软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.000000]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于 TinyRPC 的 TinyNTP 程序设计*

骆克云[1]

[1](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210023)

作者: 骆克云, E-mail: keyun@smail.nju.edu.cn

摘 要: RPCLIB 是一个利用 C++14 设计的 RPC 库，使用 msgpack 作为序列化库，使用 Asio 作为网络通信库，它同时提供了客户端和服务器端的实现。本作业基于 RPCLIB 进行代码重构和必要精简，同时将客户端库和服务器端链接库分离，最大程度降低耦合性，兼顾跨平台简化编译，形成 TinyRPC 库。在 TinyRPC 库之上，进行服务器端和客户端编程,即本次作业 TinyNTP 任务。在 TinyNTP 中,服务器端采用多线程方式响应请求,使用 Lambda 表达式绑定 gettime 请求，同时建立认证机制，只有遵循特定握手协议的客户端才能访问到数据;在客户端，使用 Qt 绘制一个数字时钟界面，通过 RPC 请求获得服务器时间，同时根据请求响应的时间矫正获取的时间，然后将数字时钟显示出来。最后,对 TinyRPC 的进一步优化作了展望.

关键词: RPC;NTP;RPCLIB;Qt;C++14
中图法分类号: TP311

TinyNTP Program Designing Based on TinyRPC Library

Luo Keyun[1]

[1](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

Abstract: RPCLIB is an RPC library designed with C++14, it uses msgpack as a serialization library, and Asio as a network communication library. It also provides client-side and server-side implementations. This work is based on RPCLIB code refactoring and necessary streamlining, which is called TinyRPC library, while the client library and server-side minimize coupling, it also provides a method of simplified cross-platform compilation. On top of the TinyRPC library, the TinyNTP task is for server-side and client-side programming. In TinyNTP task, the server-side uses multi-threaded way to respond to requests, and uses lambda expression to bind the gettime requests. And meanwhile the authentication mechanism is established so that only clients that follow a specific handshake protocol can access the data. On the client side, using Qt to draw a digital clock interface, obtaining server time via RPC request, the acquisition time is corrected according to the time of the request response at the same time, and then displaying the digital clock. Finally, the further optimization of TinyRPC is prospected.

Key words: RPC;NTP;RPCLIB;Qt;C++14

1 相关背景

1.1 RPC框架

RPC 采用 C/S 模式，客户端发送请求，服务端响应，底层网络协议为 TCP/IP。通常 RPC 除了进程间相互通信外还可以实现不同语言之间的相互通信

RPC(远程过程调用)在分布式系统中处于基础地位，核心思想是将不同进程之间的通讯抽象成简单的函数调用。基本的过程是客户端将参数序列化成数据流发送到客户端中，客户端从流中反序列化出参数后进行业务处理，得到结果后将其序列化给客户端。从这些过程可以看出一个比较完善的 RPC 框架需要提供或调用

* 分布式系统第一次作业：利用 RPC 设计一个获取时间的小程序

现成的序列化方法和底层的异步网络通信，对用户完全透明。

从调用类型来看，函数调用可以分为同步和异步模型。同步模型中，客户端发送请求后就一直等待结果，中间不能干其他事，也称为请求-响应模式，当然，这种模式存在很大的缺点，若网络状况比较差，则客户端会花费大量的时间在等待请求的响应上，同时只有客户端可以向服务器端发送请求，反之服务器端则不能向客户端发送时间通知，即缺少回调接口机制。针对这样的缺点，可以通过双向会话通信机制来改进，首先所有的方法定义的请求都是没有返回值的，客户端调用函数后不需要等待就可以做其他事，然后对服务器端定义一个回调接口用于向客户端发送请求结果和时间通知。双向会话容易产生结果混乱的问题，一个简单的想法就是在会话中添加 ID 标识，但完全手动处理 ID 实现起来很麻烦，由此引出了异步调用 RPC。异步调用 RPC 主要思想是在双向会话的基础上让客户端通过回调函数获得请求结果,这有可能导致响应的顺序被打乱，若要求强顺序一致性，那么服务器端需要同步调用或引入分布式消息队列进行排队有序返回结果。

