**密级：保密期限：**



**硕士学位论文**



**题目：基于分布式开源技术下的企业移动协同办公服务的设计与实现**

**学号：**

**姓名：**

**专业：**

**导师：**

**学院：**

**年 月日**

独创性（或创新性）声明

本人声明所呈交的论文是本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中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京邮电大学或其他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。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名： 日期：

关于论文使用授权的说明

本人完全了解北京邮电大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属北京邮电大学。学校有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许学位论文被查阅和借阅；学校可以公布学位论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存、汇编学位论文。

本学位论文不属于保密范围，适用本授权书。

本人签名： 日期：

导师签名： 日期：

基于分布式开源技术下的企业移动协同办公服务的实现

摘 要

近几年，随着智能终端迅速的普及，移动互联网用户呈现爆炸式的增长。庞大的用户基数，对移动端软件的后台服务提出了新的调战。眼下用户沟通相关的APP中已有了微信、微博等巨头，市场趋于饱和。然而建立在移动互联网上的企业协同与自动化办公却还没有优秀的产品出现。

之前的PC端的WEB服务，以SSH框架为主。用户从请求到业务处理再到数据持久化过程都被集成到了一个服务中。当在线用户数激增的时候，单点的服务很难保证满足大量的并发用户访问。跟据2/8原则，一个软件的业务，用户80%时间在使用其中20%的功能，所以一个服务并不是所有的业务都会需要很高的性能。过度集中化的框架，很难去跟据用户数变化去调整主要业务的性能。

本文通过介绍分布式服务的思想，结合目前主流的开源社区提供的分布式中间件、服务等技术。设计一个简易的、快速上手的一个高性能分布式服务框架。

论文首先对国内外的分布式技术的发展现状分析。给出一些互联网前沿公司的优秀开源的分布式组件介绍。然后对分布式的思想与特点进行系统性的阐述，结合作者在公司中一个基于移动互联网的企业协同产品设计与开发，来详述如何通过这些分布式组件来搭建一个高可用、高性能且易于开发的分布式服务框架。

本文给出的分布式系统框架，可以作为一个通用型的产品服务开发框架来参考或使用。

**关键词**：企业协同 移动互联网 分布式服务 开源技术

THE ENTERPRISE MOVEMENT COLLABORATION OFFICE SERVICE BASED ON THE DISTRIBUTED OPENSOURCE TECHNOLOGY

ABSTRACT

In recent years, with the rapid popularization of intelligent terminals, mobile Internet users show explosive growth. Large user base, mobile client software background service proposed new tune war. Now users communicate relevant APP already a micro-channel, microblogging and other giants, market saturation. However, based on the mobile Internet business collaboration and office automation products appear still not good.

PC before the end of WEB services to SSH-based framework. From a user request to the business process and then to process data persistence are integrated into one service. When the surge in the number of users online, when a single point of service is difficult to ensure access to meet a large number of concurrent users. According to 2/8 with the principle of a software business, 80% of users use the time in which 20% of the features, so the service is not a high performance all business will need. Framework excessive centralization, it is difficult to talk to change according to the number of users to adjust the performance of the main business.

This paper describes the thinking of distributed services, combined with the current mainstream of the open source community to provide a distributed middleware, services and other technologies. Design a simple, quick start of a high-performance distributed service framework.

Firstly, the paper analyzes current development situation of distributed technologies. Give some cutting-edge Internet companies excellent open source distributed components introduced. Then distributed the idea and characteristics of systematic exposition, combined with the author in the company a mobile Internet-based enterprise collaborative product design and development, to detail how these distributed components to build a highly available, high performance and easy distributed service framework development.

Distributed systems framework presented here can be used as a general-purpose product and services development framework to reference or use.

**KEY WORDS**：Enterprise collaboration, Mobile Internet, Distributed Services， OpenSource

目 录

[第一章 引言 1](#_Toc451011669)

[1.1 背景 1](#_Toc451011670)

[1.2 课题任务 2](#_Toc451011671)

[1.2.1 课题内容 2](#_Toc451011672)

[1.2.2 本人承担任务 3](#_Toc451011673)

[1.3 论文结构 3](#_Toc451011674)

[第二章 关键技术介绍 4](#_Toc451011675)

[2.1 Thrift — RPC Framework 4](#_Toc451011676)

[2.2 分布式服务框架 Zookeeper 4](#_Toc451011677)

[2.3 服务器缓存服务 Redis 5](#_Toc451011678)

[2.4 NoSql数据库 MongoDB 6](#_Toc451011679)

[2.5 本章小结 7](#_Toc451011680)

[第三章 分布式系统介绍 8](#_Toc451011681)

[3.1 分布式系统的定义 8](#_Toc451011682)

[3.2 CAP定律 9](#_Toc451011683)

[3.3 现代分布式系统的特点 9](#_Toc451011684)

[第四章 工作圈的需求分析 12](#_Toc451011685)

[4.1 系统用户角色分析 12](#_Toc451011686)

[4.2 系统功能需求分析 12](#_Toc451011687)

[4.3 系统功能的UML用例图（可选部分） 14](#_Toc451011688)

[4.4 系统的非功能需求分析 14](#_Toc451011689)

[4.5 本章小结 14](#_Toc451011690)

[第五章 工作圈分布式系统的设计 15](#_Toc451011691)

[5.1 服务模块化的设计 15](#_Toc451011692)

[5.2 分布式服务与自动化配置中心 16](#_Toc451011693)

[5.3 数据分片(Data Sharding)与数据索引中心 17](#_Toc451011694)

[5.4 Rest server设计 18](#_Toc451011695)

[第六章 系统概要设计 20](#_Toc451011696)

[6.1 自动化配置管理中心模块（configcenter） 20](#_Toc451011697)

[6.1.1 自动化配置管理中心模块概述 20](#_Toc451011698)

[6.1.2 自动化配置管理中心模块类图 21](#_Toc451011699)

[6.1.3 Env类概述 21](#_Toc451011700)

[6.1.4 ZkHelper类概述 22](#_Toc451011701)

[6.2 数据库操作模块（mongo） 22](#_Toc451011702)

[6.2.1 数据库操作模块概述 22](#_Toc451011703)

[6.2.2 数据库操作模块类图 23](#_Toc451011704)

[6.2.3 DbPool类概述 23](#_Toc451011705)

[6.2.4 TBaseDao类概述 24](#_Toc451011706)

[6.3 缓存操作模块（cache） 25](#_Toc451011707)

[6.3.1 缓存操作模块概述 25](#_Toc451011708)

[6.3.2 缓存操作模块类图 26](#_Toc451011709)

[6.3.3 NutcrackerCenter类概述 26](#_Toc451011710)

[6.3.4 NutcrackerCenterAop类概述 27](#_Toc451011711)

[6.3.5 ModRedisCenterAop类概述 27](#_Toc451011712)

[6.3.6 ModRedisCenterAop类概述 28](#_Toc451011713)

[6.3.7 LockCacheCenter类概述 28](#_Toc451011714)

[6.3.8 LockCache类概述 29](#_Toc451011715)

[6.4 Id列表基础服务模块（IdList） 29](#_Toc451011716)

[6.4.1 Id列表基础服务模块概述 29](#_Toc451011717)

[6.4.2 IdList系统模块类图 30](#_Toc451011718)

[6.4.3 IdListClient 类概述 30](#_Toc451011719)

[6.4.4 IdListServiceImpl 类概述 31](#_Toc451011720)

[6.4.5 CommonCache类概述 32](#_Toc451011721)

[6.4.6 IdListCache类概述 32](#_Toc451011722)

[6.4.7 IdListDao类概述 33](#_Toc451011723)

[6.5 账户系统（Acount）模块 33](#_Toc451011724)

[6.5.1 账户系统模块概述 33](#_Toc451011725)

[6.5.2 账户系统模块类图 40](#_Toc451011726)

[6.5.3 JsonController类概述 40](#_Toc451011727)

[6.5.4 AccountWebController类概述 41](#_Toc451011728)

[6.5.5 AccountRestController类概述 41](#_Toc451011729)

[6.5.6 Client类概述 42](#_Toc451011730)

[6.5.7 AccountServiceImpl类概述 42](#_Toc451011731)

[6.5.8 CspService类概述 43](#_Toc451011732)

[6.5.9 TokenCache类概述 44](#_Toc451011733)

[6.5.10 UserMongoDao类概述 44](#_Toc451011734)

[6.6 圈子系统（quan）模块 44](#_Toc451011735)

[6.6.1 圈子系统模块概述 44](#_Toc451011736)

[6.6.2 圈子系统模块流程图 45](#_Toc451011737)

[6.6.3 JsonController类概述 47](#_Toc451011738)

[6.6.4 RestTeamController类概述 47](#_Toc451011739)

[6.6.5 WebTeamController类概述 47](#_Toc451011740)

[6.6.6 RestTopicController类概述 48](#_Toc451011741)

[6.6.7 WebTopicController类概述 48](#_Toc451011742)

[6.6.8 RestMemberController类概述 49](#_Toc451011743)

[6.6.9 WebMemberController类概述 49](#_Toc451011744)

[6.6.10 TeamServiceImp类概述 50](#_Toc451011745)

[6.6.11 TopicServiceImp类概述 50](#_Toc451011746)

[6.6.12 MemberServiceImp类概述 51](#_Toc451011747)

[6.7 关系系统（Relation）模块 52](#_Toc451011748)

[6.7.1 关系系统模块概述 52](#_Toc451011749)

[6.7.2 关系系统模块类图 60](#_Toc451011750)

[6.7.3 JsonController类概述 60](#_Toc451011751)

[6.7.4 WebTeamMemberServiceController类概述 61](#_Toc451011752)

[6.7.5 RestTeamMemberServiceController类概述 61](#_Toc451011753)

[6.7.6 BasicTeamMemberService类概述 62](#_Toc451011754)

[6.7.7 RestBasicTeamMemberService类概述 62](#_Toc451011755)

[6.7.8 TokenCache类概述 63](#_Toc451011756)

[6.7.9 TeamMemberDao类概述 63](#_Toc451011757)

[6.8 评论系统（Comment）模块 64](#_Toc451011758)

[6.8.1 评论系统模块概述 64](#_Toc451011759)

[6.8.2 评论系统模块类图 65](#_Toc451011760)

[6.8.3 JsonController类概述 65](#_Toc451011761)

[6.8.4 WebController类概述 66](#_Toc451011762)

[6.8.5 RestController类概述 67](#_Toc451011763)

[6.8.6 Client类概述 67](#_Toc451011764)

[6.8.7 CommentServiceImpl类概述 67](#_Toc451011765)

[6.8.8 ICommentCache类概述 68](#_Toc451011766)

[6.8.9 ICommentSortCache类概述 68](#_Toc451011767)

[6.8.10 CommentDao类概述 69](#_Toc451011768)

[7 第N章 结束语 70](#_Toc451011769)

[7.1 N．1 论文工作总结 70](#_Toc451011770)

[7.2 N．2 问题和展望 70](#_Toc451011771)

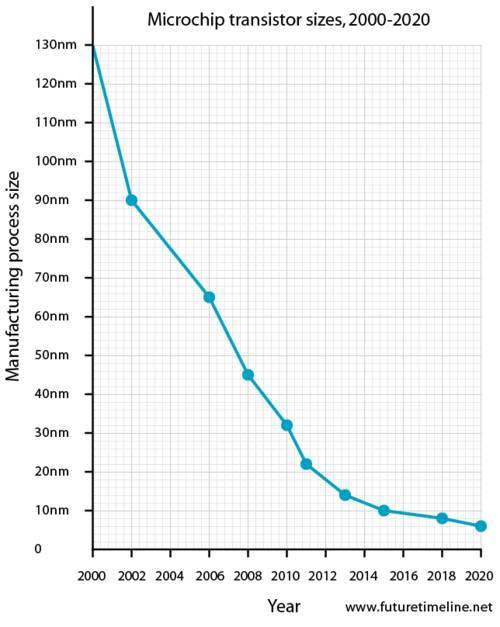
[8 附录1： 72](#_Toc451011772)

# 第一章 引言

## 背景

在一个理想的世界里，我们应该只有一个一致性模型。但是多路处理器和分布式系统中的一致性问题是一个非常难以解决的问题。当系统大多还都是单体应用，处理器性能提升还是靠缩小晶体管体积的年代，一致性问题还不是一个非常大的问题，但是从90年代开始，互联网公司大量出现，互联网公司处理的数据量远远超过了人类历史上任何一个时期的传统企业所需要处理的数据量，大型的互联网公司甚至开始自己采购零部件自己组装服务器，服务器市场上排名除了IBM，HP之外还出现了“Others”。NoSQL已经被互联网公司广泛接受，Micro Service的出现让数据和计算的分布比历史上任何时期都更加的水平，在这个背景下，分布式系统给一致性问题带来了巨大的挑战。

实际上在互联网蓬勃发展之前，有些有远见的科学家们从70年代就开始在多路处理器级别上研究一致性模型。过去的10年间，计算机处理器的速度提升越来越困难，摩尔在1965年告诉我们12个月晶体管密度提升一倍，计算性能提升一倍(1975修正为24个月)，但是2002年1月Moore在Intel开发者论坛上提出Moore’s Law预计会在2017年终结，一旦晶体管体积进入皮米级别，晶体管将不得不接近原子大小，我们很快将无法突破物理的基础限制。事实上从2012年开始晶体管密度和主频速度的提高速度明显放慢。人们不得不开始转向横向伸缩，提高处理器的并行处理能力。下图显示了在2012年出现了晶体管体积缩小速度的突然放缓。[1]



处理器的现状我们已经分析了，眼下的处理器单线程的处理能力的增长已是十分的缓慢，然而并行处理能力却是得到了很大的强化，使得Moore’s Law对于目前处理器的计算能力预见是依然有效的。为了能充分利用起处理器强大的并行能力，对处理任务的分解和并行计算有了新的要求。在操作系统的领域中，对处理的管理是以线程为单位的，一个线程只能被分配到处理器中的一个处理核心来执行。现代的服务器的处理器，核心数最高已达12而等效的处理线程数高达96（IBM 即将发布的Power8系列）。以前的集成化式的系统很难充分的挖掘现在服务器的运算能力，于是新的思路出现了。

1. 对系统进行模块化的拆分。将不通的模块服务独立出来，单独部署。这样在一个服务器上可以运行多个模块化的服务，通过拆分运行来充分利用服务器能力。
2. 利用虚拟技术，在一个物理服务器上虚机出多台虚拟服务器。可将服务同时部署到多台服务器上运行。通过虚拟化一来充分利用服务器。

以上的两点正是分布式计算中的模块化与横向扩展性的一个应用。对系统进行分布式的重构无疑是现在也是将来的互联网服务的主要方向。

## 课题任务

### 1.2.1 课题内容

开发一个基于分布式服务的高性能企业协同移动应用的后台服务。服务要根据业务内容划分模块，同时对整体服务的性能要做到可以动态的横向扩展。

### 1.2.2 本人承担任务

研究开源的分布式组件。设计和实现一个高性能，支持弹性扩展的易于维护的分布式服务框架。

实现分布式服务中的核心服务。

## 论文结构

本文共分N章，内容安排如下：

第一章 引言，介绍本课题的意义、任务、预期目标等。

第二章 XXXX，XXXX

第三章 XXXX，XXXX

第N章 结束语，对本文工作进行全面总结，给出本文所取得的成果，指出存在的不足和改进方向。

# 第二章 关键技术介绍

随着开源社区的兴起，开源成为了计算行业中的一股中间力量。越来越多的优秀技术诞生于开源社区，成长于开源社区，开源社区给了程序以更多的选择。本章选择介绍Apache社区的两个顶级项目Thrift和Zookeeper，还有优秀的Redis与MongoDB。

