学校代码：10286
分 类 号：TP311
密　　级：公开
U　D　C：004.4
学　　号：184644

东南大学

工程硕士学位论文

基于 MQTT 协议的物联网消息系统的设计与实现

（学位论文形式：应用研究）

研究生姓名：孙磊

导 师 姓 名：孔佑勇 副教授

梁蕤 高工

申请学位类别 工 程 硕 士　　学位授予单位 东 南 大 学

工程领域名称 软 件 工 程　　论文答辩日期 2021 年 5 月 26 日

研究方向 软 件 工 程　　学位授予日期 20　年　月　日

答辩委员会主席 何 洁 月　　评　阅　人 盲 审

2021 年 6 月 4 日

东南大学

工程硕士学位论文

基于 MQTT 协议的物联网消息系统的设计与实现

专 业 名 称： 软件工程

研究生姓名： 孙 磊

导 师 姓 名： 孔佑勇

校 外 导 师： 梁 蕤

DESIGN AND IMPLEMENTATION OF MESSAGE SYSTEM FOR INTERNET OF THINGS BASED ON MQTT PROTOCOL

A Thesis Submitted to

Southeast University

For the Professional Degree of Master of Engineering

BY

Sun lei

Supervised by

Associate Professor Kong You-yong

And

Senior Engineer Liang rui

College of Software Engineering

Southeast University

June 4, 2021

东南大学学位论文独创性声明

本人声明所呈交的学位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，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得东南大学或其它教育机构的学位或证书而使用过的材料。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谢意。

研究生签名：[signature] 日期：2021.6.4

东南大学学位论文使用授权声明

东南大学、中国科学技术信息研究所、国家图书馆、《中国学术期刊（光盘版）》电子杂志社有限公司、万方数据电子出版社、北京万方数据股份有限公司有权保留本人所送交学位论文的复印件和电子文档，可以采用影印、缩印或其他复制手段保存论文。本人电子文档的内容和纸质论文的内容相一致。除在保密期内的保密论文外，允许论文被查阅和借阅，可以公布（包括以电子信息形式刊登）论文的全部内容或中、英文摘要等部分内容。论文的公布（包括以电子信息形式刊登）授权东南大学研究生院办理。

研究生签名：[signature] 导师签名：[signature] 日期：2021.6.4

摘要

随着物联网行业的兴起，物联网市场规模持续增长，物联网设备数量逐年递增。种类繁多、数量庞大的设备也给物联网应用带来了新的挑战，传统的方法是企业管理者通过现场液晶查看数据或者手工记录数据的方式来监控设备。然而这种方式存在着数据碎片化影响生产监管、生产设备状态不能及时把控、难以提升产品质量和降低成本等问题。为解决上述问题，本文提出了构建物联网消息系统。

针对物联网终端设备网络带宽低、计算和存储资源有限、网络环境较差等问题，本文选取轻量级的 MQTT 协议作为消息推送协议，设计并实现了基于 MQTT 协议的物联网消息系统。本文主要工作如下：

（1）结合 MQTT3.1.1 版本协议设计并实现了基础的消息发布订阅功能。将物联网设备状态分为连接、心跳保活和离线三种，解决了分布式集群下 Session 信息存储问题。基于 HBase 设计了离线消息存储保证了消息的可靠性。

（2）针对系统安全，设计并实现了设备身份验证和 ACL 主题鉴权模块。将设备身份验证和 ACL 主题鉴权服务独立出来，以 HTTP API 的方式提供验证服务，能够根据企业业务需要提供可灵活配置的设备身份认证和主题鉴权。根据业务需要设计了 Kafka 消息桥接接口，将消息系统与业务系统进行解耦。

（3）设计并实现了去中心化的消息系统集群，基于 Akka 实现集群节点管理，方便监控各节点运行状态；基于路由表实现了集群间的消息转发，保证了消息在集群间的精确转发。设计并实现了系统监控模块，以 Web 页面方式提供对系统的监控，方便系统管理员对相关业务信息进行统计与监控。

最后对系统进行了详细的功能测试和性能测试，测试结果表明该系统不仅能够完成基本的消息推送功能，还支持企业定制化的业务需求，在性能上支持十万连接数和万级的并发，达到了预期的目的，具有较高的工程应用价值。

关键词：MQTT 协议，物联网，消息系统，集群

Abstract

With the development of IoT industry, the market as well as the number of device keeps growing. In addition, amounts of equipments for IoT applications have also brought new challenges. The traditional approach is that enterprise managers monitor equipment utilizing the LCD viewing data or manual recording data. However, such a way exists a series limitations, including data fragmentation, the production equipment status can not be in real-time control, the difficulty in the improvement of the quality of product, and the reduction in cost, etc. In order to solve the above problems, this paper proposes to build the messaging system of IoT.

Aim at the problems of low network bandwidth, limited computing or storage resources, and poor network environment, this paper selects lightweight MQTT protocol as a message push protocol, designs and implements an IoT messaging system. The main contents of this paper are as follows:

(1) Based on MQTT3.1.1 protocol, we design and implement the basic function for messags publishing and subscription. The device states of the IoT can be divided into three categories:connected, heartbeat alive and offline, which solves the problem of session information storage in distributed cluster. Offline message storage is designed based on HBase to ensure the reliability of messages.

(2) For the security of system, we designed and implemented the device authentication and ACL topic authentication module. The device authentication and ACL topic authentication services are separated. The authentication services are provided in the way of HTTP API, which can provide flexible configuration of device authentication and topic authentication according to the business needs of the enterprise. We design the Kafka message bridge interface depending on the need of business to decouple messaging system from the business system.

(3) We designed and implemented a decentralized messaging system cluster, and the cluster mode management which is based on Akka to facilitate the monitoring of the running state of ecah mode. Message forwarding between clusters is realized based on routing table, which ensures the accurate message forwarding between clusters. The system monitoring module is designed and implemented to provide the monitoring of the system in the form of Web page. This is convenient for the system administrator to count and monitor on the relevant business information.

Finally, a detailed test for function and performance has been carried out on the system. The results show that the system can complete the basic function of message push, as well as support the business requirements of enterprise customization. The system can support 100,000 connections and 10,000 levels of concurrency, which achieves the expected purpose and has a high value of engineering application.

Keywords: MQTT, IoT, Messaging System, Cluster

目录

缩略词对照表

缩略语	英文全名	中文对照
MQTT	Message Queuing Telemetry Transport	消息队列遥测传输
IOT	Internet of Things	物联网
Qos	Quality of Service	服务质量
NIO	New Input/ Output	新输入/输出
JVM	Java Virtual Machine	Java 虚拟机
ACL	Access Control Lists	访问控制列表
TSL	Thing Specification Language	物描述语言
TCP	Transmission Control Protocol	传输控制协议
HTTP	HyperText Transfer Protocol	超文本传输协议
API	Application Program Interface	应用程序接口
LVS	Linux Virtual Server	Linux 虚拟服务器

第一章 绪论

1.1 课题背景与意义

随着互联网的发展，在市场需求及新技术的驱动下物联网市场规模持续增长，物联网设备连接数逐年递增[1]。报告显示，2019 年全球物联网总连接数达到 120 亿，预计到 2025 年，全球物联网总连接数规模将达到 246 亿，全球物联网市场达到 3.9-11.1 万亿美元。物联网应用场景不断扩展，包括智能制造、智慧城市、智慧医疗、智能办公、智慧农业等[2]。以智能制造应用场景为例，设备管理、生产过程管控、企业运营管理、资源配置协同、产品研发设计是目前的热点应用[3]，然而其数据采集终端与传统的互联网终端相比，具有网络带宽小、处理器性能低、存储资源有限等特点，这就要求其使用相对轻量级、对网络条件要求较低的消息推送协议进行数据的上报、接收等工作。

MQTT 是基于发布/订阅编程模式的消息协议，最早由 IBM 提出，如今已成为 OASIS 规范。由于规范很简单，其发布/订阅模式方便消息在传感器之间传递，允许用户动态创建主题，把传输量降到最低以提高传输效率，提供服务质量管理，因此非常适合需要低功耗和网络带宽有限的 IOT 场景[4,5]。使用 MQTT 协议，设备可以很方便地连接到物联网云服务，也方便存储数据，最后应用到各种物联网业务场景。

基于 MQTT 协议的消息推送系统作为云平台服务的基础组件和与业务紧密联系的重要组成部分，能够支持安全可靠的设备连接通信能力，向下连接海量设备、支撑设备数据采集上云；向上提供云端 API，服务端通过调用云端 API 将指令下发至设备端，实现远程控制；支持各种物联网应用场景，能够满足不同用户的需求，为行业开发者赋能。

本文来源于国内某互联网公司物联网服务运营平台项目，该平台主要面向智能制造、智慧办公、智慧园区、智慧农业等物联网场景，通过工业以太网、工业光纤网络、工业总线、3G/4G、NB-IOT、MQTT、ZigBee 等各类有线和无线通信技术，接入各种工业现场设备、智能产品/装备采集设备数据，方便物联网应用开发，为用户提供通用服务能力，如可视化能力、通常设备监控能力等。

本文主要研究内容为基于 MQTT 协议的物联网消息系统的设计与实现。消息系统作为物联网服务运营平台的子系统，主要负责设备接入、设备数据采集，为上层物联网应用提供数据支撑。在实际的生产过程中，不仅仅

是将物联网设备联网，获取设备上报的数据，更需要对设备的上下线及工作情况的实时监控、挖掘数据背后隐藏的价值更好地有利于生产、支持消息系统和业务应用之间数据共享、根据业务规则引擎实现多种数据流转等。

此外，如何实现物联网设备资源的统一描述与管理、支持物联网场景下高并发的连接请求、实现设备的身份认证与权限管理、协同物联网众多服务应用等，都是需要解决的问题。因此有必要开发一套能够完成基本的消息推送功能、支持企业定制化业务需求、提供高性能高可靠的基于 MQTT 协议的物联网消息系统。

1.2 国内外研究现状

国内外近几年关于物联网的研究主要包括物联网系统使用协议及适用场景、MQTT 协议改进、物联网系统实现三个方面。

对物联网系统使用协议的研究主要包括 MQTT、COAP、AMQP 和 XMPP 四种[6]。MQTT 协议简洁、可扩展性强、省流量省电、适合物联网应用场景[7]。它是轻量级的、发布/订阅模式的消息传输协议。COAP 是轻量级的 M2M 协议，它支持请求/响应和资源/观察模式。COAP 主要用于与 HTTP 和 RESTful Web 进行交互，适用于使用统一资源描述符来指示特定资源[8]。AMQP 是一种可靠的、安全的高级消息队列协议，它支持发布/订阅模式，提供可靠的消息顺序和灵活的消息路由[9]。XMPP 是基于 XML 协议的通讯协议，目前主要应用于许多聊天系统中。它的优点是协议成熟、强大、可扩展性强，缺点是协议较复杂、冗余、费流量、费电、部署硬件成本高。

国内外一些学者对以上消息协议的特点和应用场景纷纷进行了深入的研究。Naik[10]等人从协议安全性、报文结构、消息可靠性等方面，对 IoT 系统使用的 MQTT、HTTP、CoAP、AMQP 这四种协议进行了深入的研究，分别指出了协议适用的场景、优缺点，为 IoT 系统选择推送协议提供了参考。Imane S[11]等人对 MQTT 和 CoAP 协议在智慧医疗领域的使用做了分析与比较，提出了根据传输层协议的不同来选择推送协议的观点。并从系统外部和内部两方面介绍了可能存在的安全性问题，指出了系统在安全性方面需要解决的问题。金成明[12]等人深入分析了 CoAP 和 MQTT 协议应用配电物联网的适用性，设计了基于 MQTT 和 CoAP 的配电物联网通信架构，提出了配电物联网信息数据交互方法，方便了信息模型与实时数据的高效传输。Eridani D[13]等人在辣椒作物温室原型中使用 MQTT 作为消息推送协议，从消息推送时延和系统吞吐量两方面在不同网络环境条件下做了详细的对比实验，验证了 MQTT 协议在实际场景中使用的可靠性与稳定性。

此外，对 MQTT 协议的改进也是近年来国内外研究的重点。WuXin 和 LiNing[14]通过对 MQTT 协议报文的改进，解决了原生 MQTT 协议 Retain 机制只能保留最新一条消息的不足，实现了客户端上线之后能够接收全部的离线消息。Sadeq A S[15]等人从系统 Qos 和流量控制对 MQTT 协议进行改进，将消息重要性与原生 Qos 进行关联解决了消息传输的优先级问题，将当前服务端消息处理能力反馈给客户端解决了流量控制问题。Tantitharanukul N[16]等人将 MQTT 协议中 Topic 管理模块作为一个独立系统，方便消息发布者进行主题注册和订阅者订阅感兴趣的内容，为 Topic 管理提供了新的思路。Harsha M S[17]使用 Shodan API 对 MQTT 的安全问题做了详细的测试，指出了协议本身存在的身份验证和鉴权问题，并提出了使用用户名和密码做身份验证和基于 ACL 进行主题鉴权的解决方案。Tanomwong N[18]等人使用自适应数据传播机制解决系统资源浪费问题，并从主题设计、发布/订阅流程等方面对系统的实现做了详细介绍，为 MQTT 系统设计与实现提供了参考。Matic M[19]针对 MQTT 系统单节点难以支撑海量设备问题，设计了系统集群架构并使用水平资源转发算法解决集群间共享订阅问题，为 MQTT 系统集群的设计与实现提供了参考。

随着物联网的发展，MQTT 协议自身不断的完善，国内外一些厂商也纷纷入局物联网，提供相应的物联网消息通讯服务。国内外一些企业也都将 MQTT 协议作为其物联网云平台的接入协议之一，比如百度的智能云天工物联网平台、阿里的阿里云物联网平台、微软的 Azure IOT 等。一些开源的可接入 MQTT 协议的系统实现有 EMQX、Mosquitto、HiveMQ 等。EMQX 是基于 Erlang/OTP 平台开发的开源物联网 MQTT 消息服务器。EMQX 设计目标是实现高可靠、并支持承载海量物联网终端的 MQTT 连接，支持在海量物联网设备间低延时消息路由。开源版可以免费试用，企业版则需要付费使用。Mosquitto 是一款由 C/C++语言编写的开源 MQTT 服务端，Mosquitto 可以通过桥接的方式进行集群，桥接就是靠一个 Mosquitto 实例去做转发，其他的 Broker 可以转发给它，一旦中转 Broker 不可用，整个集群就不能正常服务。HiveMQ 是企业级的 Broker，使用 Java 语言开发，功能非常齐全。它的集群设计基于 Jgroups，持久化的数据都是本地和 Jgroups 同步，具有优秀的缓存和并发访问控制，由于是收费的并没有公开的源码。

通过以上分析可知，基于 MQTT 协议搭建物联网消息系统是可行的，但是需要结合 MQTT 协议特点对消息的安全性、可靠性和系统集群架构等进行优化设计。此外，现有的物联网消息服务难以根据企业需求做功能和性能上的扩展，物联网消息系统的设计与实现仍是有待研究的方向。

1.3 本文主要工作

在深入研究了 MQTT 协议特点和开源的 MQTT 服务端实现之后，本文结合物联网消息系统在功能和性能上的需求，设计与实现了基于 MQTT 协议的物联网消息系统。本文主要工作如下：

（1）研究与分析了 MQTT 消息协议的内容与特点，使用 Netty 框架完成协议的解析和处理，并结合协议规范实现了基础的消息推送功能。

（2）解决了物联网设备 Session 存储、离线消息存储和分发、认证鉴权三个难点问题。分别基于内存和 Redis 实现了非持久化和持久化 Session 的存储，完成了分布式集群的 Session 存储。为保证消息的可靠性，设计与实现了基于 HBase 的离线消息存储。提供基于 HTTP API 的设备认证与鉴权方式，保证了接入设备的合法性，能够实现灵活且复杂的验证授权机制。

（3）设计了消息系统集群架构，解决了集群间消息的转发与处理。提供了 Kafka 消息桥接接口，完成了物联网众多服务间数据的按需分发与有效协同。基于 Akka 实现了系统的集群管理，基于路由表实现了集群间消息的精准转发，方便系统进行水平上的扩展。

1.4 论文工作安排

本文分 7 个章节对基于 MQTT 协议的物联网消息系统进行设计与实现，具体的章节安排如下：

第一章：绪论。主要包括课题背景与意义、国内外研究现状、本文主要工作和论文工作安排。

第二章：相关概念与技术。对本文涉及到的 MQTT 协议相关概念和系统开发过程中使用到的 Netty、Redis 数据库、Kafka 消息中间件、HBase 数据库和 Akka 这些技术进行介绍。

第三章：系统需求分析。从系统功能性和非功能性两方面对系统进行详细的需求分析。

第四章：系统设计。主要介绍了系统整体结构设计、各功能模块设计和存储设计。

第五章：系统实现。通过类图、时序图、流程图、系统截图等形式，详细地描述了系统各功能模块的实现过程。

第六章：系统测试。介绍了系统的测试环境和测试工具，并从基础功能和性能两方面对系统进行测试，给出了详细的测试结果与分析。

第七章：总结与展望。总结本文所做工作，提出还可以继续研究的方向。

第二章 相关概念与技术

本文使用 MQTT 作为消息推送协议、Netty 搭建网络通信模块、Redis 存储会话和其他系统数据、Kafka 进行消息桥接、HBase 存储离线数据、Akka 管理集群。本章对以上概念与技术进行分析与介绍。

2.1 MQTT 协议介绍

2.1.1 MQTT 协议基础概念

MQTT 全称为 Message Queuing Telemetry Transport（消息队列遥测传输），是一种基于发布/订阅模式的轻量级物联网消息传输协议。MQTT 近年来以简单、易实现、支持 QoS、轻量且省带宽等众多特性逐渐成为了 IoT 通讯的标准。MQTT 作为一种低开销、低带宽占用的即时通迅协议，可以用极少的代码和带宽为物联网设备提供实时可靠的消息服务，它适用于硬件资源有限的设备和宽带有限的网络环境[20]。因此，MQTT 协议广泛应用于物联网、移动互联网、智能硬件、车联网、电力能源等行业。

首先对本文所涉及到的 MQTT 中的相关概念进行解释。

（1）客户端：使用 MQTT 协议的程序或者设备，它可以连接到服务端、发布应用消息给其它相关的客户端、订阅以请求接受相关的应用消息、取消订阅以移除接受应用消息的请求和关闭连接到服务端的网络连接。

（2）服务端：在发送消息的客户端与已订阅的客户端之间充当中介角色的程序或设备，它可以接受来自客户端的网络连接、接受客户端发布的应用消息、处理客户端的订阅和取消订阅请求、转发应用消息给符合条件的已订阅客户端和关闭来自客户端的网络连接。

（3）会话：每个客户端与服务器建立连接后就是一个会话（Session），客户端和服务器之间有状态交互。会话可以存在于一个网络连接之间，也可以跨越多个连续的网络连接存在。

（4）订阅：订阅包含一个主题过滤器（Topic Filter）和一个最大的服务质量（QoS）等级。订阅与单个会话关联。会话可以包含多于一个的订阅。会话的每个订阅都有一个不同的主题过滤器。

（5）主题名：附加在应用消息上的一个标签，被用于匹配服务端已存在的订阅。服务端会向所有匹配订阅的客户端发送此应用消息。

（6）载荷：对于 PUBLISH 报文来说载荷就是业务消息，它可以是任意格式（二进制、十六进制、普通字符串、JSON 字符串、Base64）的数据。

（7）消息服务质量：MQTT 协议提供了 3 种消息服务质量等级，它保证了在不同的网络环境下消息传递的可靠性。QoS0 是发送者（可能是 Publisher 或者 Broker）发送一条消息之后，就不再关心它有没有发送到对方，也不设置任何重发机制。QoS1 包含了简单的重发机制，发布者发送消息之后等待接收者的 ACK，如果没收到 ACK 则重新发送消息。这种模式能保证消息至少能到达一次，但无法保证消息重复。QoS2 设计了略微复杂的重发和重复消息发现机制，保证消息到达对方并且严格只到达一次。

（8）清除会话：MQTT 客户端向服务器发起 CONNECT 请求时，可以通过 CleanSession 标志设置是否创建全新的会话。CleanSession 设置为 0 时。如果存在一个关联此客户标识符的会话，服务端必须基于此会话的状态恢复与客户端的通信。如果不存在任何关联此客户标识符的会话，服务端必须创建一个新的会话。CleanSession 设置为 1，客户端和服务端必须丢弃任何已存在的会话，并开始一个新的会话。

（9）保活心跳：MQTT 客户端向服务器发起 CONNECT 请求时，通过 KeepAlive 参数设置保活周期。客户端在无报文发送时，按 KeepAlive 周期定时发送 2 字节的 PINGREQ 心跳报文，服务端收到 PINGREQ 报文后，回复 2 字节的 PINGRESP 报文。服务端在 1.5 个心跳周期内，既没有收到客户端发布订阅报文，也没有收到 PINGREQ 心跳报文时，将断开客户端连接。

（10）保留消息：MQTT 客户端向服务器发布消息时，可以设置保留消息标志。保留消息会驻留在消息服务器，后来的订阅者订阅主题时可以接收到最新一条的保留消息。

（11）遗嘱消息：MQTT 客户端向服务端发送 CONNECT 请求时，可以携带遗嘱消息。MQTT 客户端异常下线时（客户端断开前未向服务器发送 DISCONNECT 消息)，MQTT 消息服务器会发布遗嘱消息。

2.1.2 MQTT 报文介绍

MQTT 报文由三部分组成，分别为固定报头（Fixed header）、可变报头（Variable header）以及有效载荷（Payload）。包含报文类型等字段的固定包头存在于所有 MQTT 报文中。可变报头的内容根据报文类型的不同而不同，一些报文中甚至不存在可变报头。有效载荷通常是与业务场景相关的数据，例如对 PUBLISH 报文来说有效载荷就是应用消息，对 SUBSCRIBE 报文来说有效载荷就是订阅列表。

MQTT 主要消息类型有 CONNECT/CONNACK、PUBLISH/PUBACK、SUBSCRIBE/SUBACK、UNSUBSCRIBE/UNSUBACK、PINGREQ/PINGRESP 和 DISCONNECT 这 11 种。其中 PINGREQ/PINGRESP 和 DISCONNECT 报文是

不需要可变头部的，也没有 Payload，也就是说它们的报文大小仅仅消耗 2 个字节。在 CONNECT 报文的可变长度头部里，有个 Protocol Version 的字段。为了节省空间，只有一个字节。所以版本号不是按照字符串"3.1.1"存放的，而是使用数字 4 来表示 3.1.1 版本。

