开源框架 (http://39.106.3.150/tags/#1560569459781)

Dubbo框架设计总结

Posted by Palmer on 06-15，2019

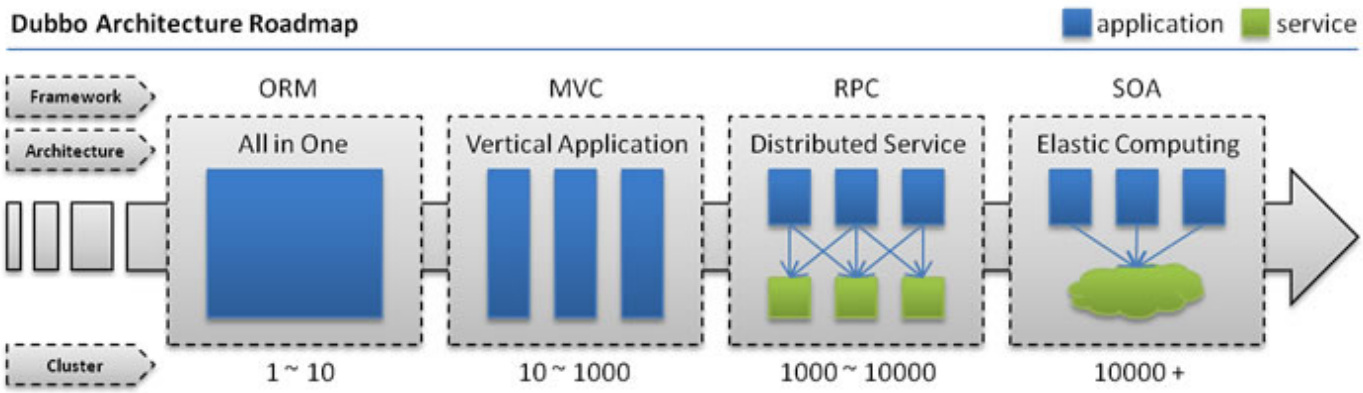
一、简述

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。

Dubbo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。

二、背景

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。



单一应用架构

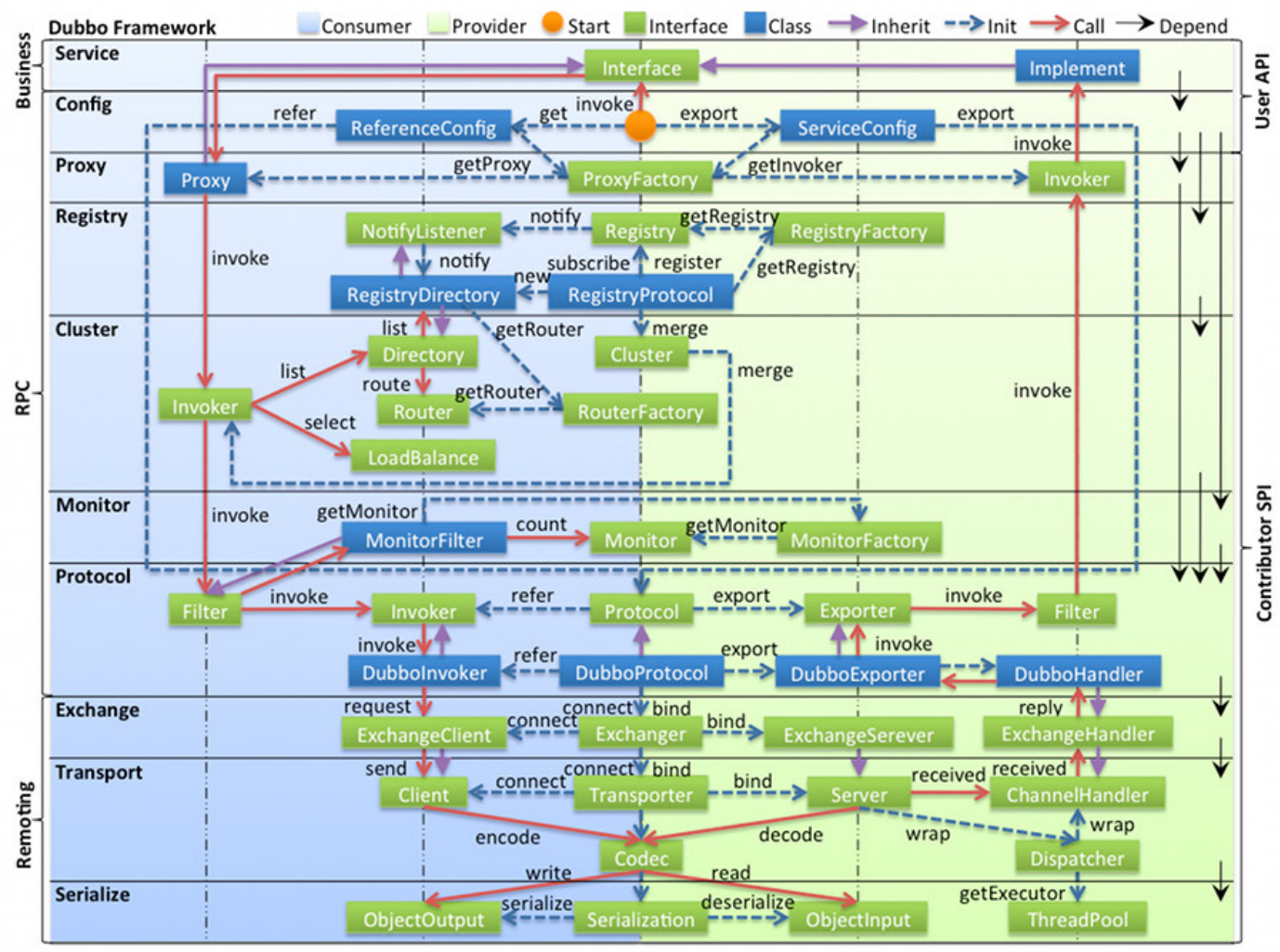
当网站流量很小时，只需一个应用，将所有功能都部署在一起，以减少部署节点和成本。此时，用于简化增删改查工作量的数据访问框架(ORM)是关键。

垂直应用架构 当访问量逐渐增大，单一应用增加机器带来的加速度越来越小，将应用拆成互不相干的几个应用，以提升效率。此时，用于加速前端页面开发的Web框架(MVC)是关键。

分布式服务架构 当垂直应用越来越多，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，使前端应用能更快速的响应多变的市场需求。此时，用于提高业务复用及整合的分布式服务框架(RPC)是关键。

