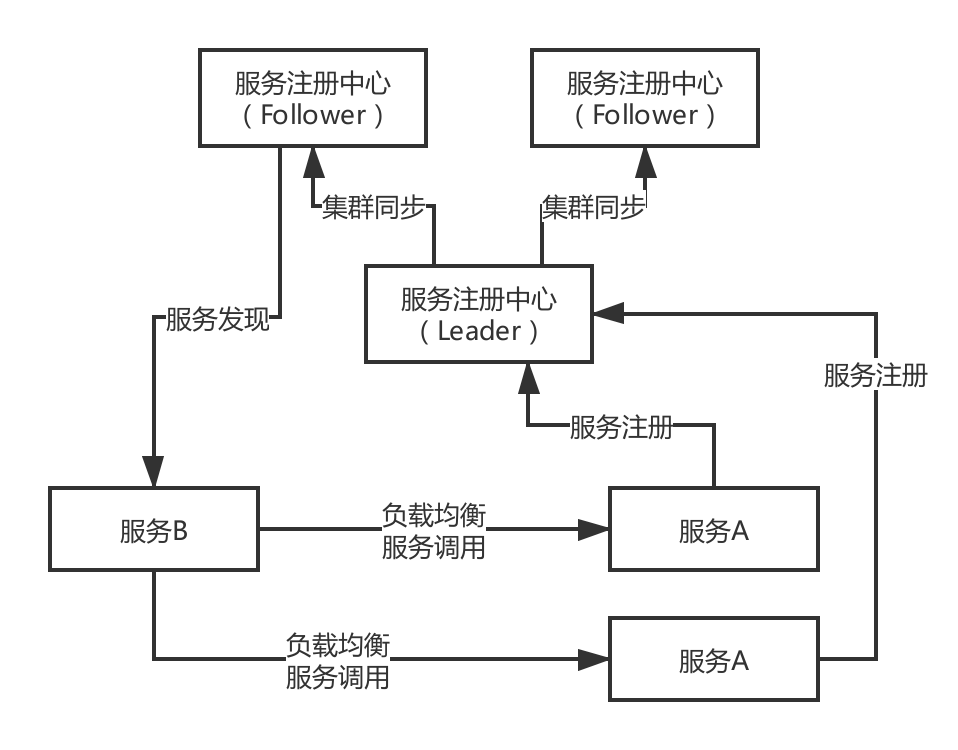
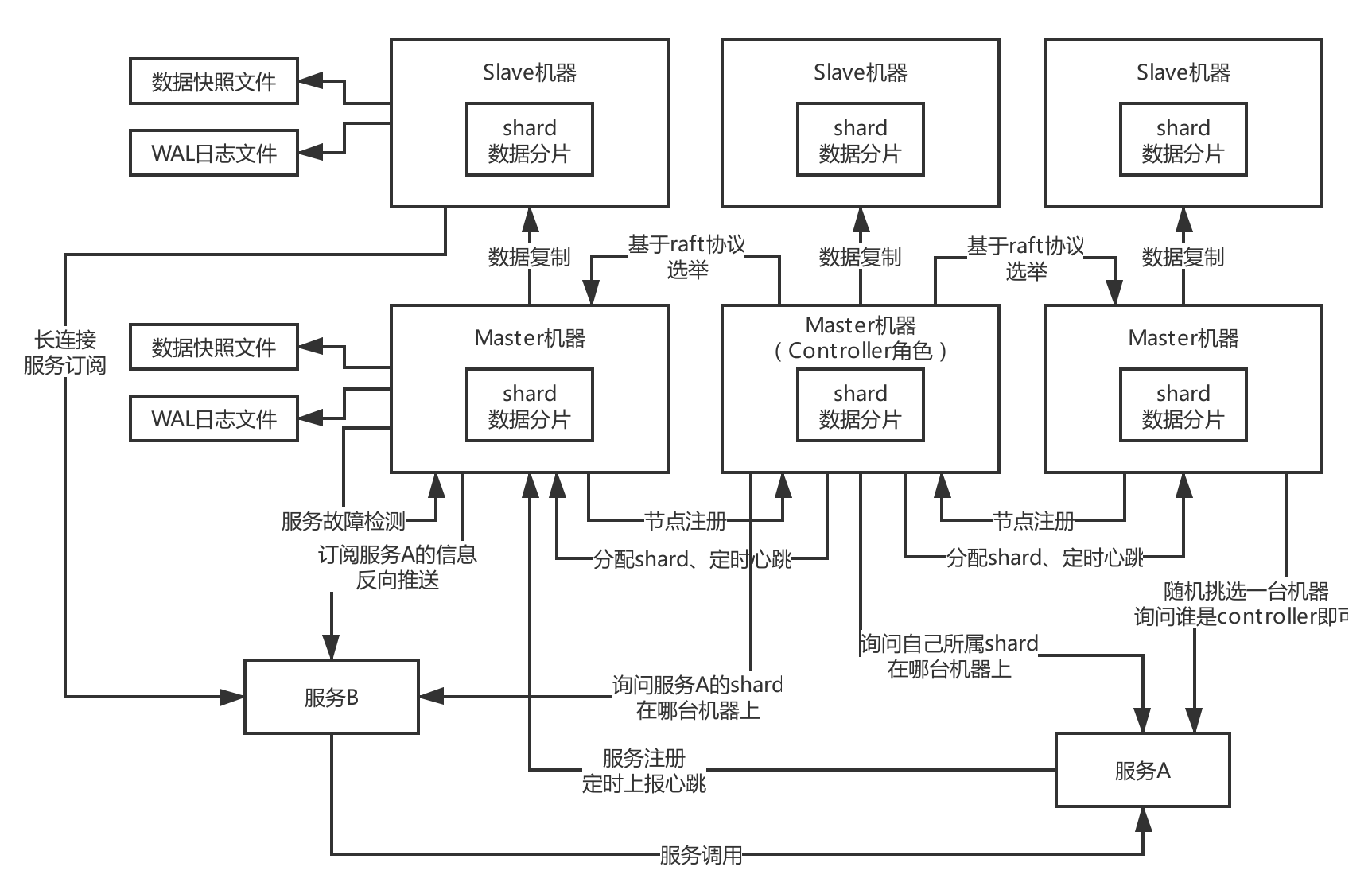
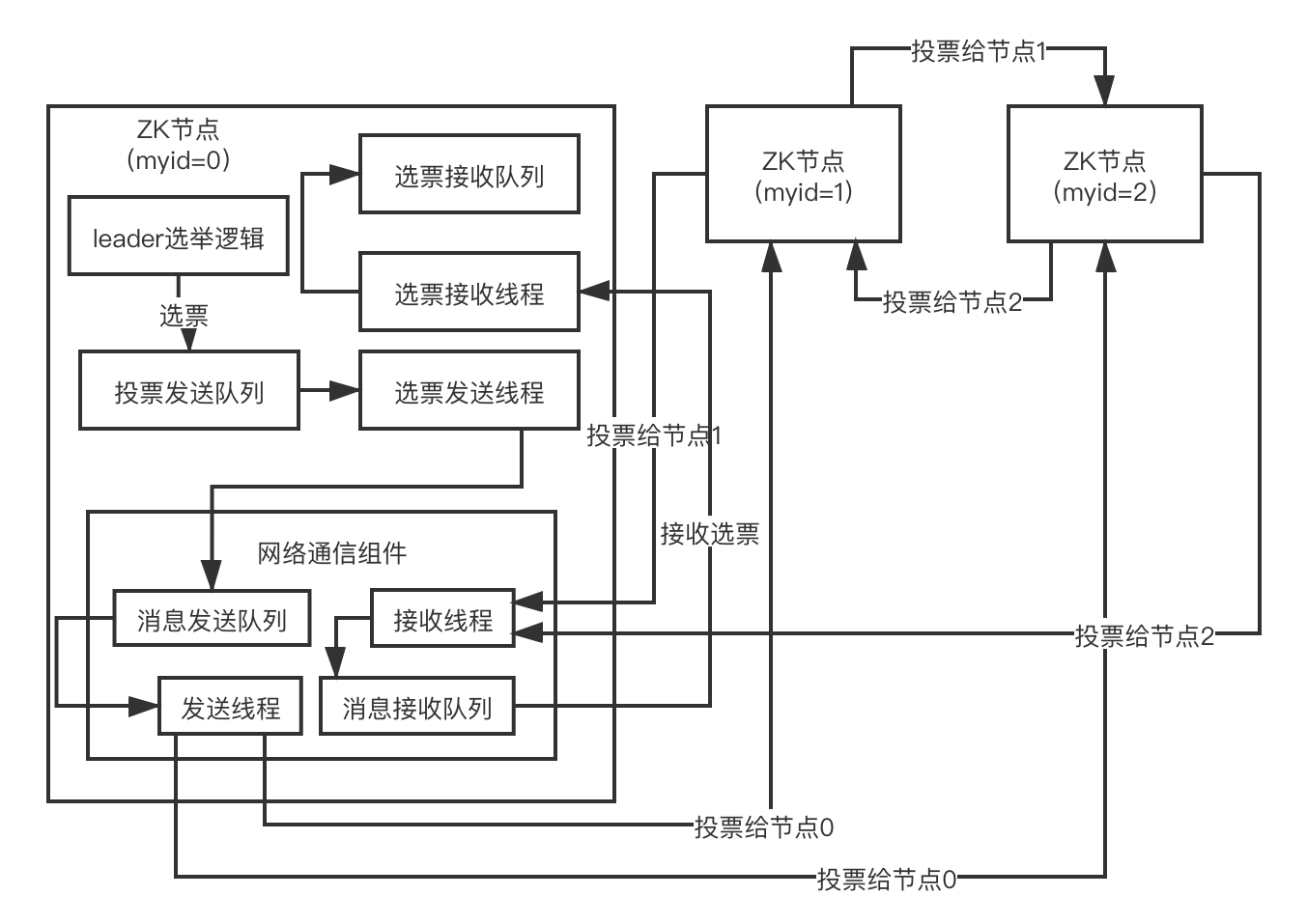
# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/016~035资料/035\_本周小作业：自己根据个人兴趣阅读Nacos的源码细节/01\_服务注册中心架构.png**



# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/036~057资料/057\_架构总结：分布式架构、高时效性、大规模服务、高可用、可伸缩、自平衡/02\_自研服务注册中心的架构设计.png**



# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/188~198资料/198\_ZK源码中已经选举出leader，其他节点加入是如何处理的？/03\_ZK源码中的leader选举细节.png**



# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/157~166资料/162\_实现主动建立连接成功之后的状态维护逻辑（续）/~$biji .docx**

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/306~315资料/308\_仿照ZK开发基于NIO的客户端（3）/笔记.docx**

《308\_仿照ZK开发基于NIO的客户端（3）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/306~315资料/307\_仿照ZK开发基于NIO的客户端（2）/笔记.docx**

《307\_仿照ZK开发基于NIO的客户端（2）》

先要站在功能使用层面讲解spring，把spring各种高阶功能以及在各种开源项目里的实践都讲清楚，spring有哪些高大上、帅气、炸裂的功能和注解，spring cloud开源项目的源码，很多spring东西，帅气的注解；spring源码，AOP、IOC和高阶功能、高阶注解的源码级实现；spring boot高阶功能和设计思想，源码剖析；spring cloud标准相关的一些源码，同时关注一下netflix、alibaba的一些组件是如何集成到spring cloud里去的；做一个集成，到时候可以把我们的服务注册中心再去进行一个集成

在pom.xml里引入一个按照spring boot和spring cloud标准实现的一个客户端依赖，在spring boot的配置文件里去配置好客户端需要的一些东西，比如说三个controller地址，自动装配跑起来，基于spring cloud标准去执行服务注册和心跳的一些动作

服务注册中心的server是独立部署和运行的

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/306~315资料/311\_仿照ZK开发基于NIO的客户端（6）/笔记.docx**

《311\_仿照ZK开发基于NIO的客户端（6）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/306~315资料/310\_仿照ZK开发基于NIO的客户端（5）/笔记.docx**

《310\_仿照ZK开发基于NIO的客户端（5）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/306~315资料/306\_仿照ZK开发基于NIO的客户端（1）/笔记.docx**

《306\_仿照ZK开发基于NIO的客户端（1）》

你应该配置几个controller候选地址，一般来说无论你有多少台master，controller候选节点给个3台就足够了，自动进行选举，有什么问题吗？随机在里面挑选一个controller候选节点（controller），跟他建立一个NIO长连接

就可以去实现后续的服务注册中心的功能了，服务注册，获取槽位分配情况，把自己的服务路由到一个slot槽位去，接着找到那个槽位所在的master节点，建立长连接，服务注册请求就可以直接过去了，心跳，心跳检查

服务发现，其他服务根据要调用的服务名称路由到一个slot槽位里去，获取到那个服务的信息所在的master，建立行连接，第一次可以拉取一下那个服务的实例列表，加一个针对那个服务的监听机制，master监听那个服务的变化，有变化反向通知给你

项目会暂停一段时间，spring源码，spring高阶功能和注解使用，全面剖析spring，参考一下spring cloud制定的一个标准，把我们的服务注册中心，类似于eureka、consul、nacos一样，集成到spring cloud

spring cloud netflix、spring cloud alibaba、spring cloud shishan

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/306~315资料/315\_仿照ZK开发基于NIO的客户端（10）/笔记.docx**

《315\_仿照ZK开发基于NIO的客户端（10）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/306~315资料/312\_仿照ZK开发基于NIO的客户端（7）/笔记.docx**

《312\_仿照ZK开发基于NIO的客户端（7）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/306~315资料/314\_仿照ZK开发基于NIO的客户端（9）/笔记.docx**

《314\_仿照ZK开发基于NIO的客户端（9）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/306~315资料/313\_仿照ZK开发基于NIO的客户端（8）/笔记.docx**

《313\_仿照ZK开发基于NIO的客户端（8）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/306~315资料/309\_仿照ZK开发基于NIO的客户端（4）/笔记.docx**

《309\_仿照ZK开发基于NIO的客户端（4）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/016~035资料/030\_服务端是如何通过定时检查心跳自动下线服务的？/笔记.docx**

《030\_服务端是如何通过定时检查心跳自动下线服务的？》

raft协议进行同步，还是仅仅停留在leader上，注册中心是如何检查心跳下线服务的，leader检查心跳，follower也检查心跳，授人以渔，nacos比较早的源码版本，0.2.x，0.几.x，比较早的一个版本

官方还宣布不是生产版本，还是开发和测试版本

nacos早期版本，raft协议，raft协议是怎么回事，给大家点了一下，基于raft协议的集群机制之下，服务注册、服务发现、心跳上报、故障检测，是如何去做的，应该是有一个思路的，希望大家顺着我说的这个源码思路

比如说如果是leader和follower同时检测心跳，都自动下线服务，leader如果自动下线服务之后，是否需要raft协议同步给follower呢？因为follower自己会检测心跳，也已经下线过服务了

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/016~035资料/033\_服务注册中心的要求：发现快、可伸缩、高稳定、高可用/笔记.docx**

《033\_服务注册中心的要求：发现快、可伸缩、高稳定、高可用》

非常关键的一个特性，就是服务发现和故障感知的时效性，几秒钟，那还是可以的，启动一个服务之后，几秒过后其他人就会发现你了，几十秒，甚至几分钟那就非常的不靠谱了，你启动服务之后，人家往往是要报错一段时间

故障感知，如果你一个服务挂掉了，几秒钟过后人家就感知到了，也是没有问题的；很长时间范围内，人家都没有感知到，比较笨的做法，他发现请求一台机器一直是报错，反复的重试，在几十秒甚至几分钟内，都一直砸重试

重试几次就把这台机器拉入黑名单里去，过一段时间再次重试

注册中心无论是并发压力，还是服务规模大了以后的内存压力，可能会导致机器压力比较大，单机压力比较大，此时务必要支持是可伸缩，直接线性扩容机器，就可以线性的增强你的注册中心的抗并发的能力，抗更大数据量的能力

一定是比较稳定，不能动不动就出现一些奇怪的问题，eureka，肯定是不太稳定，出现对他的服务器进行重启，故障，一些数据会丢失，服务已经启动一段时间了，结果死活就是发现不了

任何时候你的注册中心任何一台机器如果宕机的话，是绝对不会导致数据丢失的，高可用，一定是有其他机器可以接管他的工作的

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/016~035资料/020\_简单聊聊Raft协议：居然与ZooKeeper的ZAB协议如此相似？/笔记.docx**

《020\_简单聊聊Raft协议：居然与ZooKeeper的ZAB协议如此相似？》

ZooKeeper的ZAB协议要解决的也是一个集群部署的时候，角色的划分，选举leader，leader->follower一致性同步机制，leader崩溃恢复机制

如果说你的服务注册中心采取的是Raft协议，你要做的事情类似于zk

zk的ZAB协议本身就是跟Raft协议是类似的，只不过里面有一些细节有差别的

Peers架构中，要划分主从角色，随时可以进行热切换；RocketMQ消息中间件，他要实现Broker的高可用，采用的Raft协议，Dledger技术，基于Raft协议在你的每一组broker都是一主量从，选举出来一个Leader

raft协议，作为一个经典的分布式一致性的协议，在分布式系统里使用的还是很广泛的

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/016~035资料/021\_简单聊聊Raft协议：Nacos的集群机制比Eureka进步的地方/笔记.docx**

《021\_简单聊聊Raft协议：Nacos的集群机制比Eureka进步的地方》

eureka集群机制，纯粹的peers，各个机器角色没有划分，没有分别的，无差别的可以执行所有的操作，然后异步化的同步数据到其他机器上去。每台机器都可以写入请求，数据同步的时候，三层队列，层层打包，异步延迟同步

数据一致性很差，最终一致性的方案，可能要过几十秒，甚至上一分钟之后才能实现集群之间的数据同步

比如你注册一个服务到机器A上去，要过1分钟才能同步这个注册到机器B和机器C上去，此时你的服务连接到机器B去尝试发现注册的服务，是发现不了

崩溃恢复没有办法去做，一批服务注册到了机器A上去，还没来得及同步给机器B和机器C，机器A宕机了，此时在很长时间范围内，其他服务是无法发现注册过的服务的，那个注册过的服务发现机器A宕机了

这个服务会将他后续的心跳发送机器B，机器B发现自己这里没有他注册过的痕迹，结果他就直接发送心跳过来了，此时要求他重新在机器B这里进行注册，此时其他服务才可以在机器B上发现这个服务

机器B过几十秒后才会同步给机器C

只能适用于小公司，并发量很低，业务服务拆分不多，不是太复杂，要求不是太高的一些系统，用eureka是没有问题的

但凡是有点规模的公司，在使用eureka的时候都会发现很大的一个痛点，就是服务的注册和发现的时效性很低，还不如用zk当服务注册中心呢。nacos集群架构采用了raft协议之后，同步的时效性比eureka提高很多

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/016~035资料/035\_本周小作业：自己根据个人兴趣阅读Nacos的源码细节/笔记.docx**

《035\_本周小作业：自己根据个人兴趣阅读Nacos的源码细节》

从nacos的GitHub上下载下来对应的源码，导入intellij idea中，模仿去画图，真正帮助你去掌握看源码的能力

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/016~035资料/023\_客户端进行服务注册的时候是如何全部转发给Leader的？/笔记.docx**

《023\_客户端进行服务注册的时候是如何全部转发给Leader的？》

在Raft协议里面，一旦你敲定了角色之后，各个机器的角色会很明确，明确了角色之后，谁是leader谁是follower大家都清楚。服务会连接到机器上去执行写入的请求，服务注册、心跳上报

只有leader可以执行写入操作，follower而言也可以接收写入的请求，但是需要转发给leader去执行，zookeeper他的实现原理跟这个也是类似的

（1）服务是如何建立跟注册中心的机器之间的网络连接和通信架构的

（2）进行注册的时候，服务仅仅是发送请求给一台机器，如何在集群中挑选出来一台机器建立连接的呢？（对于zk而言，是随机挑选的策略）

（3）对那台机器是如何发送出去服务注册的请求和心跳上报的请求呢？

（4）对于follower角色的机器而言，收到写入请求之后，是如何转发给leader机器的呢？

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/016~035资料/019\_简单聊聊Raft协议：Leader的崩溃恢复机制/笔记.docx**

《019\_简单聊聊Raft协议：Leader的崩溃恢复机制》

有服务，注册的请求还没发送过去，leader就崩溃了，此时进入了在剩下的两台follower中重新选举leader过程中，还没有诞生一个新的leader，服务注册的请求是失败的，无法发送，你需要不停的重试

如果说你的注册请求发送到了一个leader上去，此时uncommitted状态，没有发送uncommitted请求到其他follower上去，leader崩溃了，此时这个注册请求是失败的，需要服务不停的重试，选举出来新的Leader

如果说你的注册请求发送到了leader上去，也发送了uncommitted请求到其他follower上去，部分follower收到了请求，但是还没达到半数follower返回ack给Leader，leader就挂了，会导致服务也认为服务注册强求失败了

如果说已经超过了半数的uncommitted请求的ack给leader了，服务注册请求已经成功了，此时leader崩溃了，选举一个新的leader，会去接收其他follower的ack，如果超过半数follower有ack，直接commit操作

