



本科毕业设计(论文)

题目：智能家居物联网服务平台设计

院（系）：	<u>计算机科学与工程学院</u>
专 业：	<u>物联网工程</u>
班 级：	<u>14060610</u>
学 生：	<u>兰 洋</u>
学 号：	<u>14060610105</u>
指导教师：	<u>白小军</u>

2018 年 6 月

题目：智能家居物联网服务平台设计

院（系）： 计算机科学与工程学院
专 业： 物联网工程
班 级： **14060610**
学 生： 兰 洋
学 号： **14060610105**
指导教师： 白小军

2018 年 6 月

西安工业大学毕业设计（论文）任务书

院（系）计算机学院 专业 物联网工程 班 14060610 姓名 兰洋 学号 14060610105

1. 毕业设计（论文）题目： 智能家居物联网服务平台设计
2. 题目背景和意义： 智能家居是物联网技术的一个重要应用领域，近几年来得到了广泛的研究。智能家居系统一般采用三层体系结构构建：首先使用各类传感器感知周围的环境信息并组网传输，然后通过网络层将感知到的数据传递给后台服务器进行存储和处理，最后上层应用从服务层获取数据，实现业务管理、数据可视化及反馈控制等各项功能。
3. 设计(论文)的主要内容（理工科含技术指标）： 本课题拟研究搭建智能家居应用的云端服务平台，该平台作为系统的中间层，一方面与 ZigBee 数据采集系统通信，获取、处理采集到的数据，并实现数据的持久化管理；另一方面与应用前端通信，为应用层提供数据查询等服务，并能辅助应用前端，实现智能家居设备的反馈控制功能。
4. 设计的基本要求及进度安排（含起始时间、设计地点）：
第 1 周到第 2 周 选题、收集资料、理解题目、确定开发工具、准备开题报告；
第 3 周至第 6 周 完成软件的需求分析、概要设计、总体框架（模型）设计；
第 7 周至第 12 周 完成各模块的设计；初步完成程序代码的编写工作；
第 13 周至第 14 周 对整个系统进行组合及调试，修改；
第 15 周至第 18 周 撰写论文并准备答辩；设计地点：校内。
5. 毕业设计（论文）的工作量要求 上机时数不少于 200 机时，撰写不少于 15000 字的论文
 - ① 实验（时数）*或实习（天数）： 18 周
 - ② 图纸（幅面和张数）*： 不限
 - ③ 其他要求： 查阅文献资料不少于 30 篇，其中英文文献不少于 5 篇。

指导教师签名： 年 月 日

学生签名： 年 月 日

系（教研室）主任审批： 年 月 日

说明：1 本表一式二份，一份由学生装订入附件册，一份教师自留。

2 带*项可根据学科特点选填

智能家居物联网服务平台设计

摘要

随着物联网、大数据、云计算等技术的发展成熟，推动了物联网应用的蓬勃发展，智能家居作为物联网技术的一个重要应用领域，近几年来得到了广泛的研究，也出现了大量的应用产品。目前智能家居设计中主要存在缺少理论依据，现有系统控制能力差、扩展性不足、服务器资源利用效率不高，已有平台接入门槛高、兼容性差等问题。

针对以上问题，拟采用物联网四层体系架构，即在原来物联网三层体系架构加入一个平台层，连接物联网应用层和感知层的技术方案，研究开发一套智能家居物联网服务平台。

本文研究了智能家居系统相关理论和诸多技术方案，开发了一套基于 RESTful 风格服务的 Web PaaS/IoT PaaS 混合的智能家居物联网服务平台，该平台一方面为应用层提供 Web Service 接口和 MQTT 接口，另一方面为感知层提供 MQTT 协议的事件接口，数据存储方面采用 InfluxDB 存储时序数据。智能家居服务平台，结合智能家居网关和应用前端，实现海量设备接入、设备数据云端持久化存储和远程控制等重要功能。

经过严格的测试，这套智能家居物联网服务平台可以有效解决目前智能家居系统的诸多问题，能够很好地提供安全可靠的服务。

关键字：智能家居；物联网平台；MQTT；Web Service；REST

Design of IoT Service platform for smart home system

Abstract

With the maturity of technologies such as Internet of Things, big data, cloud computing, the development of Internet of Things applications has been booming. Smart homes, as an important application area of Internet of Things technology, have been widely studied in recent years, and a large number of applications have also emerged product. At present, smart home design mainly exists, lacks theoretical basis, existing system control capability is poor, scalability is insufficient, server resource utilization efficiency is not high, and there are problems such as high access threshold and poor compatibility of the platform.

In view of the above issues, the four-tier architecture of the Internet of Things is proposed, that is, a platform layer is added to the original three-tier architecture of the Internet of Things to connect the technology solutions of the application layer and the perception layer of the Internet of Things, and a set of smart home internet of things service platforms is researched and developed.

This article studies the theory and many technical solutions of smart home systems, and develops a web-based PaaS/IoT PaaS hybrid smart home networking service platform based on RESTful-style services. The smart home service platform adopts Web Service technology to provide web interfaces on the one hand. On the other hand, the MQTT protocol event interface is implemented based on the Web interface. InfluxDB time series storage is adopted for smart home device data, combined with a smart home gateway to achieve concurrent access to massive smart home IoT devices and persistent storage of device data clouds, and remote intelligent control. Other important functions.

After a rigorous software testing process, this smart home IoT service platform can effectively solve many problems in current smart home systems and can provide a safe and reliable service.

Keywords: Smart Home; IoT Platform; MQTT; Web Service; REST

目录

摘 要.....	I
Abstract.....	II
1 绪论.....	1
1.1 选题背景和研究意义.....	1
1.1.1 选题背景.....	1
1.1.2 研究意义.....	1
1.2 国内外研究现状.....	2
1.2.1 国外研究现状.....	2
1.2.2 国内研究现状.....	2
1.3 主要研究内容.....	3
1.3.1 智能家居的总体结构.....	3
1.3.2 本文的研究内容.....	3
1.4 论文结构.....	4
2 相关技术介绍.....	5
2.1 Web Service.....	5
2.1.1 REST Web Service.....	5
2.1.2 Flask 框架.....	6
2.2 物联网通信协议 MQTT.....	6
2.2.1 MQTT 消息传递机制.....	6
2.2.2 HiveMQ 和 MQTT.fx.....	7
2.3 数据存储系统.....	7

2.3.1 MongoDB.....	7
2.3.2 InfluxDB.....	7
2.3.3 Redis.....	8
2.4 其他工具.....	8
3 需求分析.....	9
3.1 功能需求.....	9
3.1.1 应用层功能需求.....	9
3.1.2 感知层功能需求.....	11
3.2 性能需求.....	11
4 系统概要设计.....	12
4.1 应用层服务接口.....	12
4.1.1 用户接口.....	12
4.1.2 设备接口.....	13
4.2 消息处理.....	14
4.3 数据库设计.....	14
4.3.1 概念模型.....	15
4.3.2 逻辑模型.....	15
4.3.3 BSON 数据存储格式.....	17
4.4 软件架构设计.....	19
5 详细设计.....	20
5.1 系统模块设计.....	20
5.1.1 服务接口模块.....	20

5.1.2 事件管理模块.....	22
5.1.3 综合管理模块.....	23
5.1.4 MQTT 事件模块.....	24
5.2 系统性能优化.....	25
5.2.1 使用 gunicorn 优化.....	25
5.2.2 使用 Nginx 优化.....	26
6 系统部署和测试.....	27
6.1 系统部署.....	27
6.1.1 系统配置信息和端口分配.....	27
6.1.2 部署说明.....	27
6.2 系统测试.....	28
6.2.1 测试环境.....	28
6.2.2 单元测试.....	29
6.2.3 集成测试.....	30
6.2.4 测试结论.....	33
7 总结和展望.....	34
参考文献.....	35
致 谢.....	37
毕业设计（论文）知识产权声明.....	38
毕业设计（论文）独创性声明.....	39
附录.....	40

1 绪论

随着科技的进步,尤其是物联网技术在智能家居领域的广泛应用,目前不论是国内还是国外都在智能家居领域取得一定成果。要研究智能家居系统,就需要首先了解国内外智能家居领域发展趋势以及成果。本章主要介绍智能家居物联网服务平台的选题背景和研究意义,以及国内外的研究现状。最后阐述了本文主要研究内容。

1.1 选题背景和研究意义

本节主要从目前智能家居领域的发展情况、突出的问题介绍本文的选题背景,然后从研究智能家居系统所产生的社会价值阐述其研究意义。

1.1.1 选题背景

改革开放以来,我国在政治、经济、科技、文化等领域都取得长足发展,人民生活水平提高,人民对居家环境的质量提出更高的要求。在物联网、云计算、大数据等技术的大环境下,智能家居系统需要为用户随时随地提供安全可靠、节能并且个性化的服务,与此同时需要在保证海量设备接入数据的并发和处理能力。作为家居智能化发展到一定阶段的重要组成部分——智能家居服务平台的作用显得尤为重要。智能家居服务平台化是家居行业发展的必然趋势。近几年来智能家居得到了广泛的研究,虽然现了大量的产品,但同时智能家居系统逐渐暴露出一些问题,例如:资源受限、不够便捷、不容易扩展、网络环境不够安全等。

目前国在内外智能家居系统主要基于物联网应用的三层架构,存在很多问题。

(1) 已有解决方案与现实脱节,很难适应实际情况。

(2) 以嵌入式网关为核心构成的智能家居系统,对智能家居的控制能力有限、扩展能力差、服务器资源利用率不高。

(3) 虽然基有了一些基于平台的智能家居物联网系统,但存在接入门槛过高、各平台之间兼容性差等问题。

1.1.2 研究意义

研究基于平台的智能家居物联网服务平台可以解决原来基于三层体系架构

的智能家居系统所面临的众多问题。例如：智能家居控制能力弱、扩展性差、服务器数据压力过大等问题。

鉴于诸多智能家居问题，研究智能家居有助于促进家居行业发展和物联网技术的应用，极具学术价值。具体而言，有助于让智能家居系统在资源受限和特殊网络的环境下平稳运行，即节约能源又安全可靠还能为用户提供个性化服务。在经济发展方面，研究智能家居有助于促进经济繁荣。文化方面，有助于提高人民幸福指数，丰富物质和精神生活。最后，智能家居行业的发展有助于提高综合国力和国际影响力。

1.2 国内外研究现状

目前国内外有许多个人和组织都在研究智能家居系统，并且取得一系列成果，以下进行简要介绍。

1.2.1 国外研究现状

早期美国提出“智慧地球”概念，欧盟委员会发布《物联网——欧洲行动计划》，提出了包括芯片、技术研发等在内的 14 项框架内容。如今，亚马逊推出 AWS IoT，使互联设备可以轻松安全地与云应用程序及其他设备交互；微软推出 Azure IoT；IBM 推出 IBM Watson IoT 提供全面的云托管服务；Exosite Murano 是一个基于云的提供安全可扩展、支持端到端 IoT 软件平台。而智能家居作为物联网技术的一个重要应用领域，自然成为各巨头的争夺焦点。

早在 2013 年德国电信就推出了 Smart Home 平台，获得 ZigBee 智能家居产品的国际认证，并且提供多个套件，包括门窗监控、烟雾报警、暖气控制、智能插座等多个模块；AT&T Digital Life 更加注重智能家居系统安全问题，目前推向市场的两款套装包括安全模块和动态传感模块，除了门窗监控以外，还有监控摄像和车库监控。

1.2.2 国内研究现状

目前国内研究状况：在服务风格方面：程冬梅、王瑞聪等人提出的基于 REST 服务架构的物联网服务平台非常具有借鉴意义。在数据持久化存储方面：杨杰和张启飞提出基于 MongoDB 的物联网服务平台用户信息管理方案。系统架构方面：王易川、李文钧提出的基于 ZigBee、WiFi 和云服务器集群的智能家居系统设计方案，该系统由终端传感网层、通信网络层、云服务层组成和好地解决了目前智能家居系统的中存在的扩展性、高并发等问题。

1.3 主要研究内容

本课题拟研究物联网技术在智能家居领域的应用，实现一种基于 RESTful 服务的智能家居物联网服务平台。

1.3.1 智能家居的总体结构

本设计研究的智能家居物联网系统由终端感知节点、控制节点、网关节点、服务平台、用户 App/Web 节点组成，可实现对智能家居的远程控制。系统总体结构如图 1.1 所示。

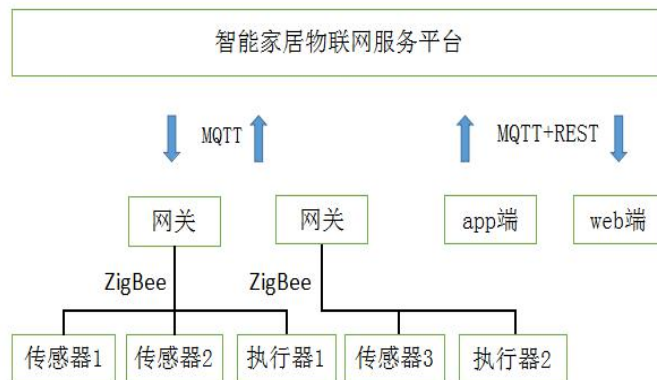


图 1.1 智能家居系统总体结构图

智能家居系统通过 ZigBee 技术组网将感知节点的数据发送到网关节点，经过处理后使用物联网主流协议 MQTT 发送至智能家居服务平台，通过平台推送给数据的订阅方（App/Web 端）。来自 App/Web 端的指令同样以 MQTT 协议发送至服务平台，然后将指令发送到网关节点，并下达到指定设备（控制节点）。

1.3.2 本文的研究内容

本文主要完成智能家居物联网服务平台设计，核心研究内容如下：

（1）应用层服务：智能家居物联网服务平台的应用层接口由 Web Service 接口和发布/订阅 MQTT 接口组成，研究 REST Web Service 服务框架和 MQTT 协议发布/订阅消息机制。

