

服务治理之 dubbo 实践

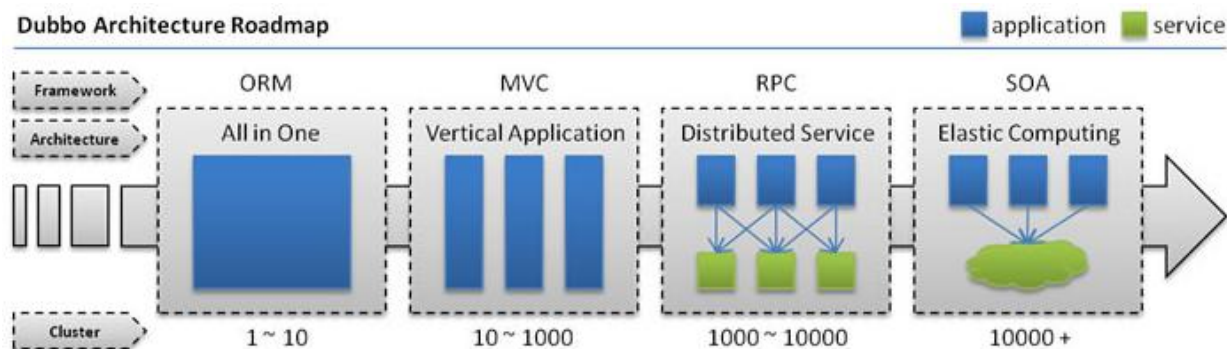
目录

- 1 dubbo 背景简介..... 4
- 2 什么是 dubbo..... 5
 - 2.1 什么是 RPC..... 5
 - 2.2 RPC 需要解决的问题..... 5
 - 2.3 Dubbo 简介及架构图..... 5
- 3 环境准备..... 7
 - 3.1 开发工具..... 7
 - 3.2 启动 zookeeper..... 7
 - 3.3 启动 dubbo-admin-server..... 8
 - 3.4 启动 dubbo-admin-ui..... 9
 - 3.5 启动 provider..... 10
 - 3.6 启动 consumer..... 10
- 4 Dubbo 实践..... 12
 - 4.1 服务提供者..... 12
 - 4.2 服务消费者..... 12
 - 4.3 启动时检查..... 13
 - 4.4 集群容错..... 14
 - 4.4.1 Failover Cluster..... 14
 - 4.4.2 Failfast Cluster..... 14
 - 4.4.3 Failsafe Cluster..... 15
 - 4.4.4 Failback Cluster..... 15
 - 4.4.5 Forking Cluster..... 15
 - 4.4.6 Broadcast Cluster..... 15
 - 4.5 负载均衡..... 15
 - 4.5.1 随机策略-Random LoadBalance..... 15
 - 4.5.2 轮询策略-RoundRobin LoadBalance..... 15
 - 4.5.3 最少活跃调用数-LeastActive LoadBalance..... 16
 - 4.6 直连提供者..... 17
 - 4.7 只订阅..... 17
 - 4.8 只注册..... 17

- 4.9 多协议..... 17
- 4.10 多注册中心..... 18
- 4.11 服务分组..... 18
- 4.12 多版本..... 19
- 4.13 分组聚合..... 19
- 4.14 结果缓存..... 20
- 4.15 泛化引用与泛化实现..... 20
- 4.16 回声测试..... 20
- 4.17 Consumer 异步调用..... 21
- 4.18 Provider 异步执行..... 22
- 4.19 延迟暴露..... 23
- 4.20 动态配置..... 23
- 5 参考资料..... 24

1 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。

2 什么是 dubbo

2.1 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。

2.2 RPC 需要解决的问题

- 通讯问题

主要是通过客户端和服务端之间建立 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。

2.3 Dubbo 简介及架构图

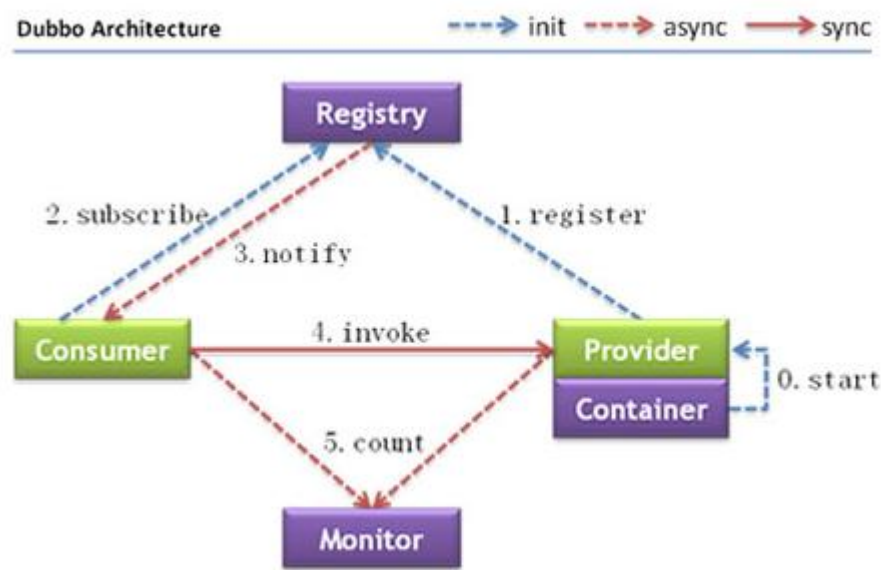
Dubbo 是：

- 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。

3 环境准备

3.1 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

3.2 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：

```
1  # The number of milliseconds of each tick
2  tickTime=2000
3  # The number of ticks that the initial
4  # synchronization phase can take
5  initLimit=10
6  # The number of ticks that can pass between
7  # sending a request and getting an acknowledgement
8  syncLimit=5
9  # the directory where the snapshot is stored.
10 # do not use /tmp for storage, /tmp here is just
11 # example sakes.
12 dataDir=E:\soft\zookeeper\zookeeper-3.4.14\data
13 dataLogDir=E:\soft\zookeeper\zookeeper-3.4.14\logs
14 # the port at which the clients will connect
15 clientPort=2181
```