跟zookeeper几乎是类似的

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/016~035资料/031\_服务是如何优雅下线以及基于Raft协议同步给Follower的？/笔记.docx**

《029\_服务是如何优雅下线以及基于Raft协议同步给Follower的？》

都会提供接口让你优雅的下线，让你发送一个请求过去，把自己从注册表里删除掉，然后再进行机器的重启

eureka源码，服务注册、心跳上报、服务下线，都是发送请求到一台机器，然后那台机器要同步给其他的机器的

如果大家之前真的跟我看了很多的源码之后，大数据架构那边讲解的zookeeper的源码这块，我觉得真的能给大家看nacos源码有很多的启发，startup.sh脚本，看看里面的jvm参数，默认的启动main类

然后一步一步去看，启动的时候代码，如何初始化各种各样的组件的，如何初始化网络连接和通信的架构的，如何启动一些后台线程的，如何发起leader选举

服务注册、服务发现、心跳上报、服务下线，都是直接从nacos的http接口源码作为入口，去跟踪每个请求对应的业务逻辑其实就可以了

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/016~035资料/025\_服务注册是如何通过Raft协议同步给Follower的？/笔记.docx**

《025\_服务注册是如何通过Raft协议同步给Follower的？》

你只要从leader的服务注册的http接口进去就可以看到对应的后续的源码了，raft协议，先把数据以uncommitted状态同步给所有的follower，后续如何接收各个follower的ack，如果你的机器之间在这里进行转发采取的是什么协议？

（1）leader同步数据给follower的时候，采取的是什么网络协议？是TCP+自定义协议？还是HTTP协议，干脆就走一个http接口

（2）在采用同步网络协议的基础之上，是如何接收follower的ack的？

（3）如果收取到了过半follower的ack，是如何处理的？是否直接判定服务注册请求成功了，返回结果给服务了？

（4）如果超过了过半follower返回ack之后，接下来是如何将服务注册标记为committed，如何写入自己的内存核心数据结构里去的？对于follower是如何发送commit请求过去的，要求别人也进行commit呢？

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/016~035资料/026\_客户端是如何进行服务发现的？/笔记.docx**

《026\_客户端是如何进行服务发现的？》

参考一下，以前的eureka的服务发现的多级缓存的架构，主要是用于避免掉对同一个内存数据结构频繁的读写出现大量的锁互斥，多线程频繁读写，避免涉及到大量的加锁和等待的过程

ReadOnlyMap + ReadWriteMap

刚开始的时候，服务发现，会先找ReadOnlyMap，如果没有就找ReadWriteMap，如果没有就从核心注册表加载出来放入ReadWriteMap中以及ReadOnlyMap中，后续服务会每隔一段时间去加载注册表

默认情况，就是从ReadOnlyMap里去加载

如果注册表出现了变更，此时会直接invalid删除ReadWriteMap中所有的数据，一个后台线程会每隔一段时间检查ReadOnlyMap和ReadWriteMap两者数据是否一致，如果不一致，就是发现ReadWriteMap里空了，此时会清空ReadOnlyMap

下次别人服务过来加载注册表，看到ReadOnlyMap是空的，直接就会从核心注册表里加载数据，填充两个map就可以了

nacos，提醒一点，在官网也是可以基于http接口进行服务的发现的，直接找到源码里controller里面的服务发现的http接口，顺着进去看，他是如何从服务注册表里发现对应的服务的数据的

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/016~035资料/029\_心跳是如何通过Raft协议同步给所有Follower的？/笔记.docx**

《028\_心跳是如何通过Raft协议同步给所有Follower的？》

心跳上报本身也是属于写的请求，nacos里看一下，心跳上报的时候是否采取的是raft协议同步给所有的follower，是同步了呢？还是不同步呢？如果同步是如何做的？如何通过raft协议去进行同步的？如果没有同步，你想一下，是否会有什么问题存在？

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/016~035资料/034\_自研服务注册中心：服务注册、服务治理、调用链路、配置中心/笔记.docx**

《034\_自研服务注册中心：服务注册、服务治理、调用链路、配置中心》

我对他的一个定位是：服务治理平台

服务注册与发现划归到服务治理的一个功能而已，比如你要部署大量的服务，服务的注册与发现、故障自动感知，你对大量服务的一个自动化的一个管理功能之一

可视化的管理大量的服务，对大量的服务进行治理和管理：监控到服务，每个服务的QPS，每个服务的可用性，每个服务的性能延迟，服务的管理，不同环境的服务的隔离，流量的管理，不同的机器的不同的权重

包括灰度发布，也可以基于这个来做，金丝雀发布，蓝绿部署

调用链路的监控，各个服务之间调用链路，链路分析，整个链路而言他的TPS是多少，性能是多少，各个环节的性能是多少

配置中心，可以对服务在治理平台上设置一些配置，高可用管理，自研熔断限流框架整合起来使用，每个服务实例状态，自动降级，手动降级开关

完全可以对标BAT内部使用的类似的服务治理平台

RPC框架去做，Dubbo源码，Dubbo和Spring Cloud二次封装，我们做一个自己的RPC框架，他二次封装了Dubbo和Spring Coud，对里面去做大量的扩展，让框架支持大量的企业级的功能

跟我们的服务注册中心整合起来

限流熔断框架，整合到我们的RPC框架里去，可以通过我们服务治理平台去管理服务的可用性状态，限流降级，服务之间的资源隔离

0.x版本，成立开源社区，在里面不停的加入各种各样的功能，一直到1.x版本，在这个过程中，就会让架构班里部分同学在自己公司里可以尝试使用，他们自己作为commitor随时有能力修复bug，增加新的功能特性

大规模推广，推成热门的开源项目，RPC框架+服务治理平台，企业级的功能，都具备，很受中小型企业的欢迎的

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/016~035资料/018\_简单聊聊Raft协议：2PC与过半写机制/笔记.docx**

《018\_简单聊聊Raft协议：2PC与过半写机制》

服务注册

心跳：每个服务定期发送心跳给leader，如果发现某个服务超过一定时间没有发送心跳，自动认为他宕机了

Raft协议他的规定，2PC+过半写机制

刚开始leader受到一个注册请求，uncommitted，他把这个uncommitted请求发送给各个follower，作为第一个阶段，各个follower收到uncommitted请求之后，返回ack给leader，如果leader收到了超过半数的follower的ack

此时leader就把这个请求标记为committed，同时发送committed请求给所有的follower，让他们也对消息进行committed

跟zk的ZAB协议是一致的

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/016~035资料/032\_基于Raft协议的注册中心的本质：容量受限于单机/笔记.docx**

《032\_基于Raft协议的注册中心的本质：容量受限于单机》

建议大家可以仿照eureka源码分析的思路，自己画一个图出来，具备这样的一个能力，并发，集合，网络，IO，Netty，底层的东西都讲完了，Raft协议这里也讲给你了，nacos源码你完全可以自己去分析和画图了

授人以渔，真的

所有的写请求都得基于leader去完成，如果你有大量的服务实例，上万服务实例，心跳间隔调的很低，100ms，1s一次，拉取服务注册表的间隔调节的很低的话，此时你的leader一定压力是会很大的

单机都要上万，甚至是几万并发的请求都是有可能的

服务注册表的数据量特别大，因为包含了大量的服务实例，几万，几十万

单台机器的内存容量是有限的，我升级leader的机器配置，24核48G，让他去一台机器抗超高并发，抗超大数据量

leader承载所有的写，leader和follower每台机器保留的都是完整的数据，所有注册表数据都在一台机器上，一台leader机器抗所有的写，从架构的角度而言，单机瓶颈，这样也是有问题的

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/016~035资料/027\_客户端是如何定期更新注册表的缓存的？/笔记.docx**

《027\_客户端是如何定期更新注册表的缓存的？》

eureka源码的时候，服务而言，他必须是每隔一段时间就去注册中心拉取一下最新的服务注册表的，nacos里面，服务的客户端而言，是如何做到定期拉取注册表的，更新服务本地的注册表缓存的

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/016~035资料/017\_简单聊聊Raft协议：如何实现Leader选举？/笔记.docx**

《017\_简单聊聊Raft协议：如何实现Leader选举？》

讲一下他的leader选举的算法

启动了之后，一般来说，3台机器，每个人都会投票给自己，选举自己当选为leader，他对自己的投票会发给其他的机器

机器01：投票自己，机器02，机器03

机器02：投票自己，机器01，机器03

机器03：投票自己，机器01，机器02

选不出来leader，3台机器，最起码需要2条机器，大多数人都认可你当leader，你才可以当选为leader

Raft协议，如果一轮投票，发现大家没有选举出来一个leader，此时如何呢？大家都走一个随机时间的等待，timeout时间过后，再次发起第二轮选举。机器01选择休眠等待3秒钟，机器02选择休眠等待1.5秒钟，机器03选择休眠等待4秒钟

第二轮选举的时候，机器02先苏醒过来，发现进入了第二轮投票，他发现此时没有人发送选票给他，他就还是选举自己当做leader，发送给了机器01和机器03

机器02：投票给自己，机器02，机器02

机器01：机器02，投票给机器02，机器02

机器03：机器02，机器02，投票给机器02

大家发现选票都投完了，发现超过半数的人（全票）都投给了机器02，此时机器02当选为leader，Raft协议本质就是通过随机休眠时间保证说一定会在某一轮中投票出来一个人当选为leader

大数据架构里的zk源码和原理

zk一般来说都是3台机器启动，一般都是说先进行leader选举，参考了Raft协议的实现，他其实是基于自己改良后的ZAB协议去进行leader选举，数据同步以及崩溃恢复

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/016~035资料/022\_集群启动的时候是如何基于Raft协议进行Leader选举的？/笔记.docx**

《022\_集群启动的时候是如何基于Raft协议进行Leader选举的？》

讲一些nacos源码实现的思路，我大概是他0.几x版本的时候，当时看过一遍，1.1.x版本，核心的一些东西应该是没有什么问题的，授人以渔的东西，讲一些我们不看源码，我们觉得如果他是基于raft协议做集群的

他的整个源码实现的思路应该是什么样子的

抛砖引玉的效果，让大家根据我讲解的思路去尝试自己去分析一下nacos源码

如果说他是基于raft协议去做的，跟我现在正在大数据架构那边讲解zookeeper源码的，ZAB协议，包括他的leader选举，类似的

画一些图，毕竟来说，eureka的源码剖析过了，花费很多时间讲解这个就没必要了

集群启动的时候，看一下nacos官网，里面有集群启动的步骤，说白了就是他的部署包放在多台机器上，使用startup.sh的脚本去分别启动，一个集群就启动起来了

（1）集群启动的时候，机器与机器之间的网络连接和网络通信的架构如何去搭建的？对于zookeeper而言，当时讲解的很详细，他在leader选举的时候用的是一个端口，而且是基于一套组件；leader与follower之间是一套网络通信的架构；BIO网络通信

（2）当连接建立之后，是如何发起一个初始化的leader选举的过程的？

（3）在leader选举的过程中，具体是采取什么样的代码细节去做的，其实是nacos在落地raft协议的时候也会引入他自己的一些东西，他判断谁是leader还会引入一个概念，term，是跟每台机器上的数据的新旧程度来定的；zk是基于zxid和myid

（4）如果用raft协议进行leader选举，随机休眠的过程

（5）如何通过网络连接互相之间进行投票，投票如何进行归档，如何判定谁是leader，如果一轮不成功如何开启下一轮选举

（6）如果leader选举成功之后，各个机器如何根据投票的结果确定自己的角色

如果不学习大数据那边的zk源码的课程，仅仅只是依靠java架构这边的课，我觉得其实也没什么问题？eureka就已经讲解的很详细了，nacos是基于raft协议的话，协议的原理基本也出来了，看源码找源码的思路

可以自己尝试分析nacos源码

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/016~035资料/024\_Leader自身的服务注册是如何完成的？/笔记.docx**

《024\_Leader自身的服务注册是如何完成的？》

只要是leader收到了一个服务注册的请求之后，如果到时候你要寻找这块的源码，nacos里面，官网里面是提供了http接口发送请求进行服务注册的，你就可以去里面找一些类似controller之类的组件，http接口的入口

他自己在本地是如何进行处理的呢？

eureka源码，核心的内存数据结构，服务注册表，ConcurrentHashMap来做

（1）对于nacos而言，他的核心服务注册表的数据结构是什么？

（2）当他刚刚接收到服务注册的请求的时候，会直接写入自己的服务注册表吗？（跟zk是一样的，zk的leader收到一条数据的时候不能直接写入内存数据结构的，他需要先执行ZAB协议规定的2PC机制+过半写机制）

（3）如何按照Raft协议尝试进行请求的uncommitted状态的标识，转发给其他follower机器的呢？

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/016~035资料/028\_存活的服务是如何持续性的上报心跳的？/笔记.docx**

《028\_存活的服务是如何持续性的上报心跳的？》

除了集群启动时候的，网络连接和选举的源码，找起来稍微有点点麻烦，服务注册、服务发现，都是可以走他的http接口的，那个源码找起来，直接去找对应的http接口就可以了，选举源码

startup.sh

找到nacos的启动类，从启动类的main方法作为一个入口开始来看就可以了

服务随时可能会重启、宕机、崩溃，每个服务都得定时上报心跳到注册中心去，注册中心就应该定时检查心跳，是否超过30秒没有上报，就自动从注册表里删除他，如果后续他恢复了，继续上报心跳

此时就应该要求他重新注册

后续也可能说是他自己就重新注册过来了

nacos客户端的源码，是运行在服务上，他是如何定期上报心跳的，注册中心是每隔多长时间检查心跳，故障感知的时效性

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/016~035资料/016\_简单聊聊Raft协议：什么是分布式一致性协议？/笔记.docx**

《016\_简单聊聊Raft协议：什么是分布式一致性协议？》

本来是预定的spring cloud alibaba套件里面的nacos服务注册中心的源码给大家快速的过一下，直接给大家把nacos作为一个服务注册中心的架构原理给大家来说一下，其实跟eureka很多东西几乎是类似的

分布式一致性的协议，跟eureka是不太一样的

集群化部署之后，必然涉及到一个问题

（1）三台机器，哪一台机器可以接受写入的请求呢？

（2）接受写入请求的机器，如何将这个请求转发同步给集群里其他的机器呢？

（3）如果有一台机器挂掉了，其他机器如何分担掉他的工作负载呢？

Eureka架构

针对第一个问题，纯粹的一个peer架构，每台机器的地位是对等的，每台机器都可以接收写入请求；针对第二个问题，每台机器接收到写入之后，进行请求打包，异步化延迟一点时间同步给集群里的其他机器；针对第三个问题，任何一台机器宕机，理论上应该是不影响集群运行的，都可以从其他机器里获取注册信息

服务如果感知到某台机器宕机了，下次读写请求直接发送到其他机器就可以了

Nacos，ZooKeeper他们是类似的，并没有采用纯粹的peers架构

类似于Raft协议的分布式一致性的

（1）集群里多台机器，只有一个人可以接收写入请求，他就是leader，集群需要选举出来一个人当做leader

（2）leader收到写入请求之后，需要采取一定的方案同步给集群里其他的机器

（3）如果leader崩溃掉了，其他的机器就会重新选举出来一个leader

划分出来一个主从的角色，leader、follower

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/276~286资料/283\_Controller候选人如何去接收槽位分配数据？（4）/笔记.docx**

《283\_Controller候选人如何去接收槽位分配数据？（4）》

假设选举都已经完成了以后，Controller开始发送这个槽位分配数据出去，作为Controller候选人也应该有自己的行为，就是等待接收槽位分配数据，单纯的仅仅依靠各个代码块自己去take message，完全可能会take混乱

有可能说某个candidate的vote投票的代码块还在运行中，controller已经开始分配槽位数据了，此时可能会导致槽位数据分配到候选人那儿，结果被投票的代码块获取到了，还不能处理

专门让读IO线程，每次读取到一条消息之后，直接从消息里先提取出来一个int，也就是消息的类型，然后根据消息类型，把消息转换为不同的对象，比如Vote之类的，然后接着把这条消息主动推送给某个代码块去进行处理

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/276~286资料/284\_对槽位分配逻辑进行重构（1）/笔记.docx**

《284\_对槽位分配逻辑进行重构（1）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/276~286资料/285\_对槽位分配逻辑进行重构（2）/笔记.docx**

《285\_对槽位分配逻辑进行重构（2）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/276~286资料/276\_回顾一下目前已经实现的代码流程/笔记.docx**

《276\_回顾一下目前已经实现的代码流程》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/276~286资料/278\_Controller将槽位分配数据发送给其他Candidate（2）/笔记.docx**

《278\_Controller将槽位分配数据发送给其他Candidate（2）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/276~286资料/281\_Controller候选人如何去接收槽位分配数据？（2）/笔记.docx**

《281\_Controller候选人如何去接收槽位分配数据？（2）》

假设选举都已经完成了以后，Controller开始发送这个槽位分配数据出去，作为Controller候选人也应该有自己的行为，就是等待接收槽位分配数据，单纯的仅仅依靠各个代码块自己去take message，完全可能会take混乱

有可能说某个candidate的vote投票的代码块还在运行中，controller已经开始分配槽位数据了，此时可能会导致槽位数据分配到候选人那儿，结果被投票的代码块获取到了，还不能处理

专门让读IO线程，每次读取到一条消息之后，直接从消息里先提取出来一个int，也就是消息的类型，然后根据消息类型，把消息转换为不同的对象，比如Vote之类的，然后接着把这条消息主动推送给某个代码块去进行处理

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/276~286资料/286\_对槽位分配逻辑进行重构（3）/笔记.docx**

《286\_对槽位分配逻辑进行重构（3）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/276~286资料/282\_Controller候选人如何去接收槽位分配数据？（3）/笔记.docx**

《282\_Controller候选人如何去接收槽位分配数据？（3）》

假设选举都已经完成了以后，Controller开始发送这个槽位分配数据出去，作为Controller候选人也应该有自己的行为，就是等待接收槽位分配数据，单纯的仅仅依靠各个代码块自己去take message，完全可能会take混乱

有可能说某个candidate的vote投票的代码块还在运行中，controller已经开始分配槽位数据了，此时可能会导致槽位数据分配到候选人那儿，结果被投票的代码块获取到了，还不能处理

专门让读IO线程，每次读取到一条消息之后，直接从消息里先提取出来一个int，也就是消息的类型，然后根据消息类型，把消息转换为不同的对象，比如Vote之类的，然后接着把这条消息主动推送给某个代码块去进行处理

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/276~286资料/280\_Controller候选人如何去接收槽位分配数据？（1）/笔记.docx**

《280\_Controller候选人如何去接收槽位分配数据？（1）》

假设选举都已经完成了以后，Controller开始发送这个槽位分配数据出去，作为Controller候选人也应该有自己的行为，就是等待接收槽位分配数据

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/276~286资料/279\_Controller将槽位分配数据发送给其他Candidate（3）/笔记.docx**

《279\_Controller将槽位分配数据发送给其他Candidate（3）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/276~286资料/277\_Controller将槽位分配数据发送给其他Candidate（1）/笔记.docx**

《277\_Controller将槽位分配数据发送给其他Candidate（1）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/147~156资料/149\_参考一下ZK源码中对监听连接请求的异常处理机制（续）/笔记.docx**

《149\_参考一下ZK源码中对监听连接请求的异常处理机制（续）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/147~156资料/148\_参考一下ZK源码中对监听连接请求的异常处理机制/笔记.docx**

《148\_参考一下ZK源码中对监听连接请求的异常处理机制》

写工业级代码的时候：

第一步，写逻辑，就是把系统的功能逻辑给写通，仅仅是第一步，代码质量是比较低的，绝对达不到工业级标准

第二步，优化代码，引入设计模式，引入一些比较好的工具类，jdk 8以后的lamda表达式，让代码更加简洁，类名、变量名，更加的清晰明了，注释写的好一些，看起来很清爽，简洁明了

第三步，考虑你写的代码运行过程中各种异常情况，设计和开发异常处理机制

第四步，引入日志，有完善的正常逻辑日志，以及异常处理日志

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/147~156资料/153\_参考一下ZK源码中对主动连接的异常处理机制/笔记.docx**

《153\_参考一下ZK源码中对主动连接的异常处理机制》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/147~156资料/151\_监听连接请求的异常无法处理时，控制系统优雅退出/笔记.docx**

《151\_监听连接请求的异常无法处理时，控制系统优雅退出》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/147~156资料/154\_实现主动发起连接的异常重试机制/笔记.docx**

《154\_实现主动发起连接的异常重试机制》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/147~156资料/147\_回顾一下已经开发完毕的Master节点建立连接的代码/笔记.docx**

《147\_回顾一下已经开发完毕的Master节点建立连接的代码》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/147~156资料/155\_主动发起连接重试失败之后，实现定时重试机制/笔记.docx**

《155\_主动发起连接重试失败之后，实现定时重试机制》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/147~156资料/150\_实现监听其他Master连接请求的异常处理机制/笔记.docx**

《150\_实现监听其他Master连接请求的异常处理机制》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/147~156资料/156\_主动发起连接重试失败之后，实现定时重试机制（续）/笔记.docx**

《156\_主动发起连接重试失败之后，实现定时重试机制（续）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/147~156资料/152\_给监听连接请求的处理过程加上完善的日志/笔记.docx**

