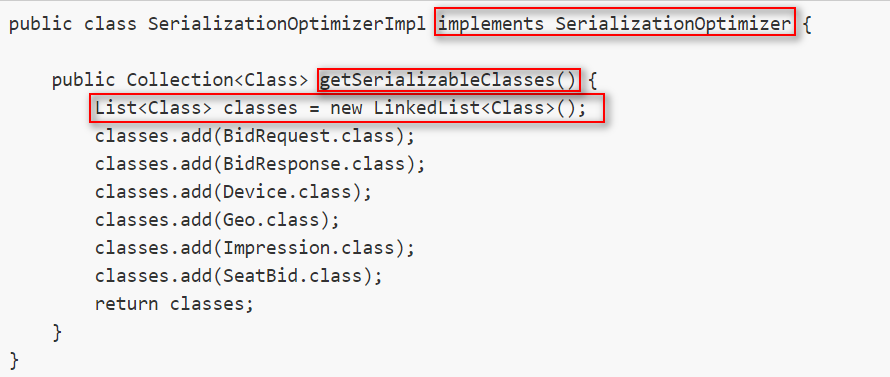
START

1. 相对于dubbo提供的RPC序列化方式，Dubbox提供了更高效的序列化方式Kryo；实际应用中建议采用kryo的序列化方式，因为它序列化和反序列化速度更快
2. 启用Kryo



1. 注册被序列化的类



1. 然后在xml配置中添加



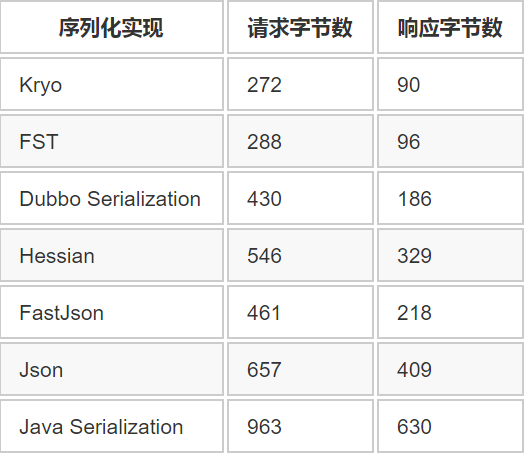
Tip: 在注册这些类后，序列化的性能可能被大大提升，特别针对小数量的嵌套对象的时候。

