**数据平台API访问控制技术的研究与实现**

**摘　要**

目前，随着互联网的不断发展，API已经广泛运用于网站、移动应用和企业管理等领域中，在日常生活和工业生产中发挥着重要的作用。与此同时，伴随着当前“互联网+”以及各类云服务的火热兴起，各行各业对于API管理的需求都在不断增加，如何更有效的对API进行管理，对企业、行业和社会而言也越发重要，API管理平台在当前的这种市场需求中应运而生。更进一步，为保证API服务的安全，需要结合API管理中遇到的实际情形，在API管理和访问控制技术间找到合适的契合点，对API访问控制技术和策略进行深入研究。

本文首先介绍了目前API管理的研究和使用情况、访问控制技术的背景知识和当前的API访问控制技术情况。接下来详细介绍了在开源API网关的基础上设计的API管理平台及其实现细节。在对API管理平台实现的基础上，结合目前常见的访问控制模型，本文设计了三种API访问控制决策算法。在文章的最后，详细介绍了利用API网关提供的相关插件实现API访问控制技术的方法。

本文设计和实现的API管理平台对传统的API管理功能进行了拓展，结合实际需要和情形设计实现的API访问控制策略极具使用价值和启发意义，为API访问控制技术的进一步研究和API细粒度管理提供了思路和方向。

**关键词**　API访问控制　API管理系统　API管理　访问控制

**Research and implementation of API access control technology for data platform**

**ABSTRACT**

Nowadays, with the continuous development of the Internet, API has been widely used in the field of websites, mobile applications and enterprise management, and plays an important role in the daily life and industrial production. At the same time, along with the rise of the "Internet +" and all kinds of cloud services, demand from all walks of life for API management is growing. Thus, how to more effectively carry out the management of API, which is becoming more and more important to the enterprise, industry and society, API management platform came into being in this situation. Further, in order to ensure the safety of API service, we need to find a suitable meeting point between the API management and access control technology, and make an intensive study of API control technology and policy combined with the actual situation.

In the paper, we firstly introduce the research and usage of API management, technical background of access control and the current circumstances of API access control technology. Next, the design and implementation of API management system, which based on an open API gateway were described in detail. On the basis of API management platform, combined with the common access control model, three API access control decision algorithms were designed. In the end, we detail introduced the implementation process of API access control techniques by using the plugins of API gateway.

Design and implementation of API management platform that introduced in this paper extends the functions of traditional API management, and API access control policy that combined with the actual needs and circumstances are valuable and meaningful. All of this provide ideas and directions for further study of API access control technology and fine-grained management API.

**KEY WORDS API access control API management system API management access control**

**目　　录**

[第一章　绪论 1](#_Toc453012471)

[1.1研究背景和研究意义 1](#_Toc453012472)

[1.2国内外研究和应用现状 1](#_Toc453012473)

[1.3论文研究内容 2](#_Toc453012474)

[1.4论文内容安排 2](#_Toc453012475)

[第二章　API管理和API访问控制技术研究 3](#_Toc453012476)

[2.1 API管理研究 3](#_Toc453012477)

[2.1.1 API与API管理现状 3](#_Toc453012478)

[2.1.2 API管理平台介绍 3](#_Toc453012479)

[2.1.3 本平台API管理功能性需求分析 5](#_Toc453012480)

[2.2访问控制技术研究 8](#_Toc453012481)

[2.2.1 访问控制技术 8](#_Toc453012482)

[2.2.2常见的访问控制模型 9](#_Toc453012483)

[2.3 API访问控制技术介绍 11](#_Toc453012484)

[第三章 API管理平台的设计与实现 13](#_Toc453012485)

[3.1 API管理平台的架构 13](#_Toc453012486)

[3.2 开源网关Kong 14](#_Toc453012487)

[3.2.1 Kong的工作流 14](#_Toc453012488)

[3.2.2 Kong的插件 14](#_Toc453012489)

[3.2.3 Kong的管理 15](#_Toc453012490)

[3.3 基于Django框架的API管理系统的实现 15](#_Toc453012491)

[3.3.1 Python-Kong-Client 16](#_Toc453012492)

[3.3.2 API管理系统管理员端的实现 18](#_Toc453012493)

[3.3.3 API管理系统用户端的实现 21](#_Toc453012494)

[第四章　API访问控制技术的设计与实现 24](#_Toc453012495)

[4.1 API访问控制决策算法的设计 24](#_Toc453012496)

[4.1.1 ABAC与API管理的结合 24](#_Toc453012497)

[4.1.2.DAC和MAC与API管理的结合 25](#_Toc453012498)

[4.1.3 RBAC与API管理的结合 25](#_Toc453012499)

[4.2基于ABAC的API访问控制的实现 26](#_Toc453012500)

[4.2.1 API网关访问控制决策流程 26](#_Toc453012501)

[4.2.2 Kong插件配置的实现 27](#_Toc453012502)

[第五章 系统测试 29](#_Toc453012503)

[5.1 功能测试 29](#_Toc453012504)

[5.1.1 用户注册功能测试 29](#_Toc453012505)

[5.1.2 用户发布API功能测试 29](#_Toc453012506)

[5.1.3 用户修改API功能测试 29](#_Toc453012507)

[5.1.4 用户删除API功能测试 29](#_Toc453012508)

[5.1.5 用户购买/使用API功能测试 30](#_Toc453012509)

[5.1.6 管理员管理API功能测试 30](#_Toc453012510)

[5.1.7 管理员管理用户功能测试 30](#_Toc453012511)

[5.2 性能测试 30](#_Toc453012512)

[第六章 总结与展望 34](#_Toc453012513)

[6.1 总结 34](#_Toc453012514)

[6.2 展望 34](#_Toc453012515)

[参考文献 36](#_Toc453012516)

[致　　谢 37](#_Toc453012517)

第一章　绪论

## **1.1研究背景和研究意义**

随着大数据技术的发展，国内外数据平台的研究与应用已经成为当前的热点。许多专家和学者认为：“大数据是一种价值观、方法论，我们面临的不是随机样本，而是全体数据；不是精确性，而是混杂性；不是因果关系，而是相关关系。这是一场思维的大变革，更是一个互动的过程——你可以用不同的角度不同的方式去做大数据，并得到不一样的结果与好处”。数据平台作为随大数据潮流产生的一种工具，对将数据价值最大化和帮助用户做出基于理性的判断和决策具有重要意义。

作为数据开放的接口，在数据平台上，广大用户不仅可以使用数据平台的API，还可以经管理员审核后发布自己的API。通过实现API访问控制技术，一方面可以提高API服务的安全性，防止数据泄密的发生；另一方面，实现API访问控制技术后，可以实现更细粒度的管理，方便管理员和用户的使用。因此，API的管理和访问控制技术需要进行深入研究。

本论文通过研究国内数据平台上API的发布、审核、以及访问控制等技术和流程，寻找主流的实现思想，并在实验室已搭建好的数据平台上，实现API的管理，包括发布、审核以及访问控制策略，为在数据平台上对API进行管理和访问控制提供了一个行之有效的方案。

## **1.2国内外研究和应用现状**

现如今，在互联网这块大网络中，各类应用通过不同的API，互相依赖着彼此的数据及服务。在这个大环境下，API管理对于企业及开发者们来讲，变得越来越重要。因此，各家云服务提供商以及SOA提供商对API管理以及API访问控制的研究和应用也最为前沿和广泛。从2012年开始，API管理领域逐渐开始升温，几家老牌的API管理平台提供商如Apigee，3scale等先后通过融资和收购来增加其API管理的新功能。一些云服务提供商也通过收购试图增强其云服务中API管理的功能，如微软收购API 管理服务公司Apiphany，将其API管理服务集成到自己的云服务Azure中；英特尔以1.8亿美元收购了API管理服务商Mashery；CA收购Layer7；Facebook收购了Parse。在国内，百度、腾讯、华为、360等公司也追随云时代的脚步，先后推出了带有API管理的云服务和API商店。

目前，虽然各家API管理平台或云平台均为用户提供了一定的API访问控制选项，但各种API访问控制策略和设定的作用和目的不尽相同，功能也或强或弱，因此API访问控制技术还有进一步研究和探索的空间和价值。

## **1.3论文研究内容**

本文的主要研究内容包括：

1. 分析研究目前国内外主流API管理、访问控制技术和API访问控制的方法与实现，理解其原理和实现方法。
2. 根据数据平台对API管理的实际需求设计出符合功能要求的API管理平台。
3. 基于开源网关实现一个完整的API管理平台。
4. 研究目前主流的访问控制策略和机制，寻找适合与API管理结合的结合点，选择合适的访问控制模型，构建合适恰当的API访问控制决策算法。
5. API访问控制技术在API管理平台上的实现。

## **1.4论文内容安排**

本文共分为6章，各章内容安排如下：

第一章　绪论，本章首先介绍本文的研究背景和研究意义，然后介绍了论文的研究内容以及论文各章节的内容安排。

第二章　API管理和API访问控制技术研究，本章首先介绍了目前国内外使用API管理的现状和主流方法，接着介绍了主流控制访问技术，最后对API访问控制技术的背景和研究现状进行了介绍。

第三章　API管理平台的设计与实现，本章详细的介绍了设计的API管理平台的基本架构、功能模块设计以及具体的程序实现。

第四章　API访问控制技术的设计与实现，本章根据第二章中介绍的常见的访问控制决策算法，结合实践中可能遇到的需要情形，设计了三种访问控制决策算法。本章的后半部分介绍了其中的一种访问控制决策算法的在API管理平台中的实现。