《152\_给监听连接请求的处理过程加上完善的日志》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/248~257资料/257\_ZK服务端是如何处理客户端发送的正常请求的？/笔记.docx**

《257\_ZK服务端是如何处理客户端发送的正常请求的？》

客户端与服务端之间的TCP连接如何建立，连接如何维护和管理；

底层的网络通信如何执行，序列化和反序列化如何做，拆包和粘包问题如何解决；

如何发送建立session的请求，在服务端开启一个session会话，返回session数据给客户端；

客户端如何接收session数据，session如何进行保活；

在session活跃的情况下，如何进行客户端的请求发送和服务端的响应

对请求进行链条式的处理，比如走一个两阶段提交，在自己本地写一下proposal，再发送proposal给其他的follower，只要超过半数机器都写成了proposal，此时就可以发送commit，自己就会先进行commit，更新内存，返回响应

最终一致性，保证强一致的

客户端一旦启动之后，第一件事情一定是连接任何一台controller candidate，然后去找controller去对自己进行分片，把自己路由到某一台master机器上去，接着就是跟master机器进行

controller是必须要知道所有的客户端分散在哪些机器上的，集群元数据的概念

controller还必须要同步这些集群元数据给其他的controller candidate，万一controller挂了，此时选举其他的candidate作为新的controller

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/248~257资料/254\_ZK服务端是如何把会话信息返回给客户端的？/笔记.docx**

《254\_ZK服务端是如何把会话信息返回给客户端的？》

客户端与服务端之间的TCP连接如何建立，连接如何维护和管理；

底层的网络通信如何执行，序列化和反序列化如何做，拆包和粘包问题如何解决；

如何发送建立session的请求，在服务端开启一个session会话，返回session数据给客户端；

客户端如何接收session数据，session如何进行保活；

在session活跃的情况下，如何进行客户端的请求发送和服务端的响应

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/248~257资料/253\_ZK服务端是如何为客户端创建一个会话的？/笔记.docx**

《253\_ZK服务端是如何为客户端创建一个会话的？》

客户端与服务端之间的TCP连接如何建立，连接如何维护和管理；底层的网络通信如何执行，序列化和反序列化如何做，拆包和粘包问题如何解决；如何发送建立session的请求，在服务端开启一个session会话，返回session数据给客户端；session如何进行保活；在session活跃的情况下，如何进行客户端的请求发送和服务端的响应

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/248~257资料/251\_看一下ZK服务端是如何监听客户端的请求的？/笔记.docx**

《251\_看一下ZK服务端是如何监听客户端的请求的？》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/248~257资料/252\_ZK服务端接收客户端请求的时候如何发生拆包怎么办？/笔记.docx**

《252\_ZK服务端接收客户端请求的时候如何发生拆包怎么办？》

客户端与服务端之间的TCP连接如何建立，连接如何维护和管理；底层的网络通信如何执行，序列化和反序列化如何做，拆包和粘包问题如何解决；发送建立session的请求，在服务端开启一个session会话；session如何进行保活；在session活跃的情况下，如何进行客户端的请求发送和服务端的响应

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/248~257资料/248\_ZK客户端在建立TCP连接后是如何发送初始请求的？/笔记.docx**

《248\_ZK客户端在建立TCP连接后是如何发送初始请求的？》

ZK客户端建立的时候是如何跟ZK服务端建立TCP连接的，ZK服务端是如何监听端口，跟很多ZK客户端建立连接，我们都已经看明白了。ZK客户端一旦在建立了TCP连接之后，就会发送一个初始的请求，就从这个初始的请求开始，看一看，ZK客户端和ZK服务端是如何进行网络通信的

对于ZK来说，网络通信，其实主要就是三块，一块是管理网络连接，一块是针对网络连接进行通信的线程，最后一块就是收发消息的内存队列

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/248~257资料/250\_深入分析ZK客户端发送数据包到服务端的拆包问题/笔记.docx**

《250\_深入分析ZK客户端发送数据包到服务端的拆包问题》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/248~257资料/249\_ZK客户端是如何通过线程把消息发送给服务端的？/笔记.docx**

《249\_ZK客户端是如何通过线程把消息发送给服务端的？》

用Protobuf提前为我们的一些通信用的实体类做好序列化的组件，需要的时候，就直接把对象通过Protobuf序列化成bytebuffer，就可以了

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/248~257资料/255\_ZK客户端如何接收服务端返回的会话数据？/笔记.docx**

《255\_ZK客户端如何接收服务端返回的会话数据？》

客户端与服务端之间的TCP连接如何建立，连接如何维护和管理；

底层的网络通信如何执行，序列化和反序列化如何做，拆包和粘包问题如何解决；

如何发送建立session的请求，在服务端开启一个session会话，返回session数据给客户端；

客户端如何接收session数据，session如何进行保活；

在session活跃的情况下，如何进行客户端的请求发送和服务端的响应

session保活，就是每隔一段时间发送心跳，ping，给服务端，服务端把你的session不停的挪到下一个分桶里去，服务端也会自动过期掉你的在过期分桶里的session，对于你后续的正常的每一个请求，都必须要带上你的sessionid，服务端而言需要去检查你的会话是否过期了

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/248~257资料/256\_建立会话之后，ZK客户端如何发送请求到服务端？/笔记.docx**

《256\_建立会话之后，ZK客户端如何发送请求到服务端？》

客户端与服务端之间的TCP连接如何建立，连接如何维护和管理；

底层的网络通信如何执行，序列化和反序列化如何做，拆包和粘包问题如何解决；

如何发送建立session的请求，在服务端开启一个session会话，返回session数据给客户端；

客户端如何接收session数据，session如何进行保活；

在session活跃的情况下，如何进行客户端的请求发送和服务端的响应

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/188~198资料/195\_ZK源码中对接收到的正常投票是如何处理的？/笔记.docx**

《195\_ZK源码中对接收到的正常投票是如何处理的？》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/188~198资料/191\_ZK源码中的leader选举是如何封装一次投票请求的？/笔记.docx**

《191\_ZK源码中的leader选举组件如何将选票发送出去？》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/188~198资料/198\_ZK源码中已经选举出leader，其他节点加入是如何处理的？/笔记.docx**

《198\_ZK源码中已经选举出leader，其他节点加入是如何处理的？》

弥补了zk课程的一个小遗憾，对leader选举算法进行这么细节的梳理和分析

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/188~198资料/194\_ZK源码中对接收到的投票异常是如何处理的？/笔记.docx**

《194\_ZK源码中对接收到的投票是如何处理的？》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/188~198资料/193\_ZK源码中如何接收别的节点发送给自己的投票？/笔记.docx**

《193\_ZK源码中如何接收别的节点发送给自己的投票？》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/188~198资料/189\_ZK源码中的leader选举方法入口都有哪些逻辑？/笔记.docx**

《189\_ZK源码中的leader选举方法入口都有哪些逻辑？》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/188~198资料/190\_ZK源码中是如何发送自己的第一个leader投票出去的？/笔记.docx**

《190\_ZK源码中是如何发送自己的第一个leader投票出去的？》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/188~198资料/188\_ZK源码中的leader选举逻辑在哪里？/笔记.docx**

《188\_ZK源码中的leader选举逻辑在哪里？》

初步的把master节点的启动、互相建立网络连接、互相进行网络通信，已经初步的都跑通了，下一步的话呢其实是应该会让master节点之间互相可以进行controller选举，我们会基于raft算法去选举出来一个controller

节点也不会太多的，我们就直接让所有的master都参与选举就可以了

但是或许我们会做很多的完善的，如果说一旦我们的集群规模变大了，此时我们可以给配置文件加入一个参数，is\_controller\_candidate=false，如果设置为true的那几台机器，那么可以作为contorller节点的候选节点

master集群里有20台机器，30台机器，此时你可以设置集群里就5台机器会参与contorller的选举

zk集群本身也是会选举leader，follower专门同步数据，做一个数据副本的备份，随时leader挂了，follower跟上，保证zk是高可用的，并不是分布式的，主从架构，但是zk这个技术本身其实不是分布式的

他里面的数据以及并发，会集中在一个leader节点上，每个节点的数据都一样，每个节点都会承担所有的并发写请求

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/188~198资料/196\_ZK源码中对收到的投票是如何进行归票处理的？/笔记.docx**

《196\_ZK源码中对收到的投票是如何进行归票处理的？》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/188~198资料/192\_ZK源码中的leader选举组件如何将选票发送出去？/笔记.docx**

《192\_ZK源码中的leader选举组件如何将选票发送出去？》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/188~198资料/197\_ZK源码中在选举出leader过后会如何处理？/笔记.docx**

《197\_ZK源码中在选举出leader过后会如何处理？》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/082~101资料/090\_在测试环境时的debug日志如何输出到文件里？/笔记.docx**

《090\_在测试环境时的debug日志如何输出到文件里？》

肯定是在Intellij IDEA里运行，可以自己跑起来测一把，也可以运行一些单元测试，集成测试，运行一些测试代码，跑系统跑一遍，这个过程中打印出来的一些debug日志，都可以在控制台里输出

接下来肯定需要在一些测试环境的机器上部署一下，系统在运行的过程中，可以跑一些自动化的测试，去测试系统运行是否正常，这个过程中肯定也会有一些debug日志会输出到一个日志文件里去

像这种文件目录，一般来说，都应该自己去改，设定一个/opt/software/ss-govern/logs

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/082~101资料/100\_写工业级代码的标准素质：处处考虑异常以及完善的日志记录/笔记.docx**

《100\_写工业级代码的标准素质：处处考虑异常以及完善的日志记录》

异常体系：大模块的自定义异常 + 细节操作的小异常，小异常如何捕获和处理，如何包装成大异常抛出去，大异常如何进行日志输出和系统退出

日志体系：debug、输出一些特定的值，观察一下运行；info，系统正常运行的情况；warn和error，warn就是小异常，处理掉，重试，或者搞定了，就不用封装成大异常往外抛了，日志打印一个warn；error，封装成大异常，对大异常进行处理，系统退出了

不同的环境下，本地调试在控制台打印日志，测试环境环境输出到文件，生产环境进行rolling文件输出

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/082~101资料/094\_在服务治理系统中引入log4j的依赖，同时加入日志/笔记.docx**

《094\_在服务治理系统中引入log4j的依赖，同时加入日志》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/082~101资料/084\_大致浏览一下日志框架所使用的配置文件/笔记.docx**

《084\_大致浏览一下日志框架所使用的配置文件》

日志框架，面向slf4j提供的日志API去进行编程，他底层使用的具体的日志框架是log4j，负责具体的日志的打印，日志打印的时候，到底怎么打，写入什么日志文件，文件放在哪里，日志文件的名称是什么，打印日志的格式

配置文件

如果对日志框架没有进行过研究的同学，可能看log4j的配置文件，会显得特别的迷糊，解决的问题有两个，第一个是，后面探索一下日志框架的源码，看看日志框架的配置文件的地址能否自定义

起码是以zookeeper为例子，搞明白日志配置文件里，各个配置项的含义是什么，我们就知道日志写入到什么文件里，放在什么目录里，日志的格式是什么，如果一个日志文件写的太大了，此时怎么办

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/082~101资料/101\_工业级中间件系统的三大基础介绍：异常体系、日志记录、metirc监控/笔记.docx**

《101\_工业级中间件系统的三大基础介绍：异常体系、日志记录、metirc监控》

必须有完善的异常体系，你在系统里必须把各种异常都可以搞出来，进行对应处理，重试，写磁盘，降级，或者默认值，系统退出，小异常 -> 大异常；日志体系，debug级别，就是你在写代码的时候是不加的

一般都是你写好了代码之后，在进行调试和测试的时候，才会加入一些debug，你可以通过观察日志，可以调试系统

在写系统的时候，比较关键的是info、warn和error，info要表明系统运行的情况；warn，小异常，可以直接处理掉，不影响系统的运行，此时你就用warn；error，大异常，系统功能不正常了，要退出了

mtric监控机制，对于任何一个系统，中间件系统，分布式业务系统，一个是日志，一个是监控，四个级别监控，机器资源负载，JVM GC和内存使用的监控，metric监控，比如说每秒处理1000个订单，报警了，订单量异常大

系统运行都必须有一套metric机制，统计metric跟外部的监控系统进行集成，open-falcon，prometheus

异常监控，系统里如果有报错和异常的话，就需要进行监控和报警

都需要分析日志，所以说完善的日志也是必须的

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/082~101资料/099\_对日志框架在各种情况下的日志输出进行测试/笔记.docx**

《099\_对日志框架在各种情况下的日志输出进行测试》

在测试环境下，一般来说，系统不会不停的运行，一般都是部署了之后，跑一些测试，确认一下系统如果运行都正常的话，然后就可以了，看一下日志，手动删除日志，测试环境的日志文件不存在说会特别大的情况

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/082~101资料/093\_如何对日志文件的滚动进行细粒度的控制？/笔记.docx**

《093\_如何对日志文件的滚动进行细粒度的控制？》

部署在自己的很多环境里，比如测试环境，或者生产环境，肯定会不停的运行，不停的输出日志，如果都是输出到一个日志文件里去，必然会导致日志文件越来越大，磁盘空间会不够的，经常自己登录到机器上去，手动删除日志文件

自动去对文件进行rolling，滚动，清理

每个日志文件多大，最多保留多少个日志文件

对于无论是java web的系统的日志，还是说中间件系统的日志，都可以搞定了。对于分布式的电商业务系统之类的东西，一般来说日志都是上传到基于ES的日志中心里去，对业务日志得精心设计一下，你要看系统的运行的情况，快速的基于ES进行检索

log4j日志框架，精通了

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/082~101资料/089\_参考ZooKeeper看如何细粒度指定日志输出格式？/笔记.docx**

《089\_参考ZooKeeper看如何细粒度指定日志输出格式？》

%d{ISO8601} [myid:%X{myid}] - %-5p [%t:%C{1}@%L] - %m%n

时间 [线程:Logger@什么类什么方法第几行] – 日志内容 换行符

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/082~101资料/082\_回顾一下配置文件加载模块的实现流程和思路/笔记.docx**

《082\_回顾一下配置文件加载模块的实现流程和思路》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/082~101资料/086\_log4j的几种日志级别以及我们在项目里如何使用？/笔记.docx**

《086\_log4j的几种日志级别以及我们在项目里如何使用？》

log4j的日志是分成几种级别的，也就是说我们可以按照不同的级别去打印日志

all：所有的级别

trace：就是对系统运行中产生的各种细小的事件，进行追踪

debug：为了调试系统的运行，打印的一些日志

info：就是对系统正常运行过程中打印的一些日志，帮助我们了解系统运行的情况

warn：系统运行过程中，发现了一些异常的情况，但是不至于让系统崩溃，进行警告

error：系统已经产生了一些异常，导致一些功能没法正常的运行了，主动退出系统

fatal：发生了绝对严重的异常，导致系统立马要崩溃

off：关闭日志输出

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/082~101资料/096\_准备好配置文件，同时在Intellij IDEA中指定启动参数/笔记.docx**

《096\_准备好配置文件，同时在Intellij IDEA中指定启动参数》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/082~101资料/088\_开发时本地调试的时候如何将日志输出到控制台？/笔记.docx**

《088\_开发时本地调试的时候如何将日志输出到控制台？》

比如说我们在系统里，根据自己的需求打印了一些日志，debug、info、warn、error，一定要适当的处理异常，打印丰富的日志，帮助自己观察系统的运行，开始自己要在本地运行一下代码，进行一下调试

直接在Intellij IDEA里运行代码来进行调试

希望我们可以把日志打印到控制台里去，怎么才能通过log4j在调试的时候，把日志输出到控制台里去？

log4j自己就内置支持一些appender，ConsoleAppender

CONSOLE appender会收到全部的INFO级别的日志，你希望INFO级别以上的日志哪些可以通过CONSOLE appender打印出来，WARN

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/082~101资料/095\_编写自己的log4j的配置文件，满足不同场景下的日志打印/笔记.docx**

《095\_编写自己的log4j的配置文件，满足不同场景下的日志打印》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/082~101资料/098\_对已经开发好的小范围异常体系进行测试/笔记.docx**

《098\_对已经开发好的小范围异常体系进行测试》

对于一个大的操作，配置解析，大的模块，对一个大的机制，可以设计一个类似ConfigurationException大的异常，大操作，大模块，大机制，如果有异常，都是会被封装到这个异常里去

在一个大操作，大模块，大机制，其实可能会有很多小异常，尽量就是用JDK原生提供的一些异常，或者第三方依赖包的一些异常，对这个小异常可以进行一定的处理，把小异常封装大异常

构成了我们的一个异常体系

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/082~101资料/085\_看一下slf4j的LoggerFactory是如何创建Logger日志组件的？/笔记.docx**

《085\_看一下slf4j的LoggerFactory是如何创建Logger日志组件的？》

slf4j底层会基于跟log4j整合的包，Log4jLoggerFactory，去创建Logger组件，还会根据类名进行缓存

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/082~101资料/092\_对于系统运行时的正常日志和异常日志如何进行区分？/笔记.docx**

《092\_对于系统运行时的正常日志和异常日志如何进行区分？》

在测试的时候，必然是让日志得输出到一个磁盘文件里去，debug日志是不输出的，info以上的日志，应该区分一下

通过一个appender输出info以上的日志到一个文件里去，info级别的日志是很多的，你的系统不停的在运转，对应的日志一直在打印，代表了你的系统运行的一个核心的情况，量是很大的，ss-govern.log

有可能你会出现warn和error级别的日志，一般来说专门用一个appender，输出到一个单独的日志文件里去，ss-govern.error，有时候如果我们发现系统有异常的话，此时就可以直接去看ss-govern.error里的异常日志

一旦你搞明白了log4j日志框架的使用之后，以后无论是什么系统，异常处理体系，电商系统，日志体系，metric监控体系，工业级的代码，代码，命名、规范、面向对象、注释、DDD去进行设计、设计模式，处处要考虑到异常，处处要考虑到日志，处处要考虑到metric监控，对系统后续可以进行监控

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/082~101资料/083\_看一下ZooKeeper源码中使用的是什么日志框架？/笔记.docx**

《083\_看一下ZooKeeper源码中使用的是什么日志框架？》

使用的都是slf4j这个日志框架的类，但凡是做过Java开发的，都会基本的日志打印这块东西，但是很多人其实对日志框架的原理，都不太了解，其实就是拷贝一个日志框架的配置文件，然后就瞎搞一通，日志也能打印

中间件系统，他的技术必然要遵循一个最小依赖的原则，能不依赖别人，尽量就别依赖别人，网络通信，仿照和参考zk，基于BIO + NIO去撸，netty，源码比较复杂，出了问题修改netty的源码

JDK提供的API去进行开发

日志框架，我们必须得依赖他，那么必须要搞明白这个框架底层的源码和机制，否则的话，有的时候打印一些日志都会导致系统出现问题，打印日志本身就是在写磁盘文件，如果日志打印的不好的话，会不会影响你的系统的性能

日志框架，好好的深入的研究一下，看看里面的源码

slf4j + slf4j-log4j + log4j，slf4j似乎看起来是一种通用的日志框架的标准API的组件，定义了一套标准的日志框架的API，你可以基于slf4j的日志API去进行日志的操作，打印一些日志之类的东西

具体负责记录日志到磁盘文件里的，似乎是一个具体的日志框架，log4j，他是负责去实现slf4j定义的一套日志API，他是具体的日志写入磁盘的实现框架

slf4j这个日志API框架，似乎是通过slf4j-log4j这个工程跟log4j日志框架进行的整合

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/082~101资料/097\_先对已经开发好的正常配置解析流程进行测试/笔记.docx**

《097\_先对已经开发好的正常配置解析流程进行测试》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/082~101资料/087\_在代码里打印的日志，哪些可以输出？输出到哪儿去？/笔记.docx**

《087\_在代码里打印的日志，哪些可以输出？输出到哪儿去？》

log4j.rootLogger=INFO, CONSOLE

问题来了，比如说你打印的日志，在代码里通过log4j的Logger组件打印的日志，是输出到哪里去的？是控制台还是文件？格式是什么？

INFO：指的是日志级别，你哪个级别的日志，可以打印出来，INFO，那么debug、info、warn、error，只有INFO级别和以上级别的日志会打印出来，INFO级别下面的debug级别的日志，是不会打印出来的

appender：一个appender代表你配置了一个日志输出的路径，你可以给每个appender起一名字，这个名字就代表了一个日志输出的路径和方式，只有INFO级别以及之上的日志会打印出来，打印到CONSOLE这个名字的appender里面去

按照CONSOLE这个名字的appender的配置去输出对应的日志

在代码里打印日志的API，Logger；按照我们需要的级别来打印不同级别的日志，在不同的场景下可以去打印；对于打印出来的日志，哪些级别的日志可以输出，通过哪个appender配置的路径去输出日志

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/001~015资料/008\_先看一下阿里开源的服务注册中心的介绍/笔记.docx**

《008\_先看一下阿里开源的服务注册中心的介绍》

https://nacos.io/zh-cn/docs/what-is-nacos.html

做一些参考，带着大家看一下目前nacos最新版本的源码，其实我们自己也可以考虑我们自研的服务注册中心要做哪些东西

nacos包含的一些特性、后续他的roadmap要做什么

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/001~015资料/011\_我们要做的工业级服务注册中心的架构有哪些特点？/笔记.docx**