由于 RPC 在分布式系统中的基础作用，目前各大公司都有自发研制相应的 RPC 库，这其中比较著名的有谷歌的 gRPC(基于 ProtoBuf 序列化，基于 HTTP/2 协议标准而设计)、FaceBook 的 Thrift(捐献给 Apache 基金会，可伸缩，使用软件堆栈代码生成引擎)，以及国内腾讯的 PhxRPC(基于 ProtoBuf 序列化，半同步半异步模型，支持动态自适应)、阿里的 Dubbo(基于 Java Interface，依托 Spring 框架，微服务体系)。从这些大企业的开源项目来看，它们大都使用第三方的序列化工具，利用代码生成技术减少编码量。

1.2 RPCLIB库

RPCLIB[1]是一个使用现代 C++设计、使用 MsgPack 序列化、利用 Asio（Boost）异步通信的跨平台的 RPC 库。RPCLIB 在设计过程中使用了一些 C++14 新特性，因此编译器需要至少满足如下条件：g++5.0/clang++3.7/MSVC 2015 Update3，RPCLIB 是第三方库自包含的，使用到了 asio 网络库、fmtlib 字符串格式化和日志输出、msgpack 序列化库。它同时提供了客户端和服务器端的实现，也可用作内部进程间的通信。RPCLIB 使用了 Msgpack 序列化库以及 Msgpack-RPC 库分配和编码对函数进程的调用。因此，从总体来说，RPCLIB 可实现服务器端的多线程运行、客户端的同步/异步调用、跨平台、无需代码生成等多种特性。

RPCLIB 在设计上使用了现代 C++编程技术，大量使用了模板元编程技术，在编译期对代码进行类型计算，并运用了一系列设计模式，主要有 pimpl(pointer to implementation)和 dispatcher。RPCLIB 在设计过程中，利用 pimpl 设计模式进行编译期封装，隐藏实现，最小化耦合，将接口分离,这样的好处就是能够最小化编译期依赖，将接口和实现分离，更加便携；同时使用集中分发设计模式，在调用一个方法时，方法中的参数被特殊对待并决定采用哪一个方法，具体来说，在 RPCLIB 中，利用元编程技术生成能够管理任意使用 msgpack 编码的消息的回调函数的包装器，然后将函数产生的结果编码成 msgpack 消息，生成的包装器具有统一的签名，它允许将它们存储在映射中，通过按名称查找正确的函数来执行调度。

2 系统分析与设计

2.1 设计目标分析

1. 服务器端产生时间并在本地显示，至少两个客户端获取服务器端时间，然后在客户端以数字时钟或模拟时钟显示服务器端的程序：该目标实际上是要求服务器端支持多线程模型，客户端使用同步通信模型，因此在 RPC 的服务端使用多个 worker 构成线程池。
2. 客户端与服务器端的通信时延：完成一次 RPC 调用考虑通信开销，因此在客户端调用服务器端前后使用一个计时器记录从调用到得到结果所需要的时间 t_0,然后将服务器端获取的时间 t 加上 $t_0/2$,即 $t=t+ t_0/2$。
3. 只有授权的客户端才能从服务器端获取时间：客户端和服务器端建立一个握手机制，这里简化成服务器端有一个 verify_id 存根，只有客户端传入的 verify_id 能模除服务器的 verify_id 才能得到时间，否则直接返回空字符串。