（2）感知层服务：智能家居物联网服务平台为感知层提供 MQTT 接口，研究 MQTT 协议发布订阅消息机制。

（3）海量数据存储：智能家居服务平台需要存储数以万计的智能家居设备提供产生的数据，实现持久化存储和时序存储，研究 MongoDB 持久化存储和 InfluxDB 时序存储数据的海量数据存储方案。

（4）高并发性：智能家居服务平台接入大量的设备，需要有很强的并发处理能力，研究 Nginx 服务器反向代理负载均衡解决方案。

1.4 论文结构

本文的正文共分为七章。

第一章：绪论，介绍本文选题背景和研究意义、国内外研究现状、主要研究内容。

第二章：相关技术介绍，包括设计的总体设计方案、具体应用的技术以及工具。

第三章：需求分析，结合当前智能家居系统用户的实际需求，本文从功能需求、性能需求以及外部接口需求等方面进行需求分析。

第四章：概要设计，论述了系统架构，在纵向上对系统进行分层，各层在横向进行模块划分，充分提高系统各模块的性能。

第五章：详细设计，给出系统各模块说明和系统类图及类间关系，以及系统重要模块实现细节。

第六章：系统部署和测试，介绍了系统的部署和测试环境，以及系统测试。

第七章：总结与展望，总结系统的总体实现情况以及给出系统后续应完善的部分。

2 相关技术介绍

本章介绍智能家居物联网服务平台相关技术，该平台一方面为感知层提供服务，采用 MQTT 协议与智能家居网关交互，另一方面为应用层提供服务，为用户提供 Web Service 服务接口和 MQTT 消息接口。

2.1 Web Service

Web Service 技术是跨平台和跨编程语言的远程调用技术，即使运行在不同的机器、不同的操作系统，无需借助任何专门或第三方软硬件就可以相互交换数据。接下来将从 Web Service 服务 API 风格、服务框架来阐述以及 Flask Web 框架。

2.1.1 REST 风格 Web 服务

REST（英文：Representational State Transfer，简称 REST）表述性状态传递，是 Roy Fielding 博士 2000 年的时候在他的论文中提出的一种软件架构风格。REST 通过统一的链接接口对相应资源进行操作，以 URI 来确定资源，充分地发挥了 HTTP 本身具备的分布式特性，将 HTTP 提供的四种基本方法（GET、POST、PUT 及 DELETE）分别对应资源的一种操作（查询、创建、修改及删除）。如图 2.1 所示。

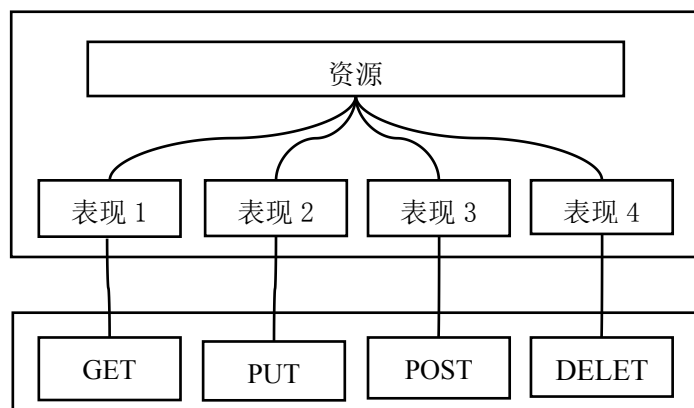


图 2.1 REST Web Service 资源方法对应关系

REST 也是一种针对网络应用的设计和开发方式，可以降低系统的耦合性，提高系统的可伸缩性，简化 API 开发。REST 风格 API 非常适合智能家居物联网服务平台多模块之间交换数据，降低模块之间耦合，保证了系统高度的可伸缩性。

2.1.2 Flask 框架

Flask 是一个使用 Python 编写的轻量级 Web 服务框架，WSGI（Web Server Gateway Interface）采用 Werkzeug，模板引擎则使用 Jinja2。Flask 被称为“微型架构”，使用简单的核心。通过 extension 来扩展很多其他功能，可以实现非常强大的 Web 应用。Python 是一种解释型的、面向对象的高级程序设计语言，Werkzeug 是 Python 的 WSGI 的实用函数库，Jinja2 是基于 Python 的模板引擎。

轻量级的 Flask Web 服务框架可以很容易构建微服务，再加上 Flask 众多的扩展，这为系统设计实现带来极大便利。所以，本文使用 Flask 设计系统的 REST Web service 接口。

2.2 物联网通信协议 MQTT

MQTT（Message Queuing Telemetry Transport，消息队列遥测传输）是一种轻量级的基于代理（broker）的发布/订阅（Publish/Subscribe）的消息协议。MQTT 协议由 IBM 推出，是为大量计算能力有限，且工作在低带宽、不可靠的网络的远程传感器和控制设备通讯而设计的协议，非常适合物联网的应用场景，如今已经成为物联网主流协议之一。

2.2.1 MQTT 消息传递机制

MQTT 协议的消息机制不同于以往的客户端/服务器模式，而是发布/订阅（pub/sub）模式。发布/订阅模式如图 2.2 所示。

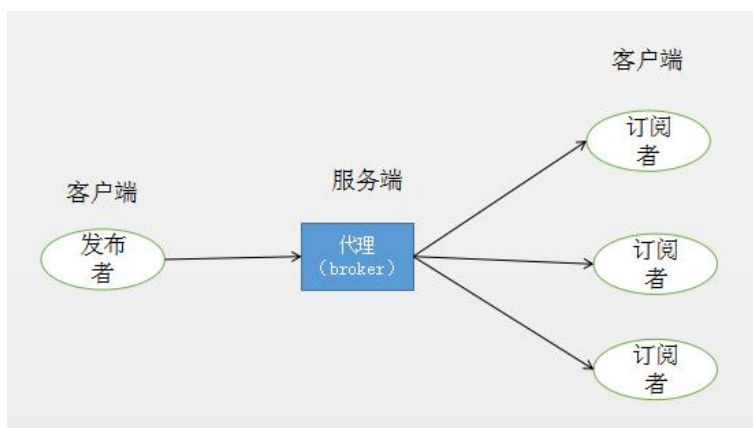


图 2.2 MQTT 协议发布/订阅模式

传统客户端/服务器模式是客户端直接连接到服务器，而发布/订阅模式需要第三个组件代理（broker），发布者和订阅者都需要连接到代理，发布者（Publisher）发布消息到特定主题（topic）经过代理过滤转发给订阅了相应主题（topic）的订阅者（Subscriber）。发布/订阅模式实现了客户端的解耦，比起传统客户端/服务

器模式有更好的扩展性。

2.2.2 HiveMQ 和 MQTT.fx

HiveMQ 是企业级 MQTT Broker，提供高性能、高可用、高扩展、高安全性的企业级服务。HiveMQ 基于事件驱动支持异步非阻塞 IO 模型、低延时高吞吐量、扩展性强。

MQTT.fx 是目前主流的 mqtt 客户端，可以快速验证是否可以与 IoT Hub 服务交流发布或订阅消息。

2.3 数据存储系统

智能家居物联网系统数据可分为两类：一类是用户信息、设备信息；另一类是设备产生数据和一些用户指令。用户信息和设备信息需要方便查询和更新，数据量大、支持横向扩展，因此这类数据使用数据实时性好、伸缩性强、表结构灵活的 MongoDB 存储。设备数据和用户命令的数据需要按时间排列，使用时序数据库 InfluxDB 存储。为了提高系统的并发访问性能，使用 Redis 作为缓存。

2.3.1 MongoDB

MongoDB 是一个基于分布式文件存储的 NoSQL 数据库，由 C++ 语言编写。旨在为 Web 应用提供可扩展的高性能数据存储解决方案。介于 SQL 与 NoSQL 之间，MongoDB 支持的数据结构非常松散，可用于对象及 JSON 数据的存储，可以存储大尺寸是数据。MongoDB 具有很好的数据实时操作性能，伸缩性好可分布式横向扩展。MongoDB 数据库基本概念如表 2.1 所示。

表 2.1 MongoDB 名词对照表

MongoDB	SQL 名词	说明
database	database	数据库
collection	table	集合/数据库表
document	row	文档/数据记录行
field	column	域/数据字段
index	index	索引
primary key	primary key	默认 _id 字段/主键

2.3.2 InfluxDB

InfluxDB 是一个开源的分布式时序、时间和指标数据库，使用 go 语言编写，

无需外部依赖。其设计目标是实现分布式和水平伸缩扩展。时序数据库全称时间序列数据库,时间序列数据库主要用于处理按时间变化的数据。InfluxDB 有三大特性:时序性,支持许多和时间相关的函数(可以求最大值、最最小、求和等多种运算);度量,对大量的数据实时计算;事件,支持任意的事件操作。同时,InfluxDB 有两种 api 访问方式:Protobuf API 和 HTTP API。

InfluxDB 数据库特有概念:

(1) tag--标签。在 InfluxDB 中,tag 是一个非常重要的部分,表名+tag 一起作为数据库的索引,是“key-value”的形式。

(2) field--数据。field 主要是用来存放数据的部分,也是“key-value”的形式。

(3) timestamp--时间戳。作为时序型数据库,时间戳是 InfluxDB 中最重要的部分,在插入数据时如果为空系统自动指定。

(4) series--序列。所有在数据库中的数据,都需要通过图表来展示,而这个 series 表示这个表里面的数据,可以在图表上几条线展示。

(5) Retention policy--数据保留策略。可以定义数据保留的时长,每个数据库可以有多个数据保留策略,但只能有一个默认策略。

(6) Point--点。表示每个表里某个时刻的某个条件下的一个 field 的数据,因为体现在图表上就是一个点,于是将其称为 point。

2.3.3 Redis

Redis 是一个开源的使用 ANSI C 语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Value 数据库,缓存和消息代理。它支持数据结构,如字符串,散列,列表,集合,具有范围查询的排序集,位图,超级日志和具有半径查询的地理空间索引。redis 会周期性地把数据写入磁盘存储,并且支持多种语言的客户端 Java, C/C++, JavaScript, Python, 使用很方便。最重要的是 redis 支持分布式主从部署,为系统的分布式部署带来极大便利。

2.4 开发工具

本项目开发中使用的开发工具有如下几种:

(1) 开发 IDE。PyCharm 是一个用于计算机编程的集成开发环境,由捷克公司 JetBrains 开发,可以帮助用户在使用 Python 语言开发时提高其效率的工具,借助 PyCharm 众多的功能,可以提高开发效率。

(2) 软件包管理工具。pip 是一个以 Python 计算机程序语言写成的软件包管理系统,提供了对 Python 包的查找、下载、安装、卸载的功能。

(3) 代码托管工具。gitHub 是一个面向开源及私有软件项目的托管平台,因为只支持 git 作为唯一的版本库格式进行托管,故名 gitHub。借助强大的代码托管功能,有利于提高开发效率。

3 需求分析

本章是对智能家居系统的需求分析，首先从用户需求出发分析了用户、租户、区域、设备管理等功能需求，然后分析了并发性、可靠性、时延等性能需求。

3.1 功能需求

智能家居系统一边要为感知层提供服务，另一边要为应用层提供服务。本节功能需求分析分别从感知层接口需求和应用层接口需求入手。

3.1.1 应用层接口需求

智能家服务平台需要为应用层提供服务，包括用应用层户订阅自己关心的设备的信息；设备归属不同的家庭成员（用户），这样就需要系统支持多用户参与，同时需要一个租户来授权用户；不同的家庭是不同的区域，每个区域可以放置多个设备；租户可以管理自己的区域。最后系统还需要一个管理员。

a. 应用层需求分析

经过需求分析，整个平台参与者有三种，分别是：租户、用户和系统管理员。各参与者的需求如下：

- (1) 用户可以自己注册、登录、注销。
- (2) 用户可以订阅自己区域（即家庭）内设备的信息信息。
- (3) 用户可以对设备进行控制。
- (4) 用户可以查看自己的历史操作记录。
- (5) 用户可以按时间查看指定设备的历史数据。
- (6) 租户可以自己注册、登录、注销。
- (7) 租户可创建自己的区域。
- (8) 租户可以授权给用户，来管理设备。
- (9) 租户可以为区域添加设备。
- (10) 平台管理员可以登录、注销。
- (11) 平台管理员可以管理现有的租户和区域。

b. 用例图

上面对智能家居物联网服务平台的用户需求分析，用例图如下。

(1) 用户。用户功能需求可为登录系统、设备管理（包括订阅设备所产生的数据、查询设备的历史数据、发出指令给特定设备、取消订阅）等。用户的用

例图如图 3.1 所示。

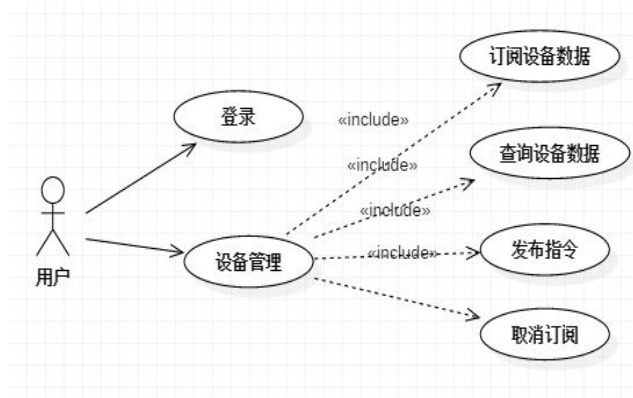


图 3.1 用户用例图

（2）租户。租户功能需要为登录系统、用户管理（包括创建用户、更新用户、查看用户、删除用户）、区域管理（创建区域、更新区域、查看区域信息、删除区域）等。用例图如图 3.2 所示。

