摘要

Abstract

[摘要 1](#_Toc435466227)

[Abstract 1](#_Toc435466228)

[1 绪论 2](#_Toc435466229)

[1.1 研究背景及意义 2](#_Toc435466230)

[1.2 国内外研究现状 3](#_Toc435466231)

[1.3 研究目标及内容 3](#_Toc435466232)

[1.4 本文章节安排 4](#_Toc435466233)

[2 相关技术分析 4](#_Toc435466234)

[3 系统需求分析与架构设计 5](#_Toc435466235)

[3.1 业务分析 5](#_Toc435466236)

[3.2 系统需求分析 5](#_Toc435466237)

[3.2.1 功能性需求 5](#_Toc435466238)

[3.2.2 非功能性需求 5](#_Toc435466239)

[3.3 系统架构设计 5](#_Toc435466240)

[3.3.1 技术架构 5](#_Toc435466241)

[3.3.2 逻辑架构 5](#_Toc435466242)

[3.3.3 部署视图 5](#_Toc435466243)

[3.3.4 系统实现环境 5](#_Toc435466244)

[3.4 本章小结 5](#_Toc435466245)

[4 系统核心功能模块及关键技术研究与实现 5](#_Toc435466246)

[4.1 邻里社交子系统 5](#_Toc435466247)

[4.1.1 邻里社交模型 5](#_Toc435466248)

[4.1.2 客户端状态管理 6](#_Toc435466249)

[4.1.3 消息缓存、推送机制 6](#_Toc435466250)

[4.2 基于微服务的社区服务管理子系统 6](#_Toc435466251)

[4.2.1 微服务的？ 6](#_Toc435466252)

[4.2.2 基于Docker的微服务容器化 6](#_Toc435466253)

[4.2.3 基于Dubbo的服务治理 6](#_Toc435466254)

[4.3 统一资源管理平台（技术，算法问题） 7](#_Toc435466255)

[4.3.1 服务注册、发现 7](#_Toc435466256)

[4.3.2 日志管理 7](#_Toc435466257)

[4.3.3 容器资源伸缩性 7](#_Toc435466258)

[4.4 本章小结 7](#_Toc435466259)

[5 系统测试及应用 7](#_Toc435466260)

[5.1 系统测试 7](#_Toc435466261)

[5.1.1 功能测试 7](#_Toc435466262)

[5.1.2 性能测试 7](#_Toc435466263)

[5.2 系统应用 7](#_Toc435466264)

[5.2.1 运行实例 7](#_Toc435466265)

[5.2.2 应用效果及分析 7](#_Toc435466266)

[5.3 本章小结 7](#_Toc435466267)

[6 总结与展望 7](#_Toc435466268)

[6.1 工作总结 7](#_Toc435466269)

[6.2 下一步工作 8](#_Toc435466270)

[参考文献 8](#_Toc435466271)

[致谢 8](#_Toc435466272)

[攻读学位期间已发表的学术论文 8](#_Toc435466273)

# 绪论

## 研究背景及意义

社区是若干社会群体或社会组织聚集在某一个领域里所形成的一个生活上相互关联的大集体，是社会有机体最基本的内容，是宏观社会的缩影。 构成社区要具备五个要素：1）有聚居的一群人；2）有一定的地域；3）有一定的生活服务设施； 4）居民群具有特定的文化背景和生活方式，居民群之间发生种种社会关系；5）为谋求规章制度具体落实，产生各种社会群体和机构。 社区蕴藏着巨大的资源优势，具有经济性、社会化、心理支持与影响、社会控制和社会参与等多种功能。

社区治理是政府与社区组织、社区公民共同管理社区公共事务的活动。在我国, 城市社区治理模式由行政型社区向合作型社区和自治型社区的发展过程, 是社会经济体制改革和社会结构调整在城市社区发展中的一种反映,它代表着我国城市社区发展的方向。建立在合作主义基础上的新型政府与社会关系、社区制逐步取代单位制以及城市街道制体制的改革,代表着我国社区发展与制度创新的基本思路。

