|  |  |
| --- | --- |
| **学校代码** | **10699** |
| **分 类 号** | **TP391.9** |
| **密 级** |  |
| **学 号** | **2016201734** |

|  |  |
| --- | --- |
| **题目** | **面向舰船平台管理的** |
|  | **SOA技术研究** |

|  |  |
| --- | --- |
| **作者** | **索鹏** |

|  |  |
| --- | --- |
| **学科、 专业** | **计算机软件与理论** |
| **指 导 教 师** | **吴健** |
| **申请学位日期** | **2019年3月** |

西 北 工 业 大 学

硕 士 学 位 论 文

（学位研究生）

|  |  |
| --- | --- |
| 题目： | 面向舰船平台管理的 |
|  | SOA技术研究 |

作 者： 索鹏

学科专业： 计算机软件与理论

指导教师： 吴 健

2019年1月

**Title: Research on SOA Technologies in Shipborne Platform Management**

**By**

**Suo Peng**

**Under the Supervision of Professor**

**Wu Jian**

A Dissertation Submitted to

Northwestern Polytechnical University

In partial fulfillment of the requirement

For the degree of

Master of Computer Software and Theory

Xi’an P. R. China

January 2019

摘 要

舰船平台管理系统采用计算机控制技术、网络技术等对舰船平台的主要系统如推进系统、电力系统、损管系统和辅机系统等实施集中监视、控制和管理。该系统要求将现代舰艇的各类运算操作、基础数据集成到一个统一的、公共计算环境中，实施分布或集中式管理。舰船平台上集成的应用应能根据使用过程中的反馈进行快速的更新。

面向服务架构(SOA)是一种构建分布式应用的软件设计风格。服务是可进行远程调用、具有完整的业务逻辑并独立运行的软件组件。客户端使用标准数据格式和协议与服务进行交互，从而使服务开发实现技术独立。微服务架构是将持续交付的软件开发理念和容器技术等新思想注入到SOA中形成的一种架构风格，采用微服务架构，可以灵活的由现有的服务组合成新的应用，并且满足现代舰船应用系统快速迭代开发和高可用等需求。

本文研究面向舰船平台管理软件服务集成技术，并提出舰船系统平台本身也采用微服务架构，这样可以保证平台的可靠性，即平台中一个模块（即一个微服务）的故障不会影响到整个平台的使用，而故障模块由于粒度小，从而可以对其进行快速的调试，维护和部署，使故障模块对系统的影响降到最低。此外通过将关键模块进行多实例部署，可以实现平台整体的高可用性。而微服务的快速部署特点使平台中的模块或应用服务根据负载情况进行动态扩展，使得系统具有更大的弹性。

论文首先介绍了SOA及微服务架构的相关概念和技术，然后对面向舰船平台管理的软件集成平台进行需求分析，使用UML对各个模块及模块间的通信过程进行建模，最终设计并实现了基于微服务架构的集成平台的技术方案。该平台为各个子系统微服务提供服务发现功能，为客户端屏蔽了服务实例的物理地址；并提供熔断机制，在服务调用出现故障或性能问题时由平台接管并直接返回错误信息，防止客户端由于异常服务调用而阻塞；平台还提供安全认证和权限管理，对客户端进行认证并审核服务调用权限。最后，论文通过构造一个虚拟应用场景对该集成平台进行实验验证，结果表明本文所设计与实现的集成平台合理、可行。

关键词：面向服务架构，服务模型，服务模型描述语言，数据接口，操作接口

**Abstract**

Shipborne platform management system uses computer control technologies and network technologies to monitor, control and manage major systems on the shipborne platform such as propulsion system, power supply system, damage control system and auxiliary equipment system. It demands the consolidation of various computing operations and fundamental data sets into a single and public computing environment to apply centralized or distributed management. Applications integrated through the shipborne platform should be updated timely in respond to feedback from daily use.

Service-Oriented Architecture (SOA) is a style of software design for building distributed applications. A service is a software component that can be remotely invoked, has self-contained business logic and runs independently. Clients use standard data format and protocol to interact with services, as a result, service development can be made technically neutral. Microservice architecture is an architectural style that combines SOA with continuous delivery software development process and new technologies such as container technology, by adopting microservice architecture, existing services can be combined flexibly to form new applications and modern shipborne applications’ requirements of fast iterative development and high availability can be met.

This paper studies service integration technologies in shipborne platform management software and suggests that the shipborne system platform itself should also adopt microservice architecture so that the resulting platform is resilient, which means faulting of one module (microservice) will not affect the whole platform, and due to its small granularity, the faulting module can be debugged, maintained and deployed more quickly such that the negative effect of the faulting module is brought to the minimum. Besides, the platform can be made highly available by deploying multiple instances of the core modules. And the fast deployment of microservices means that modules and services in the platform can be scaled dynamically according to their loads, so the system is more elastic.

This paper first introduces relevant concepts and technologies in SOA and microservice architecture, and then it analyzes the requirements of the software integration platform targeting shipborne platform management, using UML to create models for the modules and the communication processes among them. Finally, it designs and implements the technical proposal of the integration platform based on microservices architecture. The proposed platform provides service discovery function to each subsystem microservice, abstracting away the physical locations of service instances for the invoking clients; it also provides circuit breaker mechanism, which means the platform will take over and return error messages when the service is broken or when its performance suffers, so that client will not block when invoking errant services; the platform also provides authentication and authorization services, identifying the clients and checking their credentials. In the end, this paper verifies the integration platform by constructing a hypothesized application scenario, and the result shows that the platform is reasonable and its design is feasible.

**Key words**：SOA, Service Model, Service Definition Language, Data Interface, Operation Interface

目 录

[摘 要 I](#_Toc533947283)

[**Abstract** II](#_Toc533947284)

[目 录 IV](#_Toc533947285)

[1 绪论 1](#_Toc533947286)

[1.1 研究背景与意义 1](#_Toc533947287)

[1.2 国内外研究现状及发展趋势 1](#_Toc533947288)

[1.3 论文主要工作及贡献 3](#_Toc533947289)

[1.4 论文组织结构 3](#_Toc533947290)

[1.5 本章小结 4](#_Toc533947291)

[2相关概念与关键技术概述 5](#_Toc533947292)

[2.1 微服务架构 5](#_Toc533947293)

[2.2.1 微服务和单体架构对比 5](#_Toc533947294)

[2.2.2 微服务和传统SOA对比 5](#_Toc533947295)

[2.2.3 微服务架构特点 5](#_Toc533947296)

[2.2 Spring Boot 6](#_Toc533947297)

[2.3 REST 7](#_Toc533947298)

[2.4 Docker容器技术 7](#_Toc533947299)

[2.4.1 Docker简介 7](#_Toc533947300)

[2.4.2 Docker和虚拟机的区别 8](#_Toc533947301)

[2.4.3 Docker和微服务架构 9](#_Toc533947302)

[2.5 OAuth 2.0 9](#_Toc533947303)

[2.6 设计模式 10](#_Toc533947304)

[2.7 本章小结 10](#_Toc533947305)

[3 舰船平台服务管理系统的需求分析 11](#_Toc533947306)

[3.1 微服务系统面临的挑战 11](#_Toc533947307)

[3.2 舰船平台服务管理的系统目标 12](#_Toc533947308)

[3.3 舰船平台服务管理系统的功能性需求分析 12](#_Toc533947309)

[3.4 舰船平台服务管理系统的非功能性需求分析 13](#_Toc533947310)

[3.5 本章小结 13](#_Toc533947311)

[4 面向舰船平台管理框架的概要设计 15](#_Toc533947312)

[4.1 微服务架构的基本原则 15](#_Toc533947313)

[4.2 服务治理方案 15](#_Toc533947314)

[4.3平台管理框架的架构设计 15](#_Toc533947315)

[4.4平台管理核心框架的接口设计 15](#_Toc533947316)

[4.5 平台管理API网关的数据库设计 16](#_Toc533947317)

[4.6 本章小结 16](#_Toc533947318)

[5 面向舰船平台管理框架的详细设计与实现 17](#_Toc533947319)

[5.1平台管理核心框架的设计与实现 17](#_Toc533947320)

[5.2平台管理API网关的设计与实现 17](#_Toc533947321)

[5.3本章小结 17](#_Toc533947322)

[6 面向舰船平台管理框架的测试与验证 18](#_Toc533947323)

[6.1平台管理框架的功能性测试 18](#_Toc533947324)

[6.2平台管理框架的非功能性测试 18](#_Toc533947325)

[7 总结与展望 19](#_Toc533947326)

[7.1论文总结 19](#_Toc533947327)

[7.2下一步工作展望 19](#_Toc533947328)

[参考文献 20](#_Toc533947329)

[致 谢 21](#_Toc533947330)

[攻读硕士学位期间发表的学术论文和参加科研情况 23](#_Toc533947331)

[**学位论文知识产权声明书** 24](#_Toc533947332)

1 绪论

本章首先介绍了面向舰船平台管理系统的研究背景和意义，通过分布式技术的演变过程、面临的问题以及解决的手段，确立了采用SOA技术实现舰船平台管理是当前的趋势。通过讨论近年来微服务架构流行的原因和该架构本身的优势，提出使用微服务架构作为舰船平台管理服务化的实现手段。通过分析国内外企业微服务架构的使用案例，明确微服务架构中服务治理方案，提出了微服务框架的特点。最后对论文的工作和结构进行了介绍，使读者对论文的研究内容有一个初步的了解。

1.1 研究背景与意义

舰船的信息化、自动化和智能化管理是未来重要发展趋势之一。随着信息技术的发展和舰船系统和设备的日益复杂化，对舰船上的各个分系统进行统一管理和信息集成的需求也越来越明显。舰船平台管理系统就是利用计算机及网络技术将主推进系统、损管系统、电力系统、辅机系统、监控系统、三方系统、保障系统以及各类信息系统等相对独立的子系统连接起来，实现信息集成，从而对全舰实现以信息为中心的监测与管控[1]。

通过对舰船内部各个子系统的集中管理，实现各类数据的共享，从而可以实现通过分析全舰的状态迅速做出决策，实现最优的分布式实时智能控制；通过消除舰船中的“信息孤岛”，促进各个系统的协同合作，强化了舰船的综合管理能力，同时实现节能减员，产生巨大的经济效益；通过提供平台管理标准接口和规范，加快了新系统及应用的开发速度；通过将集成的各种状态信息通过卫星传输至远程监控指挥系统，有助于实现船岸一体化发展[2]。

相比民用船舶，舰船的各个系统在操作性、实时性、精确性和维护性等方面都具有更严格的要求。目前，国外新型研制的舰船都装备平台管理系统来提高舰船的信息集成和自动化程度，如美国下一代大型驱逐舰DDX就采用了平台管理系统，使得具备巡洋舰规模的该舰舰员只有100人左右，此外，英国的45型驱逐舰、德国的F124护卫舰、法国和意大利联合研制的“地平线”级驱逐舰、韩国的KDX-2等舰船上都采用类似的平台管理系统[3]。虽然我国第一艘航空母舰“辽宁号”已经下水服役，但与世界海军强国相比，特别是平台管理系统软件方面，仍存在较大差距。而国内舰船的各个子系统相对独立，并采用不同的技术标准，开发语言和体系架构，这给系统的集成和管理带来了更大的挑战[4]。因此，有必要对面向舰船平台管理的相关技术进行研究，对平台管理系统进行概要设计，掌握系统的设计方法和关键技术，加快我国舰船信息化建设进程，减少对国外技术的依赖的同时满足国防建设的需要。