第五章　系统测试，本章模拟实际情况对API管理系统进行了功能测试和性能测试，对API网关进行了性能测试，并根据测试结果对需求进行了验证。

第六章　总结与展望，对文章内容进行总结，展望了API管理和API访问控制技术的研究的前景和研究方向。

第二章　API管理和API访问控制技术研究

本章首先对目前国内外使用API管理的现状和主流方法进行了介绍和说明；接着介绍了目前学术界和工业界主流的访问控制技术。在本章的最后，简要的介绍了API访问控制技术的现状。

## **2.1 API管理研究**

伴随着当前“互联网+”以及各类云服务的火热兴起，各行各业对于API管理的需求都在不断增加。API作为连接各个应用程序和数据之间的桥梁和纽带，API管理平台应运而生，如何更有效的对API进行管理，对企业、行业和社会而言也越发重要。

**2.1.1 API与API管理现状**

API（Application Programming Interface），即应用程序接口。目的是提供[应用程序](http://baike.baidu.com/view/330120.htm)与开发人员基于某[软件](http://baike.baidu.com/view/37.htm)或硬件得以访问一组[例程](http://baike.baidu.com/view/1582587.htm)的能力或理解内部工作机制的细节，而又无需访问源码。在Web 2.0的浪潮到来之前，开放的API主要体现在桌面应用上，但现在随着大数据时代、应用经济时代、智能设备时代和物联网时代的到来，越来越多的Web应用面向开发者开放了API，也有越来越多的企业通过API对数据进行流转。本文关注的重点也正是这些基于Web的API及管理这些API的方法。

在大数据时代，API的本质是数据的流入和流出。数据以API的形式在应用和企业之间交互流转，通过 API 输出企业核心的业务处理能力，给应用及终端用户提供更为优质、便利的最小服务单元，因此也是企业发展的核心内容。国外的互联网巨头Google、Facebook、Twitter，抑或是云服务提供商Amazon，都有先进的API管理及开放平台，所有的服务功能都通过API来获取及提供；在国内的互联网企业中，腾讯、百度、阿里、新浪等也都提供了基于API的开放服务平台，通过 API提供各种应用类型、各种应用场景下的服务和处理能力。

由于在移动端涌现了各种花样繁多的应用，很多公司也急着依靠这些平台来扩大自己的影响力传播渠道，发布自己的 API。在过去的几年里年，API增长迅猛，根据 API产品目录网站ProgrammableWeb的数据，从2010年到2013年短短的3年间，在其网站上登记的API数目从2000余个升至10000余个，增长超过5倍。在这种新形势下，也催生了一门新的业务：API管理。API管理的目的就是通过扩大各大公司的API通道，增加开发者的使用率，实现流量控制，增加 API 的安全性等等手段，让某个公司的API变得安全性好，可扩展性高，并易于使用，获得更多的应用开发者青睐。

**2.1.2 API管理平台介绍**

作为企业应用的重要组成部分，API管理平台受到各家云服务提供商、企业服务提供商和SOA提供商的关注和重视。对于一般的API管理平台而言，整个API管理平台可以分为基于UI界面的前端API管理系统(API门户)和处于后台的API网关。下面将分别对目前市场上主流的API的管理系统和API网关原理和实现详细介绍。

1、API网关

从理论上来讲，API的使用者可以直接向每个API端点发送请求。每个API服务都会有一个公开的端点，API端点负责解析用户的请求并做出响应。但遗憾的是，直接使用这种方法存在着很大挑战和局限：

首先是用户的需求和每个API暴露的细粒度API不匹配。举例来说，为了实现某项功能，一个使用API服务的客户端可能需要发送7个独立请求。在更复杂的应用程序中，可能要发送更多的请求。按照Amazon的说法，他们在显示其产品页面时就调用了数百个API服务。发送如此多的请求无疑会大大的降低服务质量，增加网络延迟。

使用者直接调用API的另一个问题是，不同的API服务的提供者不同，可能会使用不同的Web协议，一个API服务可能使用Thrift二进制RPC，而另一个服务可能使用AMQP消息传递协议，这些协议往往不是浏览器友好或防火墙友好的，只适合在内部使用。在防火墙之外，应用程序应该使用诸如HTTP和Web Socket之类的协议，协议转换的需要将会成为直接使用API服务的难点之一。

这种方法的另一个缺点是，对于API开发者而言，它会使得API服务难以重构。随着时间推移，API开发者可能想要更改系统划分服务的方式。例如，我们可能合并两个API服务，或者将一个API服务拆分成两个或更多服务。然而，如果API使用者与API直接通信，那么执行这类重构工作就非常困难了。

由于这些问题的存在，API使用者与API直接通信存在众多的风险和问题，我们需要将API使用者和API端点隔离，API网关应运而生。API网关是一个服务器，是整个系统唯一的入口。从面向对象设计的角度看，它与外观模式类似。API网关封装了系统内部架构，为每个客户端提供一个定制的API。API网关除了负责服务请求路由、组合及协议转换。它还具有其它职责，如身份验证、监控、负载均衡、缓存、“请求整形（request shaping）”与管理、静态响应处理[4]。

API网关有其优点也有其不足。使用API网关的最大优点是，它封装了应用程序的内部结构。API的使用者只需要同网关交互，而不必调用特定的服务。API网关可以为多个API服务提供同一个接口，这可以大大减少API使用者调用API的次数，降低了API使用者的使用难度。

API网关也有其不足。首先它的出现为系统增加了一个必须开发、部署和维护的高可用组件。其次，API网关需要应对大量的请求，有很大的概率会成为性能的瓶颈。

一些云平台服务商如亚马逊AWS、微软Azure等提供的API管理功能中均能实现API网关的功能，但并不开源，其更多的目的是吸引用户将API部署在各自的云平台之上。SOA提供商提供的API管理套件中一般也提供有API网关，比较著名的有WSO2 Gateway，除此之外，还有一些出色的独立开源API网关如Kong、TYK、API umbrella等。

2、API管理系统

仅仅使用API网关实现对API的管理通常是不够的，在实践的过程中，只使用API网关完成对API的管理经常会遇到以下问题：

首先，API网关往往以CLI或者符合RESTful风格的HTTP请求对API网关进行管理，对于不熟悉和了解API网关的管理员，很难对API网关进行直接的管理操作。

其次，对于API的使用者来说，通过API网关难以获得API的使用方法和使用文档，也难以直观的了解API的内容、作用和限制。

除此之外，由于API网关只关心API的端点信息、参数和用户的授权信息等少量信息，而对API的具体信息，用户的具体身份、角色不关心也不了解，很难再进一步实现更多细粒度的管理及功能拓展延伸。

因此，在API网关之上，往往需要存在一个API管理系统作为API管理、介绍和展示的门户。API管理系统通常类似于CMS(Content Management System，内容管理系统)，API使用者可以在API管理系统提供的门户获得API的详情、使用文档和使用权限，管理员可以在管理系统后台对API网关和API管理系统门户的内容和数据进行管理。

目前开源的API管理系统较多，但各个API管理系统功能不一，规模有大有小，代码质量也参差不齐。市场上较为完善和流行的有apiaxle、WSO2 API Manager等。

**2.1.3 本平台API管理功能性需求分析**

根据实验室数据平台的实际需求，对于需要的完整的API管理系统，系统不仅要对数据平台内的API进行管理，还需要对外部注册用户及用户发布的API进行管理。综合而言，整个API管理系统需要完成的基本功能有：用户注册、管理员审核用户上传的API后将该API发布、用户修改API、用户删除API等功能。

下面将选取几个有代表性的操作使用系统顺序图对操作的步骤详细说明：

1、用户注册

如图2-1是用户在API注册时的系统顺序图，接下来将详细说明图中的每个步骤：

图2-1 用户注册系统顺序图

步骤1：用户在API管理系统门户的用户注册页面填写相关信息。

步骤2：API管理系统在检验注册信息无误后向API网关申请一个新用户和用户的密钥。

步骤3：API网关在用户和密钥创建成功后向API管理系统返回成功信息。

步骤4：API管理系统将收到的注册成功信息和分配的密钥传递至前端网页显示。

2、用户申请发布API



图2-2 用户提交API完整系统顺序图

如图2-2是用户提交API时完整的系统顺序图，接下来将详细说明图中的每个步骤：

步骤1：用户在API管理系统门户的API注册页面填写欲在系统中注册的API的相关信息。

步骤2：验证提交的信息无误后，API管理系统将提交的API信息保存，并返回提交成功等待审核的信息。

步骤3：管理员在管理员登陆后查看提交未审核的API信息，经检查、测试和确认后向API 网关同步提交的API。

步骤4：API管理系统向API 网关同步提交的API。

步骤5：API网关向API管理系统返回的相关成功信息。

步骤6：API管理系统向管理员提示相关API已经被成功启用和同步。

步骤7：用户可向API管理系统申请查看自己已经提交API的状态。

步骤8：API管理系统向用户显示被审核通过的用户提交API。

3、用户购买/使用API



图2-3 用户购买API系统顺序图

如图2-3是用户购买API时的系统顺序图，接下来将详细说明图中的每个步骤：

步骤1：用户在前端页面向API管理系统申请购买某API。

步骤2：API管理系统在收到用户的API购买申请后在API中

步骤3：API网关返回配置成功的信息。

步骤4：API管理系统向用户返回购买/使用成功的信息。

根据对数据平台API管理的功能需求分析，以及对市场上主流API管理系统的调查和研究，目前市场上所有API管理系统中，用户注册后只能作为系统中API的使用者，或者用户管理API的权限需要由管理员手工下放，并不能上传自己API后待管理员审核后，对API自主的进行管理，因此需要自主设计一个适合功能需求的API管理平台。