2.1.3 MQTT 发布订阅模型

基于发布/订阅模式的 MQTT 协议中有三种角色：发布者、代理和订阅者。发布者向代理发布消息，代理向订阅者转发这些消息。通常情况下，客户端的角色是发布者和订阅者，服务器的角色是代理，但实际上，服务器也可能主动发布消息或者订阅主题。MQTT 协议发布订阅模式如图 2-1 所示。

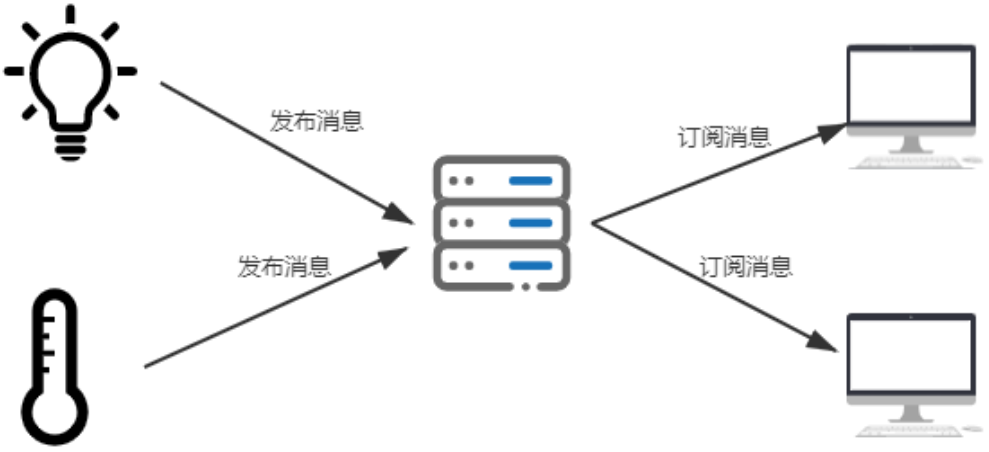

图 2-1 MQTT 发布订阅模式

2.2 相关技术简介

2.2.1 Netty

Netty 是基于 Java NIO 的网络应用框架，使用 Netty 可以快速开发网络应用，例如服务器和客户端。Netty 提供了一种新的方式来开发网络应用程序，这种新的方式使它很容易使用和具有很强的扩展性。Netty 的内部实现是很复杂的，但是 Netty 提供了简单易用的 API 从网络处理代码中解耦业务逻辑。Netty 是完全基于 NIO 实现的，所以整个 Netty 都是异步的。网络应用程序通常需要有较高的可扩展性，无论是 Netty 还是其他的基于 Java NIO 的框架，都会提供可扩展性的解决方案[21]。

Netty 的核心是支持零拷贝的 Bytebuf 缓冲对象、通用通信 API 和可扩展的事件模型。它支持多种传输服务并且支持 HTTP、Protobuf、二进制、文本、WebSocket 等一系列常见协议，也支持自定义协议。Netty 的模型是基于 Reactor

多线程模型，其中 mainReactor 用于接收客户端请求并转发给 subReactor。subReactor 负责通道的读写请求，非 IO 请求的任务则会直接写入队列，等待处理线程进行处理。Netty 优秀的线程模型结合线程池的使用，可以轻松支持万级的设备并发连接和十万级的设备连接。下面对本文涉及的 Netty 概念进行简要介绍。

（1）Bootstrap、serverBootstrap：Bootstrap 的意思是引导，其主要作用是配置整个 Netty 程序，将各个组件整合起来。serverBootstrap 是服务器端的引导类。Bootstrap 用于连接远程主机，它有一个 eventLoopGroup。serverBootstrap 用于监听本地端口，它有两个 eventLoopGroup。

（2）eventLoop：eventLoop 维护了一个线程和任务队列，支持异步提交执行任务。Netty 为了更好的利用多核 CPU 资源，一般会有多个 eventLoop 同时工作，每个 eventLoop 维护着一个 Selector 实例。

（3）eventLoopGroup：eventLoopGroup 主要是管理 eventLoop 的生命周期，可以将其看作是一个线程池，其内部维护了一组 eventLoop，每个 eventLoop 对应多个 Channel，而一个 Channel 只能对应一个 eventLoop。eventLoopGroup 提供 next 接口，可以从组里面按照一定规则获取其中一个 eventLoop 来处理任务。在 Netty 服务端编程中，一般需要提供两个 eventLoopGroup。例如 bossEventLoopGroup 和 workerEventLoopGroup。

（4）channelPipeLine：是一个包含 channelHandler 的 list，用来设置 channelHandler 的执行顺序。Channel 中包含一个 channelPipeline，channelPipeline 中维护着一个 ChannelHandlerContext 的双向链表。

（5）Channel：Channel 代表一个实体（如一个硬件设备、一个文件、一个网络套接字或者一个能够执行一个或者多个不同的 IO 操作的程序组件）的开放链接，如读操作和写操作。

（6）Futrue、ChannelFuture：Future 提供了另一种在操作完成时通知应用程序的方式。这个对象可以看作是一个异步操作结果的占位符。它将在未来的某个时刻完成，并提供对其结果的访问。Netty 的每一个出栈操作都会返回一个 ChannelFuture。Future 上面可以注册一个监听器，当对应的事件发生后会触发该监听器。

（7）ChannelInitializer：它是一个特殊的 ChannelInboundHandler，当 Channel 注册到 eventLoop 上面时，会对 Channel 进行初始化。

（8）ChannelHandler：ChannelHandler 是一个父接口，ChannelnboundHandler 和 ChannelOutboundHandler 都继承了这个接口，它们分别用来处理入栈和出栈。ChannelHandler 本身并没有提供很多方法，因为这个接口有许多的方法需要实现。

其子类在使用期间根据业务需要进行相应的方法实现。

（9）ChannelHandlerContext：允许与其关联的 ChannelHandler 与它相关联的 ChannlePipeline 和其它 ChannelHandler 来进行交互。它可以通知相同 ChannelPipeline 中的下一个 ChannelHandler，也可以对其所属的 ChannelPipeline 进行动态修改。

2.2.2 Redis

Redis 是开源的，基于内存的数据结构存储，可作为数据库，缓存和消息代理。它能够在普通的计算机上实现十万每秒的读写请求，并且支持丰富的数据结构类型，比如字符串、哈希、列表、集合、有序集合、位图和地理空间索引等。

Redis 从 3.0 版本开始支持集群，集群包括主服务器和从服务器。每个主服务器都有两个用于冗余备份的从服务器，从而保证了在主服务器不工作时服务的高可用性。

Redis 通过提供 DISCARD、EXEC、MULTI 和 WATCH 命令来实现事务处理功能。事务中的所有命令都作为独立操作执行。MULTI 命令用于标志一个事务块的开始。EXEC 命令用于触发事务中所有命令的执行。WATCH 命令用于标记要监视的指定键，以便事务有条件的执行。DISCARD 命令用于刷新事务中之前排序的命令，并将连接状态恢复到正常状态。

2.2.3 Kafka

Kafka 是开源的分布式发布/订阅消息中间件。它最初由 LinkedIn 公司开发，之后成为 Apache 项目的一部分。Kafka 提供分布式、可划分、冗余备份、持久性的日志服务。它主要用于处理活跃的流式数据。Kafka 基于主题进行分区，支持复制、不支持事务。在大数据系统中，常用于数据的汇聚与分发，比如对数据的监控、流式数据处理、日志收集分析等。

Kafka 可以起到降低系统组网复杂度和降低编程复杂度两个作用。Kafka 能够为发布和订阅提供高吞吐量；通过将消息持久化到磁盘可进行消息的持久化操作；易于向外扩展，所有的 producer、broker 和 consumer 都可以有多个，且均为分布式的，无需停机即可扩展集群；消息处理状态由客户端维护，服务端不用管理消费状态，当消费消息失败时能够自动平衡[22]。

Kafka 的整体架构非常简单，是显式分布式架构。在一个 Kafka 消息系统中包括 Producer、Consumer、Zookeeper、Broker 四个组件。

Producer 是消息的生产者或发布者，它将消息通过 Push 方式发送到 Broker。

Consumer 是消息的订阅接收者，它通过订阅相关主题消息从 Broker 端进行主动地拉取。在 Kafka 中消费者可以选择加入消费者组，一个消费者组可包含多

个消费者。同一个消费者组的消费者通过负载均衡的方式拉取消息。根据消息体量的大小，可以灵活地配置消费者组中的消费者数量。这种方式提高了 Kafka 消息系统的消费速度，有利于系统的可扩展。

Broker 端负责消息的接收与分发。在 Broker 中可以存在多个 Topic，Producer 向 Topic 进行消息的发送，Consumer 通过订阅 Topic 进行消息的消费。同一个 Topic 可以分布在多个 Broker 上面，一个 Topic 可以具有多个 Partition 分区，每一个 Partition 分区即为一个消息队列。

Zookeeper 作为整个 Kafka 系统的协调者，负责元数据的存储和协调工作。Broker 和 Consumer 的工作信息通过 Zookeeper 进行维护，Zookeeper 通过订阅关系协调 Broker 和 Consumer 的工作，实现了消息的按需分发与负载均衡。

2.2.4 HBase

HBase 是 bigTable 的开源版本，是 Apache Hadoop 的数据库，是建立在 hdfs 之上，被设计用来提供高可靠性、高性能、列存储、可伸缩、多版本的 Nosql 的分布式数据存储系统。它弥补了 hive 不能低延迟、以及行级别的增删改的缺点。

HBase 依赖于 hdfs 做底层的数据存储；依赖于 MapReduce 做数据计算；依赖于 Zookeeper 做服务协调。HBase 介于 Nosql 和 RDBMS 之间，通过主键和主键的范围来检索数据；查询数据功能简单，不支持 join 等复杂操作；不支持复杂的事务，只支持行级事务；主要用来存储结构化和半结构化的松散数据；无模式，每行都有一个可排序的主键和多个任意的列，列可以根据需要动态的增加[23]。

HBase 中有几个核心的概念：

（1）行键（Rowkey）：与 Nosql 数据库一样，Rowkey 是用于检索记录的主键，行键可以是任意字符串。在 HBase 内部，Rowkey 保存为字节数组。HBase 会对表中的数据按照 Rowkey 排序。访问 HBase 表中的行只有三种方式：通过单个 Rowkey 访问、通过 Rowkey 的范围和全表扫描。

（2）列簇：HBase 表中的每一个列，都归属于某个列簇。列簇是表的约束的一部分，必须在使用表之前定义好，而且定义好了之后不能更改。列名都是以列簇为前缀，访问控制、磁盘和内存的使用统计等都是在列簇层面进行的。列簇越多，在取一行数据时所要参与 IO、搜寻的文件就越多。

（3）时间戳：在 HBase 中通过 Rowkey 和列簇确定的为一个存储单元称为一个 cell。每一个 cell 都保存着同一份数据的多个版本，版本通过时间戳来索引。时间戳的类型是 64 位整型。时间戳可以在数据写入时自动赋值，此时时间戳是精确到毫秒的当前系统时间。每个 cell 中，不同版本的数据按照时间倒序排序，即最新的数据排在最前面。为了避免数据存在过多版本造成的管理负担，HBase

提供了两种数据版本的回收方式：保存数据的最后几个版本和保存最近一段时间内的版本。

（4）分区：分区的概念和关系型数据库的分区或者分片类似。HBase 会将一个大表的数据按照 Rowkey 的不同范围分配到不同的分区中，每个分区负责一定范围的数据访问和存储。即使是一张数据量非常大的表，由于被划分到不同的分区中，其访问延迟也非常低。

2.2.5 Akka

Akka 是一款高性能、高容错性的分布式和并行应用框架，它遵循 Apache 开源许可，底层通过 JVM 上另外一个流行的语言 Scala 实现。它基于经典的 Actor 并发模型，即所有的消息都是基于 Actor 组件进行传递，拥有如下特点：

（1）并行与并发：提供对并行与并发的高度抽象。

（2）异步与阻塞：Akka-Actor 消息通信都是基于异步非阻塞。

（3）高容错性：为跨多 JVM 的分布式模型提供强劲的容错处理，号称永不宕机。

（4）持久化：Actor 携带的状态或者消息可以被持久化，以便在 JVM 崩溃后能恢复状态。

（5）轻量级：每个 Actor 大约只占 300bytes，即 1G 内存可容纳接近 300 万个 Actor。

得益于 Actor 模型的良好实现，Akka 天生拥有分布式能力，操作一个远程 Actor 的方式和操作本地 Actor 并没有明显的区别，Akka 基本上已经屏蔽了底层网络通信细节[24]。Akka 的集群功能由 akka-cluster 模块提供，它基于去中心化的 P2P 模型，没有单点故障和单点瓶颈，可以满足多个应用场景[25]。akka-cluster 模块提供了一个可容错的、去中心化的、基于点对点的集群成员服务，解决了单点故障和单点瓶颈问题。集群成员之间使用 Gossip 协议来进行通信，集群的当前状态会被随机的在集群内部传播，集群中的成员有可能看到非最新的版本。

目前 Akka 已经在多家互联网和软件公司广泛使用，比如 eBay、Amazon、VMWare、PayPal、阿里、惠普等，涉及行业包括游戏、金融投资、医疗保健、数据分析等。

2.3 本章小结

本章对消息系统使用的 MQTT 协议和相关技术进行介绍。其中，从协议基础概念、报文格式、发布订阅模式对 MQTT 协议进行简要介绍。从设计架构和基础组件对 Netty 框架进行介绍。从使用场景和相关概念对 Redis、Kafka、HBase、

Akka 进行简要介绍。为消息系统的设计与实现奠定了技术基础。

第三章 系统需求分析

本章对消息系统进行详细的需求分析，既包括功能需求，也包括非功能需求。根据用户角色不同，分别阐述了系统用户在功能上的需求。从时效性、并发性、可扩展性三方面分析了系统在性能上的需求。

3.1 系统需求概述

基于 MQTT 协议的物联网消息系统是公司物联网服务运营平台的一部分，消息系统为整个物联网服务运营平台的基础部分，它为上层应用提供数据支撑。物联网服务运营平台架构如图 3-1 所示。

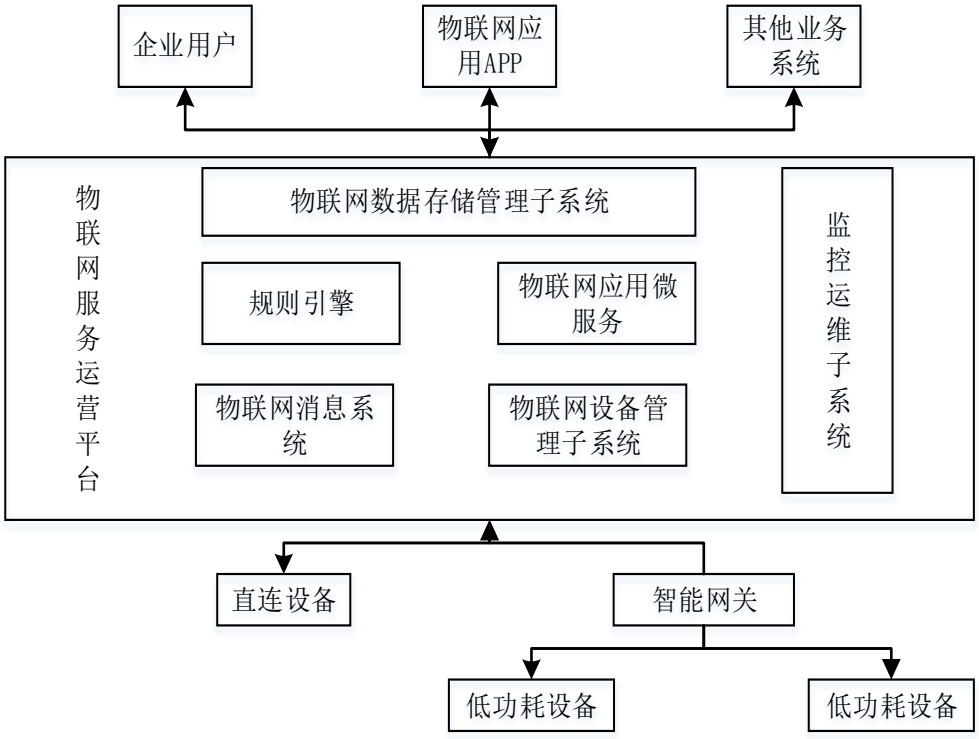

图 3-1 物联网服务运营平台架构

物联网服务运营平台在空间上将物联网体系架构分为三层，分别是物联网感知层、物联网接入层、物联网应用层。

物联网感知层由一些智能的物联网网关、智能设备/装备组成，通过物联网感知层能够采集设备运行数据、监控设备运行状态、控制物联网设备。物联网接入层提供设备管理服务、设备数据解析与存储服务、提供规则引擎进行数据流转服务，为物联网应用层提供基础的数据支撑。接入层众多的物联网微服务之间通过 HTTP API、RPC 等方式实现服务间的协同配合。

物联网应用层则根据相关需要使用接入层提供的数据进行相应的物联网服务应用的开发。物联网应用层包括企业级用户接入物联网相关数据、物联网应用 APP、其他业务应用系统。

消息系统的主要业务需求是能够支撑智能制造、智慧办公、智慧园区、智慧农业等物联网应用场景，通过接入各种工业现场设备、智能产品/装备等物联网设备，将采集的设备运行数据以 MQTT 协议的方式上报至物联网运营平台。

消息系统提供物联网设备状态管理、设备服务调用、设备监控等通用服务能力，以此支持物联网数据可视化展示、数据存储服务、第三方服务等业务相关应用[26]。

3.2 系统功能性需求分析

消息系统用户可以分为一般用户和系统管理员用户。一般用户包括使用该系统的企业用户、公司开发人员等。一般用户通过配置相关物联网设备，通过 MQTT 协议向消息系统请求建立连接，设备身份信息验证通过后能够将设备接入到系统中。

系统一般用户能够通过控制设备能够进行基本的消息发布/订阅操作、决定上报到消息平台的各种设备参数、通过向设备下发命令实现设备的远程控制与配置。

系统管理员用户具有消息系统使用的最高权限，它是系统的超级用户。系统管理员除了具有一般用户的权限之外，还能够对连接到系统的设备进行权限控制，包括身份信息的配置、发布订阅 ACL 的配置等。

此外，系统管理员需要对集群信息及时把控，主要包括系统集群的负载信息、系统连接数、系统主题等关键业务信息。一般用户和系统管理员用例图分别如图 3-2 和 3-3 所示。

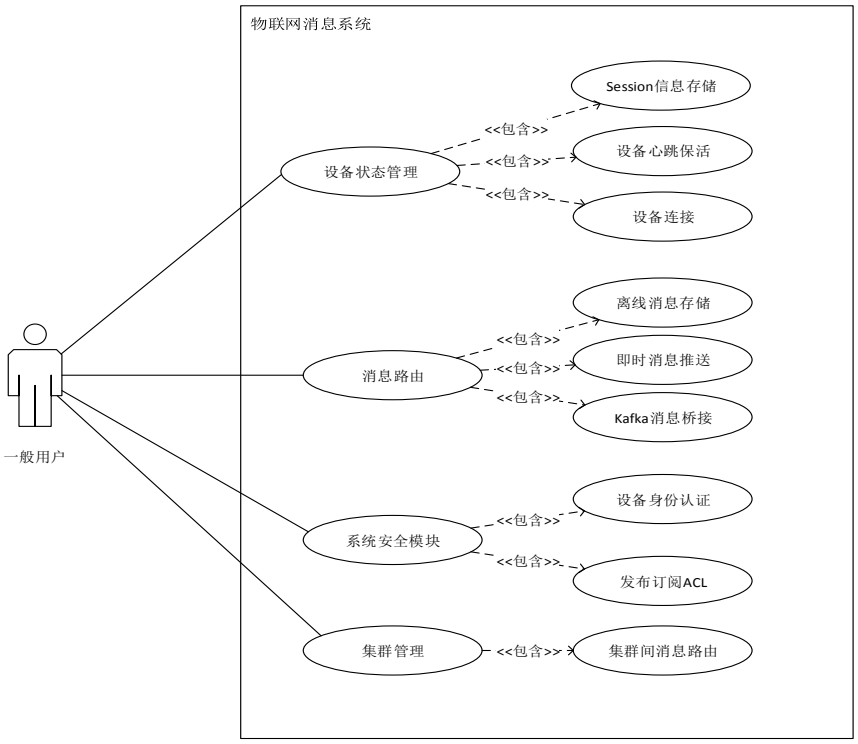

图 3-2 一般用户用例图

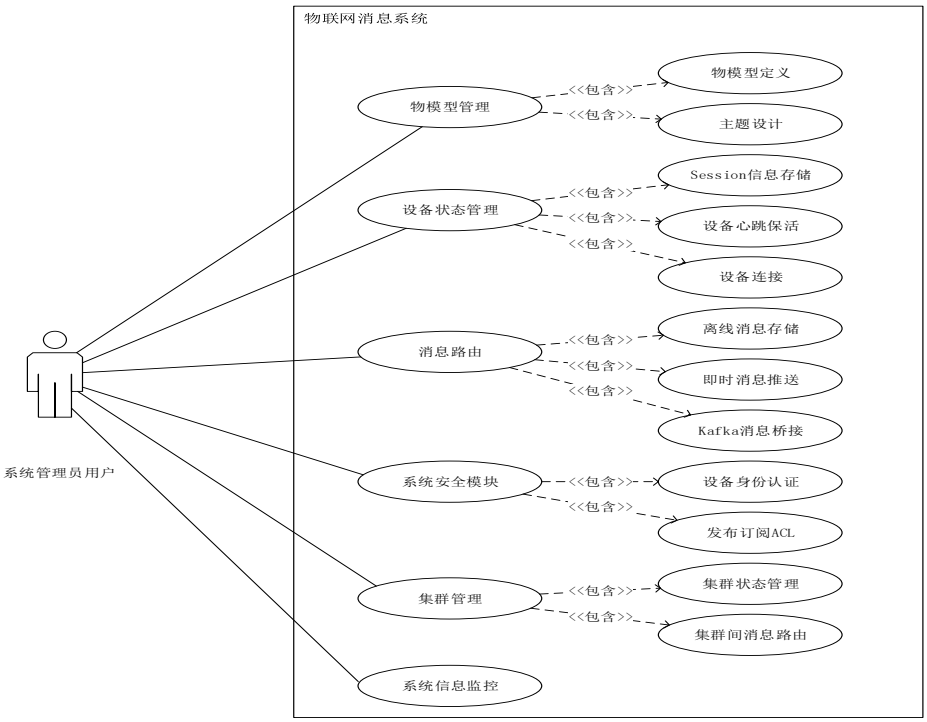

图 3-3 系统管理员用户用例图

系统具体的对功能上的需求主要有以下几点：

（1）物模型管理：物模型管理包括物模型定义和主题设计。通过物模型将物理空间中设备实体数字化，并在云端构建该实体的数据模型，以此描述实体的功能。主题模块根据主题所代表的语义对系统中的主题结构规范进行设计。

