華中科技大學

本科生毕业设计[论文]

基于 SpringBoot 的
后台微服务管理系统的设计与实现

院　　系 计算机科学与技术学院

专业班级 计卓 1401

姓　　名 俞洋

学　　号 U201415219

指导教师 胡侃　肖威

2018 年 5 月 20 日

学位论文原创性声明

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包括任何其他个人或集体已经发表或撰写的成果作品。本人完全意识到本声明的法律后果由本人承担。

作者签名：　　年　月　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保障、使用学位论文的规定，同意学校保留并向有关学位论文管理部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权省级优秀学士论文评选机构将本学位论文的全部或部分内容编入有关数据进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于 1、保密口，在　　年解密后适用本授权书

2、不保密口　。

（请在以上相应方框内打“√”）

作者签名：　　年　月　日

导师签名：　　年　月　日

摘 要

微服务是一种有别于传统架构的开发设计架构，具体表现为将一个传统的整体型应用拆分为多个微服务。微服务会通过提供基于 HTTP/JSON 的 API 端点以做到与其它服务集成在一起，通过使用微服务架构，系统可以更容易地适应不断增长的容量需求，增强自身的水平可伸缩性。

在企业的实际开发环境中，各个业务团队之间有着联系，开发人员之间也有着各种各样的服务调用关系，但是这些服务调用过程并没有被统一地管理起来，对于相同或相似的服务有着许多种调用方式和渠道，造成服务管理混乱，效率低下等问题。

本文通过对公司的开发环境进行需求调研，设计实现了一个满足开发需求的基于 SpringBoot 的微服务管理系统。本系统使用服务端程序来本地调用服务，利用 Java 的 NIO 来负责服务端和控制中心的通信，调用方通过 RESTful 方式来向控制中心调用服务，将服务调用统一管理，提高整体效率。

关键词：微服务；SpringBoot；NIO；RESTful

Abstract

Micro service is a development and design framework different from traditio nal architecture, which is specifically divided into several traditional services. Mi cro services can be integrated with other services by providing API endpoints ba sed on HTTP/JSON, and by using the micro service architecture, the system can be more easily adapted to the growing capacity requirements and enhance their level of scalability.

In the actual development environment of the enterprise, there are connections among the various business teams, and there are a variety of service call relationships among the developers. However, these service invocation processes have not been uniformly managed, and there are many kinds of calls and channels for the same or similar services, resulting in the chaos of service management and low efficiency.

In this paper, according to analysis the needs of the company's development environment, we design and implement a SpringBoot based micro service management system. The system use the server program to call the service locally, use the NIO of the Java to communicate the server and control center. The client calls the service to the control center through the RESTful mode. The service call is managed in a unified way, and the overall efficiency is improved.

Key Words： Micro service; SpringBoot; NIO; RESTful

目 录

1 绪　论

1.1 课题研究的背景和意义

由于全球化、城市化等全球大趋势对世界各国的社会和企业产生的影响，如今面对各种市场是比面对需求问题更为关键的挑战。发达国家的产品需要高度个性化，而新兴市场的产品需要适应区域需求。此外，现在还存在着一种缩短创新周期的趋势[1]。

面对这些挑战，从 IT 角度提出的解决方案是面向服务架构（SOA）和微服务的概念。

微服务的核心概念是将应用程序细粒度地分解成微小的服务。与 SOA 相比，微服务中被封装的服务是小规模的。这些围绕业务组织的服务，在包括开发、部署和维护的全生命周期中，都是相互独立的，并且彼此之间不直接通信，而是使用依赖轻量级通信协议的独立定义的接口进行通信操作，通过这种方式可以减少服务之间的依赖性。另外，微服务架构会在各个实例之间将数据存储进行拆分，而不是依赖一个中央数据库，除了数据之外，例如计算性能等资源也同样适用。

在企业的实际开发环境中，各个业务团队之间有着联系，开发人员之间也有着各种各样的服务调用关系，但这些调用过程并没有统一的管理，对于相同或相似的服务有着许多种调用方式和渠道，造成服务管理混乱，效率低下等问题，因此需要建立一个独立的服务管理体系，而微服务架构正好符合公司的开发环境需求，所以本课题将设计一个基于 SpringBoot 的微服务管理系统（以下简称 MMS 系统），用于管理服务调用过程。通过使用 MMS 系统，服务调用方只需要通过访问特定的 url 即可发动服务的调用，控制中心会将访问进行处理然后将请求以 NIO 通信的方式发送至服务端程序，服务端程序收到请求后会以请求中说明的调用方式调用相应的服务，最后将结果通过控制中心返回给服务调用方，此时调用方只需对返回结果进行解析处理即可获得服务调用结果。在整个流程中，调用方需要做的就只有向特定 url 发送包含特定参数的请求，操作方便，而服务提供方只需要提供特定的调用接口和必需的配置文件即可。MMS 系统将服务提供方与服务调用方分离，使调用过程对开发者透明化，对服务提供了集中的管理，提高了开发的效率。

1.2 国内外研究现状

2012 年 3 月在第 33 届 Java 国际会议上，微服务作为学习案例被提出。2014 年 3 月 James Lewis 和 Martin Fowler 发表了关于 Microservices 的博文，其中阐述了微服务的概念与原理以及微服务与传统单体式应用的区别[2]。

微服务架构作为一种架构方式与设计理念，目前对其还没有统一的定义，而作为微服务架构的提出者，Martin Fowler 的博文是大家公认的参照标准。他在文章中提炼出了微服务架构的九大特性：

1. 服务组件化；
2. 按业务组织团队；
3. 以做“产品”的态度开发；
4. 智能端点与哑管道；
5. 去中心化治理；
6. 去中心化管理数据；
7. 基础设施自动化；
8. 容错设计；
9. 演进式设计。

总的来说，微服务结构可以理解为一种通过细粒度的服务来进行协同开发的方法，每个服务在全生命周期内都是互相独立的进程，彼此之间通过轻量级的通讯协议进行通讯[3]。

国外的各个研究小组在微服务架构的相关开发和应用方面有许多实现的成果，比如 Matthias Vianden 等人提出了一种可应用于企业测量系统开发的参考架构，为微服务架构提供具体应用案例[4]。还有一个可以应用于微服务构建的应用的测试的框架也被研究了出来，通过使用该框架可以有效地减少维护成本[5]。 Alexandr Krylovskiy 和 Edoardo Pattiy 将微服务与 SOA 架构比较，得到了微服务架构具有技术异构性的结论，同时也指出了微服务架构会在一定程度上提高系统的复杂度[6]。Vinh D. Le，Melanie M. Neff，Royal V. Stewart 等在文献[7]中阐述了因为微服务架构具有较强的可扩展性，灵活的结构，以及不被编程语言、数据库类型、框架等所限制的优点，因此适合进行云端部署。

企业方面，目前已有许多针对微服务架构中不同场景的各种解决方案和开源框

架。服务治理方面，有 Dubbo，DubboX，Eureka 等，分布式配置管理方面，有 Disconf，Archaius，Config 等，批量任务方面，有 Elastic-Job，Azkaban，Task 等，服务跟踪方面，有 Hydra，Sleuth，Zipkin 等。在此基础上，还出现了整合了多个项目的 SpringCloud 框架。

随着互联网的发展，越来越多的系统开始使用微服务架构，例如图书馆系统[8]，掌上校园系统[9]，城市一卡通系统[10]等。同济大学也研究了一种新型云件 PasS 平台[11]，这个基于轻量级容器技术和微服务架构的平台实现了将传统软件直接部署到云端运行的功能。

1.3 论文研究内容

本文主要阐述了作者在阿博茨科技有限公司实习期间，通过了解企业具体开发工作的需求和流程，基于微服务架构的具体环境，设计并实现了满足企业开发过程中简洁高效的服务调用、统一方便的服务管理等需求的微服务管理系统 MMS 系统。

在具体的设计实现过程中，本文结合了企业的具体开发需求，对 MMS 系统进行了需求分析，并给出了具体的设计与实现方案。根据微服务的调用过程，将整个系统分为几个模块：请求接受模块，请求发送模块，服务管理模块。请求接受模块和请求发送模块主要用于服务端程序和服务管理模块之间进行通信，具体通信内容包括服务注册、服务注销、服务调用等。服务管理模块主要负责解析调用请求，转发调用请求，注册服务，注销服务等功能。

最后，本文还对 MMS 系统进行了功能测试，从测试结果的角度展现了 MMS 系统的具体功能和使用方法。

1.4 论文组织结构

本文共分为五个章节：

第一章，首先简要地介绍了本文的选题背景和研究意义，接着对国内外的研究现状进行了大致地说明，然后说明了本文的主要研究内容，最后介绍了论文的总体结构。

第二章，简要地介绍了本文在实现过程中所使用的关键技术和相关理论，同时

也说明了本文的技术选型。

第三章，分析系统需求，设计和完善了系统的基本架构，以请求和结果的信息流的方式确定了系统的模块和工作方式，同时为了开发者能尽可能方便地使用系统，对具体接口进行了优化设计。

第四章，具体设计各个模块，以信息流的流动过程为主要分析方式，明确了各个模块所实现的功能和实现方式。

第五章，测试系统能否实现所需功能，通过完成与实际工作内容一致的测试用例，展现系统是否实现了优化服务管理、提高开发效率的设计目标。

第六章，总结完成的主要内容，分析系统的不足并提出还可以改进的方向和方法。

1.5 课题来源

针对实习单位——北京阿博茨科技有限公司武汉分公司的数据团队实际开发环境需求所开发。

2 技术综述

2.1 微服务架构概述

微服务架构是一种架构模式，它提倡一个庞大的应用应该由一组较为细粒度的服务组成，通过服务之间的彼此调用和集成来完成具体的业务逻辑功能。这些服务是围绕业务组织的，而不是预定义的组件[12]。微服务作为面向服务架构（SOA）[13]的一种特定实现方法，就如 XP 是敏捷软件开发的具体方法一样[14]，本质上它依然是 SOA 的一种实践方式。

微服务架构的主要特点如下：

1) 组件化服务：即服务的开发标准组件化，在微服务架构下对服务的开发以组件为标准进行接口和功能的设计，实现功能单一，调用方便的特点，减少对其他功能和模块的依赖，同时增强自身的内聚性。
2) 按业务组织团队：以业务需求对团队进行组织和划分，可以实现服务在同一团队中完成设计、研发、测试、部署等环节，有效地提升了研发效率和资源的利用率。
3) 微自治：服务本身要实现局部自治，对该服务进行打包、部署、升级和回滚等操作都不应该影响已发布系统的其他功能和整体工作。

