**服务治理之dubbo实践**

**目录**

[1 dubbo背景简介 4](#_Toc8856)

[2 什么是dubbo 5](#_Toc7371)

[2.1 什么是RPC 5](#_Toc9357)

[2.2 RPC需要解决的问题 5](#_Toc28128)

[2.3 Dubbo简介及架构图 5](#_Toc25871)

[3 环境准备 7](#_Toc26431)

[3.1 开发工具 7](#_Toc17897)

[3.2 启动zookeeper 7](#_Toc25835)

[3.3 启动dubbo-admin-server 8](#_Toc22232)

[3.4 启动dubbo-admin-ui 9](#_Toc21609)

[3.5 启动provider 10](#_Toc24859)

[3.6 启动consumer 10](#_Toc27783)

[4 Dubbo实践 12](#_Toc12462)

[4.1 服务提供者 12](#_Toc42)

[4.2 服务消费者 12](#_Toc4176)

[4.3 启动时检查 13](#_Toc11455)

[4.4 集群容错 14](#_Toc22208)

[4.4.1 Failover Cluster 14](#_Toc26884)

[4.4.2 Failfast Cluster 14](#_Toc5843)

[4.4.3 Failsafe Cluster 15](#_Toc24385)

[4.4.4 Failback Cluster 15](#_Toc20822)

[4.4.5 Forking Cluster 15](#_Toc5676)

[4.4.6 Broadcast Cluster 15](#_Toc16084)

[4.5 负载均衡 15](#_Toc17935)

[4.5.1 随机策略-Random LoadBalance 15](#_Toc14520)

[4.5.2 轮询策略-RoundRobin LoadBalance 15](#_Toc25569)

[4.5.3 最少活跃调用数-LeastActive LoadBalance 16](#_Toc17513)

[4.6 直连提供者 17](#_Toc31352)

[4.7 只订阅 17](#_Toc6220)

[4.8 只注册 17](#_Toc27133)

[4.9 多协议 17](#_Toc29138)

[4.10 多注册中心 18](#_Toc10311)

[4.11 服务分组 18](#_Toc3326)

[4.12 多版本 19](#_Toc12490)

[4.13 分组聚合 19](#_Toc20737)

[4.14 结果缓存 20](#_Toc2362)

[4.15 泛化引用与泛化实现 20](#_Toc8932)

[4.16 回声测试 20](#_Toc10262)

[4.17 Consumer异步调用 21](#_Toc5722)

[4.18 Provider异步执行 22](#_Toc13493)

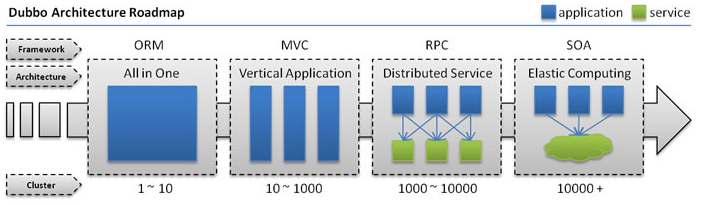
[4.19 延迟暴露 23](#_Toc13858)

[4.20 动态配置 23](#_Toc5258)

[5 参考资料 24](#_Toc10423)

# dubbo背景简介

随着互联网的发展，网站应用的规模不断扩大，常规的垂直应用架构已无法应对，分布式服务架构以及流动计算架构势在必行，亟需一个治理系统确保架构有条不紊的演进。



* 单一应用架构

当网站流量很小时，只需一个应用，将所有功能都部署在一起，以减少部署节点和成本。此时，用于简化增删改查工作量的数据访问框架(ORM)是关键。

* 垂直应用架构

当访问量逐渐增大，单一应用增加机器带来的加速度越来越小，将应用拆成互不相干的几个应用，以提升效率。此时，用于加速前端页面开发的Web框架(MVC)是关键。

* 分布式服务架构

当垂直应用越来越多，应用之间交互不可避免，将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，使前端应用能更快速的响应多变的市场需求。此时，用于提高业务复用及整合的分布式服务框架(RPC)是关键。

* 流动计算架构

当服务越来越多，容量的评估，小服务资源的浪费等问题逐渐显现，此时需增加一个调度中心基于访问压力实时管理集群容量，提高集群利用率。此时，用于提高机器利用率的资源调度和治理中心(SOA)是关键。

# 什么是dubbo

## 什么是RPC

了解dubbo之前，了解一下什么是RPC？

RPC(Remote Procedure Call Protocol)：远程过程调用。 两台服务器A、B，分别部署不同的应用a,b。当应用a需要调用应用b提供的函数或方法的时候，由于不在一个内存空间，不能直接调用，需要通过网络来表达调用的语义传达调用的数据。

## RPC需要解决的问题

* 通讯问题

主要是通过在客户端和服务器之间建立TCP连接，远程过程调用的所有交换的数据都在这个连接里传输。连接可以是按需连接，调用结束后就断掉，也可以是长连接，多个远程过程调用共享同一个连接。

* 寻址问题

A服务器上的应用怎么告诉底层的RPC框架，如何连接到B服务器（如主机或IP地址）以及特定的端口，方法的名称名称是什么，这样才能完成调用。比如基于Web服务协议栈的RPC，就要提供一个endpoint URI，或者是从UDDI服务上查找。如果是RMI调用的话，还需要一个RMI Registry来注册服务的地址。