## **2.2访问控制技术研究**

**2.2.1 访问控制技术**

访问控制是信息安全研究的一个重点，不论是传统的IT领域还是新兴的互联网领域，访问控制在其中都扮演着重要的角色，对于保证服务器软硬件、资源和用户数据的安全性起着举足轻重的作用。

访问控制是指，针对服务器端客体资源获取能力和范围进行控制的机制，当资源受到未经授权的访问时，可以依据设定好的安全策略和算法对访问进行认证和限制，以提供合理的手段对资源进行保护。可以说，访问控制存在的主要目的和意义就是保证资源的安全性，对访问进行区分和隔离，以保证正常用户的访问，而阻止非正常用户的访问。访问控制对于一个完整系统而言，是不可或缺的，对于系统安全至关重要。

访问控制包括三个要素，即主体、客体和访问控制策略，其中第三个要素，即访问控制策略是访问控制的关键技内容。如图2-4是ISO访问控制通用框架：



图2-4 ISO访问控制通用框架

主体(Subject)是指一个提出请求或要求的实体，是动作的发起者，但不一定是动作的执行者，可以是用户，也可以是任何主动发出访问请求的智能体，包括程序、进程服务等。

客体(Object)是需要接受其他主体访问的被动实体，包括所有受访问控制机制所 保护下的系统资源，在不同应用场景下可以有着不同的具体定义。比如在操作系统中 可以是一段内存空间，磁盘上面的某个文件，在数据库里可以是一个表中的某些记录，在Web上可以是一个特定的页面，网络结构中的某个广义上的数据包结构。

访问控制策略(Access Control Policy)是主体对客体的操作行为的约束条件集。简单的讲，访问控制策略是主体对客体的访问规则集，它直接定义了主体对客体可以实施的具体的作用行为和客体对主体的访问行为所做的条件约束。访问控制策略在某种程度上体现了一种授权行为，也就是客体对主体访问时，所具有的操作权限的允许。主体进行访问动作的方式取决于客体的类型，一般是对客体的一种操作，比如请求内存空间，文件的操作问题，修改数据库表中记录，以及浏览陌生服务器中的某些页面等。

在API访问控制问题中，客体是目标API，主体是意欲使用目标API的用户，对于用户发送的请求，我们应该合理的设置访问决策，根据之前设定的访问控制策略和规则库对用户的请求进行判定，从而达到对API进行访问控制的作用。

**2.2.2常见的访问控制模型**

国内外研究者己经在数据与服务的访问控制领域进行了大量研究，也产生了很多的访问控制相关理论和模型，如Discretionary Access Control(DAC)自主访问控制、Mandatory Access Control(MAC)强制访问控制、Role Based Access Control(RBAC)基于角色的访问控制、Attribute Based Access Control(ABAC)基于属性的访问控制等，并在应用、管理平台等方面得到了大量的应用，在这些系统中发挥着重要的作用，为系统安全做出了重要的贡献[15]。

1、自主访问控制

Discretionary Access Control（DAC，自主访问控制），其基本思想是系统中的主体（用户或者资源的所有者）可以根据自己的意愿，将其拥有的对客体（资源或者服务）的权限授予给其它主体。自主访问控制是一种有效的访问控制机制，能有效保护客体不被非法或者越权访问。

虽然自主访问控制是一种有效的访问控制机制，也已经在不少的系统中实现了，但是该机制也有自己的弱点。首先，自主访问控制的客体访问权是掌握在客体的所有者手中的，这样使得访问权非常的分散，不利于集中管理。其次，访问权所有者可能不是专业人员，对访问控制不了解，因而不能有效的对访问权进行管理。最后，也是最重要的一点，自主访问控制中，访问权可以传递，即将自己的访问权授予给以后，同样可以将从那里获得的访问权授予给其他人。若访问权被传递出去，管理就会非常困难，并且会带来严重的安全问题。

2、强制访问控制

Mandatory Access Control（MAC，强制访问控制），其基本思想是系统中的主体和客体都会被管理员事先分配一个固定的安全属性，而该属性是不能被除了管理员以外的人修改的，根据安全属性判断主体是否有权访问客体。文献[12]中提到，强制访问控制最先使用于军事应用中，并且常与自主访问控制相结合使用。

强制访问控制相比于自主访问控制，添加了对客体属性的划分，并通过访问之前对访问权限和安全属性进行匹配，以判断是否允许访问。某种程度上讲，强制访问控制拥有更高的安全性，可以防止非法访问。但是这种访问控制过于严格，过于复杂。由于每个主体对每个客体的安全属性都会有一个匹配值，会产生N\*N的一个对应关系，造成管理上的不便，影响系统的灵活性。

3、基于角色的访问控制

Role Based Access Control（RBAC，基于角色的访问控制），其基本思想是，建立一个角色集合，并赋予角色相应权限，在用户与资源之间加入了角色层，通过为用户分配角色，使得用户拥有角色相关的权限。该机制使得新建用户时，只需分配相应角色，而不需要赋予多种访问权限，减少了不必要的操作。并且，仅修改角色就可以有效降低管理成本和开销，有利于简化授权管理[17]。

基于角色的访问控制是一种普遍适用、更为有效的访问控制方式，也是访问控制领域的研究热点，其基于访问者身份进行授权，既符合常理，又方便快捷。文献[13]在RBAC的基础上，加入了属性，使得能够较好的进行更细致的访问控制，但将其仍有一些不足之处，不能支持开放环境的访问控制。例如：若非法访问者，获取了合法的身份信息，便可获得对应的权限，造成系统信息泄漏；若非法访问者获取了某个角色，就可以随意使用相关权限，使得权限被滥用，对系统造成损害，这种方式也违背了最小权限原则。总之，RBAC不太适合用于开放的环境，也难以做到细粒度的访问控制。

4、基于属性的访问控制

Attribute Based Access Control（ABAC，基于属性的访问控制），其主体思想是利用主体、客体以及所处环境等相关的属性来进行访问控制和授权，通过属性来对服务访问进行限制。正是由于不是直接利用主客体之间的关系，使其能适应开放的网络环境。与传统的访问控制不同，基于属性的访问控制是基于请求者和资源的属性来进行授权，因而其具备令人印象深刻的灵活性、可扩展性的特点。能够有效支持匿名访问。文献[14]提出了提升ABAC安全和性能的方法，使得的效率和安全性有了进一步的提升。目前，对于使用ABAC对API进行访问控制已经有较多的文献和实现[11]，第四章中将对API管理中使用ABAC进行访问控制具体介绍。当然，ABAC也有其缺点，因其没有细粒度的划分数据，难以对数据进行动态、细粒度的访问控制。

5、访问控制模型总结

表2-1 常见访问控制模型的优缺点对比

|  |  |  |
| --- | --- | --- |
| 访问控制策略 | 优点 | 缺点 |
| 自主访问控制 | 1、简单直观 | 1、管理分散  2、授权困难  3、效率底下 |
| 强制访问控制 | 1、管理策略简单  2、安全性高 | 1、缺乏灵活度  2、应用领域窄 |
| 基于角色的访问控制 | 1、安全性高  2、管理方便  3、系统开销小 | 1、不适合开放式环境  2、控制粒度粗  3、缺乏动态性 |
| 基于属性的访问控制 | 1、适合开放式环境  2、灵活性  3、可扩展性 | 1、动态性  2、细粒度 |

表2-1针对上述的访问控制模型的优缺点进行总结，以期选择有价值的、合适的和易于实现的API访问模型，为API的访问控制技术的设计和实现打下基础。

## **2.3 API访问控制技术介绍**

目前，对于API访问控制技术，各方尚处于摸索阶段，行业内各厂商拥有和实现的方式也不尽相同，总结而言，主要的API访问控制技术有以下：

1. 身份认证

对于每个使用API的用户，验证其是否具有使用API的权限具有十分重要的意义。目前主要使用的身份认证的方法有：

* 基础HTTP认证

其方法就是使在请求API时将用户注册时的用户名和密码放入URL的参数或者Header中作为其身份验证，其优点是验证简单方便。但是其也有非常明显的缺点，因为HTTP协议采用明文传输，所有的信息都会暴露在外面，所以直接将用户名和密码在网络中传输十分的危险。为了保证安全，最好能够使用HTTPS协议来代替HTTP协议。目前只使用基础验证作为访问API时身份认证的已经非常少了。

* Token/Key认证

其方法就是为每个用户名单独生成一个Token/Key，每次访问API时需要将Token/Key带入URL参数中或者Header中，相比基础HTTP认证，因为Token/Key为独立生成，因此可以大大提高安全性。这种方法仍然存在一定的问题，由于HTTP协议明文传输，所以一旦别人获取到这个Token/Key，就可以伪装身份使用该API。目前这种认证方式使用较多，但是对于安全要求更高的API，应该考虑使用更为安全认证方式如oAuth认证或者多种认证方式结合使用[2]。

* oAuth认证

oAuth协议为用户资源的授权提供了一个安全的、开放而又简易的标准。与以往的授权方式不同之处是oAuth的授权不会使第三方触及到用户的帐号信息（如用户名与密码）[3]。这种方式最为安全，但也会增加延迟时间和网络资源消耗。

1. 用户权限管理

用户权限管理是企业使用API管理时非常重视和关注的地方，强大的用户权限设置功能可以满足企业用户管理API时遇到的各种问题，下面将介绍几种云服务商提供的用户权限管理的方式：