2.2 SpringBoot 开源框架

Spring Boot 是由 Pivotal 团队为了简化 Spring 框架而开发的开源开发框架[15]。最突出的特点是 Spring Boot 通过使用 Java 中注解的配置方式，大大简化了 Spring 框架中使用各种配置文件进行配置的操作。另外 Spring Boot 能够集成大量的框架，解决了之前一直被人所诟病的版本依赖和稳定性问题。

Spring Boot 有以下几个技术特点[16]：

1. 可以方便地选配、组装、生成一个独立的开发初始项目；
2. 由于内置了服务器，所以可以打包成 jar 包；
3. 使用 starter POMs 的配置方式大幅度地简化了 Maven 的配置。

因为拥有众多的优点，同时还有着从 Spring 继承而来的两大特性：依赖注入

（IOC）[17]和面向切面编程（AOP）[18]，SpringBoot 在快速应用开发领域占据了统领性的地位。

SpringBoot 开发的应用 SpringApplication 处理请求加载的流程图如图 2-1 所示。

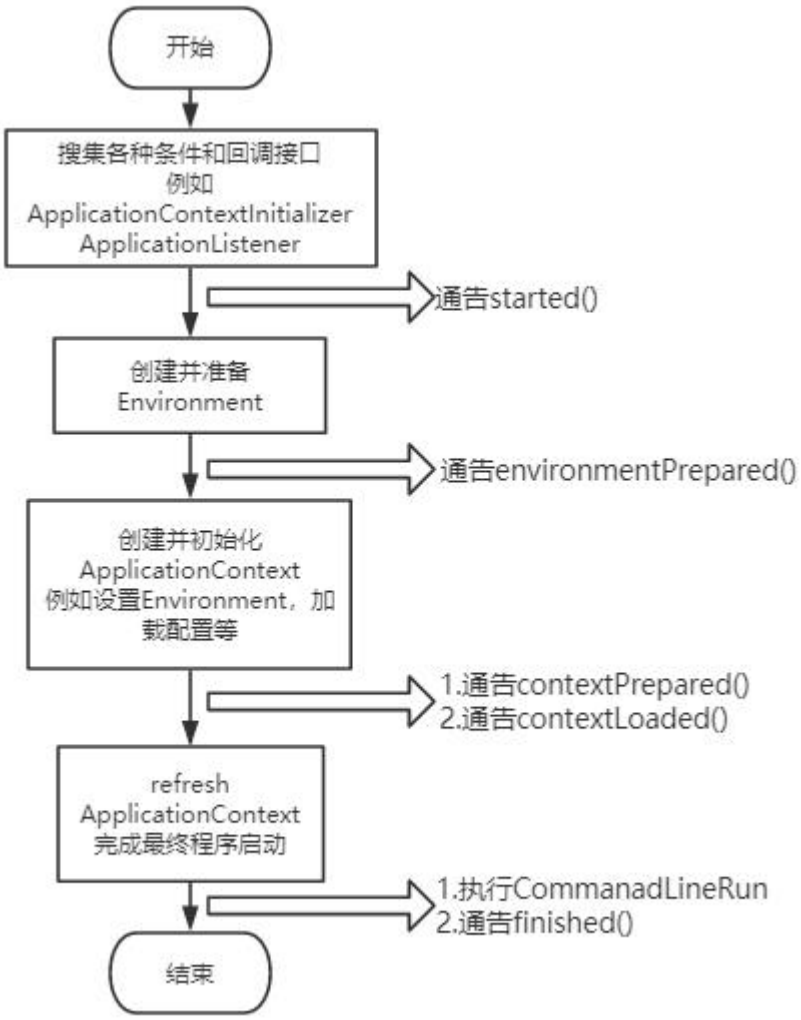

图 2-1 SpringBoot 处理请求加载

Spring 框架的开发效率因为 Spring Boot 的出现而大大提升。Spring 自 2002 年发布以来，通过不断发展，逐渐成为企业应用开发领域最流行的基础框架之一[19]。而且 Spring 除了可以用于服务层的开发之外，它简易测试和松耦合的特性在任何 Java 应用程序中都带来了益处[20]。

2.3 MyBatis 开源框架

Mybatis 原来是 Apache 公司的一个面向广大用户和企业而研究的开源项目“ibatis”[21]，在加入 Google Code 之后正式更名为 Mybatis。MyBatis 是一个持久层框架，它集成了 SQL 查询、存储以及高级映射的功能[22]。相比于原始的 JDBC

操作数据库，MyBatis 可以使用 XML 或注解的方式对配置和 Mapper 文件进行处理，十分便捷高效地完成 POJO（Plain Old Java Object）和数据库记录之间的映射操作，同时还能使开发者避免如加载驱动、建立连接等一系列琐碎而没有技术含量的操作[23]。

MyBatis 主要包含 DAO 组件和 SQL Map 组件，其架构如图 2-2 所示[24]。

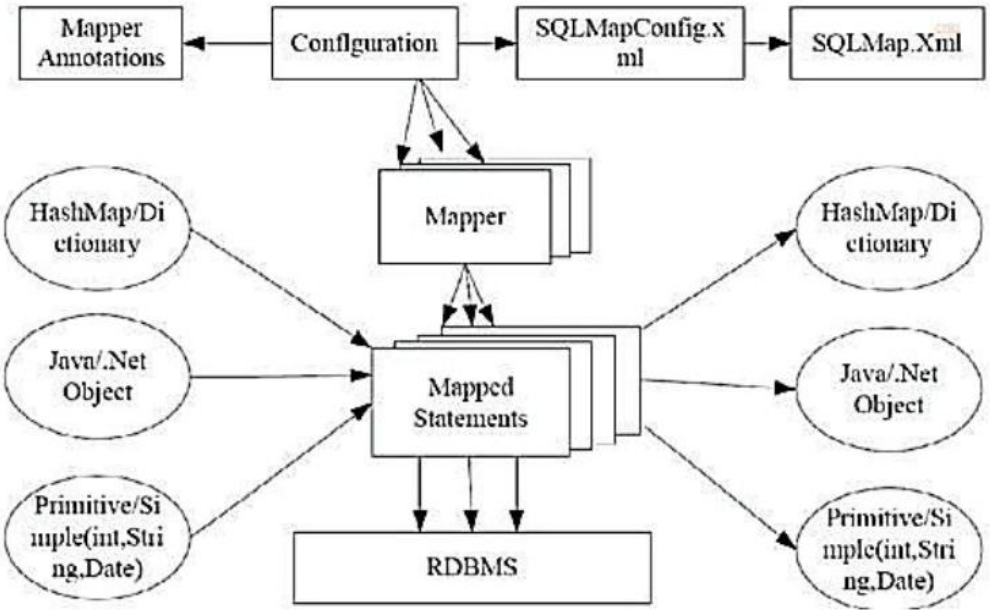

图 2-2 MyBatis 架构图

Mybatis 的 ORM 自动生成方式代替了几乎所有的 JDBC 代码和属性配置。Mybatis 主要由三层功能组成[25]：

1)API 接口层：开发者可以通过自动生成的 JDBC 封装代码向外界提供接口，同时也可以通过本地 API 对数据库直接进行操作和处理。

2)数据处理层：拥有具体的接口实例化 JDBC 代码，包括具体的 SQL 查找、解析文件等功能。它负责将调用的 API 请求转化成具体的代码，而后由代码完成数据库操作。

3)基础支撑层：负责框架最基础的如管理数据库连接、管理数据库事务、加载配置等功能，该部分将这些功能组合成一个基本组件以供上层调用。

MyBatis 的功能流程如图 2-3 所示。

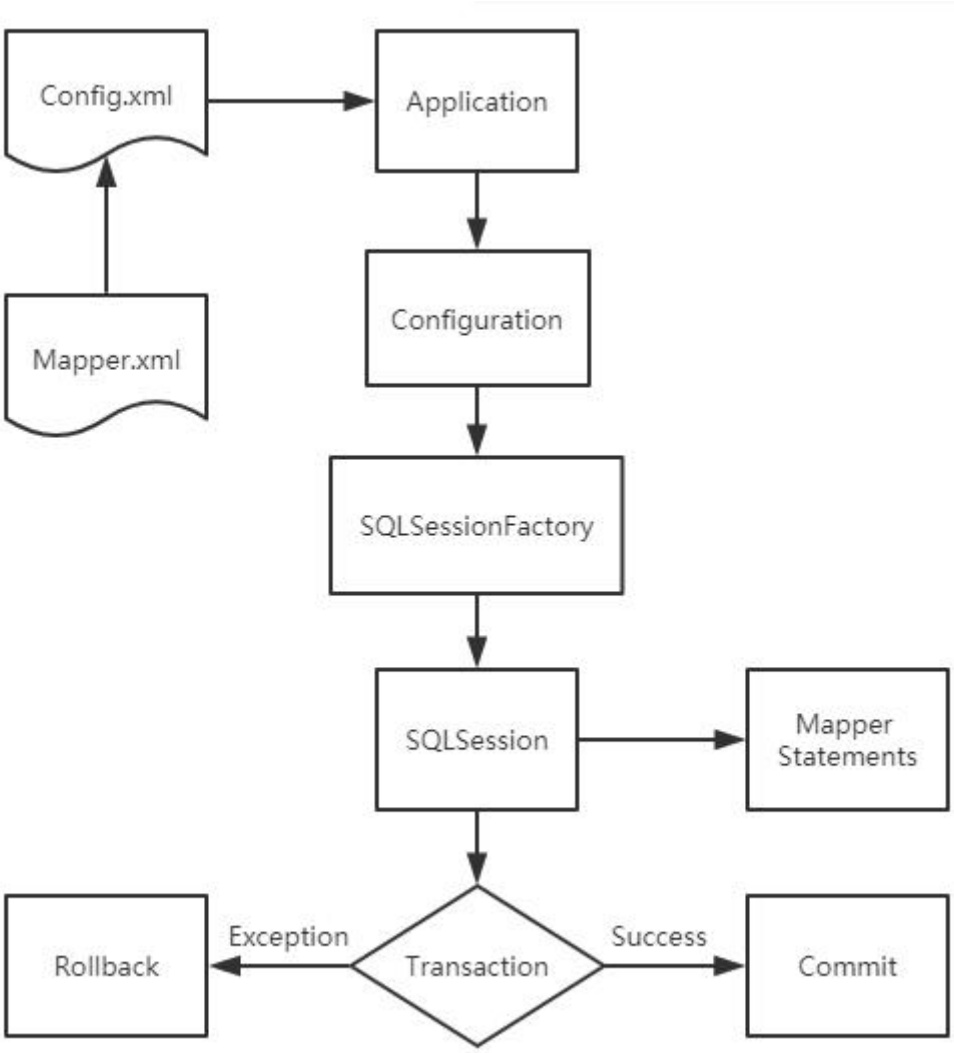

图 2-3 MyBatis 功能流程图

2.4 Netty 开源框架

对 Netty 进行说明介绍离不开 NIO 通信这个概念，因此本节将先介绍 NIO 通信，在此之后再对 Netty 进行分析。

2.4.1 NIO 通信

在大型的企业架构中，服务器程序普遍应用的是核心技术是并发技术的客户/服务器模式，这种服务器需要同时应对多个客户请求并且为它们提供服务[26]。因为每个连接都需要一个用来处理输入输出的线程，所以此种服务器程序会存在性能低下和缺乏可伸缩性的问题。而非阻塞 I/O 机制（NIO）的引入就是为了解决这

