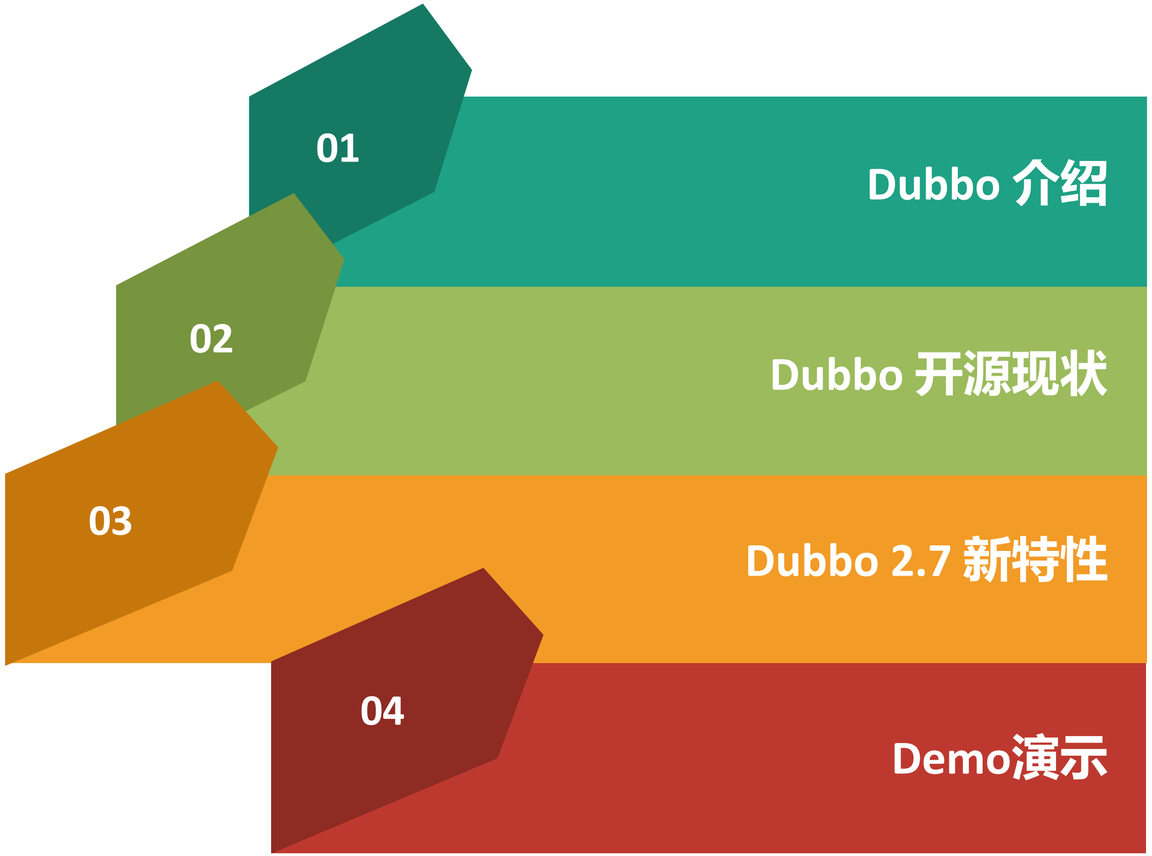
Apache Dubbo (Incubating)
Dubbo-2.7 新特性介绍