流动计算架构 当服务越来越多，容量的评估，小服务资源的浪费等问题逐渐显现，此时需增加一个调度中心基于访问压力实时管理集群容量，提高集群利用率。此时，用于提高机器利用率的资源调度和治理中心(SOA)是关键。

三、框架设计



图例说明：

图中左边淡蓝背景的为服务消费方使用的接口，右边淡绿色背景的为服务提供方使用的接口，位于中轴线上的为双方都用到的接口。

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，其它各层均为 SPI。

图中绿色小块的为扩展接口，蓝色小块为实现类，图中只显示用于关联各层的实现类。

图中蓝色虚线为初始化过程，即启动时组装链，红色实线为方法调用过程，即运行时调时链，紫色三角箭头为继承，可以把子类看作父类的同一个节点，线上的文字为调用的方法。

各层说明

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

关系说明

在 RPC 中，Protocol 是核心层，也就是只要有 Protocol + Invoker + Exporter 就可以完成非透明的 RPC 调用，然后在 Invoker 的主过程上 Filter 拦截点。

图中的 Consumer 和 Provider 是抽象概念，只是想让看图者更直观的了解哪些类分属于客户端与服务器端，不用 Client 和 Server 的原因是 Dubbo 在很多场景下都使用 Provider, Consumer, Registry, Monitor 划分逻辑拓普节点，保持统一概念。

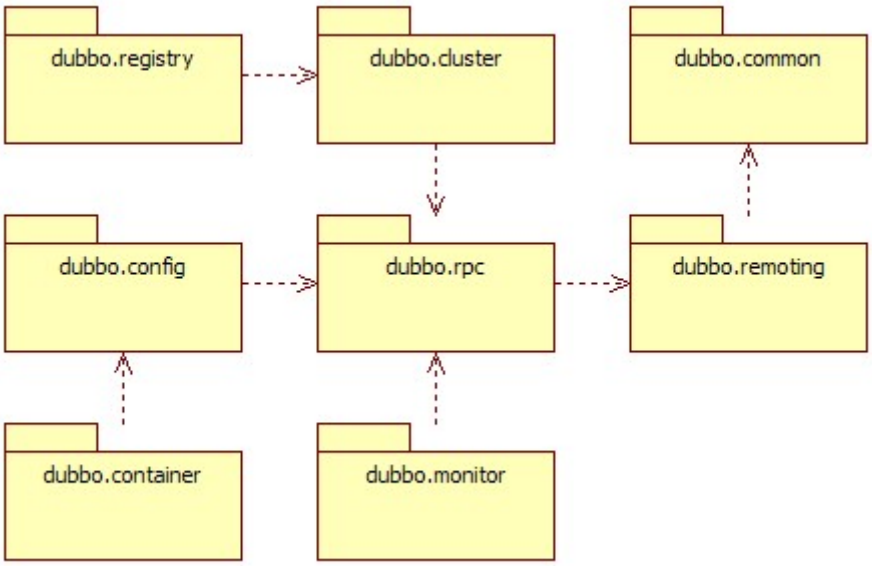
而 Cluster 是外围概念，所以 Cluster 的目的是将多个 Invoker 伪装成一个 Invoker，这样其它人只要关注 Protocol 层 Invoker 即可，加上 Cluster 或者去掉 Cluster 对其它层都不会造成影响，因为只有一个提供者时，是不需要 Cluster 的。

Proxy 层封装了所有接口的透明化代理，而在其它层都以 Invoker 为中心，只有到了暴露给用户使用时，才用 Proxy 将 Invoker 转成接口，或将接口实现转成 Invoker，也就是去掉 Proxy 层 RPC 是可以 Run 的，只是不那么透明，不那么看起来像调本地服务一样调远程服务。

而 Remoting 实现是 Dubbo 协议的实现，如果你选择 RMI 协议，整个 Remoting 都不会用上，Remoting 内部再划为 Transport 传输层和 Exchange 信息交换层，Transport 层只负责单向消息传输，是对 Mina, Netty, Grizzly 的抽象，它也可以扩展 UDP 传输，而 Exchange 层是在传输层之上封装了 Request-Response 语义。

Registry 和 Monitor 实际上不算一层，而是一个独立的节点，只是为了全局概览，用层的方式画在一起。

模块分包



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 Standlone 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。