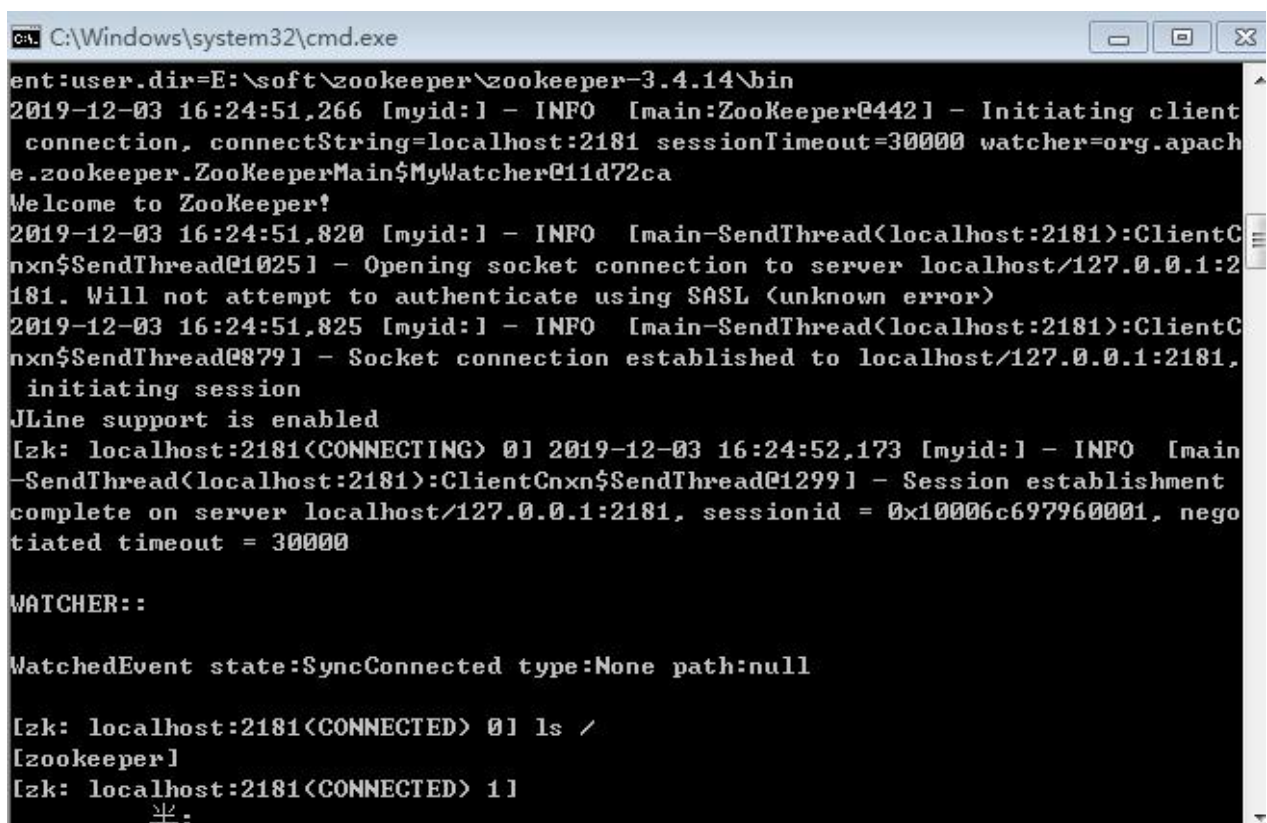
修改完毕进入到 bin 目录，启动 zkServer.cmd，服务启动成功如下图：

```

xpiring session 0x100013c37980000, timeout of 30000ms exceeded
2019-12-03 16:21:05,953 [myid:] - INFO [ProcessThread(sid:0 cport:2181)::PrepRe
questProcessor@487] - Processed session termination for sessionid: 0x100013c3798
0000
2019-12-03 16:21:05,954 [myid:] - INFO [SyncThread:0:FileTxnLog@216] - Creating
new log file: log.35e
2019-12-03 16:21:15,993 [myid:] - INFO [SessionTracker:ZooKeeperServer@355] - E
xpiring session 0x100013c37980031, timeout of 40000ms exceeded
2019-12-03 16:21:15,993 [myid:] - INFO [SessionTracker:ZooKeeperServer@355] - E
xpiring session 0x100013c37980032, timeout of 40000ms exceeded
2019-12-03 16:21:15,994 [myid:] - INFO [SessionTracker:ZooKeeperServer@355] - E
xpiring session 0x100013c37980033, timeout of 40000ms exceeded
2019-12-03 16:21:15,994 [myid:] - INFO [ProcessThread(sid:0 cport:2181)::PrepRe
questProcessor@487] - Processed session termination for sessionid: 0x100013c3798
0031
2019-12-03 16:21:15,994 [myid:] - INFO [ProcessThread(sid:0 cport:2181)::PrepRe
questProcessor@487] - Processed session termination for sessionid: 0x100013c3798
0032
2019-12-03 16:21:15,995 [myid:] - INFO [ProcessThread(sid:0 cport:2181)::PrepRe
questProcessor@487] - Processed session termination for sessionid: 0x100013c3798
0033

