

# Dubbo

[dʌboʊ]

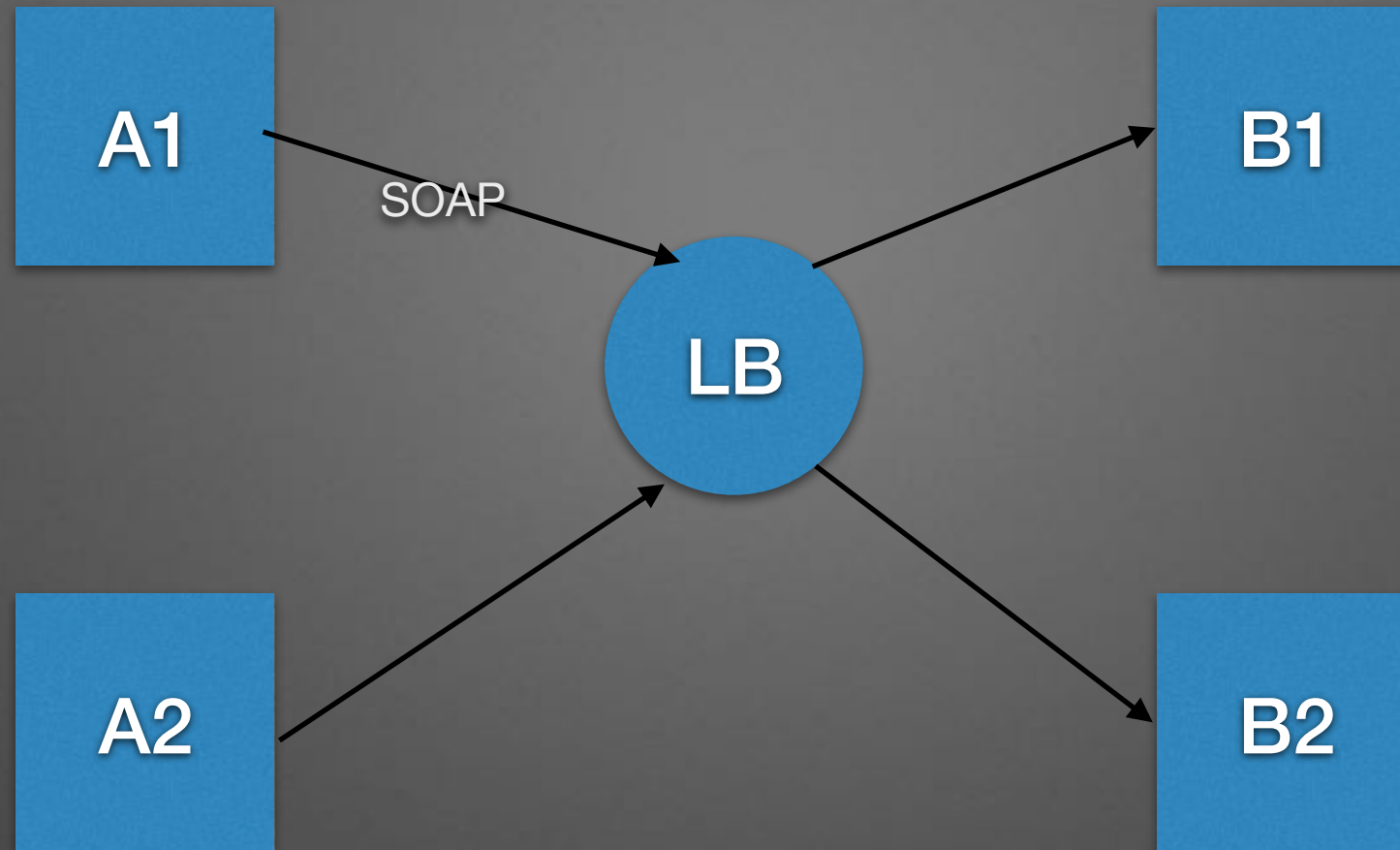
@BohrQiu

**What?**

Dubbo是阿里巴巴开源出来的一个分布式服务框架，致力于提供高性能和透明化的RPC远程服务调用方案，以及作为SOA服务治理的方案。