（2）设备状态管理：当前，工业互联网平台主要应用于设备管理、生产过程管控、企业运营管理、资源配置系统、产品研发设计及制造与工艺管理等领域。如果生产设备状态不能及时把控，将无法实时掌握设备状态，造成设备检修不及时，影响生产和工作效率。

因此，必须对设备的上线、下线、离线进行监控，及时地掌握设备运行的状态信息。设备成功连接是能够使用后续服务的前提，在连接过程中需要对设备进行认证，拒绝非法的设备连接。设备连接为长连接，为保持连接的有效性，设备每隔一段时间需要发送心跳报文，系统需要接收并反馈响应。设备 Session 存储可以分为持久和非持久两种，系统需要完成这两类 Session 的存储。

（3）消息路由：物联网设备采集的数据或者监控的状态上报至系统之后，会汇聚到系统规则引擎中，规则引擎根据预先设定的规则将消息转发至订阅了该主题的设备、相关业务系统、物联网相关业务 APP、数据库、Kafka 消息中间件等地方。对于离线的设备，结合协议规范对需要进行离线消息存储的则需要持久化消息。因此，系统需要提供这些消息路由接口方便消息依据不同的业务需要流转到不同的地方[27]。

（4）系统安全模块：服务器的资源是有限的并且非常宝贵，并且设备连接为长连接，因此必须在设备连接阶段进行设备的身份认证[28]。在物联网智能网关或者其他设备进行连接的时候，服务器需要对设备的身份进行比较合理的认证，根据不同类型的产品进行不同级别的认证方式。能够拒绝非法的设备连接请求，保证设备在连接阶段的合法性[29]。

在消息系统中，消息之间以各自的主题进行区分，不同的设备依据其物模型定义向不同的主题进行发布或者订阅消息。如果不对设备能够读写的主题进行权限鉴别，会使设备接收到或发布不符合其消息格式的数据，这将使设备异常，严重的话会导致设备的损坏。

因此，系统需要对设备能够订阅的主题进行控制，防止设备订阅了其他设备的主题消息以导致数据泄露及越权访问，或者肆意订阅主题导致主题订阅树复杂造成系统响应慢、网络阻塞等后果。此外，系统也需要对设备能够发布的主题消息进行权限控制，防止对系统的恶意攻击或者非系统管理员越权对设备进行控制。综上所述，必须在服务端对设备能够读写的主题进行权限的控制。

（5）集群管理：为保证消息系统的高可用性，系统需要以集群方式对外提供一致服务。在集群工作模式下，集群间的消息发布与消息订阅操作较单点服务器复杂，需要正确处理消息在集群间的流转和订阅信息在各节点的共享。

（6）系统信息监控：监控各个服务节点的运行状态和信息包括两个方面，一方面是监控系统运行过程中主题数量、内存使用情况、连接数、客户端订阅信息这些系统运行数据；另一方面是各服务节点的工作状态，主要监控上线、离线、异常这些服务器运行状态。

因此，需要设计相应的查询接口以及相关 Web 界面来获取系统运行时的信息，以方便系统管理员对各个节点的工作情况、内部运行数据进行监控，及时地进行服务器性能的升级，保证系统能够对外稳定地提供服务，避免因不能及时查看服务运行情况而导致系统异常或者崩溃这些事情的发生。

3.3 系统非功能性需求分析

消息系统除了需要满足功能上的需求外，为保证系统的实用性，给用户良好的体验，需要在性能上作相应约束，具体有以下三个方面。

（1）消息时延。消息系统主要用来传递物联网设备的消息，消息的及时性决定了物联网设备能否及时地接受相关指令从而迅速做出相关操作。比如在窗帘、电灯、空调的控制中，如果不能及时地传递用户下发的指令，会导致设备的响应时间延迟进而影响整个智慧办公场景下的用户体验。所以需要消息系统接收到相关消息之后，就立即进行消息的路由分发，并且响应时间应该在毫秒级别。

（2）并发性。目前公司可接入的设备数量为 5 万左右，并发数在百级。随着业务的增长，设备连接数和并发数逐渐增大，因此需要支持十万级的连接数和万级的并发连接。所以在设计系统架构时需要考虑到系统的并发性能，当设备连接数增长时能够支持高并发，保证系统的稳定性。

（3）可扩展性。整个消息系统以集群方式进行部署，对外提供统一接口。随着业务的扩展或者新需求的到来，系统需要开发新的功能以支持业务需要，因此在设计系统时需要解耦合，给这些可能的扩展点留下相应接口，方便系统的扩展。

3.4 本章小结

本章首先介绍了整个物联网服务运营平台的整体架构，展示了物联网消息系统在整个平台所承担的角色与功能。然后，从一般用户和系统管理员用户两个用户角色展示了系统的功能用例。分别阐述了各个功能模块的具体设计需求。

最后，从时效性、并发性、可扩展性对消息系统在性能上提出具体需求。本章为系统的设计设定了清晰的目标。

第四章 系统设计

本章首先对消息系统整体结构进行了设计，并将系统分为物模型与主题、设备状态管理、消息路由、系统安全、系统集群、系统监控这六个功能模块。其次，分别对各个功能模块进行了详细的设计，并对设备 Session 存储、离线消息存储与分发、认证鉴权这三个关键问题进行研究与设计。系统各个模块在功能上相互独立，通过相关接口相互调用向客户端提供服务。

4.1 系统整体设计

消息系统作为物联网服务运营平台的一部分，主要负责设备的接入与数据采集，然后根据业务需求将这些设备采集的数据存储至数据库、上报到相关业务系统等。整个消息系统以集群方式进行搭建，单点的服务节点都能够完整的支持整个消息的传递，整个集群对外提供一致性服务。结合以上分析可以得知系统网络架构如图 4-1 所示。

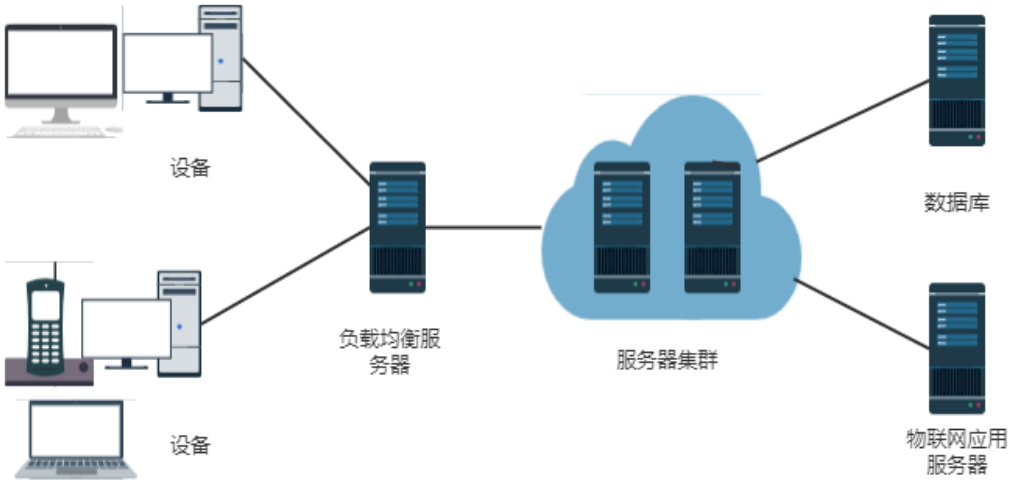

图 4-1 系统网络架构图

每个单点 MQTT 代理服务端由物模型与主题模块、设备状态管理模块、消息路由模块、系统安全模块、系统集群模块、系统监控模块组成，单个 MQTT 代理功能模块图如图 4-2 所示。

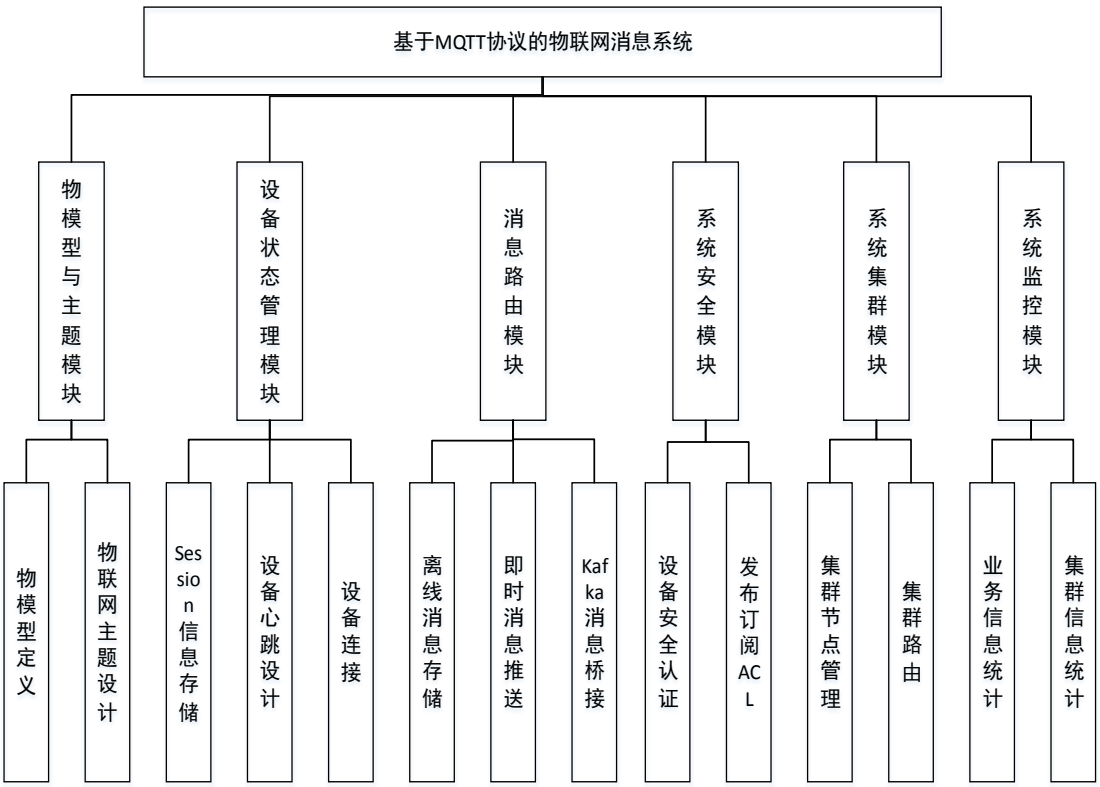

图 4-2 系统功能模块结构图

系统中每个功能模块介绍如下：

（1）物模型与主题模块。物模型是将物理空间中设备实体数字化，并在云端构建该实体的数据模型，以此描述实体的功能。主题模块根据主题所代表的语义对系统中的主题结构规范进行设计。

（2）设备状态管理模块。该模块通过将设备状态分为在线、离线、异常三种，并结合心跳机制进行设备连接的保活，完成对设备状态的管理。并解决了分布式集群中设备 Session 存储这一难点问题。

（3）消息路由模块。消息系统基础的功能就是完成消息的推送，其中包括根据 MQTT 协议完成即时消息推送、设备离线消息的推送和 Kafka 消息转发。

（4）系统安全模块。系统安全模块主要包括两方面，一方面为设备连接时的安全认证，对合法的设备进行连接以完成后续业务操作，禁止非法的设备连接；另一方面为主题鉴权模块，当设备向某一主题发布消息或者订阅某个主题时，首先需要验证该设备是否具有该主题的发布订阅权限，只允许具有权限的设备进行操作。

（5）系统集群模块。系统以集群方式进行搭建，为保证对外提供稳定的服务，需要监控每一个节点的运行情况，当有节点异常时能够及时地反馈信息给系统管理员。因为有不同的设备连接建立在不同的服务节点，当一个消息发布到某一节点时除了该节点上具有订阅了该消息的客户端，其余节点也有可能存在订阅

了该主题消息的客户端，因此需要进行集群间消息的转发。

（6）系统监控模块。系统监控模块主要监控系统运行时内存使用情况、Cpu 占用率和与业务相关的信息比如主题数、连接数、订阅关系等，最后以 Web 页面方式进行展示。

4.2 物模型与主题模块设计

4.2.1 物模型设计

物模型指将物理空间中的物联网设备实体数字化，并在云端构建该实体的数据模型，用于描述设备具体的功能。将物联网设备进行物模型设计方便将不同的设备异构化，在开发时不用关心底层设备的异构性。

物模型可以定义一个独立设备也可以定义由每个独立设备组成的复合设备。一个温度传感器、一个空调的开关这些具有单一功能的设备为独立设备。在物联网环境中有一类设备是由许多独立的设备组成，比如物联网网关。它能够采集多种物联网设备上报的数据，通过相关接口接入到系统中，集中地上报数据到物联网系统中，这一类设备称之为复合设备。

物联网消息系统在描述这些设备时采用物模型来定义设备，物模型 TSL（Thing Specification Language）是一个 JSON 格式的文件。它是物理空间中的实体，如传感器、车载装置、楼宇、工厂等在云端的数字化表示，从属性、服务和事件三个维度，分别描述了该实体是什么、能做什么、可以对外提供哪些信息。定义了物模型的这三个维度，即完成了设备功能的定义。

属性（Property）一般用于描述设备运行时的状态，如环境监测设备所读取的当前环境温度等。属性支持 GET 和 SET 请求方式。应用系统可发起对属性的读取和设置请求。服务（Service）表示设备可被外部调用的能力或方法，可设置输入参数和输出参数。相比于属性，服务可通过一条指令实现更复杂的业务逻辑，如执行某项特定的任务。事件（Event）用来描述设备运行时的事件。事件一般包含需要被外部感知和处理的通知信息，可包含多个输出参数。如某项任务完成的信息，或者设备发生故障或告警时的温度等，事件可以被订阅和推送。

4.2.2 主题设计

主题是一个 UTF-8 字符串，服务端用它来过滤每个连接的设备消息。主题由一个或多个主题级别组成。每个主题级别之间由正斜杠分割，比如 device/firstfloor/room/temp。与消息队列相比，主题非常轻量级。设备不需要在发布或订阅之前创建所需的主题，服务节点接受每个有效主题时不需要进行任何初始化。每个主题必须包含至少一个字节，并且它还可以包含空格。另外，主题是

区分大小写的。除此之外，单独的正斜线也是有效的主题。

当设备订阅主题时，它可以订阅指定的主题消息，或者可以使用通配符同时订阅更多的主题。通配符在设备订阅主题时使用，并且在发布消息时不允许使用。

主题存在两种通配符，单级和多级通配符。单个级别的通配符代表一个主题级别，单个级别的通配符以加号来表示。除了通配符以外的任意字符串所构成的主题级别，可以与由单级通配符构成的主题级别相匹配。单级通配符仅涵盖一个主题级别，而多级通配符可以涵盖任意数量的主题级别。

为了确定匹配的主题，需要多级通配符总是为主题中的最后一个字符，并且确保它前面是正斜杠，订阅具有多级通配符的主题的设备将接收所有的消息。在设计主题时，一个符合物联网应用的主题应具有以下特点：

（1）能够允许设备以主题来进行发布或订阅操作，具有良好的区分性，避免不同设备间主题的混淆。

（2）能够体现消息来源于哪个设备，比如来自哪个产品、网关或者智能设备。

（3）能够体现该消息传递的信息。比如代表设备属性的上报、服务的调用或者设备告警事件等。

4.3 设备状态管理模块设计

4.3.1 设备连接设计

物联网设备与系统建立可靠稳定的长连接是后续进行订阅、推送、取消订阅等业务请求处理和使用系统其它服务的基础。当设备向系统请求建立合法的连接时，系统会先验证 MQTT 协议的版本信息，然后通过设备认证接口验证设备身份信息防止非法的设备连接。

当系统中已经有相同的 clientid 会话信息的时候，会先将该 clientid 对应的连接断开并且及时清理之前的会话信息并释放服务器资源，然后建立当前请求的连接。如果该设备有离线消息，那么将会推送离线消息。设备连接系统流程图如图 4-3 所示。

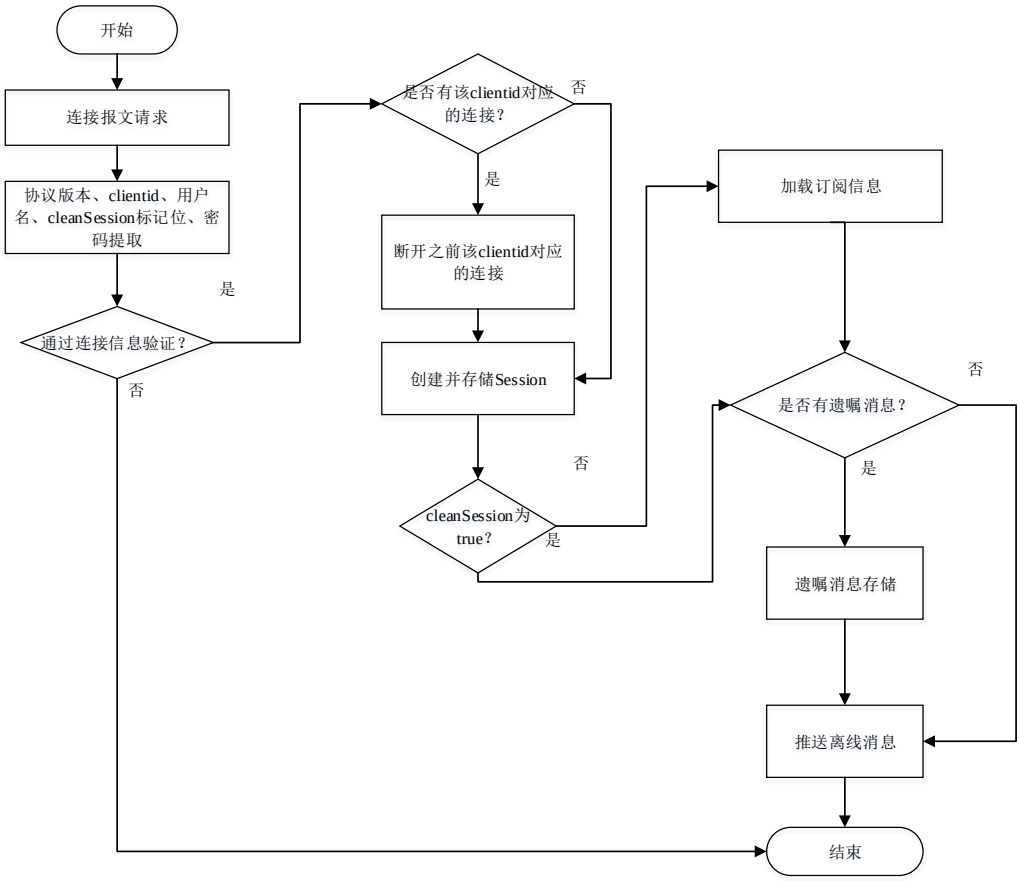

图 4-3 设备建立连接流程

4.3.2 Session 信息存储设计

服务器为每个客户端创建一个会话用来存储连接信息，单节点服务器环境下，服务器实例只需要维护建立在自身的长连接，而在集群环境下处理设备下次发起的连接的服务器节点可能与上次的服务器节点不是同一个，因此会导致系统的 Session 信息不同步，存在 Session 信息不一致的情况。所以在设计 Session 信息存储模块的时候需要实现 Session 信息的共享。

常见的 Session 存储主要有利用客户端记录 Cookie、服务器集群间会话信息的主从同步、会话信息持久化到后台数据库和利用 Redis 缓存会话信息。

利用客户端记录 Session 时，Session 记录在客户端，每次请求服务器的时候，将 Session 放在请求中发送给服务器，服务器处理完请求后再将修改后的 Session 响应给客户端。利用 Cookie 记录 Session 受 Cookie 大小的限制，能记录的信息有限，每次请求需要传递 Cookie，占用带宽影响性能。这种方式不适用于物联网设备自身能够使用的网络和存储资源非常有限的环境下。

服务器集群间会话信息的主从同步适用于集群为主从架构，主服务器用于处理客户端的连接请求，然后将连接会话信息通过后台进程同步给其他从服务器节点。这种方式系统中每个节点都需要维护全部的连接信息，需要占用较多的内存

资源，对服务器内存有较高的存储要求[30]。

会话数据持久化到后台数据库是将服务节点里的所有会话信息存储到专门用于存储 Session 信息的数据库中，保证 Session 的持久化。好处在于服务器出现问题，Session 不会丢失；缺点为如果访问量很大，会给数据库造成很大压力，还需要增加额外的开销来维护数据库。

利用 Redis 缓存会话信息是利用 Redis 统一缓存会话信息，因其高可用集群和内存读写的优势，存储性能要比数据库好。缺点为需要进行一次网络读写，额外增加系统响应的时间[31]。

结合 Redis 存储会话数据的优势以及系统的开发也需要借助 Redis 存储数据，选择了基于 Redis 设计 Session 信息的存储架构。每个客户端连接系统所使用的客户端标识是唯一的，系统以这个标识区别每个连接，在每个服务器内存中存储设备标识与连接信息的对应关系，其中连接信息包括 MQTT 协议的 clean session 标志位、Netty 中上下文信息等。Redis 中保存设备唯一标识与会话数据的对应关系，包括当前连接的订阅信息、服务器节点信息等[32]。基于 Redis 的 Session 信息存储设计如图 4-4 所示。