```

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



```

C:\Windows\system32\cmd.exe
ent:user.dir=E:\soft\zookeeper\zookeeper-3.4.14\bin
2019-12-03 16:24:51,266 [myid:] - INFO [main:ZooKeeper@442] - Initiating client
connection, connectString=localhost:2181 sessionTimeout=30000 watcher=org.apach
e.zookeeper.ZooKeeperMain$MyWatcher@11d72ca
Welcome to ZooKeeper!
2019-12-03 16:24:51,820 [myid:] - INFO [main-SendThread<localhost:2181>:ClientC
nxn$SendThread@1025] - Opening socket connection to server localhost/127.0.0.1:2
181. Will not attempt to authenticate using SASL (unknown error)
2019-12-03 16:24:51,825 [myid:] - INFO [main-SendThread<localhost:2181>:ClientC
nxn$SendThread@879] - Socket connection established to localhost/127.0.0.1:2181,
initiating session
JLine support is enabled
[zk: localhost:2181(CONNECTING) 0] 2019-12-03 16:24:52,173 [myid:] - INFO [main
-SendThread<localhost:2181>:ClientCnxn$SendThread@1299] - Session establishment
complete on server localhost/127.0.0.1:2181, sessionId = 0x10006c697960001, nego
tiated timeout = 30000

WATCHER::

WatchedEvent state:SyncConnected type:None path:null

[zk: localhost:2181(CONNECTED) 0] ls /
[zookeeper]
[zk: localhost:2181(CONNECTED) 1]
quit;

```

3.3 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>

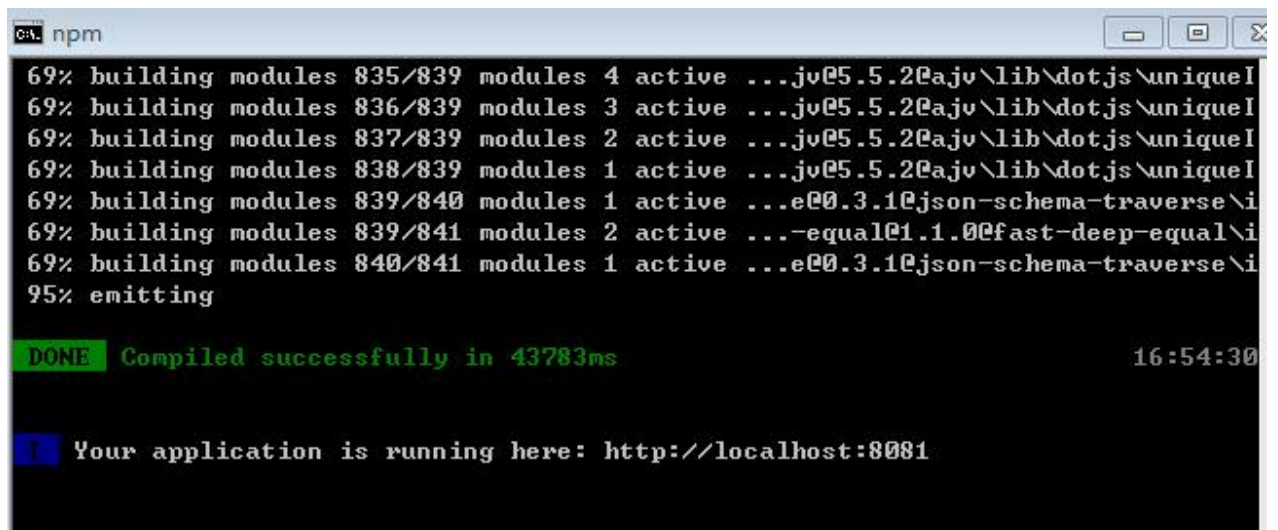

```

erenceScanner      : Scanning for api listing references
2019-12-03 16:44:40.518 INFO 3752 --- [main] .d.s.w.r.o.CachingOpera
tionNameGenerator : Generating unique operation named: disableRouteUsingPUT_1
2019-12-03 16:44:40.520 INFO 3752 --- [main] .d.s.w.r.o.CachingOpera
tionNameGenerator : Generating unique operation named: enableRouteUsingPUT_1
2019-12-03 16:44:40.590 INFO 3752 --- [main] .d.s.w.r.o.CachingOpera
tionNameGenerator : Generating unique operation named: searchServiceUsingGET_1
2019-12-03 16:44:40.657 INFO 3752 --- [main] .d.s.w.r.o.CachingOpera
tionNameGenerator : Generating unique operation named: createRuleUsingPOST_1
2019-12-03 16:44:40.659 INFO 3752 --- [main] .d.s.w.r.o.CachingOpera
tionNameGenerator : Generating unique operation named: deleteRouteUsingDELETE_1
2019-12-03 16:44:40.663 INFO 3752 --- [main] .d.s.w.r.o.CachingOpera
tionNameGenerator : Generating unique operation named: detailRouteUsingGET_1
2019-12-03 16:44:40.666 INFO 3752 --- [main] .d.s.w.r.o.CachingOpera
tionNameGenerator : Generating unique operation named: disableRouteUsingPUT_2
2019-12-03 16:44:40.669 INFO 3752 --- [main] .d.s.w.r.o.CachingOpera
tionNameGenerator : Generating unique operation named: enableRouteUsingPUT_2
2019-12-03 16:44:40.672 INFO 3752 --- [main] .d.s.w.r.o.CachingOpera
tionNameGenerator : Generating unique operation named: searchRoutesUsingGET_1
2019-12-03 16:44:40.675 INFO 3752 --- [main] .d.s.w.r.o.CachingOpera
tionNameGenerator : Generating unique operation named: updateRuleUsingPUT_1
2019-12-03 16:44:40.811 INFO 3752 --- [main] o.s.b.w.embedded.tomcat
.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''