**Why?**

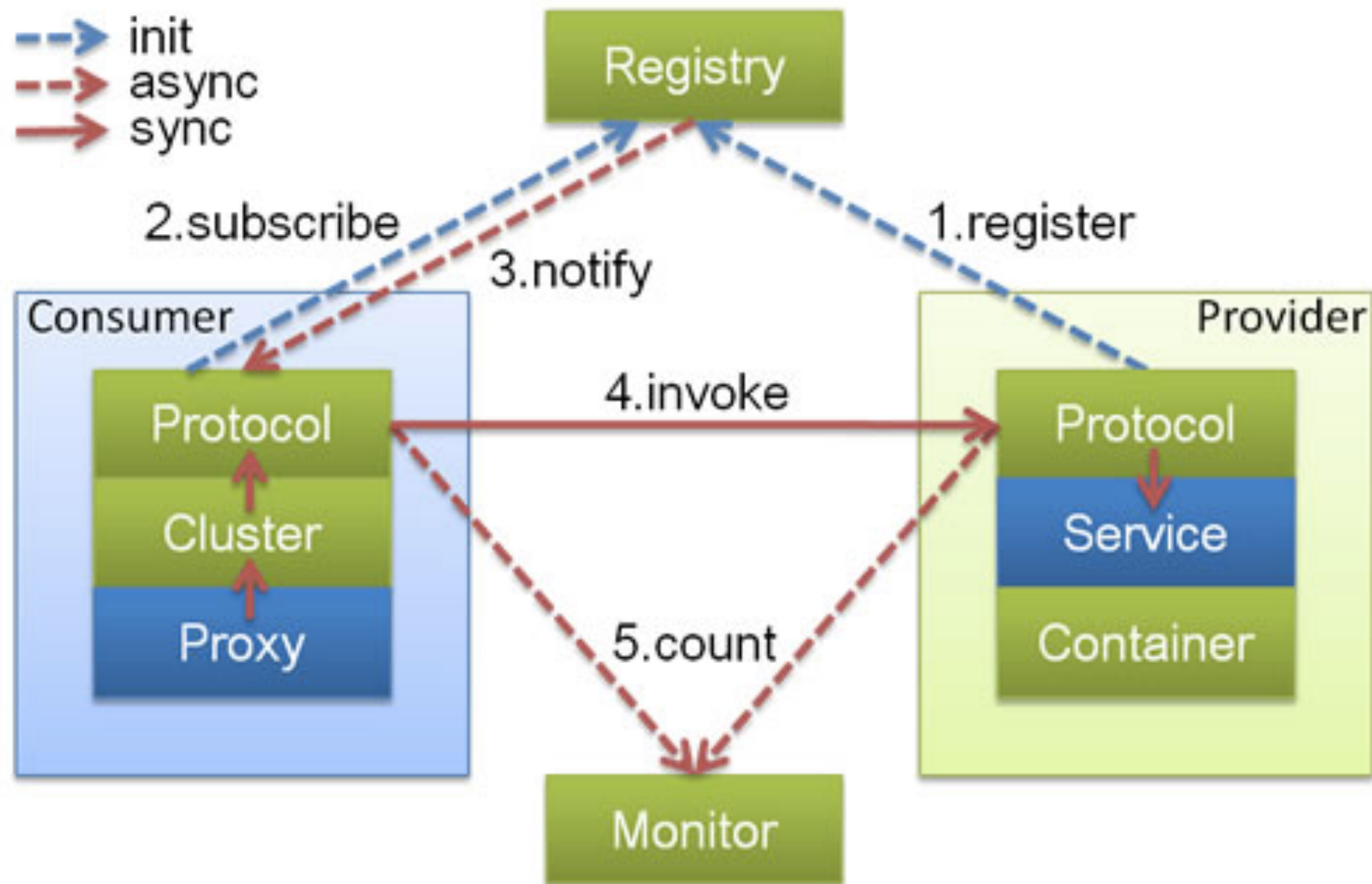
# BEFORE



1. 服务端暴露WebService接口
2. LB保证高可用(尽量保证系统可用, 系统出故障后尽快恢复)

