**Dubbo**

## 一、Dubbo概述

**1 Dubbo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。

**2 Dubbo是Alibaba开源的分布式服务框架**，它最大的特点是按照分层的方式来架构，使用这种方式可以使各个层之间解耦合（或者最大限度地松耦合）。从服务模型的角度来看，Dubbo采用的是一种非常简单的模型，要么是提供方提供服务，要么是消费方消费服务，所以基于这一点可以抽象出服务提供方（Provider）和服务消费方（Consumer）两个角色。

**3 核心部分**

**@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。

**4 电商系统的演变**

**（1）单一应用框架(ORM)**

当网站流量很小时，只需一个应用，将所有功能如下单支付等都部署在一起，以减少部署节点和成本。缺点：单一的系统架构，使得在开发过程中，占用的资源越来越多，而且随着流量的增加越来越难以维护

**（2）集群应用框架(MVC)**

业务水平拆分，集群应用架构解决了单一应用架构所面临的扩容问题，流量能够分散到各个机器中。

**（3）分布式应用架构(RPC)**

业务垂直拆分成多个应用，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

**（4）流动计算架构(SOA)**

随着服务化的进一步发展，服务越来越多，服务之间的调用和依赖关系也越来越复杂，诞生了面向服务的架构体系(SOA)，也因此衍生出了一系列相应的技术，如对服务提供、服务调用、连接处理、通信协议、序列化方式、服务发现、服务路由、日志输出等行为进行封装的服务框架

## Dubbo架构

**1 组成结构**

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: 统计服务的调用次数和调用时间的监控中心。

**2 调用流程**

0.服务容器负责启动，加载，运行服务提供者。

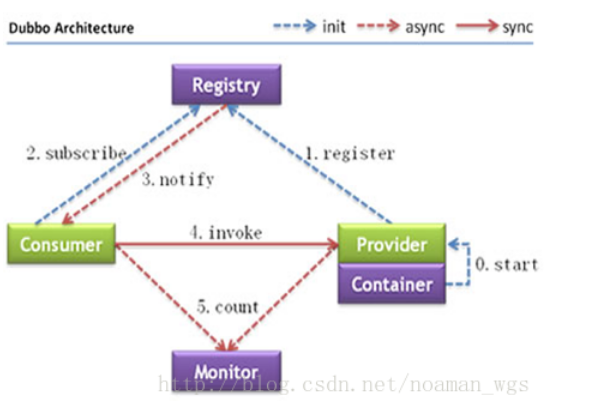
1.服务提供者在启动时，向注册中心注册自己提供的服务。

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 Dubbo注册中心**

对于服务提供方，需要发布服务，由于应用系统的复杂性，服务的数量、类型也不断膨胀；

对于服务消费方，如何获取到它所需要的服务，面对复杂的应用系统需要管理大量服务调用。

而且，对于服务提供方和服务消费方来说，他们还有可能兼具这两种角色，即既需要提供服务，有需要消费服务。

通过将服务统一管理起来，可以有效地优化内部应用对服务发布/使用的流程和管理。服务注册中心可以通过特定协议来完成服务对外的统一。

Dubbo提供的注册中心有如下几种类型可供选择：

**Multicast注册中心**

**Zookeeper注册中心**

**Redis注册中心**

**Simple注册中心**

**4 Dubbo优缺点**

优点：透明化的远程方法调用；像调用本地方法一样调用远程方法，没有任何API侵入；

软负载均衡及容错机制；可在内网替代nginx lvs等硬件负载均衡器；服务注册中心自动注册 & 配置管理;不需要写死服务提供者地址，注册中心基于接口名自动查询提供者ip；使用类似zookeeper等分布式协调服务作为服务注册中心，可以将绝大部分项目配置移入zookeeper集群。服务接口监控与治理Dubbo-admin与Dubbo-monitor提供了完善的服务接口管理与监控功能，针对不同应用的不同接口，可以进行 多版本，多协议，多注册中心管理。

缺点：只支持JAVA语言

**5 dubbo 注解配置**

@Service //Service注解暴露服务

@Configuration // javaconfig形式配置公共模块

@DubboComponentScan // 指定dubbo扫描路径

@Reference //注解引用

**6 启动时检查服务可用性**

<dubbo:reference check //设定特定服务启动检查

<dubbo:consumer check //设定所有服务启动检查

<dubbo:registry check //设定注册中心启动检查

**7 集群容错：**

<dubbo:service cluster="failsafe" />

<dubbo:reference cluster="failsafe" />

failover：失败自动切换，重试其它服务器

failfast：快速失败，只发起一次调用，失败立即报错，非幂等操作

failsafe：失败安全，忽略异常，

failback：失败自动恢复，后台记录失败请求，定时重发，

forking：同时调用多个服务器，只要一个返回成功即可，应用于实时性高的操作

broadcast：广播所有服务器，逐个调用，任意一台报错，则报错，服务更新提供者缓存应用

**8 负载均衡：**

服务端 服务端方法级 客户端 客户端方法级

random：按权重设置随机概率，碰撞概率

roundrobin：公约后的权重设置轮训概率，慢提供者积累请求问题。

leastactive：慢的提供者收到更少的请求

consistenthash：相同参数请求发送到同一提供者，

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" />

**9 线程派发策略：**

all：所有消息都派发到线程池

direct：所有消息不派发线程池，直接在io线程操作

message：只请求响应派发到线程池

execution：只请求消息派发线程池

connection：除连接断开消息，其它派发到线程池

**10 线程池：**

fixed：固定大小

cached：空闲一分钟删除，需要重建

limited：可伸缩，只增长不减少，应对大流量

直连：<dubbo:reference id="xxxService" interface="com.alibaba.xxx.XxxService" url="dubbo://localhost:20890" />

## **11十层架构**

**（1）服务接口层（Service）：**该层是与实际业务逻辑相关的，根据服务提供方和服务消费方的业务设计对应的接口和实现。

**（2）配置层（Config）：**对外配置接口，以ServiceConfig和ReferenceConfig为中心，可以直接new配置类，也可以通过spring解析配置生成配置类。

**（3）服务代理层（Proxy）：**服务接口透明代理，生成服务的客户端Stub和服务器端Skeleton，以ServiceProxy为中心，扩展接口为ProxyFactory。

**（4）服务注册层（Registry）：**封装服务地址的注册与发现，以服务URL为中心，扩展接口为RegistryFactory、Registry和RegistryService。可能没有服务注册中心，此时服务提供方直接暴露服务。

**（5）集群层（Cluster）：**封装多个提供者的路由及负载均衡，并桥接注册中心，以Invoker为中心，扩展接口为Cluster、Directory、Router和LoadBalance。将多个服务提供方组合为一个服务提供方，实现对服务消费方来透明。