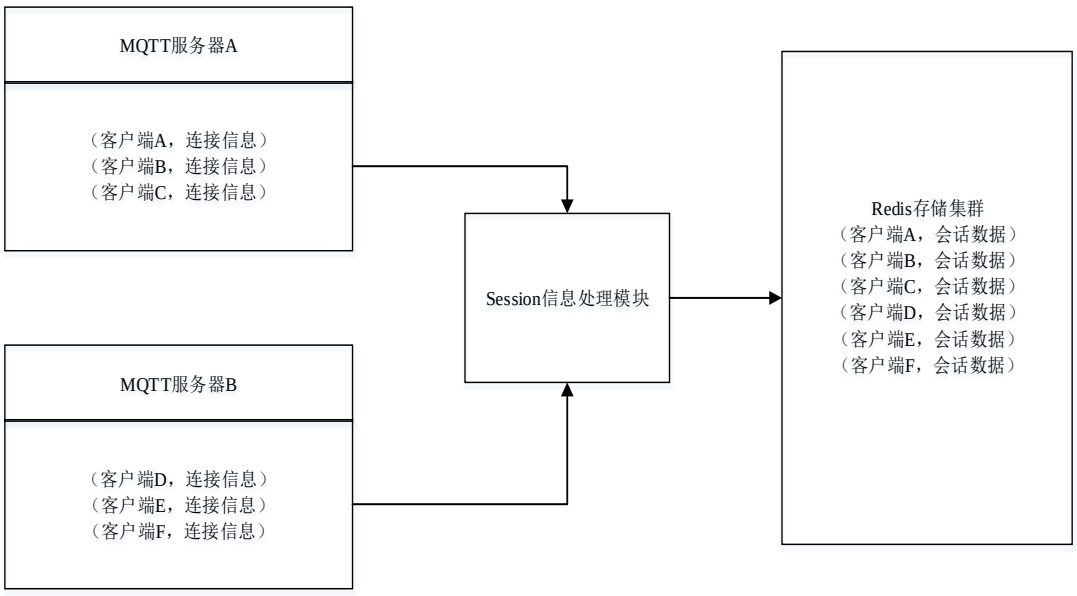

图 4-4 基于 Redis 的分布式 Session 存储设计

在以上的 Session 架构设计中，每个服务器实例在内存中保存了客户端标识与连接信息的对应关系，系统通过连接信息找到对应的长连接通道进行数据的收发。Redis 存储集群存储了客户端与会话数据的关系，当设备再次连接时系统会请求该设备对应的客户端标识是否存在之前的连接，进一步判断是否清除之前会话信息，重发离线消息等操作。并且利用 Redis 的键值过期时间能够及时地清理过期的会话数据，释放服务器资源。

4.3.3 设备心跳设计

客户端与服务端的长连接维持是靠心跳机制实现的，客户端连接时需要指定心跳时间，间隔一般为五分钟之内，本系统设置的时间间隔为 180s。成功连接系统之后客户端会根据设置的心跳间隔时间主动地向服务器发起心跳请求，服务端则通过会话通道响应请求。

服务器在 1.5 倍的心跳周期内接收不到客户端发送的 PINGREQ，可考虑关闭客户端的连接描述符。此时的关闭连接的行为和接收到客户端发送 DISCONNECT 消息的处理行为一致，但对客户端的订阅不会产生影响，不会清除客户端订阅数据。设备心跳设计如图 4-5 所示。

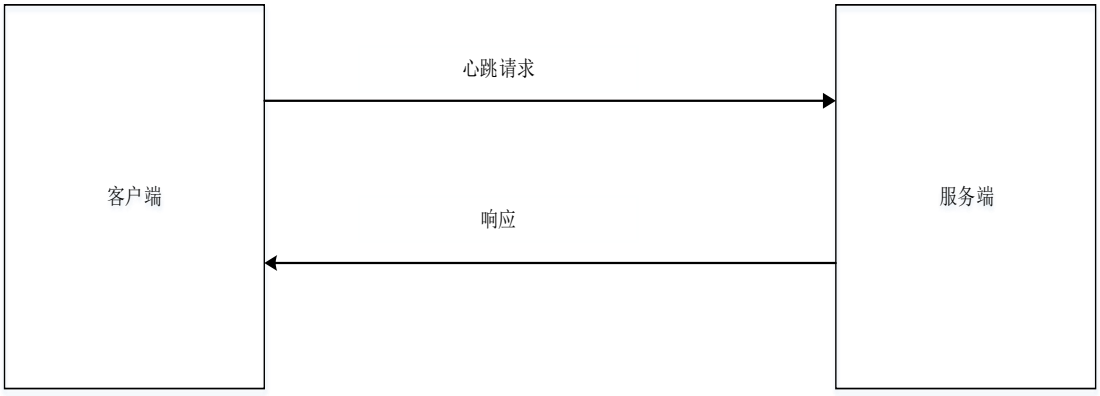

图 4-5 设备心跳设计图

4.4 消息路由模块设计

4.4.1 离线消息存储设计

物联网因其不稳定的网络环境，设备会因为网络波动而异常掉线，在掉线期间可能会有设备订阅的主题消息发布，为了保证离线设备重新连接之后能够接收到离线时的消息，系统需要存储设备订阅的 Qos 大于 0 且 cleanSession 标记位为 false 的离线主题消息。当网络恢复设备正常连接时，能够将离线时未能监听到的消息推送给设备。

物联网环境下设备消息实时上报，离线消息体量较大，如果基于内存存储会耗费较多内存资源。此外，对每个需要进行离线消息存储的设备来说，服务端需要建立与其对应的离线消息队列，这样会使得离线消息队列数量非常多。现有的消息队列服务并不能满足这种需求，因此考虑采用 HBase 对离线消息的存储进行设计与实现。

HBase 本身不提供消息队列的功能，但可以利用 HBase 的特性来实现虚拟队列的概念。如图 4-6 所示，有 4 个客户端，每个客户端对应一个虚拟队列。为每个客户端分配一个唯一的队列 ID，这样每个队列可以用 QueueID 和单调递增 ID

来组合成一个唯一的 Rowkey。同时为了保证写入的均匀性，避免数据热点问题，结合 Rowkey 设计优化原理，进行设计合理的唯一 ID 前缀来将这些 Rowkey 均匀地分布到不同的 Region[33]。为保证写入消息的有序性，需要为每个设备的每条离线消息分配唯一的 ID，比如 sitech_1024 表示设备 sitech 的第 1024 条消息。

消息在写入时以批量方式进行，首先需要对每个队列进行加锁操作，然后分配每条消息的 ID，再将离线消息写入 HBase，最后释放每个队列的锁。这种方式能够保证多个服务节点在并发地向一个设备离线消息队列写入消息时不会发生数据冲突[34]。

在读取离线消息时，通过 HBase 的 scan 操作来获取队列最小最大 ID，可以将其缓存在 Cache 中。每次读取一定长度的消息数据，确保一次读取的数据量不会太大。因为任何时刻的读取请求只可能来自一个设备，所以不需要在读取时进行加锁操作。对于已经推送的离线消息，需要及时地删除掉避免消息的重复发送。

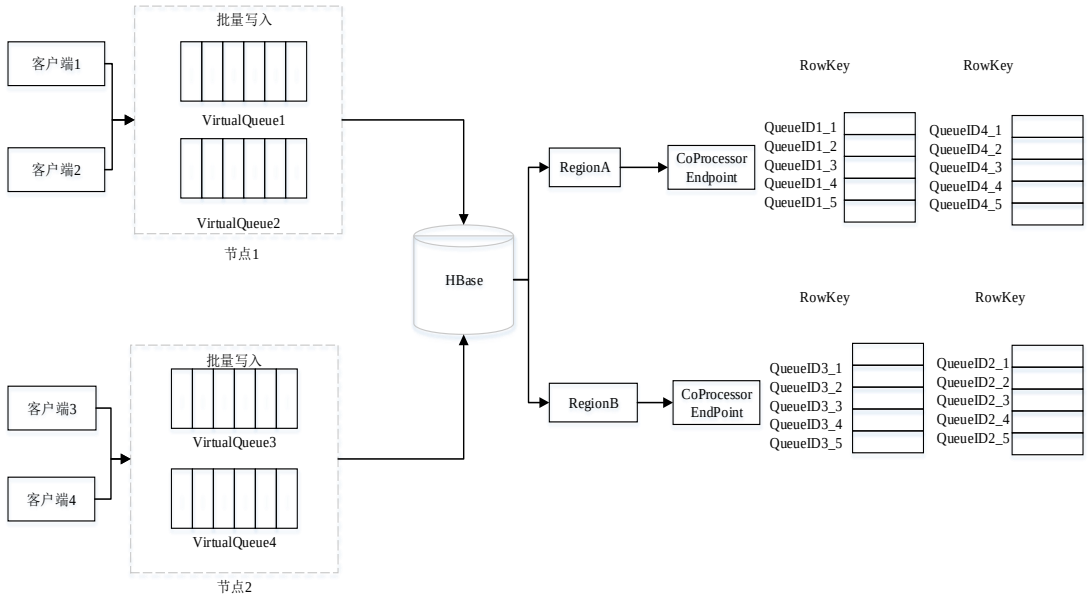

图 4-6 基于 HBase 的离线消息存储设计

4.4.2 即时消息推送设计

消息推送功能是消息系统的核心功能，消息推送模块需要正确处理设备的消息推送与接收[35]。MQTT 协议本身提供了三种消息服务质量分别为 Qos0、Qos1 和 Qos2，它保证了在不同的网络环境下消息传递的可靠性，Qos 的设计也是 MQTT 协议里的重点。

当 Qos 为 0 时，消息的分发依赖于底层网络的能力。发布者只会发布一次消息，接收者不会应答消息，发布者也不会存储和重发消息。消息在这个等级下的传输效率最高，但也有可能送达一次也可能根本没有发送成功。这种消息质量的消息一般应用于可以接受偶尔的消息丢失，设备与系统之间网络环境比较好的

场景下。

当 Qos 为 1 时，可以保证消息至少送达一次。消息系统通过简单的 ACK 机制来保证 Qos1。此时设备发送消息，会等待系统 PUBACK 报文的应答，如果在规定的时间内没有接收到 PUBACK 的应答，设备将会将消息的 DUP 置为 1 并重发消息。设备端接收到 Qos 为 1 的消息时需要回应 PUBACK 报文，设备接收端可能会多次接收到同一个消息，无论 DUP 标志如何设置，消息接收者都会把接收到的消息作为一个新的消息并发送 PUBACK 报文应答。这种消息质量一般应用在希望消息性能最优、消息接收者能够去重的场景下，同时也是物联网环境下使用最多的消息质量。

当 Qos 为 2 时，发布者和订阅者通过两次会话来保证消息只被传递一次，这是 MQTT 协议里最高的消息质量等级，消息的丢失和重复都是不可接受的，同时使用这个等级服务端会有额外的开销。当设备端发布 Qos 为 2 的消息时，会将发布的消息存储起来并等待接收者回复 PUBREC 的消息，发送者收到 PUBREC 之后，它就可以安全地丢弃掉之前发布的消息，因为此时消息已经被接收者成功接收。发布者会保存 PUBREC 消息并应答一个 PUBREL，等待接收者回复 PUBCOMP 消息，当发送者收到 PUBCOMP 消息之后会清空之前所保存的状态。

当接收者收到一条 Qos 为 2 的消息时，它需要处理此消息并返回一条 PUBREC 进行应答。当接收者收到 PUBREL 消息之后，它会丢掉所有已保存的状态，并回复 PUBCOMP。不管消息的发送者是设备还是系统，无论在哪个阶段出现丢包，都需要重新发送上一条的消息，消息接受者需要对收到的每一个消息进行应答。这种等级的使用一般应用在对数据完整性和及时性要求较高的场景下。

消息推送模块处理设备发布消息时，首先需要确定消息往哪个主题发布，然后经过系统安全模块进行主题鉴权成功之后，系统将发布的消息解析封装成系统内部能够识别的消息。

对于 Qos 大于 0 的消息，系统需要先将消息持久化落地到数据库中，再添加至消息发送队列，通过查找主题订阅树匹配出订阅了该主题消息的设备并进行消息的推送。对于 Qos 为 0 的消息则直接推送到消息发送队列，由后台进程进行异步的消息分发。

对于离线的设备，如果 cleanSession 标志位设置为 true，那么系统将丢弃该消息；如果 cleanSession 设置为 false 则需要将消息保存在离线消息队列中去，当设备下一次上线时，由离线消息推送模块推送给设备。Qos 为 2 的消息推送过程最为复杂，Qos 为 0 和 1 的消息推送过程较简单，MQTT 协议 Qos 为 2 的消息推送过程如图 4-7 所示。

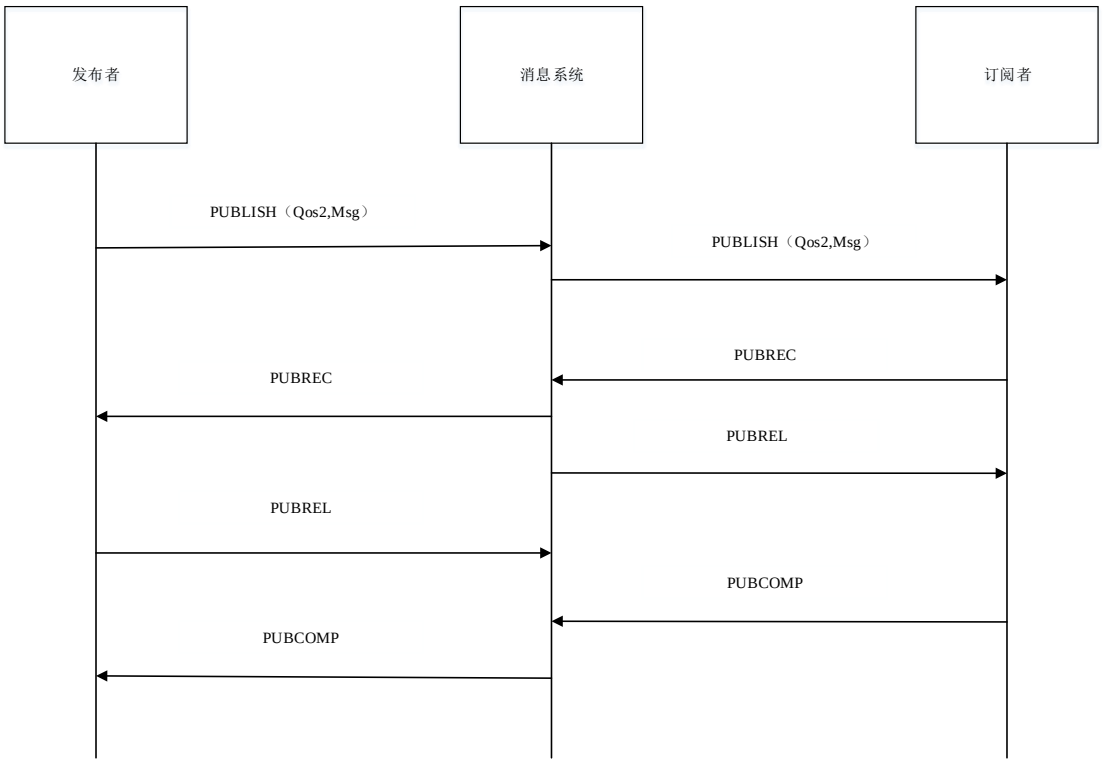

图 4-7 Qos2 消息推送过程

4.4.3 Kafka 消息桥接设计

如何实现设备采集的大规模数据在众多的物联网服务间的按需分发，是一个非常重要的问题。物联网设备上线工作产生的消息不仅仅在设备之间交互，还需要供相关应用系统使用以实现如安全审计、流量计费、设备监控、通知触发等功能，在这种工作模式下能够通过以下消息流转设计完成。

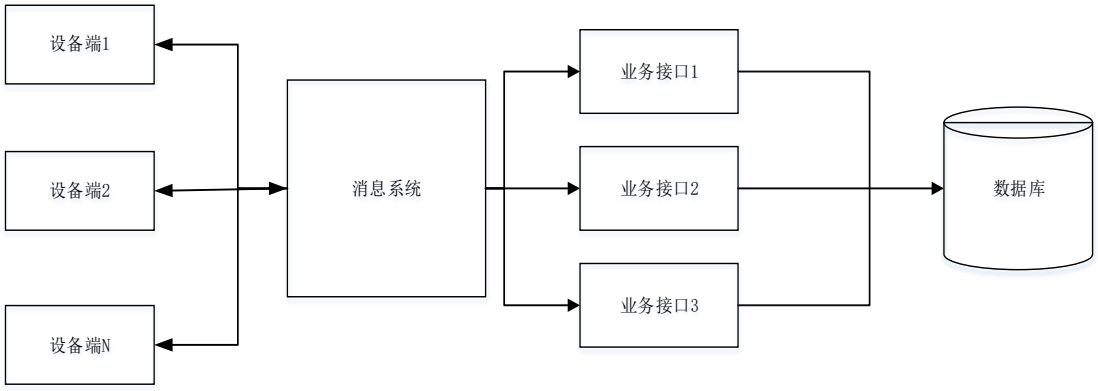

图 4-8 消息流转原型图

物联网业务应用系统通过使用消息系统提供的接口进行数据交互，这种消息流转模式在结构上是紧耦合的。这种工作模式下系统需要支持多个数据处理接口，以保证每个应用系统按照自身的需要从服务端获取数据消息。每一个业务应用系统需要与消息系统建立数据通道，额外的服务器资源开销用在这些通道的创建与

维持，物联网消息同步速度极大的影响系统消息交换的及时性。

随着系统业务的增长，整个系统的结构需要变更以维持新的业务需求。并且每个节点的消息处理速度与时序并不一致，这样当消息体量较大时，部分业务处理会出现阻塞，从而造成消息的丢失、整个系统的稳定性下降等严重后果。这种方式不方便消息系统的扩展，难以支持物联网服务的协同工作，限制了物联网系统的灵活性。

如何解决消息在分布式、松耦合的物联网应用间的按需订阅和共享数据，如何实现多个服务间的协同工作来快速应对设备的动态变化，是消息系统在架构设计上的关键问题。Kafka 基于发布/订阅模式，具有松耦合、快速、分布式、可扩展性强等特点，适用于大规模数据的收集处理。使用 Kafka 能够实现消息系统与众多物联网服务间的解耦，能够进行限流、削峰填谷、队列处理等措施[36]。

比如在智能门锁应用场景下，系统上行、下行消息数据需要供以下三个业务环节使用。消息通知：将开锁状态通知到门锁用户绑定的通知方式；状态监控：分析处理门锁定时上报的状态信息，如果电量、状态异常等需触发告警通知系统管理员；安全审计：分析上下行消息数据，记录用户开锁行为，同时防范下行指令被篡改、重放等方式攻击。

该方案中，系统会将对应主题的消息统一桥接到 Kafka 供物联网服务应用使用，实现业务系统与消息系统的解耦。该方案中使用 Kafka 作为消息服务器与应用程序之间的消息队列与消息总线。消息系统往每个主题队列末尾添加数据，每个物联网服务应用依次读取数据然后进行处理，这种架构设计既兼顾了性能与数据可靠性，又能够有效降低系统复杂度、提升系统扩展性[37]。这种方案原型设计如图 4-9 所示。

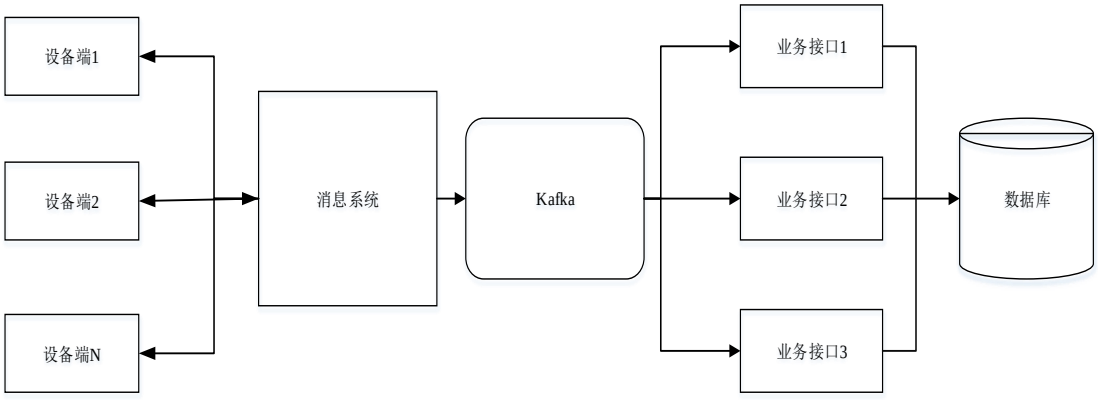

图 4-9 Kafka 消息桥接设计

4.5 系统安全模块设计

4.5.1 设备安全认证设计

MQTT 协议本身不提供安全机制，但是可以使用 CONNECT 报文中用户名和密码字段作为系统安全认证的依据。为确保接入系统的物联网设备是合法的，设备在建立连接时会验证设备的身份信息[38]。设备在连接系统之前需要在物联网设备管理平台注册设备信息（属于物联网平台其他服务不在本文研究范围内），然后获取其独有的设备秘钥。

针对设备身份安全认证，系统提供两种方式的认证：一机一密和一型一密。一机一密认证方法，即为每个设备提供唯一的设备证书，当设备与消息系统建立连接时，消息系统对其携带的设备证书信息进行认证，认证通过，系统激活设备，设备与系统间才可传输数据，一机一密认证方式的安全性较高；一型一密认证方式下，同一产品下所有设备可以使用相同设备证书，设备连接时，消息系统进行身份确认，认证通过，建立设备连接。结合 MQTT 协议规范，系统设备安全认证模块连接状态码如表 4.1 所示。

表 4.1 连接状态码

值	返回码响应	描述
0	0x00	连接已被服务端接受
1	0x01	连接被拒绝：服务端不支持客户端请求的 MQTT 协议级别
2	0x02	连接被拒绝：不合格的客户端标识符
3	0x03	连接被拒绝：MQTT 服务不可用
4	0x04	连接被拒绝：无效的用户名或密码
5	0x05	连接被拒绝：客户端未被授权连接到此服务器

4.5.2 发布订阅 ACL

发布订阅 ACL 指对发布或订阅操作的权限控制，例如拒绝设备 clientA 向 open/sitech/door 发布消息。设备订阅主题、发布消息时主题鉴权模块通过检查目标主题是否在指定数据源允许/禁止列表内来实现对设备的发布、订阅权限管理。

常见的 ACL 数据源主要有三种：配置文件、外部主流数据库和自定义 HTTP API。使用配置文件提供认证数据源，适用于变动较小的 ACL 管理；外部数据库可以存储大量数据、动态管理 ACL，方便与外部设备管理系统集成；HTTP ACL 能够实现复杂的 ACL 管理，满足物联网环境下设备对不同级别的权限需求[39]。ACL 是允许与拒绝条件的集合，消息系统中使用以下规则来描述 ACL：允许或

者拒绝某个客户端发布或者订阅某个主题。其中系统管理员可拥有超级用户身份，超级用户身份拥有最高的发布权限并且不受 ACL 的限制。

认证鉴权启用超级用户功能后，发布订阅时系统将优先检查设备是否为超级用户身份。设备具有超级用户身份时，通过授权并跳过后续 ACL 检查。此外系统 ACL 缓存当设备在命中某条 ACL 规则后，便将其缓存至内存中，以便下次直接使用，设备发布、订阅频率较高的情况下开启 ACL 缓存可以提高 ACL 检查性能。

系统选择 HTTP API 作为 ACL 数据源。使用外部自建的 HTTP 应用认证授权数据源，根据 HTTP API 返回的数据判定授权结果，能够实现灵活、复杂的 ACL 校验逻辑。系统在设备发布、订阅事件中使用当前设备相关信息作为参数，向系统的认证服务发起请求权限，通过返回的 HTTP 响应状态码来处理 ACL 授权请求。无权限则 API 返回 4xx 状态码；授权成功则 API 返回 200 状态码。