《011\_我们要做的工业级服务注册中心的架构有哪些特点？》

服务注册中心而言，非常关键以及核心的一点，其实是服务发现和故障感知的时效性，应该要做到秒级，故障感知哪怕慢一点，其实还好，本来你有一个服务，3个实例，此时有一个实例挂了

其他人没有感知到，照常会去访问这个挂掉的实例，访问失败，连续重试几次，就可以把他拉入黑名单里去了，后续就访问其他的机器，过了一段时间之后，再尝试对挂掉的黑名单里的服务实例做一个尝试

服务注册和发现的时效性，你现在部署了一个服务，结果其他人也部署了一个服务，在几分钟之内，几十秒之内都没法感知到你的服务，此时会在线上出现大量的访问失败的问题，服务发现最好是快一些

最好是故障感知也做到秒级

如果说有很多服务依赖了你的服务实例，你的服务实例挂掉了，人家都没那么快感知到，此时很多服务实例都会大面积的出现调用失败的问题，秒级让人家感知到，大幅度的提升系统的稳定性，不至于说每次重启服务，都会导致大量的服务调用失败

必然会导致频繁的有服务与注册中心之间的通信，心跳以及注册表的拉取/通知，在大规模的服务实例之下，一定会有大量的通信，高并发的访问服务注册中心，服务注册中心可以抗大规模的服务实例

比如说你的服务实例有很多很多，每个服务实例带了一些配置、元数据之类的一些东西，几万个服务实例，几十万个服务实例，此时最好是做成分布式的架构，也就是说让你的大量的服务注册表、配置、元数据都分散在不同的机器上

因为你有很多的机器，所以你必然可以抗很大规模的高并发的访问，扩容加机器，不停的去抗更高的并发量

对于每台机器都要做master-slave的高可用架构，主节点可以写入数据，备用节点就是同步数据，保证主节点突然挂掉，秒级就可以让备用节点切换成主节点，而且，所有的数据都不能够丢失

高可用、高可靠的一个架构

大规模服务实例、大量数据的分布式存储、抗高并发访问、可伸缩、主从高可靠以及高可用，提供大量的工业级的服务和功能，对外开放的API，插件机制，配置机制，元数据管理机制，服务治理机制

把这个课程里的项目做成一个开源社区，项目代码对外开源，组织咱们的学员学好了课程之后，都是这个项目里的contributor或者是committor，让一些学员如果你公司里有条件可以部署的话，用到大家的公司里去，替代eureka、consul、zk服务注册中心，用的人可以去贡献源码，我们可以自己制定roadmap路线图，0.x版本，基本可用；1.x版本；2.x，大家在自己公司里用的同学就可以贡献bug，hotfix，大家自己用的可以提建议，你就可以来写这个功能的代码，feature

贡献源码的大小，contributor、commitor

后续可以把这个东西通过我的公众号，石杉的架构笔记，狸猫技术窝，群，可以在里面进行推广，多邀请一些公司来使用，确实是可以替代eureka，spring cloud alibaba，很多功能只是规划，其实还没做

dubbo、spring cloud、k8s、service mesh/lstio进行整合，支持docker容器部署

你如果作为一个高知名度的开源项目的contributor还是commitor，那么此时你都能收获很多的东西，无论是技术还是声望

其他的项目，比如自研限流熔断框架、自研高性能网关，可以后续放慢脚步，到明年穿插着慢慢来做，明年三四月份以后的主线，必须切花到业务架构，DDD、中台、大纲里写的一套东西、高并发、高可用

同时穿插阅读中间件研发，穿插自研一些限流熔断框架，自研网关

希望在服务注册中心里加入很多很多的功能，包括服务治理的功能，是很多公司都很缺乏的，每个服务的流量，QPS，可用性，灰度发布，蓝绿部署，服务的回滚，流量权重，多环境的隔离

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/001~015资料/002\_Eureka、ZooKeeper、Consul等服务注册中心有什么问题（下）？/笔记.docx**

《002\_Eureka、ZooKeeper、Consul等服务注册中心有什么问题（下）？》

eureka，peer集群化部署，简单，比较low的一个架构，适应一些中小型公司，几百个服务实例，没有什么问题的，1000个服务实例左右，即使你是调整为让他的服务发现和故障感知的时效性提高的1s以内

对eureka单机的并发压力也不会很大的，一秒最多小2000的并发压力

ZooKeeper，跟Dubbo配合起来使用的

大数据那边的zk的课，现在在剖析他的源码，对zk掌握的会非常的好

服务注册 -> 其实就是在zk的一个znode下面创建一个新的节点

服务下线 -> 在zk里删除自己的节点

服务宕机 -> 之前在zk里创建的是临时节点，节点自动会消失

服务发现和故障感知 -> 其他所有的服务都要监听zk里一个znode的子节点的变化，如果有子节点变化，那么zk就会反向通知所有的其他服务说，有服务变化了，可能是上线、下线、崩溃

其他服务收到zk的反向通知以后，此时就会获取到zk推送过来的最新的所有服务实例的列表，无非就是znode下面所有的子节点

/dubbo/registry/services

/dubbo/registry/services/service-instance-dfd999fdsfs

你自己都可以做

如果你是中小型公司，其实没有什么问题的，服务实例很少，几百个，一千个

zk最多就是给几百个或者上千个服务实例反向推送变化的服务注册表的数据，量也不是很大，100KB，瞬间反向推送给几百个上千个服务实例，瞬间会通过网卡输出最多上百MB的数据出去

zk里的服务注册表的数据是100kb，瞬间要通过自己机器的网卡反向推送数据给1000个服务实例，千兆网卡，理论上每秒可以传输100MB左右的数据，此时100MB左右的数据，网卡打满

zk里的服务注册表的数据可能有1MB，反向推送给1万个服务实例，10GB，频繁的服务上线和下线，会导致zk频繁的打爆自己的网卡，不可能在1秒内就全部推送出去，慢慢推送，网卡被打爆掉

甚至会影响zk集群间的通信，zk可能会不稳定，甚至挂掉；或者是服务注册的时候就会遇到zk的报错，zk会极为的不稳定

大公司里，服务注册中心，一般来说是不能用zk来做的

dubbo+zk，几百个服务实例，都无所谓，中小型公司；如果是中大型公司

consul，重型，consul agent去监控服务的监控程度，跟服务端进行通信

service mesh、k8s，除非是已经有巨头提供的云平台极为极为的成熟，巨头也是在实践和积累中，传统的微服务架构，服务注册中心

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/001~015资料/010\_对工业级服务注册中心的架构进行了解/笔记.docx**

《010\_对工业级服务注册中心的架构进行了解》

分布式、高可用、可伸缩的服务注册中心，提供完善的工业级的功能，尽可能的保证服务注册与发现都是秒级以内，可以承载大规模服务实例，高并发的访问，数据一致性，数据不丢失，高可用，高可靠，可伸缩，随时可以扩容

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/001~015资料/014\_了解一下工业级服务注册中心的压测报告/笔记.docx**

《014\_了解一下工业级服务注册中心的压测报告》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/001~015资料/007\_为什么是从自研服务注册中心开始做起？/笔记.docx**

《007\_为什么是从自研服务注册中心开始做起？》

基础架构的研发课程，推向最高峰

微服务技术栈：分布式技术栈（服务注册中心、RPC框架、熔断限流框架、分布式事务框架、网关系统），配置中心、日志中心、监控中心、调用链路、服务治理、持续集成

如果你要做一个分布式系统，必然要让各个子系统之间可以互相进行通信，RPC框架，负载均衡和服务发现，服务注册中心，熔断限流框架，分布式事务框架，网关系统，把你的分布式的多个系统对外进行统一的暴露

服务注册中心，可以让分布式系统里面的子系统、服务，启动之后可以注册到服务注册中心里面去，故障感知 -> RPC框架（Dubbo源码） -> 对Dubbo进行工业级的二次开发和封装 -> 服务注册中心集成到Dubbo/Spring Cloud -> 熔断限流框架/集成 -> 分布式事务框架/集成 -> 网关系统

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/001~015资料/006\_为什么有了Spring Cloud Alibaba之后我们还要自研？/笔记.docx**

《006\_为什么有了Spring Cloud Alibaba之后我们还要自研？》

读源码，分析源码，通过源码解决问题的能力

自研一个中间件系统，从头到尾做一遍架构设计，写一遍代码，积累的是完全不同的能力，我们会带着大家示范一下如何对中间件进行二次开发，如果真的到一些大厂里去，要你做自研的话，其实这个时候也是可以的

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/001~015资料/004\_未来国内微服务技术领域的发展趋势预测/笔记.docx**

《004\_未来国内微服务技术领域的发展趋势预测》

Spring Cloud：eureka/consul + hystrix + feign/ribbon + 其他的微服务套件，很多公司是在用的，最近2年很多公司都是切换到了这套spring cloud的组件上去

Dubbo：zk + dubbo，开发分布式系统，老的一个情况，2017年以前，好多公司都使用的这套东西做分布式系统架构和开发的

大厂：自研服务注册中心 + 自研RPC框架（基于Dubbo二次开发） + 自研限流熔断狂阿基 + 自研微服务套件，10年以前开始，全部都是自研的，或者基于dubbo做一些二次开发这样子

Spring Cloud Alibaba：慢慢的有一些公司开始使用阿里推动的一些套件，nacos + dubbo + sentinal来作为自己的核心微服务套件

Service Mesh + docker + k8s，等等，诸如此类的新技术：都是大厂在探索和实践，中小公司用的还是比较少的

趋势的判断

spring cloud的这套组件：基于netflix公司的组件为核心的套件，现在在式微，eureka现在停止更新了，feign+ribbon基于http协议做rpc调用的，性能不如dubbo的纯底层的rpc通信的性能要好

未来一两年内，随着阿里的推送，越来越多的公司会逐步的切换为使用spring cloud alibaba，人家推出的时间还不太长，用的公司没那么多，知名度还没那么那么的大，替换成本也很低，都是基于spring cloud规范去做的

都是经历过阿里自身系统的打磨和实践的，工业级，生产级，阿里很重视这个项目，未来开源社区的更新频率和活跃程度也会比较高，hystrix、eureka，社区活跃程度就是很低，不太适合国内了

换一些注解和依赖包而已

长期的趋势来判断的话，service mesh这个技术，下一代的微服务技术，思路，开发一个分布式系统，还有点复杂，考虑配置服务注册中心、RPC框架，service mesh，大幅度的简化你的分布式系统的一个开发

比较复杂，技术难度要比普通的dubbo、nacos之类的要困难很多

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/001~015资料/009\_对工业级服务注册中心的一些核心概念进行了解/笔记.docx**

《009\_对服务注册中心的一些核心概念进行了解》

部署一套服务注册中心，多环境隔离呢？部署一个服务到测试环境里去，此时应该只有测试环境的其他服务可以调用到他，他也只能调用到测试环境的其他服务，不可能说因为都用一个服务注册中心，导致生产环境的服务调用了测试环境部署的服务

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/001~015资料/015\_下周课程预告：一周时间快速摸一下Nacos核心源码/笔记.docx**

《015\_下周课程预告：一周时间快速摸一下Nacos核心源码》

一周的时间，快速把Nacos核心源码的架构摸一下，核心的一些流程，服务的注册和发现，集群机制，主要是这几块东西，核心源码看一下，就可以了，下周也会给大家部署一些作业，可以把一些细节的源码去看一下

自研，自研，自研，如果说我们想要参考别人的源码，尽快进入自研的一个环节，都会有一些作业，希望大家自己去完成，如果你作业不做，后续我反正是直接按照你完成作业的基础来讲课，自研

自己玩儿一下部署，控制台，监控，跟RPC框架的整合，压测报告；下周的作业，一些细节性的源码，你没有自己看一下，如果后续你听我讲自研系统的时候，你跟不上，或者发现我讲的东西你听不懂

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/001~015资料/005\_架构班对微服务技术栈会如何学习？/笔记.docx**

《005\_架构班对微服务技术栈会如何学习？》

Spring Cloud本身的那套组件，源码级别都剖析的很详细了

Spring Cloud Alibaba里面的组件进行源码的剖析，nacos、sentinal、dubbo、seata，干剖析源码，如果你未来自己公司要替换成阿里套件，你是绝对有技术优势的，或者出去找工作面试都是没有问题的

越来越多的一些中大型的公司，哪怕是独角兽公司，都是有自己的基础架构团队，就是对开源的技术进行二次开发和改造，封装。看一个nacos源码，结合我们自己的思路，自己研发一个服务注册中心

sentinal源码，就可以自研一个限流熔断框架

dubbo源码，我们可以对dubbo做一个二次开发和封装，封装一个自己的框架，把自己的服务注册中心和限流熔断框架跟封装你的dubbo框架集成，直接跟spring cloud做一个集成怎么来做

seata源码，自研一个分布式事务框架，集成到我们的微服务套件里去，要支持跟dubbo、spring cloud分别做集成

spring cloud gateway网关源码，自研高性能的网关

2018年头半年，分布式系统架构

2018年底~2020年初，1年半，基础架构

2020年一整年，业务架构，DDD+中台+高并发、高可用、海量数据、高性能、安全性+中间件源码，局部某个系统的架构设计

2021年一整年，对标P8水平的超大规模架构设计，BAT级别的项目，很多系统组成的一个大系统，架构设计

上面这些东西，大致来说会再讲几个月，到明年后，2月份，大致在三四月搞定，然后底层技术、中间件研发、基础架构的能力，给大家就做的很扎实了

明年+后年，还要学习很多的中间件的源码，你完全可以做一个基础架构的架构师，我们自研分布式海量小文件系统，自研过一些分布式中间件系统

明年，还要讲DDD、中台架构、高并发、高可用、海量数据、高性能、安全性这样的一套复杂架构

后年，我们就要讲对标P8的一些超大型复杂架构的设计，BAT里复杂项目，大型架构的设计，就不是局部架构设计

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/001~015资料/012\_给大家留个作业：自己玩儿一下Nacos的一些使用Demo/笔记.docx**

《012\_给大家留个作业：自己玩儿一下Nacos的一些使用Demo》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/001~015资料/003\_Spring Cloud Alibaba是什么？/笔记.docx**

《003\_现在慢慢在火起来的Spring Cloud Alibaba是什么？》

eureka、zookeeper、consul作为服务注册中心，中小公司用哪个都没问题，但凡公司规模稍微大一些，绝对就是不ok了，基础架构团队，负责分布式、微服务、日志、监控、中间件的自研、二次开发、运维部署、开发一些可视化的管理工作台

一般来说都会对服务注册中心做一些手术，动刀子

阿里，美团

https://github.com/alibaba/spring-cloud-alibaba

nacos -> eureka/zk/consul -> 服务注册中心

dubbo -> feign + ribbon -> RPC框架

sentinal -> hystrix -> 限流熔断框架

rocketmq -> kafka/rabbitmq -> 消息中间件

seata -> bytetcc -> 分布式事务

遵从Spring Cloud他作为微服务框架的一个标准，定义了一套规范，包含了一些接口，各个组件如何交互：netflix（eureka + zuul + hystrix） + feign/ribbon，他们都基于规范整合到spring cloud里去

Spring Cloud Alibaba：把他的一些组件遵从Spring Cloud的规范进行了整合和集成，在Spring Cloud规范之下，选用nacos + dubbo + sentinal + seata，去开发分布式系统，微服务架构，调用链路监控、日志中心、监控中心、持续集成、服务治理

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/001~015资料/001\_Eureka、ZooKeeper、Consul等服务注册中心有什么问题（上）？/笔记.docx**

《001\_Eureka、ZooKeeper、Consul等服务注册中心有什么问题？（上）》

自研分布式服务注册中心

Eureka：Spring Cloud原生带的，netflix开源的，阅读过他的源码

集群化部署，你的服务对每台机器都可以发送心跳、拉取注册表、注册，机器之间互相进行数据同步

第一个问题，默认的参数配置下，服务发现的速度是很慢的，一般来说要几十秒或者几分钟才能发现一个新的服务，或者感知到一个服务的下线。可以通过配置他的各种参数，让他的服务发现和故障感知的时效性提高到1s以内

大量的服务会对eureka发起频繁的、高并发的请求，eureka有3台机器，你90个服务实例，30个服务实例对eureka01发起请求，30个服务实例对eureka02发起请求，30个服务实例对eureka03发起请求

Peer集群化部署的架构

eureka01要同步给eureka02和eureka03，90个服务他们每秒比如发起大量的请求，eureka每台机器都要承受所有服务的请求

eureka一台机器最多每秒抗1万并发请求，如果你的服务实例越来越多，中型公司，上百个服务都是很正常的，上千台机器很正常的，上千个服务实例是很正常的，每秒会有心跳，还会有拉取注册表，几千个服务实例

eureka单机就会出现性能瓶颈，超过他单机能承载的一个最大的量了

BAT巨头，中大型公司，部署上万个服务实例，每个服务为了应对高并发都会部署10台，甚至20台机器，几百个服务，但是部署了上万个服务实例，几十万个服务实例

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/001~015资料/013\_了解一下工业级服务注册中心的集群部署、运维管理以及可视化监控/笔记.docx**

《013\_了解一下工业级服务注册中心的集群部署、运维管理以及可视化监控》

初步的了解了一下工业级的服务注册中心，概念、架构、部署、管理、监控以及与RPC框架的整合使用

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/301~305资料/304\_参考ZK开发Server端的NIO服务器代码（4）/笔记.docx**

《304\_参考ZK开发Server端的NIO服务器代码（4）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/301~305资料/305\_参考ZK开发Server端的NIO服务器代码（5）/笔记.docx**

《305\_参考ZK开发Server端的NIO服务器代码（5）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/301~305资料/302\_参考ZK开发Server端的NIO服务器代码（2）/笔记.docx**

《302\_参考ZK开发Server端的NIO服务器代码（2）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/301~305资料/303\_参考ZK开发Server端的NIO服务器代码（3）/笔记.docx**

《303\_参考ZK开发Server端的NIO服务器代码（3）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/301~305资料/301\_参考ZK开发Server端的NIO服务器代码/笔记.docx**

《301\_参考ZK开发Server端的NIO服务器代码》

分片机制已经做好了，controller可以自动划分分片，分配给各个master机器，客户端只要连接到任何一个controller candidate，都可以拿到完整的分片数据，然后他根据一定的路由算法，就直接把自己路由到一个分片去，就可以找到那个分片所在的master机器就可以了，分布式的效果

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/102~115资料/112\_对大家利用2020年1月份赶上进度以及消化内容的期望/笔记.docx**

《112\_对大家利用2020年1月份赶上进度以及消化内容的期望》

1%的学员是能完全已经跟上进度，而且真的可以把2018~2019，1年半的纯技术的课程，真正消化掉

99%的同学，利用一个月的时间，没赶上进度的争取赶赶进度，差不多快要赶上进度的，也别松懈，消化掉，学过的东西，都自己画一些思维导图，各种图，经常的复习和巩固，项目实战的，分布式海量小文件存储系统，跟着视频撸一遍代码，不看视频再撸一遍代码

IM即时通讯系统，非常好的一个项目，业务无关，纯技术的项目，完成我布置的大量的作业，在我讲解的思路之上，把这个系统撸出来

1%跟上进度+消化课程的同学，收获到了远远比其他人多得多的东西。三期的一个学员，他已经可以自己参考那个系统架构以及学习到的底层技术，技术功底，自己独立撸出来一个简易版的Redis缓存系统

跟我学了一年左右，小公司，项目拿不出手，12月份出去找工作，完全用我们的分布式存储系统+IM系统，自己做了大量的细节的完善和作业，扩展，基于我的基础之上，自己搞定了两个非常好的系统，跟自己公司的业务结合，自己做了大量的思考和总结，分布式、高并发、高性能、高可用、可伸缩、稳定性

横扫了多个市值百亿美金级别的大独角兽的架构师的面试，4轮技术面都可以直接过，确定去的是一个小独角兽的架构师offer，离他家很近，10分钟的路

想法完全大错特错，只盯着眼前，目光短浅

技术功底

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/102~115资料/113\_目前除了课程讲解之外，正在为4月份之后的架构课做的准备工作/笔记.docx**

《113\_目前除了现在课程的讲解之外，我们正在为明年4月份之后的架构课做的准备工作》

Java架构，二三四3个月，主线课程 -> 分布式服务治理平台 v0.1版本

大数据架构，二三四3个月，主线课程 -> 大数据面试突击课程

大型的项目实战，1年以及1年多里，底层技术，技术功底

Java架构，支线课程 -> 分布式服务治理平 -> 开源社区

主线课程 -> 开始进行多个大型业务系统的项目实战，高并发、高可用、中台架构、DDD之类的真实复杂的大型项目实战

大数据架构，主线课程 -> 开始逐步进行多个大型项目实战

2020年~2021年，整整还有2年的架构课，Java还是大数据

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/102~115资料/110\_设计实现ControllerCandidate候选对象/笔记.docx**

《110\_设计实现ControllerCandidate候选对象》

如果你是一个master，那么必然代表你是一个controller的候选人，所以需要设计一个controller candidate这样的实体对象，他会负责作为一个controller候选人去跟其他的controller候选人进行连接和通信，发起投票

controller

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/102~115资料/103\_实现从配置文件中读取Master节点列表/笔记.docx**