个问题[27]。与原本的阻塞 I/O 机制不同的是，NIO 在操作一个非阻塞的连接时，调用会立即返回，这就使一个线程可以同时管理多个连接，从而使资源使用率得到很大的提升[28]。

NIO 并不为了应对每个客户端请求而使用多线程，而是通过多线程来使得多个 CPU 的处理能力得到充分利用，从而提高服务能力。基于 NIO 的多线程服务器通常使用 Reactor 模式来实现，而该模式则是基于选择器(Selector)进行开发的[29]，具体结构如图 2-4 所示[30]。

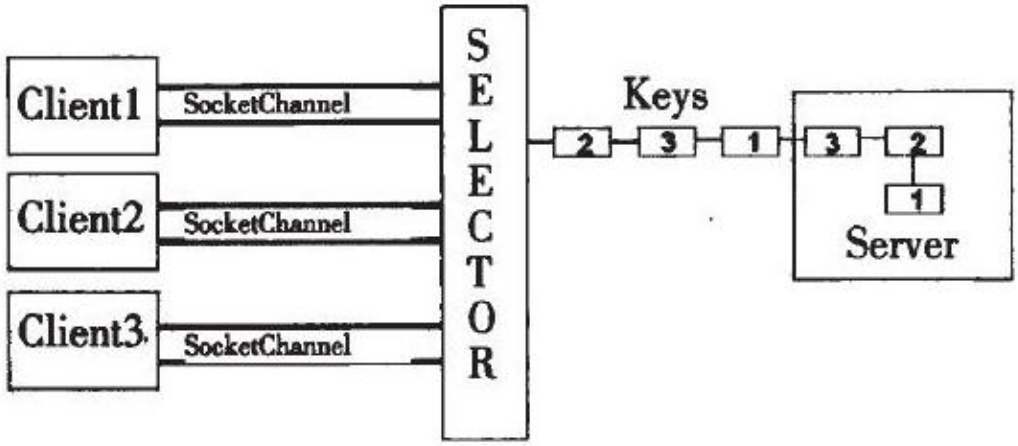

图 2-4 NIO 系统结构

2.4.2 Netty 框架

Netty 是一个高性能的、异步的、事件驱动的，基于 Java NIO 实现的客户端服务端模式的网络框架[31]。它支持包括 TCP，UDP，HTTP 和 FTP 等众多网络传输协议，并且 Netty 中与 I/O 相关的所有操作都是采取异步非阻塞的方式实现的。Netty 的逻辑架构图如图 2-5 所示。

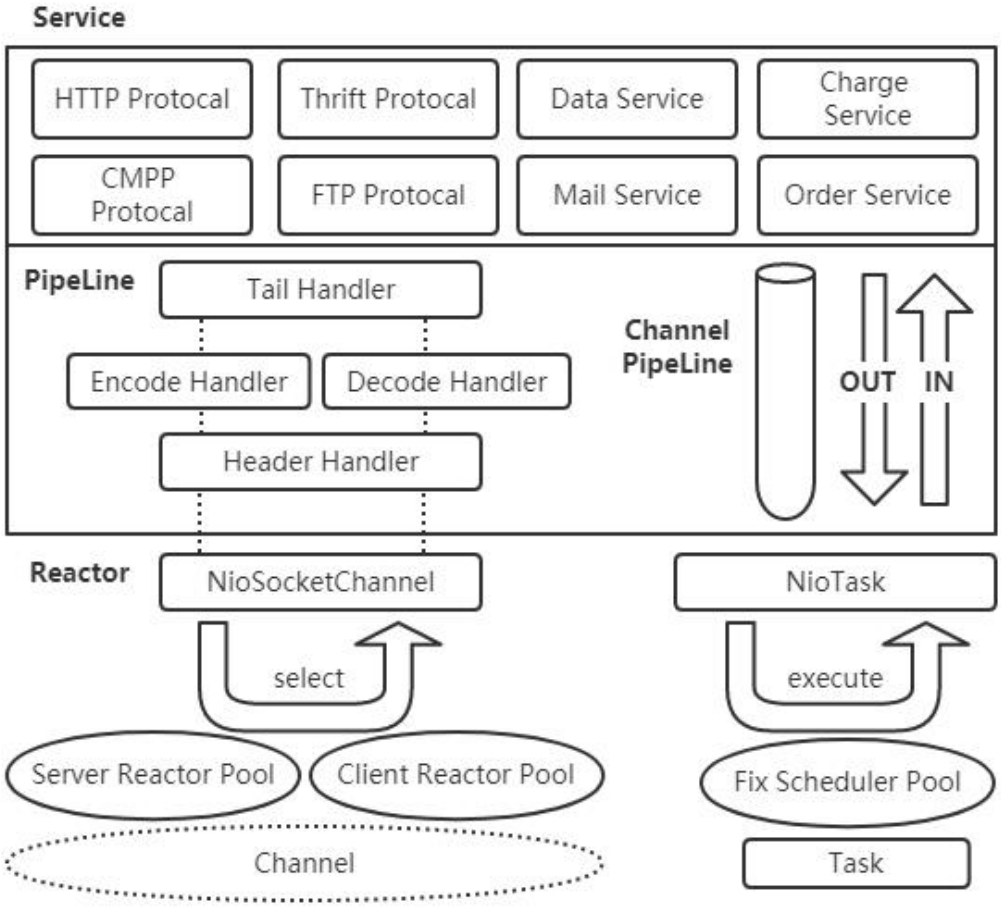

图 2-5 Netty 逻辑结构图

其中 Reactor 通信调度层主要负责监听网络的读写和连接操作，该层会将网络层的数据读取到内存缓冲区中，然后触发各种网络事件，并且将这些事件触发到 PipeLine 中以进行后续的处理。ChannelPipeLine 被称为职责链，它将事件的处理视为职责，该部分负责事件的有序传播。通过配置不同的职责链相关处理类，可以做到有选择地监听和处理事件。Service ChannelHandler，也叫做业务逻辑编排层，该层通常有两类：一类是用于特定协议的协议插件，还有一类是业务逻辑编排。

Netty 框架的优点如下[32]：

1) API 使用简单，通过操纵几个接口方法，就可以完成多个参数的调整。
2) 预置了多种协议栈和编解码器。
3) 采用异步非阻塞通信模式，并且因为使用了多路复用器，Netty 框架避免了线程膨胀的问题。
4) 高效的 I/O 线程模型，提供了多种线程模型。
5) 高性能的序列化框架，Netty 默认使用 Protobuf 来完成序列化操作，同时也支

持用户自定义序列化框架。

2.5 Spring MVC

MVC (Model View Controller)是模型、视图、控制器的缩写。M 是指业务模型，在其中会存放数据和对应的实体类，V 是指用户界面，负责渲染页面显示数据，C 是控制器，负责连接前面两者，它会根据用户的请求，将相应的数据提出并选择相应的视图进行展示。MVC 的目的是让开发人员的分工更为明确，让处理数据的研发人员只需要处理数据，让处理视图的研发人员只需要设计视图。Spring MVC 是基于 Spring 的 MVC 框架，利用 MVC 的设计方式使得开发更加灵活、高效。

在收到用户的请求时，Spring MVC 会调用前端拦截器来对其进行处理。前端拦截器会解析 url，然后把请求转发到对应的控制器，控制器在完成对请求的解析并得到请求参数之后，会调用 service 层完成业务逻辑，并返回模型和视图对象。视图对象和模型会被视图解析器处理，经过处理之后会得到最终的视图结果，该结果会被返回给请求的用户。具体流程图如图 2-6 所示。

