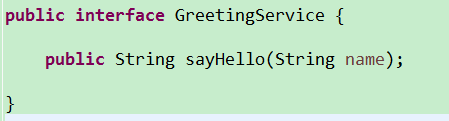
Dubbo服务配置文档

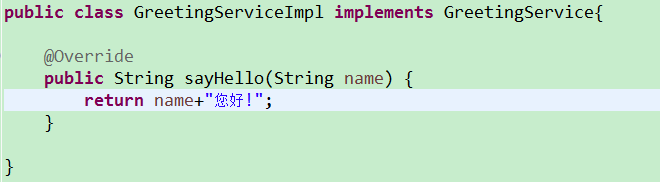
# 服务提供着

## 快速启动

### 定义服务接口



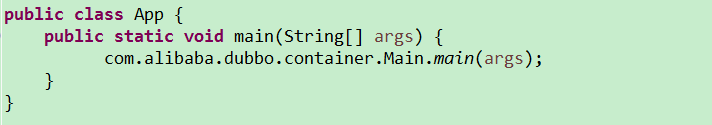
### 在服务方提供好实现



### 在spring配置声明暴露服务

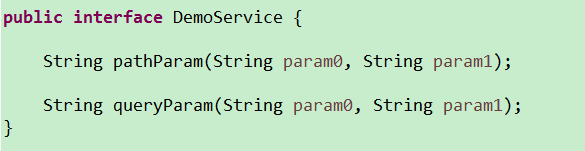


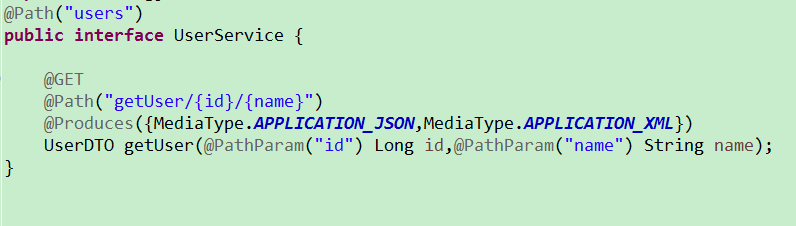
### 通过dubbo内置main方法启动dubbo服务



## rest协议服务

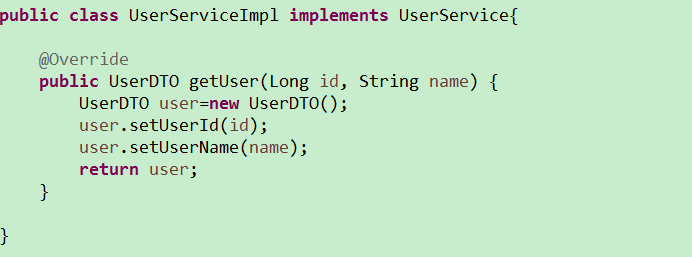
### 定义服务接口





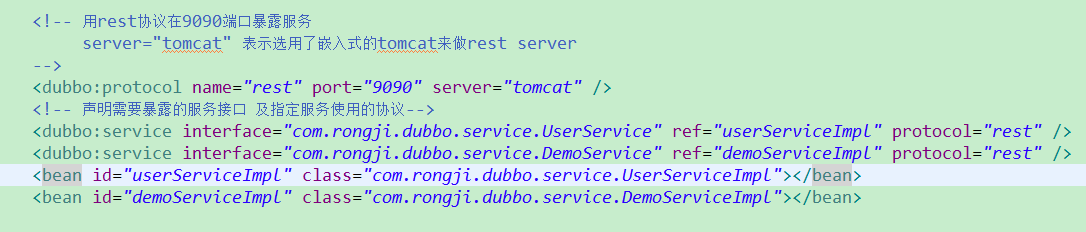
### 服务提供方提供好实现





### spring配置文件中添加此服务

目前配置的rest server使用的是内置tomcat



注意事项:

1.XML数据格式的额外要求

由于JAX-RS的实现一般都用标准的JAXB（Java API for XML Binding）来序列化和反序列化XML格式数据，所以我们需要为每一个要用XML传输的对象添加一个类级别的JAXB annotation，否则序列化将报错。例如为getUser()中返回的UserDTO添加如下：

@XmlRootElement

public class UserDTO implements Serializable {

// ...