《103\_实现从配置文件中读取Master节点列表》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/102~115资料/105\_完成master节点列表的正则表达式的校验/笔记.docx**

《105\_完成master节点列表的正则表达式的校验》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/102~115资料/115\_如果几个月内有跳槽准备的，及时找我制定短期学习计划/笔记.docx**

《115\_如果几个月内有跳槽准备的，及时找我制定短期学习计划》

1月份，二三四3个月，4月份有很多大型项目实战，筹划和准备，跟着我往前走，架构师，妥妥的，符合我们的架构师课程的定位，3年+

技术广度，技术深度，项目经验，架构设计，简历改造

（1）技术广度，面试突击系列、狸猫技术窝上我的朋友出的一些专栏

（2）技术深度，架构班里每个专题课程都是技术深度

（3）项目经验，亿级流量，es，把各种技术结合你的项目的情况，进行一个改造和包装

（4）架构设计，思维的提升，零零散散的涵盖在上述各种课程里，高并发、高可用、高性能、可伸缩

（5）简历改造：个人介绍、技术能力、项目经验、工作履历、个人学历，全面改造的漂亮一些

5年左右经验的同学，在一线城市25k~30k的范围是ok的，在二线城市20k~25k的范围是ok的，很多比较努力上进的同学，进入BAT、大厂、知名的大独角兽、知名的小独角兽了；8年~10年+的同学，在一线城市，30k~40k之间的范围，在二线城市，在25k~30k的范围，而且，同理，很多人可以进知名的公司，架构师的offer，40k+

2018年初就开始跟我学的，大部分人都在知名大厂和知名独角兽里，几十个人，老是各种忙，课程学习都没跟上

2年的课都学完，5年+的，就可以直接去面架构师职位了，30k~40k了；8~10年+的，直接去面架构师/高级架构师的职位，40k~50k，50k+的职位一般都是技术管理层的，技术leader，技术总监的了

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/102~115资料/102\_服务治理平台Master节点启动之后首先得干什么？/笔记.docx**

《102\_服务治理平台Master节点启动之后首先得干什么？》

master节点启动之后，必须得去找其他的所有的master节点建立网络连接，接下来就应该是所有的master节点互相发起投票，用类似于raft协议来进行leader选举，选举出来一个controller

其他的非controller的master，就得向controller进行注册，controller就应该知道集群里有多少台master了，包括他自己在内。controller就可以去进行shard数据分片的分配，可以均匀的把shard数据分片分配给其他的master

controller知道每台master机器上有哪些数据分片

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/102~115资料/111\_2020年1月发布课程不会太多的原因解释/笔记.docx**

《111\_2020年1月发布课程不会太多的原因解释》

Java架构最后一周的课，1月最后一周，春节前一周我也会请假回家准备过年了，1月第一周是碰上了元旦，所以也是不发课，1月实际能发课的就两周，主要是把一些服务治理平台最最基本的东西给讲了一下，搭建了一个基本的架子出来

最近两周听我的课发现我的咳嗽的频率增加了，我一直有慢性支气管炎，慢性咽炎，经常性会咳嗽，跟北京的雾霾，讲课真的很难受，嗓子特别痒，忍着，在尽力的去讲课，嗓子也特别的不舒服

开一堆药，最近要少说话

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/102~115资料/109\_实现main方法中的无限等待系统停止的功能（续）/笔记.docx**

《109\_实现main方法中的无限等待系统停止的功能（续）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/102~115资料/108\_实现main方法中的无限等待系统停止的功能/笔记.docx**

《108\_实现main方法中的无限等待系统停止的功能》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/102~115资料/104\_基于正则表达式对master节点列表参数进行校验/笔记.docx**

《104\_基于正则表达式对master节点列表参数进行校验》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/102~115资料/114\_为什么我要邀请朋友在狸猫技术窝推出一些专栏？/笔记.docx**

《114\_为什么我要邀请朋友在狸猫技术窝推出一些专栏？》

跳槽的时候，面试越来越难了，需要一些技术储备和积累，P8~P9级别的朋友，下属，我带出来的，狸猫技术窝，推出一些针对性的，面试比价热门的技术的专栏，从0开始带你成为jvm优化实战高手，救火队队长，广受好评

原子弹大侠，RocketMQ专栏，从0开始带你成为消息中间件实战高手，68块，互相推荐一下，你们互相还有推荐奖励，68 -> 24返现，四五十块

从0开始带你成为MySQL实战高手，大互联网公司的分库分表架构实战案例

spring源码，秒杀系统，专栏，安排我的朋友去出，抽空，可能也会继续在狸猫技术窝独立发售我的面试突击系列，第四季，第五季，扫盲，常见的面试题，不会照顾到一些基础，原理，实战案例

专栏，利用上下班的碎片时间，面试

架构班而言，技术深度是很强，基础的东西，架构班里是不会细致的讲解，jvm -> 剖析里面的源码实现，黑科技；RocketMQ，讲源码，二次开发和改造；MySQL，底层的一些东西，更加的系统和完善的调优的手段，周边性的数据库系统的二次开发和改造，TiDB分布式数据库的东西

Spring Cloud，做一大堆的demo，基础都是一带而过，源码

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/102~115资料/106\_对master节点列表参数的读取进行测试/笔记.docx**

《106\_对master节点列表参数的读取进行测试》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/102~115资料/107\_根据节点角色构建出来Master对象/笔记.docx**

《107\_根据节点角色构建出来Master对象》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/268~275资料/275\_Controller在内存以及磁盘中存储槽位分配（5）/笔记.docx**

《275\_Controller在内存以及磁盘中存储槽位分配（5）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/268~275资料/269\_Controller为所有Master机器进行Slot槽位分配（2）/笔记.docx**

《269\_Controller为所有Master机器进行Slot槽位分配（2）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/268~275资料/273\_Controller在内存以及磁盘中存储槽位分配（3）/笔记.docx**

《273\_Controller在内存以及磁盘中存储槽位分配（3）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/268~275资料/274\_Controller在内存以及磁盘中存储槽位分配（4）/笔记.docx**

《274\_Controller在内存以及磁盘中存储槽位分配（4）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/268~275资料/270\_Controller为所有Master机器进行Slot槽位分配（3）/笔记.docx**

《270\_Controller为所有Master机器进行Slot槽位分配（3）》

5461.3333333333

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/268~275资料/272\_Controller在内存以及磁盘中存储槽位分配（2）/笔记.docx**

《272\_Controller在内存以及磁盘中存储槽位分配（2）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/268~275资料/268\_Controller为所有Master机器进行Slot槽位分配/笔记.docx**

《268\_Controller为所有Master机器进行Slot槽位分配》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/268~275资料/271\_Controller在内存以及磁盘中存储槽位分配/笔记.docx**

《271\_Controller在内存以及磁盘中存储槽位分配》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/116~126资料/122\_重新为本地三个节点编写对应的配置文件/笔记.docx**

《122\_重新为本地三个节点编写对应的配置文件》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/116~126资料/126\_测试一下改造后的配置文件以及校验逻辑是否正常/笔记.docx**

《126\_测试一下改造后的配置文件以及校验逻辑是否正常》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/116~126资料/117\_详细参考ZooKeeper的集群内部节点通信的实现/笔记.docx**

《117\_详细参考ZooKeeper的集群内部节点通信的实现》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/116~126资料/125\_根据改造后的配置文件优化参数校验逻辑/笔记.docx**

《125\_根据改造后的配置文件优化参数校验逻辑》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/116~126资料/120\_初步设计MasterNetworkManager里面的方法逻辑/笔记.docx**

《120\_初步设计MasterNetworkManager里面的方法逻辑》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/116~126资料/119\_为集群内部的网络通信设计NodeNetworkManager组件/笔记.docx**

《119\_为集群内部的网络通信设计NodeNetworkManager组件》

QuorumPeer，代表的是zk集群内部的每一个节点，他认为每一个节点的作用其实都是一样的，所以他们都叫做QuorumPeer，Peer这个概念我们后续是可以采纳的，Master节点，分角色，Controller -> Peers

ControllerCandidate，没有人是Controller，大家都是Controller候选人

都需要一个集群内部的节点通信的网络组件，单独抽出来一个的，ZooKeeper里采取的就是QuorumCnxManager，Connection。NodeNetworkManager，所有的节点，其实都是一个Node，分为不同的类型，Master和Slave，Master之间分为不同的角色，Controller和Peer

所有的node共同组成了一个cluster集群，集群内部有很多的通信，master和slave的通信，master自己内部的controller和peer的通信，都是集群内部的node在进行一个网络通信

针对集群内部的节点之间的通信，NodeNetwrokManager，集群内部的节点之间网络通信的管理组件

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/116~126资料/118\_仔细看看ZooKeeper是如何初始化QuorumCnxManager的？/笔记.docx**

《118\_仔细看看ZooKeeper是如何初始化QuorumCnxManager的？》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/116~126资料/124\_继续优化完善配置文件中的master节点地址/笔记.docx**

《124\_继续优化完善配置文件中的master节点地址》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/116~126资料/116\_简单回顾一下之前写好的Master节点启动入口/笔记.docx**

《116\_简单回顾一下之前写好的Master节点启动入口》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/116~126资料/123\_优化MasterNetworkManager里的方法逻辑/笔记.docx**

《123\_优化MasterNetworkManager里的方法逻辑》

假设认为所有的节点一般来说是按照id从小到大来启动的，先启动id=1的节点，再启动id=2的节点，一般来说运维一个集群都是这么搞的

节点之间在内部进行通信的时候，两种通信，一种是master节点之间的通信，一种slave和master节点之间的通信，会导致很多网络通信的代码都会混杂在一起，还不如说，改动配置文件，给每个master节点再次加入一个新的端口号，专门用于和slave节点进行通信

所有的master地址中，其实应该是加入每个master节点的id，你才知道你要等待谁的连接，你要连接谁

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/116~126资料/121\_MasterNode的id序号的概念引入的作用/笔记.docx**

《121\_MasterNode的id序号的概念引入的作用》

参考一下zk的做法，给每一个zk节点的都是引入了一个id，序号，数字的，自增长的

节点，1；节点，2；节点，3

其实我们就需要在ss-govern.properties配置文件里，加入一些配置参数，每个master节点，配置自己的id，配置自己的ip地址

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/199~208资料/199\_自己设计类似ZAB协议的逻辑清晰的选举算法/笔记.docx**

《199\_自己设计类似ZAB协议的逻辑清晰的选举算法》

ZAB协议，我觉得核心思想是ok的，刚启动就给其他的节点去发送投票，每个人刚开始都是投票给他自己的，也会收到别人的投票，收到别人的投票之后，就会比较，zxid，代表的就是当前自己最近执行过的事务id

比较myid，优先倾向于投票给myid更大的节点

quorum大多数的节点，都投票给其中的一个人作为leader，此时选举就成功了，每个人都会判断出自己的角色

后来有人启动发出投票给别的节点，别人会告诉他，不用选举了，此时leader已经有了，他就直接把自己设置为follower就可以了

多个节点获取到的数据都是一样的，zk，所以他是倾向于优先选择那个数据最新的节点作为leader；多个master分布式的，每个人收到的数据是不一样的，zxid来选举controller；raft，zab，基于zab改进过来

比较节点的id大小，master集群总共部署了20台机器，其中就3台机器作为master候选节点，3台机器启动就会发起选举的投票，他们里面就是很简单，优先会把id更大的节点选举为contorller

controller节点实际上来说是会保留一些集群的元数据的，更新集群的一些元数据，然后把集群元数据的更新按照zab协议/raft协议，同步给其他的contorller候选节点，接下来再次选举的时候，一定会选举集群元数据最新的那个contorller候选节点

xid，事务id，集群元数据更新事务id，xid越大，说明集群元数据是越新的，优先选举这个节点作为controller

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/199~208资料/206\_实现获取所有controller候选节点的逻辑/笔记.docx**

《206\_实现获取所有controller候选节点的逻辑》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/199~208资料/208\_测试本周实现的全部功能是否正常运行/笔记.docx**

《208\_测试本周实现的全部功能是否正常运行》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/199~208资料/200\_引入新的配置项：是否Controller候选节点/笔记.docx**

《200\_引入新的配置项：是否Controller候选节点》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/199~208资料/203\_建立连接的时候互相发送自己是否为controller候选节点/笔记.docx**

《203\_建立连接的时候互相发送自己是否为controller候选节点》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/199~208资料/201\_实现是否为controller候选节点配置项的解析和校验/笔记.docx**

《201\_实现是否为controller候选节点配置项的解析和校验》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/199~208资料/205\_建立连接的时候互相发送自己是否为controller候选节点（续）/笔记.docx**

《205\_建立连接的时候互相发送自己是否为controller候选节点（续）》

节点1~节点10

节点3~节点5，都是controller候选节点

节点1会跟其他的9个节点建立好连接，收到人家的节点数据，封装好

节点3而言，他会跟其他的9个节点建立好连接，节点1~节点2都是他主动去建立连接的，节点4~节点9都是他被动监听别人的连接请求的，也会封装好节点数据，找其他的controller候选节点都有谁

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/199~208资料/202\_要发起投票之前需要做的准备工作分析/笔记.docx**

《202\_实现controller候选节点启动就开启第一轮投票》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/199~208资料/207\_实现向其他controller候选节点开启新一轮投票的逻辑/笔记.docx**

《207\_实现向其他controller候选节点开启新一轮投票的逻辑》

节点1、节点2、节点3，每个人都会发起第一轮投票，投票的轮次，刚开始大家都是1，这个投票的轮次，是必须要带在选票里的，发送给别人的，每个人当前处于第几轮投票，就会去接收第几轮的选票

如果你发现自己处于第一轮投票呢，别人第二轮的投票的选票过来了，此时你必须立马作废你原来的投票和收到的选票，立马进入第二轮投票；此时你处于第二轮投票里，结果人家第一轮的投票过来了，此时你需要废弃掉，不理睬他

每一轮收到其他人的选票过后，做一轮归票，做一些处理，如果说没有选出来controller，此时根据一些既定的逻辑，比如说自己下一轮的投票，就投票给id最大的那个节点，就可以了

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/199~208资料/204\_实现远程Master节点管理组件的逻辑/笔记.docx**

《204\_实现远程Master节点管理组件的逻辑》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/287~294资料/293\_Master节点阻塞等待接收自己负责的槽位范围/笔记.docx**

《293\_Master节点阻塞等待接收自己负责的槽位范围》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/287~294资料/287\_回顾已经写好的整体代码流程/笔记.docx**

《287\_回顾已经写好的整体代码流程》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/287~294资料/294\_Master节点在内存里初始化槽位以及磁盘持久化/笔记.docx**

《294\_Master节点在内存里初始化槽位以及磁盘持久化》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/287~294资料/290\_Controller在自己内存里初始化负责的槽位（2）/笔记.docx**

《290\_Controller在自己内存里初始化负责的槽位（2）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/287~294资料/292\_Master节点接收自己负责的槽位范围/笔记.docx**

《292\_Master节点接收自己负责的槽位范围》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/287~294资料/288\_Controller把槽位范围推送给各个Master节点/笔记.docx**

《288\_Controller把槽位范围推送给各个Master节点》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/287~294资料/289\_Controller在自己内存里初始化负责的槽位/笔记.docx**

《289\_Controller在自己内存里初始化负责的槽位》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/287~294资料/291\_Controller在自己本地磁盘持久化负责的槽位/笔记.docx**

《291\_Controller在自己本地磁盘持久化负责的槽位》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/073~081资料/079\_实现基于properties格式的配置解析以及异常处理/笔记.docx**

《079\_实现基于properties格式的配置解析以及异常处理》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/073~081资料/075\_为什么要参考ZooKeeper源码中的很多实现细节？/笔记.docx**

《075\_参考ZooKeeper源码中的配置文件解析模块》

对于服务治理平台，我们会吸收和包容进来很多世界级顶尖的优秀开源中间件的架构设计思想、核心机制的实现、底层代码的实现，redis、elastcisearch、kafka、zookeeper、hdfs，他们的实现和思想

在我们后面写代码的过程中，会大量的参考zookeeper的源码细节，无论是分布式架构 ，选举算法，网络通信，磁盘读写，数据同步，类似这样的一些，其实我们的这个服务治理平台跟zookeeper之间是有很大的关联性的

最好是在学习这个课程之前，把大数据架构里的zookeeper的源码都去看一下，没有看大数据里的zookeeper源码，也不要紧，我主要是参考一些源码里的片段，底层源码实现，我们完全可以在一些核心机制中

代码的实现一般来说是没有什么问题的

对于配置文件而言，xml、yml、properties，用什么方法都可以，我们就直接去参考一下zookeeper的配置的格式以及配置解析的模块，内存中是如何保存解析出来的各种配置的，都去参考一下

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/073~081资料/078\_初步开始实现服务治理平台的配置管理模块/笔记.docx**

《078\_初步开始实现服务治理平台的配置管理模块》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/073~081资料/074\_实现服务治理平台的核心启动类/笔记.docx**

《074\_实现服务治理平台的核心启动类》

Java + 大数据两套技术是同等重要的，每一个学员一个非常优惠的政策，3年+的课程体系，Java里的很多技术对学大数据很重要，大数据里的很多技术的源码剖析对Java中的一些课程同样的重要

大数据里的技术同样去跟上进度

zookeeper，他们的启动脚本，startup.sh，一般来说都是去通过java的命令，跟上一堆的jvm参数，还有一些其他的变量和参数，最终去执行一个用于启动的类，QuorumPeerMain，还有canal里面

我们先来实现一个启动类

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/073~081资料/080\_对节点角色参数进行解析以及异常校验/笔记.docx**

《080\_对节点角色参数进行解析以及异常校验》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/073~081资料/081\_对节点角色参数进行解析以及异常校验（续）/笔记.docx**

《081\_对节点角色参数进行解析以及异常校验（续）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/073~081资料/076\_研究一下ZooKeeper源码中的配置文件解析模块/笔记.docx**

《076\_研究一下ZooKeeper源码中的配置文件解析模块》

中间件系统，也是基于key-value对进行配置的

1、有专门的负责配置的类

2、基于文件流+Properties去加载配置信息

3、对加载的配置进行一定的处理

4、对配置信息进行校验

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/073~081资料/077\_从ZooKeeper源码中的配置解析模块来看异常如何处理？/笔记.docx**

《077\_从ZooKeeper源码中的配置解析模块来看异常如何处理？》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/258~267资料/267\_分析ZK服务端是如何从磁盘读取数据的（2）/笔记.docx**

《267\_分析ZK服务端是如何从磁盘读取数据的（2）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/258~267资料/262\_为什么更加倾向于采用Hash Slots机制呢？（3）/笔记.docx**

《262\_为什么更加倾向于采用Hash Slots机制呢？（3）》

shard分片这个概念，可以是这样子的，默认每台机器就有一个shard，每台机器就是一个数据分片，controller而言，必须把你这个服务和shard之间的路由关系，在自己的内存里存储一份，而且他还必须把这个路由关系同步给candidate

你可以思考一下，对于controller和candidate而言，他们是仅仅把这个路由关系放在内存里就可以了吗？肯定是不行的，必须得是把你的数据做一个持久化，所以说对于controller和candidate而言，还必须把这个路由关系在磁盘文件里写一份

未来你可以横向扩容，可能你的master会有20台机器，其中只有3台机器是controller candidate，其中只有一台机器是真正的contorller，可能controller是把你的这个服务路由到了某一个shard里去

此时他还得把这个路由关系发送给你的那个shard对应的那台master机器，对于你的master机器也必须要在内存里写一份路由关系，甚至在磁盘文件里也得写一份

就是说你的controller以及每个candidate有点类似于zookeeper一样，必须在内存里维护一个全量的服务->shard的一个路由表，这个路由表是相当的坑爹的，其实你的这份路由表也会对应很大的一份数据量

最为关键的一点，而是在于单点的瓶颈，相当的恶心的

Redis Cluster，hash slots机制

我们可以默认搞16384个slot槽位，均匀的分配在我们的每台机器上，每台机器上都有一部分的slot槽位，一旦我们选举出来 了一个controller，contorller必然会知道集群里有哪些机器，此时就可以均匀的把16384个slot槽位分配到各个机器上去

每台机器上其实就有一定的slot槽位了，对于controller自己而言，其实只要存储一份每台对应的slot槽位范围，在磁盘文件里写一份就可以了，对于每个controller candidate而言，也是同理的，他只要维护每台机器对应的slot槽位范围以及文件里写一份就可以了

controller一旦把槽位分配完了之后，除了canddiate要同步一份，还得给每台机器发送过去他负责的slot槽位的范围，这些工作都是在系统启动的时候就全部搞定了，对于每台机器而言，在磁盘里写一份自己负责的槽位的范围

另外在内存里可以初始化自己负责的槽位的数据结构

首先第一点变化，对于服务而言，在candidate里面随便找一台就可以问一下槽位和机器之间的对应关系，加载缓存在服务自己的本地就可以了，1~5336对应着机器1，5337~10389对应着机器2

hash slots，直接就对自己的服务名称算一个hash值出来，使用hash值对16384个槽位进行取模，把自己的服务对应到某一个槽位里去，接着就直接可以知道那个槽位是在哪台机器上，然后就直接找到自己的槽位所在的机器就可以了