中间件技术部

曹胜利/展图

目录



Apache Dubbo介绍

Apache Dubbo **是一款高性能、轻量级的开源**Java RPC**框架。**

1 导出服务
服务提供方通过指定端口对外暴露服务

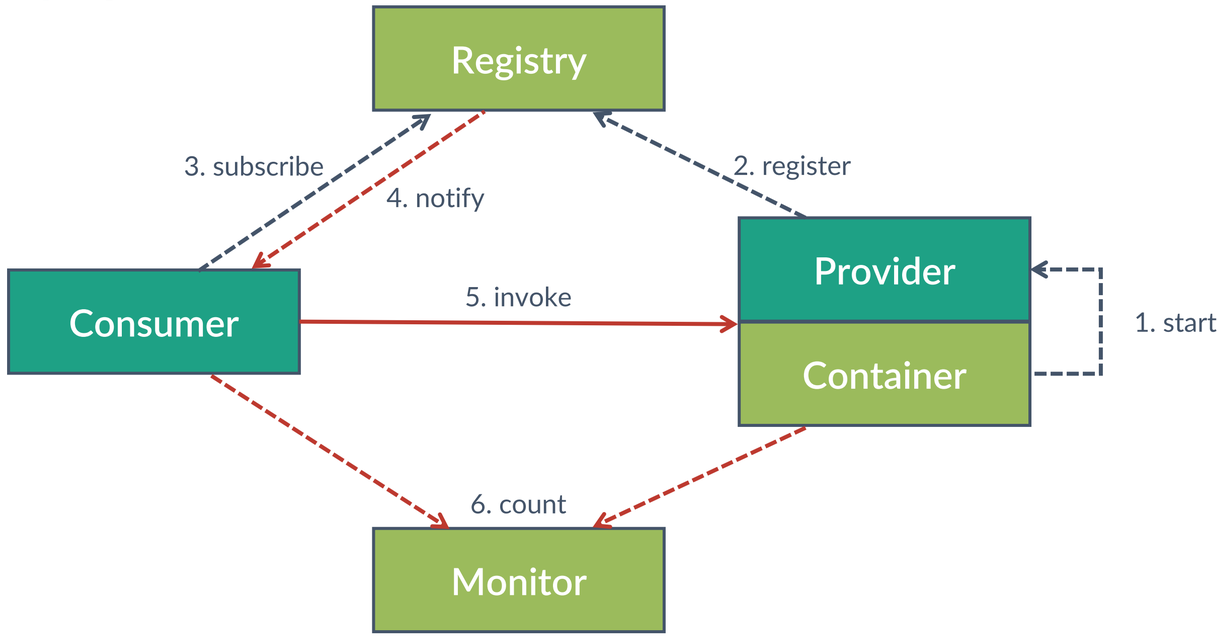
2 注册服务
提供方向注册中心注册自己的信息

3 订阅服务
服务调用方通过注册中心订阅自己感兴趣的服务

4 发现服务
注册中心向调用方推送地址列表

5 调用服务
调用方选择一个地址发起 RPC 调用

6 监控
服务提供方和调用方的统计数据由监控模块收集展示



开源推进

核心功能

三方库的版本更新 | 支持spring boot

核心功能迭代 | 2.6.X 和2.7版本并行

全新 Logo & 官网

文档 | 博客

活动 | 社区

生态系统

多语言客户端: Node, Python, PHP, GO

扩展: native Hessian, native Thrift, HTTP2

Spring Boot Dubbo Initializr

全新 Ops

拥抱Spring Boot | Admin与Monitor 融合

对接多种Registry | 2.6.X, 2.7.X版本兼容

用户交流

考拉，挖财，工行，浩鲸交流；

开源推进

请在 https://github.com/apache/incubator-dubbo/issues/1012 上提供您的信息

Dubbo 用户主要分为 3 类：互联网企业、向互联网架构转型的企业、用互联网架构做解决方案的企业。当当、去哪儿、微店、阿里巴巴作为 Dubbo 进 Apache 孵化的初始成员。

