

**毕业设计（论文）**

题 目 云环境下的外部服务接入系统设计与实现

专 业 计算机科学与技术

学 号 140410130

学 生 姬喜洋

指 导 教 师 张小东

答 辩 日 期

**摘要**

随着互联网技术的发展，现在已经进入云计算时代，传统的WEB开发模式已经逐渐被抛弃，取而代之的便是微服务开发模式。在微服务架构中，一个大应用被拆分为多个小的服务提供出来，而且这些小的系统甚至可以拥有自己的数据库框架和编程语言，小系统之间通过暴露RestApi风格的接口被各种第三方应用程序调用。为了进一步降低开发者开发自己应用程序的难度，最理想的情况应该是开发者需要什么样的服务就去调用一个API接口传入参数后获得返回数据，然后自己将各种API的返回结果进行自己的包装即可。但是一旦一个应用依赖的微服务变得很多而且这些微服务是第三方服务不能受自己控制时，对这些外部服务的管理就会变得很复杂。因此我们需要一个统一的外部服务网关来负责外部服务的接入与管理。

本文将引入外部服务接入系统的概念，通过介绍微服务架构来说明设计外部服务接入系统的必要性；通过需求分析与样例分析，逐步设计出一个云环境下的外部服务接入系统，最终实现外部服务接入、多服务源智能路由、外部服务记账与监控功能。

**关键词：**API网关；外部服务；微服务；智能路由；

**Abstract**

**Keywords:** B/S,C/S,Online Q&A, check-in, classroom interaction

目录

[摘要 I](#_Toc485906454)

[Abstract II](#_Toc485906455)

[第1章 绪论 1](#_Toc485906456)

[1.1课题背景 1](#_Toc485906457)

[1.2研究的目的和意义 1](#_Toc485906458)

[1.3国内外在该方向的研究现状及分析 2](#_Toc485906459)

[1.3.1国外现状及分析 2](#_Toc485906460)

[1.3.2国内现状及分析 3](#_Toc485906461)

[1.4该课题研究的主要内容 3](#_Toc485906462)

[1.5 本文结构 3](#_Toc485906463)

[第2章 需求分析 5](#_Toc485906464)

[2.1场景分析 5](#_Toc485906465)

[2.2需求描述 5](#_Toc485906466)

[2.3系统需求分析 6](#_Toc485906467)

[2.3.1 课前用例分析 6](#_Toc485906468)

[2.3.2上课用例分析 9](#_Toc485906469)

[2.3.3课后用例分析 11](#_Toc485906470)

[2.4本章小结 14](#_Toc485906471)

[第3章 系统设计 15](#_Toc485906472)

[3.1系统概要设计 15](#_Toc485906473)

[3.2系统体系结构 16](#_Toc485906474)

[3.3系统详细设计 17](#_Toc485906475)

[3.3.1 注册流程设计 17](#_Toc485906476)

[3.3.2 登陆流程设计 18](#_Toc485906477)

[3.3.3 学生认证流程设计 19](#_Toc485906478)

[3.3.4 签到流程设计 20](#_Toc485906479)

[3.3.5 试卷预下载流程设计 20](#_Toc485906480)

[3.3.6 答题流程设计 22](#_Toc485906481)

[3.4数据库设计 22](#_Toc485906482)

[3.5本章小结 26](#_Toc485906483)

[第4章 系统实现与测试 27](#_Toc485906484)

[4.1学生端基础库的实现 28](#_Toc485906485)

[4.1.1消息框架模块实现 28](#_Toc485906486)

[4.1.2缓存模块实现 30](#_Toc485906487)

[4.2学生端业务模块的实现 31](#_Toc485906488)

[4.2.1学生认证的业务模块实现 31](#_Toc485906489)

[4.2.2课堂讨论模块实现 34](#_Toc485906490)

[4.2.3学生课程概要及详情模块实现 35](#_Toc485906491)

[4.2.4试卷预下载模块实现 37](#_Toc485906492)

[4.2.5测试/练习模块实现 40](#_Toc485906493)

[4.2.6签到模块实现 42](#_Toc485906494)

[4.2.7欢迎页面/登陆/注册实现 46](#_Toc485906495)

