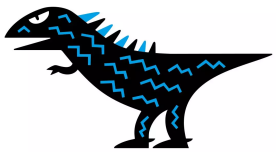
# Apollo在有赞的实践

原创 有赞技术 有赞coder 昨天



点击关注“有赞coder”

获取更多技术干货哦～

Apollo

A reliable configuration management system

作者：俞柯 & 张正

团队：有赞云

## 一. 背景和Apollo简介

在集中式开发时代，配置文件基本足够用了，因为那时配置的管理通常不会成为一个很大的问题，简单一点来说，系统上了生产之后，如果需要修改一个配置，登录到这台生产机器上，修改这个配置文件，然后reload配置文件并不是什么很大的负担。但是在互联网时代，我们的应用都是分布式系统，部署在N台机器上，如果在线上一台一台的重启机器，会造成很大的负担和不稳定。并且对于公司来说，会有多个环境区分（测试环境和线上环境），有时还需要对同一环境中的不同集群做不同的配置。因此需要一个配置中心来集中管理不同环境、不同集群的配置，修改配置后能够实时推送到应用端。

Apollo（阿波罗）是携程框架部门研发的分布式配置中心，能够集中化管理应用不同环境、不同集群的配置，配置修改后能够实时推送到应用端，并且具备规范的权限、流程治理等特性，适用于微服务配置管理场景。Apollo服务端基于Spring Boot和Spring Cloud开发，打包后可以直接运行，不需要额外安装Tomcat等应用容器。

## 二. Apollo在有赞的实践

### 2.1 部署方案

**多环境支持**

在有赞内部，环境的区分为:

- DAILY：日常环境（供开发自测的环境）
- QA：测试基准环境
- PRE：预发环境（上线前的验证）
- PROD：线上环境

在Apollo的抽象里，配置支持环境和集群粒度的隔离。其中环境的隔离是物理的隔离，不同环境是需要单独的数据库来支持的，集群则是逻辑隔离，同一环境的不同集群的数据库是共享的。这就导致一个问题，有赞需要四个环境的隔离，但是底层的RDS只支持三个环境。

如何在这种约束条件下支持四个环境的配置隔离，我们的做法是在PROD环境下创建PRE集群，虚拟为PRE环境，这个方案能够解决问题，但是会带来大量的兼容性成本。这里的兼容性成本包括：

1. client识别环境和机房的逻辑需要兼容
2. 控制台的UI需要做调整
3. 开放平台的sdk需要写兼容逻辑

而且，因为这种方案本质上破坏了环境和集群的抽象，原有的关于环境的特性以及集群的特性也丧失了，比如虚拟出来的PRE环境不能创建集群，PRE环境的应用会因为集群的降级特性读取线上环境的配置。

总结来说，多环境的支持是Apollo在有赞的实践做的最不好的地方，之所以会这样，根本原因在于对Apollo的抽象没有理解清晰，所以出现了破坏抽象的定制。

跟ops系统深度集成除了动态配置，Apollo作为配置中心的另一个重要的特性是，配置的中心化管理，将业务配置跟非业务配置隔离开来。这样隔离的好处在于，业务开发不需要关心框架和中间件的配置了，框架和中间件也能够更容易的管理各自的配置。Apollo可以解决其他中间件的配置管理问题，那Apollo自身的配置管理要怎么解决呢？我们的解决方案是通过ops系统来管理。

下面是Apollo相关的配置，通过运维系统写到每个机器上，通过读取这个文件，可以识别到当前所在的环境、机房以及其他的信息。Apollo-client就是通过读取下面的信息来识别相关信息的。

- APPLICATION-STANDARD-ENV=qa
- APPLICATION-SERVICE-CHAIN=qa
- APPLICATION_NAME=carmen-console-ng
- APPLICATION_IDC=qabb
- APOLLO_METASERVER=apollo-metaserver-qa.s.qima-inc.co

