百知教育
Baizhi Education

Dubbo实战

讲师:蒋中洲

概述

- Dubbo是阿里巴巴SOA服务化治理方案的核心框架，每天为2,000+个服务提供3,000,000,000+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,自开源后，已有不少非阿里系公司在使用Dubbo.

Dubbo是什么

Dubbo['dʌbəʊ] 是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SOA服务治理方案。其核心部分包含:

- 远程通讯: 提供对多种基于长连接的NIO框架抽象封装，包括多种线程模型，序列化，以及“请求-响应”模式的信息交换方式。
- 集群容错: 提供基于接口方法的透明远程过程调用，包括多协议支持，以及软负载均衡，失败容错，地址路由，动态配置等集群支持。
- 自动发现: 基于注册中心目录服务，使服务消费方能动态的查找服务提供方，使地址透明，使服务提供方可以平滑增加或减少机器。

Dubbo能做什么

- 透明化的远程方法调用，就像调用本地方法一样调用远程方法，只需简单配置，没有任何API侵入。
- 软负载均衡及容错机制，可在内网替代F5等硬件负载均衡器，降低成本，减少单点。
- 服务自动注册与发现，不再需要写死服务提供方地址，注册中心基于接口名查询服务提供者的IP地址，并且能够平滑添加或删除服务提供者。

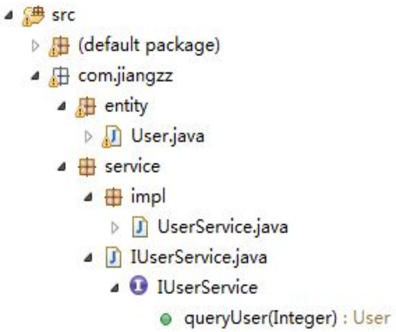
Dubbo入门案例

- 导入jar包
- 导入配置applicationContext.xml文件
- 编码IUserService接口类

服务

```
lib
    commons-logging-1.1.1.jar
    dubbo-2.5.3.jar
    javassist-3.15.0-GA.jar
    jline-0.9.94.jar
    log4j-1.2.17.jar
    netty-3.2.5.Final.jar
    slf4j-api-1.7.5.jar
    slf4j-log4j12-1.7.5.jar
    spring-aop-3.2.9.RELEASE.jar
    spring-beans-3.2.9.RELEASE.jar
    spring-context-3.2.9.RELEASE.jar
    spring-core-3.2.9.RELEASE.jar
    spring-expression-3.2.9.RELEASE.j
    zkclient-0.2.jar
    zookeeper-3.3.1.jar
```

```xml
<?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://code.alibabatech.com/schema/dubbo"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://code.alibabatech.com/schema/dubbo
    http://code.alibabatech.com/schema/dubbo/dubbo.xsd">
    <!-- 配置实体bean -->
    <bean id="userService" class="com.jiangzz.service.impl.UserService"/>
    <!-- 提供方应用信息，用于计算依赖关系 -->
    <dubbo:application name="DubboDemo01"/>
    <!-- 提供注册这，用于注册服务 -->
    <dubbo:registry address="localhost:2181" protocol="zookeeper"/>
    <!-- 用dubbo协议在20880端口暴露服务 -->
    <dubbo:protocol name="dubbo" port="20880" />
    <!-- 声明需要暴露的服务接口 -->
    <dubbo:service interface="com.jiangzz.service.IUserService" ref="userS
</beans>
```

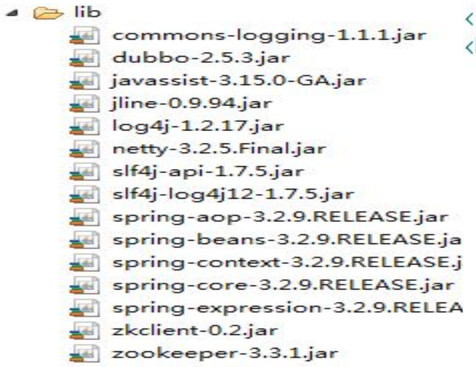


```
src
    (default package)
    com.jiangzz
        entity
            User.java
        service
            impl
                UserService.java
            IUserService.java
                IUserService
                    queryUser(Integer) : User
```

- 启动spring容器等价于发布服务

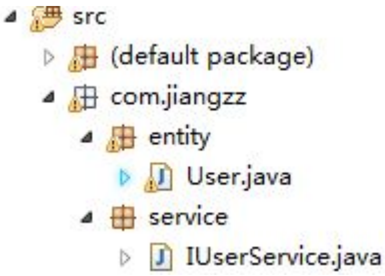
Dubbo入门案例

- 导入jar包
- 导入配置applicationContext.xml文件
- 导入IUserService接口类



```
lib
    commons-logging-1.1.1.jar
    dubbo-2.5.3.jar
    javassist-3.15.0-GA.jar
    jline-0.9.94.jar
    log4j-1.2.17.jar
    netty-3.2.5.Final.jar
    slf4j-api-1.7.5.jar
    slf4j-log4j12-1.7.5.jar
    spring-aop-3.2.9.RELEASE.jar
    spring-beans-3.2.9.RELEASE.ja
    spring-context-3.2.9.RELEASE.j
    spring-core-3.2.9.RELEASE.jar
    spring-expression-3.2.9.RELEA
    zkclient-0.2.jar
    zookeeper-3.3.1.jar
```

```xml
<?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://code.alibabatech.com/schema/dubbo"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://code.alibabatech.com/schema/dubbo
    http://code.alibabatech.com/schema/dubbo/dubbo.xsd">
     <!-- 消费方应用名，用于计算依赖关系，不是匹配条件，不要与提供方一样 -->
    <dubbo:application name="DubboDemo01_"  />
    <dubbo:registry address="localhost:2181" protocol="zookeeper" />
    <!-- 生成远程服务代理，可以和本地bean一样使用demoService -->
    <dubbo:reference id="userService" interface="com.jiangzz.service
</beans>
```

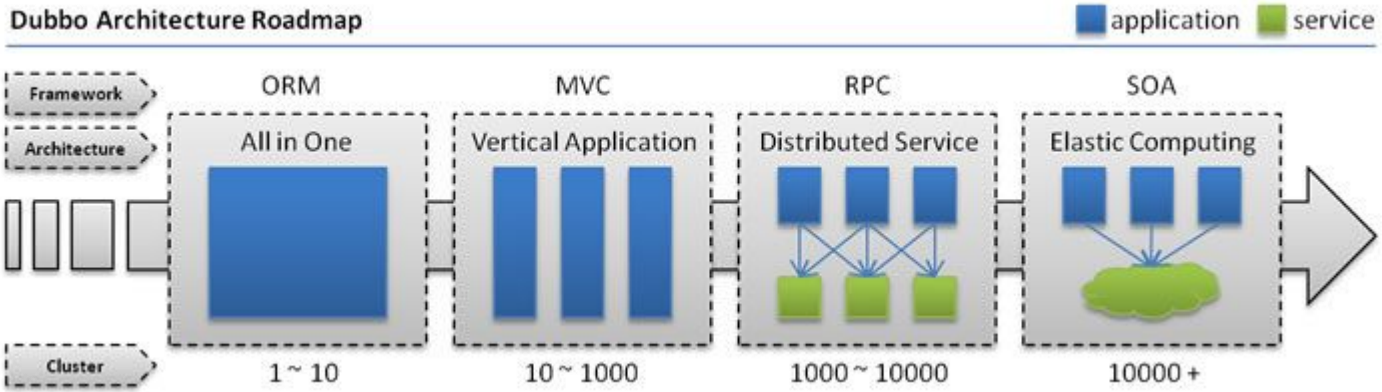


```
src
    (default package)
    com.jiangzz
        entity
            User.java
        service
            IUserService.java
```

- 编码测试案例

背景

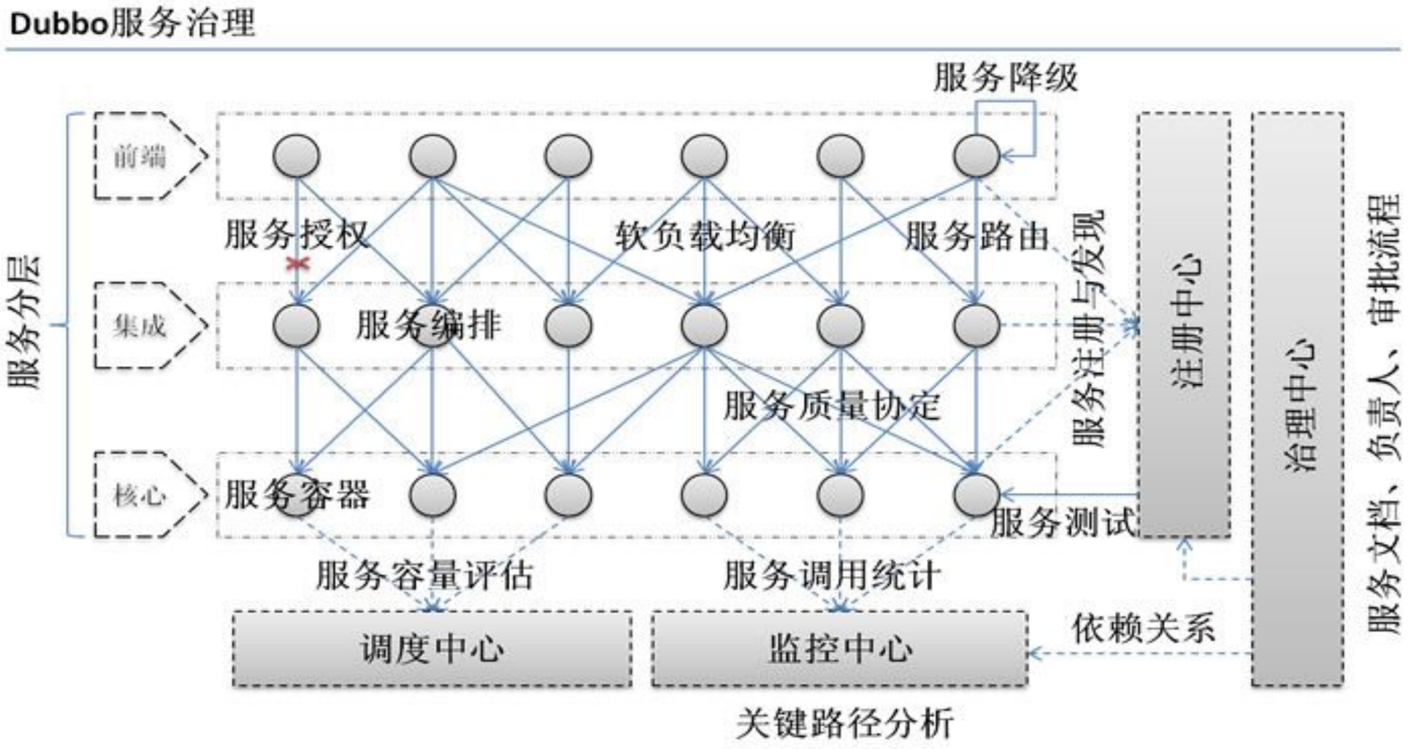
- 随着互联网的发展，网站应用的规模不断扩大，常规的垂直应用架构已无法应对，分布式服务架构以及流动计算架构势在必行，亟需一个治理系统确保架构有条不紊的演进。



背景

- 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。

需求


Dubbo服务治理
服务降级
服务分层
前端
集成
核心
服务授权
软负载均衡
服务路由
服务编排
服务质量协定
服务容器
服务注册与发现
注册中心
治理中心
服务文档、负责人、审批流程
服务测试
服务容量评估
服务调用统计
依赖关系
调度中心
监控中心
关键路径分析