1.2 国内外研究现状及发展趋势

对舰船平台管理系统的研究与同时代计算机技术的发展有着紧密的联系，由最初的单体架构朝着分布式架构演变，而分布式技术又由早期的远程过程调用(RPC, Remote Procedure Call)逐渐演化到现在的面向服务架构（SOA，Service Oriented Architecture）。RPC将发生在进程内部的过程调用扩展到分布式系统中，使得客户端进程可以调用远程服务器进程的函数。随着面向对象技术的兴起，远程方法调用（RMI，Remote Method Invocation）将面向对象的模型应用到RPC中，从而实现不同进程中对象的相互调用。面向对象的思想和分布式技术的结合产生了分布式对象技术，具有典型代表的是Java RMI和CORBA，但分布式对象技术在使用过程中出现了很多的问题，例如对象的接口描述没有对依赖进行描述，分布式对象中间件的编程模型过于底层，以及编程模型没有分离出安全、失效处理以及并发等分布式系统中常见的公共功能，这些都给分布式对象编程带来了极大的困难。为了解决这些问题，分布式组件技术应运而生，其中最具代表性的是微软公司的DCOM模型、Sun公司的EJB模型以及CORBA的组件模型CCM（CORBA Component Model），组件通过合约描述了其提供和依赖的接口，不同的组件可以根据接口的兼容性进行灵活的组合，通过使用现成的组件可以快速组合出需要的应用。国内也出现了基于分布式组件技术的舰船平台管理系统的研究[5]。但上述组件在舰船平台管理系统中的应用依然存在问题，其中DCOM组件存在二进制版本兼容问题[6]；EJB与Java面向对象编程语言紧耦合，这导致与非Java语言开发的组件进行组装时会存在问题；而CCM虽然不依赖于具体的编程语言，但其特性过于复杂，增加了开发难度和开发人员的学习成本。

SOA的出现解决了分布式构件技术的种种问题，通过将系统进行垂直切分为小粒度的服务，可以实现类似分布式构件的灵活性；通过标准化通信协议，SOA很好地解决了服务通信的兼容性；通过完整的服务治理规范，降低了开发难度。美国海军提出的“全舰计算环境(TSCE, Total Ship Computing Environment)”的概念中，提出利用企业服务总线(ESB, Enterprise Service Bus)实现应用的集成[7]。然而，使用ESB实现SOA，ESB会成为单点故障，而ESB自身作为一个庞大的单体构件不利于动态的扩展其功能，同时，ESB作为重量级中间件，对其进行功能扩展需要复杂的设置，增加用户的学习成本[8]。

近年来，作为SOA的一种实现和部署方案，微服务架构（MSA，Microservice Architecture）去除了传统SOA所使用的ESB，其采用分布式服务治理，基础设施自动化以及轻量的通信协议，并融合领域驱动设计、持续交付、小型自治团队等软件工程思想[9]。国内外涌现出许多优秀的微服务框架，目前国内以阿里巴巴的开源框架Dubbo影响力最大，其主要目标是实现高性能和透明的远程服务调用[10]。国外也有微服务框架的成功案例，比如Netflix公司和其他工程团队通过采用微服务架构，实现了系统的高可用性，应用的快速迭代开发和部署能力，提高了服务的重用性，降低了服务的响应时间[11][12]。国内外优秀的微服务框架具有共同的特点，包括提供服务注册与发现机制，灵活的通信协议支持及路由配置，提供负载均衡能力，提供自动化部署途径以及对服务进行监控、组合、降级等服务治理能力[13]。

TSCE中提出了对计算、存储、网络等硬件资源的虚拟化，实现了基础设施即服务（IaaS, Infrastructure as a Service）[7]。而微服务通过容器技术，可以快速的部署到IaaS中，从而可以充分的利用IaaS提供的基础设施[14]。采用微服务架构实现舰船平台管理是未来海军平台化建设的发展方向。

国内对于面向舰船平台管理的技术的研究处于理论探索和初步实践的阶段，目前有一些高校和研究所在开展相关工作。研究内容主要还是利用构件技术来实现舰船平台管理[5]，与国外仍存在较大差距。本文借鉴TSCE有关采用SOA来实现面向舰船平台管理的思想，并进一步提出采用微服务来实现舰船平台管理，希望能对我国舰船平台管理的发展做出贡献。

1.3 论文主要工作及贡献

本课题来自于实际横向合作项目，对面向舰船平台管理的SOA技术进行了研究，通过跟踪舰船基础设施和平台管理的发展方向，提出采用微服务架构实现舰船平台管理。基于微服务架构，研究满足面向舰船平台管理中服务注册发现、服务安全、服务生命周期管理和服务监控需求的舰船平台服务管理系统。

在设计该系统的过程中，本文参考了现有优秀微服务框架的特点和舰船平台管理的业务需求，采用软件工程的研究手段对舰船平台服务管理系统进行了需求分析，给出了功能设计方案和实现方案。该方案提供微服务开发框架，使用REST API接口规范，采用RPC调用方式，提供高性能的服务集成。框架与Spring Boot无缝集成，保证开发编码的低侵入性，并实现服务的可配置。系统采用基于Docker的部署方案，实现自动化部署和持续集成。系统通过日志分析实现对服务调用时间、次数、性能等数据的统计与追踪。系统还实现负载均衡、服务熔断、服务降级、身份认证与权限控制等机制，用于满足舰船平台服务管理的非功能性需求。

本文按照软件工程的基本方法，使用用例图进行需求分析、使用流程图描述系统业务流程、使用架构图呈现系统的各个模块、使用类图和时序图描述系统的详细设计。在使用各种图形进行建模的同时，对系统功能的设计与实现进行了详细的描述。

1.4 论文组织结构

本文通过对面向舰船管理SOA技术的研究，设计与实现基于微服务架构的舰船平台管理系统，从介绍项目背景和国内外研究现状开始，通过对各种相关技术的分析，对舰船平台管理系统进行从需求分析、概要设计、详细设计和系统验证，最终实现系统的可行性研究。

本文一共有七章，具体结构如下：

第一章，绪论。本章主要介绍论文的背景、意义、国内外的发展趋势和技术变革对舰船平台管理系统带来的影响，最后介绍了论文的组织结构。

第二章，介绍舰船平台服务管理系统中出现的关键概念和使用的部分技术。首先对微服务架构和设计模式进行介绍，然后介绍了Spring Boot，REST，Docker容器，Auth 2.0等微服务主要技术和框架。

第三章，对舰船平台服务管理系统进行需求分析。通过对系统所面临的挑战进行分析，挖掘系统的功能性和非功能性需求，并最终完成系统功能模块的划分。

第四章，对舰船平台服务管理系统进行概要设计。通过对总体架构设计，接口设计和数据库设计的描述完成对系统的微服务架构实施方案的介绍。

第五章，对舰船平台服务管理系统进行详细设计与实现。主要描述了服务定义、服务注册与发现、服务监控、安全认证、API网关以及自动化部署的详细设计与实现。

第六章，对舰船平台服务管理系统进行验证。通过模拟应用场景对系统进行测试与验证，证明系统的可行性和有效性。

第七章，总结与展望。对本文的研究进行了总结，并对下一步的工作进行安排，同时对未来的研究方向进行了展望。

1.5 本章小结

本章首先介绍了研究背景和意义，然后通过描述国内外舰船平台管理系统的发展和技术演变，结合当下面向服务架构的实现技术，提出了采用微服务架构实现舰船平台管理这一创新性理念。最后本章对论文的主要工作和组织结构进行了介绍。

2 相关概念与关键技术概述

2.1 微服务架构

微服务在2011年由一个软件架构小组首次提出，用于描述它们使用的一种实现SOA的架构机制[15]。当前被业界普遍接受的关于微服务架构的观念是由Martin Fowler提出的：微服务是将软件划分为一组单一功能的、可以独立开发部署运行的服务，服务运行于相互独立的进程之中，服务之间采用轻量的方式进行通信。

2.2.1 微服务和单体架构对比

传统的单体架构根据系统中模块间职责的不同分为表示层、业务逻辑层和数据访问层[16]。该架构的优势为系统容易进行开发和设计，并且由于采用该架构开发应用的历史已久，有大量成熟的框架可以使用，系统的部署也相对容易，只需将应用打包并复制到应用服务器上运行即可。

随着业务的不断增加，单体架构的缺陷逐渐体现出来。首先是扩展能力差，因为系统中的功能模块高度耦合，当需要对功能进行扩展的时候，会引入大量的相关依赖。另外，在单体架构下对一个模块进行修改，意味着要对整个应用进行重新的编译、测试和部署，大大增加了开发周期。单体架构中的一个模块成为瓶颈，需要对整个应用进行水平复制，这极大的浪费了硬件资源，增加了运营成本。

微服务就是为了解决单体架构的问题，通过将单体应用拆分成服务，每个服务可以独立扩展和修改。当系统性能出现问题时，只需将个别微服务进行复制即可。

2.2.2 微服务和传统SOA对比

SOA时将系统按照不同功能拆分成服务单元，并通过良好定义的接口和标准通信协议对服务进行组合的一种软件组件模型。从定义上看，传统SOA和微服务在思想上是一致的，作为SOA的一种实现方法，微服务将敏捷开发、自动化部署和云计算思想融入传统SOA，使其呈现自身的特性和侧重点。

微服务摒弃了在传统SOA中居核心地位的企业服务总线，使得服务更加独立灵活；微服务推荐使用REST形式的接口来替代Web服务的SOAP协议，使用轻量级的JSON作为推荐的数据传输格式；微服务的粒度更小，使得开发周期短，利于服务的部署和横向扩展。

2.2.3 微服务架构特点

每个微服务具有单一的职责，作为基本模块通过组合形成应用。由于使用标准化接口进行交互，每个服务可以采用不同的技术手段进行开发。微服务具有自治性，被独立的开发、部署、运行和维护。服务间的通信通常采用HTTP + JSON或AMQP等轻量级协议。微服务通常实现一个小而完整的业务功能，从而一个服务开发团队只由几个人组成，并且由开发团队负责运维。微服务之间相互解耦，增加了应用开发的灵活性并限制了模块出错的传播范围。故而近年来采用微服务架构的企业和团队逐步递增。

2.2 Spring Boot

Spring框架通过依赖注入、切面编程和模板技术将J2EE开发者从EJB复杂的开发模型中解救出来。通过依赖注入，开发者不用管理对象的创建以及对象间的依赖关系，并通过面向接口编程，增加了程序的灵活性；通过切面，Spring支持将开发过程中的公共服务提取出来，以注解的方式应用到各个需要的业务逻辑中，增加了代码的复用度；通过模板技术，Spring封装了开发过程中与业务逻辑无关的工作，如获取数据库连接和异常处理等，从而增加了代码的清晰度，降低了程序员的工作量。

Spring Boot是Spring社区对Spring框架进行的进一步精简和优化，具体而言，使用Spring Boot构建微服务，具有以下好处[18]：

1. 提供starter依赖

在开发一个Spring项目的过程中，项目需要依赖很多第三方库，例如日志库、模板库、序列化库等。通常使用Maven或Gradle将一个项目的依赖进行统一管理，尽管如此，当项目的依赖变得愈来愈多时，依赖的维护、项目依赖关系的分析以及依赖之间的兼容性都会给项目的开发和维护带来很大的困难。Spring Boot按功能领域将开发某一功能所需要的依赖统一起来，并以一个总的starter依赖项进行发布，开发者只需引入一个依赖项就可以传递获取互相兼容的一组依赖。Starter依赖项增加了项目依赖结构的清晰度，降低了依赖维护的难度，解决了依赖间的兼容性问题。

1. 提供自动配置功能

通过采用“惯例优于配置”的思想，为Spring框架的许多配置项提供了默认选项。这样一来，通过少量的配置就能完成基于Spring框架的服务开发。Spring Boot会根据类路径上配置的依赖包自动配置各种bean而不需要开发者进行显式的bean声明。Spring Boot在提供默认配置的同时也允许开发者进行显式的配置，在最大化简化开发工作量的同时保留了Spring框架的灵活度。

1. 提供命令行界面

Spring Boot提供了一个可以执行Groovy脚本的命令行， Groovy采用类似Java的语法，但对Java语法进行了精简。使用Groovy和Spring Boot命令行，允许开发者用极少量的代码实现一个微服务，由于Groovy是解释型语言，Spring Boot通过命令行界面提供了快速迭代开发的能力。

1. Spring Boot Actuator

Spring Boot Actuator为Spring Boot开发的微服务增加了许多REST API接口。通过这些接口客户端可以访问到一切有关微服务的信息，包括服务中运行的bean、服务的配置属性和环境变量、服务中每一个线程的函数调用栈、服务的健康情况、服务调用的次数统计和服务调用历史纪录等。

1. 集成运行容器

不同于传统的Spring Web应用，Spring Boot可以将开发的微服务打包成jar包，其中包含服务运行所需要的容器，不像war包需要部署到Java Servlet容器中，只需要运行环境支持JVM即可。这样做简化并统一了部署环境，符合微服务动态扩展和自动化部署的要求。

