型如图 3.7 所示。

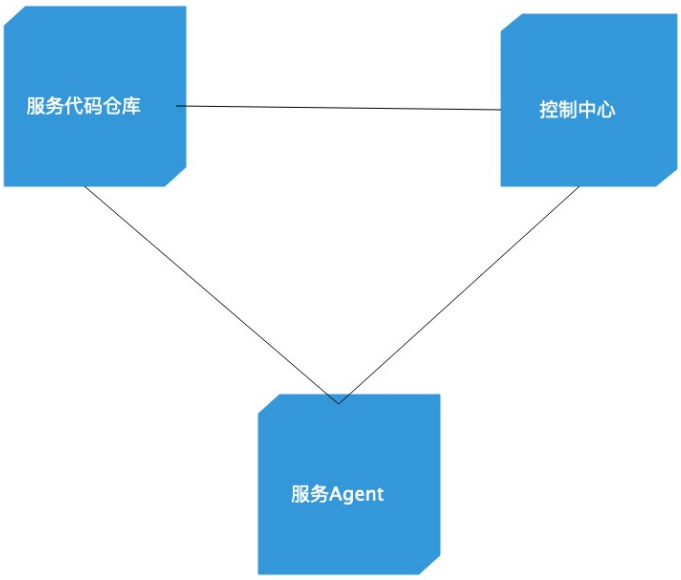

图3.7 分布式部署模型

Flower 框架统一控制整个应用集群，即多个 Flower 实例，控制中心监管集群的整体资源，消息的流转与通信如单个实例一般，从控制中心加载服务实例，然后由 ServiceActor 完成远程或本地消息的传递。

3.3 框架实现

Flower 框架的实现在基础设计之上，应用多种设计原则与模式，使框架具有更强的内聚性以及封装性，同时也是为了给开发者更友好的开发体验。其次，在易用性方面，使用配置文件、注解等形式，简化开发者的使用方式，减少对开发者代码的入侵，降低学习成本。

3.3.1 Service 到 Actor 模型的封装

Flower 框架通过继承 Akka Actor 反应式组件提供的 AbstractActor 完成将 Service 封装到 Actor 模型的工作。开发者通过 Flower 框架提供的 Service<P,R>抽象接口实现业务代码，通过面向接口编程的方式，封装成 FlowerService，倒置与用户代码的依赖。实现类图如图 3.8 所示。

AbstractFlowerActor 在 Flower 框架中进一步抽象了 Akka Actor 提供的 AbstractActor 抽象类，添加框架绑定服务的必要流程，包含正常处理流程和异常处理

流程。在一次正常处理流程中，Service 处理业务返回消息即可，如图 3.9 消息 1.1.1 所示；发生异常时，框架则启动默认异常处理方案，如图 3.9 中消息 1.1.2 所示，默认的异常处理器（DefaultExceptionHandler）将捕获异常并形成异常调用堆栈，为开发者提供调试帮助。框架提供 ExceptionHandler 接口，供开发者自定义异常处理器。