[4.2.8主界面实现 45](#_Toc485906496)

[4.3教师端的业务模块实现 51](#_Toc485906497)

[4.3.1课程/课堂管理模块 51](#_Toc485906498)

[4.3.2课堂讨论模块 54](#_Toc485906499)

[4.3.3题库管理模块 55](#_Toc485906500)

[4.3.4试卷管理模块 56](#_Toc485906501)

[4.4软件开发环境及工具 57](#_Toc485906502)

[结 论 59](#_Toc485906503)

[参考文献 60](#_Toc485906504)

[致 谢 61](#_Toc485906505)

1. 绪论

## 课题背景

在云计算和微服务架构出现之前，网络应用开发的难度都比较大。在传统的WEB开发方式中，所有的功能打包在一个WAR包中，除了WEB应用运行需要的容器相关的依赖外，基本没有外部依赖，部署在一个容器（例如TOMCAT）中，包含了DAO、Service、UI等所有的功能逻辑。这种方式的优点便是开发简单、集中管理，但是缺点也是显而易见的，具体包括：（1）效率低：开发都在一个项目，容易产生冲突；（2）维护难：代码耦合度很高，这会让新人无法下手，同时开发人员开发的同时必须不断和运维人员交流协作而不能专心于应用的开发；（3）稳定性差：一个小问题就可能导致整个应用崩溃；（4）扩展性不够：无法满足高并发下的业务需求。显而易见，传统的应用开发方式对开发人员是及其不友好的，而且在人们对信息的需求越来越巨大的今天，传统的开发方式会带来非常差的用户体验.云计算和微服务架构的出现大大改善了这一状况。微服务架构的目的便是：有效的拆分应用，实现敏捷开发和快速部署。总结了一下网上对于微服务架构的定义，其应该包括：（1）一些独立的服务共同组成大的系统；（2）每个微服务单独部署，跑在自己的线程中；（3）每个微服务都负责自己独立的业务；（4）微服务之间彼此隔离。使用微服务架构可以大大降低WEB应用开发人员的负担，同时开发出适用于云计算环境的服务组件。

## 研究的目的和意义

云计算强调的是一种服务租用模式，即开发人员可以直接租用云上已经做好的外部服务，通过按使用量计费的方式来直接享用外部服务提供的计算资源而不是自己再去实现相应的组件逻辑，这一点与微服务架构是契合的。

一个WEB开发者想使用微服务架构去开发自己的WEB应用时大概会经过这么几个过程：（1）需要什么服务？哪些服务是自己可以实现的，哪些服务是需要依赖别人做好的的？或者说把这种服务叫做外部服务；（2）开发者要去哪里得到那些我想要的外部服务？（3）怎么管理这些外部服务？如果这些外部服务有的突然不能用了或者说变化了怎么处理？(4)若外部服务的协议格式和需要的的不一样，开发者要自己做转换。如果对每一个外部服务都要单独进行配置和管理，这个工作将会非常繁琐。为此提出外部服务接入系统的概念，主要目的是打造一个高可用的服务网关，统一管理外部服务，从而减轻开发人员工作负担，简化外部服务接入流程。需要强调的是，本系统在整个云环境下应该也被看成一个服务。

## 国内外在该方向的研究现状及分析

### 国外现状及分析

国外的SAP公司在2016年推出了Yaas平台[1]，这是一个基于云的微服务生态系统，他让开发者可以自由选择任何技术应用在每一个微服务中。开发者本身能力有限，他们依靠个人力量可能无法解决一些复杂的问题，但是当各种各样的开发者聚集在一个社区时，他们一起创造、贡献，可以轻易从其他的开发团队收益，软件的部署流程进一步简化。