正是上述Spring Boot的这些特点，使得其成为一款优秀的微服务开发框架。

2.3 REST

2000年Roy Fielding在其博士论文中首次提出REST（Representational State Transfer,表述性状态转移）的概念[19]。REST包含一组架构原则和约束规范，它强调以资源为中心，将网络上可访问的所有对象都抽象成资源，整个Web被视为资源的一个集合，其中的每个资源由唯一的标识符进行标识。满足REST规范的服务就被称作RESTful服务。RESTful服务提供规范的接口对资源进行操作，对资源的访问通常是无状态、可缓存的。根据亚马逊建设服务构架的过程中总结的经验，RESTful Web服务的性能由于基于SOAP的Web服务[20]。

论文提出的舰船平台服务管理系统采用REST风格实现，其各个组件的接口设计满足RESTful API的设计规范，即使用URL路径来标识资源，用HTTP方法来表明对资源进行的操作，RESTful API的范例如表XX:

表2-XX REST API范例

|  |  |  |
| --- | --- | --- |
| URL | HTTP方法 | 接口描述 |
| /attributes | GET | 查看所有属性 |
| /attribute | POST | 创建新属性 |
| /attribute/{id} | GET | 查看ID为id的属性 |
| /attribute/{id} | DELETE | 删除ID为id的属性 |
| /attribute/{id} | PATCH | 更新ID为id的属性 |

2.4 Docker容器技术

2.4.1 Docker简介

Docker是使用容器来开发、部署和运行应用的平台。容器包含运行应用所需要的环境，包括代码、运行时环境、环境变量和配置文件等。容器间相互独立并且包含独立的网络、内存和文件子系统。容器引擎负责管理容器，容器引擎通过扩展操作系统内核，使在同一引擎上运行的容器共享同一个操作系统，从而允许在一个操作系统上运行几百个容器的同时具有接近运行在操作系统上的性能，故Docker被称为轻量级容器。

Docker主要有两种用途：持续集成和持续部署[21]。为了部署一个应用，首先在Dockerfile中使用各种命令来生成镜像，该镜像作为生成Docker容器的模板被推送到本

地的注册中心。Docker镜像通过tar文件的形式存在，故而部署一个使用Docker打包的应用只需复制一个文件到目标机器上即可。Docker镜像使用分层的设计，每次容器数据发生改变需要持久化时，Docker只保存变化的部分，这样极大地降低了镜像的大小。从而使得Docker镜像可以快速的复制和部署。Docker平台提供了一组API用来管理容器镜像和容器的执行，而Docker注册中心和DockerHub提供镜像的复用，使我们能在任何地方快速部署服务。

2.4.2 Docker和虚拟机的区别

图xx显示了Docker和虚拟机在架构上的区别[22]。在虚拟机中，虚拟机监控程序运行在宿主操作系统之上，而虚拟机操作系统则运行在虚拟机监控程序上，由监控程序负责管理。而Docker使用容器技术，使得不同的容器直接运行在宿主操作系统上，由Docker守护进程进行管理。

（a）虚拟机 （b）Docker

图 Docker与虚拟机的架构区别

作为虚拟化技术，容器比虚拟机更轻量。虚拟机通过在硬件上建立抽象来实现虚拟化，而容器中的应用通过相应的容器进程拥有自己的进程ID空间，文件系统结构和网络接口，从而实现了对内核的虚拟化。由于虚拟机包含整个操作系统环境，而该环境中的许多组件对于运行应用来说是多余的，而容器只包含支持运行应用的环境，因而运行容器的开销更小。通常运行在监控程序上的虚拟机占用主机10%-15%的资源，而Docker只使用极少量的资源[23]。有研究表明Docker容器的运行速度比虚拟机平均快26倍[23]。监控程序的开销随着在其之上运行的虚拟机的增加成指数被增长，这使得在同一台机器上可以运行的Docker容器数远超过虚拟机[24]。Docker容器本身不存在启动过程，所以容器内的应用进程可以实现瞬间启动。

2.4.3 Docker和微服务架构

Docker容器和微服务是天生的一对，Docker容器的低开销和快速部署的能力使其特别适合实现微服务架构，它们共同提升系统的模块化，促进代码复用，实现运行过程可重现，以及实现细粒度的分布式系统的扩展[25]。然而，容器技术在实现高可用的同时，也带来的一系列的挑战，其中最大的挑战是服务发现。接下来介绍的Spring Cloud Netflix就是用来解决这一问题的技术。

2.5 OAuth 2.0

舰船上不同的子系统可能由不同的研发单位研发，而不同的子系统也可能由不同的部门负责监管，这造成了舰船平台中的服务存在于不同的信任域中，每个域可能使用不同的用户名和认证机制。如何跨域进行身份管理是OAuth主要解决的问题。

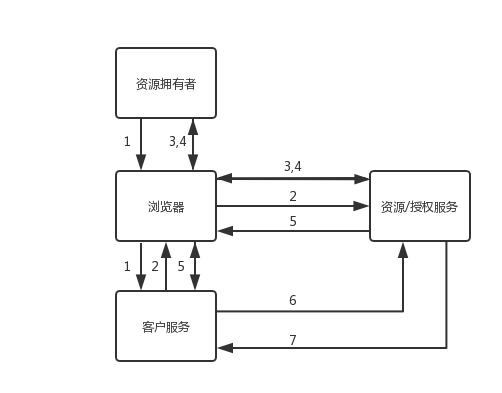
传统的进行跨域服务调用的方法是给需要请求调用的第三方应用用户名和密码来获取相应服务的调用权限，这样做会导致第三方应用和用户一样具有完全的服务请求权限，这一般不是用户想要的，用户需要限制第三方应用具有的调用权限，例如只能调用查询信息功能而不能对信息进行删除或修改。OAuth(Open Authentication)是一个开放认证协议，允许第三方应用不使用用户的用户名和密码就可以完成对用户资源请求的认证和授权，并且通过授权机制限制第三方应用可以进行的操作[26]。

OAuth 2.0中的主要概念如下：

1. 客户服务：请求授权的服务；
2. 资源拥有者：拥有客户端所请求的资源的具体用户；
3. 资源服务：提供资源的服务；
4. 授权服务：对所提供的资源拥有者的凭据进行验证并进行权限检查的服务，通常授权服务和资源服务是同一个服务，两者必然处于同一个信任域；
5. 访问令牌：由授权服务创建，让客户端用来访问资源服务的一段数据；
6. 认证码：授权服务在访问请求阶段用来进行身份检查的一段数据。

OAuth 2.0中一次服务请求验证的过程如图XX所示[27]：

1. 资源拥有者在浏览器中登陆客户服务，并在请求客户服务直接访问资源服务；
2. 客户服务返回一个响应，该响应将资源拥有者通过浏览器重定向到资源服务（该服务同时也是授权服务），将客户服务者ID和资源请求信息发送给资源服务，资源拥有者需要登陆到资源服务；
3. 资源服务让资源拥有者确认客户服务的请求；
4. 资源拥有着对请求进行确认；
5. 资源/授权服务对用户的确认进行响应，生成一个验证码，并通过浏览器将资源拥有者重定向到资源服务者，同时传送的还有生成的验证码；

图XX OAuth的一次认证请求过程

1. 客户服务将验证码和其他相关信息发送给资源/授权服务；
2. 资源/授权服务对验证码进行验证，确定消息来源后生成访问令牌，并将令牌返回给客户服务。

至此，客户服务就可以使用令牌直接访问资源服务，代替资源拥有者对资源进行访问。在OAuth的一次请求过程中的大部分都发生在后台，简化了资源拥有者的操作。

2.6 设计模式

2.7 本章小结

本章主要介绍了研究过程中遇到的相关重要概念和使用的有关技术。首先对SOA的演变过程进行了介绍，给出微服务架构实现SOA的优势，然后介绍了流行的微服务开发框架Spring Boot。接下来对微服务通信设计所使用的REST架构风格、部署方案使用的Docker技术以及安全认证所使用的Auth 2.0标准进行了描述。最后，本章对系统架构设计过程中使用的设计模式和详细设计中使用的面向对象设计模式进行了介绍。

3 舰船平台服务管理系统的需求分析

本章从微服务系统所面临的一般性挑战和面向舰船平台服务管理的系统目标出发，分析实现面向舰船平台管理的舰船平台服务管理系统的功能性和非功能性需求。本章将对各个功能模块进行建模，从而确立系统的功能性需求。此外，本章还从易用性、性能、安全性、可靠性、可管理和可扩展性等方面对系统的非功能性需求进行了分析。

3.1 微服务系统面临的挑战

现如今舰船系统的设备越来越复杂，舰船平台服务管理系统要能够对舰船内部各个子系统进行统一管理，传统的单体式架构所构建的垂直应用已经不能满足舰船平台统一管理和子系统间的交互和集成的需求。面向服务架构与单体式架构相比更加灵活，更能满足当代舰船信息一体化需求。通过将各个子系统应用的公共业务进行抽取并独立作为公共基础服务，形成平台服务管理模式下的底层基础组件。采取微服务架构实现面向服务架构相比传统的ESB方式对舰船平台服务管理系统进行设计与实现，一方面可以保证这些基础组件的高可用和负载均衡，另一方面可以降低新应用的开发难度，适应新型舰船需求灵活多变的特征。然而，采取微服务架构仍面临各种各样的挑战[27][28]。

总的来说，构建平台服务管理系统以实现微服务带来的好处，具有以下挑战：

1. 实现灵活性的同时保证高性能

传统的单体架构业务流程中的模块以函数的形式呈现，模块间的调用发生在同一个进程之中，而微服务架构将各个模块划分出来形成独立自治的服务，虽然这降低了系统维护的难度，增加了系统的灵活部署和扩展的能力，但模块间的调用也由之前的进程内的函数调用转变为进程间的远程调用，增加了调用延迟。而微服务的粒度越小，实现业务逻辑的服务调用次数就越多，如何多次远程服务调用的情况下保持系统的性能给平台服务管理系统的设计带来了巨大的挑战。

1. 实现微服务的快速开发和部署

微服务架构要求服务快速迭代开发和持续部署的同时还要实现权限控制、负载均衡、健康监控等功能。而微服务的细粒度意味着微服务开发数量的增加，如果开发人员要在每一个服务中考虑上述功能的实现，这无疑增加重复的工作，降低了开发效率。所以要求微服务系统提供开发框架，封装公共功能的实现，规范接口设计标准，提供集成开发和部署工具，实现项目的从初始化、编译到集成和部署的自动化。

1. 微服务架构中的安全性和稳定性考虑

对于任何系统来说，安全和稳定都是需要满足的特性，不安全或者不稳定的系统是无法使用的，而对于舰船系统来说，系统的安全性和稳定性有更高的要求。微服务架构中的安全性要求系统提供完善的身份认证和访问控制机制。而稳定性要求当服务出现故障或错误的时候，系统能够及时的检测并进行服务恢复。系统还应通过多实例部署、服务降级或服务熔断等机制来提高系统和服务调用的稳定性。

1. 确立微服务的治理方案

在舰船系统服务化的过程中，随着服务数量的增长，服务的规范治理显得尤为重要。当代舰船的计算基础设施逐渐走向虚拟化，以IaaS或PaaS的形式呈现。故可以参考云服务的治理模式，实现服务的独立部署、升级和扩容。云计算的弹性伸缩，多实例部署，健康监控和容量规划等服务运维措施都可以融入到舰船平台系统的设计当中。

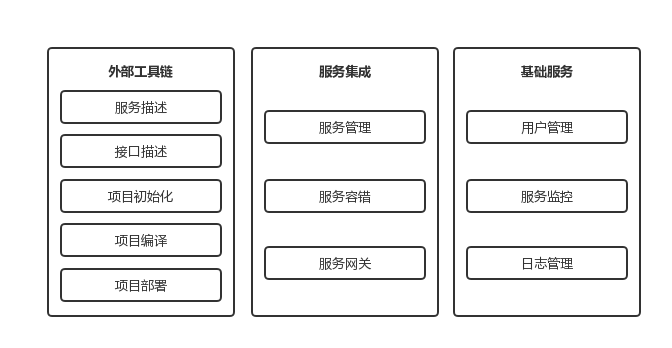
微服务治理应用到舰船系统中，要采用线上线下相结合的方式。通过服务的上下限审核、服务文档管理和服务协作规范等制度的规定来实现服务的线下治理。通过服务目录、服务监控、服务日志、用户管理和服务权限管理等手段来实现服务的线上治理。如何协调和实现统一的线上线下的服务治理方案，也是设计舰船服务化平台需要重点考虑的问题。

