**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： 2011127136**

**姓 名： 赵炜**

**专 业： 软件工程**

**导 师： 吴国仕**

**学 院： 软件学院**

**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设计与实现

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。移动互联网服务极大丰富了我们的日常生活。移动社交（微信、微博）、掌上购物（淘宝、京东、天猫）、移动支付（支付宝、微信、ApplePay）等等。现在很多的行业都在谋求能够赶上移动互联网这趟高速列车。但是想要转型移动互联网，企业的管理首先就要跟的上。首先，大部分的企业在企业协同的软件上投入很少，之前使用的大多数是ERP、CRM类型的软件，而且软件更新的速度也很慢，大部分的软件是半年或者一年一次更新，两年或更长时间才有一个大版本的发布，这样的更新速度是远远赶不上用户的需求，赶不上业务的发展的；其次，国内高速的经济发展催生了一大批创业者，他们成立的小微企业人数较少但是数量却已经达到千万级别，对于小微企业的协同软件市场国内也是刚刚起步，还很简陋；数据孤岛也式现代企业急需解决的问题，部门与部门之间难协同，企业上下游数据不同都阻碍了企业的成长。现在的企业协同市场急需符合现在移动互联网浪潮的一款企业协同服务。

云服务的出现让人们看到了解决这些问题希望。将企业协同的服务云化，把业务都放到云端将有效的打通企业内，企业与企业之间的数据；将企业协同移动化，让企业协同具备一定的社交属性，将极大的加强了企业内、企业与企业之间的协同能力，减低协同的成本。同时云化能加快软件服务的更新，使得企业协同服务能力的提升速度能够赶上社会的发展。然而，云化也对企业协同服务提出了新的挑战。将上千万的企业协同服务由云端来提供，无疑对云端的服务是一个很大的压力。本文以Apache开源社区的两大开源项目Zookeeper和Thrift为基础，结合分布式的思想设计式和实现一个高性能，高可用的一个企业协同云服务来满足需求。

**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

原始社会到工业作坊到现代企业都存在一定的制度和规矩来协同整个组织的运作，这就是所谓的管理（没有规矩不成方圆）；管理的目的是让每一个人发挥所长而不是限制每一个人，而每一个人的特长发挥需要更好的协同。协同在任何组织天然存在：因为人类社会存在专业分工、既有分工必然需要协同。从远古时代的飞鸽传信、烽火台、飞马传书、驿站、官员奏章、朝廷颁令，到近代的书信邮局、电报传真、电话、红头文件、内部会议，再到现代电子邮件、即时通讯、移动手机、视频通讯、微博微信……所有这些都是为了满足更高效实现一个组织目标而发明的协同工具。从现代文明开始，随着电脑科技的发展、企业组织的发达，也就产生了现在的协同(OA)软件。协同(OA)软件的发展与互联网的发展基本保持一致，从刚开始有邮件、电子公告板、BBS，到独立网站的出现、到分类搜索网站、到门户网站（新浪、网易）、到全局搜索引擎的出现、到强化个人分享的博客出现、到实时个人分享的社交化微博出现，到目前微信的兴起，社会互联网的发展一直带动着协同(OA)软件的发展。互联网的发展脉络是从点对点、点对面、面对面、面对点、点对点直到回归以每一个人为核心的应。

协同(OA)软件以应用为核心的阶段发展：

第一阶段：文档电子化（1991-1997）

第二阶段：电子邮件的普及（1998-2003）

第三阶段：公文行政办公和内网建设为核心（2003-2008）

第四阶段：工作流程管理为核心（2007-2013）

第五阶段：信息门户阶段（集成为核心）（2012-？）

第六阶段：？

我们看到，协同(OA)软件发展的前几个阶段中是以PC为主要的产生工具，而且数据主要的载体也是在PC端。从应用的部署来看，不管是C/S结构的方式还是B/S结构的方式，目前都是以PC端为主来部署和应用的。PC的优点，计算速度快，可视面积大；缺点，不便携，自备电源待机时间短，不适应长时间的外出工作。这样就提出了智能移动终端的需求，我们需要一个可以随身携带，使用和待机时间长而且功能丰富的终端来代替PC，智能手机无疑就是最佳的选择了。接下来我们以管理的内容来看看协同(OA)软件的发展。前三个阶段，管理的内容主要有：文档、邮件、公告、内部新闻等一些静态的资源。第四个阶段，提出了一个新概念：工作流程管理。此时企业的内部就有了很多的协同(OA)软件。员工为了完成工作可能需要使用不只一种的协同办公软件，相当的繁锁。在第五阶段，为了解决这个问题，提出了信息门户，重点在于集成这之前提供的众多种类的协同(OA)软件。起到了一点成效，提高了员工的工作效率。在这个集成的门户中，数据孤岛的问题依然没有能很好的解决。现在，越来越多的企业意识到信息流通的重要性，不只在企业的内部打通信息解决孤岛，更想要加强企业上下游之间的沟通，提升企业的决策能力。云服务，成为了一个最优解决的方案。综上，一个基于云上的企业协同服务方案成为了现在移动互联网行业中的热门方向。