当然，在对一个类做序列化的时候，可能还级联引用到很多类，比如Java集合类。针对这种情况，dubbox自动将JDK中的常用类进行了注册

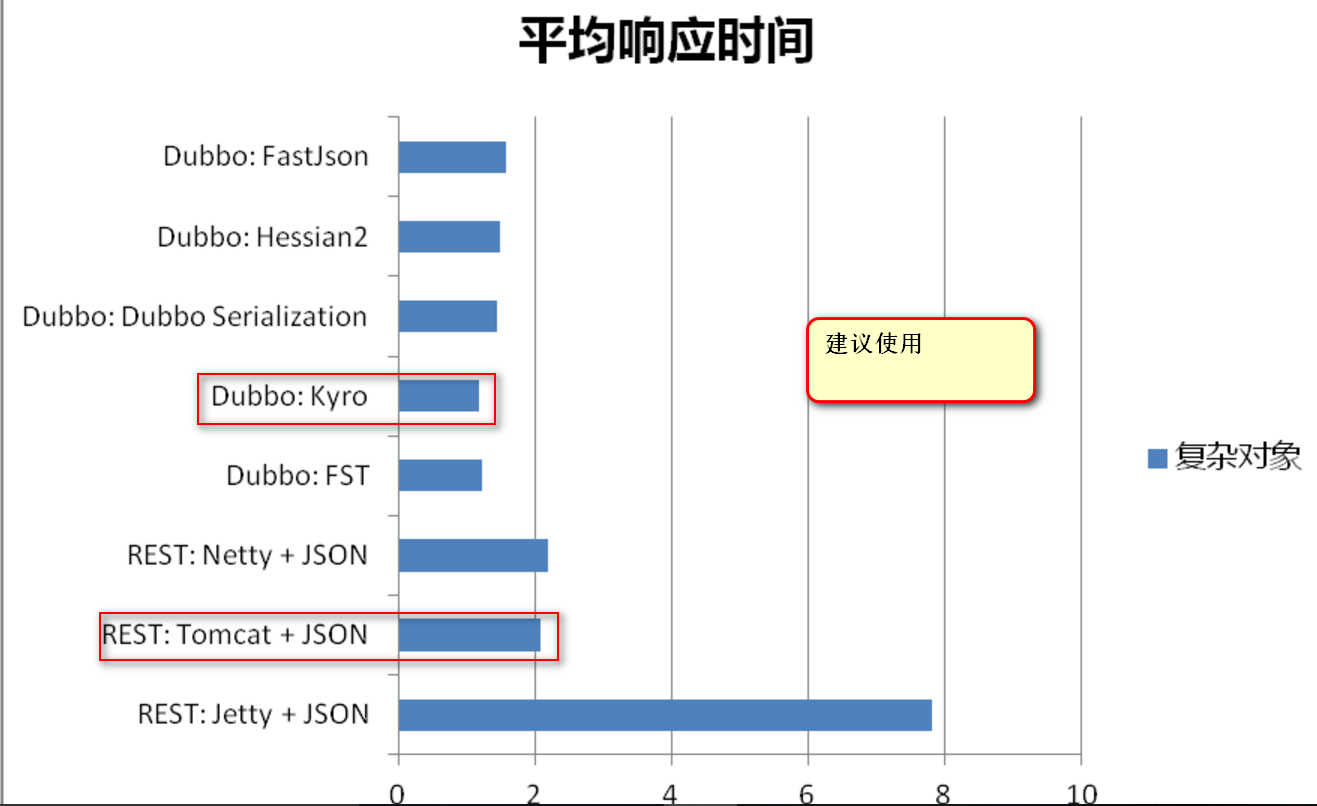
Tip: 如果被序列化的类中不包含无参的构造函数，则在Kryo的序列化中，性能将会大打折扣, 另外，Kryo和FST本来都不需要被序列化都类

实现Serializable接口，但我们还是建议每个被序列化类都去实现它，因为这样可以保持和Java序列化以及dubbo序列化的兼容性

1. 序列化之间的性能对比：
2. RPC之间不同序列化生成的字节大小比较
3. 序列化生成字节码的大小是一个比较有确定性的指标，它决定了远程调用的网络传输时间和带宽占用。
4. 针对复杂对象的结果如下（数值越小越好）：



1. Dubbo RPC中不同序列化响应时间和吞吐量对比



1. 综上所述

就目前结果而言，我们可以看到不管从生成字节的大小，还是平均响应时间和平均TPS，Kryo和FST相比Dubbo RPC中原有的序列化方式都有非常显著的改进。

1. 相对于dubbo支持RPC（java对象序列化+socket通讯），Dubbox 同时也支持restful风格远程调用；
2. Restful服务提供端详解
3. Restful 建议通过http协议中提供的四种方法实现通信：POST、DELETE、PUT、GET

Restful 风格url 

其他形式的URL来访问不同ID的用户资料

1. 服务提供端代码注解：



1. Annotation放在接口还是实现类

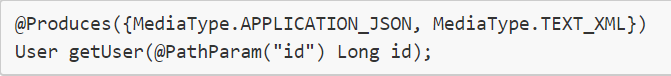
在一般应用中，我们建议将annotation放到服务实现类，这样annotation和java实现代码位置更接近，更便于开发和维护。另外更重要的是，我们一般倾向于避免对接口的污染，保持接口的纯净性和广泛适用性。

但是，如后文所述，如果我们要用dubbo直接开发的消费端来访问此服务，则annotation必须放到接口上。

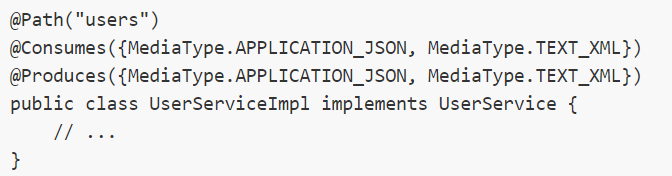
如果接口和实现类都同时添加了annotation，则实现类的annotation配置会生效，接口上的annotation被直接忽略。

