1. **给我简单介绍下你的项目？**

本项目基于Netty、Spring、Zookeeper实现了一个简单的RPC框架，可以支持服务器客户端长连接，采用IO异步调用，服务器支持心跳检测，采用JSON序列化实现编解码，基于Spring注解和动态代理实现调用，最后基于zookeeper的Watcher机制实现了客户端连接的动态管理、监听和发现功能，并实现了服务器注册功能。

1. **你的项目的流程给我介绍一下**

首先需要启动Zookeeper服务器，然后通过Spring启动RPC的server，此时RPC server会将自身注册到Zookeeper上，在rpc server启动过程中，利用ApplicationListener监听机制完成server的初始化，通过BeanPostProcessor将服务器被自定义注解标注的bean及其方法存储进一个Map当中，这样当后续客户端发送消息时，就可以通过确认消息里携带的方法取出对应的方法，并执行相应处理。

然后启动rpc client，在启动过程中，client会向zookeeper获取服务器列表并与服务器相连，同时启动过程中会对通信接口进行动态代理，因为rpc的本质是像调用本地方法那样进行网络通信，所以动态代理的目的是完成通信过程的配置，比如将调用方法传递的参数包装成一个request，并使用netty的通信接口将request发送出去，从而达到对调用者隐藏通信细节的目的。最后，基于Zookeeper的watcher机制实现了客户端对服务端节点的监视功能。

1. **你的项目的架构是怎样的？**



1. **你的项目具有哪些特别的点？为什么要做这个项目？**

不同于其他简单CURD项目，本项目实现了一个基于spring、netty和zookeeper轻量级RPC框架，涉及了应用广泛的netty框架和spring框架的使用，包含了一些简单并发问题的解决。

因为感觉普通的curd项目有时技术含量较低，作为技术⼈总有这对技术追求的热情，通过对netty的学习搭建了一个RPC框架。

1. **你知道dubbo么？那你的项目和dubbo有什么区别？x**

Dubbo是由阿里开源的一种典型RPC框架，是一个高性能和透明化的RPC远程服务调用方案

我实现的rpc项目：零配置、序列化方式采用JSON序列化、开发时的想法是高性能服务器，所以采用了Netty作为项目的服务器、使用Spring CGLIB完成动态代理、

Dubbo项目：使用spring的schemas，配置繁琐、Dubbo支持多种序列化方式、Dubbo支持Tomcat，Jetty和Netty等等、使用JavassistProxy和JDK自带的动态代理、

1. 短链接，长链接：

**短链接有什么问题？如何实现的长连接？ X**

由于采用TCP作为传输协议，所以若采用短连接，则每次传输都要建立连接和断开连接，有很多网络通信消耗，这和开发预期的高性能RPC不符合。

利用Zookeeper分布式服务器，将服务端注册入Zookeeper服务器中，而客户端每次都从zookeeper服务器中获取服务端地址，连接后将返回的ChannelFuture存进一个set里，每次发送请求都从set从取出ChannelFuture发送信息，并且发送消息完毕后并不关闭连接，这就保证了客户端和服务端的长连接。

1. **基于TCP 还是http，为什么用TCP实现**

要做长连接的话，是不能用http协议来做的，因为http协议已经是应用层协议了，并且http协议是无状态的，而我们要做长连接，肯定是需要在应用层封装自己的业务，所以就需要基于TCP协议来做

同时TCP协议可以保证到达应用层的数据是正确无误的，是可靠的网络传输协议，这对于RPC通信框架来说是不可缺少的

1. **同步阻塞调用性能瓶颈：有什么瓶颈？ X 你怎么解决的，怎么实现异步调用的？**

同步阻塞的缺点是调用时线程会阻塞在请求中，cpu此时并不会切换到其他线程，若服务器不返回消息，则客户端会一直被阻塞，无法处理其他请求。

Netty是基于NIO开发的，采用了异步非阻塞通信方式，本项目采用了netty服务框架实现了客户端服务端开发。

1. **为什么需要心跳检测,客户端发送请求服务端没回应会造成什么(X)？心跳检测机制你是怎么实现的？(X)**

在系统空闲时，例如凌晨，往往没有业务消息。如果此时链路被防火墙Hang住，或者遭遇网络闪断、网络单通等，通信双方无法识别出这类链路异常。等到第二天业务高峰期到来时，瞬间的海量业务冲击会导致消息积压无法发送给对方，由于链路的重建需要时间，这期间业务会大量失败(集群或者分布式组网情况会好一些)。

为了解决这个问题，需要周期性的心跳对链路进行有效性检测，一旦发生问题，可以及时关闭链路，重建TCP连接。

当有业务消息时，无须心跳检测，可以由业务消息进行链路可用性检测。所以心跳消息往往是在链路空闲时发送的。

使用IdleStateHandler实现心跳检测机制，IdleStateHandler提供了3种心跳检测方法：

● 读空闲超时机制:当连续周期T没有消息可读时，触发超时Handler， 用户可以基于读空闲超时发送心跳消息，进行链路检测;如果连续N个周期仍然没有读取到心跳消息，可以主动关闭链路。