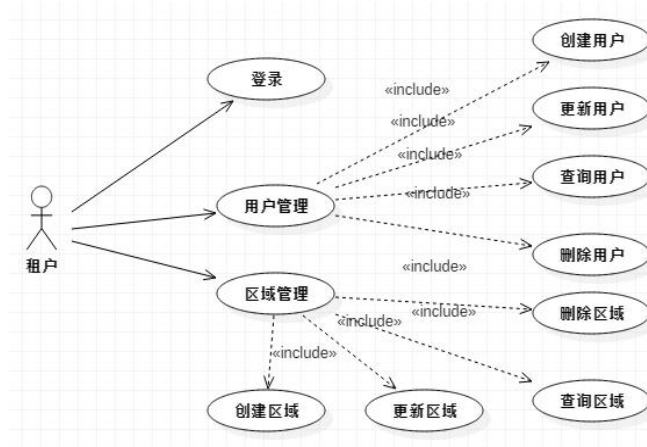


图 3.2 租户用例图

（3）平台管理员。平台管理功能需求为登录系统、站点管理（包括创建区域、更新区域、查看区域信息、删除区域）、租户管理（创建租户、更新租户、查看租户、删除租户）等。用例图如图 3.3 所示。

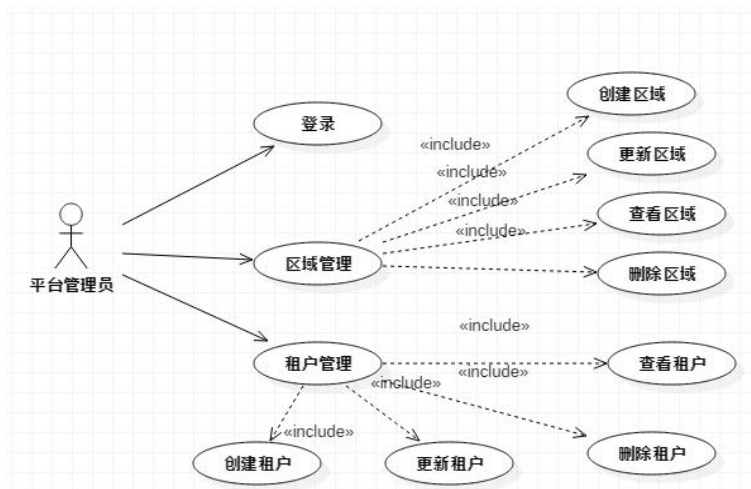


图 3.3 平台管理员用例图

3.1.2 感知层接口需求

智能家居服务平台的另一端连接着设备，需要把来自设备数据和来自应用前端是用户命令等事件进行处理后并进行持久化存储。设备侧的功能需求如下：

- （1）订阅。订阅来自设备设备数据和用户命令的 MQTT 消息。
- （2）发布。发布来自设备的数据和用户的命令到特定 MQTT 主题。
- （3）自注册。收到未注册设备的事件，先进行设备注册。
- （4）设备管理。添加设备到平台；注销故障设备。
- （5）设备事件处理。将设备产生的数据进行持久化存储，获取当前时间指定设备数据。
- （6）用户事件存储。将用户发出的命令进行持久化存储，或当前时间指定设备用户命令。

以上设备侧的功能需求，转化成用例图如图 3.4 所示。

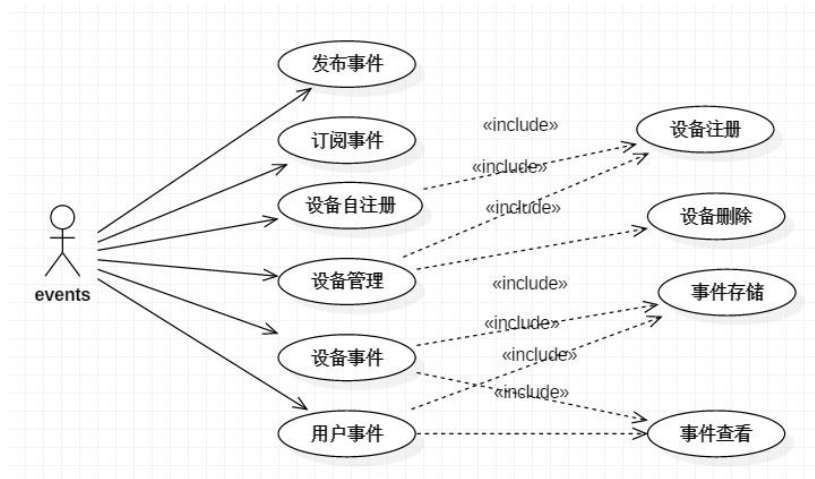


图 3.4 设备侧用例图

3.2 性能需求

本系统的设计目标是要为大量的用户和设备提供及时可靠的服务，这就需要系统能应对很高并发，为数万甚至数十万设备提供服务，由于用户实时控制设备，系统延时应尽可能小。

- （1）高并发。系统要为大量的用户和设备提供服务，并发量必须大于 1k 以上的 qps。
- （2）可靠性。系统可以长期稳定运行，提供服务可靠，不出现查询出错等异常情况。
- （3）延时低。系统延时必须小于 100ms，保证服务质量。

4 系统概要设计

经过以上需求分析，本章进行系统概要设计。主要进行应用层服务接口设计、感知层消息处理、数据库设计，最后进行系统架构设计。

4.1 应用层服务接口

在移动互联网开发领域，几乎所有的互联网系统都采用接口形式提供数据服务。物联网系统开发中，我们同样需要为移动设备提供数据访问接口，而且在开发之初就需要预先设计正确的、良好的、易于扩展的接口。接口设计的优劣关系着系统的性能以及后期开发的难以程度，因此接口设计在系统设计中举足轻重。根据需求分析，智能家居物联网服务平台的接口设计包含两部分用户接口和设备接口。

4.1.1 用户接口

REST 的核心思想是把接口按照逻辑上的资源进行划分，这些资源将按照不同的 HTTP 方法进行请求，不同的方法对应不同的含义。以下是智能家居物联网服务平台的接口设计表。

a.区域服务接口。区域服务接口表如下表 4.1 所示。

表 4.1 区域服务接口表

方法	路径	描述
GET	/sites	查看所有区域信息
GET	/sites/{siteToken}	按照站点 token 查看区域
POST	/sites	创建一个区域
PUT	/sites	更新一个区域
DELETE	/sites/{siteToken}	删除指定区域

b.租户服务接口。租户服务接口表如表 4.2 所示。

表 4.2 租户服务接口表

方法	路径	描述
GET	/tenants	查看所有租户信息
GET	/tenants/{tenantId}	按照租户 ID 查看租户
POST	/tenants	创建一个租户
PUT	/tenants	更新一个租户
DELETE	/tenants/{tenantId}	删除指定租户

c. 用户服务接口。用户服务接口表如表 4.3 所示。

表 4.3 用户服务接口表

方法	路径	描述
GET	/users	查看所有用户信息
GET	/users/{username}	按照用户名查看用户
POST	/users	创建一个用户
PUT	/users	更新一个用户
DELETE	/users/{username}	删除指定用户

4.1.2 设备接口

设备的 Web Service 接口，包括设备服务接口和设备事件接口。设备服务接口包括对设备的多种查看、创建、修改以及删除等接口。设备事件接口包括设备数据的存储和查看接口，以及用户命令的存储和查看接口。

a. 设备服务接口。设备服务接口如表 4.4 所示。

表 4.4 设备服务接口表

方法	路径	描述
GET	/devices	查看所有设备信息
GET	/devices/{hardwareId}	按照设备 ID 查看设备
GET	/devices?Type=siteToken	按照站点 token 查看设备
POST	/devices	注册一个设备
PUT	/devices	更新一个设备
DELETE	/devices/{hardwareId}	删除指定设备

b. 设备事件接口。设备事件接口如表 4.5 所示。

表 4.5 设备事件接口表

方法	路径	描述
GET	/devices/{hardwareId}/events/{etype}	查看设备历史记录
POST	/devices/{hardwareId}/events/{etype}	设备数据存储

为方便支持不同协议设备的接入，对以上事件接口进行封装，实现了 MQTT 协议的接口，经过 mqtt breker 代理来连接设备和平台。设备的 MQTT 接口包括 MQTT 事件订阅和发布两个接口。

c. 设备 MQTT 接口。设备 MQTT 接口如表 4.6 所示。

表 4.6 设备 MQTT 接口表

主题	格式	描述
/iot/input/json	json: 如 {"led": "false"}	设备数据接入
/iot/output/json	json: 如 {"led": "false"}	设备数据发送

4.2 感知层消息接口

智能家居物联网服务平台存在设备和用户两方面的接口，由于这两类接口收到的消息有不同的特点，所有处理的流程也不一样。设备接口消息并发量大、实时性要求高；用户接口消息量较小，稳定性要求高。接下来对这两种消息处理方式进行说明。

a. 用户消息。用户接口消息稳定性要求较高，调用相应的 Web Service 进行处理和存储。系统订阅“/iot/input/json”主题上的消息，当用户接口消息到来时，首先判断区域 token 是否存在，如果存在，则把命令发布到指定主题（“/iot/output/json”），然后调用用户命令存储接口进行持久化存储，最后返回结果；否则，直接忽略。用户命令处理流程如图 4.1 所示。

b. 设备消息。考虑到设备接口消息并发量大、实时性要求高的特点，引入了 Redis 缓存机制。系统订阅“/iot/input/json”主题上的消息，当有设备数据消息到达时，判断设备是否在系统中注册，如果没有注册，则调用设备注册接口注册设备，然后把设备消息内容写入 Redis 缓存，接着报消息发布到特定主题“/iot/output/json”，然后调用设备数据存储接口进行持久化存储，最后返回结果。设备数据消息处理流程如图 4.2 所示。

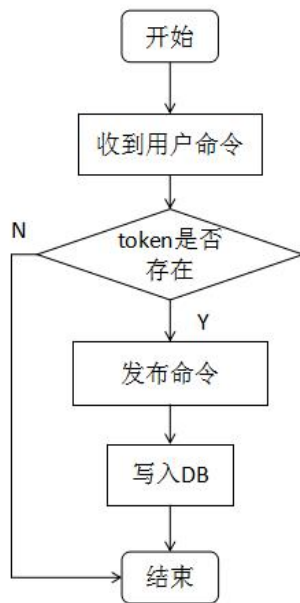


图 4.1 用户命令处理流程

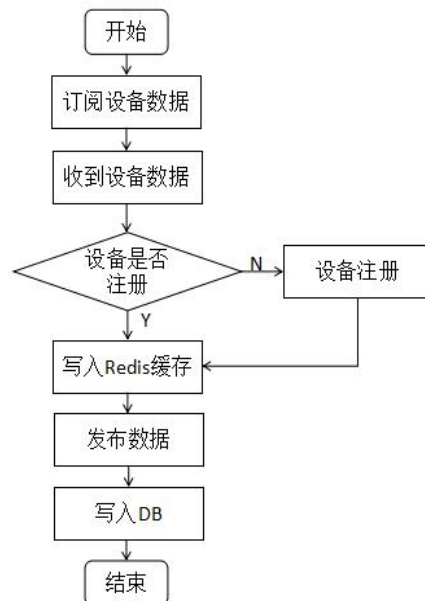


图 4.2 设备消息处理流程

4.3 数据库设计

系统选取 MongoDB 和 InfluxDB 来应对物联网海量数据和扩展性等问题，设备管理和系统管理产生的数据存储存储在 MongoDB 中，而事件数据存储存储在 InfluxDB

中。根据前面对系统需求的分析，智能家居物联网服务平台数据库表包括租户、站点、用户、设备、设备数据、用户命令。以下是数据库详细设计。

4.3.1 概念模型

概念模型是数据库逻辑结构即逻辑模型的基础。在以上需求分析和系统设计的基础上，由于实体的属性较多，图中省略了实体的属性，详细的实体关系 E-R 图，如图 4.3 所示。

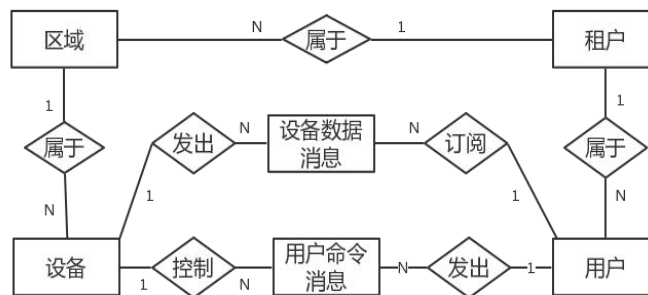


图 4.3 数据库 E-R 图

4.3.2 逻辑模型

经过以上概念模型的设计，可以将设计好的概念模型转化成某种数据库系统支持的逻辑模型。针对不同数据库系统我们做了如下设计。MongoDB 数据库设计四个集合（collection）分别用来存储租户、用户、区域、设备的信息。InfluxDB 数据库设计两个测量（measurement）用来存储设备数据和用户命令。详细设计描述如下：

（1）区域基本信息表。创建区域时，将填充的区域基本信息（如区域名、数据、描述信息等）保存到 MongoDB 数据库的站点基本信息集合中，具体设计如表 4.7 所示。

表 4.7 站点基本信息表

字段	字段名称	允许空	备注
token	区域 token	否	主键
name	区域名	否	
description	区域描述信息	是	
createdBy	创建者	是	
createdDate	创建时间	是	
metadata	数据域	是	
ext	扩展字段	是	

（2）租户基本信息表。用于存储租户的基本信息，在管理员创建租户时填充。具体设计如表 4.8 所示。

表 4.8 租户基本信息表

字段	字段名称	允许空	备注
id	租户 ID	否	主键
name	租户名	否	主键
authenticationToken	区域 token	否	
authorizedUserIds	用户组	是	
createdBy	创建者	是	
createdDate	创建时间	是	
metadata	数据域	是	
ext	扩展字段	是	

