背景

在微服务体系中，服务之间的调用、关系变得越来越复杂，而业务本身的频繁变更让运维成本大大增加，再加上不可靠的硬件及软件服务，此时一个低成本、可靠、自动容错的名字服务显得尤为重要。
对于一个分布式系统来说，名字服务是其中最核心的组件之一，也可以说它是一个分布式系统的衡量标准，更是如何治理好服务的一个基础。

微服务体系之下有几种调用关系；业务到数据库采用Dns或者Ip的方式；业务到业务采用服务注册发现或者域名的方式；业务到中间件采用Dns或者Ip的方式。

数据库现状

各大业务方需求基本类似，但是没有实现集中统一管理都是单独维护自己的服务，解决方案也各不相同，以 DBA 团队为例：

1. 业务线会与 DBA 老师先沟通好使用的域名。
2. DBA 老师会将该域名对应集群物理机节点列表维护在数据库中。
3. DBA 老师向 运维老师 发起 DNS 变更申请。
4. 申请通过后 运维老师 会手动更新 DNS 配置，reload DNS。
5. Mysql 服务异常会通过报警发出，然后由 DBA 老师人工处理（重复 2、3、4）。
6. 新增节点或者下线节点都会由 DBA 老师人工处理（重复 2、3、4）。

从 DBA 团队的实际使用情况来看有以下问题：

- 不能实现DBProxy动态的 扩/缩 容，服务扩展性不够高 。
- 数据库的DBProxy不存在探活，只能通过异常报警来人工下线不可用节点，止损时间长，无法保证高可靠 。
- 鉴权不够灵活，只能通过 IP 白名单来实现，对于有些 IP 变动频繁的服务无法实现鉴权，服务的安全性不高 。
- 人工成本较高
    - 服务 扩/缩 容更新时间 。
    - 服务 脱机/不可用 止损时间 。

总体来看各个业务线都有自己的解决方案，但是方案不够完善，存在一些弊端，服务的稳定性无法保障，急需一个服务做收敛，将问题汇总归类统一解决。

目标

指标项	指标	指标值（当前	指标值（目标）
扩展性	提升DBProxy动态缩扩容的效率	15min	3min
可靠性	降低DBProxy出现故障之后的止损时间	10min	1min
	提升DBProxy分级发布效率	10min	1min
安全性	原来是通过Ip的方式来授权，K8S下无法实现精准鉴权	0	1

解决方案

纵观Nacos、Eureka、Etcd以及Zookeeper，这些充当服务发现中心的组件，其都需要一个Client-SDK为上层应用提供根据服务名查找到一个机器IP的能力，而这就带来了一个问题，应用必须要集成这一些组件的客户端SDK，一旦组件的SDK出现问题需要升级时，就需要推动应用方的升级，这是不想见到的。而DNS协议，天然跨语言，Nacos 和 CoreDNS 结合实现名字服务，使用DNS协议作为服务发现，直接解决了跨语言的问题。

Nacos

Nacos是阿里开源的配置中心，相比与市面上常见的其他配置中心有如下亮点：

1. 支持CP也支持AP。
2. Nacos 无缝支持一些主流的开源生态。
3. Nacos 致力于帮助发现、配置和管理微服务。Nacos 提供了一组简单易用的特性集，实现动态服务发现、服务配置、服务元数据及流量管理。
4. Nacos 支持动态配置服务。动态配置消除了配置变更时重新部署应用和服务的需要，让配置管理变得更加高效和敏捷。配置中心化管理让实现无状态服务变得更简单，让服务按需弹性扩展变得更容易。
5. Nacos支持插件管理，扩展性强。