目前，我国在社区治理方面仍然存在如下的问题：1）治理主体定位不清：我国社区自治程度普遍较低，政府往往通过行政命令和强制手段直接控制社区的治理工作。2）社区管理资源匮乏：我国城市社区普遍面临资源匮乏、配套支持难以满足社区治理需要的困境。3）社区参与不足：社区治理的目标就是通过多元的社区组织实现对社区治理的参与，治理的主体既包括居民个人，政府组织，也包括非营利组织和市场组织。面对社会发展以及信息化的需要，建立统一、高效的社区治理模型成为提高社区管理质量，解决社区群众参与不足的迫切需要。

本文以某市社区实名制邻里社交平台为背景，该系统建设了统一的实名制邻里社交平台，为社区中不同角色的居民（管理群众、服务群众、生活群众）提供了社区管理、服务、社交途径。例如，居民与社区管理人员之间可通过平台进行线上问题咨询，周边商铺、商户间可通过平台对居民提供服务，居民与居民之间可通过平台增进相互间沟通与了解。

## 国内外研究现状

邻里社交是指以社区为单位的，社区内居民间以聊天、群组等方式进行的社交。在如今多种多样的基于位置、兴趣、话题、职业的社交网络中，邻里社交的出现填补了人们在这一方面的空白。如果Facebook意味着家庭和朋友，Twitter意味着名人和线上朋友，Linkedin代表着职业化网络，那么Nextdoor则意味着和邻居的亲密互动，而这种真实的互动建立在基本的信任之上。

未完

## 研究目标及内容

本文以实验室承接的社区邻里社交系统为背景，针对提高社区管理质量，增加居民参与度的需求，结合即时通讯，流式处理，Zookeeper，Dubbo等技术，设计并实现社区邻里社交系统。并通过测试及实际线上运行，验证其可行性和有效性。

围绕上述研究目标，本文的研究内容主要有：

1. 业务及需求分析：
   1. 社区内居民间需要拥有能够进行即时聊天，状态分享，信息公告交流的线上平台。
   2. 系统需要拥有良好的可扩展性及鲁棒性，能够轻松应对
2. 相关技术分析研究：分析并研究基于MQTT，WebSocket协议的即时通讯，Storm，Dubbo，Zookeeper，Thrift等相关技术。
3. 系统的设计与实现：设计并实现社区邻里社交系统，该系统可有效支持居民间社交需求。
   1. 结合MQTT，WebSocket协议，设计并实现即时通讯子系统，使用户可通过移动、PC终端进行实时通讯。
   2. 结合Dubbo以及Docker，实现基于服务发现、治理、注册的微服务系统架构。
   3. 结合Apache Yarn，并对其进行改进，使其支持Long-lived服务，并以此为基础实现对所有微服务的统一资源管理分配。