需求

- 在大规模服务化之前，应用可能只是通过RMI或Hessian等工具，简单的暴露和引用远程服务，通过配置服务的URL地址进行调用，通过F5等硬件进行负载均衡。

(1) 当服务越来越多时，服务URL配置管理变得非常困难，F5硬件负载均衡器的单点压力也越来越大。此时需要一个服务注册中心，动态的注册和发现服务，使服务的位置透明。并通过在消费方获取服务提供方地址列表，实现软负载均衡和Failover，降低对F5硬件负载均衡器的依赖，也能减少部分成本。
(2) 当进一步发展，服务间依赖关系变得错踪复杂，甚至分不清哪个应用要在哪个应用之前启动，架构师都不能完整的描述应用的架构关系。
这时，需要自动画出应用间的依赖关系图，以帮助架构师理清理关系。
(3) 接着，服务的调用量越来越大，服务的容量问题就暴露出来，这个服务需要多少机器支撑？什么时候该加机器？为了解决这些问题，第一步，要将服务现在每天的调用量，响应时间，都统计出来，作为容量规划的参考指标。其次，要可以动态调整权重，在线上，将某台机器的权重一直加大，并在加大的过程中记录响应时间的变化，直到响应时间到达阀值，记录此时的访问量，再以此访问量乘以机器数反推总容量。

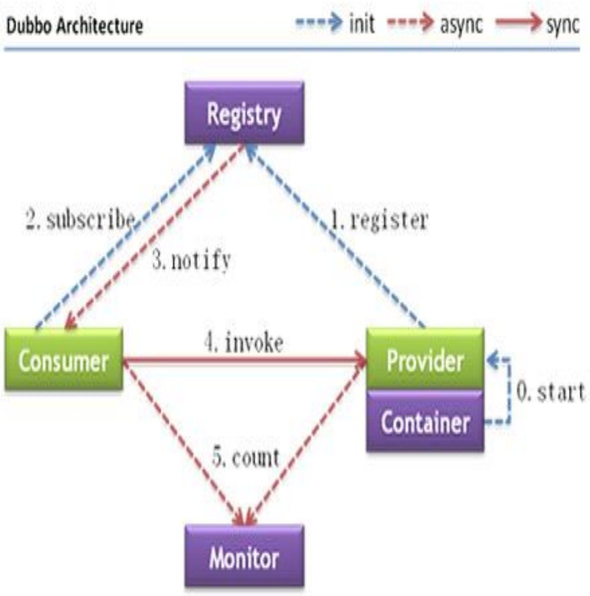
架构

❖ 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。

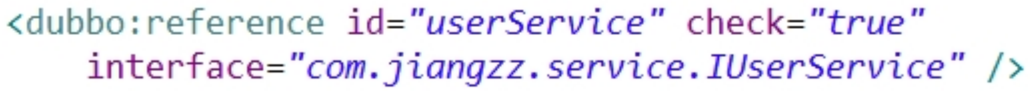
❖ 调用关系说明：

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。



启动检测

- 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，阻止Spring初始化完成，以便上线时，能及早发现问题，默认check=true。



```
<dubbo:reference id="userService" check="true"
    interface="com.jiangzz.service.IUserService" />
```

说明：一般在Spring做DI的时候是否去检测该服务类是否可用，如果设置为false时spring容器在启动的时候不会检测服务提供者是否可用。默认值是true。

警告：如果你的Spring容器是懒加载的，或者通过API编程延迟引用服务，请关闭check，否则服务临时不可用时，会抛出异常，拿到null引用，如果check=false，总是会返回引用，当服务恢复时，能自动连上。

启动检测

关闭某个服务的启动时检查：(没有提供者时报错)

```
<dubbo:reference interface="com.foo.BarService" check="false" />
```

关闭所有服务的启动时检查：(没有提供者时报错)

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

关闭注册中心启动时检查：(注册订阅失败时报错)

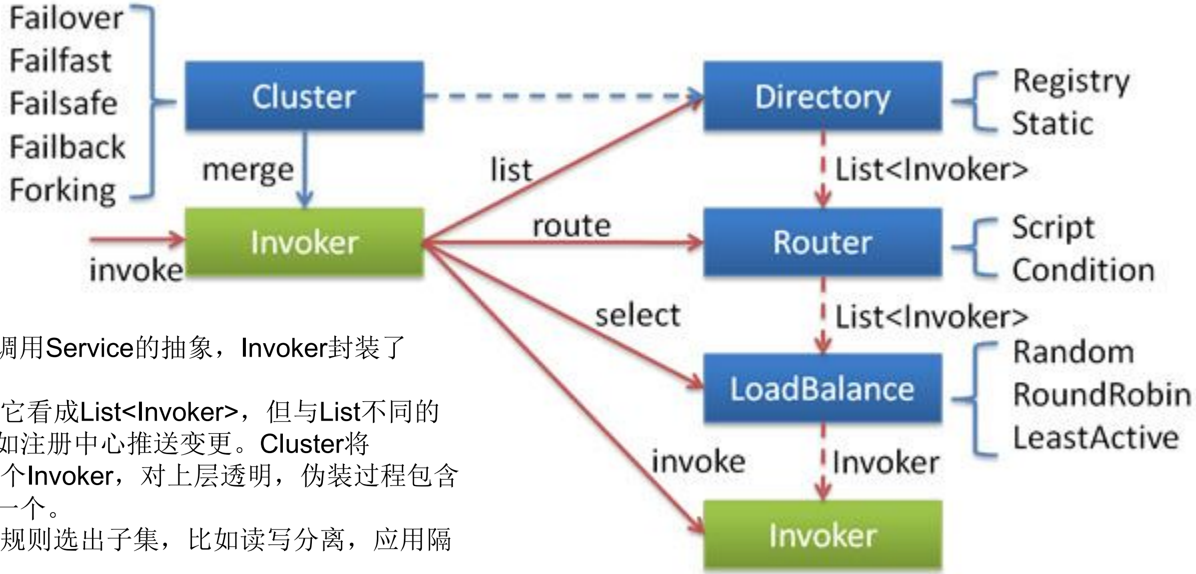
```
<dubbo:registry check="false" />
```

注意：<dubbo:reference/>默认是延迟初始化的，只用在外界使用到该bean的是时候才会去创建动态代理。可以添加init=true此时及时不引用该bean也会创建gaibean的代理对象。

```
<dubbo:reference interface="com.foo.BarService" init="true" />
```

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。



- 各节点关系：
  - 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。

集群容错

❖ Failover Cluster

- 失败自动切换，当出现失败，重试其它服务器。(缺省)
- 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)。

❖ 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。

❖ 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。

❖ 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。

❖ 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。

❖ Broadcast Cluster

- 广播调用所有提供者，逐个调用，任意一台报错则报错。(2.1.0开始支持)
- 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。

集群容错

重试次数配置如：(failover集群模式生效)

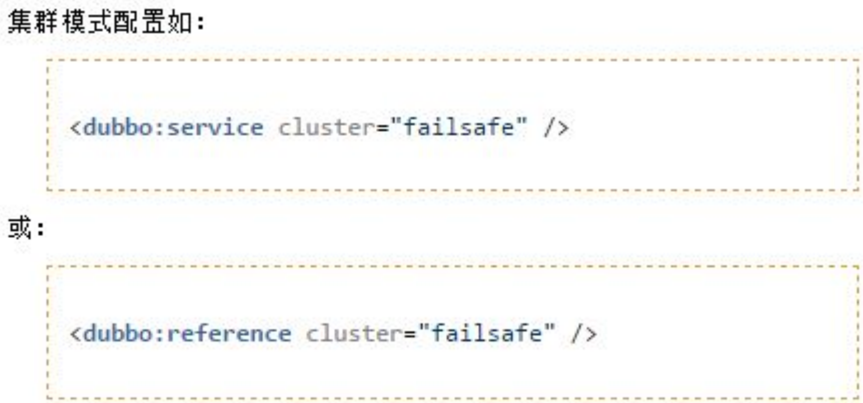
```
<dubbo:service retries="2" />
```

或：

```
<dubbo:reference retries="2" />
```

或：

```
<dubbo:reference>
    <dubbo:method name="findFoo" retries="2" /
</dubbo:reference>
```



集群模式配置如：

```
<dubbo:service cluster="failsafe" />
```

或：

```
<dubbo:reference cluster="failsafe" />
```

负载均衡

❖ 在集群负载均衡时，Dubbo提供了多种均衡策略，缺省为random随机调用。

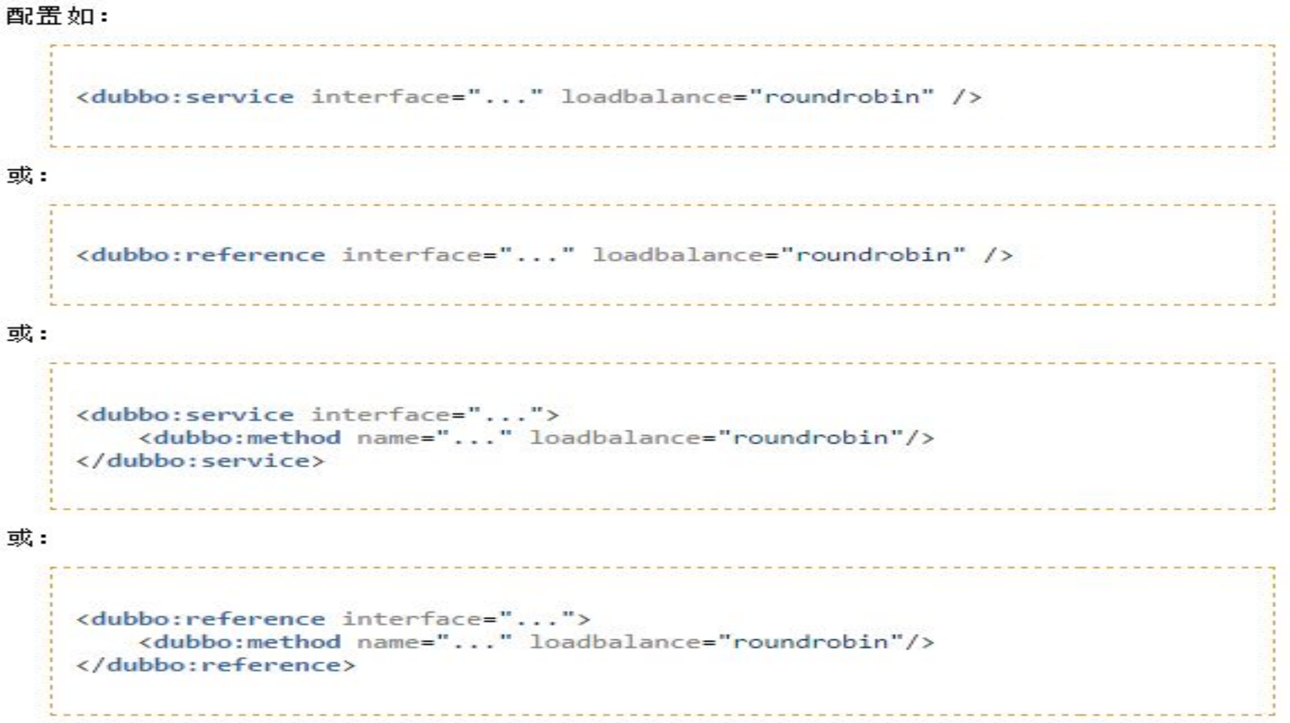
❖ Random LoadBalance
- 随机，按权重设置随机概率。
- 在一个截面上碰撞的概率高，但调用量越大分布越均匀，而且按概率使用权重后也比较均匀，有利于动态调整提供者权重。

❖ LeastActive LoadBalance
- 最少活跃调用数，相同活跃数的随机，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。

