

内部

中国科学技术大学

工程硕士学位论文



基于 Spring 和微服务架构的 寄件快递单处理系统的设计与实现

作者姓名：

工程领域：

校内导师：

企业导师：

完成时间：

Limited

University of Science and Technology of
China
A dissertation for master's degree
of engineering



**Design and Implementation of
Processing System for Sending Order
Based on Spring and
Microservice Architecture**

Author's Name:

Speciality:

Supervisor:

Advisor:

Finished time:

摘 要

随着电子商务的发展,我国的快递业务快速增长,从 2010 增长到 2014 年,我国快递的年平均增长率达到 29.6%,全国快递服务企业在 2015 年业务量累计达 206 亿 7000 万件。业务量的不断增加,给快递行业的信息系统建设带来新的挑战,尤其对快递流程的末端业务的处理提出了更高要求。目前末端快递业务仍主要依靠物流运单系统开展,末端业务如寄件业务缺少规范、灵活的线上管理。目前寄件订单服务主要将寄件单按照普通的快递单来处理,揽件时间由快递员自行安排,仅提供参考价格,存在快递员操作不规范,用户体验差,服务确定性得不到保证,系统拓展性不高等问题。

针对末端寄件服务存在的上述问题,本课题在充分调研末端寄件需求,分析业务细节的基础上,基于 Spring 框架,采用微服务架构思想,使用现有的中间件平台,设计整体系统架构,构建并实现一套完整的寄件快递单处理系统,并对系统进行测试和验证。系统实现消费者和快递员线上线下的业务协作,发起和执行寄件订单流程,基于位置进行订单分派,显示跟踪物流信息等功能,并且能接入移动端应用,对运营活动进行后台管理,以及对微服务进行方便地使用和管理。将系统部署到实际生产环境,结合实际数据,对系统的设计和实现效果做分析,最终使系统达到稳定处理一定规模访问量,提供时间确定和价格确定的可靠服务,灵活拓展末端新业务的目的。

关键词: 寄件快递单 处理系统 微服务

ABSTRACT

With the development of electronic commerce, the courier business is growing rapidly in China. The average development index of the courier business has reached 29.6% per year. The volume of the courier business has reached 20.67 billion in 2015. The rapid increase of the business volume brings new challenges to the courier business's information system, especially in the processing of the courier business's end part. At present the main business of the end part conducted relying on the logistics waybill system while the end courier business like the sending business lacks standard and flexible online management. The current order service for sending is primarily processed according to the ordinary delivery orders, and the time is arranged at random. And the service only provides the reference price. The existing problem is that the courier operations are not standardized, user experience is poor, no deterministic service quality is ensure and the system scalability is not high.

To solve these problems of the end service for sending, this subject conducts the full investigation of the needs for sending and the detailed analysis of the business, and adopts the Spring framework, the microservice architecture and the existing middleware platform to design the overall system architecture, build a complete processing system for sending order, test and validate the system. This processing system of this subject implements the online and off-line business collaboration between couriers and consumers, starts and executes the process of orders for sending, dispatches the orders based on position, displays the tracking logistics information. The system can also connect the mobile application, manage the background operational activities, use and manage the microservice conveniently. This subject deploys the system in the production environment, analyzes the result of design and implementation of the system. And this subject finally makes the system read the target of providing the stable treatment of visits which reaches a certain size, the reliable service of deterministic time and price and the flexible scalability of end business.

Key Words: sending order, processing system, microservice architecture

目 录

摘要	I
Abstract.....	II
目录	III
第一章 绪论	1
1.1 论文研究背景	1
1.2 论文研究的目的和意义	2
1.3 国内外技术研究应用现状和发展趋势	3
1.4 论文的组织安排	5
第二章 关键技术介绍	6
2.1 寄件快递单处理系统概述.....	6
2.2 微服务架构.....	6
2.2.1 软件拓展模型	6
2.2.2 三种软件拓展模型的优缺点	7
2.2.3 微服务架构的优点	8
2.3 Spring 框架	9
2.3.1 主流 web 服务框架对比	9
2.3.2 Spring 框架的组成和优势	10
2.3.2.1 Spring 框架的组成	10
2.3.2.2 Spring 框架的优势	11
2.4 HSF 框架、中间件平台和无线组件	12
2.4.1 HSF 框架介绍.....	13
2.4.2 中间件平台	13
2.4.3 无线组件	13
2.5 其他技术.....	14
2.5.1 iBatis	14
2.5.2 Mockito	15
2.6 本章小结.....	15
第三章 寄件快递单处理系统的需求分析	16
3.1 用户角色分析.....	16

3.1.1 寄件人	16
3.1.2 快递员	18
3.1.3 运营人员	19
3.2 功能性需求分析	20
3.3 非功能性需求分析	22
3.4 本章小结	22
第四章 寄件快递单处理系统的设计	23
4.1 系统总体设计	23
4.1.1 子系统划分	23
4.1.2 系统总体架构	23
4.1.3 系统的开发框架	25
4.1.4 微服务的管理方案	28
4.2 寄件订单子系统的设计	30
4.3 寄件派单子系统的设计	32
4.4 寄件运营后台子系统的设计	33
4.5 数据库设计	34
4.5.1 数据库的 ER 图	35
4.5.2 数据库的表的设计	36
4.6 本章小结	39
第五章 寄件快递单处理系统的具体实现	40
5.1 寄件订单子系统	40
5.2 寄件派单子系统	44
5.3 寄件运营后台子系统	48
5.4 微服务管理的实现	51
5.5 本章小结	53
第六章 寄件快递单处理系统的测试和分析	54
6.1 寄件快递单处理系统的测试方案	54
6.1.1 单元测试方案	54
6.1.2 集成测试方案	54
6.2 寄件快递单处理系统的测试用例	55
6.2.1 寄件订单子系统的测试用例	55
6.2.2 寄件派单子系统的测试用例	57

6.2.3 寄件运营后台子系统的测试用例	58
6.2.4 线上 API 数据统计和性能分析	59
6.3 本章小结	59
第七章 总结和展望	60
7.1 本文的主要工作	60
7.2 今后待研究的问题	60
参考文献	61
致谢	64

第一章 绪论

1.1 论文研究背景

近些年来，国内的电子商务快速发展。2014 年我国的电子商务交易额为 13.37 万亿元，同比增长 28.6%；网络零售交易额达到 2.79 万亿元，同比增长 49.7%。2015 年预计达到 20.8 万亿元，网络零售额平均增长超过 50%，达到 4 万亿元，位居世界第一^[1]。

截止 2015 年 12 月，我国的手机网民规模达到 6 亿 2000 万个，占网民总数的 90.1%，移动互联网应用已经应用到基础的商务交易、信息查询等领域，塑造了全新的社会生活形态，潜移默化地改变着网民的日常生活，移动互联网应用在未来还将更加贴近网民生活^[2]。

随着电子商务在国内的繁荣发展，快递业迅速发展，2010 年至 2014 年中国的快递发展指数达到平均每年 29.6% 的增长率。2014 年中国快递业以每年 140 亿的业务量，超过美国成为世界上的快递业务量最大的国家^{[3][4]}。而快递业持续快速发展，2015 年全国的快递企业的业务量总计完成 206.7 亿件，同比增长 48%^[5]。从服务范围上，快递基本覆盖了全国所有区域，也到达了农村。比如，实际数据表明，淘宝旗下民营快递网络的 15 家民营快递公司，可交付包裹到达 50% 以上的乡镇，菜鸟供应链管理公司的配送网络能够实现全国 95% 的区县覆盖，快递服务网点在目前也基本实现了县一级的全覆盖^[6]。

如图 1-1 和图 1-2，是中国快递业务数据统计^{[5][7][8][9][10][11]}：

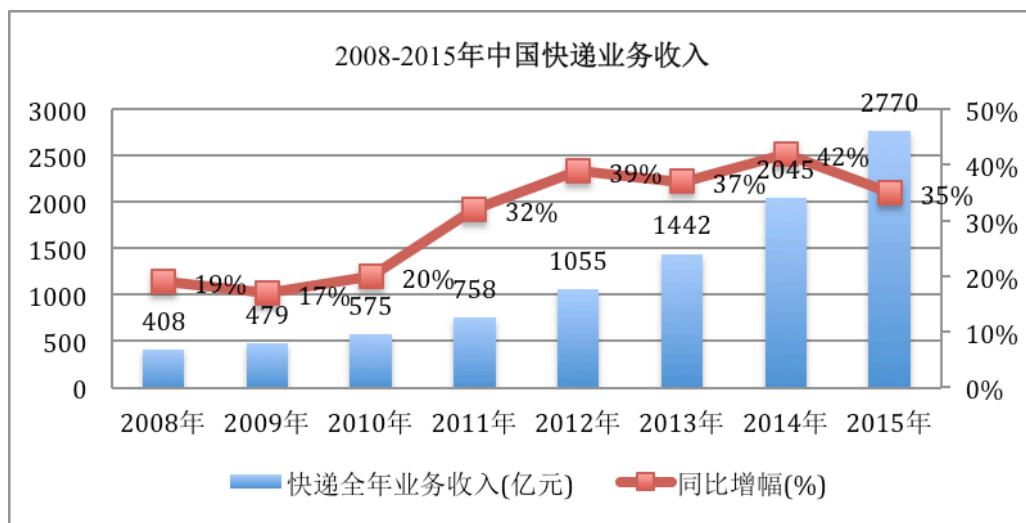


图 1-1 2008-2015 年中国快递业务收入

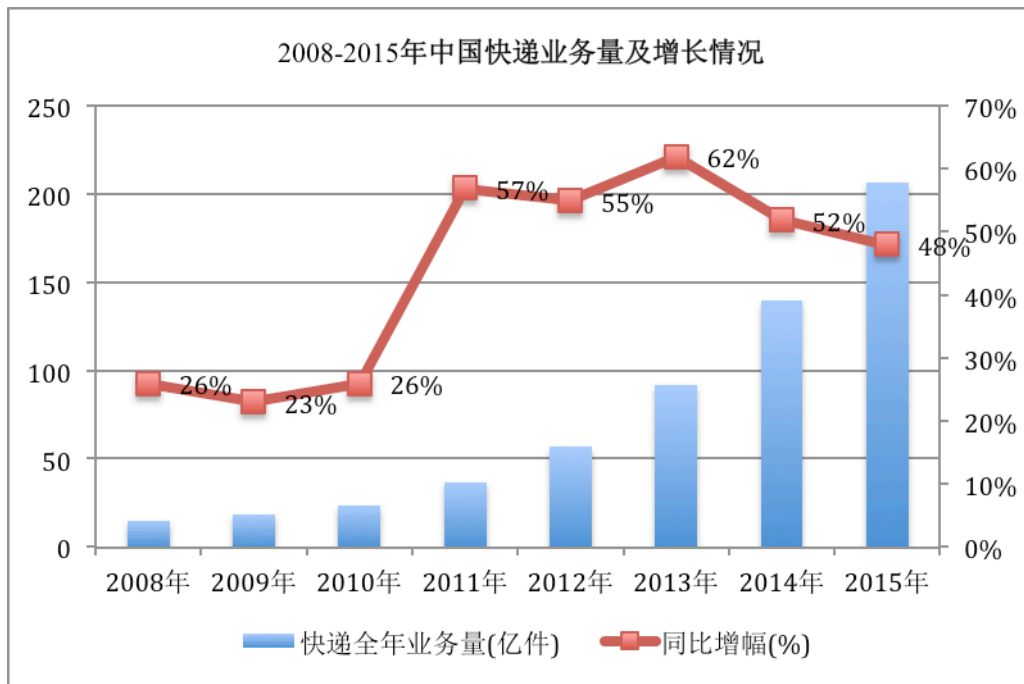


图 1-2 2008-2015 年中国快递业务量及增长情况

电商将实体经济从线下带到线上，快递行业又让商品实物从线上回归线下，互联网电商的影响，也在不断深化，不仅建立了完善的电子商务平台，更是对商品的供应、仓储、物流配送进行全渠道整合，在完善物流、服务业等方面，提高自身的软实力。

以快递员为主体的快递末端，不仅提供了主要的派送服务，也发展了上门寄件、同城寄件等业务。尤其随着电商无理由退换货等服务的提供，同城、二手交易的发展，更多寄件需求产生。

1.2 论文研究的目的和意义

近几年，移动互联网快速发展，06 年以来谷歌公司开发并继续完善了一个智能移动操作系统 Android，开放源码并授权手机厂商大量生产后，安卓手机持有量迅速上涨，苹果公司的 iOS 系统和手机硬件也吸引了大量用户，目前在智能移动手机市场在平分秋色。

对接安卓和 iOS 系统，能迅速地将软件延伸到最广泛的用户，直接面向消费者拓展寄件业务。同时减少了智能终端推广代价，初期没有终端硬件投入成本，只需要软件 APP 的营销推广，后期也只需维护应用层软件系统即可，这种方式也是移动互联网领域广泛采用的方式。

此外，移动端市场竞争激烈，需求持续变化，每月、每周都有新的活动和

业务变化，新的业务的种类也围绕核心业务在不断形成。相应的软件应用形式也在持续地更新，在保证基本用户习惯，核心流程稳定的前提下，应用可能持续向水平方向，以满足不断拓展的业务需求。伴随着软件推广和运营活动开展，订单的数量也会持续上涨，如果软件具有很好的伸缩性，则能提供良好的响应和用户体验。

目前快递员主要在网点收取寄件或者电话联系消费者，如果结合寄件订单处理，同时建立了快递员和消费者的实时消息联系，实现线上线下互操作，不仅方便了快递员，减少了沟通成本，也能够更加提供给消费者更加透明、更加确定的寄件服务。

综上，建立一套标准、规范、能直接对接移动 APP、能快速拓展的寄件快递单处理系统，具有广阔前景和实际社会价值。

1.3 国内外技术研究应用现状和发展趋势

（1）寄件快递单处理系统

国外的研究者，基于成熟的末端智能网络，在寄件快递单处理上，有具体的研究和应用实现。

比如 Michael A. Rivalto 在《System for Automated Package Pick-up and Delivery》^[12]中，围绕智能收寄设备，研究应用了一套自动化的快递件收寄处理系统，使用中央数据库负责存储收存、发送的订单信息。其中寄件订单的处理包括投寄、信息录入（目的地、快递公司、支付信息）、取件等环节，由中央服务器和数据库处理信息，系统包括发送通知给快递员的功能。

Ole-Petter Skaaksrud 和 Cameron Dee Dryden 等人在《Internet-based Shipping, Tracking, and Delivery Network Supporting A Plurality of Mobile Digital Image Capture and Processing (MICAP) Systems》^[13]中，借助图像处理，在手机端自动采集、生成快递单信息，在信息中心集中处理。

Ole-Petter Skaaksrud 等人考虑了快递包裹的提货、投送、运输等环节，以及运单信息的更新、包裹处理节点的消息通知等问题，在手机端 web 网络环境下，对处理系统做了具体实现。

国内的研究人员，如赵园园在《电子商务环境下社区智能快递系统助力快递末端配送效率提升》^[14]中，为快递末端围绕智能快递柜，建立智能配送系统，投递快递单信息，同时将系统拆分为快递存储子系统、短信支持子系统、监控子系统，描述了取寄件流程的具体细节。

许校境、郑召文在《基于 RFID 的快递系统的研究与应用》^[15]中，基于 RFID

技术实现了一套快递单处理系统,涵盖了寄件功能,并对包裹取走、短信发送、物流追踪等环节,以及单据的详细属性信息都做了分析。

武防震、姚国祥在《基于 UML 的快递系统建模》^[16]中,使用标准建模语言 UML,根据快递软件系统处理的寄件和派件业务,对快递单在物流链路变化过程中的状态和对应的软件模型做了整体设计。

以上应用和研究,主要围绕末端智能设备构建末端网络和快递单处理系统,针对寄件业务中快递员和消费者之间,基于位置、短信和寄件订单的流程处理,但在安卓、iOS 设备已经普及的前提下,缺乏实用性。在寄件快递单的处理上,对后端架构的设计缺少可扩展性,容错能力,没有提供时间和价格确定的服务保障。

随着智能手机越来越广泛的使用,逐渐增多的用户和不断增加的 APP 使用频率,一个能够直接对接智能手机 APP,连接主流快递公司的寄件订单处理系统,具有更大的适应性和实际价值。

(2) Spring 框架和微服务架构

Spring 框架是 Rod Johnson、Juergen Hoeller 等开发的,用于支持 JavaBean 构件运行的容器^[17]。Spring 框架基于依赖注入,提供了应用组件的机制和以 AOP 技术为基础的事务和日志管理功能。Spring 框架的出现,解决了传统的 J2EE 架构的 EJB 组件开发困难、数据存取效率低、单元测试难度大等问题。Spring 框架,作为一个轻量级架构,也解决了普通架构存在耦合度高、缺乏统一的事务日志管理服务不足。在行业中, Spring 框架已经得到了广泛的应用,也使得 J2EE 应用开发变得简单、高效、规范。

微服务架构已经被证明是一种可行的新型架构, Villamizar 和 M. Garces 等人指出,微服务架构能够实现高效部署可扩展的应用,在云计算技术的支持下,还能为企业级应用提供动态调整计算资源的能力。近年来,已经被国外大型互联网公司,比如 Amazon, Netflix 和 LinkedIn 等公司,在部署大型应用时所采用^[18]。国内的银行和电信行业,逐渐将微服务架构应用投入市场,例如,中国建设银行升级微信公众平台^[19],使应用由原有的单一推送功能,转变为具有多种业务功能的综合性服务平台,使用的就是微服务架构。

微服务架构将应用拆分为多个子服务,实现独立测试、部署、扩展、维护和升级,降低应用的复杂度,更加灵活高效地在云平台上扩展。微服务架构,在未来的企业业务细分,转型和升级,以及云计算平台技术迅速发展的大背景下,具有广泛的应用前景和实际的工业生产价值。

1.4 论文的组织安排

本课题是在快递业务高速发展，快递末端寄件业务不断拓展，快递员和消费者之间存在信息缺位，针对寄件服务的不足而提出的。其中涉及到多个方面的问题：服务保障、快递员和消费者的业务互操作、业务拓展、系统稳定等。课题拟基于微服务的思想，采用 Spring 框架，并结合某公司的相关技术平台，实现一个寄件快递单处理系统。课题主要包含以下 3 点内容：1.寄件快递单处理系统总体结构的设计；2.寄件订单、派单子系统，运营后台子系统的设计和实现；3.寄件快递单处理系统的测试和分析。

本文各章节安排如下：

第 1 章，绪论，介绍本文的研究背景和意义，国内外相关领域的研究现状和发展趋势，本文的主要内容和目标。

第 2 章，关键技术介绍，介绍寄件快递单处理系统使用的核心技术和方法，阐述 Spring 框架、微服务架构和所使用的平台技术和组件，说明了该系统采用的关键技术和及实现的技术方案。

第 3 章，寄件快递单处理系统的需求分析，介绍寄件快递单处理系统的具体需求，分析系统依赖的环境和要达到的目标。

第 4 章，寄件快递单处理系统的的设计，对寄件快递单处理系统的系统架构进行了设计，对子系统和子模块，做了具体细分和详细描述。

第 5 章，寄件快递单处理系统的具体实现，给出了主要子系统的实现方法，Spring 框架和微服务思想在实例中的具体应用。

第 6 章，寄件快递单处理系统的测试和分析，给出了寄件快递单处理系统的测试方案和具体测试用例。

第 7 章，总结和展望，对寄件快递单处理系统的技术思路和实现成果做了总结，分析了系统的创新点和论文的主要工作，系统的不足之处，指出了论文今后的工作和改进点。

第二章 关键技术介绍

2.1 寄件快递单处理系统概述

寄件快递单处理系统，是在寄件业务场景下，结合末端快递流程特点，抽象出寄件快递单的订单、派单处理等环节，提供服务功能、存储相关数据的系统。

本课题研究的场景涉及的基本概念，概括定义如下：

寄件：寄件人（普通消费者）联系快递员，快递员上门收取包裹，寄件人支付费用并在之后获得物流信息的过程。

快递单：在物流运单形成之前，寄件人在系统中创建，包含了快递投递必须的收件人地址、寄件人地址、物品类型和重量体积等信息，由系统指派给提供服务的快递员负责处理的电子订单的抽象。本课题研究的快递单属于订单的范畴。

对寄件快递单的处理：既包括了订单处理必须的下单、确认、分派、收款、发运等环节，还包括验证取件、回传运单，并在下单到分派环节增加了基于位置的派单信息推送，对价格的预估、调整，对分派后订单超时的处理等内容。