2.7版本新特性

Repackage

Apache的项目
org.apache.dubbo

JDK8

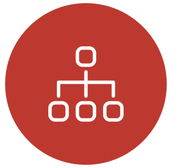
Default method
CompletableFuture

异步支持

支持更友好的使用方式
支持Provider端异步

元数据

简化注册中心配置项
丰富元数据信息



动态配置

远程配置
服务治理参数的配置

路由规则

与注册中心分离，使用配置中心存储
支持更多路由规则

JDK8 & Repackage

JDK8

- default method
- CompletableFuture
- Optional
- Lambda
- function

Apache Dubbo

- GroupId/Package: org.apache.dubbo

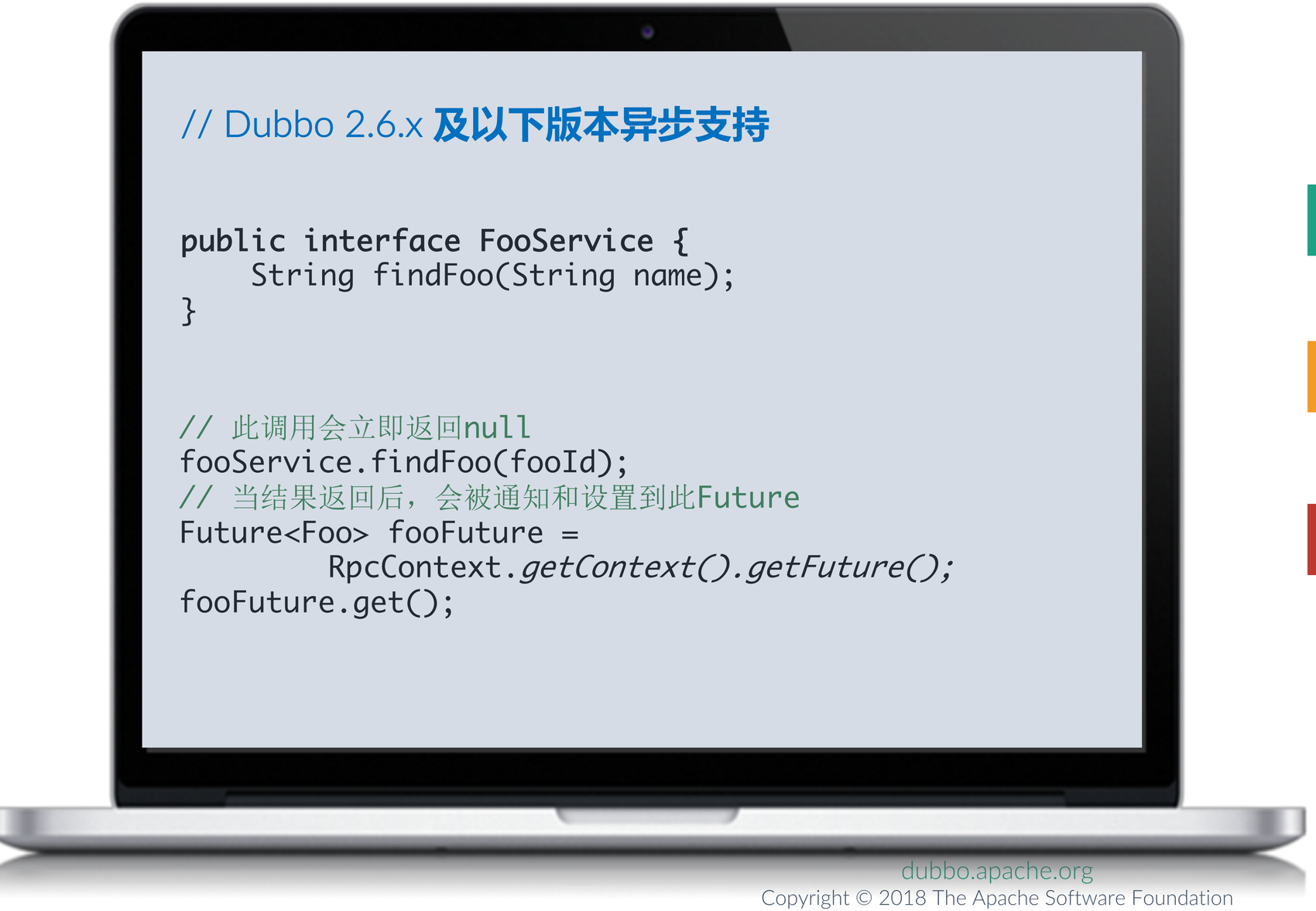
兼容包

- dubbo-compatible: 核心API/扩展兼容

实现

- JDK8 default method

异步支持



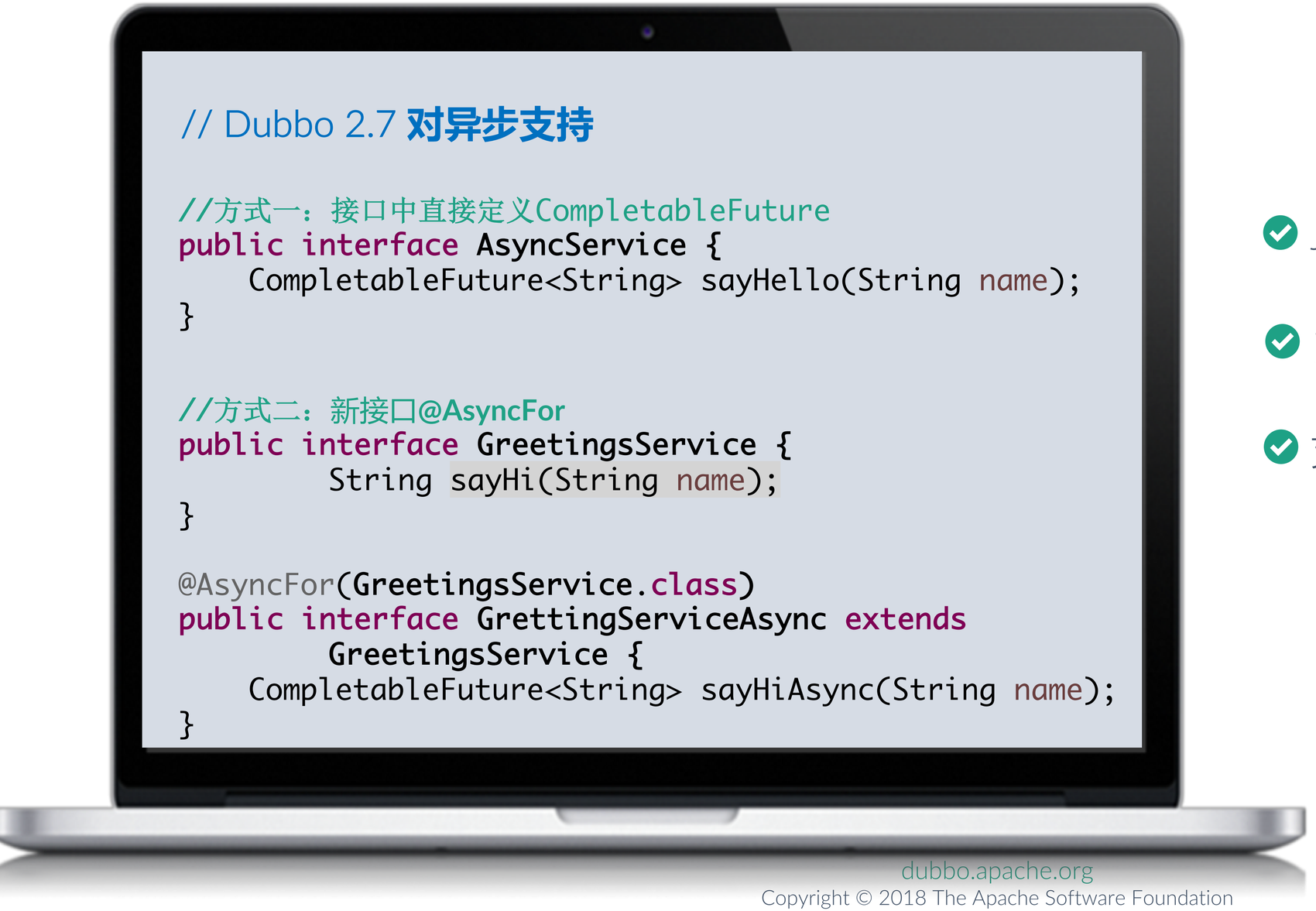
// Dubbo 2.6.x **及以下版本异步支持**

```
public interface FooService {
    String findFoo(String name);
}
```

```
// 此调用会立即返回null
fooService.findFoo(fooId);
// 当结果返回后，会被通知和设置到此Future
Future<Foo> fooFuture =
        RpcContext.getContext().getFuture();
fooFuture.get();
```

- Future获取“好奇怪”
- Future无法实现自动回调
- 不支持Provider端异步

异步支持



// Dubbo 2.7 **对异步支持**

```
//方式一：接口中直接定义CompletableFuture
public interface AsyncService {
    CompletableFuture<String> sayHello(String name);
}


//方式二：新接口@AsyncFor
public interface GreetingsService {
        String sayHi(String name);
}

@AsyncFor(GreetingsService.class)
public interface GrettingServiceAsync extends
        GreetingsService {
    CompletableFuture<String> sayHiAsync(String name);
}
```

- Jdk8新特性：CompletableFuture方式
- 支持Provider端的异步
- 支持了Promise方式

异步支持

方式一

```
public interface AsyncService {
    CompletableFuture<String> sayHello(String name);
}
```

接口


```
public class AsyncServiceImpl implements AsyncService {
    @Override
    public CompletableFuture<String> sayHello(String name) {
        RpcContext savedContext = RpcContext.getContext();
        RpcContext savedServerContext =
RpcContext.getServerContext();
        return CompletableFuture.supplyAsync(() -> {
            savedServerContext.setAttachment("server-key1", "server-
value1");
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            return "async response from provider.";
        });
    }
}
```

Provider

```
CompletableFuture<String> future =
                asyncService.sayHello("async call request");
RpcContext savedServerContext =
RpcContext.getServerContext();

future.whenComplete((v, t) -> {
   String sv =savedServerContext.getAttachment("server-
key1"));
   System.out.println(sv);
    if (t != null) {
        t.printStackTrace();
    } else {
        System.out.println("Response: " + v);
    }
});
```

Consumer

异步支持

方式二


```
@DubboAsync
public interface GreetingsService
{

    String sayHi(String name);

}
```

接口

dubbo-async-processer

编译阶段自动生成

```
@Generated("com.alibaba.dubbo.async.processor.AsyncAnnotationProcessor")
@AsyncFor(com.alibaba.dubbo.samples.api.GreetingsService.class)
public interface GreetingsServiceAsync extends GreetingsService {

    CompletableFuture<java.lang.String> sayHiAsync(java.lang.String name);
}
```

生成的异步接口

实现

```
public class GreetingsServiceImpl implements
GreetingsService {

    @Override
    public String sayHi(String name) {

        return "hello, " + name;
    }
}
```

Provider

依赖

```
//异步
CompletableFuture<String> future =
                greetingsService.sayHiAsync(“async call reqeust”);
System.out.println("async call ret :" + future.get());
//同步
System.out.println(greetingsService.sayHi("normal sync call "));
```