1. 返回数据上的格式支持

比如，我们要让上例中的getUser()方法支持分别返回JSON和XML格式的数据，只需要在annotation中同时包含两种格式即可：

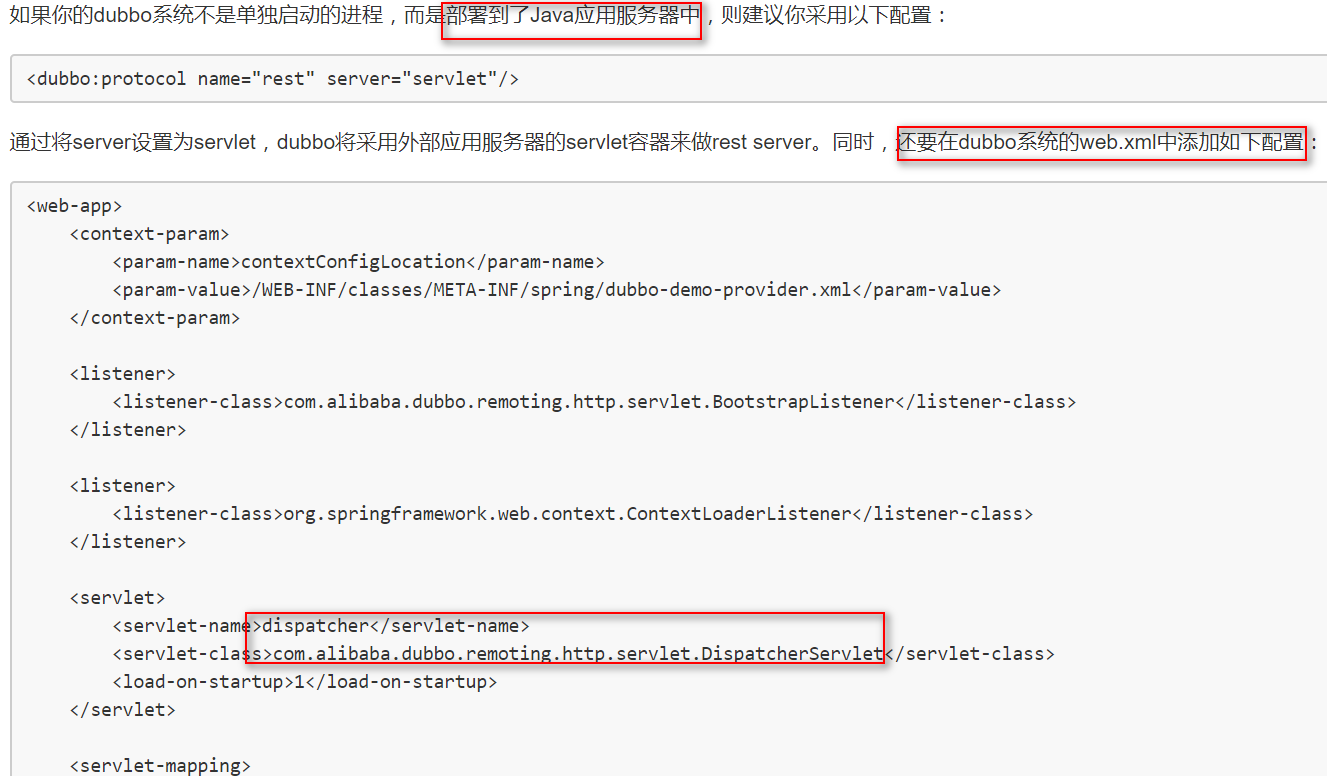


如果所有方法都支持同样类型的输入输出数据格式，则我们无需在每个方法上做配置，只需要在服务类上添加annotation即可：



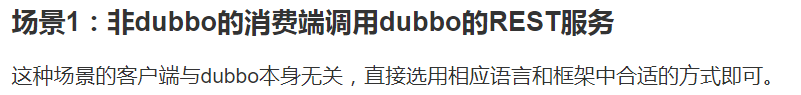
1. 配置rest Server实现



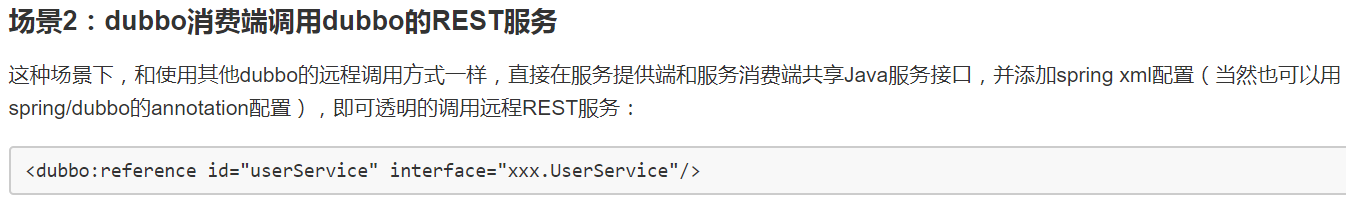


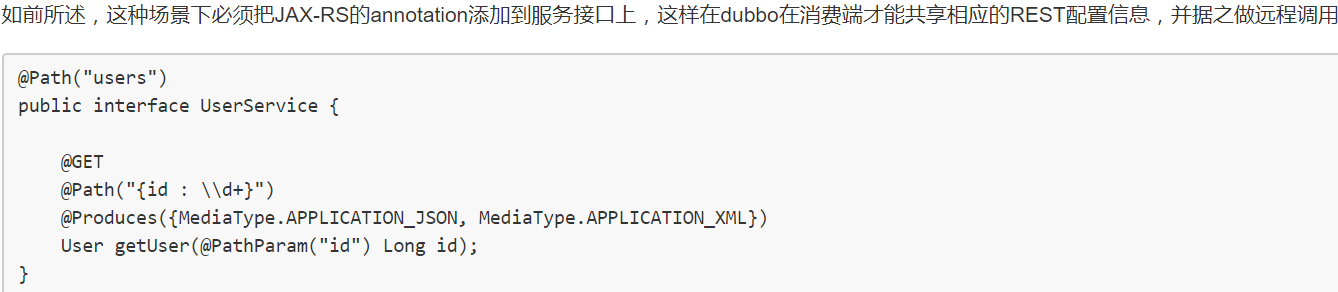
其他配置，例如服务线程数，长连接等请查看官方文档： <https://dangdangdotcom.github.io/dubbox/rest.html>

1. REST 消费端详解
2. No dubbo -> dubbo，直接写一个http client客户端，通过HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection() 建立连接请求数据；



1. Dubbo -> dubbo

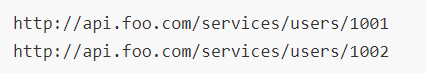




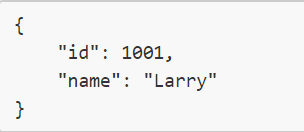
1. Dubbo -> no dubbo 客户端（消费端）dubbbo实现，我们理解no dubbo的provider端是一个开放的restful接口API，消费端直接调用即可；

