# 1、dubbo项目引入

dubbo 阿里开源的一个SOA服务治理框架，从目前来看把它称作是一个RPC远程调用框架更为贴切。单从RPC框架来说，功能较完善，支持多种传输和序列化方案。所以想必大家已经知道他的核心功能了：就是远程调用。

## 1.1 引入dubbo的三种方式

### 1.1 直接引入dubbo

实现步骤

* 创建服务端项目
  + 引入dubbo 依赖
  + 编写服务端代码
* 创建客户端项目
  + 引入dubbo 依赖
  + 编写客户端调用代码

dubbo 引入：

<dependency>

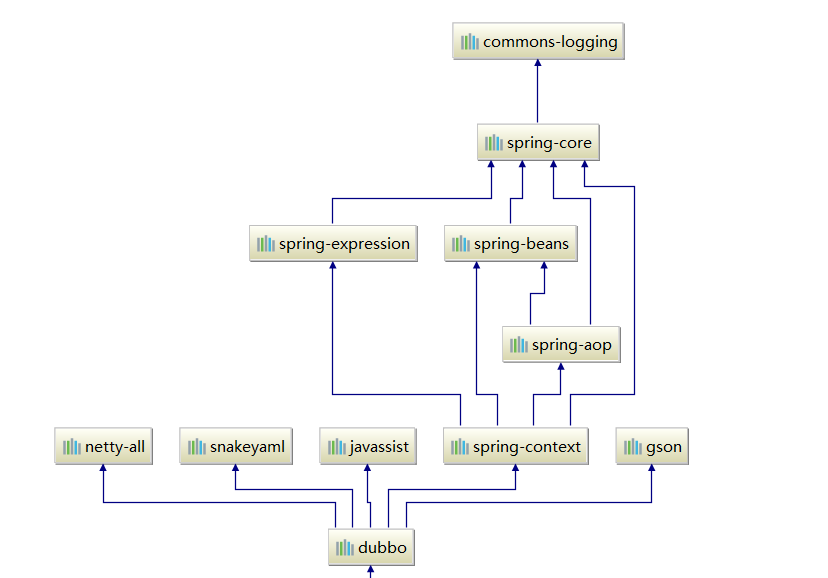
<groupId>org.apache.dubbo</groupId>

<artifactId>dubbo</artifactId>

<version>2.7.4.1</version>

</dependency>

dubbo 默认必填依懒：



服务端代码：

public void openServer(int port) {

ApplicationConfig config = new ApplicationConfig();

config.setName("simple-app");

ProtocolConfig protocolConfig=new ProtocolConfig();

protocolConfig.setName("dubbo");

protocolConfig.setPort(port);

protocolConfig.setThreads(20);

ServiceConfig<UserService> serviceConfig=new ServiceConfig();

serviceConfig.setApplication(config);

serviceConfig.setProtocol(protocolConfig);

serviceConfig.setRegistry(new RegistryConfig(RegistryConfig.NO\_AVAILABLE));

serviceConfig.setInterface(UserService.class);

serviceConfig.setRef(new UserServiceImpl());

serviceConfig.export();

}

客户端代码：

static String remoteUrl = "dubbo://127.0.0.1:12345/tuling.dubbo.server.UserService";

// 构建远程服务对象

public UserService buildRemoteService(String remoteUrl) {

ApplicationConfig application = new ApplicationConfig();

application.setName("young-app");

ReferenceConfig<UserService> referenceConfig = new ReferenceConfig<>();

referenceConfig.setApplication(application);

referenceConfig.setInterface(UserService.class);

referenceConfig.setUrl(remoteUrl);

UserService userService = referenceConfig.get();

return userService;

}

演示基于注册中心实现服集群：

* 修改服务端代码，添加multicast 注册中心。
* 修改客户端代码，添加multicast 注册中心。
* 观察 多个服务时，客户端如何调用。
* 观察 动态增减服务，客户端的调用。

# 服务端连接注册中心

serviceConfig.setRegistry(new RegistryConfig("multicast://224.1.1.1:2222"));

# 客户端连接注册中心

referenceConfig.setRegistry(new RegistryConfig("multicast://224.1.1.1:2222"));

