**YA-RPC: Yet-Another RPC Framework 设计报告**

# 引言

## 编写目的

通过学院开设的分布式系统课程学习了分布式系统的本原理，了解了远程过程调用(RPC: Remote Procedure Call)的基本概念，此次课程大作业选择的题目为“YA-RPC: Yet-Another RPC Framework”，此设计报告文档旨在说明实现该题目的整个过程，包括相关原理、项目体系结构、重点代码、测试结果，以便使读者理解整个系统的结构和功能。

## 背景

RPC由三部分组成：RPC Server、RPC Client、Registry。RPC Server作为服务提供方，暴露服务；RPC Client作为服务消费方，调用远程服务；Registry实现服务的注册与发现。这三部分协作实现RPC的简要过程为：RPC Server在Registry中注册服务，RPC Client在Registry中订阅服务，Registry将服务的信息传给RPC Client，最终RPC Client调用RPC Server提供的服务。

## 说明

分布式系统大作业选题：YA-RPC: Yet-Another RPC Framework

开发语言：Python

开发环境：PyCharm，Anaconda

设计与开发者：邓棋（202222080416）

# 系统设计与实现

## 需求分析

开发的YA-RPC框架需支持三种基本数据类型：int, float, string；支持 At-least-once 语义；需要实现两个API： float sum(float a, float b)，string uppercase(str)；并不少于2个客户端，1个服务端。

由此可分析得到该系统的非功能性需求有：

1. 访问透明性：客户端调用函数时就像调用本地函数一样。
2. 并发要求：至少实现并发数为 2 的并发调用。

## 概要设计

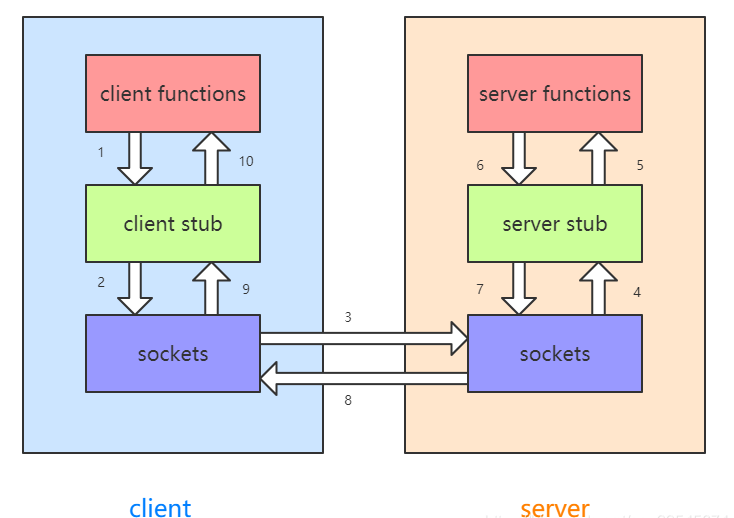
要实现RPC需要解决3个主要问题：

（1）Call ID 映射。如何确定客户端到底调用的是哪个函数？在本地调用中，函数体是直接通过函数指针来指定的，但是在远程调用中，函数指针是不行的，因为两个进程的地址空间是完全不一样的。所以，在 RPC 中所有的函数都必须有自己的一个 ID，这个 ID 在所有进程中都是唯一确定的。客户端在做远程过程调用时，必须附上这个ID。然后我们还需要在客户端和服务端分别维护函数与Call ID的对应表。当客户端需要进行远程调用时，它就查一下这个表，找出相应的 Call ID，然后把它传给服务端，服务端也通过查表，来确定客户端需要调用的函数，然后执行相应函数的代码。

（2）序列化和反序列化。客户端怎么把参数值传给远程的函数呢？在本地调用中，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。这个过程叫序列化和反序列化。同理，从服务端返回的值也需要序列化反序列化的过程。

（3）网络传输。客户端和服务端通过网络连接的，所有的数据都需要通过网络传输，因此需要有一个网络传输层把Call ID 和序列化后的参数字节流传给服务端，然后再把序列化后的调用结果传回客户端。

因此，实现RPC的基本工作原理为：部署在不同服务器上的客户端想调用服务端提供的服务，由于不在一个内存空间，不能直接调用，需要通过网络来表达调用的语义和传达调用的数据。因此需要客户端把参数转换成字节流，传给服务端，然后服务端将字节流转换成自身能读取的格式，这是一个序列化和反序列化的过程；数据准备好了之后，通过网络传输层把序列化后的参数传给服务端，然后把计算好的结果序列化传给客户端。该过程如图2.1所示。

图2.1 RPC调用过程

## 详细设计

## 系统实现与测试

# 总结