* 序列化与反序列化

当A服务器上的应用发起远程过程调用时，方法的参数需要通过底层的网络协议如TCP传递到B服务器，由于网络协议是基于二进制的，内存中的参数的值要序列化成二进制的形式，也就是序列化（Serialize）或编组（marshal），通过寻址和传输将序列化的二进制发送给B服务器。 同理，B服务器接收参数要将参数反序列化。B服务器应用调用自己的方法处理后返回的结果也要序列化给A服务器，A服务器接收也要经过反序列化的过程。

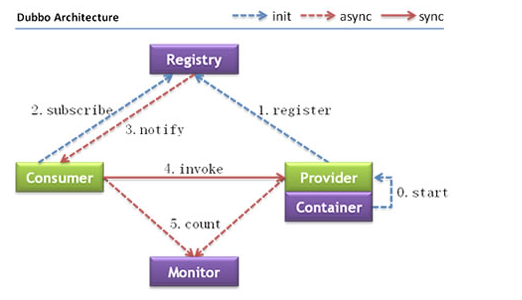
## Dubbo简介及架构图

Dubbo是：

* 一款分布式服务框架
* 高性能和透明化的RPC远程服务调用方案
* SOA服务治理方案

每天为2千多个服务提供大于30亿次访问量支持，并被广泛应用于阿里巴巴集团的各成员站点以及别的公司的业务中。

架构图：



节点角色说明：

Provider: 暴露服务的服务提供方。

Consumer: 调用远程服务的服务消费方。

Registry: 服务注册与发现的注册中心。

Monitor: 统计服务的调用次调和调用时间的监控中心。

Container: 服务运行容器。

调用关系说明：

0 服务容器负责启动，加载，运行服务提供者。

1. 服务提供者在启动时，向注册中心注册自己提供的服务。

2. 服务消费者在启动时，向注册中心订阅自己所需的服务。

3. 注册中心返回服务提供者地址列表给消费者，如果有变更，注册中心将基于长连接推送变更数据给消费者。

4. 服务消费者，从提供者地址列表中，基于软负载均衡算法，选一台提供者进行调用，如果调用失败，再选另一台调用。

5. 服务消费者和提供者，在内存中累计调用次数和调用时间，定时每分钟发送一次统计数据到监控中心。

# 环境准备

## 开发工具

JDK8

maven

IntelliJ IDEA

Springboot

Apache dubbo 2.7.3

Zookeeper

dubbo-admin ops源码

Github

Node.js

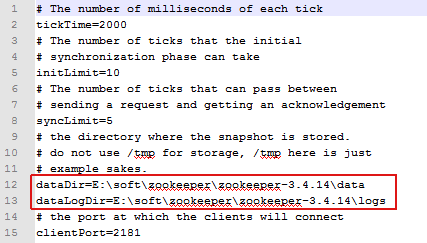
Webpack

npm/cnpm

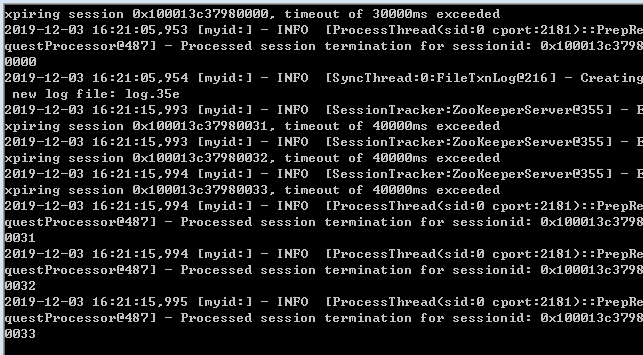
vue

## 启动zookeeper

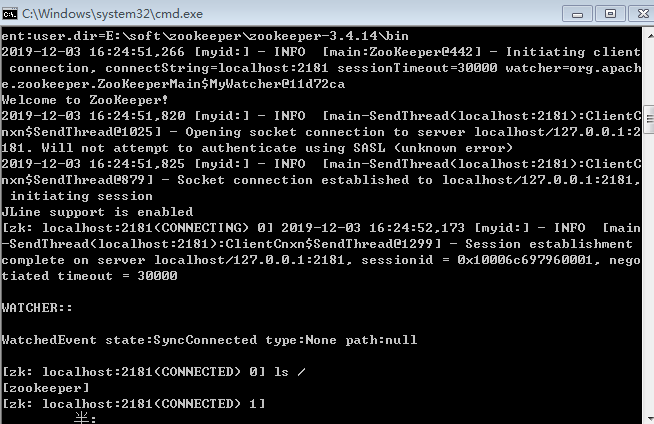
Zookeeper官网下载，解压并进入到conf目录，修改zoo.cfg文件，新增dataDir和dataLogDir目录，如图：