2.2 微服务架构

2.2.1 软件拓展模型

根据 Martin L. Abbott 的理论^[20]，软件拓展模型可以分为三个维度：

（1）X 轴向拓展：

基于负载均衡等方法，通过重复的方式，在多台机器上部署应用，运行并处理请求的方式，叫做 X 轴向拓展。

（2）Y 轴向拓展：

将应用分成多个不同的子服务，每个子服务负责一个或多个相关的功能，各个子服务负责的功能不同，这种拓展方式叫做 Y 轴向拓展。

有多种方法将应用分解为子服务，一种通过基于动词的语义进行分解，比如满足单一用例的结账服务。还有一种方法是通过名词将应用拆分，分解并建

立不同的子服务，分别负责某一个名词对应实体的相关的所有操作。应用一般使用上述两种方式拆分为对应的子服务。

（3）Z 轴向拓展：

与 X 轴向拓展类似，每个服务器运行同一份代码，但是不同点在于，每个服务器只负责整个数据的一个子集，系统负责将每个请求路由到合适的服务器组件。比如，一种路由策略将请求的某个属性作为访问如数据的主键，将不同的请求映射到对应的服务器上。或者将请求源进行区分，如普通用户和付费用户，将请求路由到不同的服务器上。这种拓展应用的方法，叫做 Z 轴向拓展。

三个维度方向上的软件模型的拓展可以用图 2-1 形象表示。

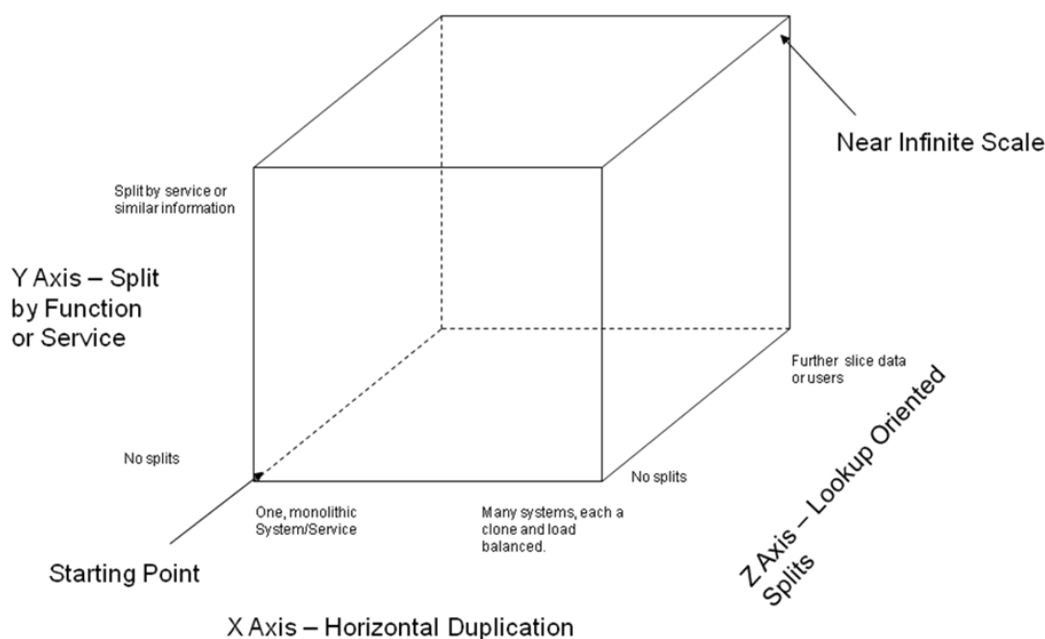


图 2-1 软件架构拓展的立方体模型

2.2.2 三种软件扩展模型的优缺点

（1）X 轴向拓展的优缺点：

X 轴向拓展的优点： 如果存在 N 份代码拷贝，运行在独立的服务机器上，则每一台服务器只承受 $1/N$ 的负载，拓展方式简单，因而广泛使用。

X 轴向拓展的缺点： 一个缺点在于每一份代码拷贝都可能访问到全部数据，机器缓存需要更大的存储空间，保证读取的效率。另一个缺点是解决代码规模增加引起的开发量的增加和应用复杂度的增加。

（2）Y 轴向拓展的优缺点：

Y 轴向拓展的优点： 基于功能将应用拆分，配合开发资源，降低每次迭代的开发量，减少了应用复杂度^[21]。

Y 轴向拓展的缺点： 对应用实现的功能要详细分析，分割功能存在部分取

舍的代价。

(3) Z 轴向拓展的优缺点:

Z 轴向拓展的优点: 每个服务器只处理数据的一个子集。它提高了缓存的效率, 减少了 I/O 流量, 减少了内存的使用量。因为在正常情况下, 一个请求分发到多个服务器上, 从提高了事务的拓展性。由于能保证部分数据访问失败, 提高了故障的隔离性。

Z 轴向拓展的缺点: 1.应用复杂度的提高。2.分配数据的分配策略相对复杂, 如果数据需要再次分配, 则复杂度更高。3.开发量增加和应用复杂度增加的问题仍然没有解决。

2.2.3 微服务架构的优点

微服务思想, 是 Y 轴向拓展的, 将整个应用拆分为多个“微服务”, 具有以下优势^{[22][23]}。

1.每个微服务相对较小, 在需求和设计上都更容易被开发人员理解; 每个功能对应的微服务可采用不同的技术方案、技术工具, 各子团队的开发人员使用对应的集成开发环境, 提高开发效率^[24]; web 容器的负担更小, 启动速度更快, 使的开发效率提高, 应用部署时间缩短^[25]。

2.每个服务独立于其他的服务, 部署上也可以独立进行, 方便更加频繁地部署新版本的服务^[26]。

3.扩展开发团队更加简单, 方便将开发人员组织在多个团队上。每个团队负责单个服务, 能独立于其他的团队开发、部署和拓展自己的服务。

4.更好的故障隔离。比如, 如果某个服务存在内存泄漏, 只有这个服务受到影响, 其他的服务能够继续处理请求。比较而言, 集成在一起的整个系统架构, 如果有一个组件不正常, 整个系统都会受到影响。

2.3 Spring 框架

2.3.1 主流 web 服务框架对比

本课题考虑的应用对接物流数据平台, 寄件快递单处理流程部分类似于电商订单的处理, 而且考虑了直接对接公司电商平台和中间件平台, 便于业务开展和技术复用, 考虑使用 Java 语言。同时, Java 语言作为一种流行的开发语言, 具有跨平台性, 以及面向对象的语言特性。对于接入主流 web 框架, 运行在 Tomcat 等开源服务器上, 具有毫无疑问的优势。同时 Java 自带垃圾回收机制,

舍弃了 C、C++对内存的管理要求，简化了开发复杂度，提高程序的可靠性和开发速度，对于 web 程序而言是一个很好的选择。目前 JavaEE 等规范建立了 web 开发标准，对于企业级的 web 应用，应对复杂的平台环境，达到企业级集成，高效的生产能力，企业级的应用组件，未来可保证的技术支持等方面，具有难以替代的优势^{[27][28]}，本课题主要考虑基于 Java（尤其是符合 JavaEE 平台标准）的 web 服务框架。

（1）Struts 框架

Struts 是一个 Java 的基于请求的 web 框架。Struts 2 遵循 MVC 模式，其中 C-控制器被称为动作控制器（Action Controller）。Struts 2 通过映射文件的配置将请求导向对应的动作（Action），而且通过动作将数据展示给用户。Struts 2 支持基于注解的配置方式，并且在 web 应用中使用 Action 类作为 Model。Struts 2 提供了强大的 API 支持对于过滤器的配置^{[29][30]}。Struts 在很多应用中和 Spring 搭配，应用在表现层，持久化用 Hibernate 实现，称为 SSH（Struts+Spring+Hibernate）框架组合^[31]。

（2）Play 框架

Play 框架是一个全栈的 Java Web 应用框架。Play 遵循 MVC 架构模式，主要特点是提高了开发效率，代码的热重载（hot code reloading）和程序错误信息在浏览器中直接显示。

Play 框架使用了 SBT（Simple Build Tool）提高构建速度，并且提供热重载的功能，使代码的改变的效果能在“编码+刷新”的流程中快速体现，并配合 Maven 进行代码仓库的管理，相比于其他框架代码改变能迅速获得反馈。

Play 框架提供了良好的对单元测试的支持，配合使用 Sonar 等测试工具快速实现单元测试。此外，Play 框架还能直接在浏览器中查看错误信息和发生错误的代码片段。

Play 框架使用了 JBoss Netty 作为 web 服务器，它实现了对请求的异步处理，实现了非阻塞的服务，区别于一些不支持异步处理的 JavaEE 框架。同时支持了 socket 和流处理（streaming）。

（3）Spring 框架

Spring 框架是一个建立 Java 应用的轻量级框架，支持包括 Java 的 web 应用。它也是一个基于 Java 的基于请求的框架，核心是 IoC 容器，通过依赖注入绑定对象组件。同时 Spring 还提供了事务管理和对其他各种技术框架的无缝集成^[32]。

2.3.2 Spring 框架的组成和优势

2.3.2.1 Spring 框架的组成

目前的 Spring 框架依据功能特性，分成了 20 个独立的模块，它们共同组成了 Spring 框架的核心容器、数据访问/集成、Web、AOP（Aspect Oriented Programming, 面向切面编程）、Instrumentation、消息和测试，如图 2-2 所示：

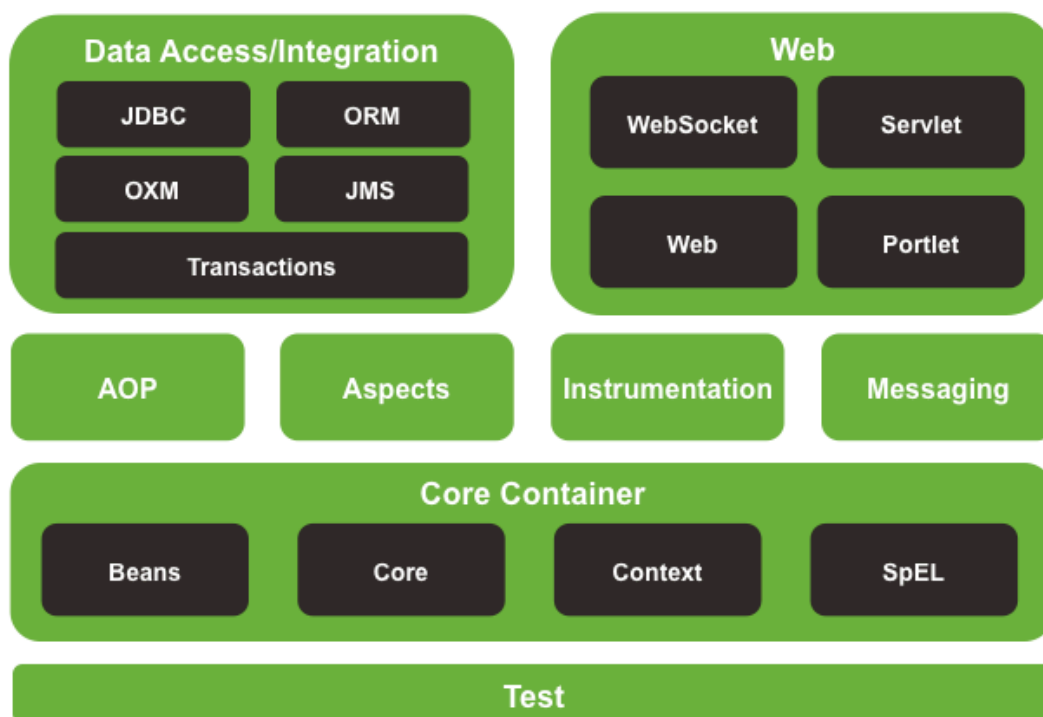


图 2-2 Spring 的组成模块

Spring 框架的核心：核心容器模块，包含了 `spring-core`，`spring-context`，`spring-beans`，`spring-context-support`，`spring-expression` 五个子模块。其中，`spring-core` 和 `spring-beans` 模块组成了整个 Spring 框架的核心组件，提供了对 IoC 和依赖注入的支持，并通过用工厂模式实现的 `BeanFactory`，将应用配置和依赖描述于程序的逻辑代码分离。

基于 `spring-core` 和 `spring-beans` 模块建立的 `spring-context` 模块，负责管理 Spring 中对象的生命周期，并提供了对事件的传递、资源文件的加载和上下文创建的支持。`spring-context` 模块还提供了对 Java EE 特性的支持。

`Spring-aop` 模块提供了对 AOP（面向切面编程）的支持，让程序能够通过方法拦截器和 `pointcut`（切入点）的定义，实现应用代码和共用的功能逻辑代码解耦的目的。

2.3.2.2 Spring 框架的优势

1. IoC (Inversion of Control 控制反转)。IoC 技术是一种将对依赖的组件的创建和管理转移到外部的技术。比如类 B 依赖于类 A 的实例，在传统编程方式中，要么实现在类 B 中实现类 A 的实例，要么通过工厂类获得类 A 的实例。采用 IoC 之后，类 A 的实例，是在运行时环境中，通过框架注入到类 B 当中的，这个操作称为依赖注入。而 IoC 是实现依赖注入的一种高效的方式，通过这种方式，Spring，作为一个容器，提供了应用类的所有依赖。通过显式配置或者自动绑定的方式，Spring 允许很少的开发量就能完成 IoC，实现了非侵入式和插件化的编程模型^[33]。

2. AOP (Aspect Oriented Programming, 面向切面编程)。AOP 是一种通过在不需要对核心业务代码做修改的情况下，实现横向的关注点（如事务管理、日志等）的技术。它可以通过预编译和运行时期动态代理的方式来实现。AOP 提供了在编程需要的时候，就能获取到像拦截器这样模块中的横切代码的能力，从而实现了横切关注点的功能^[35]。AOP 使的开发人员只需关注系统中主要业务逻辑部分，对于系统级别的日志处理、事务管理，提供了方便的切面式的实现方式^[34]。Spring 提供了基于代理的 AOP 能力。

3. Spring 还提供了对于 Junit4 的支持，方便单元测试，并且方便地集成各种开源框架。同时 Spring 还提供了数据访问\集成、web 编程的支持：如 JDBC、ORM、WebSocket、Servlet 等。

相比于 Struts 框架，Spring 提供了更好的对于后端的支持，其中 IoC 和 AOP 功能，在事务、组件充分解耦等方面，提供了对于我们的寄件快递单处理系统，这种企业级的关注于业务逻辑的系统的完整技术支持。

相比于 Play 框架，Spring 优势在于：（1）经历了很长一段时间的的发展，越来越成熟，现在有很多使用 Spring 实现的，能够直接产生联系的 web 应用程序。

（2）大量的技术文档，成熟的技术团队，开发者社区的技术支持。（3）Spring 提供了更加丰富的组件和 API 接口支持，能够集成各种开源框架，方便企业级拓展^[36]。

2.4 HSF 框架、中间件平台和无线组件

2.4.1 HSF 框架介绍

HSF (High-Speed Service Framework), 是基于 RPC (Remote Procedure Call) 的高性能分布式服务框架。HSF 的基础是 RPC, RPC 提供了远程调用的功能, 也就是, A 机器调用 B 机器上的一个对象的方法, B 机器将方法的返回值发送给 A 机器^[37], 如下图 2-3 所示。

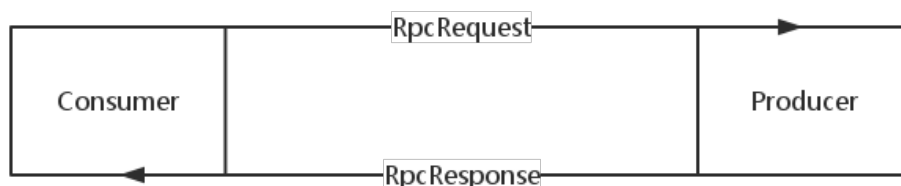


图 2-3 RPC 调用方式

RPC 的实现方式有多种, 可以是 RMI, Hessian 等。

在 HSF 中, 通过配置中心 (ConfigServer) 的设计, 让服务发布者 (Producer) 能够发布服务, 服务的调用者 (Consumer) 能够查找和调用服务, 如图 2-4 所示。

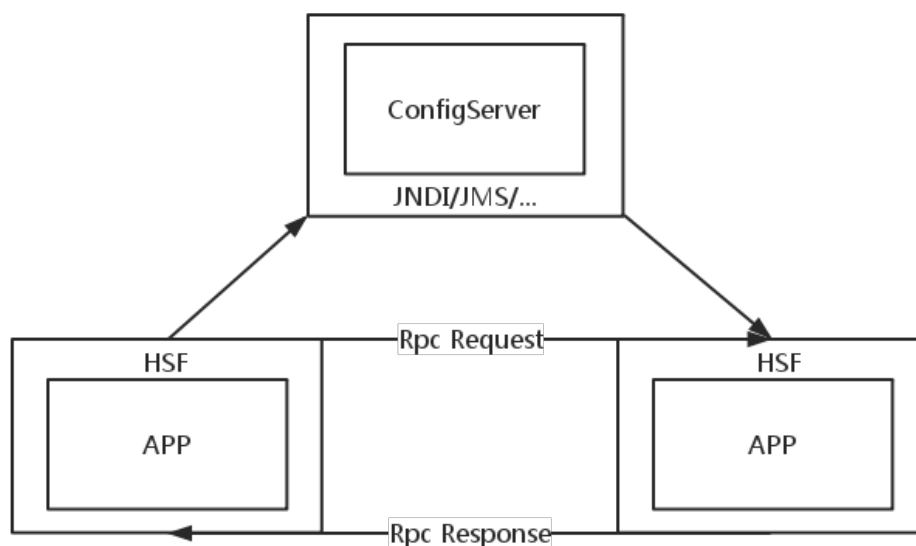


图 2-4 HSF 的配置中心设计

服务发布者负责发布服务, 它将服务注册到配置中心上, 配置中心采用数据库存储或分布式文件系统存储, 存储的内容包括服务的名称、服务发布方 (Producer) 的机器的 IP 地址、超时时间、序列化方式等信息, 并且通过索引和缓存提高查找速度。

在服务的发布者 (Producer) 和配置中心 (ConfigServer) 之间, 是典型的

CS 模式，可以通过 JNDI、JMS、WebService 等多种方式实现，调用配置中心提供的管理接口。JNDI 的接口返回所查找的服务的信息，以 Java 对象的形式组织，并序列化成字节流，通过网络传输返回给服务调用者。

服务调用者得到服务发布者的具体信息，包括 IP 地址、端口号、服务名称、超时时间等，然后利用上文描述的 RPC 实现，进行服务调用（具体实现上，HSF 实现了异步调用和对调用超时异常情况的处理）。

HSF 框架为不同的服务应用系统之间，即不同的服务集群之间的，可靠的服务调用提供了通信上的支持。

2.4.2 中间件平台

（1）Taobao Notify 消息中间件

Taobao Notify 中间件是一个 Message Oriented Middleware (MOM)，具有通用的 MOM 所具有的松散耦合和异步处理的特点^[38]，并且具有高可靠性、高可用、高效率、水平扩展、安全性，同时保证了消息 100%投递。

中间件将消息使用本地数据库存储，并使用事务操作消息的传递，能够可靠的管理消息的提交、存储、投递和订阅关系。

（2）定时任务队列中间件

所使用的定时任务队列 TMQ (Taobao Management Queue) 中间件，是基于 Redis 实现的消息队列，能够在分布式集群上快速处理定时任务。

（3）Taobao Tair 分布式缓存中间件

Taobao Tair 分布式缓存中间件，是一个分布式的 Key/Value 存储引擎，基于 hash 算法，将所有 Key 对应的 Value 分配到不同的数据节点中，最终提供负载均衡的分布式缓存服务。并通过多备份存储，在节点故障时进行数据迁移，保证了故障下的弱一致性，提供了安全策略。

（4）Taobao Diamond 分布式配置管理中间件

Taobao Diamond 分布式配置管理中间件，以 Mysql 为中心的，在分布式集群的服务器上，将数据以缓存和本地磁盘文件的形式存储，通过长轮询的方式推送数据的实时变化，实现配置的动态更新。

2.4.3 无线组件

（1）Agoo 无线消息通道

Agoo (Alibaba Cloud Mobile Push) 是一个移动消息推送服务组件。它具有向移动 APP 推送消息的功能，能够实现高效、精确、实时的消息推送，在客户端以通知的方式呈现消息。

(2) ACCS 无线数据通道

ACCS (Alibaba Cloud Channel Service) 是阿里的无线通道服务, 能够在无线客户端实现稳定的网络数据链路, 并具有弱网下的优化、抗抖动、加密传输等特征, 实现高效的数据传输, 具体应用如数据同步场景。

2.5 其他技术

2.5.1 iBatis

iBatis 框架是一个轻量级的数据映射框架, 它提供了持久化 API, 能够方便实现已有的数据库模式到应用程序的数据库持久层的转化。iBatis 框架通过 XML 编码的 SQL 映射文件, 实现了 SQL 模板的静态化和执行结果到 Java 对象的转化^[39]。使用 iBatis 的架构层次如图 2-5。