Consumer

异步支持

方式二改造：provider端异步

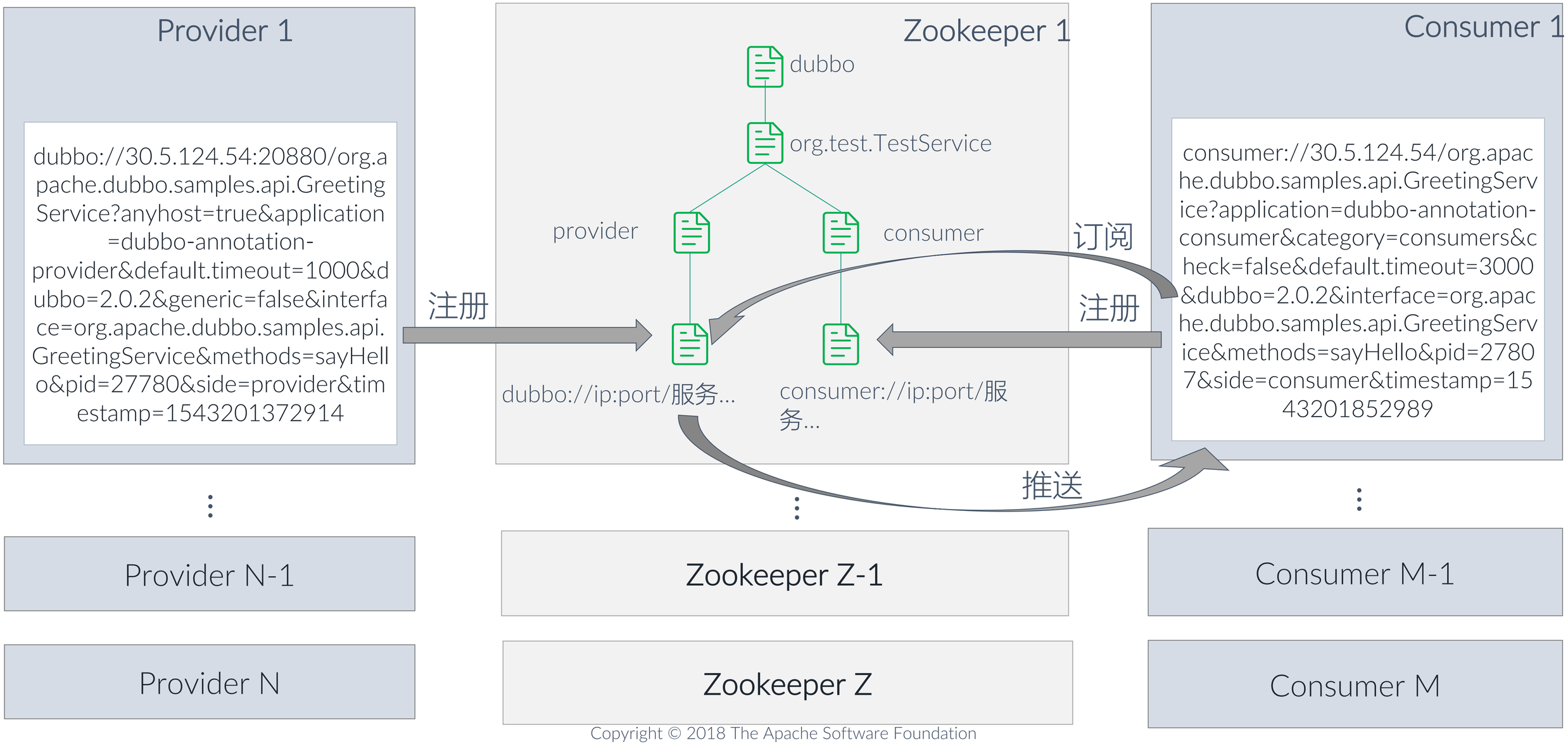
```
public class AsyncServiceImpl implements AsyncService {
    @Override
    public String sayHello(String name) {
        final AsyncContext asyncContext = RpcContext.startAsync();
        new Thread(() -> {
            asyncContext.signalContextSwitch();
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            asyncContext.write("Hello " + name + ", response from provider.");
        }).start();
        return "hello, " + name;
    }
}
```

Provider

```
<dubbo:service async="true" interface="AsyncService" ref="asyncService"/>
```