修改完毕进入到bin目录，启动 zkServer.cmd，服务启动成功如下图：



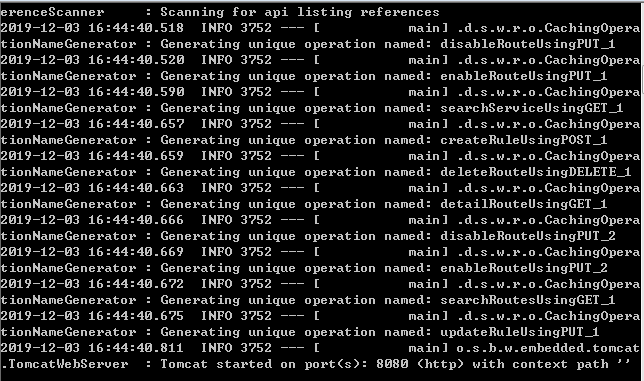
启动zookeeper客户端 zkCli.cmd 查看 根据节点信息，入下图：



## 启动dubbo-admin-server

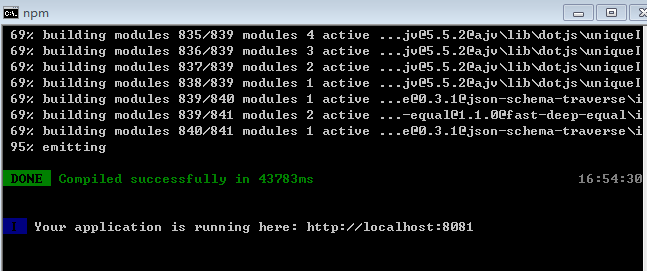
启动命令：java -jar dubbo-admin-server-0.1.jar 启动成功入下图：