}

2. 上述REST案例说明了Annotation可以放在接口或实现类上，但dubbo直接开发的消费端来访问此服务，则annotation必须放到接口上

## 常用配置示例

### 服务降级

#### mock配置

dubbo官方文档上使用一个mock配置，实现服务降级。mock只在出现非业务异常(比如超时，网络异常等)时执行。mock的配置支持两种，一种为boolean值，默认的为false。如果配置为true，则缺省使用mock类名，即类名+Mock后缀,；另外一种则是配置”return null”，可以很简单的忽略掉异常。

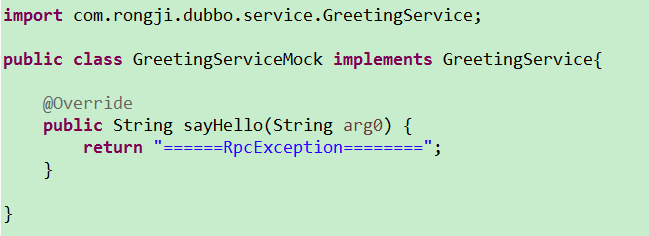
1.方式一



2.方式二:



配置mock=”true”的情况，对于上面的例子即在GreetingService的同个路径下，添加类FooServiceMock，实现如下：



### 集群

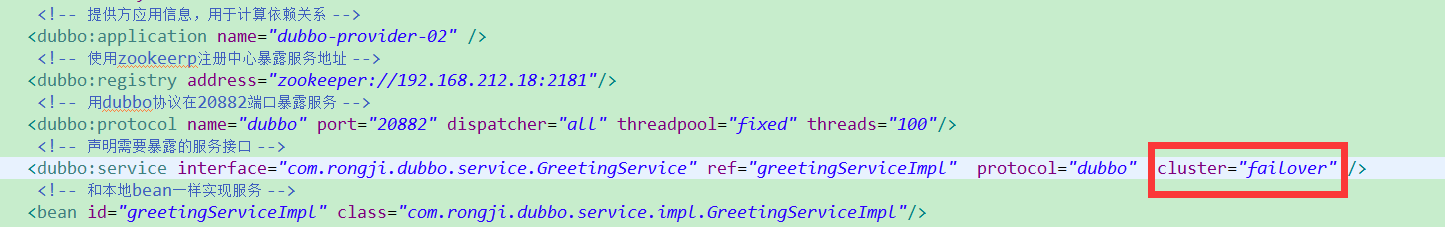
#### 多个服务提供者暴露相同的接口实现一样。

#### 在spring配置声明暴露的服务

比如dubbo-provider-01:



dubbo-provider-02:



集群容错方式

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

Failover Cluster

失败自动切换，当出现失败，重试其它服务器 [[1]](http://dubbo.apache.org/zh-cn/docs/user/demos/fault-tolerent-strategy.html#fn1)。通常用于读操作，但重试会带来更长延迟。可通过 retries="2" 来设置重试次数(不含第一次)。

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，比如新增记录。

Failsafe Cluster

失败安全，出现异常时，直接忽略。通常用于写入审计日志等操作。

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 [[2]](http://dubbo.apache.org/zh-cn/docs/user/demos/fault-tolerent-strategy.html#fn2)。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