#查看 基于UDP 占用的2222 端口

netstat -ano|findstr 2222

### 1.2 基于spring IOC 引入dubbo

在前面两个例子中 出现了,ApplicationConfig、ReferenceConfig、RegistryConfig、com.alibaba.dubbo.config.ServiceConfig等实例 ，很显然不需要每次调用的时候都去创建该实例那就需要一个IOC 容器去管理这些实例，spring 是一个很好的选择。

**提供者配置----------------------------------**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:dubbo="http://dubbo.apache.org/schema/dubbo"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd

http://dubbo.apache.org/schema/dubbo http://dubbo.apache.org/schema/dubbo/dubbo.xsd">

<!-- 提供方应用信息，用于计算依赖关系 -->

<dubbo:application name="simple-app" />

<!-- 使用multicast广播注册中心暴露服务地址 -->

<dubbo:registry address="multicast://224.5.6.7:1234" />

<!-- 用dubbo协议在20880端口暴露服务 ，当port设置为-1时代表自动分配端口号，此为生产时配置-->

<dubbo:protocol name="dubbo" port="20880" />

<!-- 声明需要暴露的服务接口 -->

<dubbo:service interface="tuling.dubbo.server.UserService" ref="userService" />

<!-- 和本地bean一样实现服务 -->

<bean id="userService" class="tuling.dubbo.server.impl.UserServiceImpl" />

</beans>

提供者服务暴露代码：

ApplicationContext context = new ClassPathXmlApplicationContext("/spring-provide.xml");

((ClassPathXmlApplicationContext) context).start();

System.in.read();

**消费者配置---------------------------------------**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:dubbo="http://dubbo.apache.org/schema/dubbo"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd

http://dubbo.apache.org/schema/dubbo http://dubbo.apache.org/schema/dubbo/dubbo.xsd">

<dubbo:application name="young-app"/>

<dubbo:registry address="multicast://224.5.6.7:1234"/>

<dubbo:reference id="userService" interface="tuling.dubbo.server.UserService"/>

</beans>

消费者调用代码：

ApplicationContext context = new ClassPathXmlApplicationContext("/spring-consumer.xml");

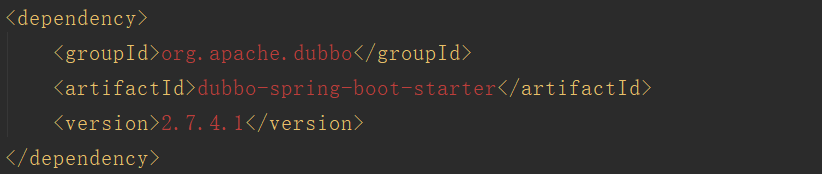
UserService userService = context.getBean(UserService.class);

UserVo u = userService.getUser(1111);

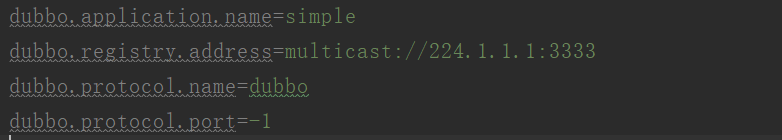
System.out.println(u);

