# 面向服务的柔性装配制造系统的研究与实现

# 摘 要

XXXX.

**关键词：XXXX；XXX**

# 第一章 综述

## 1.1 课题研究背景

随着德国工业4.0时代的到来，与中国制造2025的提出。制造业逐渐向“智能化”的方向发展。而“装配MES”是装配车间智能化的必要管理工具，现阶段只有大型企业有能力发展自己的装配MES，中小企业由于装配MES开发成本问题很难开发自己的系统。

应用软件的发展从传统On-Premises部署型解决方案，一步步由基础设施即服务的IaaS到平台即服务的PaaS再到现在软件即服务的SaaS应用模式。MES作为一种面向企业的应用化软件也迫切需要由传统的解决方案向着SaaS化模式发展。只有开发面向服务的装配MES系统，将MES系统SaaS化才可以解决开发成本高、开发周期长、数据共享性差等问题。

已有的面向服务装配MES系统主要是实现业务流程可配置性、可集成性、可重构性。本论文的面向服务是将柔性装配MES系统由原来传统部署应用模式向注册即用的服务模式转变，实现MES的SaaS应用模式。

## 1.2 面向服务的柔性装配制造系统研究现状

### 1.2.1 柔性装配制造系统发展现状

柔性装配系统（Flexible Assemble System 简称FAS），是在柔性制造系统（FMS）之后提出的。从60年代开始，随着计算机技术以及各种相关的关键技术的发展才逐步提出。到了80年代以后随着FMS的逐渐成熟FAS已开始快速发展[1]。据统计，装配的成本占到了工业产品的劳动成本的10-30%[2]，更是占到了制造成本的50%[2]。因此柔性装配系统是作为柔性制造的重要环节和组成部分，只有实现柔性装配系统才能快速响应客户的需求[4]。

随着社会需求的个性化和多样化，柔性装配系统的需求更加明显。在国内很多学者们进行柔性装配系统研究。陆远等[5]基于产品BOM的柔性装配工艺系统设计思想，通过对装配工艺系统的设计开发，实现了车间装配流程的柔性化与自动化。关桂齐等[6]，通过对减速器的柔性装配单元的控制系统进行研究，基于柔性装配单元的控制原理，提出了一种总线的控制方式，可以实现多种、中小批量减速器的柔性装配。蔡俊英等[7]，通过数据中心、电气控制系统、机器人、机械手等组成一套完整的柔性装配系统，实现了汽车生产多品种、小批量、柔性化、智能化的装配过程。袁立等[8]，针对飞机数字化柔性装配生产线，从装配工艺设计、数字化仿真、数字化柔性工装系统、先进的连接设备技术、数字化检测、数字化生产线辅助设备六个方面出发，提出了六项关键技术。高超等[9]，设计了一种导弹柔性装配系统，通过对导弹柔性装配系统中的柔性对接、质心测量两个关键技术对导弹的柔性装配系统进行了研究。卜泳等[10]，通过对翼身柔性装配的特点分析，根据装配过程的不同阶段，采用了分阶段研究的方法，将整个装配系统分解管进行拆分，从部分到整体建立一种翼身融合整体结构柔性装配系统。张铁等[11]，设计了一种摩托车发动机柔性装配单元，该系统通过机器人可以完成发动机气门组件柔性转配，并且应用了装配质量在线控制技术，保证了产品的质量。

在国外发达国家以机器人为核心的柔性装配系统发展要更早更加完善。例如，日本Seiko Epson公司开发的由55台SCARA装配机器人组成的手表柔性装配系统。可以实现8大类60中型号的手表柔性装配。日产量可达50000只，采用柔性装配生产线后成本比原来下降了63%[12]。日本的丰田汽车公司，首先采用混流生产方式的企业，通过生产线的柔性设计实现汽车混流生产[13]。其中Coronas混流生产线可以生产4000多种不同类型的汽车，实现每种车型的车门、变速器、发动机等部件的不同[14]。美国Westing House 公司开发的APAS系统，是在柔性装配系统基础上添加了视觉控制，实现了装配系统的柔性自适应。实现了每天产量13批600台左右，450中不同型号的电机生产[15]。

综上，柔性装配系统的发展跟国外发达国家还有一定的差距，在国外很多行业都已经进行了柔性装配系统的实际应用，用来提高工厂的生产效率，适应了市场多品种、小批量的需求趋势，实现同一个柔性生产线可以同时进行多种不同型号的产品生产的需求。国内的发展虽然起步较晚但是在国家政策下正在逐步追赶，各行业正在进行柔性装配系统的研究。

由于柔性装配系统的设计成本和研究难度问题，国内的各种柔性装配系统的发展现在主要是集中于航空、汽车等大型制造企业。中小企业的柔性装配应用还是较少，主要还是柔性装配管理系统的成本太高，所以柔性装配系统如果适应更多企业的发展，通用化和成本控制是研究的重点。

### 1.2.2 面向服务的软件发展趋势

面向服务的软件及软件的SaaS（Software as a service ,即软件及服务）模式应用，最早是1999提出，软件服务化源于1988年提出的ASP（Application Service Provider，即应用服务提供商）软件开发模式。与ASP相比SaaS的不同之处是，ASP模式需要服务提供商维护每个用户的不同版本程序[16]，SaaS只需要提供统一的新版本，具有不断升级的特性[17]。对比两种模式ASP只是提供了一种简单的网络托管模式，而SaaS则是作为一种共通用化的共享软件平台可以为多个企业用户同时提供服务[18]。SaaS化的优势就是通过统一的共享软件平台，可以减少人员、系统维护成本，使得企业只需要专注于业务需求不需要知道具体的软件实现细节[19]。同时，由于使用的统一的软件平台，也大大节省了每个企业的硬件成本[20]。

国外对于SaaS的研究比国内起步要早，从ASP提出开始就开始研究如何为企业和个人提供统一的应用服务[21]。

2003年，美国著名CRM（Customer Relationships Management，客户关系管理）系统供应商Salesforce，作为全球SaaS业务的开拓者，推出了世界上首款基于SaaS的CRM软件。到目前为止，基于SaaS的这款CRM软件已经占据了美国市场份额的90%[22]。

美国中小企业管理软件应用程序制造商NetSuite，是第一批将SaaS应用到企业级应用软件的公司之一。NetSuite的核心业务也是为企业提供CRM软件。除此之外还提供企业资源管理计划（ERP）以及电子商务（E-commerce）的软件应用开发[23]。

德国商用BI（Business Intelligence，商务智能）领域的领导者SAP公司，也推出了基于SaaS模式的产品Business By Design。作为市场上功能最全的SaaS化ERP产品，是中小型企业的ERP首选产品[24]，并且在2016年SAP公司推出了专门进行微服务推销的Yaas平台。目前，Yaas平台已经集成了电商、制造企业、食品企业、医疗等多个行业系统，并且随着微服务的发展还在逐渐增加。

目前，国外SaaS软件正在快速发展阶段，从SaaS提出开始，据AMR报告表明，SaaS软件与传统软件行业相比，每年的市场增长率超过了20%[25]。随着SaaS化软件的发展，国外企业越来越认可SaaS化软件，SaaS软件提供商也开始更多的进入市场，越来越多的企业开始考虑更换传统软件选择新的SaaS化软件。

在国内，虽然SaaS的应用起步较晚，但是随着云计算的提出，以及国外SaaS软件供应商的SaaS软件进入国内，越来越多的应用软件提供商开始由早起的ASP应用向SaaS应用发展。国内比较有名的几大管理软件供应商阿里巴巴、金蝶、用友、八百客、XTools等开始做出积极响应，加入到SaaS应用软件发展行列[26]。

XTools作为国内首批加入SaaS化模式的公司，作为国内专注于客户关系管理系统CRM软件的重要供应商，根据自身积累的软件业务基础，推出了适合国内的SaaS化CRM软件，并在2013年举办的媒体会上推出基于云端和移动CRM的首款“全掌控”移动CRM产品，作为创新性产品，不仅仅技术上使用了Salesforce相同的HTML5的技术，在通讯和跨平台技术上更是优于Salesforce[27]。

八百客与Xtools一样，也是首批提供进行CRM软件SaaS化的中国厂商，成立于2004你那的八百客，作为中国现在SaaS市场和技术的领导者，已经连续8年保持至企业管理软件的市场份额第一[28]。

用友作为亚太地区大型的企业管理软件用友ERP、CRM、人力资源管理等众多产品，用友从2008年开始全面按照SaaS的运营方式开始运营。也是首选从比较完善的CRM出发开始软件SaaS化的道路[29]。并且在2010年IBM宣布与用友达成合作共建元计算实验室，IBM技术的加入无疑加快了用友SaaS软件发展的进程[30]。

阿里巴巴作为国内最大的B2B平台，2007年开始投入巨资打造阿里巴巴软件公司（Alisoft），其核心业务就是SaaS应用。在2010阿里巴巴就涉足海外收购美国Vendio，作为阿里巴巴开拓海外SaaS市场的起步[30]。现在的阿里云服务更是提供云计算的IaaS、PaaS、SaaS所有服务，国内众多企业都是基于阿里云服务进行相关软件应用开发。

综上，虽然国内的SaaS软件发展较晚，但是现在也正逐步进入成熟期，开始与世界水平平齐，甚至有些行业已经开始处于世界领先水平。各行业都开始推出对应的服务化软件。根据2016年互联网云服务报告指出，目前SaaS仍然是一个快速发展的方向。虽然很多通用化软件CRM、OA、ERP等已经相继出现了很多基于SaaS的应用软件平台。但是像MES、PDM、CAPP等SaaS化还比较缓慢，未来是发展的热点方向。

### 1.2.3 服务型软件架构发展分析

随着互联网的发展，各行业软件需求激增，传统的软件开发模式已经不能满足日益增长的软件需求。根据摩尔定律，软件数量的激增维护成本相应的就会指数增长，所以一味增加软件数量会使得后期软件维护的开销任务巨大。为了应对日益增多的软件需求，软件的开发模式也一直在改变，在不同时期出现了不同的软件框架结构。

在应用软件开发初期比较流行的是一体化的软件开发方式。这种开发方式一般是一个团队在一个相同的开发环境下进行软件开发，所有功能位于一个工程中，然后采取集中打包发布，然后部署的开发方式。在一体化WEB开发模式中比较流行的就是常用的MVC（Model-View-Controller）模式[32]，将应用程序划分为三层结构即“模型-视图-控制器”开发模式。是一种常用的软件设计模式，通过将业务逻辑层、控制层、界面层进行解耦分离，提高了代码的灵活性和复用性。[33]

伴随着大规模的大型企业应用软件的不断发展，为了减少软件的异构性、减少软件内部的耦合，应对不断变化的新需求，出现了SOA（Service-oriented Architecture）架构以及SOAP技术。SOA是一种面向对象技术以及面向组件技术之上的更高级别抽象。早在1996年就有人开始提出SOA的概念，但是由于当时技术水平不足以支撑SOA架构的实现，所以并没有得到开发者的认可。后来随着互联技术WebService、SOAP的发展以及业务需求的日益复杂，SOA的再次出现才得到了广泛的应用[34]。SOA并没有十分标准的定义只是一种规范，代表了一种模型。模型中的逻辑被分解成了更小的逻辑单元，这些单元就组成了一个较大的业务逻辑块。目前世界上较大的公司Microsoft、IBM、SUN等都有自己基于SOA框架的解决方案[35]。