（3）设备基本信息表。主要用来存储设备被创建时的信息，详细设计如表 4.9 所示。

表 4.9 设备基本信息表

字段	字段名称	允许空	备注
hardwareId	设备 ID	否	主键
siteToken	区域 token	否	
comments	描述	是	
createdBy	创建者	是	
createdDate	创建时间	是	
metadata	数据域	是	
ext	扩展字段	是	

（4）用户基本信息表。用于存储用户的基本信息（如用户名、密码 hash 值、最近登录时间、创建者、创建时间等）。详细设计如表 4.10 所示。

表 4.10 用户基本信息表

字段	字段名称	允许空	备注
username	用户名	否	主键
hashedPassword	密码 hash 值	否	
lastLogin	最近登录时间	是	
status	用户状态	是	
createdBy	创建者	是	
createdDate	创建时间	是	
metadata	数据域	是	
ext	扩展字段	是	

（5）设备数据消息表。用于存储设备产生的数据，基本信息包括时间戳、

设备 ID、区域 token 等。详细设计如表 4.11 所示。

表 4.11 设备数据表

字段	字段名称	允许空	备注
time	时间戳	否	主键
hardwareId	设备 ID	否	
siteToken	区域 token	否	
eventDate	事件日期	是	
receivedDate	接收日期	是	
metadata	数据域	否	
ext	扩展字段	是	

（6）用户命令消息表。用于存来自用户端数据，基本信息包括时间戳、用户名、设备 ID、区域 token 等。详细设计如表 4.12 所示。

表 4.12 用户命令表

字段	字段名称	允许空	备注
time	时间戳	否	主键
hardwareId	设备 ID	否	
username	用户名	否	
siteToken	区域 token	否	
eventDate	事件日期	是	
receivedDate	接收日期	是	
metadata	数据域	否	
ext	扩展字段	是	

4.3.3 BSON 数据存储格式

MongoDB 采用 BSON（Binary JSON）一种类 json 的一种二进制形式的存储格式。BSON 经常作为网络数据交换的一种存储形式，其特点是高效性、轻量性、可遍历性。以下是用户、租户、区域和设备在 MongoDB 中存储形式：

（1）用户 BSON 的数据格式

```
{  "_id" : ObjectId("5ad9e506c4f2a05fdfdad221"),
  "metadata" : {  },
  "hashedPassword" : "1234561",
  "createdBy" : "admin",
  "username" : "atm",
  "createdDate" : "2018-05-09 18:44:37",
  "lastLogin" : "2018-05-09 18:44:37",
  "status" : true
}
```

（2）租户 BSON 的数据格式

```
{  "_id" : ObjectId("5ad9e53fc4f2a05fdfdad222"),
  "id" : "test",
  "name" : "test tenant",
  "metadata" : {  },
  "authenticationToken" : "123",
  "createdBy" : "admin",
  "authorizedUserIds" : [ "admin" ],
  "createdAt" : "2018-05-09 18:45:55"
}
```

（3）区域 BSON 的数据格式

```
{  "_id" : ObjectId("5af83ae5c4f2a06ff84ad0df"),
  "createdAt" : "2018-05-14 21:37:47",
  "createdBy" : "admin",
  "ext" : "1",
  "token" : "fa4c855c-56af-11e8-846f-00163e2e5c59",
  "name" : "卧室",
  "description" : "我的卧室",
  "metadata" : {
    "shidu" : "20",
    "wendu" : "20",
    "num" : "1"
  }
}
```

（4）设备 BSON 的数据格式

```
{  "_id" : ObjectId("5af90e99c4f2a06ff84ad0e1"),
  "createdAt" : "2018-05-16 15:38:36",
  "createdBy" : "admin",
  "hardwareId" : "Door001",
  "siteToken" : "fa4c855c-56af-11e8-846f-00163e2e5c59",
  "ext" : 1,
  "comments" : "",
  "metadata" : {
    "status" : 1,
    "name" : "住宅 1 号门",
    "location" : "住宅 1 号门"
  }
}
```

4.4 软件架构设计

智能家居物联网服务平台软件架构分层设计，包括数据存储系统、数据访问层、接口访问层、设备适配层。智能家居物联网服务平台的体系架构如图 4.4 所示。

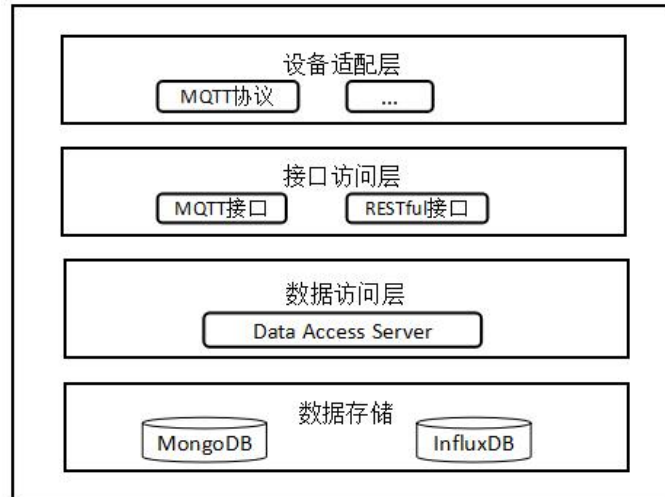


图 4.4 系统体架构图

数据存储系统，实现设备数据的持久化存储，利用 MongoDB 数据和 InfluxDB 实现对数据的持久化存储。

数据访问层位于数据存储系统和接口访问层之间，利用底层数据存储系统的数据服务数据访问层是通过服务接口模块实现的。服务接口模块主要实现系统数据访问层的接口，包括区域、租户、用户、设备和事件的内部接口。这些接口是整个系统服务的核心，负责为下层模块提供数据访问服务。服务接口模块的服务仅供系统内部模块调用。

接口访问层位于数据访问层和设备适配层之间，利用上层数据访问层服务接口模块的服务接口进行数据处理和转发，并对外提供应用程序编程接口。本层由三个模块组成，分别是事件管理模块、MQTT 事件模块、综合管理模块。事件管理模块包括对设备数据和用户命令的处理和转发功能；MQTT 事件模块订阅 MQTT broker 的消息根据消息内容调用事件管理模块接口；综合管理模块实现对上层的站点、租户、用户、设备等服务接口的封装对外提供服务。

设备适配层位于接口访问层与智能网关设备之间，利用上层接口为网关设备提供服务。设备接入层主要解决支持不同物联网协议的设备接入问题，本系统以物联网标准协议 MQTT 协议作为实例进行探索。设备适配层包含一个 MQTT broker，该层实现了 MQTT 协议设备接入智能家居服务平台。

5 详细设计

本章是系统的详细设计，主要介绍了系统各层的模块的类设计、核心类的实现、数据交换的格式，最后针对系统的性能进一步优化。

5.1 系统模块设计

智能家居物联网服务平台主要由服务接口模块、综合管理模块、事件处理模块、设备接入模块等模块组成。

5.1.1 服务接口模块

服务接口模块设计了 5 个类，分别是 site（区域）、tenant（租户）、user、device、event（事件）。区域：区域是对智能家居系统中家庭区域的抽象，一个房间多个房间或一个家庭都可以抽象成站点供平台管理。区域在创建时系统会为其分配系统唯一的 token，供设备和租户认证使用。租户：租户充当系统的管理者，并且维护一个用户列表。在智能家居实际案例中租户是区域的管理员账户，用来管理下辖的用户。用户：用户是指智能家居系统的实际操作人员，归属到相应的租户。用户可以查看订阅的设备详细数据，也可以向设备下达操作指令。设备：设备是对实际设备的映射，以便将设备状态在系统中表示和直观展现给用户。事件：事件是指设备数据、用户指令、报警信息等多种事件。

(1) 系统类图设计。系统类图设计如图 5.1 所示。

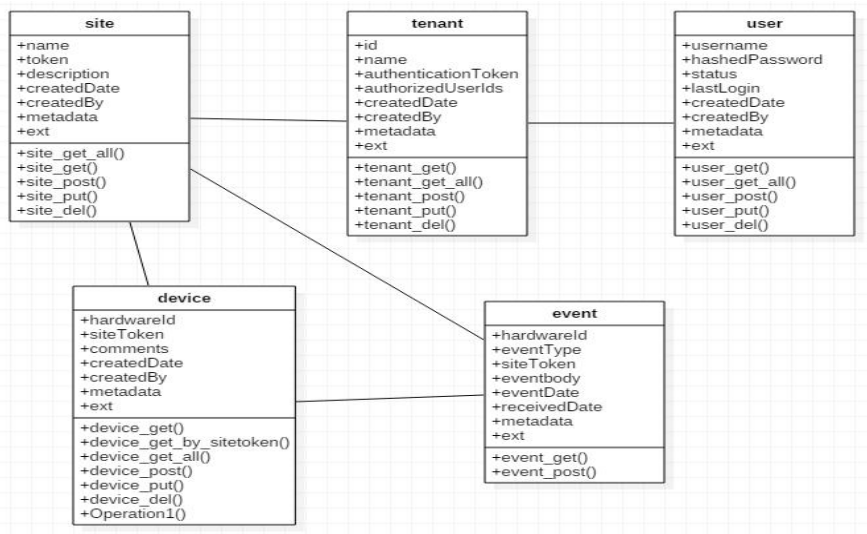


图 5.1 类图详细设计

（2）模块实现。这里介绍以下服务接口模块的实现，模块使用 Flask web 服务框架实现了各个类的服务接口，首先连接到数据存储系统，然后通过 Flask 的路由调用类的方法。以下是模块的核心代码片段。

```
@app.route('/iot/spi/<string:cate>/<string:cateid>',methods=['GET','DELETE'])
```

#实现/iot/spi/<string:cate>/<string:cateid>接口的 GET 和 POST 方法

```
def api_cate_id(cate,cateid):
    if request.method == 'GET':
        if cate == 'sites':
            #todo: 按照 token 查询区域
            res = sites.site_get(mongo,cateid)
            ...
        else:
            res = jsonify({'result':None,'code':403})
            return res
    elif request.method == 'DELETE':
        if cate == 'sites':
            #todo: 区域删除
            res = sites.site_del(mongo,cateid)
            ...
        else:
            res = jsonify({'result': None, 'code': 403})
            return res
```

#按照 token 查询站点

```
def site_get(mongo,token):
    sites = mongo.db.sites
    ex = json.loads(exp)
    res = sites.find_one({"token" : token})
    if res == None:
        #查询失败或站点不存在,code:404
        return jsonify({'result': ex,'code':404})
    else:
        #查询成功，构造响应,code:200
        res.pop("_id")
        print(res)
        return jsonify({'result':res,'code':200})
```

5.1.2 事件管理模块

事件管理模块提供的事件的接口，供 app 端和设备 MQTT 接入端使用。事件管理模块对到来的消息首先使用 Redis 缓存，然后在调用事件接口。

（1）路由设计。事件管理模块的接口路由如下。

/iot/api/devices/<string:hardwareId>/events/<string:etype>

其中<string:hardwareId>和<string:etype>字段分别是设备 ID 和事件类型。不同的设备 ID 有不同的事件类型（DevicesData、UserCommands），每个类型的事件有 GET 和 POST 之分。事件管理模块接口表与服务接口模块相似，如表 5.1 所示。

表 5.1 事件管理模块路由选择表

事件类型\HTTP 方法	GET	POST	其他
DevicesData	data_post	data_post	{'code': 403}
UserCommands	commands_get	commands_post	{'code': 403}
其他	{'code': 403}	{'code': 403}	{'code': 403}

事件管理模块接口的路由实现如下。

```
@app.route('/iot/api/devices/<string:hardwareId>/events/<string:etype>',
methods=['GET','POST'])
```

```
def api_events_get(hardwareId,etype):
```

```
    if request.method == 'GET':
```

```
        #设备数据 GET 请求
```

```
        if etype == "DevicesData":
```

```
            res = data.data_get_process(hardwareId)
```

```
        #用户命令 GET 请求
```

```
        elif etype == "UserCommands":
```

```
            res = commands.commands_get_process(hardwareId)
```

```
        else:
```

```
            res = jsonify({'result': None, 'code': 403})
```

```
    return res
```

（2）用户命令和设备数据处理。事件管理模块把事件分为设备数据和用户命令。当接口调用端推送数据，首先把数据加入 redis 缓存，然后把消息内容发布到 MQTT 代理，最后通过事件服务接口进行处理处理后返回结果；当接口调用端请求数据，首先查看 redis 是否有缓存，从缓存中返回结果；否则，通过事件服务接口获取，最后返回结果。

（3）核心代码。事件管理模块核心代码如下。

```
#设备数据 post 处理方法
```

```
def data_post_process(hardwareId,data):
```

```
    log.logger.info("call : data_post_process()")
```

```
    #连接 redis 池
```

```

rpool = rediser.redis_pool
#redis 缓存
rpool.set(hardwareId+'data', data)
#mqtt 发布
epublish.data_publish(data)
res = requests.post(urlt + hardwareId + "/events/DevicesData",
request.get_data())
log.logger.info(res.text)
res = res.json()
return jsonify(res)
#设备数据 get 处理方法
def data_get_process(hardwareId):
    log.logger.info("call : data_get_process()")
    #连接 redis 池
    rpool = rediser.redis_pool
    res = rpool.get(hardwareId+'data')
    res = None
    if res != None:
        #返回从 redis 获取的数据
        res = res.decode('utf-8')
        return jsonify({'result': res,'code':200})
    else:
        #redis 获取失败，从上次服务获取
        res = requests.get(urlt+hardwareId + "/events/DevicesData")
        res = res.json()
        return jsonify(res)