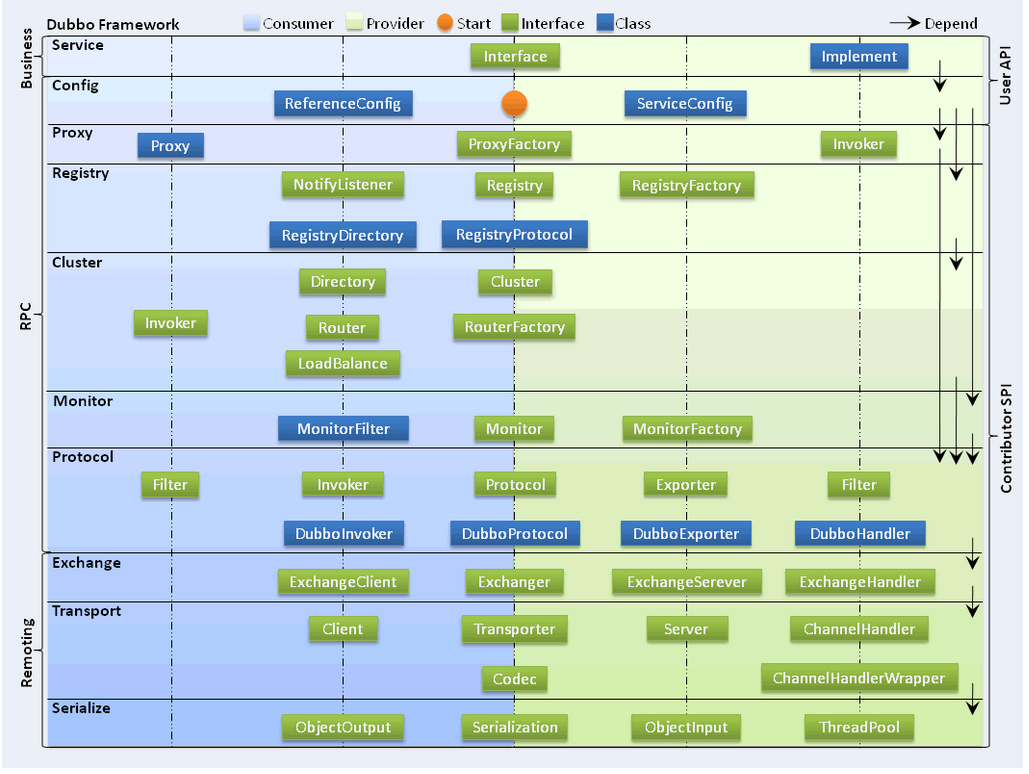
**（6）监控层（Monitor）**：RPC调用次数和调用时间监控，以Statistics为中心，扩展接口为MonitorFactory、Monitor和MonitorService。

**（7）远程调用层（Protocol）：**封将RPC调用，以Invocation和Result为中心，扩展接口为Protocol、Invoker和Exporter。Protocol是服务域，它是Invoker暴露和引用的主功能入口，它负责Invoker的生命周期管理。Invoker是实体域，它是Dubbo的核心模型，其它模型都向它靠扰，或转换成它，它代表一个可执行体，可向它发起invoke调用，它有可能是一个本地的实现，也可能是一个远程的实现，也可能一个集群实现。

**（8）信息交换层（Exchange）**：封装请求响应模式，同步转异步，以Request和Response**为中心，扩展接口为Exchanger、ExchangeChannel、ExchangeClient和ExchangeServer。**

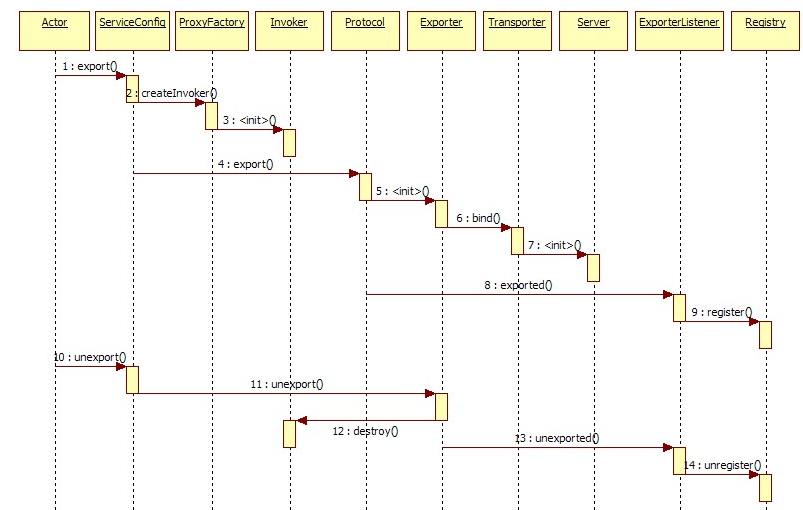
**（9）网络传输层（Transport）：**抽象mina和netty为统一接口，以Message为中心，扩展接口为Channel、Transporter、Client、Server和Codec。