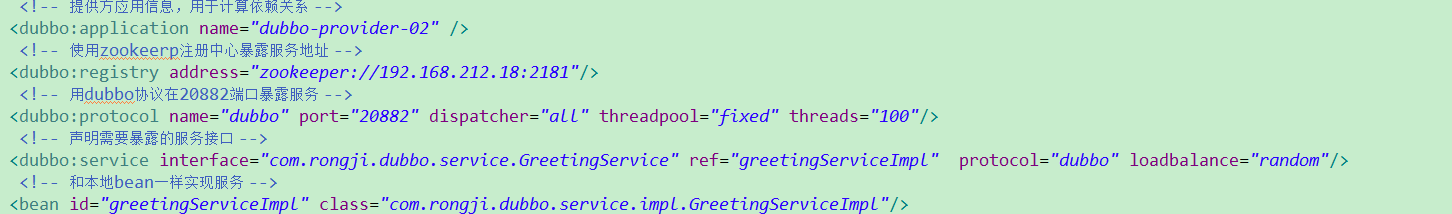
#### 多个服务提供者暴露相同的接口实现一样。

#### 在spring配置声明暴露的服务

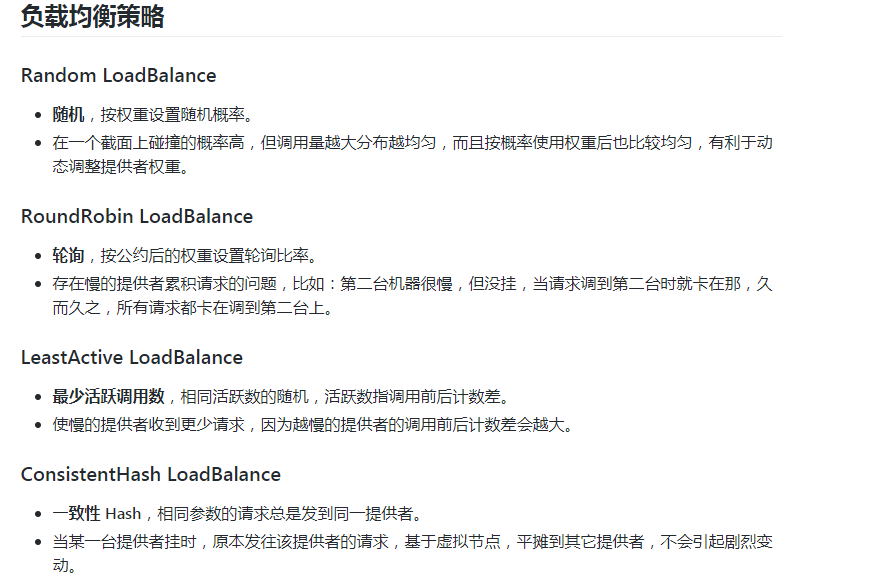
比如dubbo-provider-01:



dubbo-provider-02:



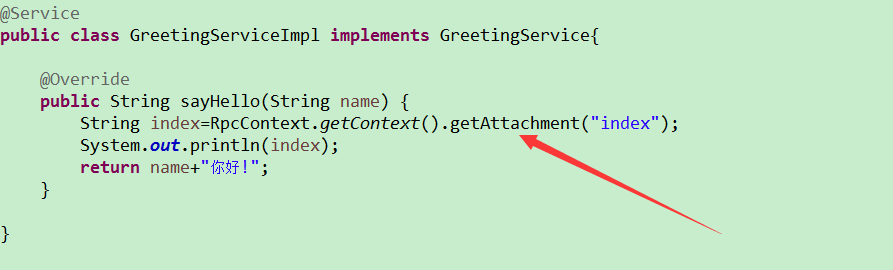
负载均衡策略有:



### 隐式传参

可以通过 RpcContext 上的 setAttachment 和 getAttachment 在服务消费方和提供方之间进行参数的隐式传递。

#### 在服务提供方端获取隐式参数

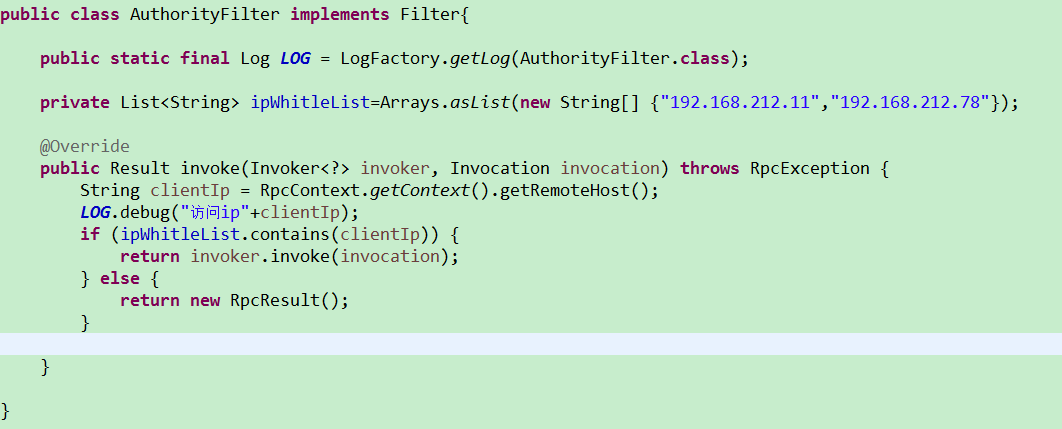


### 过滤器

比如白名单的过滤

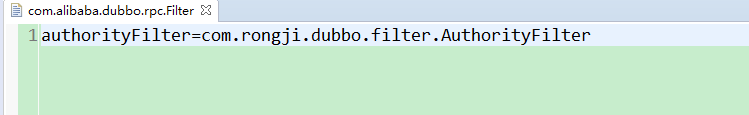
#### 扩展Filter,实现com.alibaba.dubbo.rpc.Filter接口

给dubbo接口添加白名单过滤器



#### 增加SPI配置文件

在resources目录下添加纯文本文件META-INF/dubbo/com.alibaba.dubbo.rpc.Filter，内容如下: xxxFilter=com.xxx.AuthorityFilter