启动之后访问 http://localhost:8080/swagger-ui.html

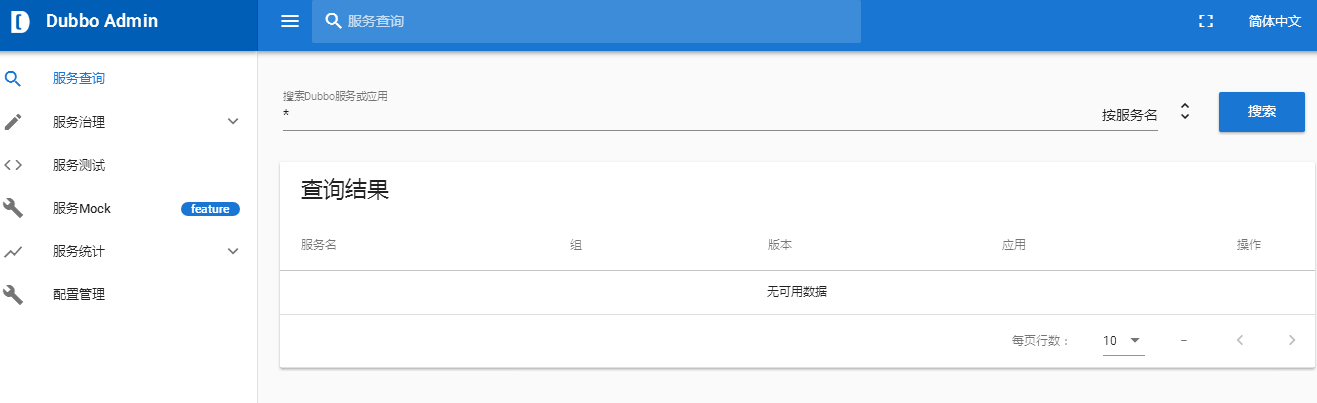


## 启动dubbo-admin-ui

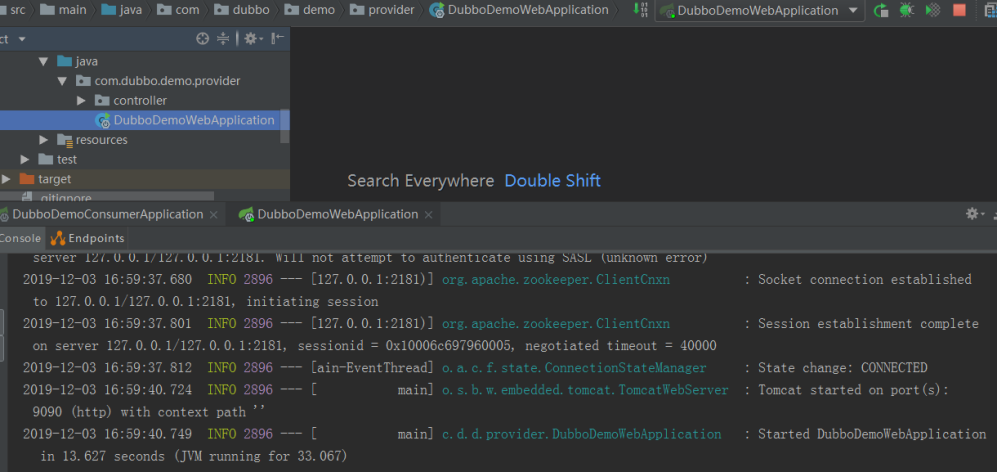
运行命令 npm run dev ，启动成功入下图



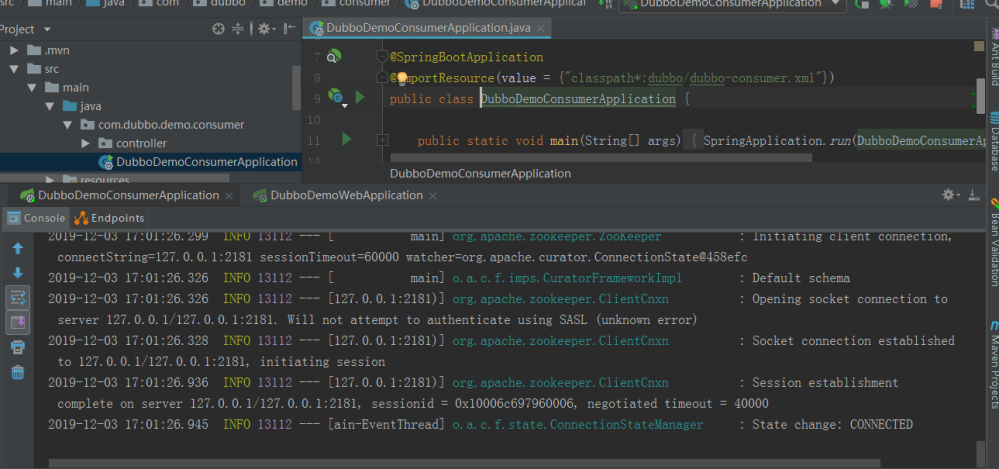
访问： <http://localhost:8081>，dubbo-admin的主界面入下图，因为provider和consumer未启动所以，admin中没有数据。