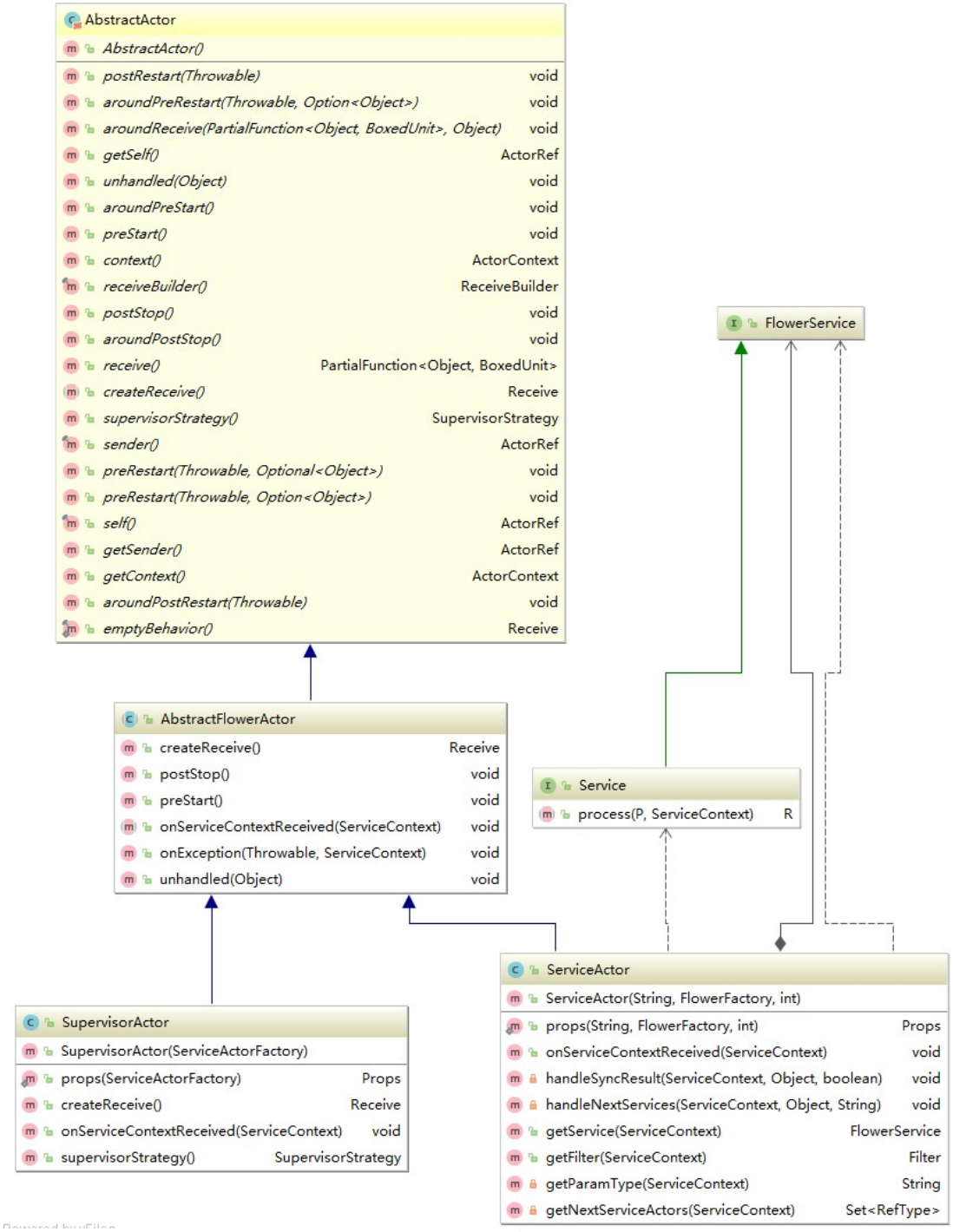

图3.8 封装 Actor 模型实现类图

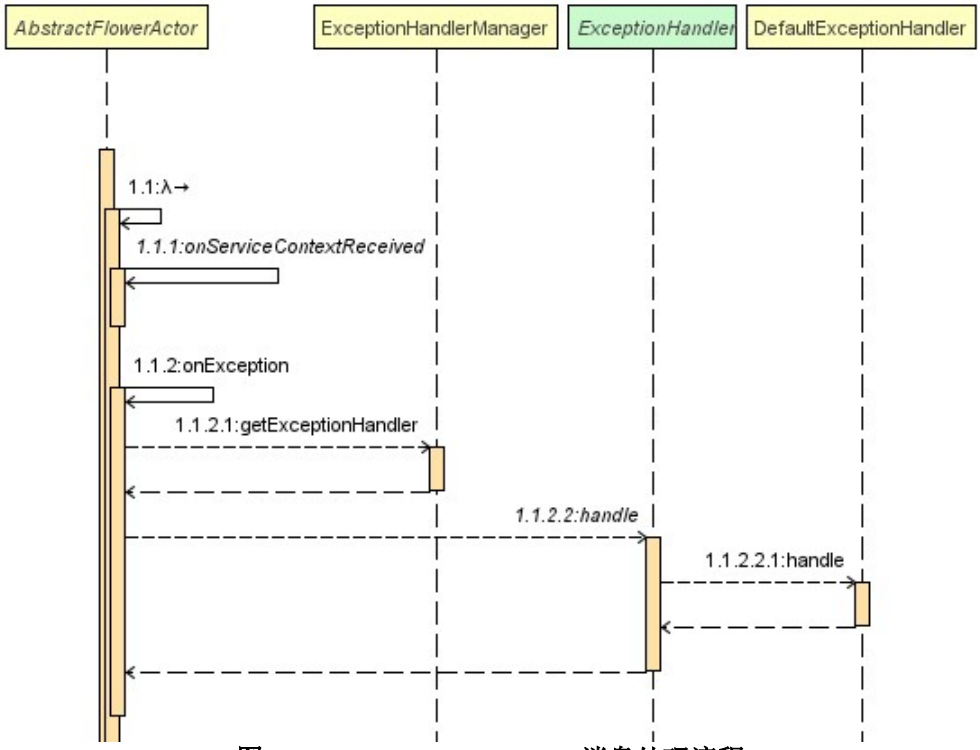

图3.9 AbstractFlowerActor 消息处理流程

SupervisorActor 是 Flower 框架为内部 Actor 对象创建的父级监察对象，实现了 Actor 模型中的监察模式。SupervisorActor 继承自 AbstractFlowerActor，与普通 Actor 对象不同的是，SupervisorActor 能够创建子 Actor 对象，并管理子 Actor 对象的生命周期。在子 Actor 对象发生错误时，接收子 Actor 对象的错误反馈消息，根据异常消息决策子 Actor 的重启与停止。

ServiceActor 是 Flower 框架将 Service 绑定到 Akka Actor 对象的具体实现。主要功能为接收消息，处理 Service<P,R>业务代码逻辑，根据流程定义进行消息分发。下级服务的消息分发通过方法 handleNextServices()完成，处理时序如图 3.10 所示。

请求下级 Service 信息时，采用懒加载的方式，从单例工厂 ServiceFactory 获取服务实例。对于聚合节点，需要为每个流程通道创建不同的内置聚合服务，因此将创建新的聚合服务服务实例。服务创建过程如图 3.10 中消息 1.1.3 所示，这也是 Flower 框架加载 Service 的时序流程，将先从本地加载服务，如果本地加载不到服务，从注册中心拉取相关的服务信息进行实例初始化。

服务实例的加载方式也为 Flower 框架流式微服务的分布式部署提供了可靠支撑。单个 Service 在 Flower 框架中具有位置透明性，且能够被复用编排，远程调用便捷。

在分布式部署过程中，只需要使用远程服务名进行混合编排即可实现分布式部署调用，不存在服务间的代码耦合。

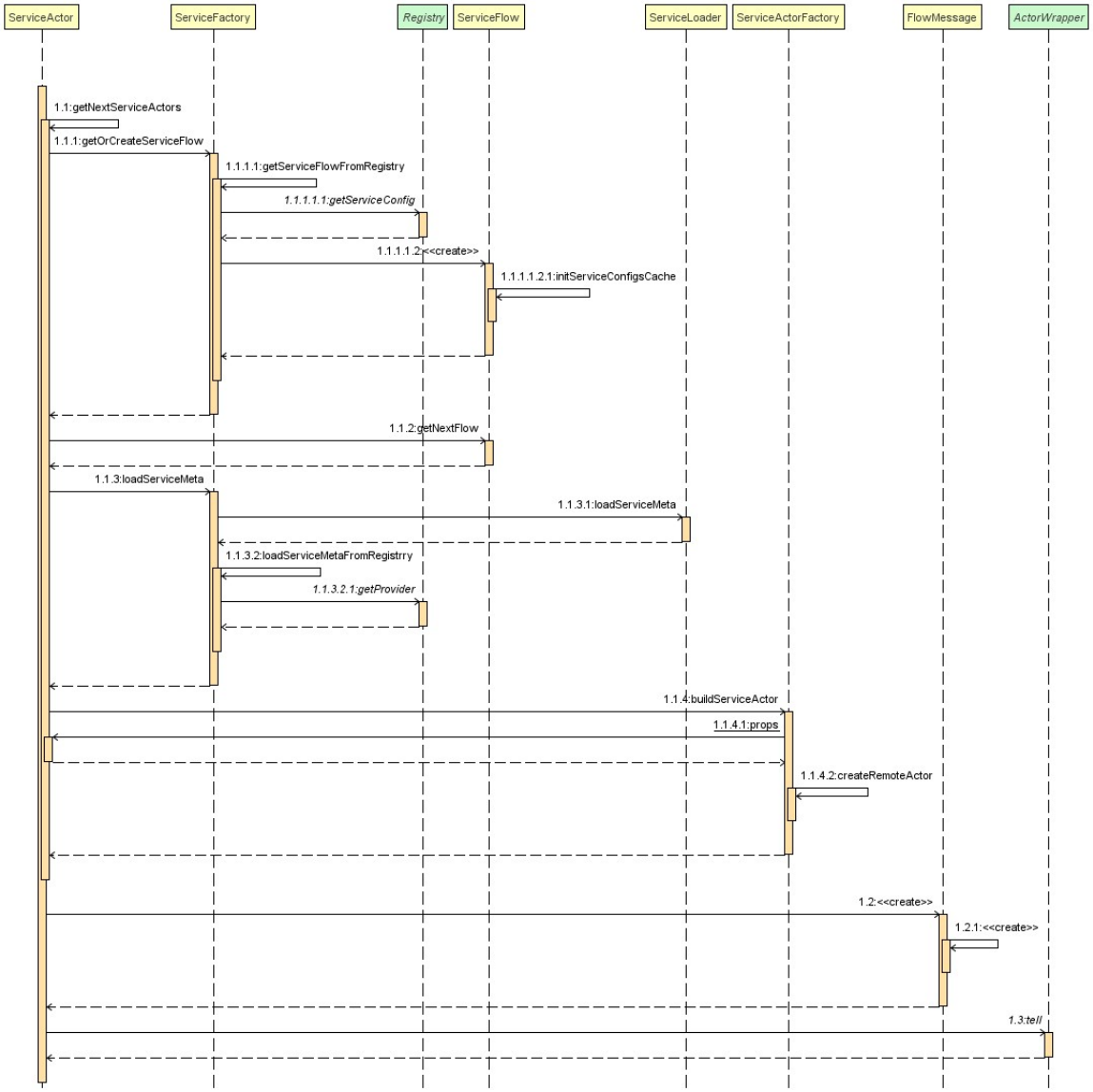

图3.10　Flower 框架下级消息处理时序

3.3.2 消息流转

消息流转实现类图如图 3.11 所示，消息流转有两种方式，一是流转到本地 Service，二是流转到远程 Service。对于消息，Flower 框架提供一个抽象 Message 接口，开发者自定义的消息需实现此接口，才能加入消息流转，被 Service 所接受。

ActorWrapper 是框架抽象的服务调用接口。getServiceName()用于获取绑定当前 Actor 对象的服务名字，通过唯一的服务名进行消息传递。消息的传递方式具有两种：