物联网设备身份验证和读写主题权限的验证接口，都以 HTTP API 请求方式完成，当设备连接时系统向验证服务发送身份信息进行验证，系统根据返回的状态码来判断是否建立连接；当设备发布或订阅主题消息时，系统会构建 HTTP 请求携带身份信息向主题验证服务鉴权，鉴权通过则可以进行发布订阅，反之，则不允许进行相关操作。

为了提高验证效率，系统对频繁进行权限验证的信息会进行缓存，当设备需要鉴权的时候，先去查询本地缓存是否有相关验证信息，本地缓存无身份信息，再去请求验证服务。整个权限鉴别模块的设计如图 4-10 所示。

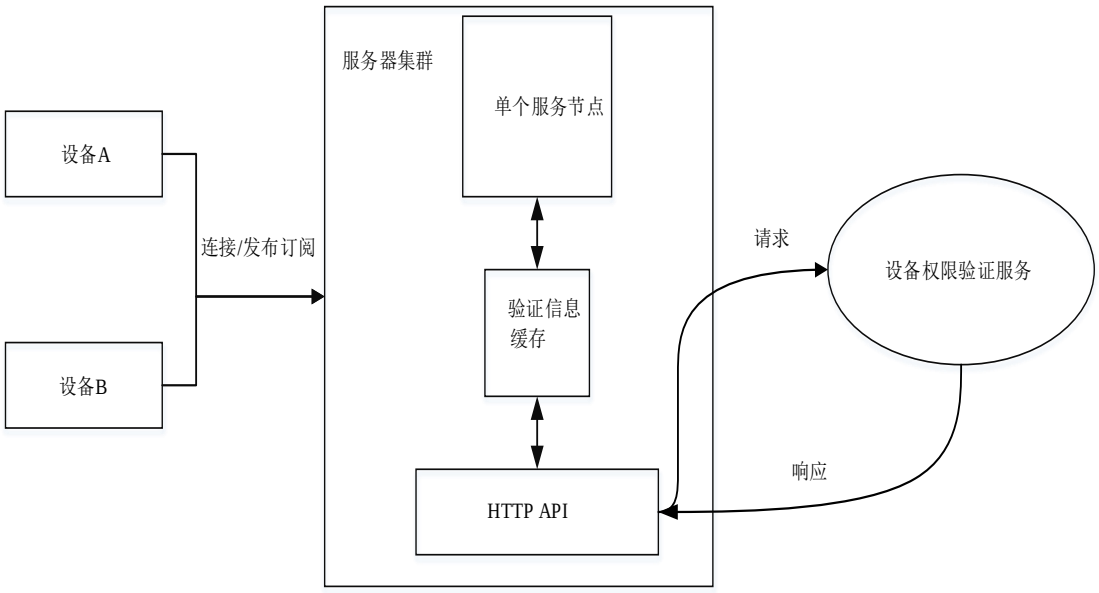

图 4-10 权限验证设计

4.6 系统集群模块设计

4.6.1 负载均衡设计

LVS（Linux Virtual Server）主要用于服务器集群间的负载均衡，它工作在网络层，可以实现高性能、高可用的负载均衡技术。消息系统依靠搭建 LVS 集群来实现负载均衡，将大量的设备连接请求按照相关负载均衡策略分配到各个服务节点[40]。

LVS 可以分为三个部分：

（1）负载均衡层。它是 LVS 的核心部分，负责将收到的请求按照相应的算法分发到后方的服务器，不做具体的业务处理。能够监控后方服务器的状态，自动剔除不能服务的机器。

（2）服务器集群。该层负责具体的业务逻辑处理。接收由负载均衡层转发的请求，并做相应业务处理。

（3）共享存储。该层主要是为上一层提供数据并保持一致。

LVS 通过控制 IP 来实现负载均衡，由 IPVS 模块负责具体实现。IPVS 有三种实现机制：VS/NAT、VS/TUN 和 VS/DR。消息系统选择 VS/DR 的工作模式，它通过将请求中的 MAC 地址进行改写，然后发送到真实的服务器处理，服务器处理后直接将响应返回给客户端，节省了 VS/TUN 中对 IP 隧道的开销，这种方式是三种负载调度机制里性能最好最高的。

4.6.2 节点管理设计

随着业务的高速发展，单台服务器可能无法承受日益增大的请求压力，在这种情况下，可以对该系统做水平扩展，让多节点同时服务，以突破单机处理能力、容灾能力的瓶颈，保证系统的高性能及高可用。

Akka 天生拥有分布式的能力，操作一个远程 Actor 的方式和操作本地 Actor 并没有明显的区别，它基本上已经屏蔽了底层的网络通信细节，基于去中心化的 P2P 模型，没有单点故障和单点瓶颈，可以满足多个应用场景，因此系统选用 akka cluster 来做集群管理。

Akka 集群由一系列的节点组成，这些节点分布在不同的 JVM 或者物理机上，每个节点都使用 hostname:port:uid 来唯一标识。当集群系统启动时，每个种子节点将会自动加入该集群，此时其他节点可以发送 join 命令，要求加入该集群。在集群运行阶段，可以根据业务运行情况动态地加入新的节点，或者停掉某个节点。在为整个集群服务的过程中，节点的状态和角色会有所变化[41]。在节点的生命周期中，节点的状态会发生一些改变，在常规情况下，节点有如下状态。

（1）joining：加入集群时的临时状态。

（2）up：正常操作状态。

（3）leaving/exiting：正常移除状态。

（4）down：不再是集群能够选择的节点。

（5）removed：彻底移除，不再是集群成员。

节点角色分为种子节点和领导节点。种子节点和普通节点并没有太大区别，它被配置在 cluster.seed-nodes 中，在一个集群中，可以配置多个种子节点。当集群启动时，不要求所有种子节点都启动，但是排在列表第一个的节点必须启动，否则其他节点没法加入集群；领导节点主要负责管理集群中成员节点的状态，比如将一个 joining 状态下的节点设置为 up 状态，或者将一个 exiting 状态下的节点设置为 removed 状态。在集群中，任何节点都可以成为 leader。

采用 akka cluster 来做集群管理，每个节点对等，不存在使用一台机器桥接做分布式所产生的单点故障隐患。每个节点监听 MemberUp、MemberDowm、MemberUnreachable、ClusterMemberState 等事件来感知其他节点的上下线，用 Akka Actor 实现节点间的消息通信。基于 Akka 的集群管理设计如图 4-11 所示。

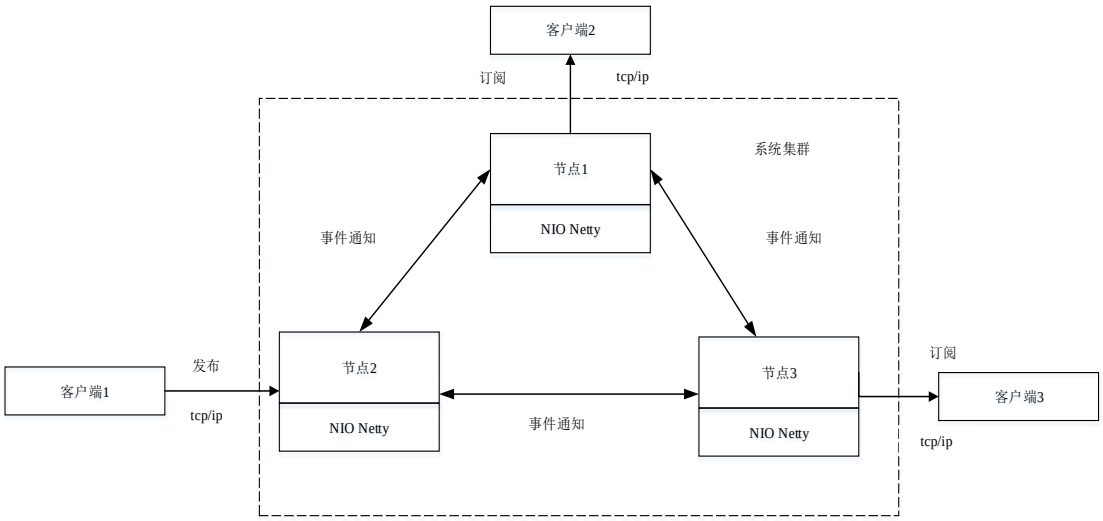

图 4-11 基于 Akka 的集群管理设计

4.6.3 集群路由设计

每个单点的服务端在运行时被称为节点，节点间通过 TCP 两两互联，组成一个网状结构。每当一个新的节点加入集群时，它会与集群中所有的节点都建立一个TCP连接。物联网设备与集群中任意节点建立连接后可以进行消息的发布，发布在单节点上的消息需要经集群路由模块将消息转发和投递给各节点上的订阅者，消息在集群间的转发过程如图 4-12 所示。

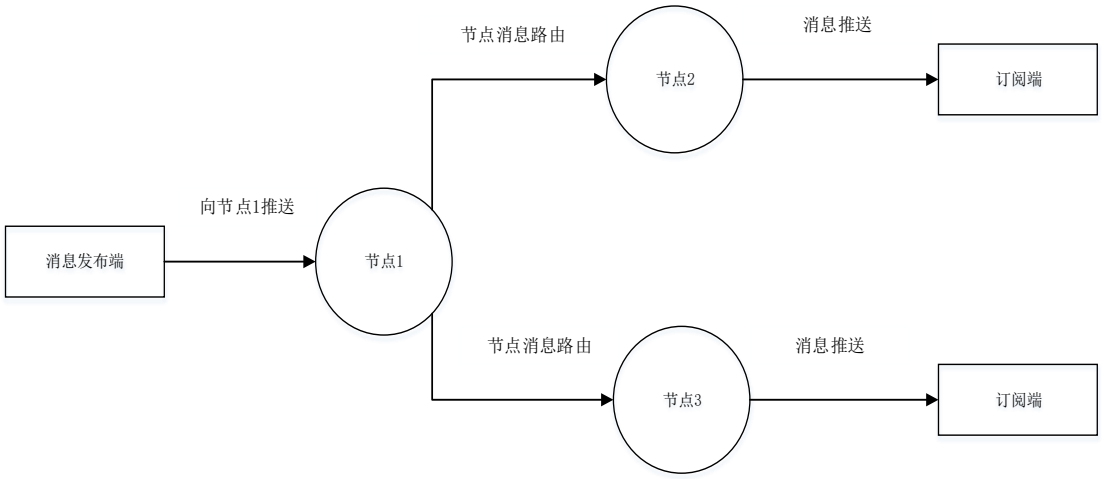

图 4-12 集群间消息转发

为了实现此过程，每个集群中的所有节点，都需要维护一份主题与节点关系映射的路由表。路由表设计如表 4.2 所示。

表 4.2 消息路由表

主题	节点
topic1	node1，node2
topic2	node3
topic3	node2，node4

当物联网设备发布消息时，所在节点会根据消息主题，查询主题订阅树匹配出订阅该主题的客户端然后将消息推送给相关订阅者，再检索路由表并将消息转发到订阅了该主题的客户端所在的节点上，再由这些节点将消息推送给相关订阅客户端。例如设备 device1 向主题 room/temp 发布消息，消息在节点间的路由与派发过程：

（1）device1 发布主题为 room/temp 的消息到节点 node1。

（2）node1 通过查询主题订阅树匹配出订阅了该主题的客户端列表，然后将消息推送给客户端。

（3）node1 通过查询路由表，得知主题 room/temp 在 node2 上有订阅者，主题 room/#在 node3 上有订阅者，然后将消息转发到 node2 和 node3。

（4）node2 和 node3 接收到 node1 转发给它们的消息后，查询自身的主题订阅树匹配出订阅者，并把消息投递给他们。

（5）消息转发和投递结束。

集群间消息转发可简述为下述两条规则：MQTT 客户端订阅主题时，所在节点订阅成功后广播通知其他节点某个主题被本节点订阅。MQTT 客户端发布消息

时，所在节点会根据消息主题检索订阅并路由消息到相关节点。

4.7 系统监控模块设计

服务器节点在运行过程当中内部的状态对外部来说是不可知的，为确保系统管理员能够及时查看掌握系统内部运行情况，需要对系统内部状态进行监控。

系统提供以 Web 界面方式展示相关信息，这些信息可以分为两类：第一类为与系统业务相关的信息，比如在线设备、主题信息、订阅信息。在线设备展示了连接到指定节点的客户端列表，能够查看设备的基本信息。主题信息记录了节点的主题数量和当前节点有哪些主题被创建。

订阅信息提供指定节点下所有的设备订阅信息，通过该指标能够查看设备订阅了哪些主题。系统在运行过程中，需要实时统计更新这些数据，以变量的形式进行存储，当需要这些信息的时候通过调用相关接口就可以获取。第二类为服务节点运行情况，主要包括当前节点的 Cpu 和内存使用及集群其他节点的关键信息，以帮助系统管理员快速掌握每个节点的状态。

通过对系统运行时内部信息的监控，系统管理员能够对每个节点运行情况进行掌控，能够及时发现非法的连接和订阅，能够对服务节点的负载情况及时评判，以应对和解决业务增长所带来的问题。

4.8 数据存储设计

系统数据存储模块主要包括设备身份信息、权限信息、与业务相关的系统运行时内部信息等的存储。

系统使用 MySQL 数据库存储设备权限规则，这样设计的好处在于可以存储大量数据、动态管理设备权限，方便与外部设备管理系统集成；使用 Redis 数据库存储设备在线状态信息、连接列表、连接详细信息、持久化 Retain 消息。具体的数据库字段和键值对设计如以下表格所示。

（1）设备权限表。设备权限表记录了设备享有哪些主题的读写权限，在设备进行发布或订阅消息的时候需要查询该表，设备权限表存储在 Mysql 数据库中，设备权限表设计如表 4.3 所示。

表 4.3 设备权限表

Column	Type	Check	Primary key
Id	Int(11)	Not null	Yes
Allow	Int(1)	Default 1	NO
Ip	Varchar(60)	Default null	No
deviceName	Varchar(100)	Default null	No
clientId	Varchar(100)	Default null	No
Access	Int(2)	Not null	No
Topic	Varchar(100)	Not null	No

（2）设备身份信息表。设备身份信息表记录了设备的身份信息，是设备能否建立连接的重要依据，设备信息表存储在 Mysql 数据库中，设备信息表设计如表 4.4 所示。

表 4.4 设备身份信息表

Column	Type	Check	Primary key
Id	Int(11)	Not null	Yes
deviceName	Varchar(100)	Default null	No
productSecret	Varchar(100)	Default null	No
Is_superuser	Int(2)	Default 0	No
Created	Datetime	Default null	No

（3）设备在线状态。设备上下线时，需要更新在线状态、上下线时间至 Redis 数据库。以 Hash 类型 mqtt：client：{clientid}格式的 key 记录，其键值设计如表 4.5 所示。

表 4.5 设备在线状态键值对设计

Field	Value
State	离线为 0 在线为 1
Online	时间戳
Offline	时间戳

（4）节点连接列表。节点连接列表记录了当前节点下有哪些设备连接，设备上线时间，以 Hash 类型 mqtt：node：{nodeName}格式的 key 进行记录存储在 Redis 数据库中，其键值设计如表 4.6 所示。

表 4.6 节点连接列表键值设计

Field	Value
Clientid	时间戳

（5）订阅信息。订阅信息记录了每个设备以何种 Qos 等级订阅了哪些设备，以 Hash 类型 mqtt：sub：{clientId}格式 key 在 Redis 中初始化代理订阅 Hash。其键值设计如表 4.7 所示。

表 4.7 订阅信息键值设计

Field	Value
Topic	消息质量等级

（6）消息详情表。消息详情表记录了需要保留在 Redis 数据库中的 Retain 消息。以 Redis Hash 类型 mqtt：msg：{messageId}为 key 进行存储，其键值设计如表 4.8 所示。

表 4.8 消息详情键值设计

Field	Value
Id	消息 id
From	设备标识
Qos	消息质量等级
Topic	消息主题
Payload	消息内容
Ts	时间戳

（7）持久化 Retain 消息。系统需要保留设备发布的消息中字段 Retain 为 true 的消息，每次仅保留最新的 Retain 消息，当订阅了该主题的设备连接的时候需要将该条 Retain 消息推送给设备。以 string 类型 mqtt：retain：{topic}为 key 存储在 Redis 数据库中，其键值设计如表 4.9 所示。

表 4.9 Retain 消息键值设计

Field	Value
Topic	消息 id

4.9 本章小结

本章根据系统需求分析对消息系统做出了明确的设计，首先给出了系统的整体结构设计，然后将系统分为六大功能模块。分别从各功能模块展开详细的功能模块的设计，以流程图、结构图、表格的方式详细地展示了各个部分的设计，为系统的实现打下基础。

第五章 系统实现

本章结合第四章对各个功能模块的设计，给出了系统编码时使用到的关键类、函数和相关接口，并结合类图、时序图、流程图等形式详细地阐述了系统各功能模块的实现过程。

5.1 物模型与主题模块实现

5.1.1 物模型实现

物模型是对设备在云端的功能描述，包括设备的属性、服务和事件。系统通过定义一种物的描述语言来描述物模型，称之为 TSL（即 Thing Specification Language），采用 JSON 格式，可以根据 TSL 组装上报设备的数据。比如对设备属性上报，采用以下数据格式进行，其他类型的物模型数据上报格式类似。

```
{
"id": "",
"timestamp": 0,
"method": "thing.event.property.post",
"version": "1.0",
"productKey": "",
"deviceName": "",
"params": {
${identifier_1}: ${value_1}
...
${identifier_n}: ${value_n}
}
}
```

其中 id 指其身份标识，timestamp 为时间戳，method 表示消息内容的类型，version 为版本信息，productKey 指设备所属产品的唯一标识，deviceName 为设备名称，params 为设备上传的数据信息。设备的节点类型为网关时，主题中的 productKey 和 deviceName 是指网关设备的，报文中的 productKey 和 deviceName 是网关下子设备的。当设备节点类型为直连设备时，主题和报文中的 productKey 和 deviceName 都是指直连设备的。

5.1.2 主题实现

系统将物联网设备主题分为数据上行和数据下行两大类，数据上行指设备将

自身的运行状态、事件上报到系统中；数据下行指系统管理员或用户为控制设备向设备下达命令。数据上行分为三类：设备属性上报、设备事件上报、设备状态上报。数据下行分为三类：设备属性设置、设备属性获取、设备服务调用。其对应的主题实现如表 5.1 和 5.2 所示。

表 5.1 数据上行主题实现

主题类型	主题格式
设备属性上报	/sys/${ProductKey}/${DeviceName}/thing/event/property/post
设备事件上报	/sys/${ProductKey}/${DeviceName}/thing/event/${tsl.event.identifier}
设备状态上报	/sys/${ProductKey}/${DeviceName}/thing/event/status/post

表 5.2 数据下行主题实现

主题类型	主题格式
设备属性设置	/sys/${ProductKey}/${DeviceName}/thing/service/property/set
设备属性获取	/sys/${ProductKey}/${DeviceName}/thing/service/property/get
设备服务调用	/sys/${ProductKey}/${DeviceName}/thing/service/${tsl.service.identifier}

5.2 设备状态管理模块的实现

5.2.1 设备连接实现

设备建立连接是实现后续发布或订阅等操作的基础，设备在连接时通过发送 Connect 报文连接系统。实现过程为系统首先监听设备发送的请求，然后判断请求类型，当监听到设备发送的请求为 Connect 类型的消息，则调用 ConnectProcessor 类进行处理，该类会解析出 Connect 消息报文中的协议版本、设备标识符、clean Session 标志位、用户名、密码，然后调用方法 deviceConnectAuth()验证协议版本、设备标识符、设备身份信息，如果验证不通过则根据验证的结果返回给设备端相应的失败连接返回码。

当设备通过 deviceConnectAuth()方法验证后，系统会调用 getClient（）方法查询系统之前是否有该 clientid 下对应的连接，如果有则调用 removeClient（）方法清除之前的连接。如果 clientSession 标志位为 true 则调用 createSession（）创建设备 Session 信息，反之，则调用 reloadClientSession（）重新加载设备的订阅信息。如果 willFlag 标志位为 true，则调用 storeWillMsg（）方法存储设备的遗嘱信息，返回给客户端 ackMessage，至此连接建立。最后调用 offlineMsg（）判断设备是否有离线消息并进行推送。设备连接时序如图 5-1 所示。

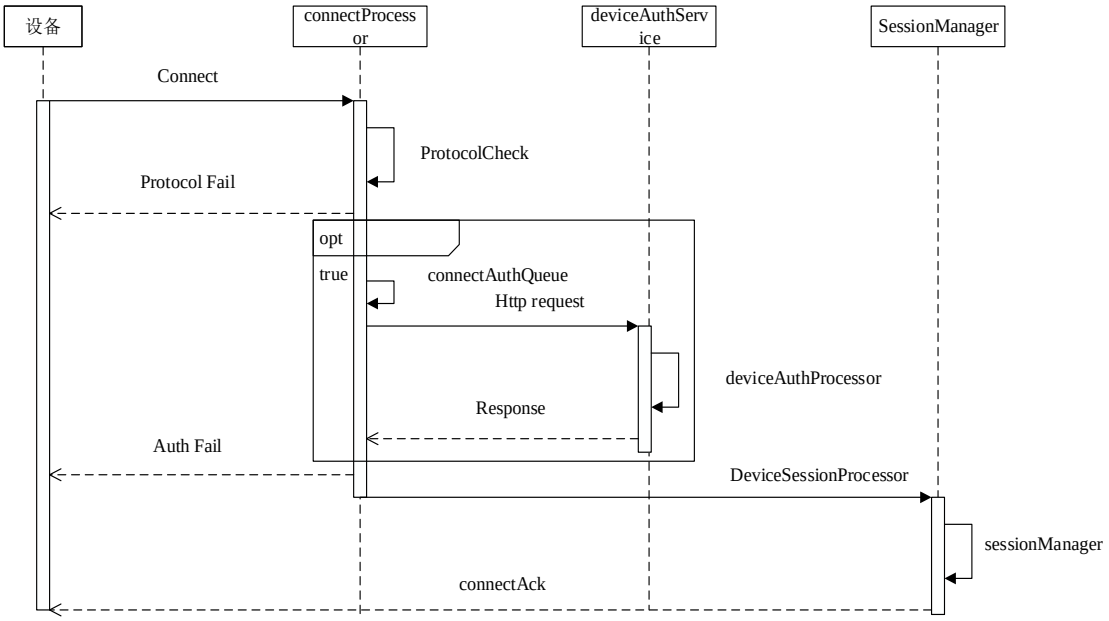

图 5-1 设备连接时序图

5.2.2 Session 信息存储实现

系统定义了两种 Session 信息的存储，分别为持久会话和非持久会话。持久会话为cleanSession字段设置为false的客户端Session，非持久会话为cleanSession字段设置为 true 的客户端 Session。持久会话在客户端断开重连后，之前的订阅数据、离线期间接收的消息依然存在；非持久会话在断开连接后就会清空所有数据。