# ISSUE

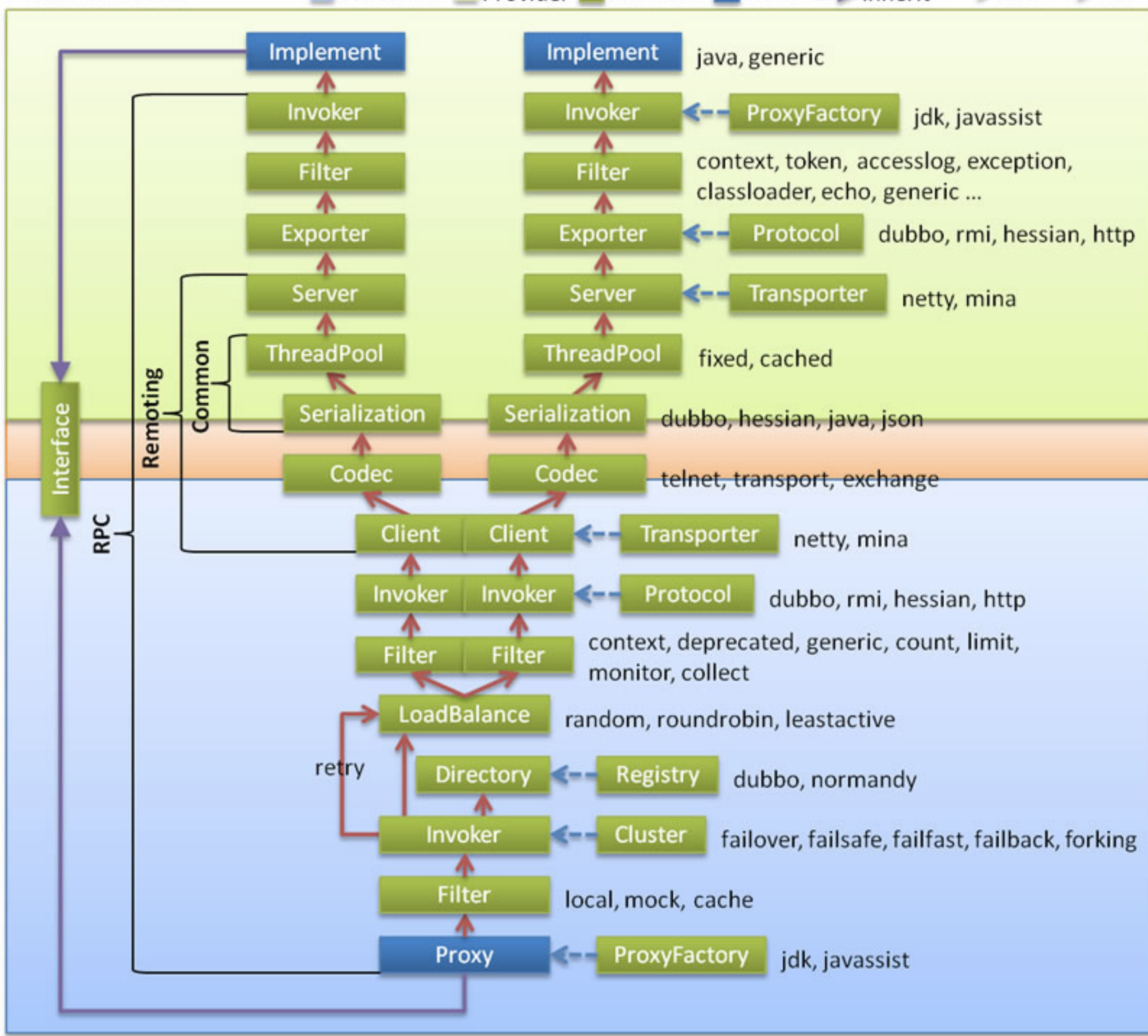
- 1.服务消费者需要知道服务提供者ip信息
- 2.soap协议需要优化
- 3.服务兼容性需要保证
- 4.LB维护成本大，遇到LB bug很难排查
- 5.服务更新(关流量，重启服务，开流量)麻烦
- 6.完全没有服务治理能力