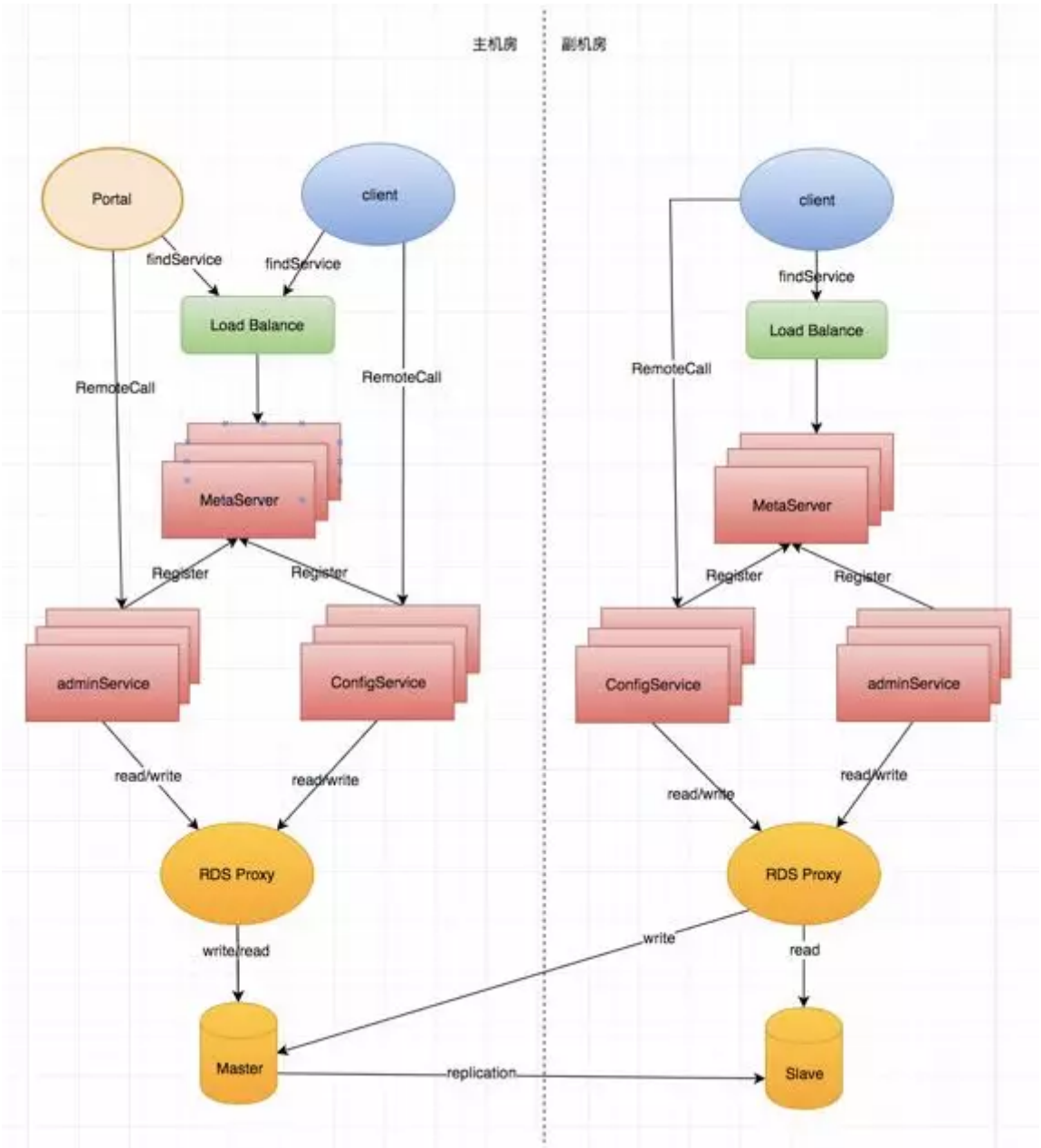
方案的好处在于，能够减少业务方错误配置带来的答疑量。

**双机房支持**为了带来最高的稳定性，有赞线上核心应用都是双机房部署，双机房部署能够避免单点问题，增强稳定性。Apollo作为核心的组建，也需要支持双机房部署。

双机房部署要解决的主要问题是，数据如何在两个机房间同步，因为Apollo底层使用mysql存储配置数据，所以这个问题就变为不同机房的mysql数据库如何进行数据的同步，以及某个节点不可用的情况下如何切换。

这里有必要对有赞的RDS系统做一个介绍，RDS能够自动实现mysql的主从切换，应用通过RDS proxy来跟server交互，默认情况下，RDS proxy会将写流量路由到master，读流量按照配置进行路由。在master节点故障的时候，RDS可以自动实现主从切换，并将写流量路由到新的master节点。

**双机房部署图如下**:



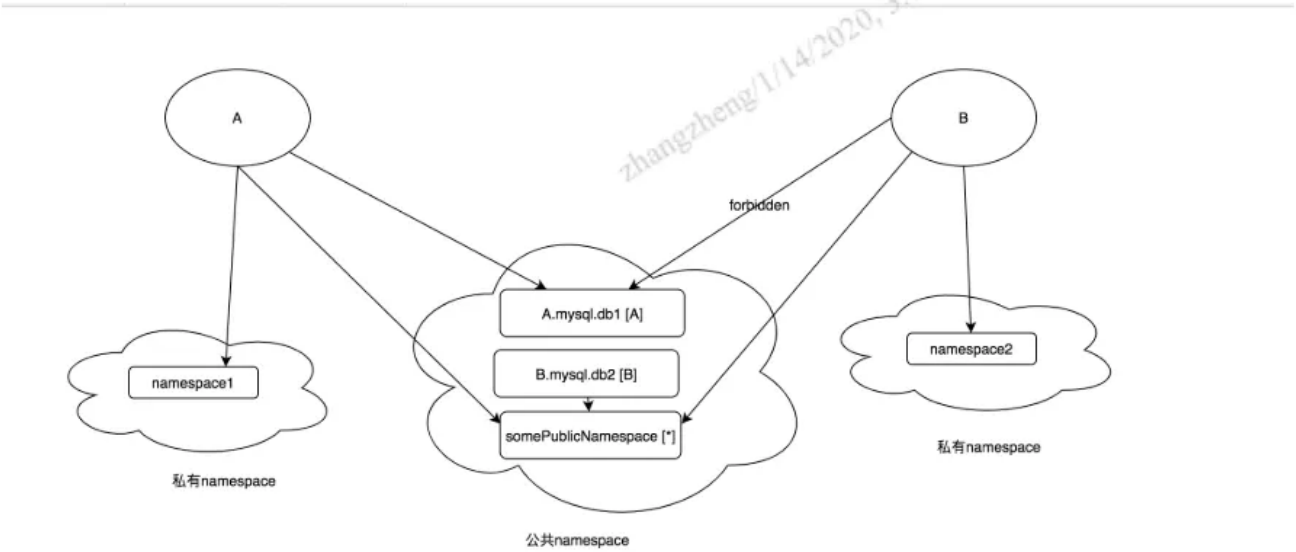
## 2.2 上云

有赞不仅仅作为一家SAAS公司，也涉及云的业务，具体说来，就是有赞会将核心业务沉淀为中台，中台暴露扩展点，外部开发者可以使用这些扩展点来定制自己的SAAS软件。外部开发者的应用通常托管在有赞的SAAS云上，这类云上的应用也有动态配置的需要，Apollo上云正是为了满足这个需求。