## Thrift — RPC Framework

Apache Thrift 是 Facebook 实现的一种高效的、支持多种编程语言的远程服务调用的框架。

目前流行的服务调用方式有很多种，例如基于 SOAP 消息格式的 Web Service，基于 JSON 消息格式的 RESTful 服务等。其中所用到的数据传输方式包括 XML，JSON 等，然而 XML 相对体积太大，传输效率低，JSON 体积较小，新颖，但还不够完善。本文将介绍由 Facebook 开发的远程服务调用框架 Apache Thrift，它采用接口描述语言定义并创建服务，支持可扩展的跨语言服务开发，所包含的代码生成引擎可以在多种语言中，如 C++，Java，Python，PHP，Ruby，Erlang，Perl，Haskell，C#，Cocoa，Smalltalk 等创建高效的、无缝的服务，其传输数据采用二进制格式，相对 XML 和 JSON 体积更小，对于高并发、大数据量和多语言的环境更有优势。[2]

## 分布式服务框架 Zookeeper

Zookeeper 分布式服务框架是 Apache Hadoop 的一个子项目，它主要是用来解决分布式应用中经常遇到的一些数据管理问题，如：统一命名服务、状态同步服务、集群管理、分布式应用配置项的管理等。

数据模型

Zookeeper 会维护一个具有层次关系的数据结构，它非常类似于一个标准的文件系统，如图 1 所示：

图 1 Zookeeper 数据结构[3]

Zookeeper 这种数据结构有如下这些特点：

每个子目录项如 NameService 都被称作为 znode，这个 znode 是被它所在的路径唯一标识，如 Server1 这个 znode 的标识为 /NameService/Server1

znode 可以有子节点目录，并且每个 znode 可以存储数据，注意 EPHEMERAL 类型的目录节点不能有子节点目录

znode 是有版本的，每个 znode 中存储的数据可以有多个版本，也就是一个访问路径中可以存储多份数据

znode 可以是临时节点，一旦创建这个 znode 的客户端与服务器失去联系，这个 znode 也将自动删除，Zookeeper 的客户端和服务器通信采用长连接方式，每个客户端和服务器通过心跳来保持连接，这个连接状态称为 session，如果 znode 是临时节点，这个 session 失效，znode 也就删除了

znode 的目录名可以自动编号，如 App1 已经存在，再创建的话，将会自动命名为 App2

znode 可以被监控，包括这个目录节点中存储的数据的修改，子节点目录的变化等，一旦变化可以通知设置监控的客户端，这个是 Zookeeper 的核心特性，Zookeeper 的很多功能都是基于这个特性实现的。

Zookeeper 作为一个分布式的服务框架，主要用来解决分布式集群中应用系统的一致性问题，它能提供基于类似于文件系统的目录节点树方式的数据存储，但是 Zookeeper 并不是用来专门存储数据的，它的作用主要是用来维护和监控你存储的数据的状态变化。通过监控这些数据状态的变化，从而可以达到基于数据的集群管理。

Zookeeper 作为 Hadoop 项目中的一个子项目，是 Hadoop 集群管理的一个必不可少的模块，它主要用来控制集群中的数据，如它管理 Hadoop 集群中的 NameNode，还有 Hbase 中 Master Election、Server 之间状态同步等。

## 服务器缓存服务 Redis

Redis 是一个开源（BSD许可）的，内存中的数据结构存储系统，它可以用作数据库、缓存和消息中间件。它支持多种类型的数据结构，如字符串（strings），散列（hashes），列表（lists），集合（sets），有序集合（sorted sets）与范围查询，bitmaps，hyperloglogs和地理空间（geospatial）索引半径查询。Redis 内置了复制（replication），LUA脚本（Lua scripting），LRU驱动事件（LRU eviction），事务（transactions）和不同级别的磁盘持久化（persistence），并通过Redis哨兵（Sentinel）和自动分区（Cluster）提供高可用性（high availability）。

你可以对这些类型执行原子操作，例如：字符串（strings）的append 命令；散列（hashes）的hincrby命令；列表（lists）的lpush命令；集合（sets）计算交集sinter命令，计算并集union命令和计算差集sdiff命令；或者在有序集合（sorted sets）里面获取成员的最高排名zrangebyscore命令。

为了实现其卓越的性能，Redis 采用运行在内存中的数据集工作方式。根据您的使用情况，您可以每隔一定时间将数据集导出到磁盘，或者追加到命令日志中。您也可以关闭持久化功能，将Redis作为一个高效的网络的缓存数据功能使用。Redis 同样支持主从复制（能自动重连和网络断开时自动重新同步），并且第一次同步是快速的非阻塞试的同步。[5]

## NoSql数据库 MongoDB

1、存储形式，相较于传统的关系型数据库，它可以使用一个表存储原来需要使用多表关联才能存储的数据库。

2、存储动态性，相较于传统的数据库当要增加一个属性值的时候要对表大动，mongodb的面向文档的形式可以使其属性值轻意的增加和删除。而原来 的关系型数据库要实现这个需要有很多的属性表来支持。

Mongodb存储特点

3、Mongodb有两种数据写入模式，

• 一种是高效的fire-and-forget模式就是只管向数据库服务器提交数据不等数据库服务器的回应。这个是数据库默认模式。

• 另外一种是安全模式，就是写入的同时还要与服务器同步，当数据的安全性要求高的时候适合用安全模式。

4、Mongodb的日志功能。

• 日志功能是帮助在系统Down机的时候恢复数据完整性做的，默认情况下Mongodb是开启日志功能的。每一个数据库操作都会先记录日志，所以当down机重启动服务器的时候数据库服务器能够通过日志文件恢复之前未完成的操作。

• 当然，开启日志必然影响性能，所以用户需要根据自己的实际应用来选择一个合适的工作模式。

• 如果你打算使用无日志模式，最好使用拷贝集群模式，这样可以降低数据丢失的风险。

Mongodb的索引

5、Mongodb支持关系型数据库的所有索引模式（升序，降序，唯一，复合，地理空间索引），同时也支持二级索引（通过B-tree实现）。每一个collection支持64个索引

6、支持ad hoc queries用户自己可以定义自己的查询。

Mongodb的集群

7、Mongodb的集群模式是主从模式的集群，其中主服务器只有一个，支持读写，而从服务器可以有多个只支持读取操作。集群的主要目的是通过数据的冗余实现failover模式，从而提高数据库服务器的可用性。当主服务器断掉后其中的一个从服务器会自动升级为主服务器，其它drive client只要通过failover就可以自动切换到新的主服务器，正常工作。而当原来的主服务器再恢复工作的时候，它将自动转为从服务器。

Mongodb的可扩展性

8、Mongodb采用了最新的低成本的横向扩展模式，相对于传统的单结点纵向扩展，可以节约成功，而且有更好的可靠性，更好的数据处理性能。

更好的数据处理性能。

## 本章小结

随着开源社区的蓬勃发展，开源的框架也越来越多。在开源的世界里有一句名言：不要重复发明轮子。我们要充分利用开源的力量来提高我们的工作效率与工作质量。本章所述的组件都是开源社区的优秀成果，本文论述的分布式框架设计，其思想与实现都是与这些开源的组件紧密结合的。

# 第三章 分布式系统介绍

分布式系统并不是什么新鲜词，在上个世纪七八十年代就已经有各种分布式系统出现。只是在互联网时代，分布式系统才大放异彩，尤其是Google更是把分布式系统运用到了极致。Google整个的软件构架都是基于各种各样的分布式系统，诸如Borg、MapReduce、BigTable等。正是这些分布式系统，使得Google可以处理高并发请求响应以及海量数据处理等。Apache旗下的Hadoop、Spark、Mesos等分布式系统，把大数据处理相关技术变得非常亲民，让更多企业客户体会到了分布式系统的便利。

## 分布式系统的定义

我们把分布式系统定义成一个其硬件或软件组件分布在连网的计算机上，组件之间通过传递消息进行通信和动作协调的系统。这个简单的定义覆盖了所有可有效部署连网计算机的系统。[6]

由一个网络连接的计算机可能在空间上的距离不等。它们可能分布在地球上不同的洲，也可能在同一栋楼或同一个房间里。我们定义的分布式系统有如下显著特征：

并发：在一个计算机网络中，执行并发程序是常见的行为。用户可以在各自的计算机上工作，在必要时共享诸如Web页面或文件之类的资源。系统处理共享资源的能力会随着网络资源（例如，计算机）的增加而提高。在本书的许多地方将描述有效部署这种额外能力的方法。

缺乏全局时钟：在程序需要协作时，它们通过交换消息来协调它们的动作。密切的协作通常取决于对程序动作发生的时间的共识。但是，事实证明，网络上的计算机与时钟同步所达到的准确性是有限的，即没有一个正确时间的全局概念。这是通信仅仅是通过网络发送消息这个事实带来的直接结果。

故障独立性：所有的计算机系统都可能出故障，一般由系统设计者负责为可能的故障设计结果。分布式系统可能以新的方式出现故障。网络故障导致网上互连的计算机的隔离，但这并不意味着它们停止运行，事实上，计算机上的程序不能够检测到网络是出现故障还是网络运行得比通常慢。类似的，计算机的故障或系统中程序的异常终止（崩溃），并不能马上使与它通信的其他组件了解。系统的每个组件会单独地出现故障，而其他组件还在运行。

我们看一下现代分布式系统的几个例子，包括Web搜索、多人在线游戏和金融交易系统，也考察今天推动分布式系统发展的关键趋势：现代网络的泛在特性，移动和无处不在计算的出现，分布式多媒体系统不断增加的重要性，以及把分布式系统看成一种实用系统的趋势。接着本章强调资源共享是构造分布式系统的主要动机。资源可以被服务器管理，由客户访问，或者它们被封装成对象，由其他客户对象访问。

构造分布式系统的挑战是处理其组件的异构性、开放性（允许增加或替换组件）、安全性、可伸缩性（用户的负载或数量增加时能正常运行的能力）、故障处理、组件的并发性、透明性和提供服务质量的问题。

## CAP定律

在理论计算机科学中，CAP定理（CAP theorem），又被称作布鲁尔定理（Brewer's theorem），它指出对于一个分布式计算系统来说，不可能同时满足以下三点：

• 一致性（Consistence) (等同于所有节点访问同一份最新的数据副本）

• 可用性（Availability）（对数据更新具备高可用性）

• 容忍网络分区（Partition tolerance）（以实际效果而言，分区相当于对通信的时限要求。系统如果不能在时限内达成数据一致性，就意味着发生了分区的情况，必须就当前操作在C和A之间做出选择[3]。）

根据定理，分布式系统只能满足三项中的两项而不可能满足全部三项[4]。理解CAP理论的最简单方式是想象两个节点分处分区两侧。允许至少一个节点更新状态会导致数据不一致，即丧失了C性质。如果为了保证数据一致性，将分区一侧的节点设置为不可用，那么又丧失了A性质。除非两个节点可以互相通信，才能既保证C又保证A，这又会导致丧失P性质。[7]

## 现代分布式系统的特点

1。对服务器硬件要求低

• 对服务器硬件可靠性不做要求，允许服务器硬件发生故障，硬件的故障由软件来容错。所以分布式系统的高可靠性是由软件来保证。

• 对服务器的性能不做要求，不要求使用高频CPU、大容量内存、高性能存储等等。因为分布式系统的性能瓶颈在于节点间通讯带来的网络开销，单台服务器硬件性能再好，也要等待网络IO

2。强调横向可扩展性

横向可扩展性（Scale Out）是指通过增加服务器数量来提升集群整体性能。纵向可扩展性（Scale Up）是指提升每台服务器性能进而提升集群整体性能。纵向可扩展性的上限非常明显，单台服务器的性能不可能无限提升，而且跟服务器性能相比，网络开销才是分布式系统最大的瓶颈。横向可扩展性的上限空间比较大，集群总能很方便地增加服务器。而且分布式系统会尽可能保证横向扩展带来集群整体性能的（准）线性提升。比如有10台服务器组成的集群，横向扩展为100台同样服务器的集群，那么整体分布式系统性能会提升为接近原来的10倍。

3。不允许单点失效（No Single Point Failure）

单点失效是指，某个应用服务只有一份实例运行在某一台服务器上，这台服务器一旦挂掉，那么这个应用服务必然也受影响而挂掉，导致整个服务不可用。例如，某网站后台如果只在某一台服务器上运行一份，那这台服务器一旦宕机，该网站服务必然受影响而不可用。再比如，如果所有数据都存在某一台服务器上，那一旦这台服务器坏了，所有数据都不可访问。

因为分布式系统的服务器都是廉价的PC服务器，硬件不能保证100%可靠，所以分布式系统默认每台服务器随时都可能发生故障挂掉。同时分布式系统必须要提供高可靠服务，不允许出现单点失效，因此分布式系统里运行的每个应用服务都有多个运行实例跑在多个节点上，每个数据点都有多个备份存在不同的节点上。这样一来，多个节点同时发生故障，导致某个应用服务的所有实例都挂掉、或某个数据点的多个备份都不可读的概率大大降低，进而有效防止单点失效。

通常情况，不要让服务器满负荷运行，服务器长时间满负荷运行的话，出故障的概率显著升高。所以分布式系统采用一大堆中低性能的PC服务器，尽可能把负载均摊到所有服务器上，让每台服务器的负载都不高，保证集群整体稳定性。

4。分布式系统尽可能减少节点间通讯开销

如前所述，分布式系统的整体性能瓶颈在于内部网络开销。目前网络传输的速度还赶不上CPU读取内存或硬盘的速度，所以减少网络通讯开销，让CPU尽可能处理内存的数据或本地硬盘的数据，能显著提高分布式系统的性能。典型的例子就是Hadoop MapReduce，把计算任务分配到要处理的数据所在的节点上运行，从而避免在网络上传输数据。

5。分布式系统应用服务最好做成无状态的

应用服务的状态是指运行时程序因为处理服务请求而存在内存的数据。分布式应用服务最好是设计成无状态。因为如果应用程序是有状态的，那么一旦服务器宕机就会使得应用服务程序受影响而挂掉，那存在内存的数据也就丢失了，这显然不是高可靠的服务。把应用服务设计成无状态的，让程序把需要保存的数据都保存在专门的存储上，这样应用服务程序可以任意重启而不丢失数据，方便分布式系统在服务器宕机后恢复应用服务。[8]

比如，在设计网站后台的时候，对于用户登陆请求，可以把登陆用户的session相关信息保存在Redis或Memcache等缓存服务中，这样每个网站的后台实例不保存用户登录状态，这样即使重启网站后台程序也不丢失用户的登录状态信息；如果把用户的session相关信息保存在网站后台程序的内存里，那一旦受理用户登录的网站后台程序实例挂掉，必然有用户的登录状态信息会丢失。

## 本章小节

总而言之，分布式系统是大数据时代企业级应用的首选平台，它有良好的可扩展性，尤其是横向可扩展性（Scale Out），使得分布式系统非常灵活，能应对千变万化的企业级需求，而且降低了企业客户对服务器硬件的要求，真正能做到应用服务层面的弹性扩展（auto-scaling）。