## 本文章节安排

社区治理系统元数据模型与

# 相关技术分析

综述用到的所有的技术，分析，对比，研究，有自己的观点。

解决整篇论文的技术基础

# 系统需求分析与架构设计

## 业务分析

## 系统需求分析

### 功能性需求

### 非功能性需求

## 系统架构设计

### 技术架构

### 逻辑架构

### 部署视图

### 系统实现环境

## 本章小结

# 系统核心功能模块及关键技术研究与实现

## 邻里社交子系统

### 邻里社交模型

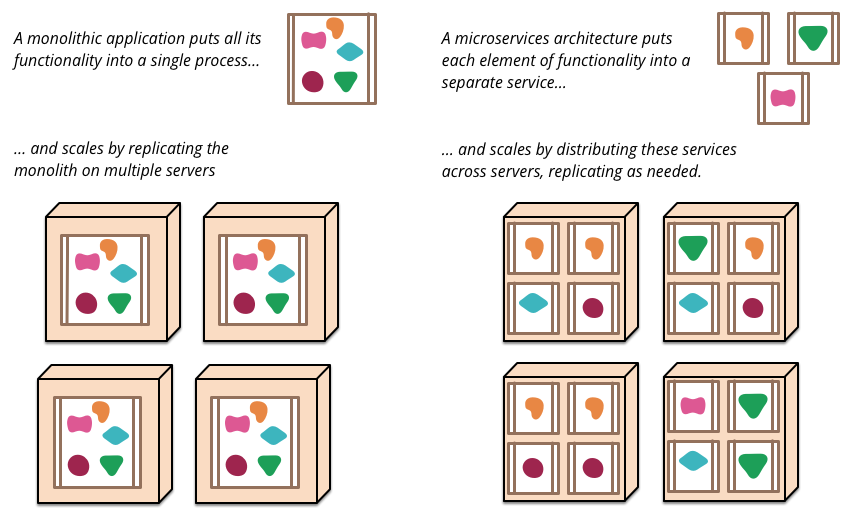
### 客户端状态管理

### 消息缓存、推送机制

## 基于微服务的社区服务管理子系统

微服务是一种软件架构风格，它代表着一种开发多个运行在独立进程、环境中的使用轻量级通信协议进行交互的小型服务来代替传统的单一应用的大型应用的软件架构风格。这些微服务是围绕着业务功能进行划分的，并且可独立部署、升级、管理。

为了更好的解释微服务架构，我们将其与传统的单一应用架构进行比较。企业级应用通常包括三个部分：1）客户端接口（通常包括运行在客户浏览器中的HTML页面以及javascript代码），2）数据库（通常为关系型数据库中的多个表集合），3）服务器端应用（处理HTTP请求，执行业务逻辑，获取更新数据）。而服务器端应用通常是一个逻辑上的集合，对于服务器端的任何修改都会导致对整个服务器端应用的重新编译以及部署。单一应用架构是可行的，但随着应用范围以及复杂度的增加，其缺点也显得越来越明显：每次更改都要求整个服务器端重新编译以及部署，在扩容时也只能够对整个应用进行扩容，而不能够只针对资源需求更低的一个模块来进行扩容。

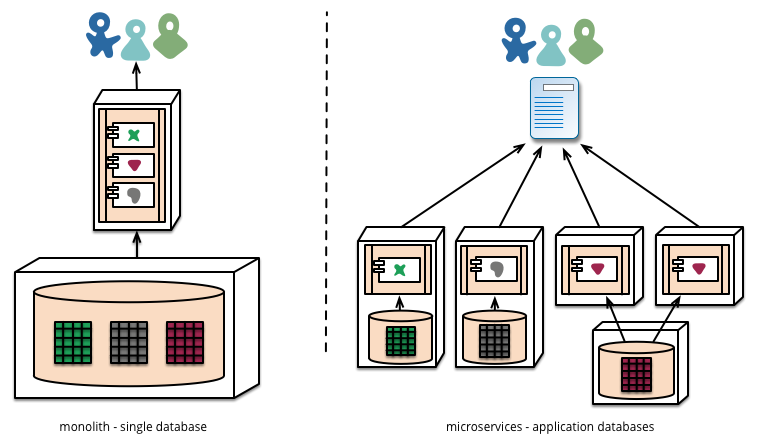


上述的问题就引出了一个新的软件架构：微服务架构。微服务架构将传统的单一应用以一组独立的服务来构成，与此同时，这些服务是可以单独部署以及水平扩展的。每一个这样的服务都拥有着严格的模块边界，甚至允许不同的服务使用不同的开发语言来进行开发。

微服务架构会使用依赖包，但其主要通过将软件分割成服务来进行模块化。我们定义依赖包为链接入程序并通过in-memory程序调用使用的程序模块，而服务为使用RPC或者网络请求进行交互的进程。使用服务作为系统模块有两大优势：首先服务可独立部署，这样对一个单独的模块进行更改不会影响到系统中的其他模块，另一个优势则是服务间通过显示的远程调用来进行通讯，从而减小程序间的接口定义难度。然而，微服务架构也是有缺点的。首先，远程调用比进程内函数调用开销更大，其次，将程序职责在服务间进行迁移的难度也会更加的复杂。

去中心化治理：中心化治理的会导致整个程序开发使用单一的技术和平台。然而，这样的解决方案是受到很大限制的，不是每个问题都能通过单一的技术来解决。针对具体的问题选择最适合的技术显然是一个更加的做法。微服务架构则将这样的方法带入了软件开发之中。