❖ RoundRobin LoadBalance
- 轮循，按公约后的权重设置轮循比率。
- 存在慢的提供者累积请求问题，比如：第二台机器很慢，但没挂，当请求调到第二台时就卡在那，久而久之，所有请求都卡在调到第二台上。

❖ ConsistentHash LoadBalance
- 一致性Hash，相同参数的请求总是发到同一提供者。
- 当某一台提供者挂时，原本发往该提供者的请求，基于虚拟节点，平摊到其它提供者，不会引起剧烈变动。
- 算法参http://en.wikipedia.org/wiki/Consistent_hashing。
- 缺省只对第一个参数Hash，如果要修改，请配置 <dubbo:parameter key="hash.arguments" value="0,1" />
- 缺省用160份虚拟节点，如果要修改，请配置 <dubbo:parameter key="hash.nodes" value="320" />

负载均衡



配置如：

```
<dubbo:service interface="..." loadbalance="roundrobin" />
```

或：

```
<dubbo:reference interface="..." loadbalance="roundrobin" />
```

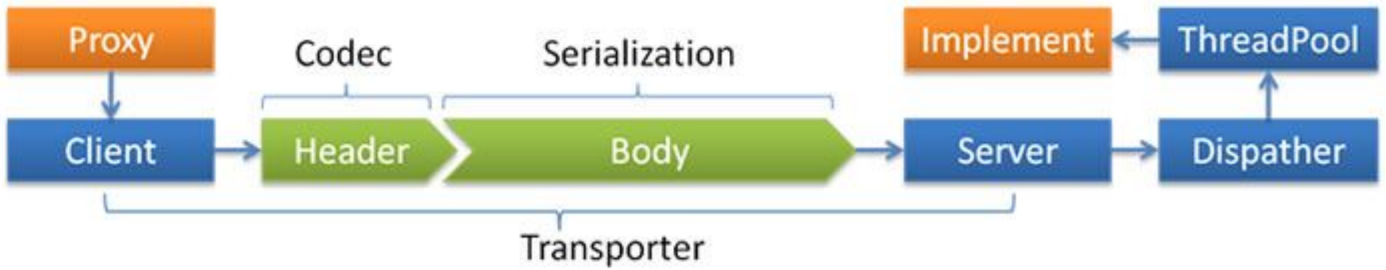
或：

```
<dubbo:service interface="...">
    <dubbo:method name="..." loadbalance="roundrobin"/>
</dubbo:service>
```

或：

```
<dubbo:reference interface="...">
    <dubbo:method name="..." loadbalance="roundrobin"/>
</dubbo:reference>
```

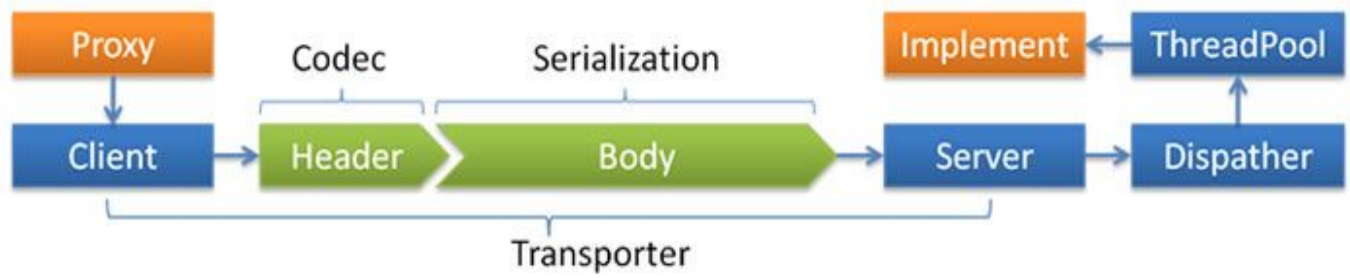
线程模型



❖ 事件处理线程说明

- 如果事件处理的逻辑能迅速完成，并且不会发起新的IO请求，比如只是在内存中记个标识，则直接在IO线程上处理更快，因为减少了线程池调度。
- 但如果事件处理逻辑较慢，或者需要发起新的IO请求，比如需要查询数据库，则必须派发到线程池，否则IO线程阻塞，将导致不能接收其它请求。
- 如果用IO线程处理事件，又在事件处理过程中发起新的IO请求，比如在连接事件中发起登录请求，会报“可能引发死锁”异常，但不会真死锁。

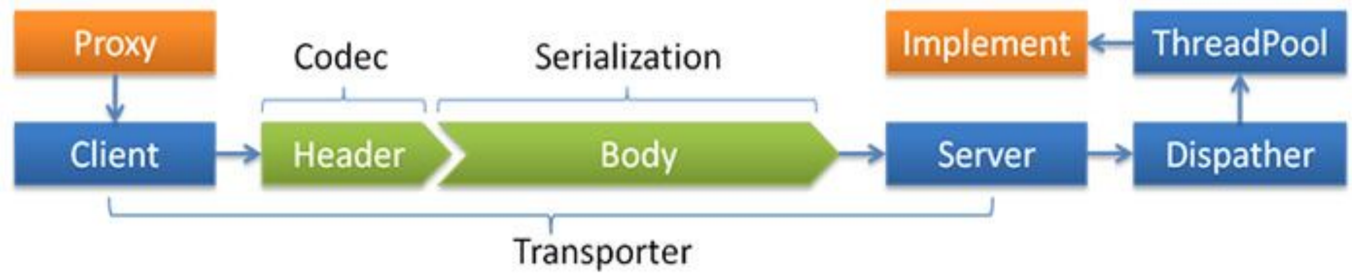
线程模型



❖Dispatcher

- all 所有消息都派发到线程池，包括请求，响应，连接事件，断开事件，心跳等。
- direct 所有消息都不派发到线程池，全部在IO线程上直接执行。
- message 只有请求响应消息派发到线程池，其它连接断开事件，心跳等消息，直接在IO线程上执行。
- execution 只请求消息派发到线程池，不含响应，响应和其它连接断开事件，心跳等消息，直接在IO线程上执行。
- connection 在IO线程上，将连接断开事件放入队列，有序逐个执行，其它消息派发到线程池。

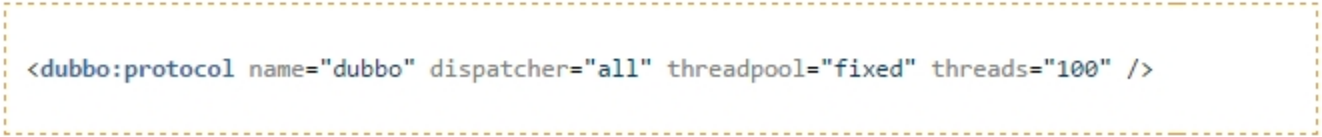
线程模型



❖ThreadPool

- fixed 固定大小线程池，启动时建立线程，不关闭，一直持有。(缺省)
- cached 缓存线程池，空闲一分钟自动删除，需要时重建。
- limited 可伸缩线程池，但池中的线程数只会增长不会收缩。(为避免收缩时突然来了大流量引起的性能问题)。

配置如：



```
<dubbo:protocol name="dubbo" dispatcher="all" threadpool="fixed" threads="100" />
```

多协议

- 不同服务在性能上适用不同协议进行传输，比如大数据用短连接协议，小数据大并发用长连接协议。

```
<?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://code.alibabatech.com/schema/dubbo"
    xsi:schemaLocation="http://www.springframework.org/schema/beanshttp://www.springframework.org/schema/beans/spring-
beans.xsdhttp://code.alibabatech.com/schema/dubbohttp://code.alibabatech.com/schema/dubbo/dubbo.xsd">

    <dubbo:application name="world"  />
    <dubbo:registry id="registry" address="10.20.141.150:9090" username="admin" password="hello1234" />

    <!-- 多协议配置 -->
    <dubbo:protocol name="dubbo" port="20880" />
    <dubbo:protocol name="rmi" port="1099" />

    <!-- 使用dubbo协议暴露服务 -->
    <dubbo:service interface="com.alibaba.hello.api.HelloService" version="1.0.0" ref="helloService" protocol="dubbo" />
    <!-- 使用rmi协议暴露服务 -->
    <dubbo:service interface="com.alibaba.hello.api.DemoService" version="1.0.0" ref="demoService" protocol="rmi" />

</beans>
```

多协议

- 多协议暴露服务

```
<?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://code.alibabatech.com/schema/dubbo"
    xsi:schemaLocation="http://www.springframework.org/schema/beanshttp://www.springframework.org/schema/beans/spring-
beans.xsdhttp://code.alibabatech.com/schema/dubbohttp://code.alibabatech.com/schema/dubbo/dubbo.xsd">

    <dubbo:application name="world"  />
    <dubbo:registry id="registry" address="10.20.141.150:9090" username="admin" password="hello1234" />

    <!-- 多协议配置 -->
    <dubbo:protocol name="dubbo" port="20880" />
    <dubbo:protocol name="hessian" port="8080" />

    <!-- 使用多个协议暴露服务 -->
    <dubbo:service id="helloService" interface="com.alibaba.hello.api.HelloService" version="1.0.0" protocol="dubbo,hessian" />

</beans>
```

多协议

```
<?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://code.alibabatech.com/schema/dubbo"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://code.alibabatech.com/schema/dubbo
    http://code.alibabatech.com/schema/dubbo/dubbo.xsd">
     <!-- 消费方应用名，用于计算依赖关系，不是匹配条件，不要与提供方一样 -->
    <dubbo:application name="DubboDemo01_"  />
    <dubbo:registry address="localhost:2181" protocol="zookeeper" />
    <!-- 生成远程服务代理，可以和本地bean一样使用demoService -->
    <dubbo:reference id="userService" check="true"
        interface="com.jiangzz.service.IUserService" protocol="rmi"/>
</beans>
```

指定协议

多注册中心

❖ 多注册中心注册

比如：中文站有些服务来不及在青岛部署，只在杭州部署，而青岛的其它应用需要引用此服务，就可以将服务同时注册到两个注册中心。

```
<?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://code.alibabatech.com/schema/dubbo"
    xsi:schemaLocation="http://www.springframework.org/schema/beanshttp://www.springframework.org/schema/beans/spring-
beans.xsdhttp://code.alibabatech.com/schema/dubbohttp://code.alibabatech.com/schema/dubbo/dubbo.xsd">

    <dubbo:application name="world"  />

    <!-- 多注册中心配置 -->
    <dubbo:registry id="hangzhouRegistry" address="10.20.141.150:9090" />
    <dubbo:registry id="qingdaoRegistry" address="10.20.141.151:9010" default="false" />

    <!-- 向多个注册中心注册 -->
    <dubbo:service interface="com.alibaba.hello.api.HelloService" version="1.0.0" ref="helloService" registry="hangzhouRegistry,qingdaoRegistry" />

</beans>
```

多注册中心

- 不同服务使用不同注册中心

比如：CRM有些服务是专门为国际站设计的，有些服务是专门为中文站设计的。

```
<?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://code.alibabatech.com/schema/dubbo"
    xsi:schemaLocation="http://www.springframework.org/schema/beanshttp://www.springframework.org/schema/beans/spring-
beans.xsdhttp://code.alibabatech.com/schema/dubbohttp://code.alibabatech.com/schema/dubbo/dubbo.xsd">

    <dubbo:application name="world"  />

    <!-- 多注册中心配置 -->
    <dubbo:registry id="chinaRegistry" address="10.20.141.150:9090" />
    <dubbo:registry id="intlRegistry" address="10.20.154.177:9010" default="false" />

    <!-- 向中文站注册中心注册 -->
    <dubbo:service interface="com.alibaba.hello.api.HelloService" version="1.0.0" ref="helloService" registry="chinaRegistry" />

    <!-- 向国际站注册中心注册 -->
    <dubbo:service interface="com.alibaba.hello.api.DemoService" version="1.0.0" ref="demoService" registry="intlRegistry" />

</beans>
```