一种含有发送者，主要用于需要同步获取返回值的服务调用，即消息返回模式；另一种没有发送者，服务无需关心发送者，只需要按照设定处理消息即可，是 Flower 框架中最为普遍的消息流转模式。

ActorRefWrapper 是本地服务的调用的封装实现，用于本地的消息流转调用；当调用的服务属于远程服务时，则需要使用 ActorSelectionWrapper 进行远程服务的调用。服务的通信即 Actor 对象的消息传递，Flower 框架将这一部分封装在反应式组件 Akka Actor 中执行。

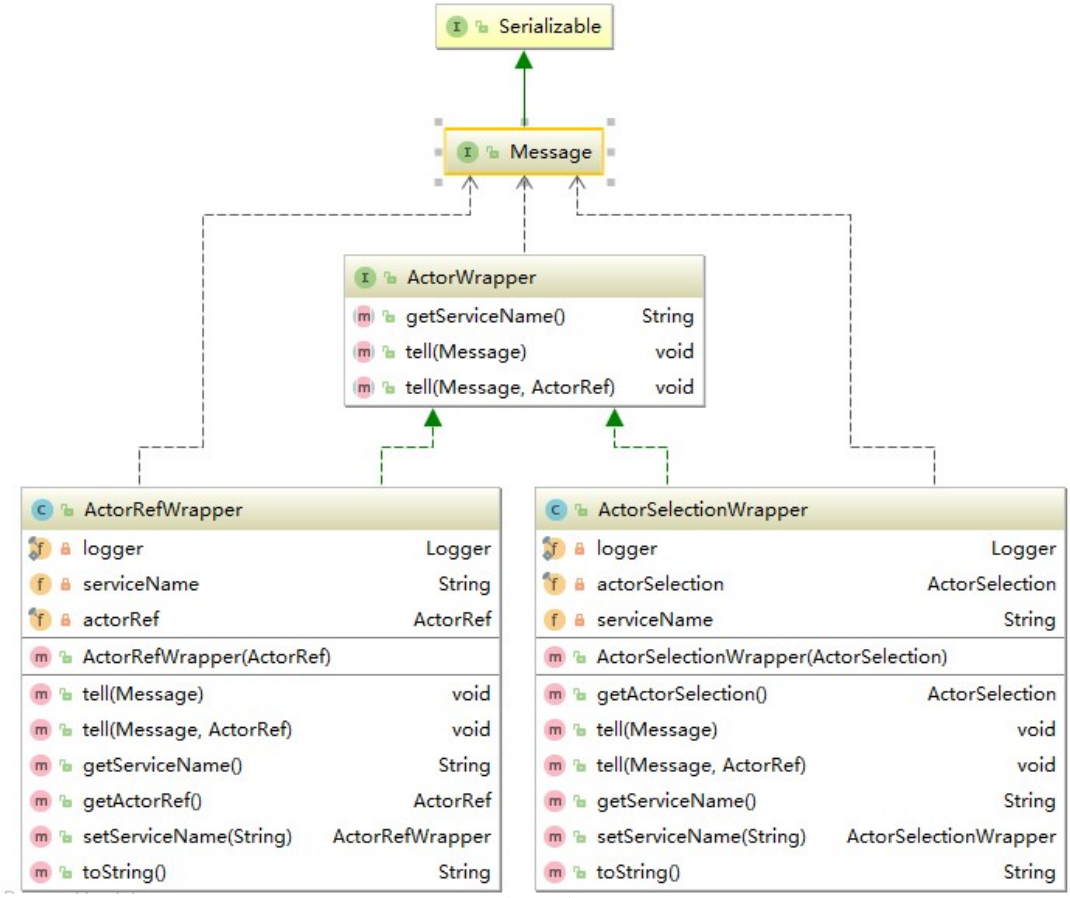

图3.11 消息流转实现类图

3.3.3 Flower 容器注解

为方便开发者向容器注册服务与流程，Flower 框架提供流程注解以及服务注解。注解是 Flower 框架加载用户代码的方式，简单的注解不仅仅能够提高开发者对 Flower 框架的理解，同时也是简化开发的一项重要工作。注解的定义如图 3.12 所示。

被 FlowService 注解的 Service<P,R>子类都会在容器初始化时，自动被容器加载并注入框架内部。通过 FlowerServiceUtil. getServiceName()方法获得开发者注解的服务名，服务名是 Flower 框架中能够唯一向该服务发送消息的抽象引用。

FlowerType.AGGREGATE 枚举类型用于标记接收聚合消息的后继服务。聚合消息通过 Flower 框架内置的 AggregateService 来实现。通过使用缓存的方式，记录每一次到达的消息，被聚合的分支消息全部缓存成功之后，自动封装成 List 列表并流转到 FlowerType 为 AGGREGATE 的后继服务。

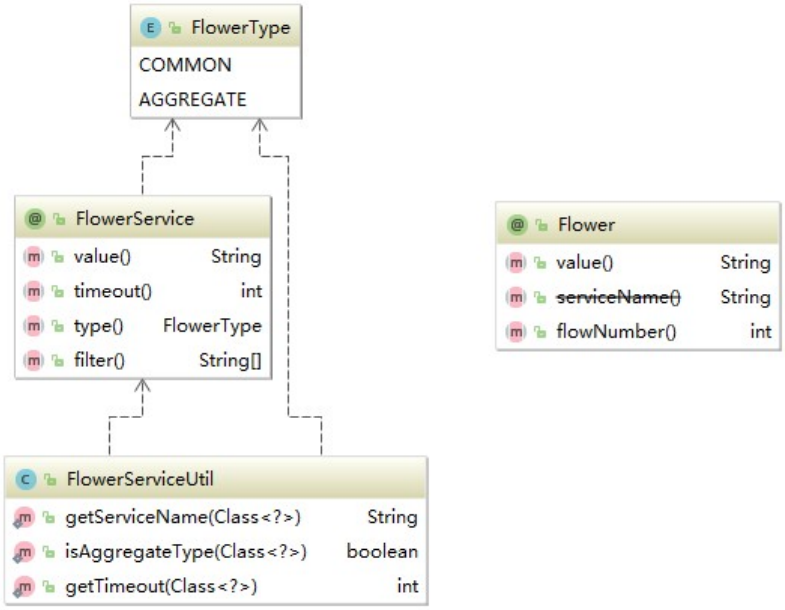

图3.12 Flower 容器注解

Flower 用来注解流程定义，具有 value 以及 flowNumber 两个注解参数，分别作为流程的命名以及初始构建的通道数量。流程命名和服务命名，组成一个唯一的流程服务调用路径；通道数量为相同流程的请求提供负载。在 Web 开发中，通常给 FlowerController 网关添加 Flower 注解，一个 FlowerController 即一个流程定义，也就是一个通道实现，通过 Flower 注解来定义流程名字以及通道数量。

3.3.4 Web 开发实现

(1) HTTP 请求处理

Flower 框架利用 Servlet3 的异步操作，代理 Spring 解析后的 HTTP 请求，封装成流程通道的第一个服务的消息。Flower 框架中一次 HTTP 请求的处理过程如图 3.13 所示。

FlowerController 以及 Flower 容器由 Flower 框架提供。其中 FlowerController 包含 Flower 框架中流程通道的入口，也是 HTTP 请求进入 Flower 容器执行异步操作的前提；FlowerController 同时也是 Spring 中的控制器，用于接收请求参数以及请求路径的路由。框架提供的 FlowerController 抽象实现类如图 3.14 所示。