# Dubbo Extension

Consumer Provider Interface Class Inherit Init Call





- 高性能NIO通讯及多协议集成
- 服务注册与发现
- 软负载均衡与容错
- 依赖分析与降级
- 服务监控
- 透明化的远程方法调用
- 优雅停机

**How?**

# provider

```
public interface DemoService {  
    String sayHello(Person person);  
}
```

```
public class DemoServiceImpl implements  
DemoService {
```

```
    public String sayHello(Person person) {  
        return "Hello," + person.getName() ;  
    }  
}
```

# provider

<!--配置注册中心为zookeeper-->

```
<dubbo:registry protocol="zookeeper"
address="xxx" file="xxx"/>
```

<!--配置协议-->

```
<dubbo:protocol name="dubbo" port="20882"
threads="200" />
```

<!--开启监控-->

```
<dubbo:monitor protocol="registry"/>
```

<!--配置提供者-->

```
<dubbo:provider timeout="60000"
cluster="failfast" register="true"/>
```

# provider

<!-- 注册bean到spring容器 -->

```
<bean id="demoService"  
      class="DemoServiceImpl"/>
```

<!-- 暴露服务到zookeeper -->

```
<dubbo:service interface="DemoService"  
               ref="demoService" version="1.0" />
```

# consumer

<!--配置注册中心为zookeeper-->

<dubbo:registry protocol="zookeeper"  
address="xxx" file="xxx"/>

<!--开启监控-->

<dubbo:monitor protocol="registry"/>

<!--配置消费者-->

<dubbo:consumer check="false"  
loadbalance="roundrobin"/>

<!--配置服务代理-->

<dubbo:reference id="demoService"  
                  interface="DemoService"  
version="1.0" />



# consumer

//代码中获取DemoService代理

```
DemoService demoService =  
    context.getBean("demoService",  
DemoService.class);
```

@Autowired

```
private DemoService demoService;
```

**Attentions!**

## 1.配置优先级

服务超时和重试次数，由服务提供方来配置。因为服务提供方更能清楚的了解服务的能力服务注册与发现

## 2.修改统计日志级别

```
<logger name="com.alibaba.dubbo.monitor.dubbo.DubboMonitor"
level="warn"></logger>
```

## 3.zookeeper client

使用zkclient作为zookeeper的client

## 4.兼容性

服务接口增加方法，或服务模型增加字段，可向后兼容，删除方法或删除字段，将不兼容。不兼容时需通过变更版本号升级。

## 5.序列化

服务参数及返回值使用POJO对象，实现Serializable

## 6.本地开发时：

`<dubbo:registry address="xxxx" register="false" />` 配置注册为false，避免在本地启动项目时，发布到zookeeper，被其他用户使用到