整体上按照分层结构进行分包，与分层的不同点在于：

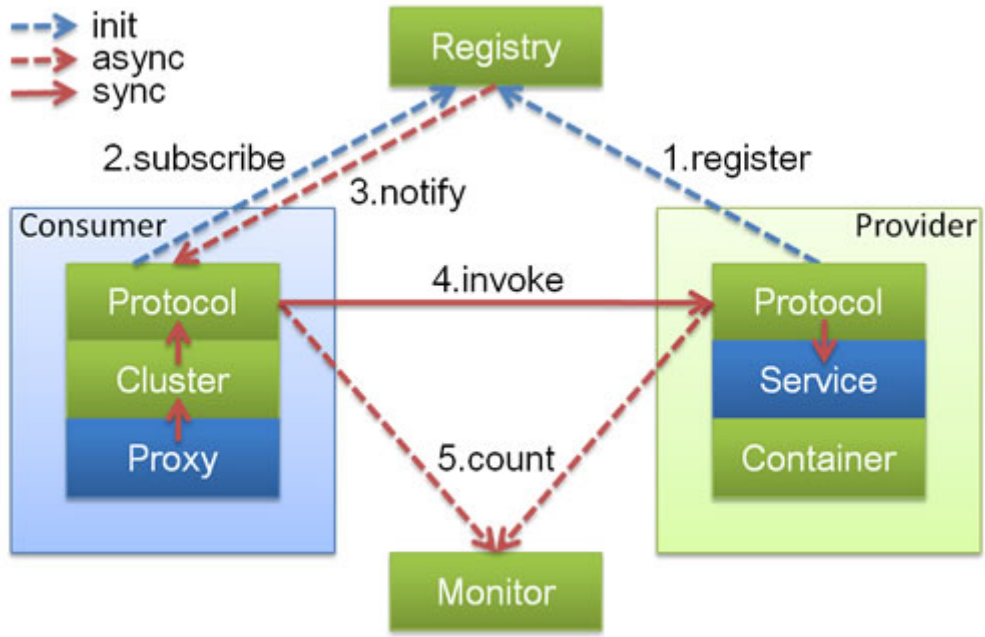
container 为服务容器，用于部署运行服务，没有在层中画出。

protocol 层和 proxy 层都放在 rpc 模块中，这两层是 rpc 的核心，在不需要集群也就是只有一个提供者时，可以只使用这两层完成 rpc 调用。

transport 层和 exchange 层都放在 remoting 模块中，为 rpc 调用的通讯基础。

serialize 层放在 common 模块中，以便更大程度复用。

依赖关系



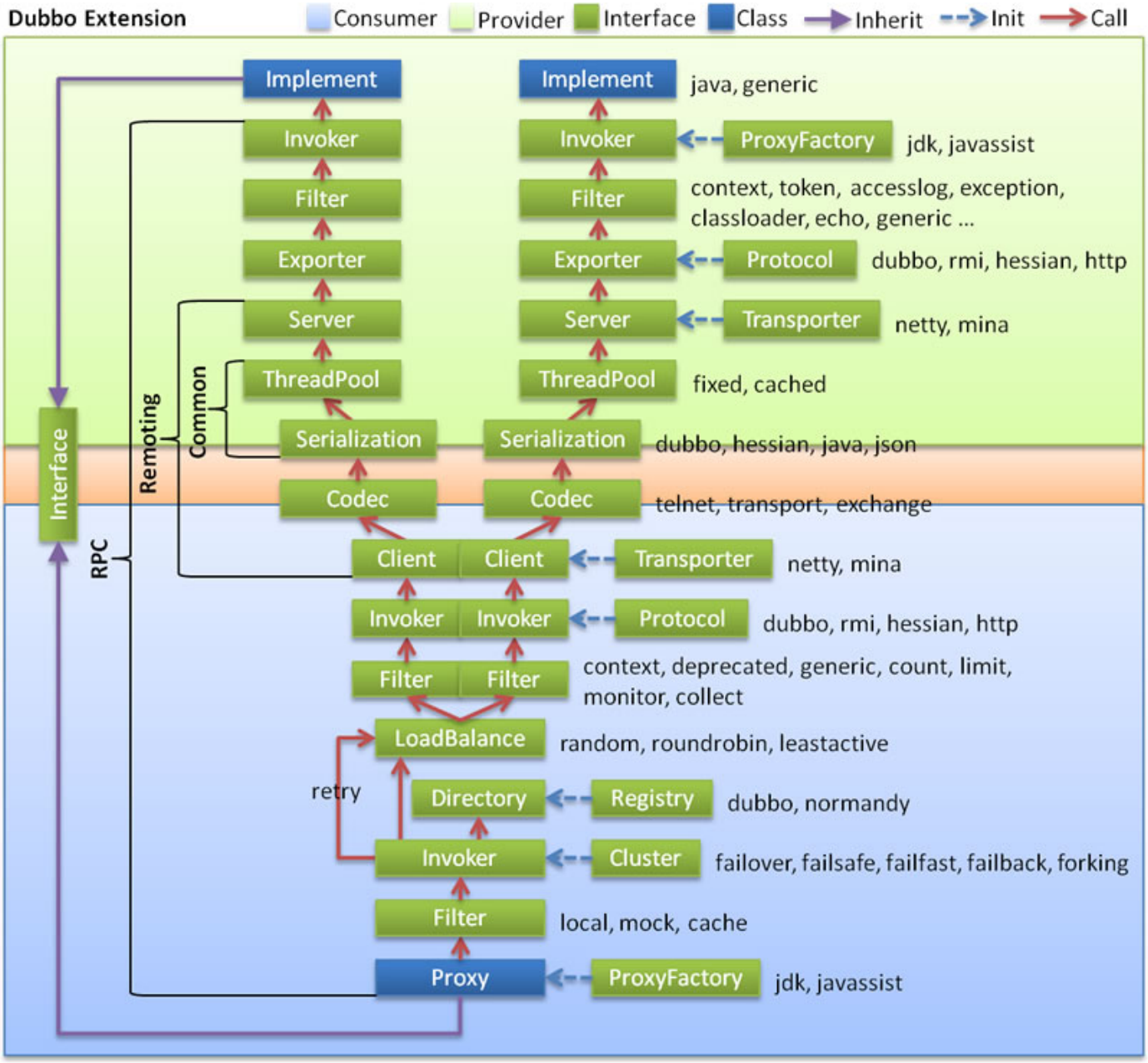
图中小方块 Protocol, Cluster, Proxy, Service, Container, Registry, Monitor 代表层或模块，蓝色的表示与业务有交互，绿色的表示只对 Dubbo 内部交互。

图中背景方块 Consumer, Provider, Registry, Monitor 代表部署逻辑拓扑节点。

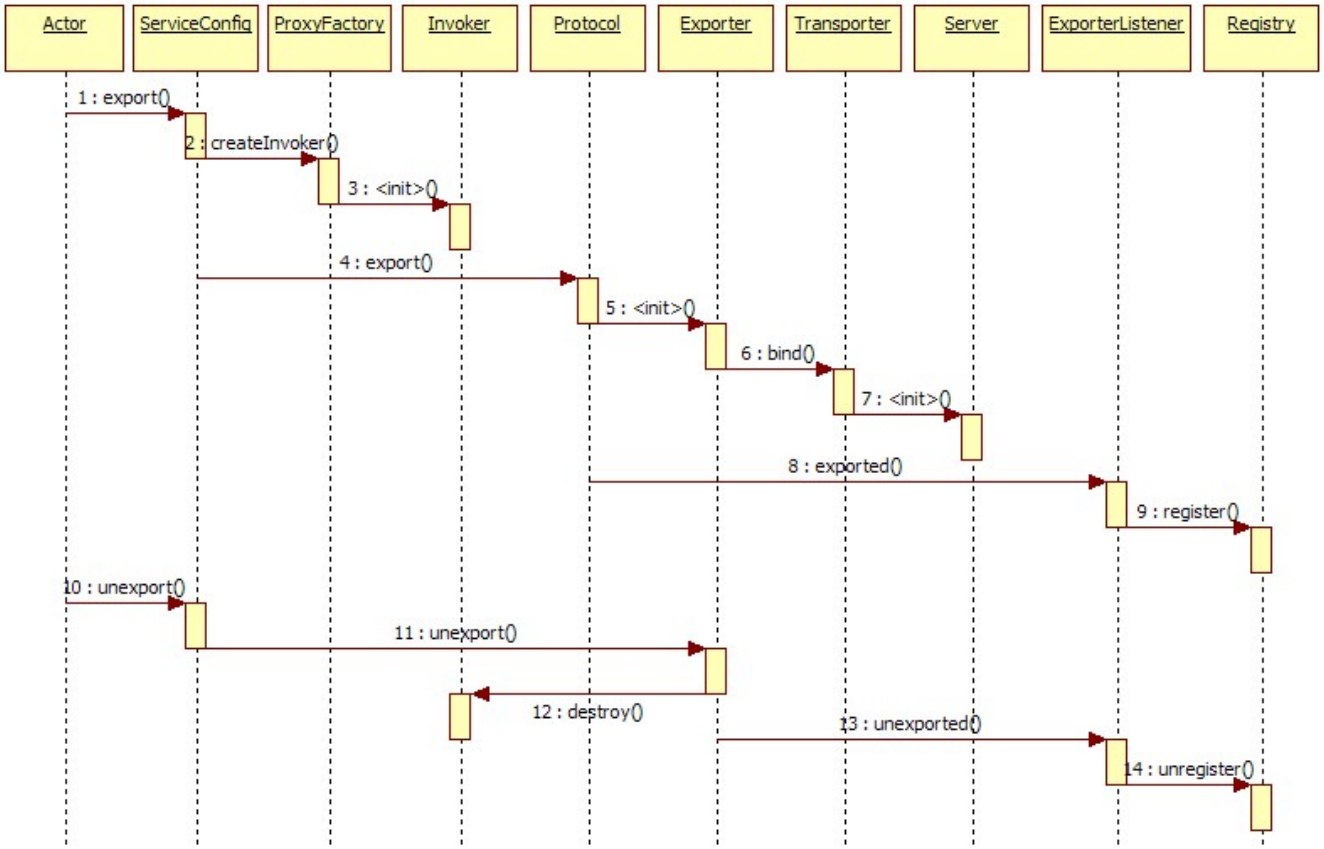
图中蓝色虚线为初始化时调用，红色虚线为运行时异步调用，红色实线为运行时同步调用。