```

3.4 启动 dubbo-admin-ui

运行命令 `npm run dev`，启动成功入下图



```

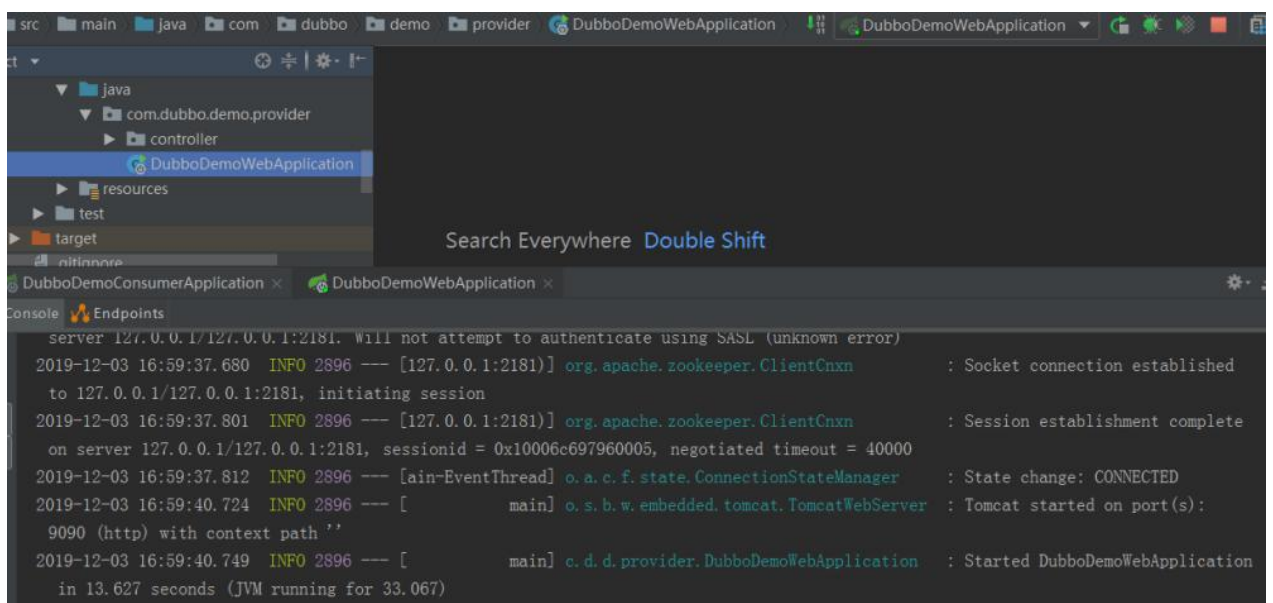
ca npm
69% building modules 835/839 modules 4 active ...jv@5.5.2@ajv\lib\dotjs\uniqueI
69% building modules 836/839 modules 3 active ...jv@5.5.2@ajv\lib\dotjs\uniqueI
69% building modules 837/839 modules 2 active ...jv@5.5.2@ajv\lib\dotjs\uniqueI
69% building modules 838/839 modules 1 active ...jv@5.5.2@ajv\lib\dotjs\uniqueI
69% building modules 839/840 modules 1 active ...e@0.3.1@json-schema-traverse\i
69% building modules 839/841 modules 2 active ...-equal@1.1.0@fast-deep-equal\i
69% building modules 840/841 modules 1 active ...e@0.3.1@json-schema-traverse\i
95% emitting
DONE Compiled successfully in 43783ms 16:54:30
Your application is running here: http://localhost:8081

```

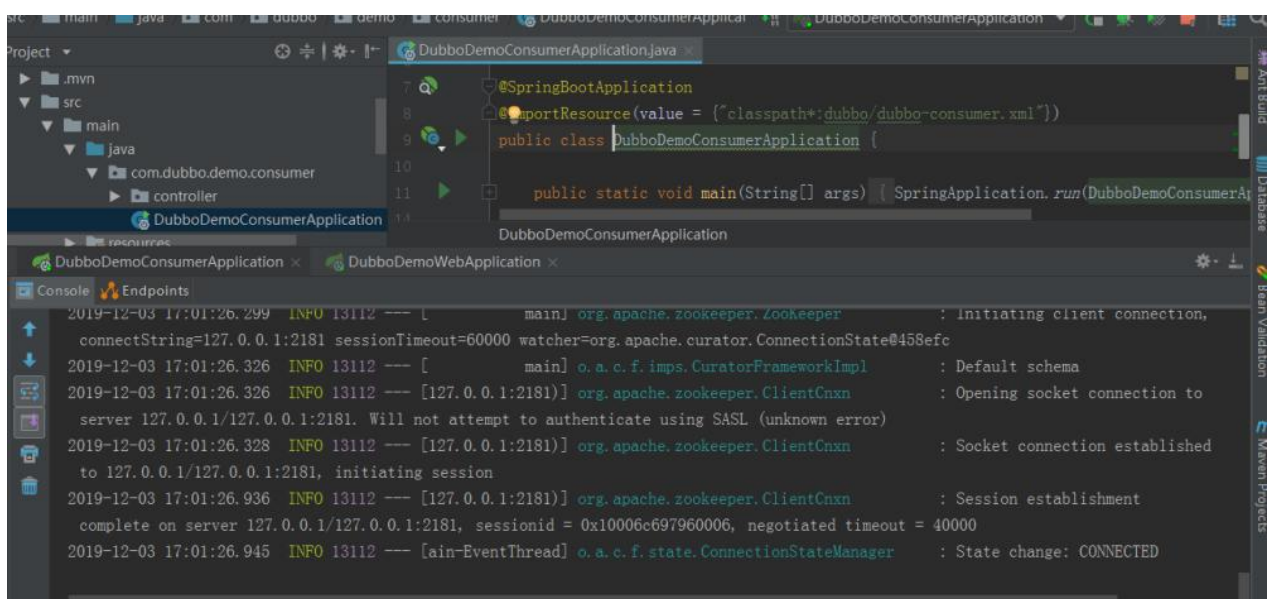
访问：<http://localhost:8081>，dubbo-admin 的主界面入下图，因为 provider 和 consumer 未启动所以，admin 中没有数据。