如果说基于hash slots槽位机制来做，这个就非常的方便了，首先是你的controller没必要维护你所有的服务和shard之间的路由关系了，所以你的controller也绝对不会成为单点的瓶颈，其实唯一要维护的仅仅是少量的槽位范围和机器的对应关系

对于你的服务而言，没必要说一定要从controller去加载路由映射关系，你随便找一台candidate也可以拿到你的槽位范围和机器的对应关系，假设你要服务注册了，没必要找controller了

直接自己hash路由到slot，发送请求到slot所在的机器上去，直接注册过去就可以了，服务发现的时候也是同理的，直接根据你要发现的服务名称hash一下路由到某个slot里去，接着发送请求到那个机器去加载数据就可以了

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/258~267资料/261\_为什么更加倾向于采用Hash Slots机制呢？（2）/笔记.docx**

《261\_为什么更加倾向于采用Hash Slots机制呢？（2）》

shard分片这个概念，可以是这样子的，默认每台机器就有一个shard，每台机器就是一个数据分片，controller而言，必须把你这个服务和shard之间的路由关系，在自己的内存里存储一份，而且他还必须把这个路由关系同步给candidate

你可以思考一下，对于controller和candidate而言，他们是仅仅把这个路由关系放在内存里就可以了吗？肯定是不行的，必须得是把你的数据做一个持久化，所以说对于controller和candidate而言，还必须把这个路由关系在磁盘文件里写一份

未来你可以横向扩容，可能你的master会有20台机器，其中只有3台机器是controller candidate，其中只有一台机器是真正的contorller，可能controller是把你的这个服务路由到了某一个shard里去

此时他还得把这个路由关系发送给你的那个shard对应的那台master机器，对于你的master机器也必须要在内存里写一份路由关系，甚至在磁盘文件里也得写一份

就是说你的controller以及每个candidate有点类似于zookeeper一样，必须在内存里维护一个全量的服务->shard的一个路由表，这个路由表是相当的坑爹的，其实你的这份路由表也会对应很大的一份数据量

最为关键的一点，而是在于单点的瓶颈，相当的恶心的

Redis Cluster，hash slots机制

我们可以默认搞16384个slot槽位，均匀的分配在我们的每台机器上，每台机器上都有一部分的slot槽位，一旦我们选举出来 了一个controller，contorller必然会知道集群里有哪些机器，此时就可以均匀的把16384个slot槽位分配到各个机器上去

每台机器上其实就有一定的slot槽位了，对于controller自己而言，其实只要存储一份每台对应的slot槽位范围，在磁盘文件里写一份就可以了，对于每个controller candidate而言，也是同理的，他只要维护每台机器对应的slot槽位范围以及文件里写一份就可以了

controller一旦把槽位分配完了之后，除了canddiate要同步一份，还得给每台机器发送过去他负责的slot槽位的范围，这些工作都是在系统启动的时候就全部搞定了，对于每台机器而言，在磁盘里写一份自己负责的槽位的范围

另外在内存里可以初始化自己负责的槽位的数据结构

首先第一点变化，对于服务而言，在candidate里面随便找一台就可以问一下槽位和机器之间的对应关系，加载缓存在服务自己的本地就可以了，1~5336对应着机器1，5337~10389对应着机器2

hash slots，直接就对自己的服务名称算一个hash值出来，使用hash值对16384个槽位进行取模，把自己的服务对应到某一个槽位里去，接着就直接可以知道那个槽位是在哪台机器上，然后就直接找到自己的槽位所在的机器就可以了

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/258~267资料/264\_开始分析ZK服务端写磁盘文件的源码片段/笔记.docx**

《264\_开始分析ZK服务端写磁盘文件的源码片段》

先写一些数据头，然后基于数据内容算一个checksum校验和出来，写 校验和，接着就是写你的数据内容的长度，最后是写数据内容进去

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/258~267资料/266\_分析ZK服务端是如何从磁盘读取数据的（1）/笔记.docx**

《266\_分析ZK服务端是如何从磁盘读取数据的（1）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/258~267资料/263\_从ZK服务端的拦截器链条切入找写磁盘入口/笔记.docx**

《263\_从ZK服务端的拦截器链条切入找写磁盘入口》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/258~267资料/258\_系统架构目前实现到什么地方了？/笔记.docx**

《258\_系统架构目前实现到什么地方了？》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/258~267资料/259\_到底是用Shard路由机制还是Hash Slots机制？/笔记.docx**

《259\_到底是用Shard路由机制还是Hash Slots机制？》

shard分片这个概念，可以是这样子的，默认每台机器就有一个shard，每台机器就是一个数据分片，controller而言，必须把你这个服务和shard之间的路由关系，在自己的内存里存储一份，而且他还必须把这个路由关系同步给candidate

你可以思考一下，对于controller和candidate而言，他们是仅仅把这个路由关系放在内存里就可以了吗？肯定是不行的，必须得是把你的数据做一个持久化，所以说对于controller和candidate而言，还必须把这个路由关系在磁盘文件里写一份

未来你可以横向扩容，可能你的master会有20台机器，其中只有3台机器是controller candidate，其中只有一台机器是真正的contorller，可能controller是把你的这个服务路由到了某一个shard里去

此时他还得把这个路由关系发送给你的那个shard对应的那台master机器，对于你的master机器也必须要在内存里写一份路由关系，甚至在磁盘文件里也得写一份

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/258~267资料/260\_为什么更加倾向于采用Hash Slots机制呢？/笔记.docx**

《260\_为什么更加倾向于采用Hash Slots机制呢？》

shard分片这个概念，可以是这样子的，默认每台机器就有一个shard，每台机器就是一个数据分片，controller而言，必须把你这个服务和shard之间的路由关系，在自己的内存里存储一份，而且他还必须把这个路由关系同步给candidate

你可以思考一下，对于controller和candidate而言，他们是仅仅把这个路由关系放在内存里就可以了吗？肯定是不行的，必须得是把你的数据做一个持久化，所以说对于controller和candidate而言，还必须把这个路由关系在磁盘文件里写一份

未来你可以横向扩容，可能你的master会有20台机器，其中只有3台机器是controller candidate，其中只有一台机器是真正的contorller，可能controller是把你的这个服务路由到了某一个shard里去

此时他还得把这个路由关系发送给你的那个shard对应的那台master机器，对于你的master机器也必须要在内存里写一份路由关系，甚至在磁盘文件里也得写一份

就是说你的controller以及每个candidate有点类似于zookeeper一样，必须在内存里维护一个全量的服务->shard的一个路由表，这个路由表是相当的坑爹的，其实你的这份路由表也会对应很大的一份数据量

最为关键的一点，而是在于单点的瓶颈，相当的恶心的

Redis Cluster，hash slots机制

我们可以默认搞16384个slot槽位，均匀的分配在我们的每台机器上，每台机器上都有一部分的slot槽位，一旦我们选举出来 了一个controller，contorller必然会知道集群里有哪些机器，此时就可以均匀的把16384个slot槽位分配到各个机器上去

每台机器上其实就有一定的slot槽位了，对于controller自己而言，其实只要存储一份每台对应的slot槽位范围，在磁盘文件里写一份就可以了，对于每个controller candidate而言，也是同理的，他只要维护每台机器对应的slot槽位范围以及文件里写一份就可以了

controller一旦把槽位分配完了之后，除了canddiate要同步一份，还得给每台机器发送过去他负责的slot槽位的范围，这些工作都是在系统启动的时候就全部搞定了，对于每台机器而言，在磁盘里写一份自己负责的槽位的范围

另外在内存里可以初始化自己负责的槽位的数据结构

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/258~267资料/265\_分析写入os cache的数据如何刷入磁盘/笔记.docx**

《265\_分析写入os cache的数据如何刷入磁盘》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/157~166资料/160\_成功建立连接的状态维护的异常情况处理/笔记.docx**

《160\_成功建立连接的状态维护的异常情况处理》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/157~166资料/163\_实现等待跟其他所有节点成功建立连接的逻辑/笔记.docx**

《163\_实现等待跟其他所有节点成功建立连接的逻辑》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/157~166资料/164\_检查伪分布式master集群启动的配置环境和日志环境/笔记.docx**

《164\_检查伪分布式master集群启动的配置环境和日志环境》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/157~166资料/161\_实现主动建立连接成功之后的状态维护逻辑/笔记.docx**

《161\_实现主动建立连接成功之后的状态维护逻辑》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/157~166资料/166\_修复完全部bug之后给大家讲解代码流程/笔记.docx**

《166\_修复完全部bug之后给大家讲解代码流程》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/157~166资料/157\_回顾一下目前为止已经写好的代码逻辑/笔记.docx**

《157\_回顾一下目前为止已经写好的代码逻辑》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/157~166资料/159\_实现连接建立成功之后的状态维护逻辑（续）/笔记.docx**

《159\_实现连接建立成功之后的状态维护逻辑（续）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/157~166资料/158\_实现连接建立成功之后的状态维护逻辑/笔记.docx**

《158\_实现连接建立成功之后的状态维护逻辑》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/157~166资料/162\_实现主动建立连接成功之后的状态维护逻辑（续）/笔记.docx**

《162\_实现主动建立连接成功之后的状态维护逻辑（续）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/157~166资料/165\_启动伪分布式master集群验证其内部连接功能/笔记.docx**

《165\_启动伪分布式master集群验证其内部连接功能》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/295~300资料/295\_已经完成的槽位分配机制的代码流程回顾（1）/笔记.docx**

《295\_已经完成的槽位分配机制的代码流程回顾（1）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/295~300资料/299\_正式开始测试槽位分配机制（1）/笔记.docx**

《299\_正式开始测试槽位分配机制（1）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/295~300资料/297\_已经完成的槽位分配机制的代码流程回顾（3）/笔记.docx**

《297\_已经完成的槽位分配机制的代码流程回顾（3）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/295~300资料/300\_正式开始测试槽位分配机制（2）/笔记.docx**

《300\_正式开始测试槽位分配机制（2）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/295~300资料/298\_为槽位机制的全流程测试准备好配置文件/笔记.docx**

《298\_为槽位机制的全流程测试准备好配置文件》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/295~300资料/296\_已经完成的槽位分配机制的代码流程回顾（2）/笔记.docx**

《296\_已经完成的槽位分配机制的代码流程回顾（2）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/036~057资料/056\_生产级代码的标准：随时存在的网络故障和异常体系/笔记.docx**

《056\_生产级代码的标准：随时存在的网络故障和异常体系》

工业级和生产级是啥意思，工业级的意思是，你的架构设计考虑到了复杂环境里各种需求，和各种情况，架构设计出来，是完全可以面向复杂的业务场景。生产级，你的实际的代码质量，考虑到了各种生产环境里的异常问题、性能问题、磁盘问题、内存问题，代码在生产环境里运行是没有问题的

DFS系统，分布式海量小文件系统，工业级，不是生产级，架构比较复杂的，能解决复杂的工业问题，但是不是生产级，代码层面很多的细节，都没有解决各种各样的异常和很多的问题，所以真正跑道生产环境里去，很多bug

很多crud的业务系统，技术很简单，java web，代码层面，可以考虑到很多完善的代码封装，以及异常的处理，他可以运行在生产环境，但是他不是工业级的，解决的是一些很简单的问题，crud，不是复杂的工业场景

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/036~057资料/055\_集群扩容之后，Controller如何对shard进行rebalance？/笔记.docx**

《055\_集群扩容之后，Controller如何对shard进行rebalance？》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/036~057资料/038\_模仿Redis Cluster设计我们的主从同步架构/笔记.docx**

《038\_模仿Redis Cluster设计我们的主从同步架构》

模仿redis cluster里面的主从同步架构，我们是允许一个master挂载多个slave，他可能是要做多机房容灾部署的，你可以通过参数去配置，master -> slave的数据复制，是同步的？还是异步的？多个slave的话，对1个slave是同步的，其他slave是异步的？

提供多种数据复制的策略

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/036~057资料/057\_架构总结：分布式架构、高时效性、大规模服务、高可用、可伸缩、自平衡/笔记.docx**

《057\_架构总结：分布式架构、高时效性、大规模服务、高可用、可伸缩、自平衡》

工业级的一套架构，开源的服务注册中心的水平，没有一个可以达到我们这个项目的

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/036~057资料/054\_Controller宕机后如何基于Raft协议重新选举？/笔记.docx**

《054\_Controller宕机后如何基于Raft协议重新选举？》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/036~057资料/050\_如果Slave机器挂了如何进行长连接转移？/笔记.docx**

《050\_如果Slave机器挂了如何进行长连接转移？》

一旦slave挂了，之前跟这个slave建立上连接的服务，立马随机从组里重新挑选一台机器出来，可能是master，也可能是其他的slave

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/036~057资料/053\_对宕机的Master机器，如何采用Slave角色进行恢复？/笔记.docx**

《053\_对宕机的Master机器，如何采用Slave角色进行恢复？》

必须得以slave的方式去进行一个启动，让他去连接到当前最新的master上去，跟新的master进行数据恢复，另外一个新的master感知到有slave挂载上来了，此时必然就会让一部分的机器转移长连接到新的slave上去，进行rebalance

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/036~057资料/043\_选举出Controller之后，如何给各个机器分配shard？/笔记.docx**

《043\_选举出Controller之后，如何给各个机器分配shard？》

选举controller

在某一轮投票中，每台机器都会发现大部分人都投票给了某台机器作为controller，此时在这一轮投票结束之后，每台机器都会把当选controller的机器作为controller去对待，每台机器都会默认自己的角色不是controller

作为controller自己，他其实会检查一下自己当前在本地磁盘上存储的元数据文件，刚开始启动一个集群一定是空的

此时他就需要进行一下集群里各个机器分配到的shard初始化，256个shard，此时要均匀分配给，master+slave，3个master，80多个shard，作为controller，你需要等待各个其他的master对你去进行一个注册

其他人就应该注册到controller上去，kafka的话，基于zk选举controller，broker注册到zk上去的，controller进行一个感知，我的思路是不要引入zk，避免他的可用性降低。注册的过程中，就会附带过去自己当前本地管理了哪些shard

此时由controller负责去分配shard给各个机器，你可以去采取一个均匀分配，每个机器分配到的shard是差不多的，此时每个master就知道自己应该负责管理哪些shard了

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/036~057资料/040\_数据分片之后的痛点：服务如何寻找自己所属的分片？/笔记.docx**

《040\_数据分片之后的痛点：服务如何寻找自己所属的分片？》

eureka：你随便找一个节点进行服务注册就可以了

基于类似raft协议的服务注册中心：zookeeper，等等，你找到他们的Leader去写入注册信息就可以了

数据分片，每个服务在进行注册的时候，都应该是找到一个属于自己的分片的，然后去定位到自己的分片所在的master机器，去进行服务注册，可以模仿Redis Cluster的slot槽位机制，一个key-value对是如何定位到自己的slot

根据key去进行定位

更加简单一点的办法，其实就是模仿类似于Elasticsearch也是可以的，shards分片，写入数据的时候，需要进入他的shard分片，随机的策略进入分片，根据某个key进行hash，用hash值对shard分片的数量进行取模

你这条数据的读写都是基于某个shard来走的

服务A，把自己的服务名称:ip地址:port端口号，做一个hash，通过hash对服务注册中心里所有的shard数量进行取模，就可以轻松的定位到自己是在哪个shard里面的，下一步，就是找到自己所属的shard所在的机器

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/036~057资料/039\_模仿ZooKeeper设计我们的预写日志以及数据快照的存储机制/笔记.docx**

《039\_模仿ZooKeeper设计我们的预写日志以及数据快照的存储机制》

master和slave先后都故障崩溃了，这个时候会不会导致放在内存里的数据就会全部丢失呢？导致一部分的服务注册表的数据就没有了，模仿ZooKeeper设计预写日志+数据快照组成的一套存储机制

类似于Redis的AOF+RDB数据存储机制

WAL：write ahead log，预写日志

日志，就代表了对每个服务的注册、心跳、下线、故障

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/036~057资料/041\_模仿Kafka的集群Controller机制的设计/笔记.docx**

《041\_模仿Kafka的集群Controller机制的设计》

我是归属于哪个shard，哪个数据分片的，模仿redis cluster设计的数据分片机制，redis cluster他是有固定的的slots，16384个槽位。我们作为一个服务注册中心，我们的shard数量本身是不可变的

默认的shard数量其实可以设计的多一些，服务注册中心而已，他能够最多是横向扩展多少台master机器呢？十几台，20台，肯定是抗住现在BAT级别的百万服务实例的规模，其实都是可以了

shard分片的数量，可以设计个256个分片，默认服务注册中心就不会超过100台机器

他就是用自己的服务名称:ip地址:port端口号，做一个hash，对256取模，就知道自己是属于哪个分片的了，比如说我是属于分片65的，分片65在哪台机器上呢？谁来管控哪个分片在哪里呢？

模仿Kafka，Controller机制，大数据架构里的kafka和zookeeper的课程，管控集群里每个Topic的partiton都分布在哪儿，partition进行rebalance的时候，也都是controller去进行一个管控的

会有一台机器，他的角色就是controller，这个Controller他必须去感知到集群里有多少台服务器这样，他应该负责去均匀的把集群里的shard分配给各个机器去管理，机器01负责管理的shard001~shard056，机器02负责管理的shard057~shard121

服务注册的时候，就应该去找Controller机器，Controller机器他是知道每台机器管理着哪些分片的，他就会告诉你，你就可以连接到那台机器上去，对你所属的分片，执行服务的注册这样子

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/036~057资料/037\_模仿Redis Cluster的数据分片架构设计/笔记.docx**

《037\_模仿Redis Cluster的数据分片架构设计》

我们肯定是要让我们的服务注册中心的服务注册表的数据，以后可能还会有其他的数据，是进行数据分片，才可以进行分布式的架构设计，模仿的就是Redis Cluster，可以去回顾一下我最早还是在2017年讲的一个《亿级流量电商详情页系统架构设计》的课程

Redis Cluster

他划分出来了一大堆的slot，16384个slots，每个槽位里可以放一些数据，你可以自己决定要部署多少台redis机器，16384个slots会分布在不同的机器上，10台机器，每台机器可能有1000多个slot

然后你写入和读取，都是根据你的key路由到一台机器的slot上去写入和读取

如果机器数量出现了扩容或者是缩减，此时他比较知名的一个算法是一致性hash算法，他会采取一个环的思路，虚拟槽位，基于一套算法，尽可能减少你的对数据的读写请求的缓存失效的问题

服务注册中心而言，会采用Redis Cluster数据分片机制里的一部分而已，Elasticsearch，2017年出的一套《Elasticsearch顶尖高手系列课程》，他的数据分片机制，其实就是所谓的shard机制

用shard来代表这个含义的

数据分片，shard是跟分片是对应起来的

服务注册表，分布式存储

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/036~057资料/042\_如何选举Controller：基于ZooKeeper还是Raft协议？/笔记.docx**

《042\_如何选举Controller：基于ZooKeeper还是Raft协议？》

对于Kafka而言，他选举controller，他是基于zk来搞的，我们是不希望在服务注册中心中引入一个zk的，多一个依赖，万一zk挂了，kafka也挂了。万一zk挂了，就会导致你的服务注册中心就会挂掉

基于raft协议，直接在master机器中自己去选举出来一个controller节点

每台机器都应该在一个配置文件里需要感知到集群里有哪些机器的，此时他刚开始都是投票给自己的，发送投票给别人

每台机器收到的其他人的选票，都是分别投给他们自己的

此时每台机器都陷入一个随机的休眠，肯定会有第一台机器苏醒，会投票给自己，选票发送给其他人，其他人苏醒之后，会发现已经有选票了，此时他们就会投票给这个人，发送选票出去

第二轮，最多第三轮，就会发现超过半数的机器都投票给一个人作为leader，controller

如果说他成为leader之后，他必须有一个后台线程定时的发送心跳给其他的机器，如果发现leader死掉了，此时其他机器重新选举一个controller出来

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/036~057资料/051\_等Slave机器恢复之后，如何自动进行长连接rebalance？/笔记.docx**

《051\_等Slave机器恢复之后，如何自动进行长连接rebalance？》

slave重新启动：向master进行注册，进行数据恢复，让自己跟master保持一致

长连接rebalance

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/036~057资料/036\_Peers以及Raft协议下的服务注册中心的缺陷回顾/笔记.docx**

《036\_Peers以及Raft协议下的服务注册中心的缺陷回顾》

目前市面上已有的一些服务注册中心的缺陷，consul，基于raft协议，nacos

这一周，对我们的自研服务注册中心进行一个完整的架构设计

eureka：纯peers，任何一个节点都可以接收读写请求，每台机器都要承载所有的写入的请求，把心跳上报的频率调节的很小，把拉取注册表的频率调节的很小，保证你的服务发现的时效性，注册表可能会很大，每个人都频繁的来拉取

zookeeper以及类似的基于raft协议的注册中心：类似的问题

单机抗所有并发，无法伸缩

单机抗所有数据，内存可能不够，没有办法进行伸缩

单机网卡流量有限，服务太多会打满网卡

机器，都是用千兆网卡，理论上每秒就是传输100+MB，一般还到不了，每秒也就传输几十MB这样子

单机，单机，单机

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/036~057资料/047\_心跳机制如何实现：基于数据分片机制正常上报/笔记.docx**