本地远程测试(测试本地启的应用)，使用直连的方式

## 7.配置上Dubbo缓存文件

当应用重启过程中，Dubbo注册中心不可用时则应用会从这个缓存文件读取服务提供者列表的信息，进一步保证应用可靠性

## 8.服务端口

不要使用随机端口，使用随机端口，上面的缓存文件就没有任何作用了。

## 9.停机

Dubbo是通过JDK的ShutdownHook来完成优雅停机的，所以如果用户使用"kill -9 PID"等强制关闭指令，是不会执行优雅停机的，只有通过"kill PID"时，才会执行。运维的同学要注意下了。开发的童鞋在classpath下新建dubbo.properties配置文件来设置服务关机的等待时间，

## 10.多服务实现时编程调用特定服务

`com.yjf.common.dubbo.DubboRemoteProxyFacotry#getProxy`

## 11.timeout

请仔细评估服务的超时时间

## 12.版本号

建议选用两位，比如1.2，版本兼容的升级就不改版本号，不然，每次改版本号，client也要跟着改。对于不兼容的升级，我们需要修改版本号和依赖的facade版本

## 13.跨机房

重庆和天津共用一套zookeeper，zookeeper集群搭建到天津，重庆的应用发布到天津的zookeeper集群，暴露自己的物理ip，运维童鞋打通vpn，天津的应用可以通过重庆服务的物理ip访问。以后可以在天津搭建ZooKeeper Observers



## 14.多版本共存

理论上是支持多版本共存的, 但是支付宝不建议多版本长期共存。  
我们可以通过dubbo的版本来实现服务的平滑升级

首先服务提供者发布新版本服务, 服务消费者逐步升级, 服务消费者全部升级完后, 服务提供者全部升级到新版本服务。

## 15.启动时检查

```
<dubbo:consumer check="false" />
```

## 16.hessian序列化

字段重名问题,如果order继承链中有重名字段, 在反序列化时会出现字段丢失.

**FIX IT**

Closed

Issue #1 · created by  秋波 9 months ago ·  9 months ago

+ New Is

## 传输大数据导致oom的bug


com.alibaba.dubbo.remoting.exchange.codec.ExchangeCodec#encodeResponse中在处理超大数据时,会多次把数据记录进日志,导致oom.

Closed

Issue #3 · created by  秋波 9 months ago ·  9 months ago

## dubbo服务关闭时,不等待正在执行的任务执行完毕.

1 participant

 秋波 @qzhanbo · 9 months ago

修改关闭dubbo服务执行步骤为: 1.provider通知consumer,不要发新请求过来 2.provider等待服务执行完毕  
3.consumer等待服务执行完毕 3.provider关闭 4.cusumer关闭

## dubbo和spring容器关闭顺序调整导致空指针异常

我们今天测试使用kill来关闭maven-tomcat-plugin,我看日志里面里的关闭顺序:

### 1. 关闭dubbo shutdownhook

取消服务注册/关闭注册中心的连接/关闭

### 2. 关闭spring 容器

每个dubbo服务是一个ServiceBean对象(实现DisposableBean接口),在关闭时,又会去调用取消服务注册,这个时候zookeeper客户端已经关闭了,所以会导致空指针异常.

我们正常的关闭行为是先关闭spring容器,然后运行com.yjf.common.util.ShutdownHooks(里面我注册了一个dubbo的服务关闭钩子),这样是不会出现这个异常的.

Closed

Issue #6 · created by 秋波 2 months ago · 2 months ago

+ New Issue

Reopen

Edit

## 编程式获取dubbo服务代理对象造成oom

编程式的方式使用ReferenceConfig创建服务代理对象，当指定版本的服务不存在时，会报出异常

java.lang.IllegalStateException: Failed to check the status of the service XXXX,此ReferenceConfig对象所创建的资源没有被正确销毁，最终导致OOM