* RAM (Resource Access Management)资源访问管理

RAM是阿里云使用的用户权限管理方式。通过RAM，管理员可以集中管理使用的用户（比如员工、系统或应用程序），以及控制用户可以访问您名下哪些资源的权限。从权限角度看，管理员与RAM用户是一种root与user的关系，管理员对资源拥有一切操作控制权限，而RAM中的用户只能拥有管理员授予管理的权限。

* IAM（Identity and Access Management）。

作为云服务商的标杆，AWS使用IAM使用作为其用户权限管理的方式。IAM具有非常大的灵活性，不仅可以为用户定制群组和角色，还可以手动为每个群组、角色和用户设定访问控制策略。

* ACL(Access Control List)访问控制列表

与路由器中的ACL功能相似，在一些平台中，管理员用户可以通过配置ACL中的白名单和黑名单从而达到对用户限制使用和访问的功能。目前如APIcloud等使用了该方法。

1. 其他功能

除了身份验证和用户权限管理两项基本的访问API控制功能，还有一些平台为用户提供了更多的API访问控制功能，如IP访问限制、速率限制等。

第三章 API管理平台的设计与实现

API管理平台作为实现API管理和API访问控制技术的基础，依据对API管理的研究，本章首先介绍了设计的API管理平台的架构；接着依据从下到上的原则，先对选取的开源网关Kong的结构和原理进行了介绍和说明。在本章的最后，详细的介绍了API管理系统实现的具体过程和细节。

## **3.1 API管理平台的架构**

基于对实验室数据平台对于API管理功能的分析，现有的开源API管理系统很难完全适应实验室的数据平台对API管理系统功能的要求，因此依据实验室数据平台的实际情况和具体需求，结合现有主流API管理系统的设计思路和结构，设计出了适合实验室数据平台的API管理平台。



图3-1 API管理平台框架图

整个API管理平台的框架图如图3-1所示，在API管理系统的底层，使用了开源API网关Kong。Kong依赖于非关系型数据库Apache Cassandra对API网关中的相关数据进行存储。在 Kong中，还可以在核心部件之上部署一些的插件，方便对API网关的功能进行扩展。在下一章，我们将利用这些插件，实现API的访问控制的一部分功能。

在API网关的上层，API管理系统由python语言编写并基于Django web框架，API的相关信息和用户信息使用Django自带的ORM框架进行映射和管理，存储在MySQL数据库中[7]。API管理系统的管理员端使用Django框架自带的Django admin工具完成，通过Django admin可以快速的完成管理员端的开发，实现对API的数据和用户数据进行管理。为了减少API网关中的存储开销，加快API网关的效率，一般来说API网关中并不对API的细节和用户的具体内容进行存储，因此一部分API访问控制的功能需要模仿页面的访问控制，在API管理系统中实现。

在API网关和API管理系统之间，我们使用了开源Python-kong-client作为Django框架和开源网关Kong交互的工具。

根据本小节中API平台图的模块划分，下面的两节将分别对选用的API网关和设计的API管理系统的实现细节进行详细介绍。

## **3.2 开源网关Kong**

API网关Kong由世界上规模最大的API市场Mashape将其API层开源而来，根据其官方的说法，Kong管理着Mashape上的超过15000个API，每月承受着10亿次以上的用户请求。

Kong使用Lua语言编写[10]，利用OpenResty平台搭建在代理服务器Nginx之上。得益于Nginx优秀的性能和均衡负载能力，根据其官方的测试，在两分钟内Kong能够以平均10ms的延迟时间相应117185次请求且只有一次发生超时。此外，Kong还支持集群系统，能够轻松方便的根据实际流量需求增加、减少节点。

Kong强大的性能、优秀的插件可扩展性和开放的开源社区环境是选择Kong作为API网关的重要原因。

**3.2.1 Kong的工作流**



图3-2 API网关Kong工作流示意图

如图3-2是Kong在其官网上给出的Kong的工作流。当API网关工作时，用户使用API时，需先向API网关发出请求，API网关收到请求后会将请求转向目标API端点，并执行身份验证、访问控制、管理和统计等插件功能，最终把目标API返回的结果返回给请求用户。

**3.2.2 Kong的插件**

在众多API网关中，Kong中最具特色的一个特性是可以通过插件扩展已有功能，插件在API请求响应循环的生命周期中被执行。用户不仅可以直接使用官方提供的HTTP基本认证、密钥认证、CORS(Cross-origin Resource Sharing，跨域资源共享)、TCP、UDP、文件日志、API请求限流、请求转发以及nginx监控等插件，还可以使用Lua语言编写自己插件。

为了实现访问控制功能和数据平台其他部分功能的协调，API管理平台选取使用的API插件有:

* Key Authentication(密钥验证)：启用密钥验证后，每个用户在向API网关请求时须在查询字符串参数中或者请求头中加入为自己分配的API-Key以验证身份。
* ACL (访问控制列表)：使用API访问控制列表需要对API网关中的用户和API同时进行配置。每个用户的访问控制列表插件可以配置用户所属的组，而在API的访问控制列表插件可以将各个组添加到白名单或黑名单，从而达到访问控制的作用。
* Rate Limiting(速率限制)：可以通过配置对每秒、每分钟、每小时、每日、每月和每年的访问速率进行限制，当访问速率超过设置的请求时，API网关会拒绝用户的请求并返回相应的错误信息。
* UDP Log(UDP日志)：每次请求相应后将此次请求相应的日志发送到设定IP地址和端口，日志的内容包括请求大小、请求中携带的参数、响应状态、响应大小、验证信息等。

**3.2.3 Kong的管理**

Kong默认需要占用两个端口用来通信，分别为8000端口和8001端口。

8000端口为代理端口，Kong通过监听8000端口到来的请求执行其网关的功能。

8001端口负责对网关管理和操作。为了对API网关进行管理，在Kong的内部存在一个RESTful风格的API，因此在对Kong管理时，需要使用遵循RESTful风格的GET、POST、PATCH和DELETE四种HTTP请求类型对API进行管理操作。在实际使用API网关时，还应该设置防火墙对8001端口进行保护或者禁止外界对8001端口的访问，以保证API网关管理的安全。

## **3.3 基于Django框架的API管理系统的实现**

考虑到API网关Kong已经存在开源的python客户端，因此整个API管理系统选择使用python语言实现，用户端和管理员端Web App框架也相应的选择使用python语言下相对较为完善的Django框架。

与绝大多数Web 框架一样，Django同样基于MVC构造[8]。但是在Django中，控制器接受用户输入的部分由框架自行处理，所以 Django 里更关注的是模型(Model)、模板(Template)和视图(Views)，称为 MTV模式[9]。它们各自的职责如表3-1所示：

表3-1 Django的MTV层次-职责对应表

|  |  |
| --- | --- |
| 层次 | 职责 |
| 模型（Model），即数据存取层 | 处理与数据相关的所有事务： 如何存取、如何验证有效性、包含哪些行为以及数据之间的关系等。 |
| 模板(Template)，即表现层 | 处理与表现相关的决定： 如何在页面或其他类型文档中进行显示。 |
| 视图（View），即业务逻辑层 | 存取模型及调取恰当模板的相关逻辑。模型与模板之间的桥梁。 |

依据3.1节中API管理系统模块的划分和设计，以下三个小节将对API管理系统中三个主要功能模块具体的实现进行了介绍。

**3.3.1 Python-Kong-Client**

Python-Kong-Client是使用Python语言写成的开源Kong客户端，通过使用Python语言下的HTTP库requests将HTTP请求封装，以函数调用的形式替代手动发送HTTP请求对Kong进行管理，同时也能够降低上层API管理系统实现的复杂性和耦合性，方便进行集中统一控制。

在API管理系统实际运行时，利用Python-Kong-Client提供的不同接口可以创建多个不同的客户端同时对Kong进行管理操作。

部分API管理系统使用到的提供的客户端接口及其方法如下：

1. API管理客户端

* class APIAdminClient(APIAdminContract, RestClient):
* def \_\_init\_\_(self, api\_url):

通过实例化该类，能够创建出一个API管理客户端，调用下面介绍的类中的方法能够对API执行相应API管理的操作。

* def create(self, upstream\_url, name=None,request\_host=None, request\_path=None,strip\_request\_path=False,preserve\_host=False):

通过传递进入相应的参数并调用该方法，能够在Kong中新建一个API。

* def create\_or\_update(self, upstream\_url, api\_id=None, name=None, request\_host=None, request\_path=None,strip\_request\_path=False, preserve\_host=False):

通过传递进入相应的参数并调用该方法，如果api\_id或api\_name在Kong中不存在，则新建一个API，否则对参数选中的API进行修改。

* @backoff.on\_exception(backoff.expo, ValueError, max\_tries=3)  
   def delete(self, name\_or\_id):

通过调用该方法，传入API的名称或者ID，能够删除Kong中对应的API，当尝试删除失败时，会再尝试执行该方法3次。

1. 插件管理客户端

* Class APIPluginConfigurationAdminClient(APIPluginConfigurationAdminContract, RestClient):
* def \_\_init\_\_(self, api\_admin, api\_name\_or\_id, api\_url):

实例化该类时，需要传入使用插件的API的名称或ID。通过实例化该类，能够创建出一个管理API插件的客户端，调用下面介绍的类中的方法能够执行对API的插件执行创建、更新、删除等操作。

* def create(self, plugin\_name, enabled=None,consumer\_id=None, \*\*fields):

通过调用该方法，传入创建API插件的编号以及插件配置的json格式的配置信息，能够在API上创建一个对应的插件。

* def create\_or\_update(self, plugin\_name,plugin\_configuration\_id=None, enabled=None,consumer\_id=None, \*\*fields):