API网关方面，目前国外有一些非常好的开源的案例可以参考：（1）Kong Server：Kong是一款基于Nginx\_Lua模块写的高可用，易扩展由Mashape公司开源的API Gateway项目。KONG本身提供包括HTTP基本认证、密钥认证、CORS、TCP、UDP、文件日志、API请求限流、请求转发及NGINX监控等基本功能。目前，Kong在Mashape管理了超过15,000个API，为200,000开发者提供了每月数十亿的请求支持。（2）Netflix zuul[4]: Zuul是一种提供动态路由、监视、弹性、安全性等功能的边缘服务,是Netflix出品的一个基于JVM路由和服务端的负载均衡器。(3) apiaxle[5]: Nodejs 实现的一个 API 网关。这些开源的网关都有不同的适用条件，其中Kong Server在性能方面比较好但是维护起来困难，学习成本高，短期内的项目开发并不适用。Zuul是由java编写的，底层采用的容器技术可能会对其性能有一定的影响，但是Zuul本身和Springcloud整合得比较好，本身也实现了过滤机制，非常适合作为一个简单的反向代理替代产品。

### 国内现状及分析

国内方面我找到了两个比较成熟的API网关产品，一个是阿里云的API网关，一个是成都小豹科技公司的小豹API网关。

阿里云的API网关中的第三方API分两种，一种是在阿里云API市场上架的可以公开购买使用权的API，另一种是没有上架API市场，但是API提供商可以将API接入网关后提供给合作伙伴，从而利用阿里云API网关提供的一些服务。在阿里云的官方文档中，说明了阿里云API网关提供的以下服务：

1. 提供防攻击、防重放、请求加密、身份认证、权限管理、流量控制等多重手段保证 API 安全，降低 API 开放风险。
2. 提供 API 定义、测试、发布、下线等全生命周期管理，并生成 SDK、API 说明文档，提升 API 管理、迭代的效率。
3. 提供便捷的监控、报警、分析、API 市场等运维、运营工具，降低 API 运营、维护成本。

可以看出来，阿里云是面向商业的API网关，其有一个巨大的云API市场，其中的API产品有免费的也有1分钱/100次这种类型收费的API。阿里云后台通过API分组来管理API，API授权分两种情况：（1）用户直接从云市场购买API，购买后用户直接获得该API的使用权，但是需要将该使用权授权给自己的APP后才能正常访问API，也就是说API的授权单位是App而不是用户个人，最后App使用自己的AppID与其他约定好的密钥信息去访问API获得数据。（2）用户不是从云市场购买API，而是直接在线下直接联系API提供商构成合作伙伴关系，用户提供AppID给API提供商，API提供商在自己后台通过检索AppID来直接授予相应应用使用该API的权限。这种情况其实是用户与API提供商利用阿里云API网关提供的服务而已。

相比阿里云API网关的大手笔，小豹API网关则显得更为小巧，我们可以看到一些其技术细节。在小豹API网关的帮助文档中[7]提及了其架构图（图1-3-2）：

1. 基于Netty NIO的响应式架构；分布式缓存基于Redis；数据库基于Mysql，分布式配置基于ZooKeeper
2. API配置缓存，运行时不依赖DB，配置更新后自动通知各网关节点；
3. 支持自定义组件，动态加载，在不中断网关服务的情况下重新加载配置和运行组件；
4. API服务连续异常后自动熔断和自我恢复，访问异常、超时处理；
5. 网关核心运行过程不写磁盘IO，避免磁盘IO性能影响网关吞吐量；
6. Docker容器化支持，拆分网关、管理服务、第三方中间件依赖等镜像，便于灵活扩容。