# 第四章 工作圈的需求分析

本章将介绍分布式服务框架实际应用的一个项目。这个项目已经是公司对外布的一个成熟的产品了，目前已有60万的用户与20多万的企业在应用这个产品。

## 系统用户角色分析

陌生人社交用微博，朋友沟通用微信，工作沟通用什么？微信，微博？目前在企业协同领域上有所作为移动APP还是很少，这就是工作圈切入的一个痛点。

工作圈重点是依托于目前的智能手机平台，为企业用户提供一站全能式办公协同功能。企业用户可以通过即时消息系统在工作圈中进行消息、语音沟通，也可以发起多人的会话。工作圈的主打的工作功能列表中为企业用户提供了外勤签到、内勤签到、审批（请示、请假、报销、借款等审批类功能）、公告、企业文件柜、电话会议等一系列企业办公的功能。工作圈中的圈子功能可以让工作上的交集的同事朋友们平等在其中分享自己的工作经验、工作成果以及想对同事、领导或是部门、公司说的话、提出的改进的意见等。当别的同事在圈子中发表了他的分享，圈中的成员可以对这个分享进行评论，点赞或是在其它的圈中进行转载。

## 系统功能需求分析

### 企业通讯录

有企业一个清晰简单的组织结构树。可以通过姓名、电话或是邮件等关键信息的搜索公司中同事的信息。也可以将同事的电子名片转发给相关的人。

可以很方便的和同事通过电话，工作邮件，短信，消息方式进行联系。也可以通过工作圈提供的即时通讯功能联系或是多个人一同发起会话。

企业管理员可以新增，删除员工，调整部门组织结构。

### 圈子

将工作上有交集的同事，朋友们组织在一起，大家可以平等的在圈子中发表在公司的一些工作成果或是发起一项讨论等。

圈子中的角色有圈子管理员与圈子成员。

可以邀请他人加入圈子。

### 贴子

贴子是发表在圈子中文章。可以被圈子中的用户浏览。

贴子可以发表500字的文章，可以上传图片，附件。可以被评论，可以被点赞。

在贴子中可以@圈子中的用户，被@到的用户会收到一条关于这篇贴子的提醒。

### 评论

可以对圈子中的贴子发表评论。评论中也可以上传图片或是附件。

评论也可以用来回复他人的评论

当用户评论贴子的时候，发贴人会收到提醒。回复他人的评论时，被回复用户也会收到提醒。

评论中可以@圈子中的用户。被@到的用户会收到一条关于这条评论的提醒。

### 赞

可以对贴子点赞。

### 工作功能

工作功能是建立在企业通讯录之上，为企业员工用户提供协同办公功能的服务。

公告：发布公告，提示员工有新的未读公告。公告发布范围可以限定是部门内或是全公司。

审批：员工可以通过这个功能提交日常的请示、请假、报销等审批类型的单子给上级领导。收到审批单的领导会收到一条推送消息提示有待您审批的单据。员工的请假单时间可以转换成为考勤日历给企业管理员查看导出

签到：读取员工的地理位置信息，生成一条签到记录提交给相关的领导知会。

文件柜：接入的第三方开发商(ISV)为工作圈企业用户提供的企业网盘功能。

电话会议：第三方电信运营商为工作圈企业用户提供的付费语音电话会议服务。

### 即时通信（IM）

工作圈企业用户可以通过即时通信服务进行沟通。可以向用户发送文字、语音、表情、图片及文件等。

工作圈的@人与评论帖子的通知也是通过即时通讯完成的。

即时通信服务还可以为工作功能提供相关的消息推送服务。比如发布公告，提交审批等等。

## 系统的非功能需求分析

### 数据存储持久性

定义：承诺在合同期内数据保存不丢的概率，即每月完好数据/\_(\_每月完好数据+\_每月丢失数据)\_。

工作圈的数据持久性不低于99.99999999%。数据持久性按服务周期统计，一个服务周期为一个自然月，如不满一个月不计算为一个服务周期。

按工作圈的数据持久性计算，如用户在工作圈存储一百亿条工作或是圈子相关的业务数据，每月最多只有1条业务数据发生数据丢失的可能性。

### 数据可迁移性

承诺用户能够控制数据或主机镜像的迁移，保证启用或弃用该云服务时，数据能迁入和迁出。

用户可以通过API，SDK等工具对存储在工作圈上的数据进行读写操作，并根据需要进行迁移。

工作圈不会对用户上传的数据做任何的修改。

### 数据私密性

承诺用户应有加密或隔离等手段保证同一资源池用户数据互不可见，并且在用户授权的情况下，云服务商才能获得数据。

用户存储在工作圈上的数据，在未经用户合法授权的情况下，其他用户无法访问其数据。

工作圈为从访问接口上进行权限控制和隔离，保障用户数据的私密性。

### 服务可用性

承诺用户业务可用性为合同期内每月单个用户云服务业务可用时间的概率，即每月实际可用时间/每月(实际可用时间+不可用时间)。

工作圈的服务可用性不低于99.99%。可用性按服务周期统计，一个服务周期为一个自然月，如不满一个月不计算为一个服务周期。工作圈所提供的服务在连续的5分钟或更长时间不可使用方计为不可用时间，不可使用的服务时间低于5分钟的，不计入不可用时间。工作圈不可用时间不包括日常系统维护时间、由用户原因、第三方原因或不可抗力导致的不可用时间。

工作圈服务可用性的计算方法如下：

月总请求次数低于1万的用户不做统计；

失败请求数 = 返回错误5xx的请求数量；

估算的失败请求数 = 前7天用户单位时间请求数的平均值 × 服务不可用时间 （当出现服务不可用且无失败请求返回时）；

工作圈的可用性 =（失败请求数+估算的失败请求数）/总请求数（正常请求数+失败请求数+估算的失败请求数）。

### 故障恢复能力

告知用户如出现故障时，故障恢复的能力。

工作圈为付费用户的云服务提供7×24小时的运行维护，并以在线工单和电话报障等方式提供技术支持，具备完善的故障监控、自动告警、快速定位、快速恢复等一系列故障应急响应机制。

## 本章小结

本章介绍了工作圈这个软件的用户需求以及非功能性的需求。下面我们就会结合之前的开源技术与分布式的思想来进行这个项目的设计，以期能实现本章中的各个需求。

# 第五章 工作圈分布式系统的设计

## 服务模块化的设计

对服务进行模块划分，有助于对业务进行更清晰的梳理。同时在软件设计的层面上，好的模块划分更有利于程序员架构师对项目的维护与扩展，减少开始的代码量。

模块划分是分布式服务设计的第一步。有了上面的业务需求，对工作圈的模块第一步的划分很好理解，根据上面的需求功划分能得到：用户、圈子（贴子）、评论、赞、IM消息、工作功能等模块服务。因为贴子也是圈子的一个成员属性故划到圈子服务中。而评论和赞的服务，可以通用化用在其它的一些业务上，比如用在工作的审批中。所以做为一个基础的服务可以独立出来。

所有的业务模块，基本都需要操作数据库与缓存。所以将对MongoDB数据库与Redis缓存的连接与操作服务封装成公共模块的服务，开发时要求开发人员统计使用这些公共服务完成对持久层的开发。这样不仅可以减少在持久层上的的代码开发量，同时也可以应用链接池技术提升服务的吞吐性能，预防因为人为的疏忽导致的链接泄漏。

在进行模块划分设计需要考虑的要点：

1.高内聚，低耦合。模块之前要相互独立，仅数据耦合。

2.提取功能相同的代码到一个公共服务包进封装。减少开发代码量。

3.模块服务要是无状态的，便于模块服务的模向扩容，同时也为restful服务打基础

最终模块划分如下：

公共模块：

gongzuoquan-mongo，提供数据库接入操作的封装。

gongzuoquan-cache，提供对缓存服务的接入操作。

gongzuoquan-configcenter，提供自动化配置中的管理操作。

gongzuoquan-util，提供一些公共的工具类方法

核心业务服务

gongzuoquan-idcenter，提供全局主键生成服务。

gongzuoquan-idlist，提供数据主键索引中心。

gongzuoquan-setting，提供用户个性化设置配置服务。

业务服务

gongzuoquan-account，用户账户服务

gongzuoquan-quan，圈子、贴子服务

gongzuoquan-comment，评论服务

gongzuoquan-favorite，赞服务

gongzuoquan-app，工作应用服务

## 分布式服务与自动化配置中心

模块划分完成之后，这个框架还不能被称为分布式框架。因为它还没有满足前面（3.3 现代分布式系统的特点）中提到的特性。下面来尝试进一步的满足这些特性。

横向可扩展性（Scale Out）。意味着一个业务服务模块是可以部署很多个副本，这样就带来一个问题，如何保证所有的请求都能平均的分发到所有的服务上去？这时候我们需要一个自动化配置中心。在一个模块服务启动时，将启动的服务器在网络上的IP地址和提供服务的端口注册到自动化配置中心服务中，这样就完成了服务状态的发布。然后任一客户端在需要调用这个服务的时候，先在自动化配置中心中读取一下这个服务的服务状态，取得了可以提供这个服务的服务器地址列表，然后再以一个算法随机的地址发起请求，完成一次调用。这样就满足了服务的横向可扩展性（Scale Out）。

不允许单点失效（No Single Point Failure）。只要对自动化配置中心的功能稍加改动，便能满足这一特性。只要能将服务状态列表中的故障节点检测到，并从中剔出除们，客户端便不会受到影响。只要这个服务状态列表中仍有一台服务器上的服务是正常的，就会不造成整个服务失效。当然，这里我们也要考虑到这个自动化配置中心服务本身会出问题，需要保证这组服务也是高可用的服务。

对故障列表的检测，有两种方式：1.在启动的服务与自动化配置中心之间建立一个心跳，心跳消失即失效；2.从客户端来检测，如果客户端调用服务失败后，将其从自动化配置中心中剔除然后访问下一个服务地址。第一种方式，因为心跳会有延时。如果在延时中发生问题，那么在这一段时间内就会有一定机率的请求会因为请求到故障节点上。第二种方式，需要做到客户端实时监听自动化配置中心的服务器状态变化，这样虽然会避免请求故障节点，但是在客户端数量过大的时候会大量的占用自动化配置中心服务的链接。工作圈目前的设计，客户端服务还是可控的，所以选择了第二种实时性较高的处理方式。

在实际的开发中，Apache Zookeeper服务成为了自动化配置中心服务的不二之选。工作圈在Zookeeper约定以/gongzuoquan为根目录，在下面以模块名建立服务配置与服务状态的目录。客户端只要建立相应节点上的监听就可以得到相应服务的当前服务状态。同样，也可以将数据库和缓存服务的地址与链接配置存到Zookeeper中相应的节点下。当服务器的数据库或是缓存进行迁移数所据的时候，监听节点的变化，重新建立指向新服务的链接池就可以了。

## 数据分片(Data Sharding)与数据索引中心

大数据时代，数据的持久化性能会影响服务的整体的性能。如果一个模块下的业务都存储在一个表中，那么随着数据量的上升性能必然会下降。解决的办法是对数据库中的数据进行水平扩展(分表，分库等)，控制住单表中的数据量不会超过一个阀值即可。这个阀值的界定不是固定的，会根据数据库的选择以及运行的环境不同而不同。比如我们需要数据库插入一条业务数据的响应时间要在1ms以内，需要我们在实际的运行环境中对数据库进行一个insert压力测试，得到在多少条数据的情况下单表的插入响应时间超过了1ms（这个时间需要根据业务场景来设定），那么这个阀值就是我们估算的一个单表数据量上限的指标。假设这个值是f，需求预估的整个系统容量是m，那个分表数t = m / f。如果数据库服务器不只一台，那么我们可以再将数据散列到不同的数据库上去，此时假设有n台数据库服务，那么t = m / (f \* n)。

对数据进行分表，分库的插入的过程我们叫数据分片((Data Sharding)。当系统分片的策略有很多，例如常见的有以下几种：

根据ID特征：例如对记录的ID取模，得到的结果是几，那么这条记录就放在编号为几的数据分区上。

根据时间范围：例如前100万个用户数据在第1个分区中，第二个100万用户数据放在第2个分区中。

基于检索表：根据ID先去一个表内找到它所在的分区，然后再去目标分区进行查找。

在这些数据分片策略之中没有哪个有绝对的优势，选择哪种策略完全是根据系统的业务或是数据特征来确定的。值得强调的是：数据分片不是银弹，它对系统的性能和伸缩性（Scalability）带来一定好处的同时，也会对系统开发带来许多复杂度。例如，有两条记录分别处在不同的服务器上，那么如果有一个业务是为它们建立一个“关联”，那么很可能表示“关联”的记录就必须在两个分区内各放一条。另外，如果您重视数据的完整性，那么跨数据分区的事务又立即变成了性能杀手。最后，如果有一些需要进行全局查找的业务，光有数据分片策略也很难对系统性能带来什么优势。

工作圈的数据分片主要使用的是ID取模的方式，但是对这个策略进行了改进。首先引入一个gongzuoquan-idcenter的服务，用来随机生成数据在工作圈服务内全局唯一的主键id。这样，id在进行取模分片的时候就可以较为均匀的分布在所有的分表中。

同时，为了解决跨区数据查询的问题，我们需要一个数据索引中心（类拟于hadoop的nameService的概念)。相一组有关系的数据id索引在一起保存起来，在查询的时候通过这个索引可以得到这些数据的id，然后再通过id取模来得到分库分表的数据，从而查询到需要的业务数据。工作圈中承担这部份工作的核心服务是gongzuoquan-idlist。

## Rest server设计

Rest server是整个工作圈服务的表现层，是和工作圈移动端或是WEB端通信的一层。工作圈终端借由Rest Server这一层提供的Rest api，来和工作圈服务进行数据的交换。

分布式的Rest server。

1.Nginx负载均衡

分布式的Rest server意味着一个Rest服务器可以运行在多个服务器上。那么就需要一个前端的负载均衡来分发从互联网传来的请求。可用的服务也很多，Nginx是一个开源的优秀的http服务器，可以满足目前的需求。

2.无状态的服务设计

其实提出这点反而是为了解决用户的登录状态问题，假设用户登录的请求发到了Rest A服务上，对用户的数据请求又发送到了Rest B服务上。那Rest B怎么能知道用户的登录状态呢。在以前的单点式系统中，有一个Session的概念。用户登录后建立在服务器，请求的时候服务端会查询这个Session，登出后销毁。但是在分布式的Rest server上主机之前内存的数据不共享，无法得知用户的Session状态。

工作圈的解决方式是登录后返回一个加密的token给终端。不是在服务端保存用户的登录状态，而是在用户使用的终端上保存这个token。在Android iOS系统上是由工作圈App软件保存在手机的数据库中，PC WEB因为没有数据库，将这个token保存在Cookie中。在每次请求的时候将WEB端会自动的带上Cookie供服务器查询，手机端的App需要程序从手机上读取这个请求加到Http Header中来便于服务检查。

## 最终架构图

结合上面的设计，我们最终得到了如下图示的一个完整的服务系统框架



# 第六章 系统概要设计

## 自动化配置管理中心模块（configcenter）

### 自动化配置管理中心模块概述

自动化配置管理中心封装了对zookeeper服务的操作。在这个模块中定义了软件在不同的阶段需要使用到的环境：开发环境、测试环境、模拟环境、生产环境，使每个环境的配置相互隔离。同时也为服务提供统一的配置监听入口，在配置变更的时候执行服务指定的处理逻辑。

工作圈中的所有的服务都依赖于这个模块。

### 自动化配置管理中心模块类图

图3-1 configcenter模块类图

### Env类概述

Env是一个枚举类，枚举了工作圈的环境信息：开发环境、测试环境、模拟环境、生产环境。

类功能定义

|  |  |  |
| --- | --- | --- |
| **序号** | **功能点** | **功能点详细内容** |
| 1 | getSystemConfigType | 获取当前系统运行的Env变量 |
| 2 | checkEnvIsProd | 当前运行是否是生产环境 |
| 3 | isInte | 当前运行是否是测试环境 |
| 4 | isDev | 当前运行是否是开发环境 |
| 5 | isAli | 当前运行是否是阿里云环境 |

### ZkHelper类概述

通过启动参数识别服务运行环境，连接指定的zookeeper服务。协助服务进行zookeeper对指定的配置节点变化进行监听.