服务配置

元数据改造



元数据改造

Analysis

性能

推送量大 -> 存储数据量大 -> 网络传输量大 -> 延迟严重

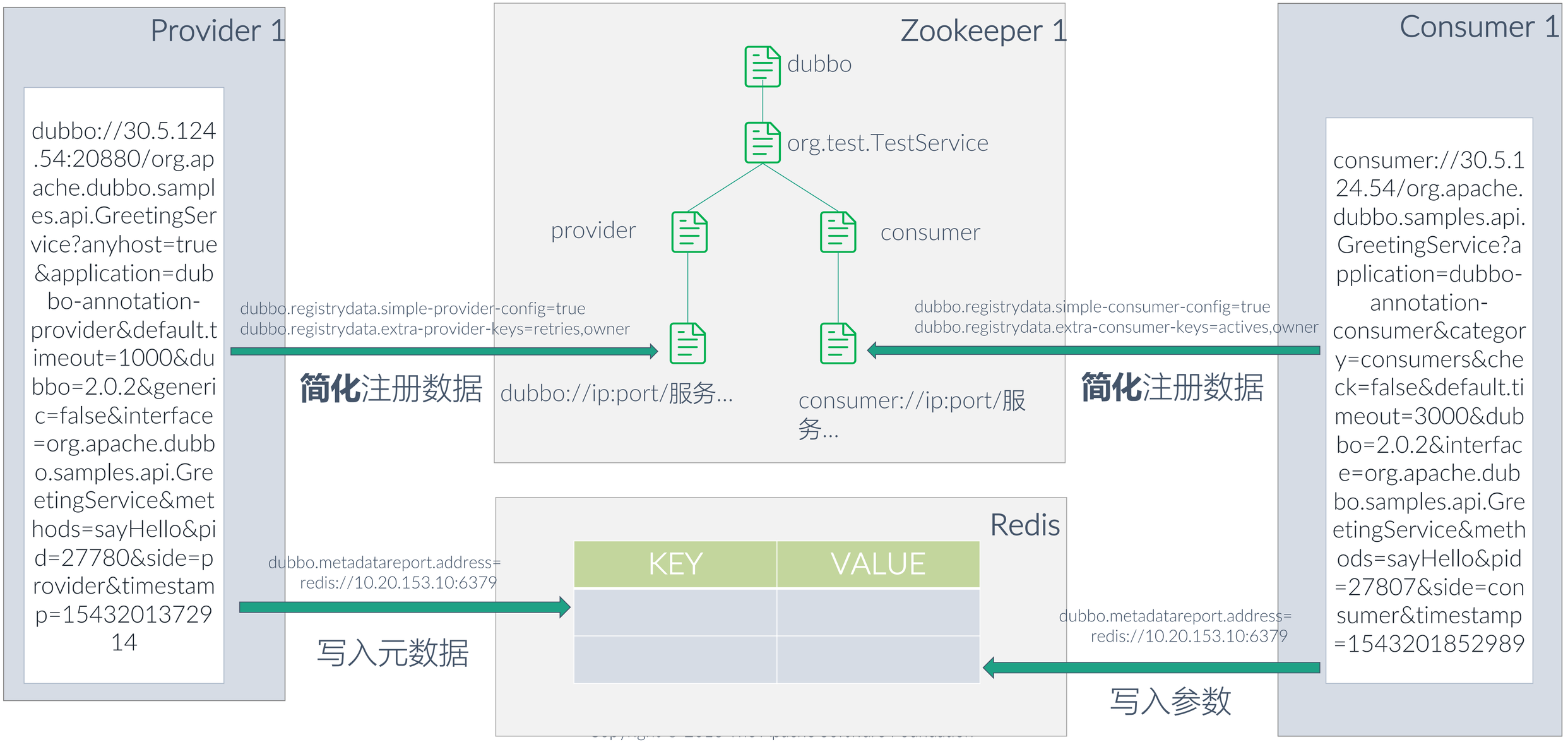
参数分析

生产者端注册30+参数，有接近一半是不需要作为注册中心进行传递
消费者端注册25+参数，只有个别需要传递给注册中心

新需求

OPS-服务测试需要元数据信息

元数据改造


Provider 1
dubbo://30.5.124.54:20880/org.apache.dubbo.samples.api.GreetingService?anyhost=true&application=dubbo-annotation-provider&default.timeout=1000&dubbo=2.0.2&generic=false&interface=org.apache.dubbo.samples.api.GreetingService&methods=sayHello&pid=27780&side=provider&timestamp=1543201372914
Zookeeper 1
dubbo
org.test.TestService
provider
consumer
dubbo.registrydata.simple-provider-config=true
dubbo.registrydata.extra-provider-keys=retries,owner
简化注册数据
dubbo://ip:port/服务...
consumer://ip:port/服务...
dubbo.registrydata.simple-consumer-config=true
dubbo.registrydata.extra-consumer-keys=actives,owner
简化注册数据
Consumer 1
consumer://30.5.124.54/org.apache.dubbo.samples.api.GreetingService?application=dubbo-annotation-consumer&category=consumers&check=false&default.timeout=3000&dubbo=2.0.2&interface=org.apache.dubbo.samples.api.GreetingService&methods=sayHello&pid=27807&side=consumer&timestamp=1543201852989
Redis
KEY
VALUE
dubbo.metadatareport.address=redis://10.20.153.10:6379
写入元数据
dubbo.metadatareport.address=redis://10.20.153.10:6379
写入参数

动态配置中心

动态配置

静态配置

动态配置缺少远程dubbo.properties

服务治理的配置

只有服务级别的配置

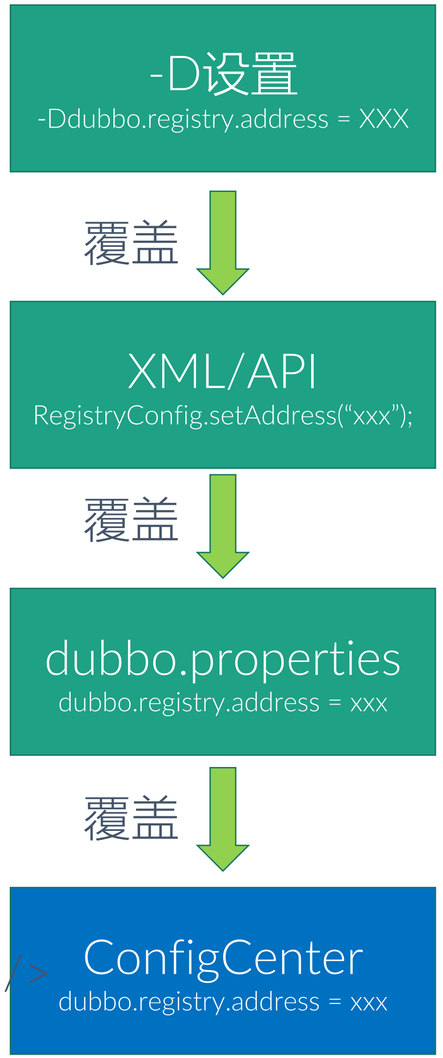
一个服务可以有多个配置项

存储：和注册中心共用

动态配置中心

动态配置

支持类似于Spring Cloud Config的远程配置方式托管
支持新的覆盖关系




```
<dubbo:configcenter address="zookeeper://127.0.0.1:2181"/>
```

