### **一、介绍**

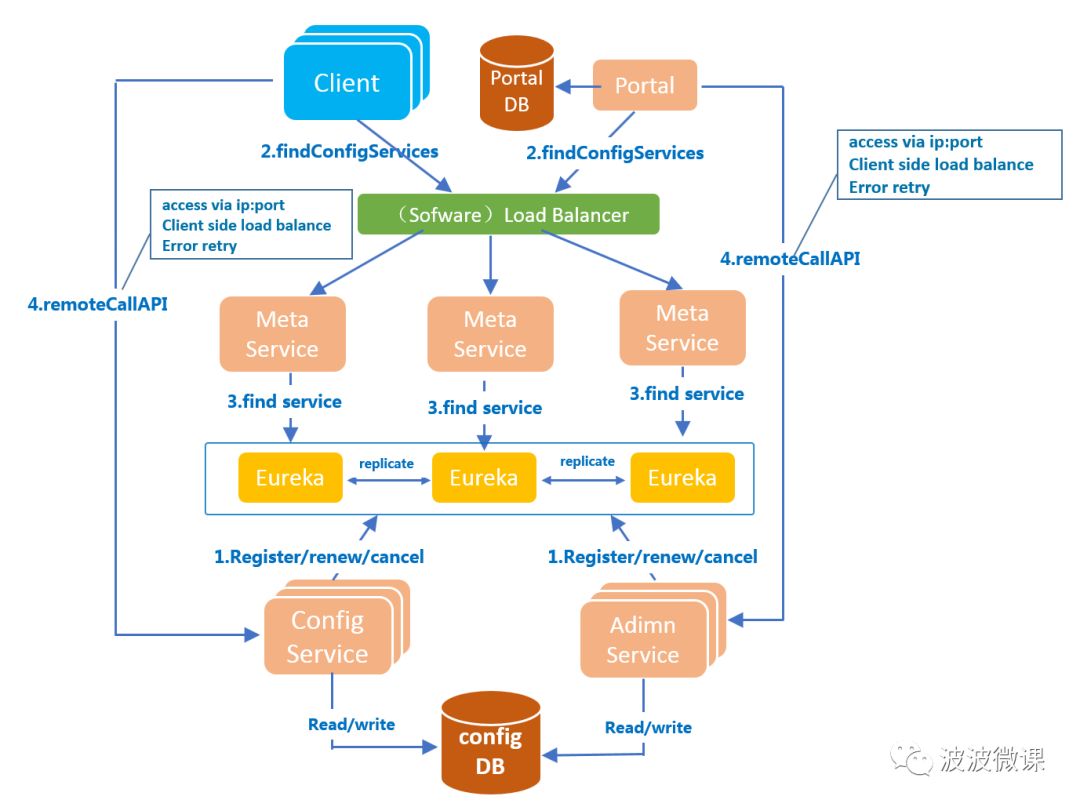
Apollo（阿波罗）[参考附录1]是携程框架部研发并开源的一款生产级的配置中心产品，它能够集中管理应用在不同环境、不同集群的配置，配置修改后能够实时推送到应用端，并且具备规范的权限、流程治理等特性，适用于微服务配置管理场景。

Apollo目前在国内开发者社区比较热，在Github上有超过5k颗星，在国内众多互联网公司有落地案例，可以说Apollo是目前配置中心产品领域Number1的产品，其成熟度和企业级特性要远远强于Spring Cloud体系中的Spring Cloud Config产品。

Apollo采用分布式微服务架构，它的架构有一点复杂，Apollo的作者宋顺虽然给出了一个架构图，但是如果没有一定的分布式微服务架构基础的话，则普通的开发人员甚至是架构师也很难一下子理解。为了让大家更好的理解Apollo的架构设计，我花了一点时间把Apollo的架构按我的方式重新剖析了一把。只有完全理解了Apollo的架构，大家才能在生产实践中更好的部署和使用Apollo。另外，通过学习Apollo的架构，大家可以深入理解微服务架构的一些基本原理。

### **二、架构和模块**

下图是Apollo的作者宋顺给出的架构图：



Apollo架构图by宋顺

如果没有足够的分布式微服务架构的基础，对携程的一些框架产品(比如Software Load Balancer(SLB))不了解的话，那么这个架构图第一眼看是不太好理解的(其实我第一次看到这个架构也没有看明白)。在这里我们先放一下，等我后面把这个架构再重新剖析一把以后，大家再回过头来看这个架构就容易理解了。