近两年随着国内的Docker容器的发展[36]，微服务架构（MicorServices Architecture）逐渐在国内软件开发领域变得流行起。由于SOA定义的模糊性以及具体实现的难度，在SOA基础之上提出了一种微服务的软件架构，微服务与SOA有着很多相似之处[37]，“微服务”一词源于Martin Fowler的名为Microservies微博。2014年给出了正式定义，微服务架构风格是一种将单个应用开发成一套小的服务的方法，每个服务在其独立的进程内运行，他们之间通过如HTTP等轻量级机制进行通信。这些服务都围绕业务进行构建，通过充分自动化的部署进行独立部署。现在比较流行的完整微服务架构有Spring开源社区的Spring Cloud以及谷歌的Kubemetes还有阿里的Dubbo，Dubbo只提供核心功能，不过作为国内自主开发架构在国内拥有企业在使用。

微服务的架构概念比较新，从企业的实践中发展而来。目前国外对微服务的框架研究较多。Alshu qayran等人[38]，对微服务的研究现状进行了综合论述，对微服务设计的负载均衡、熔断、消息等进行了总结。Hassan等人[39]讨论微服务的拆分问题，讨论了微服务粒度粗细对于系统性能和维护的成本平衡问题。Salah等人[40]对SOA单个服务于微服务的区别进行了论述，并对服务集群管理、数据存储进行了总结。Ueda等人[41]将单体系统与微服务系统的负载均衡进行了比较，指出性能上单体应用要高于微服务系统。Khazaei等人[42]就微服务的开发效率与单体模式开发进行了对比分析，表明微服务的开发效率高于单体模式，但是维护成本会有所提高。Savchenko等人[43]对微服务的测试进行研究，表明微服务测试需要更高的测试要求。国内也做了一些微服务框架研究工作，尤其是阿里的Dubbo虽然没有提供完整的微服务功能，但是基于Dubbo可以很容易选择不同开源项目完成一个微服务软件架构。

微服务的软件架构作为当前最新的软件基础框架，拥有可扩展性、快速响应、可重用性、松耦合、高容错、功能独立等多方面优势。但是没有完美的框架，微服务也面临很多问题，包括安全性、数据一致性、整体软件性能等问题。但是，瑕不掩瑜，对于现在复杂软件功能需求，尤其是SaaS软件的发展，原来越多的软件在开发时候选择微服务框架结构作为首选的开发模式。微服务被各大软件厂接受的事实也说明微服务在当今拥有的优势。

## 1.3 课题研究意义

### 1.3.1 研究内容

（1）柔性装配MES的SaaS应用模式研究

如果将传统的企业应用模式向着SaaS化的模式转变是本论文研究的核心内容，从ME系统的研究和应用发展历程来看，大致可以分为四个阶段，从20实际70年代提出专用的MES（point MES），到20世纪80年代的集成的MES（Integrated MES），再到20世纪90年代中期可集成的MES( Integratable MES，I-MES)，最后是现在智能化的MES[44]。总体上MES的发展主要是在实时性、智能性以及集成性的逐步发展。现在MES正逐渐与新兴学科联系，在近些年来，随着云智造、物联网、以及中国制造2025等概念的提出[45]，MES的服务化也正逐步兴起。

而柔性装配执行系统的发展是在90年代开始发展可集成的MES后才发展起来的。人们在制造行业自动化的设计、开发和研究在开始主要是集中在柔性制造系统上，而柔性装配系统的发展相对而言比较落后，到了80年代，随着制造系统的发展装配制造系统也获得了飞速的发展[46]。现在柔性装配制造系统除了在实时性、智能性以及集成性的发展也在逐步向服务化方向发展。所以将MES进行SaaS化是现在MES系统发展的趋势，也是未来MES系统的主要应用形式。

（2）基于微服务Spring Cloud框架的设计

Microservice架构模式就是将整个Web应用组织为一系列小的Web服务。这些小的Web服务可以独立地编译及部署，并通过各自暴露的API接口相互通讯。它们彼此相互协作，作为一个整体为用户提供功能，却可以独立地进行扩。

Spring Cloud是一个基于Spring Boot实现的微服务架构开发工具，对微服务架构提供了全面的技术支持。它为微服务架构中涉及的配置管理、服务治理、断路器、智能路由、微代理、控制总线、全局锁、决策竞选、分布式会话和集群状态管理等操作提供了一种简单的开发方式[47]。

使用Spring Cloud 框架结合Bdf2框架，设计装配MES系统的整体系统框架。可以更加方便跟其他系统进行系统集成，实现系统功能扩展。基于微服务框架可以根据需求通过网络对松散耦合的粗粒度应用组件进行分布式部署、组合和使用是实现SaaS灵活应用的基础。

（3）SaaS化MES平台设计与开发

MES系统包含基础数据管理、设备管理、任务管理、质量管理、库存管理等众多功能[48][48][47][48][48][48][48]。所有功能的SaaS化研究不是本文重点，论文主要是就如果开发一个SaaS化的MES平台框架做研究。主要是基于SaaS的基本特征多租户、可配置、注册即用来设计一个基础的应用平台，所有的功能可以基于平台再进行相应的设计。MES系统的功能可以基于微服务框架在平台基础上进行具体服务的增加，最终实现完整的SaaS化MES系统。

（4）柔性装配MES库存管理平台应用实例

课题的主要目的是面向服务的柔性装配MES的整体架构的搭建，为了测试系统架构的可行性，使用面向服务的柔性装配MES中可以单独使用的库存模块作为原型测试系统。通过设计库存管理平台并验证。

这部分内容包括设计库存模块的信息模型、功能模型、过程模型，以及描述库存模块及其组成部分的特点。基于服务的柔性装配MES的整体架构开发面向服务的柔性装配MES库存模块。

### 1.3.2 研究意义

采用微服务服务的框架Spring Cloud，以软件即服务的SaaS软件应用模式为目标，设计面向服务的柔性装配MES，满足多租户模式需求和共有云与私有云部署方式。

可以通过私有云和公共云部署来满足大型企业数据安全和中小企业对服务化MES软件的需求。

在多租户使用模式下，实现各企业制造数据的知识积累与数据共享，并且通过微服务框架的良好扩展性与低耦合性可以与其他系统进行集成，实现关联企业间数据共享，从而支持制造企业的车间装配系统内部及制造企业之间的制造数据共享。

## 1.3 论文组织结构

第一章：综述，首先是从柔性制造系统、面向服务软件、以及服务型框架三个方面介绍了面向服务的柔性装配制造系统国内外的发展现状。然后论述了柔性装配MES系统SaaS化相关研究内容，并阐明了论文研究的意义。

第二章：课题相关的理论和关键技术，分析了实现柔性装配制造系统的关键实现技术。主要柔性装配制造系统的业务通用化、微服务框架的搭建以及软件SaaS服务化等方面阐述其关键技术的具体实现方法。

第三章：系统SaaS应用模式研究，主要是从SaaS化的三个方面实现多租户模式、实现可配置性、实现伸缩性进行设计，并且讨论了SaaS的应用层、数据层及网络层的安全机制。

第四章：SaaS化MES软件平台总体设计，从SaaS化需求出发，基于微服务的软件框架结构，首先是基于Spring Cloud微服务架构进行系统框架设计，然后根据SaaS基本特征进行相关平台功能设计。

第五章：面向服务的柔性装配制造系统验证，介绍了系统的开发环境，实现了服务化的MES平台，并通过库存管理实例对系统进行了验证。

第六章：总结，总结了论文的主要研究成果以及不足之处，并对论文的下一步研究方向进行了说明。

# 第二章 课题相关的理论和关键技术

## 2.1 IaaS、PasS、SaaS应用模式研究

SaaS属于云计算服务之一，云计算技术作为一种全新的联网应用模式，根据软件发服务化程度由低到高可以分为：基础设施即服务（IaaS），平台即服务（PaaS）和软件即服务（SaaS）三类[49]。各层可独立提供云服务，下一层的架构也可以为上一层云计算提供支持[50]。越来越多的企业软件开发开始从原来的On-Premises传统软件解决方案向云计算服务中迁移。



图 1 云计算服务模式图

### 2.1.1 IaaS基础设施即服务

基础设施即服务（IaaS）作为云服务的底层，主要是提供资源服务包括IT设施、存储、网络以及其他相关的设施。IaaS所指的服务是需要用户自己在服务器上进行开发环境搭然后进行相关软件开发、打包、部署。

一般情况下IaaS服务提供商，一般提供计算机能力、存储、网络等服务的组合，打包成一个对外IaaS服务包。IaaS包括4个技术与4个业务特征。

四个技术特征

1. 1划N：可以将一台高性能的服务器，通过技术划分为多个数据隔离，相互间不影响的服务，提供给租户使用。从而充分利用计算机物力资源，提高资源利用率。
2. N划1：可以将多个物理设备虚拟化为一个高性能服务资源，提供给租户只用，租户只需要关注虚拟设备的性能不想需要知道具体实现方法。
3. 弹性：IaaS提供的虚拟化服务，可以根据用户的实际需求进行性能的动态扩展，并且可以根据用户的需求对资源进行实时修改和变更。。
4. 智能：IaaS可以提供虚拟资源的实时监控，可以支持租户应用程序的自动化部署。

四个业务特征

1. 用户获取的是IT资源：用户通过网络获取的是完整的计算机和存储设备资源，拥有完整的计算使用环境。
2. 用户通过网络获取资源：用户对IaaS资源的获取都是通过网路，不会知道实际IT设备，只能通过网络连接对应的虚拟资源。
3. 用户可以自助配置：用户通过网络访问资源可以根据用户自己实际使用需求进行先关的配重。
4. 按需计费：所有的虚拟资源是按需付费使用，用户可以根据实际需求选择按小时、按月租等不同方式选择计费方式。

一般来说IaaS服务总体技术架构主要分为服务层、管理层、虚拟化层和资源层四层结构。



图 2 IaaS的技术架构

服务层、管理层、虚拟化层和资源层四层服务是IaaS架构的基础部分，是IaaS对用户提供服务的基础。现在国际上最著名的IaaS服务提供商就是亚马逊的网络服务（Amazon Web Services，AWS），提供了存储、数据库管理、信息排队和计算机处理四项核心服务[51]。

### 2.1.2 PaaS平台即服务

平台即服务（PaaS）则是抽象掉了硬件和操作系统的细节部分，具有通用的开发环境，可以进行无缝扩展，是将软件研发平台作为一种服务。开发者只需要关注自己的业务逻辑，不需要关注开发环境等底层服务。

PaaS作为云计算中的中间层服务，也可以说为中间件及服务。虽然其处于云计算服务中的第三层但是出现的时间是最晚的。PaaS平台的主要功能是为开发者提供简化和辅助作用，主要提供的功能有：

应用开发环境：可以为开发者提供操作系统、应用服务器、语言执行环境以及分布式运行环境，提供开发需要的基本IDE、SDK等加快应用的开发、部署[52]。

应用组件池以及管理：以API的形式提供消息队列、数据库、缓存等多种功能服务。

服务、应用以及资源的管理：可以根据需要对系统性能进行随时修改，可以实现资源的监控、扩展、管理以及精确计量消耗的计算资源。

第一个PaaS云平台是Salesforce在2007年推出的force.com，用户可以在平台上轻松开发应用并且进行部署。PaaS平台的广泛推广还是2008年谷歌推出Google App Engine，它可以支持普通的Web应用开发，所以使得更多的人开始熟悉PaaS云平台。

