让调用远程服务调用像调用本地方法那样简单。

一、Dubbo总结

### 1、概念

#### 1.1 什么是Dubbo

Apache Dubbo(incubating)是一款高性能、轻量级的开源Java RPC框架，它提供了三大核心能力：面向接口的远程方法调用，智能容错和负载均衡，以及服务自动注册和发现。简单来说 Dubbo 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。

Dubbo的Github 地址：<https://github.com/apache/incubator-dubbo>。Dubbo是由阿里开源，后来加入了Apache。正是由于Dubbo的出现，才使得越来越多的公司开始使用以及接受分布式架构。

#### 1.2 什么是RPC？RPC原理是什么？

RPC（Remote Procedure Call）—远程过程调用，它是一种通过网络从远程计算机程序上请求服务，而不需要了解底层网络技术的协议。比如两个不同的服务A、B部署在两台不同的机器上，那么服务A如果想要调用服务B中的某个方法该怎么办呢？使用HTTP请求当然可以，但是可能会比较慢而且一些优化做的并不好。RPC的出现就是为了解决这个问题。

RPC原理：



1. 服务消费方（client）调用以本地调用方式调用服务；
2. client stub接收到调用后负责将方法、参数等组装成能够进行网络传输的消息体；
3. client stub找到服务地址，并将消息发送到服务端；
4. server stub收到消息后进行解码；
5. server stub根据解码结果调用本地的服务；
6. 本地服务执行并将结果返回给server stub；
7. server stub将返回结果打包成消息并发送至消费方；
8. client stub接收到消息，并进行解码；
9. 服务消费方得到最终结果。

时序图：



#### 1.3 为什么要用Dubbo

Dubbo的诞生和SOA分布式架构的流行有着莫大的关系。SOA面向服务的架构（Service Oriented Architecture），也就是把工程按照业务逻辑拆分成服务层、表现层两个工程。服务层中包含业务逻辑，只需要对外提供服务即可。表现层只需要处理和页面的交互，业务逻辑都是调用服务层的服务来实现。SOA架构中有两个主要角色：服务提供者（Provider）和服务使用者（Consumer）。



#### 1.4 什么是分布式

分布式就是我们把整个系统拆分成不同的服务然后将这些服务放在不同的服务器上减轻单体服务的压力提高并发量和性能。比如电商系统可以简单地拆分成订单系统、商品系统、登录系统等等，拆分之后的每个服务可以部署在不同的机器上，如果某一个服务的访问量比较大的话也可以将这个服务同时部署在多台机器上。

### 2、Dubbo的架构

#### 2.1 Dubbo架构图解



上述节点简单说明：

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：统计服务的调用次数和调用时间的监控中心

Container：服务运行容器

调用关系说明：

1. 服务容器负责启动，加载，运行服务提供者。
2. 服务提供者在启动时，向注册中心注册自己提供的服务。
3. 服务消费者在启动时，向注册中心订阅自己所需的服务。
4. 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。
5. 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。
6. 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。

重要知识点总结：

* 注册中心负责服务地址的注册与查找，相当于目录服务，服务提供者和消费者只在启动时与注册中心交互，注册中心不转发请求，压力较小
* 监控中心负责统计各服务调用次数，调用时间等，统计先在内存汇总后每分钟一次发送到监控中心服务器，并以报表展示
* 注册中心，服务提供者，服务消费者三者之间均为长连接，监控中心除外
* 注册中心通过长连接感知服务提供者的存在，服务提供者宕机，注册中心将立即推送事件通知消费者
* 注册中心和监控中心全部宕机，不影响已运行的提供者和消费者，消费者在本地缓存了提供者列表
* 注册中心和监控中心都是可选的，服务消费者可以直连服务提供者
* 服务提供者无状态，任意一台宕掉后，不影响使用
* 服务提供者全部宕掉后，服务消费者应用将无法使用，并无限次重连等待服务提供者恢复

#### 2.2 Dubbo工作原理



图中从下至上分为十层，各层均为单向依赖，右边的黑色箭头代表层之间的依赖关系，每一层都可以剥离上层被复用，其中，Service和Config层为API，其他各层均为SPI。