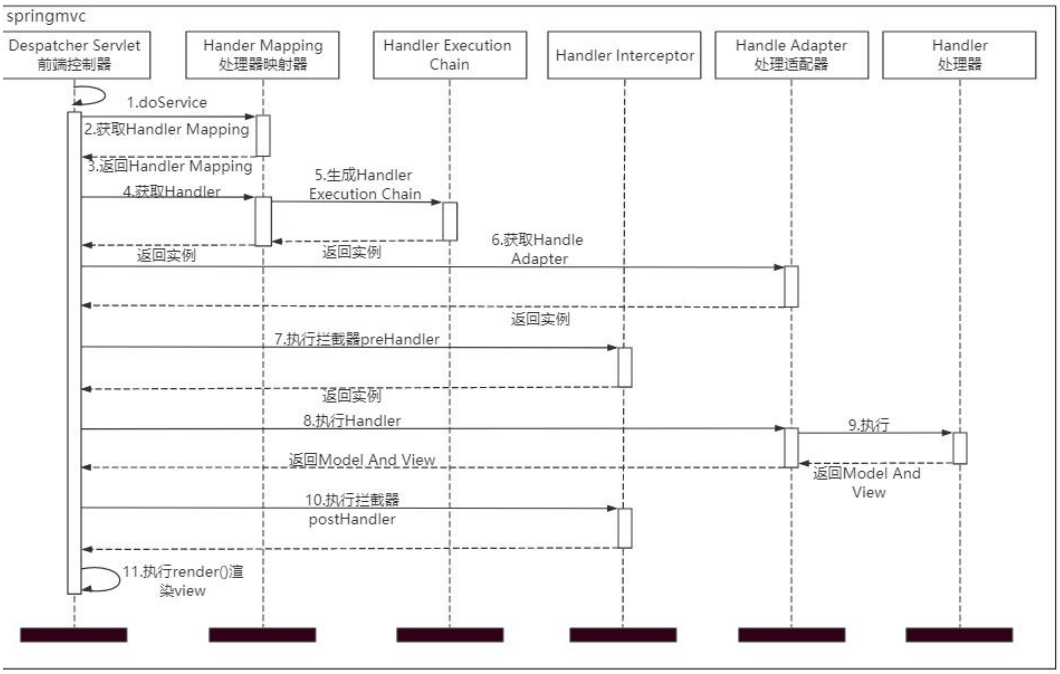

图 2-6 SpringMVC 流程图

2.6 本章小结

本章节对实现 MMS 系统过程中所使用的技术与理论进行了介绍和分析。微服

务架构是 MMS 系统的设计基础架构，整体的开发初衷和设计思路都是在微服务架构的基础上实现的。Spring Boot 框架在系统中用于控制中心的接受服务调用模块和查询页面模块的开发，其中使用了 MVC 的设计架构和思想，而 MyBatis 作为一个持久层的框架在系统中帮助完成了操作数据库的功能。Netty 作为一款优秀的 NIO 框架，负责实现了网络通信的功能，其在系统中处于十分关键的地位。

3 微服务管理系统的分析与设计

3.1 系统需求分析

3.1.1 系统用例模型图

根据功能需求，分析得到在 MMS 系统中，一共存在 3 个参与者：管理员，服务调用者，服务提供者，系统的用例模型图如图 3-1 所示。

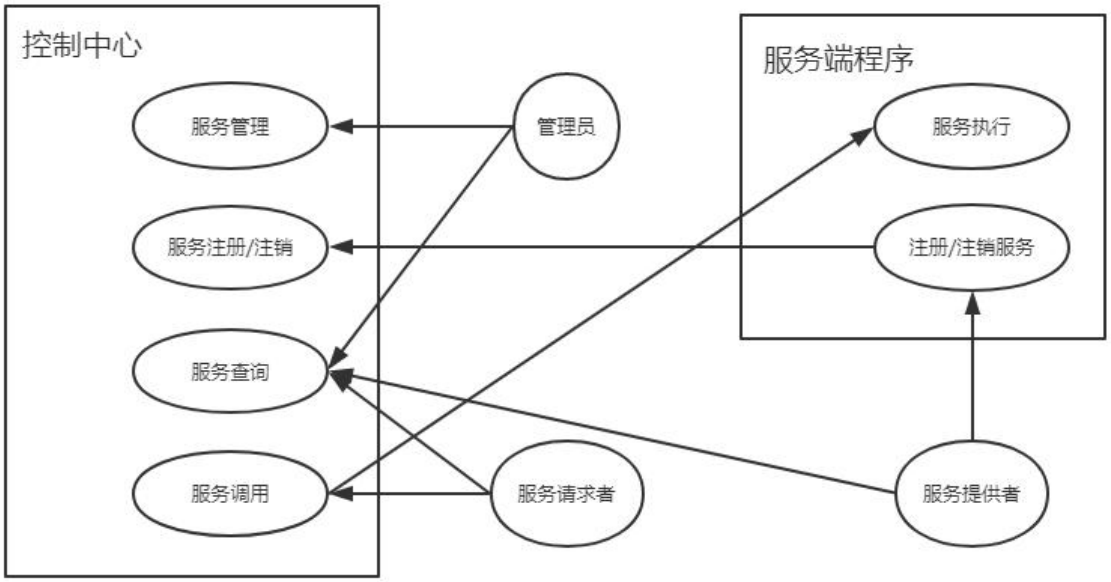

图 3-1 用例模型图

由用例图可知，MMS 系统中一共有 6 个用例，其中服务查询为公共开放用例，服务管理为管理员权限用例，服务注册/注销和服务执行为系统内部调用用例。

3.1.2 控制中心用例描述

用例 1：服务管理，即对已注册服务进行管理，包括对服务的删除、更新、更改操作。具体描述如表 3-1 所示。

表 3-1 服务管理用例描述表

ID	Center1
名称	服务管理
参与者	管理员，目标是对已注册的服务进行管理。
描述	该用例描述管理员对已注册服务的管理，包括删除、更新、更改操作。

表 3-1（续）

前置条件	服务已经注册
后置条件	管理服务，改变运行状态等信息。
正常流程	1. 进入已注册服务的信息管理部分。 2. 对服务进行更改。

用例 2：服务注册/注销，即对服务端程序发送的服务注册/注销请求进行处理。具体描述如表 3-2 所示。

表 3-2 服务注册/注销用例描述图

ID	Center2
名称	服务注册/注销
参与者	服务端程序，目标是注册服务。
描述	该用例描述控制中心处理服务注册/注销请求，包括对服务的新增和删除。
触发条件	控制中心收到服务注册/注销请求。
前置条件	控制中心服务注册/注销服务端处于启动状态。
后置条件	新增已注册服务或删除已注册服务，得到注册成功结果后服务端启动。
正常流程	1. 服务端程序发送服务注册/注销请求。 2. 控制中心处理请求，新增已注册服务或删除已注册服务。 3. 控制中心返回结果。 4. 服务端得到成功结果后启动。

用例 3：服务查询，即查询已注册服务信息。具体描述如表 3-3 所示。

表 3-3 服务查询用例描述表

ID	Center3
名称	服务查询
参与者	管理员，服务请求者，服务提供者
描述	该用例描述控制中心提供的服务查询功能。
触发条件	参与者打开查询页面。
前置条件	控制中心的查询页面功能处于启动状态，有服务已注册。
后置条件	参与者查询到已注册服务的信息。

表 3-3（续）

正常流程	1. 打开已注册服务查询页面。 2. 查看已注册服务信息，包括服务地址，服务类型，服务运行状态等。 3. 根据结果做出相应行动，如服务请求者请求某一服务，管理员更改某一服务等。

用例 4：服务调用，即服务请求者对控制中心发起对已注册服务的调用请求。具体描述如表 3-4 所示。

表 3-4 服务调用用例描述表

ID	Center4
名称	服务调用
参与者	服务请求者。
描述	该用例描述服务请求者向控制中心发起服务调用请求。
触发条件	服务请求者发起调用请求。
前置条件	服务请求者构造符合规范的请求。
后置条件	服务请求者得到服务调用结果。
正常流程	1. 服务请求者发起调用请求。 2. 控制中心处理请求，解析得到执行命令。 3. 控制中心将执行命令发送给服务端程序，得到结果。 4. 控制中心将结果返回给服务请求者。

3.1.3 服务端程序用例描述

用例 1：服务执行，即服务端程序执行从控制中心收到的执行命令。具体描述如表 3-5 所示。

表 3-5 服务执行用例描述表

ID	Server1
名称	服务执行
参与者	控制中心。
描述	该用例描述服务端程序执行从控制中心收到的执行命令。
触发条件	服务端程序收到来自控制中心的执行命令。
前置条件	服务端程序处于启动状态，控制中心发送执行命令。

表 3-5（续）

后置条件	服务端程序返回服务调用结果。
正常流程	1. 控制中心发送执行命令。 2. 服务端程序执行收到的命令。 3. 服务端程序返回服务执行结果。

用例 2：注册/注销服务，即服务提供者对控制中心发起注册/注销服务的请求。具体描述如表 3-6 所示。

表 3-6 注册/注销服务用例描述表

ID	Server2
名称	注册/注销服务
参与者	服务提供者。
描述	该用例描述服务提供者向控制中心发起注册/注销服务请求。
触发条件	服务提供者运行服务端程序。
前置条件	服务提供者提供了符合规范的配置文件。
后置条件	服务提供者得到注册/注销服务结果。
正常流程	1. 服务提供者运行服务端程序。 2. 服务端程序根据配置信息发送注册/注销服务请求。 3. 服务端程序得到控制中心返回的结果，如果注册成功则进入启动状态。

3.2 系统结构设计

MMS 系统的设计目标是将团队中原本混乱的服务调用过程规范地管理起来，使服务调用变得更加便捷，从而提高开发效率。出于这一目的，并结合实习的工作内容，MMS 系统选择以 HTTP 访问作为服务调用方的请求方式，同时选择 SpringBoot 这一款能大大提升 Web 应用开发的优秀的开发框架来作为系统的基础框架。

在 MMS 系统的设计中，有三个与服务调用相关的部分，它们分别是服务调用方，服务控制中心，服务端程序，它们三者在服务调用过程中所处的地位和相互之间的关系如图 3-2 所示：

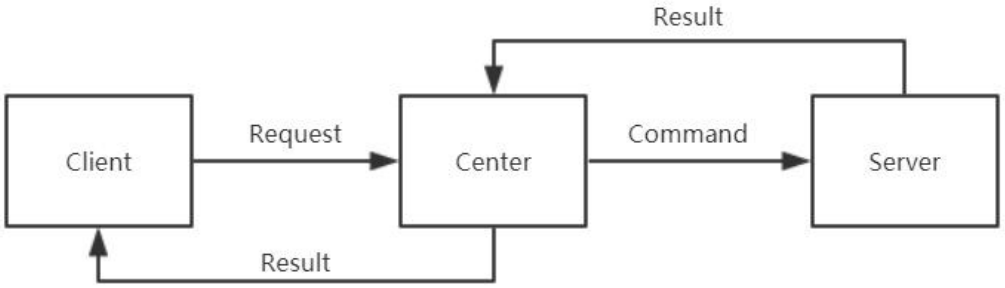

图 3-2 三部分关系图

其中服务调用方与服务控制中心之间的通信采用 HTTP 请求的方式实现，服务控制中心与服务端程序之间的通信采用 Java 的 NIO 方式实现。

服务控制中心与服务端程序之间的通信除了服务调用相关的通信之外，还有与服务管理相关的通信，包括服务注册与注销等功能。

另外，对于服务注册信息的存储，MMS 系统采用数据库存储的方式来对这些信息进行管理，这样的存储方式将重要信息的存储与控制中心分离，避免了控制中心的意外崩溃而导致服务注册信息丢失。同时，系统还要求实现查询页面模块，该模块要求能够让开发者对于已注册服务进行快捷查询，采用数据库存储信息的方式可以让该模块简单高效地实现目标功能。