### 2.1.3 SaaS软件即服务

软件即服务（SaaS）是软件的开发、管理、部署都交给第三方，自己不需要关心技术细节问题，只需要使用提供的服务即可。SaaS作为云计算的最顶层，可以建立在IaaS或者PaaS基础之上，也可以直接提供SaaS服务。



图 3 Iaas、Paas、SaaS关系图

作为云服务的最顶层服务，是对普通用户直接提供，不像IaaS、PaaS都是需要有开发经验才可以能感受到云服务带来的便利。所以现在软件制造商更多的是偏向SaaS化服务发展。

## 2.2 微服务软件框架设计

微服务架构作为现在复杂业务软件的主流架构，是在2012被提出（28），微服务架构作为SOA的延续，微服务在SOA定义基础上做了进一步的研究。提出了一套更加明确的框架准则，微服务的核心就是将原来一个复杂业务拆分为多个可以单独运行的服务功能模块。更传统的SOA模式有以下几个特点：

1. SOA虽然也强调服务的概念，但是还是以ESB（Enterprise Service Bus，即服务总线）服务总线为核心的。微服务是强调去ESB、分布式部署、去中心化，微服务的开发方式更加灵活、能够是开发出来的应用更加轻量级、是一种增加高效的开发模式。
2. 从服务的划分上说，微服务更加注重于服务的可用性、微服务划分的粒度，每一个单独拆分出来的微服务都可以单独使用。而SOA对于服务的要求就比较模糊，只是建议将功能服务化。
3. 从软件部署上来看，微服务可以基于容器Docker[53]进行快速部署，也可以基于内置Tomcat进行部署，每一个微服务都是运行在自己单独的线程中。SOA相比较而言需要将工程发布、打包成WAR等才能进行部署。微服务与SOA的具体比较见表1。

表 1 微服务与SOA的比较

|  |  |
| --- | --- |
| SOA | 微服务 |
| 最大应用程序服务的可重用性 | 专注于解耦 |
| 系统改变时需要修改 | 系统的改变是创建一个新的服务 |
| 重点关注业务功能重用 | 更关注边界上下文 |
| 使用企业服务总线（ESB） | 使用轻量级的协议，例如：HTTP、REST等 |
| Doker等容器不受欢迎 | Docker容器与微服务很好配合 |
| 共享数据存储 | 每个微服务有独立的数据存储 |
| 共同治理和标准 | 更注重团队协作和自由选择 |

微服务作为一种分离式软件架构。每个微服务有着独立的功能，可以进行单独部署，单独提供服务。例如一个传统的登录功能，需要登录验证、返回登录信息、登录日志等，在微服务中可能登录验证、登陆日志、信息返回都是单独的微服务。每个服务专注于自己的功能，每个服务可以单独进行开发，加快开发的效率。微服务的总体结构如图[54]。



图 4 微服务平台技术架构

### 2.2.1 微服务框架选型

从微服务框架被提出就收到了各大软件开发商、众多开发者的追捧。无数的架构师和开发者为了微服务的框架发展做出了诸多努力。围绕微服务出现很多解决方案和相关技术。

服务跟踪：国外有Spring 的Sleuth、Twitter的Zipkin等，国内有京东的Hydra等。

批量任务：国外有Spring Task、LinkedIn的Azkaban，国内有当当的Elastic-Job等。

分布式配置管理：国外有Netfix的Archaius、Spring 的Config等，国内有淘宝的Diamond、360的QConf、百度的Disconf等。

服务治理：国外有Netflix的Eureka、Apache的Consul等，国内有阿里巴巴的Dubbo、当当扩展Dubbo的DubboX等。

······

这是在微服务探索初期最先出现的一批比较成熟的微服务架构组件。可以看到大部分公司都是开发一部分微服务功能组件。如果我们需要构建一个完整的微服务架构，就需要我们组合不同公司的服务组件。由于组件不是源于统一的设计所以我们在整合各种组件时候就需要进行相应的优化修改。

目前比较流行的完整微服务框架有Spring 社区的Spring Cloud 、Google 公司的Kubemetes、阿里的Dubbo等

通过三个服务框架对比最终选择Spring Cloud作为微服务框架。

表 2 从微服的关注点比较Spring Cloud 和 Dubbo

|  |  |  |
| --- | --- | --- |
| **微服务关注点** | **Spring Cloud** | **Dubbo** |
| 配置管理 | Config | — |
| 服务管理 | Eureka、Consul、Zookeeper | Zookeeper |
| 负载均衡 | Ribbon | 自带 |
| 网关 | Zuul | — |
| 分布式追踪 | Spring Cloud Sleuth | — |
| 容错 | Hystrix | 不完善 |
| 通信方式 | HTTP、Message | RPC |
| 安全模块 | Spring Cloud Security | — |

表 3 从微服务的关注点比较Spring Cloud 与 Kubernetes

|  |  |  |
| --- | --- | --- |
| **微服务关注点** | **Spring Cloud** | **Kubrnetes** |
| 配置管理 | Config | Kubernetes ConfigMap |
| 服务管理 | Eureka、Consul、Zookeeper | Kubernetes Servies |
| 负载均衡 | Ribbon | Kubernetes Servies |
| 网关 | Zuul | Kubernetes Servies |
| 分布式追踪 | Spring Cloud Sleuth | Open tracing |
| 容错 | Hystrix | Kubernetes Health Check |
| 安全模块 | Spring Cloud Security | — |
| 分布式日志 | ELK | EFK |
| 任务管理 | Spring Batch | Kubernetes Jobs |

从上面两个表对比可以看出对比Dubbo框架，Spring Cloud框架拥有更加完备的各项微服务功能模块。对比Kubernetes框架两种框架拥有功能模块基本相当，但是对于原有Spring 工程的兼容程度和开发上Spring Cloud 要更胜一筹。所以最终选择Spring Cloud 作为微服务开发框架。

Spring Cloud 是一个基于Spring Boot实现的微服务框架。本身就具有比传统的Spring 更加便捷的开发模式。依托于Spring活跃的社区资源，现在微服务拥微服务涉及的服务治理、配置管理、智能路由、断路器、控制总线、微代理、决策精选、分布式会话和全局锁等众多服务模块可以直接使用。Spring Cloud 开箱急用的生产特性，可以大大的提高开发效率。Spring Cloud 常见组件见表4

表 4 Spring Cloud基本组件表

|  |  |  |
| --- | --- | --- |
| 组件 | 名称 | 功能 |
| Spring Cloud Eureka | 服务注册与发现组件 | Eureka 提供了服务的健康管理和友好的UI界面，可以帮助开发者轻松的实现服务的注册发现功能 |
| Spring Cloud Hystrix | 熔断组件 | 除了基本的路由熔断功能外，哈可以实现服务限流、服务降级。还使用Hystrix Dashboard 实现了熔断器的健康监控功能 |
| Spring Cloud Ribbon | 负载均衡组件 | 配合其它组件使用，实现注册中心、各个服务提供者和消费者的负载均衡 |
| Spring Cloud Zuul | 路由网关 | 基本功能通过内部的API将服务通过网关统一暴露，还可以通过安全设计增加系统安全性能 |
| Spring Cloud Config | 服务配置 | 可以实现原来的本地配置文件，使用远程仓库或者本地仓库实现配置文件统一管理功能 |
| Spring Cloud Securtiy | 服务安全 | 通过Securtiy安全模块可以搭建服务授权管理服务，实现对各个服务安全访问控制 |
| Spring Cloud Sleuth | 分布式链路跟踪 | 通过链路跟踪，可以分析服务间调用关系，对整体服务架构进行 调优 |
| Spring Cloud Stream | 数据流控制 | 可以使用数据流操作实现对消息的接受和发送 |
| ······ | ······ | ······ |

表4中列举的只是基本的组件，Spring Cloud还包含Spring Cloud Consul（服务注册发现控制台）、Spring Cloud Task（任务调度管理）、Spring Cloud Bus（消息总线）等众多组件。通过基本组件就可以搭建基于Spring Cloud微服务框架的微服务系统，架构如图4所示。



图 5 简单的Spring Cloud 架构图

### 2.2.2 微服务设计原则

（1）单一职责原则

单一职责原则作为SOLID（object-oriented-design）原则之一，指的是一个开发单元（类、方法或者服务等）只需要关注整个系统功能中单独、有界限的一部分。这样我们可以帮助我们可以进行敏捷开发。。

（2）服务自治原则

服务自治是指每个微服务应当具备独立的业务能力、依赖与运行环境。在微服务架构中，服务是独立的业务单元，应该与其他服务高度解耦。每个服务从开发、测试、构建、部署，都应当可以独立运行，而不应该依赖其他服务。

（3）轻量级通讯原则

微服务之间应该通过轻量级通信机制进行交互。轻量级通信机制应该具备两点：首先是它的体量较轻；其次是它应该是跨语言、跨平台的。下面通讯协议采用的REST协议，就是一个典型的“轻量级通信机制”。

（4）微服务粒度

微服务的粒度微服务拆分的难点，使用合理的粒度划分微服务，不能一味将服务做小。应当以服务本身的业务复杂性为标准进行服务设计。微服务之间应该相对独立并保持松耦合。微服务架构的设计是一个循序渐进的过程。常常会根据业务的变化，对微服务进行重构，甚至是重新划分，从而让架构更加合理。最终使得微服务的开发、部署、测试以及运维效率很高，并且运营成本很低。

微服务的开发作为一个循序渐进的过程，不能过于求成，在需求不是很迫切时候首先使用单体开发模式，再根据稳定版的工程进行微服务的拆分也是一个合理的技术路径。只有一个微服务设计完成后各个服务的开发、测试、部署的效率都很高，用于维护成本很低时候，才可以说是一个好的微服务框架。

## 2.3 软件SaaS服务化

### 2.3.1 SaaS 软件优势

SaaS软件模式的提出，将软件从原来基于生产制造的第二产业，转变为了面向服务的第三产业。这种软件服务方式的转变是一种革命性的变革，社会的发展主要从制造活动向着服务化的方向发展，软件行业也是面向相应的问题。SaaS化软件的出现顺应了行业发展的潮流，将会引起软件的巨大变革。

软件SaaS将会给用户和软件提供商带来明显的优势。

（1）给用户的优势

1. 拿来即用：用户不需要进行安装，无需下载，无需部署，更不需要配重。用户只需要关注软件功能即可，不需要关注软件复杂的安装和配置过程。
2. 按需使用：传统的软件大部分都是打包购买或者分版本购买，往往会购买用户不需要的功能。SaaS化的软件用户可以根据需求按照功能或者使用时间进行购买，对于用户来说节省很多不必要的开销。
3. 随处可用：一般的软件使用都是需要安装对应的客户端，需要配置对应的运行环境，SaaS化的软件只需要浏览器访问即可使用。因为网络的快速发展，只要不是极其偏远的地区，网络可以说是无处不在。正是网络的无处不在才使得SaaS在未来发展根据竞争力。
4. 先天防毒：传统软件因为部署在本地点电脑上，因为电脑防护问题病毒容易入侵。SaaS的软件是运行在软件服务商的服务器上，服务器对于病毒的防御等级要不普通客户端电脑高很多，所以SaaS软件具有先天防病毒特性。
5. 风险减少：SaaS软件拥有更容易的试用方式，对比传统软件，用户可以在确定购买前进行更好的评估，基于按需使用的理念。软件使用一段时间后也可以随时更换不会有传统软件不断购买产品的风险。
6. 成本降低：传统软件一般会分为产品费用和后期维护费用，甚至有些软件的后期维护要比购买成本还要高，SaaS化软件及服务，用户购买的只是服务，不必给软件的升级维护付出额外费用。所以及成本的SaaS软件在中小型企业中很受欢迎。