多注册中心

❖ 多注册中心引用

CRM需同时调用中文站和国际站的PC2服务，PC2在中文站和国际站均有部署，接口及版本号都一样，但连的数据库不一样。

```
<?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://code.alibabatech.com/schema/dubbo"
    xsi:schemaLocation="http://www.springframework.org/schema/beanshttp://www.springframework.org/schema/beans/spring-
beans.xsdhttp://code.alibabatech.com/schema/dubbohttp://code.alibabatech.com/schema/dubbo/dubbo.xsd">

    <dubbo:application name="world"  />

    <!-- 多注册中心配置 -->
    <dubbo:registry id="chinaRegistry" address="10.20.141.150:9090" />
    <dubbo:registry id="intlRegistry" address="10.20.154.177:9010" default="false" />

    <!-- 引用中文站服务 -->
    <dubbo:reference id="chinaHelloService" interface="com.alibaba.hello.api.HelloService" version="1.0.0" registry="chinaRegistry" />

    <!-- 引用国际站站服务 -->
    <dubbo:reference id="intlHelloService" interface="com.alibaba.hello.api.HelloService" version="1.0.0" registry="intlRegistry" />

</beans>
```

服务分组

- 当一个接口有多种实现时，可以用group区分。

```
<dubbo:service group="feedback" interface="com.xxx.IndexService" />
<dubbo:service group="member" interface="com.xxx.IndexService" />
```

```
<dubbo:reference id="feedbackIndexService" group="feedback" interface="com.xxx.IndexService" />
<dubbo:reference id="memberIndexService" group="member" interface="com.xxx.IndexService" />
```

意组：(2.2.0以上版本支持，总是只调一个可用组的实现)

```
<dubbo:reference id="barService" interface="com.foo.BarService" group="*" />
```

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。

```
<dubbo:service interface="com.foo.BarService" version="1.0.0" />
```

```
<dubbo:service interface="com.foo.BarService" version="2.0.0" />
```

```
<dubbo:reference id="barService" interface="com.foo.BarService" version="1.0.0" />
```

```
<dubbo:reference id="barService" interface="com.foo.BarService" version="2.0.0" />
```

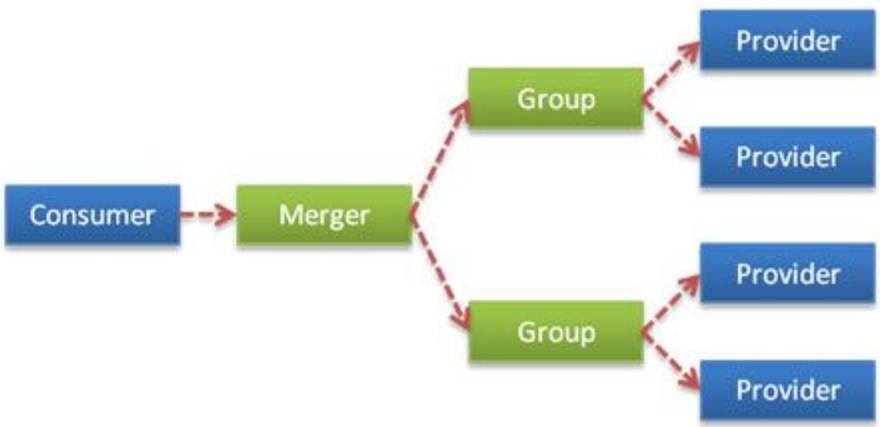
区分版本：(2.2.0以上版本支持)

```
<dubbo:reference id="barService" interface="com.foo.BarService" version="*" />
```

- 在低压力时间段，先升级一半提供者为新版本
- 再将所有消费者升级为新版本
- 然后将剩下的一半提供者升级为新版本

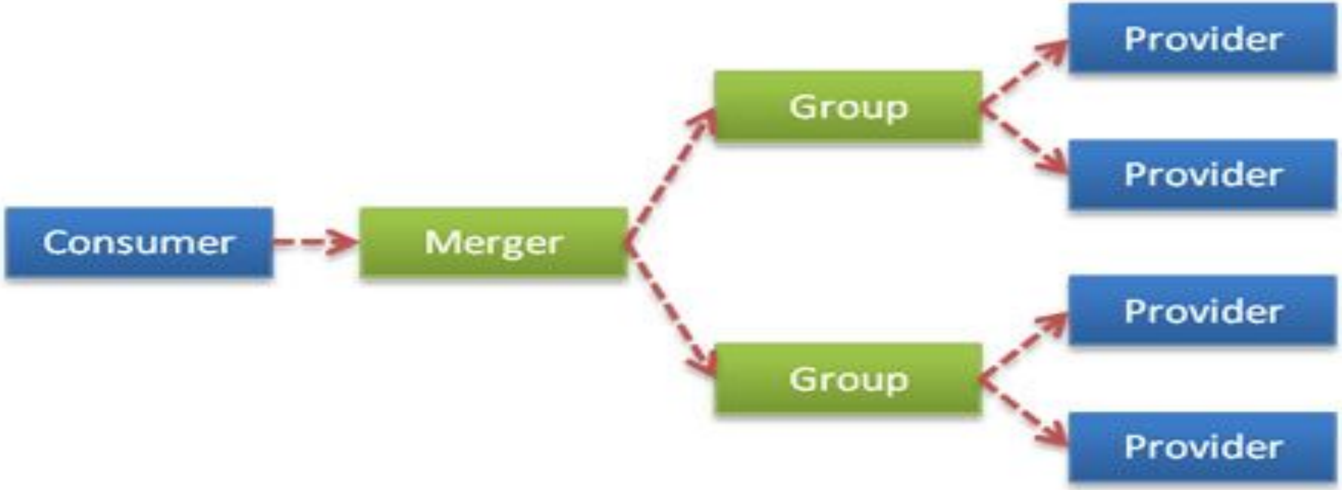
分组聚合

- 按组合并返回结果，比如菜单服务，接口一样，但有多种实现，用group区分，现在消费方需从每种group中调用一次返回结果，合并结果返回，这样就可以实现聚合菜单项。



分组聚合

默认情况下Dubbo支持结果集的合并例如:list、map、set、8种基本数据类型的合并、数组类型合并。



分组聚合

配置如：(搜索所有分组)

```
<dubbo:reference interface="com.xxx.MenuService" group="*" merger="true" />
```

或：(合并指定分组)

```
<dubbo:reference interface="com.xxx.MenuService" group="aaa,bbb" merger="true" />
```

或：(指定方法合并结果，其它未指定的方法，将只调用一个Group)

```
<dubbo:reference interface="com.xxx.MenuService" group="*">
    <dubbo:method name="getMenuItems" merger="true" />
</dubbo:service>
```

或：(某个方法不合并结果，其它都合并结果)

```
<dubbo:reference interface="com.xxx.MenuService" group="*" merger="true">
    <dubbo:method name="getMenuItems" merger="false" />
</dubbo:service>
```

分组聚合

指定合并方法，将调用返回结果的指定方法进行合并，合并方法的参数类型必须是返回结果类型本身

```
<dubbo:reference interface="com.xxx.MenuService" group="*">
    <dubbo:method name="getMenuItems" merger=".addAll" />
</dubbo:service>
```

注意：这里的addAll方法必须是方法返回值类型所具有的方法。

结果缓存

- 结果缓存，用于加速热门数据的访问速度，Dubbo提供声明式缓存，以减少用户加缓存的工作量。2.1.0以上版本支持

  - lru 基于最近最少使用原则删除多余缓存，保持最热的数据被缓存。
  - threadlocal 当前线程缓存，比如一个页面渲染，用到很多portal，每个portal都要去查用户信息，通过线程缓存，可以减少这种多余访问。
  - jcache 与JSR107集成，可以桥接各种缓存实现。

配置如：

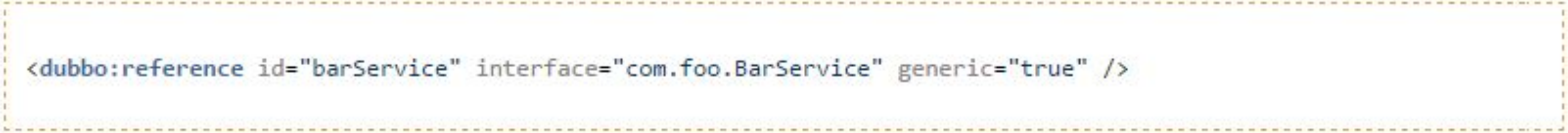
```
<dubbo:reference interface="com.foo.BarService" cache="lru" />
```

或：

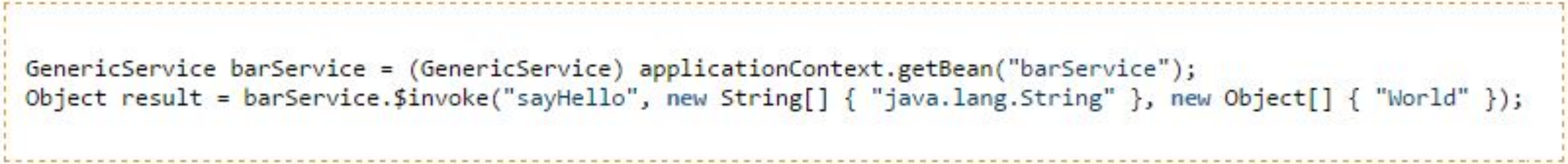
```
<dubbo:reference interface="com.foo.BarService">
    <dubbo:method name="findBar" cache="lru" />
</dubbo:reference>
```

泛化引用

- 泛接口调用方式主要用于客户端没有API接口及模型类元的情况，参数及返回值中的所有POJO均用Map表示，通常用于框架集成，比如：实现一个通用的服务测试框架，可通过GenericService调用所有服务实现。



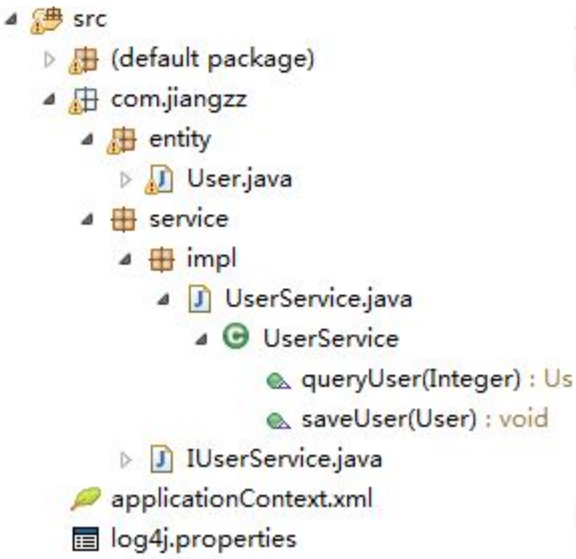
```
<dubbo:reference id="barService" interface="com.foo.BarService" generic="true" />
```



```
GenericService barService = (GenericService) applicationContext.getBean("barService");
Object result = barService.$invoke("sayHello", new String[] { "java.lang.String" }, new Object[] { "World" });
```

注意：interface里面必须写provider的全类名，result的返回值类型是HashMap。

泛化引用



```
src
  (default package)
  com.jiangzz
    entity
      User.java
    service
      impl
        UserService.java
          UserService
            queryUser(Integer) : Us
            saveUser(User) : void
      IUserService.java
  applicationContext.xml
  log4j.properties
```

服务方暴漏的服务

```xml
<?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://code.alibabatech.com/schema/dubbo"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://code.alibabatech.com/schema/dubbo
    http://code.alibabatech.com/schema/dubbo/dubbo.xsd">
    <dubbo:application name="DubboDemo03"/>
    <dubbo:registry protocol="zookeeper" address="localhost:2181"/>
    <dubbo:reference id="userService"
        interface="com.jiangzz.service.IUserService"
        generic="true"/>
</beans>
```