```

5.1.3 综合管理模块

（1）路由设计。综合管理模块的路由设计如下。

/iot/api/<string:cate>

/iot/api/<string:cate>/<string:cateid>

其中<string:cate>和<string:cateid>字段分别是类别和类别对应的 ID。

（2）核心代码。综合管理模块功能比较简单，模块核心代码如下。

#租户、区域、设备、用户管理接口

@app.route('/iot/api/<string:cate>',methods=['GET','POST','PUT'])

def api_cate(cate):

log.logger.info("call : api_cate(cate)")

```

if request.method == 'GET':
    res = requests.get(urlt + cate)
    res = res.json()
    return jsonify(res)
elif request.method == 'POST':
    data = json.loads(request.get_data().decode('utf-8'))
    res = requests.post(urlt + cate, request.get_data())
    res = res.json()
    return jsonify(res)

```

5.1.4 MQTT 事件模块

设备接入模块订阅“/iot/input/json”主题上的 json 格式消息，消息到来后根据消息类型，选择不不同的消息接口进行处理。这里对 paho.mqtt.client 的 MQTT 客户端进行封装，当订阅的消息到来时会调用 MyMQTTClass 类 on_message() 回调函数，然后解析消息类型，如果消息类型是“DevicesData”调 devaccess 类的 device_data()，如果是“UserCommands”调用 devaccess 类的 user_command()。如果设备没有注册，这里调用 devaccess 类的 register_device() 接口进行设备注册，然后在调用设备数据接口。

(1) MQTT Client 类图。设备接入模对 paho.mqtt.client 的 Client 进行封装，类图如图 5.2 所示。

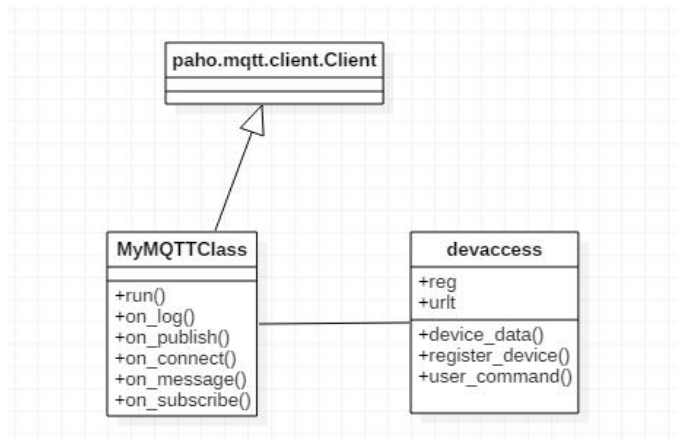


图 5.2 MyMQTTClass 类图

(2) 核心代码。设备接入事件处理核心代码。

#设备数据处理

```

def device_data(data):
    log.logger.info("call : device_data(data)")
    res = requests.post(urlt + data["hardwareId"]
        + "/events/DevicesData", json.dumps(data)).json()

```



```

print(res)
if res["result"]["hardwareId"] == ":
    #注册设备
    register_device(data)
    requests.post(urlt + data["hardwareId"]
                  + "/events/DevicesData", json.dumps(data)).json()
#用户命令处理
def user_command(data):
    log.logger.info("call : user_command(data)")
    log.logger.info(data)
    requests.post(urlt+ data["hardwareId"] + "/events/UserCommands",
                  json.dumps(data))

```

5.2 系统性能优化

Flask 默认的服务器采用的是一个同步阻塞的 web 框架，在应对物联网海量数据压力时显然无法胜任。这里我们采用 gunicorn 来提升性能。总的优化方案如下：

- （1）服务器模块优化：使用异步多线程 gunicorn 提高服务并发性能。
- （2）前端优化：使用 Nginx 负载均衡提高服务平台整体并发性能。

使用 gunicorn 提升各个 Flask 服务模块性能，之后采用 nginx 进行负载均衡实现系统的横向扩展。整体设计如图 5.3 所示。

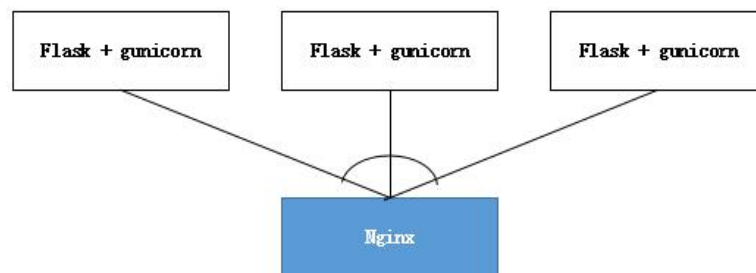


图 5.3 系统性能优化设计图

5.2.1 使用 gunicorn 优化

gunicorn 用于单个服务节点的优化，安装方式：pip install gunicorn（python3.x 版本使用 pip3 install gunicorn）。

本文使用 Gunicorn 提高 Flask 服务性能，在程序中加入如下粗体的代码即可。

```

if __name__ == '__main__':

```

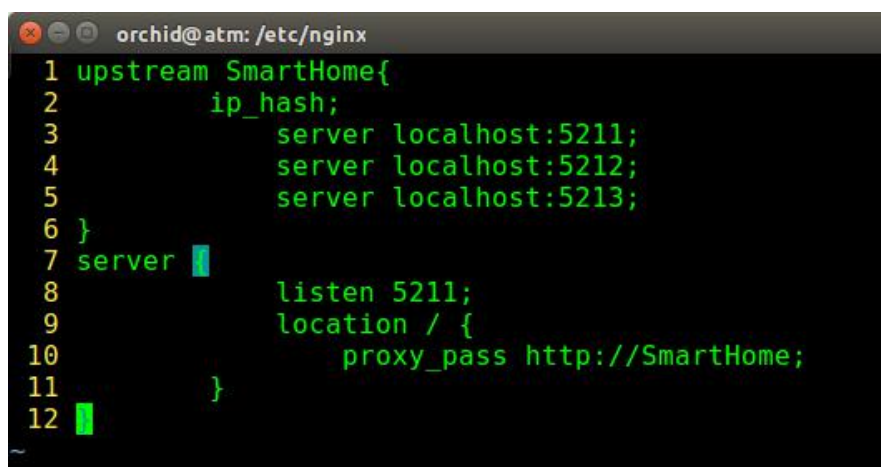
```
from werkzeug.contrib.fixers import ProxyFix
app.wsgi_app = ProxyFix(app.wsgi_app)
app.run()
```

启动多线程支持高并发，提高系统的性能

>\$ gunicorn -w 4 -b 127.0.0.1:5120 文件名(不带后缀):app

5.2.2 使用 Nginx 优化

Nginx 反向代理可以负载均衡，把客户端的请求转发给后面的服务器。这里配置了一个 Nginx 的反向代理来做负载均衡。nginx 的配置文件如图 5.4 所示。



```
orchid@atm: /etc/nginx
1 upstream SmartHome{
2     ip_hash;
3     server localhost:5211;
4     server localhost:5212;
5     server localhost:5213;
6 }
7 server {
8     listen 5211;
9     location / {
10         proxy_pass http://SmartHome;
11     }
12 }
```

图 5.4 Nginx 负载均衡配置

6 系统部署和测试

本章主要介绍智能家居物联网服务平台的部署和系统测试。

6.1 系统部署

6.1.1 系统配置信息和端口分配

智能家居物联网服务平台最终部署在阿里云平台，部署环境详细信息如下：

操作系统：Ubuntu 16.04.3 LTS (x86_64 GNU/Linux)

开发语言：python v=3.5

数据库：MongoDB v=3.6.0；InfluxDB v=1.2.4；Redis v=4.0.7

Web 框架：Flask v=0.12.2

为了方便管理和维护，系统各个模块端口使用情况定义如表 6.1 所示。

表 6.1 智能家居物联网服务平台端口使用表

归属层	模块	端口
数据存储	MongoDB	27017
	InfluxDB	8086
数据访问层	服务接口模块	5120
接口访问层	事件处理模块	5211
	综合管理模块	5122
设备适配层	设备接入模块-MQTT 代理	1883

6.1.2 部署说明

首先是系统环境的初始化，包括数据库和 MQTT 代理的启动等步骤。然后是系统的部署和启动。

a. 系统环境初始化

系统环境舒适化详细操作如下：

(1) 启动 MongoDB 数据库

```
$ sudo service mongod restart
```

(2) 启动 InfluxDB 数据库

```
$ sudo influxd run &
```

(3) 启动 Redis

```
$ redis-server &
```

(4) 启动 HiveMQ MQTT broker

```
$ ./run.sh &
```

b. 系统的部署

(1) 服务接口模块启动:

```
$ gunicorn -w 2 -b 0.0.0.0:5120 run-center:app
```

(2) 事件处理模块启动:

```
$ gunicorn -w 2 -b 0.0.0.0:5211 run-process:app
```

(3) 综合管理模块启动:

```
$ gunicorn -w 2 -b 0.0.0.0.1:5122 run-admin:app
```

(4) 数据接入模块启动:

```
$ python3.5 run-devacc.py
```

(5) 关闭系统使用如下命令:

```
$ ps -ef|grep run-|grep -v grep|awk '{print "kill -9 "$2}'|sh
```

a. 系统启动

为了方便部署,减轻部署过程的复杂性,提供了一下几个脚本: start-env.sh 用于部署环境启动数据库; start.sh 脚本用于启动各模块的服务; stop.sh 脚本用于关闭系统服务。具体的使用方法如下:

(1) 启动部署环境和数据库:

```
$ sudo sh start-env.sh
```

(2) 启动系统服务:

```
$ sudo sh start.sh
```

(3) 关闭系统服务:

```
$ sudo sh stop.sh
```

6.2 系统测试

对于智能家居物联网服务平台来说,系统的可扩展性、可靠性、安全性是几个非常重要的因素。刚开发的系统肯定会有各种各样的问题,需要通过严格的测试,才能成为成品系统。本章将介绍系统测试相关内容,包括系统测试环境、接口测试、系统测试、测试结论。

6.2.1 测试环境

处理器: 英特尔 第三代酷睿 i5-3337U

内存: 4G

硬盘: 500G

操作系统：Ubuntu 16.04.3 LTS (x86_64 GNU/Linux)

数据库：MongoDB 3.6.0; InfluxDB

浏览器：Chrome v=66.0.3359.117

6.2.2 单元测试

本小节对系统的各个模块进行单元测试。单元测试是一段自动化的代码，这段代码调用被测试的工作单元，之后对这个单元的单个最终结果的某些假设进行检验。

针对各个模块的功能，这里使用 `unittest` 进行单元测试。这里以服务接口模块的单元测试为例进行说明，以下是服务接口模块的单元测试代码片段：

```
import unittest
import requests

class device_SPI_test(unittest.TestCase):
    def setUp(self):
        self.base_url = "http://127.0.0.1:5120/iot/spi/devices"

    #创建设备接口测试
    def test_a_device_post(self):
        data={'hardwareId':"unit-test","siteToken":"","comments":"","metadata":{},"ext":{}}
        res = requests.post(self.base_url,data)
        res = res.json()
        self.assertEqual(res['code'],200)
        print (res)

    # 查看设备列表接口测试
    def test_b_device_get_list(self):
        res = requests.get(self.base_url+"?type=all").json()
        self.assertEqual(res['code'], 200)
        print (res)

if __name__ == '__main__':
    unittest.main()
```

在以上服务接口模块的测试用例基础上，还进行了事件管理模块、综合管理模块、设备接入模块的单元测试。测试结果如图 6.1 所示。



图 6.1 单元测试结果

6.2.3 集成测试

前边进行了完整的单元测试，接下来进行系统的集成测试。智能家居物联网服务平台负责对外提供 RESTful API 服务接口，接口准确性和稳定性直接影响服务的质量，这里对智能家居物联网服务平台进行外部接口测试和压力测试。

智能家居物联网服务平台的接口使用命令行工具 curl 进行手工测试，。以下是测试详细流程：

a. 用户接口测试

(1) 查询所有用户

```
curl -X GET http://127.0.0.1:5122/iot/api/users
```

(2) 查询用户

```
curl -X GET http://127.0.0.1:5122/iot/api/users/atm001
```

(3) 创建用户

```
curl -X POST http://127.0.0.1:5122/iot/api/users
```

(4) 更新用户信息

```
curl -X PUT http://127.0.0.1:5122/iot/api/users
```

```
-d '{"username":"atm001","hashedPassword":"123456","metadata":{"test":"123"},"ext":{}}'
```

(5) 删除用户

```
curl -X DELETE http://127.0.0.1:5122/iot/api/users/atm001
```

(6) 异常情况返回

```
{
  "code": 40x,
  "result": {
```

```

        "createdBy": "",
        "createdDate": "",
        "ext": {},
        "hashedPassword": "",
        "lastLogin": "",
        "metadata": {},
        "status": "",
        "username": ""
    }
}

```

b. 租户接口测试

(1) 查询所有租户

```
curl -X GET http://127.0.0.1:5122/iot/api/tenants
```

(2) 查询租户

```
curl -X GET http://127.0.0.1:5122/iot/api/tenants/test
```

(3) 创建租户

```
curl -X POST http://127.0.0.1:5122/iot/api/tenants
```

```
-d '{"id": "test1", "name": "testtenant", "authenticationToken": "", "authorizedUserIds": ["admin"], "metadata": {}, "ext": {}}'
```

(4) 更新租户信息

```
curl -X PUT http://127.0.0.1:5122/iot/api/tenants
```

```
-d '{"id": "test1", "name": "testtenant", "authenticationToken": "", "authorizedUserIds": ["admin"], "metadata": {}, "ext": {}}'
```