通过传递进入相应的参数并调用该方法，如果以插件尚不存在，则新建一个插件，否则对该插件进行修改。

* @backoff.on\_exception(backoff.expo, ValueError, max\_tries=3)  
   def delete(self, plugin\_id):

通过调用该方法，传入API插件在Kong注册时生成plugin\_id，能够删除Kong中对应的插件。

1. 用户管理客户端

* class ConsumerAdminClient(ConsumerAdminContract, RestClient):

def \_\_init\_\_(self, api\_url):

通过实例化该类，能够创建出一个用户管理的客户端，调用下面介绍的类中的方法能够对用户执行相应用户管理的操作。

* def create(self, username=None, custom\_id=None):

通过调用该方法，传入用户的用户名或者用户ID，能够在Kong内创建一个用户。

* @backoff.on\_exception(backoff.expo, ValueError, max\_tries=3)  
   def delete(self, username\_or\_id):

通过调用该方法，传入用户的用户名或者用户ID，能够在Kong内删除该用户。

1. 密钥验证管理客户端

* class KeyAuthAdminClient(KeyAuthAdminContract, RestClient):

def \_\_init\_\_(self, consumer\_admin, consumer\_id, api\_url):

实例化该类时，需要传入给用户添加密钥验证的用户ID。通过实例化该类，能够为该用户创建一个密钥验证客户端。调用下面介绍的类中的方法能够通过该客户端执行对密钥执行创建、更新、删除等操作。

* def create(self, key=None):

调用该方法时，可以传入想要设置的密钥，也可以置为空；如果传入的密钥为空，Kong会自动生成一个密钥。

* def delete(self, key\_auth\_id):

调用该方法时，传入Kong生成密钥时自动生成的key-auth-id，能够在Kong内删除该密钥。

1. 用户访问控制列表管理客户端

* class AclAdminClient(AclAdminContract, RestClient):

def \_\_init\_\_(self, consumer\_admin, consumer\_id, api\_url):

通过实例化该类，能够为该用户创建一个访问控制列表客户端。调用下面介绍的类中的方法能够通过该客户端执行访问控制列表执行创建、更新、删除等操作。

* def create(self, group):

调用该方法时，传入为该用户设置的所在组的名称，能为用户分配一个新的组。

* def create\_or\_update(self, acl\_id=None, group=None):

调用该方法时，传入创建时生成的acl\_id和新的组名，能够更改ACL中的组。

* def delete(self, acl\_id):

调用该方法时，传入传入创建时生成的acl\_id，能够删除该ACL。

**3.3.2 API管理系统管理员端的实现**

作为API管理系统管理端，管理员在登录管理员端后能够直接对API和用户执行全部的管理操作：API的注册、信息查看、修改、插件的增删、API开启关闭和向API网关同步等功能；用户的注册、删除、信息查看、增加验证密钥、增加用户访问控制组、开启关闭和向API网关同步等功能。

实现API管理系统管理员端时，使用了Django框架自带的Django admin管理工具。作为Django的特色功能和最强大的部分之一，Django admin管理工具能够自动生成Admin界面，它直接读取模型中的元数据，经过些许配置就能提供一个强大的、生产环境就绪的界面，使管理员能立即用它向站点中添加内容。

下面将分别介绍底层模型的建立和使用Django admin对API管理和用户管理的实现。

1. Model模型的建立

Django内置了自己的ORM(Object Relational Mapping)对象关系映射工具，Django还提供了一套自动生成的用于数据库访问的API，因此可以使用符合Django ORM支持语法的语句代替SQL语言对数据库进行新建、查找、修改、删除等操作。模型的每个属性都表示为数据库中的一个字段。

图3-3为Django中模型的UML图，下面将对各个类所代表的含义分别详细说明：



图3-3　API管理系统模型UML图

* KongProxyModel：该类是与Kong同步的抽象基类，在数据库中不会直接保存该表，主要包括Kong新建时生成Kong-id、创建时间、更新时间、是否同步等字段，所有需要与Kong进行交互的类都需要继承该类。
* APIReference：该类是保存API信息的类，继承自KongProxyModel。保存的主要字段除了API网关中必须的API的名称、API目标端点地址、访问API网关时的相对路径等信息，还有API注册时填写的中文名、对API的介绍、选择的API类型和API每秒每日访问速率限制以及上传的API的logo图片。
* ParameterReference：该类保存用户向API网关发起HTTP请求时应该携带在URL中的参数、参数意义和参数默认值。类中的api字段使用外键与APIReference关联。
* HeaderReference：该类保存用户向API网关发起HTTP请求时应该携带在Header中的参数、参数意义和参数默认值等信息。类中的api字段使用外键与APIReference关联。
* ErrorReference：该类保存终端API对用户发起的HTTP请求时出现错误可能返回的错误代码、错误意义等信息。类中的api字段使用外键与APIReference关联。
* PluginConfigurationReference：该类保存在Kong的插件相关的配置信息，继承自KongProxyModel，api字段使用外键与APIReference关联。类中字段主要有插件的名称、json格式的配置信息、开关等。
* ConsumerReference：该类是保存用户信息的类，同样继承自KongProxyModel。保存的字段分别是用户名、用户的ID和在API网关内的启用关闭。
* BuyReference：该类是用户保存用户购买信息的类，保存的字段有购买者、购买的API和购买时间。购买者consumer字段外键与APIReference关联，购买的API字段通过外键与ConsumerReference关联。
* ConsumerAuthentication：该类是继承自KongProxyModel的抽象类，唯一字段consumer使用外键与ConsumerReference关联，是后续保存Kong中用户的验证信息和访问控制表分组信息的基类。
* KeyAuthReference：该类继承自ConsumerAuthentication，用来保存用户使用密钥验证时的密钥信息，除了继承而来consumer用户字段，还有用来保存密钥的字段。
* AclReference：该类继承自ConsumerAuthentication，用来保存用户在访问控制是所处的组。除了继承而来consumer用户字段，还有用来保存分组信息的group字段。

1. API管理的实现

对于管理员而言，Kong的插件对其不应该是透明的，在API上传后，Kong的插件会被按照预先的设定配置好，管理员既可以在对Kong插件的使用方法和机制有一定的了解后，自主的配置、管理相应的API的插件；也可以对直接使用默认创建好的插件配置。

为了实现对API的管理，在建立的模型上，需要为APIReference模型创建一个ModelAdmin类，将该类在admin注册后，在后台管理网站就能实现对API信息的管理：

* class APIReferenceAdmin(ActionButtonModelAdmin):

admin.site.register(APIReference, APIReferenceAdmin)

通过配置list\_display、fieldsets等属性能够设置在列表页显示的项目和在详情页显示的具体内容；通过配置readonly\_fields属性可以将创建时间、修改时间置为只读类型。最后配置inlines将与API紧密相关的URL参数表、Header参数表、错误参数表和插件参数表显示在详情页中。

图3-4是实现后的API管理系统管理员端对API管理的示意图：



图3-4 API管理系统管理员端截图

1. 用户管理的实现

与API管理的实现相似，为ConsumerReference模型创建一个ModelAdmin后，配置list\_display、fieldsets、readonly\_fields等属性即能快速建立一个对于用户管理的后台。完成后的用户管理列表如图3-5所示：

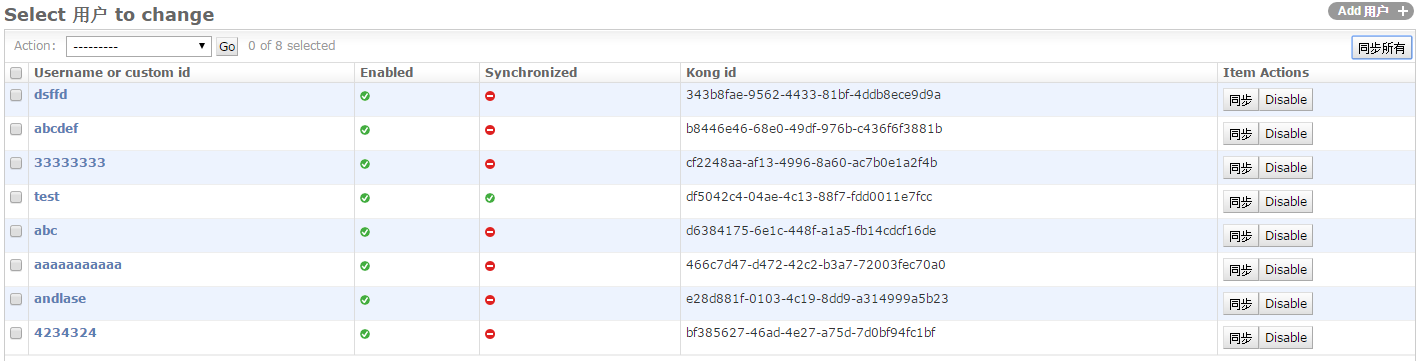


图3-5 API管理系统用户端截图

1. 与API网关同步的实现

在管理员端的API列表页和用户列表页上，各自存在启用/关闭、同步和全部同步三种Action按钮。为了实现API管理系统与API网关同步的功能，在管理端中实现了4个Action函数用以执行同步全部API、同步单个API、同步所有用户、同步单个用户四种操作。

* def synchronize\_api\_references(request, queryset=None):
* def synchronize\_api\_reference(request, pk, toggle\_enable=False):
* def synchronize\_consumer\_references(request, queryset=None):
* def synchronize\_consumer\_reference(request, pk, toggle\_enable=False):

在四个Action函数前，使用了修饰器@staff\_member\_required。通过使用该修饰器，django框架可以限定当前处于管理员状态时四个函数才会被成功调用，以此可以防止其他用户通过URL访问对API网关非法管理操作。