3.2 舰船平台服务管理的系统目标

舰船平台服务管理的主要目标是为了推进舰船平台化发展，引入服务治理规范，设计和实现面向服务管理的微服务开发和集成框架。

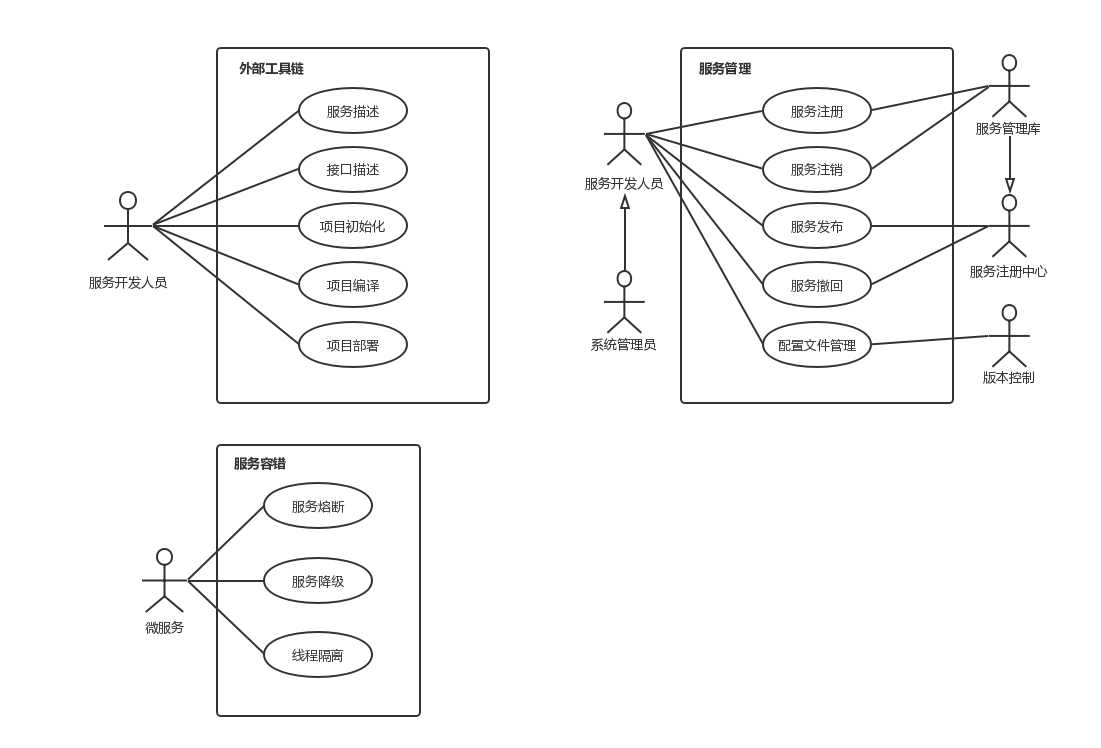
经过研究现有的微服务开发和集成框架，首先，为了降低服务的开发成本，舰船平台服务管理系统应提供服务开发框架，封装服务通信、服务生命周期和健康管理等模板代码，降低开发人员的工作量，提高开发效率。系统还应该提供服务注册发现、监控、身份认证与访问权限控制等服务管理功能，并提供服务配置和部署的自动化工具，形成规范的服务综合治理平台。

3.3 舰船平台服务管理系统的功能性需求分析

根据舰船平台管理系统的系统目标，可以将系统按功能划分为三个模块，如图XX所示：

图XX 舰船平台服务管理系统功能架构

3.3.1 外部工具链

外部工具链是用来支持服务开发与部署的自动化工具的统称。外部工具链主要供服务开发人员使用，其用例图如图XX所示。其主要功能包括服务描述、接口描述、项目初始化、项目编译和项目部署。

图XX 外部工具链的用例图

具体的需求描述如下：

1. 服务描述

系统应支持通过JSON文件描述服务的主要信息，包括服务的标识、服务的描述、服务版本信息、服务的接口、服务开发的依赖、服务地址等服务描述信息。

1. 接口描述

使用接口定义语言对接口进行描述，包括服务的名称、接口的名称、接口接受的参数类型、接口返回的结果类型。

1. 项目初始化

系统应提供开发框架，在引入服务描述文件后，根据文件内容生成项目结构，目录框架和项目描述文件pom.xml。

1. 项目编译

系统支持采用Maven进行项目的自动化构建，编译脚本根据接口描述文件在目录框架中生成响应的服务代码框架，框架集成了与服务集成相关的功能。

1. 项目部署

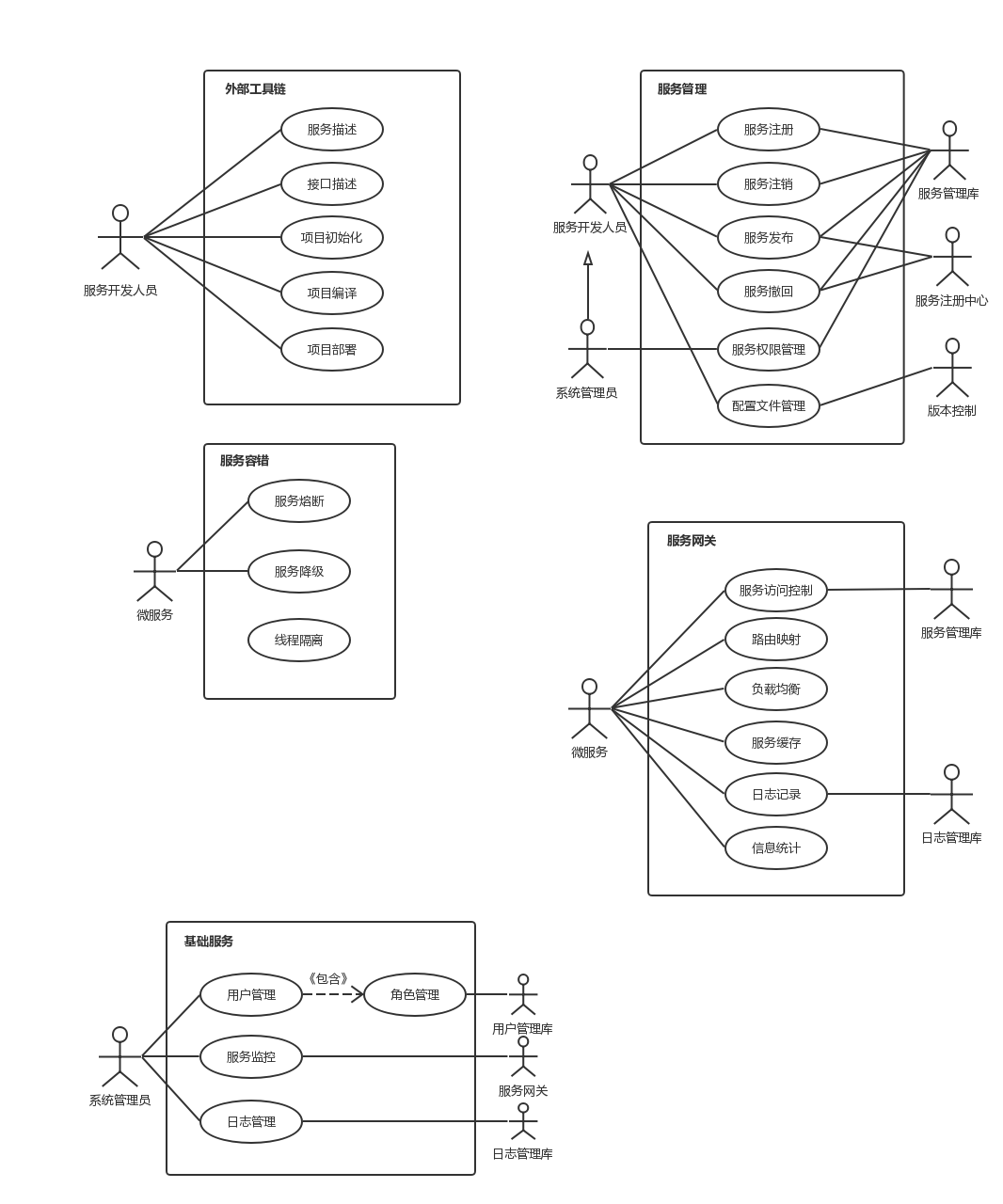
系统提供项目的自动化部署脚本，将Maven构建的jar包制作成Docker镜像，并根

据服务描述文件的内容将镜像部署成Docker容器。

3.3.2 服务集成

服务集成包括了系统在实现服务调用过程中所需的一系列功能，进一步可以划分为服务管理、服务容错和服务网关三个功能子模块。

1. 服务管理

服务管理主要是系统管理员和服务开发人员对服务的元数据和配置文件进行管理，其用例图如图XX所示。

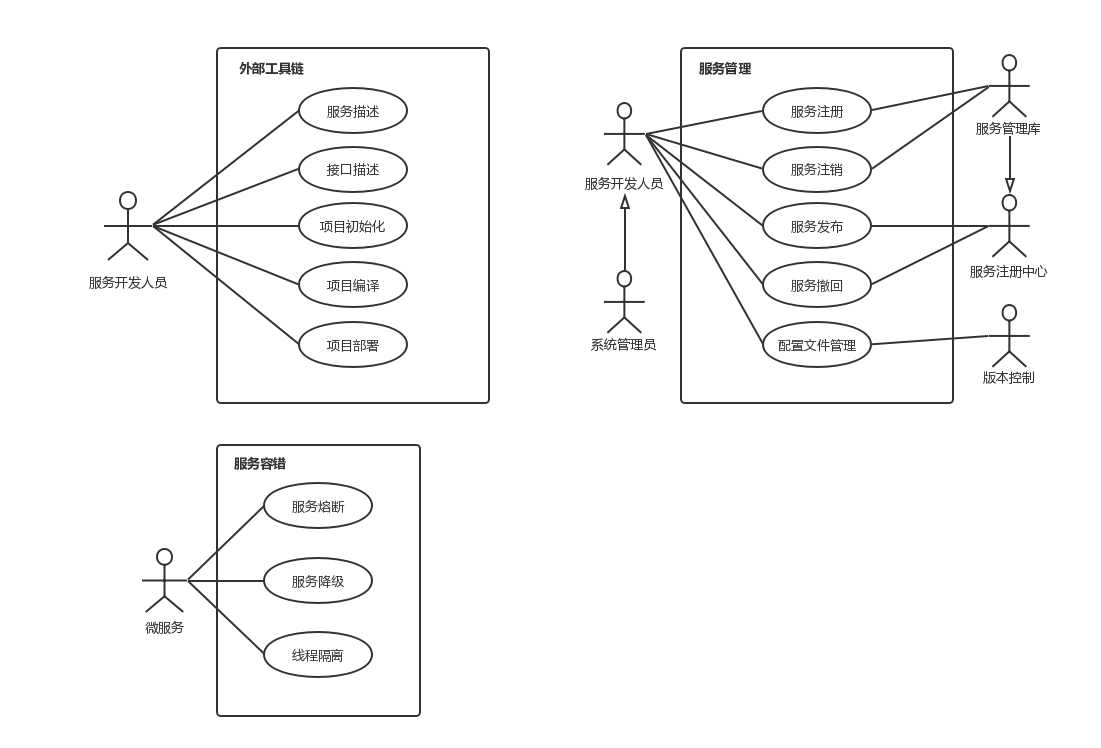
图XX 服务管理的用例图

服务管理包括的功能有：

1. 服务注册/注销：开发并测试好的服务完成部署后，需借助本功能向系统注册，注册信息保存于服务管理库中，包括服务的分类信息和描述服务的各种元数据。当然，服务注销是服务注册的反操作，它从服务管理库中删除服务的分类与描述信息。
2. 服务发布/撤回：系统管理员或服务开发者利用此功能将已注册的服务通过注册中心向外界发布，使其对外部可见。服务撤回是服务发布的反操作。
3. 服务权限管理：对权限的增删改查。服务开发人员可以对平台中所有已发布的服务进行选择并订购，服务管理员对订购行为进行审核，若审核通过，则给予订购者以服务授权。服务授权的结果是给订购者颁发一个OATH授权码，在服务运行阶段，所有对授权服务的访问都依据该授权码进行访问控制。对来自于同一子系统或同一用户的服务，缺省认为它们之间当然具有相互访问（接口调用）的授权，除非服务提供者另外指定访问权限。本功能当然也包含取消服务的调用权限的操作。
4. 配置文件管理：负责将一个服务的配置进行集中管理和版本控制，保证服务实例的一致性，当修改服务配置后，服务的所有实例将接到通知并更新配置信息。
5. 服务容错