(5) 删除租户

```
curl -X DELETE http://127.0.0.1:5122/iot/api/tenants/test1
```

(6) 异常返回

```

{
    "code": 40x,
    "result": {
        "authenticationToken": "",
        "authorizedUserIds": [],
        "createdBy": "",
        "createdDate": "",
        "ext": {},
        "id": "",
        "metadata": {},
        "name": ""
    }
}

```

}

c. 区域接口测试

(1) 查询所有区域

```
curl -X GET http://127.0.0.1:5122/iot/api/sites
```

(2) 查询特定区域

```
curl -X GET http://127.0.0.1:5122/iot/api/sites/
7b0db932-43c2-11e8-a7b2-705ab6ad9cae
```

(3) 创建一个新区域

```
curl -X POST http://127.0.0.1:5122/iot/api/sites
-d'{"token":"","name":"test","description":"testsite","metadata"
:
{ }, "ext": { } }'
```

(4) 更新一个区域

```
curl -X PUT http://127.0.0.1:5122/iot/api/sites
-d'{"token":"714567ca-6888-11e8-846f-00163e2e5c59","name":"test","descr
iption":"testsite","metadata": { }, "ext": { } }'
```

(5) 删除一个区域

```
curl -X DELETE
http://127.0.0.1:5122/iot/api/sites/714567ca-6888-11e8-846f-00163e2e5c59
```

(6) 异常返回结果

```
{
  "code": 40X,
  "result": {
    "createdBy": "",
    "createdDate": "",
    "description": "",
    "ext": { },
    "metadata": { },
    "name": "",
    "token": ""
  }
}
```

d. 设备管理接口测试

(1) 查询所设备

```
curl -X GET http://127.0.0.1:5122/iot/api/devices?type=all
```

(2) 查询设备

```
curl -X GET
http://127.0.0.1:5122/iot/api/devices/test1234560z
```

(3) 创建设备


```
curl -X POST http://127.0.0.1:5122/iot/api/devices
-d'{"hardwareId":"test123456","siteToken":"","comments":"","metadata":{},"ext":{}}'
```

(4) 更新设备信息

```
curl -X PUT http://127.0.0.1:5122/iot/api/devices
-d'{"hardwareId":"test123456","siteToken":"","comments":"","metadata":{},"ext":{}}'
```

(5) 删除设备

```
curl -X DELETE
http://127.0.0.1:5122/iot/api/devices/test123456
```

a. 设备 MQTT 接口测试

这里使用 MQTT.fx 对系统设备消息发布/订阅接口进行测试，并且在订阅接口订阅“/iot/output/json”主题消息，收到相应的测试数据如图 6.2 所示。

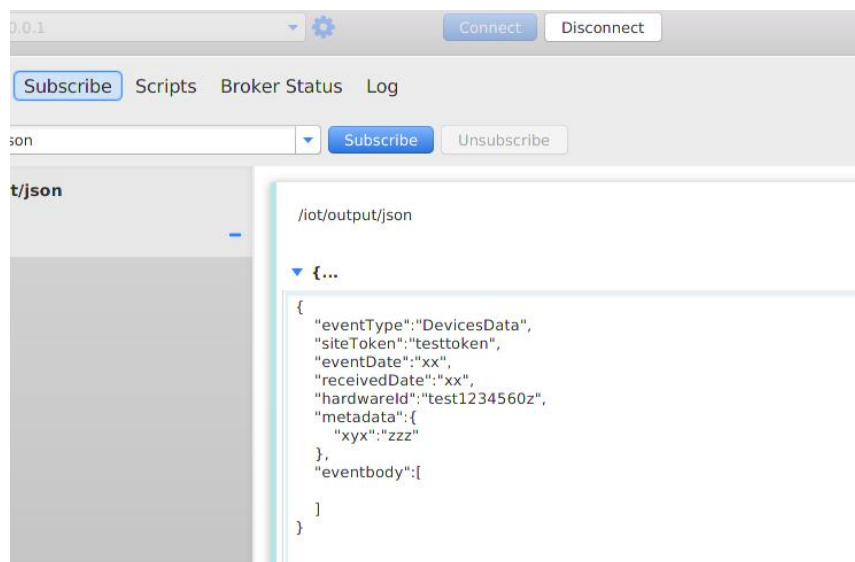


图 6.2 消息订阅测试

6.2.4 测试结论

这次测试的内容主要是对智能家居物联网服务平台的接口进行测试，包括用户、租户、区域、设备等接口的测试。单元测试，确保系统各模块逻辑正确、功能正确，集成测试确保系统服务稳定可靠。经过以上测试，智能家居物联网服务平台系统稳定，功能符合预期。

7 总结和展望

7.1 总结

本文针对目前国内外智能家居行业所面临的一些问题进行研究，例如：理论研究和实践分离；智能家居系统控制能力有限、扩展能力差、服务器资源利用率不高；已有的一些平台之间协议不兼容等问题。本文针对以上诸多问题对物联网技术应用于智能家居系统进行研究。本文研究了基于 RESTful 服务的 web PaaS/IoT PaaS 混合的服务框架，着重研究 RESTful API、MQTT 物联网发布订阅协议，flask+gunicorn+nginx 的高性能系统优化，Redis+MongoDB+InfluxDB 的持久化存储。在以上的研究和实践，形成了基于 RESTful 服务的 web PaaS/IoT PaaS 混合的高弹性、高稳定性、高扩展性的智能家居物联网服务平台解决方案。

7.2 展望

本文对智能家居物联网服务平台进行了深入研究，虽然取得了一定的研究成果，但是由于现有软硬件资源、行业技术的限制，该系统依然有诸多问题需要改进。首先，安全问题一直伴随着物联网发展，本系统虽然加强了安全措施，但安全问题不容小觑。其次，系统开发基于 python 语言，虽然在开发效率上较有优势，但运行效率远不及较为底层的 C/C++ 高效。最后，平台的智能化、个性化服务也是今后智能家居服务平台发展方向。

智能家居行业的发展是当前科学技术聚集的焦点。鉴于以上问题，智能家居服务平台接下来应该立足于传统软件、硬件技术，结合日益成熟的大数据、云计算技术，能很好地解决系统的性能问题。然而，系统的更多的需求是智能分析处理的个性化服务，则依赖于当前的区块链、人工智能等技术。在融合以上传统技术与新兴技术的基础上，智能家居行业必将产生前所未有的巨变。在融合以上传统技术与新兴技术的基础上，智能家居行业必将产生前所未有的巨变。

参考文献

- [1] Jong Hyuk .Intelligent systems and smart homes .Inf Syst Front (2009) 11:481 - 482
- [2] 程冬梅, 王瑞聪, 刘燕, 等. 基于 REST 架构风格的物联网服务平台研发[J]. 计算机工程与应用, 2012, 48(14):74-78.
- [3] 孙其博, 刘杰, 黎彝等. 物联网:概念、架构与关键技术研究综述 .北京邮电大学学报 .1007-5321(2010)03-0001-09
- [4] Charlie Wilson • Tom Hargreaves .Smart homes and their users: a systematic analysis and key challenges .10.1007/s00779-014-0813-0
- [5] YUAN Xiao-Chen, CHEANG Chak-Fong, LI Jian-Qing 等 .Implementation of a Remote Control System for Smart Home .信息通信技术
- [6] 陈云. 基于物联网和云计算的智能家居系统的应用. 技术分析. 1674-1900 (2017) 20-0029-01
- [7] 杜经纬, 李海涛, 梁涛. 国内外物联网研究现状及展望. 中国科学院地理科学与资源研究所, 北京 100101
- [8] 李 金凤. 一种基于MQTT发布/订阅机制的冷藏运输智能监控系统 [J]. 互联网天地, 2014 (7) 30-33
- [9] 吕莉, 罗杰. 智能家居及其发展趋势. 计算机与现代化. 1006-2475 (2007) 11-0018-03
- [10] 邱实, 汪明, 李旭等. 基于物联网的智能家居管控系统设计. 建筑电气. 1003 - 8493. 2017. 07. 011
- [11] 魏迎 . 综合布线集成环境的设计与实现 [D] . 西安: 西安电子科技大学, 2014
- [12] 龚华明, 阴躲芬. 物联网三层体系架构及其关键技术浅析[J]. 科技广场, 2013(2):20-23.
- [13] M. J. Kaur, P. Maheshwari. Building smart cities applications using IoT and cloud-based architectures [D] International Conference on Industrial Informatics and Computer, 2016
- [14] 丛林. 基于技术、应用、市场三个层面的我国物联网产业发展研究 [D] . 沈阳: 辽宁大学, 2016
- [15] 胡晓喻, 陈庆奎 . 智能家居接入服务器策略的设计与实现 . 计算机工程与设计. 1000-7024(2017)02-0544-0
- [16] 泉琳. “互联网+”, 向纵深挺进[J]. 科学新闻, 2017(3).
- [17] 俞文俊, 凌志浩. 一种物联网智能家居系统的研究[J]. 自动化仪表, 2011, 32(8):56-59.

- [18] 俞文俊, 凌志浩. 一种物联网智能家居系统的研究[J]. 自动化仪表, 2011, 32(8):56-59.
- [19] 王明霞. 探寻物联网的发展轨迹[J]. 政工导刊, 2017(4):25-27.
- [20] 宁方华, 牛建瑞, 俞武嘉, 等. Windows Azure 平台数据存储的访问控制研究[J]. 浙江理工大学学报, 2014, 31(1):75-78.
- [21] 刘芳, 陈义明. 一种面向大数据分析的物联网平台架构[J]. 电脑知识与技术, 2017, 13(20):3-4.
- [22] 杨运平, 吴成宾. 一种无线环境监测数据采集与推送系统[J]. 成都大学学报(自然科学版), 2015, 34(1):59-62.
- [23] 申斌, 张桂青, 汪明, 等. 基于物联网的智能家居设计与实现[J]. 自动化与仪表, 2013, 28(2):6-10.
- [24] Ramsey B W, Mullins B E, White E D. Improved tools for indoor ZigBee warwalking[C]// Local Computer Networks Workshops(LCN Workshops), 2012 IEEE 37th Conference on. IEEE , 2012:921-924.
- [25] Sajal K. Das. Mobility and Resource Management in Smart Home Environments. Lecture Notes in Computer Science.. 2004, 1109-1111

致 谢

本设计在白小军老师的悉心指导和严格要求下业已完成，从课题选取、方案论证到具体设计和调试，无不凝聚着老师的心血和汗水。论文从零到一的过程中，每走一步对我来说都是新的尝试与挑战，这也是我在大学期间独立完成的最令我鼓舞的项目。在这段时间里，我学到了很多知识，从开始片面的认识到逐渐加深了解，再到熟悉。在这个过程中，不但学会了独立思考问题、查阅资料、分析和讨论问题、解决问题，还学会了整理资料、分享知识。看着自己的作品一步步完善起来，每一次完善都是收获颇丰，每一次试验的成功都让我欢欣鼓舞。

力不足者，中道而废。本设计能够顺利的完成，也归功于各位任课老师的认真负责，使我能够很好的掌握和运用专业知识，并在设计中得以体现。正是有了他们的悉心帮忙和支持，才使我的毕业设计顺利完成，在此向西安工业大学的全体老师表示由衷的谢意。感谢他们四年来的辛勤栽培。在四年的本科学习和生活期间，切身感受着导师的精心指导和无私的关怀，我将受益终身。

本文在写作过程中参考了大量的文献资料，主要文献资料已开列出来在此向所有的作者表示深深的感谢！

由于本次毕业设计的时间有限，对于系统的设计还有很多不完善的地方，我会在以后的学习中加以修改，希望老师和同学提出宝贵意见。

毕业设计（论文）知识产权声明

本人完全了解西安工业大学有关保护知识产权的规定，即：本科学生在校攻读学士学位期间毕业设计（论文）工作的知识产权属于西安工业大学。本人保证毕业离校后，使用毕业设计（论文）工作成果或用毕业设计（论文）工作成果发表论文时署名单位仍然为西安工业大学。学校有权保留送交的毕业设计（论文）的原文或复印件，允许毕业设计（论文）被查阅和借阅；学校可以公布毕业设计（论文）的全部或部分内容，可以采用影印、缩印或其他复制手段保存毕业设计（论文）。

（保密的毕业设计（论文）在解密后应遵守此规定）

毕业设计（论文）作者签名：

指导教师签名：

日期：

毕业设计（论文）独创性声明

秉承学校严谨的学风与优良的科学道德，本人声明所呈交的毕业设计(论文)是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，毕业设计（论文）中不包含其他人已经发表或撰写过的成果，不包含他人已申请学位或其他用途使用过的成果。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了致谢。

毕业设计（论文）与资料若有不实之处，本人承担一切相关责任。

毕业设计（论文）作者签名：

指导教师签名：

日期：



毕业论文外文资料

院（系）： 计算机科学与工程学院

专 业： 物联网工程

班 级： 14060610

学 生： 兰 洋

学 号： 14060610105

指导教师： 白小军

外文出处： <http://www.ijetae.com>

附 件： 1.译文； 2 原文

2018 年 6 月

附件 A

设备和客户端授权设计以解决物联网安全问题

摘要

本文提出了针对物联网设备和客户端（用户控制这些设备）的授权服务器设计方法来解决物联网的安全问题。授权服务器（AS）是开放授权（OAuthTM）协议中描述的参与者之一。在成功认证资源所有者并获得授权之后，负责向客户端发布访问令牌。当前版本的授权服务器只支持“资源所有者密码凭据”授权授予类型。资源所有者身份验证将基于目录服务器交互来实现。令牌被表示为一个 128 位的一成不变的 UUID（本地 java 实现）。在 RFC4122 中描述的生成过程：通用唯一标识符（UUID）URN 命名空间，UUID 是使用密码强伪随机数生成的。密码强随机数最小符合 FIPS140-2 中指定的统计随机数生成器测试，密码模块的安全要求，输出序列是密码强的，如 RFC 中所描述的 1750：安全性的随机性建议。