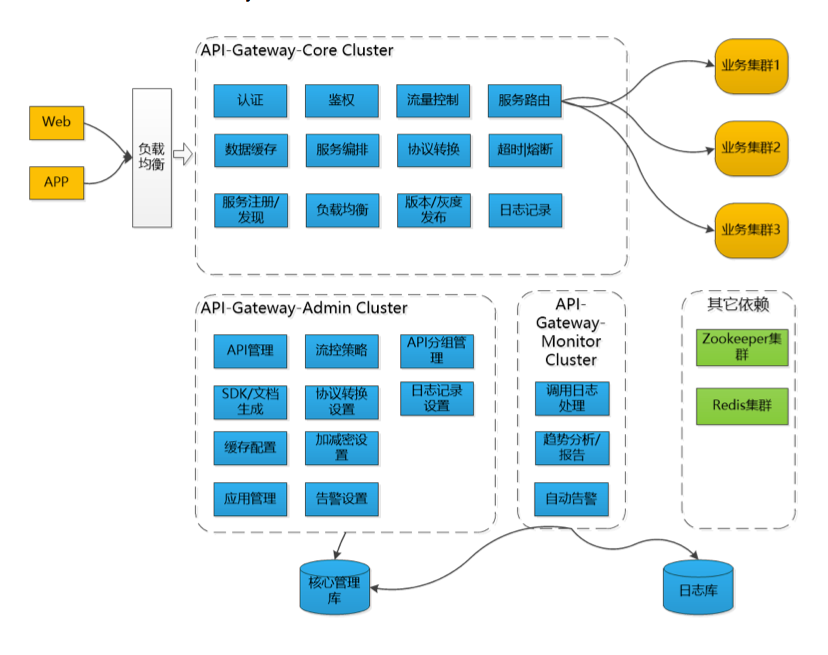


图1-3-2：小豹API网关的架构图

小豹API网关的开发者认为他们的网关相比其他API网关更加适合作为企业的OpenAPI网关、微服务网关、API管理服务平台，对于这三种API网关其帮助文档也给予了说明：

1. Open API。企业将自己的数据、服务等使用开发平台向外开放，一般用rest的方式向外提供，像腾讯公司的QQ开发平台、淘宝开放平台、微信开放平台等。 Open API开放平台一定会涉及到应用的接入、API权限管理、调用次数管理等，需要有一个统一的入口进行管理，这正是API网关可以发挥作用的时候。
2. 微服务网关。在微服务架构中，有一个组件是必不可少的，那就是微服务网关，微服务网关处理了负载均衡，缓存，路由，访问控制，服务代理，监控，日志等。API网关在微服务架构中正是以微服务网关的身份存在。
3. API服务管理平台。上述的微服务架构对企业来说有可能实施上是困难的，企业有很多遗留系统，要全部抽取为微服务器改动太大，对企业来说成本太高。但是由于不同系统间存在大量的API服务互相调用，因此需要对系统间服务调用进行管理，清晰地看到各系统调用关系，对系统间调用进行监控等。 API网关可以解决这些问题，我们可以认为如果没有大规模的实施微服务架构，那么对企业来说微服务网关就是企业的API服务管理平台。

通过对比两家API网关的使用流程就可以发现，阿里云和小豹的API网关有很多相似点，比如基础的API登记接入，API分组创建和应用创建，最后都是使用AppID附加一些约定的授权信息去访问API网关获得API响应数据。但是他们都没有对返回数据做处理，目前都是直接透传回用户，在实际场景中，最好是可以让用户定制返回内容。另外，这两个API网关都没有提供智能路由方案，当一种服务请求后台对应着多个服务提供商时，最好能自动让API网关选一个“最好”的转接给用户，这便是我要研究和实现的。

## 该课题研究的主要内容

本课题将研究第三方服务的特点，设计一种云环境下的外部服务的统一接入方案。采用B/S架构，研究主要内容包括: （1）外部服务登记管理，（2）外部服务接入接出适配，（3）外部服务调用以及智能路由处理，（4）外部服务的监控，（5）服务记账，（6）用户管理及权限管理。本课题中的重点是智能路由算法和外部服务返回报文的格式转换算法。

## 本文结构

本文主要内容如下：

第一章，绪论。主要阐述课题的背景、目的和意义以及国内外研究现状。

第二章，相关技术介绍。主要介绍系统开发用到的关键技术，对于一些核心概念要简要展开讲解。

第三章，需求分析。从API调用者，API提供商，网关平台管理员三种角色出发说明具体的用户需求、功能需求，并且从需求出发介绍需要的开发环境与工具要求。

第四章，系统设计。详细阐述系统从整体架构、底层子模块到系统数据库的设计流程，为系统的实现奠定了基础。

第五章，各模块实现。简要阐述各模块实现的过程，说明模块之间的调用逻辑。

第六章，平台测试。寻找Bug，找出不足。

1. 相关技术介绍

## Spring相关

### Spring

Spring是一个开源框架，它是为了解决企业应用开发的复杂性而创建的。框架的主要优势之一就是其分层架构，分层架构允许使用者选择使用哪一个组件，同时为J2EE应用程序开发提供集成的框架。Spring使用基本的JavaBean来完成以前只可能由EJB完成的事情。Spring的核心是控制反转IOC和面向切面AOP[8]。