4. 服务器端可以运行在 Windows 系统或 Linux 系统之上：服务器端运行环境不受限制，这里由于实现了 TinyRPC 的跨平台编译，所以均可以运行。
5. 客户端系统至少能够垮 Windows 和 Linux 系统：客户端使用 Qt 编写，Qt 本身就可以跨平台，所以无须过多考虑。
6. 客户端和服务器端代码耦合度要低：将 TinyRPC 的客户端和服务器端的公共部分抽取出来编译成一个静态库，然后将客户端和服务器端分别与之链接，形成 libtinyrpc_client.a 和 libtinyrpc_server.a 库，然后进行 TinyNTP 开发时，只需要将客户端程序和服务器端程序与这两个库链接形成可执行文件即可，最大程度降低耦合。
7. 客户端和服务器端的运行时状态可调整：将服务器端和客户端的 IP、端口和握手信号写到配置文件中，运行时先读取配置文件中的值，建立客户端和服务器端。
8. 一些客户端能持续的向服务器端发送请求：客户端设计成每隔 200 毫秒向服务器端发送请求，直到手动停止。
9. 使用 RPC/RMI 作为通信机制：TinyRPC 使用 RPC 通信机制，底层使用 Asio 异步通信模型，使用 TCP 连接。

2.2 系统环境要求与整体结构

1. Linux (Ubuntu14.04, Ubuntu16.04，64 位):cmake3.4 以上，gcc/g++ 5.0 以上(支持 C++14 语法，Ubuntu14.04 需要升级 gcc/g++)， Qt5.2.1 及以上版本。
2. Windows (window10，64 位): cmake3.4 以上，Visual Studio 2015 Update3(支持 C++14 语法) 或 MinGW GCC/G++5， Qt5.2.1 及以上版本。
3. 程序结构：

图 1 TinyRPC 程序代码目录结构

4. 说明：该项目是一个 C++ CMake 项目，编写 CMakeLists 文件实现跨平台，编译时，可使用脚本 build_linux.sh 或 build_win.bat 实现一键编译。编译好的库存放在 lib 文件夹下，最终可执行文件存放在 bin 目录下，编译的中间文件夹为 build 目录，可删除。

2.3 TinyRPC库的设计

1. 总体结构：

TinyRPC 库包括 include 文件夹（第三方依赖：asio, msgpack）和 tinyrpc 文件夹(核心项目，配置文件解析以及 Log 日志库 fmtlib)，他们都统一在 tinyrpc 命名空间内。原始的 RPCLIB[2]代码结构按照头文件和实现文件分开存储，比较混乱。因此，针对原始项目，主要精简了针对不同架构的 cmake

文件，在 CMakeLists 文件中针对 Linux 和 Windows 平台将编译参数作相应调整，将所有依赖转移到 include 文件下，将 RPCLIB 中去掉了 this_session 类，重写了 server、client 和 pimpl.h 类，删除了 client_error 类，增添了配置文件处理函数。

2. 核心库

```
.
├─── client              #客户端，生成 libtinyrpc_client.a 库
├─── CMakeLists.txt      #CMakeLists 编写模块
├─── core                #通用核心模块，生成 libtinyrpc_core.a，libfmt.a 库
├─── detail              #内部隐藏类模块
└─── server              #服务器端，生成 libtinyrpc_server.a 库
```

3. 内部隐藏类模块

该模块主要定义了基本类型的模板元运算，以及日志输出等，并定义了 pimpl 宏定义和服务器端的会话，该模块对外部用户不可见。

4. 通用核心模块

该模块主要定义了日志输出、配置文件读取、dispatcher 任务调度器和请求响应等。其中 dispatcher 任务调度器定义了服务器端绑定指定函数功能，该绑定实现了 Lambda 表达式函数，类成员方法，普通函数等多种形式。