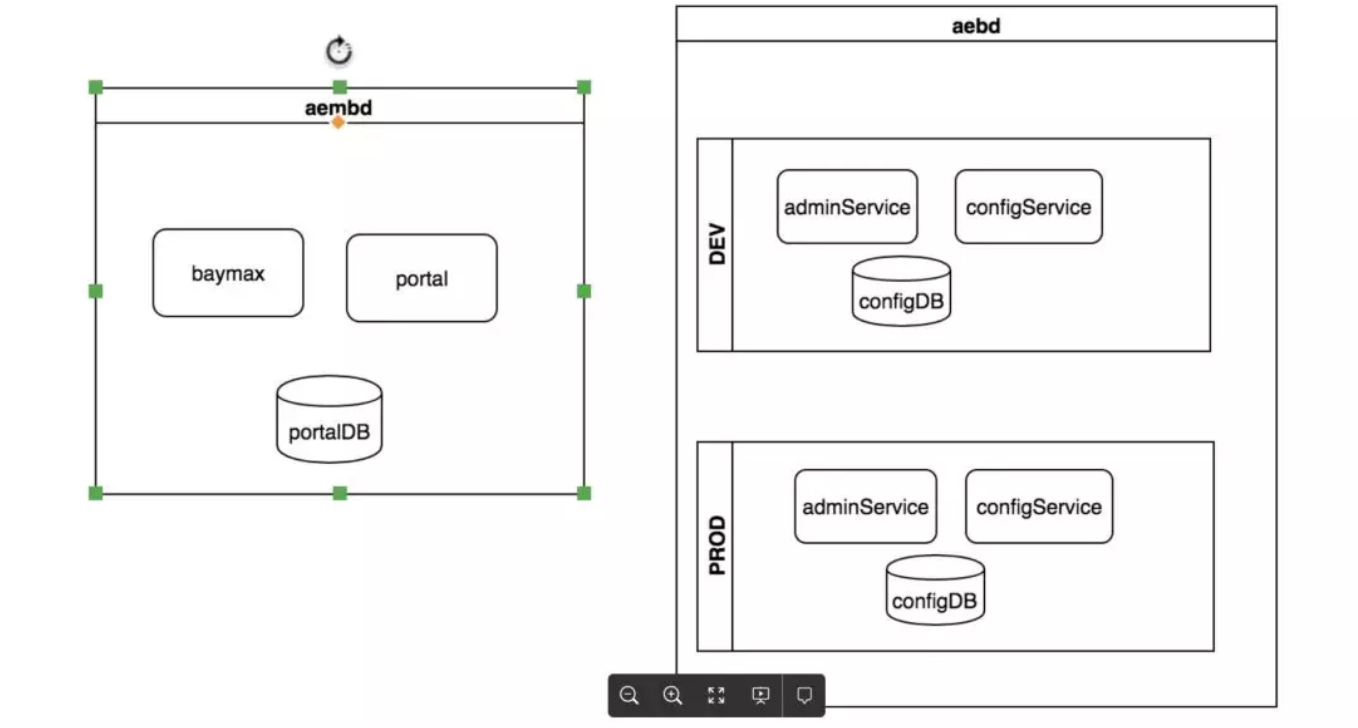
为了满足上云的需要，Apollo需要解决安全性问题，开源的版本缺乏对安全性方面的管控，具体体现在：

1. 任何应用都可以通过伪造应用名的方式来访问其他应用的配置
2. 任何应用都可以访问公共的配置，而某些公共的配置只能被特定的几个应用访问

我们给出的解决方案是，给应用颁发凭证，Apollo server会校验凭证以验证应用的身份。同时扩展Namespace的模型，增加scope属性，用来限制该Namespace可被访问的应用列表，比如Namespace的scope为 ["A","B"],表明这个Namespace可以被A应用和B应用访问。




除了安全性问题，Apollo上云还需要解决另外一个问题，如何在一个环境中部署多个Apollo环境？因为云上的Apollo面向的是开发者，开发者使用的是有赞线上的环境，然而对于开发者应用，也是需要区分测试环境和线上环境的。这就带来一个需求，就是Apollo需要在线上环境支持开发者的测试环境和线上环境的配置隔离。下面展示了线上环境，Apollo的部署情况：




如上图所示，portal部署在aembd机房，configservice和adminservice在线上环境部署两个服务（由于公司发布系统的限制，一个应用在一个环境只能部署一个服务，所以要实现这种部署的话，是需要给adminservice和configservice分别申请两个应用的），分别作为开发者的DEV环境和PROD环境。baymax是面向开发者的控制台，portal并不直接面向开发者。这种实现方案带来了很大的运维成本，体现在需要额外申请多个应用和数据库，后续数据库DDL变更的成本更高。

其实Apollo的设计里面，是支持环境和集群两个纬度的配置隔离的，所以针对这种需求场景，是可以使用集群隔离的特性的。两个纬度的配置隔离的特性也体现出Apollo设计人员的前瞻性，这里仅仅是其中一个例子，另外一个例子是资源配置的托管，这个放在后续再做介绍。