**（10）数据序列化层（Serialize）：**可复用的一些工具，扩展接口为Serialization、 ObjectInput、ObjectOutput和ThreadPool。



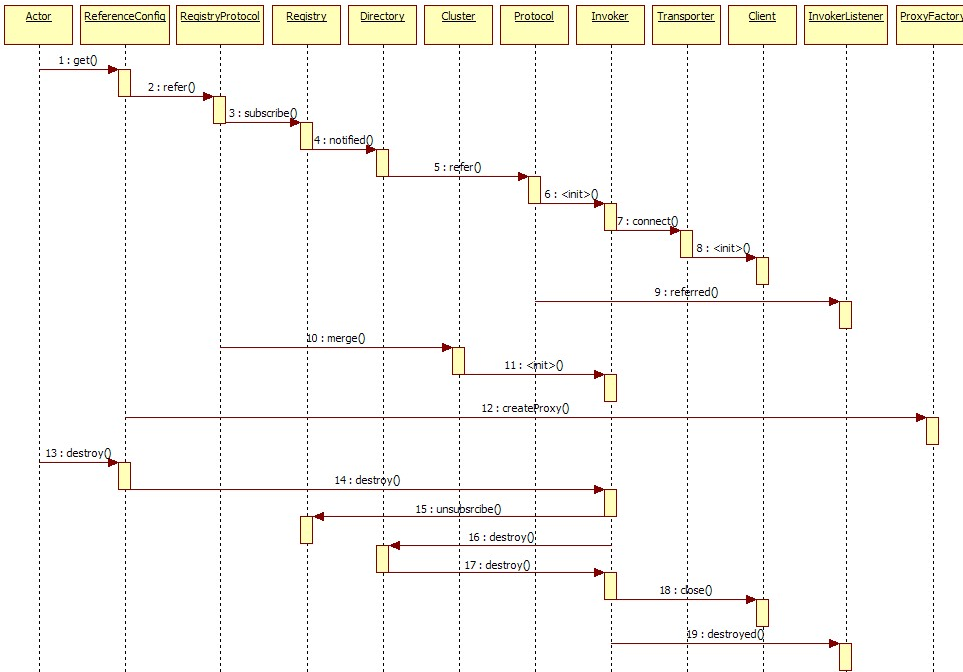
**12 注册/注销服务**

服务的注册与注销，是对服务提供方角色而言，那么注册服务与注销服务的时序图，如图所示



**13 服务订阅/取消**

为了满足应用系统的需求，服务消费方的可能需要从服务注册中心订阅指定的有服务提供方发布的服务，在得到通知可以使用服务时，就可以直接调用服务。反过来，如果不需要某一个服务了，可以取消该服务。下面看一下对应的时序图，如图所示：



**14 协议支持**

Dubbo支持多种协议,在通信过程中，不同的服务等级一般对应着不同的服务质量，那么选择合适的协议便是一件非常重要的事情。你可以根据你应用的创建来选择。

例如，使用RMI协议，一般会受到防火墙的限制，所以对于外部与内部进行通信的场景就不要使用RMI协议，而是基于HTTP协议或者Hessian协议。

**Dubbo协议**

**Hessian协议**

**HTTP协议**

**RMI协议**

**WebService协议**

**Thrift协议**

**Memcached协议**

**Redis协议**

**Rest协议**

**（1）dubbo 协议 (默认)**

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

1、dubbo默认采用dubbo协议，dubbo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

2、他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:provider protocol="dubbo" />

<dubbo:service protocol="dubbo" />

<dubbo:protocol id="dubbo1" name="dubbo" port="20880" />

<dubbo:protocol id="dubbo2" name="dubbo" port="20881" />.

<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” />

3、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。

4、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" />

5、为什么要消费者比提供者个数多？

因dubbo协议采用单一长连接，假设网络为千兆网卡(1024Mbit=128MByte)，根据测试经验数据每条连接最多只能压满7MByte(不同的环境可能不一样，供参考)，理论上1个服务提供者需要20个服务消费者才能压满网卡

6、为什么不能传大包？

因dubbo协议采用单一长连接，如果每次请求的数据包大小为500KByte，假设网络为千兆网卡(1024Mbit=128MByte)，每条连接最大7MByte(不同的环境可能不一样，供参考)，单个服务提供者的TPS(每秒处理事务数)最大为：128MByte / 500KByte = 262。单个消费者调用单个服务提供者的TPS(每秒处理事务数)最大为：7MByte / 500KByte = 14。如果能接受，可以考虑使用，否则网络将成为瓶颈。

7、为什么采用异步单一长连接？

因为服务的现状大都是服务提供者少，通常只有几台机器，而服务的消费者多，可能整个网站都在访问该服务，比如Morgan的提供者只有6台提供者，却有上百台消费者，每天有1.5亿次调用，如果采用常规的hessian服务，服务提供者很容易就被压跨，通过单一连接，保证单一消费者不会压死提供者，长连接，减少连接握手验证等，并使用异步IO，复用线程池，防止C10K问题。

8、接口增加方法，对客户端无影响，如果该方法不是客户端需要的，客户端不需要重新部署；输入参数和结果集中增加属性，对客户端无影响，如果客户端并不需要新属性，不用重新部署；

输入参数和结果集属性名变化，对客户端序列化无影响，但是如果客户端不重新部署，不管输入还是输出，属性名变化的属性值是获取不到的。

9、总结：服务器端 和 客户端 对 领域对象 并不需要完全一致，而是按照最大匹配原则。

**（2）、rmi 协议**

Java标准的远程调用协议。

连接个数：多连接

连接方式：短连接

传输协议：TCP

传输方式：同步传输

序列化：Java标准二进制序列化

适用范围：传入传出参数数据包大小混合，消费者与提供者个数差不多，可传文件。

适用场景：常规远程服务方法调用，与原生RMI服务互操作

1、RMI协议采用JDK标准的java.rmi.\*实现，采用阻塞式短连接和JDK标准序列化方式

如果正在使用RMI提供服务给外部访问（公司内网环境应该不会有攻击风险），同时应用里依赖了老的common-collections包（dubbo不会依赖这个包，请排查自己的应用有没有使用）的情况下，

存在反序列化安全风险。请检查应用：将commons-collections3 请升级到3.2.2版本：

将commons-collections4 请升级到4.1版本，新版本的commons-collections解决了该问题

2、如果服务接口继承了java.rmi.Remote接口，可以和原生RMI互操作，即：

提供者用Dubbo的RMI协议暴露服务，消费者直接用标准RMI接口调用，或者提供方用标准RMI暴露服务，消费方用Dubbo的RMI协议调用。

3、如果服务接口没有继承java.rmi.Remote接口，缺省Dubbo将自动生成一个com.xxx.XxxService$Remote的接口，并继承java.rmi.Remote接口，并以此接口暴露服务，

4、设置了<dubbo:protocol name="rmi" codec="spring" />，将不生成$Remote接口，而使用Spring的RmiInvocationHandler接口暴露服务，和Spring兼容。

<dubbo:protocol name="rmi" port="1099" />.