动态配置中心

动态配置

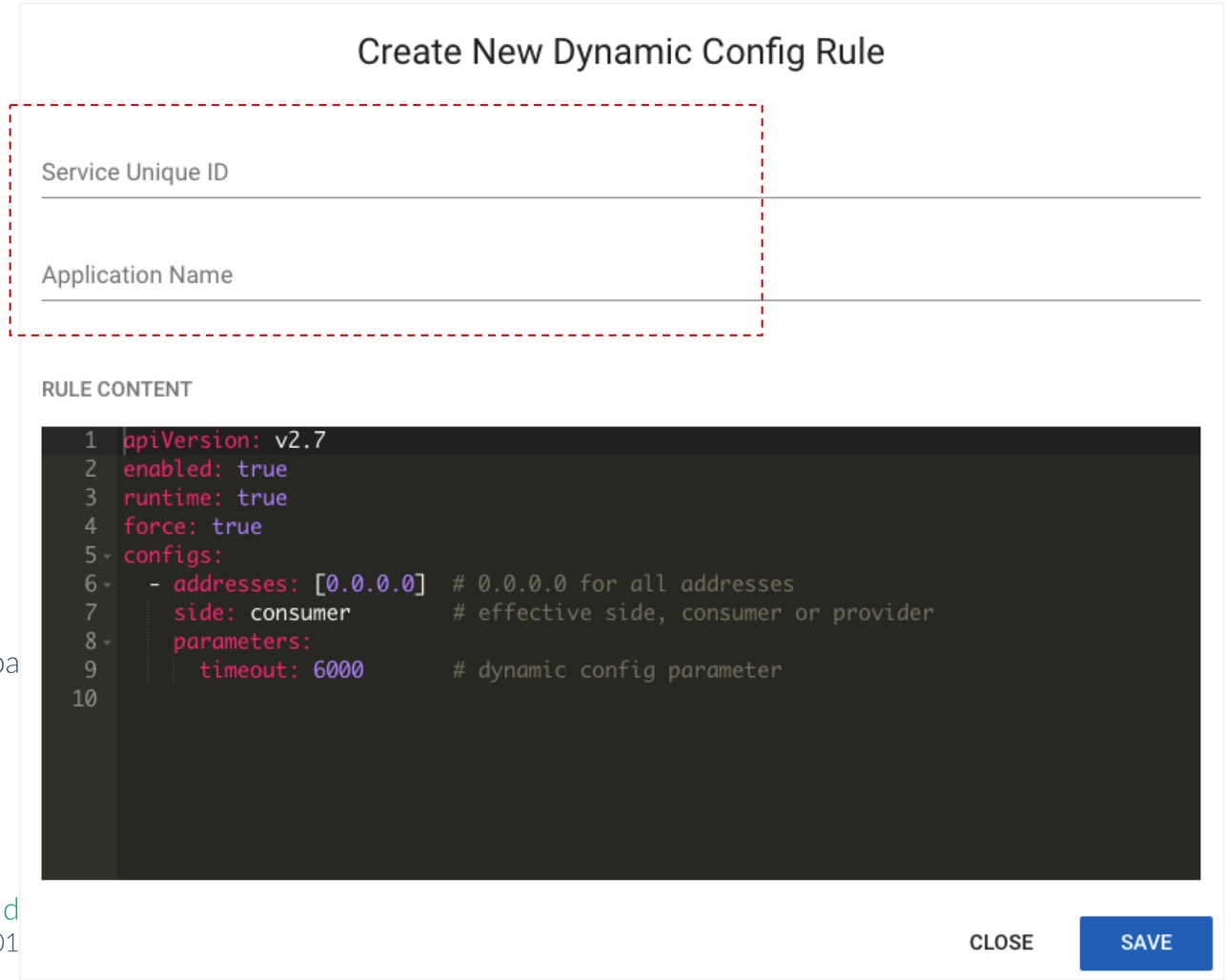
支持类似于Spring Cloud Config的远程配置方式托管
支持新的覆盖关系