将所有的企业协同业务放在云上的优点很多，同样带来的挑战也很大。截止2015年下半年，国内的注册企业数已达7000万家，从业人员还要番几番。为如此庞大的用户群提供服务，一几台高性能服务器的提供服务是远远不够的。眼下服务器性能的增长已经到达一个瓶颈，而分布式服务越来越受到大家的重视。基于分布式的思想，将服务部署在多个服务器上，将用户的请求以一个固定的算法平均分布到不同的服务器上去处理。由于企业服务的特殊性，大部份员工不是7×24在线使用，所以这个服务还需要具务弹性扩展的能力，不仅能随着用户的的增长来增加自己的服务能力，同时又能在空闲时间来收缩服务器的数量达到对节约成本的目的。

本文的核心是通过Zookeeper与Thrift技术来构建一个具有高性能以及高可用性的企业协同云服务。分布式服务的构建是一个相对复杂的过程，需要考虑数据一致性、服务高可用性以及分区等特性的取舍以及实现方式。本文在对协同云服务进行设计的同时，采用分布式的策略来进行构建。使用Zookeeper与Thrift来实现分布式的服务，指出系统的设计难点与瓶颈，给出相应的解决方案。

## 课题任务

### 课题内容

本课题的主要目标是基于畅捷通公司（以下简称为畅捷通）的用户系统设计和实现了一款企业协同云服务——工作圈的后台服务。工作圈是一款基于云服务的协同办公软件，它将为畅捷通公司的所有企业客户提供企业协同的云服务，包括：企业组织架构与员工的管理（企业通讯录）、办公OA功能（轻应用下的公告，请示、审批、请假、报销、派活、签到等功能）、沟通功能（IM聊天，群聊等）、企业上下游协同能力（圈子、贴子，赞等）。工作圈有PC WEB、Android以及iOS平台的软件包。本文重点首先是设计一个满足上述所有功能的并同时为三个平台的软件包提供服务的协同云后台服务，着重于分析需求，设计服务整体框架与模块等。

### 本人承担任务

本人在项目中承担了对服务整体技术架构的设计与实现，开发了项目技术架构中一些核心的组件与服务。开发的组件有：数据持久化组件(gongzuoquan-mongo)，缓存服务存取组件(gongzuoquan-cache)。开发的核心服务有：全局唯一标识生成的服务(gongzuoquan-idcenter)，分布式数据索引服务(gongzuoquan-idlist)。

## 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，

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

本章将针对在该论文中涉及的分布式系统的思想与运用到的核心技术进行介绍。运用到的核心技术主要包括：Apache的两个顶级开源项目Zookeeper与Thrift，还有Redis服务与Mongo数据库。这些技术都是在最近十年的移动互联网中，应对上亿用户的高并发而开发的组件。

## 分布式系统

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。

## 开源技术

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。

### 高性能HTTP服务器Nginx

Nginx ("engine x") 是一个高性能的 HTTP 和 反向代理 服务器，也是一个 IMAP/POP3/SMTP 代理服务器。 Nginx 是由 Igor Sysoev 为俄罗斯访问量第二的 Rambler.ru 站点开发的，它已经在该站点运行超过两年半了。 Igor 将源代码以类 BSD 许可证的形式发布。尽管还是测试版，但是，Nginx 已经因为它的稳定性、丰富的功能集、示例配置文件和低系统资源的消耗而闻名了。在本文中将使用Nginx作为云服务的前端请求的代理与负载均衡。

## 本章小结

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）。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 企业协同办公云服务的需求分析