#### 在服务提供方暴露的接口增加对应的过滤器



### 令牌验证

#### 服务提供方token spring 配置

可以全局设置开启令牌验证：

*<!--随机token令牌，使用UUID生成-->*

<dubbo:provider interface="com.rongji.dubbo.service.GreetingService" token="true" />

*<!--固定token令牌，相当于密码-->*

<dubbo:provider interface="com.rongji.dubbo.service.GreetingService" token="123456" />

也可在服务级别设置：

*<!--随机token令牌，使用UUID生成-->*

<dubbo:service interface="com.rongji.dubbo.service.GreetingService" token="true" />

*<!--固定token令牌，相当于密码-->*

<dubbo:service interface="com.rongji.dubbo.service.GreetingService" token="123456" />

还可在协议级别设置：

*<!--随机token令牌，使用UUID生成-->*

<dubbo:protocol name="*dubbo*" token="true" />

*<!--固定token令牌，相当于密码-->*

<dubbo:protocol name="dubbo" token="123456" />

### 多版本

老版本服务提供者配置：



新版本服务提供者配置：



# 服务消费者

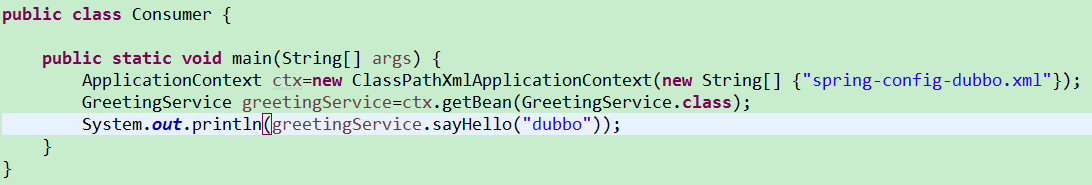
## 快速启动

### 打包服务提供者暴露的接口jar给服务消费者

### 通过spring配置引用远程服务



### 加载spring配置并调用远程服务

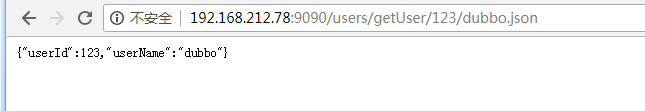


## rest协议服务

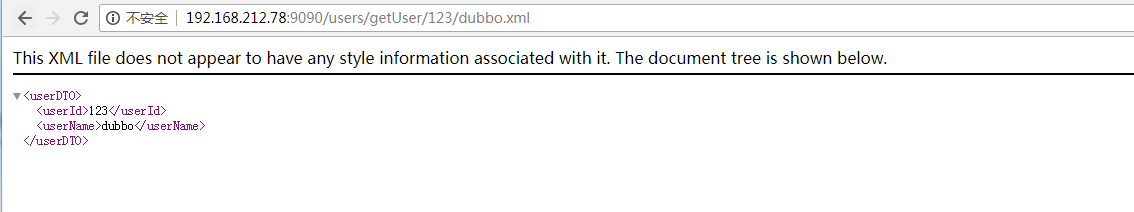
### 客户端访问



返回json数据



返回xml数据

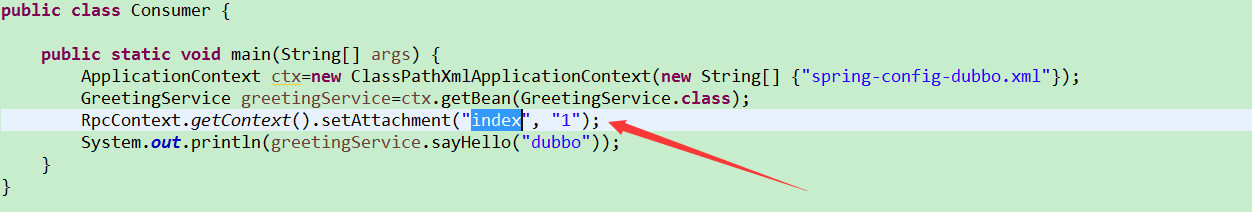


## 常用配置示例

### 隐式传参

可以通过 RpcContext 上的 setAttachment 和 getAttachment 在服务消费方和提供方之间进行参数的隐式传递。

#### 在服务消费方端设置隐式参数



### 本地存根

#### Stub服务实现类

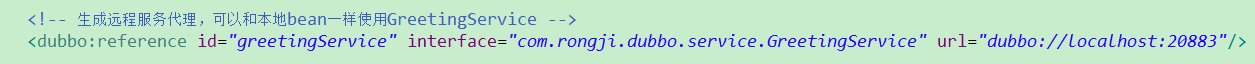


#### 在服务消费者spring配置暴露的接口增加Stub实现类



### 直连提供者

不需要提供注册中心



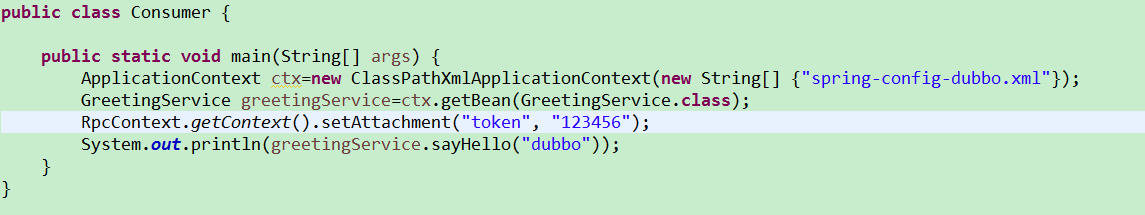
### 令牌验证

#### 直连服务提供者的消费端调用

如果服务提供者token=”true”,则消费端应该没有办法调用(推测),如果服务提供者token=”123456”,

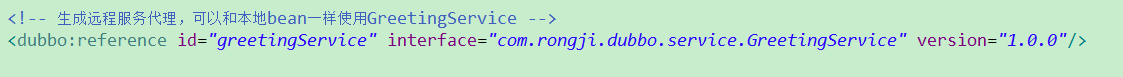