层说明：

* 第一层：service层，接口层，给服务提供者和消费者来实现的
* 第二层：config层，配置层，主要是对dubbo进行各种配置的
* 第三层：proxy层，服务接口透明代理，生成服务的客户端 Stub 和服务器端 Skeleton
* 第四层：registry层，服务注册层，负责服务的注册与发现
* 第五层：cluster层，集群层，封装多个服务提供者的路由以及负载均衡，将多个实例组合成一个服务
* 第六层：monitor层，监控层，对rpc接口的调用次数和调用时间进行监控
* 第七层：protocol层，远程调用层，封装rpc调用
* 第八层：exchange层，信息交换层，封装请求响应模式，同步转异步
* 第九层：transport层，网络传输层，抽象mina和netty为统一接口
* 第十层：serialize层，数据序列化层，网络传输需要

### 3、Dubbo负载均衡策略

比如我们的系统中的某个服务的访问量特别大，我们将这个服务部署在了多台服务器上，当客户端发起请求的时候，多台服务器都可以处理这个请求。那么，如何正确选择处理该请求的服务器就很关键。假如，你就要一台服务器来处理该服务的请求，那该服务部署在多台服务器的意义就不复存在了。负载均衡就是为了避免单个服务器响应同一请求，容易造成服务器宕机、崩溃等问题。

Dubbo提供的负载均衡策略：

在集群负载均衡时，Dubbo提供了多种均衡策略，默认为random随机调用。可以自行扩展负载均衡策略，参见：[负载均衡扩展](https://dubbo.gitbooks.io/dubbo-dev-book/content/impls/load-balance.html)。

#### 3.1 Random LoadBalance（基于权重的随机负载均衡机制）

* 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。



#### 3.2 RoundRobin LoadBalance（不推荐，基于权重的轮询负载均衡机制)

* 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。



#### 3.3 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。

#### 3.4 ConsistentHash LoadBalance

* 一致性 Hash，相同参数的请求总是发到同一提供者。(如果你需要的不是随机负载均衡，是要一类请求都到一个节点，那就走这个一致性hash策略。)
* 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](http://en.wikipedia.org/wiki/Consistent_hashing" \t "_blank)
* 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" />

### 4、配置方式

#### 4.1 xml配置方式

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>

#### 4.2 注解配置方式

消费方基于基于注解的服务级别配置方式：

@Reference(loadbalance = "roundrobin")

HelloService helloService;

### 5、zookeeper宕机与dubbo直连的情况

zookeeper宕机与dubbo直连的情况在面试中可能会被经常问到，所以要引起重视。

在实际生产中，假如zookeeper注册中心宕掉，一段时间内服务消费方还是能够调用提供方的服务的，实际上它使用的本地缓存进行通讯，这只是dubbo健壮性的一种体现。

dubbo的健壮性表现：

1. 监控中心宕掉不影响使用，只是丢失部分采样数据
2. 数据库宕掉后，注册中心仍能通过缓存提供服务列表查询，但不能注册新服务
3. 注册中心对等集群，任意一台宕掉后，将自动切换到另一台
4. 注册中心全部宕掉后，服务提供者和服务消费者仍能通过本地缓存通讯
5. 服务提供者无状态，任意一台宕掉后，不影响使用
6. 服务提供者全部宕掉后，服务消费者应用将无法使用，并无限次重连等待服务提供者恢复

我们前面提到过：注册中心负责服务地址的注册与查找，相当于目录服务，服务提供者和消费者只在启动时与注册中心交互，注册中心不转发请求，压力较小。所以，我们可以完全可以绕过注册中心——采用 dubbo 直连 ，即在服务消费方配置服务提供方的位置信息。

xml配置方式：

<dubbo:reference id="userService"

interface="com.zang.gmall.service.UserService" url="dubbo://localhost:20880" />

注解方式：

@Reference(url = "127.0.0.1:20880")

HelloService helloService;