关键字

OAS (Open Authorization Server), JSON(Java Script Object Notation), LDAP (Lightweight Directory Access Protocol), UUID (Universally Unique Identifier), FIPS

一、引言

物联网的承诺远远超过单个设备的承诺。移动性管理是一个快速发展的例子，物联网设备的影响。想象一下，如果突然交付给你的组织的每一个包裹都产生了一个内置的 RFID 芯片，它可以连接到你的网络，并将自己识别到它的连接的物流系统。或者你可以想象一个医疗环境，其中检查室中的每一个仪器都连接到网络上，通过传感器收集病人数据。即使在像农业这样的工业中，想象一下每一只家畜都是数字追踪的，以监测其位置、健康和行为。物联网的可能性是无限的，所以可以显化的设备数量也是如此。

长期以来，物联网和机器对机器（M2M）生态系统的兴起一直是人们所期待的。由于生态系统覆盖了云计算和大数据等主要趋势，企业需要做好准备，以安全地应对新一轮的连接智能设备，并保护随之而来的数据。为了更好地理解即将到来的浪潮中的安全现实，BeCHAM 研究已经分析了其对 M2M/IOT 市场 Oracle 进行的近期研究调查的相关结果。通过对 BeCHAM 研究的更多最近的研究，重点分析了 IOT 时代的安全关注和方法。

- 1、系统方法在确保任何物联网中的迫切性程序？
- 2、IOT 最新的安全理念如何影响程序的成功？
- 3、强强化设备、数据、识别和更多的最佳实践是什么？
- 4、如果改进是差异化的关键，那么如何在不牺牲安全性的情况下进行创新，这往往是至高无上的？
- 5、对设备制造商和 ISV 的影响是什么？更多的成本或更多的机会？

二、相关工作

2.1 我们是如何得到网络安全的进化的：

保护数据是自前两台计算机相互连接以来一直存在的问题。通过互联网的商业化，安全担忧扩大到覆盖个人隐私、金融交易和虚拟盗窃的威胁。在物联网中，

安全是安全的。无论是非自愿还是恶意，与起搏器、汽车或核反应堆的控制交织在一起对人类的生命构成危险。安全控制与网络进展并行，从 20 世纪 80 年代末的第一包过滤防火墙到更精细的协议和应用感知防火墙、入侵检测和预测系统（IDS/IPS）和安全事件和事件管理（SIEM）解决方案。这些控制措施试图阻止企业网络中的恶意活动，如果他们获得了访问权，就会发现它们。

如果恶意软件完成了防火墙，则在签名匹配和黑名单上创建的反病毒技术将介入识别和解决问题。未来，随着恶意软件的扩展和避免发现先进技术，白名单技术开始取代黑名单。类似地，随着添加的设备开始进入企业网络，开发了许多访问控制系统，以验证设备和坐在他们之后的用户，并授权那些用户和设备进行特定的动作。

2.2 个新威胁：挑战和挑战：

在 IOT 世界中涉及这些相同的实践或变体需要大量的重新设计来解决设备约束。例如，BANIN 需要太多的磁盘空间，以用于物联网应用。嵌入式设备是设计用于低功耗，具有小硅形式，往往具有有限的连通性。它们通常具有与它们的任务一样多的处理能力和存储器。它们通常是无头的，也就是说，没有一个人能够输入身份验证 ID 或决定应用程序是否应该被信任；他们必须创建自己的判断和决定是否接受命令或执行任务。无限多样性的物联网应用立场同样广泛的各种各样的安全挑战。例如：

在工厂地面自动化中，操作机器人系统的深度嵌入式可编程逻辑控制器（PLC）通常与企业 IT 基础设施相结合。在保护 IT 基础设施投资和利用可获得的安全控制的同时，如何保护这些 PLCs 免受人为干扰？

同样地，核反应堆的控制系统也附属于基础设施。他们如何及时地获得软件更新或安全补丁，而不损害功能安全性，或者在每次推出补丁时都产生显著的重新认证成本？

一个聪明的仪表，能够发送能量使用数据到公用事业运营商进行动态计费或实时电网优化，必须能够屏蔽这些信息免受未经授权的使用或启示。电力使用量下降的信息可以表明一个家是空的，使它成为入室盗窃的最好目标或更糟。

三、提议工作

3.1 自下而上的建筑安全：

知道没有一个单独的控制将充分保护一个设备，我们如何应用我们在过去 25 年中获得的来实现各种场景的安全性？我们这样做是通过一个多层的安全方法，开始时，当电源被应用时，启动一个可信的计算基线，并锚定信任不可篡改的不可改变的事物。安全必须通过设备一直解决。生命周期，从最初的设计到操作环境：安全引导：当电源首次被引入设备时，使用密码生成的数字签名验证设备上软件的合法性和准确性。访问控制：接下来，应用多种形式的储备和访问控制。建立在操作系统中的强制或基于角色的入口控制限制了设备组件和应用程序的特权，因此它们只访问他们想做的工作设备认证的资源：当设备插入到网络中时，它应该在 T 之前进行身份验证。接收或传输数据。FifWrand 和 IPS：该设备还需要防火墙或深度包来克服在设备更新和补丁中注定要终止的流量：在设备进入之后操作，它将开始接收热点补丁和软件更新。运营商需要推出补丁，并且设备需要验证它们，以不消耗带宽或损害设备的有用安全性的方式。

3.2 端到端安全解决方案：

设备和网络级别的安全对物联网的运行是至关重要的。同样的智能，使设备

能够完成他们的任务也必须使他们能够识别和抵消威胁。幸运的是，这并不需要革命性的方法，而是在 IT 网络中被证明成功的措施的演变，修改为物联网的挑战和连接设备的约束。而不是寻找一个尚未存在的解决方案，或提出一种革命性的安全方法，风河专注于提供当前最先进的 IT 安全控制，优化了新的和极其复杂的嵌入式应用程序驱动的物联网。

为了减轻 IOT 设备和客户机（控制这些 IOT 设备的应用）的安全挑战，我们建议下面的授权和认证模型。这将解决当前的物联网设备的安全问题。

3.3 提出的高层次设计概述：

授权服务器（AS）是开放授权（OAuthTM）协议中描述的参与者之一。在成功认证资源所有者并获得授权之后，负责向客户端发布访问令牌。当前版本的授权服务器只支持“资源所有者密码凭据”授权授予类型。资源所有者身份验证将基于目录服务器交互来实现。

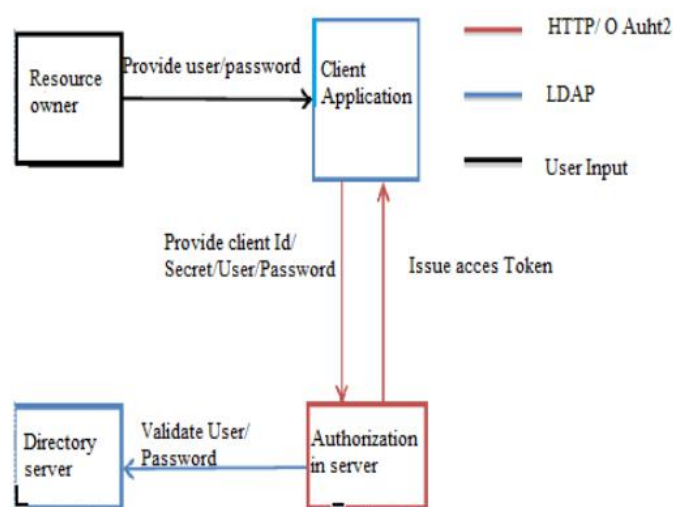


图 1：交互概述

3.4 设计概述：

这一部分包含开放式授权服务器的总体架构、逻辑和物理设计以及部署拓扑设计。本节简要介绍了用于设计过程的堆栈的 OAS 体系结构和关键组件，这些细节将在后续章节中详细介绍。OAS 组件与客户端实体交互，并使用“资源所有者密码”提供客户端令牌的管理。

在 AOUTH 2 协议规范中描述的证书授权授权。OAS 基于这样的业务关系（用户、客户机和令牌之间）：客户端在同一时间段内只有一个访问/刷新令牌对。客户端可以使用任何用户凭据来获取令牌，并且在将来的尝试中不使用其他凭据来避免。

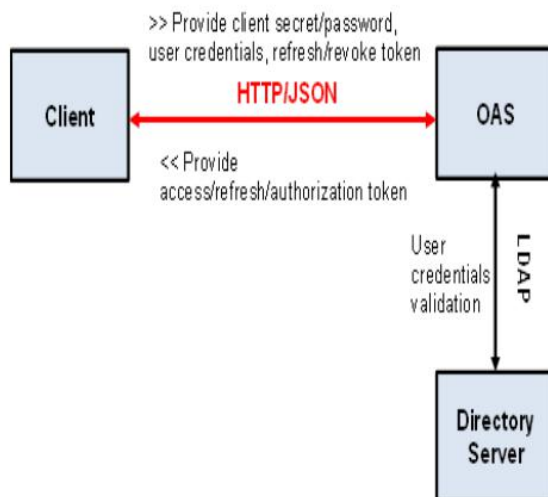


图 2：接口和通信协议

OAS 使用 HTTP/JSON 与所有客户端交互。在请求处理过程中，OAS 使用 LDAP 与目录服务器进行交互。OAS 服务器只支持“资源所有者密码授权授权”。下图说明了获取令牌所需的步骤。

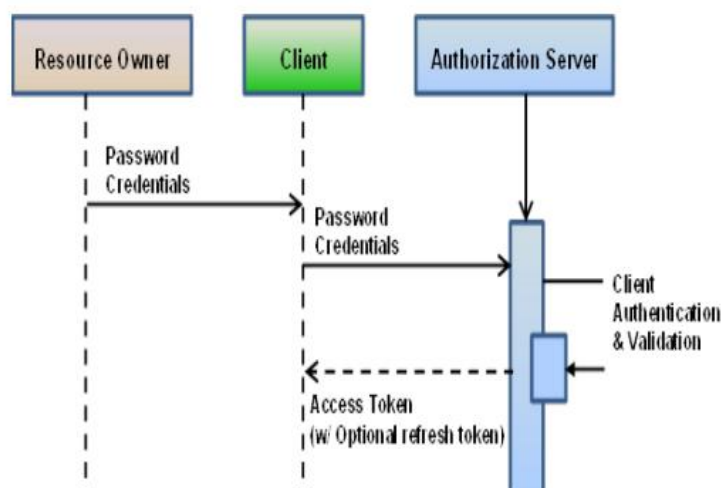


图 3：资源所有者密码授权流程

图 3 所示的流程包括以下步骤：

- 1、资源所有者为客户端提供用户名和密码。
- 2.客户端通过从资源所有者接收的凭据来请求来自 OAS 的令牌。在进行请求时，客户端用授权服务器。
- 3、授权服务器验证客户端并验证资源所有者凭据，如果发现有效，则发出访问令牌。

3.5 客户端认证：

客户端通过将其凭据（客户端 ID /客户端秘密）作为请求报头（HTFEP 基本认证方案在[RCF2617]中定义）而对其进行身份认证。客户端秘密在所有客户端上都有一个可预配置的值。这意味着在请求中包含任何客户端 ID，如果客户端机密与可信秘密匹配，则客户端将被授权。

3.6 资源拥有凭证验证:

在请求中提供的资源所有者凭据将使用“绑定”方法对目录服务器进行验证。OAS 将尝试使用资源所有者凭据绑定到目录服务器，并执行搜索查询。在绑定或搜索失败的情况下，用户将变得未经授权。可以配置搜索查询。

在使用目录服务器 OAS 验证用户凭据之前:

如果用户名（例如<用户名>@ABC.com）包含与一个预定义域（例如 ABC）在一个配置中匹配的域部分，将从用户名删除域部分（例如<用户名>），然后将其传递到目录服务器进行验证。

如果用户名（<用户名>@XYZ.com）包含与 Apple 配置中的任何预定义域（例如 ABC）不匹配的域部分，因为它不会从用户名删除域部分（例如<用户名>@XYZ.com），并将其传递给用于验证的目录服务器。

如果用户名（例如<用户名>）不包含域部分，AS 将将其传递到目录服务器以进行验证。

3.7 authorization 令牌加密机制:

如果任何的恶意用户的用户名和密码，然后我知道 the can get hold of the OAS 旁路授权令牌 and can with Application to authenticate and possibility of getting there may be for one and 注册用户授权令牌，令牌授权另一用户由于是静态的爱 for all users)。so to avoid such OAS 的情况下，授权令牌 encrypts the combination of which is uconnect ID 时，访问令牌，in and 随机种子。is the 令牌加密使用 RSA 加密算法。The RSA 加密授权令牌，然后使用 Base64 编码 encoding is the above is also prefixed 生成令牌（separated by 结肠）配置静态 value to get with the information about the 客户。

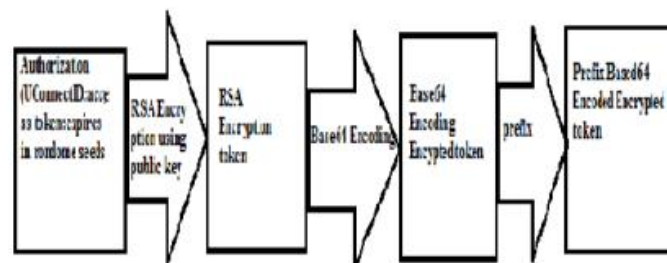


图 3：加密流程

四、结论与未来展望

具有通信驱动能力的设备的扩散正在使物联网更加接近，其中感测和驱动功能无缝地融合到背景中，并且通过丰富的新信息源的访问使得新的能力成为可能。但是随着数百万设备和客户端应用程序（移动应用）数量的不可控制的增加，它们的安全性将是一个挑战，需要通过选择底层到顶部的设计方法和多个可扩展性来解决。跨越不同的平台。在设备和网络级的安全性是物联网操作的关键。使设备能够执行任务的智能还必须使它们能够识别和抵消安全威胁。幸运的是，这并不需要革命性的方法，而是在 IT 网络中证明成功的措施的演变，适应 IOT 和连接设备的约束的挑战。代替搜索一个尚未存在的解决方案，我们需要在两个设备以及网络级上处理安全性。

附件 B



International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 5, Issue 3, March 2015)

Device And Client Authorization Design Approach To Address IOT's Security Concerns

Ashwini A.Varhekar¹, Prof. S. P. Mankar²

¹Computer science and Engineering SGBAU, Guruji Ward, Arvi, Maharashtra, India

²Information Technology SGBAU, Guruji Ward, Arvi, Maharashtra, India

Abstract— This paper deals with addressing security concerns of IOT by proposing Authorization Server design approach for both IOT devices as well as Client (The user controlling these devices). Authorization Server (AS) is one of the actors described in Open Authorization (OAuth) protocol. AS take care of issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization. Current version of Authorization Server supports "Resource Owner Password Credentials" authorization grant type only. The resource owner authentication will be implemented based on Directory Server interactions. Token is represented as a 128-bit immutable universally unique identifier UUID (native java implementation). Generation process described in RFC 4122: A Universally Unique Identifier (UUID) URN Namespace, UUID is generated using a cryptographically strong pseudo random numbers. A cryptographically strong random number minimally complies with the statistical random number generator tests specified in FIPS 140-2, Security Requirements for Cryptographic Modules, An output sequence is cryptographically strong, as described in RFC 1750: Randomness Recommendations for security.