CoreDns

CoreDNS作为CNCF中托管的一个域名发现的项目，原生集成Kubernetes，它的目标是成为云原生的DNS服务器和服务发现的参考解决方案。

CoreDNS有以下3个特点。

1. 插件化（Plugins）。基于Caddy服务器框架，CoreDNS实现了一个插件链的架构，将大量应用端的逻辑抽象成插件的形式暴露给使用者。CoreDNS以预配置的方式将不同的插件串成一条链，按序执行插件链上的逻辑。
2. 配置简单化。引入表达力更强的DSL，即Corefile形式的配置文件（也是基于Caddy框架开发的）。
3. 一体化的解决方案。区别于Kube-dns“三合一”的架构，CoreDNS编译出来就是一个单独的可执行文件，内置了缓存、后端存储管理和健康检查等功能，无须第三方组件辅助实现其他功能，从而使部署更方便，内存管理更安全。

核心要点

1、基础设施为了保障高可用性，需要实现双活架构；中心实现高可用

2、基础设施为了保障高可用性，需要做多级缓存；让业务单元就近形成闭环

3、基础设施需要兼容多种寻址协议（Http、gRPC、Dns）

技术需求

提升DBProxy的扩容效率

为了提升DBProxy的扩容效率，我们就需要把扩容流程中能自动化的部分全部自动化起来，通过自动化来提升效率，降低错误率。

1）动态注册的能力

需要给DBProxy提供OpenApi或者SDK；以供DBProxy启动的时候注册到名字服务里来

2）动态调整DBProxy的负载均衡

需要给DB的Paas提供负载均衡（权重调整）变更的接口，以供DB的Paas层调用相应的API调整每个DBProxy的流量规则；PAAS平台可以主动的调整每个 Proxy的负载均衡策略。

提升DBProxy出现故障之后的止损效率

为了提升DBProxy出现故障之后的止损效率，我们就需要把扩容流程中能自动化的部分全部自动化起来，通过自动化来提升效率，降低错误率。

1）需要对DBProxy的生命周期

DBProxy需要提供状态探测（readiness），处于Ready状态之后才能对其它服务可见。这个健康检查的策略目前定的是：

- periodSeconds：检查执行的周期，默认为10秒，最小为1秒
- timeoutSeconds：检查超时的时间，默认为1秒，最小为1秒
- successThreshold：从上次检查失败后重新认定检查成功的检查次数阈值（必须是连续成功），默认为1
- failureThreshold：从上次检查成功后认定检查失败的检查次数阈值（必须是连续失败），默认为1

2）动态下线的能力

通过生命周期的管理，如果感知到DBProxy已经失联；业务方在访问数据库的时候就不能把失联的DBProxy分配给业务方。

提升DBProxy的分级发布效率

为了提升DBProxy的扩容效率，我们就需要把扩容流程中能自动化的部分全部自动化起来，通过自动化来提升效率，降低错误率。

1）动态注册的能力

需要给DBProxy提供OpenApi或者SDK；以供DBProxy启动的时候注册到名字服务里来

2）需要对DBProxy的生命周期

DBProxy需要提供状态探测（readiness），处于Ready状态之后才能对其它服务可见。

3）动态调整流量控制能力

需要给DB的Paas提供负载均衡（比例调整）变更的接口，以供DB的Paas层调用相应的API调整每个DBProxy的流量规则；PAAS平台可以主动的调整每个 Proxy的负载均衡策略。

提升DB的安全性

1）根据服务名动态的授权

原来是根据ip来授权的，在k8s下服务重新部署之后，ip变化幅度比较大，在通过ip来授权的方式就行不通了；分析之后，每个微服务都是有名字的，我们可以根据名字来做授权，这样不管ip怎么变（不管是新增、还是删除）的场景都能支持。