最后，作为使用 MMS 系统的服务提供者，需要提供符合系统要求的服务调用接口，具体为调用参数的设计，对可以为空的参数必须提供默认值，同时要求提供具有可读性的调用错误信息。此外，还有一些必要的配置文件需要提供，具体为调用该服务所必需的 IP 地址和端口地址，以及调用参数的具体说明和返回结果的具体说明。

3.3 系统工作流程

前面的内容介绍了 MMS 系统的整体设计，本小节将在此基础上对 MMS 系统的工作流程进行介绍。

首先，在服务调用方调用某个服务之前，该服务需要进行注册。注册功能由服务端程序实现，服务端程序通过扫描服务提供者编写的符合系统规范的配置文件，从而获取到该服务相关的 IP 地址、监听端口、调用参数说明、返回结果说明

等信息，然后通过 Java 的 NIO 通信方式将相关信息发送到控制中心，控制中心收到注册请求后对请求进行解析，完成解析后将相关信息写入存储服务注册信息的数据库，根据写入结果向服务端程序返回注册是否成功的信息，至此服务注册流程结束。示意图如图 3-3 所示。

服务注销流程与注册流程相同，只是服务端程序发送的参数不同而已，故不再赘述。

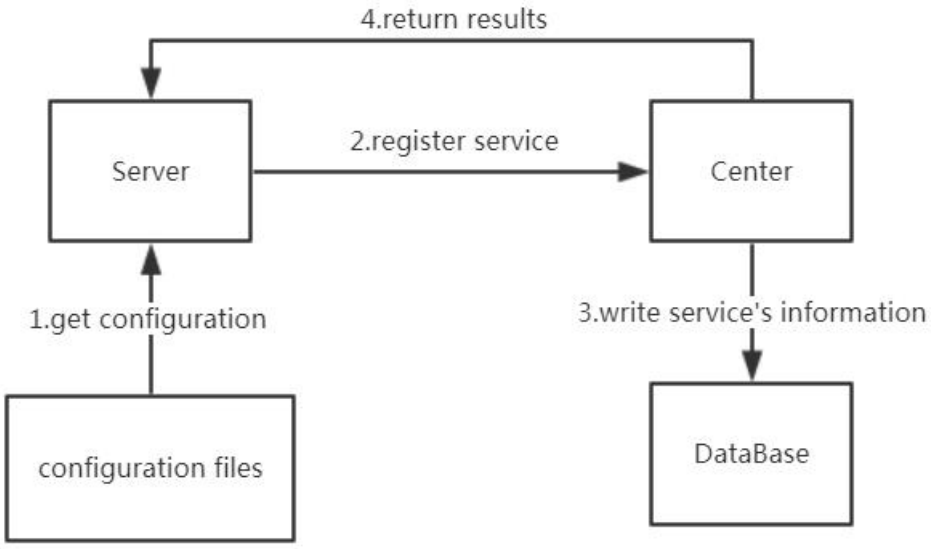

图 3-3 服务注册示意图

完成了服务注册之后，服务调用者就可以通过系统提供的 Web 页面查询到已注册的服务信息，具体包括服务名，服务类型（python 还是 Java），服务 IP 和服务端口，服务运行状态，调用参数说明和调用结果说明。服务调用者得知上述信息后就可以按照系统规范构造需要发送请求的 url，并且根据“调用结果说明”这一信息对得到的结果进行解析处理。

从服务调用方发出的调用请求将最先到达服务控制中心，服务控制中心对请求信息进行处理，根据 url 确定被调用的服务和服务类型（是 python 还是 Java），通过请求中包含的参数确定被调用服务的 IP 和端口，以及传递给被调用服务的参数内容，然后服务控制中心会查询已注册服务的信息来对调用进行准确性的检查，在确认服务调用正确后，利用上述内容构造调用命令，接着发送构造好的调用命令至服务端程序，由服务端程序执行命令并返回结果。至此服务调用流程结束，整体流程如图 3-4 所示。

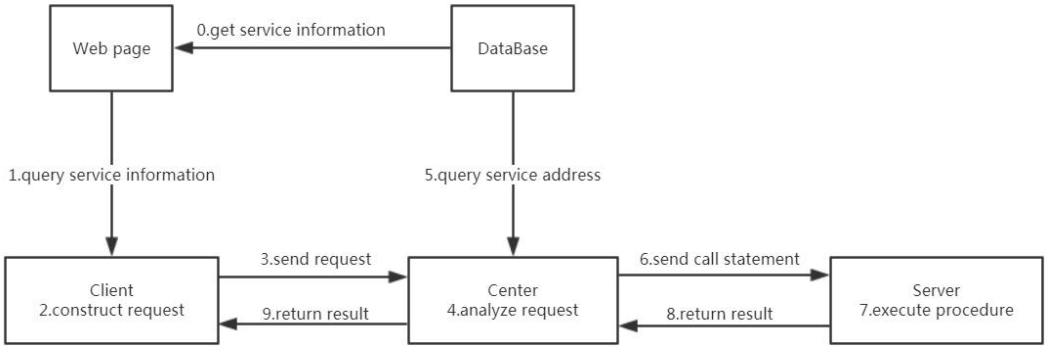

图 3-4 服务调用示意图

3.4 系统规范与协议设计

在实现 MMS 系统的过程中，为了能正常地实现的功能，制定了一些规范和协议，同时为了让使用者能方便地使用 MMS 系统，在设计和实现的过程中本着简洁高效的原则制定了它们。本小节中会对这些规范和协议进行介绍和说明。

首先是服务调用者，服务调用者在使用MMS系统时需要构造符合规范的POST请求，具体规范如表 3-7 所示。

表 3-7 服务调用者请求规范

	规范格式	具体说明
url	http://{IP}:{PORT}/{TYPE}/call	IP：控制中心所在 IP。 PORT：控制中心所监听的端口。 TYPE：所调用的服务类型，具体值为 python 或 java。

请求参数	{
	“ip”: {IP},
	“port”: {PORT},
	“name”: {NAME},
	“param{N}”: {PARAM}(可为空)
	}
		IP：被调用服务所在 IP。
		PORT：被调用服务所监听的端口。
		NAME：被调用服务的服务名称。
		N：从 1 开始的正整数，表示参数的序号。
		PARAM：参数的值，需要与被调用服务提供的参数顺序对应。

接着是服务提供者，服务提供者需要提供三个配置文件，分别是服务本身相关信息的文件——ServiseConfig.txt，调用参数的说明文件——ParamConfig.txt，调用返回结果的说明文件——ResultConfig.txt，考虑到服务提供者可能在不同的系统环

境上部署服务，所以配置文件的类型采用了 txt 文件。在配置文件中，配置参数以一行一个的方式填写，具体规范如表 3-8 所示。

表 3-8 服务提供者配置文件规范

<table>
<tr><td></td><td colspan="2">ServiseConfig.txt</td><td colspan="2">ParamConfig.txt</td></tr>
<tr><td>行数</td><td>规范格式</td><td>具体说明</td><td>规范格式</td><td>具体说明</td></tr>
<tr><td>第一行</td><td>{NAME}</td><td>NAME:服务名称。</td><td>Param1—{EXPLANATION}</td><td rowspan="6">具体行数视参数个数而定。EXPLANATION：对参数的解释，需要说明参数的作用，应该传入哪些值，是否可以为空，默认值等必要信息。</td></tr>
<tr><td>第二行</td><td>{TYPE}</td><td>TYPE：服务类型，具体值为 python 或 java。</td><td>Param2—{EXPLANATION}</td></tr>
<tr><td>第三行</td><td>{RUN/STOP}</td><td>RUN/STOP：注册还是注销，具体值为 register 或 stop。</td><td>Param3—{EXPLANATION}</td></tr>
<tr><td>第四行</td><td>{IP}</td><td>IP：服务所在 IP。</td><td>……</td></tr>
<tr><td>第五行</td><td>{PORT}</td><td>PORT：服务所在端口。</td><td>……</td></tr>
</table>

<table>
<tr><td colspan="3">ResultConfig.txt</td></tr>
<tr><td>行数</td><td>规范格式</td><td>具体说明</td></tr>
<tr><td>第一行</td><td>{RESULTNAME}—{EXPLANATION}</td><td rowspan="5">具体行数视结果个数而定。RESULTNAME：返回结果的名称，其中不能出现“—”。EXPLANATION：对返回结果的解释说明。</td></tr>
<tr><td>第二行</td><td>{RESULTNAME}—{EXPLANATION}</td></tr>
<tr><td>第三行</td><td>{RESULTNAME}—{EXPLANATION}</td></tr>
<tr><td>第四行</td><td>……</td></tr>
<tr><td>第五行</td><td>……</td></tr>
</table>

最后是 MMS 系统内部的通信协议，在某些通信过程中使用的消息体类型为 Java 的 Map 类型，这是一个 Key-Value 类型，下面对使用的 Key-Value 要求进行说明。

服务注册环节，服务端程序向控制中心发送的注册/注销服务请求的 Key-Value 要求如表 3-9 所示。

表 3-9 服务注册/注销请求说明

Key	Value 类型	说明
ip	String	服务所在 IP。

表 3-9（续）

name	String	服务名称。
type	String	服务类型。
port	String	服务所在端口。
paramFormat	List<String>	由参数说明组成的 List。
resultFormat	List<String>	由结果说明组成的 List。
ifregister	String	表明是注册还是注销。

控制中心执行操作后向服务端程序直接返回表示结果的字符串信息。

控制中心在向服务端程序发送服务的执行命令时是直接发送已经构造好的执行命令字符串，而服务端程序向控制中心返回执行结果时会使用 Key-Value，该处的 Key-Value 比较简单，具体如表 3-10 所示。

表 3-10 服务调用返回结果说明

Key	Value 类型	说明
{RESULTNAME}	String	此处的 Key 值即为服务配置文件中的结果名称，Value 值即为对应的结果值。
……	……	