下面是Apollo的七个模块，其中四个模块是和功能相关的核心模块，另外三个模块是辅助服务发现的模块：

#### **四个核心模块及其主要功能**

1. ****ConfigService****

* 提供配置获取接口
* 提供配置推送接口
* 服务于Apollo客户端

1. ****AdminService****

* 提供配置管理接口
* 提供配置修改发布接口
* 服务于管理界面Portal

1. ****Client****

* 为应用获取配置，支持实时更新
* 通过MetaServer获取ConfigService的服务列表
* 使用客户端软负载SLB方式调用ConfigService

1. ****Portal****

* 配置管理界面
* 通过MetaServer获取AdminService的服务列表
* 使用客户端软负载SLB方式调用AdminService

#### **三个辅助服务发现模块**

1. ****Eureka****

* 用于服务发现和注册
* Config/AdminService注册实例并定期报心跳
* 和ConfigService住在一起部署

1. ****MetaServer****

* Portal通过域名访问MetaServer获取AdminService的地址列表
* Client通过域名访问MetaServer获取ConfigService的地址列表
* 相当于一个Eureka Proxy
* 逻辑角色，和ConfigService住在一起部署

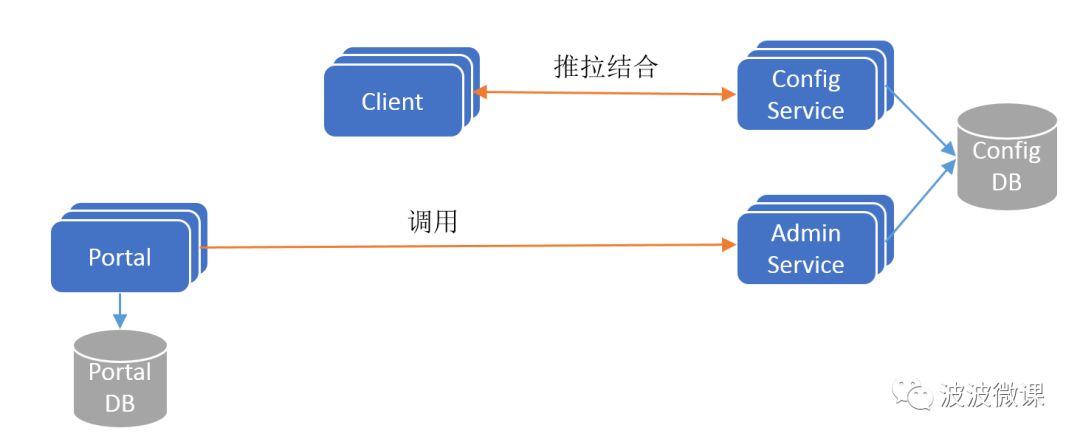
1. ****NginxLB****

* 和域名系统配合，协助Portal访问MetaServer获取AdminService地址列表
* 和域名系统配合，协助Client访问MetaServer获取ConfigService地址列表
* 和域名系统配合，协助用户访问Portal进行配置管理