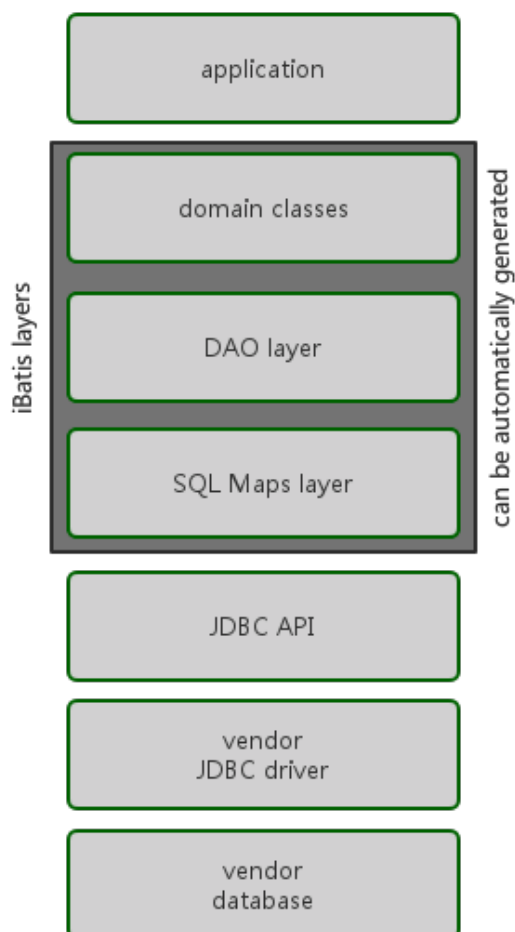


图 2-5 iBatis 应用的架构层次

除了 iBatis, 在 Java EE 领域, 应用的比较多的还有 Hibernate。Hibernate 是严格以对象为中心的持久化方式, 而 iBatis 则是围绕传统数据库模式, 进行

设计和实现。

在数据库的数据模式完整、确定，且没有很高吞吐量要求的场景下，Hibernate 能较好地组织和工作在应用架构中。Hibernate 提供了很好的通过 Java 对象操作数据库的机制，但是也给程序带来较高的复杂度，尤其是面对传统数据库模型，使用 Hibernate 编写的代码往往包含了复杂的 SQL 查询代码。

在传统数据库模型的应用场景，且查询使用占比很大的情况下，iBatis 更加简洁，给程序结构增加的复杂度也更少，从而更具优势。

2.5.2 Mockito

Mockito 框架是一个 mocking（一种支持单元测试的方法）框架，它通过代理、方法拦截等方式，将被测试类和它的依赖分离开来，提供对单元测试的支持。而且 Mockito 还提供了更简洁的 API 支持，测试结果可读性高，对程序错误的验证过程也很简洁^[40]。

Mockito 是一个开源项目，而且已经得到了越来越广泛的应用，允许单元测试类的编写，减少了额外的对于结果预期和解耦系统依赖的工作量。而且能和 junit 配合，使系统满足单元测试的验证标准。

2.6 本章小结

本章在分析了微服务架构的优势，并且将 Spring 框架和其他主流服务框架进行比较后，选择了 Spring 作为本课题的开发框架。同时选用了定时任务队列，和其他现有的中间件，以及其他组件，后者主要是以接口的方式提供功能。iBatis 在数据库方面提供了优秀的持久化支持，而 Mockito 在单元测试方面提供了良好的支持。

微服务架构因其扩展性高、灵活性强、适用于分布式云平台，能适应变化的多开发团队合作，近年逐渐被广泛应用于企业级的 web 服务架构搭建，具有很大的优势和前景。Spring 框架、其他中间件和组件的结合使用，使得整个寄件快递单处理系统的功能能够成熟、稳定地得到实现。

第三章 寄件快递单处理系统的需求分析

3.1 用户角色分析

寄件订单处理系统包括寄件人、快递员和后台运营三种用户，基于三个角色，针对系统设计的用例图如图 3-1、图 3-2 和图 3-3。

3.1.1 寄件人

系统最主要的用户是寄件人，也就是发起寄件流程的用户，主要是普通消费者，是我们的需求场景中最重要用户。

针对寄件人，我们用图 3-1 所示的用例图，表示寄件人的用户需求。

首先，寄件人使用手机应用端，一般是 Android 系统或者 iOS 系统的智能手机应用，应用具有访问和上传用户地理位置的权限。通过统一的方式登录后，开始使用 APP 的功能，当用户选择进入寄件模块时，手机应用端会从服务端系统获取用户在之前的寄件历史记录，所以我们的寄件快递单处理系统要满足寄件人的查询历史寄件记录的需求。

同时，寄件用户在手机 APP 端的寄件模块的寄件流程入口时，手机应用端会上传当前用户的地理位置信息，包括用户所在的经纬度。系统根据用户上传的经纬度，查询、解析并返回用户的具体地理位置，包括所在的省市区街道等，减少了用户的填写工作，同时给系统提供了更多的真实位置信息。

当用户创建寄件快递单，发起订单流程时，需要填入寄件所必须的信息，包括寄件人姓名、寄件人联系方式、收件人姓名、收件人联系方式、寄出物品的体积重量和种类等信息。此外，寄件用户还能对服务的类型进行选择，这要求系统提供寄件服务内容查询的功能。

寄件人在下单后，系统收到用户提交的关于寄件订单的各种信息后，对所在位置的服务商或快递员的寄件服务状态进行校验，满足校验要求后返回订单创建成功的结果。

寄件人使用的手机客户端得到订单创建成功结果后，会进入等待接单状态，寄件人在等待订单被系统处理以及快递员接单的整个过程中，能实时查看寄件订单的进度情况。寄件人也能取消订单，重新填写寄件信息，或者在订单被接单之前退出寄件订单处理流程。

寄件人不取消订单，寄件订单会被推送给附近的快递员，包括小件员或者上门寄件服务提供者（本文统一称为快递员角色），快递员能够接单。寄件人在接单过程中，能够查看寄件订单的创建时间、寄件订单详情、是否被接单、接单快递员的信息等。

寄件人在线下等待快递员上门的过程中，也可以使用本系统提供的发送消息的功能，和接单的快递员进行联系，确定上门时间和商量额外的寄件相关信息。

寄件人在快递员上门时，向快递员出示寄件人系统在寄件人手机应用上显示的取件码，系统会根据寄件人和快递员的当前地理位置的距离，以及快递员输入的取件码，验证取件操作成功。包裹打包相关的线下活动，可以在上门取件时进行，也可以在快递员返回网点后再进行装箱打包，在这个过程中，寄件人根据系统给出的价格标准，以及快递员提供的包裹实际重量和体积等，进行寄件快递单的支付，支付的内容也可能还包括包装费用等。

寄件人支付完成，等待快递员在网点发出包裹，并上传物流运单号。系统还要满足寄件人对包裹的物流信息查询的功能，在快递物流信息查询模块，寄件人能继续跟进寄出包裹的物流信息，也可以回到寄件模块查看历史寄件记录。

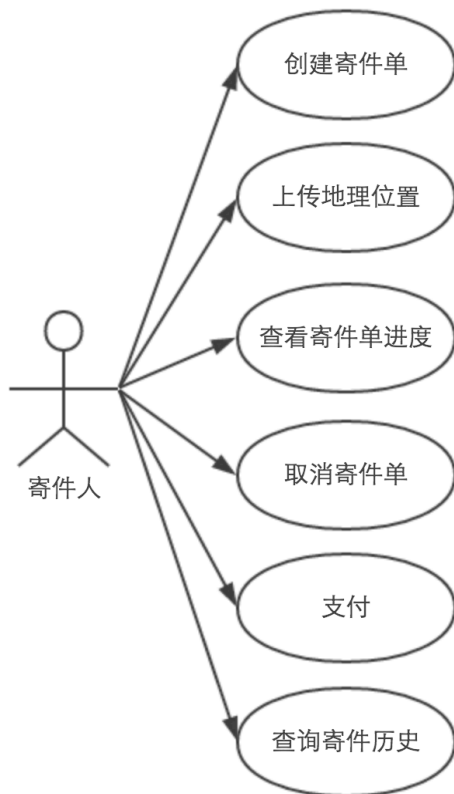


图 3-1 寄件人用例图

3.1.2 快递员

在寄件这个应用场景中，快递员是第二个出现的角色，也是承接寄件服务的主要角色，负责了寄件订单从被接单，到寄件快递单处理结束的整个流程。

在寄件用户首先在寄件系统创建寄件订单后，系统确认订单创建成功，并将寄件快递单指派给快递员，快递员接受指派，并进入到寄件快递单在本系统的处理流程中。

正常状态下，快递员在外部系统会周期性上传工作状态，包括工作时间、服务范围等，同时维护自身提供的寄件服务信息，本系统将这些信息返回给寄件人的手机应用端。同时，利用本系统的服务，快递员能上传包括地理位置、服务标识等信息，这样系统就能够基于地址位置等，将寄件订单指派给附近的快递员，让快递员选择继续做接单操作。

系统将寄件订单下发给一个或多个快递员，快递员在这种情况下，收到推送的分派任务，可以在快递员的手机应用端，查看派送的订单信息，确认满足接单条件，比如服务范围和寄件类型，继续选择接受或拒绝派单。

快递员在接受系统指派的某个订单后，需要在规定的时间内上门取件，系统也会进行取件前的计时。快递员上门取件，需要在取件时，索取寄件人的取件码，同时上传和更新自己的地理位置，这样让系统验证快递员的取件行为真实发生。

快递员在上门取件时，对寄件人的包裹进行打包，或者带回快递包裹至快递网点再进行打包和寄送往订单上填写的目的地。在这个过程中，快递员可以要求寄件人进行支付操作，系统负责记录支付行为，只有当支付成功，并且寄出的快递包裹产生的物流运单成功上传后，寄件快递单的处理流程才算完成。

如果快递员在系统规定时间没有上门取件，系统会记录快递员的注册信息和超时取件这个事件，进行后续的业务逻辑处理，这是保证服务时效性的一个方面。

另外，如果在系统承诺的规定的时间内，派送出去的寄件订单，没有快递员做接单操作，这种情况下系统会在业务层进行处理。系统将没有被接单的快递订单重新发送给外部的预约寄件系统，这个预约寄件系统属于合作服务商的子系统，同时本系统会将订单转给外部的结算子系统，对寄件人进行可能的补偿或补偿，比如优惠券等，这是保证服务的时效性的另一个方面。

对于发送到外部系统的寄件快递单，是由外部的合作服务商直接提供寄件服务，主要通过电话联系等形式，在线下进行服务，本寄件快递单处理系统只负责向该系统发出预约信息相关的通知。

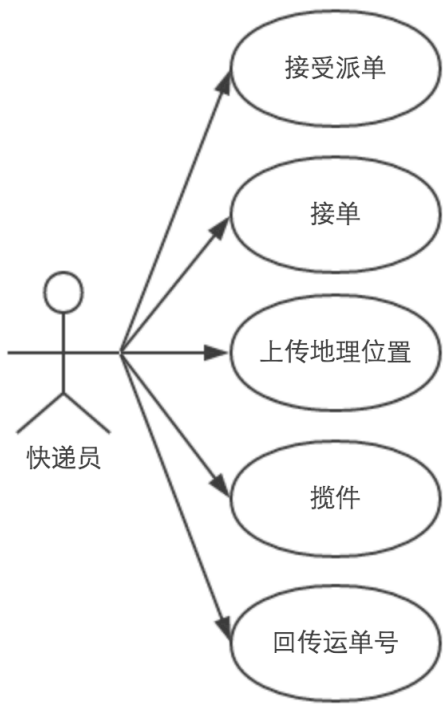


图 3-2 快递员用例图

3.1.3 运营人员

本系统面向的快递行业的末端寄件业务，服务的对象是普通消费者（寄件人），方式主要是消费者线上创建寄件快递单，快递员上门取件，之后在网点寄出。这种形式属于新拓展的功能领域，具有运营方面的需求。

一方面快递行业的业务部门需要面向消费者，推送关于这种形式的寄件业务的活动信息，比如优惠活动、服务上下线等信息。另一方面，有些对于手机端 APP 应用的管理，需要在运营后台接入管理，比如对手机应用的补丁包上传操作，往往和运营维护信息的发出，是同时进行的。

本系统是一个直接依托于公司开发的互联网移动端产品的寄件快递单处理系统，也要满足上述后台运营活动人员对相关的运营活动和线上配置的操作的需求。

目前，作为一个新拓展的快递末端业务，寄件快递单的处理在整个系统建立的初期，需要执行的运营任务相对简单。根据目前的需要，结合业务开展、订单处理和系统运行中可能出现的问题，拟实现的运营方面的功能，主要包括：

- （1）向目标用户或所有用户推送运营内容相关的消息，包括寄件服务变更消息、运营活动开展的消息、优惠消息等 APP 消息。

(2) 在开发人员协助下，上传手机端 APP 应用的补丁包（Hotpatch 包），和补丁包相关的维护信息，对发布的 APP 存在的缺陷进行修补，同时根据需要在手机应用端做出提醒。

(3) 对手机应用 APP 端的首页的展示栏界面（Banner 页面），包括展示的内容和相关配置，如图片、链接、有效时间范围等内容，做线上配置和操作。

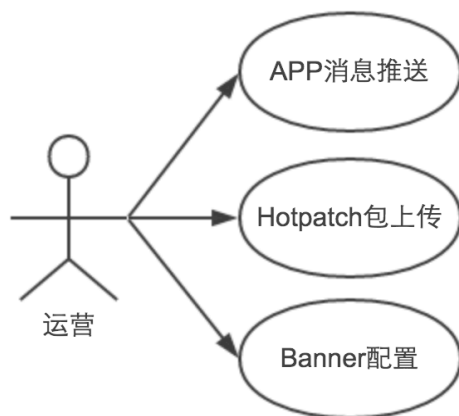


图 3-3 运营用例图

3.2 功能性需求分析

本课题拟实现的寄件快递单处理系统，包含了一个完整的订单系统所具有的核心功能，包括订单的创建、确认、分派、支付和发运。在支付部分和具体的发运环节，针对快递行业的寄件服务特点，增加了回传运单号、物流跟踪信息查看的内容，并将支付活动和取件发运活动同时进行，增加整个流程的灵活性。

如图 3-4 的活动图所示，整个系统要处理的核心业务流程描述如下：

(1) 寄件用户下单发起寄件订单流程，系统做判断后返回下单成功，寄件快递单生成。

(2) 系统判断创建的快递单是否满足派单条件，如果不满足派单条件，结束整个处理流程，满足派单条件时，内部的派单系统进行派单。对应地，快递员收到派单信息后，可以做接单操作。

(3) 快递员接单之后，寄件人进行支付操作和快递员负责上门取件操作，回传运单号的操作，是同时进行的，在这两个子流程都结束之后，包裹在快递网点被寄出，整个寄件快递单的处理过程最终完成。

此外核心流程图没有体现出的，业务异常或系统异常的处理，系统也要考虑进行对应的处理。

核心业务流程用图 3-4 所示的活动图表示。

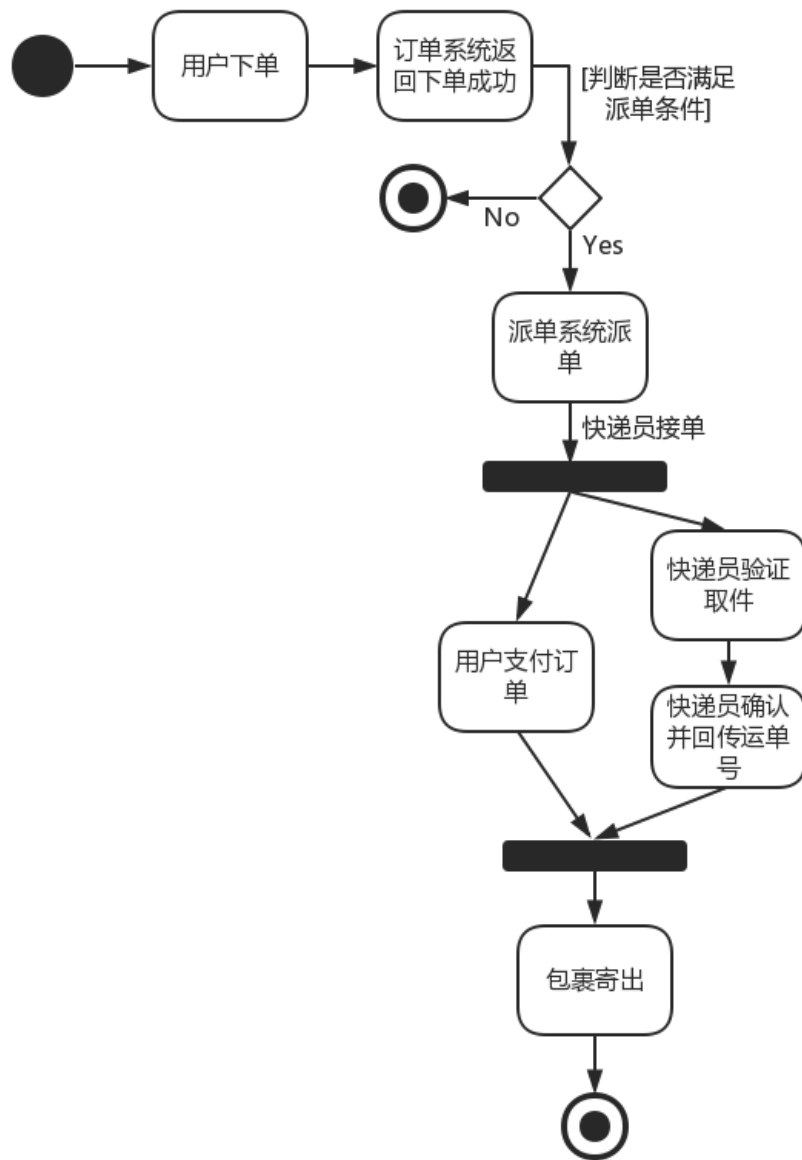


图 3-4 核心业务流程图

围绕核心业务流程，考虑寄件快递单处理场景的实际情况，仔细分析本课程设计的寄件快递单处理系统，可以得到拟实现的功能如下：

- (1) 寄件人通过移动应用提交寄件信息，系统提供校验、记录、生成订单的功能。
- (2) 基于地理位置，获取附近小件员信息和其他服务提供者信息的功能。
- (3) 根据位置信息和服务类型，将订单推送给附近快递员的功能。

(4) 快递员抢单，揽件和回传运单号的功能。

(5) 寄件人查看寄件订单的服务计时和状态变化的功能，查看历史寄件单的功能，通过公司物流平台服务跟踪物流信息的功能。

(6) 使用结算平台服务，寄件人进行支付，快递员查看支付状态、结算结果的功能。

(7) 运营人员使用运营后台，发布寄件服务相关消息，优惠活动消息的功能。

(8) 运营人员使用运营后台，进行 APP 相关配置上传，APP 补丁包上传等功能。

3.3 非功能性需求分析

本系统处理的是快递行业在末端配送上的寄件业务，并计划在未来基于寄件业务，推出同城配送等其他末端业务，对系统进行横向拓展性提出要求。

同时，整个系统的开发，分别在若干个小组中进行，每个子系统自身能简单、规范地暴露自身的服务，子系统能方便的进行调用和联系，实现松散的耦合。

目前接入的客户端是安卓或 iOS 手机客户端，装机用户 400 多万，每日活跃用户接近 20 万，在运营活动开展期间能达到 40 万。对于寄件快递单处理系统，从服务端考虑，日均 PV（页面浏览量）至少要求达到 100 万，峰值要求达到约 5000 次每分钟。

3.4 本章小结

本章分析了寄件快递单处理系统面对的三个用户角色：寄件人、快递员、运营人员。寄件人和快递员通过快递单的处理发生联系，并具有各自需要实现的功能需求。运营人员面向整个使用群体，具有必需的运营方面的需求。本章还借助整个寄件快递单处理的流程图，对系统的功能点进行了详细分析，对系统的非功能需求，结合具体数据做了分析。

第四章 寄件快递单处理系统的设计

4.1 系统总体设计

4.1.1 子系统划分

寄件快递单处理系统划分为：寄件订单子系统，寄件派单子系统，寄件运营后台子系统，结算子系统，反作弊子系统等，本课题研究实现寄件订单子系统，寄件派单子系统，寄件运营后台子系统。