非持久会话放在内存里，只在连接的节点上存在，连接断开或节点崩溃后清空。当设备端连接时，会调用 createTransientSession（）方法，以 clientid 为 key，clientSession 为 value 保存在 ConcurrentHashMap 中。其中 clientSession 保存了设备的 clientid，cleanSession 标志位和 ChannelHandlerContext 上下文信息。

对于持久化会话，即使设备主动断开，下次再连接的时候也需要有其对应的连接信息，而且在集群模式下，设备的下一次连接可能建立在不同的节点上，因此不仅在内存里保存其 Session 信息，还需要利用 Redis 存储 Session 信息以实现 Session 信息的共享。具体实现过程为设备在每次建立连接的节点内存里和非持久化会话一样保存一份以 clientid 为 key，clientSession 为 value 的 ConcurrentHashMap。另外调用 createPersistent（）方法，将设备的订阅信息、clientSession标志位存储在Redis数据库中，存储类型为Redis的Hash数据结构，以 clientid 为 key，字段为 cleanSession 和 subscription 订阅信息。对于会话的存储和获取主要由 ClientSessionProvider 和 ClientSessionPersistenceImpl 来实现。其类图如图 5-2 所示。

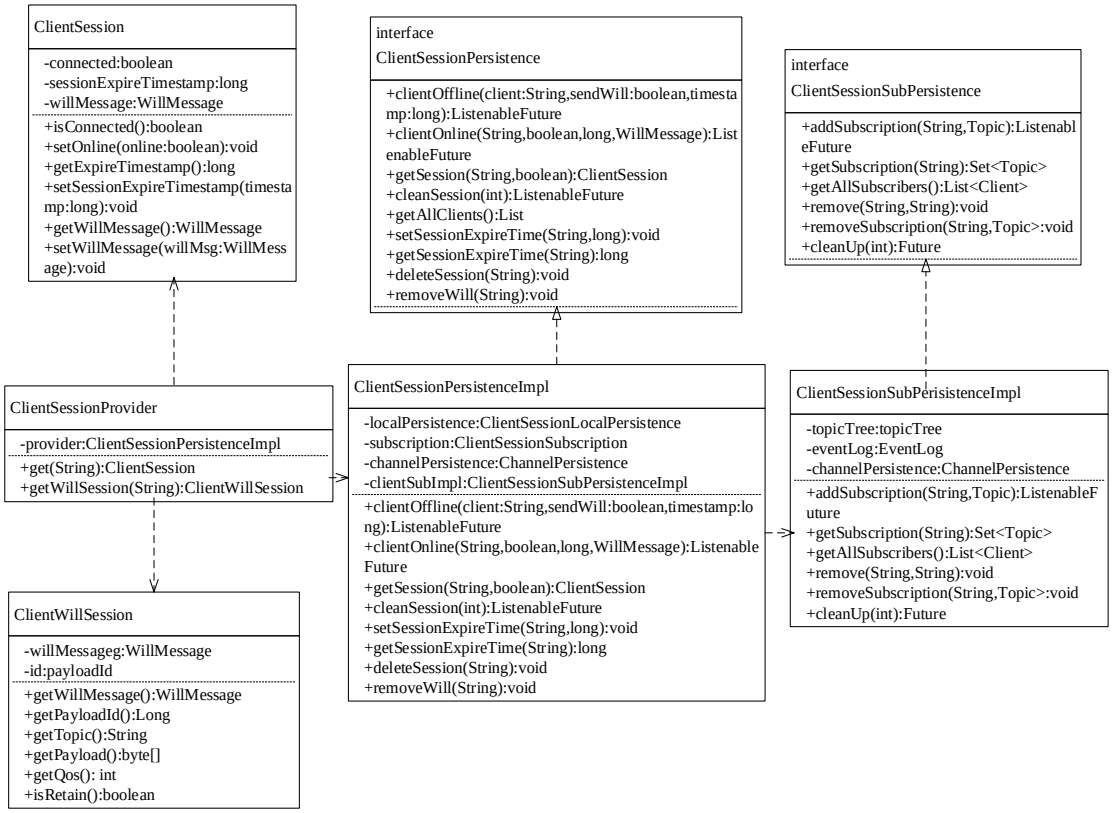

图 5-2 Session 处理类图

5.2.3 设备心跳实现

设备与系统建立连接之后，需要由系统维护与设备之间的 Tcp 长连接，并定时检测每个连接的状态，及时断开不活跃的连接，释放宝贵的服务器内存资源。

根据 MQTT 协议规范，维持连接的状态需要由设备端主动向服务节点根据客户端设定的心跳间隔时间定时发送 pingreq 请求，系统接收到 pingreq 请求之后返回给客户端 pingresp 消息，以此来保持设备的连接状态。如果在规定时间内没有收到设备发送的心跳请求，服务端则会认为设备已经离线并且主动断开该设备的连接。

使用 Netty 框架提供的 IdleStateHandler 类来实现连接的读写事件和空闲事件的检查。根据设备端设置的心跳时间，服务端在 1.5 倍的心跳时间内若没有接收到来自设备的数据包，则服务端会触发 NettyConnectHandler 类下的方法 userEventTriggered（），该方法会找到当前设备对应的连接并及时关闭，根据 Session 类型决定是否需要清除连接对应的 Session 信息，释放服务器资源。其中主要涉及的类为 IdleStateHandler 和 NettyConnectHandler。NettyConnectHandler 类图如图 5-3 所示。

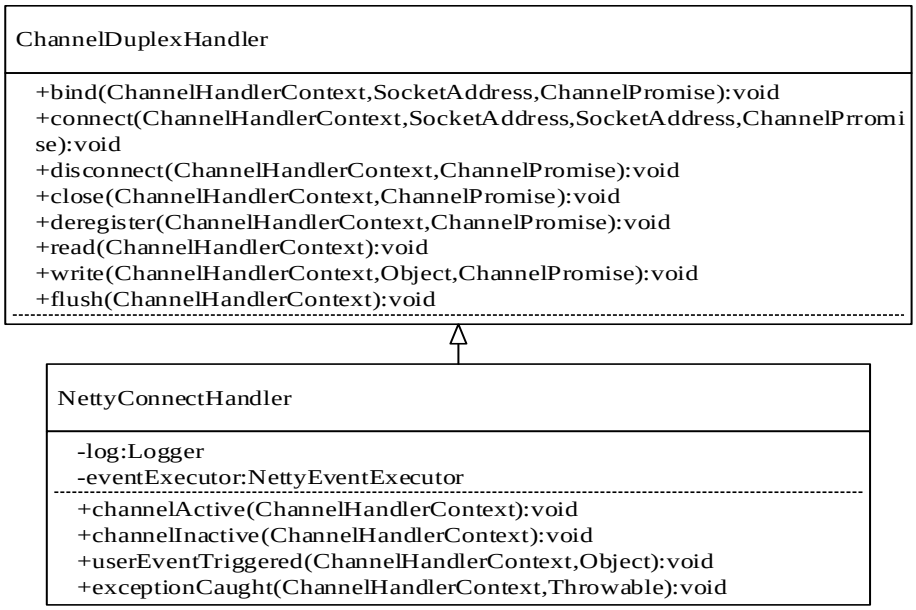

图 5-3 NettyConnectHandler 类图

5.3 消息路由模块的实现

5.3.1 即时消息推送实现

消息推送的前提是设备端成功建立与系统的连接，当设备端需要向某一主题推送消息时，首先需要发送 publish 报文请求，通过系统的协议处理模块进行处理，其中类 PublishProcessor 的 processRequest（）方法将发布的主题、Qos 质量级别和消息内容提取出来，封装成系统内部的 innerMqttMessage 类。

然后调用发布权限 ACL 模块通过 publishVerify（）方法验证当前设备是否具有该主题的发布权限。HTTP 状态码返回 200 表示权限验证通过，验证通过之后根据不同的 Qos 分别调用 processMessage（）方法进行消息推送。

在推送之前需要查询本地的主题订阅树匹配出哪些设备订阅了该主题消息，具体的匹配过程为：将要推送的主题按照“/”分为一个 Token 数组，每个 Token 对应一个主题层级，依次查找出每个 Token 层级对应的设备订阅列表得出所有订阅了该主题消息的设备。

最后将主题消息写入到每个设备对应的消息队列中去，最后根据设备的 clientid 查找其对应的 channelContext 和设备在线状态信息，对于在线设备通过其 channel 通道调用 ctx.writeAndFlush（）方法将消息推送给指定的客户端，对于离线设备根据离线消息存储模块判断是否需要进行离线消息存储。消息推送的流程图如图 5-4 所示。

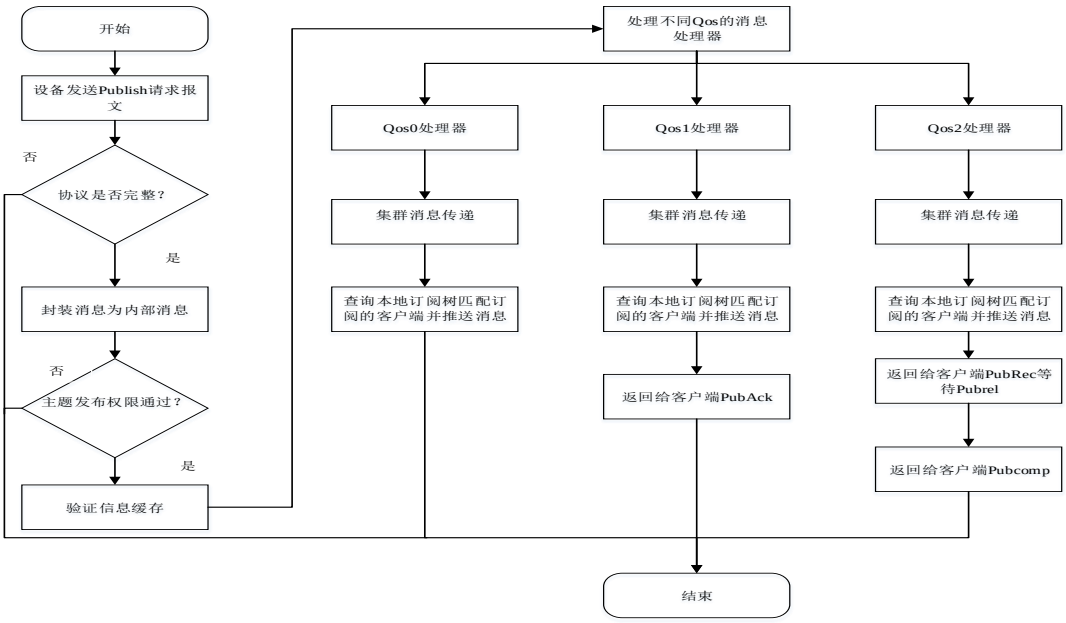

图 5-4 消息推送流程图

5.3.2 离线消息存储实现

物联网环境下网络资源有限，设备掉线情况时有发生，为了保证设备掉线重连之后能够接收到离线时订阅的消息，结合 MQTT 协议规范，系统需要存储 cleanSession 字段为 false 且 Qos 大于 0 的设备离线消息。

当有消息推送到系统时，通过查询订阅树匹配出订阅了该主题消息的设备，通过设备状态查询模块调用 getClientStatus（）方法查询设备是否在线，对于 Qos 大于 0 的消息如果设备离线并且 cleanSession 字段为 false 则调用 offlineMessageStore()方法进行离线消息存储。

离线消息存储分为 HBase 数据库加载模块和数据写入模块两部分，其中系统在启动时需要读取 HBase 数据库配置文件相关参数并建立数据库连接，系统运行时通过离线数据写入模块将设备离线消息写入数据库。

在系统初始化时通过 HbaseConfigura 类创建 Configuration 对象，读取配置文件将数据库的连接地址、端口号以参数形式配置到 Configuration 实例对象 conf 中。当监听到离线消息时调用 addRowData（）将消息以字节数组形式写入到数据库中，方法中参数由数据库表名、列族、列名以及消息内容组成。

其中为了保证海量离线消息的秒级查询，Rowkey 设计十分关键，通过 Rowkey 需要知道该条消息所属的设备信息，存储的时间以及消息的顺序。在 offlineMessageStore（）方法处理逻辑中，通过调用 generateRowKey（）方法生成每条消息的 Rowkey，Rowkey 组成为 clientid+timestamp+messageId。clientid 为离线设备的 ID，timestamp 为消息存储时的时间戳，messageId 为该设备对应的离线消息 ID。

最后消息保存在 list 列表里进行批量提交入表，在下一次设备上线时通过离线设备消息重发接口推送设备离线时的消息，并及时运行后台程序删除推送过的离线消息。

5.3.3 Kafka 消息桥接实现

Kafka 消息桥接模块根据预先设置的规则将需要转发到 Kafka 的主题消息筛选并写入对应的 Kafka 主题中去，其他业务微服务根据需要通过订阅 Kafka 的主题来消费消息。

在 Kafka 消息桥接模块中，消息系统作为生产者角色向 Kafka 主题消息队列发送消息。首先在 Kafka 服务端创建属性、事件和服务三大主题，分别作为设备上报不同类型消息的存储主题。

在代码实现中通过 Properties 类实例化 props 对象，用来设置 Kafka 服务端的主机名和端口号、是否等待所有副本节点的应答、消息发送最大尝试次数、一批消息处理大小、发送缓存区内存大小、指定 key 和 value 的序列化方式。

使用 KafkaProducer 类带有发送回调函数的 send（）方法将设备消息发送到 Kafka 各个主题中，通过 Kafka 提供的 Future 机制进行消息异步监听判断是否成功写入，并对发送失败的消息进行处理。其中需要将设备发送过来的消息主题与要写入到 Kafka 里的消息主题做对应处理，实现方式为将原生消息主题里的主题类型提取，分别与 Kafka 主题对应。

物联网服务系统根据自身业务需要订阅 Kafka 主题消息，在对应的 Consumer 类中设置消息监听器 Listener，当监听到消息时调用 messageConsume（）方法对消息进行处理。Kafka 消息桥接时序图如图 5-5 所示。

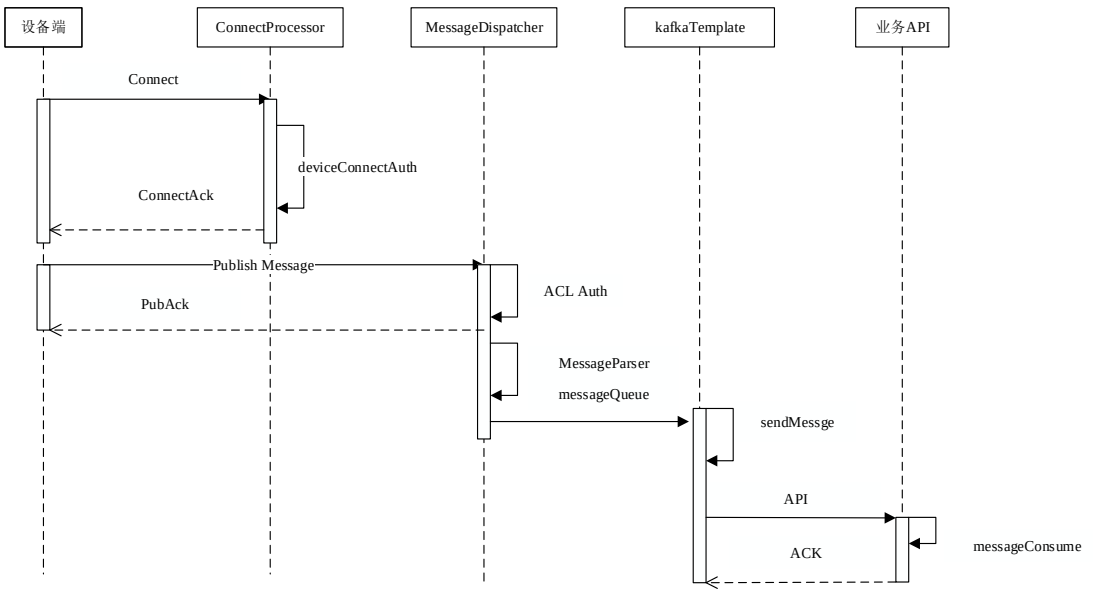

图 5-5 Kafka 消息桥接时序图

5.4 系统安全模块的实现

5.4.1 设备连接认证实现

采取一型一密验证方式的设备在物联网设备注册平台进行设备注册时，平台会为这种型号的设备颁发唯一的设备秘钥，即同一种型号的设备共用唯一的设备秘钥。采取一机一密验证方式的设备在物联网设备注册平台进行设备注册时，平台会为每个设备颁发唯一的设备秘钥。

设备身份认证服务依赖部署在其他服务器上的服务实现，基于这个认证服务系统的返回值，系统决定设备身份控制。在设备连接时通过报文解析模块对 connect 连接报文进行解析，提取出 clientid、username 和 password 三个参数，然后通过 deviceAuth（）方法向设备身份验证服务发送 http 请求。

Http 请求采用 HttpClient 开源框架实现，实现时首先创建 HttpClient 实例，设置 Http 连接主机服务超时时间，通过方法 getHTTPConnectionManager（）获取连接管理对象，通过 getParams（）获取参数对象，创建 Get 方法实例，设置 get 请求超时时间，最后设置请求重试机制，默认重试次数为 3，最后发送身份验证请求并监听状态码，根据不同的状态码消息体返回给设备响应。设备身份验证服务由注解@Controller 和@RequestMapping 标注，拦截接收到的设备身份认证请求后将收到的设备标识、用户名和密码这三个参数与 Mysql 数据库中保存的设备身份信息进行比对。

如果身份信息一致则返回状态码 200 代表验证通过，允许设备建立连接；反之，返回状态码 401 代表身份信息验证失败，系统需要拒绝该设备的连接请求并返回给客户端相应错误信息。整个连接验证的时序图如图 5-6 所示。

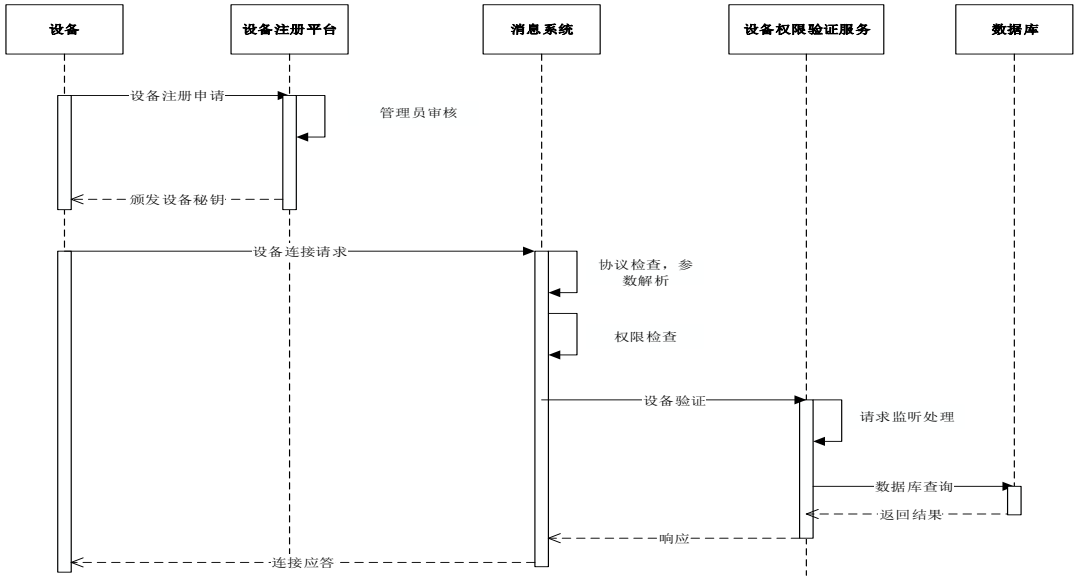

图 5-6 设备身份验证时序图

5.4.2 发布订阅 ACL 实现

系统在设备发布、订阅事件中使用当前设备标识、IP 地址、用户名、密码和要发布或订阅的主题作为参数，向自定义的认证服务发起请求权限，通过返回的 HTTP 响应状态码来处理 ACL 授权请求。若没有权限，API 返回 401 状态码；若授权成功，API 返回 200 状态码；如果忽略授权，API 返回 200 状态码且消息体 ignore。并且在配置文件中配置 HTTP 请求信息、请求地址以及请求方法。

对于系统管理员等超级用户，系统将跳过 ACL 查询。对于非超级用户进行发布、订阅认证时，系统将使用当前设备信息填充并发起 ACL 授权查询请求，查询出该设备在 HTTP 服务器端的授权数据。如果 HTTP 请求方法为 GET 时，请求参数将以 URL 查询字符串的形式传递；POST、PUT 请求则将请求参数以普通表单形式提交。发布订阅 ACL HTTP 请求发送实现过程与身份验证类似。

设备主题鉴权服务同样以注解@Controller 和@RequestMapping 标注，当拦截到系统发送过来的请求时，解析传递过来的设备标识、ip 地址、用户名、密码、要发布或订阅的主题信息并与 Mysql 数据库中的权限信息进行匹配，数据库中 access 字段为 1 则代表该设备具有该主题的发布权限，为 2 则代表该设备具有该主题的订阅权限，为 3 则代表具有发布和订阅的权限。

5.5 系统集群模块的实现

5.5.1 节点管理实现

Akka 集群由一系列节点组成，这些节点分布在不同的 JVM 或者物理机上，每个节点都使用 hostname:port:uid 来唯一标识。当集群系统启动时，每个种子节点将会自动加入该集群，此时其他节点可以发送 join 命令给这些种子节点，要求加入该集群。任何节点状态的变化都会通过 gossip 协议传输到整个集群网络的每个节点，以保证集群状态的一致性。

在将项目配置成集群环境之前，首先在项目中加入 Akka 集群模块的依赖，在 maven 中进行配置。种子节点和普通节点没有太大区别，它被配置在 cluster.seed-nodes 中，在一个集群中也可配置多个种子节点，根据集群规模进行配置，在本文中配置两个种子节点。在启动时，种子节点不必全部启动，但是排在 seed-nodes 列表中的第一个节点必须启动，否则其他节点没法 join 进集群。

将 akka 集群启动之后，每个节点的状态都会被传播到其他节点，为了得到这些信息，需要由一个 actor 订阅相关事件。在 clusterMessagActor 中，首先通过 Cluster.get(getContext().system))方法得到当前集群对象，然后调用 subscribe 方法让当前 Actor 订阅 UnreachableMember、MemberEvent 事件，其中 UnreachableMember 事件会在某个节点被故障检测器认定为不可达时触发，

