SSE208 Project1 RPC 实现

一、 目的

- 了解 RPC 框架，包括其原理、应用场景。
- 按要求动手实现一套自己的 RPC 协议

二、 前言

2.1 什么是 RPC

RPC（全称 Remote Procedure Call ）是一种进程间通信方式[1]。它允许程序调用另一个地址空间（通常是共享网络的另一台机器上）的过程或函数，而不用程序员显式编码这个远程调用的细节。即无论是调用本地接口/服务的还是远程的接口/服务，本质上编写的调用代码基本相同。

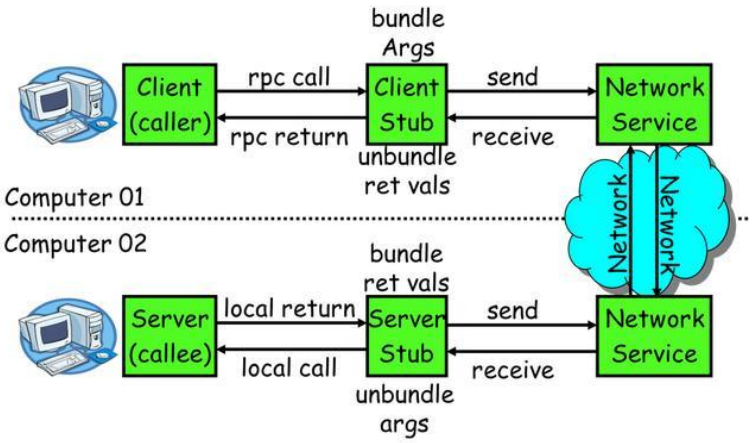
比如两台服务器 A，B，一个应用部署在 A 服务器上，想要调用 B 服务器上应用提供的函数或者方法，由于不在一个内存空间，不能直接调用，这时候就可以应用 RPC 框架的实现来解决。