## 反序列化失败，异常信息被吞噬

dubbo-rpc/dubbo-rpc-default/src/main/java/com/alibaba/dubbo/rpc/protocol/dubbo/DecodeableRpcInvocation.java

```
...    @@ -69,12 +69,6 @@ public class DecodeableRpcInvocation extends RpcInvocation implements Codec, Dec
69    69        if (!hasDecoded && channel != null && inputStream != null) {
70    70            try {
71    71                decode(channel, inputStream);
72    -        } catch (Throwable e) {
73    -            if (log.isWarnEnabled()) {
74    -                log.warn("Decode rpc invocation failed: " + e.getMessage(), e);
75    -            }
76    -            request.setBroken(true);
77    -            request.setData(e);
78    72        } finally {
79    73            hasDecoded = true;
80    74        }
...    @@ -105,13 +99,7 @@ public class DecodeableRpcInvocation extends RpcInvocation implements Codec, Dec
105   99        pts = ReflectUtils.desc2classArray(desc);
106   100        args = new Object[pts.length];
107   101        for (int i = 0; i < args.length; i++) {
108   -            try {
109   -                args[i] = in.readObject(pts[i]);
110   -            } catch (Exception e) {
111   -                if (log.isWarnEnabled()) {
112   -                    log.warn("Decode argument failed: " + e.getMessage(), e);
113   -                }
114   -            }
102   +            args[i] = in.readObject(pts[i]);
115   103        }
116   104    }
117   105    setParameterTypes(pts);
```

# 修复注册中心为zookeeper，管理控制台删除路由失败的问题 #13



Merged

liangfei0201 merged 2 commits into `alibaba:master` from `unknown repository` on 2 Jan 2014



Conversation 0



Commits 2



Files changed 2



bohrqiu commented on 13 Sep 2013



这个问题是因为，在启用路由规则时，去掉了enabled参数。在删除路由规则时Route.toUrl方法会始终加上enabled参数，这样导致删除时zookeeper中path不存在，删除失败。



# 新增SerializerFeature.WriteEnumUsingName用于在序列化时输出枚举的name()值。 #300

 **Merged**

yakolee merged 2 commits into `alibaba:master` from `bohrqiu:master` on 9 Mar

 Conversation 0

 Commits 2

 Files changed 6



bohrqiu commented on 1 Mar

新增SerializerFeature.WriteEnumUsingName用于在序列化时输出枚举的name()值。

Release Notes(1.1.42) [https://github.com/alibaba/fastjson/wiki/Release-Notes\(1.1.42\)](https://github.com/alibaba/fastjson/wiki/Release-Notes(1.1.42)) “修复序列化时SerializerFeature.WriteEnumUsingToString不生效的bug”，这种做法会引起问题。如果重写了枚举的toString方法，会导致反序列化失败。而且，此fix ([6a2b502](#))

此修改序列化枚举时默认用枚举的name()值，和以前的版本保持一致，并且修改了所有使用WriteEnumUsingToString特性的代码，让他真正生效。



# dubbo增加传输数据压缩特性

## 1. 特性说明

由于我们存在跨机房的远程调用，数据压缩可以减少TCP数据包传输时间。

1. 数据包更小，传输时间减少
2. 在网络质量很差的情况下，TCP重传时间减少

由于此特性需要客户端和服务端同时升级，所以此次dubbo版本升级，我们并不会启用此特性，待所有系统升级到此版本的dubbo后，我们在需要跨机房传输的系统上开启此特性。

## 2. 性能测试

### 2.1. 压缩比

```
NotDeflation:2084
Deflation:201
```

原始报文:2084byte，压缩后大小:201byte，压缩比：201/2084=9%

### 2.2. 压缩对性能的影响

#### 2.2.1 cpu/mem

```
//基于压缩的序列化和反序列化
CompressTest.testAllDeflation: [measured 10000 out of 10100 rounds, threads: 4 (all cores)]
  round: 0.00 [+ 0.00], round.block: 0.00 [+ 0.00], round.gc: 0.00 [+ 0.00], GC.calls: 9, GC.time: 0.03, time.total: 3.33, time.warmup: 0.08,

//非压缩情况下的序列化和反序列化
CompressTest.testAllNonDeflation: [measured 10000 out of 10100 rounds, threads: 4 (all cores)]
  round: 0.00 [+ 0.00], round.block: 0.00 [+ 0.00], round.gc: 0.00 [+ 0.00], GC.calls: 9, GC.time: 0.02, time.total: 1.66, time.warmup: 0.03,
```