**3.3.3 API管理系统用户端的实现**

在API管理系统用户端的前端，使用了Bootstrap框架。Bootstrap 是一个用于快速开发 Web 应用程序和网站的前端框架，为开发人员创建接口提供了一个简洁统一的解决方案[6]。

在API管理系统的用户端，用户除了可以注册API后，同样应该可以对自己向平台提交的API进行全生命周期管理，包括提交API、修改API信息、改变API的访问权限、删除API等。对于用户端的用户而言，Kong中的各种插件对其应该是透明的，系统需要根据用户的设置自动的配置Kong中相应的插件。下面根据网页功能的划分介绍用户端的具体实现。

1. 用户注册

当收到用户注册请求时，需要在数据库中注册一个用户，还需要在用户之上为其新建一个验证密钥。新建用户对象和密钥验证对象的语句如下：

* consumer = ConsumerReference(username=username)
* KeyAuth = KeyAuthReference(consumer=consumer)

在建立之后，还需向API网关同步新建用户和密钥的请求，核心语句如下：

* client = factory.get\_kong\_client()
* logic.synchronize\_consumer(client, obj, toggle=False)

1. 注册API

在注册API的页面，需要为用户提供一个表单用以填写API的相关信息。Django框架提供了ModelForm功能，可以直接从模型自动映射出一个表单。根据Model生成表单的定义语句如下

* class APIForm(forms.ModelForm):

class Meta:  
 model = APIReference

在用户提交后，系统应该能对提交的表单进行验证，如表单中的API名称在系统中是否已经存在、API地址是否合法等。由于表单由模型生成而来，因此根据模型字段类型和属性的设置，Django框架可以自动对表单的内容进行验证。

表单验证成功后，系统将API所有者、API网关地址添加到实例中并保存，等待管理员审核后与API网管同步。

在用户成功提交API后，系统能够自动向设定的邮箱发送邮件以提醒管理员尽快对用户提交的API进行审核发布，加快审核发布的速度。使用Django自带的发送电子邮件模块能够容易的做到这一点：

* send\_mail(u'用户'+str(username)+u'发布了一个新的API', message, 'sender@xxx.com',['receiver@xxx.com'], fail\_silently=False)

1. 修改API

修改API的前端模版和表单都可以直接复用注册API的部分，只是传入模版的表单应该将初值赋为该API当前存在数据库中信息。创建带有初始数据表单的核心语句如下：

* form = APIForm\_modify(instance=api)

1. 查看API列表

在API列表页，根据当前访问控制的设置，用户能够看到具有使用资格的API。因此相应用户的请求时，需要向模版动态的传入当前数据库中所有开启的API。核心语句如下：

* api\_list = APIReference.objects.filter(enabled\_\_exact=True)

1. 查看API详情和使用API

在API列表页中可以点击API的logo图片查看API的详情，并能申请使用该API。

在动态生成API详情页面时，除了需要将API的实例传入模板中，还需要传入用户的API-key，方便用户直接在API详情页直接获得完整的URL路径。

当用户点击使用按钮后，系统能够自动在该用户的ACL插件中新添group，以获得对该API的使用权。

1. 用户中心

在用户中心，用户可以看到自己发布API的相关信息，并能执行管理API的访问控制策略、删除API等操作。

实现时，首先根据存在Session中的信息反查出用户名，通过用户名获得该用户实例，使用api中的owner外键反查得到该登录用户申请的所有API实例。除了看到自己发布的API，用户在用户中心页面还可以查看自己购买的API。查看购买的API时，通过外键反向找到该用户购买的所有API，将购买API的信息返回。核心语句如下：

* username = get\_username(request) #获得用户名
* person = ConsumerReference.objects.get(username = username)#通过用户名获得用户实例
* apis = person.infos.all()#获得用户所有的API
* buy\_apis = person.Buy\_consumer.all()#获得用户购买的API

第四章　API访问控制技术的设计与实现

API作为数据开放的接口，如何保证API的安全性，防止因API安全性的缺失导致数据泄漏具有十分重要的研究意义。

根据文中介绍的常见的访问控制模型和API访问控制技术，本章首先介绍了三种API访问控制决策算法的设计；接下来介绍了其中一种访问控制决策算法在API管理系统中的实现。

## **4.1 API访问控制决策算法的设计**

访问控制决策算法是实现访问控制的核心内容，本节将介绍在实际情况下，将API访问控制的需求和常见的访问控制模型结合后，设计出的三种API访问控制决策算法。

**4.1.1 ABAC与API管理的结合**

在主体用户使用API时，发送的请求中携带的各种信息以及外部环境当前的状态都可以成为决定请求能否被允许的属性，如API请求中携带的参数、请求中携带的API密钥、请求的时间、被访问API当前的状态等，都可以成为ABAC决策中的属性：

文献[11]创新性的首次给出了一个基于ABAC的API访问控制模型，基于ABAC的API访问控制决策一般算法如4-1所示：算法的输入为REQ和POL和ATT，其中REQ表示主体的访问请求，请求中包括欲访问的客体APIi和主体身份User；POL表示策略的集合，集合中的每个poli都为设定的一个策略；ATT是该次访问的属性集，属性集中的各属性atti可能来自于REQ，也有可能来自外部环境和API的状态等。算法的输出Decision是布尔型，表示对于请求的决策。

|  |
| --- |
| 算法4-1　基于ABAC的API访问控制决策算法 |
| 输入：  REQ<APIi, User>  POL = {pol1,pol2,…polm}  ATT = {att1,att2,…attm} |
| 输出:  Decision : Boolean |
| BEGIN  For all(poli ∈POL) do  IF ATT\not\rightarrowpoli THEN  Decision ← Refuse  END IF  END For  Decision ← Accept  END. |

**4.1.2.DAC和MAC与API管理的结合**

在实际对API的管理和使用中，会遇到以下情形:

情形一：API开发者希望只有自己能够使用API，但是鉴于对API使用情况统计和对API访问速率进行限制等功能的需求，API被注册在了API管理系统中。

情形二：API是公开的或者受限使用，当API处于调试状态或者API自身发生改变后，API无法正常使用，此时API对于其他用户而言应该是不可见的，但是API所有者需要对API正常使用以对API进行调试。

综合两种情形，我们需要在API管理系统中设置一种访问控制模式——私有模式：只有API拥有者能够通过API管理系统访问API，该API对于用户来说应该是不可见的，用户也不能访问该API。

基于DAC和MAC私有模式的访问控制决策的算法如下所示：算法的输入为REQ，REQ表示主体用户的访问请求，请求中包括欲访问的客体APIi和主体身份User；算法的输出Decision是布尔型，表示对于请求的决策。

|  |
| --- |
| 算法4-2　基于DAC和MAC的API访问控制决策算法 |
| 输入：  REQ<APIi, User> |
| 输出:  Decision : Boolean |
| BEGIN  IF Owner(APIi) = User THEN  Decision ← Accept  ELSE  Decision ← Refuse  END. |

**4.1.3 RBAC与API管理的结合**

更多的时候，我们需要根据用户的身份来对进行更细粒度的管理，在实际情况中，我们会遇到以下情形：

情形一：某学校建立了一个API用于方便学生使用学号查询电话号码。显然对于来自非该学校的学生身份的请求，API管理应该能够拒绝请求。

情形二：某医院建立一个API用于查询病人的历史病历、等信息。处于对病人隐私安全的考虑，API应该仅允许医生身份和病人本人对于API的使用。

综上两种，根据主体用户的身份、角色作为决策依据，使用RBAC模型来对API进行管理不失为一种合理的方法[1]，基于RBAC的API访问控制决策算法如下所示：算法的输入为REQ和APIi\_ACCROLE ,其中REQ表示主体的访问请求，请求中包括欲访问的客体APIi和主体身份User，APIi\_ACCROLE表示APIi所允许访问的角色的集合。算法的输出Decision是布尔型，表示对于请求的决策。

|  |
| --- |
| 算法4-3 基于RBAC的API访问控制决策算法 |
| 输入：  REQ<APIi, User>  APIi\_ACCROLE={accrole1,accrole2,…accrolem} |
| 输出:  Decision : Boolean |
| BEGIN  For all(rolei∈ROLE\_OF(USER)) do  IF rolei∈APIi\_ACCROLE THEN  Decision ← Accept  END IF  END For  Decision ← Refuse  END. |

## **4.2基于ABAC的API访问控制的实现**

对于普通用户而言，用户能够点击每个API详情页面上的购买/使用按钮，来达到使用该API的功能。对于没有购买该API的用户和无法确认其身份的用户，向API网关请求该API应该被拒绝。此外，还应该根据API发布者在发布时填写的速率请求对API访问的速率进行限制，以防止目标API服务器因过载或恶意攻击等因素导致的服务崩溃发生[5]。

因此，用户每次向API网关请求时，请求中携带的密钥、用户的ACL状态和API当时所处的状态是ABAC中的属性。允许访问的策略为请求中密钥验证成功、用户对API具有使用权和API当前的访问速率在正常范围内。

幸运的是，在使用API网关Kong的插件后，在执行API网关的功能之前，插件能够自动被执行并根据当时的状态信息正确合理的做出决策。下面两个小节将分别介绍使用选取的Key-auth、ACL和Rate Limiting后访问控制决策流程以及在API管理系统中配置Kong插件的实现。