### **三、架构剖析**

#### **Apollo架构V1**

如果不考虑分布式微服务架构中的服务发现问题，Apollo的最简架构如下图所示：



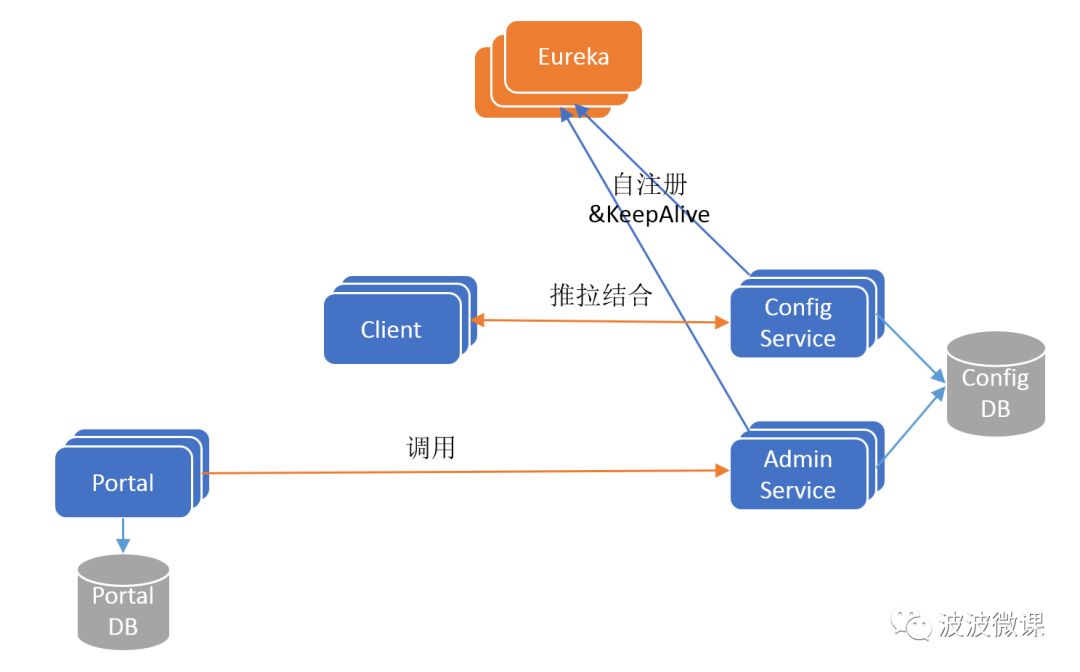
Apollo V1架构by波波

要点：

1. ConfigService是一个独立的微服务，服务于Client进行配置获取。
2. Client和ConfigService保持长连接，通过一种推拉结合(push & pull)的模式，在实现配置实时更新的同时，保证配置更新不丢失。
3. AdminService是一个独立的微服务，服务于Portal进行配置管理。Portal通过调用AdminService进行配置管理和发布。
4. ConfigService和AdminService共享ConfigDB，ConfigDB中存放项目在某个环境中的配置信息。ConfigService/AdminService/ConfigDB三者在每个环境(DEV/FAT/UAT/PRO)中都要部署一份。
5. Protal有一个独立的PortalDB，存放用户权限、项目和配置的元数据信息。Protal只需部署一份，它可以管理多套环境。

#### **Apollo架构V2**

为了保证高可用，ConfigService和AdminService都是无状态以集群方式部署的，这个时候就存在一个服务发现问题：Client怎么找到ConfigService？Portal怎么找到AdminService？为了解决这个问题，Apollo在其架构中引入了Eureka服务注册中心组件，实现微服务间的服务注册和发现，更新后的架构如下图所示：



