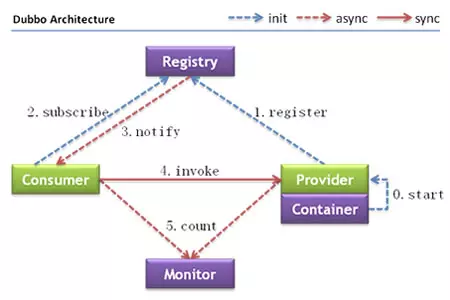
**Dubbo**

#### Dubbo原理

1. Dubbo架构图



1. Provider

Provider是服务提供者。相对于Consumer来说Provider是透明的，上一次调用服务的位置（IP地址）和下一次调用服务的位置，是不确定的。这个地方就需要使用注册中心来实现软负载。

1. Consumer

Consumer是服务消费者。消费是去invoke（同步）提供者。

1. Container

Container是服务容器。

1. Registry

Registry是服务注册中心。Provider首先启动服务并且register到注册中心，并且注册中心会notify（通知）Consumer。同时，Consumer会subscribe服务，如果没有订阅到，会不断尝试订阅。

1. Monitor

Monitor是一个监控，Consumer与Provider是通过异步的方式发送消息到Monitor的。

1. Dubbo的工作过程

（1）初始化

首先就是将服务装载进容器中，然后准备注册服务。和Spring类似，Spring启动时也是首先解析bean然后把bean装载进容器中。Dubbo也是首先读配置文件解析服务。

（2）解析服务

（a） 基于dubbo.jar内的Meta-inf/spring.handlers配置，spring在遇到dubbo名称空间时，会回调DubboNamespaceHandler类。

（b） 所有的dubbo标签，都统一用DubboBeanDefinitionParser进行解析，基于一对一属性映射，将XML标签解析为Bean对象。生产者或者消费者初始化的时候，会将Bean对象转会为url格式，将所有Bean属性转成url的参数。 然后将URL传给Protocol扩展点，基于扩展点的Adaptive机制，根据URL的协议头，进行不同协议的服务暴露和引用。

（3）暴露服务

（a）直接暴露服务端口

（b）向注册中心暴露服务

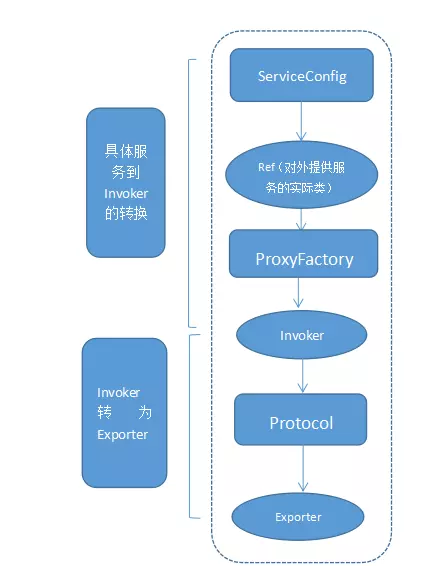
（4）引用服务

（a）直接引用服务

（b）从注册中心发现引用服务

3. 服务提供域消费过程

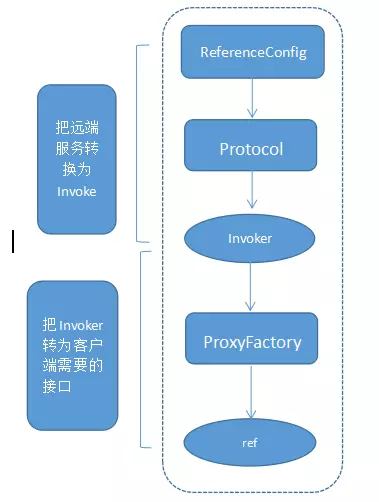
（1） 服务提供过程



暴露服务的主过程：

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。

（2） 服务消费过程



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

4. 底层通信机制

Dubbo里使用到了Socket（采用apache mina框架做底层调用）来建立长连接，发送、接收数据，底层使用apache mina框架的IoSession进行发送消息。

Dubbo底层使用Socket发送消息的形式进行数据传递，结合了mina框架，使用IoSession.write()方法，这个方法调用后对于整个远程调用(从发出请求到接收到结果)来说是一个异步的，即对于当前线程来说，将请求发送出来，线程就可以往后执行了，至于服务端的结果，是服务端处理完成后，再以消息的形式发送给客户端的。于是这里出现了2个问题：

（1）当前线程怎么让它“暂停”，等结果回来后，再向后执行？

（2）正如前面所说，Socket通信是一个全双工的方式，如果有多个线程同时进行远程方法调用，这时建立在client server之间的socket连接上会有很多双方发送的消息传递，前后顺序也可能是乱七八糟的，server处理完结果后，将结果消息发送给client，client收到很多消息，怎么知道哪个消息结果是原先哪个线程调用的？

