Dubbo序列化和反序列化模块 需求与功能分析

曹瀚云

2018K8009907008

1. Dubbo简介

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。

下图1描述了Dubbo从单一应用架构，垂直应用架构，到分布式服务架构，再到流动计算架构的改变，是Dubbo官网对Dubbo架构发展演变的总结。

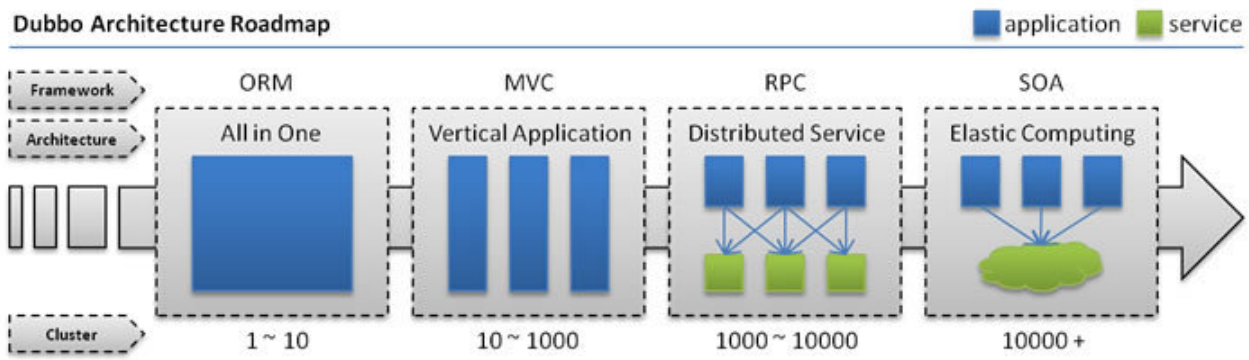

图 1. Dubbo架构发展演变

根据dubbo.Apache.org官方网站上的定义，Apache Dubbo™ 是一款微服务框架（Microservices Framework），它提供高性能 RPC 通信、服务发现、流量管理等服务治理能力，为你提供构建大规模微服务集群所需的全套解决方案。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。于是dubbo应运而生了。

Dubbo的应用场景可以简单理解如下：此前在网站开发时，整个系统处于一个服务器上，不同层次之间可以直接进行方法的调用。但面对目前的分布式系统，不同服务器之间的方法调用必须通过远程过程调用（Remote Procedure Call）来实现。将服务的提供方记为Provider，将调用远程服务的消费方记为Consumer，为了便于不同的提供方和消费方构建起调用的相互关系，设立一个注册中心，记为Register，负责记录管理生产者的服务注册以及消费者的服务调用。此外，为了统计服务的调用次数和调用时间，设立一个监控中心，记为Moniter，最后将服务的提供者放入一个服务运行容器中，记为Container。

