手写_RPC_框架项目简历写法

建议

注意，以下简历写法仅供参考，根据你自己的简历丰富度、以及对于项目的理解情况有选择地去写。**如果你自己还没有实现项目或者不理解，建议赶紧跟着鱼皮的教程把它弄懂，再写到简历上！**

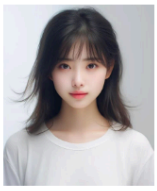
此外，本项目的部分知识，其实可以运用到你做的其他项目中，可以把该项目的部分亮点和你之前的项目进行整合。比如：

- 需求分析、拆解项目模块、系统设计的套路
- Vert.x、Etcd、SPI 机制、设计模式、负载均衡、容错机制、重试机制、注解驱动、动态代理、反射的运用实践

> 简历参考，写同款：https://laoyujianli.com/share/LchlxL <https://laoyujianli.com/share/LchlxL>

项目-更新

老鱼



13818996520 mycv@gmail.com
应届毕业生 求职意向：后端开发 深圳

教育经历

老鱼大学 计算机科学与技术 本科 2020-09 ~ 2024-07

专业技能

- 熟悉 Java，如集合类、注解、IO 流、JDK 序列化、异常处理等，能熟练运用反射、SPI 机制、动态代理提升项目的可扩展性
- 熟悉 SSM + Spring Boot 框架，能够独立开发基于 Spring Boot Starter 的 SDK
- 熟悉 Vert.x 反应式框架，能够基于它实现高性能的 HTTP / TCP 服务器和客户端
- 熟悉分布式应用开发：比如 Dubbo RPC 框架、Etcd 注册中心、负载均衡、设计重试和容错机制保证服务稳定性等
- 熟悉并实践过多种设计模式，比如工厂模式、装饰者模式、双检锁单例模式、代理模式等
- 熟练使用 Git、Maven、IDEA、Markdown 语法、浏览器控制台提高开发协作效率

项目经历

老鱼高性能 RPC 框架 2023-12 ~ 2024-03

后端开发 杭州 https://github.com/liyupi/yu-rpc

项目介绍：

基于 Java + Etcd + Vert.x + 自定义协议实现。开发者可以引入 Spring Boot Starter，通过注解和配置文件快速使用框架，像调用本地方法一样轻松调用远程服务；还支持通过 SPI 机制动态扩展序列化器、负载均衡器、重试和容错策略等。

主要工作：

1. 核心架构：包括消费方调用、序列化器、网络服务器、请求处理器、服务注册器模块。
2. 网络服务器：基于 Vert.x 的 HTTP 服务器，实现服务提供者和消费者的高性能网络通信。
3. 服务注册器：使用线程安全的 ConcurrentHashMap 存储本地服务注册信息，可以根据服务名称获取到对应实现类，并通过反射完成方法调用。
4. 序列化器：为便于扩展，编写通用的序列化器接口，并基于 Java 原生的 Object 和 ByteArray 输入输出流实现 JdkSerializer 序列化器，使得对象能够网络传输。
5. 请求处理器：基于 Vert.x 的 Handler 接口实现对请求的异步处理，将请求数据反序列化后，从服务注册器中找到服务实现类并通过反射机制调用。
6. 消费方调用：基于 JDK 动态代理 + 工厂模式实现，为指定服务接口类生成可发送 HTTP 请求的代理对象，实现远程方法的无感知调用。

个人优势

- 有较强的自学和理解能力，曾阅读 Etcd、ZooKeeper 等技术的官方文档自学，并运用到项目中实现注册中心

- 有较强的问题解决能力，能够利用 GitHub Issues 区、AI 工具、搜索引擎、Stack Overflow 等自主解决问题
- 有较强的调研分析和系统设计能力，曾参考 Dubbo 的设计方案，自主设计实现了一款高性能 RPC 框架