《047\_心跳机制如何实现：基于数据分片机制正常上报》

shard概念是非常，机器崩溃的时候，集群扩容，用一个shard去聚合多个服务的数据，服务就是跟一个shard严密的绑定在一起的，让我们的每个服务跟master之间建立一个长连接其实就可以了

心跳，应该频率高一些，服务就通过一个跟master机器的一个长连接，每秒钟都上报一个心跳过去

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/036~057资料/049\_服务发现的长连接能否基于Slave机器进行？/笔记.docx**

《049\_服务发现的长连接能否基于Slave机器进行？》

服务A的shard对应的应该是一个机器组的概念，模仿RocketMQ里面的broker group形成的一个概念，进行主从同步的几台机器，应该是可以形成一个组的，此时我们可以从这个组里的master+slaves里面，随机挑选出来一台机器

对于slave，他可以仅仅是复制心跳，对于slave角色而言，是不会启动检查心跳的线程

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/036~057资料/046\_一种比较好的服务发现方式：订阅感兴趣的服务信息/笔记.docx**

《046\_一种比较好的服务发现方式：订阅感兴趣的服务信息》

dubbo，在xml里配置你想要调用的所有服务；spring cloud feign，都是基于注解标识出来哪些是调用服务的接口。在系统启动的时候，我们是直接可以知道当前这个服务他未来将要调用的其他服务都有哪些

就直接可以针对你要调用的服务，去发送订阅请求到服务注册的机器上去

服务B调用服务A，只能通过服务A的服务名称去找他的机器所在的shard的，纠正一点，服务注册的时候，我们去定位服务所属的shard，不应该服务名称:ip地址:port端口号，根据他去hash然后找到shard

服务A部署了10台机器，每台机器上的服务A启动之后，都应该仅仅根据服务A的服务名称，去进行hash，路由到一个shard上去，去找到shard所在的机器，把当前服务A的这台机器，服务实例注册到那个shard里去

服务B启动的时候，直接对想要调用的其他服务，根据服务名称进行hash，然后路由到一个shard

任何一个服务的上线或者下线，仅仅会让服务注册中心反向推送少量的变动数据给少量的依赖这个服务的其他服务就可以了

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/036~057资料/044\_一个非常重要的思考：定时拉取注册表好不好？（上）/笔记.docx**

《044\_一个非常重要的思考：定时拉取注册表好不好？》

目前有两种模式

第一种模式：eureka、nacos，基本上都是采取的拉模式，pull模式，他的客户端会每隔30秒去拉取一次注册表，第一次是全量拉取注册表，后续是每隔30秒拉取增量注册表，就是最近一段时间又变化的注册表信息

第二种模式：推送，zookeeper，你是作为一个客户端是去监听他的代表注册表的一个节点的子节点的变化，如果有服务上线下线，此时zk会把代表注册表的所有变动后的子节点给你反向推送过来

第一种模式，比如说此时你有100个服务，最近30秒内有20个服务出现了变动，上线或者下线的情况，对于你来说，但是你只是关注其中1个服务的上线或者下线的变动，其他的服务对你而言是没有任何意义的

建议把我们最早的spring cloud里的eureka的源码分析那块去看一下

第二种模式，直接基于zk作为服务注册表，更加糟糕了，比如说你有1000个服务，但凡任何一个服务有变动之后，此时都会导致zk直接推送1000个服务给你，让你自己去感知到里面服务的变化

zk需要给其他所有服务，反向推送完整的注册表数据，此时经常会瞬间打爆网卡，每秒只能传输100mb

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/036~057资料/048\_采用什么样的频率依据心跳进行故障感知？/笔记.docx**

《048\_采用什么样的频率依据心跳进行故障感知？》

定时去检测各个shard里面的服务，最近一段时间是否上报了心跳，如果心跳正常，那么就不用做什么了，但是如果超过了一定的时间，没有心跳过来，此时你就应该认为这个服务就故障了

立马就反向通知订阅这个服务的其他服务就可以了

如果master机器发现某个服务默认超过5秒钟都没有心跳，立即就认为他是宕机了

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/036~057资料/052\_如果Master机器挂了，如何让Slave热切换为Master？/笔记.docx**

《052\_如果Master机器挂了，如何让Slave热切换为Master？》

对于一个组内的master而言，他应该是说要定时的给slave发送心跳的，告诉slave自己当前其实还活着，每秒钟发送一次心跳，各个slave一旦发现比如说超过5秒没收到这个master发送过来的心跳，此时大家就判断说，这个master必死

必须得有一个新的master出现，多个slave是否需要进行一下选举呢？

也可以基于raft协议去进行一个选举，如果只有一个slave必然是这个slave去进行接管；在这里，不要基于raft协议随便去选一个master出来，各个slave之间还是进行投票，不是随机休眠挑选一个slave

每个人去比较，谁之前从master接收到的数据是最多的，就让谁当新的master，完全是仿照zookeeper来做的；如果大家接收到的数据是一样多的，此时可以投票让slaveId最大的那个机器成为新的master

对于其他的服务而言，他们应该都转变为每隔5秒钟，去Controller节点尝试拉取这个分组里最新的master是谁，Controller主要就是管集群里有哪些机器，shard在各个分组的分配是如何定的

新的master选举出来之后，就去找controller进行一个注册

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/219~227资料/226\_实现Slave节点连接请求监听线程的逻辑（2）/笔记.docx**

《226\_实现Slave节点连接请求监听线程的逻辑（2）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/219~227资料/220\_为Master节点启动一个线程监听Slave连接请求/笔记.docx**

《220\_为Master节点启动一个线程监听Slave连接请求》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/219~227资料/227\_实现Slave节点连接请求监听线程的逻辑（3）/笔记.docx**

《227\_实现Slave节点连接请求监听线程的逻辑（3）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/219~227资料/225\_重构Slave节点连接的接收消息队列/笔记.docx**

《225\_重构Slave节点连接的接收消息队列》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/219~227资料/223\_完成Master节点启动Slave连接请求监听线程/笔记.docx**

《223\_完成Master节点启动Slave连接请求监听线程》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/219~227资料/224\_实现Slave节点连接请求监听线程的逻辑/笔记.docx**

《224\_实现Slave节点连接请求监听线程的逻辑》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/219~227资料/222\_Master网络连接管理组件的代码重构（续）/笔记.docx**

《222\_Master网络连接管理组件的代码重构（续）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/219~227资料/219\_承上启下，回顾已经完成的代码以及接下来要做的计划/笔记.docx**

《219\_承上启下，回顾已经完成的代码以及接下来要做的计划》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/219~227资料/221\_Master网络连接管理组件的代码重构/笔记.docx**

《221\_Master网络连接管理组件的代码重构》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/127~136资料/136\_细看ZK源码：如何通过读写线程以及队列，进行数据的发送和接收？/笔记.docx**

《136\_细看ZK源码：你发起的连接建立之后，如何准备读写线程和队列资源？》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/127~136资料/130\_细看ZK源码：鸟瞰一下QuorumCnxManager的整体源码/笔记.docx**

《130\_细看ZK源码：鸟瞰一下QuorumCnxManager的整体源码》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/127~136资料/129\_细看ZK源码：QuorumCnxManager节点通信组件都负责些什么？/笔记.docx**

《129\_细看ZK源码：QuorumCnxManager节点通信组件都负责些什么？》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/127~136资料/128\_为什么要彻底吃透ZooKeeper源码中的节点内部通信机制？/笔记.docx**

《128\_为什么要彻底吃透ZooKeeper源码中的节点内部通信机制？》

在大数据那边的课程里，深度的、系统的剖析了ZooKeeper的完整源码，zk就是peers集群的机制，也就是说对于他来讲，他集群里每个节点能干的事儿都是一样的，无非是peers里有一个人是leader，专门负责处理请求

集群里的其他节点相当于是备用节点一样的关系，专门是把数据在其他节点里复制一份，避免说单节点宕机导致数据丢失

zk集群里的每个节点在启动的时候，其实是要干跟我们的分布式服务治理平台一样的事情的，他也需要让每个节点互相之间建立用于内部通信的网络连接，id比较大的节点去连接id比较小的节点，通过这个方式避免重复建立连接

多参考优秀的开源项目呢，分布式海量小文件存储系统，网络通信这块几乎是照搬了Kafka的网络通信那块的架构，kafka、zookeeper、hdfs、rocketmq、elasticsearch，类似于这些最流行的中间件

随便你挑选一个最流行和最成熟的中间件，参考他里面的架构设计和源码细节实现，都是没问题的，尤其是网络通信这种

主要就是去参考zk的节点内部网络通信机制的实现细节

又可以仔细看一下zk的集群内部基于ZAB协议（raft协议改造过来的），每一行代码都仔细看一下，我们把自己的master node的leader选举机制给做了

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/127~136资料/134\_细看ZK源码：连接建立之后，读写线程以及队列如何准备？/笔记.docx**

《134\_细看ZK源码：别人跟你建立连接之后，要做哪些事情？》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/127~136资料/131\_细看ZK源码：基于BIO的节点内部通信以及reuseAddress网络参数/笔记.docx**

《131\_细看ZK源码：Listener线程是如何监听其他节点的连接请求的？》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/127~136资料/132\_细看ZK源码：基于BIO监听和建立网络连接的生产级代码/笔记.docx**

《132\_细看ZK源码：基于BIO监听和建立网络连接的生产级代码》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/127~136资料/127\_梳理一下已经设计好的系统核心组件以及交互流程/笔记.docx**

《127\_为什么要彻底吃透ZooKeeper源码中的节点内部通信机制？》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/127~136资料/133\_细看ZK源码：如何避免两台机器之间的重复连接问题？/笔记.docx**

《133\_细看ZK源码：别人跟你建立连接之后要如何管理这个连接？》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/127~136资料/135\_细看ZK源码：你如何主动向其他节点发起连接请求？/笔记.docx**

《135\_细看ZK源码：你如何主动向其他节点发起连接请求？》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/228~237资料/231\_编写Slave节点配置文件解析校验逻辑（2）/笔记.docx**

《231\_编写Slave节点配置文件解析校验逻辑（2）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/228~237资料/232\_编写Slave节点的启动流程的业务代码/笔记.docx**

《232\_编写Slave节点的启动流程的业务代码》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/228~237资料/233\_实现Slave节点去连接Master节点的过程（1）/笔记.docx**

《233\_实现Slave节点去连接Master节点的过程（1）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/228~237资料/230\_编写Slave节点配置文件解析校验逻辑（1）/笔记.docx**

《230\_开始开发Slave节点的启动入口代码》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/228~237资料/237\_再说分布式服务治理平台未来开发计划/笔记.docx**

《237\_再说分布式服务治理平台未来开发计划》

服务注册和发现，把路由机制、数据分片、持久化机制、主从同步机制、服务注册功能、服务基于长连接的发现功能、服务心跳功能、故障感知功能、master+slave故障转移、master之间的rebalance、controller重选举机制、多种部署架构（单节点、单主单从、3主3从、多主3候选人多从）

把我们的分布式服务治理平台，无缝对接spring cloud和dubbo等主流服务框架技术栈，去对接的过程中，参考spring cloud标准和规范，spring boot自动装配机制，spring高级注解机制，去跟主流框架技术栈适配

接下来就是专注开发服务治理的一系列的功能，服务监控、链路追踪、日志中心、配置中心，等等一系列的功能，就是基于服务治理平台去全方位的管控你的分布式服务，对于中小公司而言，只要你能够接入我们的平台，直接适配你现在用的dubbo、spring cloud这种主流技术栈

还要做一个高性能的API网关

把微服务框架技术栈都重新做一遍，走类似dubbo的RPC框架，做很多的高阶功能的封装，做成企业级的开发框架，熔断、限流、隔离，分布式事务，包括对很多第三方开源中间件的适配（MQ、redis、mongodb）

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/228~237资料/234\_实现Slave节点去连接Master节点的过程（2）/笔记.docx**

《234\_实现Slave节点去连接Master节点的过程（2）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/228~237资料/236\_测试Slave节点启动以及连接Master节点的过程/笔记.docx**

《236\_测试Slave节点启动以及连接Master节点的过程》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/228~237资料/229\_为三个master节点编写对应Slave节点的配置文件/笔记.docx**

《229\_编写一份Slave节点的配置文件》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/228~237资料/235\_实现Slave节点去连接Master节点的过程（3）/笔记.docx**

《235\_实现Slave节点去连接Master节点的过程（3）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/228~237资料/228\_回顾上周代码逻辑以及本周Slave启动流程说明/笔记.docx**

《228\_回顾一下上周写好的代码逻辑》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/177~187资料/177\_实现写IO线程获取待发送请求的代码逻辑/笔记.docx**

《177\_实现写IO线程获取待发送请求的代码逻辑》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/177~187资料/178\_实现阻塞式获取待发送消息的逻辑与分析/笔记.docx**

《178\_实现阻塞式获取待发送消息的逻辑与分析》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/177~187资料/182\_完成读IO线程读取消息的代码逻辑/笔记.docx**

《182\_完成读IO线程读取消息的代码逻辑》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/177~187资料/186\_实现获取别的节点发送过来的选举投票的逻辑/笔记.docx**

《186\_实现获取别的节点发送过来的选举投票的逻辑》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/177~187资料/187\_基于简化版的raft投票算法测试master节点间通信/笔记.docx**

《187\_基于简化版的raft投票算法测试master节点间通信》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/177~187资料/183\_为读IO线程实现完善的异常处理机制/笔记.docx**

《183\_为读IO线程实现完善的异常处理机制》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/177~187资料/184\_初步实现一个简单的基于raft协议的controller选择算法/笔记.docx**

《184\_初步实现一个简单的基于raft协议的controller选择算法》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/177~187资料/180\_为写IO线程实现完善的异常处理机制/笔记.docx**

《180\_为写IO线程实现完善的异常处理机制》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/177~187资料/179\_仿照ZK源码基于IO流发送消息给远程master节点/笔记.docx**

《179\_仿照ZK源码基于IO流发送消息给远程master节点》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/177~187资料/181\_实现读IO线程以阻塞方式来读取数据的逻辑/笔记.docx**

《181\_实现读IO线程以阻塞方式来读取数据的逻辑》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/177~187资料/185\_初步实现一个简单的基于raft协议的controller选择算法（续）/笔记.docx**

《185\_初步实现一个简单的基于raft协议的controller选择算法（续）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/058~072资料/066\_给我们的项目起一个名字吧：ss-govern，服务治理平台！/笔记.docx**

《066\_给我们的项目起一个名字吧：ss-govern，服务治理平台！》

服务治理平台，微服务套件，中间件项目，几个月，未来两三年，未来N多年，在这块要做的一些事情，大家应该都很清楚了

服务注册 -> 配置管理 -> 服务治理 -> 跟rpc框架、网关的集成

ss-govern，治理，石杉架构班做的服务治理平台

ss-gateway

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/058~072资料/064\_未来5~10年在中间件架构上的长远规划：技术组件国产化/笔记.docx**

《064\_未来5~10年在中间件架构上的长远规划：技术组件国产化》

架构班未来一直持续的办下去，每年都会更新一些最新的课程，这是必然的，两三年主要专注于做国产的最优秀的微服务套件，包含很多的中间件，都是自研的，替代大量的国外开源的技术

支线课程呢个一直去做的一个事儿，每次做好了一个项目之后，就会孵化出来一个开源社区，回馈和服务于自己国家里大量的公司，用自己国产化的技术，比进口的国外的开源技术，更好，容易用，功能更加强大，迭代速度更快，中文文档，中国人交流起来方便

更多技术组件的国产化，我行业里干了十多年，技术水平是有的，阅读源码（流行的开源项目的源码，基本上都读过），自研中间件，设计大型架构，5年~10年的一个时间和光景，利用架构班的教学，不停的带大家去自研一些中间件

缓存，国外开源的redis，我们可以不可以自己去做一下，做一个融合了redis架构思想的高性能、高并发的分布式kv存储，服务于大量公司；MQ，RocketMQ已经发展的很好了，做的也挺好的；elasticsearch，分布式搜索引擎系统，我们也可以自研；分布式数据库；类似于spring的框架，都是自己去写

中国人而言，我们是不是可以做的比老外更好，我们完全可以去做更好的开源项目，有一天，我们做的开源项目，国际化和推广，是否可以推到海外去

大数据领域，整个国内，大数据领域的技术全部是国外，hadoop、spark、hive、flink、hbase、druid、clickhouse，我们是不是可以考虑全部都自己重新做一遍，做中国人版本的开源系统，做的比老外的那套东西，健壮，稳定，好用，功能强大

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/058~072资料/067\_在IntelliJ IDEA中搭建工程结构：拆分为server和client两个模块/笔记.docx**

《067\_在IntelliJ IDEA中搭建工程结构：拆分为server和client两个模块》

ss-govern-common

ss-govern-server

ss-govern-client

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/058~072资料/065\_中间件架构技术支线课程以及开源社区，对架构班学员的机会/笔记.docx**

《065\_中间件架构技术支线课程以及开源社区，对架构班学员的机会》

很多不太懂行情的同学，眼界稍微窄小一些的同学，可能会考虑到一个问题，项目明明都是架构班内部的项目，付费了学费进来学的，怎么能说开源代码就开源代码呢？有的人可能会觉得有点想不通

中间件架构自研的课程，提升你的技术，积累中间件项目的经验，出去面试，每次大家都是说我在自己公司里自研了XXX项目，到底做的如何，有没有真正如你所说那么好的效果呢，打一问号

小公司，自研的一些项目，最多技术还不错

中间件项目，开源社区，孵化 -> 推广，国内某一个领域里数一数二的开源项目，知名度极高，如果你还在里面参与了一些后续的开发，成为了知名开源项目的commitor，对你未来哪怕是求职

含金量出来，差了10个数量级的水准

RocketMQ的commitor，Spark的commitor，人家看待你的眼光都不一样了，这个是大牛，做开源项目之后会有很多的好处，比如说你有时候可以参与一些活动，meetup，演讲和分享，服务于一些公司的咨询

可能就是有一些人把你当做大牛来挖你了

RocketMQ，Hadoop，Spark，Kafka，如果没有一些特别好的课程，比如我讲解的一些课程，正常人去看一个开源社区的源码，根本是看不懂的，大部分人对开源社区的项目，仅仅是使用和了解原理而已，生产环境的优化

少数的人会看明白他的源码，看明白源码跟参与我们的架构班的课程，跟着一步一步手撸出来，完全不是一个数量级的水准

开源社区之后，最大的机会都是在架构班学员里面，你去参与开源项目，成为commitor之后，对你的好处实际上来说是最大的，对行业的好处也是很大的，大量的公司可以来使用我们做出来的特别好用的开源项目

大部分工程师可以来使用我们开源出去的技术，自己看看源码，掌握一些原理

参与了我们的这个架构班的课程，可以跟着我从0开始手撸出来一个复杂的中间件项目，commitor获取巨大的行业声望，任何一个开源项目，commitor都会有二三十人那样子，服务治理平台

后续还会有很多项目，大家只要努力学习，抓住机会，每个人都一定可以在至少一个开源项目里，找到自己的机会

参加架构班课程 -> 做中间件架构项目 -> 开源社区 -> commitor -> 对所有架构班同学，价值最大化

非架构班的普通人 -> 使用开源技术在自己的项目里，产生价值，稍微了解一些原理 -> 你们每个人在没有参加我的架构班的时候，对各种开源技术不就是这个水平，大致用一下，大致了解他的原理

很多技术，有剖析源码的书，光看那些书，还是看不懂源码

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/058~072资料/062\_微服务套件中的其他项目的规划展望和技术思路/笔记.docx**

《062\_微服务套件中的其他项目的规划展望和技术思路》

自研一些东西，自研微服务套件，2年左右的时间，服务治理平台（服务注册中心的基础之上进行的一个扩展），二次封装企业级的RPC框架，高性能网关，隔离熔断框架，分布式事务框架，链路追踪系统，日志中心系统，集群监控系统，替代掉目前广泛使用的国外开源的一些技术

微服务系统，高并发、高可用、高性能、海量数据、安全性、可伸缩、多机房，高大上，不是每个公司都需要使用的，28法则，80%的公司，可能都不需要上面说的一大堆高端、大气、复杂的大型架构设计

并没有那么多的用户量，跟着我把架构师课程里后续的大量的高端复杂的架构，都学会，可以进入20%的有很大的技术难度和挑战的一些公司里去，他们会出更高的薪资去挖技术最好的一批牛人

（1）基于DDD的复杂业务架构的设计

（2）基于全套微服务组件的分布式系统的架构设计

（3）在这个公司的特定领域里面，往往会有一些额外的特殊技术需求，爬虫，加密

微服务套件，可以是面向20%的中大型公司，也可以是面向80%的中小型公司，无论是大公司，还是小公司，其实都需要这样的一个套件，他的面向性是最广阔的

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/058~072资料/071\_并发以及数据结构的技术选型：基于之前的JDK源码课程的内容/笔记.docx**

《071\_并发以及数据结构的技术选型：基于之前的JDK源码课程的内容》

并发的一些东西，很多线程可能会访问一些共享的内存数据结构，也本身就需要使用一些数据结构，list、map、set、队列，volatile、synchronized、读写锁、cas、线程安全的集合类、线程池，很多东西会需要进行使用