ZkHelper类功能定义

|  |  |  |
| --- | --- | --- |
| **序号** | **功能点** | **功能点详细内容** |
| 1 | getInstance | 获取一个ZkHelper的实例 |
| 2 | init | 通过启动参数初始化zookeeper连接 |
| 3 | checkEnv | 重新初始化 |
| 4 | closeZk | 关闭zookeeper连接 |
| 5 | delete | 删除节点 |
| 6 | addNode | 新增节点 |
| 7 | subscribeDataChanges | 节点数据变化监听 |
| 8 | subscribeChildChanges | 节点子节点变化监听 |
| 9 | getChildren | 获取path节点下的儿子节点列表 |
| 10 | getValue | 获取节点数据 |
| 11 | createEphemeral | 创建临时节点 |
| 12 | setPathData | 节点设置值 |

## 数据库操作模块（mongo）

### 数据库操作模块概述

数据库操作模块为工作圈提供了一个对mongoDB数据库操作封装的工具。这样做的好处有以下几点:

* + - 1. 开发者关注更少的代码，不需要关注数据库的操作具体的实现。
      2. 对数据库的资源进行统一管理，避免资源泄漏。
      3. 方便维护。当数据库升级之后，只需要将这模块的实现升级，然后依赖此模块的服务更新版本即可。

工作圈中的所有需要持久化数据的服务都依赖于这个模块。

### 数据库操作模块类图



图3-1 mongo模块类图

### DbPool类概述

MongoDB连接池的封装。通过zkHelper从注册中心监听某服务的MongoDB集测配置，初始化MongoDB集群的连接池。通过工作圈服务应用的数据分片key来计算数据映射的节点，并从池中取出此节点的连接供应用来使用。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | getInstance | 获取DbPool的实例 |
| 2 | initialPoolByURI | 初始化连接池，设置参数。读zookeeper |
| 3 | initialPool | 初始化连接池，设置参数。读zookeeper |
| 4 | getMongoDataBase | 获取DB实例 |
| 5 | getCollection | 获取MongoDB的集合 |
| 6 | releaseRs | 释放资源 |

### TBaseDao类概述

MongoDB某集合数据的操作类，封装了数据的增，删，改，查操作。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | getInstance | 获取一个TBaseDao的实例 |
| 2 | findByParams | 简单的条件查找 |
| 3 | findByParamsDBObject | 复杂的条件查找 |
| 4 | findByParamsDBObject2 | 复杂的条件查找（返回MongoDB原生对象） |
| 5 | findByParamsDBObjectWithOrderLimit | 复杂条件查找,增加limit和sort |
| 6 | findByParamsDBObjectWithOrderLimit | 复杂条件查找,支持limit skip和sort |
| 7 | findOne | 条件查找唯一记录 |
| 8 | findOne2 | 条件查找唯一记录 |
| 9 | findOneById | 根据主键查找 |
| 10 | addorupdate | 添加或者修改记录 |
| 11 | dropCollection | Drop一个集合 |
| 12 | addBatch | 批量插入（这个慎用，因为一次插入是一个库和表，所以不能走分表逻辑） |
| 13 | update | 修改对象 |
| 14 | updateOrAdd | 修改对象，当对象不存在的时候创建 |
| 15 | remove | 删除对象 |
| 16 | getCount | 获取记录数 |
| 17 | createIndex | 创建索引 |
| 18 | db2Bean | MongoDB对象转换成普通java bean对象 |
| 19 | bean2BasicDb | 对象数据转为MongoDB对象 |

## 缓存操作模块（cache）

### 缓存操作模块概述

缓存操作模块为工作圈提供了一个对Redis缓存操作封装的工具。这样做的好处有以下几点：

* + - 1. 开发者关注更少的代码，不需要关注缓存的操作具体的实现。
      2. 对缓存的资源进行统一管理，避免资源泄漏。
      3. 方便维护。当缓存升级之后，只需要将这模块的实现升级，然后依赖此模块的服务更新版本即可。

工作圈中的需要对数据存取加速的服务都依赖于这个模块。

### 缓存操作模块类图



图3-1 cache模块类图

### NutcrackerCenter类概述

Redis基于Nutcracker组件连接池的封装。通过zkHelper从注册中心监听某服务的Nutcracker集测配置，初始化Nutcracker集群的连接池。通过工作圈服务应用的数据分片key来计算数据映射的节点，并从池中取出此节点的连接供应用来使用。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | getInstance | 实例化服务池 |
| 2 | getRedisList | 获取redis实例地址列表 |
| 3 | initialPool | 初始化redis 池 |
| 4 | getJedis | 从公用redis集群获取resource |
| 5 | returnJedis | 归还redis链接 |
| 6 | returnBrokenJedis | 归还坏redis链接 |
| 7 | getResult | 处理任务返回结果 |

### NutcrackerCenterAop类概述

NutcrackerCenterAop是NutCrackerCenter的Spring 切面编程（AOP）的封装。它首先初始化一个NutcrackerCenter。通过Spring切面式编程，配置到业务服务的缓存层中，以AOP代理的方式执行业务缓存方法的调用。调用时，从池中获取一个连接绑定到处理线程上，在处理完之后由代理处理连接资源的归还与释放，然后再返回处理结果。从而达到防止资源泄漏。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | init | Spring AOP代理类初始化 |
| 2 | invoke | Spring AOP代理方法调用 |
| 3 | getJedis | 获取线程绑定上的Jedis连接 |

### ModRedisCenterAop类概述

Redis基于Redis自己的分布式组件ShardingRedis连接池的封装。通过zkHelper从注册中心监听某服务的Redis集测配置，初始化Redis集群的连接池。通过工作圈服务应用的数据分片key来计算数据映射的节点，并从池中取出此节点的连接供应用来使用。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | getInstance | 实例化服务池 |
| 2 | getRedisList | 获取redis实例地址列表 |
| 3 | initialPool | 初始化redis 池 |
| 4 | getJedis | 从公用redis集群获取resource |
| 5 | returnJedis | 归还redis链接 |
| 6 | returnBrokenJedis | 归还坏redis链接 |
| 7 | getResult | 处理任务返回结果 |

### ModRedisCenterAop类概述

ModRedisCenterAop是ModRedisCenter的Spring 切面编程（AOP）的封装。它首先初始化一个ModRedisCenter。通过Spring切面式编程，配置到业务服务的缓存层中，以AOP代理的方式执行业务缓存方法的调用。调用时，从池中获取一个连接绑定到处理线程上，在处理完之后由代理处理连接资源的归还与释放，然后再返回处理结果。从而达到防止资源泄漏。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | init | Spring AOP代理类初始化 |
| 2 | invoke | Spring AOP代理方法调用 |
| 3 | getJedis | 获取线程绑定上的Jedis连接 |

### LockCacheCenter类概述

Redis基于Redis自己的分布式组件ShardingRedis连接池的封装。通过zkHelper从注册中心监听某服务的Redis集测配置，初始化Redis集群的连接池。通过工作圈服务应用的数据分片key来计算数据映射的节点，并从池中取出此节点的连接供应用来使用。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | getInstance | 实例化服务池 |
| 2 | getRedisList | 获取redis实例地址列表 |
| 3 | initialPool | 初始化redis 池 |
| 4 | getJedis | 从公用redis集群获取resource |
| 5 | returnJedis | 归还redis链接 |
| 6 | returnBrokenJedis | 归还坏redis链接 |
| 7 | getResult | 处理任务返回结果 |

### LockCache类概述

LockCache是在LockCacheCenter上实现的一分布式缓存锁的实现。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | getLock | 获取资源 |
| 2 | releaseLock | 释放锁资源 |

## Id列表基础服务模块（IdList）

### Id列表基础服务模块概述

工作圈的Id列表基础服务为工作圈提供了一个集中的ID维护服务。

工作圈的数据使用了松散的分布式存储，这样可以在大并发下获得很高的吞吐量与较低的延迟。但是这样的数据不变于业务处理维护。所以需要这样的一个基础的id列表维护服务来组织数据之间的关系。

工作圈中的圈子模块，贴子模块，轻应用模块以及其它的一些基础模块都依赖于这个服务。

### IdList系统模块类图



图3-1 IdList模块类图

### IdListClient 类概述

封装对Thrift接口的调用，基于zookeeper进行 多点分发和监控。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | getResourceIdByScore | 获取一定时间段的增量数据 从小到大 |
| 2 | getResourceIdByScoreDesc | 获取一定时间段的增量数据 从大到小 |
| 3 | getResourceIdWithScoreByScoreDesc | 获取一定时间段的增量数据 with score 从大到小 |
| 4 | addOrUpdateResourceId | idlist添加记录 |
| 5 | existMember | idlist 判断某记录是否存在 |
| 6 | isExistKeys | idlist 判断某KEY是否存在 |
| 7 | addOrUpdateResourceIds | idlist添加多记录 |
| 8 | rmResourceId | idlist 移除记录 |
| 9 | rmResourceIds | idlist 移除多条记录 |
| 10 | getKeySize | 根据key获取size |
| 11 | getResourceIdByKey | 根据key 分页取idlist |
| 12 | clear | 清除掉一个key |
| 13 | getIdScores | 通过resourceid 获取对应的score值 |
| 14 | batchGetResourceIdsByScoreDesc | 批量获取idlist的方法 |
| 15 | batchAddOrUpdateResourceIds | 批量创建idlist的方法 |
| 16 | batchRmResourcesIds | 批量在一批idlist删除一个id的方法 (叉积的方式处理的) |
| 17 | batchRmResourcesIdsOne2One | 批量在一批idlist删除一个id的方法 (一一对应的处理) |

### IdListServiceImpl 类概述

服务端的接口实现类，实现Thrift的Iface接口，Client通过Thrift远程调用。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | getResourceIdByScore | 获取一定时间段的增量数据 从小到大 |
| 2 | getResourceIdByScoreDesc | 获取一定时间段的增量数据 从大到小 |
| 3 | getResourceIdWithScoreByScoreDesc | 获取一定时间段的增量数据 with score 从大到小 |
| 4 | addOrUpdateResourceId | idlist添加记录 |
| 5 | existMember | idlist 判断某记录是否存在 |
| 6 | isExistKeys | idlist 判断某KEY是否存在 |
| 7 | addOrUpdateResourceIds | idlist添加多记录 |
| 8 | rmResourceId | idlist 移除记录 |
| 9 | rmResourceIds | idlist 移除多条记录 |
| 10 | getKeySize | 根据key获取size |
| 11 | getResourceIdByKey | 根据key 分页取idlist |
| 12 | clear | 清除掉一个key |
| 13 | getIdScores | 通过resourceid 获取对应的score值 |
| 14 | batchGetResourceIdsByScoreDesc | 批量获取idlist的方法 |
| 15 | batchAddOrUpdateResourceIds | 批量创建idlist的方法 |
| 16 | batchRmResourcesIds | 批量在一批idlist删除一个id的方法 (叉积的方式处理的) |
| 17 | batchRmResourcesIdsOne2One | 批量在一批idlist删除一个id的方法 (一一对应的处理) |
| 18 | rebuildidlistCache | 重新构造idlist cache |
| 19 | batchRebuildListCache | 批量重建缓存 |

### CommonCache类概述

IdList数据库中的数据到Redis缓存的返序列化数据集合业务。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | getIdListPoByKey | 获取idlist key的value |
| 2 | getIdListPoByKey | 批量获取idlist key的value |
| 3 | setIdListPoByKey | 获取一定时间段的增量数据 with score 从大到小 |
| 4 | batchSetIdListPoMap | batch set idlist key的value |
| 5 | Del | rm key |

### IdListCache类概述

IdList依赖使用Redis缓存服务进行Id的排序和计算业务。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | addId | idlist添加add 添加成功同时会标记操作结果为成功 |
| 2 | addIds | idlist添加多个id 一次最多200个记录 |
| 3 | delId | 删除idlist 一条id |
| 4 | delkey | 删除key |
| 5 | delIds | 删除idlist 多个id |
| 6 | batchDelIds | 删除idlist 多个id |
| 7 | batchDelIds | 批量删除idlist 多个id |
| 8 | getIdList | 获取一个key对应的idlist -1全部 |
| 9 | getIdListByScore | 获取一个key在score时间之间的idlist 从小到大 |
| 10 | getIdListByScoreDesc | 获取一个key在score时间之间的idlist 从大到小 |
| 11 | batchGetIdListByScoreDesc | 批量获取一个key在score时间之间的idlist 从大到小 |
| 12 | getIdListWithScoreByScoreDesc | 获取一个key在score时间之间的idlist with score 从大到小 |
| 13 | getIdListCount | 获取某个key的总size |
| 14 | getFlagByKey | 获取key的健康状态 |
| 15 | setFlagByKey | set key的健康状态 |
| 16 | existMember | idlist key是否存在resourceId |
| 17 | getBetchListByIdList | 获取一批ID的IDLIST |
| 18 | existKey | idlist 是否存在这个key |

### IdListDao类概述

封装工作圈IdList的数据持久化业务

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | Addorupdate | 添加记录 |
| 2 | batchAddOrUpdate | 批量添加 |
| 3 | update | 通过时间截来分页获取资源下的评论 |
| 4 | Find | 查找 |
| 5 | batchFind | 批量查询 |
| 6 | rmComments | 获取所有被删除的评论 |