图中只包含 RPC 的层，不包含 Remoting 的层，Remoting 整体都隐含在 Protocol 中。

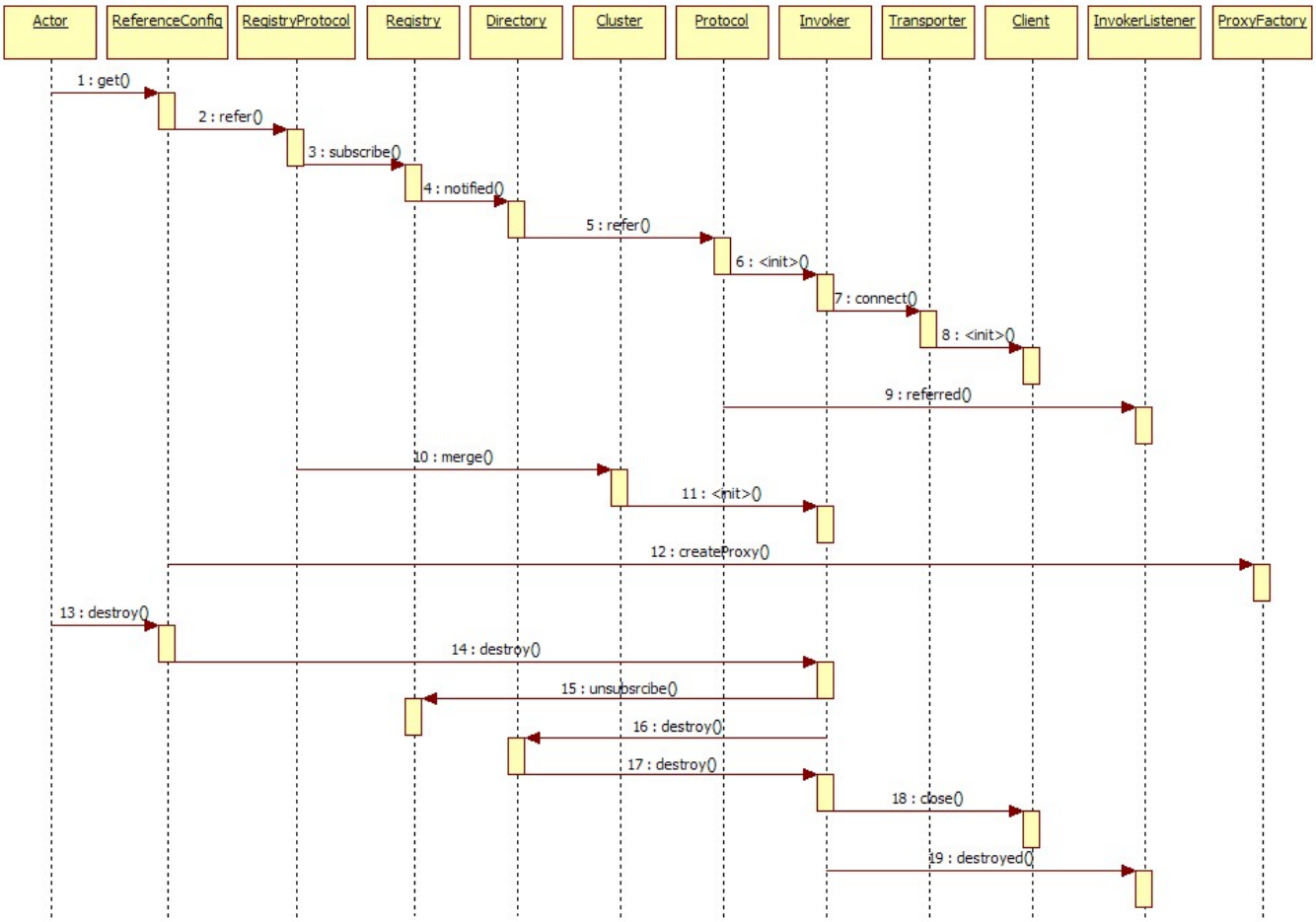
调用链


Dubbo Extension
Consumer
Provider
Interface
Class
Inherit
Init
Call
Implement
Invoker
Filter
Exporter
Server
ThreadPool
Serialization
Codec
Client
Invoker
Filter
LoadBalance
Directory
Invoker
Filter
Proxy
Implement
Invoker
Filter
Exporter
Server
ThreadPool
Serialization
Codec
Client
Invoker
Filter
Filter
LoadBalance
Directory
Invoker
Filter
Proxy
ProxyFactory
Protocol
Transporter
Protocol
Transporter
ProxyFactory
Cluster
Registry
java, generic
jdk, javassist
context, token, accesslog, exception, classloader, echo, generic ...
dubbo, rmi, hessian, http
netty, mina
fixed, cached
dubbo, hessian, java, json
telnet, transport, exchange
netty, mina
dubbo, rmi, hessian, http
context, deprecated, generic, count, limit, monitor, collect
random, roundrobin, leastactive
dubbo, normandy
failover, failsafe, failfast, failback, forking
local, mock, cache
jdk, javassist
Interface
Remoting
Common
RPC
retry

