第二章 关键技术介绍

2.1 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。

2.2 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。

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 从设计模式角度来看，是一个基于观察者模式设计的分布式服务管理框架，它负责存储和管理大家都关心的数据，然后接受观察者的注册，一旦这些数据的状态发生变化，Zookeeper 就将负责通知已经在 Zookeeper 上注册的那些观察者做出相应的反应，从而实现集群中类似 Master/Slave 管理模式，关于 Zookeeper 的详细架构等内部细节可以阅读 Zookeeper 的源码

下面详细介绍这些典型的应用场景，也就是 Zookeeper 到底能帮我们解决那些问题？下面将给出答案。

统一命名服务（Name Service）

分布式应用中，通常需要有一套完整的命名规则，既能够产生唯一的名称又便于人识别和记住，通常情况下用树形的名称结构是一个理想的选择，树形的名称结构是一个有层次的目录结构，既对人友好又不会重复。说到这里你可能想到了 JNDI，没错 Zookeeper 的 Name Service 与 JNDI 能够完成的功能是差不多的，它们都是将有层次的目录结构关联到一定资源上，但是 Zookeeper 的 Name Service 更加是广泛意义上的关联，也许你并不需要将名称关联到特定资源上，你可能只需要一个不会重复名称，就像数据库中产生一个唯一的数字主键一样。

Name Service 已经是 Zookeeper 内置的功能，你只要调用 Zookeeper 的 API 就能实现。如调用 create 接口就可以很容易创建一个目录节点。

配置管理（Configuration Management）

配置的管理在分布式应用环境中很常见，例如同一个应用系统需要多台 PC Server 运行，但是它们运行的应用系统的某些配置项是相同的，如果要修改这些相同的配置项，那么就必须同时修改每台运行这个应用系统的 PC Server，这样非常麻烦而且容易出错。

像这样的配置信息完全可以交给 Zookeeper 来管理，将配置信息保存在 Zookeeper 的某个目录节点中，然后将所有需要修改的应用机器监控配置信息的状态，一旦配置信息发生变化，每台应用机器就会收到 Zookeeper 的通知，然后从 Zookeeper 获取新的配置信息应用到系统中。

2.3 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， 列如：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.

2.4 NoSql数据库 MongoDB

Mongodb无模式优点

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。

第三章 分布式系统介绍

3.1 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。

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。

3.2 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：[1][2]

• 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。

3.3 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。

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。

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 分布式系统的设计

需求分析

系统需求分析

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。

工作圈的主打功能详述：

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。

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：

公共服务：

gongzuoquan-mongo，提供数据库接入操作的服务。

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，提供一个公共的工具类方法

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。