4.1.2 系统总体架构

整体系统的架构图如图 4-1 所示。

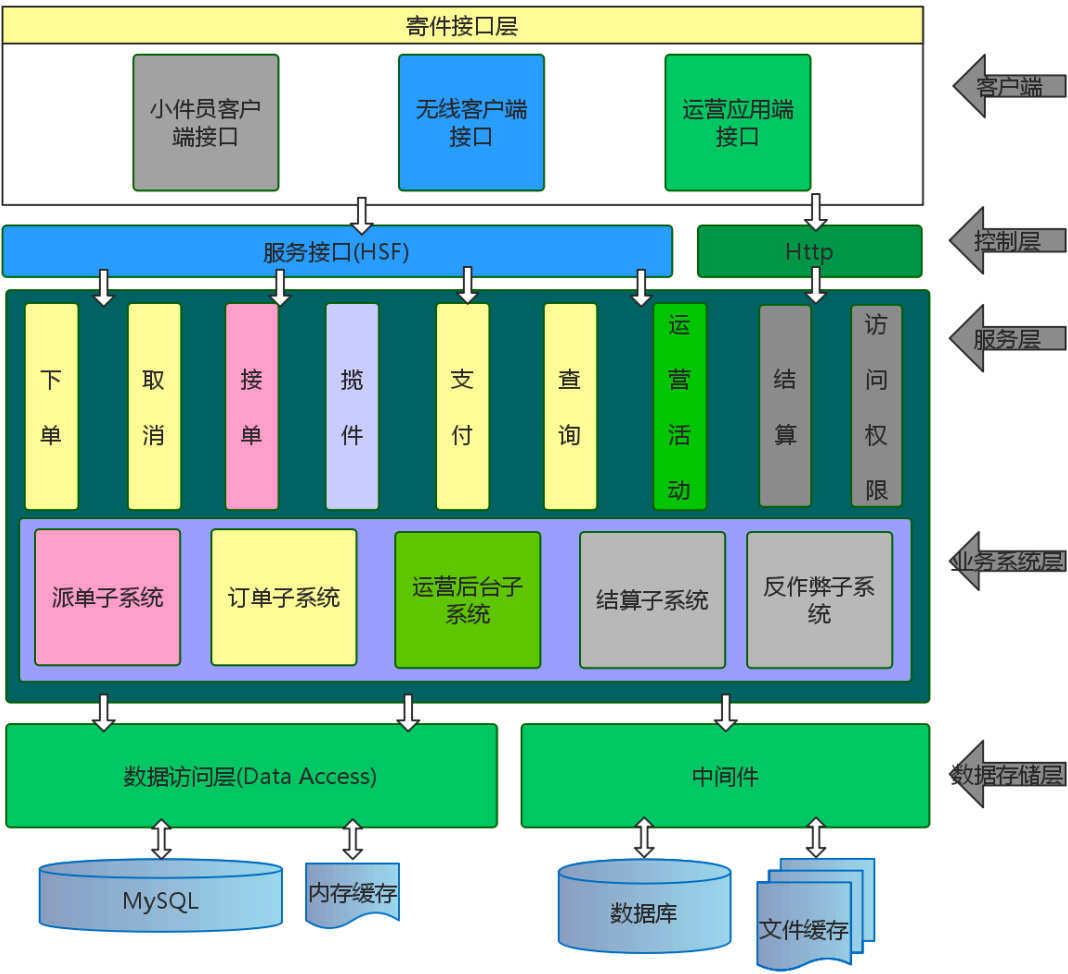


图 4-1 整体系统架构图

首先，系统主要对接的是移动应用客户端（手机 APP），面向普通消费者即寄件人，以及快递员使用的手机端应用，此外系统还要对接内部运营人员使用的运营后台前端 web 应用，因此系统的在上层划分出寄件接口，对接小件员客户端（快递员用户使用）接口，无线客户端（普通消费者使用）接口和运营应用端接口（后台运营人员使用）。

寄件接口层所使用的寄件相关的服务功能，被规范为统一的 HSF 服务的形式的，进行查找和使用。此外，也有以 Http 形式的外部服务功能提供的功能给寄件接口层使用。HSF 服务接口和 Http 接口抽象出来的构成了系统的控制层，对接口的规范进行统一管理，包括超时时间、接口命名、参数名称等都进行规范化管理。

控制层的下面是服务层，服务层是整个系统使用的微服务组成的层次，属于整个系统的核心，服务层的上层完成接口的规范和连接，下层完成功能逻辑和数据存储。服务层遵循“尽可能单一”的原则，将系统的功能拆分为一个个微服务，所有这些微服务共同定义了整个系统功能设计。

具体实现的微服务，核心的包括：下单、取消、接单、揽件、支付、查询订单、运营活动、结算、访问权限等，这些功能都是以微服务的形式对上层模块提供服务的。系统的寄件接口层，都使用这些微服务，以及它们的组合，连接客户端，提供接口功能。

业务系统层，是对整个系统的各子系统的抽象表示，这些子系统也是按照微服务的思想，根据业务功能进行划分的。在我们的寄件快递单处理系统中，划分出来的子系统包括了核心的订单子系统、派单子系统、运营后台子系统和其他的子系统。本文主要探讨和研究前三个主要的子系统设计实现和整体的系统架构设计。这些子系统采用独立的技术框架实现，分别独立地进行部署，可以独立地选择数据存储方式和存储实现。子系统之间通过微服务的方式实现联系，实现了子系统之间的松散耦合，便于系统的横向拓展。

整个系统的数据存储采用 MySQL 就行持久化存储，同时通过内存缓存加快数据访问过程，以及实现数据的临时存储。数据访问层，是对系统的存储数据的结构进行的抽象，比如 Java 语言下，以对象的形式表示要存取的数据。

此外，为了达到完整地提供寄件快递单处理系统的业务功能，系统还需要依赖外部的中间件，达到拓展系统能力，支持上层服务功能的目的。比如 Taobao Notify 消息中间件，实现系统之间的消息的持久化存储和投递；Taobao Tair 分布式缓存中间件，实现分布式缓存；Taobao Diamond 分布式配置管理中间件，实现分布式配置的更新和查询。这些中间件平台包括对底层的数据存储过程的

抽象，以及对网络通信方式的封装，对上层提供功能服务接口。

下面对寄件快递单处理系统的整体系统架构，划分出的各层概述如下：

（1）寄件接口层：包括小件员客户端接口、无线客户端接口、运营应用客户端接口，主要负责客户端应用的接入；

（2）控制层：主要使用 HSF 框架的方式对服务进行规范管理和通信联系，部分外部接口使用基于 Http 的方式进行通信和联系，系统主要以这两种方式进行服务的访问控制，HSF 方式是实现微服务管理和功能的基础；

（3）服务层：将寄件功能抽象成各个微服务，以微服务的方式独立实现和提供各种寄件流程必须的服务，包括下单、取消、接单、揽件、支付、查询订单、运营活动、结算、访问权限等；

（4）业务系统层：通过各个子系统分别实现对应的功能模块，支撑上层的功能服务，划分出来的子系统包括订单子系统、派单子系统、运营后台子系统等，子系统之间相互独立实现；

（5）数据层：提供对于数据的非持久化缓存、持久化存储，以及对于静态配置等数据访问的功能。数据层还包括使用的中间件，中间件负责提供分布式数据缓存等。

整个寄件快递单处理系统按照微服务的思想，拆分为多个独立提供微服务的子系统，各子系统之间通过 HSF 方式进行通信，并且通过消息中间件 Notify 等方式实现子系统之间的弱依赖。

4.1.3 系统的开发框架

寄件快递单处理系统被拆分为多个系统，本文主要探讨的是寄件订单子系统、寄件派单子系统、寄件运营后台子系统这三个核心的子系统。子系统可以继续划分出对应的内部层次和模块。

（1）寄件订单子系统和寄件派单子系统

按照微服务的思想，各子系统独立实现自身功能，可以采用不同的开发语言和技术框架。本课题拟实现的寄件订单子系统和派单子系统，属于核心子系统，负责处理寄件订单和寄件派单流程，按照系统整体的架构设计，进行层次划分，这两个子系统实现的功能和操作的数据对象十分接近，采用同一套技术开发框架。

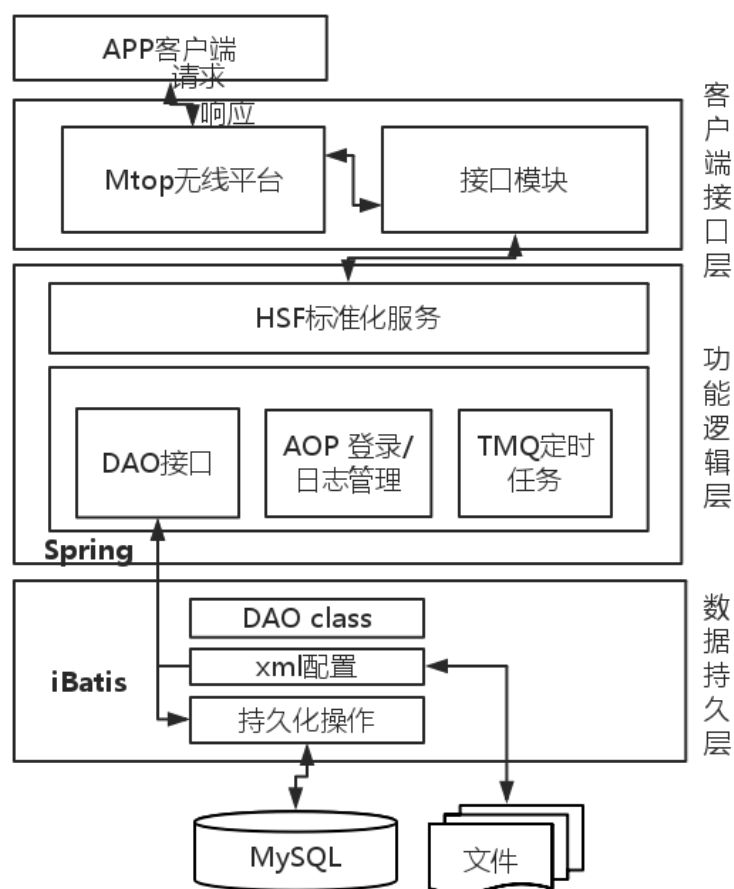


图 4-2 寄件订单子系统和派单子系统的技术框架

如图 4-2 所示，寄件订单和派单两个子系统的技术框架都是在 Spring 基础上设计的。整体设计上，划分为多个层次。从上至下，依次划分为客户端接口层、功能逻辑层、数据持久层。

其中，客户端接口层对外连接 APP 客户端，手机 APP 应用向客户端接口层发送请求，客户端接口层返回响应。手机客户端所使用的无线网络通信服务，是由 Mtop 无线平台提供的，本系统在该层负责实现对接的接口模块，对外提供手机客户端应用使用的接口，对内依赖底层模块提供的标准化服务。

功能逻辑层，负责实现子系统的功能，这些功能都是通过 HSF 的形式标准化为微服务，为上层模块提供接口来使用，同时通过对微服务管理，这些微服务的功能也能被其他的外部系统所使用。

功能逻辑层，主要基于三个子模块：包含了数据持久化存储操作的数据访问对象 DAO(Data Access Object)接口，使用面向切面编程 AOP(Aspect Oriented Programming)实现的登录和日志管理功能，通过一种定时任务队列中间件 TMQ(Taobao Management Queue)实现定时任务管理的功能。

功能逻辑层的 DAO 接口是基于 iBatis 封装的抽象数据存储过程，一般是通

过配置 xml 形式的资源文件，将 DAO 对象映射到数据库实体表，并基于 iBatis 提供的基本操作，封装出 DAO 接口作为数据存储过程。

功能逻辑层的用户登录验证和登录状态，以及服务端日志的打印等，需要依赖于 Spring AOP，一方面减少了开发量，另一方面实现了这些通用功能和其他寄件功能逻辑的解耦合。

功能逻辑层的 TMQ 定时任务队列，可以为寄件订单子系统和寄件派单子系统的超时无接单、接单后取件超时等场景，提供任务超时方面的功能支持。

数据持久层，依赖于 iBatis 将抽象的 Java 对象，映射到数据库表的实体，进行持久化操作。DAO 类的定义，可以映射某数据库表的所有列属性，也可以只映射一数据库表的一部分列属性。DAO 接口的操作，可以基于任意列属性增加 SQL 条件。xml 配置文件，静态定义了系统所需的基本的 SQL 操作，这些操作都是基于 DAO 对象定义的，并且被上层的 DAO 接口的实现类直接加载和使用，灵活地实现数据的持久化操作。

底层的数据库存储选用 MySQL，静态配置放在配置文件当中。

（2）寄件运营后台子系统

按照微服务思想，子系统的技术框架是相互独立，数据存储和应用部署都是独立的，在设计时也可以单独设计。我们的寄件运营后台子系统，对接的是应用是 web 前端应用，在浏览器中展示，实现的功能是后台管理功能，提供给内部的后台运营人员使用。采用的技术框架设计如图 4-3 所示。

表现层和客户端接口层对接 web 应用端。具体包括了两个子模块：AngularJS 框架和 HSF JS 客户端，AngularJS 框架负责 web 应用必需的模型视图控制（Model、View、Controller）等内容，HSF JS 客户端负责实现前端框架和后端 HSF 接口的联系。

功能逻辑层基于 NodeJS，NodeJS 在和 AngularJS 框架的配合使用上更有优势，这一层主要负责实现具体的运营所需的功能，并规范为 HSF 标准化服务。

运营后台子系统目前在数据存储上的需求很少，主要在前端浏览器中就可以完成，持久化存储目前设计成为依赖于其他的子系统进行，另外在功能的实现上，还要依赖于外部的 HSF 标准化服务，比如 Banner 首页配置，就需要依赖于 Taobao Diamond 分布式配置管理服务。

运营后台子系统的技术框架相对简单，但是基于前端框架 AngularJS 和后端 NodeJS 框架平台，可以灵活方便地进行拓展。

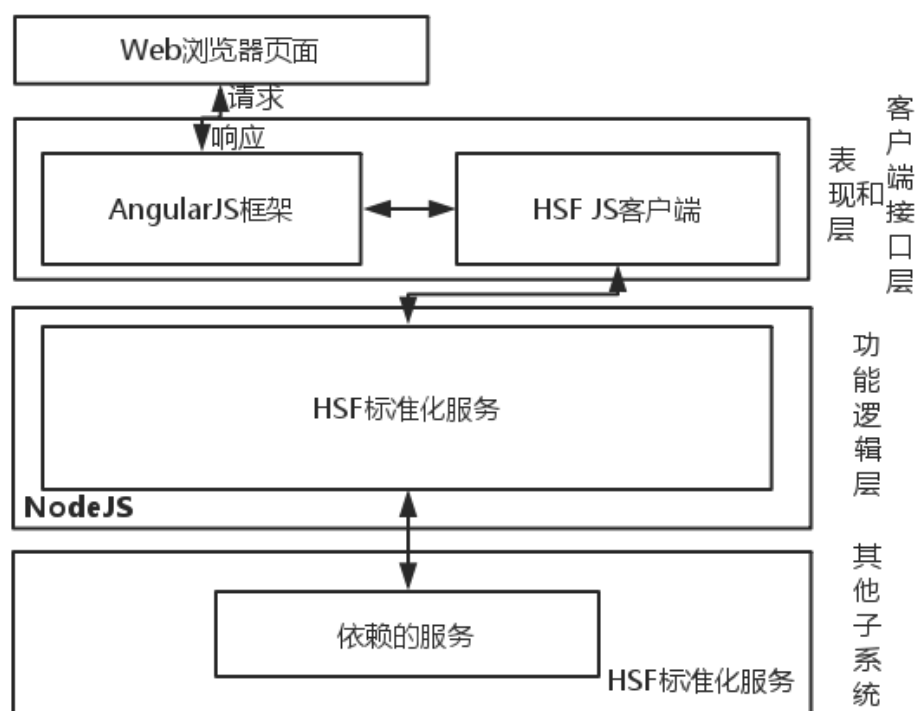


图 4-3 运营后台子系统的技术框架

运营后台子系统也是通过 HSF 的方式对外提供标准化微服务，并且依赖其他子系统提供的微服务，表现层使用 AngularJS 框架实现 Web 界面的展示，数据控制等，客户端接口层使用 HSF 的 JS 客户端访问后端的 HSF 标准化服务，这些服务是在 NodeJS 上提供的，整个技术栈不同于寄件订单子系统和派单子系统的 Java 技术栈，是基于 JavaScript 语言的框架结构。

运营后台子系统依赖其他的子系统平台实现持久化存储，本身是一个相对简单的子系统，是整个快递单处理系统的一个拓展部分。

4.1.4 微服务的管理方案

基于 HSF，每个 APP Server（子系统的服务器）都对外提供自己的微服务，微服务是通过服务功能或操作的资源对象进行划分的，每个微服务的功能尽可能的单一和独立，各个子系统都对外提供微服务，也部分依赖于其他子系统提供的微服务，实现自身功能。

对于一个完整的寄件订单、派单和运营后台系统，围绕订单处理流程，实现功能点，进行拓展后，得到的微服务数量在一百余个。每个微服务都有自己的服务名称、参数列表、返回值、超时限制等。

对微服务进行管理，一方面方便了开发人员，减少在查找，理解，引用微

服务上工作量，另一方面通过 HSF，对这些微服务进行标准化定义，并基于底层通信实现，达到规范和统一地使用微服务功能的目的。

微服务管理上，本文的主要工作在：

（1）标准化定义

每个子系统都是以微服务的形式对外提供功能，微服务定义的标准化，便于系统开发人员的开发，也通过统一的调用方式的完善，减少错误率，增加系统的成熟度和稳定性。

对于微服务的标准化，参考了目前主流的接口定义语言 IDL（Interface Definition Language），比如 Apache Thrift 等。定义的要求上，最小化定义包括服务的名称、实现类、版本号，拓展定义的内容包括重试次数、超时时间（都有默认参数）等其他配置。在后续的标准化定义的实现上，可以根据不同的开发框架和开发语言，结合使用方式，做具体的实现。

（2）服务发现和共享

为了方便开发人员查找、引用微服务，需要对微服务的元信息（meta data），如包名称、服务名称、版本号、接口信息、依赖等，进行统一的存储，并提供检索查找功能。在使用上，还要能够支持下载依赖文件，自动下载依赖，自动构建编译打包等，这些内容已经成为业界的标准，并提供了成熟的工具。

本文采用开源的 Sonatype Nexus 和 Apache Maven 管理工具。

Sonatype Nexus 是一个组织、存储、分发软件组件的仓库管理工具，提供了对代码仓库如 Maven 仓库的代理，还能对工程的部署目标进行存储和索引。Sonatype Nexus 基于集中式管理方案，能保证将不同版本的软件组件分发到用户端，并能保证原信息的及时更新，在流行的开发环境如 Eclipse 等还提供了完整的插件支持。

在本系统的设计中，对应的软件组件就是微服务的元信息，本文通过 Sonatype Nexus 将微服务服务组件的元信息，存储在远程仓库，实现方便的查找检索、引用下载、构建包下载，简化开发过程，统一了微服务元信息管理的方式。

Apache Maven 提供了对服务进行依赖管理，构建，编译，部署的自动化方法。Apache Maven Repository（Maven 仓库）提供了对发布的软件包，和软件包元数据的集中式存储，还提供了在构建工程过程中必需的依赖寻找和下载等功能。本系统依赖的大部分开源组件，都存储在 Apache Maven Repository 中。

在 Nexus 和 Maven 两个工具的使用上，开发人员通过 Nexus 检索和发现开发需要使用的程序的 Artifact（工件）信息，同时每个子系统将自身程序打包后

存储在 Maven Repository（Maven 仓库）中，这种使用方式的图像化表示如图 4-4 所示。

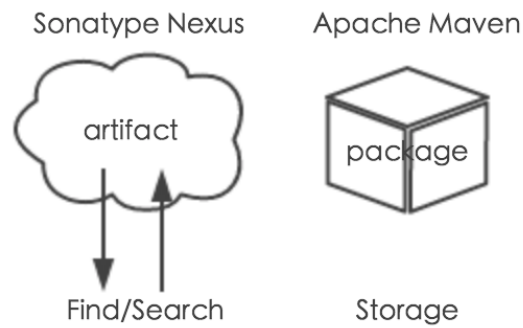


图 4-4 Nexus 和 Maven 工具的使用

4.2 寄件订单子系统的设计

本系统的寄件订单子系统的模块划分如图 4-5 所示。

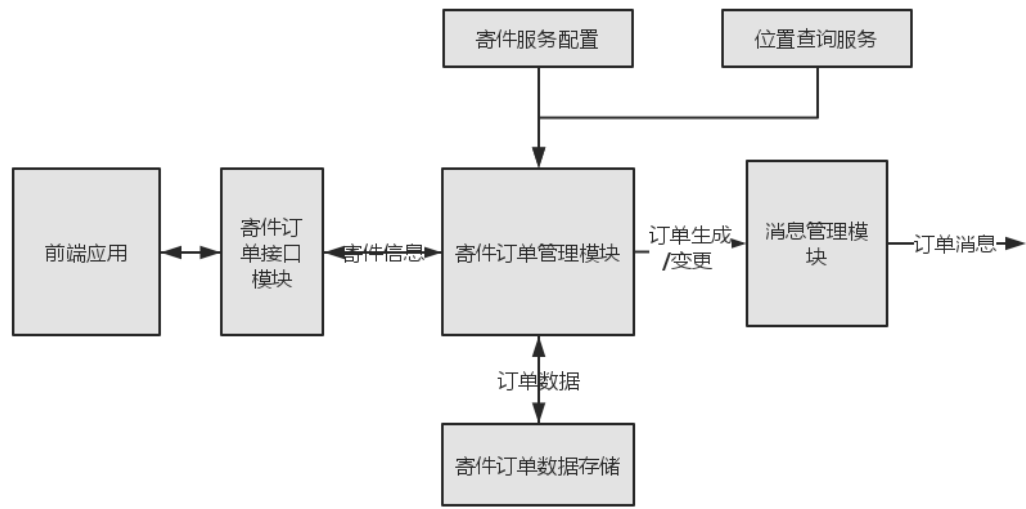


图 4-5 寄件订单子系统

对寄件订单子系统的模块划分，主要是以系统模块的功能区别作为划分标准。将寄件订单系统，划分为：寄件订单接口模块、寄件服务配置模块、位置查询服务模块、消息管理模块、寄件订单数据存储模块这几个主要的子模块。