3.5 本章小结

本章节主要对 MMS 系统进行了结合实际工作内容的需求分析，得到了系统用例并对其进行了描述，在此之后，介绍了 MMS 系统的大致设计结构，同时也介绍了系统的大致工作流程，包括请求是如何产生和发送的，服务是如何注册和被调用的。对整体流程的介绍，也间接地说明了各个模块的设计意义和具体功能，而对于各模块的具体内容将在下一章进行介绍。最后，对系统中使用的规范与协议进行了介绍，这些规范与协议以便捷高效为目标做了尽可能地优化。

4 系统各模块功能及设计

4.1 组件概述

前面介绍了 MMS 系统的主要功能部分为 Center 和 Server，而这两个主要功能部分又由各个具体的功能组件组成，它们的主要关系如图 4-1 所示。

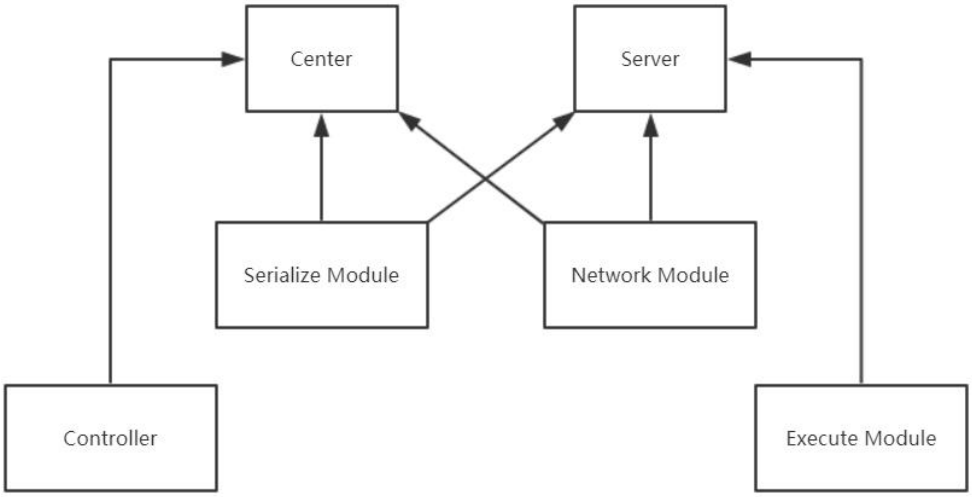

图 4-1 组件调用图

其中 Serialize 组件和 Network 组件是公共组件，同时属于 Center 部分和 Server 部分，而 Network 组件根据具体所属的部分在信息处理的方法上进行了响应的改动。

下面对各个组件进行概述。

Serialize 组件：

该组件的功能是对将要在 NIO 通信中传递的信息进行序列化和反序列化的处理。首先介绍一下序列化的概念，Java 序列化是指把 Java 对象转换为字节序列的过程，而 Java 反序列化是指把字节序列恢复为 Java 对象的过程。通过对网络信息传播的方式我们可以得知，当两个进程进行远程通信时，数据会以二进制字节的形式在网络上传播。因此，为了在 Center 部分和 Server 部分之间进行消息传递，需要将 Java 对象进行序列化与反序列化的处理。具体来说，发送方需要将 Java 对象转换为字节序列，而接收方需要将字节序列进行反序列化处理从而得到相应的

Java 对象。Java 本身提供了序列化的方案，但是它具有三个明显的缺点，分别是：

1. 无法跨语言；
2. 序列化后的码流太大；
3. 序列化性能太低。

基于上述三个缺点，在实际开发中通常不会选择 Java 序列化作为编解码的框架。而实际上，业界中有很多优秀的序列化框架可供选择，在 MMS 系统中选择使用 Protostuff 来帮助完成信息序列化的功能。Serialize 组件提供 serialize 方法和 deserialize 方法来完成信息的处理。

Network 组件：

该组件负责 Center 部分和 Server 部分的网络通信，通信方式为 NIO。网络通信有主要四种 I/O 模型，这几种模型的功能和特性对比如下表 4-1 所示：

表 4-1 几种 I/O 模型的功能和特性对比

	同步阻塞 I/O（BIO）	伪异步 I/O	非阻塞 I/O（NIO）	异步 I/O（AIO）
I/O 线程：客户端个数	1:1	M：N（M 可以小于 N）	1：M（1 个线程处理多个连接）	0：M（不需要线程，被动回调）
I/O 类型（阻塞）	阻塞 I/O	阻塞 I/O	非阻塞 I/O	非阻塞 I/O
I/O 类型（同步）	同步 I/O	同步 I/O	同步 I/O（I/O 多路复用）	异步 I/O
API 使用难度	简单	简单	非常复杂	复杂
调试难度	简单	简单	复杂	复杂
可靠性	非常差	差	高	高
吞吐量	低	中	高	高

通过对比以上四种 I/O 模型和考虑系统的实际应用场景，决定选择 NIO 作为系统中的通信方式。同样的，由于 Java 本身的 NIO 方案拥有以下几个缺点而不选择使用：

1. NIO 的类库和 API 繁杂；
2. 需要额外技能做铺垫；
3. 可靠性差；
4. 具有不少 BUG。

因此在 MMS 系统中选择使用业界最流行的 NIO 框架之一 Netty 框架来帮助完成网络通信功能。在系统中，Network 组件负责网络通信中的信息发送和信息接受，同时会使用 Serialize 组件来进行信息的序列化与反序列化操作。

Controller 组件：

该组件作为 SpringBoot 框架中的一部分，主要负责对服务调用请求进行解析，然后将解析结果构造成执行命令通过 Network 组件发送到 Server 部分。

Execute 组件：

该组件负责执行服务。在 Server 部分收到 Controller 组件发送的执行命令时，将调用该组件执行命令，同时该组件会将返回结果按服务提供者提供的结果格式处理成对应的 HashMap 对象，然后将结果通过 Network 组件返回给 Center 部分。

4.2 Serialize 组件

首先是对公共组件 Serialize 组件的说明，该组件的功能是对网络通信中的信息进行序列化和反序列化操作。

在前面的组件概述中提到，Serialize 组件选择 Protostuff 框架来进行对 POJO 的编解码操作，并且提供了 serialize 方法和 deserialize 方法来完成对信息的处理。这里通过对组件的 serialize 方法和 deserialize 方法的分析来说明本组件。

serialize 方法的方法签名如下：

public static <T> byte[] serialize(T obj)

其中 obj 参数为想要对其进行序列化的 Java 对象。在传入参数之后，serialize 方法会先利用 Java 的反射机制获得一个 obj 的 Class 类对象 cls，接着会根据 cls 对象得到一个 Schema 类的对象 schema。

Schema 类是 Protostuff 框架中一个用于存储如何进行序列化操作的相关信息的类，Schema 包含：

1. 对象进行序列化的逻辑；
2. 对象进行反序列化的逻辑；
3. 对象必填字段的验证；
4. 对象字段名称到字段编号的映射；
5. 对象的实例化。

在得到对象 schema 和对象 cls 后，生成一个 LinkedBuffer 类的实例对象 buffer，

该对象是一块默认大小的缓存空间，之后调用 Protostuff 框架的序列化方法，将 schema，cls 和 buffer 传入该方法中，就可以返回得到一个经过序列化处理后的字节数组。

接着是 deserialize 方法，该方法的方法签名如下：

public static <T> T deserialize(byte[] data, Class<T> cls)

其中 data 参数为想要进行反序列化的字节数组，cls 参数为反序列化的目标类型。在传入参数之后，deserialize 方法会先通过反射机制得到一个 cls 类的实例对象 obj，同样的也会生成一个 Schema 类的对象 schema，之后以 data，obj 和 schema 为参数调用 Protostuff 框架的反序列化方法，返回得到一个反序列化后生成的 Java 对象。

以上就是 Serialize 组件的具体实现。

4.3 Network 组件

接着是对公共组件 Network 组件的说明，该组件的功能是负责各个程序之间的网络 NIO 通信。

在前面的组件概述中提到，该组件选择 Netty 框架来帮助实现通信过程，在实际通信过程中存在客户端和服务器两个部分，但两个部分的实现方法没有太大差异，所以下面以服务器部分为例来介绍 Network 组件。

服务端程序的启动通信方法为 bind 方法，该方法的方法签名如下：

public void bind(int port)

其中 port 参数为想要监听的端口号。在传入参数被调用之后，bind 方法会创建两个 NioEventLoopGroup 线程组实例。NioEventLoopGroup 线程组包含了一组 NIO 线程用于处理网络事件，在这里，两个线程组分别用于处理客户端的连接和进行 SocketChannel 的网络读写。在创建两个线程组实例之后，bind 方法会创建一个 ServerBootstrap 对象 bootstrap，该对象是 Netty 框架的一个用于启动 NIO 服务端的辅助启动类，将两个线程组作为参数传入对象 bootstrap，同时设置创建的 Channel 类型，配置 TCP 参数，最后还需要一个负责处理具体事件的处理类 ChildChannelHandler，将该类绑定后，通信过程中的 I/O 事件都会由它来处理，至此完成对象 bootstrap 的配置，接着调用它的 bind 方法绑定监听端口，完成绑定操作后会返回得到一个 ChannelFuture 类的实例对象 future，最后调用 future 的 sync

方法进行挂起监听。具体流程如图 4-2 所示。

图 4-2 服务端启动流程图

现在对处理类 ChildChannelHandler 进行说明，该类是 Netty 框架中一个抽象类的实现类，通过该类的 initChannel 方法可以按顺序地添加需要的处理类。在服务端系统中按序添加了用于处理 TCP 粘包/拆包问题的分隔符解码器，使用 Serialize 组件实现的解码器以及执行服务端功能的处理类。

客户端的具体实现与上述基本一致，唯一不同的是使用了对象 future 的 connect 方法进行连接操作而不是绑定端口操作。

4.4 服务端程序 Server

在使用 MMS 系统时，服务提供者需要将服务端程序和相关配置文件放在其提供的服务的同一目录下，在启动时通过命令行参数的方式传入控制中心 Center 所对应的 IP 和端口以便发送服务注册的请求。从前面系统工作流程小节的内容可以得知，服务端程序的工作流程大致是读取配置文件，发送服务注册请求，挂起等待调用，被调用时执行服务。下面将从服务端程序工作流程的角度来对其进行具体分析。

4.4.1 读取配置文件

前面提到过，服务端程序 Server 需要服务提供者同时提供三个配置文件，现以具体内容如表 4-2 所示的配置文件为例。

表 4-2 配置文件用例

	ServiseConfig.txt	ParamConfig.txt	ResultConfig.txt
实际值		Test（type，ifdebug）	按序返回 Name，Sex，Age
第一行	Tools	Param1—类型，值为“python”或“java”。	Name—教师名称