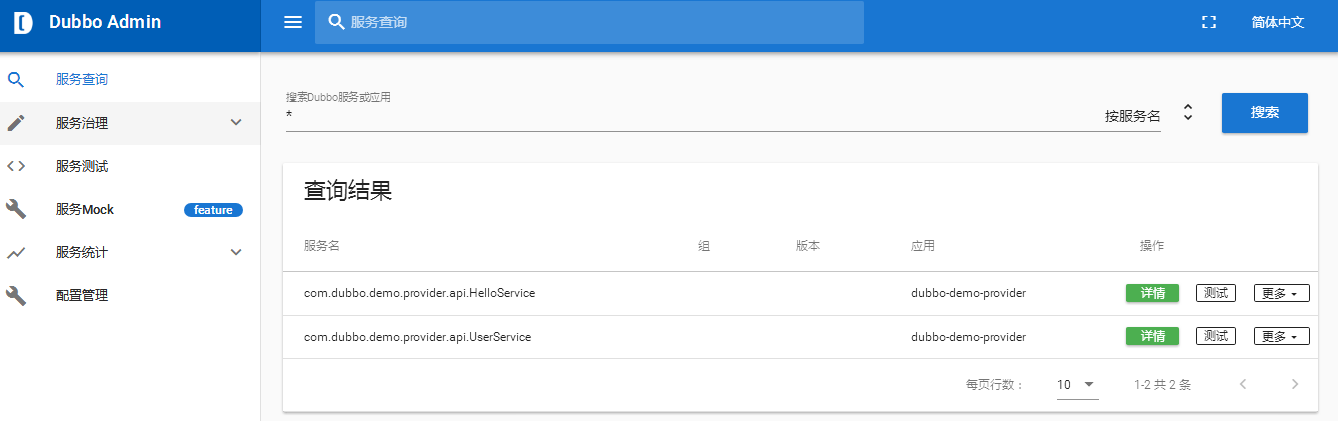
## 启动provider



## 启动consumer



Provider和consumer都启动之后再次访问dubbo-admin就可以看到注册的服务和消费者信息了。



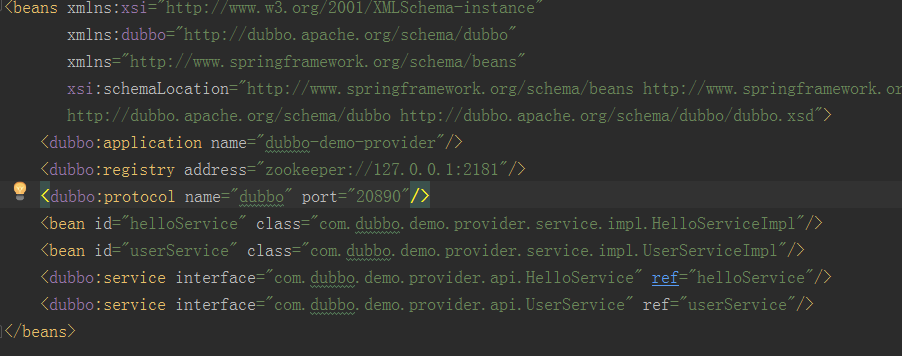
详情：



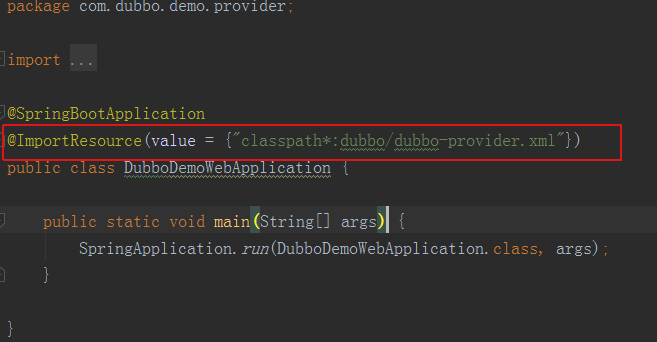
# Dubbo实践

## 服务提供者

dubbo-provider.xml：

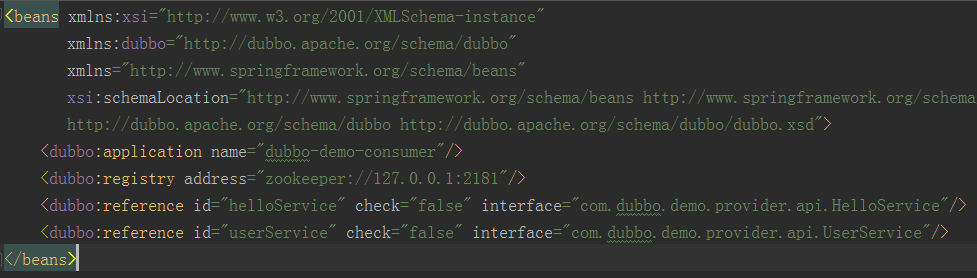


配置文件引入：

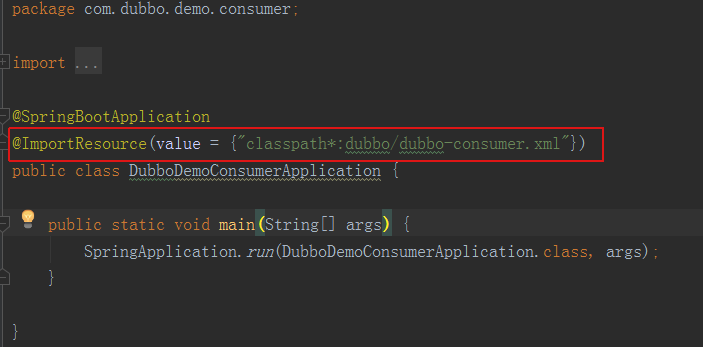


## 服务消费者

dubbo-consumer.xml:



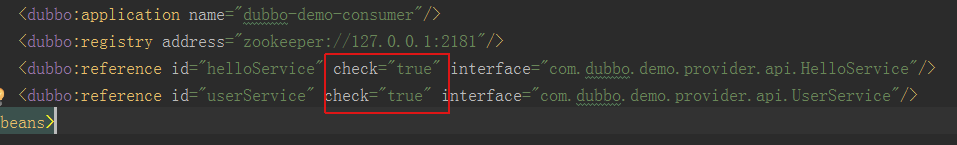
配置文件引入：



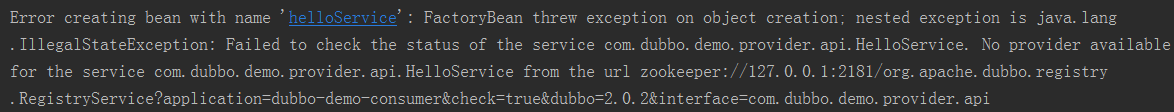
## 启动时检查

Dubbo 缺省会在启动时检查依赖的服务是否可用，不可用时会抛出异常，默认 check="true"。可以通过 check="false" 关闭，当 check="false"，总是会返回引用，当服务恢复时，能自动连上。

关闭服务提供者并修改reference的服务为check="true"或者不配置。

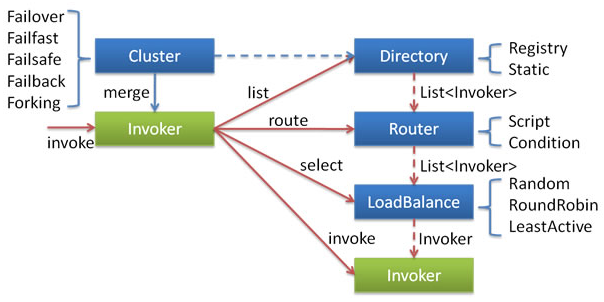


启动consumer则会报错。



修改 check=”false” 则可以启动成功。

## 集群容错



在集群调用失败时，Dubbo 提供了多种容错方案，缺省为 failover 重试。

* 这里的 Invoker 是 Provider 的一个可调用 Service 的抽象，Invoker 封装了 Provider 地址及 Service 接口信息
* Directory 代表多个 Invoker，可以把它看成 List<Invoker> ，但与 List 不同的是，它的值可能是动态变化的，比如注册中心推送变更
* Cluster 将 Directory 中的多个 Invoker 伪装成一个 Invoker，对上层透明，伪装过程包含了容错逻辑，调用失败后，重试另一个
* Router 负责从多个 Invoker 中按路由规则选出子集，比如读写分离，应用隔离等
* LoadBalance 负责从多个 Invoker 中选出具体的一个用于本次调用，选的过程包含了负载均衡算法，调用失败后，需要重选

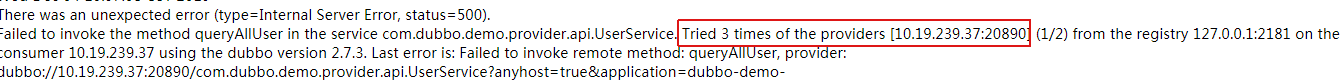
### Failover Cluster

失败自动切换，当出现失败，重试其它服务器，可通过 retries="2" 来设置重试次数(不含第一次)。

设置 retries=”2”,并关闭provider服务。

访问 <http://127.0.0.1:9091/hello/queryAllUser.do>





### Failfast Cluster

快速失败，只发起一次调用，失败立即报错。通常用于非幂等性的写操作。

失败自动切换，当出现失败，重试其它服务器，可通过 retries="2" 来设置重试次数(不含第一次)。

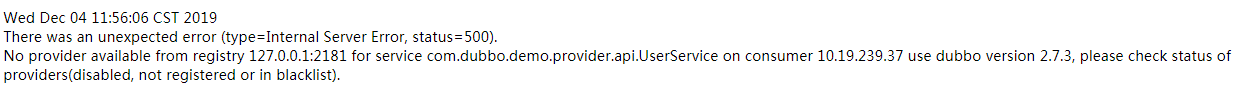
失败自动切换，当出现失败，重试其它服务器，可通过 retries="2" 来设置重试次数(不含第一次)。