5. 服务器端模块

该模块提供了阻塞式 run，非阻塞式 run(多个工作线程),以及绑定的函数和返回错误信息。

6. 客户端模块

该模块主要定义了同步调用和异步调用，发送通知信息等。

2.4 服务器端设计

1. 读取配置文件，获得 ip，port 和握手信号 verified_id。
2. 建立一个服务器连接。
3. 绑定一个 gettime 函数，以 Lambda 的方式：获取本地时间，转换成字符串，本地输出，并返回时间：方法如下：

```
srv.bind("gettime", [=](int verify_id){
        if (verify_id % verified_id != 0){
            return std::string();
        }

        time_t current_time;
        struct tm *ret_time;
        time(&current_time);
        ret_time = localtime(&current_time);

        char* result = asctime(ret_time);
        std::string time(result);
        std::cout<< "Current Time:" << result << std::endl;

        return time.substr(11, 8);
    });
```

图 2　服务器端绑定的 gettime 函数

4. 阻塞运行或非阻塞运行，这里使用了非阻塞运行，每个线程处理一个请求，线程数量为 CPU 的核数。
5. 阻塞主线程。

2.5 客户端设计

1. 读取配置文件，获得 ip，port 和握手信号 verify_id。
2. 初始化数字时钟：根据 ip 地址，端口号建立一个 TCP 连接，设置一个定时器，每 200 毫秒发送一次请求，然后不停地循环。
3. 显示时间：使用 Qt 中的 LCD 中的 8 位数码管显示，每次从服务器请求时间时记录所消耗的时间，然后用得到的时间加上消耗的时间的一半来矫正网络时延，如果返回的字符串不为空则说明请求成功，显示时间，否则返回:

```
1. void DigitalClock::showTime()
2. {
3.     QTime time_base, current_time, flag;
4.     QString text = "88:88:88";
5.     flag.start();
6.     QString time = QString::fromStdString(this->client.call("gettime", this->verify_id).as<std::s
   tring>());
7.     int cost_time = flag.elapsed() / 2;
8.
9.     if (!time.isNull()) {
10.         time_base = QTime::fromString(time, "hh:mm:ss");
11.
12.         current_time = time_base.addMSecs(cost_time);
13.         text = current_time.toString("hh:mm:ss");
14.     }
15.
16.     display(text);
17. }
```

图 3　客户端时间显示

3 实现演示

1. 首先在 Linux 或 Windows 系统上编译程序，使用提供的脚本便可以一键编译，生成的可执行文件在 bin 目录，若要运行，则先切换到 bin 目录，然后在终端输入需要执行的文件，确保目录下有配置文件 server.conf 和 client.conf，若没有则使用默认值，即服务器地址为 0.0.0.0:20160,客户端地址为 127.0.0.1:20160。若端口或 IP 不匹配则客户端将无法接受输出。
2. 服务器端位于 Linux(Ubuntu14.04, 114.212.87.26:20160)，客户端位于 Windows(Windows10, 114.212.83.87:20160)，客户端有三个，服务器端握手信号为 2016，客户端为 2016,4032,6048，持续的向服务器端请求时间：