在企业中，除了办公自动化系统之外，还有财务、库存、生产、销售、人力等管理系统。由于大量的信息孤岛式的建设，系统之间很少能够紧密协调起来，用户经常需要进行退出一个系统然后再进入另一个系统，并且常常发现数据不一致……因此，利用协同(OA)软件的信息门户与信息协同优势，通过集成的理念去构建岗位工作门户、以及面向对象的OA理念开始盛行。同时，协同(OA)软件在移动信息化方面也是首当其冲。并且随着移动互联带来的组织模式的变革，社交化的管理需求开始显露出来，协同(OA)软件也正在进入自我变革期。作为协同(OA)软件在当下最为根本的几个特征我们可以概括如下：基于组织的、基于web的、基于流程的、基于知识的、基于集成的。协同(OA)软件的绝大多数用户关注点聚焦在：流程管理、知识管理、信息门户、沟通协助和移动应用等方面，现阶段的协同(OA)软件主流厂商也是多以工作流程为核心；但，历史永远是动态发展，目前还有不少用户在关注：行政办公、邮件管理和内部信息发布；一些走的前面的用户已经十分关注信息门户和业务数据集成、移动办公；更进一步地，部分的厂商正在开始打造以岗位工作目标为核心的信息门户、流程引擎、内容管理、目标任务、移动引擎、融合集成、报表引擎，引导协同(OA)软件向社交化转型。

协同(OA)软件绝大多数的用户关心流程管理、知识管理、信息门户和沟通协作等应用，主流协同(OA)软件也已经形成以工作流程为核心的共识，并且以此为基础，开始引入目标绩效、计划任务、管理报表等与组织管理、日常办公息息相关的内容和方案实现；而与此同时，以个人工作门户为标志的平台集成理念，以及依托移动互联的移动办公平台开始引领协同OA未来的发展趋势，如移动APP、微信企业号集成应用等等。所以现代的协同办公服务应该是这样：移动应用为第一媒介；组织战略目标落地为第一要义；流程引擎、内容引擎、集成引擎、报表引擎、建模引擎、消息引擎六大引擎必备；各种应用云端采集部署：降低部署成本，加强智力交换；平台化是大趋势：个性化与标准化的有效统一；更加凸显人为核心的社交化自我管理模式；柔性组织、扩大组织边界：从组织应用为核心到个人发展为核心的模式---任何人都有自己终生唯一的网络办公空间ID，随时可以和任何企业组织进行对接。

随着人类电脑科技的进步，协同(OA)软件一定是向着更加还原协同工作自然原貌的方向发展：如果我想请假、报销、出差，只要对着手机说一下就可以了；如果我想获得某些经验，手机上马上有人跳出来告诉我做好这个事情的关键知识点—我可以不用打扰到当前的他，历史的他会来告诉我如何做；如果我想召开会议，大家马上可以通过移动设备进行会面交流，系统自动帮助梳理出会议纪要并自动提醒每一个人要做的事情；任何人所做的事情都可以通过移动设备自动记录下来，并很快反馈到每一个人每一天的工作目标和工作成效中，并告知与整个公司目标的一致性和推进成效；手持设备能够自动记录我对事件的想法和判断，并针对这些想法和判断提供一些信息依据和人员支持。

上述的内容不仅是企业协同的现在的需求，也是未来发展的方向。本章接下来就会以这些点切入，来简述工作圈这样一个基于移动云服务的企业协同软件是怎么去满足这些需求的。

## 系统用户角色分析

企业内的角色划分，一般分为：决策者、管理者、员工。他们在企业中起到一个什么样的作用，决定了他们在这个系统之中的能力大小。

决策者：一般指的就是企业的领导。他们在企业中仅是很少的一部份人，但是确是企业的头脑。他们主要的工作就是决定企业发展的方向，决定企业的主要目的与制定企业的管理方式。在工作圈中，他们定位应该是权限最大的一部份用户。做为信息企业内信息的发布者和企业人员的管理者。因为还需要对企业的发展方向做决策，所以他们还是企业运营数据，分析的查看者。

管理者：一般是企业员工的管理人员。主要工作是管理员工的日常工作，转达决策者的一些信息。在工作圈中，对他们的定位就是企业内部各功能模块的管理员。比如：签到查看人，圈子管理员等角色。

员工：就是企业中具本工作的执行者了。他们代表企业的绝大多数人。主要工作就是完成管理者传达的工作，汇报工作结果。在工作圈中，对他们的定位就是数据的主要生产者。上、下班打卡，每日、每周、每月工作总结、发布工作进度、问题或是解决方案等。