## 账户系统（Acount）模块

### 账户系统模块概述

工作圈的用户体系是畅捷通CIA云平台用户管理的一个子集，用户数据与CIA通过消息总线保持同步；CIA云平台下所有子系统注册的账号均可在工作圈直接登录。

工作圈账户系统，支撑工作圈各个端的单点登录、权限校验、注册、用户/企业信息查询和变更，并且基于畅捷通云平台的智能平台，为工作圈用户提供智能推荐、搜索服务。

同时account模块负责对注册转化率统计进行埋点，供工作圈内部分析统计，用于了解用户在注册环节中的转化率。

移动端注册转化率：

1、埋点统计数：

 A、安装 APP 的用户数

 B、安装 APP 成功的用户数

 C、启动 APP 的用户数

 D、进入注册页面（填手机号）的用户数

 E、进入获取验证码页面的用户数

 F、注册成功后首次进入工作圈首页的用户数

2、按漏斗或数据表格形式依次统计注册环节的转化率：

 A->B->C->D->E-> F

Web 端邀请进圈子和邀请加入企业的注册转化率：

1、埋点统计数：

 A、点击邀请链接进入 HTML 着陆页的用户数

 B、注册/加入成功的用户数

 C、点击使用已有帐号加入链接的用户数

 D、通过已有帐号加入成功的用户数

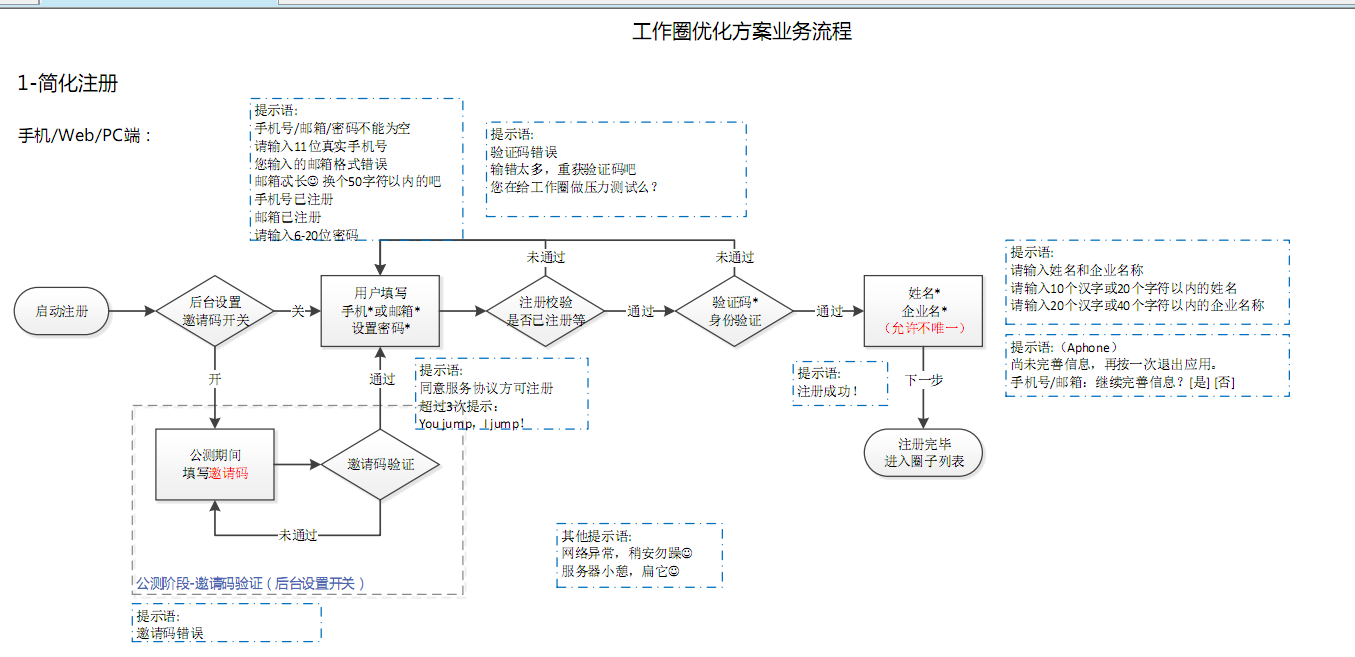
 E、通过已有帐号加入提示“已是本圈成员无需重复加入”的用户数

2、按漏斗或数据表格形式依次统计邀请注册环节的转化率：

 A->B

 A->C->D

 A->C->E

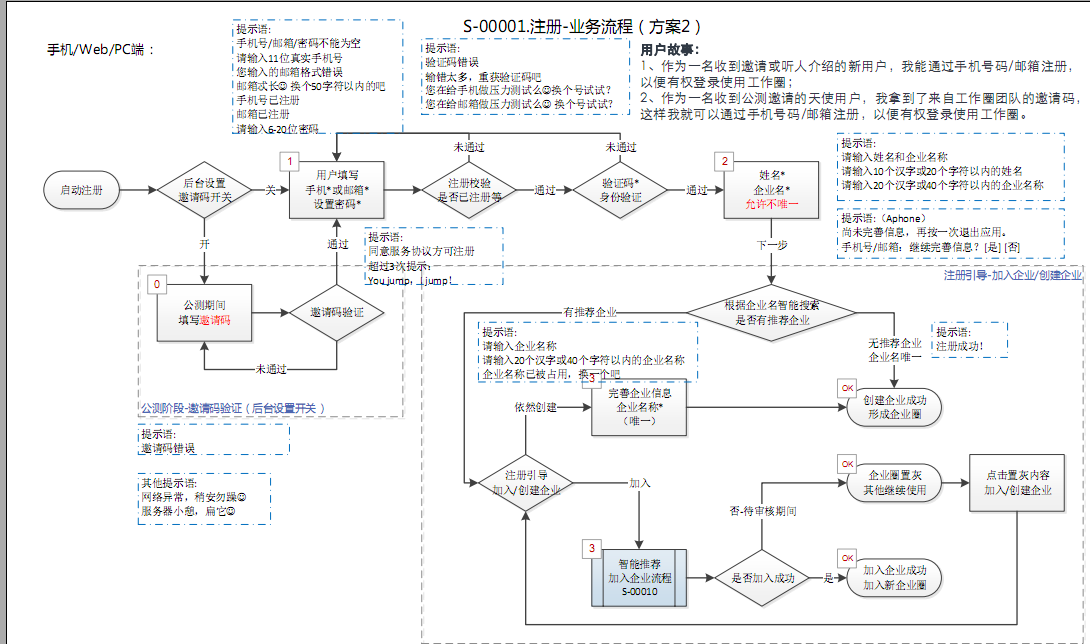
注册流程：

说明：

* 1. 相对228版简化注册流程；
  2. 取消注册环节的主动申请加入企业；
  3. 注册时填写的企业名仅仅用于个人名片中的工作单位，与用户真正的隶属企业无关；

用户需求：

1、作为一个初次接触工作圈的用户，无法理解在注册环节就必须选择加入企业或创建企业的门槛；

2、作为用户仅仅是想先体验一下工作圈的功能，即使在注册环节强制要求建立企业归属，用户填写的也未必是真实企业；

说明：

1、密码至少6位，字母数字及符号组成；

2、验证码标准4位数字；

3、姓名和企业名必填；

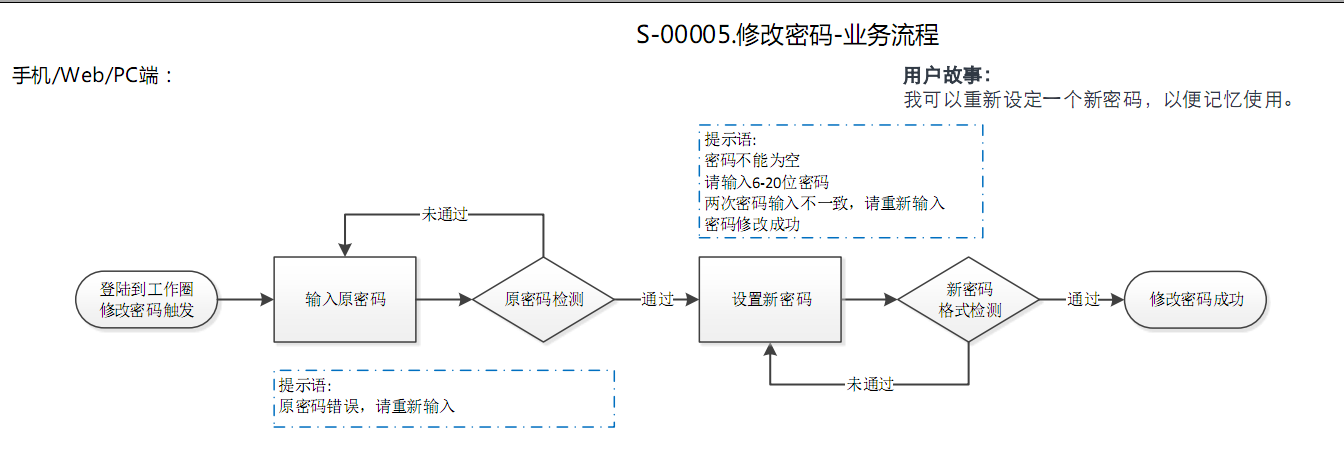
4、企业名不做唯一校验，但是在智能推荐加入企业页面选择 创建企业 按钮后，企业名需要唯一性校验；

5、根据企业名及手机号/邮箱，智能推荐备选加入企业；

6、有企业邀请的在前，关键字智能推荐的企业在后；

7、以上为业务流程，重点阐述业务关键内容，最后的程序实现要参考UE流程；

8、注意手机端和Web端UE流程的不同；

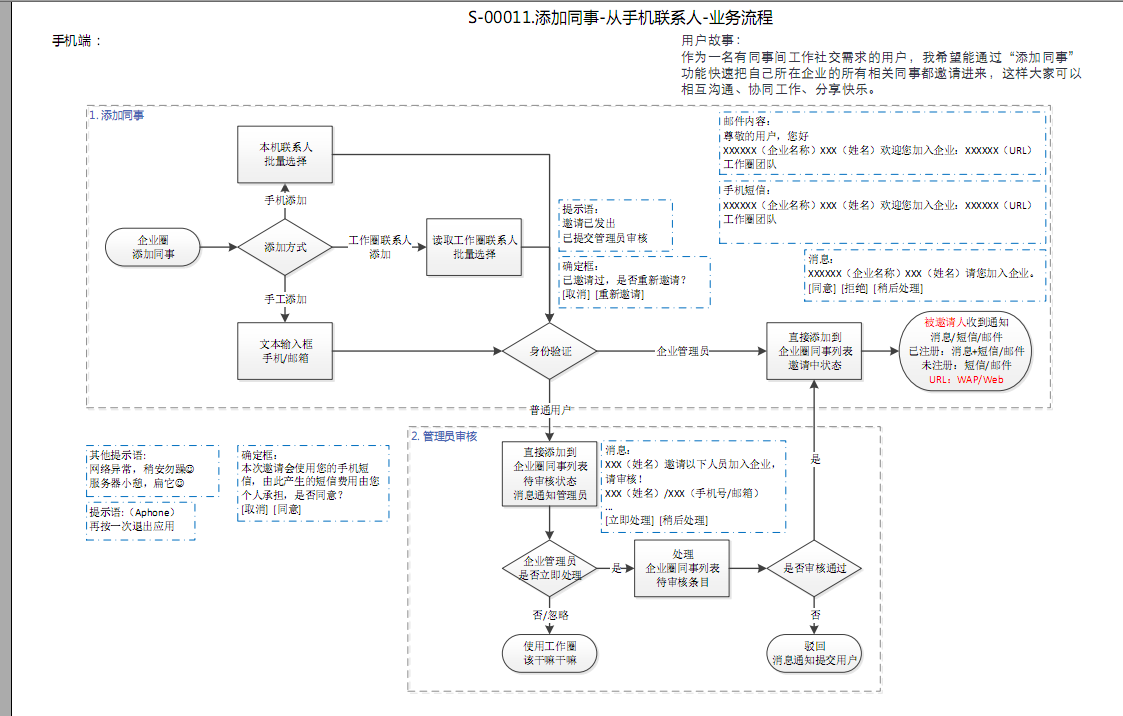
修改密码流程：

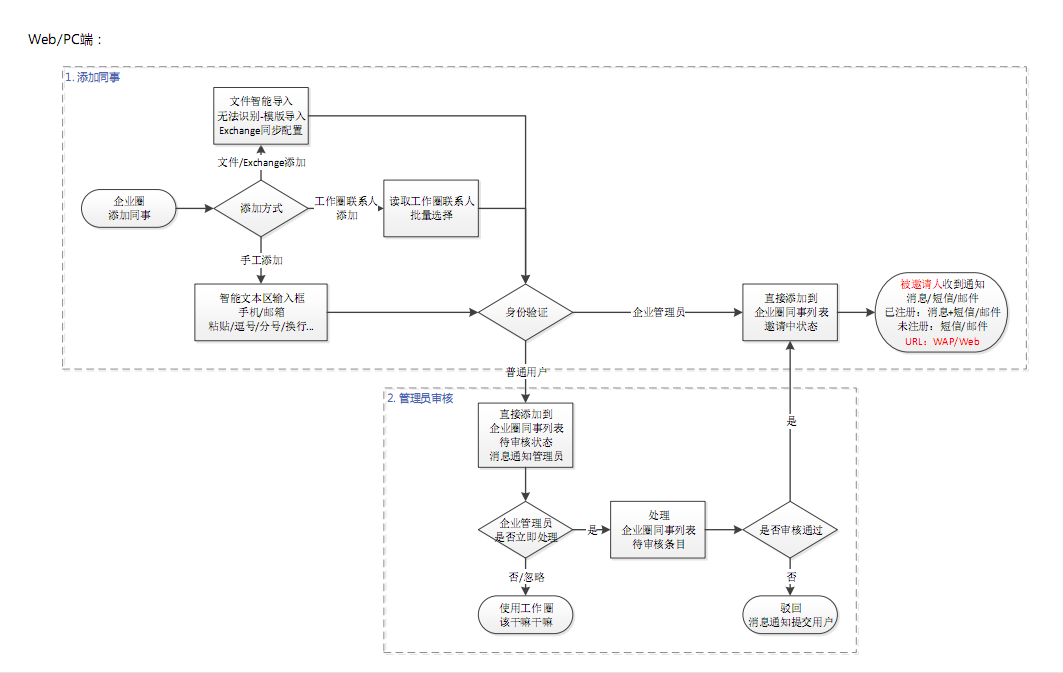
说明：

1、用户需要成功登录工作圈内才可以进行修改密码操作；

2、在设置新密码前需要输入原密码，密码验证通过后方可设置新密码；

3、新密码格式要求同注册；

添加同事：



说明：

企业圈同事列表等同于企业人员列表，包含以下几种状态：

1、活跃：已注册并成功加入企业的用户；

2、邀请中：被添加的同事还未确认加入该企业时的状态；

3、待审核：企业内普通用户发起的添加同事状态+企业外某注册用户主动申请加入该企业时的状态；

4、停用：企业管理员停用某企业内用户+某企业内用户成功加入别的企业时的状态；

以上四种状态管理员可进行的操作（普通用户无权）：

1、活跃：[停用]操作，触发后变为停用状态；

2、邀请中：[删除]操作，发出的邀请失效，相应记录直接删掉；

3、待审核：[同意]/[拒绝]操作，同意后变为邀请中状态或活跃状态；拒绝直接删掉本记录；

4、停用：有需要工作交接的停用用户，会有[工作交接]操作，如果该同事又重新加入本企业，状态变为活跃，只要其账号（手机/邮箱）不变；

以上同事列表要求显示内容：

1、企业内普通用户只需看到 活跃和邀请中 状态的同事；

2、企业管理员能够看到四种状态的同事；

3、 活跃、邀请中、停用 状态的同事，需要显示头像、姓名、手机、邮箱； 待审核 状态的同事还要显示 发起人 字段（提交添加同事的企业内普通用户或企业

外主动申请加入本企业的用户，显示为姓名）；

215版可以不显示停用状态的企业内用户，但是必须提前设计，为以后客户等业务APP做准备；

215版对于有需要工作交接的停用状态用户必须显示；

被邀请人收到通知:

1、如果是已注册用户，直接消息通知+短信/邮件；

2、如果是未注册用户，手机短信或邮件通知：通知模版：XXX（企业名）欢迎您加入工作圈，请访问下面链接完成加入：XXX（URL短链）

3、被邀请人点击URL后，后台会判断用户使用终端，如果来自移动端会自动接入WAP页，如果是PC端，会自动接入Web页，两种页面内容一致，主要包括如下字段：

XXX（企业名）邀请您加入

a。手机号/邮箱（灰色不可修改）；

b。姓名\*；

b。设置密码\*/登录密码\*

### 账户系统模块类图



图3-1 Account模块类图

### JsonController类概述

Controller的基类，封装统一较常用的方法，如参数校验、安全校验等。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | checkSig | 签名校验 |
| 2 | checkImageNo | 验证码校验 |
| 3 | changeTokenRel | Token切换 |