失败自动切换，当出现失败，重试其它服务器，可通过 retries="2" 来设置重试次数(不含第一次)。

配置 cluster="failfast"



关闭provider之后访问 <http://127.0.0.1:9091/hello/queryAllUser.do> 抱错如下：



### Failsafe Cluster

失败安全，出现异常时，直接忽略。

配置 cluster="failsafe" 并关闭provider 访问<http://127.0.0.1:9091/hello/queryAllUser.do>

### Failback Cluster

失败自动恢复，后台记录失败请求，定时重发。通常用于消息通知操作

### Forking Cluster

并行调用多个服务器，只要一个成功即返回。通常用于实时性要求较高的读操作，但需要浪费更多服务资源。可通过 forks="2" 来设置最大并行数

### Broadcast Cluster

广播调用所有提供者，逐个调用，任意一台报错则报错。通常用于通知所有提供者更新缓存或日志等本地资源信息。

## 负载均衡

### 随机策略-Random LoadBalance

Dubbo默认使用的此策略，调用量越大分布越均匀。

访问 <http://127.0.0.1:9091/hello/queryAllUser.do> ，会发现请求被随机的分发到两台应用中。

### 轮询策略-RoundRobin LoadBalance

轮询策略 就是平均一对一的分发到每台服务器。



或者在dubbo-admin中配置：

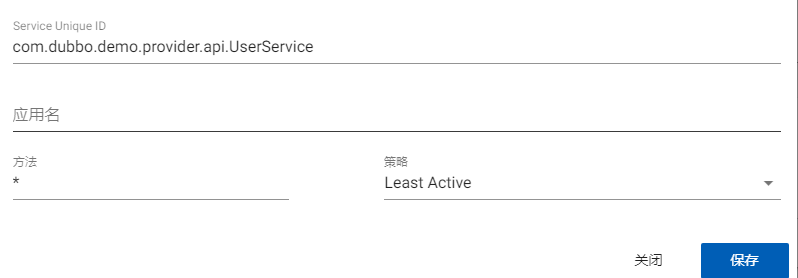


配置完成之后 访问 <http://127.0.0.1:9091/hello/queryAllUser.do> 观察日志会发现请求被轮询的分发到两台服务器。

### 最少活跃调用数-LeastActive LoadBalance

活跃数指调用前后计数差 使慢的提供者收到更少请求，因为越慢的提供者的调用前后计数差会越大。

在dubbo-admin中配置



或者

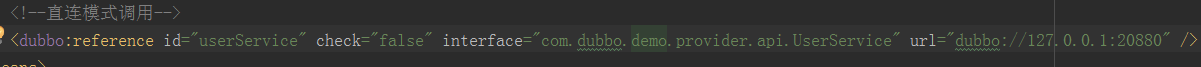


## 直连提供者

在开发及测试环境下，经常需要绕过注册中心，只测试指定服务提供者，这时候可能需要点对点直连服务提供者

设置consumer的配置文件之后访问 <http://127.0.0.1:9091/hello/queryAllUser.do>

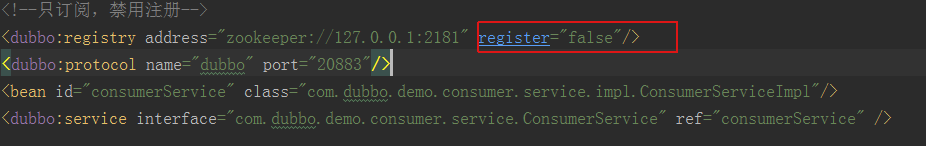
发现请求只会发送指定机器上。



## 只订阅

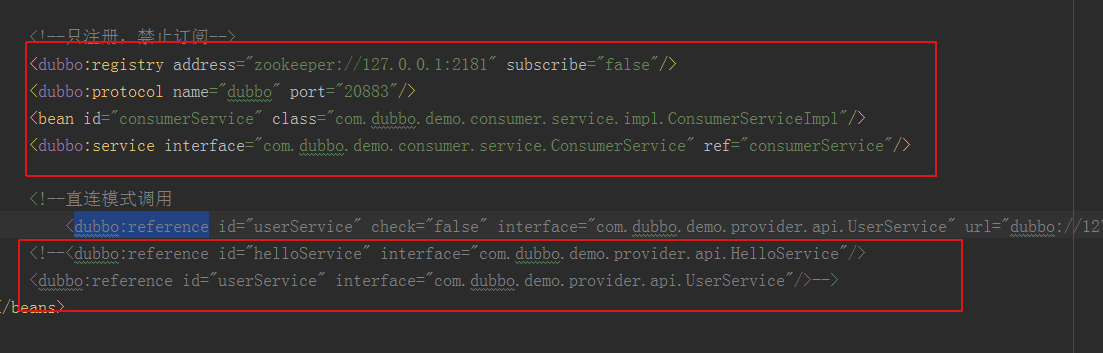
在开发及测试环境下，防止影响现有服务的使用，启用只订阅模式，设置如下。

改为register=”true” 观察admin中的服务注册变动情况。



## 只注册

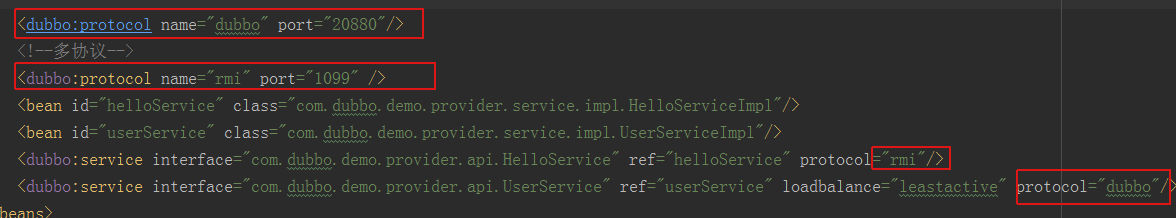
<dubbo:registry address="zookeeper://127.0.0.1:2181" subscribe="false"/> 修改为只注册时 ，发现服务启动不了，因为所有的dubbo:reference 也要注释掉。太假了



## 多协议

Dubbo目前支持的协议有dubbo、rmi、hessian、http、webservice、thrift、memcahed、redis、rest