对于大型企业，国有企业来说，他们有完整的OA办公系统，同时也有很多的专用服务商为他们提供构件基于他们业务场景下的协同办公系统。也就是说他们使用的系统大多数都是针对他们量身定制的。他们不是工作圈的主要目标。工作圈的主要用户群是一些没有雄厚财力，而且管理流程和主要业务相对简单的中小企业或者是小微企业。相对简单的管理和业务使得他们的需求更加容易抽象，这样工作圈的适用面就会更加广。同时企业人数相对大型企业要少的多，员工的转型相对要更加的容易，这样有助于软件的推广。

## 系统功能需求分析

### 用户账户管理与企业管理

工作圈的账户系统是畅捷通CIA云平台的子系统，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。之所以采用公司的CIA云平台用户账户，其一是为了消除数据弧岛，其二是为了工作圈将来的平台化做准备。畅捷通的在企业办公自动化上有很多的优秀的软件和云服务，需要一个协同云服务来帮助它们打通这些数据，而打通数据的第一步就是需要用户模型统一。用户的账户在CIA云平台都有一个唯一的ID值叫userId，可以在不同的系统，不同的业务功能中以userId识别用户信息。账户管理模块的主要功能是为工作圈各个终端提供单点登录、权限校验用户/企业信息查询和变更等功能。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。

工作圈的企业也是沿用了畅捷通CIA云平台的模型，企业的数据同用户一样通过消息总线与CIA保持同步。工作圈的企业是以orgId这个值做为企业在系统中的唯一标识，同时企业的名称也不能重复。这样可以便于用户在注册进入工作圈的时候，可以直接加入企业。

工作圈企业管理的最主要功能就是企业通讯录。微信就是以手机通讯录为入口，占据了移动互联网时代的入口。这个案例在企业协同市场依然有效，可以说做好了企业的通讯录管理，企业协同能力就完成了一大半。提交审批单的时候，需要企业通讯录才能知道提交给哪个领导；提交报销单，请假单，工作报告也是类似的；开会时发起电话会议需要企业通讯录才能知道参与人的联系方式；更不用提在工作中找人的便利性。最重要的，是挖掘企业与企业之前的关系这种跨企业之间的协同关系。而这个目前没有能够提供这样能力的服务。

企业通讯录主要的功能：展现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。

### 圈子与帖子

圈子在工作圈中的定位是一组关系相对松散的集合，这个集合的创建是以一个工作上共同的目标为关系来创建的，比如一个项目，一个目标或者是一个团队等。没有严格的限制只有一家公司的同事才可以进入。在圈子中，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。

帖子承载了圈子中最关键的信息，主要是以文字来构成，辅以图片文件等，同时可以@其他圈子成员关注。贴子可以发表500字的文章，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。

圈子中人人平等的特性其实会为企业带来一种新的管理模型：扁平化管理。在这样的管理模式中领导的指令下达不再是层层下传，而可直达所有的员工。这样的做法有利也有弊。针对小微企业人数少，若在层层传达信息容易缺失；对于大型企业，如果有很多的员工在里面容易造成信息淹没难以分辨有价值的信息。所以圈子内的人员数不是无限增长的。根据公司在小微企业调研的结果，暂时定为一个圈子人数上限不会超过1000人。这个设定，在将来工作圈的运行过程中还会根据用户的活跃程度和需求调整。

### 评论

评论的功能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。

评论的功能看似与帖子相同，然而在工作圈中却把它作为一个单独的服务提供出来是因为其价值需要。在本章的引言中提到了现代的企业协同的一大特点就是社交化，评论则是社交的一个主要的切入点，完成了信息的聚合与传播。在开放式的圈子模块中引入评论可以促进圈子成员之间的交流，同样在我们的工作日报或是报销单之类的其它电子办公的场景中引入评论同样也可以引入评论的机制让更多的用户参与进来。

可以看到在企业协同办公的很多场景中都需要评论功能，单独为了每个模块去写评论功能是不合适的。它不仅低效，而且分散了评论的数据，不便于数据的聚合与挖掘。而提供一个独立的服务让所有的模块复用则会提高开发的效率，促进数据的聚合。

### 赞

赞与评论一样，是社交式应用的一个主要的切入点。操作简便而且更加容易吸引用户。同样的，用户的点赞数据也有很多可以挖掘的地方。所以这里也作为一个单独的服务模块提供服务。

### 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。可以说工作功能这个模块就是工作圈办公OA的入口。工作提供的功能有：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。等等。