<dubbo:provider protocol="rmi" />

<dubbo:service protocol="rmi" />

<dubbo:protocol id="rmi1" name="rmi" port="1099" />

<dubbo:protocol id="rmi2" name="rmi" port="2099" />

<dubbo:service protocol="rmi1" />

<dubbo:protocol name="rmi" codec="spring" />

**（3）、hessian 协议**

基于Hessian的远程调用协议。

连接个数：多连接

连接方式：短连接

传输协议：HTTP

传输方式：同步传输

序列化：表单序列化

适用范围：传入传出参数数据包大小混合，提供者比消费者个数多，可用浏览器查看，可用表单或URL传入参数，暂不支持传文件。

适用场景：需同时给应用程序和浏览器JS使用的服务。

1、Hessian协议用于集成Hessian的服务，Hessian底层采用Http通讯，采用Servlet暴露服务，Dubbo缺省内嵌Jetty作为服务器实现。

2、Hessian是Caucho开源的一个RPC框架：http://hessian.caucho.com，其通讯效率高于WebService和Java自带的序列化。

依赖:

<dependency>

<groupId>com.caucho</groupId>

<artifactId>hessian</artifactId>

<version>4.0.7</version>

</dependency>

可以和原生Hessian服务互操作，即：

提供者用Dubbo的Hessian协议暴露服务，消费者直接用标准Hessian接口调用，或者提供方用标准Hessian暴露服务，消费方用Dubbo的Hessian协议调用。

参数及返回值需实现Serializable接口

参数及返回值不能自定义实现List, Map, Number, Date, Calendar等接口，只能用JDK自带的实现，因为hessian会做特殊处理，自定义实现类中的属性值都会丢失。

<dubbo:protocol name="hessian" port="8080" server="jetty" />

<dubbo:provider protocol="hessian" />

<dubbo:service protocol="hessian" />

<dubbo:protocol id="hessian1" name="hessian" port="8080" />

<dubbo:protocol id="hessian2" name="hessian" port="8081" />

<dubbo:reference id="helloService" interface="HelloWorld" url="hessian://10.20.153.10:8080/helloWorld" />

<dubbo:protocol ... server="jetty" />

<dubbo:protocol ... server="servlet" />

3、web.xml配置:

<servlet>

<servlet-name>dubbo</servlet-name>

<servlet-class>com.alibaba.dubbo.remoting.http.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>dubbo</servlet-name>

<url-pattern>/\*</url-pattern>

</servlet-mapping>

4、注意：如果使用servlet派发请求

协议的端口必须与servlet容器的端口相同，

协议的上下文路径必须与servlet应用的上下文路径相同。

**（4）、http 协议**

基于http表单的远程调用协议。参见：[HTTP协议使用说明]

连接个数：多连接

连接方式：短连接

传输协议：HTTP

传输方式：同步传输

序列化：表单序列化

适用范围：传入传出参数数据包大小混合，提供者比消费者个数多，可用浏览器查看，可用表单或URL传入参数，暂不支持传文件。

适用场景：需同时给应用程序和浏览器JS使用的服务。

1、采用Spring的 HttpInvoker 实现

<dubbo:protocol name="http" port="8080" />

<!-- Jetty Server -->

<dubbo:protocol ... server="jetty" />

<!-- Servlet Bridge Server -->

<dubbo:protocol ... server="servlet" />

2、web.xml配置

<servlet>

<servlet-name>dubbo</servlet-name>

<servlet-class>com.alibaba.dubbo.remoting.http.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>dubbo</servlet-name>

<url-pattern>/\*</url-pattern>

</servlet-mapping>

3、注意:如果使用 servlet 派发请求

协议的端口<dubbo:protocol port="8080" />必须与servlet容器的端口相同，

协议的上下文路径<dubbo:protocol contextpath="foo" />必须与servlet应用的上下文路径相同。

**（5）、webservice 协议**

基于WebService的远程调用协议

连接个数：多连接

连接方式：短连接

传输协议：HTTP

传输方式：同步传输

序列化：SOAP文本序列化

适用场景：系统集成，跨语言调用

1、基于CXF的 frontend-simple 和 transports-http 实现。

2、CXF是Apache开源的一个RPC框架：http://cxf.apache.org，由Xfire和Celtix合并而来 。

<dependency>

<groupId>org.apache.cxf</groupId>

<artifactId>cxf-rt-frontend-simple</artifactId>

<version>2.6.1</version>

</dependency>

<dependency>

<groupId>org.apache.cxf</groupId>

<artifactId>cxf-rt-transports-http</artifactId>

<version>2.6.1</version>

</dependency>

可以和原生WebService服务互操作，即：

提供者用Dubbo的WebService协议暴露服务，消费者直接用标准WebService接口调用，或者提供方用标准WebService暴露服务，消费方用Dubbo的WebService协议调用。

参数及返回值需实现Serializable接口

参数尽量使用基本类型和POJO。

<dubbo:protocol name="webservice" port="8080" server="jetty" />

<dubbo:provider protocol="webservice" />

<dubbo:service protocol="webservice" />

<dubbo:protocol id="webservice1" name="webservice" port="8080" />

<dubbo:protocol id="webservice2" name="webservice" port="8081" />

<dubbo:reference id="helloService" interface="HelloWorld" url="webservice://10.20.153.10:8080/com.foo.HelloWorld" />

http://10.20.153.10:8080/com.foo.HelloWorld?wsdl

<dubbo:protocol ... server="jetty" />

<dubbo:protocol ... server="servlet" />

web.xml

<servlet>

<servlet-name>dubbo</servlet-name>

<servlet-class>com.alibaba.dubbo.remoting.http.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>dubbo</servlet-name>

<url-pattern>/\*</url-pattern>

</servlet-mapping>

3、注意:如果使用servlet派发请求：

协议的端口必须与servlet容器的端口相同，

协议的上下文路径必须与servlet应用的上下文路径相同。

**（6）、thrift 协议**

当前 dubbo 支持的 thrift 协议是对 thrift 原生协议的扩展，在原生协议的基础上添加了一些额外的头信息，比如service name，magic number等。使用dubbo thrift协议同样需要使用thrift的idl compiler编译生成相应的java代码，后续版本中会在这方面做一些增强

依赖

<dependency>

<groupId>org.apache.thrift</groupId>

<artifactId>libthrift</artifactId>

<version>0.8.0</version>

</dependency>

配置:

<dubbo:protocol name="thrift" port="3030" />

Thrift不支持null值，不能在协议中传null

**（7）、memcached 协议**

可以通过脚本或监控中心手工填写表单注册memcached服务的地址：

RegistryFactory registryFactory = ExtensionLoader.getExtensionLoader(RegistryFactory.class).getAdaptiveExtension();

Registry registry = registryFactory.getRegistry(URL.valueOf("zookeeper://10.20.153.10:2181"));

registry.register(URL.valueOf("memcached://10.20.153.11/com.foo.BarService?category=providers&dynamic=false&application=foo&group=member&loadbalance=consistenthash"));

然后在客户端使用时，不需要感知Memcached的地址：

<dubbo:reference id="cache"

interface="http://10.20.160.198/wiki/display/dubbo/java.util.Map" group="member" />