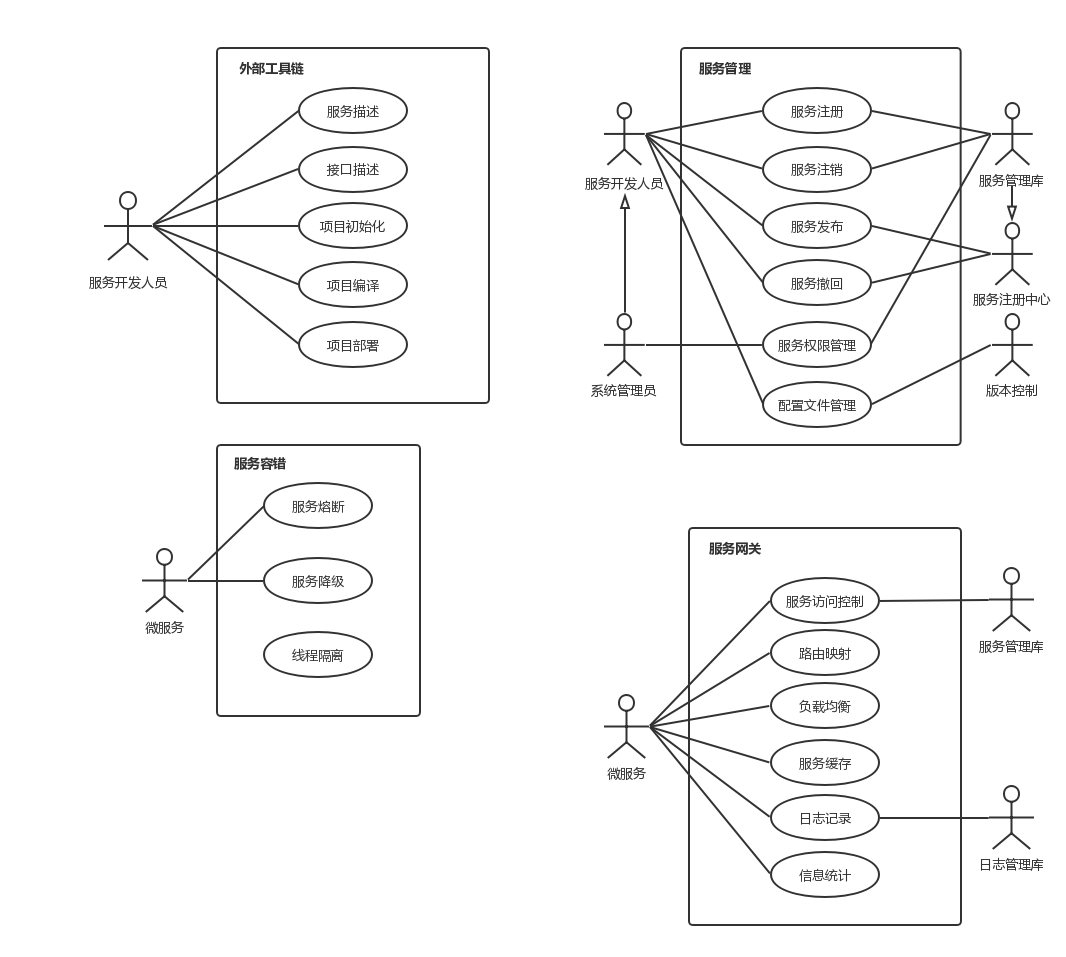
服务容错主要是为调用其他服务的微服务提供failfast、failover和隔离机制，保证服务调用的性能和可用性，其用例图如图XX所示。

服务容错包括的功能有：

图XX 服务容错的用例图

1. 服务熔断：这是处理服务故障并借以提高系统性能的机制。当对服务的访问结果进行统计分析后，如果服务调用和失败次数超过一定阙值，系统应阻断对它的进一步访问，而直接代理它向服务请求者返回服务不可用信息。
2. 服务降级：这是在服务出现故障时，系统通过用户注册的备选方案对其他可替代的服务进行调用或返回默认值，从而使系统不会因为没有调用成功而返回空值导致事务失败。
3. 线程隔离：这是防止一个微服务调用线程的阻塞导致虚拟机线程资源耗尽而无法调用其他微服务的情况，通过将每个微服务的调用限制在一个线程池中，可以将调用阻塞的影响限制在线程池范围内，而机器整体的线程资源不会受影响。
4. 服务网关

服务网关依据服务管理所产生的信息，对所部署并发布的服务的调用提供支持，所有对所发布的服务的访问均须经过服务过滤（根据授权和发布订阅关系）和路由（相当于反向代理），其用例图如图XX所示。

服务网关的主要功能包括：

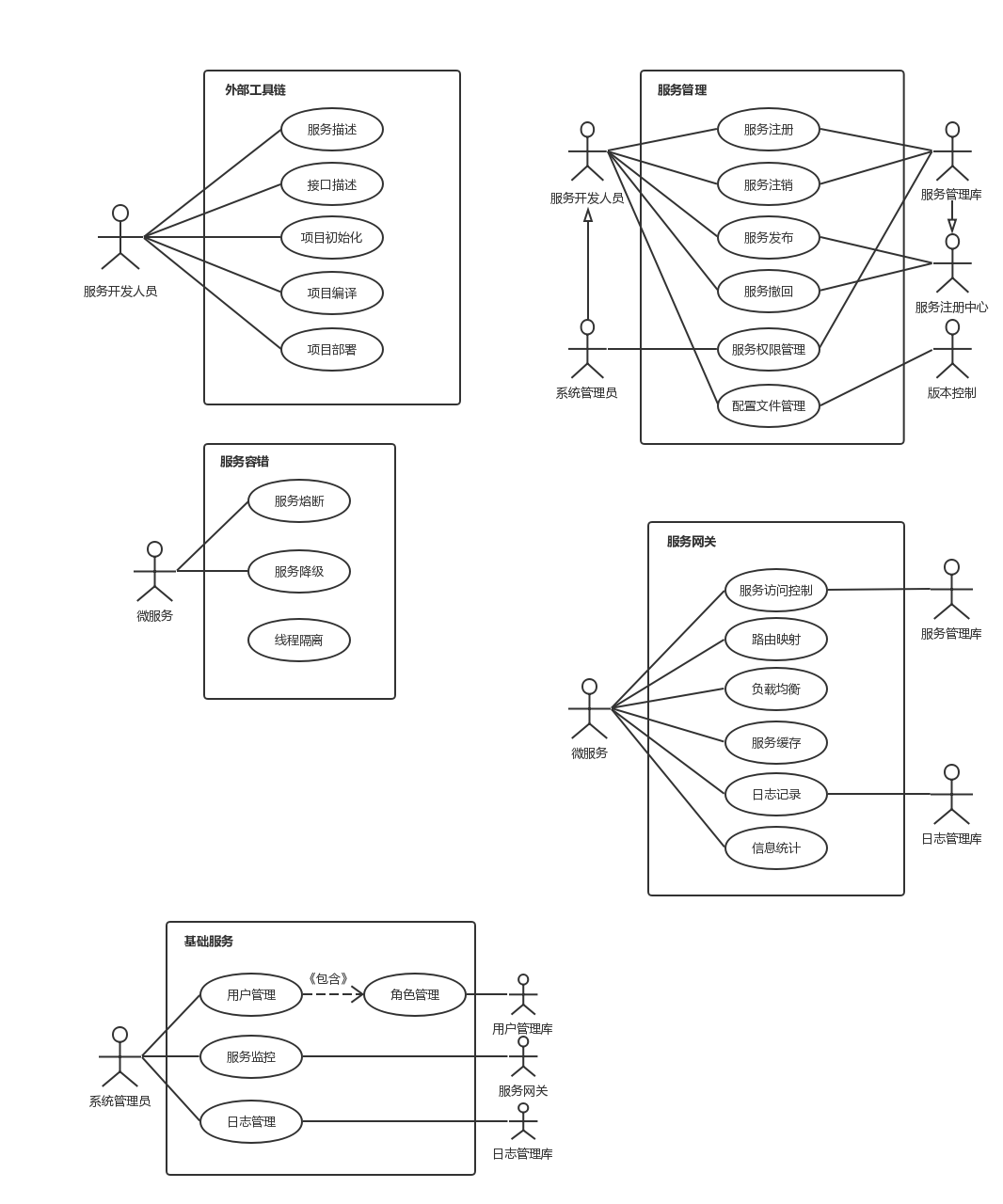
图XX 服务网关的用例图

1. 服务访问控制：依据服务服务订购与授权信息，对每一个发布的服务的接口调用同其提供的令牌进行权限核查与控制。
2. 路由映射：系统对外提供统一的服务调用URL，并提供从该路径到具体服务实例的地址的转换。
3. 负载均衡：如果后台被调用服务启动了多个实例，可从服务注册中心获取请求服务的所有实例，通过轮询或随机等策略将服务调用请求交给不同的实例进行处理，防止每次服务调用都访问同一个实例。系统在进行负载均衡的同时保持服务会话的完整性，且后台服务应该是无状态的。
4. 服务缓存：这是提升服务性能的机制之一。系统依据一定的原则，对后台服务调用的结果进行缓存，对后续到达的相同访问请求，直接利用缓存结果进行响应。
5. 日志记录：系统对服务的调用接口、调用参数、调用结果、请求调用服务的信息等记录到日志，用于系统的监控和审计。
6. 信息统计：系统对接口的调用次数和服务的响应时间等进行统计，用于对系统的性能进行分析。

3.3.3 基础服务

基础服务包括用户管理、服务监控和日志管理三个子功能，其用例图见图XX。

该功能模块的具体功能描述如下：

图XX 基础服务模块的用例图

1. 用户管理

用户管理部分主要使用基于角色的访问控制模型(RBAC)对用户和权限进行管理。主要包括用户管理和角色管理。

1. 用户管理：对用户信息的增删改查。
2. 角色管理：对角色的增删改查。
3. 服务监控

服务监控通过服务网关收集的信息向系统管理员显示系统的运行状况，服务监控包括服务健康监控和服务异常监控。

1. 服务健康监控：对服务实例的运行状况进行监控，包括服务实例是否正在运行，服务响应时间是否超时等。
2. 服务异常监控：监控系统组件的运行状况，发生异常提醒运维人员及时进行修复。
3. 日志管理

日志管理包括对日志的查询和汇总，方便系统管理员对系统的审计跟踪。

3.4 舰船平台服务管理系统的非功能性需求分析

舰船平台服务管理系统除了要满足上述的功能性需求以外，还要满足易用、高性能、安全、可靠、可管理和可扩展性等非功能性需求。

1. 易用性

任何系统的开发目标都是为了降低使用者的工作量，舰船平台服务管理系统应该通过提供公共模板简化服务开发，提供自动部署工具以简化服务部署，通过良好的UI界面和自动化反应机制以简化服务监控。

1. 高性能

舰船平台的性能要求虽然不像机载平台那么高，但由于面对作战要求的特殊性，要求能及时对数据进行处理并对事件进行响应。对于API网关在原请求处理时间上增加的额外时间不得多于50ms。由于一个业务可以包含多个微服务调用，这要求平台上的服务的请求处理时间不得超过1秒，对可能超时的请求，应采用异步方式进行处理。

1. 安全性

对于服务间传递的信息，要通过HTTPS协议进行加密通讯，对安全性要求高的数据一律采用加密手段进行传输，防止数据被非法劫持或篡改。

对于舰船平台服务管理系统，还应对服务调用请求进行身份认证和权限控制，防止服务的越权请求调用，同时应建立健全的安全规范，避免系统发生重大安全隐患。

1. 可靠性

对于API网关和服务注册中心等平台的核心部件，应保持高可用和可靠性。舰船平台的核心组件应能保证99.99%的高可用，并且应考虑战时出现部分基础设施损毁的情况下仍保证平台功能的可用性。