工作功能的入口，重点在于把控企业协同办公应用的入口。将来的规划可以把企业通讯录的服务能力以API的形式开放，提供其他厂商以内部轻应用的形式为工作圈开发一些企业协同办公的功能。上面提到的电话会议与文件柜就是试点应用。积累了足够的用户之后，更进一步的，可以以这个入口与企业通讯录为基础来打造畅捷通公司的企业协同办公生态圈。

### 即时通信（IM）

即时通信服务，作为工作圈企业用户的一个实时通道。可以进行用户之间点对点聊天，群聊、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。

即时通信，首先会让人们想到腾讯的两款标志性产品，QQ与微信。腾讯围绕这两款拳头软件打造了一个生态帝国，为我们揭示了现代移动互联网改变人们生活的原因。在企业协同的领域又如何，不妨来设想一个场景：员工出差或者是休假在外，老板突然有一个任务要下达员工却不能及时返回，此时这个任务如何完成？之前的做法，老板电话通知员工，然后在邮件告知细节，员工完成之后再邮件回老板，然而员工如果处在一个没有网络的环境怎么办？这个场景，每一个问号都是过去办公OA上的一个短板：异地办公，无网络环境等。所以，移动互联网的即时通信对现代的企业协同尤其重要。可以说是现代企业协同的基石。

## 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

本章通过分析企业协同办公发展的历史以及现代企业协同云的特点及诉求，来为我们的企业协同云服务——工作圈定义需求。在本章，我们揭示了企业协同行业的现状与移动互联网的发展，指出现代化的企业协同需要结合这两者才能满足现代企业对协同管理的需求。同时分析了即时通信，社交化办公的场景给出了如何将两者结合的方向。给出了工作圈的模块需求，并且分析了这些模块需求在企业协同中起到的作用与价值，还提供了一些参考的数值。在3.3章节中给出了现代云服务的一些非功能性质的要求。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。

# 第四章 企业协同办公云服务的总体设计与实现

结合前两章的关键技术介绍与需求分析，本章我们将开始进行工作圈的云服务总体设计阶段。这一阶段的主要目标是在满足功能性需求与非功能性需求的基础上完成云服务的整体架构设计。

## 工作圈的设计目标

之前我们提到了工作圈的针对的用户市场是中小型企业与小微企业。那么在国内这个市场有多大呢？根据工伤总局2014年发布的《全国小型微型企业发展情况报告(摘要)》：截至2013年年底，全国各类企业总数为1527.84万户。其中，小型微型企业1169.87万户，占到企业总数的76.57%。将4436.29万户个体工商户纳入统计后，小型微型企业所占比重达到94.15%。可以看出工作圈面对的是一个有千万级企业，上亿用户的这么一个市场。所以，对于工作圈服务的规划已经不能再像以前的企业门户那样只考虑几千人的用户去设计实现了，应该去像移动互联网服务那样去设计。在第二章的时候，提到了分布式的概念和技术，就是近几年解决移动互联网并发问题的方案，现在同样也适用于工作圈。结合分布式几个显著特性开始设计工作圈整体的架构。

## Rest server设计

工作圈的用户终端有三种：Android、iOS和PC Web。这三个端可以简单的区分成为PC端和移动终端。这样区分是因为这两种端对应的场景和对后台的数据交互要求都是不一样的。PC端的特点：运行在浏览器中（B/S结构），网络环境信号较强，数据传输几乎没有成本，终端的尺寸较大可以展示的内容丰富，终端更换的频率较高；移动端特点：原生的APP程序运行在移动智能操作系统中（C/S结构），容易处在弱网络环境中，数据传输有成本，终端的尺寸有限，终端不易更换。从对比来看，移动端需要轻量级的增量更新接口，这样意味着移动端的数据在从服务端下行之后需要自行缓存在终端。这样可以避免在弱网环境下程序请求不到数据，同时节省的数据重复传输的成本。PC端应用因为是B/S结构不便于本地缓存数据，所以在查询数据的时候接口应该设计成为全量分页性的接口，根据页码来下行查询数据。然而站在具体的功能业务上，同一个功能的业务逻辑却又是相同的，只是对于不同类型的终端提供的数据交互接口不一样。后台的服务就不能只是一个server的设计了，首先应该把业务逻辑封装成为一层，我们先简称为server层。然后在server层之上封装一个rest层，给不同的终端提供不同的服务。



### Nginx负载均衡

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。同样，在面对大规模用户高并发的访问时一台服务器的能力是有限的，我们需要多台服务器来分担压力。站在最前面的负载服务器就尤其重要，这里选择了前面介绍过的Nginx服务器来作为HTTP请求的代理网关与负载。