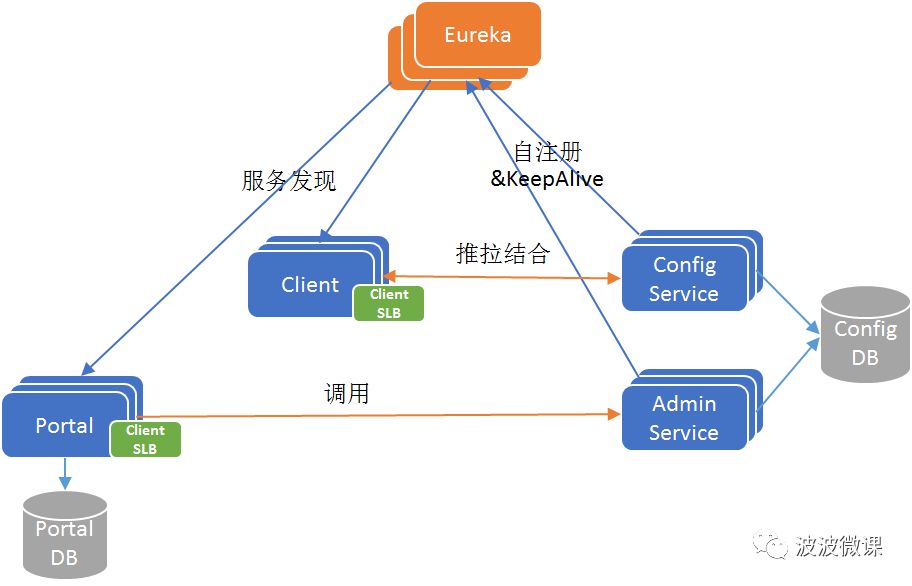
Apollo V2架构by波波

要点：

1. Config/AdminService启动后都会注册到Eureka服务注册中心，并定期发送保活心跳。
2. Eureka采用集群方式部署，使用分布式一致性协议保证每个实例的状态最终一致。

#### **Apollo架构V3**

我们知道Eureka是自带服务发现的Java客户端的，如果Apollo只支持Java客户端接入，不支持其它语言客户端接入的话，那么Client和Portal只需要引入Eureka的Java客户端，就可以实现服务发现功能。发现目标服务后，通过客户端软负载(SLB，例如Ribbon)就可以路由到目标服务实例。这是一个经典的微服务架构，基于Eureka实现服务注册发现+客户端Ribbon配合实现软路由，如下图所示：



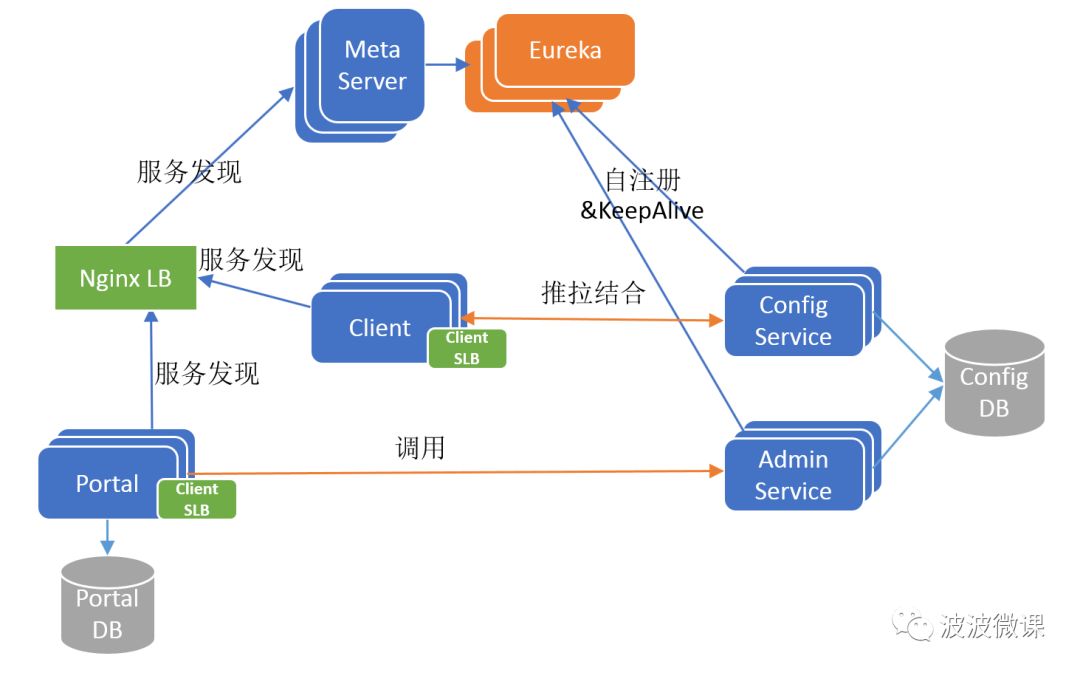
Apollo V3架构by波波

#### **Apollo架构V4**

在携程，应用场景不仅有Java，还有很多遗留的.Net应用。Apollo的作者也考虑到开源到社区以后，很多客户应用是非Java的。但是Eureka(包括Ribbon软负载)原生仅支持Java客户端，如果要为多语言开发Eureka/Ribbon客户端，这个工作量很大也不可控。为此，Apollo的作者引入了MetaServer这个角色，它其实是一个Eureka的Proxy，将Eureka的服务发现接口以更简单明确的HTTP接口的形式暴露出来，方便Client/Protal通过简单的HTTPClient就可以查询到Config/AdminService的地址列表。获取到服务实例地址列表之后，再以简单的客户端软负载(Client SLB)策略路由定位到目标实例，并发起调用。