暴露服务时序


Actor
ServiceConfig
ProxyFactory
Invoker
Protocol
Exporter
Transporter
Server
ExporterListener
Registry
1 : export()
2 : createInvoker()
3 : <init>()
4 : export()
5 : <init>()
6 : bind()
7 : <init>()
8 : exported()
9 : register()
10 : unexport()
11 : unexport()
12 : destroy()
13 : unexported()
14 : unregister()

引用服务时序


Actor
ReferenceConfig
RegistryProtocol
Registry
Directory
Cluster
Protocol
Invoker
Transporter
Client
InvokerListener
ProxyFactory
1 : get()
2 : refer()
3 : subscribe()
4 : notified()
5 : refer()
6 : <init>()
7 : connect()
8 : <init>()
9 : referred()
10 : merge()
11 : <init>()
12 : createProxy()
13 : destroy()
14 : destroy()
15 : unsubsrcibe()
16 : destroy()
17 : destroy()
18 : close()
19 : destroyed()

领域模型

在 Dubbo 的核心领域模型中：

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。

Invocation 是会话域，它持有调用过程中的变量，比如方法名，参数等。

基本设计原则

采用 Microkernel + Plugin 模式，Microkernel 只负责组装 Plugin，Dubbo 自身的功能也是通过扩展点实现的，也就是 Dubbo 的所有功能点都可被用户自定义扩展所替换。

采用 URL 作为配置信息的统一格式，所有扩展点都通过传递 URL 携带配置信息。

四、性能测试

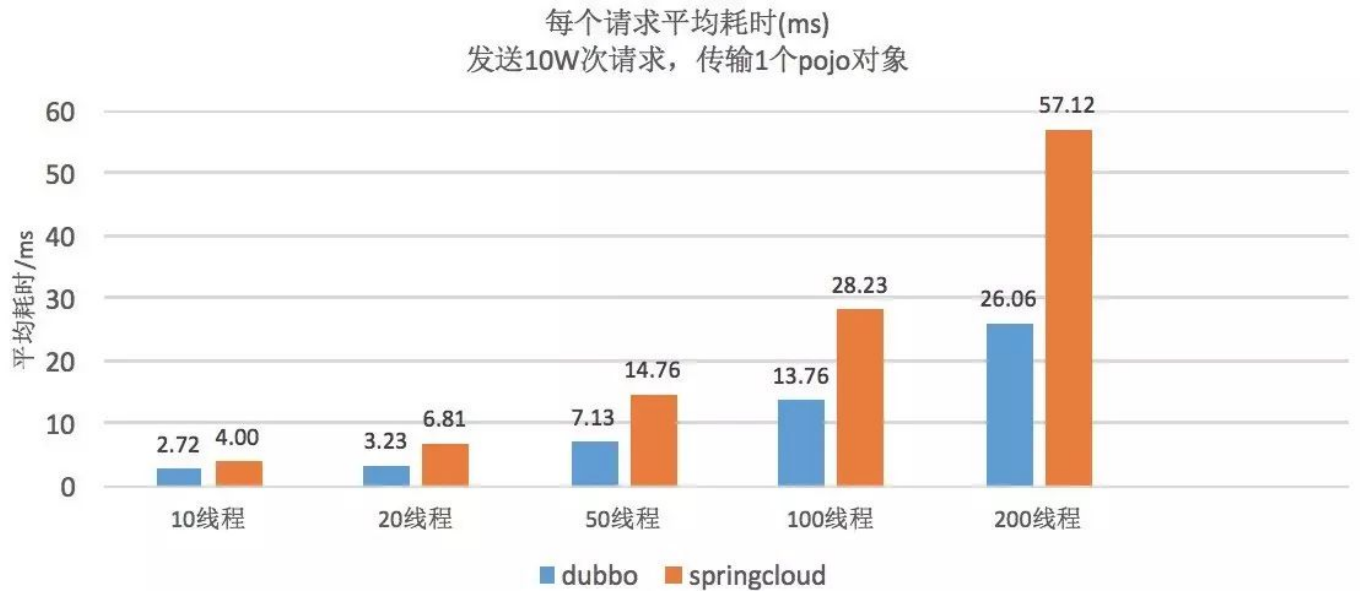
由于Spring Cloud与Dubbo天生使用的协议层面不一样，前者是HTTP，后者是TCP(使用的是Netty NIO框架，序列化使用的阿里定制版Hessian2)，导致两个框架的性能差距略大,基本上是三比一的差距！Dubbo官方TPS是1W左右，与测试最高值接近。由此也得出，框架的性能可能对一个真实的请求(Request)影响并不是很大，或者说并不起决定性作用，也许真正影响性能的是你的业务代码，比如数据库访问以及IO，当然了，框架的性能在一些对性能要求敏感的应用来说也是要考虑的。

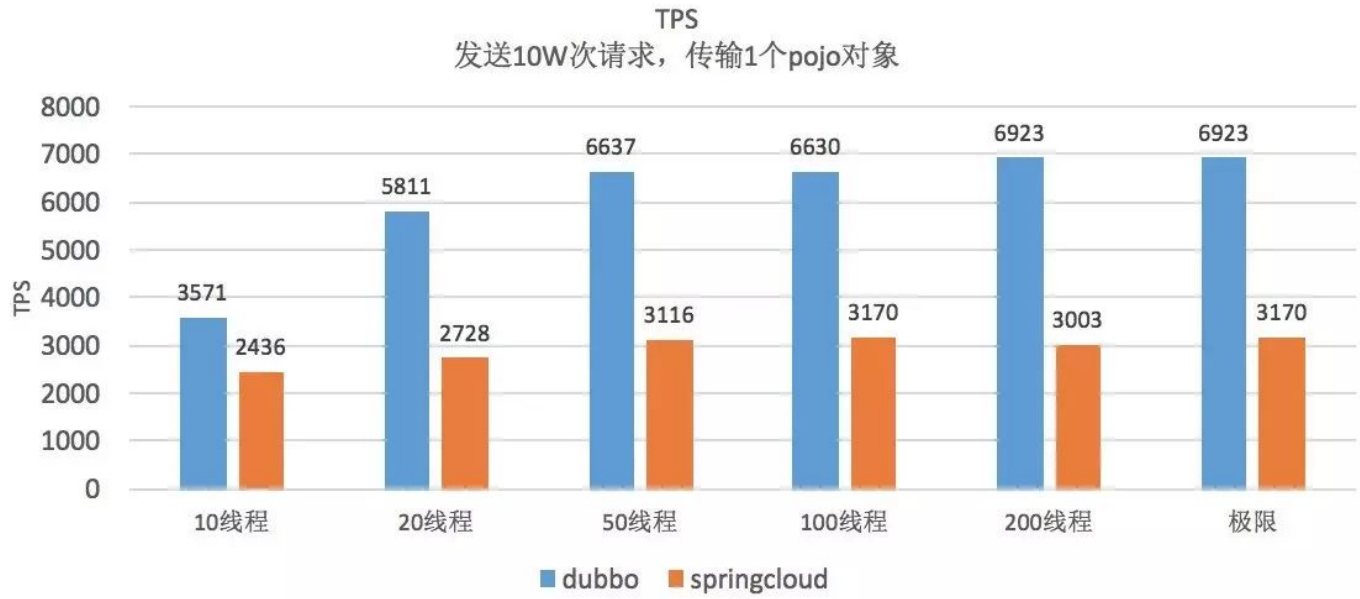
另外根据Dubbo官方说法，Dubbo在小数据量的情况下表现卓越，这和测试数据也是吻合的，在50个属性的pojo对象下，Dubbo性能确实下降了。