RPC 会隐藏底层的通讯细节（不需要直接处理 Socket 通讯或 Http 通讯）

RPC 是一个请求响应模型。客户端发起请求，服务器返回响应（类似于 Http 的工作方式）

RPC 在使用形式上像调用本地函数（或方法）一样去调用远程的函数（或方法）。

2.2 RPC 基本流程



图一： RPC 调用流程图

一次完整的 RPC 调用流程（同步调用）如图一所示：

1）客户端（client）以本地调用方式调用服务；

2）client stub 接收到调用后负责将方法、参数等组装成能够进行网络传输的消息体；

3）client stub 找到服务地址，并将消息发送到服务端；

4）server stub 收到消息后进行解码；

5）server stub 根据解码结果调用本地的服务；

6）本地服务执行并将结果返回给 server stub；

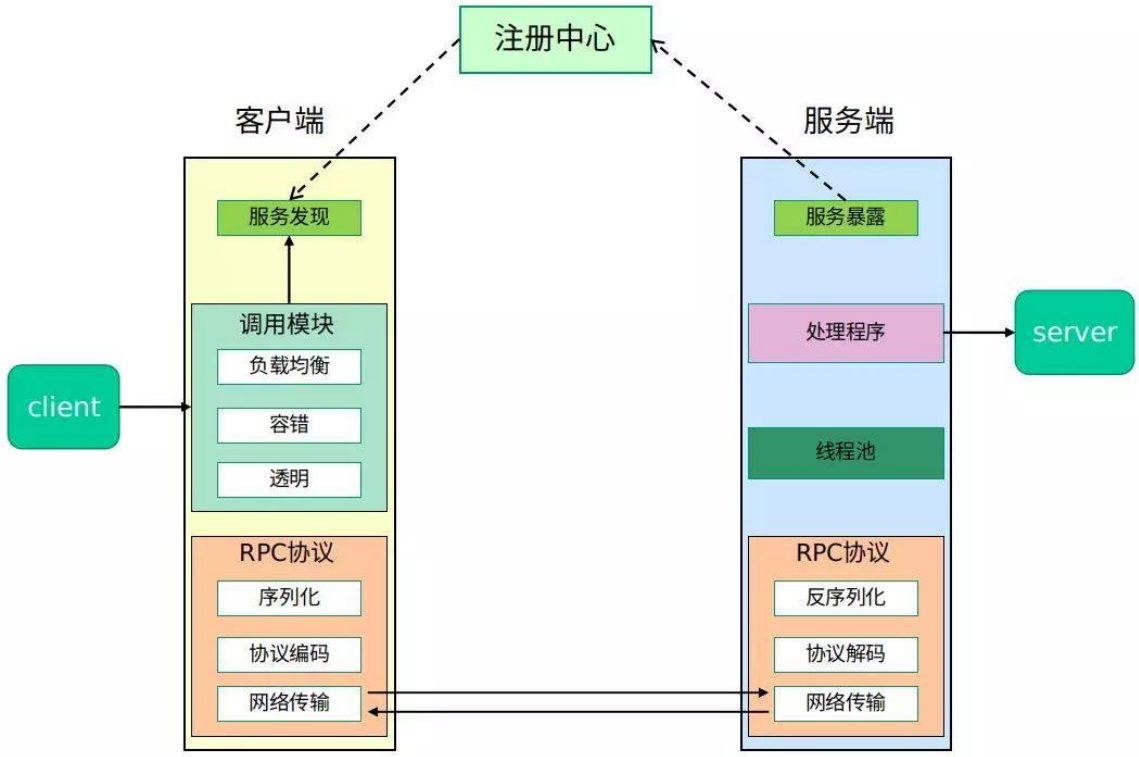
7）server stub 将返回结果打包成消息并发送至消费方；

8）client stub 接收到消息，并进行解码；

9）客户端得到最终结果。

2.3 RPC 框架

典型的 RPC 框架如图二所示



图二 典型的 RPC 框架

PRC 框架主要包括以下功能：

（1）**序列化。**本地过程调用中，我们只需要把参数压到栈里，然后让函数自己去栈里读就行。但是在远程过程调用时，客户端跟服务端是不同的进程，不能通过内存来传递参数。这时候就需要客户端把参数先转成一个字节流，传给服务端后，再把字节流转成自己能读取的格式。

序列化的方式有很多，通用的语言格式比如 JSON、XML，目前主流高效的开源序列化框架有 fastjson、Hessian、Protobuf 等。

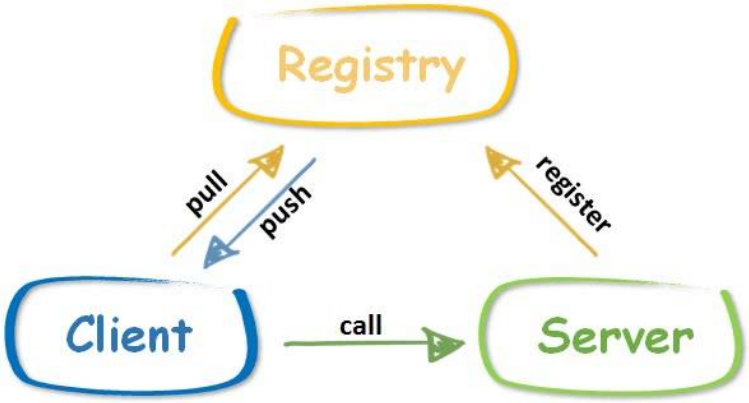
（2）**服务注册。**在 RPC 中，服务注册是指将服务提供者的信息注册到服务器或服务注册中心，让服务消费者可以通过服务注册中心查找到服务提供者的位置和配置信息。服务发现是指服务消费者从服务器或服务注册中心查找到可用的服务提供者，并连接到服务提供者，以便进行服务调用。

（3）**网络通信。**远程调用往往用在网络上，因此我们需要实现网络通信，把需要调用过程信息发送到服务器。通信方式可以使用 TCP、UDP，甚至直接使用 http

（4）**服务发现 & 服务注册中心。**服务器把自己的服务注册到服务注册中心，客户端只需要向服务注册中心询问服务器的地址，就可以连接并调用服务端所发布的服务，而不是硬编码服务器的地址。

加入服务注册中心后，RPC 架构如下图三所示。具体来说：

1. 服务端启动后，向注册中心发送注册消息，注册中心得知该服务已经启动，处于可用状态。一般来说，服务端还需要定期向注册中心发送心跳，证明自己还活着。
2. 客户端向注册中心询问，当前哪天服务是可用的，注册中心将可用的服务列表返回客户端。
3. 客户端根据注册中心得到的服务列表，选择其中一个发起调用。



图三 服务复现 & 服务注册中心

（5）**负载均衡。**在高并发的场景下，需要多个节点或集群来提升整体的吞吐能力。常用的负载均衡策略有：

- 随机选择策略：从服务列表中随机选择一个。
- 轮询算法：依次调度不同的服务器。比如假设有 n 台服务器，本次调度的服务器为 i，下次调度的服务器 j = (i + 1) mode n。
- 加权轮询：在轮询算法的基础上，为每个服务实例设置一个权重，高性能的机器赋予更高的权重，也可以根据服务实例的当前的负载情况做动态的调整，例如考虑最近 5 分钟部署服务器的 CPU、内存消耗情况。
- 哈希/一致性哈希策略：依据请求的某些特征，计算一个 hash 值，根据 hash 值将请求发送到对应的机器。一致性 hash 还可以解决服务实例动态添加情况下，调度抖动的问题。一致性哈希的一个典型应用场景是分布式缓存服务。感兴趣的可以阅读并动手写分布式缓存。