MemberEvent 事件为成员状态的顶级事件。在 onReceive 方法中，通过匹配不同事件来了解节点状态信息，通过 Member 获取当前节点的地址信息、状态信息、角色信息，当集群出现问题时及时发送消息给系统管理员。

5.5.2 集群路由实现

整个系统以去中心化集群方式进行搭建，一个系统中存在多个服务节点，物联网设备发布消息时，当前节点不仅需要将该消息推送给本节点上的订阅客户端，还需要将消息路由给其他服务节点以保证消息的精准转发。此过程中，消息路由表是整个转发过程的关键，为了保证路由表的正确与完整性，需要在 MQTT 客户端订阅主题时更新主题路由表，在 MQTT 客服端发布消息时根据消息主题 Topic 检索订阅并路由消息到相关节点。当单个服务节点接收到订阅请求时，订阅模块调用 clusterRemoteSubscribe()方法将该订阅的主题和当前节点信息广播给其他服务节点，其他服务节点监听到该消息判断该消息类型，然后主动更新内存中保存的主题和节点的路由表信息。

当设备发布消息时，首先调用消息路由模块将消息推送至订阅了该主题消息的客户端，消息路由过程与消息路由模块一致，然后以主题 Topic 为参数查询路由表，匹配出订阅了该主题消息的客户端所在的节点集合，然后通过 remoteMessage（）将消息路由到这些服务节点。当其他节点接收到来自集群间的消息，判断消息类型为主题发布类，然后通过查询本地订阅主题树，匹配出订阅了该主题的客户端，通过消息推送模块将消息推送给这些客户端，至此完成整个集群间消息的推送。集群间消息路由流程图如图 5-7 所示。

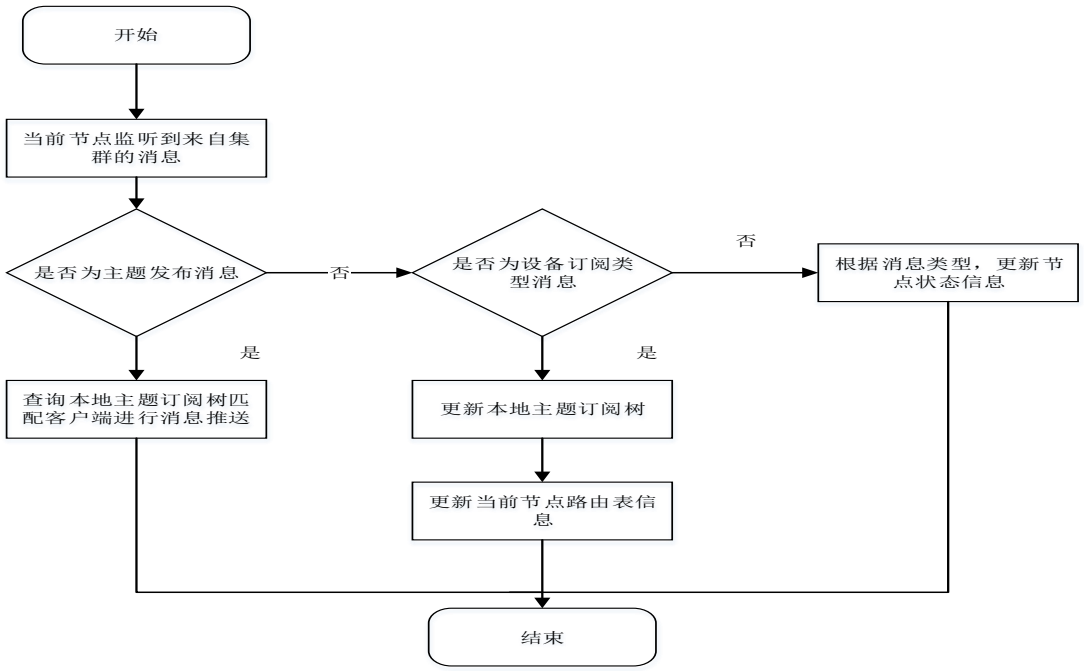

图 5-7 集群间消息路由流程图

5.6 系统监控模块实现

系统监控模块以变量的方式存储每个服务节点的设备连接数、主题订阅信息、在线客户端等信息存储在 Redis 数据库中。当系统管理员需要查看该类信息时，通过查询 Redis 数据库就可以获取这类信息。

比如在设备连接时，通过 redisTemplate.opsForValue().getAndSet()方法，记录并更新当前连接的设备变量 onlineClient 的值；在设备订阅主题消息时通过 storeSubscription()方法记录并更新订阅的客户端和主题信息，其他统计信息的实现方式类似。

对于服务器节点的 Cpu 占用率、内存使用情况等这类信息，系统通过脚本获取这些信息定时更新并记录在 Redis 数据库中。

通过访问 Web 界面即可对服务节点的负载及运行状态进行监控，及时发现非法设备连接与服务运行异常，保证服务的稳定。系统监控 Web 页面如图 5-8 所示。

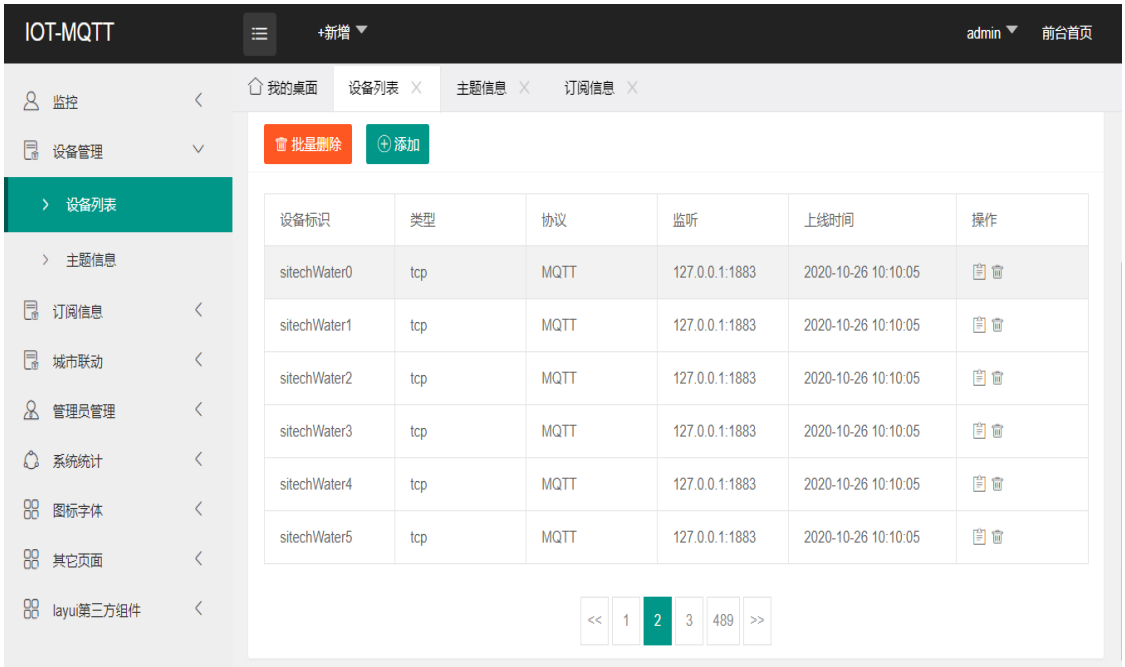

图 5-8 系统监控 Web 页面

5.7 系统存储模块实现

系统存储模块分为基于内存的数据存储和基于数据库的数据存储，其中设备发布的消息、设备端与 Channel 映射的 map 集合、主题订阅树、路由表信息保存在每个服务器的节点中；设备端 Session 信息、保留消息、遗嘱消息保存在 Redis 数据库中。

其中 AbstractMqttStore 为抽象类接口，具体实现由 RedisMqttStore 类完成，RedisMqttStore 负责由 Redis 数据库存储的设备端 Session 信息、willMessage、retainMessage 和 subscribeInfo 信息。RedisTemplate 类负责数据库的初始化、加载和关闭操作，分别由函数 init（）、operate（）和 close（）完成。其中 init（）函数负责配置 Redis 数据库连接池信息；operate()提供操作 Redis 不同数据类型的接口；close（）负责数据库连接的关闭；clusterConfig 类用于配置系统的集群信息。基于 Redis 数据库的存储类图如图 5-9 所示。

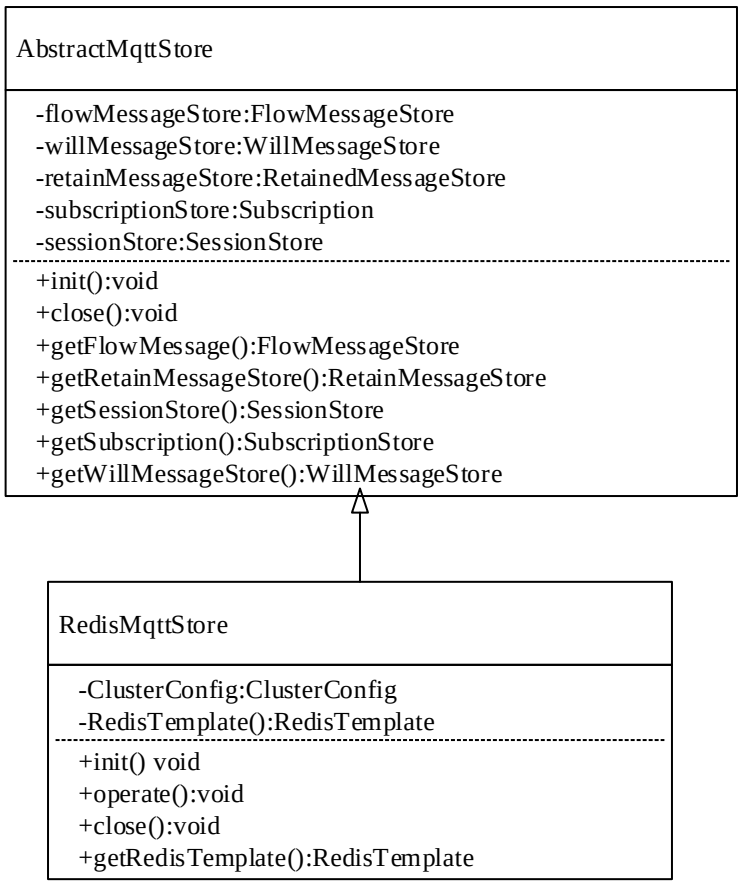

图 5-9 Redis 数据库存储类图

5.8 本章小结

本章结合系统设计章节，在各个功能模块的实现中，以类图、时序图、流程图、系统截图等方式展示了系统实现时的细节。对各个功能模块的实现细节的描述，展示了系统实现过程中遇到的难点及解决方案，描述了各个功能模块间的联系，完成了系统的开发工作。

第六章 系统测试

本章从功能性和非功能性两方面对系统进行测试，首先介绍了系统的测试环境，然后以编写测试用例的方式详细地描述了各个功能模块的测试过程及结果。最后从消息时延、系统连接数、集群并发数、集群连接数对系统的性能进行测试，并给出了测试结果及分析。

6.1 测试环境

消息系统测试所需服务器在同一个局域网内，一共所需服务器 14 台，其中 4 台用于部署 mqtt 服务，2 台用于部署负载均衡软件，5 台用于部署系统所需数据库和业务应用服务，3 台用于部署 mqtt 客户端测试工具，系统部署图如图 6-1 所示。

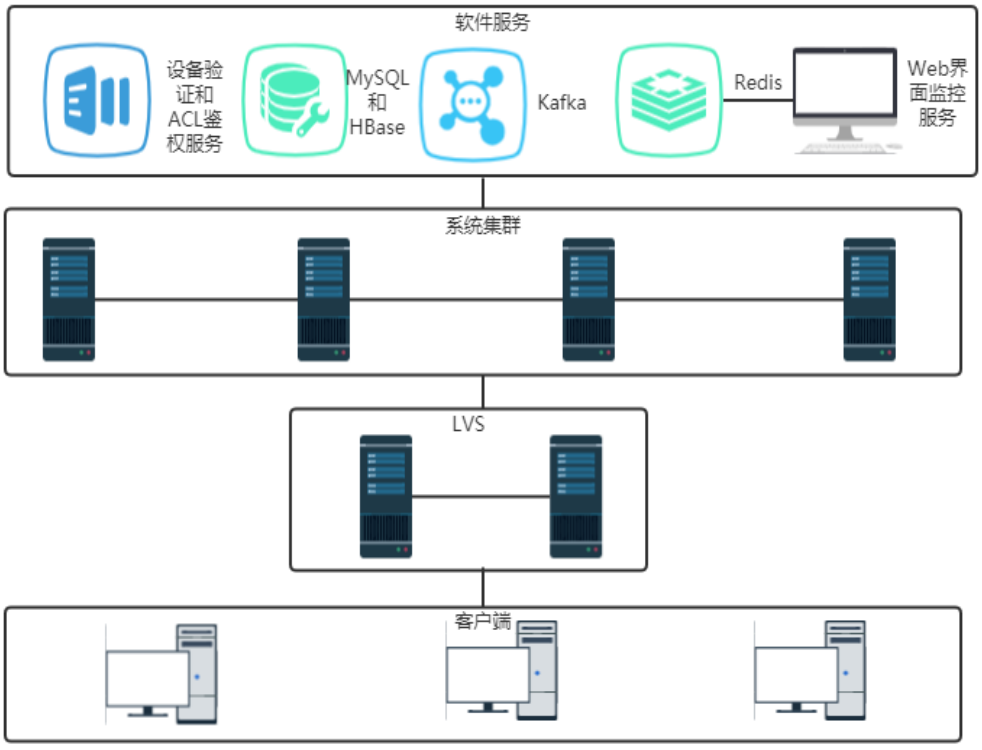

图 6-1 系统部署图

服务器硬件配置：4 核 2.4GHz，内存 8G，500G 硬盘，Linux 操作系统。

所需软件环境：Redis 数据库、HBase 数据库、Kafka 消息中间件、MySQL 数据库、设备验证和 ACL 鉴权服务、Web 界面监控服务、Emqtt_benchmark 客户端、MQTTBox 客户端以及 Jmeter 测试软件，软件环境所需 10 台机器，具体的配置如表 6.1 所示。

表 6.1 软件环境配置表

服务器编号	IP 地址	说明
1	172.18.232.188	Redis
2	172.18.232.189	HBase、MySQL
3	172.18.232.190	Kafka
4	172.18.232.158	设备验证和 ACL 鉴权服务
5	172.18.232.159	Web 界面监控服务
6	172.18.232.160	Emqtt_benchmark
7	172.18.232.161	Emqtt_benchmark
8	172.18.232.162	MQTTBox、Jmeter
9	172.18.232.163	LVS
10	172.18.232.164	LVS

6.2 系统功能性测试

MQTTBox 是一个带有可视化界面的 MQTT 客户端工具，它支持创建具有多种连接设置的 MQTT 客户端，满足功能性测试的要求，因此选用 MQTTBox 作为测试系统功能的工具。

通过 MQTTBox 客户端工具来验证消息系统是否能够满足系统在功能上的需求，主要从设备身份认证、设备发布消息、设备订阅主题消息、设备接收消息、心跳机制和消息重发这六大基本功能展开测试。

以设备身份认证功能为例介绍功能测试流程，首先在 MQTTBox 客户端参数设置界面填写正确的设备身份信息、服务节点地址、心跳间隔等信息、保存设置的参数、点击连接按钮。参数设置和结果分别如图 6-2 和 6-3 所示。从图 6-3 可知，设备成功连接且符合预期。然后在 MQTTBox 客户端参数设置界面填写错误的设备身份信息、服务节点地址、心跳间隔等信息、保存设置的参数、点击连接按钮。结果如图 6-4 所示，设备不能正常连接，符合预期。

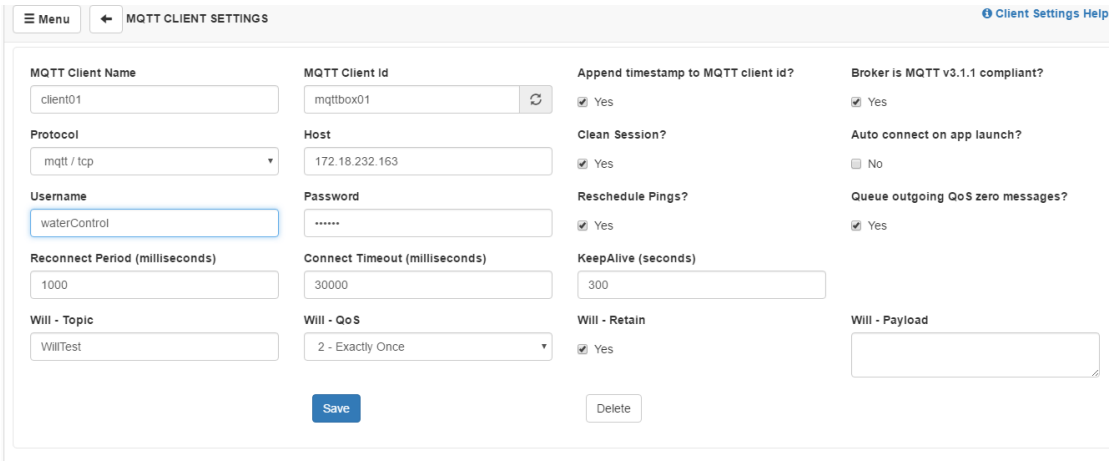

图 6-2 MQTTBox 客户端参数设置

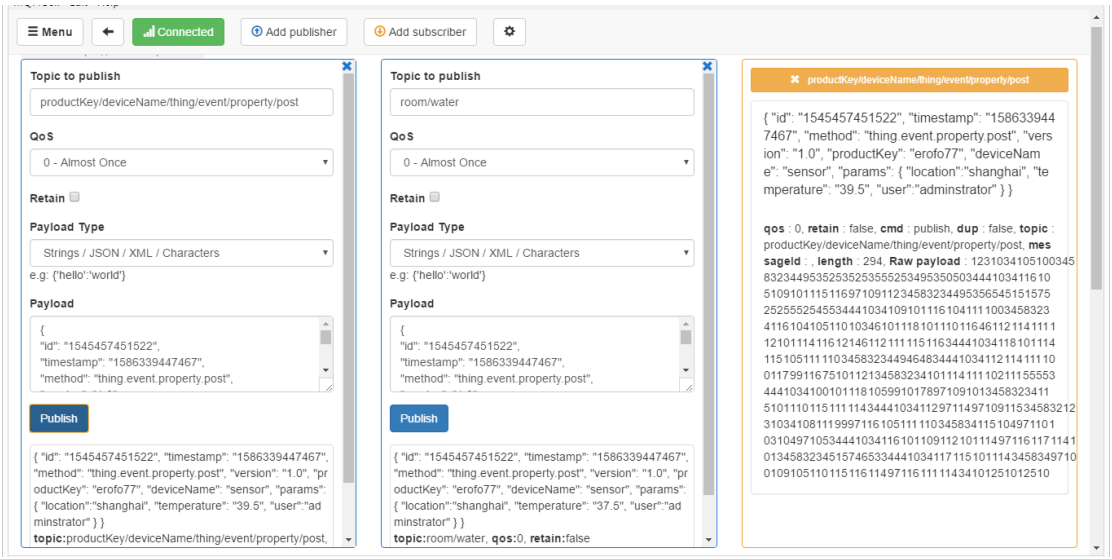

图 6-3 连接成功结果图

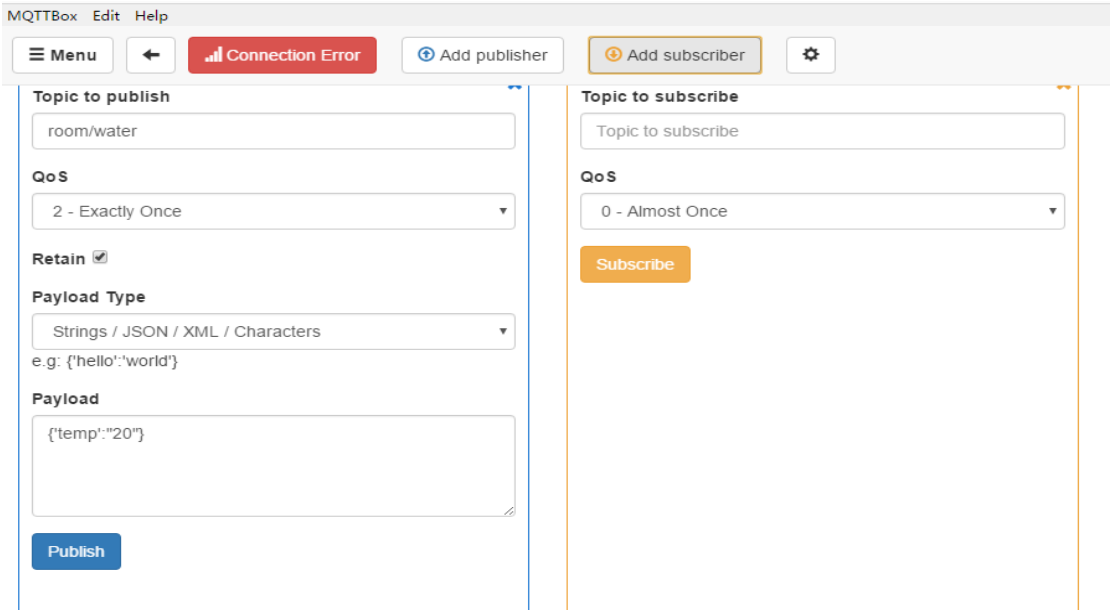

图 6-4 连接失败结果图

设备发布消息、设备订阅主题消息、设备接收消息、心跳机制和消息重发这五个功能模块的测试以测试用例方式给出。测试用例由用例名称、预置条件、测试步骤、预期结果和测试结果组成，表 6.2 至表 6.6 给出了各功能模块的测试用例。

表 6.2 设备发布主题消息测试用例表

用例名称	设备发布消息功能测试
预置条件	Clientid 为 mqttbox01 的设备成功连接、ACL 服务运行正常
测试步骤	（1）向 ACL 权限记录表中配置设备 mqttbox01 与主题 productKey/deviceName/thing/event/property/post 的读写权限，使其具有对该主题的发布权限。

	（2）MQTTBox 端设置 mqttbox01 客户端发布主题为 productKey/deviceName/thing/event/property/post、Qos 参数为 0、retain 设置为 true、payload 内容为{
	"id": "1545457451522",
	"timestamp": "1586339447467",
	"method": "thing.event.property.post",
	"version": "1.0",
	"productKey": "erofo77",
	"deviceName": "sensor",
	"params": {
	"location":"shanghai",
	"temperature": "37.5",
	"user":"adminstrator"
	}
	}的 mqtt 消息。
	（3）点击发布按钮。有预期结果 1.
	（4）删除上述 ACL 权限记录表中的 mqttbox01 与对应主题 productKey/deviceName/thing/event/property/post 的读写权限信息。
	（5）在 MQTTBox 端设置 mqttbox01 客户端发布主题为 productKey/deviceName/thing/event/property/post、Qos 参数为 0、retain 设置为 true、payload 内容为{
	"id": "1545457451522",
	"timestamp": "1586339447467",
	"method": "thing.event.property.post",
	"version": "1.0",
	"productKey": "erofo77",
	"deviceName": "sensor",
	"params": {
	"location":"shanghai",
	"temperature": "37.5",
	"user":"adminstrator"
	}
	}的 mqtt 消息。
	（6）点击发布按钮，有预期结果 2.