现在还有一个问题，MetaServer本身也是无状态以集群方式部署的，那么Client/Protal该如何发现MetaServer呢？一种传统的做法是借助硬件或者软件负载均衡器，例如在携程采用的是扩展后的NginxLB（也称Software Load Balancer），由运维为MetaServer集群配置一个域名，指向NginxLB集群，NginxLB再对MetaServer进行负载均衡和流量转发。Client/Portal通过域名+NginxLB间接访问MetaServer集群。

引入MetaServer和NginxLB之后的架构如下图所示：

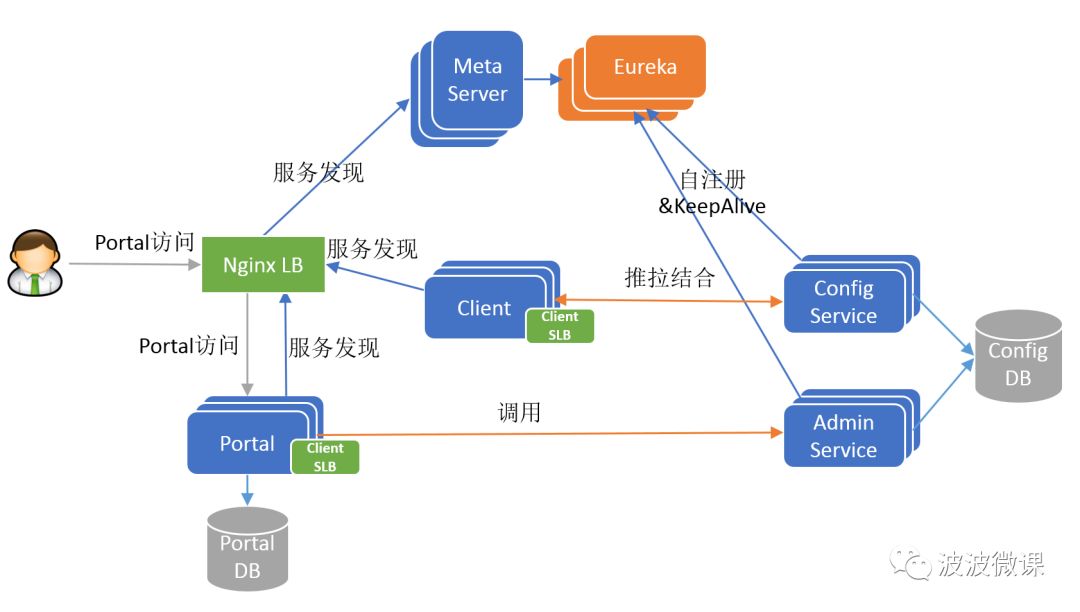


Apollo V4架构by波波

#### **Apollo架构V5**

V4版本已经是比较完整的Apollo架构全貌，现在还剩下最后一个环节：Portal也是无状态以集群方式部署的，用户如何发现和访问Portal？答案也是简单的传统做法，用户通过域名+NginxLB间接访问Portal集群。

所以V5版本是包括用户端的最终的Apollo架构全貌，如下图所示：



Apollo V5架构by波波

### **四、结论**

1. 经过我在第三部分的剖析之后，相信大家对Apollo的微服务架构会有更清晰的认识，作为一个思考题，大家再回头看一下第二部分宋顺给出的架构图，现在是否能够理解？它和波波的架构是如何对应的？提示一下，宋顺的视角是一个从上往下的俯视视角，而我的是一个侧面视角。

2. ConfgService/AdminService/Client/Portal是Apollo的四个核心微服务模块，相互协作完成配置中心业务功能，Eureka/MetaServer/NginxLB是辅助微服务之间进行服务发现的模块。

3. Apollo采用微服务架构设计，架构和部署都有一些复杂，但是每个服务职责单一，易于扩展。另外，Apollo只需要一套Portal就可以集中管理多套环境(DEV/FAT/UAT/PRO)中的配置，这个是它的架构的一大亮点。。

4. 服务发现是微服务架构的基础，在Apollo的微服务架构中，既采用Eureka注册中心式的服务发现，也采用NginxLB集中Proxy式的服务发现。

5. 波波近期和极客时间合作，推出《微服务架构实践160讲》课程，对微服务8大基础组件的架构和实践进行全面系统的剖析，其中第二个模块课程会专注剖析Apollo配置中心的架构和实践，欢迎大家关注。

### **五、附录**

https://github.com/ctripcorp/apollo