（2）给供应商带来优势

1. 节省销售成本：传统的软件在推广和销售需要投入巨大的营销成本，因为用户对于软件的各项功能需要做很多考虑。SaaS化软件用户可以根据功能按需使用，只要是基本业务一致用户就会有试用的兴趣。SaaS软件的销售会更加注重于市场的占有率，对于软件厂商来说会节省很多成本。
2. 节省维护成本：传统软件的维护、升级软件提供商需要给每个客户进行相同操作，维护人员会有很大的成本。使用SaaS模式，软件提供商只需要维护一套或者几套即可，维护人员不需要去客户现场就可以解决所有问题。使得SaaS软件在提高服务质量的同时，维护成本也大大降低。
3. 稳健的经营模式：传统软件的收入主要是来自于产品的第一次交付费用，后续的费用占到比重很少。SaaS化的软件是以服务来进行销售，软件服务商会有持续的收入，随着软件的升级对软件质量的不断提高。软件企业和客户之间可以产生良好的循环，就可以互惠互利的长期合作下去，达到双赢的目的。

任何事物都不完美，SaaS软件也是一样，SaaS也是有一些不足需要设计软件时候重点考虑。例如依赖互联网虽然现在网络基本无处不在但是对网络的依赖是SaaS一个明显不足。数据安全性与保密性，因为用户数据都在服务商服务器上，对于敏感数据的保护以及数据备份需要着重考虑。

### 2.3.1 SaaS 成熟度模型选择

根据SaaS应用是否具有可配置性、高性能、可伸缩的特征，将SaaS成熟度模型分为了四个等级。

SaaS成熟度比较表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 成熟度 | Leve l | Leve 2 | Leve 3 | Leve 4 |
| 主要特征 | 托管运营按需使用付费 | 多租户模式可配置性 | 高性价比统一维护 | 支持可伸缩性动态资源分配 |
| 关键技术 | 实现托管运营 | 可配置性租户管理 | 单实例架构安全隔离 | 负载均衡资源管理 |
| 初期投入 | 低 | 中等 | 较高 | 高 |
| 单用户成本 | 高 | 中等 | 低 | 低 |
| 目标用户 | 对个性化需求、安全性隔离要求较高大客户 | 对安全隔离性有要求，同时关注价格行业的客户 | 对价格比较敏感，安全隔离性要求不高的中小企业用户 | 大量的中小企业用户及个人用户 |

虽然从应用架构的角度，同时具备可配置性、高性能和可伸缩性的第四级SaaS成熟度模型是我们最为理性的应用架构。但是综合商业模式、实现成本及复杂度等各方面考虑，并不是所有的SaaS应用都必须达到第四级成熟度模型。具体每个应用应该选择哪个级别的SaaS成熟度模型，需要综合考虑。实际考虑因素一般分为以下三点：

* 产品所面向的客户群的特征与需求。
* 产品的租户数量级别。
* 实际开发能力与开发、改造成本之间的平衡。

### 2.3.2 SaaS 成熟度模型渐进步骤

虽然四个级别的SaaS成熟度模型是按照配置型、多租户的高性能、伸缩性的顺序逐渐增加。但是实际我们的目标都是最终的第四级成熟度模型。所以一般是按照下面四个步骤进行逐步开发。

* 实现多租户（Multi-Tenant）架构

Mutil-Tenan架构是SaaS应用的基本特征，也是实现SaaS规模的基本要素。

* 实现多租户架构下的高性能

高性能是调高用户体验，降低成本的重要需求。在没有实现伸缩性系统框架之前，优化和提升单个系统的性能。是提升系统客户规模、容量的一个重要方法。

* 实现可配置性

租户需求变化非常广，可配置性的要求范围也是相当广泛。能够满足所有租户的需求的系统几乎不可能实现。因为一般情况下来说，满足的客户需求越多，对系统的要求也就越来越多，系统的复杂度会越来越大。所以，有效的分析租户的需求变化，做出针对性的设计，实现可配置性，已取得可配置性和系统的复杂度之间的平衡才是系统可配置性设计的重点。

* 实现伸缩性

伸缩性包含了多个方面，应用层的、数据层、框架层等各个方面、一般来说应用层的伸缩性是最容易实现的一个方面。而数据库层面的伸缩性是最难实现的，数据库的分库分表是很难实现的。而且对于很多系统来说前期用户量不是很大的时候专注于应用层的伸缩性已经足够。

软件SaaS话演变模式可以用下图表示：



SaaS软件成熟度模型渐进演过的过程

## 2.4 本章小结

本章详细分析了面向服务的柔性装配系统相关理论和关键技术。软件微服务框架方面，从微服务框架的选型到具体设计的基本原则进行说明。软件服务化方面首先对云计算的三个方面IaaS、PaaS、SaaS三个方面进行分析，说明了软件SaaS是以后软件发展的趋势。最后对软件SaaS的成熟度模型进行了对比，确定了需要具体实现的功能。

# 第三章 SaaS化MES软件平台总体设计

## 3.1 系统平台总体解决方案

SaaS化MES系统平台的构建，是MES服务化的基础。基于SaaS化MES系统平台可以根据根据具体业务需求增加对应的业务功能。平台构建的基础就是基于微服务的架构设计可以满足功能性服务的按需增加。目标是实现系统的SaaS化模式，可以满足用户注册即用的使用模式。



图 6 系统总体功能图

系统整体设计分为三部分系统基础框架、服务框架。分别对应微服务的框架设计、SaaS模式设计。

## 3.2 基于微服务的架构设计

### 3.2.1 系统框架设计



服务交互层

业务逻辑层

组合服务层

RestFul

RestFul

RestFul

短信验证码服务等第三方服务

可扩展组件

持久层

展现层

图 7 基于微服务的系统架构图

作为系统的基础架构，分成展现层、组合服务层、持久层、可扩展组件4层结构。而作为核心的组合服务层内部细分为业务逻辑层和服务交互层。

（1）展现层设计

基于Dorado使用可视化的集成开发方式进行界面开发。采用RestFul将Dorado与后台服务进行解耦，将展现层与组合业务层进行分开，采用RestFul接口调用方式进行数据交互。所以在Dorado基础上可以使用Html、jsp等基本前端技术直接设计界面或者Bootstrap、React等前端开发框架。可以实现多种技术组合开发。

（2）组合业务层设计

* 服务交互层
* 微服务注册中心设计

微服务的注册中心是一个云端服务发现，一个基于 RestFul 的服务，用于定位服务，以实现云端中间层服务发现和故障转移。考虑系统的稳健性，考虑发生故障的情况，使用高可以的注册中心。

* 多服务负载均衡设计

提供云端负载均衡，有多种负载均衡策略可供选择，可配合服务发现和断路器使用。

* 服务容错保护机制设计

熔断器，容错管理工具，旨在通过熔断机制控制服务和第三方库的节点,从而对延迟和故障提供更强大的容错能力。

* 服务通讯协议设计

一般服务间通讯协议有RPC和RestFul两种。虽然RPC的调用方式可以调高服务间的数据传输效率，但是为了降低服务间的耦合性这里使用了基于Http协议的ResFul作为服务间传输协议，可以与其他程序无障碍集成。后期考虑性能RestFul支持Google开发的gRPC可以作为服务内调用规范将RestFul作为外部调用规范。

* 服务安全性设计

基于spring security的安全工具包，为应用程序添加安全控制。程序内部的传输使用自定义的安全访问协议可以在工程原有安全上进一步增加系统安全性。

* 业务逻辑层

将业务拆分为各个微服务进行单独部署，统一使用RestFul进数据交互。依托于服务交互层的负载均衡、服务熔断等提高业务服务整体性能。

（3）持久层设计

采用hibernate一级缓存、二级缓存以及Redis缓存技术来缓解数据库读写压力，提高数据交互性能。

（4）可扩展组件设计

使用基于RestFul的服务间通讯方式，可以实现跟任意第三方服务的服务互相调用。

### 3.2.2 系统微服务设计

一个完整的微服务框架一般需要具备服务注册与发现、服务的负载均衡、服务的容错、服务网关、服务配置统一管理、链路追踪六个模块。结合实际工程将微服务分为：

* 注册发现服务中心：buaa-mes-eureka
* 业务服务：buaa-mes-other
* 网关服务：buaa-mes-gatway
* 分布式配置中心服务：buaa-mes-config
* UI服务：buaa-mes-front

网关服务、分布式配置中心、分布式消息、注册中心作为系统功能型服务。业务服务根据负载均衡、熔断保护做统一设置。其中前台UI是作为展现服务，是跟用户直接进行数据交互。具体结构如图8，因为其中前台集群作为负载均衡的一种会在负载均衡中作相关介绍，所以只会对前面四个方面进行介绍。



图 8系统微服务设计图

3.2.2.1 服务注册发现设计：buaa-mes-eureka

微服务系统是由多个独立功能的服务组成，一个复杂的微服务系统通常由上百个微服务组成。例如Netflix公司的系统就是由600个微服务构成，每一个服务又由多个服务实例构成，如果没有一个统一的管理方案，服务的维护和服务间的维护将需要很大的人力资源。服务注册发现中心就是用来统一管理这些服务实例，方面查看每一个服务的状态。

服务注册是指服务向注册中心注册一个服务实例，将自己的服务名称、IP、地址提供给服务注册中心。服务发现是指同时在服务注册中心注册的服务之间的相互调用，一般是通过轻量级通讯协议例如HTTP协议进行通讯。服务的注册发现可以如图9表示。



图 9 服务的治理-服务注册和发现

目前比较主流的服务注册组件主要有Zookeeper、Etcd、Consul、Eureka等，虽然Spring Cloud 可以使用Eureka、Consul和Zookeeper作为注册和发现使用，我们这里选择Eureka。

Eureka最开始是亚马逊云计算平台AWS使用的主要组件，是Spring Cloud Netflix中的一部分。这里选择Eureka作为服务和注册主要有以下几点理由。

1. Eureka是一个开源项目，作为Netflix的开源项目之一，经过了Netflix生产环境的的检验，在基础性能和功能上都比较完备。
2. Eureka 是Spring Cloud整体框架中的首推服务和注册组件，可以与其他组件更好的对接。
3. Eureka与后面所选择的负载均衡、熔断处理、网关控制都是使用的Netflix开源系列可以，更好的搭建完整的服务管理结构。

一个完整的Erueka架构主要包括三部分。

* Eureka-Register-Service：服务注册中心，是一个单独的服务，可以采用集群构建高可用服务注册中心，提供服务的注册和发现功能。
* Eureka-Provider-Service：服务提供者，主要给其他服务提供服务。
* Eureka-Consumer-Service：服务消费者，主要是消费服务提供者提供的服务。