预期结果	预期结果 1：该条主题消息发布成功。 预期结果 2：该条主题消息发布失败。
测试结果	与预期结果相符，测试通过。

表 6.3 设备订阅主题消息测试用例表

用例名称	设备订阅主题消息功能测试
预置条件	Clientid 为 mqttbox02 的 MQTTBox 客户端成功连接、ACL 服务运行正常。
测试步骤	（1）向 ACL 权限记录表中增加一条 clientid 为 mqttbox02，主题为 productKey/deviceName/thing/event/property/post 的权限记录，使其具有该主题的订阅权限。

	（2）设置 clientid 为 mqttbox02 的 MQTTBox，使其订阅了 productKey/deviceName/thing/event/property/post 主题。 （3）点击订阅按钮。有预期结果 1. （4）在 ACL 权限记录表中删除 clientid 为 mqttbox02，主题为 productKey/deviceName/thing/event/property/post 的权限记录，使其不具有该主题的订阅权限。 （5）设置 clientid 为 mqttbox02 的 MQTTBox，使其订阅了 productKey/deviceName/thing/event/property/post 主题。 （6）点击订阅按钮，有预期结果 2。
预期结果	预期结果 1：clientid 为 mqttbox02 的客户端订阅主题 productKey/deviceName/thing/event/property/post 成功。 预期结果 2:clientid 为 mqttbox02 的客户端订阅主题 productKey/deviceName/thing/event/property/post 失败。
测试结果	与预期结果相符，测试通过。

表 6.4 设备接收主题消息测试用例表

用例名称	设备接收主题消息功能测试
预置条件	Clientid 为 mqttbox03 的客户端成功连接且已经订阅 productKey/deviceName/thing/event/property/post 主题，clientid 为 mqttbox04 的客户端成功连接，且具有任意主题的发布权限，服务节点运行正常。

测试步骤	（1）设置 mqttbox04 的客户端向主题 productKey/deviceName/thing/event/property/post 发布消息，Qos 任选，payload 设置为
	{
	"id": "1545457451522",
	"timestamp": "1586339447467",
	"method": "thing.event.property.post",
	"version": "1.0",
	"productKey": "erofo77",
	"deviceName": "sensor",
	"params": {
	"location":"shanghai",
	"temperature": "37.5",
	"user":"adminstrator"
	}
	}
	（2）点击发布按钮，有预期结果 1.
	（3）设置 mqttbox04 的客户端向主题 productKey/deviceName/thing/event/status 发布消息，Qos 任选，payload 设置为
	{
	"id": "1545457451522",
	"timestamp": "1586339447467",
	"method": "thing.event.property.post",
	"version": "1.0",
	"productKey": "erofo77",

	"deviceName": "sensor",
	"params": {
	"location":"shanghai",
	"temperature": "37.5",
	"user":"adminstrator"
	}
	}
	（4）点击发布按钮，有预期结果 2。

预期结果	预期结果 1：clientid 为 mqttbox03 的客户端接收到 clientid 为 mqttbox04 发布的消息。 预期结果 2：clientid 为 mqttbox03 的客户端未接收到来自非订阅主题的消息。
测试结果	与预期结果相符，测试通过。

表 6.5 心跳机制测试用例表

用例名称	心跳机制测试
预置条件	Clientid 为 mqttbox04 的客户端成功连接到消息系统，设置心跳间隔为 300s
测试步骤	（1）保持 mqttbox04 客户端的连接，查看服务端控制台信息每隔 450s 是否收到心跳请求并响应。
预期结果	查看控制台信息，服务端每隔 450s 接收到客户端的心跳请求并应答。
测试结果	与预期相符，测试通过

表 6.6 离线消息重发测试用例表

用例名称	离线消息重发测试
预置条件	客户端 mqttbox05 与 mqttbox06 成功连接系统并具有主题 sitech/shanghai/zhicheng/room01/temp 的发布订阅权限。
测试步骤	（1）在设置界面设置 mqttbox06 客户端订阅主题 sitech/shanghai/zhicheng/room01/temp 且 Qos 为 1，cleanSession 标志位为 false。 （2）mqttbox06 发送断开请求，在此期间 mqttbox05 客户端往该主题发布 10 条消息。 （3）mqttbox06 客户端重新连接。
预期结果	Mqttbox06 客户端重新连接后接受到服务端主动推送的 10 条离线消息。
测试结果	与预期结果相符，测试通过。

6.3 系统性能测试

本节对系统进行性能上的测试来验证是否满足系统在性能上的需求，主要从单节点连接数、三种 Qos 消息时延、集群并发数和连接数来展开测试。

6.3.1 单节点性能测试

首先是对单节点的连接负载测试，通过在两台机器上执行 Emqtt_benchmark 程序模拟设备连接对系统进行连接数测试。通过多次执行 Emqtt_benchmark 命令分别设置 5000、10000、20000、30000、38000 的连接数，通过 Web 监控页面统计连接成功率，得出的结果如表 6.7 所示。

表 6.7 单节点负载测试

连接数	Cpu 占用率	结果
5000	10.2%	成功
10000	18.5%	成功
20000	28.3%	成功
30000	41.6%	成功
38000	55.1%	成功

通过上表可以看出单节点的连接数在 38000 之内，设备连接稳定。在实际测试过程中，当连接数超过 38000 时出现少量连接拒绝现象，主要原因是受限于服务器硬件。

接着通过客户端程序分别模拟设备三种 Qos 消息发送处理过程。针对三种 Qos 消息，使用客户端程序在每次连接建立成功后发送不同 Qos 的一条消息，在测试期间总计发送 100000 条消息，分多次实验计算每种 Qos 消息从发送到接收的时间间隔，得到结果如表 6.8 所示。

表 6.8 消息时延测试

Qos	平均时延(ms)
Qos0	264
Qos1	371
Qos2	578

从上表可以看出 Qos 为 0 的消息时延最低，Qos 为 2 的消息时延最高，这与不同 Qos 消息的处理复杂度相符。因为 Qos 为 0 的消息在系统进行处理时只需要进行推送，不需要持久化，而 Qos 为 1 和 2 的消息则需要进行接收反馈和持久化操作，其中 Qos2 处理过程比 Qos1 更复杂。整体的消息处理时延都小于 1s，满足了企业实际应用场景在时延上的要求。

6.3.2 集群性能测试

对消息系统能支持的并发量进行测试，实现方式为采用 Jmeter 测试软件分别模拟数量为 2000、4000、6000、8000、10000 的并发连接。以 8000 并发连接为例，在 Jmeter 中设置并发连接数和相关参数并点击执行，最后生成测试报告，结果如图 6-5 所示。并发量测试结果如图 6-6 所示。

Requests	Executions			Response Times (ms)					
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct
Total	**8000**	**0**	**0.00%**	**3305.41**	**2454**	**4427**	**3872.00**	**3932.90**	**4049.00**
MQTT Connect	8000	0	0.00%	3305.41	2454	4427	3872.00	3932.90	4049.00

图 6-5 8000 并发测试结果图

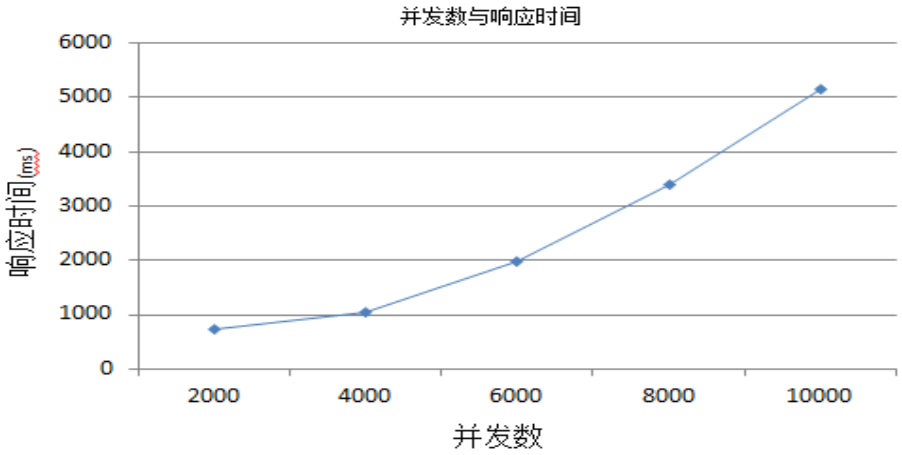

图 6-6 并发量测试结果图

从图 6-6 可以看出，消息系统随着并发数的增加系统响应时间也逐渐增大，能够稳定地支持万级的并发连接，且系统响应时间在 6s 以下，能够满足企业对于物联网环境下设备的并发需求。

通过 Emqtt_benchmark 测试软件分别模拟 10000、20000、40000、60000、80000、100000 的连接数并进行消息推送，测试由四节点组成的消息集群平均的 Cpu 占用率、支持的连接数以及消息推送时延测试结果如表 6.9 所示。

表 6.9 集群负载测试表

连接数	平均 Cpu 占用率	消息时延
10000	8.5%	<1s
20000	14.6%	<1s
40000	19.4%	<1s
60000	25.6%	<1s
80000	32.5%	<1s
100000	41.2%	<1s

由表 6.9 可知消息系统在集群环境下能够处理更多的设备连接数，4 节点集群处理 100000 连接非常稳定，且 Cpu 负载较低均在 50%以下，消息推送时延在 1s 以内，满足了企业对系统的需求。

6.4 本章小结

本章首先给出了系统的测试环境，包括系统在硬件和软件上的要求，然后对系统各个功能模块进行测试，以测试用例的形式给出了功能测试的结果。其次，对系统单节点进行系统连接和消息时延测试，给出了测试方法和测试结果。最后，对系统的集群负载和消息时延进行测试。测试结果表明，系统在功能和性能上均能满足预期，具有较高的工程应用价值。

第七章 总结与展望

7.1 总结

本文主要研究了 MQTT 协议在物联网消息通信方面的应用。结合公司真实的物联网应用场景和定制化的企业需求，本文从需求分析、功能模块设计、具体实现以及功能和性能测试详细地介绍了基于 MQTT 协议的物联网消息系统的设计与实现。本文主要的工作和贡献如下：

（1）研究与分析了物联网消息系统开发过程中使用到的相关协议和技术，包括 MQTT 协议基础概念、常见报文类型以及 MQTT 发布订阅模型、常用于服务端开发的 Netty 技术框架、用于数据存储的 HBase 数据库、消息中间件 Kafka 以及分布式节点开发技术 Akka 框架，为系统的实现奠定了理论和技术基础。

（2）结合 MQTT 协议特点和公司业务需求，对消息系统进行了功能和性能上的需求分析。主要包括实现完整的 MQTT 协议需求、公司对消息系统功能模块的要求、物联网设备安全的需要以及消息系统性能上对并发设备数、消息推送时效性和系统稳定性方面的需求。

（3）根据需求分析，完成了消息系统的整体架构设计和各功能模块的详细设计。使用物模型对物联网设备资源进行统一描述并设计了消息主题格式规范。研究与分析了物联网设备 Session 存储、离线消息存储和分发、认证鉴权、集群管理这些难点问题，并给出了具体的解决方案。

（4）结合设计模块给出了系统各功能模块的详细实现，以类图、时序图、流程图、接口设计等形式展示了实现过程时的详细信息。最后编写测试用例对系统进行功能和性能测试。测试结果表明，消息系统满足预期目标，具有较高的工程应用价值。

7.2 展望

本文设计的基于 MQTT 协议的物联网消息系统在功能和性能上已基本满足系统的业务需求，但考虑到本人能力及时间有限，系统还有一些方面能够继续设计与优化，主要包括：

（1）本系统目前只支持 MQTT 协议，未来还可以实现基于 HTTP 以及自定义消息协议的接口，这样可以根据不同需求更好地支持物联网应用。

（2）系统目前支持的协议版本为 MQTT3.1.1，未来可以做 MQTT5.0 的版本升级，支持 MQTT 协议的新特性比如共享订阅等。

（3）可以设计消息推送的优先级，对于一些要求消息实时性比较高的消息能够优先推送，比如优先推送联网设备告警事件。

致谢

又是一年樱落季节，今年已是在苏的第三个年头。时光总是悄无声息地从指间溜走，仿佛昨天刚收到东大的录取通知书，转眼间就要毕业离开校园奔赴下一个征程。回顾读研的日子，忙碌且充实。在同学和老师的陪伴下，一路走来，倒也顺利。

首先感谢治学严谨、认真负责的孔佑勇老师。孔老师无论是在学习还是生活上都给予我莫大帮助。老师细心指导与耐心的教诲，时时刻刻提醒着我要更加地努力、认真、专注。祝老师和家人在未来的日子里，平安健康，万事顺遂。

其次，感谢景晨阳、晋兴飞、孙长江、娄汕汕等同学在生活和学习上的陪伴，你们幽默、乐观、努力、勤奋的人格魅力会继续鼓励我前行。

然后，感谢张诗莹、周桔、赵华等全体苏州研究院的教师，是你们精心安排的每一场学术活动讲座丰富了我的学习生活。感谢实习期间梁蕤师傅、任慧哲、郑茹等同事在项目上对我的指导与生活上的关心。

同时，感谢百忙之中抽出时间参加论文审阅的专家和老师们，谢谢你们的仔细审阅。

最后，感谢父母、女朋友和其他家人们，谢谢你们一直支持、督促、鼓舞着我前行。希望你们不要再这么劳累，闲下来的时候静静地享受一下生活，未来我会继续努力的。

参考文献

[1] Joshua A.Internet of Things (IoT): The Technology, Architecture and Applications – Prospects in Nigeria[J]. Internet of Things and Cloud Computing, 2020, 8(4): 41-45.
[2] Ibtihal A, Mohammed A. EDTD-SC: An IoT Sensor Deployment Strategy for Smart Cities[J]. Sensors, 2020, 20(24): 7191.
[3] 陈江宁. 工业互联网驱动下的数字资产优化配置[J]. 中国发展观察, 2020, 1(24): 33-36+25.
[4] Sasaki Y, Yokotani T. Performance Evaluation of MQTT as A Communication Protocol for IoT and Prototyping[J]. Advances in Technology Innovation, 2019, 4(1): 21-29.
[5] 羊月祺. 基于物联网的医疗设备运行环境与状态监测系统设计与实现[D]. 东南大学, 2019.
[6] AI-Masri E, Kalyanam K R, Batts J, et al. Investigating Messaging Protocols for the Internet of Things (IoT)[J]. IEEE Access, 2020, 8(1): 94880-94911.
[7] Kavitha J M, Rajasekaran T. Comparison of Integrated Sensor Data With Cloud Using Zigbee With MQTT and CoAP Protocols In Real Time Applications[J]. Journal of Critical Reviews, 2020, 7(11): 988-994.
[8] Sharu B, Dilip K. Distance-based Congestion Control Mechanism for CoAP in IoT[J]. IET Communications, 2020, 14(19): 3512-3520.
[9] 吴晗. 基于 AMQP 的消息中间件的设计和实现[D]. 东南大学, 2019.
[10] Naik N. Choice of Effective Messaging Protocols for IoT Systems: MQTT, CoAP, AMQP and HTTP[C]. 2017 IEEE International Systems Engineering Symposium (ISSE). Austria: Vienna, 2017. 1-7.
[11] Imane S, Tomader M, Nabil H, et al. Comparison Between CoAP and MQTT in Smart Healthcare and Some Threats[C]. 2018 International Symposium on Advanced Electrical and Communication Technologies (ISAECT). Morocco: Rabat, 2018. 1-4.
[12] 金成明, 刘雪松, 杨睿, 崔伟, 李涛. 基于 CoAP 与 MQTT 的配电物联网通信架构设计[J]. 电气自动化, 2020, 42(05): 102-105.
[13] Eridani D, Martono K T, Hanifah A A, et al. MQTT Performance as a Message Protocol in an IoT based Chili Crops Greenhouse Prototyping[C]. 2019 4th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE). Indonesia: Yogyakarta, 2019. 184-189.
[14] Xin Wu, Ning Li. Improvements of MQTT Retain Messgae Storage Mechanism[C]. 2018 2nd IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC). China: Xi'an, 2018. 957-961.
[15] Sadeq A S, Hassan R, Al-rawi S S, et al. A Qos Approach for IoT Environment Using MQTT Protocol[C]. 2019 International Conference on Cybersecutiry(ICoCSec). Malaysia: Negeri Sembilan, 2019. 59-63.
[16] Tantitharanukul N, Osathanunkul K, Kittikorn H, et al. MQTT-Topics Management System For Sharing of Open Data[C]. 2017 International Conference on Digital Arts, Media and Technology(ICDAMT). Thailand: Chiang Mai, 2017. 62-65.
[17] Harsha M S, Bhavani B M, Kundhavai K R, et al. Analysis of vulnerabilities in MQTT security using Shodan API and implementation of its countermeasures via authentication and

ACLs[C]. 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI). India: Bangalore, 2018. 2244-2250.
[18] Tanomwong N, Jaikaeo C. Adaptive Middleware for Costly Data Generation over MQTT[C]. 2018 5th International Conference on Business and Industrial Research(ICBIR). Thailand: Bangkok, 2018. 63-68.
[19] Matic M, Antic M, Ivanovic S, et al. Scheduling Messages Within MQTT Shared Subscription Group In The Clustered Cloud Architecture[C]. 2020 28th Telecommunications Forum(TELFOR). Serbia: Belgrade, 2020. 1-4.
[20] Nurwarsito H, Yulihardi F S. Obstructive Sleep Apnea Patient's Heart Beat Monitoring System from Android Smartphone Using MQTT Protocol[J]. International Journal of Innovative Technology and Exploring Engineering (IJITEE), 2020, 9(11): 265-270.
[21] 魏井辉, 吕明. 基于 Netty 通信的消息推送系统的设计与实现[J]. 工业控制计算机, 2020, 33(12): 57-59.
[22] 任培花, 苏铭. 基于 Kafka 和 Storm 的车辆套牌实时分析存储系统[J]. 计算机系统应用, 2019, 28(10): 74-79.
[23] Yan Li, Zhang Zheqing, Yang Dan. Temporal RDF(S) Data Storage and Query With HBase[J]. Journal of Computing and Information Technology, 2019, 27(4): 17-30.
[24] Malleswari TYJ, Vadivu G. Adaptive Deduplication of Virtual Machine Images Using Akka Stream to Accelerate Live Migration Process in Cloud Environment[J]. Journal of Cloud Computing, 2019, 8(1): 1-12.
[25] Upadhyay U, Jain A. Elastic Akka Cluster with Websockets[J]. International Journal of Innovative Technology and Exploring Engineering (IJITEE), 2020, 9(9): 501-505.
[26] 孙海滨, 张敬超. 基于 MQTT 协议的跨平台工业级物联网消息传输系统实现与设计[J]. 软件, 2020, 41(08): 168-171.
[27] Manas M, Sinha A, Sharma S, et al. A Novel Approach for IoT Based Wearable Health Monitoring and Messaging System[J]. Journal of Ambient Intelligence and Humanized Computing, 2019, 10(7): 2817-2828.
[28] Tuna G, Kogias D G, Gungor C, et al. A Survey on Information Security Threats and Solutions for Machine to Machine (M2M) Communications[J]. Journal of Parallel and Distributed Computing, 2017, 109(11): 142-154.
[29] Vatsal G, Sonam K, Neelam T. MQTT Protocol Employing IoT Based Home Safety System With ABE Encryption[J]. Multimedia Tools and Applications, 2020, 80(2): 2931-2949.
[30] 张杰, 刘凯, 周立军. 采用 Redis 高并发应用系统设计与实现方法[J]. 计算机与数字工程, 2020, 48(05): 1222-1226.
[31] Peng Li, Baozhou Luo, Wenjun Zhu, He Xu. Cluster-based Distributed Dynamic Cuckoo Filter System for Redis[J]. International Journal of Parallel, Emergent and Distributed Systems, 2020, 35(3): 340-353.
[32] 王建永, 何旻诺, 方宽, 唐乐. 分布式集群系统中 Session 会话的高效共享方法研究[J]. 电子设计工程, 2020, 28(06): 136-139+148.
[33] Wang Haiyang, Liu Dawei, Li Xuemei. A Distributed Management System for Massive GIS Data Using HBase[J]. Frontiers in Artificial Intelligence and Applications, 2018, 309(1): 787-793.
[34] Sharma M, Bundele M. Analysis of NoSQL Schema Design Approaches Using HBase for GIS Data[J]. Procedia Computer Science, 2019, 152(1): 59-65.

[35] 徐龙光, 何顶新. 基于 Netty 的消息推送服务器集群设计与实现[J]. 软件导刊, 2018, 17 (04) : 118-119+123.
[36] 刘英杰. 基于 Kafka 和时序数据库的物联网数据测控系统设计与实现[D]. 山东大学, 2020.
[37] 郝鹏海, 徐成龙, 刘一田. 基于 Kafka 和 Kubernetes 的云平台监控告警系统[J]. 计算机系统应用, 2020, 29(08): 121-126.
[38] Dey S, Hossain A. Session-Key Establishment And Authentication in A Smart Home Network Using Public Key Cryptography[J]. IEEE Sensors Letters, 2019, 3(4): 1-4.
[39] 闫宏强, 王琳杰. 物联网中认证技术研究[J]. 通信学报, 2020, 41(07): 213-222.
[40] 刘成. 基于服务器集群的负载均衡系统的设计与实现[D]. 南京邮电大学, 2020.
[41] 李斐. 基于 Akka 的分布式集群运维系统设计与实现[D]. 东南大学, 2017.

作者简介

1. 基本情况

孙磊，男，河南信阳人，1995 年 11 月出生，东南大学软件学院软件工程专业 2018 级硕士研究生。

2. 教育背景

2013.09-2017.07　河南财经政法大学，　本科，　专业：计算机科学与技术
2018.09-至今　　东南大学，硕士，专业：软件工程