或者点对点直连

<dubbo:reference id="cache"

interface="http://10.20.160.198/wiki/display/dubbo/java.util.Map"

url="memcached://10.20.153.10:11211" />

自定义接口

<dubbo:reference id="cache"

interface="com.foo.CacheService" url="memcached://10.20.153.10:11211" />

方法名建议和memcached的标准方法名相同，即：get(key), set(key, value), delete(key)。

如果方法名和memcached的标准方法名不相同，则需要配置映射关系：(其中”p:xxx”为spring的标准p标签)

<dubbo:reference id="cache" interface="com.foo.CacheService"

url="memcached://10.20.153.10:11211" p:set="putFoo"

p:get="getFoo" p:delete="removeFoo" />

**（8）、redis 协议**

可以通过脚本或监控中心手工填写表单注册redis服务的地址：

RegistryFactory registryFactory = ExtensionLoader.getExtensionLoader(RegistryFactory.class).getAdaptiveExtension();

Registry registry = registryFactory.getRegistry(URL.valueOf("zookeeper://10.20.153.10:2181"));

registry.register(URL.valueOf("redis://10.20.153.11/com.foo.BarService?category=providers&dynamic=false&application=foo&group=member&loadbalance=consistenthash"));

然后在客户端使用时，不需要感知Redis的地址：

<dubbo:reference id="store"

interface="http://10.20.160.198/wiki/display/dubbo/java.util.Map" group="member" />

点对点直连

<dubbo:reference id="store"

interface="http://10.20.160.198/wiki/display/dubbo/java.util.Map"

url="redis://10.20.153.10:6379" />

自定义接口

<dubbo:reference id="store" interface="com.foo.StoreService" url="redis://10.20.153.10:6379" />

方法名建议和redis的标准方法名相同，即：get(key), set(key, value), delete(key)。

如果方法名和redis的标准方法名不相同，则需要配置映射关系：(其中”p:xxx”为spring的标准p标签)

<dubbo:reference id="cache" interface="com.foo.CacheService"

url="memcached://10.20.153.10:11211"

p:set="putFoo" p:get="getFoo" p:delete="removeFoo" />

**（9） Rest协议**

<!-- use netty server -->

<!--<dubbo:protocol name="rest" port="8888" keepalive="true" server="netty" iothreads="5" threads="100" contextpath="services"/>-->

<!-- use tjws server -->

<!--<dubbo:protocol name="rest" port="8888" server="tjws" contextpath="services"/>-->

<!-- use tomcat server -->

<dubbo:protocol name="rest" port="8888" threads="500" contextpath="services" server="tomcat" accepts="500"

extension="com.alibaba.dubbo.demo.extension.TraceInterceptor,

com.alibaba.dubbo.demo.extension.TraceFilter,

com.alibaba.dubbo.demo.extension.ClientTraceFilter,

com.alibaba.dubbo.demo.extension.DynamicTraceBinding,

com.alibaba.dubbo.demo.extension.CustomExceptionMapper,

com.alibaba.dubbo.rpc.protocol.rest.support.LoggingFilter"/>

**15 Maven dubbo-modules:**

dubbo-common 公共逻辑模块，包括Util类和通用模型。

dubbo-remoting 远程通讯模块，相当于Dubbo协议的实现，如果RPC用RMI协议则不需要使用此包。

dubbo-rpc 远程调用模块，抽象各种协议，以及动态代理，只包含一对一的调用，不关心集群的管理。

dubbo-cluster 集群模块，将多个服务提供方伪装为一个提供方，包括：负载均衡、容错、路由等，集群的地址列表可以是静态配置的，也可以是由注册中心下发。

dubbo-registry 注册中心模块，基于注册中心下发地址的集群方式，以及对各种注册中心的抽象。

dubbo-monitor 监控模块，统计服务调用次数，调用时间的，调用链跟踪的服务。

dubbo-config 配置模块，是Dubbo对外的API，用户通过Config使用Dubbo，隐藏Dubbo所有细节。

dubbo-container 容器模块，是一个Standalone的容器，以简单的Main加载Spring启动，因为服务通常不需要Tomcat/JBoss等Web容器的特性，没必要用Web容器去加载服务。

**16 暴露服务：**

a、 直接暴露服务端口

在没有使用注册中心的情况，这种情况一般适用在开发环境下，服务的调用这和提供在同一个IP上，只需要打开服务的端口即可。 即，当配置 or ServiceConfig解析出的URL的格式为： Dubbo：//service-host/com.xxx.TxxService?version=1.0.0 基于扩展点的Adaptiver机制，通过URL的“dubbo：//”协议头识别，直接调用DubboProtocol的export（）方法，打开服务端口。

b、向注册中心暴露服务：

和上一种的区别：需要将服务的IP和端口一同暴露给注册中心。

ServiceConfig解析出的url格式为：

registry://registry-host/com.alibaba.dubbo.registry.RegistryService?export=URL.encode(“dubbo://service-host/com.xxx.TxxService?version=1.0.0”)

基于扩展点的Adaptive机制，通过URL的“registry：//”协议头识别，调用RegistryProtocol的export方法，将export参数中的提供者URL先注册到注册中心，再重新传给Protocol扩展点进行暴露： Dubbo：//service-host/com.xxx.TxxService?version=1.0.0

**18 引用服务：**

a、直接引用服务：

在没有注册中心的，直连提供者情况下， ReferenceConfig解析出的URL格式为： Dubbo：//service-host/com.xxx.TxxService?version=1.0.0

基于扩展点的Adaptive机制，通过url的“dubbo：//”协议头识别，直接调用DubboProtocol的refer方法，返回提供者引用。

b、从注册中心发现引用服务：

此时，ReferenceConfig解析出的URL的格式为： registry://registry-host/com.alibaba.dubbo.registry.RegistryService?refer=URL.encode(“consumer://consumer-host/com.foo.FooService?version=1.0.0”)

基于扩展点的Apaptive机制，通过URL的“registry：//”协议头识别，就会调用RegistryProtocol的refer方法，基于refer参数总的条件，查询提供者URL，如： Dubbo：//service-host/com.xxx.TxxService?version=1.0.0

基于扩展点的Adaptive机制，通过提供者URL的“dubbo：//”协议头识别，就会调用DubboProtocol的refer（）方法，得到提供者引用。 然后RegistryProtocol将多个提供者引用，通过Cluster扩展点，伪装成单个提供这引用返回。