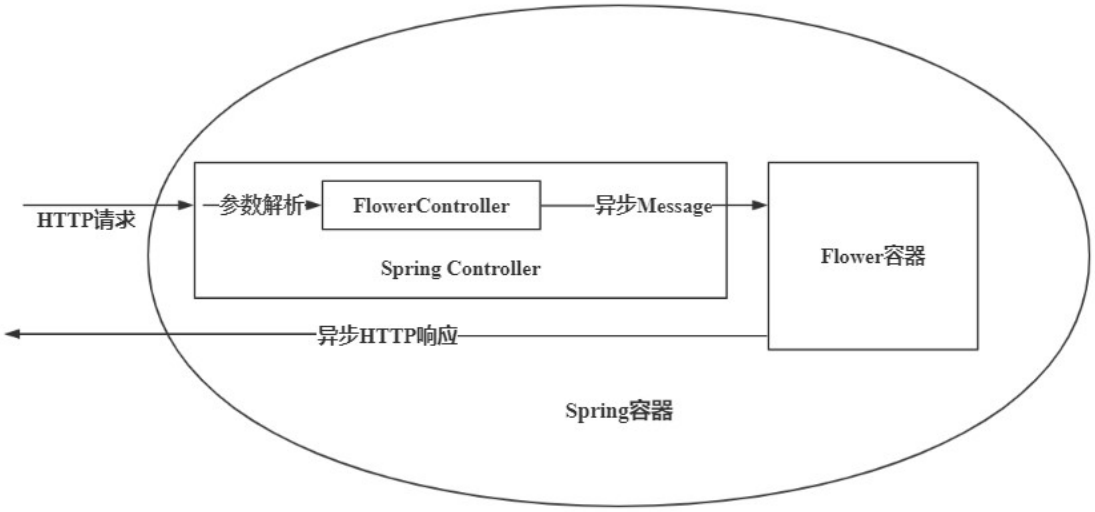

图3.13 Flower 框架中一次 HTTP 请求的处理过程

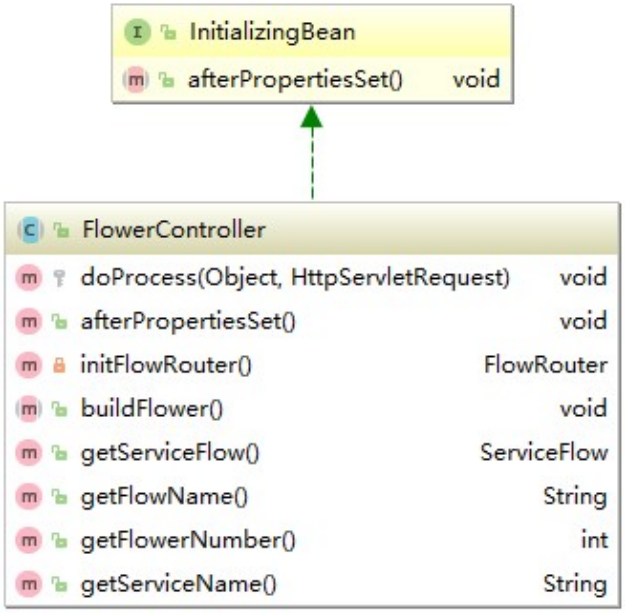

图3.14 FlowerController 抽象实现

FlowerController 中的 doProcess()方法接收 Spring 框架解析之后的 HTTP 请求参数，并从 HttpServletRequest 对象中获取异步操作上下文，交给 Flower 框架进行后续处理。Flower 框架接收 HTTP 异步请求的时序如图 3.15 所示。FlowRouter 流程路由

器将该封装后的消息通过异步调用的方式，传递到相应的流程通道，由 ServiceRouter 服务路由器完成后续的服务路由；同时在服务上下文产生一个 Web 对象用于存储端点位置。

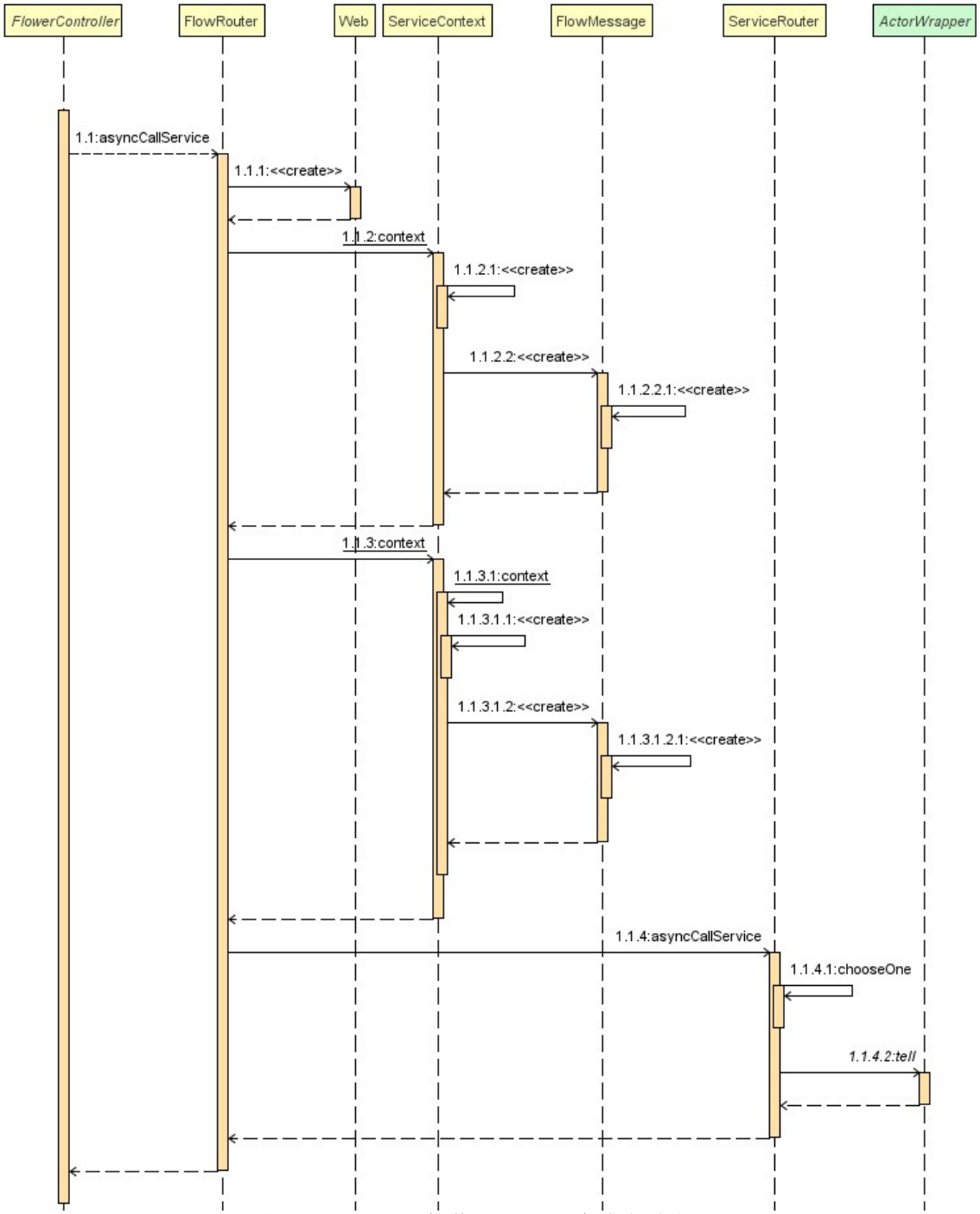

图3.15　Flower 接收 HTTP 异步请求时序图

FlowerMessage 继承自 Message 接口，是 Message 的默认实现，流程内流转的消息都会被默认封装成 FlowerMessage，存储着开发者传递消息模型的字节序列以及唯一的消息编号。