1. 可管理性

系统中服务的部署、运行和调用可以实现监控、日志、数据采集和追踪。通过提供图形用户界面队伍进行管理配置、监控统计、追踪审计，并可以对系统自身的运行情况进行监控和管理。

1. 可扩展性

可扩展性包括系统的功能模块可通过插件进行功能扩展，平台可通过控制版本兼容性对服务接口进行扩展，服务实例部署应可以动态扩展等。

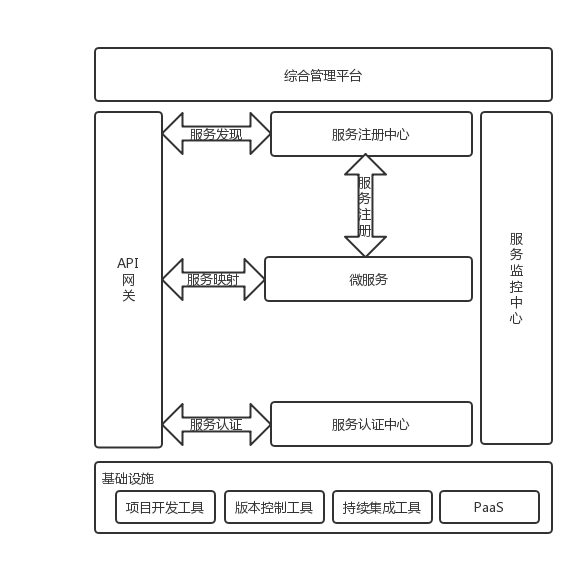
3.5 本章小结

本章对基于微服务架构的舰船平台服务管理系统进行需求分析。采用微服务架构需要保证服务调用的性能，实现服务开发部署的快速迭代，保证系统的安全稳定并制定规范的治理方案。所以舰船平台管理系统应提供易于使用开发框架和服务描述框架，提供服务调用过程中所需的服务注册发现、路由映射，提供身份认证、权限控制、负载均衡、熔断等机制来保证安全性和稳定性，提供健康监测、信息统计、日志记录来实现服务治理，同时提供自动化部署以实现微服务的伸缩性。系统在满足上述功能行需求的同时，还应满足易用、可扩展和可靠性能非功能性需求。

4 舰船平台服务管理系统核心模块的概要设计

完成了对系统的需求分析后，本章主要描述舰船平台服务管理系统的概要设计，给出舰船平台服务治理方案、然后对方案中的核心模块API网关以及与其关键的系统模块进行了模块的划分、接口设计以及数据库设计。

4.1 舰船平台服务治理方案

舰船服务平台化的核心在于提出一个合理完整的服务治理方案，该方案包括系统服务化所需的开发、集成、部署环境和安全、监控、管理手段。治理方案应该遵循微服务的基本原则，即服务小而自治、服务间通过轻量级REST API通信、服务独立部署、失效隔离和有效的日志和跟踪[30]。图XX展示了面向舰船平台管理的服务治理方案的架构。

图XX 舰船平台服务治理架构

该治理方案包含以下几个部分：

1. 服务综合管理平台

服务综合管理平台负责对整个系统进行管理，是系统的综合门户。通过该平台服务开发者可以对服务进行注册撤销和各种查询工作，可以查询微服务列表，查找每个微服务的接口信息、实例信息、相互依赖关系、统计数据、服务运行状态等系统应用运行的关键数据和信息。管理员可以在查询的基础上增加对用户、角色和权限进行管理。平台同时集成监控中心，可以对系统关键组件的运行状况进行监控，对系统的异常进行处理。

1. API网关

API网关作为调用所有微服务的入口，提供统一的调用接口，并将该接口映射到具体的服务实例地址。API网关作为反向代理，接受服务调用请求并对请求进行处理，如身份认证、路由映射、报文转换和日志记录等，然后将请求发送到服务实例进行处理。

1. 服务注册中心

服务注册中心负责服务描述信息、服务接口信息以及服务实例信息的存放。服务注册中心提供API进行服务注册、查询和上下线功能。API网关通过服务注册中心查找服务实例的地址来进行路由映射的功能。

1. 服务监控中心

服务监控中心进行服务健康检查，对服务日志进行采集，并对服务的运行状态进行监控。当监控到服务故障时，及时更新服务注册中心，并通过管理平台通知运维人员进行检测与维护。开发人员可以通过监控中心采集的日志来分析调用过程中的安全漏洞和性能缺陷。通过对服务调用进行追踪可以发现性能瓶颈，从而使开发人员可以对系统进行有效的优化。

1. 服务认证中心

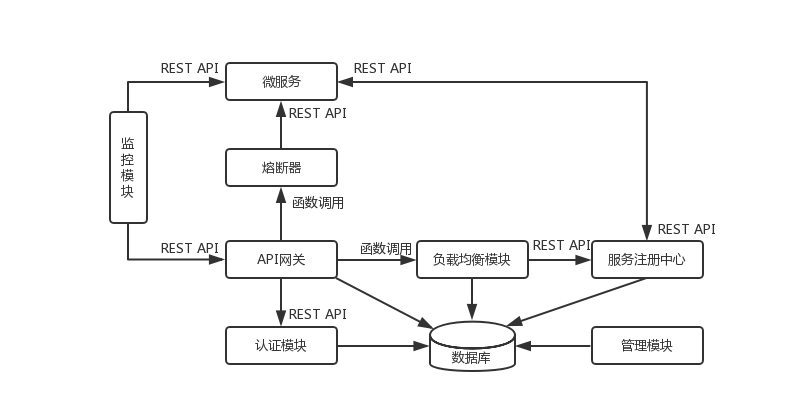
服务认证中心负责验证令牌的生成、管理和分发。令牌用于调用过程中对服务调用者进行身份认证和调用权限的确认。服务认证中心通过验证服务调用者传递的令牌确认调用者身份，然后查找与身份相关的调用权限，决定是否允许调用请求。系统的关键组件如注册中心、监控中心等也以微服务的形式存在，故也受到服务认证中心的保护。

1. 基础设施

基础设施包含支持服务开发、部署、运维的系统工具和服务运行环境。各个工具的主要功能如下：

1. 项目开发工具：负责服务开发过程中的目录结构管理，项目依赖管理以及代码生成和调试等辅助开发工具。
2. 版本控制工具：负责服务代码和服务配置的版本控制，记录代码和配置文件的版本信息。
3. 持续集成工具：对编译、版本控制、代码生成以及部署进行整合，利用自动化脚本完成代码提交到版本控制工具后进行自动编译、打包、Docker镜像的生成以及自动部署到PaaS平台。
4. PaaS平台：将舰载硬件计算资源整合后进行虚拟化的平台。平台提供一致的操作系统环境，支持Docker镜像的部署和管理，以及对平台本身进行监控和管理。

4.2 模块设计

舰船平台服务管理系统的模块划分如图XX所示：

图XX 舰船平台服务管理系统模块

4.2.1 API网关

API网关是调用系统管理的服务的统一入口，通过提供统一的服务调用的URL，屏蔽了具体服务实例的路由细节。API网关作为反向代理拦截服务调用的请求，并通过一系列处理后代理客户端发出请求，并将处理结果返回给客户端。API网关发出请求前进行的处理包括：

1. 身份认证：根据请求中包含的令牌向认证模块发出请求，确定服务调用者身份，检查服务调用权限，若身份验证通过且调用者具有调用权限，则进行后续处理，否则返回HTTP 403响应码。
2. 日志记录：将记录调用者的ID，调用时间，调用的服务及接口等调用信息记录到日志库中。
3. 路由映射：将对外暴露的服务地址转换为服务实例的具体地址，这一功能是通过调用负载均衡模块完成的。
4. 请求报文处理：对HTTP请求报文进行处理，包括报文格式转换，报文头内容替换，根据报文头的相关字段实现动态路由等。
5. 调用次数统计：对接口的调用次数进行更新，并更新相关的数据库。

API网关进行服务调用时，不是直接将请求发送到服务实例，而是调用熔断器来实现服务容错的相关功能，具体过程间熔断器模块的描述。

在API网关接收到响应结果并将结果返回给客户端前，API网关还可以进行相应的处理：

1. 响应时间统计：对调用的响应时间进行计算，并更新相关的统计数据。
2. 响应报文处理：对HTTP响应报文进行处理，包括报文格式转换，报文头内容替换等。

API网关处了进行相应的业务逻辑处理，还提供REST API接口暴露网关的运行数据、统计数据和路由数据供监控模块使用。

4.2.2 负载均衡模块

负载均衡模块负责将服务调用请求平均分发到每一个服务实例中，从而平衡系统的负载。负载均衡模块以库函数的形式供API网关调用。其通过服务ID发送REST请求到服务注册中心获取服务实例列表，然后根据数据库中有关该服务的负载均衡策略运行负载均衡算法，从服务实例列表中选出一个服务实例返回给API网关进行路由映射。

4.2.3 熔断器

熔断器负责具体的服务调用，它以代理模式代替API网关进行服务调用，通过代理机制，熔断器可以在被调用的服务发生故障或相应时间过长的情况下做出处理。通常的处理方法有：

1. 服务熔断：中断服务调用，直接抛出异常。
2. 服务降级：通过配置的策略在服务调用出现异常时选择其他方案对调用进行相应，比如调用候选服务或返回默认值等。

4.2.4 服务注册中心

服务注册中心时服务治理方案中的一个重要组件，负责服务的注册与发现。服务注册中心通过提供一系列REST API来进行服务实例查询，服务列表查询，服务实例注册等功能。服务注册中心通过在对服务实例进行注册的同时更新相关数据库，供管理模块使用。服务注册中心通过定期调用微服务的健康接口来检测服务实例的运行状况，当发现服务故障时更新实例列表，剔除发生故障的服务实例信息。

4.2.5 认证模块

认证模块对应服务治理方案中的服务认证中心，负责令牌的生成、分发和验证。认证模块生成的访问令牌与相关的访问权限一起存储在数据库中。认证模块提供REST API供API网关调用，返回令牌认证信息，并进一步进行权限的检查。

4.2.6 监控模块

监控模块对应服务治理方案中的服务监控中心，通过对微服务暴露出的状态信息接口和API网关提供的统计信息接口的调用，对系统上运行的微服务和各个组件进行健康监控，运行状态监控，统计信息监控等。该模块采用MVC模式进行设计，可以集成到服务治理方案中的综合管理平台上。

4.2.7 管理模块

管理模块负责对系统用户、服务、权限和令牌等进行管理。主要通过对数据库的操作实现对各个实体的增删改查。对系统参数如负载均衡等进行配置，并实现对系统中各类信息的查看，包括系统日志、服务列表、服务运行信息等。该模块采用MVC模式进行设计，可以集成到服务治理方案中的综合管理平台上。

4.2.8 微服务

微服务作为被系统管理的对象，其本身通过代码框架集成了很多统计服务运行信息的功能，并通过REST API接口形式暴露给系统相关组件进行调用。微服务在启动时，通过调用服务注册中心的API进行服务注册，将服务实例的地址等相关信息传递给注册中心进行保存。微服务提供健康监测接口供服务注册中心和监控模块调用。

该架构中系统的关键组件如API网关，认证模块和服务注册中心也作为微服务运行，故这些组件也提供相应的API供监控模块和管理模块进行调用，实现系统的运行态的监控和配置管理。

4.2.9 数据库

数据库用来存储系统中的各种实体的相关信息，包括用户、服务、令牌、权限、接口、服务实例、日志等。系统的各个组件都与数据库进行交互，认证模块从数据库中获取权限信息，API网关将日志记录到数据库中，负载均衡模块从数据库中获取负载均衡策略，服务注册中心向数据库中记录服务实例的信息，管理模块通过数据库对各类信息进行查看和管理。

本系统使用关系型数据库来实现该模块，数据库的具体设计见4.4节。

4.3 接口设计

4.4 数据库设计

本章详细地阐述了服务模型的设计方案，对其中的一些关键性的技术细节以Java编程语言为实现参考进行了深入讨论，包括服务模型描述语言SDL、SDL到Java的映射规范、生成的Java代码框架以及服务治理等方面的设计，这些细节对于实际的系统应用开发具有指导意义。针对本章设计方案，在下一章给出一个设计原型并测试验证。