**19 暴露服务主过程：**

首先ServiceConfig类拿到对外提供服务的实际类ref，然后将ProxyFactory类的getInvoker方法使用ref生成一个AbstractProxyInvoker实例，到这一步就完成具体服务到invoker的转化。接下来就是Invoker转换到Exporter的过程。 Dubbo处理服务暴露的关键就在Invoker转换到Exporter的过程，下面我们以Dubbo和rmi这两种典型协议的实现来进行说明： Dubbo的实现： Dubbo协议的Invoker转为Exporter发生在DubboProtocol类的export方法，它主要是打开socket侦听服务，并接收客户端发来的各种请求，通讯细节由dubbo自己实现。 Rmi的实现： RMI协议的Invoker转为Exporter发生在RmiProtocol类的export方法，他通过Spring或Dubbo或JDK来实现服务，通讯细节由JDK底层来实现。

**20 服务消费的主过程：**

首先ReferenceConfig类的init方法调用Protocol的refer方法生成Invoker实例。接下来把Invoker转为客户端需要的接口

## 入门实例

**1 操作步骤**

1）. 安装Zookeeper,启动；

2）. 创建MAVEN项目，构建Dubbo+Zookeeper+Spring实现的简单Demo；

3）. 安装Dubbo-admin，实现监控;

**2 Zookeeper介绍与安装**

本Demo中的Dubbo注册中心采用的是Zookeeper。为什么采用Zookeeper呢？

Zookeeper是一个分布式的服务框架，是树型的目录服务的数据存储，能做到集群管理数据 ，这里能很好的作为Dubbo服务的注册中心。Dubbo能与Zookeeper做到集群部署，当提供者出现断电等异常停机时，Zookeeper注册中心能自动删除提供者信息，当提供者重启时，能自动恢复注册数据，以及订阅请求安装完成后，进入到bin目录，并且启动zkServer.cmd，这个脚本中会启动一个java进程：

(注：需要先启动zookeeper后，后续dubbo demo代码运行才能使用zookeeper注册中心的功能)

**3 创建MAVEN项目**

项目结构：

主要分三大模块：

dubbo-api : 存放公共接口；

dubbo-consumer :　调用远程服务；

dubbo-provider : 提供远程服务。

**4 下面将详细叙述代码构建过程。**

1) 首先构建MAVEN项目，导入所需要的jar包依赖。

需要导入的有spring, dubbo, zookeeper等jar包。

(详情参看后面提供的项目代码)

2)创建dubbo-api的MAVEN项目(有独立的pom.xml，用来打包供提供者消费者使用)。

在项目中定义服务接口：该接口需单独打包，在服务提供方和消费方共享。

package com.alibaba.dubbo.demo;

import java.util.List;

public interface DemoService {

List<String> getPermissions(Long id);

}

3)创建dubbo-provider的MAVEN项目(有独立的pom.xml，用来打包供消费者使用)。

实现公共接口，此实现对消费者隐藏：

package com.alibaba.dubbo.demo.impl;

import com.alibaba.dubbo.demo.DemoService;

import java.util.ArrayList;

import java.util.List;

public class DemoServiceImpl implements DemoService {

public List<String> getPermissions(Long id) {

List<String> demo = new ArrayList<String>();

demo.add(String.format("Permission\_%d", id - 1));

demo.add(String.format("Permission\_%d", id));

demo.add(String.format("Permission\_%d", id + 1));

return demo;

}

}

需加入公共接口所在的依赖

用Spring配置声明暴露服务

<?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/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://code.alibabatech.com/schema/dubbo

http://code.alibabatech.com/schema/dubbo/dubbo.xsd">

<!--定义了提供方应用信息，用于计算依赖关系；在 dubbo-admin 或 dubbo-monitor 会显示这个名字，方便辨识-->

<dubbo:application name="demotest-provider" owner="programmer" organization="dubbox"/>

<!--使用 zookeeper 注册中心暴露服务，注意要先开启 zookeeper-->

<dubbo:registry address="zookeeper://localhost:2181"/>

<!-- 用dubbo协议在20880端口暴露服务 -->

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

<!--使用 dubbo 协议实现定义好的 api.PermissionService 接口-->

<dubbo:service interface="com.alibaba.dubbo.demo.DemoService" ref="demoService" protocol="dubbo" />

<!--具体实现该接口的 bean-->

<bean id="demoService" class="com.alibaba.dubbo.demo.impl.DemoServiceImpl"/>

</beans>

启动远程服务：

package com.alibaba.dubbo.demo.impl;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.io.IOException;

public class Provider {

public static void main(String[] args) throws IOException {

ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("provider.xml");

System.out.println(context.getDisplayName() + ": here");

context.start();

System.out.println("服务已经启动...");

System.in.read();

}

}

4)创建dubbo-consumer的MAVEN项目(可以有多个consumer，但是需要配置好)。

调用所需要的远程服务：

通过Spring配置引用远程服务：

<?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/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="demotest-consumer" owner="programmer" organization="dubbox"/>

<!--向 zookeeper 订阅 provider 的地址，由 zookeeper 定时推送-->

<dubbo:registry address="zookeeper://localhost:2181"/>

<!--使用 dubbo 协议调用定义好的 api.PermissionService 接口-->

<dubbo:reference id="permissionService" interface="com.alibaba.dubbo.demo.DemoService"/>

</beans>

启动Consumer,调用远程服务：

package com.alibaba.dubbo.consumer;

import com.alibaba.dubbo.demo.DemoService;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Consumer {

public static void main(String[] args) {

//测试常规服务

ClassPathXmlApplicationContext context =

new ClassPathXmlApplicationContext("consumer.xml");

context.start();

System.out.println("consumer start");

DemoService demoService = context.getBean(DemoService.class);

System.out.println("consumer");

System.out.println(demoService.getPermissions(1L));

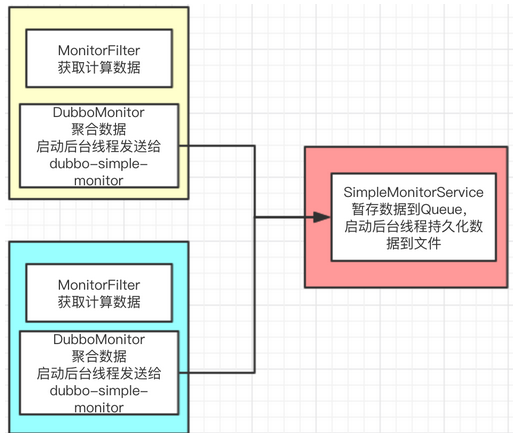
}

}

5）运行项目，先确保provider已被运行后再启动consumer模块：

运行提供者：消费者成功调用提供者所提供的远程服务：

## 四、[dubbo-monitor计数监控](https://www.cnblogs.com/java-zhao/p/8279053.html)



**1 MonitorFilter向DubboMonitor发送数据**

DubboMonitor将数据进行聚合后（默认聚合1min中的统计数据）暂存到ConcurrentMap<Statistics, AtomicReference<long[]>> statisticsMap，然后使用一个含有3个线程（线程名字：DubboMonitorSendTimer）的线程池每隔1min钟，调用SimpleMonitorService遍历发送statisticsMap中的统计数据，每发送完毕一个，就重置当前的Statistics的AtomicReference<long[]>