监控需求

1）Metric指标监控

需要通过指标信息监控系统中每个节点上的SLA。

1、延迟时间（平均、最小、最大）；2、并发数（平均、最小、最大）；3、错误数（平均、最小、最大）。

通过黄金3指标来实时呈现系统的健康度

2）全链路监控

需要把链路上所有节点的信息通过图表方式呈现，方便定位问题。

3）Logging查询

需要把各个节点的日志信息收拢到日志平台，方便定位问题。

Web平台管理需求

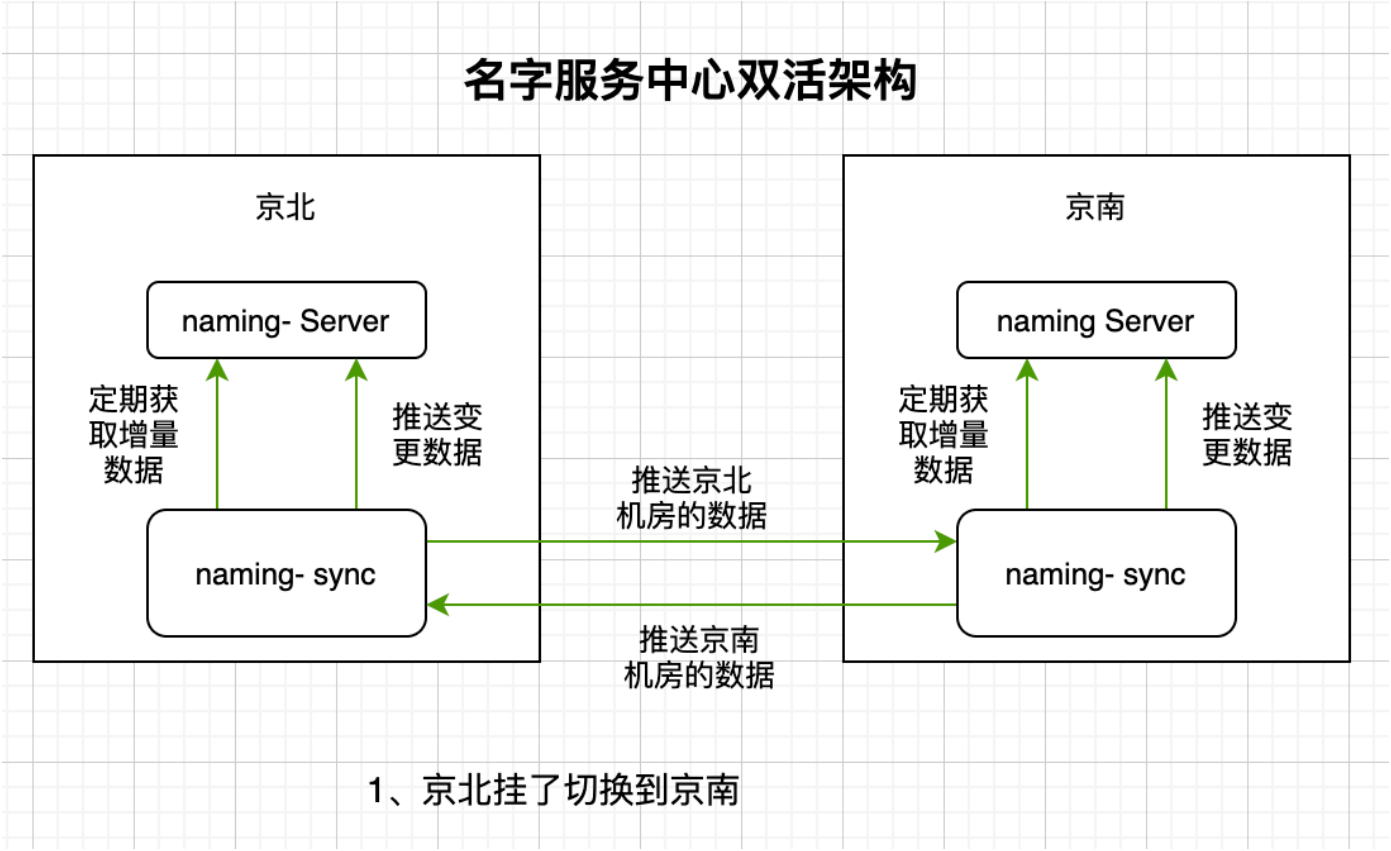
1）需要有web页面管理DBProxy的ACL配置

2）需要有web页面管理DBProxy服务的实例信息

3）需要有web页面管理DBProxy的订阅者

架构设计

中心双活方案1



1、两边机房的数据通过同步组件来做数据同步

2、架构更简单

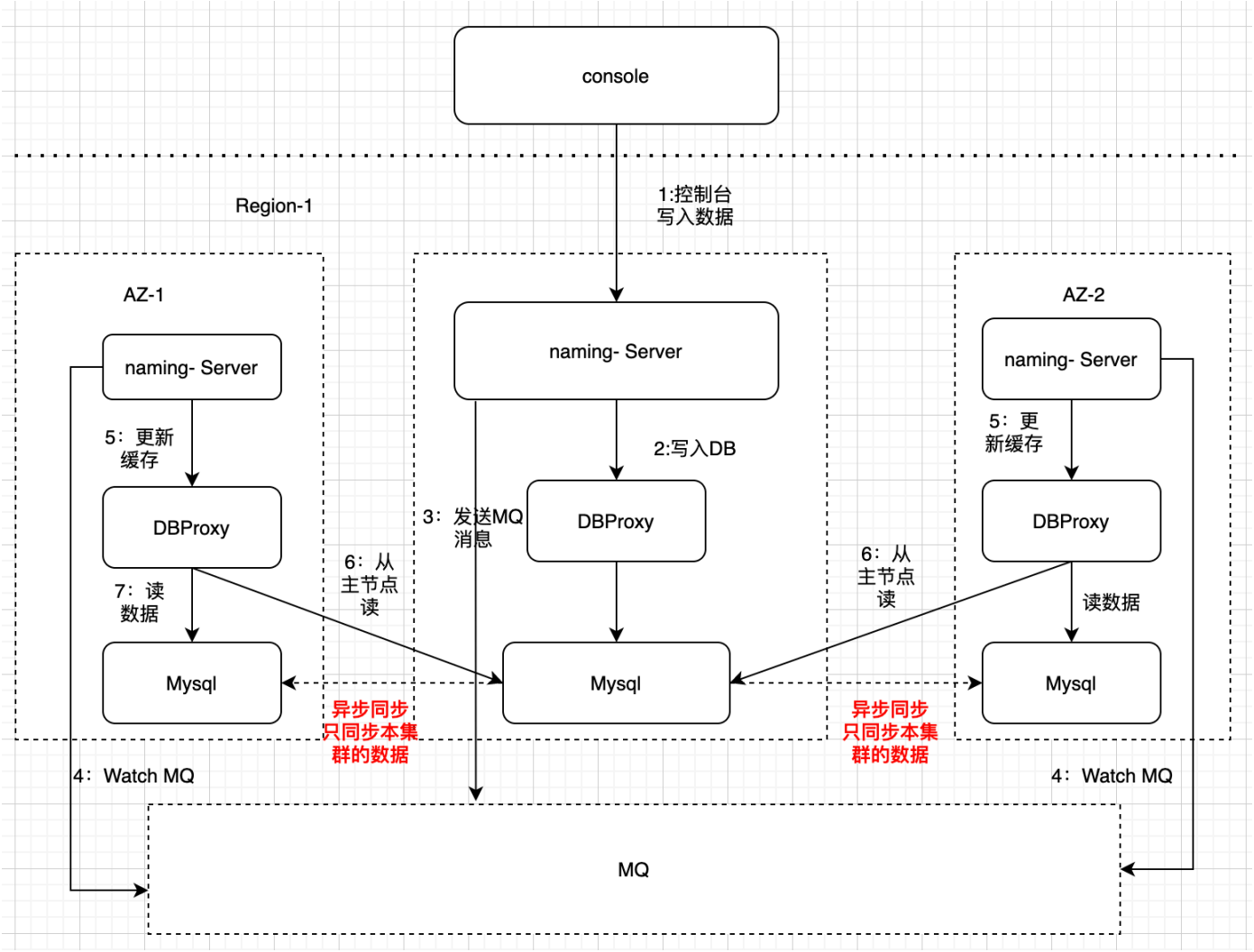