客户端的代码

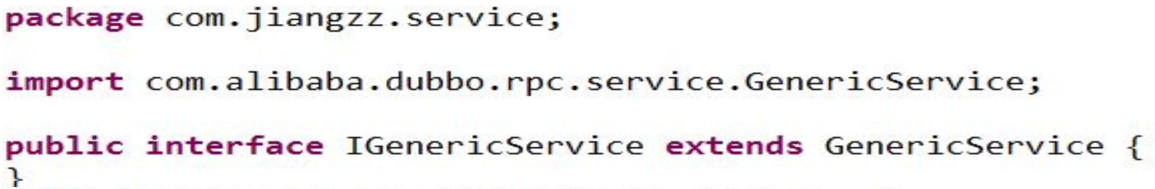
泛化引用

```
package com.jiangzz.service;

import java.util.List;

import com.jiangzz.entity.User;

public interface IUserService {
    public User queryUser(Integer id);
    public void saveUser(User user);
    public void saveAllUser(List<User> users);
}
```

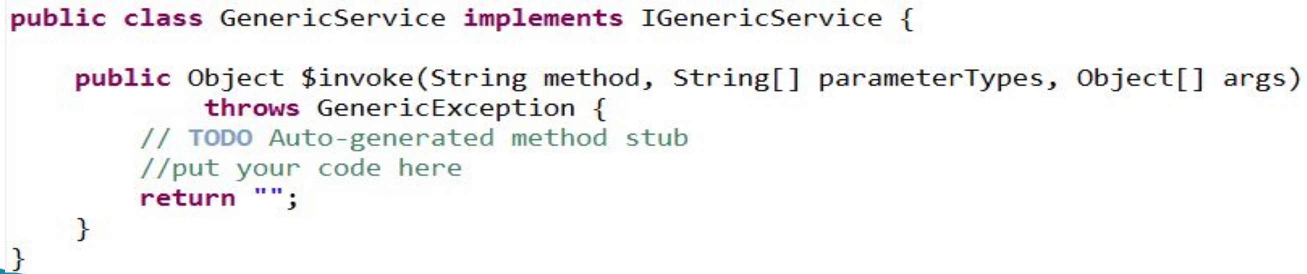
```
GenericService bean = (GenericService) ctx.getBean("userService");
Object result1 = bean.$invoke("queryUser", new String[]{"java.lang.Integer"},  new Object[]{1});
Map<String, Object> map = new HashMap<String, Object>();
map.put("class", "com.jiangzz.entity.User");
map.put("name", "zhangsan");
map.put("id", 1);
map.put("birthDay", new Date());
bean.$invoke("saveUser", new String[]{"com.jiangzz.entity.User"}, new Object[]{map});
Map<String, Object> mapUser = new HashMap<String, Object>();
// 注意：如果参数类型是接口，或者List等丢失泛型，可通过class属性指定类型。
mapUser.put("class", "com.jiangzz.entity.User");
mapUser.put("name", "zhangsan");
mapUser.put("id", 1);
mapUser.put("birthDay", new Date());
List<Object> objects=new ArrayList<Object>();
objects.add(mapUser);
mapUser.put("id", 2);
objects.add(mapUser);
bean.$invoke("saveAllUser", new String[]{"java.util.List"}, new Object[]{objects});
```

泛化实现

- ❖ 泛接口实现方式主要用于服务器端没有API接口及模型类元的情况，参数及返回值中的所有POJO均用Map表示，通常用于框架集成，比如：实现一个通用的远程服务Mock框架，可通过实现GenericService接口处理所有服务请求。



```
package com.jiangzz.service;

import com.alibaba.dubbo.rpc.service.GenericService;

public interface IGenericService extends GenericService {
}
```



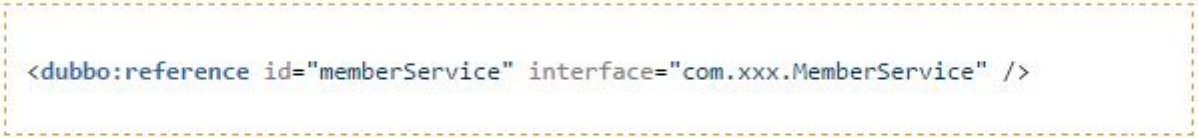
```
public class GenericService implements IGenericService {

    public Object $invoke(String method, String[] parameterTypes, Object[] args)
            throws GenericException {
        // TODO Auto-generated method stub
        //put your code here
        return "";
    }
}
```

泛化实现

```
<?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://code.alibabatech.com/schema/dubbo"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://code.alibabatech.com/schema/dubbo
    http://code.alibabatech.com/schema/dubbo/dubbo.xsd">
    <!-- 配置实体bean -->
    <bean id="userService" class="com.jiangzz.service.impl.GenericService" />
    <!-- 提供方应用信息，用于计算依赖关系 -->
    <dubbo:application name="DubboDemo01"/>
    <!-- 提供注册这，用于注册服务 -->
    <dubbo:registry address="localhost:2181" protocol="zookeeper"/>
    <!-- 用dubbo协议在20880端口暴露服务 -->
    <dubbo:protocol name="dubbo" port="20880" />
    <!-- 声明需要暴露的服务接口 -->
    <dubbo:service interface="com.jiangzz.service.IGenericService"  ref="userService"/>
</beans>
```

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。所有服务自动实现EchoService接口，只需将任意服务引用强制转型为EchoService，即可使用。



```
<dubbo:reference id="memberService" interface="com.xxx.MemberService" />
```

```
MemberService memberService = ctx.getBean("memberService"); // 远程服务引用

EchoService echoService = (EchoService) memberService; // 强制转型为EchoService

String status = echoService.$echo("OK"); // 回声测试可用性

assert(status.equals("OK"))
```

上下文信息

- 上下文中存放的是当前调用过程中所需的环境信息。
- 所有配置信息都将转换为URL的参数，参见《配置项一览表》中的“对应URL参数”一列。

(1) 服务消费方

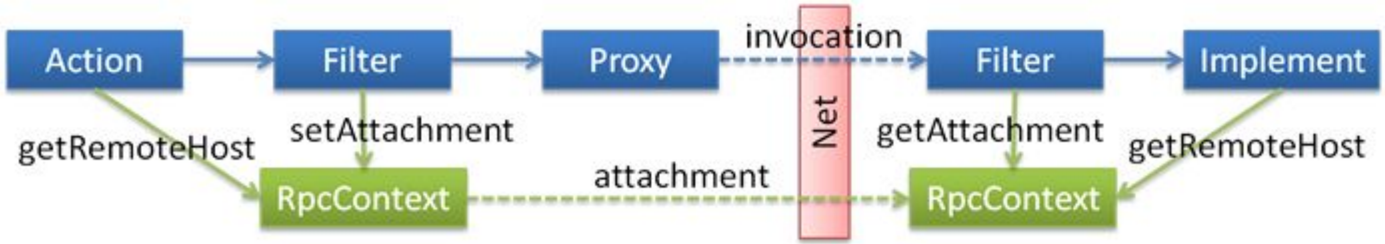
```
xxxService.xxx(); // 远程调用
boolean isConsumerSide = RpcContext.getContext().isConsumerSide(); // 本端是否为消费端，这里
String serverIP = RpcContext.getContext().getRemoteHost(); // 获取最后一次调用的提供方IP地址
String application = RpcContext.getContext().getUrl().getParameter("application"); // 获取
// ...
yyyService.yyy(); // 注意：每发起RPC调用，上下文状态会变化
// ...
```

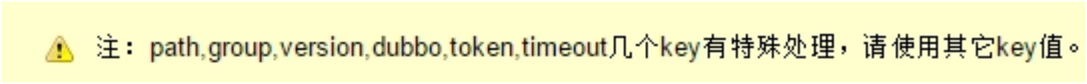
上下文信息

(2) 服务提供方

```
public class XxxServiceImpl implements XxxService {

    public void xxx() { // 服务方法实现
        boolean isProviderSide = RpcContext.getContext().isProviderSide(); // 本端是否为提供
        String clientIP = RpcContext.getContext().getRemoteHost(); // 获取调用方IP地址
        String application = RpcContext.getContext().getUrl().getParameter("application");
        // ...
        yyyService.yyy(); // 注意：每发起RPC调用，上下文状态会变化
        boolean isProviderSide = RpcContext.getContext().isProviderSide(); // 此时本端变成消
        // ...
    }

}
```

隐式传参




⚠ 注：path,group,version,dubbo,token,timeout几个key有特殊处理，请使用其它key值。

隐式传参

⚠ 注：path,group,version,dubbo,token,timeout几个key有特殊处理，请使用其它key值。

(1) 服务消费方

```
// 隐式传参，后面的远程调用都会隐式将这些参数发送到服务器端，类似cookie，用于框架集成，不建议常规业务使用
RpcContext.getContext().setAttachment("index", "1");
xxxService.xxx(); // 远程调用
// ...
```

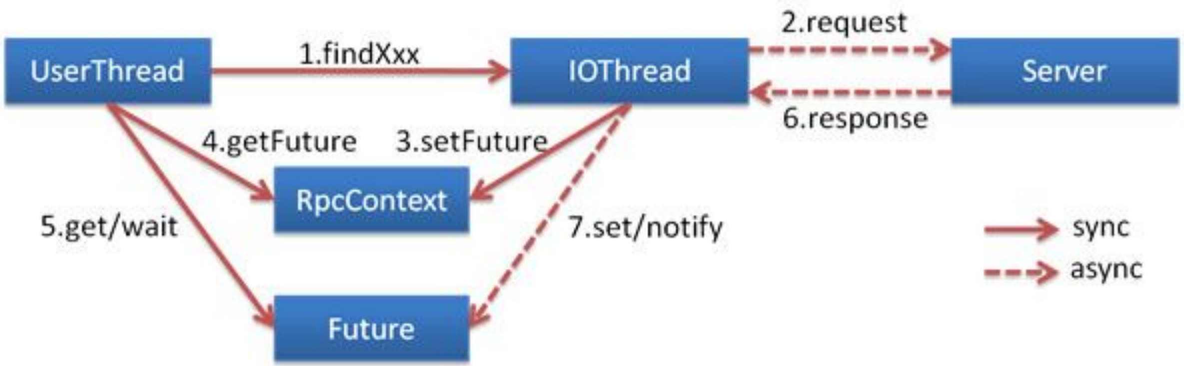
【注】 setAttachment设置的KV，在完成下面一次远程调用会被清空。即多次远程调用要多次设置。

(2) 服务提供方

```
public class XxxServiceImpl implements XxxService {

    public void xxx() { // 服务方法实现
        // 获取客户端隐式传入的参数，用于框架集成，不建议常规业务使用
        String index = RpcContext.getContext().getAttachment("index");
        // ...
    }
}
```

异步调用

- 基于NIO的非阻塞实现并行调用，客户端不需要启动多线程即可完成并行调用多个远程服务，相对多线程开销较小。



异步调用

配置声明：

consumer.xml

```xml
<dubbo:reference id="fooService" interface="com.alibaba.foo.FooService">
      <dubbo:method name="findFoo" async="true" />
</dubbo:reference>
<dubbo:reference id="barService" interface="com.alibaba.bar.BarService">
      <dubbo:method name="findBar" async="true" />
</dubbo:reference>
```