## 2.3 中间件配置托管

Apollo的核心抽象Namespace，能够解决多个应用共享配置的需求，能够为这类多个应用共享的配置提供统一管理的入口。在有赞内部，各个中间件和框架的配置，都是由Apollo来集中管理的，比如dubbo、分布式锁、调用链等等。集中托管能够带来很多的好处，比如减少业务方的配置成本以及因为配置错误引起的答疑量， 便于后续对配置的变更。

这里可以举一个例子，大家知道现在service-mesh的概念很火，service-mesh能够很好的解决多语言调用的问题，而有赞内部除了java以外，node也是一个主要的开发语言。所以公司开发了一套service-mesh组件，并且急需要将java语言的rpc框架替换成service-mesh的解决方案。这种例子中，rpc框架迁移到service-mesh只需要修改rpc框架的某些配置项，因为rpc框架的配置是集中管理的，

所以修改很容易。值得一提的是，Apollo对于这种修改场景，为了保证稳定性，还提供了强大的灰度特性。

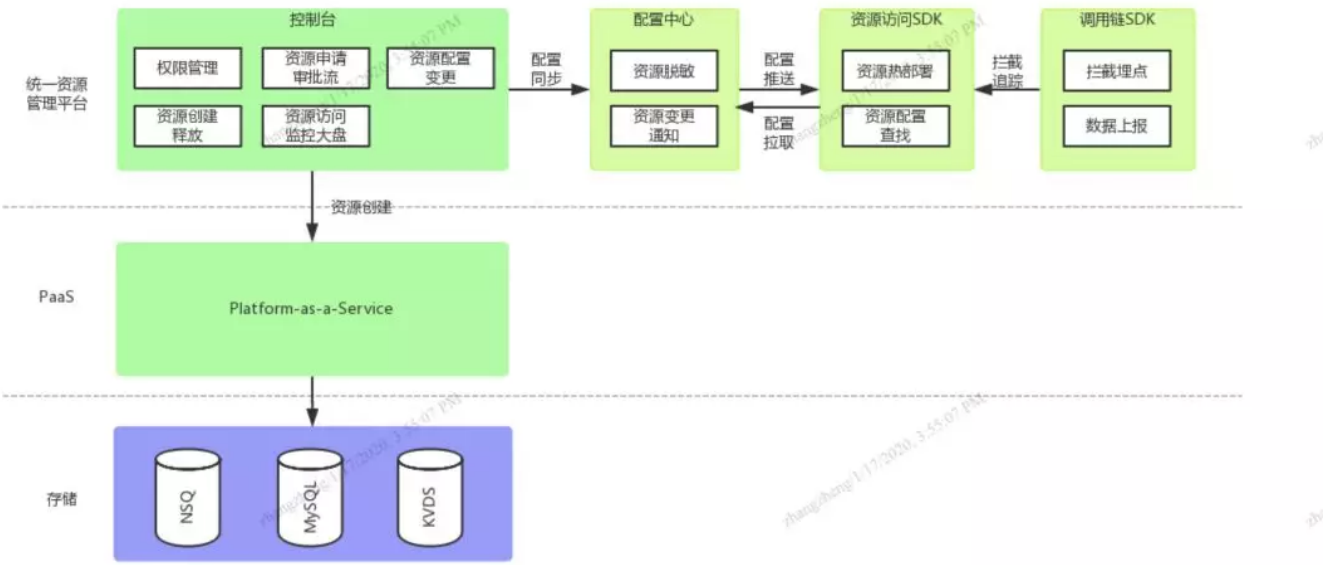
## 2.4 资源配置托管

资源的概念可以理解为应用运行时依赖的基础组件，比如数据库、消息中间件、缓存系统等等。一个完整的运维流程包括，资源的申请、资源的配置、资源的监控、资源的回收等等。

在使用Apollo托管资源配置之前，有赞的资源配置是托管在另外一个静态配置系统的，还有另外相当大的一部分是脱离管控的，散落在应用代码中。在公司的静态配置系统中，应用对资源配置的引用是通过复制的方式，而非引用的方式，对于资源的管理者，看不到使用该资源配置的应用，对资源配置的变更也需要推动业务方去修改应用的拷贝，对于散落在应用代码中的配置，要推动改造就更加不可能了。

除了配置管理方面的问题，针对数据库的配置，有对用户隐藏的需求，直接把用户名、密码暴露出去，容易带来不可控的风险。解决这个问题的方法是使用统一资源名，业务方只需要感知统一资源名，配置中心将统一资源名跟具体的资源配置映射。