**4.2.1 API网关访问控制决策流程**

如图4-1是在Kong中设置Key-auth、ACL和Rate Limiting三个插件后Kong执行时的决策流程图。



图4-1 使用插件后的API网关决策流程图

当Kong收到来自用户的请求时，首先在请求的URL参数中或Header中寻找密钥，如果找到则继续，否则返回没有找到API-key的错误信息。

接着Kong将对用户请求中携带的密钥进行验证，如果密钥正确则能确定其身份并继续，否则返回密钥错误信息。

下一步Kong将根据当前API网关中该API插件中ACL的设定，判断该用户的ACL插件中设置的所在的组是否具有访问该API的权限，如果没有访问该API的权限，则返回不能使用的错误信息，否则进行下一步。

最后Kong将对根据API发布者设定的该API访问速率限制进行判断，如果该API目前的访问速率已经超出了API发布者设定的每秒速率或每日速率，则返回超过速率限制错误信息。只有当目前该API的访问速率在设定范围内，才执行API网关的功能，将用户的请求转发到目标API上。

**4.2.2 Kong插件配置的实现**

为了实现上节中的利用Kong插件实现基于ABAC的访问控制，每一个API被上传时相关插件能够自动被配置。系统中为配置插件预留的接口如下：

* def ConfigPlugin(API):

ConfigPlugin 函数是API发布时配置相关插件的函数，调用该函数需要传入需要配置API的实例。

调用该函数后，首先判断该API上传时是否为对每秒访问速率和每日访问速率有限制，如果有则为其配置Rate Limiting插件，否则跳过该步；接下来分别配置KEY\_AUTHENTICATION插件和ACL插件，配置的核心语句如下：

* plugin=PluginConfigurationReference(api=API,plugin=Plugins.RATE\_LIMITING, config=obj)
* plugin=PluginConfigurationReference(api=API,plugin=Plugins.KEY\_AUTHENTICATION, config=json.dumps({}))
* plugin = PluginConfigurationReference(api=API, plugin=Plugins.ACL, config=json.dumps(acl))

# 第五章 系统测试

本章对设计、实现的API管理系统中的模块、功能以及性能进行测试，以期发现错误，提高性能。在任何系统的开发中，测试都是其中很重要的一部分。测试有助于发现系统的漏洞和不完善的地方，有助于改进系统功能、提高系统性能。本系统主要从API管理系统的功能和性能以及API访问控制技术性能两个方面进行测试。

## 5.1 功能测试

本节主要针对文章中实现的API管理系统进行功能测试。以对系统提供的各项功能进行测试和验证，根据各个测试用例，检查系统是否实现了要求的功能。

**5.1.1 用户注册功能测试**

1. 测试目的：验证用户是否能够正确完成注册，并自动为用户分配自己的API-key
2. 测试方案：在API管理系统用户端用户注册页面新注册一个用户Test\_register。
3. 预测结果: 新用户Test\_register注册成功，其信息可以在管理员端以及数据库中查询到；KeyAuthReference中存在为用户创建的API-key
4. 测试结果：与预期一致。

**5.1.2 用户发布API功能测试**

1. 测试目的：1）验证用户能否正确发布API；2）验证用户发布API时提交的表单有错误能否正确提示错误。
2. 测试方案：1）在API管理系统用户端发布API页面提交一个新的API；2)在提交的表单中人为设置几处不合法数据。
3. 预测结果: 使用合法的数据提交API后，其信息可以在管理员端以及数据库中查询到；当提交的表单不合法时，提示不合法的表单域和原因。
4. 测试结果：与预期一致。

**5.1.3 用户修改API功能测试**

1. 测试目的：1）验证用户能否正确修改自己发布的API；2）验证用户修改API时提交的表单有错误能否正确提示错误。
2. 测试方案：1）在API管理系统用户中心修改自己已经发布的API；2)在提交的表单中人为设置几处不合法数据。
3. 预测结果: 使用合法的数据提交API后，其信息可以在管理员端以及数据库中查询到；当提交的表单不合法时，提示不合法的表单域和原因。测试结果：与预期一致。
4. 测试结果：与预期一致。

**5.1.4 用户删除API功能测试**

1. 测试目的：验证用户在用户中心网页对自己发布的API点击删除按钮后能否正确的在API管理系统和API网关中删除API。
2. 测试方案：模拟用户在用户中心网页对自己发布的API点击删除按钮。
3. 预测结果: 在管理员端中和API网关中查看API均被删除，与该API有关的相关购买信息及API插件均被删除。
4. 测试结果：与预期一致。

**5.1.5 用户购买/使用API功能测试**

1. 测试目的：验证用户在API详情页面点击购买/使用按钮后能否对API实现购买操作。
2. 测试方案：模拟用户登陆后在API详情页面点击购买/使用按钮。
3. 预测结果: 点击后提示购买成功，用户使用自己的API-key依据API详情页面中给出的使用方法向API网关地址发送请求，能够给用户返回API的返回信息。
4. 测试结果：与预期一致。

**5.1.6 管理员管理API功能测试**

1. 测试目的：验证管理员登录后在管理端能否对API进行新增、修改、删除、审核启用等操作。
2. 测试方案：在管理员端API管理页面分别对新建API、修改API、删除API、启用API和同步API进行测试。
3. 预测结果: 对API等新增、删除和修改操作均能正常工作。点击启用 和同步按钮后能够完成在API网关中的新增、修改和删除操作。
4. 测试结果：与预期一致。

**5.1.7 管理员管理用户功能测试**

1. 测试目的：验证管理员登录后在管理端能否对用户进行新增、删除、启用和关闭等操作。
2. 测试方案：1)在管理员端用户管理页面分别对新建用户、删除用户、启用用户和关闭用户进行测试。2)新建用户成功后查看是否为用户创建一个API-key。
3. 预测结果: 对用户的新增和删除操作均能正常工作。点击启用和同步按钮后能够完成在API网关中的新建一个用户，查看创建的用户能够看到为其分配的API-key。
4. 测试结果：与预期一致。

## 5.2 性能测试

本节主要针对使用Django框架的API管理系统和后台的API网关进行性能测试，测试机基于Openstack，为其分配了2GB内存和双核CPU。性能测试所使用的工具为Locust。

Locust是一个简单易用的分布式负载测试工具，主要用来对网站进行负载压力测试，也可以对其他系统进行测试。Locust使用Python编写模拟用户行为的代码，无需繁琐的配置，由于Locust完全是事件驱动的，因此在单台机器上能够支持几千并发用户访问。与其它许多基于事件的应用相比，Locust并不使用回调，而是使用gevent，由于gevent是基于协程的，因此可以用同步的方式来编写异步执行的代码。测试流程如图5-1所示：



图5-1 性能测试流程图

1. API管理系统性能测试

为了测试使用Django框架的API管理系统的性能，我们在测试中分别模拟生成10、30、50、100个用户每隔1-3秒随机对API管理系统的首页、API列表页面、几个API详情页和用户中心随机访问，测试运行图如图5-2所示：

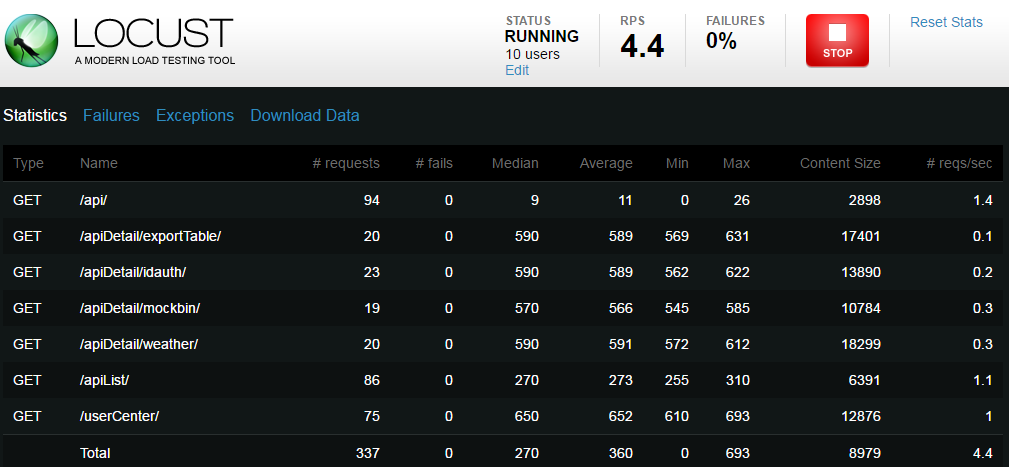


图5-2 API管理系统性能测试运行图

在每组完成超过5000次模拟访问后，测试结果如表5-1所示：

表5-1 基于Django开发服务器的API管理系统性能测试表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 模拟用户数 | 最小响应时间(ms) | 最大响应时间(ms) | 中位响应时间(ms) | 平均响应时间(ms) | 错误率 |
| 10 | 0 | 1029 | 560 | 376 | 0.00% |
| 30 | 0 | 21834 | 2232 | 2165 | 0.00% |
| 50 | 79 | 23723 | 5899 | 5885 | 0.00% |
| 100 | 130 | 60114 | 13000 | 13423 | 0.00% |

根据Django官方的说法，Django自带的开发测试服务器目的只是用来测试使用，并不适合直接部署服务，因此在用户数量增加后性能衰退的很快，下面将APP利用WSGI接口部署在Nginx服务器后再次测试结果如表5-2所示。

表5-2 基于uwsgi和Nginx服务器的API管理系统性能测试表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 模拟用户数 | 最小响应时间(ms) | 最大响应时间(ms) | 中位响应时间(ms) | 平均响应时间(ms) | 错误率 |
| 10 | 0 | 1198 | 320 | 341 | 0.00% |
| 30 | 9 | 4978 | 1900 | 2006 | 0.00% |
| 50 | 79 | 7765 | 4500 | 4519 | 0.00% |
| 100 | 1184 | 14485 | 11000 | 10572 | 0.00% |