配置dubbo-provider.xml,同一个服务可以使用多种协议，用逗号分隔。



访问 <http://127.0.0.1:9091/hello/sayHello.do?userName=ninnnn1222> 发现日志中  


在dubbo-admin中也可以看到，对应的协议的端口号

## 多注册中心

可以但单独配置：

<dubbo:registry id="local" address="zookeeper://127.0.0.1:2181"/>  
<dubbo:registry id="zyc" address="zookeeper://10.19.41.133:2181" default="false"/>

也可以这样配置：

<dubbo:registry id="local" address="zookeeper://127.0.0.1:2181,zookeeper://10.19.41.133:2181"/>

指定注册中心，不指定，默认注册到所有的注册中心，可以达到一个服务使用多个注册中心或者多个服务使用不通的注册中心。



## 服务分组

当一个接口有多种实现时，可以用 group 区分。group="\*" 多个组则用逗号分隔

使用设置group属性

<dubbo:service group="user" interface="com.dubbo.demo.provider.api.UserService" ref="userService"/>

在admin中可以查看分组信息，访问

<http://127.0.0.1:9091/hello/queryAllUser.do> 发现报错，找不到服务，因为，在consumer端也要执行group信息，之后访问就可以找到服务。

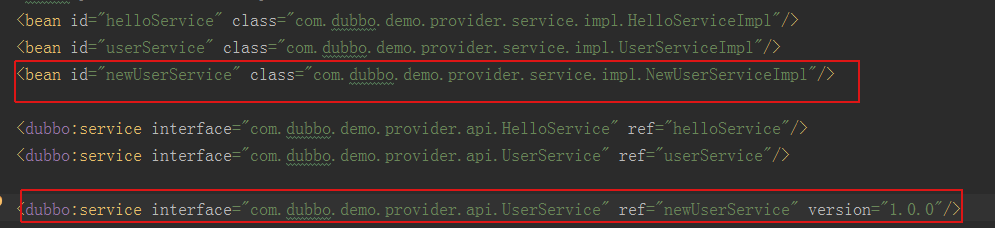
## 多版本

当一个接口实现，出现不兼容升级时，可以用版本号过渡，版本号不同的服务相互间不引用。

在dubbo-provider.xml文件中配置新版本的service，在dubbo-consumer.xml中指定版本号调用。

访问 <http://127.0.0.1:9091/hello/queryAllUser.do>

dubbo-provider.xml:



dubbo-consumer.xml:



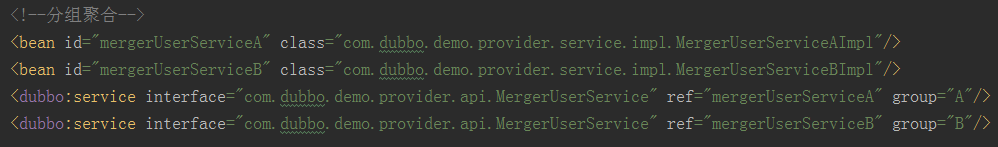
## 分组聚合

返回一个接口多种实现的结果组合

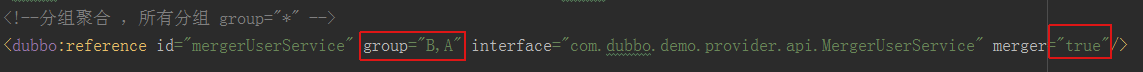
访问：<http://127.0.0.1:9091/hello/queryUserForMerger.do>，可以看到返回结果包含

mergerUserServiceA，mergerUserServiceB的结果。

dubbo-provider.xml



dubbo-comsumer.xml



## 结果缓存

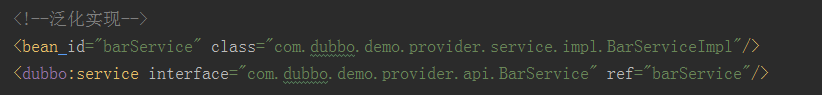
参考dubbo官网文档

## 泛化引用与泛化实现

泛化实现：方法入参类型为Map

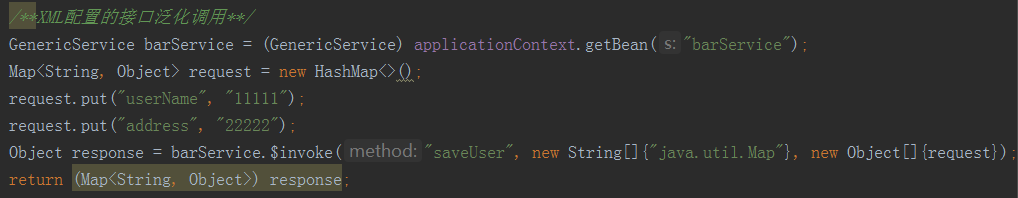
访问：<http://127.0.0.1:9091/hello/saveUserForGeneric.do>