专业技能

后端

1. 熟悉 Java，如集合类、注解、IO 流、JDK 序列化、异常处理等，能熟练运用反射、SPI 机制、动态代理提升项目的可扩展性。
2. 熟悉 Java 常用类库，如 Hutool 工具库、JSON 序列化库、HttpClient 网络请求库、Guava Retrying 重试库、Logback 日志框架等。
3. 熟悉 SSM + Spring Boot 框架，能够独立开发基于 Spring Boot Starter 的 SDK。
4. 熟悉 Vert.x 反应式框架，能够基于它实现高性能的 HTTP / TCP 服务器和客户端。
5. 熟悉分布式应用开发：比如 Dubbo RPC 框架、Etcd 注册中心、负载均衡、设计重试和容错机制保证服务稳定性等。
6. 熟悉并实践过多种设计模式，比如工厂模式、装饰者模式、双检锁单例模式、代理模式等
7. 熟练使用 Git、Maven、IDEA、Markdown 语法、浏览器控制台提高开发协作效率

项目经历

项目名称：XX RPC 框架（比如 “yu-rpc 高性能 RPC 框架”）

建议根据自己对项目的学习理解程度，自己想个有区分度的名字，其他名称参考：

- XX RPC
- XX 可扩展 RPC 框架
- XX 高可用 RPC 框架
- XX RPC - 基于 Etcd 的 RPC 框架
- XX RPC - 基于 Vert.x 的 RPC 框架
- XX RPC - 自定义协议 RPC 框架
- XX RPC - 简单易用的 RPC 框架
- XX RPC - 注解驱动的 RPC 框架

GitHub 代码地址：https://github.com/liyupi/yu-rpc <https://github.com/liyupi/yu-rpc>

建议大家也把项目放到代码仓库中，并且在主页文档里补充项目架构图、项目模块等介绍信息。

但是注意不要抄袭鱼皮的写法！一定要加上自己的理解和扩展！否则一下就被面试官查出来了。

扩展思路可以参考鱼皮手写 RPC 框架教程的第 12 节。

项目介绍

精简版

> 简单直接地突出亮点，适合简历内容丰富的同学

基于 Java + Etcd + Vert.x + 自定义协议实现。开发者可以引入 Spring Boot Starter，通过注解和配置文件快速使用框架，像调用本地方法一样轻松调用远程服务；还支持通过 SPI 机制动态扩展序列化器、负载均衡器、重试和容错策略等。

详细版

> 把核心亮点前置并补充简单的介绍，适合简历内容不多的同学

参考 Dubbo 开源项目自主设计实现的 Java 高性能 RPC 框架。开发者只需引入 Spring Boot Starter，就能通过注解和配置的方式快速使用框架，实现像调用本地方法一样轻松调用远程服务。

本项目有较多的实现亮点：基于 Vert.x TCP 服务器 + 自定义协议实现网络传输；基于 Etcd 实现注册中心以完成服务的注册消费；还支持通过 SPI 机制动态扩展序列化器、负载均衡器、重试和容错策略等。

主要工作

> 根据自己对不同模块的掌握程度，选 6 个左右去写并适当调整文案，灵活一点。 强烈建议结合下面的扩展思路多完善下项目，增加一些区分度！

后端（简易版）

1．核心架构：包括消费方调用、序列化器、网络服务器、请求处理器、服务注册器模块。
2．网络服务器：基于 Vert.x 的 HTTP 服务器，实现服务提供者和消费者的高性能网络通信。

3. 服务注册器：使用线程安全的 ConcurrentHashMap 存储本地服务注册信息，可以根据服务名称获取到对应实现类，并通过反射完成方法调用。
4. 序列化器：为便于扩展，编写通用的序列化器接口，并基于 Java 原生的 Object 和 ByteArray 输入输出流实现 JdkSerializer 序列化器，使得对象能够网络传输。
5. 请求处理器：基于 Vert.x 的 Handler 接口实现对请求的异步处理，将请求数据反序列化后，从服务注册器中找到服务实现类并通过反射机制调用。
6. 消费方调用：基于 JDK 动态代理 + 工厂模式实现，为指定服务接口类生成可发送 HTTP 请求的代理对象，实现远程方法的无感知调用。