项目的整体架构图如下，在这个项目中，Apollo配置中心只是整个资源运维流程的一步，承担了资源配置的统一管理、配置脱敏、变更通知等重要功能。

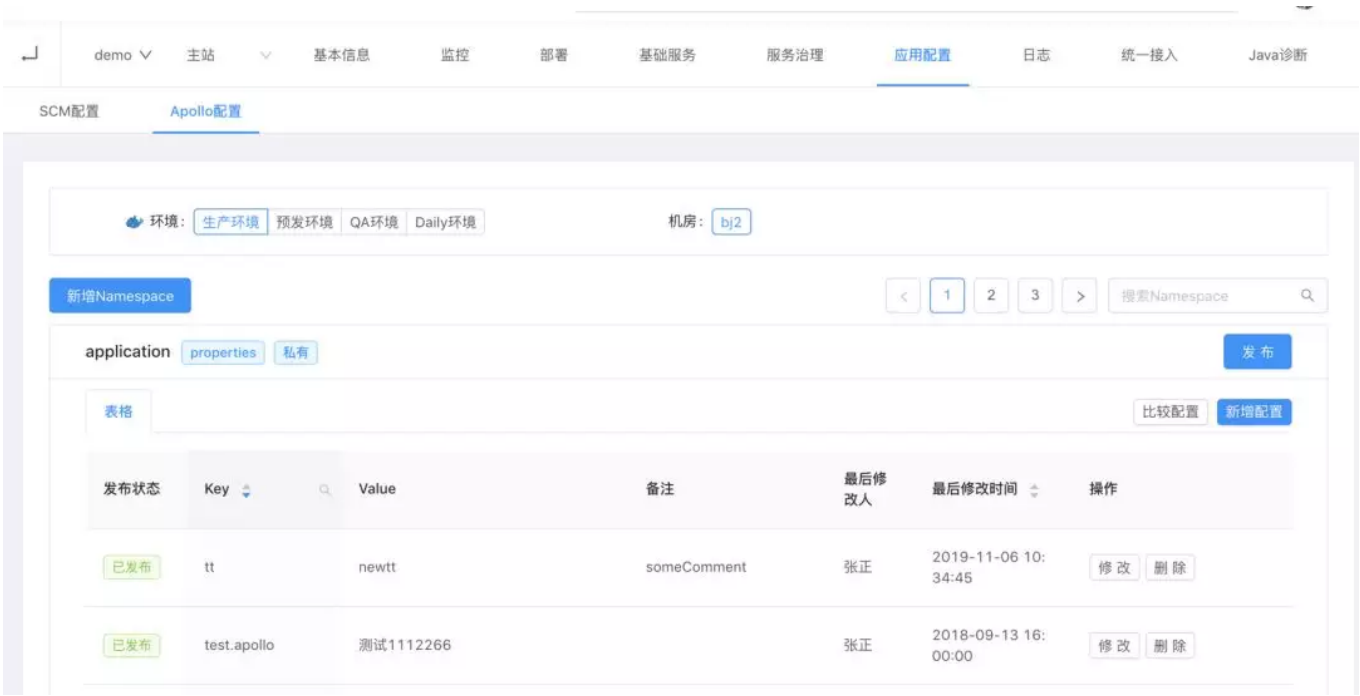



## 2.5 接入OPS系统提升产品体验

有赞有一套自己的ops系统，所有应用的管理、发布，中间件的申请，数据库权限申请等都在这个平台。所以如果把Apollo控制台的功能迁移统一维护的ops系统，可以大大的增加维护和管理，提升用户体验，并且有助于后续的继续迭代。但是如果需要挨个对接之前Apollo的接口，需要很多的工作

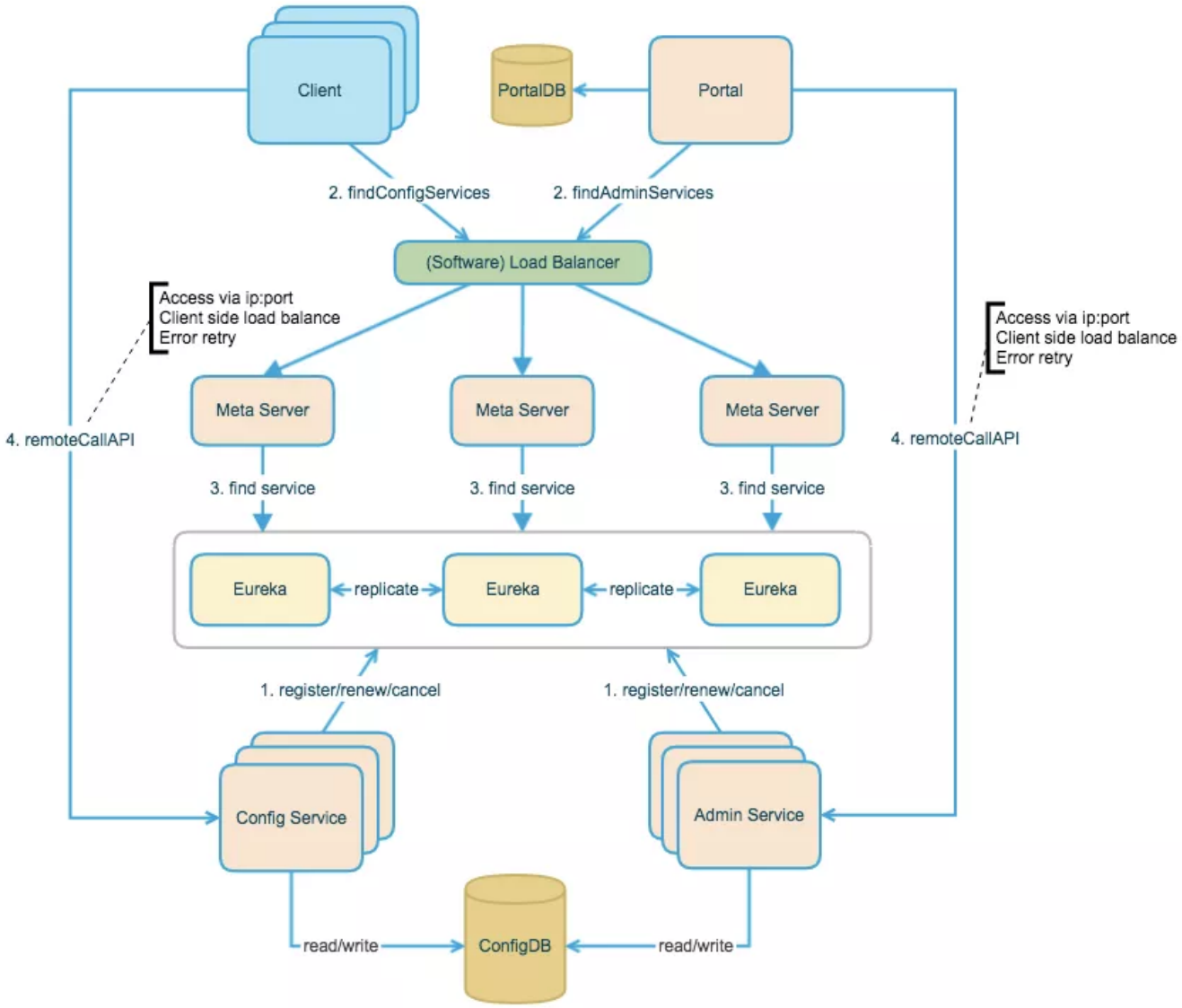
量，而且传递的参数也有可能因为不一致导致抛错。除此之外，对于有赞线上不同机房的部署，希望能在ops统一展示不同机房的名称，而Apollo默认就是default集群。