4.5 本章小结

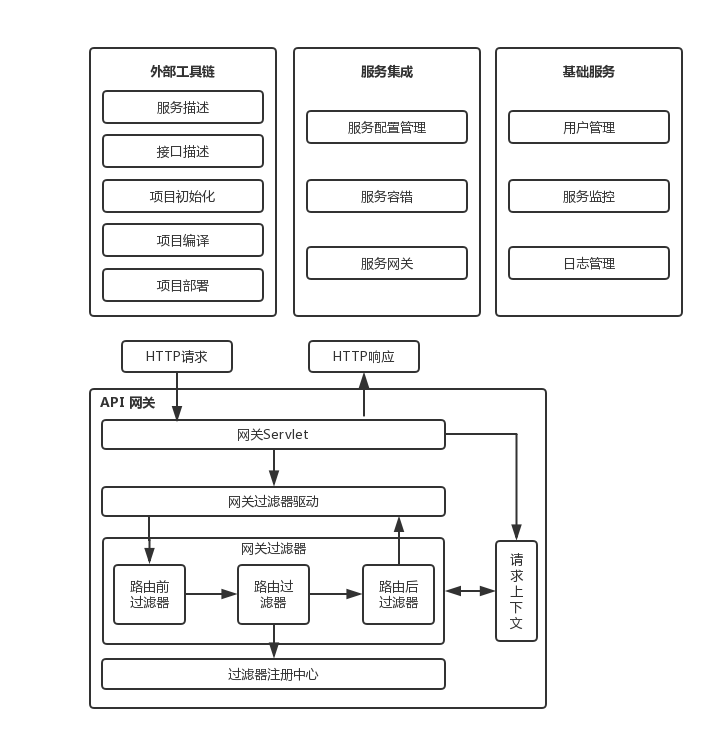
本章首先给出了舰船平台服务治理方案，然后根据舰船平台服务管理系统的需求分析，对管理系统的核心模块进行了架构分析。同时，本章还设计了系统主要模块的接口，分析了系统中各实体之间的关系，设计了系统所需的数据库的表结构。最后给出了系统所需的最重要的数据库表的字段、属性和描述信息。

5 面向舰船平台管理系统的详细设计与实现

本章将使用流程图、类图和时序图来描述面向舰船平台管理系统核心模块的设计与实现，具体包括API网关、负载均衡模块、熔断器、服务注册中心以及认证模块的设计与实现。

5.1 API网关的设计与实现

5.1.1 API网关的架构

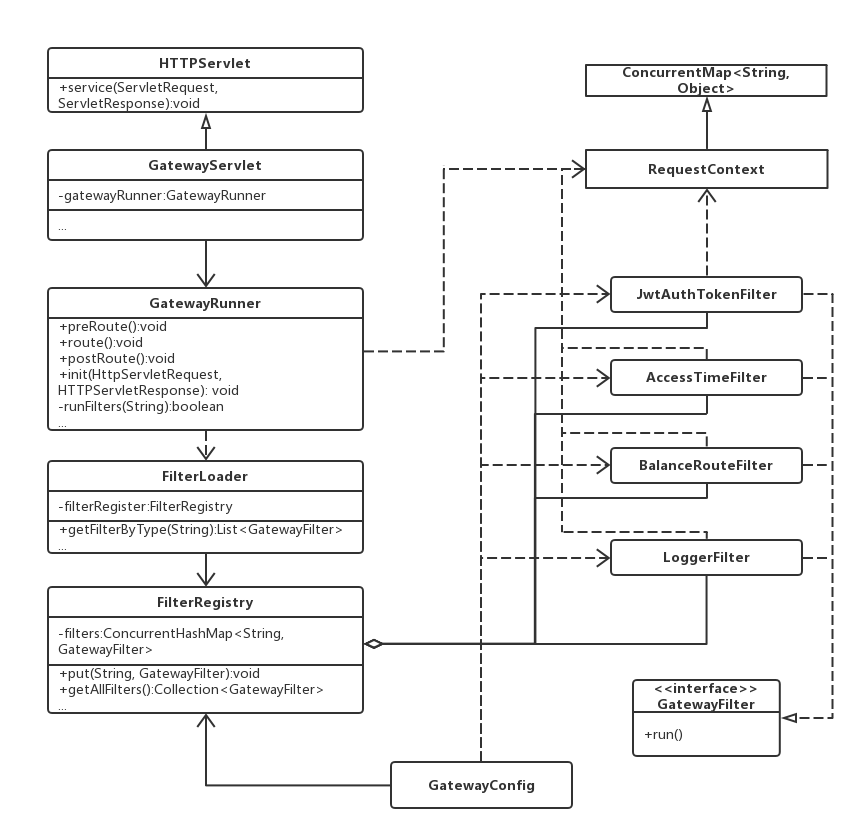
API网关的架构如图XX所示：

图XX API网关架构

对其中的各个模块的描述如下：

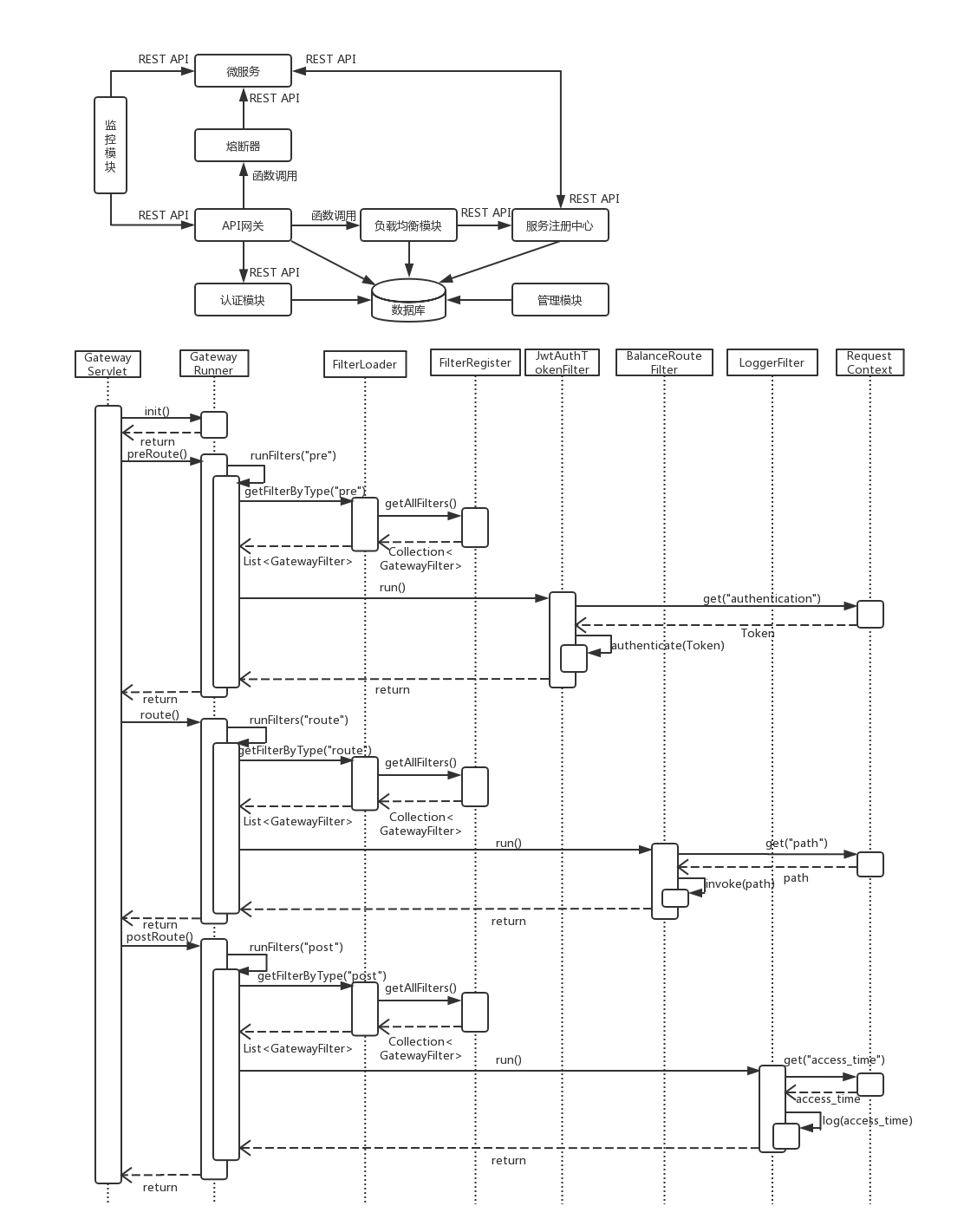
1. 网关Servlet: 用于处理HTTP请求并对生成HTTP响应；
2. 网关过滤器驱动：驱动网关过滤器按路由前过滤器、路由过滤器和路由后过滤器的顺序执行；
3. 网关过滤器：网关过滤器是API网关的核心组件，负责实现API网管身份验证、路由映射、日志记录等功能，分为路由前过滤器、路由过滤器和路由后过滤器：
4. 路由前过滤器：负责实现路由映射前API网关所执行的功能的一组过滤器，主要实现身份验证和日志记录的功能，该模块负责与认证模块与日志库进行交互。
5. 路由过滤器：负责实现路由映射和服务调用，该模块与负载均衡模块进行交互，获取实例地址并调用目标服务。
6. 路由后过滤器：负责实现服务调用后API网关所执行的功能的一组过滤器，该模块实现调用统计与日志功能。
7. 请求上下文：作为网关过滤器实现信息共享的对象。网关Servlet将HTTP请求的相关信息封装到该对象中，例如可以将访问令牌封装到该对象中，供负责身份验证的路由前过滤器使用。该对象还管理各过滤器共享的信息，例如实现服务调用响应时间统计的调用时间就记录在该对象中。
8. 过滤器注册中心：负责存储API网关所使用的所有过滤器对象，在网关过滤器驱动进行对过滤器进行调用的过程中，首先在该注册中心中取出相关的过滤器，然后按顺序进行执行。

5.1.2 API网关的核心类设计

API网关的核心类及之间的关系如图XX所示：

图XX API网关类图设计

其中GatewayConfig为网关的配置类，负责配置网关所需使用的bean，GatewayConfig在网关启动时实例化类JwtAuthTokenFilter, AccessTimeFilter, BalanceRouteFilter和LoggerFilter四个过滤器类。这四个类实现了GatewayFilter 接口的run方法，run方法用来实现每个过滤器的功能。其中JwtAuthTokenFilter和AccessTimeFilter为路由前过滤器，分别实现调用请求的身份认证功能和对调用时间进行统计。BalanceRouteFilter为路由过滤器，实现路由映射和服务调用功能，该类调用负载均衡模块对象方法进行服务实例的选择，然后调用断路器对象方法进行服务调用。LoggerFilter为路由后过滤器，负责实现日志功能，并和AccessTimeFilter一同实现服务响应时间的计算。GatewayConfig在初始化过滤器类的bean之后，将它们注册到FilterRegistry类中，FilterRegistry类使用一个ConcurrentHashMap<String, GatewayFilter>对象将各个过滤器bean按类别进行存放，使用字符串pre，route和post分别表示路由前过滤器、路由过滤器和路由后过滤器。

API网关处理服务调用请求的过程见图XX：

图XX API网关处理服务请求时序图

GatewayServlet类即为网关Servlet，该类通过实现HttpServlet的service方法实现对网关调用请求的处理。GatewayServlet实现网关过滤器驱动，调用GatewayRunner类中的方法来按顺序执行过滤器和使用当前服务调用请求的信息初始化RequestContext。GatewayServlet实现的主要代码如下：

public void service (ServletRequest servletRequest, ServletResponse servletResponse) {

gatewayRunner.init((HTTPServletRequest) servletRequest,

(HTTPServletResponse) servletResponse);

gatewayRunner.preRoute();

gatewayRunner.route();

gatewayRunner.postRoute();

}

RequestContext类是一个ConcurrentHashMap类，其负责将字符串映射到任意类对象。RequestContext对象除了映射HTTP请求报文头中的字段和值，还负责在各个过滤器之间传递消息。JwtAuthTokenFilter使用RequestContext对象中存储的HTTP报文头中的Authentication字段读取令牌信息，进行身份认证。BalanceRouteFilter读取报文头中的服务名进行服务映射。AccessTimeFilter和LoggerFilter通过在RequestContext中存放服务调用时间来计算服务响应时间。