基本原理如下：

* client一个线程调用远程接口，生成一个唯一的ID（比如一段随机字符串，UUID等），Dubbo是使用AtomicLong从0开始累计数字的
* 将打包的方法调用信息（如调用的接口名称，方法名称，参数值列表等），和处理结果的回调对象callback，全部封装在一起，组成一个对象object
* 向专门存放调用信息的全局ConcurrentHashMap里面put(ID, object)
* 将ID和打包的方法调用信息封装成一对象connRequest，使用IoSession.write(connRequest)异步发送出去
* 当前线程再使用callback的get()方法试图获取远程返回的结果，在get()内部，则使用synchronized获取回调对象callback的锁， 再先检测是否已经获取到结果，如果没有，然后调用callback的wait()方法，释放callback上的锁，让当前线程处于等待状态。
* 服务端接收到请求并处理后，将结果（此结果中包含了前面的ID，即回传）发送给客户端，客户端socket连接上专门监听消息的线程收到消息，分析结果，取到ID，再从前面的ConcurrentHashMap里面get(ID)，从而找到callback，将方法调用结果设置到callback对象里。
* 监听线程接着使用synchronized获取回调对象callback的锁（因为前面调用过wait()，那个线程已释放callback的锁了），再notifyAll()，唤醒前面处于等待状态的线程继续执行（callback的get()方法继续执行就能拿到调用结果了），至此，整个过程结束。

需要注意的是，这里的callback对象是每次调用产生一个新的，不能共享，否则会有问题；另外ID必需至少保证在一个Socket连接里面是唯一的。

现在，前面两个问题已经有答案了，

（1）当前线程怎么让它“暂停”，等结果回来后，再向后执行？

答：先生成一个对象obj，在一个全局map里put(ID,obj)存放起来，再用synchronized获取obj锁，再调用obj.wait()让当前线程处于等待状态，然后另一消息监听线程等到服务端结果来了后，再map.get(ID)找到obj，再用synchronized获取obj锁，再调用obj.notifyAll()唤醒前面处于等待状态的线程。

（2）正如前面所说，Socket通信是一个全双工的方式，如果有多个线程同时进行远程方法调用，这时建立在client server之间的socket连接上会有很多双方发送的消息传递，前后顺序也可能是乱七八糟的，server处理完结果后，将结果消息发送给client，client收到很多消息，怎么知道哪个消息结果是原先哪个线程调用的？

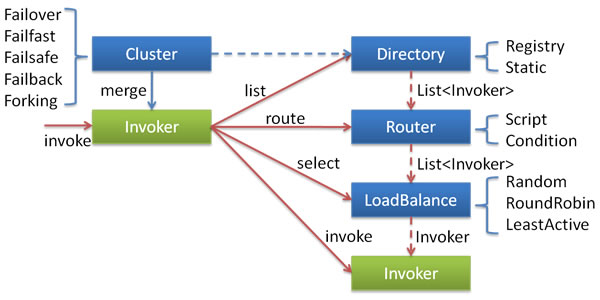
答：使用一个ID，让其唯一，然后传递给服务端，再服务端又回传回来，这样就知道结果是原先哪个线程的了。

#### Dubbo应用

https://dubbo.gitbooks.io/dubbo-user-book/

1. 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。



（1） 集群容错模式

· Failover Cluster

失败自动切换，当出现失败，重试其它服务器 [1](https://dubbo.gitbooks.io/dubbo-user-book/content/demos/fault-tolerent-strategy.html#fn_1)。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)。

重试次数配置如下：

<dubbo:service retries="2" />

或

<dubbo:reference retries="2" />

或

<dubbo:reference>

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

</dubbo:reference>

· 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](https://dubbo.gitbooks.io/dubbo-user-book/content/demos/fault-tolerent-strategy.html#fn_2)。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。

（2）集群模式配置

按照以下示例在服务提供方和消费方配置集群模式

<dubbo:service cluster="failsafe" />

或

<dubbo:reference cluster="failsafe" />

2. 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。可以自行扩展负载均衡策略

（1）负载均衡策略

· 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。

· 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的问题，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。

· 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。

· 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。

算法参见：<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" />

（2）配置

· 服务端服务级别

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

· 客户端服务级别

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

· 服务端方法级别

<dubbo:service interface="...">

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

</dubbo:service>

· 客户端方法级别

<dubbo:reference interface="...">

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

</dubbo:reference>

3. 多协议

* dubbo协议

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。

* 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

* 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。

* 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。

* 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 。

* thrift协议
* memcached协议
* redis协议

（1）不同服务不同协议

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>

（2）多协议暴露服务

需要与 http 客户端互操作

<?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>