劣势：

1、需要做数据的对账

2、所有数据都在内存中

3、延迟不太好控制

中心双活方案2



优势：

1、两边机房的数据通过数据层来做数据同步

2、数据的延迟和一致性在数据层有比较好的保障

3、从职责上讲架构更优雅

劣势：

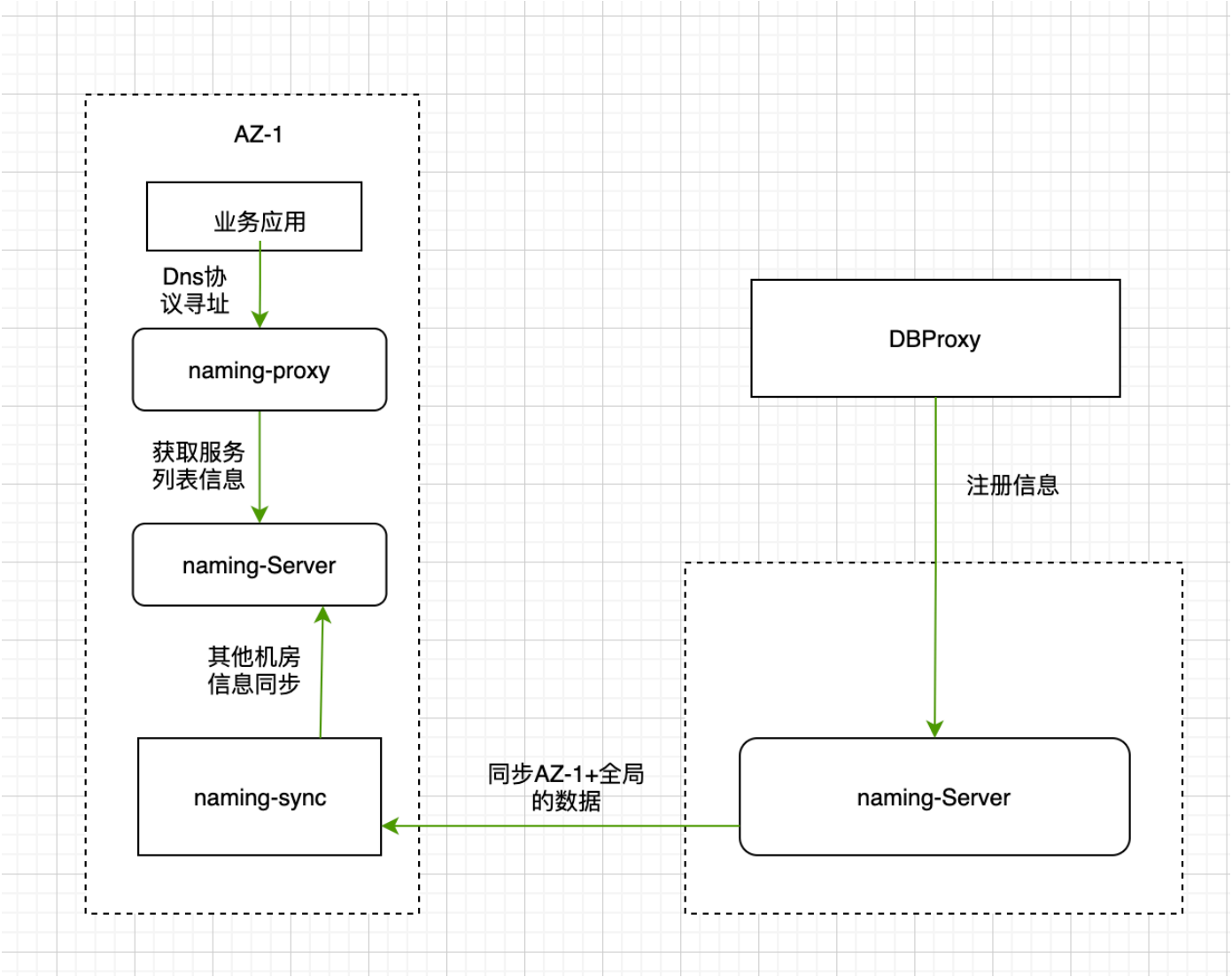
1、对数据层和MQ依赖比较严重

2、链路比较长，全链路监控成本高，排查问题也比较麻烦

3、数据层的高可用和名字服务形成双向依赖

DBProxy寻址

DBProxy的寻址是采用DNS协议来寻址；只有2级缓存（Naming-proxy和Naming-Server）



1、DBProxy动态注册到中心节点；

2、中心节点需要知道同步给哪个同步组件

需要提供给DBProxy的能力如下

https://wiki.zhiyinlou.com/pages/viewpage.action?pageId=142714858

1）注册

提供Open-Api的方式给数据库PAAS使用；

操作流程：PAAS平台把DBProxy发布之后，再通过Open-Api的方式把DBProxy的信息注册到名字服务中；后期名字服务会结合服务树来自动获取，不需要显示的注册

2）探活

制定统一的探活地址和策略（tcp）；

3）鉴权

提供Open-Api的方式给DBProxy使用

操作流程：

1）在控制台在DBProxy添加服务授权信息

2）业务方发起DBProxy的服务名查找的时候，先去名字服务获取DBProxy的服务列表，如果DBProxy服务没有对请求的服务授权就拒绝此次请求

3）DBProxy通过OpenApi的方式把授权过的服务列表获取下来，然后在订阅这些服务的实例列表信息；业务过来访问的时候，DBProxy可以根据ip地址，找到服务信息，然后在把服务信息和授权列表进行比对，是否可以通过。