调用代码：

```java
fooService.findFoo(fooId); // 此调用会立即返回null
Future<Foo> fooFuture = RpcContext.getContext().getFuture(); // 拿到调用的Future引用，当结果返回后，会被通知和设置到此Future。

barService.findBar(barId); // 此调用会立即返回null
Future<Bar> barFuture = RpcContext.getContext().getFuture(); // 拿到调用的Future引用，当结果返回后，会被通知和设置到此Future。

// 此时findFoo和findBar的请求同时在执行，客户端不需要启动多线程来支持并行，而是借助NIO的非阻塞完成。

Foo foo = fooFuture.get(); // 如果foo已返回，直接拿到返回值，否则线程wait住，等待foo返回后，线程会被notify唤醒。
Bar bar = barFuture.get(); // 同理等待bar返回。

// 如果foo需要5秒返回，bar需要6秒返回，实际只需等6秒，即可获取到foo和bar，进行接下来的处理。
```

异步调用

你也可以设置是否等待消息发出：(异步总是不等待返回)

- sent="true" 等待消息发出，消息发送失败将抛出异常。
- sent="false" 不等待消息发出，将消息放入IO队列，即刻返回。

```
<dubbo:method name="findFoo" async="true" sent="true" />
```

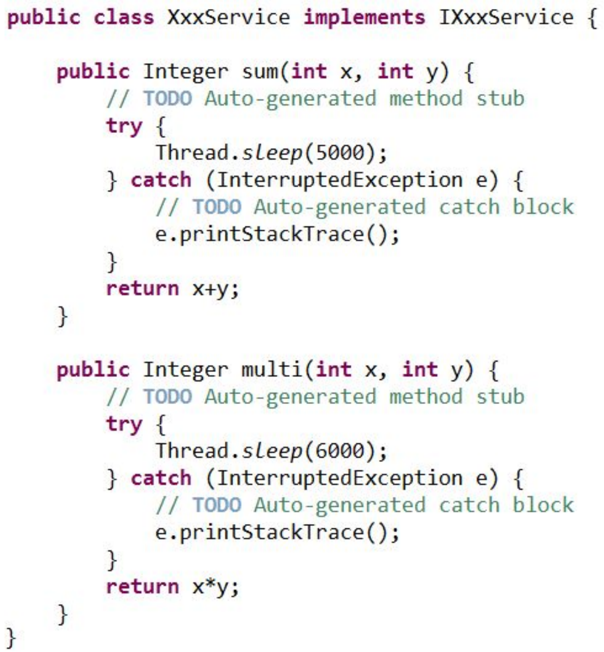
如果你只是想异步，完全忽略返回值，可以配置return="false"，以减少Future对象的创建和管理成本：

```
<dubbo:method name="findFoo" async="true" return="false" />
```

注意： 当return=false就代表Future#get()不再是阻塞的了， 同时返回值永远是null。

异步调用

服务端

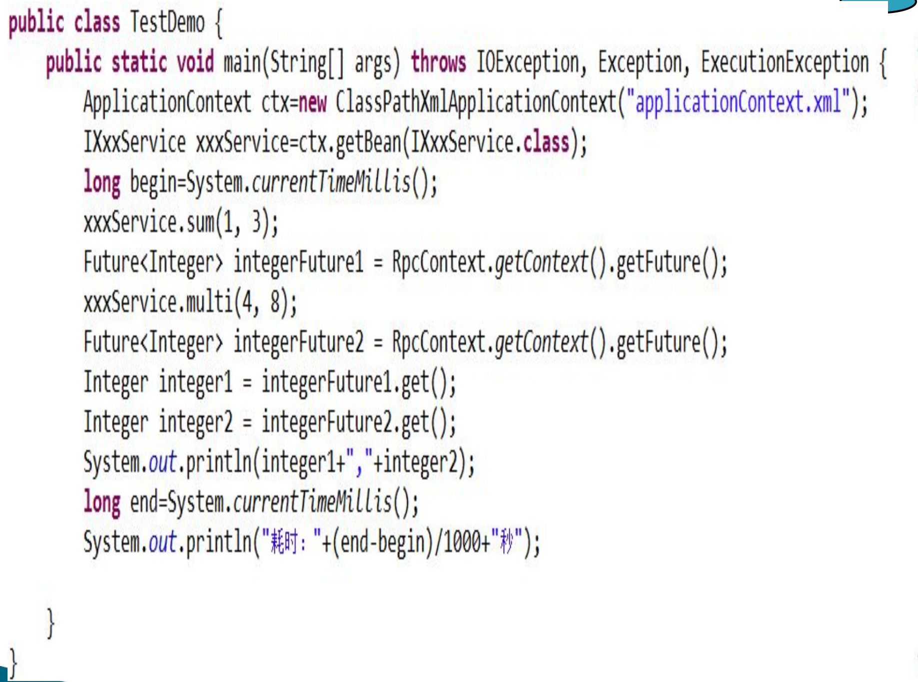


```
public class XxxService implements IXxxService {

    public Integer sum(int x, int y) {
        // TODO Auto-generated method stub
        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return x+y;
    }

    public Integer multi(int x, int y) {
        // TODO Auto-generated method stub
        try {
            Thread.sleep(6000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return x*y;
    }
}
```

```
<?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://code.alibabatech.com/schema/dubbo"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://code.alibabatech.com/schema/dubbo
    http://code.alibabatech.com/schema/dubbo/dubbo.xsd">

    <bean id="xxxService" class="com.jiangzz.service.impl.XxxService"/>
    <dubbo:application name="DubboDemo04"/>
    <dubbo:registry protocol="zookeeper" address="localhost:2181"/>
    <dubbo:protocol name="dubbo" port="20880" />
    <dubbo:service interface="com.jiangzz.service.IXxxService"
        ref="xxxService"/>
</beans>
```

异步调用

客户端，耗时6秒

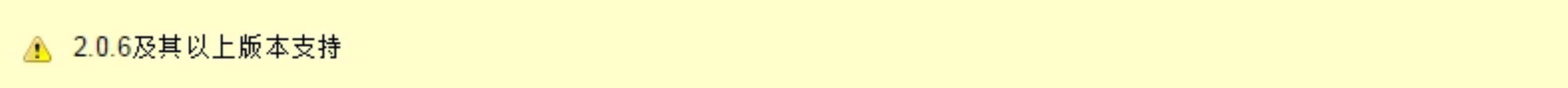


```
public class TestDemo {
    public static void main(String[] args) throws IOException, Exception, ExecutionException {
        ApplicationContext ctx=new ClassPathXmlApplicationContext("applicationContext.xml");
        IXxxService xxxService=ctx.getBean(IXxxService.class);
        long begin=System.currentTimeMillis();
        xxxService.sum(1, 3);
        Future<Integer> integerFuture1 = RpcContext.getContext().getFuture();
        xxxService.multi(4, 8);
        Future<Integer> integerFuture2 = RpcContext.getContext().getFuture();
        Integer integer1 = integerFuture1.get();
        Integer integer2 = integerFuture2.get();
        System.out.println(integer1+","+integer2);
        long end=System.currentTimeMillis();
        System.out.println("耗时："+(end-begin)/1000+"秒");

    }
}
```

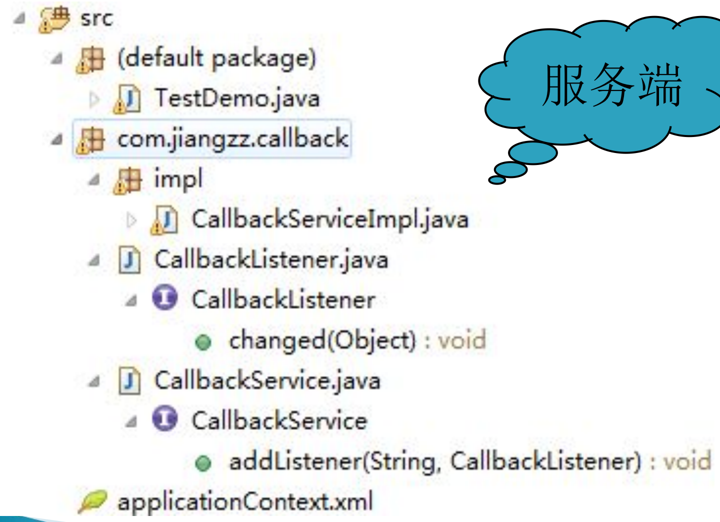
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://code.alibabatech.com/schema/dubbo
    http://code.alibabatech.com/schema/dubbo/dubbo.xsd">

    <bean id="xxxService" class="com.jiangzz.service.impl.XxxService"/>
    <dubbo:application name="DubboDemo04"/>
    <dubbo:registry protocol="zookeeper" address="localhost:2181"/>
    <dubbo:protocol name="dubbo" port="20880" />
    <dubbo:service interface="com.jiangzz.service.IXxxService"
        ref="xxxService"/>
</beans>
```

参数回调

- 参数回调方式与调用本地callback或listener相同，只需要在Spring的配置文件中声明哪个参数是callback类型即可，Dubbo将基于长连接生成反向代理，这样就可以从服务器端调用客户端逻辑。



⚠ 2.0.6及其以上版本支持



```
src
  (default package)
    TestDemo.java
  com.jiangzz.callback
    impl
      CallbackServiceImpl.java
    CallbackListener.java
      CallbackListener
        changed(Object) : void
    CallbackService.java
      CallbackService
        addListener(String, CallbackListener) : void
  applicationContext.xml
```

服务端

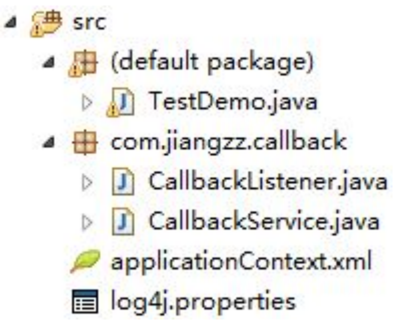
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://code.alibabatech.com/schema/dubbo
    http://code.alibabatech.com/schema/dubbo/dubbo.xsd">

    <bean id="callbackService" class="com.jiangzz.callback.impl.CallbackServiceImpl"/>
    <dubbo:application name="DubboDemo04"/>
    <dubbo:registry protocol="zookeeper" address="localhost:2181"/>
    <dubbo:protocol name="dubbo" port="20880" />
    <dubbo:service interface="com.jiangzz.callback.CallbackService"
        ref="callbackService" connections="1" callbacks="1000">
            <dubbo:method name="addListener" >
                <dubbo:argument callback="true"
                    type="com.jiangzz.callback.CallbackListener"/>
            </dubbo:method>
    </dubbo:service>
</beans>
```

参数回调

- 参数回调方式与调用本地callback或listener相同，只需要在Spring的配置文件中声明哪个参数是callback类型即可，Dubbo将基于长连接生成反向代理，这样就可以从服务器端调用客户端逻辑。

⚠ 2.0.6及其以上版本支持

客户端

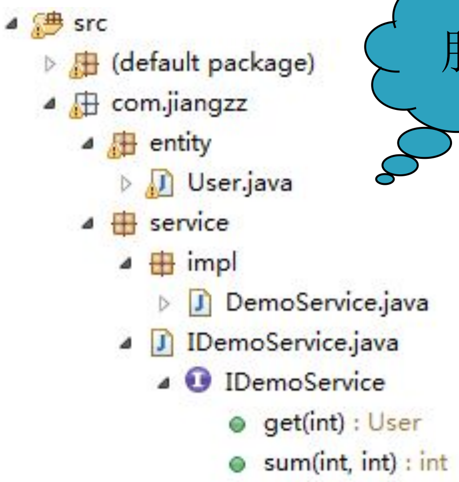

```
src
  (default package)
    TestDemo.java
  com.jiangzz.callback
    CallbackListener.java
    CallbackService.java
  applicationContext.xml
  log4j.properties
```


```
<?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://code.alibabatech.com/schema/dubbo"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://code.alibabatech.com/schema/dubbo
    http://code.alibabatech.com/schema/dubbo/dubbo.xsd">

    <dubbo:application name="DubboDemo07"/>
    <dubbo:registry protocol="zookeeper" address="localhost:2181"/>
    <dubbo:reference id="callbackService"
        interface="com.jiangzz.callback.CallbackService"/>
</beans>
```

