Dubbo Mesh hands-on

By 徐杨清

云原生蓬勃发展的过程

- Mesh 模式
- Proxy less 模式
- **Multi-Runtime**

架构演进趋势

- 重client -> 轻client

- 框架 -> Mesh　集群能力转移

- 控制面承载与其他中间件交互能力（类比于apiserver 屏蔽了etcd）

Dubbo 功能升级时的阻碍

Dubbo client 重逻辑，dubbo clinet 直接与各种中间件交互

导致依赖非常重

导致中间件选型发生调整需要升级client 的jar 版本

导致dubbo 的新特性，新能力很难运用于生产

Dubbo 3 规划

- 1.多语言支持　　->　多语言的 sdk 包 / mesh化的rest部署方案
- 2.依赖冲突严重　->　thin sdk，中间件交互交给控制面
- 3.网关穿透性差异构体系无法互通　->　基于http2的triple协议
- 4.集群管理不透明治理能力不友好 -> xds 支持/控制面抽出
- 5.可观测性差 -> dubbo traceing

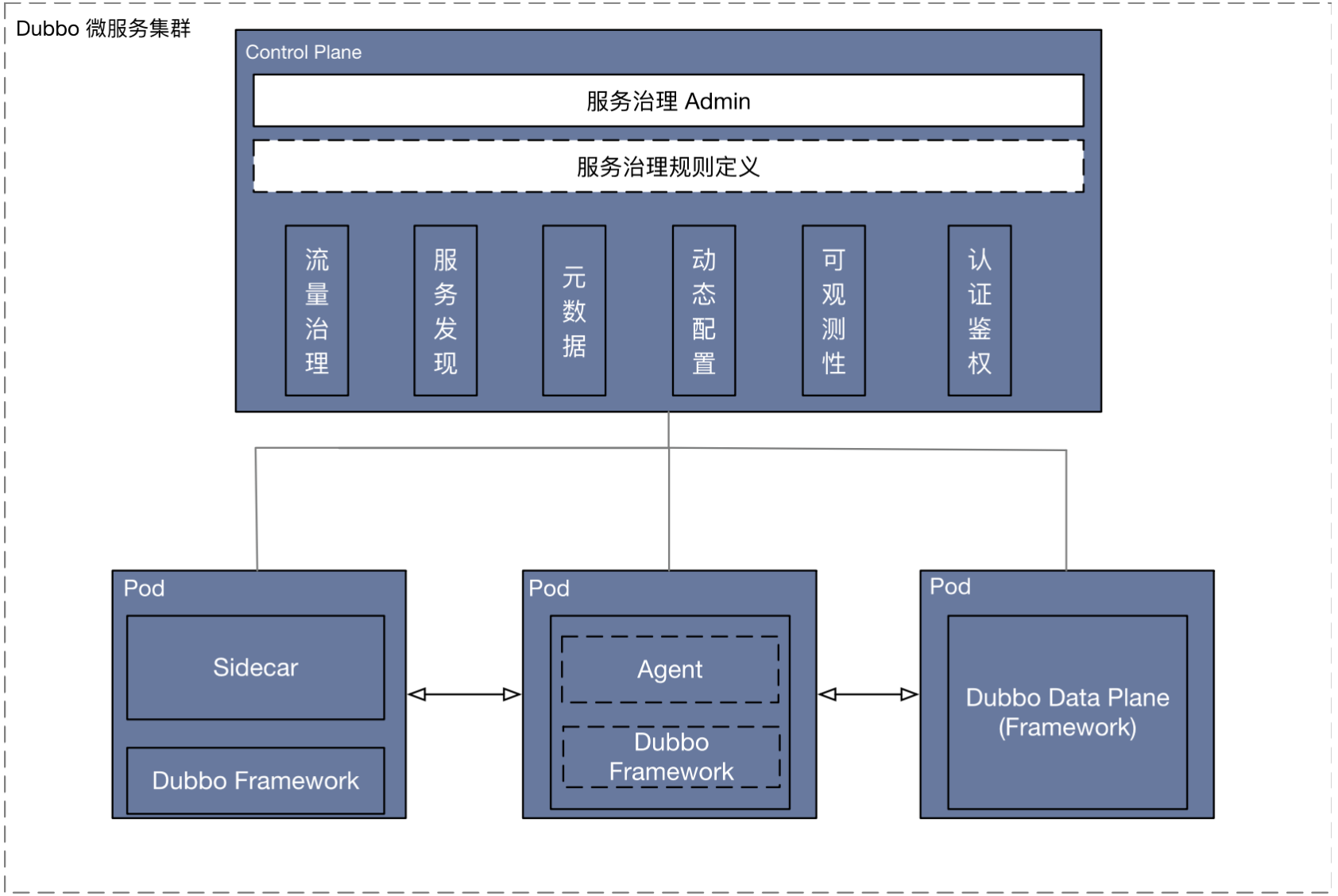
控制面数据面分离的设计

- 控制面
  专注于策略的决策，配置，控制器集群独立部署
  由控制面向其他中间件交互，允许依赖多种中间件client
  架构选型上需要替换中间件单独升级控制面就可以了

- 管控协议
  负责将控制面的控制指令，配置模型下发到数据面

- 数据面
  专注于策略的执行，只依赖协议相关模型，干净纯粹

诗和远方



目前dubbo thin sdk 已经成功抽出来了

距离理想的架构只差控制面了

- 本次demo使用istio 作为dubbo 的控制面采用proxy less的方式部署，需要提前安装Istio

Istio 安装

- 下载 istio 文件
- `$ curl -L https://istio.io/downloadIstio | sh -`

- 配置istiohome 等环境变量

echo ‘export ISTIO_HOME=下载目录' >> /etc/profile echo ‘
export PATH=$PATH:$ISTIO_HOME/bin' >> /etc/profile

istioctl install --set profile=demo –y --set values.global.jwtPolicy=first-party-jwt