通过两次测试结果对比可以表明，使用uwsgi和Nginx服务器后各项指标均有所提升，特别是得益于Nginx的均衡负载的能力，最大响应时间得到了有效的控制。鉴于测试服务器性能较差，且数据库为使用mysql远程链接，因此API管理系统在性能测试中未能完全发挥出最佳水准，相信在改进优化后还可以提高一定的性能。

1. API网关性能测试

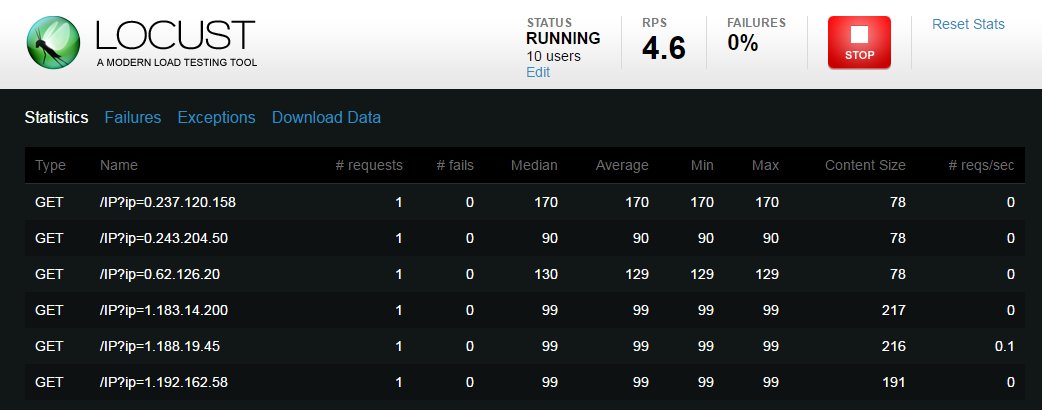
我们选取了百度API store中免费的查询IP地址信息API作为性能测试的API，在测试中我们分别模拟生成10、30、50、100个用户，为了防止查询相同内容时缓存对结果产生的影响，我们采取每1-3秒钟随机生成一个IP地址的方法向API请求查询。测试运行图如图5-3所示：

图5-3 API网关性能测试运行图

每组查询超过5000次后，测试结果分析表如表5-3所示。

表5-3 API网关性能测试统计表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 模拟用户数 | 最小响应时间(ms) | 最大响应时间(ms) | 中位响应时间(ms) | 平均响应时间(ms) | 错误率 |
| 10 | 79 | 4099 | 110 | 138 | 0.00% |
| 30 | 75 | 1460 | 120 | 150 | 0.00% |
| 50 | 79 | 3134 | 120 | 155 | 0.00% |
| 100 | 130 | 60114 | 140 | 173 | 0.00% |

测试结果表明虽然随着使用用户数的增大，API网关的响应速度也随之变慢，但是相应速度均在可接受范围内，且错误率一直保持为0，因此Kong能够满足一般情况下作为API网关的性能需求。

# 第六章 总结与展望

## 6.1 总结

本文在对目前市场上主流的API市场以及API管理技术及方案研究后，设计并实现了一个符合需求的完整API管理平台，在完成的API管理平台的基础上，对API访问控制相关算法进行了设计和实现。本文完成的工作有：

1. API管理和API访问控制技术研究综述

本文的第二章中对目前国内外使用API管理的现状和主流方法进行了介绍和说明；接着介绍了目前学术界和工业界主流的访问控制技术的主要内容和特点。第二章的最后，介绍了目前主流的API访问控制技术的相关技术背景。

1. API管理平台的设计与实现

API管理平台作为实现API管理和API访问控制技术的基础，依据第二章中对于API管理部分的研究，第三章首先介绍了根据需求设计的API管理平台的架构；接着依据从下到上的原则，对选取的开源网关Kong的结构和原理进行了介绍和说明。然后，详细的介绍了利用python下的Django框架实现API管理系统的具体过程和细节。

1. API访问控制决策算法

根据第二章中介绍的目前主流的访问控制模型，结合实际情况的需求，分别基于ABAC、MAC和DAC以及RBAC设计了三种适用范围不同的访问控制决策算法。

1. API访问控制技术的实现

在文章的最后，利用开源网关提供的相关插件，实现了基于ABAC模型的API访问控制技术。

经过努力，本人成功完成了开题之初设定的研究目标，完成了对API管理平台和API访问控制技术的设计和实现。经过本次研究和实践工作，自己的科研能力得到了提高，同时也提高了自身的工程能力，对于科研工作中实践和理论的结合有了更深入的体会。

## 6.2 展望

随着各方面对API管理的重视，未来对API管理和访问控制管理的研究展望如下：

1. API统计与分析

除了为用户提供的基本上传API、删除API和修改API等基本操作，以后还可以考虑为用户提供统计和分析功能，通过该功能，可以对使用自己上传的API的使用情况和用户情况能够有更好的了解和掌握，方便进一步对API的功能进行配置和改进。

1. API计费的功能的实现

目前国内外主流的API市场如百度API store、数据堂和Mashape等，大多数使用

按次数收费或者按时间收费的模式。但是在Kong中并没有直接提供相应的插件，下一步，如何在设计的API管理平台中完成计费功能，还需要进一步对Kong的原理进行理解后写插件完成。

1. 更多的API访问控制技术

除了在本文中介绍到的MAC、DAC、RBAC和ABAC三种基本模型能与API管理相结合。如Bell-Lapadula机密性模型、Biba完整性模型、Clark-Wilson模型和Chinese Wall模型，在未来，它们能否与API管理找到契合点，能否融入API管理中，能否在API管理中找到自己的作用，还需要进进一步的研究。

总之，作为未来数据和信息管理和控制的重要环节，API管理还需要投入更多的研究工作，值得挖掘更大的潜力和价值。

参考文献

[1] Sirisha A, Kumari G G. API access control in cloud using the Role Based Access Control Model[C]. Trendz in Information Sciences & Computing (TISC), 2010. IEEE, 2010:135 - 137.

[2] Wu M Y, Lee T H. Design and implementation of cloud API access control based on OAuth[C]. TENCON Spring Conference, 2013 IEEE. 2013:485-489.

[3] Windley P J. API Access Control with OAuth: Coordinating interactions with the Internet of Things.[J]. Consumer Electronics Magazine IEEE, 2015, 4(3):52-58.

[4] Gangrade A, Nigam B. Securing API Using API Gateway[J]. Networking and Communication Engineering, 2015, 7(4): 143-146.

[5] Michels O, Loveless C. Methods for analyzing, limiting, and enhancing access to an internet API, web service, and data: U.S. Patent 9,027,039[P]. 2015-5-5.

[6] Efron B, Tibshirani R J. An introduction to the bootstrap[M]. CRC press, 1994.

[7] Forcier J, Bissex P, Chun W. Python web development with Django[M]. Addison-Wesley Professional, 2008.

[8] 刘志凯, 张太红. Django框架在web开发中的应用[J]. 农业网络信息, 2015(2):51-52.

[9] 刘班. 基于Django快速开发Web应用[J]. 电脑知识与技术, 2009, 5(7):1616-1618.

[10] Ierusalimschy R. Programming in Lua, Third Edition[M]. Lua.Org, 2013.

[11] Keltoum B, Mohamed B, Mohamed B. A Novel Access Control Model for Securing Cloud API[J]. on Networking and Advanced Systems, 2015: 67.

[12] 鲍连承, 赵景波. 访问控制技术综述[J]. 电气传动自动化, 2006, 28(4): 1-5.

[13] Kuhn D R, Coyne E J, Weil T R. Adding attributes to role-based access control[J]. IEEE Computer, 2010, 43(6): 79-81.

[14] Hur J. Improving security and efficiency in attribute-based data sharing[J]. Knowledge and Data Engineering, IEEE Transactions on, 2013, 25(10): 2271-2282.

[15] 涂山山. 云计算环境中访问控制的机制和关键技术研究[D]. 北京邮电大学, 2014.

[16] 左忠诚. EPCglobal框架下动态数据服务访问控制技术的研究与实现[D]. 北京邮电大学, 2015.

[17] 王峰, 张彬, 李国辉,等. RBAC访问控制机制原理分析与实现[J]. 2009.

致　　谢

在论文即将完成之际，自己的本科阶段也就要结束了。在本科的四年的时间里，不论是在学习、科研还是生活中，我获益匪浅，这段经历将会是我人生中最宝贵的财富。在此，我要对培养我的北京邮电大学以及学校的各位老师和同学致以真诚的谢意。作为一名北邮人，我感到非常的自豪。

我要感谢我的毕业设计指导老师同样也即将是我研究生阶段的导师——孙岩教授，孙老师平易近人，学识渊博，在选题、开题、工程编写和毕业设计论文写作各个方面都给予了我很大的帮助、建议和支持。同时，孙老师渊博的学识、严谨的作风和敬业的精神都值得我不断学习。

我要感谢罗红老师。罗红老师治学严谨，以身作则，在前期调研阶段和项目完成阶段都给予了我莫大的指导和帮助。我还要感谢902实验室的张帅学长和魏邦联学长，他们在毕业设计从选题、项目前期调研到项目最终完成都给予了我很多技术上的帮助。

特别得，我要感谢我的家人，你们的养育之恩永生难忘。我取得的任何一点成绩，都离不开你们的关爱和帮助。祝爸爸妈妈身体健康，诸事顺遂。

最后，我要感谢即将帮助我完成答辩的所有老师、专家，你们辛苦了。