表 4-2（续）

第二行	python	Param2—是否是调试模式，值为“1”或“0”。	Sex—教师性别
第三行	register		Age—教师年龄
第四行	127.0.0.1		
第五行	8080		

服务端程序 Server 会使用 Java 的 File I/O 相关方法按照系统规范去读取以上配置文件的内容，然后会得到如下信息：

1. 该服务的名称为 Tools，程序类型为 python，处于可调用状态需要被注册，该服务可以通过 127.0.0.1:8080 的地址被调用。

2. 该服务拥有 2 个可传入的参数，这 2 个参数按传入顺序分别用于类型的表述和运行模式的表述。

3. 该服务会返回 3 个运行结果，这 3 个运行结果的名称按顺序分别为 Name，Sex，Age，它们对应的含义为教师名称，教师性别和教师年龄。

这些信息会被存入一个 Map 类型的对象 serviseConfig 中，这个包含着所有服务注册相关信息的对象将会被发送到控制中心以完成服务注册操作。

4.4.2 发送服务注册/注销请求

在得到包含着服务注册相关信息的对象 serviseConfig 后，服务端程序会将其放入一个用于通信的包装类 IOBody 中，然后通过 Network 组件的客户端实现将这个包装类发送出去。

控制中心 Center 会对服务注册请求进行处理，并且会返回处理结果，服务端程序在收到请求之后会对其进行解析，根据注册结果的成功与否会执行开始挂起监听或输出错误信息的操作。

服务注销请求与服务注册请求相同，由配置文件 ServiseConfig.txt 中的参数决定对该服务是进行注册还是注销。

4.4.3 执行服务

在 Network 组件的服务端实现收到控制中心 Center 发来的服务调用命令后，服务端程序会利用 Java 执行外部程序的功能按照命令执行服务，然后对程序在命令

行输出的结果进行读取并按照配置文件 ResultConfig.txt 中描述的内容对结果进行映射，最后得到一个 HashMap 类型的对象 result，同样的，这个对象会被放入包装类 IOBody 中并返回给控制中心 Center。

4.5 控制中心 Center

控制中心 Center 在使用时只需部署到服务器上即可。从前面系统工作流程小节的内容可以得知，控制中心的工作内容大致是处理服务注册和处理服务调用。下面将从这两个功能来对其进行具体分析。

4.5.1 处理服务注册

控制中心中处理服务注册的部分其实也是一个服务端，可以称之为 RegisterServer，其使用 Network 组件的服务端实现进行启动并挂起等待，当收到服务端程序 Server 发来的服务注册请求后，RegisterServer 会将对象 serviseConfig 从包装类 IOBody 中提出，按照对象 serviseConfig 中包含的参数来确定是对服务进行注册操作还是注销操作，如果是注册则将信息存入数据库中，如果是注销则从数据库中删除相关服务的学习，然后将操作结果返回给服务端程序 Server。

开发者可以通过控制中心提供的查询网页查询到已注册服务的相关信息，管理员还可以通过对数据库的查询得到这些信息。

4.5.2 处理服务调用

本系统采用 Web 服务的方式向使用者提供服务调用功能，当使用者按系统要求的规范构造好 http 请求并发送到控制中心后，控制中心会对请求中的参数进行解析和处理，得到服务类型、服务 IP、服务所在端口、服务名称以及调用参数。得到相关信息后，控制中心会从数据库中查询该服务是否存在，如存在则构造包含调用参数的执行命令，并将该执行命令传递给服务请求客户端程序 ServiceRequestClient，该程序会将执行命令发送给处于挂起等待状态的服务端程序 Server，在得到 Server 返回的结果后还有一个判断处理，如果调用失败则将数据库中该服务的运行状态信息改为异常，开发者或者管理者通过查询得到这个信息后可以对服务进行检查并处理。具体流程如图 4-3 所示。

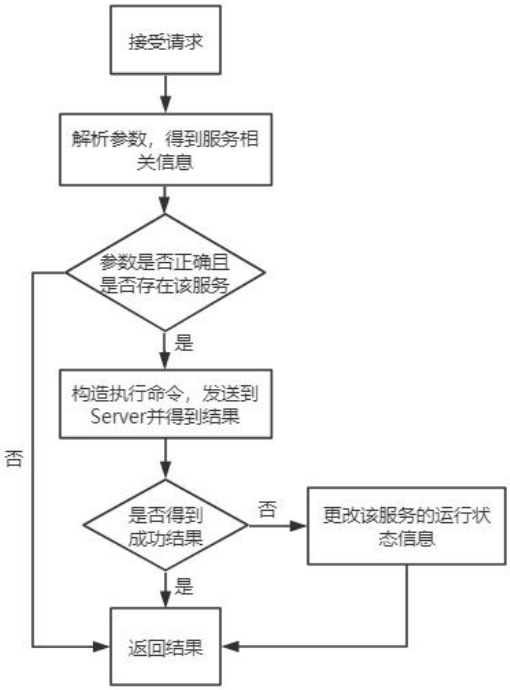

图 4-3 服务调用流程图

控制中心会将结果以 JSON 的格式返回给服务调用者，服务调用者根据查询得到的结果格式说明进行解析即可得到服务结果。

4.6 本章小结

本章介绍和说明了系统各模块的功能与设计。先是对作为公共组件的 Serialize 组件和 Network 组件进行了单独地介绍，然后以工作流程的角度对服务端程序 Server 和控制中心 Center 的实现进行了分析。至此，本文对于 MMS 系统的设计与实现已经基本分析完毕，后面将从实际应用角度对系统功能进行测试。

5 性能测试与分析

5.1 测试环境

为了避免出现具体 IP 地址，测试选择在本机环境上进行，但考虑到系统使用 Java 开发，Java 的跨平台可移植的特性可以保证测试除了具体的 IP 不同之外与实际应用环境没有区别。

测试环境为 Intel Pentium CPU G3240，4GRAM，64bit Windows 7，软件运行环境为 JRE 1.8.0_171-b11，Python 2.7.13。

5.2 测试结果

首先是服务提供者准备服务和相关配置文件，文件结构如下图 5-1 所示：

```
D:\Tool>dir
 驱动器 D 中的卷没有标签。
 卷的序列号是 AE38-5F67

 D:\Tool 的目录

2018/05/18  15:13    <DIR>          .
2018/05/18  15:13    <DIR>          ..
2018/05/18  15:10               126 ParamConfig.txt
2018/05/18  15:08                79 ResultConfig.txt
2018/05/16  17:24         3,353,362 Servise-1.0-SNAPSHOT.jar
2018/05/18  15:07                40 ServiseConfig.txt
2018/05/18  15:07             7,780 Tools.py
               5 个文件      3,361,387 字节
               2 个目录 384,431,017,984 可用字节
```

图 5-1 测试服务端目录

接下来分别在 8082 端口和 8081 端口运行控制中心和服务注册服务器，接着运行服务端程序 Service-1.0-SNAPSHOT.jar，命令行参数为 127.0.0.1 和 8081，服务端程序先进行服务注册，向服务注册服务器发送注册信息，注册成功后进入监听状态。此时可以通过访问控制中心的查询页面进行对已注册服务信息的查询操作，通过数据库也可以看到记录，结果如图 5-2 所示：

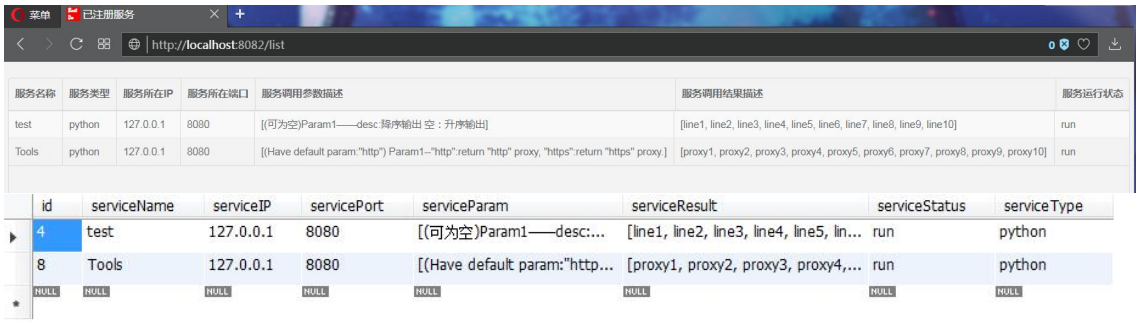

图 5-2 已注册服务查询

本次注册服务的具体信息为：服务名：Tools；服务类型：python；服务 IP：127.0.0.1；服务所在端口：8080；服务调用参数：Param1——“http”：返回 http 类型的网络代理，“https”：返回 https 类型的网络代理；服务调用结果：返回 10 个网络代理，结果名为 proxy1-10。

在完成服务端程序的启动，开始请求服务。请求服务的测试 python 代码如图 5-3 所示：

```
payload = {"ip": "127.0.0.1", "port": "8080", "name": "Tools"}
headers={'content-type': 'application/json'}
r = requests.post("http://localhost:8082/python/call", data=json.dumps(payload), headers=headers)
print r.text
```

图 5-3 测试 python 代码

这段代码将对控制中心发起服务调用请求，并将得到的结果打印出来，此时并没有传入服务调用参数，参数将采用默认值“http”，即结果将返回 http 类型的网络代理，结果如下图 5-4 所示：

```
D:\>python RequestTools.py
{"result":{"proxy7":"\"http\": \"119.28.194.66:8888\"","proxy6":"\"http\": \"61.135.217.7:80\"","pro
xy9":"\"http\": \"1.119.193.36:8080\"","proxy8":"\"http\": \"122.114.31.177:808\"","proxy1":"\"http\
": \"61.135.217.7:80\"","proxy10":"\"http\": \"120.26.110.59:8080\"","proxy3":"\"http\": \"122.114.3
1.177:808\"","proxy2":"\"http\": \"119.28.194.66:8888\"","proxy5":"\"http\": \"1.119.193.36:8080\"",
"proxy4":"\"http\": \"218.241.234.48:8080\""}}
```