事件通知

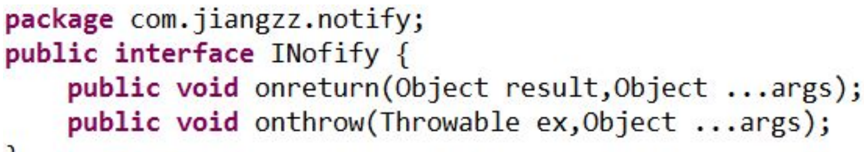
- 在调用之前，调用之后，出现异常时，会触发oninvoke, onreturn, onthrow三个事件，可以配置当事件发生时，通知哪个类的哪个方法。

服务端

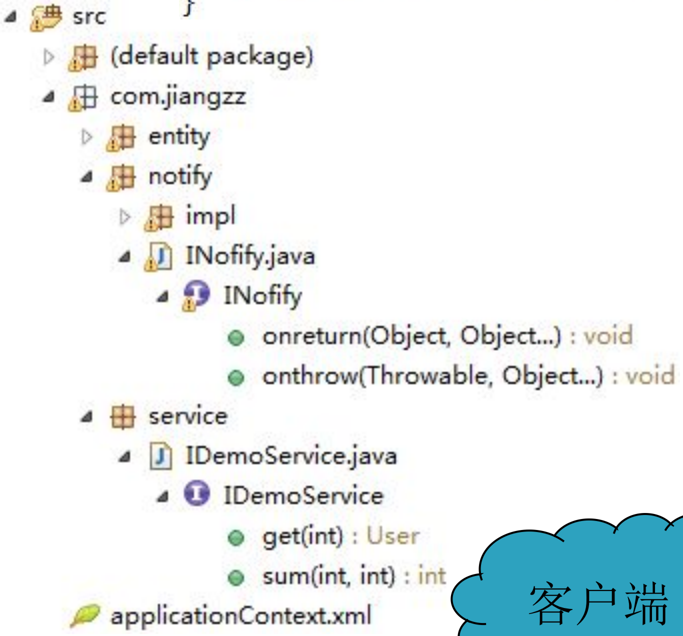


```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
    xsi:schemaLocation="http://www.springframework.org/schema/be
    http://www.springframework.org/schema/beans/spring-beans.xs
    http://code.alibabatech.com/schema/dubbo
    http://code.alibabatech.com/schema/dubbo/dubbo.xsd">

    <bean id="demoService" class="com.jiangzz.service.impl.Demo
    <dubbo:application name="DubboDemo07"/>
    <dubbo:registry protocol="zookeeper"
        address="localhost:2181"/>
    <dubbo:protocol name="dubbo" port="20880" />
    <dubbo:service interface="com.jiangzz.service.IDemoService"
              ref="demoService"/>
</beans>
```

事件通知



```java
package com.jiangzz.notify;
public interface INofify {
    public void onreturn(Object result,Object ...args);
    public void onthrow(Throwable ex,Object ...args);
}
```



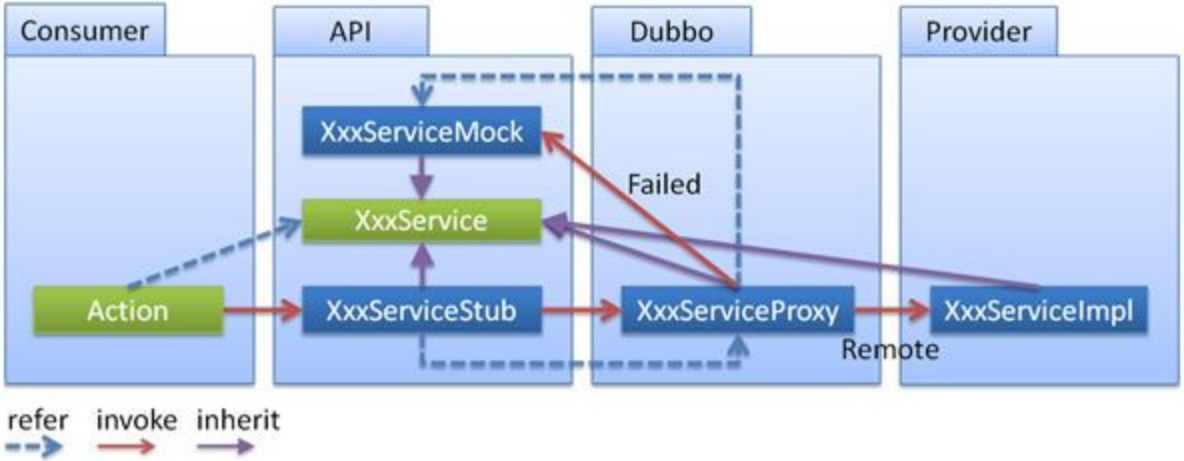
- src
  - (default package)
  - com.jiangzz
    - entity
    - notify
      - impl
      - INofify.java
        - INofify
          - onreturn(Object, Object...) : void
          - onthrow(Throwable, Object...) : void
    - service
      - IDemoService.java
        - IDemoService
          - get(int) : User
          - sum(int, int) : int
  - applicationContext.xml

```xml
<?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://code.alibabatech.com/schema/dubbo"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://code.alibabatech.com/schema/dubbo
    http://code.alibabatech.com/schema/dubbo/dubbo.xsd">

    <dubbo:application name="DubboDemo07_cleint"/>
    <dubbo:registry protocol="zookeeper" address="localhost:2181"/>

    <bean id ="notify" class = "com.jiangzz.notify.impl.Notify" />
    <dubbo:reference id="demoService" interface="com.jiangzz.service.IDemoService">
            <dubbo:method name="get" async="false" onreturn = "notify.onreturn"
                    onthrow="notify.onthrow" />
            <dubbo:method name="sum" async="false" onreturn = "notify.onreturn"
                    onthrow="notify.onthrow" />
    </dubbo:reference>
ans>
```

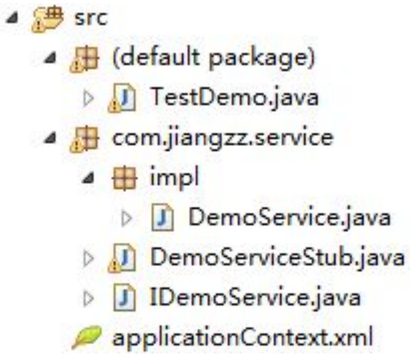
客户端

本地存根

- 远程服务后，客户端通常只剩下接口，而实现全在服务器端，但提供方有些时候想在客户端也执行部分逻辑，比如：做ThreadLocal缓存，提前验证参数，调用失败后伪造容错数据等等，此时就需要在API中带上Stub，客户端生成Proxy实，会把Proxy通过构造函数传给Stub，然后把Stub暴露组给用户，Stub可以决定要不要去调Proxy。



本地存根



- src
  - (default package)
    - TestDemo.java
  - com.jiangzz.service
    - impl
      - DemoService.java
    - DemoServiceStub.java
    - IDemoService.java
  - applicationContext.xml

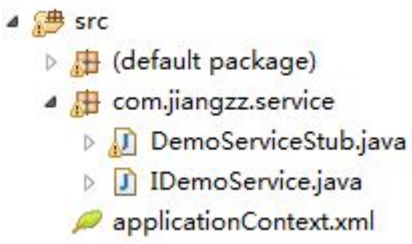
```
<bean id="demoService" class="com.jiangzz.service.impl.DemoService"/>

<dubbo:application name="DubboDemo08"/>
<dubbo:registry protocol="zookeeper"
    address="localhost:2181"/>
<dubbo:protocol name="dubbo" port="20880" />

<dubbo:service interface="com.jiangzz.service.IDemoService"
    stub="com.jiangzz.service.DemoServiceStub" ref="demoService" />
```

服务端

注意：端需要书写两个类都要实现IDemoService接口，继而实现客户端代码的调用。

本地存根

客户端



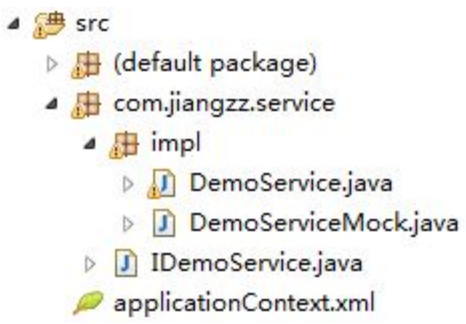
```
<dubbo:application name="DubboDemo08"/>
<dubbo:registry protocol="zookeeper"
    address="localhost:2181"/>
<dubbo:reference id="demoService" interface="com.jiangzz.service.IDemoService" />
<bean id="demoServiceStub" class="com.jiangzz.service.DemoServiceStub">
    <constructor-arg index="0" ref="demoService"></constructor-arg>
</bean>
```

本地伪装

- Mock通常用于服务降级，比如某验权服务，当服务提供方全部挂掉后，客户端不抛出异常，而是通过Mock数据返回授权失败。

- Mock是Stub的一个子集，便于服务提供方在客户端执行容错逻辑，因经常需要在出现RpcException(比如网络失败，超时等)时进行容错，而在出现业务异常(比如登录用户名密码错误)时不需要容错，如果用Stub，可能就需要捕获并依赖RpcException类，而用Mock就可以不依赖RpcException，因为它的约定就是只有出现RpcException时才执行。

本地伪装



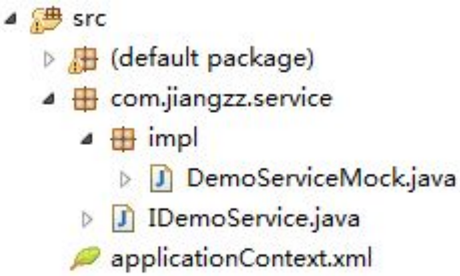
```
src
  (default package)
  com.jiangzz.service
    impl
      DemoService.java
      DemoServiceMock.java
    IDemoService.java
  applicationContext.xml
```

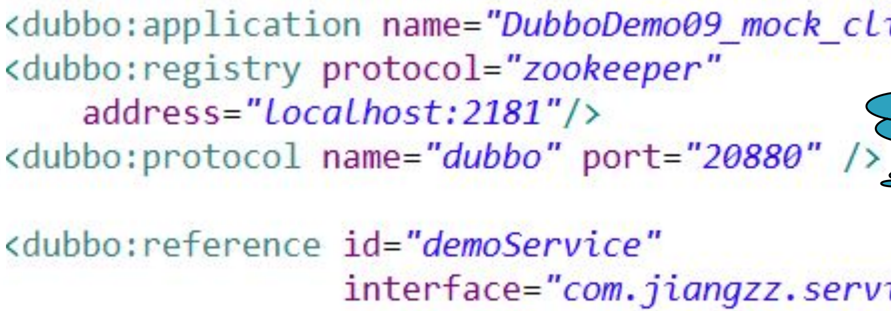
```
<bean id="demoService" class="com.jiangzz.service.impl.DemoService"/>

<dubbo:application name="DubboDemo09"/>
<dubbo:registry protocol="zookeeper"
    address="localhost:2181"/>
<dubbo:protocol name="dubbo" port="20880" />

<dubbo:service ref="demoService"
               interface="com.jiangzz.service.IDemoService"
               mock="com.jiangzz.service.impl.DemoServiceMock" />
```