GatewayRunner类方法说明如下：

1. preRoute(): 按顺序执行所有的路由前过滤器；
2. route(): 按顺序执行所有的路由过滤器；
3. postRoute(): 按顺序执行所有的路由后过滤器；
4. init(HTTPServletRequest, HTTPServletResponse): 初始化RequestContext对象；
5. runFilters(String): 调用所有由参数表示的类别的过滤器，其中参数去pre, route和post其中一个值，分别表示路由前过滤器、路由过滤器和路由后过滤器。preRoute, route和postRoute方法都使用runFilter来实现，例如，preRoute的实现：

public void preRoute() throws Exception{

if (!runFilters(“pre”)) throws Exception;

}

GatewayRunner的runFilters方法通过FilterLoader类的getFiltersByType实现，该方法通过调用FilterRegistry的getAllFilters方法获得注册表中的所有过滤器，然后根据参数所指定的类型对返回结果进行过滤，得到参数所指定的一类过滤器的列表，最后通过调用每个过滤器的run方法来执行其功能。

5.2 负载均衡模块的设计与实现

测试环境：测试实际配置了四台计算机，每一台都运行一个Tomcat服务器，四台计算机之间通过交换机连接以保证在同一局域网内。其中三台计算机，每一台的服务器上都各自部署着一个服务模型以及各自对应的服务配置文件，分别为Sonar、Grapher、Monitor。另外的一台服务器上部署着消息代理Proxy和发布订阅配置文件Mapping.xml，实现对数据发布订阅的统一路由管理。计算机环境配置如表5-1所示。

表5-1计算机环境配置表

|  |  |
| --- | --- |
| 软硬件项目 | 软硬件环境 |
| 操作系统 | Win7 64位旗舰版 |
| 内存 | DDR4 8G |
| 处理器 | 英特尔Core i7-6700 3.4GHz |
| 开发环境 | Eclipse 3.2 |
| 运行容器 | Apache-Tomcat-7.0.77 |
| Java版本 | JDK 8u102 |

5.3 熔断器模块的设计与实现

5.4 服务注册中心的设计与实现

5.5 认证模块的设计与实现

5.3 本章小结

本章通过模拟应用场景下服务模型代码框架原型程序的运行，对程序的功能是否达到预期效果进行了测试验证。首先文章对模拟应用场景做了简要描述同时说明本次试验的真实软硬件环境，接着对验证原理和验证方法进行了讨论，随后文章说明了测试验证的具体流程并从两方面对原型程序进行了测试验证。最后文章对运行结果进行了分析讨论，表明本文设计实现的服务模型切实可行、有效。

6 面向舰船平台管理框架的测试与验证

6.1 平台管理框架的功能性测试

为满足我国海军信息化建设的需求，本文通过对分布式技术的研究引入面向服务的舰载指控系统应用开发方式，利用SOA技术以尽可能复用已有的系统模块，在降低系统耦合度的同时进一步简化系统开发人员的研发过程，提高指控系统的开发效率。

本人通过查阅大量国内外关于服务计算、分布式通信以及运行容器方面的技术资料，结合实际工作中对分布式构件技术的理解，针对分布式服务架构下新型指控系统所用到的SOA技术进行了研究，提出了舰载指控系统服务化体系下服务模型的概念，并从服务模型的定义以及服务模型中接口的交互机制等方面对其进行了深入的研究与分析，随后对服务模型中所涉及的一系列关键技术的设计进行了讨论，其中对代码框架的设计大量应用SOA相关技术与理论，包括：RESTFul框架、JAX-RS规范和具体的Jersey实现技术等。在文章的最后，对于服务模型的原型程序在模拟应用场景下的功能是否符合预期结果进行了测试验证，通过对测试结果的分析，证明文中对代码框架的设计以及SOA技术的使用正确且有效。

6.2 平台管理框架的非功能性测试

本文结合实际项目实践的经历，提出了服务模型的概念并对该模型在具体开发过程中存在的技术问题进行了详细的分析和设计，第五章的测试验证结果证明该设计切实有效，具有实际应用价值。但由于本人对分布式计算相关知识的理解不够深刻，对SOA相关技术的掌握程度有限，因此文中关于服务模型的设计仍然存在一些不足之处，结合现有工作成果，未来还需要在以下方面进行完善：

文中对于服务模型在Java版本代码框架中使用的Web服务技术主要依赖于REST框架，而在实际设计过程中发现REST框架并不完全适配当前设计，REST理念是对资源的直接操作，而文中服务模型是先请求服务提供的操作接口，接口中操作的业务处理才是对资源的操作。后期可以针对该模型制定一套专用的通信机制，包括：通信格式的约定、请求方式的设计、支持但不限于HTTP协议进行通信消息的传输等。

文中使用对Tomcat服务器作为运行容器，后期可以改用Jetty服务器。Jetty相比Tomcat在提供SOA服务方面除去了无用功能，更加轻量级。同时根据Jetty提供的API可以对运行容器的功能进行定制，更加灵活。

本文对于服务模型代码框架的设计主要满足于舰载指控系统功能方面的需求，对于代码框架的性能尚需进一步优化，如采用二进制方式替代Json方式序列化运行效率更高且数据更小。

7 总结与展望

7.1 论文总结

7.2 下一步工作展望

参考文献

1. 吕云飞, 王旋, 张军. 基于DDS技术的舰船平台综合管理系统[J]. 舰船科学技术, 2011, 33(5):47-52.
2. 郭蒙. 海洋工程船综合平台管理系统研究[D]. 大连海事大学, 2013.
3. 吴旭东. 舰船监控系统支撑平台研究与设计[D]. 武汉理工大学, 2013.
4. 宁小敏, 阳斌, 王奕, et al. 基于Web的船舶信息集成管理系统的设计及实现[J]. 中国舰船研究, 2011, 06(3):99-102.
5. 熊瑛, 许建. 船舶综合平台管理系统概念设计[J]. 船海工程, 2009, 38(5):36-39.
6. 许军海. 指控系统构件集成框架设计与实现[D]. 西北工业大学, 2017.
7. 董晓明, 冯浩, 石朝明, et al. 全舰计算环境体系结构和系统集成框架[J]. 中国舰船研究, 2014, 9(1):8-13.
8. Villamizar M, Garces O, Castro H, et al. 2015 10th Computing Colombian Conference (10CCC) - Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud[J]. 2015:583-590.
9. Zimmermann O. Microservices tenets[J]. Computer Science - Research and Development, 2017, 32:1-10.
10. 刘伟强. 基于Dubbo的分布式航空交通管理系统的研究与实现[D]. 2015.
11. Alshuqayran N, Ali N, Evans R. A Systematic Mapping Study in Microservice Architecture[C]// IEEE International Conference on Service-oriented Computing & Applications. IEEE, 2016.
12. Gouigoux J P, Tamzalit D. From Monolith to Microservices: Lessons Learned on an Industrial Migration to a Web Oriented Architecture[C]// IEEE International Conference on Software Architecture Workshops. 2017.
13. Francesco P D, Lago P, Malavolta I. Migrating Towards Microservice Architectures: An Industrial Survey[C]// 2018 IEEE International Conference on Software Architecture (ICSA). IEEE Computer Society, 2018.
14. Kratzke N. About Microservices, Containers and their Underestimated Impact on Network Performance[C]// Cloud Computing 2015. 2015.
15. Johannes Thönes. Microservices[J]. IEEE Software, 2015, 32(1):116-116.
16. Familiar B. From Monolithic to Microservice[M]. 2015.
17. Thomas E. SOA Principles of Service Design[M]// Soa: principles of service design. Prentice Hall Press, 2007.
18. Walls C. Spring Boot in Action[M]. 2016.
19. Fielding, Thomas R. Architectural styles and the design of network-based software architectures[C]// University of California, Irvine, 2000.
20. 冯新扬, 沈建京. REST和RPC:两种Web服务架构风格比较分析[J]. 小型微型计算机系统, 2010, 31(7):1393-1395.
21. Ismail B I, Goortani E M, Karim M B A, et al. Evaluation of Docker as Edge computing platform[C]// Open Systems. IEEE, 2016.
22. Preeth E N, Mulerickal F J P, Paul B, et al. Evaluation of Docker containers based on hardware utilization[C]// International Conference on Control Communication & Computing India. IEEE, 2016.
23. Anderson, Charles. Docker [Software engineering][J]. IEEE Software, 2015, 32(3):102-c3.
24. Ismail B I, Jagadisan D, Khalid M F. Determining Overhead, Variance & Isolation Metrics in Virtualization for IaaS Cloud[M]// Data Driven e-Science. Springer New York, 2011.
25. Stubbs J, Moreira W, Dooley R. Distributed Systems of Microservices Using Docker and Serfnode[J]. 2015.
26. Chen E Y, Pei Y, Chen S, et al. Oauth demystified for mobile application developers[C]//Proceedings of the 2014 ACM SIGSAC conference on computer and communications security. ACM, 2014: 892-903.
27. Leiba B. Oauth web authorization protocol[J]. IEEE Internet Computing, 2012, 16(1): 74-77.
28. Esposito C, Castiglione A, Choo K K R. Challenges in delivering software in the cloud as microservices[J]. IEEE Cloud Computing, 2016, 3(5): 10-14.
29. Ghofrani J, Lübke D. Challenges of Microservices Architecture: A Survey on the State of the Practice[J]. ZEUS 2018, 2018: 1.
30. Newman S. Building Microservices[M]. 2015.

致 谢

时光荏苒，转眼间到了毕业前期，回顾两年半的研究生学习生涯，我倍感怀念和感恩。首先要感谢的是我的导师，您严谨认真的治学精神和耐心和蔼的谆谆教导让我在研究生学习阶段有了很大的提高，但最重要的是您不忘在教学之余向我传授谦逊做人、踏实做事的道理，让我在走上工作岗位这一竞争更加激烈的人生道路上能够礼貌待人、脚踏实地。我觉得我将最怀念的是您和我们一起探讨计算机理论时那种不仅要知其然，还要知其所以然的研究态度，和您对我们不正确观点的耐心纠正和指引。我将把这种认真对待基础知识，戒骄戒躁的精神应用到以后的工作当中。

同时我也要感谢教研室的各位老师，感谢您们为我们创造的科研和学习氛围。从入学向我们推荐有用的计算机书籍到日常生活中对我们的嘘寒问暖，让我能够在科研的道路上做到心中有数、松弛有度。

我还要感谢教研室的5位同级的同学，和你们在一起对课堂知识的讨论、对项目的研究和对生活的憧憬将成为我在研究生阶段的美好回忆。你们和教研室其他师兄师姐和师弟师妹们一起，使我感受到一个大家庭的温暖，让我在研究生学习和生活期间不会感到孤独。

我还要感谢我的父母，正式他们的默默支持和鼓励，在我最低落的时候给予我开导和安慰，才让我在研究生阶段能够毫不气馁、勇往直前。我只有继续在今后的工作当中努力奋斗，才能回报他们的养育和教导之恩。

最后感谢在百忙之中抽出时间评阅论文的老师们，您们辛苦了！

攻读硕士学位期间发表的学术论文和参加科研情况

硕士在读期间参与的课题有：

1. ZZ软件构件研发与测试支持子系统.
2. 面向舰船平台管理服务化系统研究.

**西北工业大学**

**学位论文知识产权声明书**

本人完全了解学校有关保护知识产权的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属于西北工业大学。学校有权保留并向国家有关部门或机构送交论文的复印件和电子版。本人允许论文被查阅和借阅。学校可以将本学位论文的全部或部分内容编入有关数据库进行检索，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。同时本人保证，毕业后结合学位论文研究课题再撰写的文章一律注明作者单位为西北工业大学。

保密论文待解密后适用本声明。

学位论文作者签名： 指导教师签名：

年 月 日 年 月 日

———————————————————————————————————————

**西北工业大学**

**学位论文原创性声明**

秉承学校严谨的学风和优良的科学道德，本人郑重声明：所呈交的学位论文，是本人在导师的指导下进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容和致谢的地方外，本论文不包含任何其他个人或集体已经公开发表或撰写过的研究成果，不包含本人或其他已申请学位或其他用途使用过的成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式表明。

本人学位论文与资料若有不实，愿意承担一切相关的法律责任。

学位论文作者签名：

年 月 日