则服务消费者需在远程调用接口前加上

RpcContext.getContext().setAttachment("token", "123456");

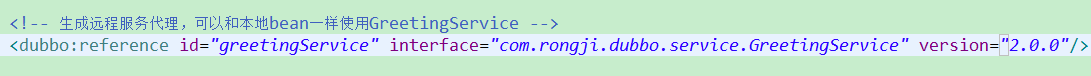


### 多版本

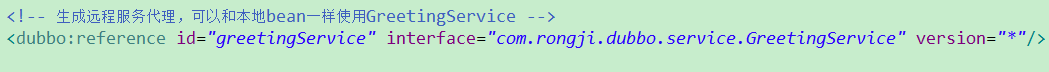
老版本服务消费者配置：



新版本服务消费者配置：



如果不需要区分版本，可以按照以下的方式配置



# demo

## dubbo-demo工程

### 各个子模块说明

1.dubbo-provider-01为服务提供者,启动方式为main

2.dubbo-provider-02为服务提供者,启动方式为(包含了main启动和外置tomcat容器启动),该模快使用了dubbo协议和rest协议举例

main启动:通过com.alibaba.dubbo.container.Main.main(args)进行main启动的默认加载的spring配置文件是resources/META-INF/spring/下的，rest服务的发布使用了内置tomcat容器。

通过tomcat容器启动:rest服务的发布使用了外置tomcat容器。

3.dubbo-consumer为消费者

4.dubbo-api为服务暴露接口

### 导出jar启动

1.dubbo-provider-01模块依赖了dubbo-api,所以先执行dubbo-api模块的maven install;

2.执行dubbo-provider-01模块的maven install

3.拷贝dubbo-provider-01/target下的zip

# 运维手册

## 使用pinpoint做分布式跟踪

### 使用Pinpoint-agent启动需要监控的项目

pinpoint-agent部署:

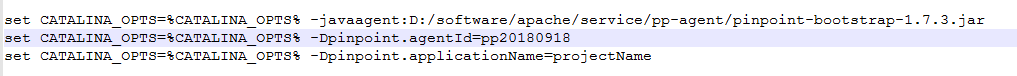
1.新建目录：mkdir pp-agent(我这边路径是D:\software\apache\service\pp-agent)

2.将[pinpoint-agent-1.7.3.tar.gz](https://github.com/naver/pinpoint/releases/download/1.7.3/pinpoint-agent-1.7.3.tar.gz) 拷贝到pp-agent目录并解压

3.配置pinpoint.config：profiler.collector.ip=127.0.0.1这是指pinpoint-collector的地址，如果是同一服务器，则不用修改。其它默认。

安装pinpoint-collector启动后，自动就开启了9994，9995，9996的端口了，这里默认即可。如果有端口需求，要去pinpoint-collector的配置文件 中，修改这些端口。

1.使用tomcat容器启动dubbo服务,以下配置放在catalina.bat下



2.使用java命令启动则为:

java-jar-javaagent:D:/software/apache/service/pp-agent/pinpoint-bootstrap-1.7.jar -Dpinpoint.agentId=pp20180918 -Dpinpoint.applicationName=projectName