### 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。

## 服务模块化的设计

模块划分是分布式服务设计重要的一环。低成本，横向扩展是分布式服务的一个主要特点。从需求分析，可以预见的是工作圈服务的用户帐户、即时通信这两个相关的服务访问量会比其他的功能要大一些。为了能做到性能可以弹性扩展，就是在有个别服务的请求量比较大的时候对，为这个服务动态增加计算资源，需要对整个服务的功能合理的划分模块服务。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，减少开发的代码量。每个模块专注与自己的功能实现，如果有对于其他模块的依赖，则通过标准的RPC API调用，这符合软件工程中的高内聚，低耦合的要求。

### 功能模块划分

根据第三章的功能需求分析，可以进行功能模块划分。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。在进行公共模块划分设计需要考虑的要点：

1.高内聚，无耦合。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。

我们对工作圈的功能模块定义如下：

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

gongzuoquan-imp，即使通信服务

### 公共模块划分

在工作圈的规划里，数据持久存储使用MongoDB服务，数据内存缓存使用Redis服务。所有的业务模块，基本都需要操作数据库与缓存。将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一使用这些公共服务完成对数据持久与缓存的操作。这样不仅可以减少在数据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。

工作圈的服务层承担了业务运算的主要工作，这一层服务对系统运算资源敏感，所以这一层的主要考虑就是动态横向扩展能力。这里先不讨论如何去实现这个横向扩展能力，但是需要指出为了实现这个横向扩展能力，需要为每一个模块服务约定一个服务的配置规范，符合这个约定的服务就能实现动态横向扩展能力。将这个公共模块定义为：gongzuoquan-configcenter。

在开发服务的过程中，工程师需要很多的重复的工具类型的功能：比如时间格式化，数据加密，数据计算MD5摘要、字符串判空等等的功能，为了每一个不同的模块服务去编写一份工具是不合适的。将所有被重复使用的工具封装到gongzuoquan-util中，工程师在开发的时候直接复用即可。最终公共模块划分如下：

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

公共模块不同于功能模块，它们不应该成为一个独立的服务，而是成为其它服务所依赖的低层插件。

### 核心模块服务

工作圈的核心服务是对分布式服务与数据存储提供抽象化服务的模块。在分布式系统之中，数据的流转处理是依赖于一个全局唯一的主键ID，需要一个gongzuoquan-idcenter的服务来为其它的服务生成一个全局唯一的ID。

后面会提到，为了提高存储服务的性能，工作圈会为数据库进行分片，同时在每一个数据库分片中会对其中的数据集合也进行分片。对数据的分片结果就是查询困难，需要一个数据分片的索引中心来维护所有的数据分片。

工作圈的核心服务定义如下

gongzuoquan-idcenter，提供全局主键生成服务。

gongzuoquan-idlist，提供数据主键索引中心。

## 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（2.1.3 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。

### 分布式服务远程调用（RPC）

服务层的模块规划完成了，接着要解决Rest层对服务层的业务服务调用。在第二章介绍的Apahce Thrift就是一个优秀的RPC框架。Thrift使用自己的IDL语言来定义远程服务的接口，然后使用Thrift工具生成基于定义的服务端接口以及可以调用服务的客户端。

以用户帐户服务gongzuoquan-account举例，Thrift会为我们生成一个接口定义程序gongzuoquan-account-core。gongzuoquan-account-server在这个包的基础上实现它定义的服务接口；gongzuoquan-account-client在这个包的基础上封装的客户端的调用方法。gongzuoquan-account-rest通过引用client实现了访问server的目的，结构图如下。

gongzuoquan-account



### 横向可扩展性（Scale Out）

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？ 前面完成了REST与SERVER层的剥离与远程调用，在已知server服务的IP前提下就可以进行调用。在有多台服务实例情况下，简单的办法就是维护一个IP数组，然后随机的从数组中选出一个IP来进行调用。

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）。

Zookeeper是这个自动化中心实现的最佳选择。服务以一个约定的路径：/gongzuoquan/account/cluster/在Zookeeper服务中注册一个节点，然后把服务自身的IP地址写入到这个节点下，如下图：



### 不允许单点失效（No Single Point Failure）

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。（如下图4.4.3.1）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。

4.4.3.1 心跳策略



在实际的开发中，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。

## 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



## 本章小节

# 第五章 核心模块服务的设计与实现

# 第六章 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）