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/058~072资料/061\_真正的宏远规划：技术国产化，替代Spring Cloud的微服务套件/笔记.docx**

《061\_真正的宏远规划：技术国产化，替代Spring Cloud的微服务套件》

服务治理平台这块的一个定位，就是站在服务注册中心的基础之上，然后做一些拔高，在里面可以加很多的服务治理的功能，完善的监控，高阶的治理功能，都是你运作一个几十个、几百个服务实例，必须的一些功能

hadoop、spark、flink、kafka、hbase，所有的技术都是国外的大公司去开源的，都是国外的大公司为了满足自己的庞大的数据量的需求，自己研发了一些分布式大数据系统，然后对外开源

了解那些技术如何使用，了解他们的原理，了解一些 优化技巧，大数据架构课程里，阅读人家的一些源码，做一点二次开发，稍微做一点改动和优化，就已经算做是国内顶尖的技术高手了

spring + hibernate + struts2，spring + spring mvc + mybatis，国内公司用的各种各样的技术，都是来源于国外的开源技术，我们国内的行业情况，大家都是盯着人家开源的一些技术和好东西

spring cloud，服务注册中心，国外公司开源的

阿里巴巴，spring cloud alibaba，dubbo替代feign+ribbon，nacos替代eureka，sentinal替代hystrix，替代掉spring cloud里面一些netflix等国外公司开源的一些组件，按照spring cloud的规范，集成到spring cloud技术体系里去

国家，应该走的一个趋势，是核心技术国产化，芯片、操作系统，核心技术，慢慢华为、京东方等顶尖大公司，都在做自研，替代国外的一些技术；作为一个Java领域的码农，也得有一点志气

自研一些东西，自研微服务套件，2年左右的时间，服务治理平台（服务注册中心的基础之上进行的一个扩展），二次封装RPC框架，自研高性能网关，隔离熔断框架，分布式事务框架，链路追踪系统，日志中心系统，集群监控系统，替代掉目前广泛使用的国外开源的一些技术

官方文档，全部是用中文来写，方便国内的工程师去学习和使用，我们的社区全都中文的开源社区，大家都在里面反馈问题，解决问题，升级和迭代开源技术

架构班里，一点一点自己研发出来一套微服务套件，替代掉spring cloud里的各种技术，spring cloud本身就是一个规范，基于spring、spring boot的一套规范，集成了一些国外开源的技术到里面去

跟spring、spring boot、spring cloud做集成和整合，单独用，整合到你目前的技术体系里去，也是可以的

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/058~072资料/059\_这个项目将要专注的领域：以服务粒度为中心的治理/笔记.docx**

《059\_这个项目将要专注的领域：以服务粒度为中心的治理》

服务治理平台，服务注册中心 -> netflix开源出来的是eureka，dubbo作为一个RPC框架使用的是zookeeper作为服务注册中心 -> 分布式系统，必然会拆分很多子系统，拆分很多的服务，各个服务之间要通信，服务注册中心 + RPC框架

三四年以前，自研RPC框架 + 自研服务注册中心，中小公司用的Dubbo + ZooKeeper，互相之间知道对方的地址，此时就可以基于TCP协议建立长连接，然后进行网络通信了，本质就是如此

两三年之前，Spring Cloud开始冒头，国外整合了netflix开源的微服务的一些组件，还有其他公司开源的微服务的组件，只不过是以spring社区为核心整合了一些国外大公司开源出来的组件，形成了一个微服务系统的开发套件

feign + ribbon作为RPC框架，eureka作为服务注册中心，hystrix作为分布式系统的隔离熔断框架，zuul、gateway项目作为分布式系统的网关系统，开源出来，凭借的是spring社区的强大影响力

最近两三年，各种中小公司用的主要都是spring cloud，dubbo + zookeeper的组合用的少了，他缺乏了很多的功能，比如隔离熔断框架，hystrix，zuul作为一个网关非常的简单，起码可以用

链路监控、跟mq的整合、测试套件，spring cloud就是把国外各种开源框架采用一个通用的协议集成和整合起来，让他来覆盖一个分布式系统/微服务系统需要使用的方方面面的技术和框架

光是一个spring cloud是不够用的，中小公司刚开始从0启动搭建一个系统的时候，用spring cloud还是挺方便的，用着用着就会发现不满足需求了，有很多服务实例，几十个服务实例，几百个，上千个服务实例，几万个服务实例

只有一个简单的服务注册中心，是满足不了我们对一个优点规模的微服务系统进行管理和治理的需求了，有一个可视化的面板，可以看到很多东西，比如说我们的服务的拓扑架构图，静态化的拓扑图

在我们生产环境里，各个服务之间是如何进行调用的，包含哪些调用链路，从我的入口服务开始，每个服务的QPS是多少，可用性是多高，性能是多高，在一个调用链路中，各个服务环节的性能是多高

线上生产环境里，各个系统部署的机器上一些资源的负载情况，cpu、内存、磁盘、jvm gc、网络，我希望能够看到这些东西

分析、项目配置的管理灰度发布、蓝绿部署、流量权重、自动降级、手动降级、接口版本、服务负载的自动告警、机器负载的自动告警、服务扩容的自动告警、调用链路的性能分析和优化、调用链路的日志

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/058~072资料/068\_网络通信如何进行技术选型：Netty还是NIO？/笔记.docx**

《068\_网络通信如何进行技术选型：Netty还是NIO？》

netty，还是nio，还是bio，个人是比较推崇zookeeper他的一个网络通信架构的，kafka、zookeeper，我们已经看过的开源项目，分布式系统，他们都可以支撑大量的客户端的连接，集群内部也都要进行大量的通信

甚至包括hadoop，hdfs，网络通信代码我们也都看过

hdfs、kafka、zookeeper，nio+bio混合用，集群内部通信其实使用bio就可以了，如果说是跟客户端建立长连接，其实是用nio就可以了，他们也都没有使用netty，而且也都使用了很多年，运行了很多年，也没什么问题

netty很好用，我们在自研im系统的时候，直接基于netty来开发网络通信程序是很方便的，他是一个框架，对于任何一个分布式系统而言，网络通信都是至关重要的一个环节，不能出现问题

netty，万一，有一些bug，底层有一些地方不如你的心意，难道你还要去修改netty底层的源码吗？如果你是一个IM系统，偏向于业务一些的系统，其实用netty来开发是可以的，可以让你专注于你的业务功能的实现

中间件系统，大数据系统，最好还是基于nio+bio混合编程，实现自己的网络通信架构，用的API是最原始的API，即使出了一些问题，也是可以自己来解决的，哪怕是说基于nio+bio进行网络通信和编程

你只要自己摸索摸索，自己都能解决

集群内部的通信，master和master，master和slave，都可以采用BIO架构去进行通信，如果是建立大量的客户端的长连接，都可以用NIO架构去进行通信，就可以了，具体的网络通信的过程，可以参考zookeeper里的源码

大数据架构课程里的zookeeper，源码，讲解的非常的细致和透彻，现在我们每一个人都可以对zk的源码很了解和很熟悉，在进行一些网络编程的时候，我可能就会带着大家去参考zk的源码

如果说zk这样一个成熟的世界级的分布式系统，他的网络通信架构那样写没问题的，那么我们在自己的中间件里这样写也一定是没问题的

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/058~072资料/063\_微服务套件的全线路roadmap和时间节点规划/笔记.docx**

《063\_微服务套件的全线路roadmap和时间节点规划》

自研一些东西，自研微服务套件，2年左右的时间，服务治理平台（服务注册中心的基础之上进行的一个扩展），二次封装企业级的RPC框架，高性能网关，隔离熔断框架，分布式事务框架，链路追踪系统，日志中心系统，集群监控系统，替代掉目前广泛使用的国外开源的一些技术

到明年的4月份左右

自研分布式服务治理平台（服务注册 -> 配置中心 -> 跟dubbo/springcloud/开源网关进行集成 -> 服务治理 -> 可视化工作台） -> 二次封装企业级的RPC框架，高性能网关，隔离熔断框架 -> 自研中间件的能力提到最高，历届阿里天池中间件大赛的赛题以及优秀项目的思路分析

v0.x的一个版本，就是不停的迭代和增加功能

这个项目就可以从架构班课程里毕业了，可以自己孵化开源项目，apache基金会那套孵化，完全我们自己做也是可以的，v0.x版本，基本可用了，我们会在全体架构班的学员里找一些同学，尝试开始落地到自己的公司里去

一边观察他在生产环境的一些表现，修复一些bug、根据大家的需求增加一些新的功能，这个过程里一直迭代的就是v0.x版本

v1.0版本，就是生产可用的版本，就会全面铺开，在架构班里只要有同学可以在自己公司里使用，就可以去自己使用，申请成为这个开源项目的commitor，跟着一起来维护这个开源项目

几十家公司使用都没有什么问题了，v1.x系列版本了，我就可以利用我的一些个人公众号等等资源，去做这个开源项目的推广，知名开源项目，后续就是以架构班参与进来的学员为一个主体，不断的迭代和演进这个开源项目

在架构班课程体系里，在明年4月份之后，会单独拉一个课程分支，持续不断的讲解更多的中间件项目的自研，二次封装企业级的RPC框架，高性能网关，隔离熔断框架，分布式事务框架，链路追踪系统，日志中心系统，集群监控系统，替代掉目前广泛使用的国外开源的一些技术

但是到明年4月份之后，我们的一个主线课程，会变成复杂业务架构的设计，DDD，中台，高并发，高可用，海量数据，高性能

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/058~072资料/070\_序列化协议：自定义协议 + Protobuf序列化/笔记.docx**

《070\_序列化协议：自定义协议 + Protobuf序列化》

hdfs、kafka、分布式海量小文件系统，自定义协议，几个字节是什么东西，几个字节是什么东西，好处在于纯粹自定义，节约空间的，对一些复杂结构的对象，传输的话，就很麻烦了，那就不好办了

zookeeper的序列化协议，自定义协议，header，body（一些复杂结构的对象，使用im系统中的protobuf的方式对复杂对象进行序列化和反序列化）

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/058~072资料/072\_给每个人的提醒：尽快跟上进度，吃透内容，大数据也要同时跟上/笔记.docx**

《072\_给每个人的提醒：尽快跟上进度，吃透内容，大数据也要同时跟上》

Java架构这块，在今年下半年，实际上来说，我们的技术难度不是太大，netty，im核心代码，讲了讲一些架构设计，硬核的项目实战，分布式服务治理平台，大数据那块，也有一些课程，zookeeper课程，讲解的非常的透彻和硬核

im项目给吃透了，有什么问题都可以来问我的，im系统出去都是你一个很大的优势

hdfs，分布式存储系统的架构，直接指导和影响了分布式海量小文件存储系统的架构，kafka、zookeeper

分布式文件系统上，都没吃透，还停留在2019年上半年的进度

少量的人，是直接跟上我的进度的

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/058~072资料/069\_磁盘读写的技术选型：参考Zookeeper的底层实现/笔记.docx**

《069\_磁盘读写的技术选型：参考Zookeeper的底层实现》

参考一下zookeeper的磁盘读写的技术实现就可以了，他其实在他的实现原理中，也是有自己的一套磁盘读写机制，每次增删改的时候，都会在磁盘文件里写入一条事务日志，每隔一段时间会把内存里的数据写一份到磁盘上去做成快照

到时候在写代码之前，我会直接带着大家去参考一下zk磁盘读写的细节性的代码，我们直接参考一下就可以自己写了，如果zk这样工业级的系统，都可以在磁盘读写上采取这样的思路来做

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/058~072资料/060\_罗马不是一天建成的：服务治理平台的演进roadmap/笔记.docx**

《060\_罗马不是一天建成的：服务治理平台的演进roadmap》

分布式服务治理平台的架构 -> 服务注册 -> 配置管理 -> 可视化工作台 -> 跟dubbo/spring cloud进行集成-> 跟开源的主流网关系统进行集成（zuul）-> 服务治理（服务监控，机器监控，服务管理）

基于dubbo/spring cloud进行二次封装

自研高性能网关系统

服务治理平台 -> 微服务架构套件

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/209~218资料/211\_开启第一轮投票：把票投给别的候选节点（续）/笔记.docx**

《211\_开启第一轮投票：把票投给别的候选节点（续）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/209~218资料/218\_对controller投票选举机制进行完整测试/笔记.docx**

《218\_对controller投票选举机制进行完整测试》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/209~218资料/210\_开启第一轮投票：把票投给别的候选节点/笔记.docx**

《210\_开启第一轮投票：把票投给别的候选节点》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/209~218资料/215\_第一轮投票失败后，调整选票开启下一轮投票/笔记.docx**

《215\_第一轮投票失败后，调整选票开启下一轮投票》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/209~218资料/209\_初始化自己第一轮投票的选票/笔记.docx**

《209\_初始化自己第一轮投票的选票》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/209~218资料/214\_收到选票之后开始进行归票以及quorum判定（续）/笔记.docx**

《214\_收到选票之后开始进行归票以及quorum判定（续）》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/209~218资料/213\_收到选票之后开始进行归票以及quorum判定/笔记.docx**

《213\_收到选票之后开始进行归票以及quorum判定》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/238~247资料/238\_初步看一下ZooKeeper构造函数的执行流程/笔记.docx**

《238\_初步看一下ZooKeeper构造函数的执行流程》

Master节点的启动以及跟其他所有Master节点之间的网络连接的互相建立，任何一个Master只要成功的启动之后，就代表说他已经跟其他所有的Master节点都建立了网络连接了，还会在至少3个controller candidate之间选举出来一个controller

你可以启动所有的slave，slave可以跟master正常保持一个长连接，你的服务治理平台的server端一旦启动好了之后，就代表所有的master、controller、slave都已经ready，都在运行中了

客户端这块，正常来说就应该找到我们的某一台服务器，然后去建立一个长连接，进行服务的注册以及维持一个心跳，你当然自己也可以从那台服务器去进行一个服务发现，去获取你需要的服务的实例的地址列表

本周我们就细致的分析一下zk的客户端跟server端建立长连接的过程

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/238~247资料/245\_ZK服务端启动的时候是如何监听端口号的？/笔记.docx**

《245\_ZK服务端启动的时候是如何监听端口号的？》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/238~247资料/247\_ZK服务端如何维护管理建立好的连接？/笔记.docx**

《247\_ZK服务端如何维护管理建立好的连接？》

服务端如何启动的时候监听端口号，客户端如何发起连接，服务端如何接收到一个连接请求，客户端和服务端如何共同阻塞住一起完成连接，客户端完成连接之后是如何维护管理这个连接的，服务端是如何维护管理大量客户端的网络连接的，接下来客户端和服务端是如何准备好进行读写IO通信的

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/238~247资料/243\_ZK客户端最终如何完成与服务端的连接建立？/笔记.docx**

《243\_ZK客户端最终如何完成与服务端的连接建立？》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/238~247资料/246\_ZK服务端如何与客户端建立连接？/笔记.docx**

《246\_ZK服务端如何与客户端建立连接？》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/238~247资料/240\_客户端网络通信组件是如何进行封装的？/笔记.docx**

《240\_客户端网络通信组件是如何进行封装的？》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/238~247资料/244\_ZK客户端与服务端建立成功连接后会做什么？/笔记.docx**

《244\_ZK客户端与服务端建立成功连接后会做什么？》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/238~247资料/239\_客户端是如何基于NIO实现用于通信的Socket？/笔记.docx**

《239\_客户端是如何基于NIO实现用于通信的Socket？》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/238~247资料/241\_客户端网络通信组件底层的两个线程启动后会干什么？/笔记.docx**

《241\_客户端网络通信组件底层的两个线程启动后会干什么？》

ZooKeeper -> 网络通信组件 -> 网络通信线程 -> 检查是否跟ZK Server建立连接，如果没有建立连接，则跟一台zk server建立长连接，基于NIO来实现

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/238~247资料/242\_详细分析ZK客户端与服务端建立连接的过程/笔记.docx**

《242\_详细分析ZK客户端与服务端建立连接的过程》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/137~146资料/141\_初步实现建立网络连接的代码逻辑/笔记.docx**

《141\_初步实现建立网络连接的代码逻辑》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/137~146资料/143\_实现id比自己小的master节点列表的解析逻辑/笔记.docx**

《143\_实现id比自己小的master节点列表的解析逻辑》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/137~146资料/138\_实现标准的系统运行变量以及重试参数定义/笔记.docx**

《138\_实现基于ServerSocket进行网络连接监听的代码》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/137~146资料/140\_实现解析master节点内部网络通信端口号的逻辑/笔记.docx**

《140\_实现解析master节点内部网络通信端口号的逻辑》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/137~146资料/142\_初步实现Master内部网络连接的读写线程/笔记.docx**

《142\_初步实现Master内部网络连接的读写线程》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/137~146资料/137\_实现监听其他MasterNode连接请求的线程/笔记.docx**

《137\_实现MasterNode等待监听其他节点的连接请求》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/137~146资料/145\_完成主动连接id较小的Master的过程/笔记.docx**

《145\_完成主动连接id较小的Master的过程》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/137~146资料/139\_实现基于ServerSocket进行网络连接监听的代码/笔记.docx**

《139\_实现基于ServerSocket进行网络连接监听的代码》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/137~146资料/144\_主动跟id比自己小的master节点建立网络连接/笔记.docx**

《144\_主动跟id比自己小的master节点建立网络连接》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/137~146资料/146\_梳理一下本周完成的Master节点内部连接过程/笔记.docx**

《146\_梳理一下本周完成的Master节点内部连接过程》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/167~176资料/174\_开始实现发送消息给其他节点的方法逻辑/笔记.docx**

《174\_开始实现发送消息给其他节点的方法逻辑》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/167~176资料/172\_参考一下ZK源码里的RecvWorker是如何接收消息的？/笔记.docx**

《172\_参考一下ZK源码里的RecvWorker是如何接收消息的？》

如果你是节点1，你要发送一条消息给节点2

此时你应该是把消息发送到节点2的队列里去，跟节点2的长连接的SendWorker会从节点2的队列里获取消息，通过IO流发送出去

节点2，跟节点1之间的长连接的Socket是对应了一个RecvWorker的，就是这个节点2的RecvWorker会从跟节点1的长连接的Socket里读取数据，读取到了之后，就接收到了一条消息，放入一个接收消息的队列就可以了

节点2而言，消息接收队列，有一个就可以了，节点2应该专门有一个请求处理线程，从消息接收队列里去获取请求，一个请求一个请求的处理

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/167~176资料/168\_迄今为止开发好的代码流程梳理和总结/笔记.docx**

《168\_迄今为止开发好的代码流程梳理和总结》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/167~176资料/167\_上周的视频没声音问题说明以及小bug修改/笔记.docx**

《167\_上周的视频没声音问题说明以及小bug修改》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/167~176资料/176\_封装从接收队列里获取消息的代码逻辑/笔记.docx**

《176\_封装从接收队列里获取消息的代码逻辑》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/167~176资料/173\_参考ZK源码里请求放入接收队列以后，是如何处理的？/笔记.docx**

《173\_参考ZK源码里请求放入接收队列以后，是如何处理的？》

RecvWorker不停的从对应的一个长连接的Socket里读取数据，有请求就接收过来，塞入recvQueue里去，你什么地方需要接收消息的时候，就从recvQueue里去进行一个获取，我觉得是一点都不好的

节点之间的通信的时候，但凡出现了网络异常直接退出，就说明你肯定是有问题了，此时的话呢系统就无法运行了

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/167~176资料/175\_基于发送队列实现发送请求方法的逻辑/笔记.docx**

《175\_基于发送队列实现发送请求方法的逻辑》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/167~176资料/169\_参考一下ZK源码中的节点间发送请求是如何做的？/笔记.docx**

《169\_参考一下ZK源码中的节点间发送请求是如何做的？》

节点间的请求发送和请求接收，响应返回和响应接收

网络长连接都已经建立好了，接着其实就是互相之间要发送请求，接收请求，处理请求，返回响应，接收响应，处理响应，底层的网络通信代码封装好

选举，controller选举，主控节点，ZAB协议和算法

服务注册作为服务治理平台的第一个功能，让大量的 服务粒度的数据分布式存储在多个节点中，QPS、接口性能、可用性，服务粒度的统计监控，就可以做了，服务的配置管理，等等，一系列的服务治理就可以做了

节点1和节点2建立好了长连接

节点1这里是有一个跟节点2的长连接Socket，针对这个Socket，节点1其实是有两个IO线程的，一个是读线程，一个是写线程

如果节点1要给节点2发送请求，其实是把请求交给跟节点2的长连接的Socket对应的写线程，那个线程是有跟节点2的长连接Socket的，通过 这个Socket，就可以把请求发送到节点2去了

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/167~176资料/170\_仔细分析ZK源码中的SendWorker如何发送请求给其他节点/笔记.docx**

《170\_仔细分析ZK源码中的SendWorker如何发送请求给其他节点》

# **File Path: /Users/xiaotingting/Downloads/学习资料/14\_自研分布式微服务注册中心unzip/167~176资料/171\_参考ZK源码里的SendWorker线程结束的资源清理机制/笔记.docx**

《171\_参考ZK源码里的SendWorker线程结束的资源清理机制》