服务端

注意：端需要书写两个类都要实现IDemoService接口，继而实现客户端代码的调用。

本地伪装





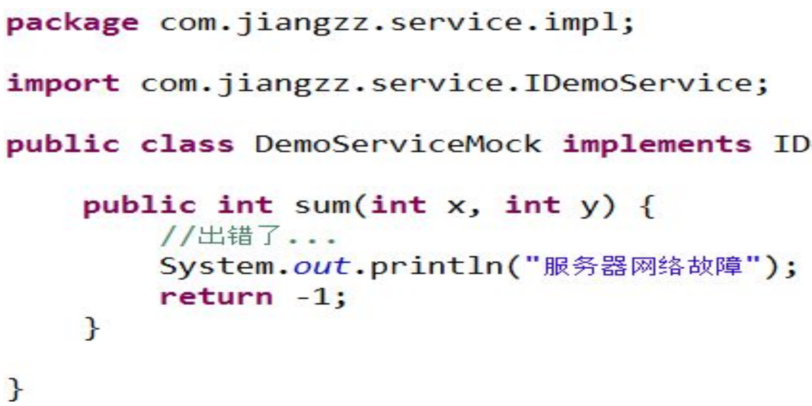
```
<dubbo:application name="DubboDemo09_mock_client"/>
<dubbo:registry protocol="zookeeper"
    address="localhost:2181"/>
<dubbo:protocol name="dubbo" port="20880" />

<dubbo:reference id="demoService"
                 interface="com.jiangzz.service.IDemoService"/>
```

客户端

请考虑改为Mock实现，并在Mock中return null。

如果只是想简单的忽略异常，在2.0.11以上版本可用：

```
<dubbo:service interface="com.foo.BarService" mock="return null" />
```



```
package com.jiangzz.service.impl;

import com.jiangzz.service.IDemoService;

public class DemoServiceMock implements IDemoService {

    public int sum(int x, int y) {
        //出错了...
        System.out.println("服务器网络故障");
        return -1;
    }

}
```

延迟暴露

- 如果你的服务需要Warmup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5秒暴露服务：

```
<dubbo:service delay="5000" />
```

延迟到Spring初始化完成后，再暴露服务：(基于Spring的ContextRefreshedEvent事件触发暴露)

```
<dubbo:service delay="-1" />
```

- 强烈建议不要在服务的实现类中有applicationContext.getBean()的调用，全部采用IoC注入的方式使用Spring的Bean。
- 如果实在要调getBean()，可以将Dubbo的配置放在Spring的最后加载。
- 如果不想依赖配置顺序，可以使用<dubbo:provider deplay=”-1” />，使Dubbo在Spring容器初始化完后，再暴露服务。
- 如果大量使用getBean()，相当于已经把Spring退化为工厂模式在用，可以将Dubbo的服务隔离单独的Spring容器。

并发控制

- 限制com.foo.BarService的每个方法，服务器端并发执行（或占用线程池线程数）不能超过10个：

```
<dubbo:service interface="com.foo.BarService" executes="10" />
```

```
<dubbo:service interface="com.foo.BarService">
    <dubbo:method name="sayHello" executes="10" />
</dubbo:service>
```

- 限制com.foo.BarService的每个方法，客户端并发执行（或占用连接的请求数）不能超过10个：

```
<dubbo:service interface="com.foo.BarService" actives="10" />
<dubbo:reference interface="com.foo.BarService" actives="10" />
```

- 限制com.foo.BarService的sayHello方法，每客户端并发执行（或占用连接的请求数）不能超过10个：

```
<dubbo:service interface="com.foo.BarService">
    <dubbo:method name="sayHello" actives="10" />
</dubbo:service>
```

```
<dubbo:reference interface="com.foo.BarService">
    <dubbo:method name="sayHello" actives="10" />
</dubbo:service>
```

注意：如果<dubbo:service>和<dubbo:reference>都配了actives，<dubbo:reference>优先。

并发控制

- Load Balance均衡

```
<dubbo:reference interface="com.foo.BarService" loadbalance="leastactive" />
<dubbo:service interface="com.foo.BarService" loadbalance="leastactive" />
```

配置服务的客户端的loadbalance属性为leastactive，此Loadbalance会调用并发数最小的Provider（Consumer端并发数）。

连接控制

- 限制服务器端接受的连接不能超过10个：（以连接在Server上，所以配置在Provider上）

```
<dubbo:provider protocol="dubbo" accepts="10" />
<dubbo:protocol name="dubbo" accepts="10" />
```

- 限制客户端服务使用连接连接数：(如果是长连接，比如Dubbo协议，connections表示该服务对每个提供者建立的长连接数)

```
<dubbo:reference interface="com.foo.BarService" connections="10" />
<dubbo:service interface="com.foo.BarService" connections="10" />
```

令牌

- 防止消费者绕过注册中心访问提供者
- 在注册中心控制权限，以决定要不要下发令牌给消费者
- 注册中心可灵活改变授权方式，而不需修改或升级提供者

Registry
2. Manage Token
3. Authorize Token

Provider
1.Generate Token
6. Verify Token

Consumer
4. Receive Token
5. Access with Token

令牌

- 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.foo.BarService" token="true" />
!--固定token令牌，相当于密码-->
<dubbo:provider interface="com.foo.BarService" 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.foo.BarService" token="true" />
<!--固定token令牌，相当于密码-->
<dubbo:service interface="com.foo.BarService" 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" />
```

Registry
2. Manage Token
3. Authorize Token

Provider
1.Generate Token
6. Verify Token

Consumer
4. Receive Token
5. Access with Token

路由

- 向注册中心写入路由规则：(通常由监控中心或治理中心的页面完成)

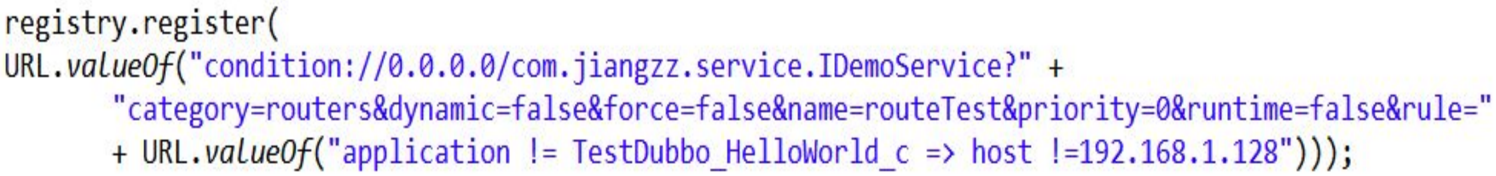
```
RegistryFactory registryFactory = ExtensionLoader.getExtensionLoader(RegistryFactory.class)
        .getAdaptiveExtension();
Registry registry =  registryFactory.getRegistry(URL.valueOf("zookeeper://192.168.1.102:2181"));
registry.register(
 URL.valueOf("condition://0.0.0.0/com.jiangzz.service.IDemoService?" +
       "category=routers&dynamic=false&force=false&name=routeTest&priority=0&runtime=false&rule="
       + URL.valueOf("host=192.168.1.102=>host!=192.168.1.102")));
```

说明：如果是192.168.1.102的请求不允许192.168.1.102响应
等价写法：host=192.168.1.102=>host!=$host

IP路由

路由

- 向注册中心写入路由规则：(通常由监控中心或治理中心的页面完成)



```
registry.register(
URL.valueOf("condition://0.0.0.0/com.jiangzz.service.IDemoService?" +
        "category=routers&dynamic=false&force=false&name=routeTest&priority=0&runtime=false&rule="
        + URL.valueOf("application != TestDubbo_HelloWorld_c => host !=192.168.1.128")));
```

说明:给TestDubbo_HelloWorld_c 添加额外机器

App应用名路由

路由

- 向注册中心写入路由规则：(通常由监控中心或治理中心的页面完成)

```
RegistryFactory registryFactory = ExtensionLoader.getExtensionLoader(RegistryFactory.class)
         .getAdaptiveExtension();
 Registry registry =  registryFactory.getRegistry(URL.valueOf("zookeeper://192.168.1.102:2181"));

 registry.register(
 URL.valueOf("condition://0.0.0.0/com.jiangzz.service.IDemoService?" +
        "category=routers&dynamic=false&force=false&name=routeTest&priority=0&runtime=false&rule="
        + URL.valueOf("application = TestDubbo_HelloWorld_front => host =192.168.1.128")));
 registry.register(
        URL.valueOf("condition://0.0.0.0/com.jiangzz.service.IDemoService?" +
               "category=routers&dynamic=false&force=false&name=routeTest&priority=1&runtime=false&rule="
               + URL.valueOf("application = TestDubbo_HelloWorld_back => host =192.168.1.102")));
```

注意：做前后台分离，要求priority值必须不一样。

App应用名路由

协议参考手册

- dubbo://
- rmi://
- hessian://
- http://
- webservice://
- thrift://
- memcached://
- redis://

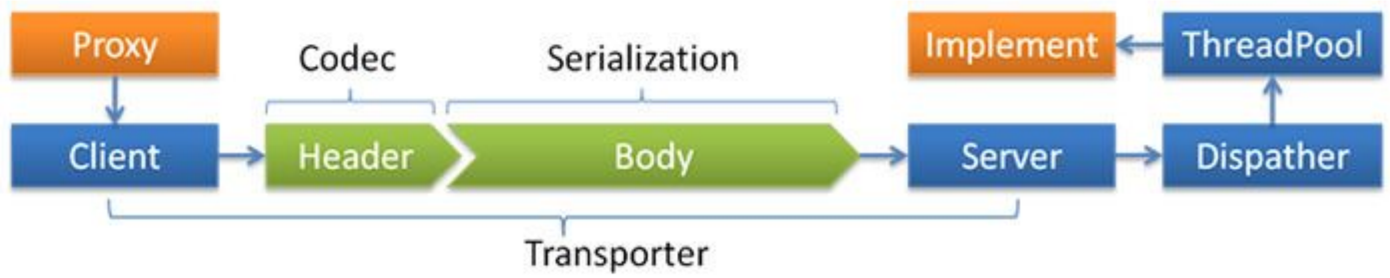
dubbo://

- Dubbo缺省协议采用单一长连接和NIO异步通讯，适合于小数据量大并发的服务调用，以及服务消费者机器数远大于服务提供者机器数的情况。
- Dubbo缺省协议不适合传送大数据量的服务，比如传文件，传视频等，除非请求量很低

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

```
<dubbo:protocol name=“dubbo” port=“9090” server=“netty” client=“netty” codec=“dubbo” serialization=“hessian2”
charset=“UTF-8” threadpool=“fixed” threads=“100” queues=“0” iothreads=“9” buffer=“8192” accepts=“1000”
payload=“8388608” />
```

dubbo://



- ❖ Transporter
  - mina, netty, grizzy
- ❖ Serialization
  - dubbo, hessian2, java, json
- ❖ Dispatcher
  - all, direct, message, execution, connection
- ❖ ThreadPool
  - fixed, cached

dubbo://

- Dubbo协议缺省每服务每提供者每消费者使用单一长连接，如果数据量较大，可以使用多个连接。

```
<dubbo:protocol name="dubbo" connections="2" />
```

<dubbo:service connections=”0”>或<dubbo:reference connections=”0”>表示该服务使用JVM共享长连接。(缺省)
<dubbo:service connections=”1”>或<dubbo:reference connections=”1”>表示该服务使用独立长连接。
<dubbo:service connections=”2”>或<dubbo:reference connections=”2”>表示该服务使用独立两条长连接。

- 为防止被大量连接撑挂，可在服务提供方限制大接收连接数，以实现服务提供方自我保护。

```
<dubbo:protocol name="dubbo" accepts="1000" />
```

dubbo://

缺省协议，使用基于mina1.1.7+hessian3.2.1的tbremoting交互。
连接个数：单连接
连接方式：长连接
传输协议：TCP
传输方式：NIO异步传输
序列化：Hessian二进制序列化
适用范围：传入传出参数数据包较小（建议小于100K），消费者比提供者个数多，单一消费者无法压满提供者，尽量不要用dubbo协议传输大文件或超大字符串。
适用场景：常规远程服务方法调用

结束

百知教育
Baizhi Education