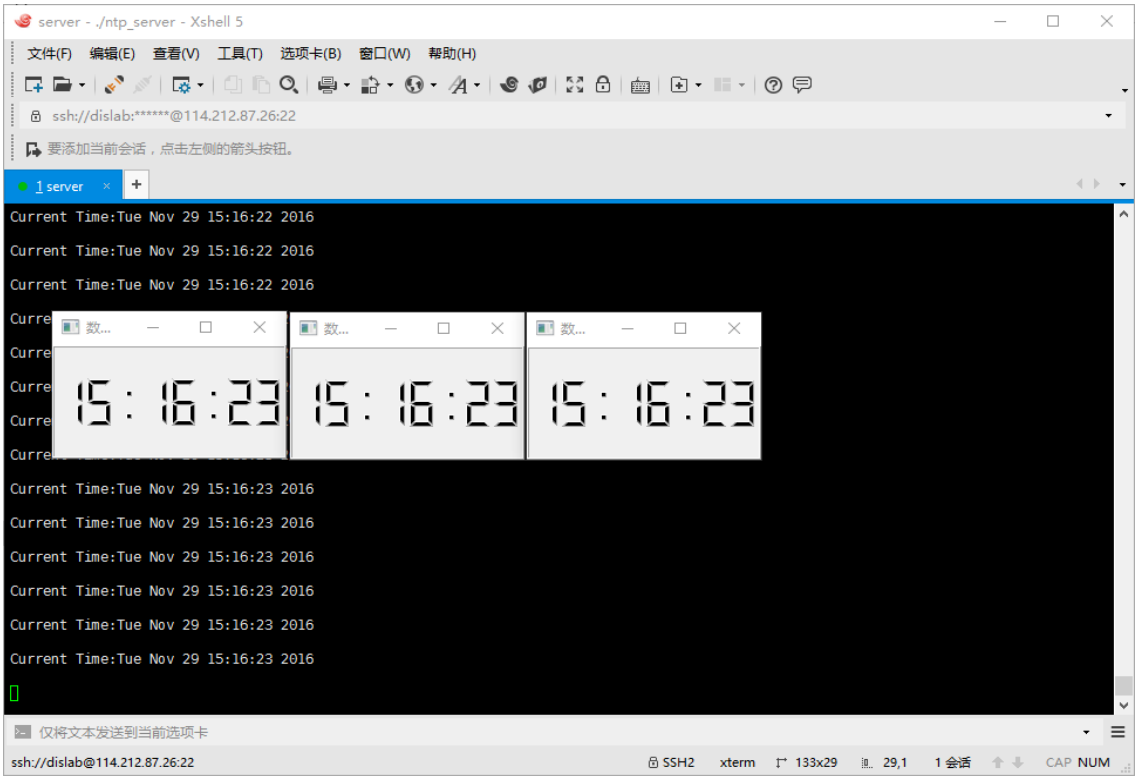

图 4　客户端 Windows，服务器端 Linux

3. 服务器端位于 Windows(Windows10, 111.195.215.157:20160，北京市教育网),客户端位于 Linux(Ubuntu16.04, 218.94.142.58:20160，南京市教育网), 客户端有三个，服务器端握手信号为 2016，客户端为 2016,4032,6049，持续的向服务器端请求时间：