### AccountWebController类概述

这个类是给PC浏览器访问提供的http接口，基于SpringMVC框架 。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | userLogin | 登录接口 |
| 2 | userLogout | 退出接口 |
| 3 | getEmailVerifyCode | 发送邮箱验证码 |
| 4 | getMSISDNVerifyCode | 发送手机短信验证码 |
| 5 | userRegisterMobile | 手机用户注册 |
| 6 | userRegisterEmail | 邮箱用户注册 |
| 7 | resetPasswd | 找回密码接口 |
| 8 | modifyPassword | 修改密码接口 |
| 9 | getAtUser | 常用联系人搜索接口 |
| 10 | searchContacts | 搜索联系人 |
| 11 | getKaptchaImage | 图片验证码接口 |
| 12 | getUserInfo | 根据userId查询用户信息 |
| 13 | updateUserInfo | 修改用户信息 |
| 14 | face2faceJoinEnterprise | 面对面加入企业 |
| 15 | joinByWechatInviteCode | 通过微信邀请码加入企业 |
| 16 | joinByInviteCode | 通过qq邀请码加入企业 |

### AccountRestController类概述

给移动端提访问提供的http接口 。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | userLogin | 登录接口 |
| 2 | userLogout | 退出接口 |
| 3 | getEmailVerifyCode | 发送邮箱验证码 |
| 4 | getMSISDNVerifyCode | 发送手机短信验证码 |
| 5 | userRegisterMobile | 手机用户注册 |
| 6 | userRegisterEmail | 邮箱用户注册 |
| 7 | resetPasswd | 找回密码接口 |
| 8 | modifyPassword | 修改密码接口 |
| 9 | getAtUser | 常用联系人搜索接口 |
| 10 | searchContacts | 搜索联系人 |
| 11 | getKaptchaImage | 图片验证码接口 |
| 12 | getUserInfo | 根据userId查询用户信息 |
| 13 | updateUserInfo | 修改用户信息 |
| 14 | face2faceJoinEnterprise | 面对面加入企业 |
| 15 | joinByWechatInviteCode | 通过微信邀请码加入企业 |
| 16 | joinByInviteCode | 通过qq邀请码加入企业 |

### Client类概述

封装对Thrift接口的调用，基于zookeeper进行 多点分发和监控。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | userLogin | 登录接口 |
| 2 | userLogout | 退出接口 |
| 3 | getEmailVerifyCode | 发送邮箱验证码 |
| 4 | getMSISDNVerifyCode | 发送手机短信验证码 |
| 5 | userRegisterMobile | 手机用户注册 |
| 6 | userRegisterEmail | 邮箱用户注册 |
| 7 | resetPasswd | 找回密码接口 |
| 8 | modifyPassword | 修改密码接口 |
| 9 | getAtUser | 常用联系人搜索接口 |
| 10 | searchContacts | 搜索联系人 |
| 11 | getKaptchaImage | 图片验证码接口 |
| 12 | getUserInfo | 根据userId查询用户信息 |
| 13 | updateUserInfo | 修改用户信息 |
| 14 | face2faceJoinEnterprise | 面对面加入企业 |
| 15 | joinByWechatInviteCode | 通过微信邀请码加入企业 |
| 16 | joinByInviteCode | 通过qq邀请码加入企业 |
| 17 | isQuanUser | 校验是否工作圈用户 |
| 18 | isCookieValid | 校验cookie是否有效 |
| 19 | syncAddTeamUser | 与智能平台同步圈子用户信息 |
| 20 | syncDelTeamUser | 与智能平台同步圈子用户信息 |
| 21 | isFirstLoginByDeviceType | 校验账号是否在某一终端类型第一次登录 |

### AccountServiceImpl类概述

服务端的接口实现类，实现Thrift的Iface接口，Client通过Thrift远程调用。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | userLogin | 登录接口 |
| 2 | userLogout | 退出接口 |
| 3 | getEmailVerifyCode | 发送邮箱验证码 |
| 4 | getMSISDNVerifyCode | 发送手机短信验证码 |
| 5 | userRegisterMobile | 手机用户注册 |
| 6 | userRegisterEmail | 邮箱用户注册 |
| 7 | resetPasswd | 找回密码接口 |
| 8 | modifyPassword | 修改密码接口 |
| 9 | getAtUser | 常用联系人搜索接口 |
| 10 | searchContacts | 搜索联系人 |
| 11 | getKaptchaImage | 图片验证码接口 |
| 12 | getUserInfo | 根据userId查询用户信息 |
| 13 | updateUserInfo | 修改用户信息 |
| 14 | face2faceJoinEnterprise | 面对面加入企业 |
| 15 | joinByWechatInviteCode | 通过微信邀请码加入企业 |
| 16 | joinByInviteCode | 通过qq邀请码加入企业 |
| 17 | isQuanUser | 校验是否工作圈用户 |
| 18 | isCookieValid | 校验cookie是否有效 |
| 19 | syncAddTeamUser | 与智能平台同步圈子用户信息 |
| 20 | syncDelTeamUser | 与智能平台同步圈子用户信息 |
| 21 | isFirstLoginByDeviceType | 校验账号是否在某一终端类型第一次登录 |

### CspService类概述

封装工作圈对云平台的账户系统（CIA）的访问接口 。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | sendRegisterMobileMsg | 发送手机验证码 |
| 2 | registerEnterpriseWithMobile | 通过手机激活玛注册企业客户信息 |
| 3 | sendEmailRegisterCode | 发送邮箱验证码 |
| 4 | registerEnterpriseWithEmailCode | 通过邮件激活玛注册企业客户信息 |
| 5 | validateCode | 判断验证码是否正确 |
| 6 | updateUser | 修改用户信息 |
| 7 | resetPasswd | 找回密码接口 |
| 8 | modifyPassword | 修改密码接口 |
| 9 | getAtUser | 常用联系人搜索接口 |
| 10 | searchContacts | 搜索联系人 |
| 11 | findUserByIdentify | 根据用户标示查询用户信息 |
| 12 | authFromCsp | 账户校验 |
| 13 | logout | 退出 |

### TokenCache类概述

封装工作圈对云平台的账户系统（CIA）的访问接口 。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | saveOrUpdateCache | 保存或修改缓存 |
| 2 | getCacheUser | 从缓存中获取缓存的用户 |
| 3 | recordUserLastLoginTime | 记录用户上一次登录工作圈的时间 |
| 4 | saveInviteCode | 邀请用户加入企业，验证码 |

### UserMongoDao类概述

账户系统对mongo的操作 。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | saveOrUpdateUser | 保存或变更用户信息 |
| 2 | updateUserInfo | 修改用户信息 |
| 3 | findUser | 查询用户信息 |
| 4 | isFirstLoginByDeviceType | 校验用户是否第一次登录某终端类型 |

## 圈子系统（quan）模块

### 圈子系统模块概述

工作圈的圈子系统是创建编辑圈子和帖子，并邀请或申请加入圈子的模块。

工作圈圈子系统，包括创建编辑删除圈子、创建删除展示帖子，邀请加入圈子，申请加入圈子等功能。

创建圈子时会默认创建一条帖子，告知用户已经创建了一个新圈子；获取圈子列表功能，可以查看我所有的圈子列表,以便找到并进入我关心的圈子；解散圈子后用户看不到已解散圈子的所有信息；邀请和申请加入圈子功能实现了圈子添加成员的作用。

### 圈子系统模块流程图



图5-1 圈子创建编辑流程图



图5-2 邀请添加圈子成员（泳道图）

### JsonController类概述

Controller的基类，封装统一较常用的方法，如参数校验、安全校验等。

类功能定义

表5-1

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | checkSig | 签名校验 |
| 2 | checkImageNo | 验证码校验 |
| 3 | changeTokenRel | Token切换 |
|  |  |  |

### RestTeamController类概述

给移动端提供的http接口 。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | addTeam | 添加圈子 |
| 2 | getTeamInfo | 获取圈子信息 |
| 3 | getTeamList | 获取圈子列表 |
| 4 | updateTeam | 修改圈子基本信息 |
| 5 | updateInvitelevel | 修改圈子的邀请级别 |
| 6 | removeTeam | 解散圈子 |
| 7 | getTeamByToken | 根据圈子token获取圈子信息 |
| 8 | getLogos | 获取圈子logo |
| 9 | findUserOrgsNum | 获取企业未读数和未审核数 |
| 10 | findUserComplexOrgInfo | 获取用户部门列表和企业用户数据 |

### WebTeamController类概述

给PC浏览器访问提供的http接口。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | addTeam | 添加圈子 |
| 2 | getTeamInfo | 获取圈子信息 |
| 3 | getTeamList | 获取圈子列表 |
| 4 | updateTeam | 修改圈子基本信息 |
| 5 | updateInvitelevel | 修改圈子的邀请级别 |
| 6 | removeTeam | 解散圈子 |
| 7 | getTeamByToken | 根据圈子token获取圈子信息 |
| 8 | memberStatus | 获取圈子信息并返回用户和圈子的关系 |
| 9 | getLogos | 获取圈子logo |
| 10 | uploadTeamLogo | 上传圈子logo |
| 11 | getUnReadCount | 获取圈子中帖子的未读数 |
| 12 | findUserComplexOrgInfo | 获取用户部门列表和企业用户数据 |
| 13 | checkTeamMembers | 检查圈子成员 |

### RestTopicController类概述

给移动端访问提供的http接口。

类功能定义

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 功能点 | | 功能点详细内容 |
| 1 | addTopic | | 发帖 |
| 2 | removeTopic | | 删除帖子 |
| 3 | getTeamTopics | | 获取某圈子的帖子 |
| 4 | getTeamTopicsWithoutTeam | | 获取某圈子的帖子结果不返还圈子信息 |
| 5 | getTopicInfo | | 获取单个帖子 |
| 6 | setAnchor | | 帖子置顶 |
| 7 | getUserTopics | | 获取某用户的帖子 |
| 8 | collect | | 收藏帖子 |
| 9 | uncollect | | 取消收藏x |
| 10 | getCollectedTopics | | 获取收藏的帖子列表 |
| 13 | | checkTeamMembers | 检查圈子成员 |

### WebTopicController类概述

给PC浏览器访问提供的http接口。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | addTopic | 发帖 |
| 2 | removeTopic | 删除帖子 |
| 3 | getTeamTopics | 获取某圈子的帖子 |
| 4 | getTopicInfo | 获取单个帖子 |
| 5 | setAnchor | 帖子置顶 |
| 6 | getFullContent | 获取帖子的全部内容 |
| 7 | getUserFeeds | 获取用户的瀑布流 |
| 8 | getUserTopics | 获取用户的帖子 |
| 9 | collect | 收藏帖子 |
| 10 | uncollect | 取消收藏 |
| 11 | getCollectedTopics | 获取收藏的帖子列表 |

### RestMemberController类概述

给移动端访问提供的http接口。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | invite | 邀请用户加入圈子 |
| 2 | apply | 用户申请加入圈子 |
| 3 | acceptInvite | 同意邀请 |
| 4 | acceptAcpply | 接受申请 |
| 5 | refuseInvite | 拒绝邀请 |
| 6 | refuseAcpply | 拒绝申请 |
| 7 | cancelInvite | 取消邀请 |
| 8 | getInviteRecord | 获取邀请记录 |
| 9 | getApplyRecord | 获取申请记录 |
| 10 | getTeamApplyRecords | 获取某圈子的申请记录 |
| 11 | getUserApplyRecords | 获取某用户的申请记录 |
| 12 | getTeamInviteRecords | 获取某圈子的邀请记录 |
| 13 | getUserInviteRecords | 获取某用户的邀请记录 |
| 14 | getPendingApplies | 获取待定的申请 |
| 15 | inviteBatch | 批量邀请成员 |
| 16 | inviteBatchByAccAndId | 通过账号或userId批量邀请成员 |

### WebMemberController类概述

给pc浏览器访问提供的http接口。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | invite | 邀请用户加入圈子 |
| 2 | apply | 用户申请加入圈子 |
| 3 | acceptInvite | 同意邀请 |
| 4 | acceptAcpply | 接受申请 |
| 5 | refuseInvite | 拒绝邀请 |
| 6 | refuseAcpply | 拒绝申请 |
| 7 | cancelInvite | 取消邀请 |
| 8 | getInviteRecord | 获取邀请记录 |
| 9 | getApplyRecord | 获取申请记录 |
| 10 | getTeamApplyRecords | 获取某圈子的申请记录 |
| 11 | getUserApplyRecords | 获取某用户的申请记录 |
| 12 | getTeamInviteRecords | 获取某圈子的邀请记录 |
| 13 | getUserInviteRecords | 获取某用户的邀请记录 |
| 14 | inviteBatch | 批量邀请成员 |
| 15 | batchByExcel | Excel批量邀请 |
| 16 | authJoin | 验证账号并加入圈子 |
| 16 | inviteBatchByAccAndId | 通过账号或userId批量邀请成员 |

### TeamServiceImp类概述

服务端的接口实现类，实现Thrift的Iface接口，Client通过Thrift远程调用。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | getTeamInfo | 获取单个圈子信息 |
| 2 | getTeamList | 获取圈子列表 |
| 3 | createTeam | 创建圈子 |
| 4 | createDefaultTopic | 创建默认帖子 |
| 5 | removeTeam | 解散圈子 |
| 6 | updateTeam | 修改圈子 |
| 7 | getTeamStatusList | 移动端获取圈子信息 |
| 8 | getWebTeamStatusList | Web端获取圈子信息 |
| 9 | getTeamInfoByToken | 根据token获取圈子信息 |
| 10 | getUnReadCount | 获取圈子未读数 |
| 11 | updateTeamMasterid | 修改圈子的圈主 |

### TopicServiceImp类概述

服务端的接口实现类，实现Thrift的Iface接口，Client通过Thrift远程调用。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | getTopicInfoWithDeleted | 获取单个帖子信息，包含删除的帖子 |
| 2 | getTopicInfo | 获取帖子信息 |
| 3 | getTopicInfoList | 获取帖子列表 |
| 4 | getTeamTimelineBefore | 获取某圈子的帖子 |
| 5 | getUserTimelineBefore | 获取某人某时间后的数量的帖子 |
| 6 | getAnchorTopics | 获取置顶帖子 |
| 7 | create | 创建帖子 |
| 8 | createWithoutCheck | 创建帖子不需要验证用户权限 |
| 9 | remove | 删除帖子 |
| 10 | setAnchor | 置顶帖子 |
| 11 | getRemTopicids | 获取删除帖子id |
| 12 | getUserFeeds | 获取用户帖子瀑布流 |
| 13 | isHaveNewTopic | 获取指定用户置顶时间新帖子数 |
| 14 | collect | 收藏 |
| 15 | uncollect | 取消收藏 |
| 16 | getCollectedTopics | 获取收藏的帖子 |

