
RPC
原理
核心是代理机制
本地代理存根: Stub
本地序列化反序列化
网络通信
远程序列化反序列化
远程服务存根: Skeleton
调用实际业务服务
原路返回服务结果
返回给本地调用方
序列化
语言原生的序列化，RMI，Remoting
二进制平台无关，Hessian，avro，kyro，fst等
文本，JSON、XML等
网络传输
TCP/SSL
HTTP/HTTPS
查找实现类
通过接口查找服务端的实现类，一般通过注册方式
Dubbo
整体架构
config 配置层：对外配置接口，以 ServiceConfig, ReferenceConfig 为中心，可以直接初始化配置类，也可以通过 spring 解析配置生成配置类
proxy 服务代理层：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
registry 注册中心层：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, RegistryService
cluster 路由层：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, LoadBalance
monitor 监控层：RPC 调用次数和调用时间监控，以 Statistics 为中心，扩展接口为MonitorFactory, Monitor, MonitorService
protocol 远程调用层：封装 RPC 调用，以 Invocation, Result 为中心，扩展接口为Protocol, Invoker, Exporter
exchange 信息交换层：封装请求响应模式，同步转异步，以 Request, Response 为中心，扩展接口为 Exchanger, ExchangeChannel, ExchangeClient, ExchangeServer
transport 网络传输层：抽象 mina 和 netty 为统一接口，以 Message 为中心，扩展接口为Channel, Transporter, Client, Server, Codec
serialize 数据序列化层：可复用的一些工具，扩展接口为 Serialization, ObjectInput,ObjectOutput, ThreadPool
SPI扩展
ServiceLoader机制META-INF/dubbo/接口全限定名，文件内容为实现类
Dubbo的SPI扩展，最关键的SPI：Protocol
启动时装配，并缓存到ExtensionLoader中
服务暴露
InjvmProtocol
DubboProtocol
HessianProtocol
RmiProtocol
WebServiceProtocol
服务引用
ServiceReference，ReferenceConfig createProxy 中 创建 Invoker
集群
Router
选取此次调用可以提供服务的invoker集合
LoadBalance
从上述集合选取一个作为最终调用者，Random，RoundRobin
泛化调用
GenericService
当我们知道接口、方法和参数，不用存根方式，而是用反射方式调用任何服务
隐式传参
Context模式
RpcContext.getContext().setAttachment("index", "1");此参数可以传播到RPC调用的整个过程
Mock
使用场景
分布式服务化改造
开放平台
直接作为BFF给前端（Web或Mobile）提供服务（不建议使用）
最佳实践
开发分包
建议将服务接口、服务模型、服务异常等均放在 API 包中，因为服务模型和异常也是API 的一部分，这样做也符合分包原则：重用发布等价原则(REP)，共同重用原则(CRP)
服务接口尽可能大粒度，每个服务方法应代表一个功能，而不是某功能的一个步骤，否则将面临分布式事务问题，Dubbo 暂未提供分布式事务支持
服务接口建议以业务场景为单位划分，并对相近业务做抽象，防止接口数量爆炸
不建议使用过于抽象的通用接口，如：Map query(Map)，这样的接口没有明确语义，会给后期维护带来不便。
参数配置
默认规则
通用参数以 consumer 端为准，如果consumer端没有设置，使用provider数值
建议在 Provider 端配置的 Consumer 端属性
timeout：方法调用的超时时间
retries：失败重试次数，缺省是 2
loadbalance：负载均衡算法 3，缺省是随机 random
actives：消费者端的最大并发调用限制，即当 Consumer 对一个服务的并发调用到上限后，新调用会阻塞直到超时，可以配置在方法或服务上
建议在 Provider 端配置的 Provider 端属性
threads：服务线程池大小
executes：一个服务提供者并行执行请求上限，即当 Provider 对一个服务的并发调用达到上限后，新调用会阻塞，此时 Consumer 可能会超时。可以配置在方法或服务上。
运维与监控
Admin功能较简单，大规模使用需要定制开发，整合自己公司的运维监控系统
重试与幂等
服务调用失败默认重试2次，如果接口不是幂等的，会造成业务重复处理
幂等设计
去重
类似乐观锁机制
Spring Cloud
Config
Eureka
Feign
核心功能
Feign的核心功能就是，作为HTTP Client访问REST服务接口
优势
全都基于注解，简单方便
跟XXTemplate一样，内置了简化操作，OOP
跟其他组件，ribbon，hytrix联合使用
Ribbon
Ribbon是用于云环境的一个客户端内部通信（IPC）库
特性
负载均衡
容错
多协议支持(HTTP, TCP, UDP)，特别是异步和反应式下
缓存和批处理
Hytrix/Alibaba Sentinel