其中，寄件订接口模块，是为了更好地接入客户端应用的模块，包括对客户端的请求进行解析和转换，对返回给客户端的数据进行重新封装和转换，以适应客户端的通信数据格式需求。

寄件订单数据存储模块，负责对寄件订单的持久化存储，寄件订单子系统

有独立的数据库存储，存储的数据模型是寄件订单表。

寄件服务配置模块，负责将寄件服务方信息，比如快递员或快递服务商的工作时间、服务范围、价目表等信息，这些信息通过配置服务提供给的，配置服务是基于 Taobao Diamond 分布式配置管理中间件实现的，这个配置模块需要对配置内容进行解析和转换，将得到的配置数据对象以配置服务的形式提供给其他模块使用。

位置查询服务模块，负责提供基于经纬度的具体位置信息的查询，在本系统中，主要是代理外部系统的位置服务，将查询外部服务得到的结果返回。

寄件订单管理模块，主要基于寄件订单的核心业务流程，根据订单的状态转换关系，提供对订单对象的各种操作，如创建订单、取消订单、接单、取件、支付、发运等操作。这些操作涉及订单数据的改变，都会通过寄件订单数据存储模块进行持久化存储。

消息管理模块，负责推送寄件订单系统要向其他子系统发出的消息，主要包括订单的创建，状态的变更等内容的消息。比如寄件派单子系统订阅寄件订单子系统的订单成功创建的消息，才能进行后续的派单操作。

在这些模块的基础上，寄件订单子系统接入客户端应用，并与其他子系统配合运行。接入的应用端具体应用目前包括：小件员 SDK，消费者（寄件人）APP。这些应用接入寄件订单接口模块，使用订单子系统的服务功能。

客户端应用获取得到的是寄件相关信息，如寄件服务信息，某个寄件订单信息，全量的寄件订单数据存储数据库表中。其他子系统接受到的推送数据，是简单的状态变化相关信息，包括原始状态，变更后状态，订单标识等。其他在模块之间传输的数据，还包括静态配置的快递服务方即寄件服务信息，从外部系统得到的地理位置查询结果数据。

在设计消息管理模块时，根据微服务的思想，同时让整个系统拆分出来的子系统，充分解耦，实现独立迭代开发。设计上目标是使寄件订单子系统，与业务流程的下游的派单子系统，通过消息中间件 Notify 联系。寄件订单系统，在订单创建、变更时，以消息生产者的身份，向订阅这个消息的派单系统发出对应的 Notify 消息。接收到消息的一方再使用寄件订单系统提供的微服务，继续进行寄件派单的逻辑处理，实现自身功能。

4.3 寄件派单子系统的设计

寄件派单子系统的模块划分如图 4-6 所示。

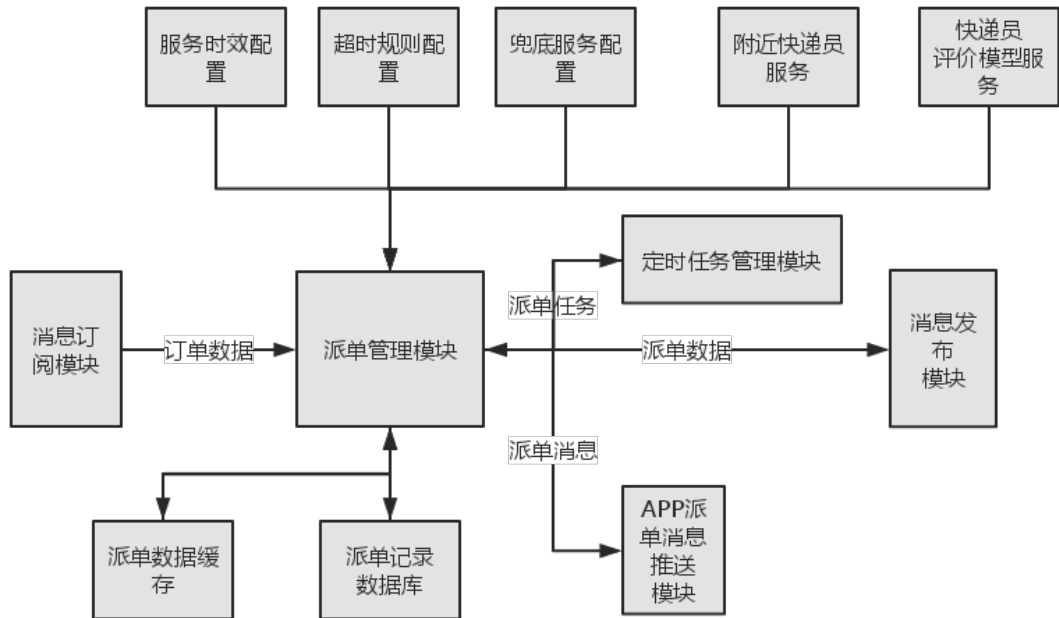


图 4-6 寄件派单子系统

寄件派单子系统，负责订阅寄件订单子系统的订单状态变更消息，同步本地派单数据，并进行派单相关的操作。

寄件派单子系统，依据模块的功能为划分标准，可以划分：服务实效配置、超时规则配置、兜底服务配置这些配置模块，快递员相关的附近快递员服务模块、快递员评价模型模块，基于 Notify 消息的消息订阅模块、消息发布模块，基于中间件和无线组件的定时任务模块、APP 派单消息推送模块，用于数据存储的派单数据缓存和派单记录数据库存储模块。

服务实效配置、超时规则配置、兜底服务配置这三个配置模块，是为寄件派单子系统提供配置查询功能的，这些配置变化的频率不高，主要是静态配置。实现方式可以依赖 Taobao Diamond 分布式配置管理中间件实现。

服务实效配置，主要包含合作服务商和快递员承诺的接单时间、上门取件时间等内容，超时规则配置，是对超时情况下的逻辑规则的进一步配置，如优惠券规则、惩罚规则等，具体的超时规则内容和逻辑处理主要在外部的补贴系统、小件员系统执行。兜底服务配置，兜底服务是指在没有快递员接单的情况下将寄件快递单转给的外部寄件服务，兜底服务配置包括是否允许使用兜底服务，兜底服务的唯一标识等。

附近快递员服务模块，依赖外部小件员系统提供的快递员位置信息，对距离进行计算，对寄件人和快递员所在街道区域进行匹配，提供附近快递员列表。

快递员评价模型模块，主要是代理外部的评价系统的服务内容，提供对快递员的评价信息查询服务。

服务端系统之间的消息通知，通过消息订阅模块获取和消息发布模块发出，订阅的消息如订单数据的变更消息，发送的消息如派单任务成果执行，订单被下发给快递员，以及无附近快递员，派单任务失败的消息。

定时任务管理模块，是对派单任务的支持，一方面将派单任务延迟，组成消息组合批量下发给客户端，另一方面对分派后的派单，检查是否超时，无人接单，进行处理。

派单数据缓存，拟设计成基于 Taobao Tair 分布式缓存中间件的数据缓存模块，提供派单记录的缓存功能，加快访问派单记录的速度。

派单记录数据的数据库存储模块，负责派单记录的持久化存储，每个配单任务的执行时间，执行内容，执行结果，都会记录到派单记录表中。

APP 派单消息推送模块，是利用无线消息组件，将派单消息下发给手机 APP 客户端的模块，主要负责封装派单消息对象，并调用组件进行下发操作。

派单管理模块，负责将这些子模块的功能联系起来，是一个 Manager（管理者）的角色，保证整个子系统按照合理的条件和顺序进行处理。

4.4 寄件运营后台子系统的设计

运营后台子系统的模块划分如图 4-7 所示。

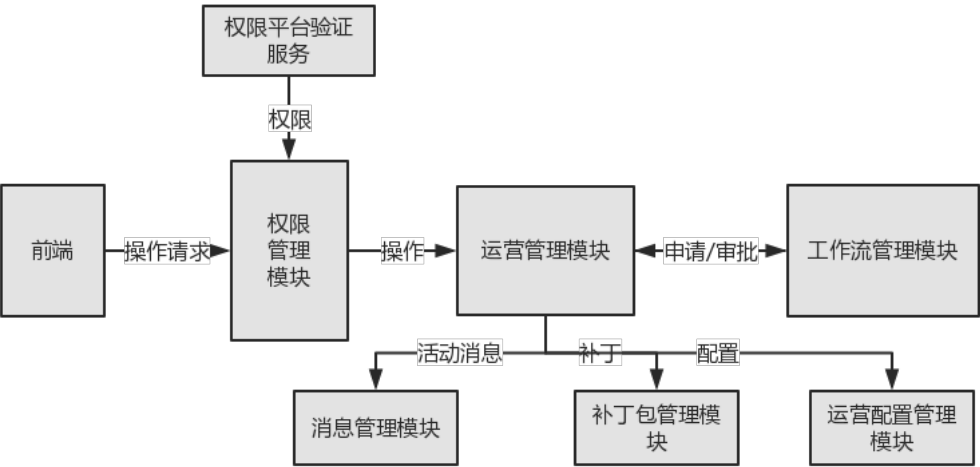


图 4-7 运营后台子系统

基于微服务思想，整个寄件快递单处理系统可以方便地拓展，满足前后端需求，运营子系统正是拓展出的子系统，要实现寄件服务消息，运营活动消息的推送，APP 补丁包的发布，运营配置的管理等功能。

寄件快递单处理系统的运营后台子系统，是为寄件运营活动提供功能的，具体划分出多个子模块。这些子模块包括：权限管理模块， workflow 管理模块，消息管理模块，补丁包管理模块，运营配置管理模块。

权限管理模块，提供权限访问控制的功能，运营后台是面向公司内部运营人员使用的，需要使用运营后台而没有权限的用户，应该跳转到外部系统的权限平台去申请权限，有权限访问的用户，也要通过对登录身份和权限角色组的验证。权限管理模块设计成代理，主要是代理外部系统的权限平台实现权限访问控制功能，比如对前端操作页面的访问控制、操作子菜单可见性的访问控制等。

workflow 管理模块，负责创建审批 workflow，在运营用户在运营后台进行操作时，对于涉及重要数据的变更操作，比如改变服务端线上配置，向全部移动应用端推送消息等，就需要创建审批 workflow，目的是获得审批人员审批，让操作真正生效。每一个运营操作对应的工作流的设计和实现，都是在外部 workflow 平台进行的，不属于本文探讨的内容。

消息管理模块，负责在运营后台子系统内，实现向手机应用端发送消息的功能，功能实现上可以是特定的部分用户，或全体用户，操作的执行会触发创建审核任务。

寄件运营后台，在使用过程上，用户对于运营后台的操作，经过权限验证，workflow 审批后，实现了面向移动端的消息推送，运营管理配置（包括移动端运营相关文案的配置）等功能。

4.5 数据库的设计

数据库设计的过程主要包含了：（1）确定要存储在数据库当中的数据；（2）确定不同数据之间的关系；（3）在数据关系的基础上建立数据模型的逻辑结构。

（1）确定要存储在数据库中的数据，也是对我们的寄件快递单处理系统进行应用分析和设计的一部分，我们在上一章节的需求分析中，已经对涉及的寄件订单信息、派单信息、运营操作信息、物流运单信息的应用场景做了介绍。

在本章的后面会对具体数据域进行定义。

（2）确定数据之间关系，包括数据集内部的依赖关系，以及数据集之间的

关联关系。本系统在之前的分析中已经建立数据实体，它们的关联关系相对简单，主要是寄件订单和寄件派单实体之间通过订单标识的一对一的关系（派单可能不存在），寄件订单和物流运单之间的多对多的关系，单独用一个关系模型来表示，运营后台子系统的操作记录用单独数据实体表示。

（3）建立数据之上的逻辑模型，是将数据组织成为合适的逻辑结构，而且能够映射到数据存储中。我们的系统的数据模型，属于对象数据库存储模型，数据存储对象能直接映射到系统所操作的对象，数据之间的关系可以用对象之间的引用（或标识）来表示，如本系统的寄件派单对象用订单号属性存储关联的寄件订单对象的标识。

4.5.1 数据库的 ER 图

在寄件业务场景中，进行数据库的数据模型的设计，首先需要考虑设计哪些数据模型。在本系统的数据模型设计时，是围绕寄件快递单处理流程，分析出现的数据实体，以及数据实体间的关系，进行分析设计的。

首先，寄件快递单在寄件订单的处理阶段，要对生成的订单信息进行存储。在寄件处理的后期，形成物流运单，要存储运单信息。还要考虑寄件快递单和运单之间的映射关系。在系统派单过程，派单的内容、时间、目标等，要存储在派单记录中，也要进行持久化存储。对于运营操作，需要存储操作记录，便于进行后续的历史记录查询等。

本系统涉及的寄件相关的订单、派单场景属于典型关系型数据库的设计场景。考虑到目前普遍使用的 MySQL 数据库系统，具有开源、跨平台、对各种开源存储引擎的友好支持等优点。以及本课题设计的外部部门的主要存储系统也使用 MySQL 作为数据库系统，使用同类型的数据库能进行无缝对接，方便进行表的联合查询等。因此本系统的数据存储的设计实现上采用 MySQL 作为数据库系统。

本系统的数据模型主要抽象为：寄件订单表、寄件的订单与运单的映射关系表、快递运单表、派单记录表这四个实体。对于寄件运营后台，构建运营后台操作记录表，独立于前面的数据模型。其他的数据模型，如用户信息表、位置信息表，则存储在外部数据库系统中。

主要的数据库实体，寄件订单表、寄件的订单与运单的映射关系表、快递运单表、派单记录表，作为整个寄件快递单处理系统的核心数据模型，它们之间的关系，用图 4-8 的 ER 图来表示。

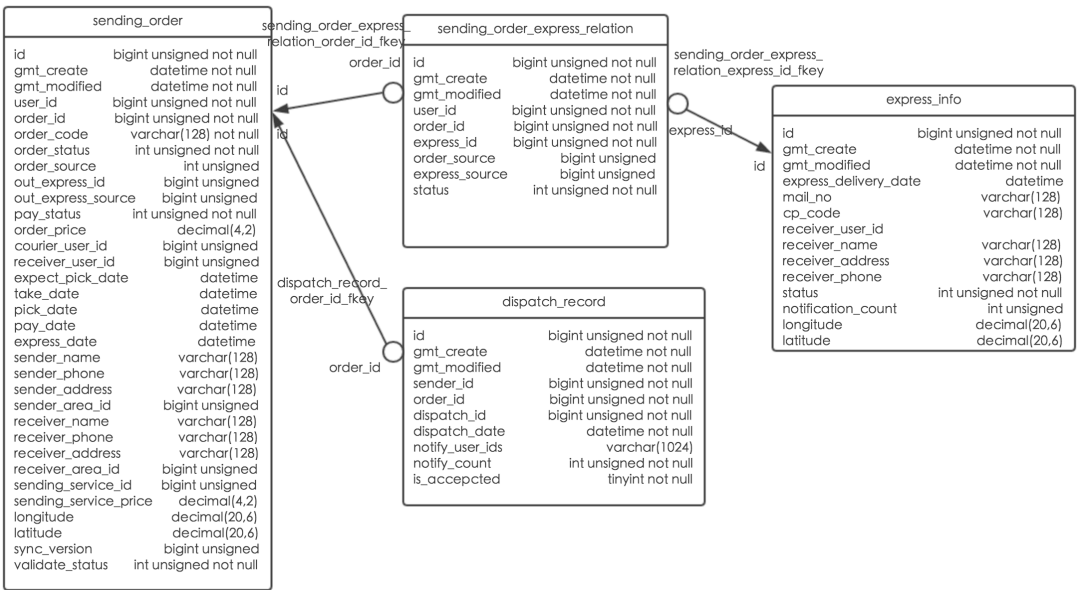


图 4-8 寄件快递单处理系统的 ER 图

4.5.2 数据库的表的设计

(1) 寄件订单表。负责存储寄件订单的创建和变更信息、地址经纬度、寄件收件地址、选择的服务类型等（注：其中的 sync_version 是为数据同步的乐观锁而设计的）。寄件订单表的结构见表 4-1。

表 4-1 寄件订单表

列	列名	数据类型	长度	允许为空	是否自增	备注
1	id	bigint unsigned			Y	数据库记录主键
2	gmt_create	datetime				记录创建时间
3	gmt_modified	datetime				记录修改时间
4	user_id	bigint unsigned				寄件用户 id
5	order_id	bigint unsigned				寄件订单 id
6	order_code	varchar	128			寄件订单号
7	order_status	int unsigned				寄件订单状态
8	order_source	int unsigned		Y		寄件订单来源
9	out_express_id	bigint unsigned		Y		外部快递单 id
10	out_express_source	bigint unsigned		Y		外部快递单来源
11	pay_status	int unsigned				支付状态
12	order_price	decimal(6,2)		Y		寄件订单价格
13	courier_user_id	bigint unsigned		Y		快递员用户 id

14	receiver_user_id	bigint unsigned		Y		收件人用户 id
15	expect_pick_date	datetime		Y		预计揽件时间
16	take_date	datetime		Y		抢单时间
17	pick_date	datetime		Y		揽件时间
18	pay_date	datetime		Y		支付时间
19	express_date	datetime		Y		外部快递物流时间
20	sender_name	varchar	128			寄件人姓名
21	sender_phone	varchar	128			寄件人手机号
22	sender_address	varchar	128			寄件人地址
23	sender_area_id	bigint unsigned				寄件人区域 id
24	receiver_name	varchar	128			收件人姓名
25	receiver_phone	varchar	128			收件人手机号
26	receiver_address	varchar	128			收件人地址
27	receiver_area_id	bigint unsigned				收件人区域 id
28	sendig_service_id	bigint unsigned				寄件服务 id
29	sending_service_price	decimal(6,2)				寄件服务价格
30	longitude	decimal(20,6)				寄件人位置经度
31	latitude	decimal(20,6)				寄件人位置纬度
32	sync_version	bigint unsigned				记录同步版本
33	validate_status	int unsigned		Y		寄件订单有效性状态

(2) 寄件订单与运单的映射关系表。存储寄件订单和运单的映射关系，见表 4-2。

表 4-2 寄件订单与运单的映射关系表

列	列名	数据类型	长度	允许为空	是否自增	备注
1	id	bigint unsigned			Y	数据库记录主键
2	gmt_create	datetime				记录创建时间
3	gmt_modified	datetime				记录修改时间
4	express_delivery_date	datetime		Y		快递派送时间
5	mail_no	varchar				运单号
6	cp_code	varchar				快递公司编码
7	receiver_user_id	bigint unsigned				收件人用户 id

8	receiver_name	varchar				收件人姓名
9	receiver_address	varchar				收件人地址
10	receiver_phone	varchar				收件人手机号
12	status	int unsigned				快递单状态
13	notification_count	int unsigned		Y		派送通知次数
14	longitude	decimal		Y		收件人位置经度
15	latitude	decimal		Y		收件人位置纬度

（3）运单表。存储揽件并发送后产生的物流运单数据见表 4-3。

表 4-3 运单表

列	列名	数据类型	长度	允许 为空	是否 自增	备注
1	id	bigint unsigned			Y	数据库记录主键
2	gmt_create	datetime				记录创建时间
3	gmt_modified	datetime				记录修改时间
4	express_delivery_date	datetime		Y		快递派送时间
5	mail_no	varchar				运单号
6	cp_code	varchar				快递公司编码
7	receiver_user_id	bigint unsigned				收件人用户 id
8	receiver_name	varchar				收件人姓名
9	receiver_address	varchar				收件人地址
10	receiver_phone	varchar				收件人手机号
12	status	int unsigned				快递单状态
13	notification_count	int unsigned		Y		派送通知次数
14	longitude	decimal		Y		收件人位置经度
15	latitude	decimal		Y		收件人位置纬度

（4）派单记录表。存储派单记录，见表 4-4。

表 4-4 派单记录表

列	列名	数据类型	长度	允许 为空	是否 自增	备注
1	id	bigint unsigned			Y	数据库记录主键
2	gmt_create	datetime				记录创建时间
3	gmt_modified	datetime				记录修改时间

4	sender_id	bigint unsigned				寄件用户 id
5	order_id	bigint unsigned				寄件订单 id
6	dispatch_id	bigint unsigned				派单记录 id
7	dispatch_date	datetime				派单时间
8	notify_user_ids	varchar	1024	Y		通知的快递员用户 id 集合
9	notify_count	bigint unsigned				通知次数
10	is_accepted	tinyint				是否被接单

（5）运营后台操作记录表。存储运营后台操作记录，其中，工单详细信息、审批人信息等存储在外部的工作流平台（注：表中 action_feature 字符串主要存储操作的系统参数信息）。表的结构见表 4-5。

表 4-5 运营后台操作记录表

列	列名	数据类型	长度	允许为空	是否自增	备注
1	id	bigint unsigned			Y	数据库记录主键
2	gmt_create	datetime				记录创建时间
3	gmt_modified	datetime				记录修改时间
4	work_id	bigint unsigned				操作人工号
5	work_name	varchar	128			操作人姓名
6	work_flow_id	bigint unsigned				工单 id
7	action_type	bigint unsigned				操作类型
8	action_feature	varchar	1024	Y		操作特征
9	action_status	bigint unsigned				操作状态
10	action_result	varchar	128	Y		操作结果