4）负载均衡

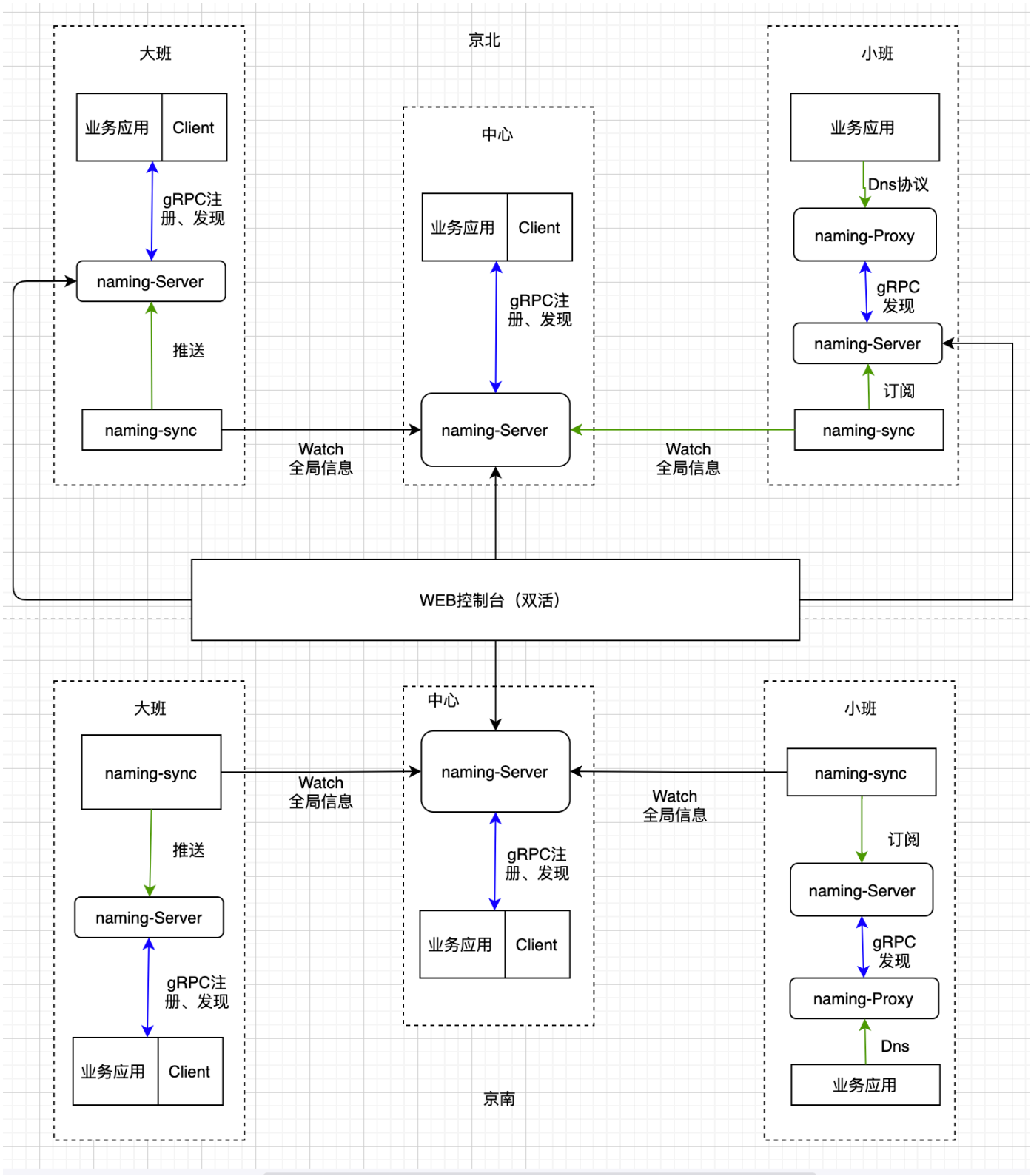
提供Open-Api的方式给DBProxy（权重的方式）

5）限流熔断

基于Mesh的控制面来实现；

未来的一些想法

单元化架构



1、业务的东西流量尽量在单元内形成闭环

2、单元内的同步组件需要Watch全局共享的服务列表信息

3、AZ内的Naming-Server和中心的Naming-Server采用同步组件的方式做松耦合的交互

4、通过Dns协议寻址有2级缓存，第一级是Proxy，第二级是Server

5、通过Client寻址有3级缓存：Client-》Server-》Proxy；Server挂掉之后，优先使用Client，Client也失效之后，使用Proxy

架构优化

- 解决名字服务单主架构写入瓶颈问题，升级成多主架构；
- 新增调度中心解决名字服务多集群的路由问题；
- 实现公有云的标准版架构（多个事业部混用一个大的物理集群）；

LocalDns

- 解决公共 DNS 节点位置影响域名解析准确性的问题；
- 解决内部使用公共 DNS 不稳定的问题；
- 优化内外网解析；