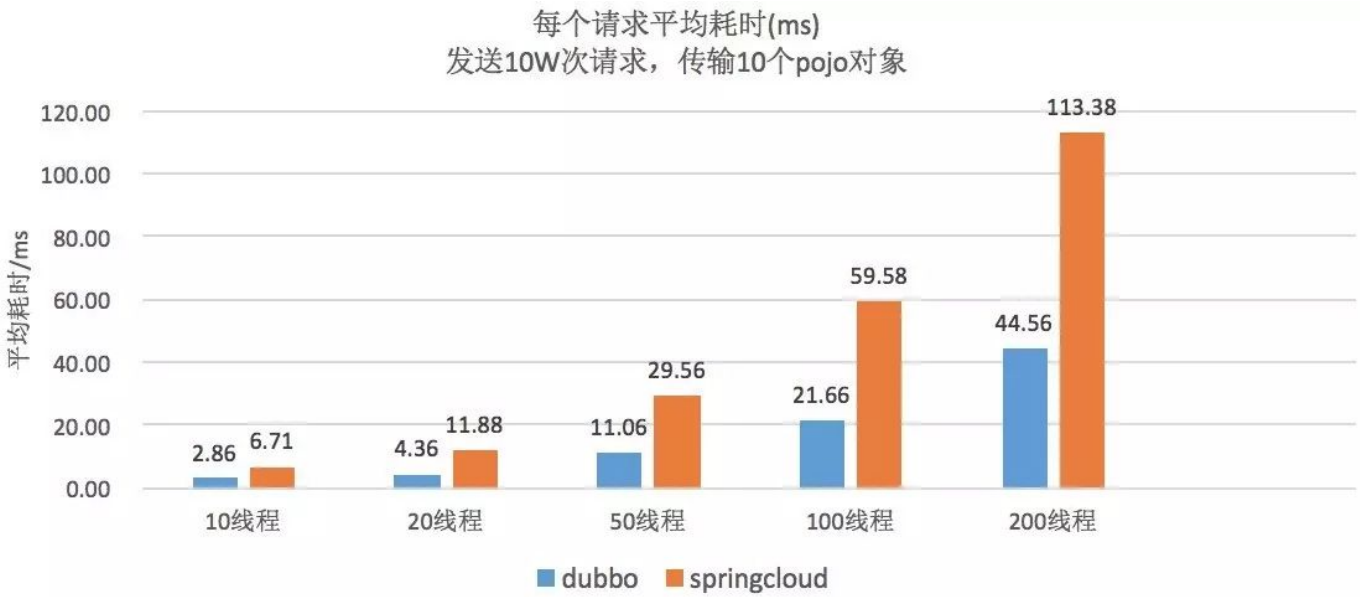
另外Spring Cloud默认的feigh client是使用jdk的urlconnection来做HTTP的请求，考虑这种做法的性能问题，尝试接入了httpclient包来测试，结果发现httpclient更慢，最后引入了开源的okhttp包，综合发现，okhttp和Spring Cloud的feign client结合是性能最高的。

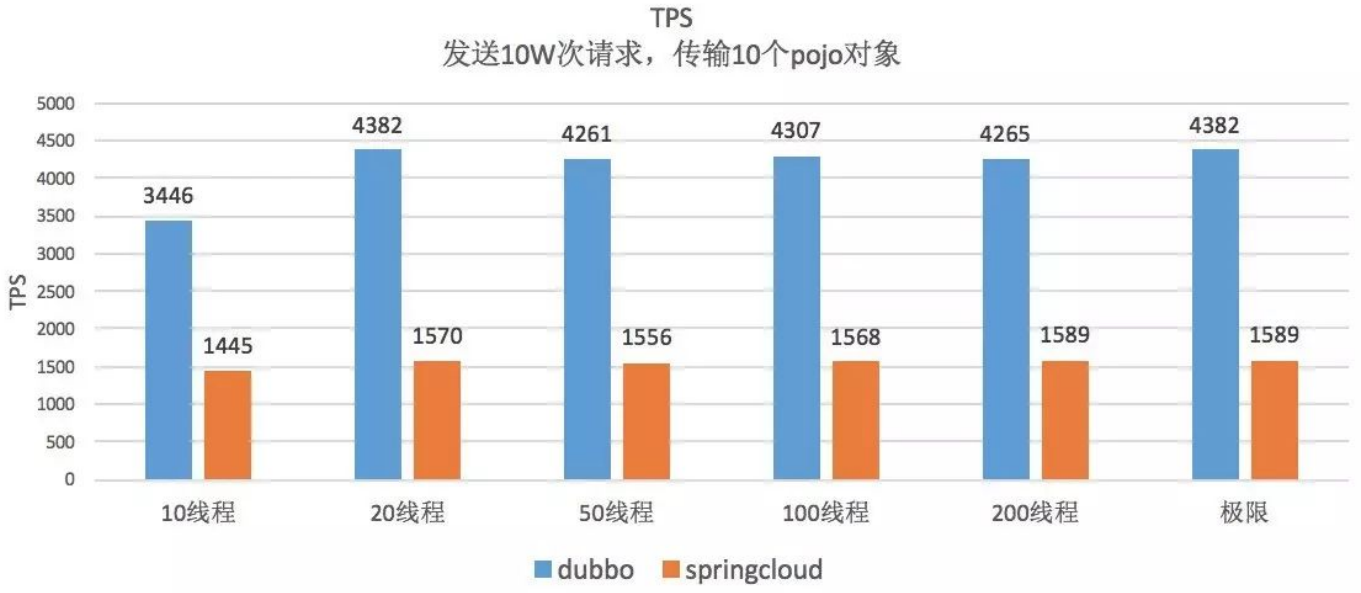
还有用RestTemplate进行测试，性能要比用Feigh还要好一些。大概能提升百分之十到十五。