为了解决这个问题，我们在Apollo之前加了一层代理（Apollo-ops），ops系统所有的请求都会发到Apollo-ops，再由Apollo-ops统一转换成Apollo的http请求报文格式，获取请求结果。对于特殊的请求和新增操作Apollo的接口，可以Apollo-ops添加接口，这样可以减少对Apollo源码的侵入。ops控制台界面如图所示：




# 三. Apollo架构与设计

## 3.1 Apollo架构图




上图简要描述了 Apollo 的总体设计，从下往上看：

- Config Service 提供配置的读取、推送等功能，服务对象是Apollo客户端
- Admin Service 提供配置的修改、发布等功能，服务对象是Apollo Portal（管理界面）
- 通过Apollo的发布界面可以多环境、集群管理配置
- Config Service 和 Admin Service 都是多实例、无状态部署，所以需要将自己注册到
- Eureka 中并保持心跳，在 Eureka 之上架了一层 Meta Server 用于封装 Eureka 的服务发现接口。Apollo提供了MetaServiceProvider SPI，用户可以注入自己的MetaServiceProvider来自定义Meta Server定位逻辑
- Client 通过域名访问Meta Server获取Config Service服务列表（IP+Port），而后直接通过IP+Port访问服务，同时在 Client 侧会做 load balance、错误重试

为了简化部署，我们实际上会把Config Service、Eureka和Meta Server三个逻辑角色部署在同一个JVM进程中。

## 3.2 Apollo核心设计

### 3.2.1 Namespace设计

Namespace是配置项的集合，类似于一个配置文件的概念。Namespace的获取权限分为两种：private（私有的）和public（公共的）。private权限的Namespace，只能被所属的应用获取到。一个应用尝试获取其它应用private的Namespace，Apollo会报“404”异常。public权限的Namespace，能被任何应用获取到。

Namespace类型有三种：

1. 私有类型
2. 公共类型
3. 关联类型

私有的Namespace具有private权限，默认的“application”的Namespace就是私有类型的。

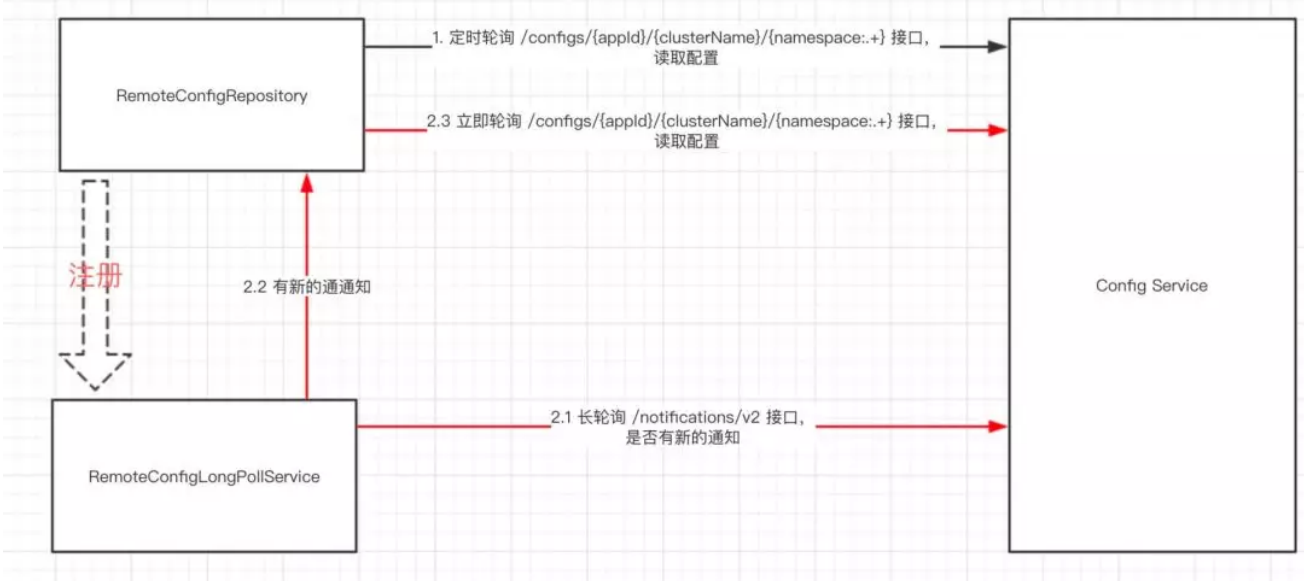
公共类型的Namespace具有public权限。公共类型的Namespace相当于游离于应用之外的配置，且通过Namespace的名称去标志公共Namespace，所以公共的Namespace的名称必须全局唯一。