异构架构

- 解决K8s内访问Kvm内服务寻址和负载均衡问题；
- 优化现在通过域名方式进行寻址访问的问题，减少中间链路，提升效率；

协同方

1、IAAS团队：需要和IAAS团队一起共创，Kvm和Pod的网络互联互通，跨集群的Pod和Pod互通如何解决

2、PAAS团队：需要和PAAS团队一起共创，Kvm和K8s、以及中间件服务之间的相互寻址如何解决

3、SRE团队：需要和SRE团队一起共创，内网DNS服务如何搭建

4、监控团队：需要和监控团队一起共创，把名字服务的立体监控体系搭建起来

项目名称

毕方

项目组人员

孵化阶段

1）负载均衡（@赵宇(赵宇)+ @杨普光(杨普光)）

实现权重的即可

2）基于服务名的安全（@赵宇(赵宇)+@杨普光(杨普光)

3）DBProxy的状态管理（@秦强强(秦强强)）

包含：探活、状态管理、注册

4）Demo和演示方案（ @高兴亿(高兴亿) ）

包含：把Demo实现，以及负载均衡、安全、分级发布、止损这些功能的演示和证明这些功能生效了

验证阶段

服务端：秦强强、杨普光

客户端：赵宇、高兴亿

产品：待确认

前端：待确认

测试：待确认