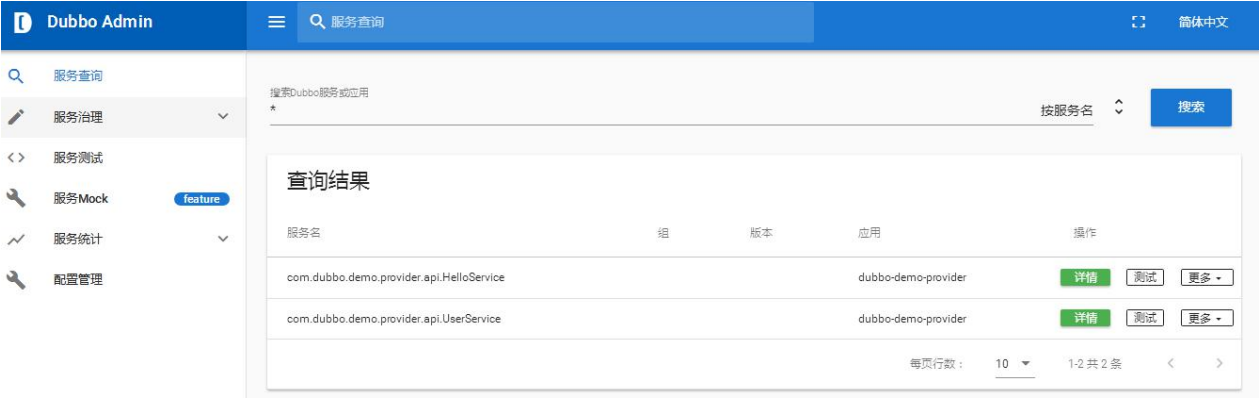
3.5 启动 provider



3.6 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：

服务信息

提供者 消费者

IP地址	端口	超时(毫秒)	序列化	权重	操作
10.19.239.37	20890			100	URL

每页行数：5 1-1 共 1 条

元数据

方法名	参数列表	返回值
queryAllUser		java.util.List<com.dubbo.demo.provider.dmo.User>
deleteUser	java.lang.String	void

4 Dubbo 实践

4.1 服务提供者

dubbo-provider.xml:

```
<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:dubbo="http://dubbo.apache.org/schema/dubbo"
  xmlns="http://www.springframework.org/schema/beans"
  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans.xsd
    http://dubbo.apache.org/schema/dubbo http://dubbo.apache.org/schema/dubbo/dubbo.xsd">
  <dubbo:application name="dubbo-demo-provider"/>
  <dubbo:registry address="zookeeper://127.0.0.1:2181"/>
  <dubbo:protocol name="dubbo" port="20890"/>
  <bean id="helloService" class="com.dubbo.demo.provider.service.impl.HelloServiceImpl"/>
  <bean id="userService" class="com.dubbo.demo.provider.service.impl.UserServiceImpl"/>
  <dubbo:service interface="com.dubbo.demo.provider.api.HelloService" ref="helloService"/>
  <dubbo:service interface="com.dubbo.demo.provider.api.UserService" ref="userService"/>
</beans>
```

配置文件引入:

```
package com.dubbo.demo.provider;

import ...

@SpringBootApplication
@ImportResource(value = {"classpath*:dubbo/dubbo-provider.xml"})
public class DubboDemoWebApplication {

    public static void main(String[] args) {
        SpringApplication.run(DubboDemoWebApplication.class, args);
    }
}
```

4.2 服务消费者

dubbo-consumer.xml:


```
<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:dubbo="http://dubbo.apache.org/schema/dubbo"
        xmlns="http://www.springframework.org/schema/beans"
        xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema
        http://dubbo.apache.org/schema/dubbo http://dubbo.apache.org/schema/dubbo/dubbo.xsd">
    <dubbo:application name="dubbo-demo-consumer"/>
    <dubbo:registry address="zookeeper://127.0.0.1:2181"/>
    <dubbo:reference id="helloService" check="false" interface="com.dubbo.demo.provider.api.HelloService"/>
    <dubbo:reference id="userService" check="false" interface="com.dubbo.demo.provider.api.UserService"/>
</beans>
```

配置文件引入：

```
package com.dubbo.demo.consumer;

import ...

@SpringBootApplication
@ImportResource(value = {"classpath*:dubbo/dubbo-consumer.xml"})
public class DubboDemoConsumerApplication {

    public static void main(String[] args) {
        SpringApplication.run(DubboDemoConsumerApplication.class, args);
    }
}
```