服务提供方编码方式

- 基本同dubbo2，业务开发方几乎无感

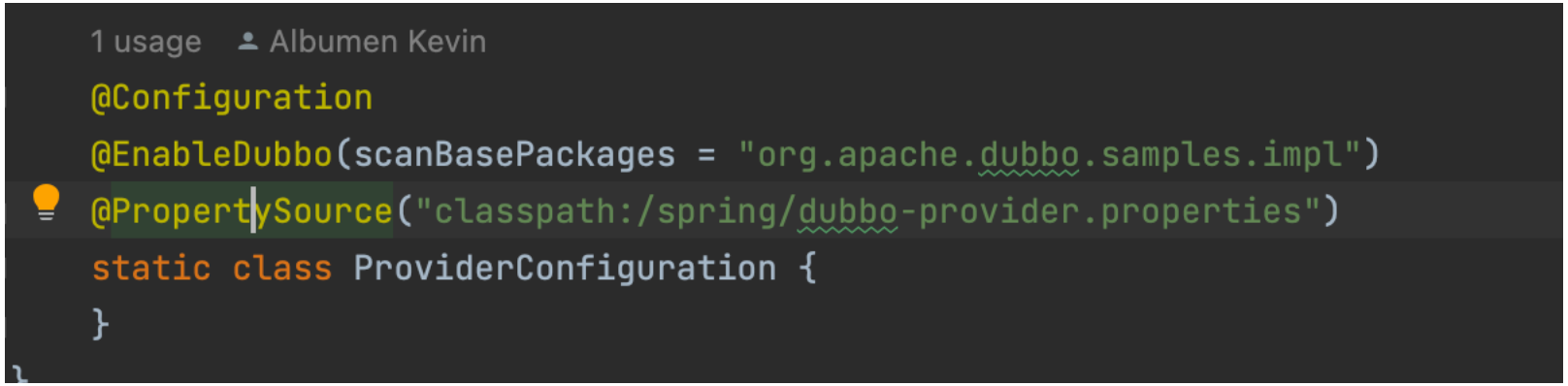
编译写想要暴露的接口

```
package org.apache.dubbo.samples.api2

public interface GreetingService {

    String sayHello(String name);

}
```

编写接口的实现

```java
package org.apache.dubbo.samples.impls;

import org.apache.dubbo.common.utils.NetUtils;
import org.apache.dubbo.config.annotation.DubboService;
import org.apache.dubbo.samples.api.GreetingService;

@DubboService(version = "1.0.0")
public class AnnotatedGreetingService implements GreetingService {

    @Override
    public String sayHello(String name) {
        System.out.println("greeting service received: " + name);
        return "hello, " + name + "! from host: " + NetUtils.getLocalHost();
    }

}
```

启用dubbo


1 usage
Albumen Kevin
@Configuration
@EnableDubbo(scanBasePackages = "org.apache.dubbo.samples.impl")
@PropertySource("classpath:/spring/dubbo-provider.properties")
static class ProviderConfiguration {
}

provider配置如下

- dubbo.application.name=dubbo-samples-xds-provider
  dubbo.application.metadataServicePort=20885
  dubbo.registry.address=xds://istiod.istio-system.svc:15012
  dubbo.protocol.name=tri
  dubbo.protocol.port=50051
  dubbo.application.qosEnable=true
  dubbo.application.qosAcceptForeignIp=true

因为部署方式是proxyless的关系，部分sidecar的职责需要dubbo client 承载，所以 resource controller 对pod 做存活检测以及Qos逐出用到的接口需要打开

服务消费方

- 对比dubbo2 下需要提供 providerby 信息，填写k8s下服务名

```
Albumen Kevin
@Component("annotatedConsumer")
public class GreetingServiceConsumer {

    1 usage
    @DubboReference(version = "1.0.0", providedBy = "dubbo-samples-xds-provider")
    private GreetingService greetingService;

    1 usage  Albumen Kevin
    public String doSayHello(String name) { return greetingService.sayHello(name); }

}
```

因为原生的XDS 协议，服务发现只做到了应用这个粒度（service）
应用 和 接口的映射关系目前只能由编码方维护
我们会推出更吻合dubbo2用户使用习惯的interface discover service来处理这部分映射关系

启用dubbo

```
1 usage  Albumen Kevin
@Configuration
@EnableDubbo(scanBasePackages = "org.apache.dubbo.samples.action")
@PropertySource("classpath:/spring/dubbo-consumer.properties")
@ComponentScan(value = {"org.apache.dubbo.samples.action"})
static class ConsumerConfiguration {

}
```

Consumer添加配置如下

- dubbo.application.name=dubbo-samples-xds-consumer
  dubbo.application.metadataServicePort=20885
  dubbo.registry.address=xds://istiod.istio-system.svc:15012
  dubbo.application.qosAcceptForeignIp=true

部署到K8S

- Maven打包
- mvn clean package –DskipTests

- 编写docker file 并打包镜像
- docker build -t apache/dubbo-demo:dubbo-samples-xds-provider_0.0.1 .

- 编写k8s 部署文件并部署到k8s
- 通过k8s 下service的概念做端口透出，就可以被使用方法使用了

关注 Dubbo 公众号，获得最新资讯