一般设计时候，服务提供者也会有服务消费者功能，服务消费者也可以提供对外服务。这里主要说明服务注册中心设计，具体服务提供者和消费者在业务服务中心会再做说明。服务注册中心界面如图10。

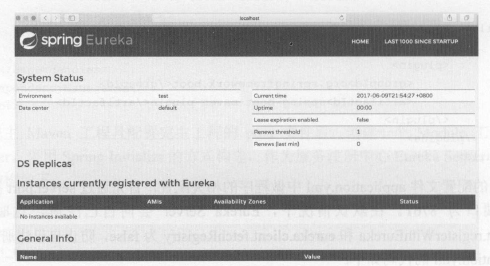


图 10 服务注册中心（需要更换）

图10展示的服务注册中心是单机注册，一般情况下考虑发生故障的情况，需要使用高可用注册中心，需要使用两个注册中心及以上，如图11，为两个注册中心相互注册。

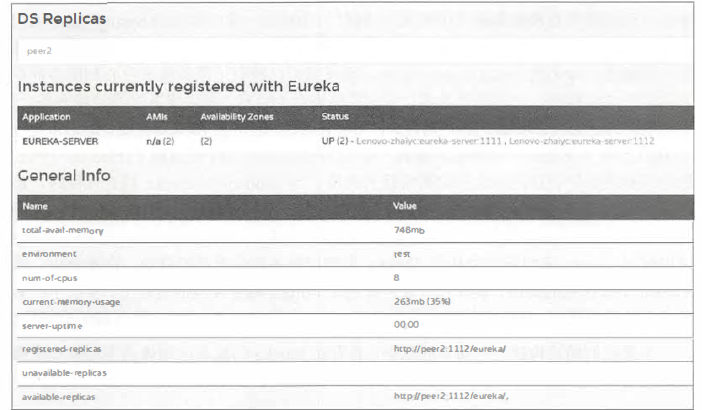


图 11 高可用注册中心（需要替换）

图12是高可用注册中心架构图。说明了多注册中心是如何保证服务故障时候服务器不至于瘫痪不可用。



图 12 Eureka 高可用架构图

每一个服务消费者和服务提供者都会向服务注册中心进行注册，并且默认每30s发送一次心跳，来续约服务。如果有服务不可用，默认90s后将会将其从服务列表剔除，服务注册中心之间是相互注册，如果有一个注册中心不可以，将会将所有消费者和服务提供者转移到正常注册中心，保障了服务注册中心的高可用性。具体使用的注册中心有许多时间、规则配置参数见表5。

表 5 Erueka系统参数配置表

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 说明 | 使用值 | |
| enabled | 启用Eureka客户端 | | TRUE |
| eureka.client.registry-fetch-interval-seconds | eureka client间隔多久去拉取服务注册信息 | | 30 |
| eureka.instance.lease-expiration-duration-in-seconds | eureka server至上一次收到client的心跳之后，等待下一次心跳的超时时间 | | 90 |
| eureka.instance.lease-renewal-interval-in-seconds | eureka client发送心跳给server端的频率 | | 30 |
| eureka.server.enable-self-preservation | 是否开启自我保护模式 | | TRUE |
| eureka.server.eviction-interval-timer-in-ms | eureka server清理无效节点的时间间隔 | | 60 |

除了表中参数还有众多参数可以配置，这里只做了部分参数调整。

3.2.2.2业务服务的负载均衡以及熔断设计：buaa-mes-other

除了功能型微服务，其他的业务型微服务都需要拥有负载均衡和熔断机制设计。一般来说负载均衡和熔断是同时使用保证服务的高可用（High Avilability）。

1. 服务的负载均衡设计

在微服务架构中，服务之间的通讯一般是通过轻量级的通讯协议，Spring Cloud使用的是HTTP协议进行通讯。因为网络因为网络延迟、阻塞等原因往往会造成系统的不稳定性。所以实际中生产环境中使用的服务往往是集群部署的。多个相同的服务提供者、服务消费者，它们之间的相互调用就涉及到了负载均衡的问题。

负载均衡从无到有到高可用负载均衡如图13所示。注册中心使用多个注册服务保证注册服务的高可用，服务提供者也高可用保证服务的高可用。从而确保了整个系统的高可用。



图 13 负载均衡模式高可用服务

框架设计时候，负载均衡采用基于Netfilx的Ribbon实现各个服务的负载均衡。虽然不像注册中心、网关、分布式配置中心是需要运行在单独的服务单元上。负载均衡几乎会使用到每个微服务中，是整个微服务的架构的基础功能。

负载均衡对于系统的高可用、缓解网络压力以及系统的扩容都是重要手段之一。负载均衡除了服务在设计时候采用软件负载均衡还有很多其他实现方法，例如硬件设备F5等，还有一些可以通过负载均衡容器例如Nginx来实现系统的负载均衡。在后面的UI服务的集群设计时候会重点介绍Nginx的使用。所有的负载均衡均可以用图14表示。



图 14 负载均衡模式

负载均衡的实现无论是通过硬件的实现还是通过软件的实现，都是通过负载均衡中心维护一个可以使用的清单，通过心跳检测机制保证清单汇总的服务可用。当有服务器的请求时候，负载均衡中心会根据某种算法（比如按照权重、线性轮询、按流量负载等）从所有可用服务汇总选择一个作为服务提供者，具体的算法选择下面会做详细论述。

因为前段UI服务使用的不是Spring Cloud 的基础框架Spring boot，加上前台的框架爱BDF2使用了Spring Security的安全验证模块导致不能跟微服务使用同一的负载均衡和熔断组件，所以单独使用其他组件进行集群处理到达前端集群的方式。考虑BDF2推荐采用Tomcat作为启动容器，所以采用跟Tomcat+Ngnx作为前端负载均衡使用。

跟后面要说的UI集群相比，采用程序控制实现负载均衡最主要的不同就是服务清单存储位置。使用容器或者设备进行负载均衡时候，所有的服务清单就会存储在设备或者容器清单中。而使用服务器的负载均衡，所有的清单维护都交给了注册中心，通过注册中心服务清单的配合，就可以更加容易的实现多服务实例的负载均衡。

结合Spring Cloud 的Eureka注册中心，在使用Ribbon时候，只需要保证所有需要使用的服务都向Eureka进行了服务注册，然后使用Ribbon功能插件包，对相关类使用@LoadBalance负载均衡过的RestTemplate来实现对应服务调用，系统会根据使用的负载均衡算法进行调用服务的负载均衡调用。

负载均衡使用的RestTemplate的使用方法一般通过两种方式进行使用分别是GET请求和PUT请求，架构所使用的的部分示例方法见表6和表7，可以满足系统的所有调用请求需要。

表 6 RestTemplate Get请求方法示例表

|  |  |  |  |
| --- | --- | --- | --- |
| 方法  类型 | 方法名 | 服务器端 | 说明 |
| get方法 | PVoid  RString | @RequestMapping("/PVoidRString")  public String PVoidRString(){  return "return-void";  } | 无参数传递使用get方法，不写请求类型默认为RequestMethod |
| PString  RString | @RequestMapping(value = "/PStringRString")  public String PStringRString (@RequestParam String name){  return "return-"+name;  } | String 参数可以使用get方法,get使用请求参数@RequestParam |
| …… | …… | …… |

表 7 RestTemplate Post请求方法示例表

|  |  |  |  |
| --- | --- | --- | --- |
| 方法类型 | 方法名 | 服务器端 | 说明 |
| Post方法 | PDomain  RString | @RequestMapping(value = "/PDomainRString" ,method = RequestMethod.POST)  public String PDomainRString(@RequestBody Employee employee){  return "return-"+employee.getEmployeeName();  } | 传递实体需要使用post方法，Post使用参数@RequestBody |
| PDomain  RDomain | @RequestMapping(value = "/PDomainRDomain")  public Employee PDomainRDomain(@RequestBody Employee employee){  Employee employeeReturn = new Employee();  employeeReturn.setEmployeeName("return-"+employee.getEmployeeName());  return employeeReturn;  } | post方法传递实体返回实体 |
| …… | …… | …… |

负载均衡策略设计，前面已经提出负载均衡中心会根据某种算法（比如按照权重、线性轮询、按流量负载等）从所有可用服务汇总选择一个作为服务提供者具体的负载策略可以分为以下几种RandomRule随机策略，RoundRobinRule线性轮询策略，WeightedEesponseTimeRule权重策略，还有综合策略ClientConfigEnabledRoundRobinRule等。在前期业务不是特别负载，各个服务的具体开销不是很明显时候，这里综合考虑使用比较通用的RoudRobinRule策略作为负载均衡策略如图15。



图 15 轮询策略实现逻辑图

具体的实现逻辑代码见附录1轮询策略实现主函数。

Spring Cloud Ribbon也是需要很多参数进行自己的服务配置，这里就不做单独使用时候具体参数介绍。实际使用一般是和Eureka进行整合使用，参数配置也会有所变化，甚至部分参数可以通过Eureka 进行配置。

负载均衡还有另外一个关键点就是重试机制，这点与微服务中常提到的CAP理论有关，即同时满足“一致性”、“可用性”以及“分区容错”是一件不可能的事情。CAP是在2000年的PODC会议上提出的，同时满足CAP的软件架构是不可能存在的，所以一般系统架构都是以其中的两个为实现重点。CAP理论图如图15。



图 16 CAP理论示意图图 17

其中Consisitency，指数据的强一致性。如果数据库写入了某一个数据，之后读取的这个数据都是最新新写入的。如果某个数据写入失败，之后所有读取的这个数据都是失败之前的正常数据。Availability，服务的可用性，只服务一直保持高可用。Partition-tolerance，分区容错，防止错误数据的出错。

Spring Cloud 注册中心与ZooKeeper的最大区别就是Spring Cloud 强调的是CAP中的AP理论，而ZooKeeper强调的是CP理论。所以为了服务的高可用Spring Cloud牺牲了一部分一致性。在服务有故障时候会选择多次重试，而不是直接剔除最大可能的保证服务的可用性。所以在使用Ribbon时候可以配置重试机制相关参数进行服务的重试配置。

1. 服务器的容错

在微服务的实际落地项目中，往往一个服务会有多个服务实例。有时候在高并发的系统中一个服务实例会有数十个。如果采用负载均衡机制访问到某个实例总是不能提供服务时候，如果不能做出对应的处理将会影响到所有使用当前实例的系统功能。如图14中所示，一个微服务有众多的服务，当其中E服务不可用时候，所有访问E服务的请求将不能返回正常的数据，进而影响使用这些请求的服务，还会影响到依赖于E的其他服务，如果E实例提供的是核心业务可能就会由于一个服务的问题导致整个系统的瘫痪，这个就是微服务中著名的雪崩效应。



图 18 服务的依赖性

为了解决服务间依可能引起的雪崩问题，分布式系统都引入了熔断机制。熔断器（Circuit Breaker）源于物理中的电路知识。它的作用是当电路故障时候迅速切断电路，从而达到保护电路的作用。熔断器的原理如图15，当一个请求的失败次数在规定阈值范围内，熔断器是处于关闭状态的。一旦出错次数超过阈值，这时候熔断器将停止服务或者进行其他设定操作，这个具体策略后面会进行详细说明，从而确保服务的稳定性。熔断器会在一定时间后处于半开状态进行服务请求测试，如果小于阈值将会关闭，反之如果还是大于设定的阈值将会继续处于开启状态。