(2) Spring 容器集成

Spring 容器在启动的时，框架自动启动 Flower 容器，并将 FlowerService 注入 Spring 容器，加载成为 Spring Bean 对象，共享 Spring 组件，实现 Flower 容器在 Spring 中的集成。实现类图如图 3.16 所示。

FlowerComponentScan 注解用于 Spring-boot 启动类，为 Spring 容器增加扫描 Flower 框架组件的包路径，以便 Spring 发现并加载开发者实现的 FlowerService 以及 FlowerController。FlowerComponentScanRegistrar 用于支撑 FlowerComponentScan 注解的功能，实现了 Spring 提供的 ImportBeanDefinitionRegistrar 接口，将在实例启动后自动扫描组件。

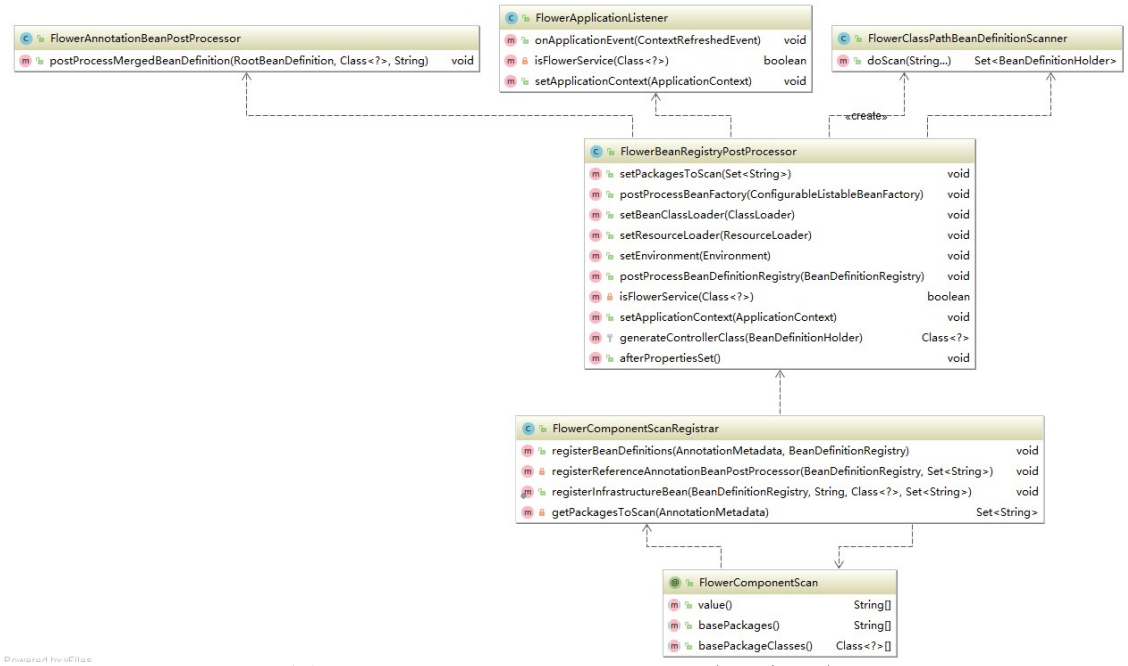

图3.16 FlowerService 注入 Spring 容器实现类

FlowerApplicationListener 在 onApplicationEvent()方法中监听 Spring 容器的启动，Spring 容器启动后，将被扫描发现的 FlowerService 注入 Flower 容器。

FlowerAnnotationBeanPostProcessor 将 FlowerService 注解加入到 Spring 自动注入的列表，是实现 Spring 中集成 FlowerService 至关重要的一个步骤，使 FlowerService 完美融入 Spring，可以轻松使用 Spring 的功能。

FlowerBeanRegistryPostProcessor 以及 FlowerClassPathBeanDefinitionScanner 则是具体的 FlowerService 注入过程的实现类。使用单例工厂的方法创建 FlowerService 对象，并封装到 Actor 模型。

注入 Spring 容器后的 FlowerService 就如同开发 Spring 一般简洁，没有晦涩难懂的框架依赖与代码逻辑，可以使用 Spring 提供的自动注入功能，注入数据库等组件，无入侵的实现一个反应式系统。

3.4 社会、安全、法律、文化、易用性因素

(1) 社会影响。Flower 框架的设计将流式计算应用到反应式系统的开发，是一次创新的探索，为社会带来一种新的系统开发方式，这样的方式更加符合现代人的交互规则，更容易被开发者所接受。Flower 框架也为团队开发提供便利，互不耦合的服务代码将单一职责划分的更清晰，更容易被开发团队所接收。

(2) 安全性考虑。Flower 框架是一个基于消息驱动的反应式编程框架，消息中不仅仅含有框架内部驱动服务需要的参数，也含有用户的数据。为了保护用户的数据，框架在传递消息时，先对消息进行字节序列化操作，拷贝并封装成统一的消息结构，然后再进行传递；对外隐藏消息细节，保护用户消息中的数据。

(3) 法律因素。Flower 框架选用开源、免费的 Akka Actor 作为底层的反应式组件，为 Flower 框架以后的发展以及推广减少法律方面的纠纷。同时，Flower 框架基于 Apache 2.0 版本开源协议发布，符合开源软件开发流程。

(4) 文化影响。反应式编程在开源社区的热度持续上升，Flower 框架也是潮流中的一部分，为反应式编程服务，目的是为了简洁、高效地得到一个反应式系统。其次，Flower 框架的设计结合微服务，应用流式计算的思想，提供可分布部署的反应式微服务。反应式与微服务的结合，符合现代高并发环境的发展需要，是一次文化的碰撞与创新。

(5) 易用性考虑。框架的使用是为了加快系统的开发过程，Flower 框架同样需要做到这一点。Flower 框架不仅仅是为了得到一个反应式系统，还需要为开发者着想，提高框架的易用性，让开发者能够低成本学习、快速应用、高效开发。

第4章 系统测试

4.1 测试方法

Flower 框架的现有功能已经初步完成，但是并不能直接发布代码，为了保证框架功能的完整性，需要对框架进行基本的功能测试。此外，为了直观的感受到 Flower 框架带来的好处，需要对其做性能测试，并与相关系统做比较。功能测试可以采用 Junit 单元测试方案，直接为框架书写测试用例；性能测试则主要集中对 Web 应用程序进行测试，对其做接口的压力测试。

4.2 测试方案及计划

4.2.1 测试方案

(1) 功能测试

使用 Junit 测试工具，编写相关的测试用例，验证测试结果的通过与否。

使用 Junit 测试代码验证框架功能的完整性，只是保证了代码的正常编译，运行是完整的，框架的实际情况需要通过真实的项目实践，在实践中推演框架的优势及不足，也为框架以后的发展提供方向。

(2) 性能测试

设计一个测试场景，分别使用 Flower 框架和 Spring 框架开发两套解决方案，在同一台计算机上运行实例，使用 JMeter 测试工具对该场景执行压力测试，收集并分析测试报告。

两次实例测试的服务器参数如表 4.1 所示。

表4.1 服务器的配置

项目		基本配置
硬件配置	CPU	Intel 酷睿 i5-5200U CPU 2.7GHz
	内存	4GB
	硬盘	500GB
	显卡	NVIDIA
软件配置	操作系统	Window10
	数据库服务器	MySql5.6