### MemberServiceImp类概述

服务端的接口实现类，实现Thrift的Iface接口，Client通过Thrift远程调用。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | getTeamInviteRecords | 获取某圈子的邀请记录 |
| 2 | getUserInviteRecords | 获取某用户的邀请记录 |
| 3 | getInviteRecord | 获取一条邀请记录 |
| 4 | inviteBatch | 批量邀请 |
| 5 | invite | 邀请加入圈子 |
| 6 | acceptInvite | 接受邀请加入圈子 |
| 7 | refuseInvite | 拒绝邀请加入圈子 |
| 8 | cancelInvite | 取消邀请 |
| 9 | getTeamApplyRecords | 获取圈子申请记录列表 |
| 10 | getUserApplyRecords | 获取用户申请记录列表 |
| 11 | getApplyRecord | 获取申请记录 |
| 12 | getApplyRecordList | 获取申请记录列表 |
| 13 | apply | 申请加入圈子 |
| 14 | acceptApply | 接受申请 |
| 15 | refuseApply | 拒绝申请 |
| 16 | cancelApply | 取消申请 |
| 17 | getTeamApplyCount | 获取申请加入圈子数 |
| 18 | acceptInviteByUserId | 通过用户ID接受邀请并加入圈子 |

## 关系系统（Relation）模块

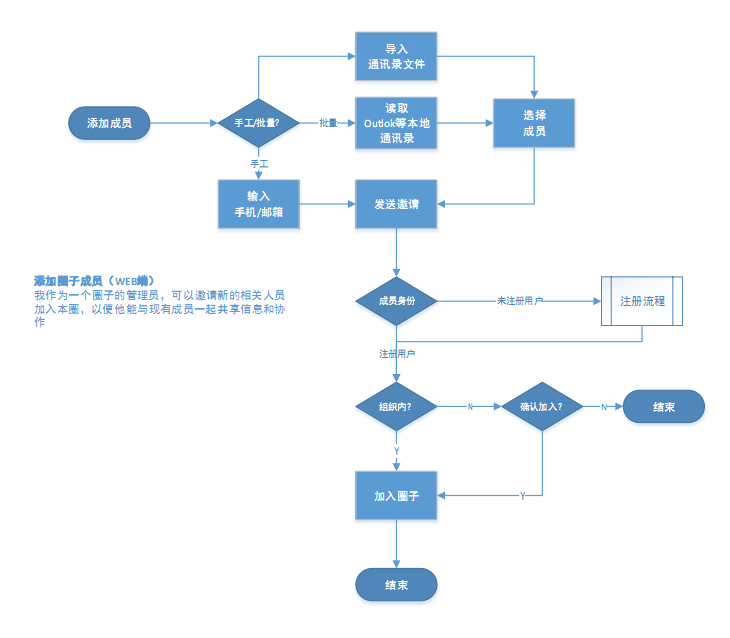
### 关系系统模块概述

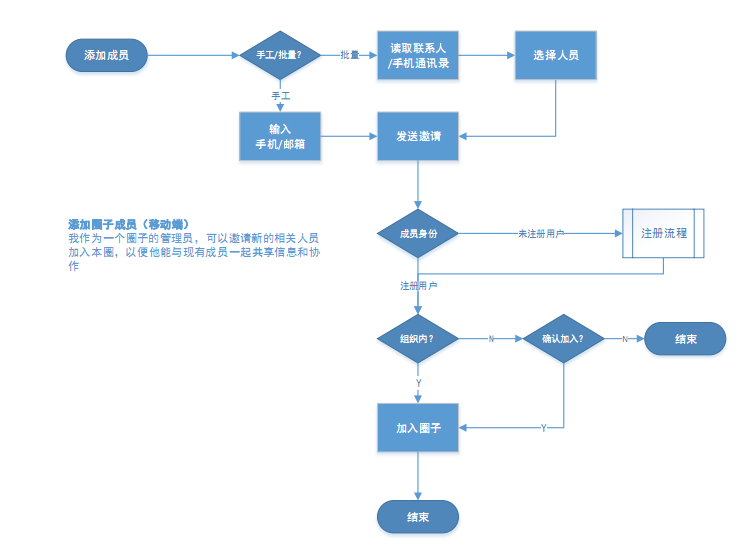
工作圈的关系系统是存储用户与圈子的所属关系，主要维护包含人所对应的圈子、圈子所对应的人、人在圈子的角色、人的置顶圈子等数据。

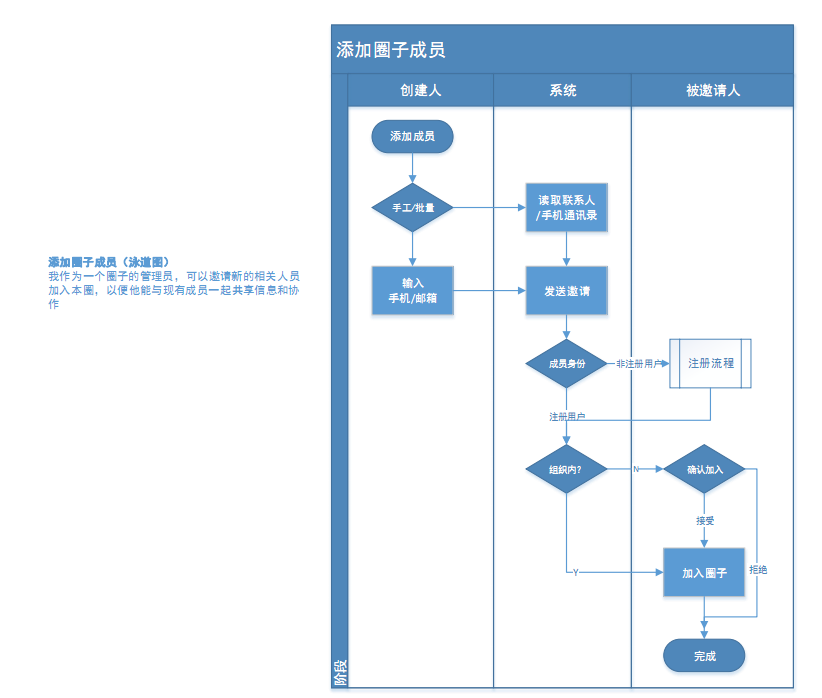
**查看圈子列表：**

1. 圈子列表（交互优先级）：圈子的名称、未读数量、logo、最新的会话
2. 圈子排序
3. 方案一：按动态（发言和回复）的时间倒序
4. 方案二：智能排序（参见S-00035）
5. 未读数量：圈子里的未读条数，
6. 圈子的置顶（限WEB端）
7. 允许设置三个圈子置顶，置顶后圈子将固定在最上方，不会因为动态而改变顺利。
8. 超过三个圈子，系统提示不允许再置顶。
9. 取消置顶后，该圈子的排序将重排。
10. 移动端加载规则
11. 切换到首页时，会自动加载圈子列表内容及圈子排序
12. 圈子中有动态更新，每30秒会推送给移动端，排序不刷新，只刷新消息数量；
13. 手动向下滑动时重新加载列表内容和排序
14. 圈子列表智能排序规则
15. 权重高：是否有未读、最新更新时间、我进入圈子的频率、我在圈子中的浏览时间
16. 权重中：通知我的频率、我发贴的次数频率、我回复的次数比率、是否我是圈主。
17. 权重低：未读消息数量

**添加圈子成员流程：**







1. 邀请权限
2. 圈子管理中选中“限圈主邀请”，就只有管理员能发出邀请
3. 圈子管理中选中“成员均可邀请”，圈子中的所有成员均可以邀请
4. 通过二维码邀请的相关权限请参考S-00077的需求描述
5. 手工邀请
6. [输入完整邮箱地址\*@\*.\*（最长50](mailto:输入完整邮箱地址*@*.*（最长50)个字符），发出邀请
7. 或输入11位手机号，发出邀请
8. Web端输入框支持批量输入、支持通过Excel复制粘贴，支持自动解析块显示（详见交互说明）；
9. 批量邀请
10. 读取手机本地通讯录，选择要邀请的联系人，支持搜索（移动端）
11. 读取工作圈联系人，选择要邀请的联系人，支持搜索（所有端）
12. 通过邀请微信好友添加圈子成员（移动端S-00077）
13. Web端隐藏Excel导入；
14. 参与搜索的字段及权重排序：姓名（汉字、全拼、简拼）、手机号、邮箱、电话、企业、部门、职位、地址；
15. 未注册用户：

如果被邀请人还没有注册，TA收到邀请后，需要完成注册并加入圈子

1. 邀请通知
2. 已注册用户（工作圈用户）：会收到系统通知
3. 未注册用户（非工作圈用户）：收到短信或邮件通知TA注册。完成注册后，会收到系统通知，参见已注册用户的流程。
4. 注意，这里的工作圈用户特指登录使用过工作圈的用户，如果已是CSP用户，如会计家园的某用户，虽然从来没有安装登录使用过工作圈，但可以用原有账号登录工作圈，这类用户也要归属到未注册用户类，被邀请时依然走短信或邮件通知，因为没有登录过工作圈，也就无从谈起收到系统通知
5. 圈子成员限制（本版本限1000人上限）
6. 邀请人在邀请时达到圈子用户上限（提示语：圈子成员达到XX人上限，您无法再邀请新的用户加入圈子）

提示规则：圈子成员等于或超过圈子限制时提示

提示需求：在手工邀请、批量邀请时判断，不允许发出邀请

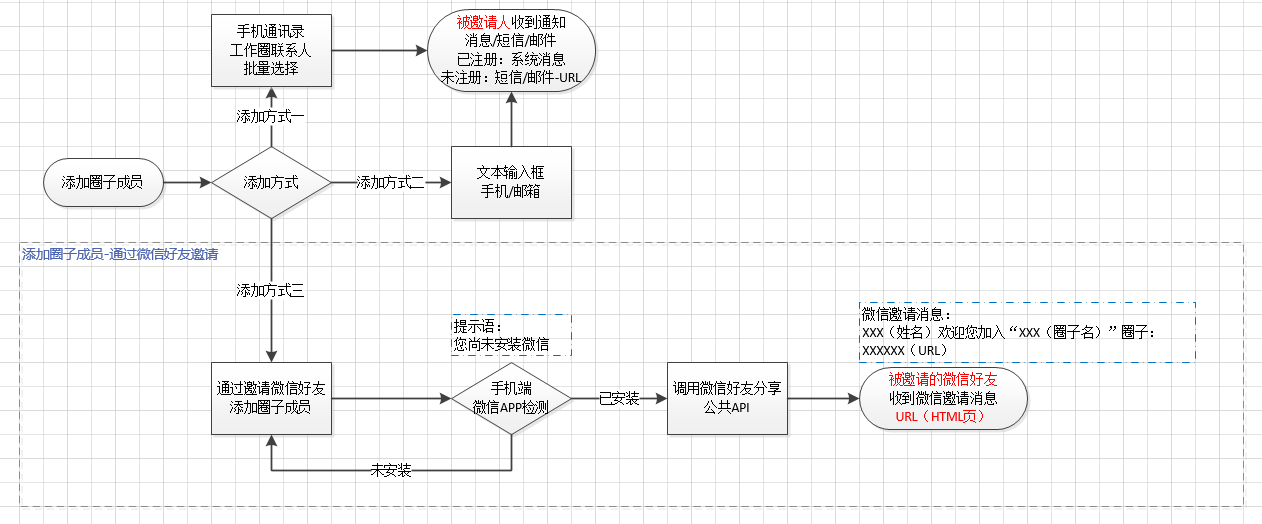
1. 被邀请人接受时达到圈子用户上限（提示语：圈子成员已满，您无法加入圈子，请与圈主联系）

提示规则：接受邀请时，圈子成员已达到上限时提示

提示需求：在接受邀请时判断，不允许该用户加入圈子

1. 用户人数限制只限是已加入的圈子成员，不包含邀请中的用户和退出的成员
2. 加入圈子（注册用户）
3. 用户收到邀请后，无论是否在同一企业，都需要确认是否加入，同意后才可以加入到圈子中，拒绝则不进入圈子
4. 关于未注册用户（非工作圈用户）收到圈子邀请后（短信/邮件），通过URL（要求短链形式）着陆到HTML页面完成加入的过程：
5. 用户点击链接后，会着陆到加入圈子的HTML页；根据用户发起端（移动端/PC端）进行页面自动适配；两种页面内容一致（页面中包含圈子名称等，具体参见UE交互图）
6. 加入圈子HTML页的需求说明：
   1. 默认URL着陆页是注册新用户（默认被邀请的手机号/邮箱）并加入圈子，用户填写完相关注册信息后自动跳转到下载页
   2. 如果用户已有工作圈账号，在上面的URL默认着陆页中有“使用已有账号登录并加入圈子”的链接，点击后会自动切换到使用已有账号加入圈子的页面（类似登录），验证通过后会自动跳转到下载页，下载页上需要有打开工作圈的链接
7. URL着陆页的相关提示同注册和申请加入圈子
8. URL需要在后台做处理，比如token的形式，以保证一个URL只处理一次有效加入
9. 需要后台考虑并设计URL链接的有效期，比如可以设定N天，本版暂定有效期30天
10. 选择企业通讯录（添加公司同事）
11. 在添加成员页面中支持“选择企业通讯录”入口，允许复选，不可以添加自己
12. 如果用户没有加入企业（属于虚拟企业），不显示该入口
13. 如果用户有多企业，默认企业与工作模块的企业一致，排序按加入企业的时间顺序。用户可以选择其它企业，然后再添加所选企业的同事。
14. 调用企业通讯录，显示成员的头像、姓名、邮箱或手机（两者都有优先显示邮箱）。如果该成员（关联的账号）已是工作圈用户的直接发邀请消息；非工作圈用户会收到注册邀请（短信/邮件），如果用户的手机和邮箱都有，优化用邮件通知。
15. 已添加用户：如果通讯录中的成员（关联的账号）已经加入该圈子，则不允许再次添加。
16. 成员搜索：支持姓名（全拼、简拼、姓名）、手机、邮箱
17. 其它限制和规则同圈子成员邀请的规则一致。
18. 邀请选项
19. 仅限圈主邀请
20. 全员邀请，需圈主审核（如是圈主邀请，则不需要再次审核，被邀请人同意后，即可加入。创建圈子时，默认选中项）
21. 全员邀请，无需圈主审核（原所有成员均可邀请，升级为该项）
22. 二维码规则
23. 仅限圈主邀请（二维码仅圈主可见）
24. 全员邀请，需圈主审核（全员可见，申请时需圈主审核）
25. 全员邀请，无需圈主审核（全员可见，申请时不需圈主审核即可加入）
26. 圈子达到上限，不允许加入圈子
27. 成员或圈主邀请时提示“圈子成员达到XX人上限，您无法再邀请新的用户加入圈子”
28. 用户被邀请确认时提示“圈子成员已满，您无法加入圈子，请与圈主联系”。
29. 圈主审核时时提示“圈子成员达到XX人上限，新的用户无法加入圈子”
30. 用户进行二维码申请确认时，提示“圈子成员已满，您无法加入圈子，请与圈主联系”。
31. 被邀请人已是圈子成员，不能重复加入
32. 成员或圈主邀请时提示“该用户已经在本圈里”
33. 用户已在本圈中，确认时被邀请时提示“您已经在本圈子里，无需重复加入”
34. 圈主审批时提示“该用户已经在本圈里”
35. 二维码申请确认时，提示“您已经在本圈子里，无需重复加入