● 写空闲超时机制: 当连续周期T没有消息要发送时，触发超时Handler，用户可以基于写空闲超时发送心跳消息，进行链路检测;如果连续N个周期仍然没有接收到对方的心跳消息，可以主动关闭链路。

● 读写空闲超时机制：以上两者的结合

重写userEventTriggered方法，根据用于业务需要，判断event触发事件，当产生读空闲、写空闲、读写空闲时所需要进行的操作，如在发生读写空闲时，服务端发送“pong”，客户端发送“ping”做回应。

1. **什么叫做序列化？Java原生序列化问题(-)？为什么用JSON，怎么实现的JSON序列化，和JAVA序列化速度的差异？(x)**

序列化就是将类对象编码成可在网络中传输的二进制数据，jdk原生序列化有下面三个问题：

1. Java序列化机制是Java内部的一种对象编解码技术，无法跨语言使用。例如对于异构系统之间的对接，Java序列化后的码流需要能够通过其他语言反序列化成原始对象(副本)，目前很难支持。

2. 相比于其他开源的序列化框架，Java序列化后的码流太大，无论是网络传输还是持久化到磁盘，都会导致额外的资源占用。

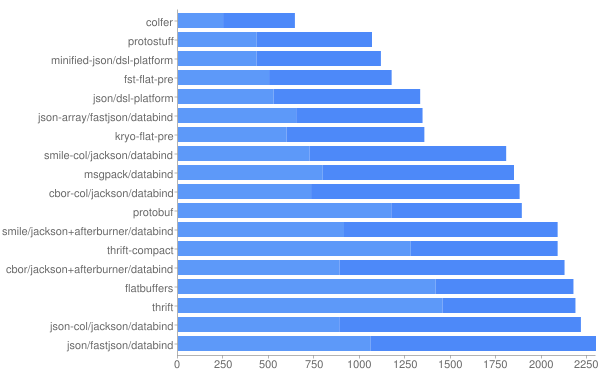
3. 序列化性能差，资源占用率高（主要是CPU资源占用高)。

使用fastjson的原因是这种序列化方式简单明了，且相比jdk序列化编解码速度快，序列化后码流较小。

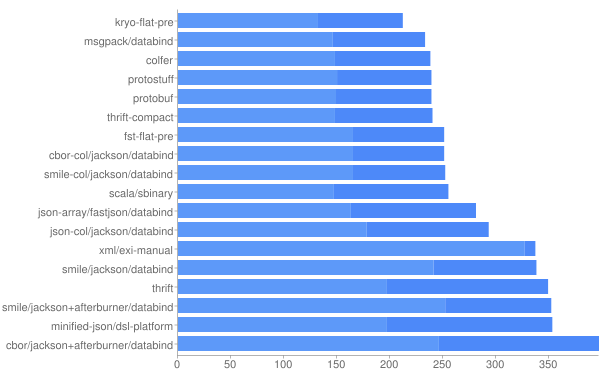
1. **为什么用JSON，应该还有其他更快的吧？(-)**
2. **你还知道哪些序列化的方式？**

以上几个问题

速度方面：



序列化后大小方面：



其他序列化方式主要有：protobuf和protostuff，hessian等

Hessian比JSON性能要高，但对java一些常见的对象类型不支持，如Linked系列、Locale类等等

Protobuf是由谷歌公司开发的数据语言，支持java、C++、python等多平台，序列化后体积较小，转化性能较高。但需要预编译IDL。

Protostuff 不需要依赖 IDL 文件，可以直接对 Java 领域对象进行反/序列化操作，在效率上跟 Protobuf 差不多，生成的二进制格式和 Protobuf 是完全相同的，可以说是一个 Java版本的 Protobuf 序列化框架。但不支持null，也不支持单纯的 Map、List 集合对象，需要包在对象里面。

1. **你是怎么基于动态代理进行的请求处理？用的哪一种？为什么使用这种?(X)**

使用SpringCGLIB进行动态代理，在intercept方法里对方法进行横切扩展，主要是将请求参数包装成ClientRequest类，并使用Netty客户端将请求发送出去

使用SpringCGLIB的优点就是可以对类进行代理，而jdk的动态代理只能针对接口

1. **解决进程通信问题这个我不太懂，你的项目哪里遇到进程通信问题了？ (X)**

当大量客户端同时向服务器发送请求时，服务器响应可能会产生并发问题，导致出现进程通信问题。

1. **你是怎么解决进程通信问题的？(X)**

通过引入wait和notify机制，使用ReentryLock和Condition进行控制。

① 创建锁

② 在主线程获取数据前，先等待结果

③ 在获取结果时，上锁，防止多线程同时获得结果

④ 如果没有获得结果，则进入Condition进行等待，并在finally释放锁

⑤ 获得结果，则调用signal唤醒锁。

1. **你还知道哪些解决进程通信问题的方式？**

Synchronized:重量级锁 volatile乐观锁 CAS机制

**以上几个问题在项目中没有遇到，这里记录一下以防被问到进程间通信相关**

1. **TCP/IP 拆包粘包能简单介绍下么？你怎么解决的呢？有没有更好的解决方式？(X Header Body)**

由于TCP协议是基于流的协议，没有消息边界，客户端发送数据时，由于TCP的neagle算法，为了减少通信消耗，会将多次发送的数据组合在一起发送，而在发送过程中有可能会对数据包进行分片，而服务端收到消息时，由于没有指定消息边界，因此多条消息的数据有可能连在一起，这就是粘包现象，还有一种可能是一条消息过长，因此被分片后服务器得到的数据只有消息的部分，这就是半包。