SimpleMonitorService将这些聚合数据塞入BlockingQueue<URL> queue中（队列大写为100000）

SimpleMonitorService使用一个后台线程（线程名为：DubboMonitorAsyncWriteLogThread）将queue中的数据写入文件（该线程以死循环的形式来写）

SimpleMonitorService还会使用一个含有1个线程（线程名字：DubboMonitorTimer）的线程池每隔5min钟，将文件中的统计数据画成图表

**2 SimpleMonitorService理解为一个服务提供者；**

**provider和consumer都是一个服务消费者，所以二者的DubboMonitor中的MonitorService实例都是一个代理实例**。dubbo-monitor计数监控不支持异步调用下的数据监控

**3 dubbo-monitor使用**

在配置文件中添加：

<dubbo:monitor address="10.211.55.5:9090" />

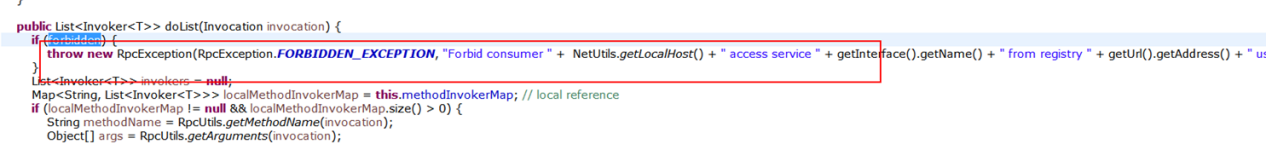
4 问题解析

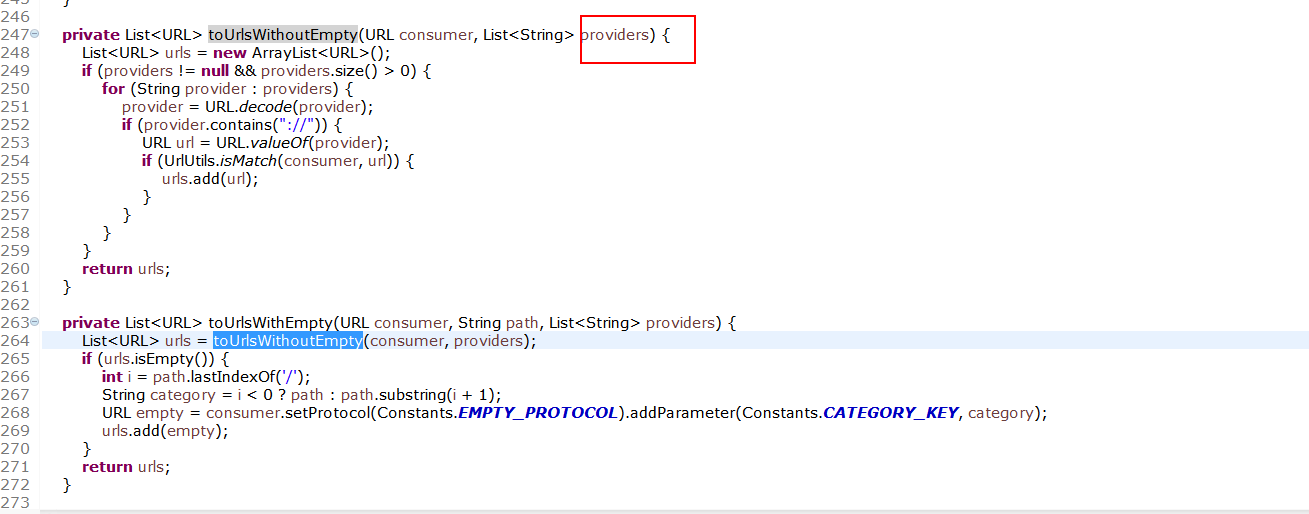
1. 问题报错

com.alibaba.dubbo.rpc.RpcException: Forbid consumer 10.10.50.156 access service com.alibaba.dubbo.monitor.MonitorService from registry 10.10.50.134:2181 use dubbo version 2.8.4, Please check registry access list (whitelist/blacklist).  
        at com.alibaba.dubbo.registry.integration.RegistryDirectory.doList(RegistryDirectory.java:579)  
        at com.alibaba.dubbo.rpc.cluster.directory.AbstractDirectory.list(AbstractDirectory.java:73)  
        at com.alibaba.dubbo.rpc.cluster.support.AbstractClusterInvoker.list(AbstractClusterInvoker.java:260)  
        at com.alibaba.dubbo.rpc.cluster.support.AbstractClusterInvoker.invoke(AbstractClusterInvoker.java:219)  
        at com.alibaba.dubbo.rpc.cluster.support.wrapper.MockClusterInvoker.invoke(MockClusterInvoker.java:72)  
        at com.alibaba.dubbo.rpc.proxy.InvokerInvocationHandler.invoke(InvokerInvocationHandler.java:52)  
        at com.alibaba.dubbo.common.bytecode.proxy1.collect(proxy1.java)  
        at com.alibaba.dubbo.monitor.dubbo.DubboMonitor.send(DubboMonitor.java:113)  
        at com.alibaba.dubbo.monitor.dubbo.DubboMonitor$1.run(DubboMonitor.java:70)  
        at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)  
        at java.util.concurrent.FutureTask.runAndReset(FutureTask.java:308)  
        at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.access$301(ScheduledThreadPoolExecutor.java:180)  
        at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run(ScheduledThreadPoolExecutor.java:294)  
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)  
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)  
        at java.lang.Thread.run(Thread.java:748)