**添加圈子成员流程**



**管理圈子成员：**

1. 成员列表（按优先级）
2. 头像：只有活跃的成员有头像
3. 姓名：只有活跃的成员有姓名
4. 邮箱：以邀请时发送的邮箱为准
5. 手机：以邀请时发送的手机号为准
6. 状态：活跃、邀请中、删除（用户不可见）
7. 操作：
8. 邀请：圈主或有权限的成员邀请新的成员加入（参见S-00031添加圈子成员）
9. 删除：圈主或圈子管理员移除圈子里的成员。或删除邀请中的记录，此时邀请行为无效，被邀请人接受邀请也无法进入到圈子。
10. 重邀：对没有确认的用户重新发出邀请，邀请方式与上一次方式一致。
11. 接受：非组织内的用户接到邀请，接受邀请后会进入圈子（活跃状态）
12. 拒绝：非组织内的用户接到邀请，拒绝后不会进入圈子（停用状态）
13. 变更圈主（所有者）：原所有者可以把圈子的所有者变更为其他人，同时自己将不在成为所有者，变更后新的所有者将会收到一条通知

（暂不支持）

1. 管理员设定：可以指定多个成员为该圈子管理员，只对活跃成员有效，也可以撤消管理员。（暂不支持）
2. 成员状态
3. 活跃：已加入圈子的成员
4. 邀请中：没有确认加入或没有注册的用户。

删除（停用）：被圈主移除或主动退出的成员，拒绝加入的用户（组织外）。该状态对用户不可见。

### 关系系统模块类图



图3-1 Relation模块类图

### JsonController类概述

Controller的基类，封装统一较常用的方法，如参数校验、安全校验等。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | checkSig | 签名校验 |
| 2 | internalServerError | 统一异常返回 |
| 3 | roleValueInvalid | 角色参数校验 |
|  |  |  |

### WebTeamMemberServiceController类概述

给PC浏览器访问提供的http接口 。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | getTeamsByUser | 获取一个用户所属的所有圈子的列表，可以根据成员类型进行过滤 |
| 2 | getUsersByTeam | 获取一个圈子所属的所有用户的列表 |
| 3 | getUsersByPage | 分页获取一个圈子所属的所有用户的列表 |
| 4 | getTeamMember | 获取圈子成员关系对象 |
| 5 | removeTeamMember | 退出圈子，移除成员 |
| 6 | updateTeamMember | 更改圈子成员关系信息 |
| 7 | getChangedUsers | 增量获取圈子的成员变化 |
| 8 | getChangedTeams | 增量获取成员的圈子变化 |
| 9 | checkUserCanBeInvited | 邀请一个圈子的用户到另一个圈子，判断用户是否已在圈子里 |
| 10 | transferTeamOwner | 圈主移交 |

### RestTeamMemberServiceController类概述

给移动端访问提供的http接口 。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | getTeamsByUser | 获取一个用户所属的所有圈子的列表，可以根据成员类型进行过滤 |
| 2 | getUsersByTeam | 获取一个圈子所属的所有用户的列表 |
| 3 | getUsersByPage | 分页获取一个圈子所属的所有用户的列表 |
| 4 | getTeamMember | 获取圈子成员关系对象 |
| 5 | removeTeamMember | 退出圈子，移除成员 |
| 6 | updateTeamMember | 更改圈子成员关系信息 |
| 7 | getChangedUsers | 增量获取圈子的成员变化 |
| 8 | getChangedTeams | 增量获取成员的圈子变化 |
| 9 | checkUserCanBeInvited | 邀请一个圈子的用户到另一个圈子，判断用户是否已在圈子里 |
| 10 | transferTeamOwner | 圈主移交 |

### BasicTeamMemberService类概述

http接口的业务实现层，与client进行交互。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | getTeamsByUser | 获取一个用户所属的所有圈子的列表，可以根据成员类型进行过滤 |
| 2 | getUsersByTeam | 获取一个圈子所属的所有用户的列表 |
| 3 | getUsersByPage | 分页获取一个圈子所属的所有用户的列表 |
| 4 | getTeamMember | 获取圈子成员关系对象 |
| 5 | removeTeamMember | 退出圈子，移除成员 |
| 6 | updateTeamMember | 更改圈子成员关系信息 |
| 7 | getChangedUsers | 增量获取圈子的成员变化 |
| 8 | getChangedTeams | 增量获取成员的圈子变化 |
| 9 | checkUserCanBeInvited | 邀请一个圈子的用户到另一个圈子，判断用户是否已在圈子里 |
| 10 | transferTeamOwner | 圈主移交 |

### RestBasicTeamMemberService类概述

http接口的业务实现层，与client进行交互 ，组合依赖BasicTeamMemberService的实现，尽量简洁代码。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | getTeamsByUser | 获取一个用户所属的所有圈子的列表，可以根据成员类型进行过滤 |
| 2 | getUsersByTeam | 获取一个圈子所属的所有用户的列表 |
| 3 | getUsersByPage | 分页获取一个圈子所属的所有用户的列表 |
| 4 | getTeamMember | 获取圈子成员关系对象 |
| 5 | removeTeamMember | 退出圈子，移除成员 |
| 6 | updateTeamMember | 更改圈子成员关系信息 |
| 7 | getChangedUsers | 增量获取圈子的成员变化 |
| 8 | getChangedTeams | 增量获取成员的圈子变化 |
| 9 | checkUserCanBeInvited | 邀请一个圈子的用户到另一个圈子，判断用户是否已在圈子里 |
| 10 | transferTeamOwner | 圈主移交 |

### TokenCache类概述

Redis缓存交互类。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | getTeamsByUserId | 获取一个用户所属的所有圈子的列表，可以根据成员类型进行过滤 |
| 2 | getUsersByTeamId | 获取一个圈子所属的所有用户的列表 |
| 3 | getUsersByPage | 分页获取一个圈子所属的所有用户的列表 |
| 4 | getTeamMember | 获取圈子成员关系对象 |
| 5 | removeTeamMember | 退出圈子，移除成员 |
| 6 | updateTeamMember | 更改圈子成员关系信息 |
| 7 | getChangedUsers | 增量获取圈子的成员变化 |
| 8 | getChangedTeams | 增量获取成员的圈子变化 |
| 9 | checkUserCanBeInvited | 邀请一个圈子的用户到另一个圈子，判断用户是否已在圈子里 |
| 10 | transferTeamOwner | 圈主移交 |
| 11 | addOrUpdate | 添加或更新缓存 |

### TeamMemberDao类概述

与MongoDB交互类。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | getTeamsByUserId | 获取一个用户所属的所有圈子的列表，可以根据成员类型进行过滤 |
| 2 | getUsersByTeamId | 获取一个圈子所属的所有用户的列表 |
| 3 | getUsersByPage | 分页获取一个圈子所属的所有用户的列表 |
| 4 | getTeamMember | 获取圈子成员关系对象 |
| 5 | removeTeamMember | 退出圈子，移除成员 |
| 6 | updateTeamMember | 更改圈子成员关系信息 |
| 7 | getChangedUsers | 增量获取圈子的成员变化 |
| 8 | getChangedTeams | 增量获取成员的圈子变化 |
| 9 | checkUserCanBeInvited | 邀请一个圈子的用户到另一个圈子，判断用户是否已在圈子里 |
| 10 | getTeamOwner | 获取圈子的圈主 |
| 11 | addOrUpdate | 添加或更新缓存 |
| 12 | updateTeamStatus | 修改圈子状态，用于关闭、删除圈子 |

## 评论系统（Comment）模块

### 评论系统模块概述

工作圈的评论系统为工作圈提供了一个公共评论/回复服务。工作圈中涉及评论与回复相关业务的功能都可以使用这个服务来发布相关的评论信息。

评论系统为工作圈中的贴子，轻应用中的审批、请假、任务等功能提供了评论和回复评论的功能

并且基于畅捷通云平台的智能平台，为工作圈用户提供智能推荐、搜索服务。

### 评论系统模块类图



图3-1 Comment模块类图

### JsonController类概述

Controller的基类，封装统一较常用的方法，如参数校验、安全校验等。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | checkSig | 签名校验 |
| 2 | checkImageNo | 验证码校验 |
| 3 | changeTokenRel | Token切换 |
|  |  |  |

### WebController类概述

给PC浏览器访问提供的http接口 。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | addComment | 新增一条评论 |
| 2 | addAppComment | 新增一条轻应用评论 |
| 3 | removeComment | 发送邮箱验证码 |
| 4 | getCommentsPaged | 删除一条评论 |
| 5 | getCommentsPaged | 获得一个主题下的评论列表 |
| 6 | getCommentNum | 获取贴子回复数量 |
| 7 | getCommentNums | 获取一组贴子回复数量 |
| 8 | isCommentExists | 判断评论是否存在 |

### RestController类概述

给移动端提访问提供的http接口 。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | addComment | 新增一条评论 |
| 2 | addAppComment | 新增一条轻应用评论 |
| 3 | removeComment | 删除一条评论 |
| 4 | getCommentsPaged | 删除一条评论 |
| 5 | getCommentsByTime | 获取一个时间截之后的贴子的所有分页评论列表 |
| 6 | getCommentsByTime1 | 获取一个时间截之后的贴子的所有分页评论列表 |
| 7 | getCommentNum | 获取贴子回复数量 |
| 8 | getCommentNums | 获取一组贴子回复数量 |
| 9 | isCommentExists | 判断评论是否存在 |
| 10 | addLeaveSheet | 新增一条请假的评论 |

### Client类概述

封装对Thrift接口的调用，基于zookeeper进行 多点分发和监控。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | addComment | 新增一条评论 |
| 2 | addAppComment | 新增一条轻应用评论 |
| 3 | removeComment | 删除一条评论 |
| 4 | getComment | 获取某一条评论详情 |
| 5 | getComments | 获得一个主题下的所有评论列表 |
| 6 | getCommentsPaged | 获得一个主题下的分页评论列表 |
| 7 | getCommentNum | 获取贴子回复数量 |
| 8 | getCommentNums | 获取一组贴子回复数量 |
| 9 | isCommentExists | 判断评论是否存在 |
| 10 | getCommentsByTime | 获取一个时间截之后的贴子的所有评论列表 |
| 11 | getCommentsByTime1 | 获取一个时间截之后的贴子的所有评论列表 |
| 12 | getRmCommentsByTime | 获取一个时间截之后的贴子的删除ID列表 |
| 13 | updateCommentBatch | 清洗数据，更新commentList的replytouser |
| 14 | getBatchCommentsByIds | 批量获取评论 |

### CommentServiceImpl类概述

服务端的接口实现类，实现Thrift的Iface接口，Client通过Thrift远程调用。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | addComment | 新增一条评论 |
| 2 | addAppComment | 新增一条轻应用评论 |
| 3 | removeComment | 删除一条评论 |
| 4 | getComment | 获取某一条评论详情 |
| 5 | getComments | 获得一个主题下的所有评论列表 |
| 6 | getCommentsPaged | 获得一个主题下的分页评论列表 |
| 7 | getCommentNum | 获取贴子回复数量 |
| 8 | getCommentNums | 获取一组贴子回复数量 |
| 9 | isCommentExists | 判断评论是否存在 |
| 10 | getCommentsByTime | 获取一个时间截之后的贴子的所有评论列表 |
| 11 | getCommentsByTime1 | 获取一个时间截之后的贴子的所有评论列表 |
| 12 | getRmCommentsByTime | 获取一个时间截之后的贴子的删除ID列表 |
| 13 | updateCommentBatch | 清洗数据，更新commentList的replytouser |
| 14 | getBatchCommentsByIds | 批量获取评论 |

### CommentAsyncService类概述

封装工作圈发送评论相关的通知消息业务。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | addCommentAsnyc | 增加评论的时候，发送评论通知 和 智能平台调用 （spring 异步） |

### ICommentCache类概述

封装工作较圈评论数据cache业务处理。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | addComment | 新增一条评论 |
| 2 | removeComment | 删除一条评论 |
| 3 | getComment | 查询一条评论 |
| 4 | updateCommentBatch | 清洗数据，更新commentList的replytouser |

### ICommentSortCache类概述

封装工作较圈评论idlist的cache业务处理。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | getComments | 获取资源下的所有评论 |
| 2 | getPagedComments | 获取资源下的评论，分页 |
| 3 | getCommentsByTime | 通过时间截来分页获取资源下的评论 |
| 4 | getRmCommentsIdByTime | 通过时间截来获取资源最进删除的评论 |
| 5 | getCommentsByIds | 通过评论ID来分页获取资源下的评论 |
| 6 | rmComments | 获取所有被删除的评论 |

### CommentDao类概述

封装工作较圈评论数据持久化业务处理。

类功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能点 | 功能点详细内容 |
| 1 | addOrUpdateComment | 添加/更新一个贴子的评论 |
| 2 | findCommentByResource | 贴子的评论列表 |
| 3 | findRmCommentsByResource | 贴子的已删队评论列表 |
| 4 | removeComment | 取消一条评论 |
| 5 | findCommentByCommentid | 通过一个评论的ID，查询评论 |
| 6 | findEffectiveCommentByCommentid | 通过一个评论的ID，查询有效评论 |

# 第七章 结束语



## 论文工作总结

本文分析了分布式计算的思想，并结合现代开源社区组件设计了一个高性能，高可用的分布式系统框架。给出了这个框发布式计算核心的服务注册中心，服务横向扩展以及数据分片算法的实现等。这些模块既可以组合在一起成为一个完整的分布式服务框架，同时也可以独立出来为其它的一些服务系统提供解决方案。

本文取得的主要成果有：

1. 阐述了分布式思想
2. 设计了一个以开源技术为基础的分布式服务框架
3. 以分布式服务框架为基础，完成了一个基于移动互联网的企业协同办公软件服务

## 问题和展望

本文探讨的 这个分布式框架，有两个不太明显的性能瓶颈。说不太明显，是因为这两个问题是在两个数据量非常大的情况下会出现的。一个是分布式注册中心，当注册的服务非常之多的时候，它的性能会有问题。另一个是分布式数据索引服务，当业务数据量过大，而且没有很好的进行分片的时候会对这个服务有较大的影响。

互联网与移动互联网将人类的沟通拉进了一个前所未有的地球村，从世界的一个地方发送一个信息到遥远的另一方不过短短的数秒，人们之前的交流变的非常便利，分布式的服务在其中做了很大的贡献。本文所描述的框架还有许多可以扩展的地方。

Docker，如果说近两年虚拟化最火的热词无疑就是这个技术了。它以最轻量级的方式为程序运行打造一个完整的虚拟机环境，可以快速的分发、部署与运行。这样，分布式服务就能方便的在业务高峰时段进行弹性扩展与收缩。

Spark，一个基于Hadoop的流数据处理组件。互联网的分布式服务和大数据处理两者是相辅相成的，当用户量达到千万级别的时候，产生的数据是非常可观的，同时其中的价值也是巨大的。

参考文献

[1] 吴强 分布式系统一致性的发展历史（一） 点融黑帮 2015年9月8日

[2] L Lamport Time, Clocks, and the Ordering of Events in a Distributed System Operating Systems July 1978 Fig.1

[3] 黄晓军,张静，张凯 Apache Thrift - 可伸缩的跨语言服务开发框架 IBM developWorks 2012年1月16日

[4] 许令波 分布式服务框架 Zookeeper -- 管理分布式环境中的数据IBM developWorks 2010年11月18日

[5] canezk, icbd, 许瑞等 译 Redis介绍 Redis.io 2016年4月27日

[6] 金蓓弘，马应龙等 译 分布式系统：概念与设计 原书第5版 机械工业出版社 2013年3月1日

[7] Alexander Misel, Abadcafe, ZéroBot等 CAP定理 Wikipedia.org 2015年12月11日

[8] 王璞 分布式系统的特点以及设计理念 InfoQ 2015年6月22日

致 谢

对于提供各类资助、指导和协助完成论文研究工作的单位及个人表示感谢。致谢应实事求是，切忌浮夸与庸俗之词。（用小四宋体）