服务治理参数配置

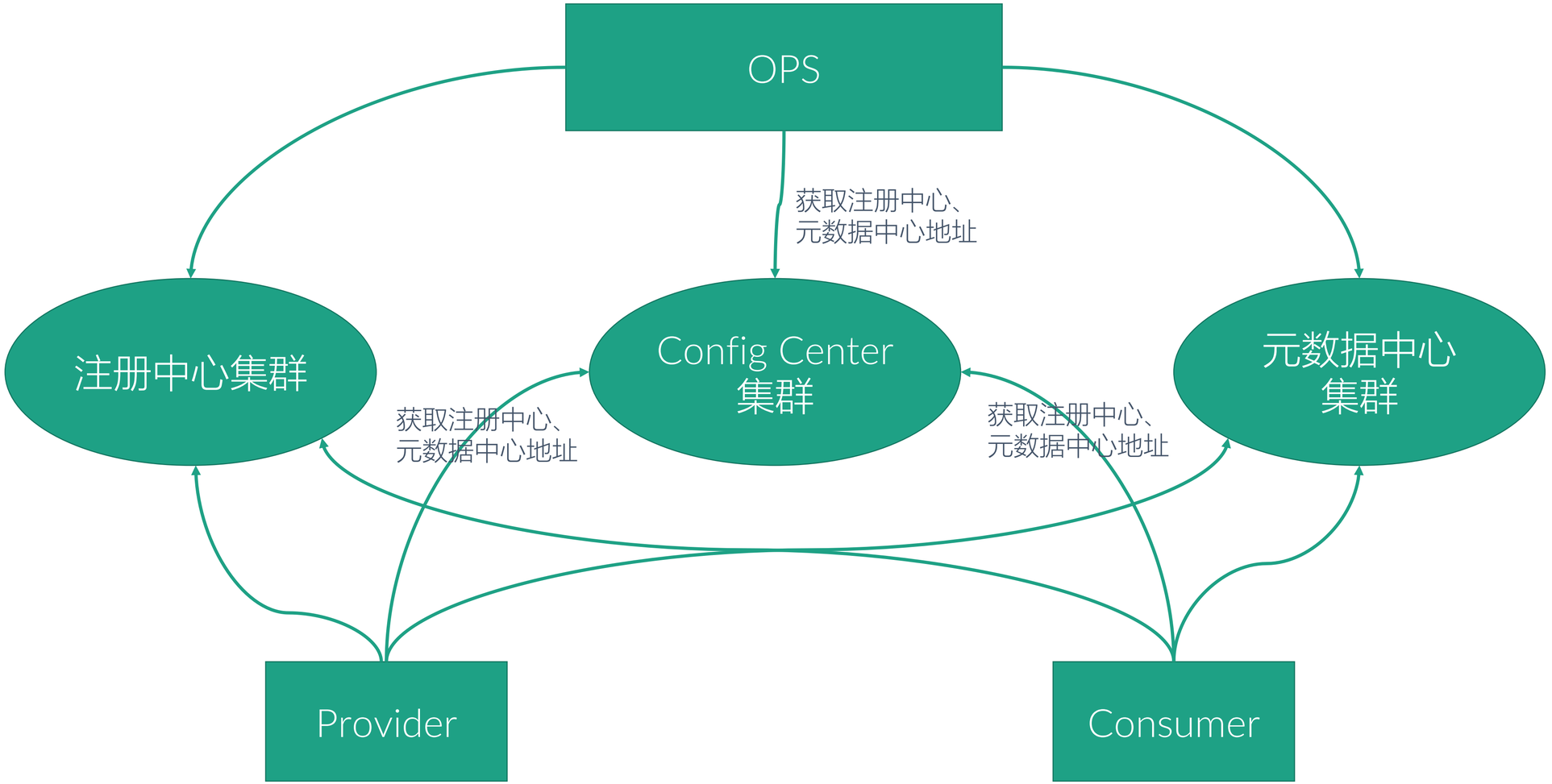
支持了应用级别，服务级别的配置
兼容Override配置

SPI定义

定义了新的SPI，可以自定义动态配置中心，接口： org.apa
默认支持Apollo，Nacos，Zookeeper作为配置中心



三个中心



路由规则配置

功能

只支持服务粒度的路由

支持的路由方式不够，如tag

存储

路由规则存储在注册中心

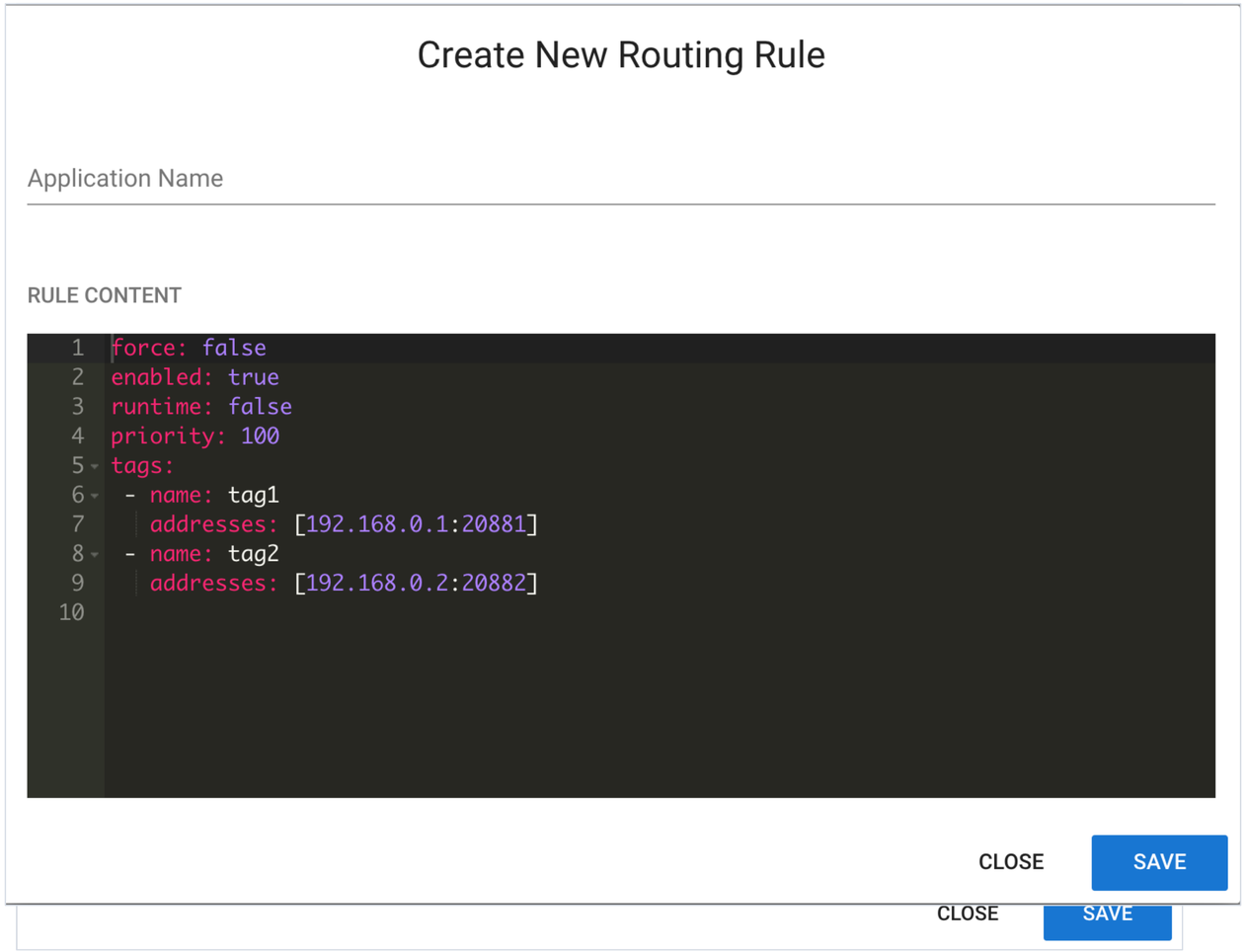
复杂不确定

一个服务或应用允许定义多条路由规则

设计不够优雅

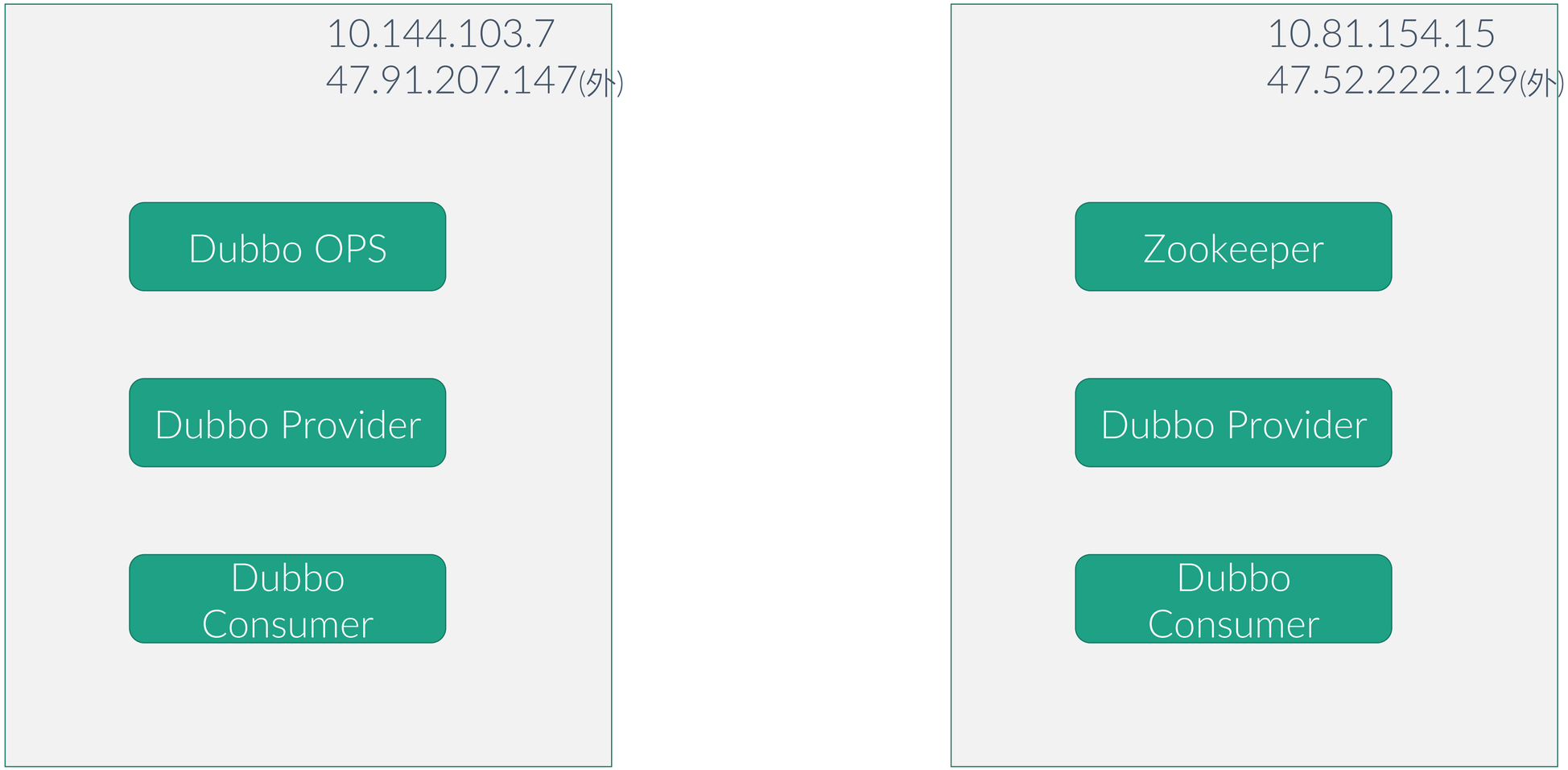
缓存失效

路由规则配置



- 存储：从注册中心转换到配置中心
- 功能增强：支持应用级别路由，支持Tag路由增强
- 精确的路由规则：每个服务能对应到精确的规则
- 实现方式上进行优化

Demo



阿里中间件Aliware开发者中心

323人

扫一扫群二维码，立刻加入该群。

Home

dubbo.apache.org