### 1.3 spring boot方式引入dubbo

引入依赖



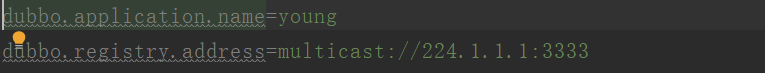
Server端配置



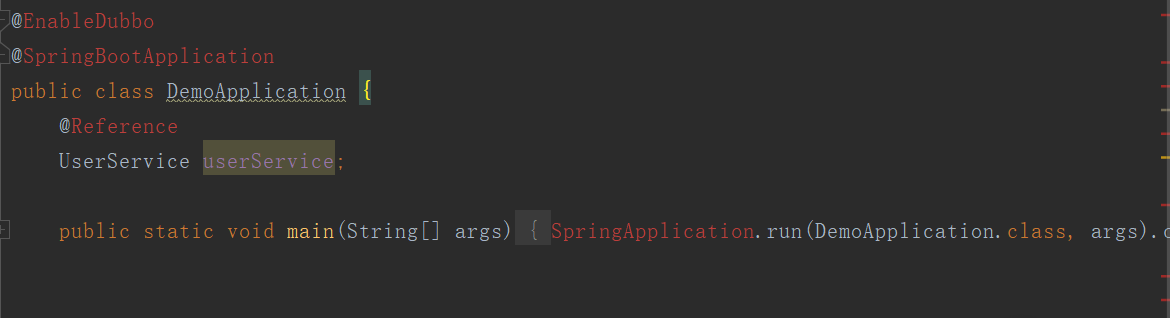
使用@EnableDubbo注解引入dubbo配置



客户端配置

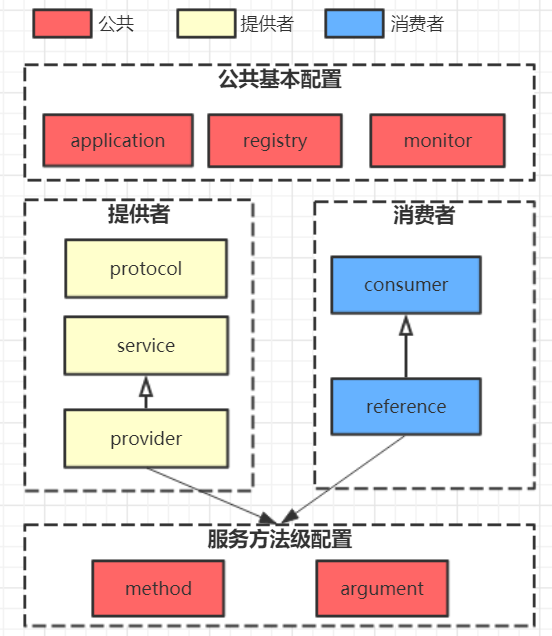


使用@Reference 注解注入RPC调用的类实例



## 1.2 dubbo的配置属性

|  |  |  |
| --- | --- | --- |
| **标签** | **用途** | **解释** |
| <dubbo:application/> | 公共 | 用于配置当前应用信息，不管该应用是提供者还是消费者 |
| <dubbo:registry/> | 公共 | 用于配置连接注册中心相关信息 |
| <dubbo:protocol/> | 服务 | 用于配置提供服务的协议信息，协议由提供方指定，消费方被动接受 |
| <dubbo:service/> | 服务 | 用于暴露一个服务，定义服务的元信息，一个服务可以用多个协议暴露，一个服务也可以注册到多个注册中心 |
| <dubbo:provider/> | 服务 | 当 ProtocolConfig 和 ServiceConfig 某属性没有配置时，采用此缺省值，可选 |
| <dubbo:consumer/> | 引用 | 当 ReferenceConfig 某属性没有配置时，采用此缺省值，可选 |
| <dubbo:reference/> | 引用 | 用于创建一个远程服务代理，一个引用可以指向多个注册中心 |
| <dubbo:method/> | 公共 | 用于 ServiceConfig 和 ReferenceConfig 指定方法级的配置信息 |
| <dubbo:argument/> | 公共 | 用于指定方法参数配置 |



### Application：用于配置应用的信息

Name：必填，当前应用名称，用于注册中心计算应用间依赖关系，注意：消费者和提供者应用名不要一样，此参数不是匹配条件，你当前项目叫什么名字就填什么，和提供者消费者角色无关，比如：kylin应用调用了morgan应用的服务，则kylin项目配成kylin，morgan项目配成morgan，可能kylin也提供其它服务给别人使用，但kylin项目永远配成kylin，这样注册中心将显示kylin依赖于morgan

Version：当前应用的版本

Owner：应用负责人，用于服务治理，请填写负责人公司邮箱前缀

Organization：组织名称(BU或部门)，用于注册中心区分服务来源，此配置项建议不要使用autoconfig，直接写死在配置中，如china,intl,itu,crm,asc,dw,aliexpress