图 19 熔断器机制

熔断器的重要性，不仅体现在防止系统雪崩上面，还有以下作用。

* 将资源进行隔离，如果只是服务的某个API出现故障导致调用失败，使用熔断器只会隔离问题接口，不会影响整个服务使用。
* 服务降级功能，当服务处于正常状态，访问量突然激增时候，熔断器也可以进行保护，防止服务瘫痪。
* 自我修复功能，服务器没有响应有时候可能是网络问题，熔断器的存在会在一定时间内自动重试，如果还是不能修复才需要人工干预。

在只用了Eureka和Ribbon的微服务工程中，在主程序目录上使用@EnableHystrix注解可以让服务自动使用默认熔断器策略。一旦服务调用出错会启动容错机制阻断故障服务，通过负载均衡转移到正常服务上，保证了线程的正常运行。

通过使用熔断器的监控状态监控可以对服务的可用性和健壮性进行监控。熔断监控器Hystrix Dashboard作为与熔断器Hystrix配合使用的图形化展示界面，可以提供友好的图形化展示界面如图20。



图 20 Hystrix Dashboard主页面（需要替换）

通过输入对应的服务地址可以进入性能监控界面图21。

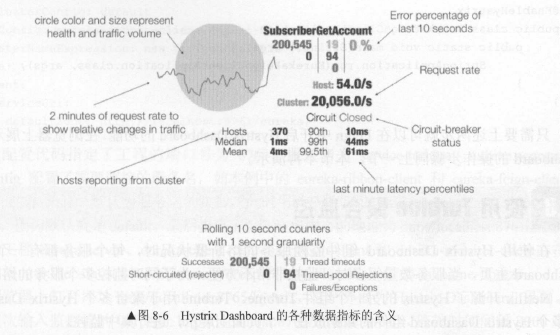


图 21 Hystrix Dashboard 的数据指标的含义（需要替换）

通过监控台的各种指标可以实时监控各个服务的安全性能，综合使用负载均衡和容错机制，微服务框架可以自动修复一部分由于网络延迟等引起的服务故障问题，其他的服务故障就需要使用监控台来进行监控处理。

3.2.2.3 网关服务设计：buaa-mes-gatway

微服务系统一般是将资源以API接口的形式对外提供服务，API接口的提供一般都是由服务网关（API网关）统一暴露。内部服务不直接对外提供API资源。这样就可以将内部服务隐藏起来。使用网关层的主要作用有：

* 网关将所有服务API资源统一管理，对外进行暴露，外部资源对内部服务资源进行调用时候只需要知道对外的接口即可。可以防止外部对内部敏感资源的调用。
* 网关可以做对调用API资源的外部服务做一些身份、权限的认证设置，可以对内部服务进行保护。
* 网关可以对服务进行实时监控，实施日志输出，对请求进行记录。
* 网关可以基于监控功能，在高流量情况下对服务进行降级。
* 使用网关，可以通过API接口方便测试。

在平台框架设计中，网关的配置是直接对服务集群进行配置，网关信息直接是对服务集群生效配合注册中心与负载均衡使用。表8是一个服务集群的网关配置实例。

表 8 网关的服务集群配置示例

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 值/参数 | | 描述 |
| zuul.routes.registerService.  path | /registerService/\*\* | 外部访问地址 | |
| zuul.routes.registerService.  serviceId | mes-registerServer | 服务名称 | |
| zuul.routes.registerService.  customSensitiveHeaders | TRUE | 是否隐藏敏感信息 | |

外部访问地址的URL格式可以参照表9按照需求设计，这里为了可以使用服务所有功能使用“\*\*”作为结尾。服务名称即为在注册中心注册的服务名称。最后一个参数是对浏览路径的保护，不会在访问中暴露内部实际服务地址。

表 9 URL路径规则表

|  |  |
| --- | --- |
| URL路径 | 说明 |
| /exampleService/？ | 它可以匹配/exampleService/之后拼接一个任务字符的路径， 比如/exampleService/a 、/exampleService/b 、/exampleService/c |
| /exampleService/\* | 它可以匹配/exampleService/之后拼接任意字符的路径， 比如/exampleService/a 、/exampleService/aaa 、/exampleService/bbb。但是它无法匹配/exampleService/a/b |
| /exampleService/\*\* | 它可以匹配/exampleService/\*包含的内容之外， 还可以匹配形如/exampleService/a/b的多级目录路径 |

通过网关的使用，我们不必关心具体服务的地址，只需要知道官网地址，以及服务的名称即可完成服务的调用，这样就避免了具体服务的改变对程序的影响。服务的地址变动只需要对网关中配置地址进行修改即可。为了修改配置不至于引起程序的修改就是下面分布式配置服务的设计。

3.2.2.4 分布式配置服务设计：buaa-mes-config

软件开发过程中往往会使用大量的配置文档，在单体架构中因为工程的单一性，配置文件一般都是在工程中直接修改。当时当使用微服务架构进行开发时候。如果没有服务的配置文件都在工程中修改，随着服务的增加将会是一个十分复杂的问题。所以在服务架构中一般使用同一的配置管理模式。常用的有阿里的Diamond、百度的Disconf、Spring Cloud 的Config组件等。平台使用的是Config文件配置管理组件管理模式如图22。使用同一配置管理的过程大致流程为：

* 首先，在配置服务启动后，通过配置信息读取远程仓库或者本地仓库中的配置信息。一般远程仓库可以使用（GitHub、Coding等）。
* 配置服务启动后，根据远程配置更新配置信息，放在服务器中备用。
* 启动服务后，服务会从配置中心获取各自配置信息。
* 远程配置信息修改过后，会通过Post请求进行刷新。



图 22 服务配置统一管理

平台的配置中心是使用的远程Git仓库作为配置管理，采用消息bus作为消息请求刷新。配置中心中的仓库配置见表10。云端配置如图23.

表 10 远程仓库配置

|  |  |  |
| --- | --- | --- |
| **参数名称** | **值/参数** | **描述** |
| spring.cloud.config.server.git.uri | <https://gitee.com/liuchenshuo/spring_cloud/> | 远程仓库地址 |
| spring.cloud.config.server.git.search-paths | spring\_cloud\_in\_action/config-repo | 配置模式 |
| spring.cloud.config.server**.**git.username | liuchenshuo@126.com | 仓库用户名 |
| spring.cloud.config.server.git.password | \*\*\*\*\*\* | 密码 |

远程仓库地址是作为远程配置地址使用，配置模式是作为远程配置目录使用。其中用户名与密码为远程仓库验证使用。

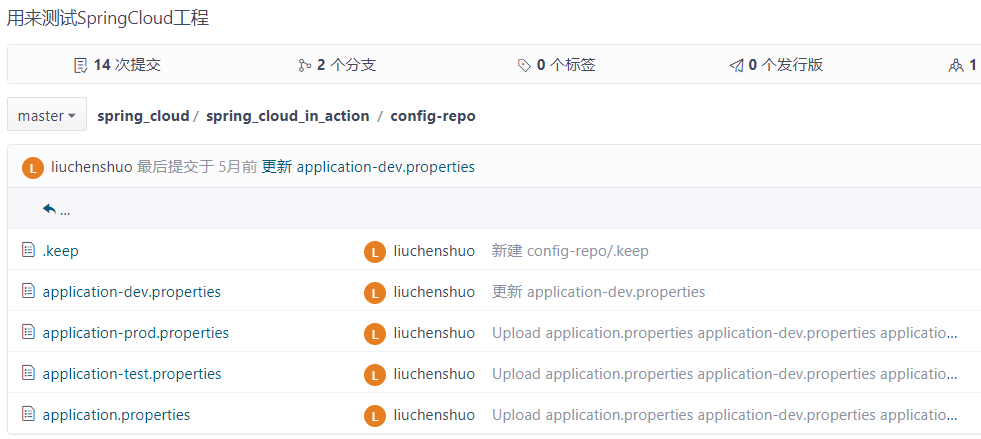


图 23 远程仓库配置

远程仓库的配置这里只是显示了基本的开发版、测试版、发布版的配置目录，一般每个服务都会有单独的配置文件，对应服务的相关阶段。

服务的刷新采用消息代理，常用的消息有ActiveMQ、Kafka、RabbitMQ等，因为Spring Cloud的兼容型问题，只支持Kafka和RabbitMQ，这里选用RabbitMQ作为消息发送。RabbitMQ是基于Erlang的高性能、可伸缩消息服务器，管理界面如图24.

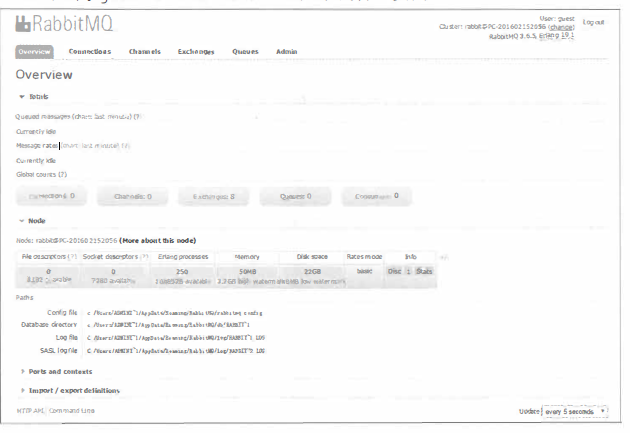


图 24 RabbitMQ主界面

通过Config中的RabbitMQ配置，结合远程仓库即可实现对应的消息刷新。其中RabbitMQ配置见表11。

表 11 Config的RabbitMQ配置参数表

|  |  |  |
| --- | --- | --- |
| **参数名称** | **值/参数** | **描述** |
| spring.rabbitmq.host | localhost | 消息服务地址 |
| spring.rabbitmq.port | \*\*\*\*\*\* | 端口 |
| spring.rabbitmq.username | springCloud | 消息服务名称 |
| spring.rabbitmq.password | \*\*\*\*\*\* | 密码 |

其中消息服务地址、端口为实际配置IP，端口，消息服务器名称和密码用于和消息服务进行验证。

### 3.2.3 传输协议REST设计

常见的微服务通讯协议有REST和RPC协议。REST全称为representational state transfer标识状态转移，是一种架构风格。RPC全称为Remote Procedure Call是一种远程调用过程，通过网络从远程计算机上进行服务请求，不需要了解底层技术协议。两者详细比较见表12。

表 12 RPC与REST对比表

|  |  |  |
| --- | --- | --- |
| **维度** | **RPC** | **REST** |
| 耦合性 | 强耦合 | 松散耦合 |
| 消息协议 | 二进制thrift、protobuf、avro | 文本型XML、JSON |
| 通讯协议 | TCP为主，也可以是HTTP | HTTP/HTTP2 |
| 性能 | 高 | 一般低于RPC |
| 接口契约IDL | Thrift、protobuf idl | Swagger |
| 客户端 | 强类型客户端、一般自动生成，可支持多语言客户端 | 一般http client可访问，也可支持多语言 |
| 案例 | dubbo、motan、tars、grpc、thrift | spring boot/mvc、Jax-rs |
| 开发者友好 | 客户端比较方便，但是二进制消息不可读 | 文本消息开发者可读、浏览器可直接访问查看结果 |
| 对外开放 | 需要转换成REST/文本协议 | 直接对外开放 |