4.3 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。

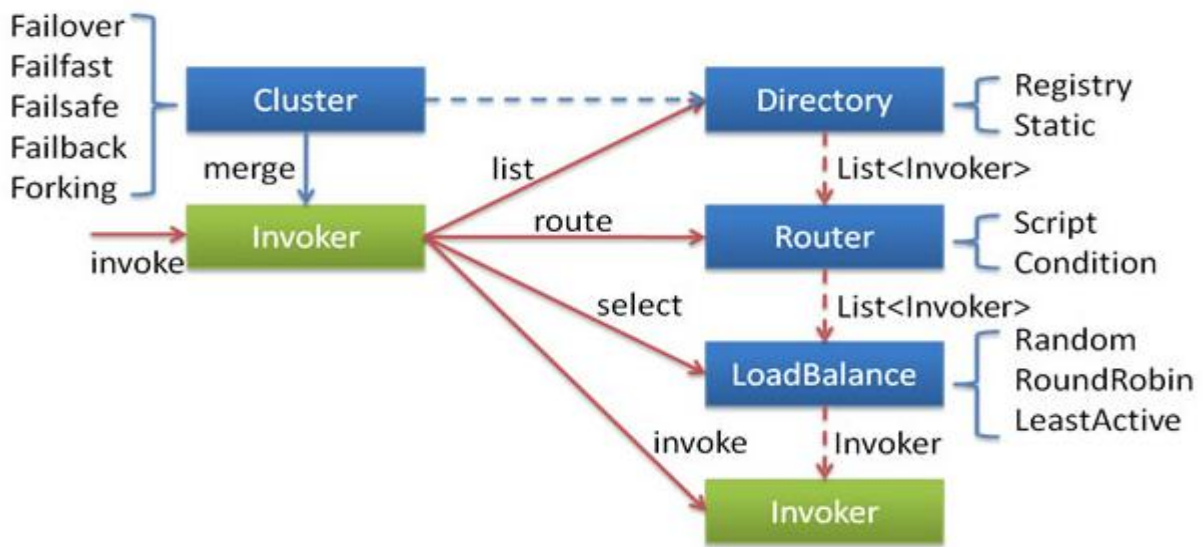
```
<dubbo:application name="dubbo-demo-consumer"/>
<dubbo:registry address="zookeeper://127.0.0.1:2181"/>
<dubbo:reference id="helloService" check="true" interface="com.dubbo.demo.provider.api.HelloService"/>
<dubbo:reference id="userService" check="true" interface="com.dubbo.demo.provider.api.UserService"/>
</beans>
```

启动 consumer 则会报错。

```
Error creating bean with name 'helloService': FactoryBean threw exception on object creation; nested exception is java.lang
.IllegalStateException: Failed to check the status of the service com.dubbo.demo.provider.api.HelloService. No provider available
for the service com.dubbo.demo.provider.api.HelloService from the url zookeeper://127.0.0.1:2181/org.apache.dubbo.registry
.RegistryService?application=dubbo-demo-consumer&check=true&dubbo=2.0.2&interface=com.dubbo.demo.provider.api
```

修改 `check="false"` 则可以启动成功。

4.4 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

4.4.1 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>

```
<dubbo:reference id="userService" check="true" interface="com.dubbo.demo.provider.api.UserService" retries="2"/>
```

There was an unexpected error (type=Internal Server Error, status=500).
 Failed to invoke the method queryAllUser in the service com.dubbo.demo.provider.api.UserService. Tried 3 times of the providers [10.19.239.37:20890] (1/2) from the registry 127.0.0.1:2181 on the consumer 10.19.239.37 using the dubbo version 2.7.3. Last error is: Failed to invoke remote method: queryAllUser, provider: dubbo://10.19.239.37:20890/com.dubbo.demo.provider.api.UserService?anyhost=true&application=dubbo-demo-

4.4.2 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。

配置 cluster="failfast"

```
<dubbo:reference id="userService" check="true" interface="com.dubbo.demo.provider.api.UserService" cluster="failfast"/>
```

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

```
Wed Dec 04 11:56:06 CST 2019
There was an unexpected error (type=Internal Server Error, status=500).
No provider available from registry 127.0.0.1:2181 for service com.dubbo.demo.provider.api.UserService on consumer 10.19.239.37 use dubbo version 2.7.3, please check status of providers(disabled, not registered or in blacklist).
```

4.4.3 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>

4.4.4 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

4.4.5 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

4.4.6 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。

4.5 负载均衡

4.5.1 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。

4.5.2 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。

```
<dubbo:service interface="com.dubbo.demo.provider.api.UserService" ref="userService" loadbalance="roundrobin"/>
```

或者在 dubbo-admin 中配置：

新建负载均衡规则

Service Unique ID

com.dubbo.demo.provider.api.UserService

应用名

方法

*

策略

Round Robin

关闭

保存

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。

4.5.3 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

Service Unique ID

com.dubbo.demo.provider.api.UserService

应用名

方法

*

策略

Least Active

关闭

保存

或者

```
<dubbo:service interface="com.dubbo.demo.provider.api.UserService" ref="userService" loadbalance="leastactive"/>
```


4.6 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。

```
<!--直连模式调用-->
<dubbo:reference id="userService" check="false" interface="com.dubbo.demo.provider.api.UserService" url="dubbo://127.0.0.1:20880" />
</dubbo>
```

4.7 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。

```
<!--只订阅，禁用注册-->
<dubbo:registry address="zookeeper://127.0.0.1:2181" register="false"/>
<dubbo:protocol name="dubbo" port="20883"/>
<bean id="consumerService" class="com.dubbo.demo.consumer.service.impl.ConsumerServiceImpl"/>
<dubbo:service interface="com.dubbo.demo.consumer.service.ConsumerService" ref="consumerService" />
```