去中心化数据管理：在对概念模型以及业务逻辑进行去中心化的同时，微服务架构同样对数据进行了去中心化的管理。每个微服务都独自管理自己的数据，这意味着微服务可根据具体的业务需求选择合适的数据库（关系型或非关系型），但同时也意味着在服务间数据一致性问题会更加难以处理。



测试部署自动化：随着云平台的日渐成熟，测试部署自动化在近年中发展迅速。而云平台大大减少了构建、部署、运行微服务的复杂度。

预防软件错误：微服务架构需要在设计时将服务异常纳入考虑范围，由于任何服务调用都可能因为服务提供者失效或网络失联而失败，因此客户端需要很好地处理这一类情况。

迭代式设计：当尝试将一个系统服务化时，我们将面临一个至关重要的问题：怎样将系统划分成不同的服务？应该使用怎样的原则来进行划分？由于一个服务的关键要素是独立部署升级，因此在系统划分时应该考虑在其需要重构时不会影响到其他的协作者。

### 微服务的？

### 基于Docker的微服务容器化

### 基于Dubbo的服务治理

## 统一资源管理平台（技术，算法问题）

### 服务注册、发现

### 日志管理

### 容器资源伸缩性

## 本章小结

# 系统测试及应用

## 系统测试

### 功能测试

### 性能测试

## 系统应用

### 运行实例

### 应用效果及分析

## 本章小结

# 总结与展望

## 工作总结

## 下一步工作

参考文献

致谢

攻读学位期间已发表的学术论文

- 相关技术分析

-- 容器化技术 (Docker & Kubernetes)

-- 资源管理平台 (YARN, Mesos, Borg, Corona)

-- 微服务架构 (Microservice architecture)

-- 即时通讯协议 (MQTT, XMPP) -------

-- 消息中间件 (Rabbitmq, 中间件的使用场景) -----

- 核心功能模块及关键技术研究与实现

-- 邻里社交子系统

--- 邻里社交模型

1. 描述社交模型 (邻里间社交、实名制、以户为单位，一对一、群组、不同身份角色交互)

2. 定义消息结构 （支持文本、语音、视频等信息）

3. 描述一部分具体用例 （一对一聊天，群组聊天，居民咨询居委会人员政策、流程问题等）

--- 社交XXX（参与者？）状态管理

1. 社交XXX（参与者？）状态管理需求 （管理客户端状态（实现广告推送，在线统计等），消息推送基石）

2. 管理方案 （心跳 + expired status(redis expired key))

3. 状态管理架构、流程

4. 具体实现

对ONLINE\_STATUS\_SERVER进行描述，怎样进行客户端状态管理，管理的机制，设计方案

--- 社交消息缓存、推送机制

1. 消息缓存队列定义

2. 消息分发微服务

3. 用户在线、离线状态判断及推送决策

4. 具体架构及实现 (高可用)

消息中间件的使用，队列的定义，消息分发微服务，用户在线、离线状态判断

-- 基于微服务的社区服务管理子系统

--- 服务划分及容器化

1. 对系统中服务进行描述，划分

2. 阐述相应的技术选型

3. 架构描述

4. 微服务容器化及部署策略（Docker + Kubernetes)

--- 服务注册与发现

1. 服务的注册与发现 (Service level) （查询与治理的关系，SOA上确定）

--- 服务治理

1. 可放入监控、调度

2. 服务的调用统计、容量评估

3. 服务的软负载均衡（软负载均衡算法，基于调用统计与机器性能的调度策略）

-- XXX（我做的工作的要点）的资源管理服务子系统（这里放到4.1）

先阐述问题，技术路线和解决方案，应用效果

先提一个资源管理服务的模型（资源管理需要做哪些事情，怎么做）

小论文内容，更改YARN使其支持Long-lived service

加入新的Docker容器管理框架Kubernetes的使用

--- 服务注册与发现 (framework level)

--- 日志管理（结合Docker与不结合Docker解决方案） 日志收集、日志聚合

--- 容器资源伸缩性（结合Docker以及不结合Docker）（scaling设计，auto-scaling机制应用）