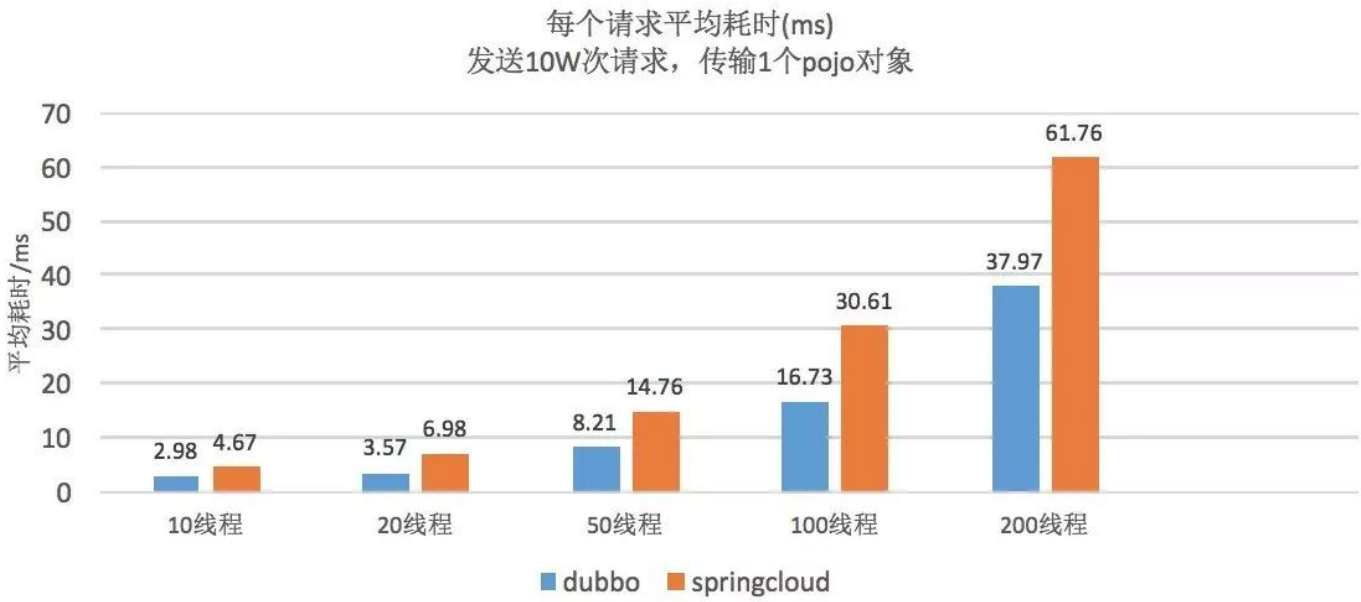
虽然Spring Cloud在性能上与Dubbo有天生的劣势，但考虑到Spring Cloud作为一套专门的微服务框架，再加上RESTful风格的API的趋势，从综合的角度，Spring Cloud无疑是你所在的公司未来微服务化进程中不可缺少的选择之一！


每个请求平均耗时(ms)
发送10W次请求，传输1个pojo对象
平均耗时/ms
60
50
40
30
20
10
0
2.72
4.00
3.23
6.81
7.13
14.76
13.76
28.23
26.06
57.12
10线程
20线程
50线程
100线程
200线程
dubbo
springcloud


TPS
发送10W次请求，传输1个pojo对象
TPS
8000
7000
6000
5000
4000
3000
2000
1000
0
3571
2436
5811
2728
6637
3116
6630
3170
6923
3003
6923
3170
10线程
20线程
50线程
100线程
200线程
极限
dubbo
springcloud


每个请求平均耗时(ms)
发送10W次请求，传输10个pojo对象
平均耗时/ms
120.00
100.00
80.00
60.00
40.00
20.00
0.00
2.86
6.71
4.36
11.88
11.06
29.56
21.66
59.58
44.56
113.38
10线程
20线程
50线程
100线程
200线程
dubbo
springcloud





成熟度

功能成熟度

Feature	Maturity	Strength	Problem	Advise	User
并发控制	Tested	并发控制		试用	
连接控制	Tested	连接数控制		试用	
直连提	Tested	点对点直连服务提供方，用于测		测试环	Alibaba

Feature	Maturity	Strength	Problem	Advise	User
供者		试		境使用	
分组聚合	Tested	分组聚合返回值，用于菜单聚合等服务	特殊场景使用	可用于生产环境	
参数验证	Tested	参数验证，JSR303验证框架集成	对性能有影响	试用	LaiWang
结果缓存	Tested	结果缓存，用于加速请求		试用	
泛化引用	Stable	泛化调用，无需业务接口类进行远程调用，用于测试平台，开放网关桥接等		可用于生产环境	Alibaba
泛化实现	Stable	泛化实现，无需业务接口类实现任意接口，用于Mock平台		可用于生产环境	Alibaba
回声测试	Tested	回声测试		试用	

隐式传参	Stable	附加参数		可用于生产环境	
异步调用	Tested	不可靠异步调用		试用	
本地调用	Tested	本地调用		试用	
参数回调	Tested	参数回调	特殊场景使用	试用	Registry
事件通知	Tested	事件通知，在远程调用执行前后触发		试用	
本地存根	Stable	在客户端执行部分逻辑		可用于生产环境	Alibaba
本地伪装	Stable	伪造返回结果，可在失败时执行，或直接执行，用于服务降级	需注册中心支持	可用于生产环境	Alibaba
延迟暴	Stable	延迟暴露服务，用于等待应用加		可用于	Alibaba

Feature	Maturity	Strength	Problem	Advise	User
露		载warmup数据，或等待spring加载完成		生产环境	
延迟连接	Tested	延迟建立连接，调用时建立		试用	Registry
粘滞连接	Tested	粘滞连接，总是向同一个提供方发起请求，除非此提供方挂掉，再切换到另一台		试用	Registry
令牌验证	Tested	令牌验证，用于服务授权	需注册中心支持	试用	
路由规则	Tested	动态决定调用关系	需注册中心支持	试用	
配置规则	Tested	动态下发配置，实现功能的开关	需注册中心支持	试用	
访问日志	Tested	访问日志，用于记录调用信息	本地存储，影响性能，受磁盘大小限制	试用	