4.8 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

```
<!--只注册，禁止订阅-->
<dubbo:registry address="zookeeper://127.0.0.1:2181" subscribe="false"/>
<dubbo:protocol name="dubbo" port="20883"/>
<bean id="consumerService" class="com.dubbo.demo.consumer.service.impl.ConsumerServiceImpl"/>
<dubbo:service interface="com.dubbo.demo.consumer.service.ConsumerService" ref="consumerService"/>

<!--直连模式调用
<dubbo:reference id="userService" check="false" interface="com.dubbo.demo.provider.api.UserService" url="dubbo://127.0.0.1:20880" />
<!--<dubbo:reference id="helloService" interface="com.dubbo.demo.provider.api.HelloService"/>
<dubbo:reference id="userService" interface="com.dubbo.demo.provider.api.UserService"/>-->
</beans>
```

4.9 多协议

Dubbo 目前支持的协议有 dubbo、rmi、hessian、http、webservice、thrift、memcached、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。

```
<dubbo:protocol name="dubbo" port="20880"/>
<!--多协议-->
<dubbo:protocol name="rmi" port="1099" />
<bean id="helloService" class="com.dubbo.demo.provider.service.impl.HelloServiceImpl"/>
<bean id="userService" class="com.dubbo.demo.provider.service.impl.UserServiceImpl"/>
<dubbo:service interface="com.dubbo.demo.provider.api.HelloService" ref="helloService" protocol="rmi"/>
<dubbo:service interface="com.dubbo.demo.provider.api.UserService" ref="userService" loadbalance="leastactive" protocol="dubbo"/>
</beans>
```

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

```
$.RemoteInvocationTraceInterceptor - Finished processing of RmiServiceExporter remote call: com.dubbo.demo.provider.api?
```

在 dubbo-admin 中也可以看到, 对应的协议的端口号

4.10 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。

```
<dubbo:service interface="com.dubbo.demo.provider.api.HelloService" ref="helloService" registry="local"/>
<dubbo:service interface="com.dubbo.demo.provider.api.UserService" ref="userService" registry="local,zyc"/>
```

4.11 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。

4.12 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:

```
<bean id="helloService" class="com.dubbo.demo.provider.service.impl.HelloServiceImpl"/>
<bean id="userService" class="com.dubbo.demo.provider.service.impl.UserServiceImpl"/>
<bean id="newUserService" class="com.dubbo.demo.provider.service.impl.NewUserServiceImpl"/>

<dubbo:service interface="com.dubbo.demo.provider.api.HelloService" ref="helloService"/>
<dubbo:service interface="com.dubbo.demo.provider.api.UserService" ref="userService"/>
<dubbo:service interface="com.dubbo.demo.provider.api.UserService" ref="newUserService" version="1.0.0"/>
```

dubbo-consumer.xml:

```
<dubbo:reference id="userService" interface="com.dubbo.demo.provider.api.UserService" version="1.0.0"/>
```

4.13 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

```
<!--分组聚合-->
<bean id="mergerUserServiceA" class="com.dubbo.demo.provider.service.impl.MergerUserServiceAImpl"/>
<bean id="mergerUserServiceB" class="com.dubbo.demo.provider.service.impl.MergerUserServiceBImpl"/>
<dubbo:service interface="com.dubbo.demo.provider.api.MergerUserService" ref="mergerUserServiceA" group="A"/>
<dubbo:service interface="com.dubbo.demo.provider.api.MergerUserService" ref="mergerUserServiceB" group="B"/>
```

dubbo-comsumer.xml

```
<!--分组聚合，所有分组 group="*" -->
<dubbo:reference id="mergerUserService" group="B,A" interface="com.dubbo.demo.provider.api.MergerUserService" merger="true"/>
```

4.14 结果缓存

参考 dubbo 官网文档

4.15 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>

```
Map<String, Object> saveUser(Map<String, Object> request);
```

<!--泛化实现-->

```
<bean id="barService" class="com.dubbo.demo.provider.service.impl.BarServiceImpl"/>
<dubbo:service interface="com.dubbo.demo.provider.api.BarService" ref="barService"/>
```

Xml 配置泛化调用：

<!--泛化引用-->

```
<dubbo:reference id="barService" interface="com.dubbo.demo.provider.api.BarService" generic="true" />
```

/**XML配置的接口泛化调用**/

```
GenericService barService = (GenericService) applicationContext.getBean("barService");
Map<String, Object> request = new HashMap<>();
request.put("userName", "11111");
request.put("address", "22222");
Object response = barService.$invoke("saveUser", new String[]{"java.util.Map"}, new Object[]{request});
return (Map<String, Object>) response;
```

Api 形式泛化调用：

/**api方式的接口泛化调用**/

```
ReferenceConfig<GenericService> reference = new ReferenceConfig<GenericService>();
reference.setInterface("com.dubbo.demo.provider.api.BarService");
reference.setGeneric(true);
GenericService genericService = reference.get();
Map<String, Object> request = new HashMap<>();
request.put("userName", "11111");
request.put("address", "22222");
Map<String, Object> response = (Map<String, Object>) genericService.$invoke("saveUser",
    new String[]{"java.util.Map"}, new Object[]{request});
return response;
```