这种现象发生的原因有以下几种：应用程序write写入的字节大于套接字发送缓冲区；TCP进行MSS分片；以太网帧payload大于MTU。

我采用分隔符的方法解决TCP的粘包半包问题，使用DelimiterBasedFrameDecoder这一handler，指定分隔符为”/r/n”，每次发送消息时都会在消息后面加上这一分隔符，这样服务器就可以在应用层分辨出消息边界。

更好的解决方法可以采用“消息头+消息体”的方法，每条请求的消息头表明该消息的长度，可以使用LengthFieldBasedFrameDecoder这一handler标明数据长度。其他方法还有固定长度：FixedLengthFrameDecoder，或者短连接方式（但这不符合开发目的）

1. **你是怎么基于BeanPostProcessor 机制和 ApplicationListener 机制实现客户端的自启动与基于注解的服务调用**

首先客户端类TcpClient的静态代码块会初始化一个netty客户端，因此在BeanPostProcessor中完成请求的包装后，调用TcpClient.send()方法就实现了客户端的初始化自启动。

而服务端则将初始化代码放在一个start()方法里，同时服务端实现了ApplicationListener<ContextRefreshedEvent>，并在onApplicationEvent方法里调用服务端的start方法，这样当Spring启动完成后，监听器监听到ContextRefreshedEvent，就会启动netty server。

基于注解的服务调用方面，我将service的实现类标注了@Remote注解，这是一个自定义注解，同时在Spring启动过程中会有一个BeanPostProcessor，对@Remote注解标注的类进行处理，主要是将service接口的各种方法与接口进行映射；而客户端则对service的接口标注了@RemoteInvoke，在BeanPostProcessor的实现类中对被@RemoteInvoke标注的接口完成方法与接口的映射，这样服务器的map和客户端的map里关于方法和类名都是一致的，因此服务端可以知道该执行什么处理。

1. **注册中心你是怎么实现的？用的什么？**

对每个服务器设置一个序列号，每次启动一个服务器都会将该服务器注册同一路径下，保存该服务器地址和端口号

用的是基于zookeeper的服务注册

1. **为什么用Zk？(X)**

zookeeper中curator中的watcher机制，可以实现客户端连接的动态管理、监听和发现功能，并实现了服务器注册功能，扩展项目功能。

1. **你的服务结点结构是怎样的？**

路径/+IP地址+"#"+端口号+"#"（+权重+"#"）

1. **那么当有新的服务上线或者旧服务下线的时候，你怎么保证得到最新结果？(X)**
2. **客户端怎么发现服务？**
3. **怎么动态监听链接？(X)**
4. **给我讲一讲Watcher机制 ( X )**

以上四个问题一并回答

使用CuratorWatcher，构建一个实现Curator接口的ServerWatcher实现类，当客户端启动时，将watcher注册到自身的CuratorFramework上，并让其监视zookeeper的/netty目录节点，当该节点有变动——增加或减少时，产生的event就会触发CuratorWatcher的process方法，在这个方法里清除先前保存的服务器信息，重新查询所有服务端节点并连接，将返回的channelFuture保存到一个set里。

由于一个watcher注册一次只会监听一次，因此在process方法里需要再次注册watcher到CuratorFramework上

1. **给我讲一讲你项目中你认为最难的点以及你的解决方式?(-)**

基于Spring注解和动态代理实现对客户端和服务端请求和响应的调用

视频、博客、百度

1. **你的项目有哪些改进地方？你想改进哪里？**
2. 序列化方式，可以采用速度更快的protobuf或者protostuff
3. 针对客户端连接，改进负载均衡算法
4. 给服务端增加流量控制功能
5. 就注册中心而言，采用eureka代替zookeeper可以获得更高的服务可用性，当然这会导致数据一致性方面不是很好，客户端获取到的节点是过时的，不过由于有wacther的存在，请求失败一两次是可以接受的
6. **你这个项目的创新点有哪些？**

本项目没有单纯的基于netty开发RPC，同时也采用Spring、Zookeeper等开发框架，完善了项目的功能

1. **做这个项目的意义是什么？**

首先，做项目的整个过程，使我对Spring、netty和RPC框架有了一个整体的了解，巩固了java基础知识，增强了自己实战经验，提高了发现问题、解决问题的能力，对以后从事java相关工作有很大的帮助。

1. **项目的性能如何？哪些地方可以继续优化从而提高性能？**

RPC调用user.save方法，一万次5秒，10万次27秒，

调节jvm参数，避免频繁FULL GC，选择protobuf等更好的序列化方式。