如果在不同部分需要共享配置获取中间件客户端需要共享时，可以使用公共类型的Namespace。关联类型又可以称为继承类型，关联类型具有private权限。关联类型的Namespace继承于公共类型的Namespace，用于覆盖公共的Namespace的某些配置项。

### 3.2.2 Client端实现

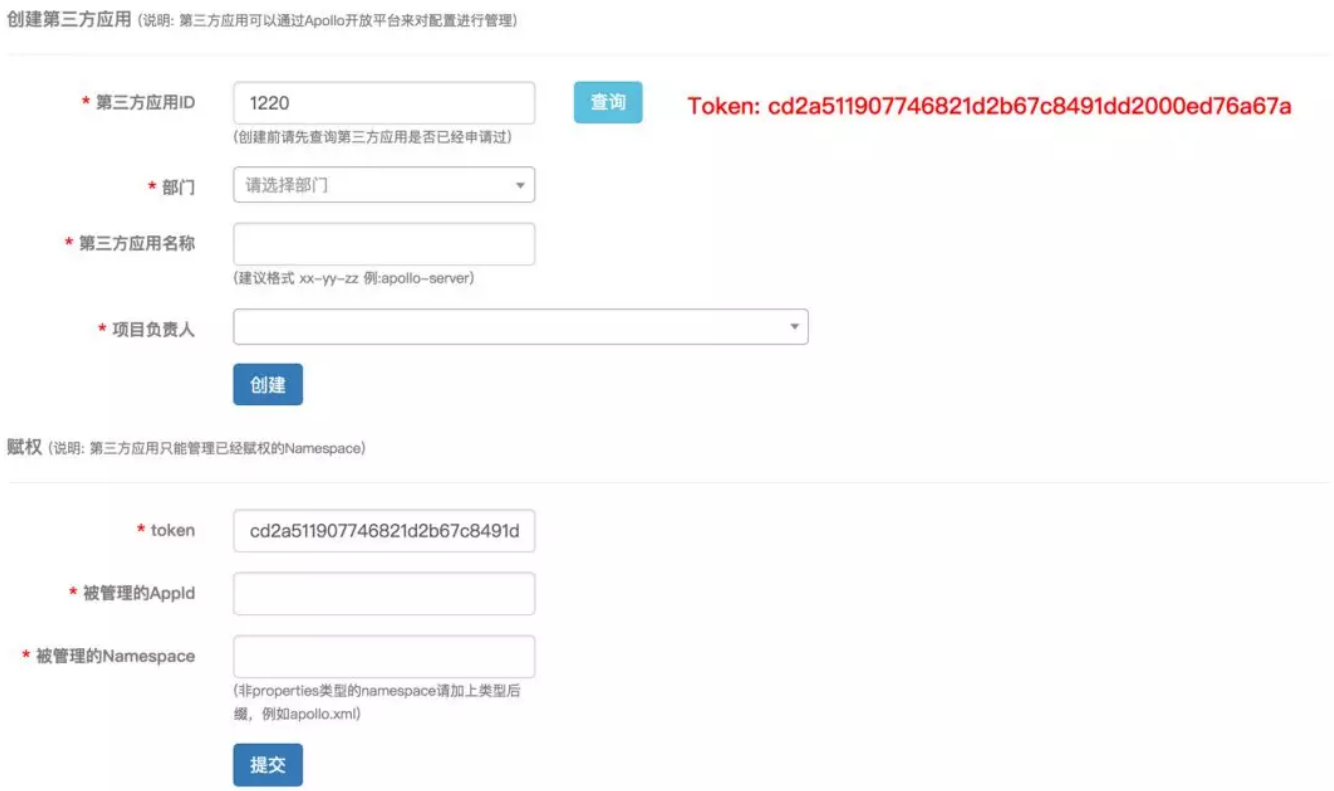
Client通过轮询的方式，从Config Service读取配置。Client的轮询包含两部分：

1. RemoteConfigRepository，定时轮询Config Service的配置读取/configs/{appId}/{clusterName}/{namespace:.+}接口。
2. RemoteConfigLongPollService，长轮询Config Service的配置变更通知/notifications/v2接口。当有新的通知时，触发RemoteConfigRepository，立即轮询Config Service的配置读取/configs/{appId}/{clusterName}/{namespace:.+}接口。整体流程如图所示：




### 3.2.3 Apollo开放平台

Apollo提供了一套的Http REST接口，使第三方应用能够管理配置。虽然Apollo系统本身提供了Portal来管理配置，但是在一些特殊的场景下，需要程序自己去管理。第三方应用负责人需要提供一些第三方应用基本信息来生成一个Token。