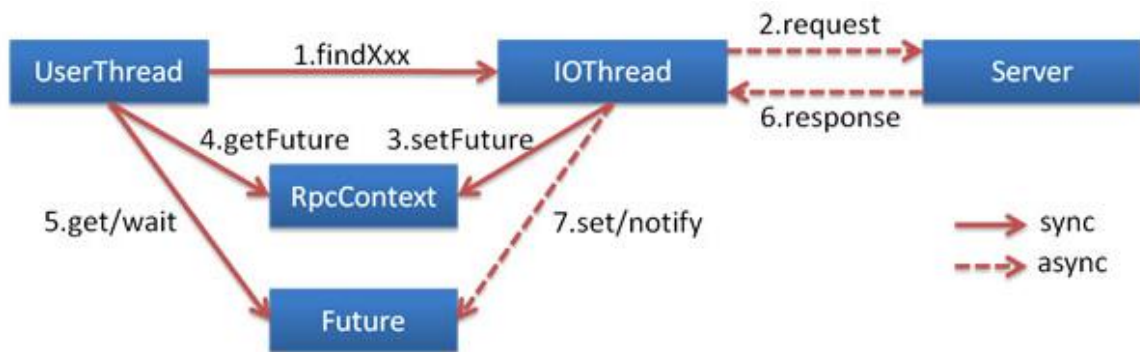
4.16 回声测试

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。

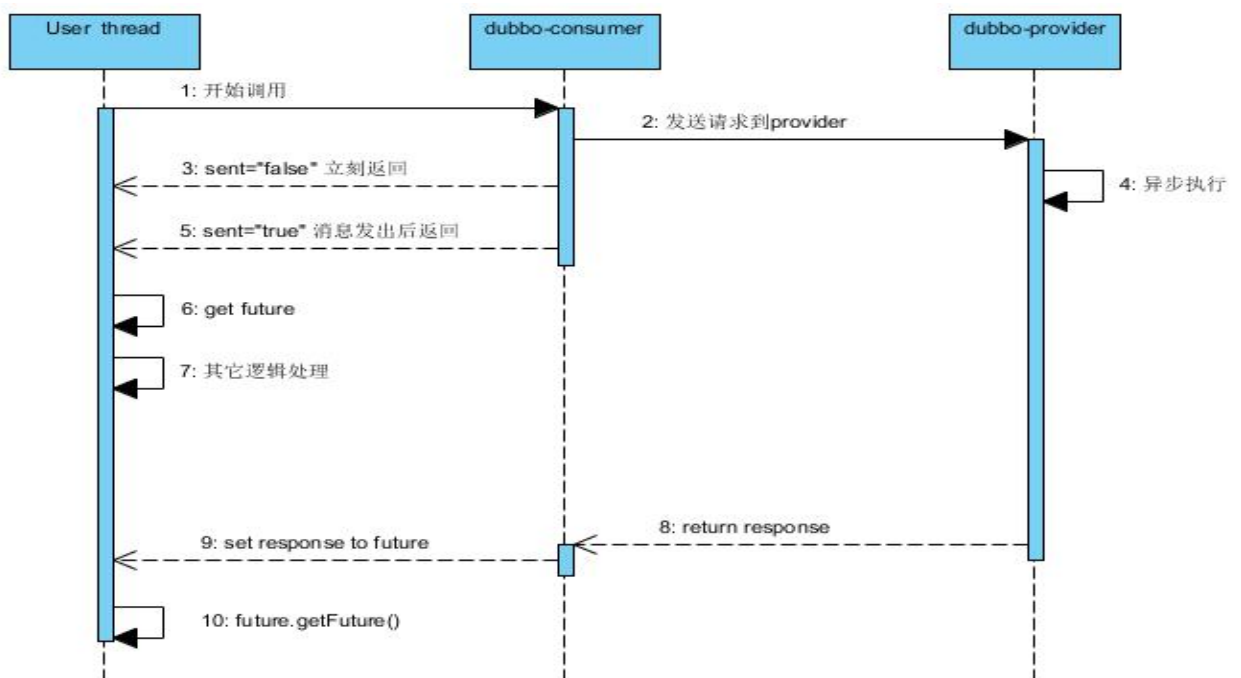
访问 <http://127.0.0.1:9091/hello/echo.do>

```
@RequestMapping("echo.do")
public String echo() {
    EchoService echoService = (EchoService) helloService;
    String status = (String) echoService.$echo("hello");
    assert(status.equals("hello"));
    return status;
}
```

4.17 Consumer 异步调用



时序图:



Dubbo-consumer.xml 配置:


```
<!--异步调用-->
<dubbo:reference id="asyncService" timeout="10000" interface="com.dubbo.demo.provider.api.AsyncService">
    <dubbo:method name="sayHello" async="true"/>
</dubbo:reference>
```

Consumer 端代码

```
//调用
LOGGER.info("异步调用开始...");
CompletableFuture<String> future = asyncService.sayHello(name: "async call request");
//回调
future.whenComplete((v, t) -> {
    if (t != null) {
        t.printStackTrace();
    } else {
        LOGGER.info("异步执行结果: {}", v);
    }
});
LOGGER.info("可以处理器其他逻辑");
// 早于结果输出
LOGGER.info("先与返回结果执行.");
return future.get();
```

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.

4.18 Provider 异步执行

Provider 端代码:

```
@Override
public CompletableFuture<String> sayHello(String name) {
    LOGGER.info("AsyncServiceImpl.sayHello");
    RpcContext rpcContext = RpcContext.getContext();
    // 建议为supplyAsync提供自定义线程池, 避免使用JDK公用线程池
    return CompletableFuture.supplyAsync(() -> {
        System.out.println(rpcContext.getAttachment(key: "consumer-key1"));
        try {
            Thread.sleep(millis: 5000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return "async response from provider." + name;
    });
}
```

Provider 端的配置文件:

<!--异步实现-->

```
<bean id="asyncService" class="com.dubbo.demo.provider.service.impl.AsyncServiceImpl"/>
<dubbo:service interface="com.dubbo.demo.provider.api.AsyncService" ref="asyncService"/>
```

4.19 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` 进行延迟暴露，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。

```
<dubbo:service interface="com.dubbo.demo.provider.api.DelayService" ref="delayService" delay="20000"/>
```

4.20 动态配置

5 参考资料

Demo 源码地址:

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

<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>