我测试了10000次序列化和反序列化，总的来说，对gc影响几乎没有。cpu开销加倍，但是平均耗时仍然比较小。

#### 2.2.2 线程模型

如果在I/O线程中执行压缩和解压操作，会非常影响dubbo的性能。

目前我们采用的线程模型如下：

1. 心跳包在I/O线程序列化和反序列化
2. 业务数据包在业务线程序列化和反序列化

心跳包数量较小，而且报文也很小，在I/O线程中执行对性能影响很小。按照当前的线程模型评估，开启压缩功能对业务吞吐量影响很小。

# 在Dubbo服务实现类上的方法上增加@Transactional，服务注册失败

1 participant



qzhanbo



秋波 @qzhanbo · 10 months ago

在`com.alibaba.dubbo.config.spring.AnnotationBean#postProcessAfterInitialization`中，  
获取代理对象的`TargetClass`，但是此对象也是cglib生成的对象。

需要获取到`target`对象

Assignee

Select

Milestone

Closed

Issue #31 · created by  秋波 about a year ago ·  about a year ago

+ New Issue

Reopen

 Edit

## [EventListener]优化dubbo ServiceBean

spring 4.2 引入@EventListener，可以很方便的实现事件编程。

dubbo ServiceBean实现ApplicationListener接口，如果dubbo服务很多，每次发送一个事件，所有的dubbo服务ServiceBean都要遍历一遍，性能不好，需要优化成ApplicationListener<ContextRefreshedEvent>

Closed

Issue #14 · created by  秋波 5 months ago ·  5 months ago

## loancoop项目在shiroRealm中使用dubbo引用为null

1 participant



秋波 @qzhanbo · 5 months ago

原因分析:

`PostProcessorRegistrationDelegate#registerBeanPostProcessors`注册spring容器所有使用到的BPP，这里的步骤为：

1. 注册所有实现`PriorityOrdered`接口的BPP
2. 注册所有实现`Ordered`接口的BPP
3. 注册其他BPP

`ShiroFilterFactoryBean`和`AnnotationBean`都在第三步初始化，且`ShiroFilterFactoryBean`比`AnnotationBean`先注册。

当`ShiroFilterFactoryBean`注册时会初始化`shiroRealm`，所以在`shiroRealm`类里声明的`@Reference`字段没有被扫描，为null。

以前修改`AnnotationBean`实现了`PriorityOrdered`接口，但是后面给回退了，具体原因忘了，现在修改`AnnotationBean`实现`Ordered`接口。

相关issue:

<http://gitlab.yiji/qzhanbo/dubbo/issues/9>

<http://gitlab.yiji/qzhanbo/yiji-boot/issues/93>

## dubbo yiji-2.5.10发布 ☆ 𠄎

发件人：qiubo <qiubo@yiji.com>

时 间：2016年1月15日(星期五) 下午3:47

收件人：📧技术人员群 <technician@yiji.com>

抄 送：培根（李培跃） <pei-gen@yiji.com>; 铁拳（刘刚） <tie-quan@yiji.com>; 之恒 <zhi-heng@yiji.com>

主要优化了 @Reference 启动时间，以 commonservices 为例，启动耗时减少32s。

```
2016-01-15 13:27:40.150 INFO [Main] Main:57-- Started Main in 147.886 seconds (JVM running for 149.589)
2016-01-15 13:34:00.589 INFO [Main] Main:57-- Started Main in 114.596 seconds (JVM running for 116.739)
```

# Resources.



- <http://dubbo.io/>
- <http://dubbo.io/Developer+Guide-zh.htm>
- SOA
- confluence