1. 原因分析

没有服务提供者造成，没有一个项目注册了监控中心。







1. 解决措施

任意一台使用dubbo的项目配置

<dubbo:monitor address="10.211.55.5:9090" />

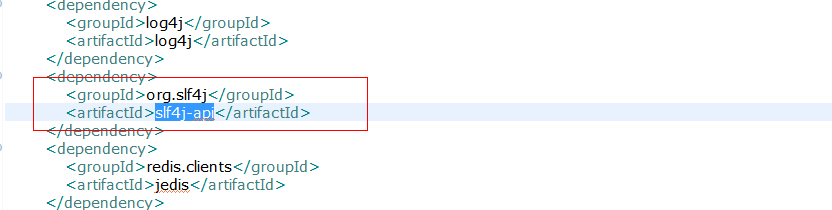


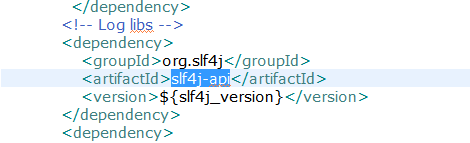
## 五、dubbo-demo工程

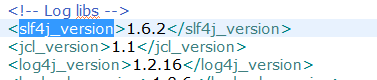
**1 dependencyManagement 和 dependencies**

Maven 使用dependencyManagement 元素来提供了一种管理依赖版本号的方式。通常会在一个组织或者项目的最顶层的父POM 中看到dependencyManagement 元素。

使用pom.xml 中的dependencyManagement 元素能让所有在子项目中引用一个依赖而不用显式的列出版本号。Maven 会沿着父子层次向上走，直到找到一个拥有dependencyManagement 元素的项目，然后它就会使用在这个dependencyManagement 元素中指定的版本号。dependencyManagement里只是声明依赖，并不实现引入，因此子项目需要显式的声明需要用的依赖。

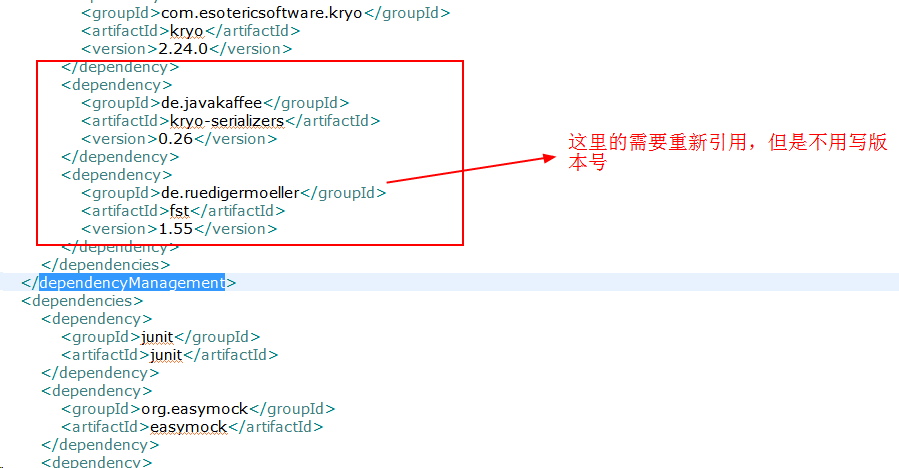






dependencies相对于dependencyManagement，所有声明在dependencies里的依赖都会自动引入，并默认被所有的子项目继承。





# **2 [Maven中plugins和pluginManagement的区别](https://www.cnblogs.com/EasonJim/p/6845012.html" \t "https://www.cnblogs.com/EasonJim/p/_blank)**

pluginManagement是表示插件声明，即你在项目中的pluginManagement下声明了插件，Maven不会加载该插件，pluginManagement声明可以被继承。

pluginManagement一般是用来在父POM中定义，提供给子POM使用，子POM也可以覆盖这个定义，而且你在父POM中定义了版本之后，子模块中直接应用groupId和artifactId，而不用指定版本，同时也方便统一管理；而在父POM中的pluginManagement并不会介入到Maven的生命周期。

plugins就是直接引入一个plugin，而且可以绑定到Maven相关的生命周期上。

pluginManagement主要是为了统一管理插件，确保所有子POM使用的插件版本保持一致，类似dependencies和dependencyManagement。

**但是在父pom中，如果使用这个标签来包裹plugins插件，当在此项目根目录运行对应的mvn命令时，如果在子pom中没有直接像上面再次引用这个plugin，那么不会触发这个plugin插件，只有在子pom中再次引用了之后，才会在对应的子项目路径下触发这个plugin.**

**3 Maven的几个常用plugin**

**maven-compiler-plugin编译Java源码，一般只需设置编译的jdk版本**

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.6.0</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

**maven-dependency-plugin用于复制依赖的jar包到指定的文件夹里**

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-dependency-plugin</artifactId>

<version>2.10</version>

<executions>

<execution>

<id>copy-dependencies</id>

<phase>package</phase>

<goals>

<goal>copy-dependencies</goal>

</goals>

<configuration>

<outputDirectory>${project.build.directory}/lib</outputDirectory>

</configuration>

</execution>

</executions>

</plugin>

**maven-jar-plugin打成jar时，设定manifest的参数，比如指定运行的Main class，还有依赖的jar包，加入classpath中**

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-jar-plugin</artifactId>

<version>2.4</version>

<configuration>

<archive>

<manifest>

<addClasspath>true</addClasspath>

<classpathPrefix>/data/lib</classpathPrefix>

<mainClass>com.zhang.spring.App</mainClass>

</manifest>

</archive>

</configuration>

</plugin>

**maven-antrun-plugin在maven中运行Ant任务，比如在打包阶段，对文件进行复制**

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-antrun-plugin</artifactId>

<version>1.7</version>

<executions>

<execution>

<phase>package</phase>

<goals>

<goal>run</goal>

</goals>

<configuration>

<target name="copy">

<delete>

<fileset dir="target" includes="\*.properties"></fileset>

</delete>

<copy todir="target">

<fileset dir="files"></fileset>

</copy>

</target>

</configuration>

</execution>

</executions>

</plugin>

**wagon-maven-plugin用于一键部署，把本地打包的jar文件，上传到远程服务器上，并执行服务器上的shell命令**

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>wagon-maven-plugin</artifactId>

<version>1.0</version>

<configuration>

<serverId>crawler</serverId>

<fromDir>target</fromDir>

<includes>\*.jar,\*.properties,\*.sh</includes>

<url>sftp://59.110.162.178/home/zhangxianhe</url>

<commands>

<command>chmod 755 /home/zhangxianhe/update.sh</command>

<command>/home/zhangxianhe/update.sh</command>

</commands>

<displayCommandOutputs>true</displayCommandOutputs>

</configuration>

</plugin>

tomcat7-maven-plugin用于远程部署Java Web项目

<plugin>

<groupId>org.apache.tomcat.maven</groupId>

<artifactId>tomcat7-maven-plugin</artifactId>

<version>2.2</version>

<configuration>

<url>http://59.110.162.178:8080/manager/text</url>

<username>linjinbin</username>

<password>linjinbin</password>

</configuration>

</plugin>