Architecture：用于服务分层对应的架构。如，intl、china。不同的架构使用不同的分层。

Environment：应用环境，如：develop/test/product，不同环境使用不同的缺省值，以及作为只用于开发测试功能的限制条件

Complier：默认javassist，Java字节码编译器，用于动态类的生成，可选：jdk或javassist

Logger：slf4j默认值，日志输出方式，可选：slf4j,jcl,log4j,log4j2,jdk

### Registry：注册中心配置

id: string 注册中心引用BeanId，可以在<dubbo:service registry=””>或<dubbo:reference registry=””>中引用此Id

address：<host:port> 服务发现必填，注册中心服务器地址，如果地址没有端口缺省为9090，同一集群内的多个地址用逗号分隔，如：ip:port,ip:port，不同集群的注册中心，请配置多个<dubbo:registry>标签

protocol:默认dubbo协议，注册中心地址协议，支持dubbo, multicast, zookeeper, redis, consul(2.7.1), sofa(2.7.2), etcd(2.7.2), nacos(2.7.2)等协议

port：9090，注册中心缺省端口，当address没有带端口时使用此端口做为缺省值

username：登陆注册中心的用户名，如果注册中心不需要验证，则可以不填

password：注册中心密码，不需要可以不填

transport：默认netty，网络传输方式，可选mina，netty

timeout：默认5000毫秒，注册中心请求超时时间（毫秒）

session：60000，注册中心会话超时时间（毫秒），用于检测提供者非正常断线后的脏数据，比如用心跳检测的实现，此时间就是心跳间隔，不同注册中心实现不一样

file： 使用文件缓存注册中心地址列表及服务提供者列表，应用重启时将基于此文件恢复，注意：两个注册中心不能使用同一文件存储

wait：默认0，停止时等待通知完成的时间（毫秒）

check：布尔值，默认true，注册中心不存在时或者宕机是否报错

register：布尔值默认true，是否向此注册中心注册服务，如果设为false，将只订阅，不注册，即只调用注册中心的服务，而本身不注册上去不会被调用

subscribe：布尔值，默认为true，是否向此注册中心订阅服务，如果设为false，将只注册，不订阅

dynamic：布尔值，默认为true，服务是否动态注册，如果设为false，注册后将显示为disable状态，需人工启用，并且服务提供者停止时，也不会自动取消注册，需人工禁用。

Group：默认为dubbo分组，服务注册分组，跨组的服务不会相互影响，也无法相互调用，适用于环境隔离。

Simplified：布尔值，默认为false，注册到注册中心的URL是否采用精简模式的（与低版本兼容）

Extra-keys：在simplified=true时，extraKeys允许你在默认参数外将额外的key放到URL中，格式："interface,key1,key2"。

Provider：服务提供者，可以为service、protocol提供默认配置，类似于模板

Id：默认值dubbo，协议BeanId，可以在<dubbo:service proivder="">中引用此ID，如果service中没有配置则使用第一个

Protocol：默认dubbo，协议

Host：服务主机名，多网卡选择或指定VIP及域名时使用，为空则自动查找本机IP，建议不要配置，让Dubbo自动获取本机IP

Threads：服务线程池大小（固定大小），默认200

Payload：默认8M，请求及响应数据包大小限制，单位：字节

Path：提供者上下文路径，为服务path的前缀

Server：dubbo协议缺省为netty，http协议缺省为servlet，协议的服务器端实现类型，比如：dubbo协议的mina,netty等，http协议的jetty,servlet等

Client：dubbo协议的默认值为netty，协议的客户端实现类型，比如：dubbo协议的mina,netty等

Codec：协议编码方式，默认dubbo

Serialization：dubbo协议缺省为hessian2，rmi协议缺省为java，http协议缺省为json，协议序列化方式，当协议支持多种序列化方式时使用，比如：dubbo协议的dubbo,hessian2,java,compactedjava，以及http协议的json,xml等

Default：默认false，是否为缺省协议，用于多协议

Filter：服务提供方远程调用过程拦截器名称，多个名称用逗号分隔

Listener：服务提供方导出服务监听器名称，多个名称用逗号分隔

Threadpool：线程池，默认固定线程池线程池类型，可选：fixed/cached/limit(2.5.3以上)/eager(2.6.x以上)

Accepts：默认0（0应该是不限制），服务提供者最大可接受连接数

Version：默认值0.0.0，服务版本，建议使用两位数字版本，如：1.0，通常在接口不兼容时版本号才需要升级

Group：服务分组，当一个接口有多个实现，可以用分组区分

Delay：默认值0，延迟注册服务时间(毫秒)- ，设为-1时，表示延迟到Spring容器初始化完成时暴露服务