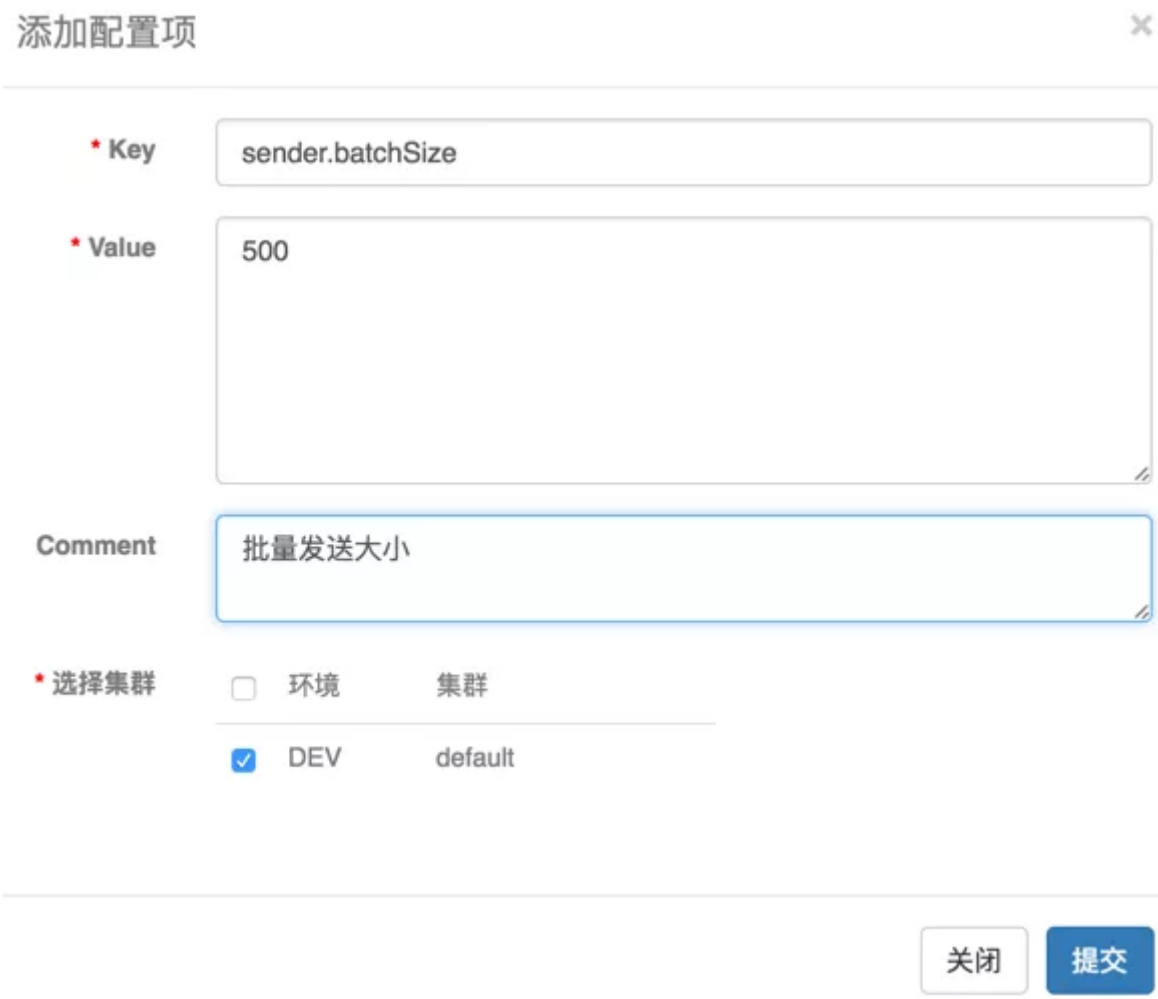

Token获取成功后，可以给Token绑定可以操作的特定Namespace，或者直接赋予整个应用下所有Namespace的权限。


## 3.3 Apollo控制台

在有赞，Apollo分为4个环境，分别是daily、qa、pre、prod，在不同环境下可以分别创建不同的集群，在不同集群下可以创建3中类型的Namespace（私有、公共、关联）。

Namespace名称全局唯一，创建需要项目管理员的权限，创建页面如下：


成功创建Namespace后，可以点击新增配置来创建配置项，创建完后，提交配置项：




配置只要在发布后才会真的被应用使用到，所以在编辑完配置后，需要发布配置。


## 四. 未来展望

Apollo已经是一套很成熟的开源配置管理中心软件，但是由于技术的更新，在一些技术细节上能有更好的优化。

1. **webSocket替代http长轮询**：Apollo在获取配置信息时，会发起一个长轮询，即客户端发送一个超时时间很长的Request，服务器hold住这个连接（Apollo默认是30s），在有新数据达到时返回Response。http1.1的连接默认使用长连接，虽然长连接在一个TCP连接上可以传输多个Request/Response消息对，但是本质上还是会造成资源的浪费、实时性不强等问题。而webSocket只需要建立一次Request/Response消息对，之后都是TCP连接，避免了需要多次建立Request/Response消息对而产生的冗余头部信息。当Apollo配置被修改后，服务端可以通知客户端，客户端再来获取最新配置，整个流程可以在一个webSocket中进行。
2. **配置中心统一**：将公司的静态配置中心和动态配置中心融合起来，使用apollo替换scm的配置中心，这样做有两个好处，一是apollo相对于公司的静态配置中心，提供了更强大的配置集中管理功能，可以提升基础组件的配置管理能力。另一个原因是减少业务方的使用成本和开发的维护成本。
3. **用ETCD替换Eureka注册中心**：这个规划主要考虑到两点原因，一是将注册中心的功能从configservice中解耦出来，解耦出来之后configservice就可以切换到容器中，运维成本降低。另一个原因是因为ETCD是公司核心的注册中心组件，稳定性更高。

## 相关链接

1. 有赞环境解决方案
2. 有赞云如何支持多语言
3. 电商云应用框架
4. 电商云BEP 网关详解
5. 有赞 Bond 分布式锁