REST与RPC相比，RPC虽然效率比REST要高，但是RPC调用显得比较繁琐，REST基于HTTP协议更规范、更标准、更通用，与其他外部平台更可以无缝链接，所以综合考虑使用REST作为传输协议。

REST是基于HTTP采用JSON进行数据传输，因为程序中需要对大量的对象数据进行序列化与反序列化，这里设计了通用化的反序列化工具类JsonUtils。下面是具体的Java代码实现见附录2JSON反序列主类。

配合JSON的反序列化类可以满足平台服务调用过程中的所有数据传输要求。

### 3.2.4 数据库连接设计

Spring Cloud微服务框架默认使用的是JPA数据库连接方式，Spring Data JPA是一种数据库连接方式，可以轻松实现服务器与数据库的数据交互。但是默认的数据库连接方式智能进行数据交互，性能一般。为了提高数据库连接性能方便数据监控在JPA之上采用的Druid的数据库连接池。

Druid是目前国内公认的最好的数据库连接池，在功能、性能、扩展性方面要远超JPA以及其他数据库连接池。Druid是阿里巴巴的开源项目，已经在阿里巴巴近600个实际项目中，经过了长时间的时间检验。服务中引入Druid相关配置包后，进行Druid配置主要属性见表13。

表 13 Druid数据池配置表

|  |  |  |
| --- | --- | --- |
| **参数名称** | **值/参数** | **描述** |
| pring.jpa.database | oracle | 数据库类型 |
| spring.datasource.driver-class-name | oracle.jdbc.driver.OracleDriver | 数据库驱动 |
| spring.datasource.url | jdbc:oracle:thin:@192.168.1.101:1521:MM | 数据库地址 |
| spring.datasource.username | MES | 数据库名 |
| spring.datasource.password | \*\*\*\*\*\*\* | 数据库密码 |
| spring.jpa.hibernate.ddl-auto | none | 是否自动更新表 |
| …… | …… | …… |

系统数据库采用的是Oracle，配置好对应的数据库驱动，再配置好数据库相关的数据库名称和密码等。表中只是列出了基本的数据库连接属性，还有众多性能相关属性这里不做展示。通过使用Druid不仅可以提高数据库连接性能还可以采用可视化的方法查看数据库连接语句的执行效率，如图

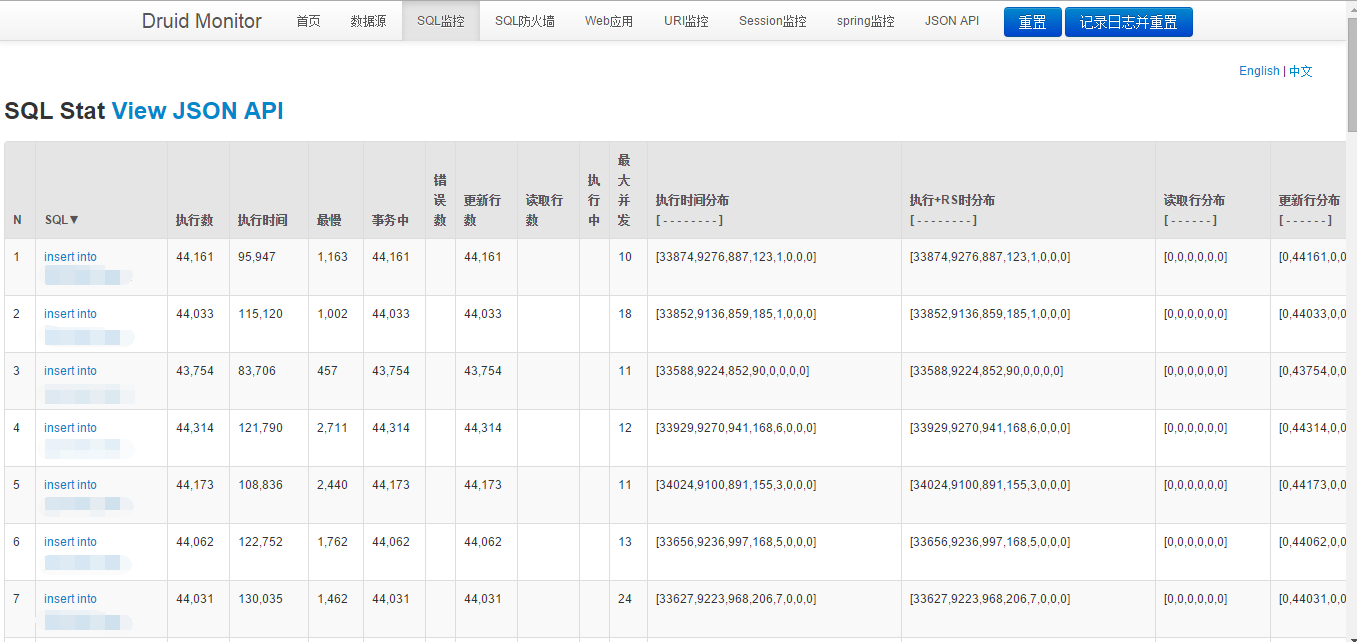


图 25 Druid监控界面

通过Druid的监控功能可以统计分析SQL的执行性能，通过SQL执行日志可以分析数据库连接的健康情况。

## 53.3 平台SaaS化详细设计

### 3.3.1 多租户设计

多租户的管理主要包括注册、订购、计费等。SaaS模式下的服务软件是通过互联网可以随时随地开通、订购和使用的产品

（1）多租户数据存储设计

3.1.2 数据隔离

数据隔离主要有完全独立模式、部分独立模式、完全共享模式三种，考虑后面方便扩张数据库和应用层面的水平扩展这里选用共享数据库模式。所以在数据库设计层面选用共享数据库、共享表模式，用户采用预留字段模式方便后面系统的水平扩张，因为使用了完全共享模式，所以对于多租户这里采用所有相关表添加租户字段。这种模式虽然会降低一部分系统性能，但是考虑实际并发级别不会太高，综合比较这种方式将会是一种比较经济实用的。通过这样数据库的设计模式还可以支持数据库和应用服务器的水平扩展模式。

（2）多租户用户管理

### 3.3.2 系统可配置设计

（1）功能可配置性

通过租户管理订阅服务实现租户对原子功能的取舍、可以通过微服务进行功能配置与定制化开发。

（2）界面可配置性

租户可在基础功能界面上自定义界面展示效果，对界面布局进行微调。

（3）数据可配置性

租户可在一定约束下设定各自系统运行参数

（4）流程可配置性

### 3.3.3 系统基础功能设计

（1）数据导入功能

（2）协同管理功能

## 3.4 系统安全设计

### 3.4.1 应用安全

为了租户在使用SaaS应用时候能享受到与传统应用一样的高可用性和高可靠性。应用安全包括身份认证、权限管理、应用监控等方面采取安全测试。

（1）身份认证

登录采用集中式认证方式，前台框架使用BDF2框架，BDF权限模块采用Spring Security3实现，并对Spring Security3作了一定的扩展，利用Spring Security3来保护系统认证安全。

（2）权限管理

权限管理，是实现用户使用系统功能的访问控制，确保有效用户可以正常使用系统的同时，可以有效的方式非法的用户和无权限的用户对系统功能进行使用。与传统应用的权限管理不同，SaaS的权限管理分为对租户的权限控制，和租户内部权限控制。

### 3.4.2 数据安全

（1）数据隔离

因为采用了共享数数据库、共享表模式，数据隔离显示更加重要，为了数据的安全性，将所有的数据表都通过对应的租户Id进行隔离。确保了每个租户数据的独立性。

（2）数据加密

对于一些敏感数据，例如系统登录的密码采用了MD5或SHA加上SALT进行二次机密存储方式，可以有效防止SQL注入，极大地增加了用户密码的安全性。

### 3.4.3 网络安全

（1）网络传输安全

传输协议从原来的http协议转换为https协议，采用ssl证书对数据传输进行加密。

（2）攻击防范

为了保证SaaS系统安全，尽量减少暴露在公网上的内容，只把系统的界面部分进行暴露，后台服务与前台进行隔离，防止对后台的攻击。

（3）网络监控

监控SaaS系统可能出现的故障，并在第一时间进行报警，后面采用微服务的框架，可以进行所有服务的集中监控，通过负载均衡熔断机制增加系统的稳定性。

## 4.4 本章小结

# 第四章 面向服务的柔性装配制造系统验证

## 4.1 系统开发环境与技术

### 4.1.1 系统开发环境

* Windows7、8、10 32/64位操作系统
* Java语言开发工具包(JDK1.8+)
* Eclipse J2EE版本+Dorado插件
* Google浏览器、360浏览器或者使用谷歌内核浏览器
* Oracle11g数据库

### 4.1.2 系统运行环境

（1）客户端

* 操作系统没有限制
* Google浏览器、360浏览器或者使用谷歌内核浏览器

（2）服务端

基于SaaS应用环境用户只需要客户端进行远程访问。

SaaS服务提供者系统运行环境：

* 前台服务服务器：
* apache-tomcat-8.0.26 服务容器
* Java语言开发工具包(JDK1.8+)
* 后台微服务服务器：
* Java语言开发工具包(JDK1.8+)
* 数据库服务器：
* Oracle11g数据库
* Redis缓存

## 4.2 系统平台管理功能

超级管理员是管理各个租户的最高权限，所有操作都是针对所有租户的基础操作，可以进行系统基础定义。

### 4.2.1系统基础参数设置

可以对系统默认的一些统一报表地址、各个服务地址、一些基本参数进行设置。还可以对公司间的关联关系进行管理，以便于公司间数据交互。

### 4.2.2系统文件管理

可以对基础数据、员工等需要使用的上传文件模板进行管理。

### 4.2.3业务模式设置

可以设定不同的业务模式，每种业务模式的界面，已经对应的人员角色等可以进行配置。

### 4.2.4查看租户

可以查看现在系统的所有租户统计信息例如:租户详情、租户的员工人数等。

## 4.3 装配库存管理实例

### 4.3.1 多种模式应用

超级管理员可以进行库存模式配置，对每个模式的功能进行编排，租户注册可以选择跟自己业务合适的模式进行注册，系统会根据选择生成对应的系统菜单。

### 4.3.2 基于微服务的功能定制

对于需要定制功能的用户，可以在前台、后台服务两个方面进行定制，通过自身系统参数、菜单配置进行使用。

### 4.3.2 基础功能

（1）页面配置

每个用户可以根据自己实际使用要求进行界面中表格和其他控件的调整。

（2）组织结构管理

根据租户实际情况建立公司组织机构。

（3）员工信息管理

根据租户实际情况添加自己员工信息，支持模板导入。

（4）基础数据管理

根据租户实际情况添加基础数据，支持模板导入。

（5）库存管理

根据选择模式进行库存功能使用。

## 4.4 本章小结

# 第五章 总结

# 致谢

值此论文完成之际，两年半年弹指一挥间，有太多的感触。首先向导师杨建军老师表示由衷的感谢！本论文是在杨老师精心地指导下完成的。杨老师严谨的学术作风、渊博的学识和认真的工作态度深深感染并鼓励着我，使我在学习上受益匪浅，一直是我学习的榜样。