dubbo.io

GitHub

github.com/apache/incubator-dubbo

github.com/dubbo

Mailing List

dev@dubbo.apache.org

IM

gitter.im/alibaba/dubbo

钉钉群：21711817

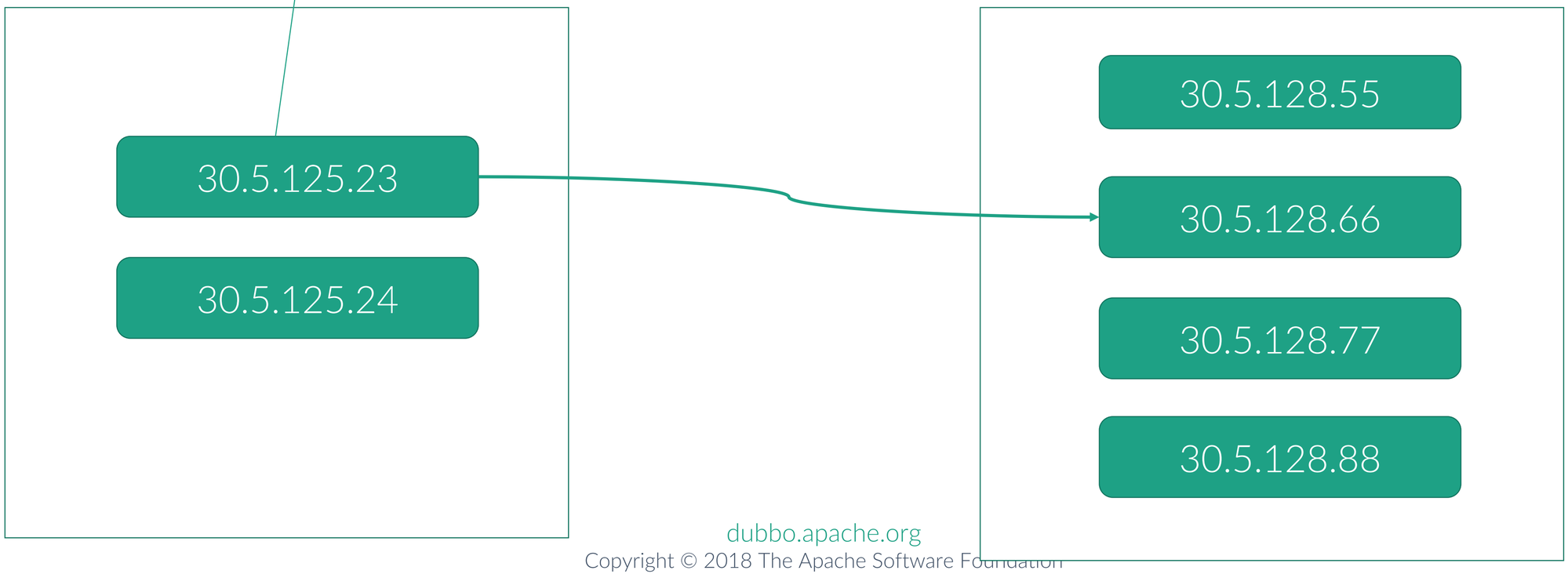
Thank you !

路由规则

org.apache.dubbo.rpc.RpcContext.*getContext*().setAttachment(”tag”,"gray");

增加打标规则：

```
tags:
  - name: gray
    addresses: [30.5.128.66:20880]
```



路由规则

增加打标规则：

```
tags:
  - name: canary
    addresses: [30.5.120.16:20880]
```

增加打标规则：

```
tags:
  - name: canary
    addresses: [30.5.128.66:20880]
```