4.2.2 测试计划

Flower 框架的功能测试环节是在每一次核心代码的修改之后，提交更新代码前都必须做的一项工作。在团队项目的开发过程中，这是保证我们的代码能够被团队所接受的前提。其次 Flower 框架在起步初期就有相关的应用项目在跟进，我们也可以在项目的使用中去发现框架本身存在的缺陷以及可提升空间。

主要的测试工作围绕以下计划展开：

（1）消息处理模式测试

消息是 Flower 框架的一等公民，消息驱动着整个框架的运转，对消息处理模式的测试不可或缺。应重点测试消息聚合以及消息分叉功能。消息分叉产生的直接效应是服务的并行，是 Flower 框架性能提升的一项重要指标；而消息聚合则是并行服务结果的收集方案。

（2）流程定义测试

流程定义即服务的编排、业务的处理逻辑， Flower 必须正确有效的执行用户编排的流程定义。

（3）Web 模块的 Spring 整合测试

Spring 框架是目前热门的开发框架，它已经成为了一名 Java 开发者最基本的入门课程，对于 Flower 框架而言，Spring 的熟知程度给 Flower 的应用带来很大的便利，熟悉 Spring 的开发者一定能够在极短的时间内使用 Flower 框架，简单易用也是我们的开发目标。在这一部分的测试中，除了对功能的完整性有要求，还需要尽可能的简单使用。我们已经在这一方面做了很大的努力，但是希望能做到更简单易用。

（4）性能测试

性能测试主要体现在大量并发用户访问时，对比系统是否达到承受能力边界、解析请求能力是否依旧存在，分析系统运行资源情况，以验证 Flower 框架设计是否确实能够带来性能提升。我们关注的参数有响应时间，并发数，吞吐量、系统压力极值。

（5）分布式部署测试

分布式部署模块主要测试 Service 服务混合编排的能力。Service 服务混合编排即通过注册中心编排远程的 Service 服务，定义流程，完成业务逻辑。混合编排的能力是分布式部署中必不可少的技术，也是实现流式计算的重要组成部分。

4.3 测试过程及结果分析

4.3.1 消息简单传递测试

（1）测试用例

执行 common.sample.programflow.Sample.java 测试文件，该测试文件分别创建 ServiceA, ServiceB, ServiceC 三个 Service，调用 5 次如下的流程定义：

ServiceA-> ServiceB ->ServiceC

ServiceA<String,String>功能输出文本：I am ServiceA！同时原样返回消息。

ServiceB<String,MessageA>功能：接收文本消息，转换成大写序列，并返回含有执行列号的消息。

ServiceC<MessageA,Void>功能：接收带序列号的消息，并打印文本。

传递的初始消息为：Hello World!

（2）执行结果

I am ServiceA！

I am ServiceA！

I am ServiceA！

I am ServiceA！

I am ServiceA！

HELLO WORLD! I am 4

HELLO WORLD! I am 3

HELLO WORLD! I am 1

HELLO WORLD! I am 2

HELLO WORLD! I am 0

（3）结果分析

本测试用例是对简单消息传递的测试，消息正确地被 ServiceA 处理并驱动了后续流程。验证了 Flower 框架的消息传递功能，同时从测试用例的结果输出顺序可以证明，Flower 框架中 Service 的执行确实是异步的。

4.3.2 消息分叉测试

（1）测试用例

定义测试流程：流程编排如图 4.1 所示。

ServiceA<String,String>输出文本：I am ServiceA！同时原样返回消息。

ServiceB<String,Void>输出文本：I am ServiceB！

ServiceC<String,Void>输出文本：I am ServiceC！

传递的初始消息为：Hello World!

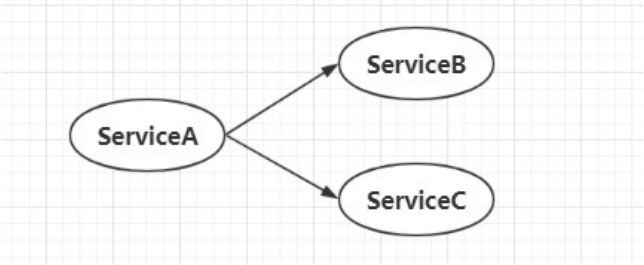

图4.1　消息分叉流程编排

（2）执行结果

I am ServiceA！

I am ServiceC！

I am ServiceB！

（3）结果分析

本测试用例对 Flower 框架中，消息分叉模式进行验证，从输出结果看出，ServiceA 的返回消息同时驱动了 ServiceB 以及 ServiceC 的执行，以此证明了通过消息分叉的流程编排可以正确的驱动服务，达到服务并行的效果。

4.3.3 消息聚合以及文本流程定义测试

（1）测试用例

创建 sample.flow 文件，并在文件中定义流程，sample.flow 文件内容如图 4.2 所示。

```
service1 -> service2
service1 -> service3
service2 -> service5
service3 -> service5
service5 -> service4
```

图4.2　sample.flow 文件

创建 sample.services 文件，在文件中定义 Service，文件内容如图 4.3 所示。

```
service1 = com.ly.train.flower.common.sample.textflow.service.Service1
service2 = com.ly.train.flower.common.sample.textflow.service.Service2
service3 = com.ly.train.flower.common.sample.textflow.service.Service3
service4 = com.ly.train.flower.common.sample.textflow.service.Service4
service5 = com.ly.train.flower.common.service.impl.AggregateService
```

图4.3　sample.services 文件

（2）执行结果

I am Service1！

I am Service3！Send Message: 3

I am Service2！Send Message: 2

I am Service4！Sum of Messages: 5

（2）结果分析

本测试用例中，流程编排以及服务的定义通过额外的文件来完成，同时定义的流程中，使用了 Flower 框架内置的聚合服务，因此本测试用例还对聚合服务进行了测试。sample.flow 文件和 sample.services 分别定义了流程和 Service，其中 Service5 是 Flower 框架内置的消息聚合服务，用于聚合分叉的消息，Service4 接收聚合服务的返回结果，并求和。从输出的结果可以看出，服务以及流程定义被正确加载，聚合服务正常执行。

4.3.4 整合 Spring 及性能测试

（1）测试用例

设计测试场景：通过学生的 id 编号查询学生的班级信息，选课信息以及个人信息。这三个操作互不干扰，最后将结果集合并返回。

开发 Flower Web 示例程序，集成 Spring：流程编排如图 4.4 所示。

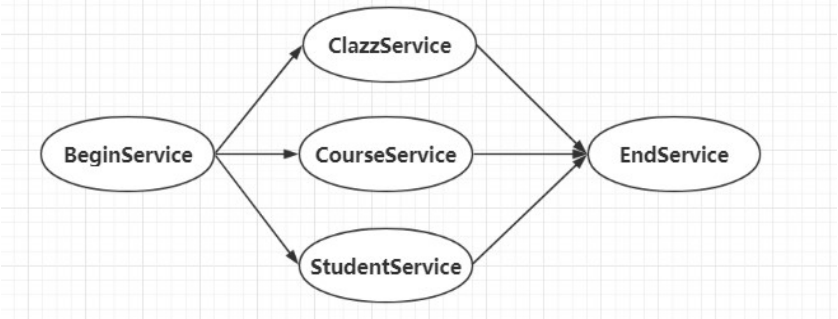

图4.4 整合 Spring 及性能测试流程编排

BeginService 接收网关 FlowerController 提供的 HTTP 请求参数，即用户 Id,将用户 id 分发至 ClazzService、CourseService 以及 StudentService 并行查询数据库，获取相应的消息。EndService 通过 FlowerService 注解成为消息聚合服务：