### SpringMVC

SpringMVC框架通过实现Model-View-Controller模式来很好地将数据、业务与展现进行分离。从这样一个角度来说，Spring MVC和Struts、Struts2非常类似。Spring MVC的设计是围绕DispatcherServlet展开的，DispatcherServlet负责将请求派发到特定的handler。通过可配置的 handler mappings、view resolution、locale以及theme resolution来处理请求并且转到对应的视图[9]。Spring MVC请求处理的工作流程如图2-1所示。



图2-1 Spring MVC处理流程图

### SpringBoot

Spring与SpringMVC两个框架需要我们手动配置的地方非常多，但是在实际编码时我们发现系统，类库，框架应该有合理的默认值，而非要求提供不必要的配置，可是使用Spring或者SpringMVC的话依然需要我们进行非常多的配置工作，这样不仅增加工作量而且在跨平台部署时非常容易出问题。Spring Boot是为简化配置应运而生，使用Spring Boot可以快速创建一个基于Spring的项目，而让这个Spring项目跑起来我们只需要很少的配置就可以了。Spring Boot主要有如下核心功能：

1. 独立运行的Spring项目
2. 内嵌Servlet容器
3. 提供starter简化Maven配置
4. 自动配置Spring
5. 准生产的应用监控
6. 无代码生成和xml配置

### SpringCloud

SpringCloud是基于SpringBoot的分布式框架，它为开发人员提供了快速构建分布式系统的一些工具，包括配置管理、服务发现、断路器、路由、微代理、事件总线、全局锁、决策竞选、分布式会话等等。它运行环境简单，可以在开发人员的电脑上跑，适合于现在的云计算开发环境。其核心组件包括服务注册与发现中心（Eureka）、负载均衡相关组件（Ribbon与Feign）、断路器（Hytrix），微服务网关（Zuul）等。

## Mybatis

原生的java语言访问Mysql数据库需要引入JDBC驱动，但是如果直接通过JDBC技术进行数据库的增删改查操作，会产生大量重复代码，数据访问层要写大量的SQL语句而且非常不易于维护，所以企业开发中经常将JDBC封装成操作方便的持久层框架，我们经常见到的的便是Hibernate和Mybatis，Hibernate框架中使用面向对象的思维去操作数据库，这虽然很方便但是也限制了SQL语句的灵活性，不适合用于复杂查询。之所以选择Mybatis是因为它配置简单，对SQL语句的支持也非常灵活，可以通过简单的XML配置和注解将JAVA的POJOs映射成数据库中的记录。文档[10]中也提到了Mybatis支持动态SQL，更符合对于SQL掌握较好的用户，但是缺点也很明显，它需要用户写很多的SQL代码。

## Netflix-Zuul

之所以单独拿出来Zuul是因为它在本项目中的地位非常重要。Netflix-Zuul是Netflix开源的微服务网关，它可以和SpringCloud体系中的Eureka、Ribbon、Hystrix等组件配合使用。同时Zuul本身已经提供了一系列的过滤器，这些过滤器从类型上可以分为三种：

“pre”型过滤器，主要用于请求转发之前判断当前请求是否带有符合要求的身份认证信息，只有当请求中具有符合要求的验证信息时才进行路由，对于不符合认证要求的网关请求，在这一步就可以直接打回，避免对后端服务器施加压力。

“route”型过滤器，主要用于自定义路由，Zuul的底层默认使用的是Apache的HTTP Client，通过读取一张路由表来实现请求的转发，如果需要在请求到达时动态的修改即将路由的URL，可以在这种过滤器中重新构造HTTPClient进行动态转发。

“after”型过滤器，主要用于路由结束后对返回报文的处理，以及一些资源释放工作。

在SpringCloud体系中，Zuul主要是从配置文件里面读取路由映射关系，由于我要做的是面向外部服务的接入，所以需要修改Zuul的配置，让其从数据库中读取路由信息实现自动路由转发。

## FastJSON

JSON(JavaScript Object Notation, JS 对象简谱)是一种轻量级的数据交换格式，它具有非常简洁和清晰的层次结构，非常易于阅读和编写，也非常容易机器解析和生成。目前，大部分的API都使用JSON进行数据交互，在前后端分离的架构中，前端与后端便是通过JSON字符串来传递信息，前端与后端只需要按照一份标准的接口文档各自进行自己的开发即可。