4.6 本章小结

本章将寄件快递单处理系统划分为的寄件订单子系统、寄件派单子系统和寄件运营后台子系统，并具体划分了内部功能模块，还对数据库进行了设计，对寄件订单表、寄件订单和运单的映射关系表、运单表、派单记录表和运营后台操作记录表，进行了具体字段的定义。

第五章 寄件快递单处理系统的具体实现

5.1 寄件订单子系统

基于微服务思想，将子系统实现的功能做划分成独立的子服务，每个子服务只提供尽可能独立且单一的功能，包括订单操作、订单查看、订单搜索、服务商信息查询等服务，具体列举如下：

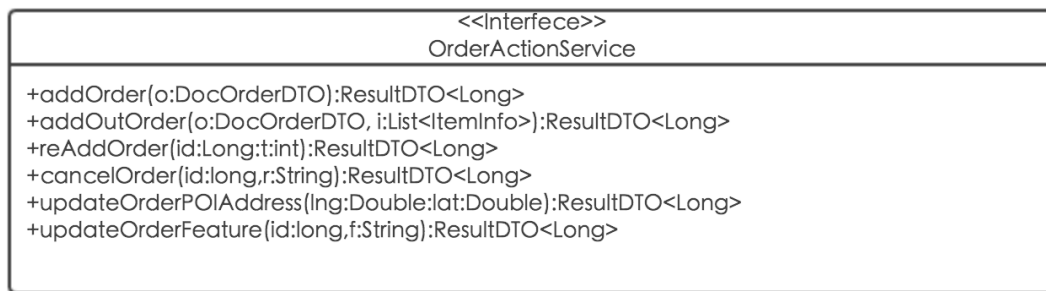


图 5-1 OrderActionService 的类图

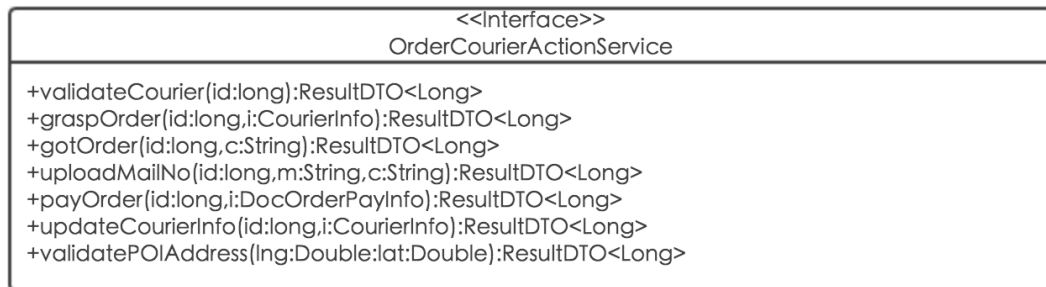


图 5-2 OrderCourierActionService 的类图

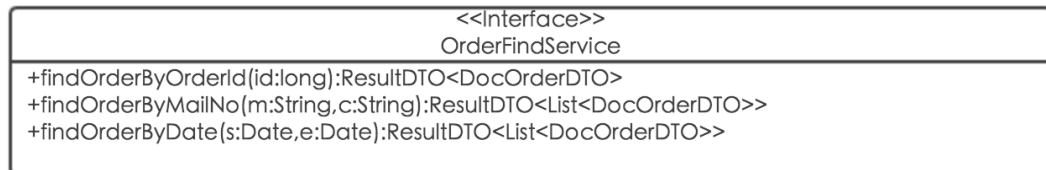


图 5-3 OrderFindService 的类图

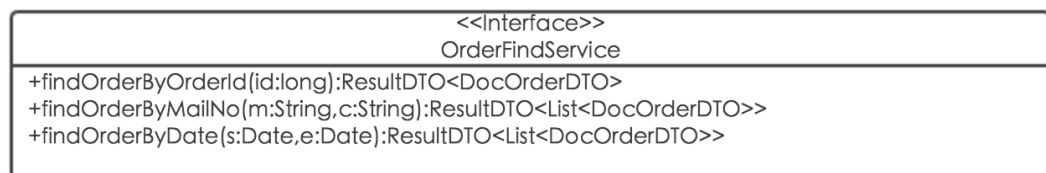


图 5-4 OrderQueryService 的类图

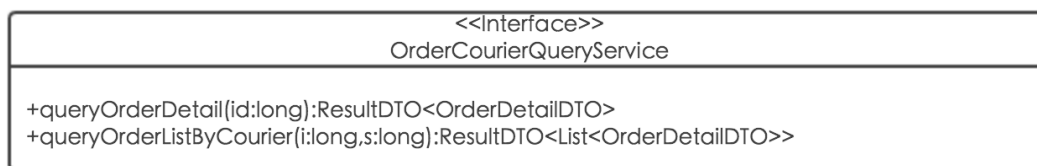


图 5-5 OrderCourierQueryService 的类图

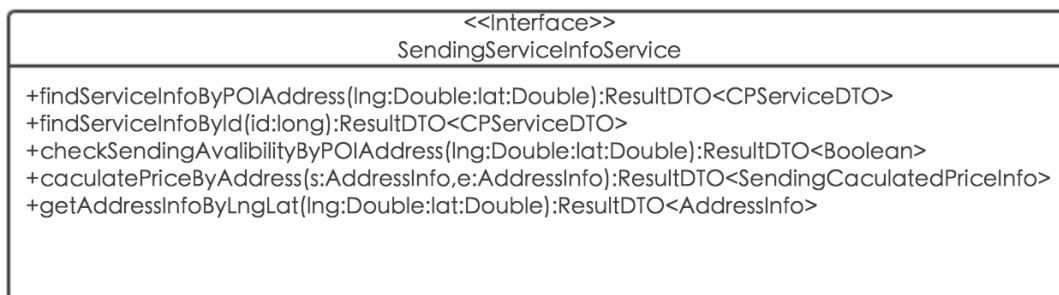


图 5-6 SendingServiceInfoService 的类图

以上列举了寄件订单子系统的核心服务，在实现上尽可能地让这些服务独立承担单一的功能。

其中，`OrderCourierActionService` 服务类和 `OrderCourierQueryService` 服务类，将快递员对寄件订单操作和查询功能分开，分别实现对应的微服务。`OrderActionService` 和 `OrderQueryService` 在实现上的思想类似，也是将寄件人对寄件订单的操作和查询功能分离。

实现的服务类依赖对应的管理器类，或其他的服务类进行工作。比如，操作订单的服务，依赖定时任务服务，实现上 `OrderActionService` 类依赖 `TimeNotifyProducer` 类。对于订单操作服务本身，实现的具体细节在管理类 `OrderActionManager` 中实现，服务类更多关注数据格式的解析以及依赖服务的组装。

订单搜索和订单详情查询服务，在实现上共用了订单查询管理类，主要是对数据库进行 sql 查询的封装。`OrderValidationService` 服务类主要提供了快递员基于位置的验证功能，快递员订单操作中的取件等环节，都需要依赖于这个功能来实现。

在 Spring 中，上面的订单操作、位置验证等子系统内部的服务，按照微服务标准编写，同时声明为 Bean 组件，直接通过注入的方式实现相互联系，整体实现上简洁明了。

这些类之间的主要依赖关系如图 5-7 所示（类方法的参数从略）。

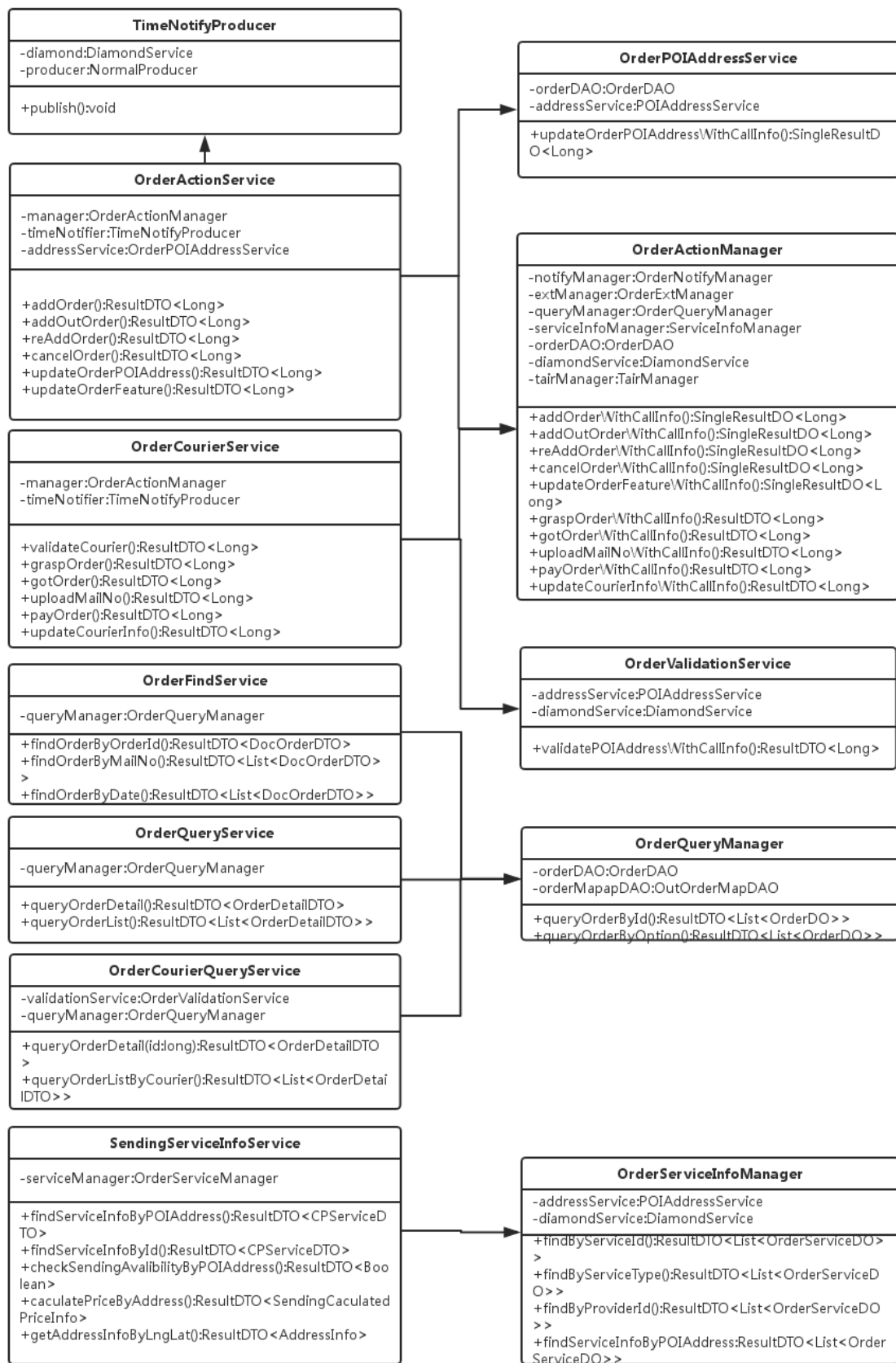


图 5-7 订单子系统关键类的类图

基于以上要实现的微服务类，和它们依赖的管理类和其他服务类，编写相关代码。其中，创建订单部分的核心代码如下：

```
@Override
public ResultDTO<Long> addOrder(DocOrderCreateDTO order) {
    ResultDTO<Long> result = new ResultDTO<Long>();
    if (!DocValidation.checkCreateDocOrderDO(result, order)) {
        return result;
    }

    RpcCallInfo call = getCallInfo("NBOrderService#addOrder");
    if (logger.isInfoEnabled()) {
        logger.info("IN@NBOrderService.addOrder,order={}, callInfo={}",
new Object[]{order, call});
    }

    try {
        return orderManager.addOrder(order, call);
    } catch (LogisticsException e) {
        logger.error("error@NBOrderService.addOrder,order={}", new
Object[]{order, e});
        throw e;
    } catch (Exception e) {
        logger.error("error@NBOrderService.addOrder,order={}", new
Object[]{order, e});
        DocValidation.setFaildResult(result, ServiceErrorCodeEnum.S02);
    }

    if (logger.isInfoEnabled()) {
        logger.info("success@NBOrderService.addOrder, result={}", new
Object[]{result});
    }
    return result;
}
```

5.2 寄件派单子系统

寄件派单子系统也通过接口的形式定义为微服务，对外暴露这些微服务提供的功能，包括派单记录、小件员位置信息、派单管理、订单状态同步等功能，具体用下面的类图表示。

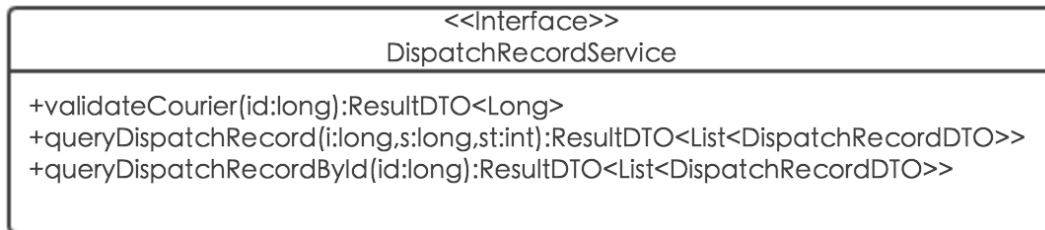


图 5-8 DispatchRecordService 的类图

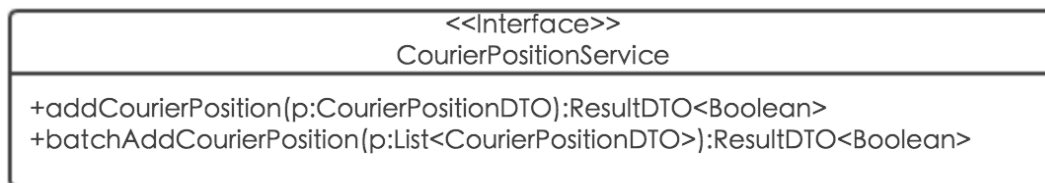


图 5-9 CourierPositionService 的类图

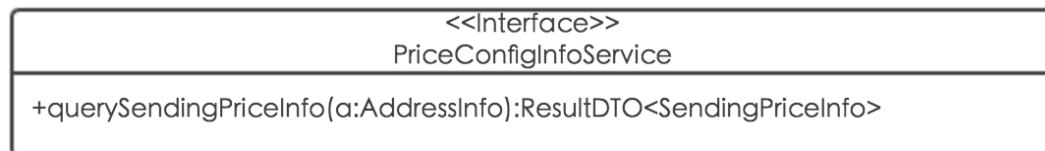


图 5-10 PriceConfigInfoService 的类图



图 5-11 OrderDispatchManager 的类图

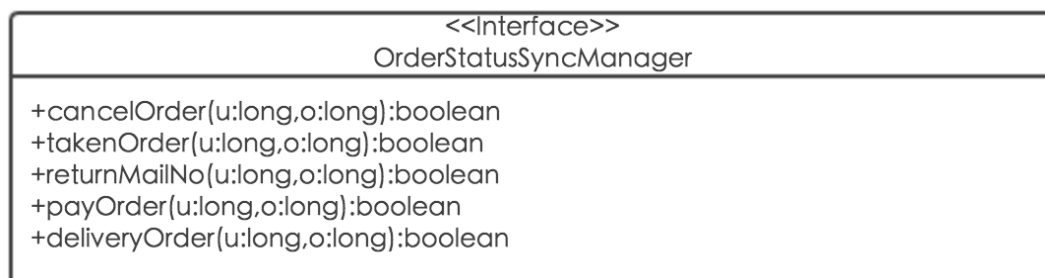


图 5-12 OrderStatusSyncManager 的类图

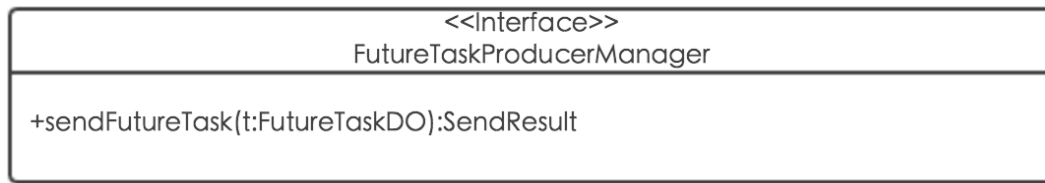


图 5-13 FutureTaskProducerManager 的类图

这些服务类依赖持久化层和对应的管理类，或其他的服务类进行工作，主要类之间的依赖关系如图 5-14 所示（类方法的参数从略）。

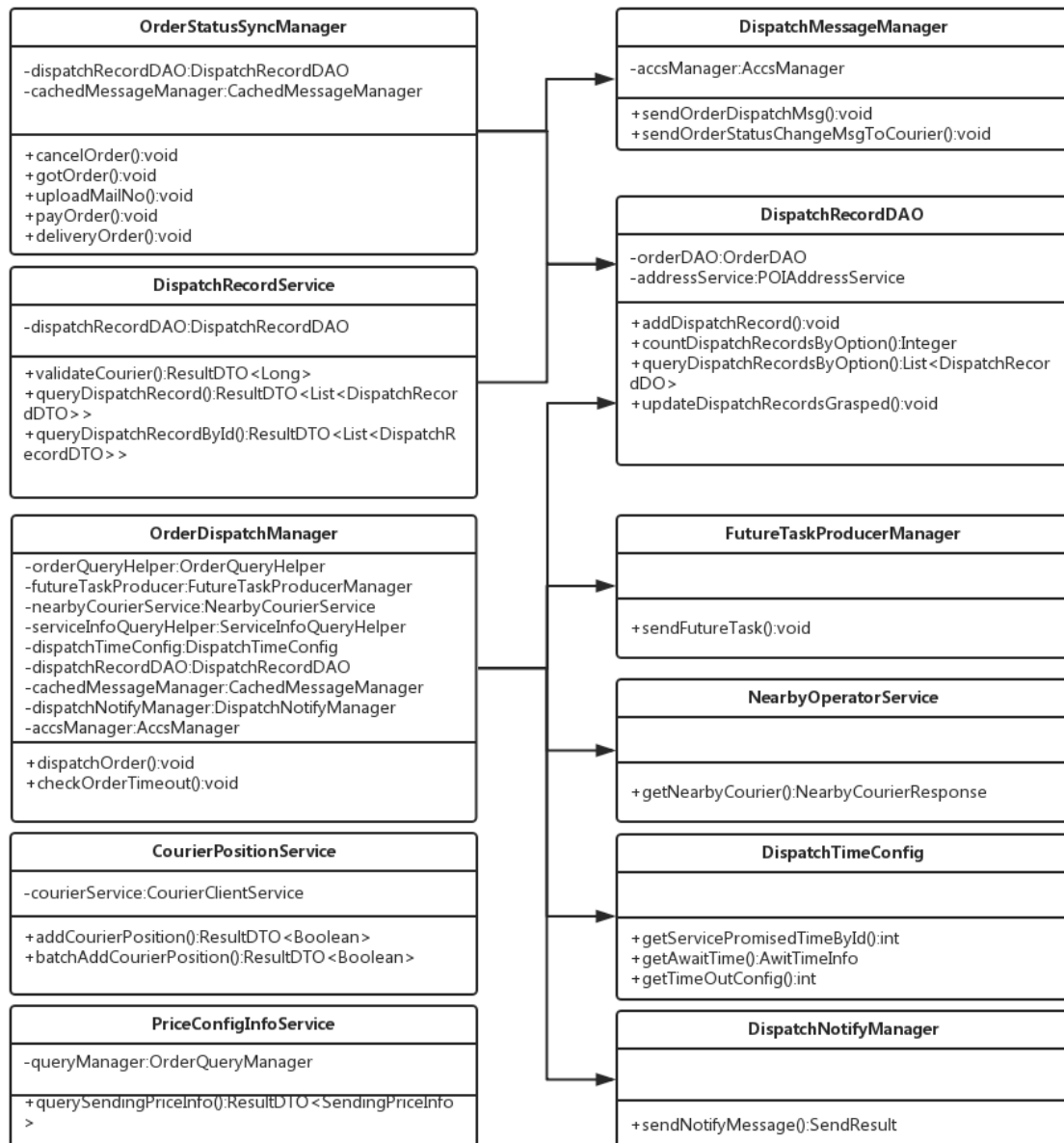


图 5-14 派单子系统关键类的类图

基于上面的微服务接口的设计，对应管理类 and 依赖的相关服务类的设计，

实现派单子系统。其中，派单处理部分的核心代码如下：

```

@Override
public void dispatchOrder(long userId, long orderId) {
    // 查询订单的信息
    OrderDO orderDO =
nbOrderQueryHelper.queryOrderDOByUserOrderId(userId, orderId);
    // 判断订单是否可派送
    boolean isDispatch = checkOrderCanDispatch(orderDO);
    if (!isDispatch) {
        logger.error("OrderDispatchManagerImpl.dispatchOrder
checkOrderCanDispatch:" + isDispatch + " userId:" + userId + " orderId:" +
orderId);
        return;
    }
    DispatchDO dispatchDO =
dispatchDOHelper.convertOrderDOToDispatchDO(orderDO);
    //延迟 5 秒派发
    createTimingDispatchTask(orderDO, DateUtils.addSeconds(new
Date(),5));
    return;
    // 根据发件人地址查询附近的小件员
    NearbyCourierResponse result =
nearbyCourierService.getNearbyCouriers(dispatchDO.getSendDetailAddress(),
dispatchDO.getPoi(), dispatchDO.getSenderLongitudeD(),
dispatchDO.getSenderLatitudeD());
    if (!result.isSuccess()) {
        logger.error("nearbyCourierService fail!, sendAddress:" +
dispatchDO.getSendDetailAddress()
        + " poi:" + dispatchDO.getPoi() + " LongitudeD: " +
dispatchDO.getSenderLongitudeD()
        + " LatitudeD:" + dispatchDO.getSenderLatitudeD() + "
errorMsg:" + result.getErrorMsg()
        + " orderId: " + orderId + " userId:" + userId);
    }
}