分布式事务	Research	JTA/XA三阶段提交事务	不稳定	不可用	

策略成熟度

Feature	Maturity	Strength	Problem	Advise	User
Zookeeper注册中心	Stable	支持基于网络的集群方式，有广泛周边开源产品，建议使用dubbo-2.3.3以上版本（推荐使用）	依赖于Zookeeper的稳定性	可用于生产环境	
Redis注册中心	Stable	支持基于客户端双写的集群方式，性能高	要求服务器时间同步，用于检查心跳过期脏数据	可用于生产环境	
Multicast注册中心	Tested	去中心化，不需要安装注册中心	依赖于网络拓扑和路由，跨机房有风险	小规模应用或开发测	

Feature	Maturity	Strength	Problem	Advise	User
Simple注册中心	Tested	Dogfooding，注册中心本身也是一个标准的RPC服务	没有集群支持，可能单点故障	试用	
Feature	Maturity	Strength	Problem	Advise	User
Simple监控中心	Stable	支持JFreeChart统计报表	没有集群支持，可能单点故障，但故障后不影响RPC运行	可用于生产环境	
Feature	Maturity	Strength	Problem	Advise	User
Dubbo协议	Stable	采用NIO复用单一长连接，并使用线程池并发处理请求，减少握手和加大并发效率，性能较好（推荐使用）	在大文件传输时，单一连接会成为瓶颈	可用于生产环境	Alibaba
Rmi协议	Stable	可与原生RMI互操作，基于TCP协议	偶尔会连接失败，需重建Stub	可用于生产环境	Alibaba
Hessian协议	Stable	可与原生Hessian互操作，基于HTTP协议	需hessian.jar支持，http短连接的开销大	可用于生产环境	

Feature	Maturity	Strength	Problem	Advise	User
Netty Transporter	Stable	JBoss的NIO框架，性能较好（推荐使用）	一次请求派发两种事件，需屏蔽无用事件	可用于生产环境	Alibaba
Mina Transporter	Stable	老牌NIO框架，稳定	待发送消息队列派发不及时，大压力下，会出现FullGC	可用于生产环境	Alibaba
Grizzly Transporter	Tested	Sun的NIO框架，应用于GlassFish服务器中	线程池不可扩展，Filter不能拦截下一Filter	试用	
Feature	Maturity	Strength	Problem	Advise	User
Hessian Serialization	Stable	性能较好，多语言支持（推荐使用）	Hessian的各版本兼容性不好，可能和应	可用于生产环	Alibaba

Feature	Maturity	Strength	Problem	Advise	User
			用使用的Hessian冲突，Dubbo内嵌了hessian3.2.1的源码	境	
Dubbo Serialization	Tested	通过不传送POJO的类元信息，在大量POJO传输时，性能较好	当参数对象增加字段时，需外部文件声明	试用	
Json Serialization	Tested	纯文本，可跨语言解析，缺省采用FastJson解析	性能较差	试用	
Java Serialization	Stable	Java原生支持	性能较差	可用于生产环境	
Feature	Maturity	Strength	Problem	Advise	User
Javassist ProxyFactory	Stable	通过字节码生成代替反射，性能比较好（推荐使用）	依赖于javassist.jar包，占用JVM的Perm内存，Perm可能要设大一些：java -XX:PermSize=128m	可用于生产环境	Alibaba
Jdk ProxyFactory	Stable	JDK原生支持	性能较差	可用于生产环境	

Feature	Maturity	Strength	Problem	Advise	User
Failover Cluster	Stable	失败自动切换，当出现失败，重试其它服务器，通常用于读操作（推荐使用）	重试会带来更长延迟	可用于生产环境	Alibaba
Failfast Cluster	Stable	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	如果有机器正在重启，可能会出现调用失败	可用于生产环境	Alibaba
Failsafe Cluster	Stable	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	调用信息丢失	可用于生产环境	Monitor
Failback Cluster	Tested	失败自动恢复，后台	不可靠，重启丢失	可用于	Registry

Feature	Maturity	Strength	Problem	Advise	User
		记录失败请求，定时重发，通常用于消息通知操作		生产环境	
Forking Cluster	Tested	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	需要浪费更多服务资源	可用于生产环境	
Broadcast Cluster	Tested	广播调用所有提供者，逐个调用，任意一台报错则报错，通常用于更新提供方本地状态	速度慢，任意一台报错则报错	可用于生产环境	
Feature	Maturity	Strength	Problem	Advise	User
Random LoadBalance	Stable	随机，按权重设置随机概率（推荐使用）	在一个截面上碰撞的概率高，重试时，可能出现瞬间压力不均	可用于生产环境	Alibaba
RoundRobin LoadBalance	Stable	轮询，按公约后的权重设置轮询比率	存在慢的机器累积请求问题，极端情况可能产生雪崩	可用于生产环境	
LeastActive LoadBalance	Stable	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，使慢的机器收到更少请求	不支持权重，在容量规划时，不能通过权重把压力导向一台机器压测容量	可用于生产环境	
ConsistentHash LoadBalance	Stable	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	压力分摊不均	可用于生产环境	
Feature	Maturity	Strength	Problem	Advise	User
条件路由规则	Stable	基于条件表达式的路	有些复杂多分支条件	可用于	Alibaba

Feature	Maturity	Strength	Problem	Advise	User
		由规则，功能简单易用	情况，规则很难描述	生产环境	
脚本路由规则	Tested	基于脚本引擎的路由规则，功能强大	没有运行沙箱，脚本能力过于强大，可能成为后门	试用	
Feature	Maturity	Strength	Problem	Advise	User
Spring Container	Stable	自动加载META-INF/spring目录下的所有Spring配置		可用于生产环境	Alibaba
Jetty Container	Stable	启动一个内嵌Jetty，用于汇报状态	大量访问页面时，会影响服务器的线程和内存	可用于生产环境	Alibaba
Log4j Container	Stable	自动配置log4j的配置，在多进程启动时，自动给日志文件按进程分目录	用户不能控制log4j的配置，不灵活	可用于生产环境	Alibaba

dubbo官网文档内容 (http://dubbo.apache.org/zh-cn/)

← PREVIOUS POST (HTTP://39.106.3.150/ARCHIVES/INTERVIEW_BASE)

NEXT POST → (HTTP://39.106.3.150/ARCHIVES/15598824

开源框架 (http://39.106.3.150/tags/#1560569459781) Spring (http://39.106.3.150/tags/#spring)
设计模式 (http://39.106.3.150/tags/#1559888728999) linux (http://39.106.3.150/tags/#linux)
SpringBoot (http://39.106.3.150/tags/#springboot) 大数据 (http://39.106.3.150/tags/#1559363598973)
区块链 (http://39.106.3.150/tags/#1559363594390) Java (http://39.106.3.150/tags/#java)

FRIENDS

 (https://github.com/PowehiEdge)