Keywords— OAS (Open Authorization Server), JSON (Java Script Object Notation), LDAP (Lightweight Directory Access Protocol), UUID (Universally Unique Identifier), FIPS

I. INTRODUCTION

The promises of IOT go far beyond those for individual devices. Mobility management is a rapidly evolving example of the impact of IOT devices. Imagine if suddenly every packages delivered to your organization came up with a built-in RFID chip that could connect to your network and identify itself to its connected logistics system. Or you can picture a medical environment in which every instrument in the exam room connected to the network to transmit patient data collected via sensors. Even in the industries like farming, imagine if every domestic animal were digitally tracked to monitor its location, health and behavior. The IOT's possibilities are limitless, and so is the number of devices that could manifest [1]

The IOT and the rise of a machine-to-machine (M2M) ecosystem have been long anticipated. As the ecosystem covered with major trends like the cloud computing and big

Data, businesses need to be prepared to securely address the new wave of connected intelligent devices and protect the data that comes with them. To help better understand the realities of security in this coming wave, Beecham Research has analyzed the relevant results of its recent research survey conducted for Oracle of the M2M/IOT market. This analysis of security concerns and methods for the IOT era are Enriched by additional recent studies by Beecham Research focused on key points: 1. How imperative is a system approach in securing any IOT program? 2. How are the newest ideologies of security for IOT affecting program success? 3. What are the best practices in strongly fortifying devices, data, identify and more? 4. If improvement is key to differentiation, how do you deliver innovation without compromising on security, which is often supreme? 5. What are the effects on device manufacturers and ISV's? More cost or more opportunity.

II. RELATED WORK

How we got here the evolution of network security:

Protection of data has been a question ever since the first two computers were connected to each other. By means of the commercialization of the Internet, security worries expanded to cover personal privacy, financial dealings, and the threat of virtual theft. In IOT, security is attached from safety. Whether involuntary or malicious, interfering with the controls of a pacemaker, a car, or a nuclear reactor poses danger to human life.

Security controls have advanced in parallel to network advancement, from the first packet-filtering firewalls in the late 1980s to more refined protocol- and application-aware firewalls, intrusion detection and anticipation systems (IDS/IPS), and security event and event management (SIEM) solutions. These controls struggled to keep malicious activity off of corporate networks and detect them if they did gain access.



International Journal of Emerging Technology and Advanced Engineering

Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 5, Issue 3, March 2015)

If malware accomplished to breach a firewall, antivirus techniques created on signature matching and blacklisting would kick in to identify and cure the problem.

Future, as the universe of malware expanded and techniques for avoiding finding advanced, whitelisting techniques started replacing blacklisting. Similarly, as added devices started coming onto corporate networks, many access control systems were developed to authenticate both the devices and the users sitting after them, and to authorize those users and devices for specific actions.

2.2 New threats constrain and challenges:

Relating these same practices or variants of them in the IOT world needs substantial reengineering to address device constraints. Banning, for example, needs too much disk space to be practical for IOT applications. Embedded devices are designed for little power consumption, with a small silicon form and often have limited connectivity. They typically have only as much processing capability and memory as desirable for their tasks. And they are often “headless”—that is, there isn’t a human being functioning them who can input authentication IDs or decide whether an application should be trusted; they must create their own judgments and decisions about whether to accept a command or execute a task. The endless diversity of IOT applications stances an equally wide variety of security challenges. For example:

- In factory ground automation, deeply embedded programmable logic controllers (PLCs) that operate robotic systems are typically combined with the enterprise IT infrastructure. How can those PLCs be protected from human interference while at the same time defending the investment in the IT infrastructure and leveraging the security controls obtainable?
- Equally, control systems for nuclear reactors are attached to infrastructure. How can they get software updates or security patches in a timely mode without impairing functional safety or incurring significant recertification costs every time a patch is rolled out?
- A clever meter—one which is able to send energy usage data to the utility operator for dynamic billing or real-time power grid optimization—must be able to shield that information from unauthorized usage or revelation. Information that power usage has dropped could indicate that a home is empty, making it an best target for a burglary or worse.

III. PROPOSED WORK

3.1 Building Security from Bottom up approach:

Knowing no one lone control is going to adequately protect a device, how do we apply what we have acquired over the past 25 years to implement security in a variety of scenarios? We do so through a multi-layered approach to security that starts at the beginning when power is applied, launches a trusted computing baseline, and anchors that trust in something immutable that cannot be tampered with. Security must be addressed all the way through the device lifecycle, from the initial design to the Operational environment:

Secure booting: When power is first introduced to the device, the legitimacy and veracity of the software on the device is verified using cryptographically generated digital signatures.

Access control: Next, diverse forms of reserve and access control are applied. Mandatory or role-based entrance controls built into the operating system limit the privileges of device components and applications so they access only the resources they want to do their jobs

Device authentication: When the device is plugged into the network, it should authenticate itself prior to receiving or transmitting data.

Firewalling and IPS: The device also wants a firewall or deep packet going-over capability to control traffic that is destined to terminate at the device

Updates and patches: After the device is in operation, it will start receiving hot patches and software updates. Operators need to roll out patches, and devices need to validate them, in a way that does not consume bandwidth or impair the useful safety of the device.

3.2 End to end security solutions:

Security at both the device and network levels is acute to the operation of IOT. The same intelligence that enables devices to accomplish their tasks must also enable them to recognize and counteract threats. Luckily, this does not require a revolutionary approach, but rather an evolution of measures that have proven successful in IT networks, revised to the challenges of IOT and to the constraints of connected devices. Instead of searching for a solution that does not yet exist, or proposing a revolutionary approach to security, Wind River is focusing on delivering the current state-of-the-art IT security controls, optimized for the new and extremely complex embedded applications driving the Internet of Things.



International Journal of Emerging Technology and Advanced Engineering
 Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 5, Issue 3, March 2015)

To mitigate this security challenges at both IOT devices and client (Application which controlling these IOT devices) we are suggesting below Authorization and Authentication model. This will address current security concerns of the IOT devices.

3.3 Proposed high level design overview:

Authorization Server (AS) is one of the actors described in Open Authorization (OAuth) protocol. AS take care of issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization. Current version of Authorization Server supports "Resource Owner Password Credentials" authorization grant type only.

The resource owner authentication will be implemented based on Directory Server interactions.

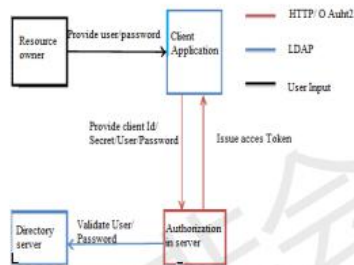


Figure 1: Interactions Overview

3.4 Design Overview:

This section contains overall Architecture, Logical and Physical Design, and Deployment Topology Design for Open Authorization Server. This section provides a brief overview of OAS architecture and key components of the stack that have been used to design processes, which are detailed in subsequent sections. The OAS component interacts with Client entities and provides management of client tokens, using "Resource Owner Password Credential" authorization grant described in OAuth 2.0 protocol specifications.

OAS is based on such business relationships (between user, client and tokens): A client can have only one access/refresh token pair associated with it in the same period of time. A client can use any user credentials to obtain tokens and not avoided to use other credentials in future attempts.

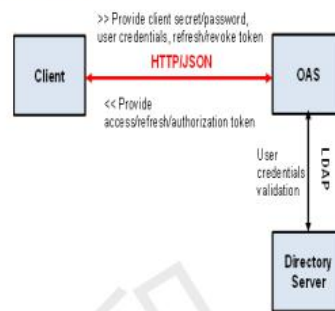


Figure 2: Interfaces and communication Protocol

OAS interacts with all clients using HTTP/JSON. During processing of the request, OAS interacts with Directory Server using LDAP. OAS server supports only "Resource Owner Password" authorization grant. Below figure illustrates the steps needed to obtain tokens.

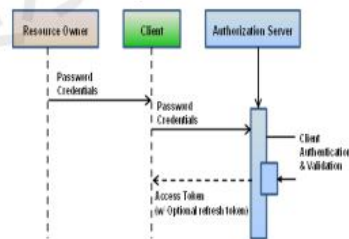


Figure 3: Resource Owner Password authorization flow

The flow illustrated in Figure 3 includes the following steps:

1. The resource owner provides the client with its username and password.
2. The client requests tokens from OAS by including the credentials received from the resource owner. When making the request, the client authenticates with the authorization server.
3. The authorization server authenticates the client and validates the resource owner credentials, and if found valid, it issues an access token.



International Journal of Emerging Technology and Advanced Engineering

Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 5, Issue 3, March 2015)

3.5 Client Authentication:

Client authenticates itself by including its credentials (client id/client secret) in request as authentication header (HTTP Basic authentication scheme defined in [RFC2617]).

Client secret has a single pre-configurable value across all clients. It means that whatever client id is included in the request, if client secret matches with the trusted secret then client will be authorized.

3.6 Resource owned credentials validation:

Resource owner credentials provided in request will be validated against Directory Server using "binding" approach. OAS will try to bind to Directory Server using resource owner credentials and perform a search query. In case binding or search fails, user will become unauthorized. Search query could be configured.

Before validating user credentials with Directory Server OAS checks:

- If user name (e.g. <username>@abc.com) contains the domain part that matches with one of the predefined domains (e.g. abc.com) in AS configuration, AS will remove the domain part from user name (e.g. <username>) and then pass it on to the Directory Server for validation.
- If user name (<username>@xyz.com) contains the domain part that does not match with any of the predefined domains (e.g. abc.com) in AS configuration, AS will not remove the domain part from user name (e.g. <username>@xyz.com) and pass it on to the Directory Server for validation as is.
- If user name (e.g. <username>) does not contain the domain part, AS will pass it on to Directory Server for validation as is.

3.7 Authorization token encryption mechanism:

If any malicious user knows the username and password then he can get hold of authorization token and can bypass the OAS to authenticate with application and there may be possibility of getting authorization token for one user and register for another user since the authorization token is static (same for all users).

So to avoid such situations, OAS encrypts the authorization token which is combination of uconnect ID, access token, expires in and random seed. The token is encrypted using RSA encryption algorithm. The RSA encrypted authorization token is then encoded using Base64 encoding. The above generated token is also prefixed (separated by colon) with the configurable static value to get information about the client.

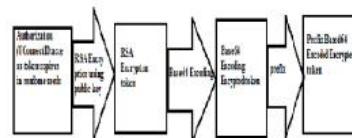


Figure3: Flow of Encryption

IV. CONCLUSIONS AND FUTURESCOPE

The proliferation of devices with communicating-actuating capabilities is bringing closer the vision of an Internet of Things, where the sensing and actuation functions seamlessly blend into the background and new capabilities are made possible through access of rich new information sources. But with the insurmountable increase in the number of millions devices and client application (Mobile application) controlling them security of them will be a challenge and it needs be addressed by choosing bottom to top designing approach and multiple scalability across different platform.

Security at both the devices as well as network levels is critical to the operation of IOT. The intelligence that enables devices to perform their tasks must also enable them to recognize & counteract security threats. Fortunately, this does not require a revolutionary approach, but rather an evolution of measures that have proven successful in IT networks, adapted to the challenges of IoT and to the constraints of connected devices. Instead of searching for a solution that does not yet exist, we need to make address the security at both devices as well as network level.

REFERENCES

- [1] [wr_security-in-the-internet-of-things.pdf](#).
- [2] OWASP_Internet_of_Things_Top_Ten_Project
- [3] Aberer, K., Hauswirth, M., Salchi, A.: Middleware Support for the Internet of Things. In: 5th
- [4] <http://www.cisco.com/web/solutions/trends/iot/security.html>
- [5] Bohn, H., Bobek, A., Golasowski, F.: SIRENA - Service Infrastructure for Realtime Embedded
- [6] Networked Devices: A Service Oriented Framework for Different Domains.
- [7] Developing Solution's for IOT - White paper by Intel corp.
- [8] Securing future designs security for IOT by harber research.com.
- [9] Security concern of IOT devices by Machantosh Magazine.
- [10] Security of M2M communication - by m2m alliance.com.
- [11] Native Java implementation for UUID
<http://docs.oracle.com/javase/6/docs/api/java/util/UUID.html>