```

```

    }
    // 派送列表
    List<Long> dispatchList = new ArrayList<Long>();
    // 推送派单消息
    List<JSONObject> messages = new ArrayList<JSONObject>();
    String message = sendAccsMsgToUser(orderId, userId, facilitatorName,
    MsgContentConstants.SERVER_TYPE_FACILITATOR, 0);
    messages.add(JSON.parseObject(message));
    dispatchUsers.add(facilitatorName);
    dispatchList.add(facilitatorsDTO.getId());

    if(messages != null && !messages.isEmpty()){
        JSONArray array = new JSONArray(messages.size());
        for(int i=0,size=messages.size();i<size;i++){
            array.add(messages.get(i));
        }
        cachedMessageManager.putMessageByOrderId(
    orderId,MsgTypeConstants.C_ORDER_DIAPATCHER,array.toJSONString()
    );
    }
    if (dispatchList.size() > 0) {
        List<Long> levelOne = new ArrayList<Long>(); // 第一批派送
        List<Long> levelTwo = new ArrayList<Long>(); // 第二批派送
        // 进行分级别派送
        for (Long dispatchId : dispatchList) {
            // 判断是否要分批派单
            if (awaitTimeConfig.isAwait()) {
                createDispatchOrderTask(userId, dispatchId, dispatchDO);
            } else {
                dispatchOrderToUser(userId, dispatchId, dispatchDO);
            }
        }
    }
    // 启动定时检查任务,检查是否没人抢单

```

```

        createCheckOrderDeliveryTask(userId, dispatchDO);
        // 启动超时判断任务，检查订单是否超时，超时通知 SDK
        createCheckOrderTimeoutTask(orderId, userId);
    }
}

```

5.3 寄件运营后台子系统

寄件运营后台子系统，对外提供运营需要的微服务功能，包括首页 Banner 配置、APP 消息推送、Hotpatch 包上传，这些服务要依赖于系统内的权限管理服务功能，这些功能都比较单一，用下面的类图表示：



图 5-15 AOLRightPermissionService 的类图

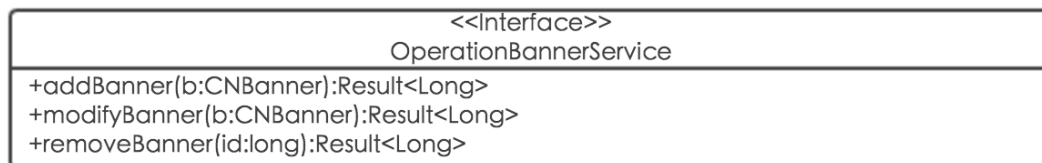


图 5-16 OperationBannerService 的类图

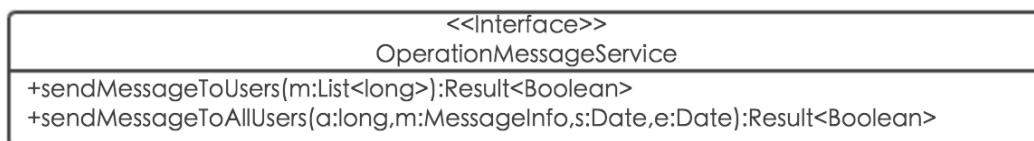


图 5-17 OperationMessageService 的类图

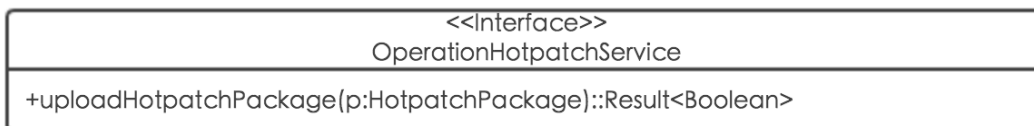


图 5-18 OperationHotpatchService 的类图

这些服务类依赖其他的管理器，或其他的 service 类进行工作，主要类之间的依赖关系如图 5-19 所示（类方法的参数从略）。

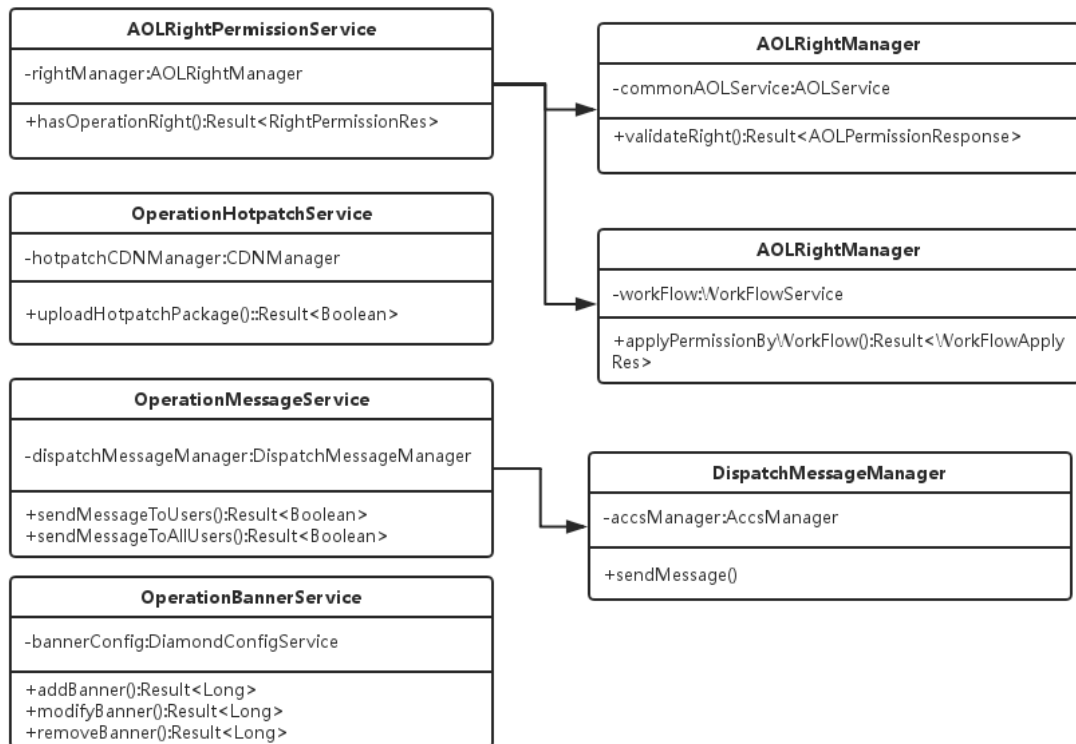


图 5-19 运营后台子系统关键类的类图

运营后台子系统，对接了前端 web 页面，运营人员需要在浏览器中通过指定的网址，访问运营后台，使用后台系统提供的上述功能。在浏览器中打开的运营后台的效果如图 5-20 至图 5-23 所示。

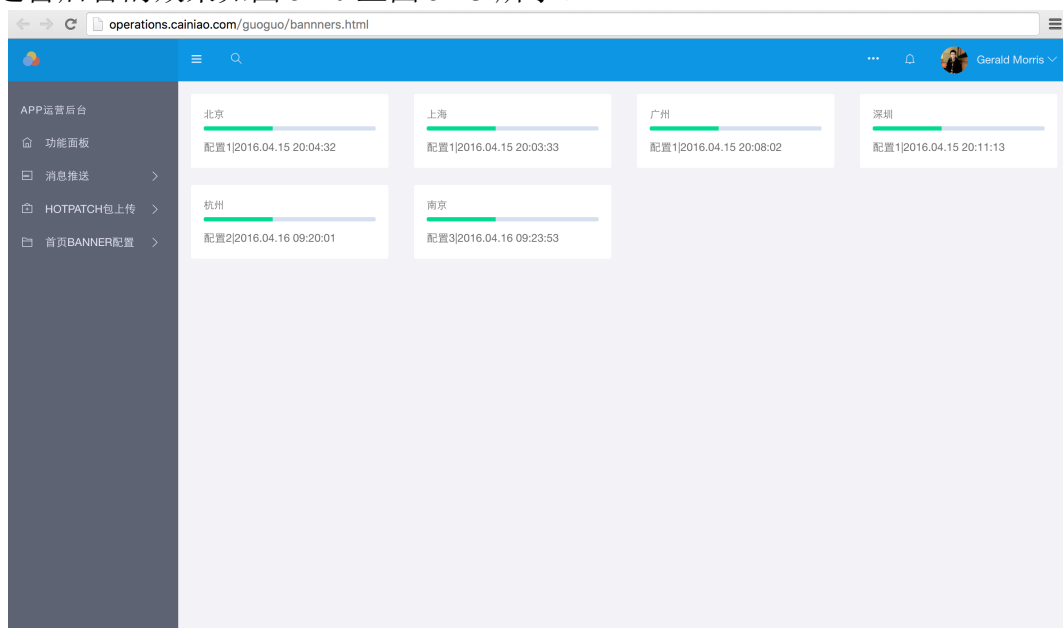


图 5-20 信件运营后台的 Banner 配置主页面

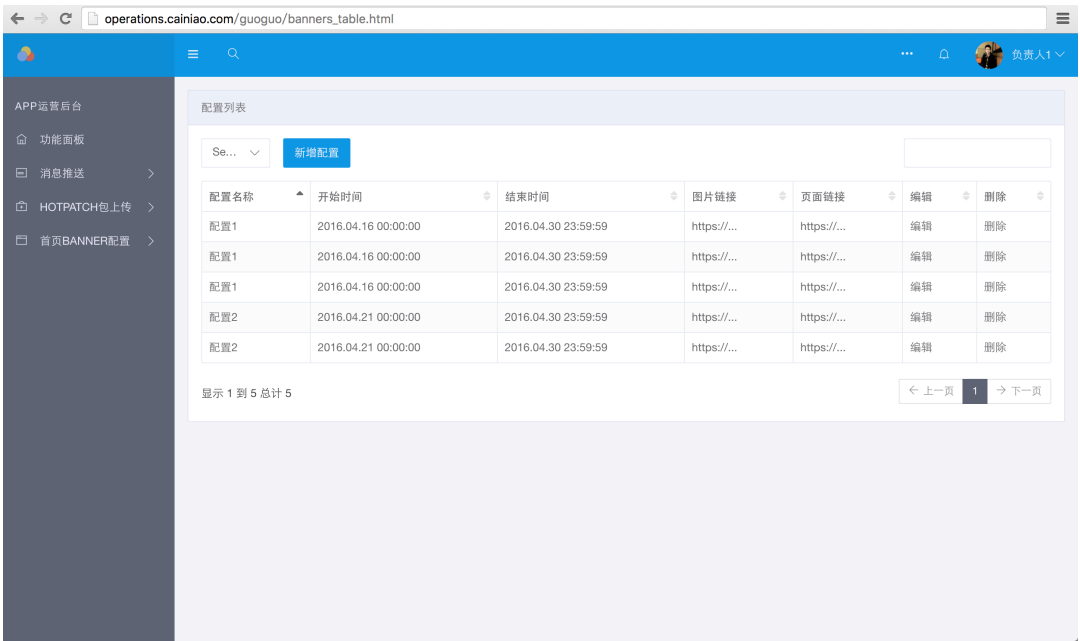


图 5-21 寄件运营后台的 Banner 配置详情页面

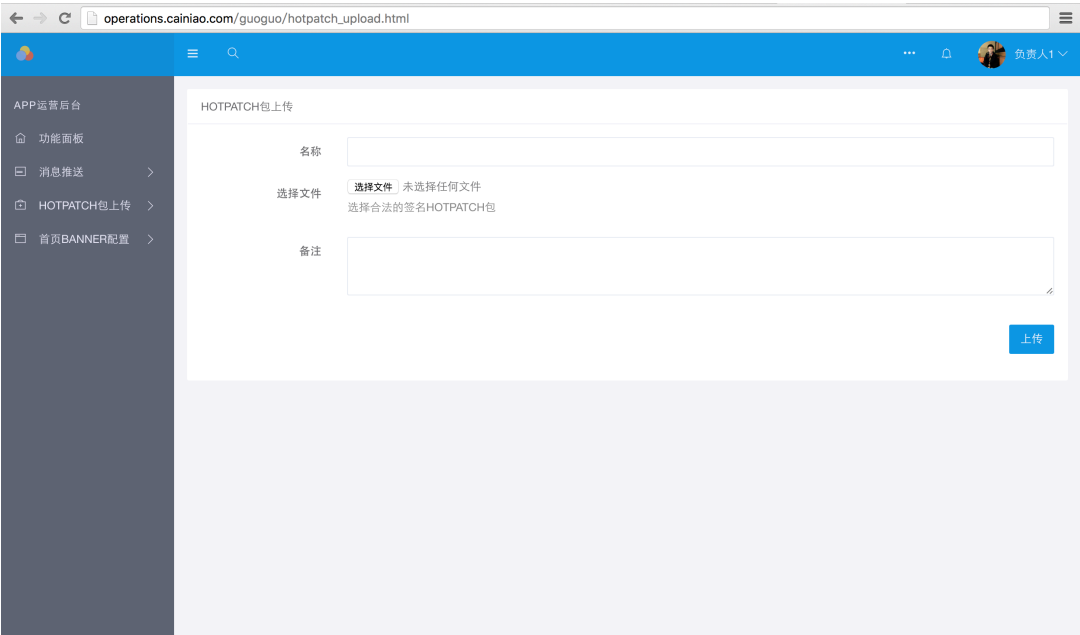


图 5-22 寄件运营后台的 Hotpatch 包上传页面

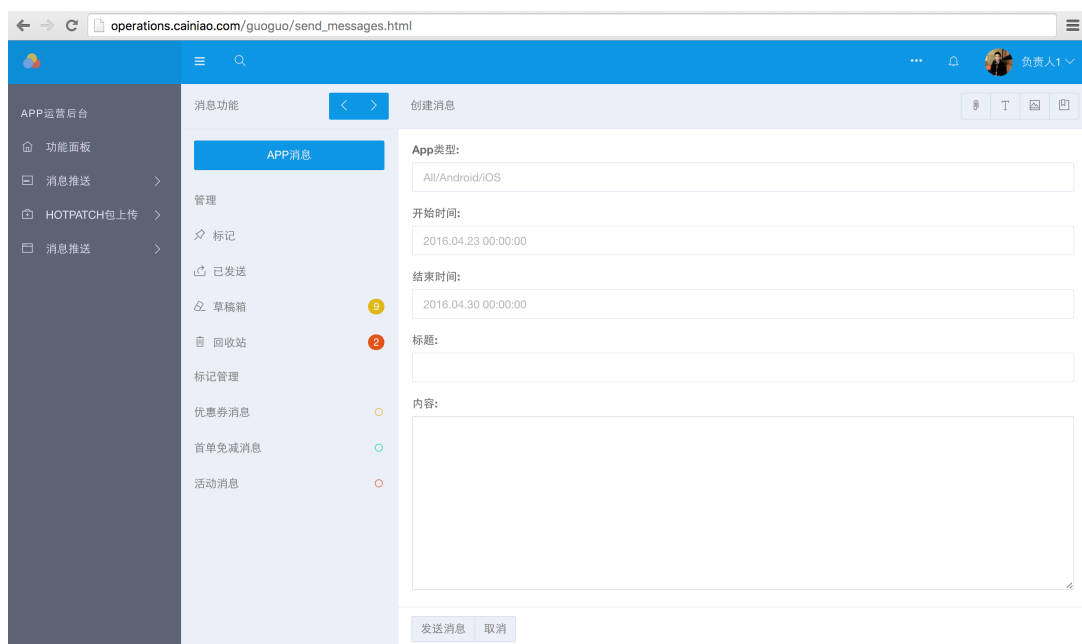


图 5-23 寄件运营后台的 APP 消息发送页面

5.4 微服务管理的实现

(1) 微服务标准化定义实例

每个技术平台（框架），所实现的微服务的定义形式并不相同。每个拓展出来的子系统，要根据自身的开发语言，使用的表述形式，存储方法进行具体的定义和实现。

其中，对于我们的寄件快递单处理系统的寄件订单子系统和寄件派单子系统，是在 Spring 框架基础上，将每一个微服务都以 Spring Bean 的形式声明和定义，并存储在 xml 格式的配置文件中，具体定义形式的例子如下：

在 Spring Bean 的配置文件中，对服务发布方（Producer），要定义实现类，实现的接口，继承的 HSF 基础类、超时时间、服务版本等信息。定义方式采用 xml 标签的形式，提供给 Spring 容器进行加载和使用。

```
<bean id="senderServiceImpl" class="com.company.service.core.SenderServiceImpl" />
<bean id="senderServiceProvider" class="com.taobao.hsf.app.spring.util.HSFSpring
ProviderBean" init-method="init">
    <property name="serviceInterface">
        <value>com.company.service.core.SenderService</value>
```

```

</property>
<property name="target">
    <ref bean="com.company.service.core.SenderServiceImpl" />
</property>
<property name="serviceName">
    <value>com.company.service.core.SenderService</value>
</property>
<property name="clientTimeout">
    <value>5000</value>
</property>
<property name="serviceVersion">
    <value>1.0.0</value>
</property>
<property name="serviceGroup">
    <value>HSF</value>
</property>
</bean>

```

对服务调用方（Consumer）的定义：

```

<bean name="clientSenderService" class="com.taobao.hsf.app.spring.util.HSFSpringConsumerBean" init-method="init">
    <property name="interfaceName" value="com.company.service.core.SenderService" />
    <property name="version" value="1.0.0" />
</bean>

```

（2）结合 Maven 配置和 Nexus 的应用实例

对于微服务发现和查询，以及元信息的存储，是通过 Maven 和 Nexus 工具实现的。定义的格式上，Nexus 通过 Artifact 的形式，定义程序包的信息，这种形式也是一种 xml 的格式。Nexus 提供对外提供服务的查询和发现功能。。

为本系统编写的信件服务的 Artifact 实例如下：

```

<project>
    <parent>
        <groupId>com.company.service</groupId>
        <artifactId>sender-service</artifactId>

```

```
<version>1.0.0</version>
</parent>
<modelVersion>1.0.0</modelVersion>
<artifactId>core</artifactId>
</project>
```

对于程序的构建 jar 包，则存储在 Maven 仓库中。Maven 为了管理依赖和构建过程，需要使用 pom（project object model）文件作为配置文件。pom 文件是一种 xml 形式的配置文件，包含了对工程依赖的软件组件的详细定义。

本项目编写的 pom 文件中，对于 hsf 包的依赖配置如下：

```
<dependency>
  <groupId>com.taobao.hsf</groupId>
  <artifactId>hsf.connector.spring</artifactId>
  <version>2.3.1</version>
</dependency>
```

对服务发布方的依赖配置如下：

```
<dependency>
  <groupId>com.company.service</groupId>
  <artifactId>service-sender</artifactId>
  <version>1.0.0</version>
</dependency>
```

5.5 本章小结

本章结合 Spring 技术框架和微服务架构，分别对寄件订单子系统、寄件派单子系统、寄件运营后台子系统，以及微服务管理的具体实现做了叙述。本章结合微服务思想，给出了关键服务类的定义，服务类之间的关系定义。此外本章还给出了寄件订单、派单子系统关键部分的代码，微服务的关键配置定义形式，并展示了运营后台的实现效果。

第六章 寄件快递单处理系统的测试与分析

6.1 寄件快递单处理系统的测试方案

6.1.1 单元测试方案

本课题拟采用 Mockito 进行程序的单元测试，在 Spring 工程中，以接口的实现类为测试单元，用 Mockito 模拟一个接口通过依赖注入获得的其他接口所提供的功能，从而实现了测试上和其他类的解耦。

单元测试方案的应用实例：针对目标类 `SenderServiceImpl`，编写测试类 `SenderServiceImplTest`。针对依赖的位置服务接口 `AddressService` 类，实现位置服务功能的 mock（模拟）：

```
AddressService addressService = Mockito.mock(AddressService.class);  
PowerMockito.when(POIUtil.queryPOI(Mockito.anyLong(),  
Mockito.anyLong())).thenReturn(new AddressInfo());
```