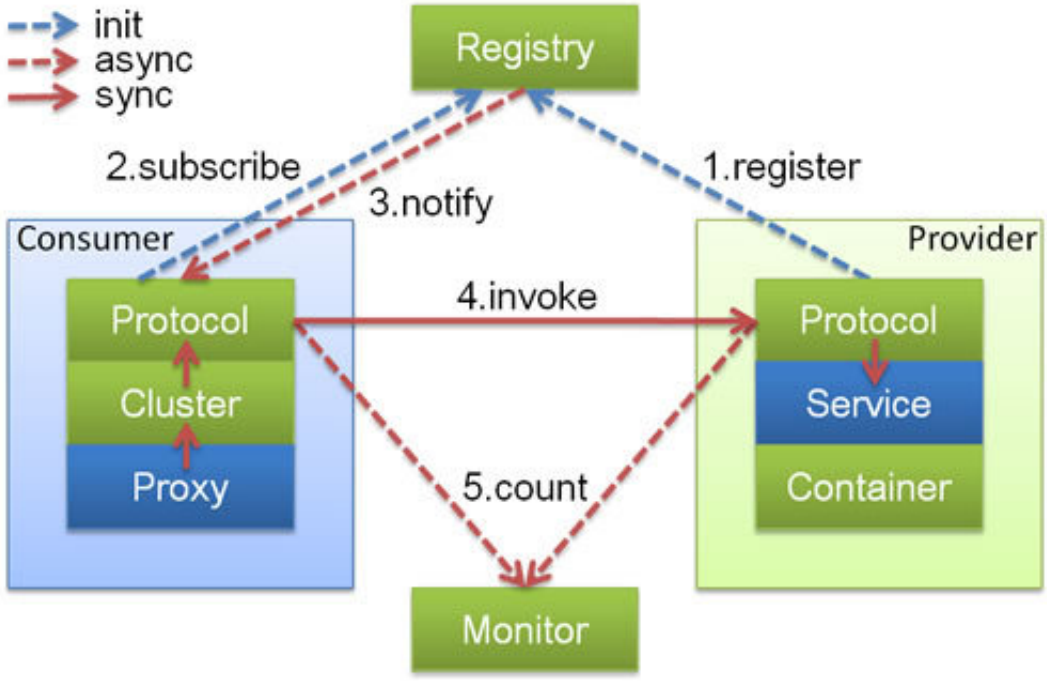

图 2. Dubbo的操作流程

按照上图数字顺序，在Dubbo下具体执行流程如下：

0. 服务容器负责启动，加载，运行服务提供者。

1. 服务提供者（生产者）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。

作为一款开源的远程服务调用的分布式框架，Dubbo 有三个主要优点：

1. 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。

2. 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。

3. 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。

2. Dubbo的设计框架

下面是dubbo.Apache.org官方网站的Dubbo设计架构图片。

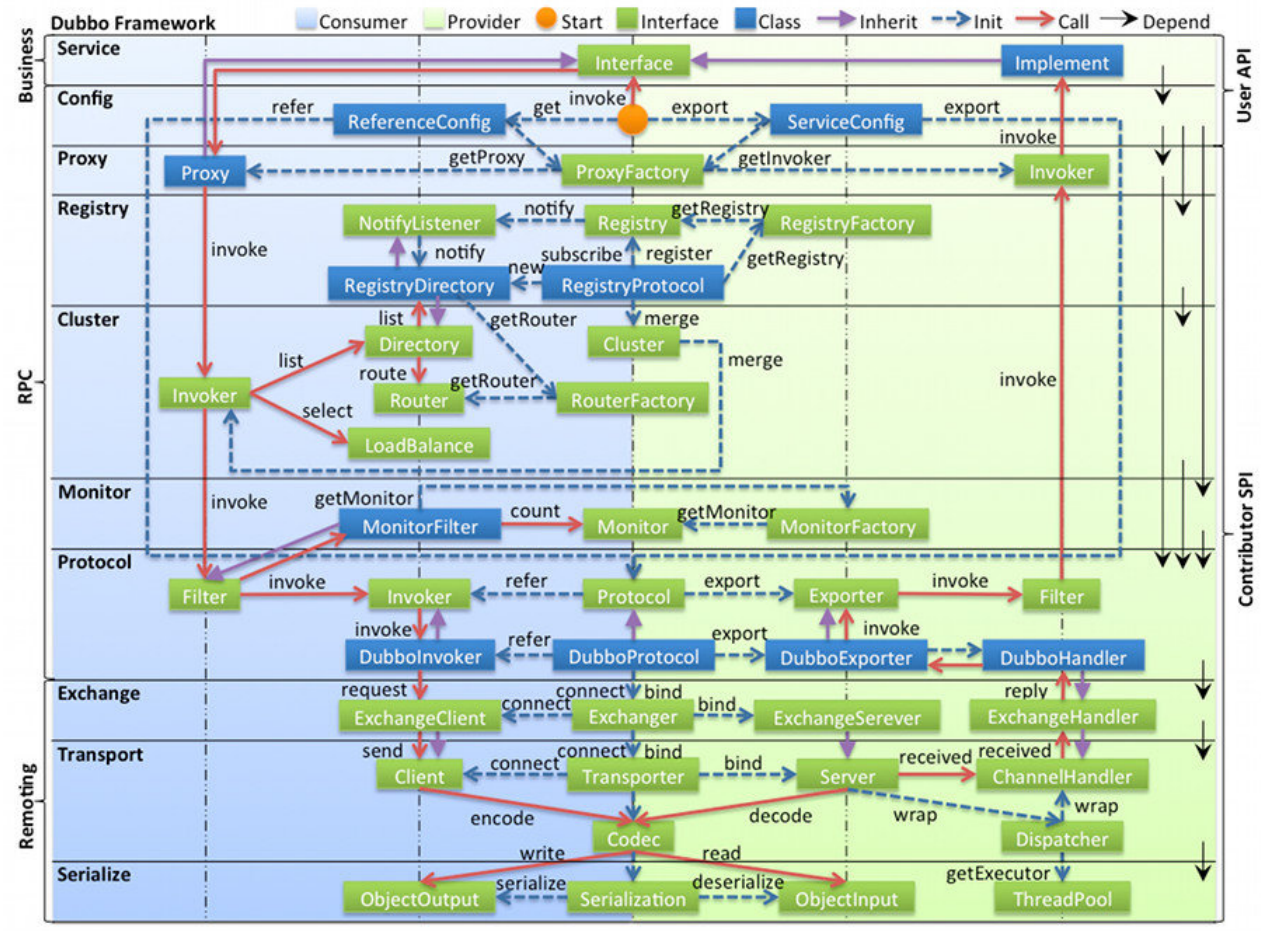

图 3. Dubbo设计架构图

层次结构分为三层： （1）**Business**：主要实现业务内容，也就是作为Dubbo用户的我们最经常见到的。其内部仅包含了Interface和其对应的实现Implement。

（2）**RPC**: 远程过程调用，内含config（配置层），proxy （服务代理层），registry（注册中心层），cluster（路由层），monitor（监控层）和protocol（远程调用层）共6层。

（3）**Remoting**：网络通信框架，内含exchange（信息交换层），transport（网络传输层）和serialize（数据序列化层）三层，其中serialize序列化层就是我此次开源项目源码分析的主要部分，它为网络通信数据的序列化和可序列化提供了一些可复用的工具，内含的扩展接口为 Serialization, ObjectInput, ObjectOutput, ThreadPool。

3. Dubbo中序列化/反序列化模块的作用