在本课题中，JSON扮演着举足轻重的角色。目前JSON的常用解析工具有很多

### 2.4 本章总结

本章介绍了这次项目中用到的一些重点技术，本项目最终应该是可以和SpringCloud体系整合，自身成为一个服务组件，开发环境选择Intelli Idea，因为这款IDE相对于eclipse有很多人性化的插件可以使用，更适合云计算分布式开发，其可以自动构建Springboot项目，调试起来也很方便。

# 第3章 需求分析

## 3.1功能需求

本系统中的用户角色主要包括：API服务提供商，API服务消费者，API网关平台管理员，API网关平台本身，系统总用例图如图所示

图3-1：系统用例总图

## 3.1.1用例描述

## 3.1.1.1 API服务提供商用例

1. API提供商注册登录：原则上API提供商与API消费者在系统注册时都应该登记自己的实名制信息并且经平台管理员审核才能进行其他工作。
2. API登记接入：API提供商登录之后进入API创建页面，正确填写必要信息之后点击创建，将API信息录入平台数据库中API转发表

主要信息包括：api名称，api最大接入量，api请求方式，api真实url地址，api参数列表（参数名称，参数位置，参数是否必选，参数样例,参数是否为常量参数），api功能描述，api记账方式，api单价，api返回格式，api正常返回结果，api异常返回结果，api心跳测试接口

1. API在线配置：API提供商成功创建API后可以在个人中心的API列表中看到自己创建的所有API，选择配置后进入配置页面，可以选择对API的信息进行修改，然后保存。
2. API在线测试：服务提供商完成创建API之后还不能马上发布API到线上，应经过测试，验证其可以正确通过网关进行访问才能将API发布到线上。测试界面应该将API登记时登记的参数直接展示出来，帮助API提供商进行测试，API提供商按照自己设定的参数格式输入参数，查看响应结果。API提供商可以在API列表项选择发布API到线上。
3. 账单管理：API提供商可以在账单管理界面清楚看到自己创建的API被调用的情况，可以根据APPID查看某个APP调用API的详细情况，支持导出报表功能。
4. 监控状态查看：通过API网关的监控中心，服务提供商可以看到自己的API最近的访问情况，平均响应时间和健康状态等信息。

## 3.1.1.2 API服务消费者用例

1. 消费者注册/登录：与API服务提供商一致
2. 创建App:消费者登录成功进入个人中心后可以在App管理页面创建App，当然这个App是虚拟的，创建成功后系统会给该App分配一个AppID与AppSecret。
3. 检索API：消费者需要前往API中心寻找符合自身需要的API，用户可以输入检索条件来获得API列表。
4. 购买API使用权：找到后在API详情页选择购买数量后就可以获得API的使用权，API的记账方式有多重，可能是按次数收费，也可能是按照流量收费。
5. API授权给应用：需要强调的是，API授权是指授权给应用而不是授权给用户，所以消费者在得到API一定数量的使用权后要将自己买到的API授权给自己的应用后让应用去带着授权信息去访问API。
6. API监控状态查看：消费者可以在监控中心看到自己买到的API的健康状况以及自己的历史请求数据。
7. 账单管理：API消费者可以在账单管理页面清楚地看到自己的API使用情况，包括详细的账单数据。
8. 定制返回信息：由于消费者需求总是多样的，可能原API返回的报文尽管内容上是其需要的，但是格式上或者说数量上和消费者需要的还是有出入的，所以这里让消费者在平台定制自己需要的返回内容，平台根据这里指定的规则对返回报文进行格式转换。

## 3.1.1.3网关平台管理员用例

1. 登录：平台管理员不用注册直接登录。
2. APi审核：服务提供商创建的API必须经过平台管理员审核通过才能上线。
3. 用户管理：API消费者与API提供商的账户管理。
4. API监控状态查看：同API提供商用例一致。