@FlowerService(type = FlowerType.AGGREGATE)

```
public class EndService extends AbstractService<List<Object>, Object> implements Flush, HttpComplete, Complete {
    @Override
    public Object doProcess(List<Object> message, ServiceContext context) throws Throwable {
            Response<List<Object>> res = R.ok(message);
            String ret = JSONObject.toJSONString(res, true);
            context.getWeb().printJSON(ret);
            return null;
    }}
```

EndService 用于接受分叉的三个 Service 的返回消息，封装成消息，返回页面。

开发 SpringMVC 方式获取相关数据的示例程序：

```
@Service
public class StudentMvcService {
    @Autowired
    ClazzDao clazzDao;
    @Autowired
    CourseDao courseDao;
    @Autowired
    StudentDao studentDao;
    public Response<List<Object>> getInfo(int id){
        List<Object> list = new ArrayList<>();
        list.add(studentDao.getById(id));
        list.add(courseDao.getByStudentId(id));
        list.add(clazzDao.getByStudentId(id));
        return R.ok(list);
    }
}
```

（2）编写 JMeter 测试脚本，并分别对以上两个 Web 应用程序执行压力测试，测试过程保证运行环境一致。

（3）结果分析

收集测试数据，绘制系统吞吐量变化曲线以及出错率变化曲线。如图 4.5 ，图 4.6 所示。

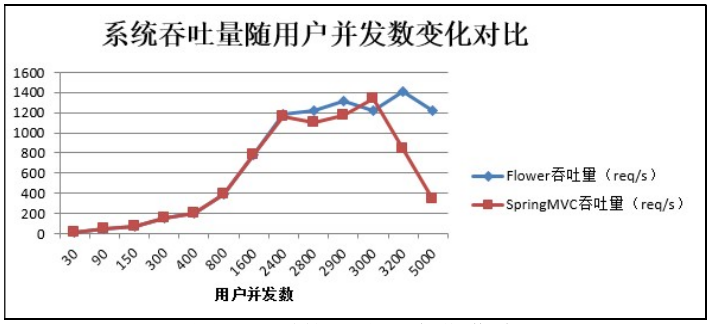

图4.5 系统吞吐量变化曲线

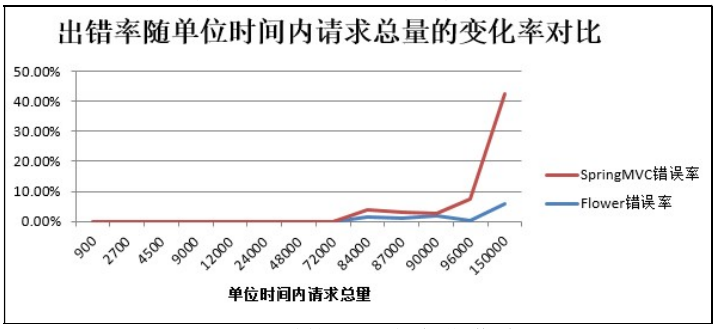

图4.6 系统出错率变化曲线

通过以上的测试结果，可以看出 Flower 框架对系统性能的提升体现在高并发下仍然保持响应，提升系统稳定性，对失败更加宽容，降低系统出错率。Flower 框架将更充分的利用系统资源，将系统的并发极值提升，推延系统崩溃压力值的出现。这也就意味着，在服务器集群中替换成 Flower 框架实现系统，即使缩减一半甚至更多的服务器，也能提供一样优质的服务。

4.3.5 分布式部署测试

（1）测试用例

在服务器运行 Flower 注册中心，注册中心已经由 Flower 框架打包，可直接使用。

创建业务 Flower 容器示例 A,并在配置文件配置注册中心地址，启动业务容器 A。

同理，为了测试多远程服务和本地服务混排，再次创建 Flower 容器实例 B,与实例 A 配置同一个注册中心。

创建网关容器，同理需要注册到注册中心，为网关容器提供 FlowerController 的实现，进行流程混排，如图 4.7 所示。

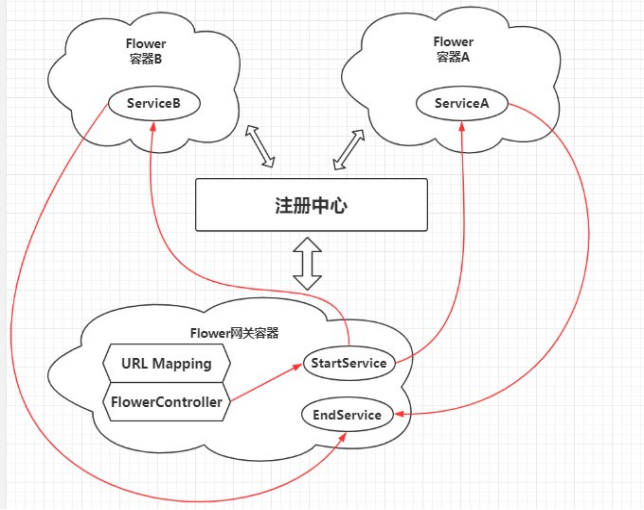

图4.7 分布式部署及流程混排

（2）结果分析

本测试用例测试了 Flower 框架在分布式部署之后，将本地服务以及远程服务混合编排时，流程定义以及服务调用的正确性。正确执行测试用例，证明 Flower 框架确实具有分布式部署的能力。

远程服务混排需要注意的是，使用服务名进行流程编排，因为服务实例不在容器内，JVM 无法获取到相应的类信息。注册中心必须优先启动，因为容器在启动之后将会自动寻找注册中心进行服务注册。

第5章 总结与展望

5.1 总结

Flower 框架是一个低学习成本的反应式编程框架，降低反应式编程的门槛，极大地简化反应式系统的开发难度，帮助现有的同步系统开辟一条更加符合现代人的异步交互方式。Flower 框架也在系统性能方面做出努力，充分利用多核处理器核心，增加系统的资源利用率，提高应用的并发承载上限，是更符合现代并发系统的开发方案。

论文主要完成了 Flower 框架服务到 Actor 模型的封装、消息处理模式的实现、HTTP 请求支持以及系统测试四个方面工作：

(1) 框架服务到 Actor 模型的封装。框架服务即 FlowerService，是 Flower 框架中最小的执行单元；利用底层的 Akka Actor 对象来加载、执行用户逻辑，形成服务，使服务能够被单独执行。 一个用户逻辑可被多个 Actor 对象加载，让服务能够并发地执行。

(2) 消息处理模式实现。消息是服务的输入与输出，也是服务执行的驱动力，是 Flower 框架中的一等公民。服务发出的消息不具有指向性，因此服务之间没有直接依赖，框架通过一套额外的流程定义，联系多个服务，形成服务通道。通过流程定义，将消息一对一的传递、一对多的分发、多对一的聚合，驱动一个又一个服务的执行，最终完成用户逻辑。当消息进行一对多分发时，通过开发内置的条件服务，根据结果控制分发的节点数量，做到运行时控制分发。

(3) HTTP 请求支持。Web 应用程序天然地运行在一个并发的网络环境下，而 HTTP 作为网络请求中最基本的协议，HTTP 请求的处理也具有并发性。Flower 框架提供 Web 开发能力，但并不直接解析 HTTP 请求，而是通过集成 Spring 的 Web 模块来处理；在实现上为每一个请求路径提供至少一个服务通道，将 Spring 解析 HTTP 请求后的请求参数封装成消息，作为服务通道中第一个服务的驱动消息，进而根据流程定义驱动整个服务流程的执行。

(4) 系统测试。论文对框架实现之后的核心功能进行了黑盒测试，通过执行测试用例程序，测试相关功能的完整性。测试用例在制作的过程中，充分考虑功能的弊端，测试其应对情况。其次，论文还在框架性能方面做出对比测试；使用 Flower 框架以及当今主流的 Spring 框架分别开发相同业务场景下的 Web 程序，在 JMeter 脚本下执