Dubbo要实现远程服务调用（RPC），其功能需要涵盖三个主要方面：（1）远程方法调用的Call ID映射；（2）序列化和反序列化；（3）网络传输层。因为网络传输的数据必须是二进制数据，而不同的服务器之间的可读取数据的格式不一定相同，所以在 RPC 调用中，需要客户端把参数先转成一个字节流，被称为序列化，传给服务端后，由服务端将调用过程的返回值转换成字节流传递给自己，最终客户端再把字节流转换成自己能读取的格式，也就是反序列化。对函数参数对象与返回值对象进行序列化与反序列化是RPC必须的一个过程。

4. Dubbo中序列化/反序列化模块粗析

在GitHub上的Dubbo仓库中，dubbo-serialization文件夹下含12个文件夹，其中一个提供api，一个为test文件夹，其余10个是不同的序列化/反序列化方法，如：FastJson，Kryo，Protobuff等（如下图展示了api和fastjson两个文件夹的目录结构，这里建议参考Maven教程）。我们主要分析各个模块的关系，暂时不去具体详述并比较这些知名的序列化/反序列化方法的具体实现。

图 4. Dubbo–serialization下目录层次结构示例

dubbo-serialization-api文件夹下共有9个Java文件，它们为Dubbo中的10种序列化/反序列化方法提供了许多接口和抽象类。SerializableClassRegistry抽象类中的registerclass方法调用了HashMap，用于提供注册表，被fst，kryo等部分序列化方法依赖。cleanable接口只被KryoObjectInput显式地实现了，为其中的cleanup方法提供了定义。其余的DataInput等接口均被10种序列化/反序列化方法实现，具体关系见下面的UML图（以FastJson为例，其余方法原理相同）。可以看出由ObjectInput继承DataInput，再由FastJsonObjectOutput实现，最后被FastJsonSerialization中的serialize方法使用这条关系线十分清晰，对于Output与Input类似，对应于deserialization。