后端（扩展版）

> 每句话冒号前的内容可以省略

1. 核心架构：包括消费方调用、序列化器、网络服务器、请求处理器、注册中心、负载均衡器、重试策略、容错策略等模块。
2. 消费方调用：基于 JDK 动态代理 + 工厂模式实现消费方调用模块，为指定服务接口类生成可发送 HTTP 请求的代理对象，实现远程方法的无感知调用。
3. 全局配置加载：使用双检锁单例模式维护全局配置对象，并通过 Hutool 的 Props 实现多环境配置文件的加载。
4. 接口 Mock：通过 JDK 动态代理 + 工厂模式实现，为指定服务接口类生成返回模拟数据的 Mock 服务对象，便于开发者测试。
5. 多种序列化器实现：定义序列化器接口，实现了基于 JSON、Kryo 和 Hessian 的序列化器，并通过 ThreadLocal 解决了 Kryo 序列化器的线程安全问题。
6. 可扩展设计：使用工厂模式 + 单例模式简化创建和获取序列化器对象的操作。并通过扫描资源路径 + 反射自实现了 SPI 机制，用户可通过编写配置的方式扩展和指定自己的序列化器。
7. 注册中心：基于 Etcd 云原生中间件实现了高可用的分布式注册中心，利用其层级结构和 Jetcd 的 KvClient 存储服务和节点信息，并支持通过 SPI 机制扩展。
8. 注册中心优化：利用定时任务和 Etcd Key 的 TTL 实现服务提供者的心跳检测和续期机制，节点下线一定时间后自动移除注册信息。
9. 消费者服务缓存：使用本地对象维护已获取到的服务提供者节点缓存，相比于每次从注册中心获取，性能提高了 xx%；并通过 Etcd 的 Watch 机制，监听节点的过期并自动更新缓存。
10. 自定义协议：由于 HTTP 协议头信息较多，基于 Vert.x TCP 服务器 + 类 Dubbo 的紧凑型消息结构（字节数组）自实现了 RPC 协议，提升网络传输性能。
11. 半包粘包问题解决：基于 Vert.x 的 RecordParser 完美解决半包粘包问题，并使用装饰者模式封装了 TcpBufferHandlerWrapper 类，一行代码即可对原有的请求处理器进行增强，提高代码的可维护性。

项目-更新

12. 负载均衡器：为提高服务提供者集群处理能力，实现了一致性 Hash、轮询、随机等不同算法的负载均衡器，并通过 SPI 机制支持开发者自行扩展。
13. 重试机制：为提高消费端调用的稳定性，基于 Guava Retrying 实现了包括 fixedWait 等多种重试策略，并通过 SPI 机制支持开发者自行扩展。
14. 容错机制：为提高系统的稳定性和可用性，设计实现了 FailOver、FailBack、FailSafe、FailFast 等多种重试策略，并通过 SPI 机制支持开发者自行扩展。
15. 注解驱动：为降低开发者的使用成本，封装了服务提供者和消费者启动类；并开发了基于注解驱动的 Spring Boot Starter，一个注解就能快速注册 Bean 为服务、以及注入服务调用代理对象。

个人评价

1. 有较强的自学和理解能力，曾阅读 Etcd、ZooKeeper 等技术的官方文档自学，并运用到项目中实现注册中心。
2. 有较强的问题解决能力，能够利用 GitHub Issues 区、AI 工具、搜索引擎、Stack Overflow 等自主解决问题。
3. 有较强的调研分析和系统设计能力，曾参考 Dubbo 的设计方案，自主设计实现了一款高性能 RPC 框架。

url=https%3A%2F%2Fwww.yuque.com%2Fu37765561%2Fak85bt%2F56534b5bbb6c0f3aca4fdaf0e