此外，还要感机械自动化学院720系的全体老师，他们敬业精神和辛勤的工作为我们创造了学术氛围浓厚的习和科研环境。感谢你们陪我共同度过人生最美好的两年半时光，是你们让我从幼稚走向成熟，衷心的感谢你们

感谢邓老师这两年半时间上在学习和课题上给予我的指导和启发。感韩宝安、周勇、方工程、黄壮师兄以及已经毕业的张晨辉、刘欣、余梦雨、文波、蒋彧琛等各位师兄在开发与课题上给予我的指导和帮助。感谢同窗两年半的温春生、雷宇翔在学习上给予我的支持和帮助。感谢我的室友田博显、彭振龙是你们在生活中给我不断的鼓励。感谢所有的同学和朋友，认识你们是我一生的财富，那些我们共同走过的日子必将成为我们人生最美好的回忆。同时我要向我的父母致以最诚挚的谢意，是你们的养育、期望和支持，使我顺利地完成学业。

最后，特别感谢在百忙之中评阅论文和参加答辩的各位老师！

# 附录一 负载均衡轮询策略代码

**public** Server choose(ILoadBalancer lb, Object key) {

**if** (lb == **null**) {

log.warn("没有负载均衡！");

**return** **null**;

}

Server server = **null**;

**int** count = 0;

**while** (server == **null** && count++ < 10) { // retry 10 次

List<Server> reachableServers = lb.getReachableServers();

List<Server> allServers = lb.getAllServers();

**int** upCount = reachableServers.size();

**int** serverCount = allServers.size();

**if** ((upCount == 0) || (serverCount == 0)) {

log.warn("没有可用的负载均衡服务" + lb);

**return** **null**;

}

**int** nextServerIndex = incrementAndGetModulo(serverCount); // incrementAndGetModulo方法内部使用nextServerCyclicCounter这个AtomicInteger属性原子递增对serverCount取模得到索引值

server = allServers.get(nextServerIndex); // 得到服务器实例

**if** (server == **null**) {

Thread.yield();

**continue**;

}

**if** (server.isAlive() && (server.isReadyToServe())) {

**return** (server);

}

server = **null**;

}

**if** (count >= 10) {

log.warn("尝试10次后仍然没有可用服务 "

+ lb);

}

**return** server;

}

# 附录二 JSON反序列化主类

**public** **class** JsonUtils {

//对象转JSON字符串

**public** **static** String object2Json(Object obj) {

String result = **null**;

**try** {

ObjectMapper objectMapper = **new** ObjectMapper();

result = objectMapper.writeValueAsString(obj);

} **catch** (IOException e) {

e.printStackTrace();

}

**return** result;

}

**public** **static** Map object2Map(Object obj) {

String object2Json = *object2Json*(obj);

Map<?, ?> result = *jsonToMap*(object2Json);

**return** result;

}

//JSON字符串转Map对象

**public** **static** Map<?, ?> jsonToMap(String json) {

**return** *json2Object*(json, Map.**class**);

}

//JSON转Object对象

**public** **static** <T> T json2Object(String json, Class<T> cls) {

T result = **null**;

**try** {

ObjectMapper objectMapper = **new** ObjectMapper();

result = objectMapper.readValue(json, cls);

} **catch** (IOException e) {

e.printStackTrace();

}

**return** result;

}

//返回的list对象妆化为对应实体对象

**public** **static** <T> List<T> conveterObjectAll(List<Object> objectList, Class<T> destObjectType) **throws** Exception {

// **TODO** Auto-generated method stub

List<T> returnList = **new** ArrayList<T>();

**for** (Object object : objectList) {

returnList.add(*conveterObject*(object,destObjectType));

}

**return** returnList;

}

**public** **static** <T> T conveterObject(Object srcObject, Class<T> destObjectType) {

String jsonContent = *object2Json*(srcObject);

**return** *json2Object*(jsonContent, destObjectType);

}

}

# 参考文献

**References:**

[1]. 姚志良, 柔性自动装配的现状和动向. 组合机床与自动化加工技术, 1994(04): 第42-45页.

[2]. Mosemann, H. and F.M. Wahl, Automatic decomposition of planned assembly sequences into skill primitives. IEEE Trans.robot.autom, 2001. 17(5): p. 709-718.

[3]. Sun, Y.M., Optimization Design for the Flexible Assembly Systems Incorporated with Human and Machines. Journal of South China University of Technology, 2001.

[4]. Joshi, S., E. Mettala and R. Wysk, CIMGEN—A COMPUTER AIDED SOFTWARE ENGINEERING TOOL FOR DEVELOPMENT OF FMS CONTROL SOFTWARE. A I I E Transactions, 1992. 24(3): p. 84-97.

[5]. 陆远与刘洁, 基于产品BOM的柔性装配工艺系统. 南昌大学学报(工科版), 2015. 37(02): 第146-150+154页.

[6]. 关桂齐等, 减速器柔性装配单元控制系统实验研究. 黑龙江科技信息, 2013(02): 第16-17页.

[7]. 蔡俊英, 汽车制造企业柔性自动化装配系统. 中国设备工程, 2018(01): 第107-109页.

[8]. 袁立, 现代飞机数字化柔性装配生产线. 航空科学技术, 2011(05): 第1-4页.

[9]. 高超, 导弹柔性装配系统关键技术研究, 2014, 哈尔滨工业大学.

[10]. 卜泳等, 翼身融合整体结构柔性装配技术规划研究. 航空制造技术, 2013(20): 第44-45+58页.

[11]. Zhang, T., M. Shao and C.X. Xie, On-line Control of Assembly Quality for Locks of Motorcycle Engine Gas Valve. Journal of South China University of Technology, 2000. 22(2): p. 146-53.

[12]. Yasunri, Y. Automatic assembly line of many type watches. in Proceedings of the 1st Asian Conference on Robotics and Its Application. 1991. Tokyo.

[13]. 曹振新等, 混流装配线负荷平衡与投产排序的优化研究. 信息与控制, 2004(06): 第660-664页.

[14]. Chow, W.M., Assembly Line Design: Methodology and Applications. 1990: M. Dekker.

[15]. Hartley, J., Robots at work :: a practical guide for engineers and managers: IFS.

[16]. 阚振, 竞争情报软件云时代发展分析——SaaS的应用. 情报科学, 2012. 30(10): 第1461-1464页.

[17]. 刘静, SaaS模式下计算机软件版权保护的困境与出路, 2015, 西南政法大学.

[18]. 经渊与郑建明, 基于SaaS的社会数字文化一体化门户服务模型研究. 情报科学, 2016. 34(09): 第31-35页.

[19]. 郭海玲与严建援, 基于SWS定制化软件实施周期与SaaS标准化软件种类的竞争模型. 系统工程, 2016. 34(11): 第131-138页.

[20]. 黄桂梅, SaaS的运营模式和定价策略研究, 2010, 西安电子科技大学.

[21]. 刘德平与李小林, ASP在中小制造企业中的应用及其前景. 商场现代化, 2008(07): 第118-119页.

[22]. 马文婷, SaaS商业模式创新评估研究, 2016, 华中科技大学.

[23]. 尚昭, 看好中国SaaS市场商机 NetSuite整装待发. 中国计算机用户, 2009(05): 第75页.

[24]. 姚磊, SAP发力SaaS. 程序员, 2010(04): 第27页.

[25]. 邢蕊, 基于SaaS的天津市制造业信息化平台的研究, 2012, 天津理工大学.

[26]. 霍小军, SaaS模式在电子政务中的应用初探. 电子政务, 2008(01): 第73-87页.

[27]. Xtools转型双核心战略:云端+移动CRM. 电子技术与软件工程, 2013(10): 第10-11页.

[28]. 端木云骢, 云服务供应链产品定价策略研究, 2013, 南京大学.

[29]. 用友SaaS新服务客源滚滚赢天下. 办公自动化, 2009(12): 第19-20页.

[30]. IBM、用友建立云计算联合实验室. 中国信息安全, 2010(06): 第78页.

[31]. 王冰睿, 阿里巴巴首次涉足海外收购 马云SaaS业务在美国再起步. IT时代周刊, 2010(15): 第60-61页.

[32]. Jivani, P., C. Chopara and M. Prashant, Over All Idea about MVC: How to use Model- View-Controller (MVC).

[33]. 渠连恩与赵珊, 基于J2EE和MVC模式在B/S系统开发中的应用研究. 电脑编程技巧与维护, 2013(08): 第71-72页.

[34]. 刘永青, 使用面向服务的体系架构(SOA)构建集团CRM系统. 科技情报开发与经济, 2007(25): 第232-233+240页.

[35]. 朱振杰, SOA的关键技术的研究与应用实现, 2006, 电子科技大学.

[36]. 高礼与高昕, Docker技术在软件开发过程中的应用研究. 软件, 2016. 37(03): 第110-113页.

[37]. Thönes, J., Microservices. IEEE Software, 2015. 32(1): p. 116-116.

[38]. Alshuqayran, N., N. Ali and R. Evans. A Systematic Mapping Study in Microservice Architecture. in IEEE International Conference on Service-Oriented Computing and Applications. 2016.

[39]. Hassan, S. and R. Bahsoon. Microservices and Their Design Trade-Offs: A Self-Adaptive Roadmap. in IEEE International Conference on Services Computing. 2016.

[40]. Salah, T., et al. The evolution of distributed systems towards microservices architecture. in Internet Technology and Secured Transactions. 2017.

[41]. Ueda, T., T. Nakaike and M. Ohara. Workload characterization for microservices. in IEEE International Symposium on Workload Characterization. 2016.

[42]. Khazaei, H., et al. Efficiency Analysis of Provisioning Microservices. in IEEE International Conference on Cloud Computing Technology and Science. 2017.

[43]. Savchenko, D.I., G.I. Radchenko and O. Taipale. Microservices validation: Mjolnirr platform case study. in International Convention on Information and Communication Technology, Electronics and Microelectronics. 2015.

[44]. 王晋, 制造执行系统的研究现状和发展趋势. 兵器装备工程学报, 2016. 37(02): 第92-96页.

[45]. 李卫东, “工业4.0”对推进“中国制造2025”的启示, 2017, 外交学院.

[46]. 魏东源与朱文坚, 柔性装配系统的总体评价. 机械, 1999(04): 第6-8页.

[47]. 王方旭, 基于Spring Cloud实现业务系统微服务化的设计与实现. 电子技术与软件工程, 2018(08): 第60-61页.

[48]. 张庆锋, MES系统的应用与研究, 2008, 辽宁科技大学.

[49]. 巴拉提艾斯卡尔, 浅论云计算服务层面-SAAS,PAAS,IAAS. 电脑迷, 2017(13).

[50]. 刘瑶, PaaS云平台技术研究与应用, 2015, 长安大学.

[51]. 俞乃博. 云计算—IaaS服务模式探讨. in 中国通信学会信息通信网络技术委员会2011年年会. 2011. 中国河南郑州.

[52]. Erl, T., R. Puttini and Z. Mahmood, Cloud Computing: Concepts, Technology & Architecture. 2013: Prentice Hall Press. 236 - 239.

[53]. Yousif, M., Microservices. IEEE Cloud Computing, 2016. 3(5): p. 4-5.

[54]. 孙杰贤, 从YaaS看传统软件企业转型. 中国信息化, 2016(08): 第60-61页.