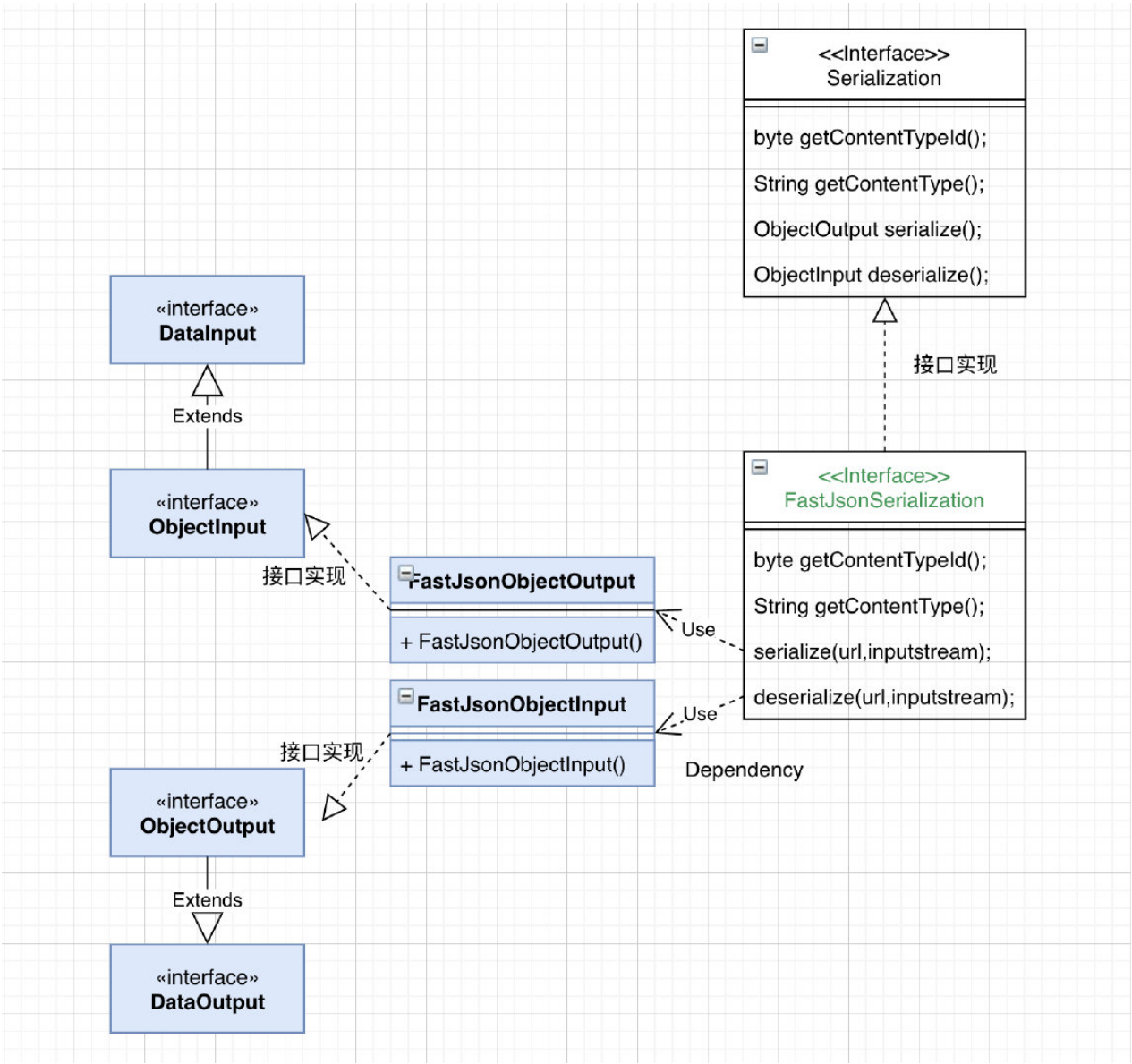

图 5. Dubbo序列化模块关系的UML图

5. 结语

以上就是我第一部分的报告了。在这个分析学习的过程中我投入了不少时间与精力，但也愈发觉得自己对Dubbo序列化/反序列化模块的理解仍有极大欠缺，我会继续研究，更好地完成之后部分的报告。