图 5-4 测试 1 结果

接着将服务调用参数设置为“https”，此时结果将返回 https 类型的网络代理，结果如下图 5-5 所示：

```
D:\>python RequestTools.py
{"result":{"proxy7":"\"https\": \"115.203.174.41:45079\"","proxy6":"\"https\": \"1.197.153.54:37648\
"","proxy9":"\"https\": \"42.51.12.2:808\"","proxy8":"\"https\": \"221.228.17.172:8181\"","proxy1":"
\"https\": \"119.162.134.147:8118\"","proxy10":"\"https\": \"171.113.159.157:808\"","proxy3":"\"http
s\": \"123.163.154.13:808\"","proxy2":"\"https\": \"114.226.128.223:6666\"","proxy5":"\"https\": \"1
06.56.102.7:8070\"","proxy4":"\"https\": \"115.210.75.91:34928\""}}
```

图 5-5 测试 2 结果

接下来进行一次错误测试，将服务调用参数设置为“ftp”，此时调用服务的请求仍会正常处理并且服务端程序也会正常调用服务，但由于服务本身的设置此时将会发生错误，对错误的具体处理由服务自行负责，然后服务会返回对应的错误

信息，这个错误信息会被当做正常结果通过系统返回给服务调用者，调用者需要自行对错误进行处理，具体结果如下图 5-6 所示：

```
D:\>python RequestTools.py
{"result":{"proxy1":"Error: Wrong Param: ftp"}}
```

图 5-6 测试 3 结果

5.3 本章小结

本章节对完成的 MMS 系统进行了应用测试，得到了系统可以正常完成服务注册和服务调用等所要求的功能的测试结果。实际上，本章测试所使用的获取网络代理服务本就是实际工作内容中所需要的一个服务，通过测试结果可以看到，本文实现的 MMS 系统可以完成实际工作的任务需求。

6 总结与展望

在这几个月的设计与实现中，微服务管理系统 MMS 基本实现了当初定下的需求：

1. 对开发团队内互相使用的服务进行统一管理。
2. 开发者可以通过 MMS 系统方便地调用服务。
3. 提供服务的开发者不需要对自己的服务做出太大改动，只需提供符合系统规范的配置文件和接口即可。

为了实现这些需求，在完成 MMS 系统的过程中，涉及了以下理论与技术：

1. 微服务架构；
2. Java 序列化；
3. Java 的 NIO 通信；
4. Web 应用的 MVC 模式；

同时分别使用了 Protostuff 框架来完成序列化功能，Netty 框架来完成 NIO 通信功能，SpringBoot 框架和 MyBatis 框架来完成 Web 应用的开发。

最后通过测试证明了本课题设计实现的 MMS 系统可以投入实际使用当中。

接下来我们讨论一下 MMS 系统目前还存在的一些局限以及日后可以继续研究继续开发的方向。

1. 目前控制中心的部署方式是单点部署，对于目前的应用环境来说单点部署已经足以，但是随着业务规模的扩大和团队规模的扩大，MMS 系统需要管理和调用的服务会越来越多，此时单点部署带来的风险的严重性和影响性也会随之上升。
2. 目前服务端程序与服务的数量比为 1：1，即每个服务的当前目录下都有一个服务端程序，随着被提供的服务越来越多，这种部署方式会带来一定的硬盘资源的浪费和运行资源的浪费。
3. 目前 MMS 系统对于异常服务的处理方式比较简单，使用者需要通过查询已注册服务的学习才可以得到具体服务是否处于正常运行状态。

针对以上的不足，MMS 系统在后续可以进行如下改进：

1. 控制中心可以进一步地细分，将查询页面和接受调用请求的部分迁移出来，通过使用例如 ZooKeeper 等框架来实现处理服务注册/注销模块和转发执行命

令模块的分布式部署。

2. 对 MMS 系统的规范进行进一步的优化，从配置文件的层面实现提供服务的层级化或结构化，当然同时也得对服务端程序进行升级。抑或对服务端程序进行优化，让服务端程序支持对位于子目录的服务的扫描，这样也可以达到减少服务端程序数量，提高服务部署聚合度的目的。

3. 可以在控制中心里添加一个新的模块，该模块会定时扫描已注册服务的信息，对于近期变化为异常运行状态的服务，该模块会向服务提供者和管理员的邮箱发送邮件进行提醒操作，此种改进方法需要添加对联系方式的存储。或者可以为 MMS 系统添加心跳检测模块，将上述提到的定时扫描改为心跳检测，对于检测异常的服务同样发送信息给服务提供者和管理员，此种方法的及时性更高，可以在服务被调用之前发现服务的异常，采用此种方法需要对控制中心和服务端程序进行升级。

致　谢

完成整个系统的过程是辛苦的，在这个过程中，我得到了外界的很多帮助，也从中学到了很多。

首先，我要感谢我的校内指导老师胡侃老师，感谢他在我完成毕业设计的过程中给予的关心和敦促。

其次，我要感谢我的校外指导老师肖威，他在我分析和设计 MMS 系统的过程提供了很大的帮助，特别是在分析系统需求的过程中提出了十分宝贵的意见，使得完成后的 MMS 系统与实际开发需求高度契合。

另外，我还要感谢陪伴我的同学们，他们认真完成自己课题的态度和坚持奋斗的精神给了我很大的鼓舞和激励，同时他们还会耐心地倾听我对自己课题的想法，帮助我梳理思路，并提出自己的建议。

还有，还要感谢本科阶段所有的老师们，是你们孜孜不倦地教诲让我在大学四年中学习到了丰富的知识，为以后进一步的学习和更深入的研究打下了坚实的基础。

最后，感谢母校，感谢在大学四年里与我朝夕相处的同学们。没有母校这四年的培养就没有现在的我。同时，作为 14 级计卓班这个集体的一员，我得到了这个集体对我的许许多多的帮助和关心。因为有你们各位，我的四年青春没有辜负。

参考文献

[1] Benjamin Götz,Daniel Schel,Dennis Bauer,Christian Henkel,Peter Einberger,Thomas Bauernhansl. Challenges of Production Microservices[J]. Procedia CIRP,2018,67.

[2] Martin Fowler, James Lewis. Microservices[EB/OL]. https://martinfowler.com/articles/microservices.html, 25 March, 2014.

[3] 陈林,应时,贾向阳.SHMA:一种云平台的监控框架[J].计算机科学,2017,44(01):7-12+36.

[4] Vianden, Lichter and Steffens. Experience on a Microservice-Based Reference Architecture for Measurement Systems[C]. Asia-Pacific Software Engineering Conference, 2014:183-190.

[5] Rahman M, Gao J. A Reusable Automated Acceptance Testing Architecture for Microservices in Behavior-Driven Development[C]. Service-Oriented System Engineering (SOSE).IEEE, 2015:321-325.

[6] Krylovskiy A, Jahn M and Patti E. Designing a Smart City Internet of Things Platform with Microservice Architecture[C]. International Conference on Future Internet of Things and Cloud. IEEE, 2015:25-30.

[7] Le V D, Neff M M,Stewart R V, et al. Microservice-based architecture for the NRDC[C]. IEEE International Conference on Industrial Informatics. IEEE, 2015:1659-1664

[8] 郭文丽,严潮斌,吴旭.基于 Android 客户端的图书馆微服务研究与实践[J].图书情报工作,2013,57(08):22-26.

[9] 操凤萍,徐锦川.掌上校园微服务系统的研究与实践[J].电脑知识与技术,2017,13(26):64-66.

[10] 温晓丽,苏浩伟,陈欢,邹大毕.基于 SpringBoot 微服务架构的城市一卡通手机充值支撑系统研究[J].电子产品世界,2017,24(10):59-62.

[11] 郭栋,王伟,曾国荪.一种基于微服务架构的新型云件PaaS平台[J].信息网络安全,2015(11):15-20.

[12] 李贞昊.微服务架构的发展与影响分析[J].信息系统工程,2017(01):154-155.

[13] Schroth C, Janner T. Web 2.0 and SOA: Converging Concepts Enabling the Internet of Services[M]. IEEE Educational Activities Department, 2007 , 9 (3) :36-41.

[14] Newman S. Building Microservices[M]. O'Reilly Media, Inc. 2015.

[15] Pivotal 团队. Spring Boot Reference Guide1.5.3.RELEASE[OL].2017.

[16] 王永和,张劲松,邓安明,周智勋.Spring Boot 研究和应用[J].信息通信,2016(10):91-94.

[17] 薄奇,许林英.Spring 框架中 IoC 的实现[J].微处理机,2008(01):147-149+153.

[18] 张国平,万仲保,刘高原.Spring AOP 框架在 J2EE 中的应用[J].微计算机信息,2007(36):254-256.

[19] 朱荣鑫. 基于微服务架构的游戏商城服务端的设计与实现[D].南京大学,2017.

[20] 刘行亮. 基于 J2EE 平台的 Spring 框架分析研究与应用[D].武汉科技大学,2006.

[21] 夏汛,陈玲.基于 Spring MVC 和 Mybatis 的动态表单设计[J].计算机光盘软件和应用,2012(20).

[22] 荣艳冬.关于Mybatis持久层框架的应用研究[J].信息安全与技术,2015,6(12):86-88.

[23] Begin C, Goodin B, Meadors L. Ibatis in Action[M]. Manning Publications Co. 2007.

[24] 徐雯,高建华.基于 Spring MVC 及 Mybatis 的 Web 应用框架研究[J].微型电脑应用,2012(7).

[25] 文欢欢,刘振宇,吴霖.基于 Mybatis 和 JDBC 的分页查询研究[J].电脑知识与技术,2015,11(25):165-167.

[26] 姜力.基于 Java NIO 反应器模式设计与实现[J].大庆师范学院学报,2008(02):27-30.

[27] 刘仕筠,盛志伟,黄健.Linux 环境并发服务器设计技术研究[J].成都信息工

程学院学报,2006(05):630-634.

[28] 王洁.JAVA NIO 在 Socket 通讯中的应用[J].成都信息工程学院学报,2003(03):258-261.

[29] 王伟平,杨思勤.基于NIO的高并发网络服务器模型的研究与设计[J].硅谷,2009(17):12-13+29.

[30] 范宝德,马建生.Java 非阻塞通信研究[J].微计算机信息,2006(36):116-119.

[31] Anderson C. James Phillips on Service Discovery[J]. IEEE Software, 2016, 33(6):117-120.

[32] 肖仲垚. 微服务通信框架的设计与实现[D].北京交通大学,2017.