服务器端：

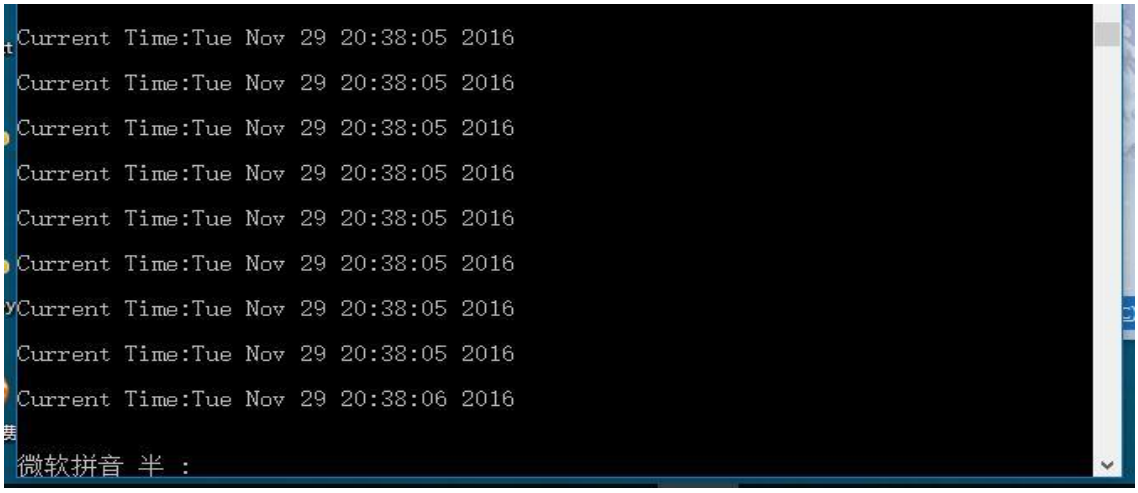

图 5　服务器端 Windows 显示

客户端：

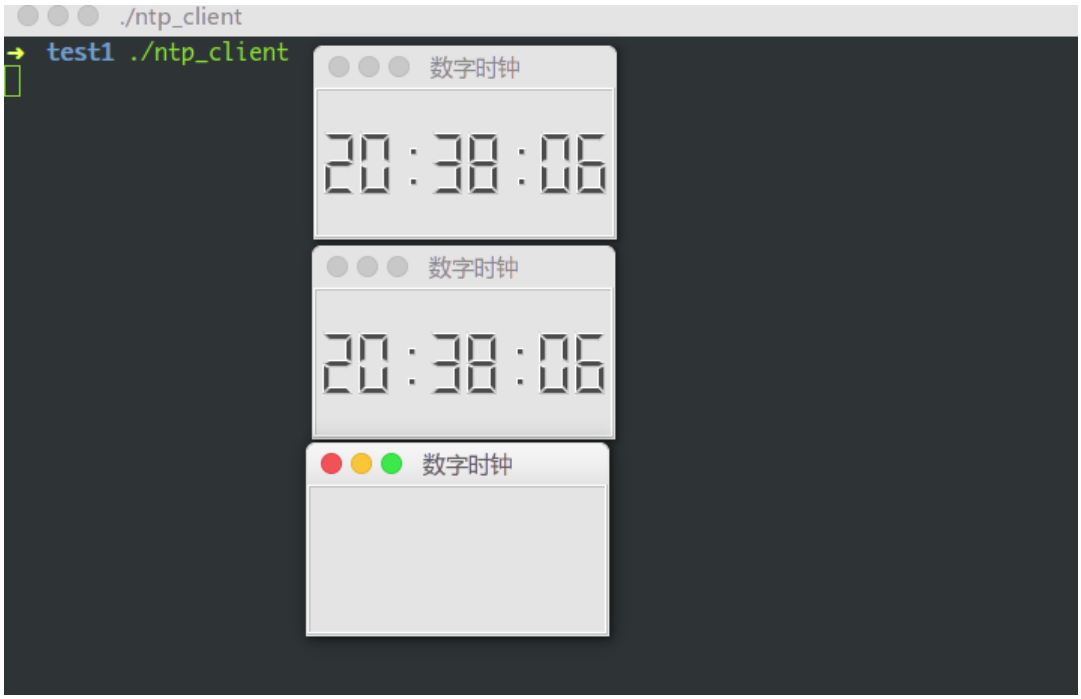

图 6　客户端端 Linux 显示

可以看出客户端 1,2 握手成功，得到时间，客户端 3 握手不成功，无法得到时间。

4　总结

RPC 跨机器间通信是分布式系统里面的一个基本问题，目前各大公司已经涌现出多种多样的 RPC 开发框架，本次作业对开源项目 RPCLIB 进行了裁剪和重构，对客户端和服务器端进行了解耦，并增添了读取配置文件模块，形成独立的跨平台的 RPC 服务器端库和客户端库，在这两个库之上构建了 TinyNTP 小程序，实现远程调用服务器端 Lambda 表达式函数获取时间并在本地显示，中间处理了诸多任务要求。

通过这次实践，我对 RPC 库的调用机制有了一个初步的了解，也发现一些问题，例如目前的授权机制是在应用层上的，没有在 TCP 底层拒绝连接未授权客户，这样会导致连接资源可能被恶意消耗，另外目前序列化框架和底层通信库还不能完全解耦，比如使用 ProtoBuf 序列化，使用 Libev,Libuv 等通信库替换 Msgpack，asio，这些都是以后需要进一步研究的。

References:

[1] RPCLIB. http://rpclib.net/

[2] rpclib source code.https://github.com/rpclib/rpclib

[3] Tanenbaum, Andrew S., and Maarten Van Steen. Distributed systems. Prentice-Hall, 2007.