（6）**超时处理。**超时处理是 RPC 框架一个比较基本的能力，如果缺少超时处理机制，无论是服务端还是客户端都容易因为网络或其他错误导致挂死，资源耗尽，这些问题的出现大大地降低了服务的可用性。因此，我们需要在 RPC 框架中加入超时处理的能力。

2.4 RPC Demo

关于 RPC 的 demo 我们提供 gRPC 的官方文档作为参考[2]，同学们可以先体验下 gRPC，然后再动手去实现自己的 RPC 框架。

三、作业要求

注：标✱项目要求必须实现，因为这是 RPC 最基本的功能，其他未标✱项目根据同学自身情况实现，不做硬性要求。

3.1 消息格式定义，以及消息序列化和反序列化（✱）

消息的格式，以及其序列化和反序列化方式可以自行定义，具体可以**参考之前我们处理 tcp 粘包的过程**，另外消息的序列化和反序列化方式也可以使用其他主流的序列化方式，如 json、xml 和 protobuf 等方式。

3.2 服务注册（✱）

RPC 服务端启动时需要注册其能支持函数。**我们要求服务端至少能同时支持注册 10 个以上的函数。**

3.3 服务发现（❋）

RPC 服务器需要为客户端提供接口，这样客户端才能知道服务端是否支持其希望调用的服务。

3.4 服务调用（❋）

在 RPC 客户端发现服务后，根据你所设置的 RPC 协议正确地调用远程服务。服务调用的输入和输出的数据格式即在 3.1 你定义的格式。

3.5 支持并发（❋）

服务端需要具有并发处理客户端请求的能力。比如，假设客户端 A 发来请求，然后服务端处理客户端 A 的请求，这时客户端 B 也发来了请求，要求服务端也能同时处理客户端 B 的请求，不能出现服务端处理完客户端 A 的请求才能处理客户端 B 的请求，导致客户端 B 需要等待。具体地，可以利用多线程或者多进程的方式，参考我们之前的编程作业！

另外，我们要求，服务端至少可以支持并发处理 10 个客户端的请求。

3.6 异常处理及超时处理（❋）

RPC 框架需要提供异常处理及超时处理的能力，包括但不限于以下几个方面。

客户端处理异常/超时的地方：

- 与服务端建立连接，导致的异常/超时
- 发送请求到服务端，写数据导致的异常/超时
- 等待服务端处理时，等待处理导致的异常/超时（比如服务端已挂死，迟迟不响应）
- 从服务端接收响应时，读数据导致的异常/超时

服务端处理异常/超时的地方：

- 读取客户端请求数据时，读数据导致的异常/超时
- 发送响应数据时，写数据导致的异常/超时

- 调用映射服务的方法时，处理数据导致的异常/超时

3.7 服务注册中心

可以支持多个服务端把自己的服务注册到服务注册中心，客户端向服务注册中心询问服务端的地址并调用。

3.8 负载均衡

为了减少服务端的负载，服务端肯定不能只有一个，客户端可以通过服务注册中心选择服务器。

因此，负载均衡功能就是把每个请求平均负载到每个服务器上，充分利用每个服务器的资源。

注：考虑成本原因，不同的服务器可以使用多个虚拟机或 docker 镜像，但不能是单机多线程或多进程。

3.9 其他要求（❋）

3.9.1 编程语言选择

我们限定编程语言为：C、C++、Java、Python 和 Golang。另外，为了让同学们更容易上手实现，我们提供一个 C 语言版本的基本代码框架，具体可以查看代码里的注释，已经写得比较清楚。