行压力测试，收集测试数据，对比二者性能。通过测试证明，Flower 框架能够准确地执行用户逻辑，有效地加强系统稳定性以及响应性，极大提升系统对并发的应对能力，提高执行效率。

5.2 展望

Flower 框架的设计理念以及实现方式并不复杂，但是其产生的效应却是能突破现有系统的一些瓶颈，无论是对开发者，还是对系统本身而言，都是一次难得的提升机会。为了更好的利用 Flower 框架开发系统，体验更优质的性能提升，更便捷的开发方式，提出以下五点建议：

(1) 提前进行流程设计。设计好 Service 服务的粒度以及编排，流程设计好了，系统的架构也就设计完成了。

(2) 进行消息设计。Flower 框架将消息作为一等公民，消息也是服务之间的接口，合理的设计消息，服务的输入输出就有了约束，在团队开发时只需要遵循消息的设计来开发服务，开发者之间不需要彼此依赖，提高团队开发的效率。

(3) 尽量提高并行服务的数量。Flower 框架不仅仅对系统的并发承载能力做出提升，同时也在消息的处理流程上提供更优的并行处理方案，并行对不同资源消耗的服务，可很大程度的提升系统的资源利用率；例如 CPU 访问密集的操作、IO 密集的操作。

(4) 请勿阻塞等待，尽可能的减少消息回复的使用。Flower 框架提供反应式编程方案，期望做异步非阻塞的操作，底层基于 Akka 构建，阻塞等待将挂起底层的 Akka 线程，在并发量增大的时候，加速系统资源的耗尽，不符合反应式编程的开发理念。

(5) Flower 框架核心提供了同步调用的方式，但是并不建议去使用它，使用同步调用的方式将失去反应式的体验，异步操作是反应式系统中最基本的元素。

Flower 框架致力成为继 Rx.Net、RxJava、RxJava2 以及 Reactor 之后的新一代反应式编程框架；框架本身处于刚刚起步的阶段，个人的力量尚且有限，功能以及性能也将持续扩展与提升，我将更加努力的学习相关技术知识，紧随时代的发展、技术的革新，为 Flower 框架增添更多的特色，使其功能更为完善、对系统性能的提升更为有效，为反应式系统提供更强的特性支撑。同时也欢迎志同道合的朋友一同加入，为 Flower 框架的发展增添一抹色彩。

致谢

首先，我要感谢我的父母，20 年来任劳任怨的供我读书学习，教我成长，教我为人处事。

感谢同程艺龙首席架构师李智慧老师对我的认可与栽培，让我有机会加入 Flower 框架的项目研发组，接触一个全新的技术领域。李智慧老师是一名资深的技术专家，曾发表多项技术专利，Flower 框架项目正是由李老师设计并发起的。在同程艺龙公司实习期间，每次聆听他的分享会，无论是技术上还是思维方式上总能收获满满。在 Flower 项目的研发过程中，遇到任何问题李老师总是耐心且细心的为我们讲解，寻求最为合适的解决方案。

感谢阮威导师，在同程艺龙实习期间对我的悉心指导；感谢为 Flower 框架贡献代码的同事们，他们在业务繁忙期间也不忘 Flower 框架的研发进度，认真为 Flower 框架提供跟进项目，确保贡献的每一行代码都正确有效。

感谢毕业设计指导老师储珺教授。储老师一名十分有资历并且很和蔼教师，由于我在校外进行毕业设计，校内的大大小小事务都是通过储老师才得以完成。从选题、开题报告到现在的论文书写、毕业答辩，都离不开储老师悉心的指导、对每一份文档的耐心批改、提出的每一句宝贵建议。

感谢班主任段喜龙老师。感谢段喜龙老师在大学四年里对我们的包容与教诲，教会我们如何从高中生转变成为一名大学生，教会我们如何在大学里为人处事。段老师同时也是我的 Java 启蒙老师，以及软件测试教师。段老师在专业领域经验丰富，每课堂我们学到的都不仅仅是技术理论，还有更多的实践经验。

感谢辅导员温素梅老师。大学四年里与温老师的交流最为密切，在学校内的琐事都是温老师帮忙整理并通知到我们每一位学生。温老师对我们的关心就如母爱一般温暖，从开学的入学典礼到如今临近毕业的就业情况，都有温老师着急的身影，担心我们适应不了大学生活，担心我们没有找到合适的工作；每次假期前夕，温老师都会在第一时间强调安全；每一次请假条的递交，温老师都会尽心尽责。感谢温老师四年来的关心与照顾。

最后感谢母校南昌航空大学四年的培养，以及同程艺龙公司在我实习期间的大力栽培，给我学习与成长的平台，让我学会生活、学会工作，更快的融入社会。

参考文献

[1] 沈哲. RxJava2.x 实战[M]. 北京:电子工业出版社,2018.

[2] Henry Lee,Eugene Chuvyrov. Reactive Extensions for. NET[M]. Apress: Beginning Windows Phone App Development,2012.

[3] Lea D. The java.util.concurrent synchronizer framework[J]. Science of Computer Programming,2005, 58(3):293-309.

[4] Davis A L. Akka Streams[M]. Apress:Berkeley,2019.

[5] Shouheng Zhang,Shanan Zhu. Server structure based on netty framework for internet-based laboratory[C]. IEEE International Conference on Control and Automation,Hangzhou,China,June 12-14,2013.

[6] Freeman E. Head First 设计模式（中文版）[M]. 北京:中国电力出版社,2007.

[7] Konrad Malawski. Why Reactive?[M]. Safari:O'Reilly Media,2016.

[8] Philipp Haller,Martin Odersky. Scala Actors: Unifying thread-based and event-based programming[J]. Theoretical Computer Science,2008,410(2):1154-1168.

[9] Khare S,An K,Gokhale A,et al. Reactive stream processing for data-centric publish/subscribe[J]. ACM Press the 9th ACM International Conference,2015,15:234-245.

[10] Vaughn Vernon. Reactive Messaging Patterns with the Actor Model: Applications and Integration in Scala and Akka [M]. Massachusetts:Pearson Education,2016.

[11] Karmani R K,Shali A,Agha G. Actor frameworks for the JVM platform[C]. Proceedings of the 7th International Conference on Principles and Practice of Programming in Java,Alberta,Canada,August 27-28,2009.

[12] 武昆，魏东岚，李家. 基于反应式设计的网站研究与实现[J]. 软件,2017(05):91-96.

[13] 张晶，王琰洁，黄小锋. 一种微服务框架的实现[J]. 计算机系统应用,2017,26(4):82-86.

[14] 王方旭. 基于 Spring Cloud 实现业务系统微服务化的设计与实现[J]. 电子技术与软件工程, 2018(8):60-61.

[15] 李智慧. 大型网站技术架构：核心原理与案例分析[M]. 北京:电子工业出版社,2013.

[16] Clement Escoffier. Building Reactive Microservices in Java [M]. Gravenstein Highway North: O’Reilly Media,2017.

[17] Dave Farley,Roland Kuhn,Martin Thompson. The Reactive Manifesto[EB/OL].

https://www.reactivemanifesto.org,2014.

[18] John Hunt. Introduction to Akka Actors[M]. New York:Springer International Publishing,2014.

[19] Michał Zielonka,Jarosław Kuchta,Paweł Czarnul. From Sequential to Parallel Implementation of NLP Using the Actor Model[M]. New York:Springer International Publishing,2018.

[20] 李斐. 基于 Akka 的分布式集群运维系统设计与实现[D]. 南京:东南大学,2017.