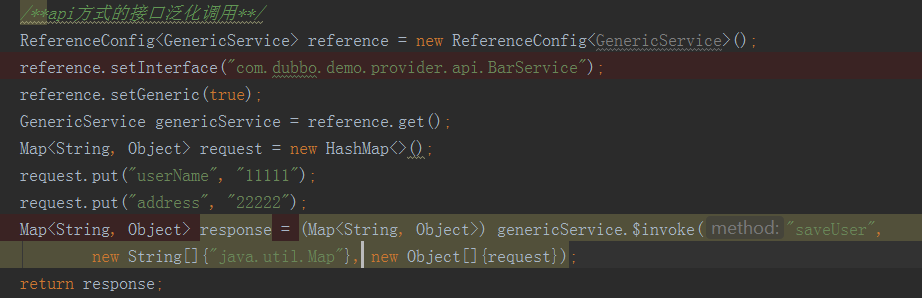


Xml配置泛化调用：





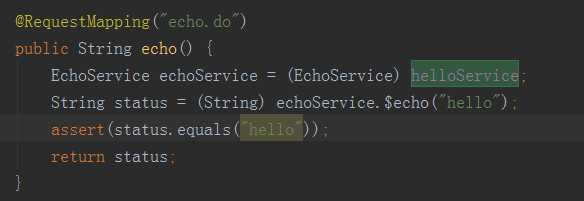
Api形式泛化调用：



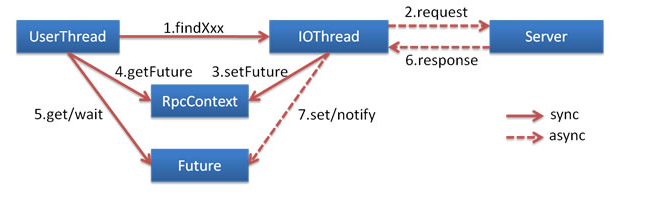
## 回声测试

回声测试用于检测服务是否可用，回声测试按照正常请求流程执行，能够测试整个调用是否通畅，可用于监控。

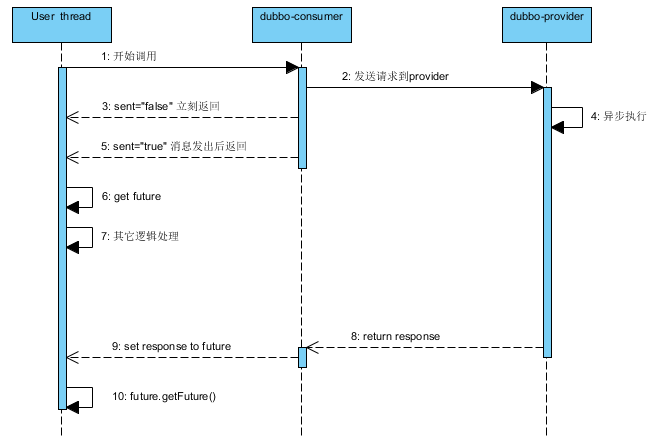
访问 <http://127.0.0.1:9091/hello/echo.do>



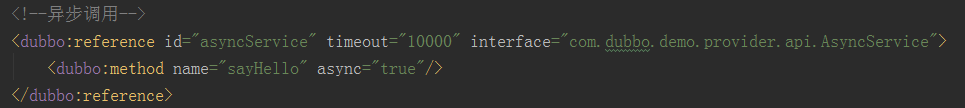
## Consumer异步调用



时序图：



Dubbo-consumer.xml配置：



Consumer端代码



访问 <http://127.0.0.1:9091/hello/async.do> 之后观察日志输出：

异步调用...

可以处理器其他逻辑

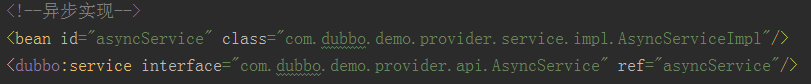
先与返回结果执行.

## Provider异步执行

Provider端代码：



Provider端的配置文件：



## 延迟暴露

如果服务需要预热时间，比如初始化缓存，等待相关资源就位等，可以使用 delay 进行延迟暴，延迟暴露是在服务端配置的，对客户端的调用是无感知的。

配置比较简单入下图。



## 动态配置

# 参考资料

Demo源码地址：

<https://github.com/lsh0721/dubbo-demo.git>

[git@github.com:lsh0721/dubbo-demo.git](mailto:git@github.com:lsh0721/dubbo-demo.git)

Dubbo参考文档：<http://dubbo.apache.org/zh-cn/docs/user/quick-start.html>

dubbo-admin ops源码地址：<https://github.com/apache/dubbo-ops>