这样，对每个单元测试的目标类，就能直接使用模拟的外部类的功能，并且能够控制不同条件下的返回值。建立单元测试入口后，继续触发测试逻辑的执行，就能达到对测试单元，在正常数据流和异常数据流情况下进行单元测试的目的。

6.1.2 集成测试方案

在完成单元测试，保证程序单元无误后，使用自底向上的策略对子系统的各个内部模块进行合成，以及对多个子系统进行集成，保证接口之间没有问题，系统的整体功能能够满足需求。

在寄件快递单处理系统的各个子系统的集成中，列举了微服务提供的服务接口，逐个编写测试用例，保证依赖的接口之间，在相互访问时没有问题，最终完成服务端系统的集成测试。

6.2 寄件快递单处理系统的测试用例

按照寄件快递单处理系统的功能点，进行测试用例的设计，包括执行的前置条件、输入、预期结果、测试结果，并在相应的环境条件下执行测试用例，对子系统的功能性进行测试和验证。

寄件订单子系统、寄件派单子系统和运营后台子系统的测试用例的具体内容和执行的结果，记录在对应的功能测试用例表。

6.2.1 寄件订单子系统的测试用例

寄件订单子系统测试用例的测试功能点主要包括寄件服务信息查询、寄件人订单操作、快递员订单操作。

表 6-1 寄件订单子系统功能测试用例表

测试用例					
功能点	用例说明	前置条件	输入	预期结果	测试结果
寄件服务信息查询	校验位置	用户已经登录	经度, 纬度	位置校验结果	与预期一致
	根据距离计算价格	用户已登录, 通过区域位置验证	寄件人区域 id, 收件人区域 id	返回对应的寄件费用	与预期一致
	服务信息	用户已登录, 通过区域位置验证	经度, 纬度	返回服务信息列表	与预期一致
	获取具体位置信息	用户已经登录	经度, 纬度	返回具体位置信息	与预期一致
寄件人订单操作	创建订单	用户已登录, 通过区域位置验证	服务 id, 包裹类型, 加价, 寄件人和收件人位置和用户信息	返回订单 id	与预期一致
	取消订单	用户已登录, 订单处于已创建状态	订单 id, 取消原因	返回订单 id	与预期一致
	重新创建订单	用户已登录, 订单处于取消状态	订单 id, 重建类型	返回订单 id	与预期一致

	计算订单支付金额	用户已登录, 订单处于已接单状态	订单 id, 标准价格, 寄件服务价格, 优惠券 id	返回实际支付价格, 优惠券实际使用价格	与预期一致
	支付订单	用户已登录, 订单处于已接单状态	订单 id, 支付类型, 标准价格, 寄件服务价格, 优惠券 id	返回实际支付价格, 订单编号, 支付状态, 支付时间	与预期一致
	查询订单记录	用户已经登录	开始时间, 结束时间	返回订单信息列表	与预期一致
	查询订单详细信息	用户已经登录	订单 id	返回订单详情	与预期一致
	更新订单的寄件人位置	用户已登录, 订单处于已创建状态	经度, 纬度	更新成功: 订单 id; 失败: -1	与预期一致
快递员订单操作	验证快递员身份	用户录入快递员, 已登录	登录 id	验证结果: 成功	与预期一致
	接单	用户已登录, 通过快递员身份验证, 订单处于已创建状态	订单 id	接单成功: 订单 id; 接单失败: -1	与预期一致
	揽收包裹	用户已登录, 通过快递员身份验证, 订单处于已接单状态	订单 id, 揽收码	成功: 订单 id; 失败: -1	与预期一致
	回传运单号	用户已登录, 通过快递员身份验证, 订单处于已揽收状态	订单 id, 运单编号	成功: 订单 id; 失败: -1	与预期一致

	验证到达寄件人位置	用户已登录，通过快递员身份验证，订单处于已接单位置	经度，纬度，订单 id	是否在区域内	与预期一致
--	-----------	---------------------------	-------------	--------	-------

6.2.2 寄件派单子系统的测试用例

寄件派单子系统测试用例的测试功能点主要包括派单信息查询、位置服务、寄件信息查询、短信服务。

表 6-2 寄件派单子系统功能测试用例表

测试用例					
功能点	用例说明	前置条件	输入	预期结果	测试结果
派单信息查询	验证快递员身份	用户录入快递员，已登录	登录 id	验证结果:成功	与预期一致
	派单记录信息	用户已登录，通过快递员身份验证	页码，页大小，派单状态	派单记录列表	与预期一致
	派单记录详情	用户已登录，通过快递员身份验证	派单记录 id	单条派单记录详情	与预期一致
位置服务	上传位置信息	用户已登录，且为注册快递员	登录 id，经度，纬度，apk 信息	上传成功/失败	与预期一致
寄件信息查询	寄件费用	用户已登录，且为注册快递员	区域 id	首重价格，续重价格	与预期一致
短信服务	发送短信	用户已登录，且为注册快递员	派单 id，消息内容	发送结果:成功	与预期一致
	获取派单短信消息记录	用户已登录，且为注册快递员	派单 id	获得短信消息列表	与预期一致

6.2.3 寄件运营后台子系统的测试用例

寄件运营后台子系统测试用例的测试功能点主要包括权限验证、消息推送、Hotpatch 包上传，首页 Banner 配置。

表 6-3 寄件运营后台子系统功能测试用例表

测试用例					
功能点	用例说明	前置条件	输入	预期结果	测试结果
权限验证	操作权限	操作用户已登录	页面路径，用户 id	正常访问页面，或被禁止访问	与预期一致
消息推送	向特定用户推送	用户录入快递员，通过权限验证	目的用户 id，消息标题，消息内容	目标用户 app 接收到消息推送	与预期一致
	向所有用户推送	用户录入快递员，通过权限验证	消息标题，消息内容	目标用户 app 接收到消息推送	与预期一致
Hotpatch 包上传	上传补丁包	用户录入快递员，通过权限验证	上传任务名称，包文件，备注	上传成功/失败	与预期一致
首页 Banner 配置	增加配置	用户录入快递员，通过权限验证	配置名称，开始、结束时间，图片链接	成功：配置 id，失败：-1	与预期一致
	编辑配置	用户录入快递员，通过权限验证	配置 id，配置名称，开始、结束时间，图片链接	成功：配置 id，失败：-1	与预期一致
	删除配置	用户录入快递员，通过权限验证	配置 id	成功：配置 id，失败：-1	与预期一致

6.2.4 线上 API 数据统计和性能分析

本课题对上线后的寄件快递单处理系统提供的服务进行管理，并统计了线上服务的 API 调用数据，其中某一天的具体的统计数据，如图 6-1 所示。

[菜鸟无线_CNWIRELESS]：一共有51个API									
API	版本	请求总次数	平均响应时间	成功请求次数	成功平均响应时间	异常请求次数	系统异常次数	业务异常次数	错误率
mtop.cnwireless.CNSenderService.queryServiceList	1.0	1,241,797	132.395	1,234,338	135.143	7,459	1,691	5,768	0.6%
mtop.cnwireless.CNSenderService.queryOrderDetail	3.2	995,510	77.105	996,489	137.625	79,021	6	79,015	0.5%
mtop.cnwireless.CNGrabService.querySenderEntry	3.2	979,000	148.487	978,986	148.278	14	7	7	0%
mtop.cnwireless.CNSenderService.querySenderOrderList	1.0	1,164,338	127.239	1,164,279	123.297	59	59	0	0.1%
mtop.cnwireless.CNSenderService.queryStationList	1.0	554,690	71.011	554,634	70.918	56	9	47	0.1%
mtop.cnwireless.NBCouponMtopService.queryCouponList	3.3	440,210	11.272	440,177	8.621	33	24	9	0.1%
mtop.cnwireless.CNGrabService.queryPoiInfoListByKeywords	1.0	40,162	210.774	40,146	209.886	16	13	3	0%
mtop.cnwireless.CNSenderService.querySenderPriceInfo	1.0	534,543	3.177	534,539	2.791	4	3	1	0%
mtop.cnwireless.CNSenderService.queryServiceList	3.2	434,244	152.663	434,231	152.395	13	2	11	0%
mtop.cnwireless.ShareService.shareLottery	1.0	432,811	41.75	432,791	39.042	20	20	0	0.1%
mtop.cnwireless.CNSenderService.queryOrderRecord	3.2	428,674	48.945	428,646	48.636	28	5	23	0.1%
mtop.cnwireless.CNAddressService.queryUserAddressInfoList	1.0	624,491	7.965	624,482	6.458	9	8	1	0%
mtop.cnwireless.CNSenderService.queryUserId	1.0	424,178	2.587	423,778	2.436	400	400	0	1.7%
mtop.cnwireless.CNAddressService.synUserAddressInfo	1.0	321,549	8.956	321,499	7.706	50	9	41	0.2%
mtop.cnwireless.CNSenderService.payOrder	1.0	17,006	23.64	16,613	23.74	393	4	389	2.3%
mtop.cnwireless.CNSenderService.queryOrderDetail	1.0	316,347	114.065	316,344	114.086	3	3	0	0%
mtop.cnwireless.CNNearPostManService.queryPostManModelList	1.0	14,998	161.145	14,998	161.145	0	0	0	0%
mtop.cnwireless.CNAddressService.queryDivision	1.0	12,959	122.942	12,948	120.618	11	11	0	0.1%
mtop.cnwireless.CNSenderService.createOrder	3.2	12,220	135.246	11,985	132.728	235	10	225	1.9%
mtop.cnwireless.CNSenderService.calculateOrderPayAmount	1.0	11,852	8.545	11,835	7.656	17	3	14	0.1%
mtop.cnwireless.NBCouponMtopService.queryCouponList	3.2	18,951	7.233	18,950	6.741	1	1	0	0%
mtop.cnwireless.CNSenderService.frozenCoupon	1.0	6,393	16.353	6,279	14.491	114	3	111	1.8%
mtop.cnwireless.CNSenderService.createOrderWithFrom	3.2	6,329	132.385	6,238	130.911	91	26	65	1.4%
mtop.cnwireless.CNSenderService.querySenderOrderListByApp	1.0	4,839	125.34	4,832	122.7	7	7	0	0.1%
mtop.cnwireless.CNGrabService.querySenderEntry	1.0	3,674	141.946	3,671	142.061	3	3	0	0.1%
mtop.cnwireless.CNSenderService.obtainCPAgingList	1.0	2,742	148.412	2,727	149.224	15	0	15	0.5%
mtop.cnwireless.CNSenderService.createOrder	1.0	2,235	145.357	2,232	141.337	3	2	1	0.1%
mtop.cnwireless.CNSenderService.cancelOrder	3.2	1,618	143.833	1,582	146.929	36	0	36	2.2%
mtop.cnwireless.CNSenderService.cancelSenderOrder	1.0	1,473	30.24	1,473	30.24	0	0	0	0%
mtop.cnwireless.CNSenderService.cancelOrder	1.0	1,171	140.294	1,169	140.532	2	2	0	0.2%
mtop.cnwireless.CNSenderService.queryOrderRecord	1.0	1,133	359.667	1,107	263.331	26	26	0	2.3%
mtop.cnwireless.CNSenderService.querySenderOrder	1.0	643	120.616	643	120.616	0	0	0	0%
mtop.cnwireless.CNSenderService.createStationSenderOrder	1.0	66	71.576	66	71.576	0	0	0	0%

图 6-1 系统 API 调用量日统计数据

从统计的数据可以看出，SenderService 的日调用次数达到 100 万余次，参考了运维监控数据，峰值也能达到 5 千每分钟，达到了系统的性能需求。

6.3 本章小结

本章给出了寄件快递单处理系统的单元测试和集成测试方案，分别对各子系统设计了具体的测试用例，并结合线上接口的统计数据，对系统的实际性能进行了分析。

第七章 总结和展望

7.1 本文的主要工作

我国电子商务带动下，快递行业的高速发展的背景下，寄件业务迅速发展，现有的寄件处理系统存在诸多问题，系统的针对性、流程的规范性、服务的确定性、系统的拓展性等等。

本文的主要工作如下：

（1）本文首先对以及寄件快递处理过程中存在的问题进行了分析，介绍了实现处理常用的技术手段，包括技术框架、软件组件、架构设计等。

（2）本文针对寄件快递单处理系统的具体需求，重新定义了寄件快递单处理的核心流程，对系统进行了分析和设计。本课题也结合轻量级 J2EE 中的 Spring 框架，分析和应用了微服务架构，对服务的统一性规范性，对外支持的多样性，系统的可拓展性进行了陈述。

（3）本文对建立的寄件快递单处理系统进行了功能和性能上的验证，保证系统在实际生产环境下满足需求的目标。本课题是在当前对寄件快递单处理研究不多的情况，尝试性地研究了寄件快递单处理系统应用，对于目前高速发展快递业有很好的借鉴意义和价值。

7.2 今后待研究的问题

本文在设计和实现寄件快递单处理系统的过程中，发现仍然存在一些问题，需要在以后的工作中进行深入的分析研究：

（1）系统安全性。本文没有单独就安全性问题，包括身份认证、数据加密等，进行深入分析探讨。为了继续构建更可靠和成熟的系统，有必要开展对系统安全性问题的分析研究。

（2）对微服务相互通信协议的研究。目前微服务架构的应用越来越广泛，微服务的通信也存在多种实现方式，微服务之间可以利用现有的 Thrift、原始 RPC 方式或者本文使用的 HSF 方式进行通信，尚却少一种针对微服务特点的跨平台多语言支持的通信协议技术。因此可以对微服务相互之间通信协议，进行研究探讨，将对微服务的应用提供有力支持。

参考文献

- [1] 中国互联网络信息中心 CNNIC. 2014 年中国网络购物市场研究报告 [R]. Beijing: CNNIC, 2015.
- [2] 中国互联网络信息中心 CNNIC. 第 37 次中国互联网络发展状况统计报告 [R]. Beijing: CNNIC, 2016.
- [3] 国家邮政局. 中国快递发展指数首次发布 [N/OL]. [2015-03-30]. http://www.mot.gov.cn/jiaotongyaowen/201510/t20151014_1897390.html.
- [4] 张意轩. 2014 年中国快递业务量达 140 亿件 跃居世界第一 [N]. 人民日报海外版, 2015-01-07.
- [5] 国家邮政局. 国家邮政局公布 2015 年邮政行业运行情况 [N/OL]. [2016-01-14]. http://www.spb.gov.cn/dtxx_15079/201601/t20160114_710673.html.
- [6] 郭雪红. 菜鸟网络打通县村物流 农村淘宝实现 100% 覆盖 [N/OL]. [2015-07-09]. http://hzdaily.hangzhou.com.cn/mrsb/html/2015-07/09/content_2010946.htm.
- [7] 国家邮政局. 国家邮政局公布 2014 年邮政行业运行情况 [N/OL]. [2015-02-24]. http://www.spb.gov.cn/yzshjd/ywgl/201503/t20150324_437767.html.
- [8] 国家邮政局. 国家邮政局公布 2013 年邮政行业运行情况 [N/OL]. [2014-01-15]. http://www.spb.gov.cn/dtxx_15079/201401/t20140115_274540.html.
- [9] 国家邮政局. 国家邮政局公布 2012 年邮政行业运行情况 [N/OL]. [2013-01-16]. http://www.spb.gov.cn/dtxx_15079/201301/t20130116_158179.html.
- [10] 国家邮政局. 国家邮政局公布 2011 年邮政行业运行情况 [N/OL]. [2012-01-20]. http://www.spb.gov.cn/dtxx_15079/201201/t20120120_156963.html.
- [11] 国家邮政局. 国家邮政局公布 2010 年邮政行业运行情况 [N/OL]. [2011-01-25]. http://www.spb.gov.cn/xytj/tjxx/201101/t20110125_159862.html.
- [12] Michael A. Rivalto. System for automated package pick-up and delivery: US, 6,690,997[P]. 2001-09-13.
- [13] Ole-Petter Skaakrud, Cameron Dee Dryden, Jeffrey Robert Smith, et al. Internet-based shipping, tracking, and delivery network supporting a plurality of mobile digital image capture and processing (MICAP) systems: US, 7,870,999[P].
- [14] 赵园园. 电子商务环境下社区智能快递系统助力快递末端配送效率提升[J]. 物流技术, 2015, 34(1):158-160.

- [15] 许校境, 郑召文. 基于 RFID 的快递系统的研究与应用[J]. 无线通信技术, 2013, 14(4):57-60.
- [16] 武防震, 姚国祥. 基于 UML 的快递系统建模[J]. 微机发展, 2003.13(3):123-125.
- [17] 李刚. 轻量级 Java EE 企业应用实战[M]. 北京: 电子工业出版社. 2008.
- [18] Villamizar M., Garces O.; Castro H., et al. [29] Ole-Petter Skaaksrud, Cameron Dee Dryden, Jeffrey Robert Smith, et al. Internet-based shipping, tracking, and delivery network supporting a plurality of mobile digital image capture and processing (MICAP) systems: US, 7,870,999[P].
- [19] 邢帆. 微服务 On Line[J]. 中国信息化, 2015.12(6):76-77.
- [20] Martin L. Abbott, Michael T. Fisher. The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise[M]. 2nd Edition. Boston: Addison-Wesley Professional, 2015.
- [21] Thijmen de Gooijer, Anton Jansen, Heiko Koziolk, et al. An Industrial Case Study of Performance and Cost Design Space Exploration[C]// Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering. New York: ACM, 2012:205-216.
- [22] Chris Richardson. Pattern: Microservices Architecture[N/OL]. [2014-05-28]. <http://microservices.io/patterns/microservices.html>.
- [23] Shahir Daya, Nguyen Van Duy, Kameswara Eati, et al. Microservices from Theory to Practice[N/OL]. [2015-08-03]. <http://ibm.com/redbooks>.
- [24] Jose Ignacio Fern, Carlos A. Iglesias, Mercedes Garijo. Microservices: Lightweight Service Descriptions For Rest Architectural Style[C]// International Conference on Agents and Artificial Intelligence (ICAART). Valencia: INSTICC Press, 2010:513-516.
- [25] Vinh D. Le, Melanie M. Neff, Royal V. Stewart, et al. Microservice-based Architecture for the NRDC[C]// Industrial Informatics (INDIN). Cambridge: 2015 IEEE 13th International Conference on, 2015:1655-1664.
- [26] 马海收, 吴振新. 微服务在数字资源长期保存系统中的发展研究[C]// 2011 图书馆信息技术的应用、服务和创新学术研讨会暨第 3 届数字图书馆与开放源代码软件(DLIB&OSS2011)学术研讨会. 北京: 图书馆学研究, 2011: 237-244.
- [27] 杨霄彩. 基于 J2EE 的订单跟踪微架构研究与应用[J]. 计算机与数字工程, 2010, 38(4): 164-166.
- [28] 张冀峰. 基于 Spring 框架的供应链订单管理系统的设计与实现[D]. 天津:

南开大学, 2014.

[29] 周会强. 基于 Struts 框架的 Java Web 应用开发研究[J]. 科技通报, 2012, 28(6): 36-37.

[30] Niolet D'mello, Larkins Carvalho. Struts 2 - The modern web application framework[J]. International Journal of Modern Engineering Research, 2013, 3(3):1854-1856.

[31] 顾海振. 基于 Spring、Struts 和 Hibernate 整合应用的货物管理系统[D]. 山东: 山东大学, 2014.

[32] 范为. 基于 Spring 框架面向多业务 Web 平台开发框架研究[J]. 现代计算机, 2012, 12(23):131-133.

[33] 张文字, 许明健, 薛昱. 论 spring 的零配置与 XML 配置[J]. 计算机系统应用, 2015, 24(2):270-275.

[34] 何思平, 方美琪. Spring AOP 技术在电子商务中的应用[C]// 2008 年 IT 服务促进企业信息研讨会论文集. 北京: 信息化纵横, 2008: 32-34.

[35] Thomas Wurthinger, Danilo Ansaloni, Walter Binder, et al. Safe and atomic run-time code evolution for Java and its application to dynamic AOP[J]. ACM SIGPLAN Notices: A Monthly Publication of the Special Interest Group on Programming Languages, 2011, 46(10):603-615.

[36] Ankur Bawiskar, Prashant Sawant, Vinayak Kankate, B.B. Meshram. Spring Framework: A Companion to JavaEE[J]. International Journal of Computational Engineering & Management, 2012, 15(3):41-49.

[37] Benussi. L, Bianco. S, Colafranceschi. S. A new approach in modeling the behavior of RPC detector[J]. Nuclear Instruments and Methods in Physics Research, Section A. Accelerators, Spectrometers, Detectors and Associated Equipment, 2012, 661(S1):131-133.

[38] Kai Sachs, Samuel Kounev, Alejandro Buchmann. Performance modeling and analysis of message-oriented event-driven systems[J]. Software and systems modeling, 2013, 12(4):54-67.

[39] Don Denoncourt. POJO Programming with iBATIS[J]. iSeries news, 2006, 0(322):61-62.

[40] Mostafa, S. Xiaoyin Wang. An Empirical Study on the Usage of Mocking Frameworks in Software Testing[C]// The 14th International Conference on Quality Software(QSIC 2014). Dallas: IEEE, 2014:127-132.