## 3.1.1.4 API网关本身用例

1. API路由转发：App向网关发送过来API请求时如果该API背后有多个候选的服务提供商，网关应该综合各种因素为用户挑选一个“最好”的API发送请求，如果只有一个API提供商则不用智能路由。
2. API请求鉴权：App请求到达网关，网关要检查其有没有访问对应API的权限，如果没有权限，则将请求直接打回返回错误码。若已经授权，便将其转发至对应API真实Url地址。
3. API服务记账：对于每一次App的请求，网关这里都要进行记录，当发现请求的App所属用户购买的API使用量已经用完时便直接返回错误码不再为其转发。
4. 协议格式转换：协议格式转换分两部分，一部分是请求报文格式转换，这部分主要是参数的转换，另一部分是响应报文的格式转换，这里应该根据API消费者之前指定好的映射规则来实现转换。

## 3.2 性能需求

系统性能需求如表3-2所示：

在（2c,4G）\*3的集群下：

|  |  |  |
| --- | --- | --- |
| 大类 | 指标 | 备注 |
| 综合场景负载测试 | 系统处理能力（QPS） | >500笔/秒 |
| 平均响应时间 | <=3秒 |
| 交易成功率 | 大于等于99% |
| 稳定性测试 | 稳定运行时间 | 24小时 |
| 指标 | 系统峰值的65% |
| 交易成功率 | 大于等于99.9% |
| 系统资源 | CPU使用率 | <=75% |
| 内存使用率 | <=75% |
| 磁盘繁忙率 | <=75% |

表3-2：系统性能需求

## 3.3安全需求

（1）系统级安全：平台要具有数据分离与备份恢复机制，接入消费者认证机制，接入企业认证机制，以及敏感信息的加密机制。

（2）应用级安全：包括用户管理，日志操作等

3.4 本章总结

## 本章从功能需求、性能需求、安全需求三个方面阐述了系统的需求分析工作。为下一步的系统设计做好了基础。

**结 论**

**参考文献**

1. 孙杰贤.从YaaS看传统软件企业转型[J].中国信息化,2016(08):60-61.
2. 吴坤安,黄文思,韩泽华,黄屏发,李芸,王荣烨,陈伟仑.基于Docker的数据库微服务系统设计与实现[J].国外电子测量技术,2017,36(12):57-62.
3. 唐瑞. 基于Kubernetes的容器云平台资源调度策略研究[D].电子科技大学,2017.
4. netflix.spring-cloud-netflix-zuul[EB/OL].https://github.com/spring-cloud/spring-cloud-netflix/tree/master/spring-cloud-netflix-zuul.
5. apiaxle. Documentation[EB/OL]. http://apiaxle.com/docs.html.
6. Alibaba[EB/OL].https://help.aliyun.com/document\_detail/29464.html?spm=a2c4g.11186623.6.539.EFhEOs.
7. 成都小豹科技有限公司. 小豹Api网关帮助文档[EB/OL]. http://www.xbgateway.com/why\_use\_api.html.
8. Praveen Gupta. Spring Web MVC Framework for rapid open source J2EE application development: a case study[J]. International Journal of Engineering Science and Technology Vol. 2010, 2（6）, 1684-1689.
9. Paul Deck. Spring MVC学习指南[M]. 北京：人民邮电出版社，2015：25-42.
10. 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.

**致 谢**

将近一个学年的毕业设计已接近尾声，毕业设计让我学到了很多有用的东西，不仅仅是知识上的进步，更重要的是我得到大家给我的无微不至的关怀和帮助。毕业设计的过程就是我们成长的过程，在这个过程中我们相互帮助、相互协作、团结一致，无论是学习还是工作都让我们变得成熟。

经过了这几个月的设计经历，我获得了各位老师和同学的帮助，才能使我的毕业设计最终圆满完成。在这里衷心感谢我的毕业设计导师初佃辉老师，从前期开题阶段，到系统开发阶段，再到最后论文定稿阶段，他的言传身教将使我受益终生。在此特别的感谢张小东老师给予我关键性的指导意见，感谢老师的严格教导和谆谆教诲，是他治学严谨、认真负责的态度为我们做出了榜样。感谢学院给我们提供良好的设计条件和良好的物质保证，感谢院领导对我们的关心和支持。

这次能圆满完成任务，我感到骄傲和欣慰，在这里我由衷的感谢帮助过我的人，由衷地对他们说声谢谢！