3.9.2 运行教程

你需要在设计文档详细说明程序的运行教程，包括客户端和服务端，具体的，服务端的启动参数必须包括：

1）-h，帮助参数，输出参数帮助，具体格式可以自行定义，清晰易懂即可

2）-l，服务端监听的 ip 地址，需要同时支持 IPv4 和 IPv6，可以为空，默认监听所有 ip 地址，即 0.0.0.0

3）-p，服务端监听的端口号，不得为空

客户端的启动参数必须包括：

1）-h，帮助参数，输出参数帮助，具体格式可以自行定义

2）-i，客户端需要发送的服务端 ip 地址，需要同时支持 IPv4 和 IPv6，，不得为空

3）-p，客户端需要发送的服务端端口，不得为空。

比如，以 C,C++,或其他编译型语言（直接编译成可执行文件）为例子的客户端和服务端的启动命令：

服务端：./server -l 192.168.3.1 -p 2333

客户端：./client -i 192.168.3.1 -p 2333

3.9.3 不能利用现有框架或第三方库（除了消息序列化和反序列化外）

除了消息序列化和反序列化外，不得使用任何现有的框架或第三方库。当然操作系统和语言本身提供的库（非第三方库）是允许使用的，比如 socket 编程，我们需要用到类 Unix 的 POSIX 接口去实现。**除此之外的功能必须由你们自己设计并实现。**

3.9.4 其他

代码不得相互抄袭。当然，一些设计思路可以相互借鉴，但代码必须自己实现。**发现有严重的抄袭行为，直接 0 分处理！**

本作业只要求按照上述要求去实现，并不违反规则，具体使用什么方法、什么设计思路，由同学们自由发挥。

3.10 加分项

为了鼓励同学们更加深入了解 RPC，并让框架达到企业级的水平，我们设置了一个加分项：同学们之间共同设计同一套 RPC 协议，每个同学分别用不同的编程语言去实现（注意：不得同一种语言实现同一套 RPC 协议的多个实现版本）。另外还需实现跨语言之间的相互调用（比如同学 A 用 Java 语言实现了客户端 client，同学 B 用 C 语言实现了服务端，客户端可以通过共同定义的 RPC 协议调用服务端的服务），否则不视为同一套 RPC 协议，不参与加分。

注：上述加分项需在代码、设计文档及其他文件中体现，否则不予加分。

四、大作业成果提交

1、将所有文件打包在一个.zip 压缩文件当中，上传到作业系统，内容包括：

（1）所有实现代码。**注：必须是文本类型的文件，如果是编译型语言（如，C、C++），不要上传可执行文件，另外，如果是 C 或 C++语言，需要提供 Makefile 文件；**

（2）设计文档（5 页以上），需要清晰体现自己如何实现作业要求的关键思路，其内容必须包括但不限于：

- 对应 3.1。你是如何定义消息格式的？以及你是如何进行消息序列化和反序列化？
- 对应 3.2。服务端是如何注册服务的，用了什么数据结构来组织服务？注册服务的接口是什么？
- 对应 3.3。服务端提供给客户端的服务发现接口是什么？服务端是如何从你为服务所设计的数据结构中找到服务的？
- 对应 3.4。服务调用中的输入和输出的数据结构是什么，你是如何将请求消息和响应结果组织到这个数据结构中的？
- 对应 3.5。你是如何实现并发的？用多线程还是多进程？是否用到线程池或进程池之类的技术以及你是如何实现的？是否用了 IO 多路复用的方式？如果用了 IO 多路复用的方式，使用 select、poll 还是 epoll 的哪一种？为什么选它？
- 对应 3.6。你是如何设计异常和超时处理过程的？
- （可选）对应 3.7。服务端是如何将自己的服务注册到服务注册中心的？是否永久注册？还是有心跳检测来实现服务保活？
- （可选）对应 3.8。你是怎么设计负载均衡策略的？
- 你使用的网络传输层协议是什么？为什么用它而不是别的？
- 每点要求都需要有详细的解释，另外还需要辅以关键代码的截图或代码文本片段！
- 代码启动教程！！！

（3）**（可选）**其他体现你工作的文件，比如视频、PPT 等；

2、提交截至时间（Due date）：2023 年 7 月 8 日 11:59 am（二十周）

五、评分标准

本作业满分为 100 分（100 分封顶，基础项 100 分+加分项 10 分），各个要求点的评分标准如下。

要求点（对应第三章的作业要求）	分数
3.1 消息格式定义，以及序列化和反序列化	10
3.2 服务注册	10
3.3 服务发现	10
3.4 服务调用	10
3.5 支持并发	10
3.6 异常处理及超时处理	10
报告文档质量	优：40，良：20，差：10
加分项：多个同学合作实现同一套 RPC 协议，每个同学分别利用不同的编程语言去实现，且可以互相调用	10

六、参考阅读

[1] Birrell A D, Nelson B J. Implementing remote procedure calls[J]. ACM Transactions on Computer Systems (TOCS), 1984, 2(1): 39-59.
[2] “GRPC 官方文档中文版.” https://doc.oschina.net/grpc?t=60133. N.p., n.d. Web.

GOOD LUCK!