二、服务之间的调用为什么不直接用HTTP而用RPC

如果你要开发分布式程序，你也可以直接基于HTTP接口进行通信，但是为什么要用Dubbo呢？从Dubbo提供的下面四点特性来说为什么要用Dubbo：

1. 负载均衡——同一个服务部署在不同的机器时该调用那一台机器上的服务。
2. 服务调用链路生成——随着系统的发展，服务越来越多，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。Dubbo 可以为我们解决服务之间互相是如何调用的。
3. 服务访问压力以及时长统计、资源调度和治理——基于访问压力实时管理集群容量，提高集群利用率。
4. 服务降级——某个服务挂掉之后调用备用服务。

另外，Dubbo除了能够应用在分布式系统中，也可以应用在现在比较火的微服务系统中。不过，由于Spring Cloud在微服务中应用更加广泛，所以，我觉得一般我们提Dubbo的话，大部分是分布式系统的情况。

微服务是架构设计方式，分布式是系统部署方式，两者概念不同。两者的关系是，系统应用部署在超过一台服务器或虚拟机上，且各分开部署的部分彼此通过各种通讯协议交互信息，就可算作分布式部署，生产环境下的微服务肯定是分布式部署的，分布式部署的应用不一定是微服务架构的，比如集群部署，它是把相同应用复制到不同服务器上，但是逻辑功能上还是单体应用。

### 1、RPC解决了什么问题？

从上面对RPC介绍的内容中，概括来将RPC主要解决了：让分布式或者微服务系统中不同服务之间的调用像本地一样简单。

### 2、常见的RPC框架

* RMI（JDK自带）：JDK自带的RPC，有很多局限性，不推荐使用。
* Dubbo：Dubbo是阿里巴巴公司开源的一个高性能优秀的服务框架，使得应用可通过高性能的 RPC 实现服务的输出和输入功能，可以和 Spring框架无缝集成。目前 Dubbo 已经成为 Spring Cloud Alibaba 中的官方组件。
* gRPC：gRPC是可以在任何环境中运行的现代开源高性能RPC框架。它可以通过可插拔的支持来有效地连接数据中心内和跨数据中心的服务，以实现负载平衡，跟踪，运行状况检查和身份验证。它也适用于分布式计算的最后一英里，以将设备，移动应用程序和浏览器连接到后端服务。
* Hessian：Hessian是一个轻量级的remotingonhttp工具，使用简单的方法提供了RMI的功能。 相比WebService，Hessian更简单、快捷。采用的是二进制RPC协议，因为采用的是二进制协议，所以它很适合于发送二进制数据。
* Thrift：Apache Thrift是Facebook开源的跨语言的RPC通信框架，目前已经捐献给Apache基金会管理，由于其跨语言特性和出色的性能，在很多互联网公司得到应用，有能力的公司甚至会基于thrift研发一套分布式服务框架，增加诸如服务注册、服务发现等功能。

### 3、既有HTTP，为什么用RPC进行服务调用

RPC只是一种概念、一种设计，就是为了解决不同服务之间的调用问题，它一般会包含有传输协议和序列化协议这两个。

HTTP是一种协议，RPC框架可以使用HTTP协议作为传输或者直接使用TCP作为传输协议，使用不同的协议一般也是为了适应不同的场景。

计算机网络的五层协议的体系结构是指：应用层（HTTP）、传输层（TCP）、网络层、数据链路层、物理层。

应用层的任务是通过应用进程间的交互来完成特定网络应用。HTTP属于应用层协议，它会基于TCP/IP的通信协议来传递数据。HTTP协议工作于客户端-服务端架构之上。浏览器作为HTTP客户端通过URL向HTTP服务器端即WEB服务器发送所有请求。Web服务器根据接收到的请求后，向客户端发送响应信息。HTTP协议建立在TCP协议之上。

运输层的主要任务就是负责向两台主机进程之间的通信提供通用的数据传输服务。TCP是传输层协议，主要解决数据如何在网络中传输。相比于UDP，TCP提供的是面向连接的，可靠的数据传输服务。