Provider端有时候可能需要客户端提供private token进行API授权的验证

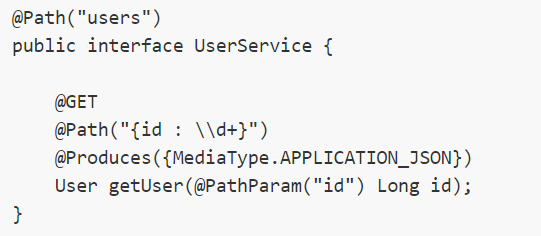
示例：我们要调用如下外部服务，users/{id}



返回的数据格式是json



接口声明如下：



配置：直接配置外部的RESTFUL地址即可，同样可以配置客户端可启动的最大连接数和超时时间：



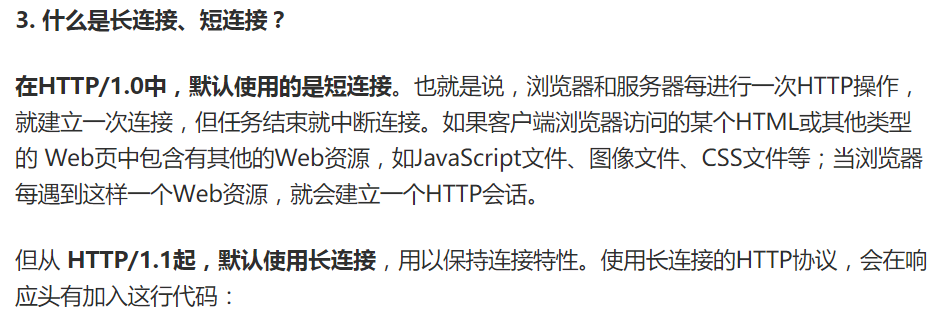
1. **结论：**RESTFUL和 dubbo rpc性能测试

性能优化建议：

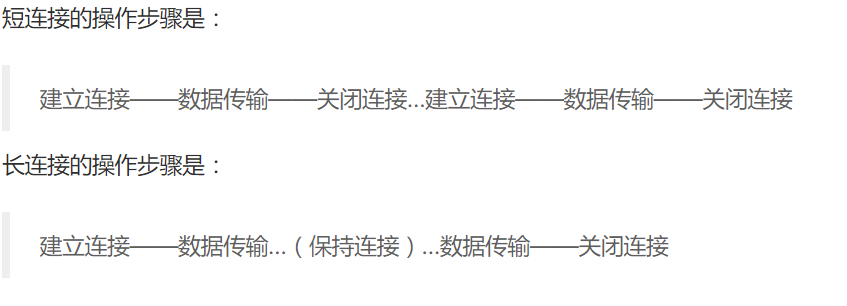
1. dubbo RPC（特别是基于高效java序列化方式如kryo，fst）比REST的响应时间和吞吐量都有较显著优势，内网的dubbo系统之间优先选择dubbo RPC。
2. 在REST的实现选择上，仅就性能而言，目前tomcat7和netty最优
3. 在REST中JSON数据格式性能优于XML



1. **拓展知识点：**长短连接概念，优劣，建议使用场景
2. 概念：



1. 长短连接的操作过程



1. 长短连接的优缺点

长连接可以省去较多的TCP建立和关闭操作，减少浪费，节约时间。对于频繁请求资源的客户来说，较适用于长连接

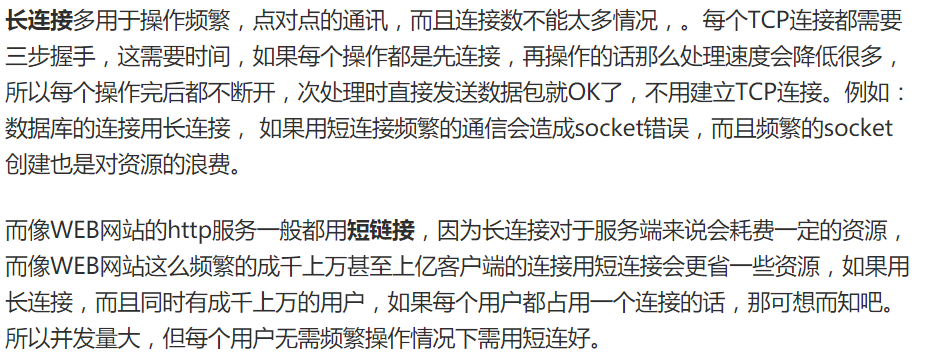
不过Client和server之间的连接如果一直不关闭的话，会存在一个问题，随着客户端连接的越来越多，server迟早有抗

不住的一天；这个时候就需要server采取一些措施

比如关闭一些长时间没有读写的连接，也可以限制每个客户端的最大长连接数；这样可以完全避免某个蛋疼的客户端连累后端服务。

短连接对于服务器管理较为简单，存在的连接都是有用的连接。但如果客户请求频繁，将在TCP的建立和关闭操作上浪费时间和带宽

1. 什么时候使用长连接，什么时候使用短连接



划重点：那么对于使用dubbo框架搭建的工程来说，controller--->war 和用户建立的是短连接；dao和service ----->作为微服务的provider端和controller--->war建立的是长连接

