2 Socket 编程 UDP

UDP 网络编程

V1 版本 - echo server

简单的回显服务器和客户端代码

备注: 代码中会用到 地址转换函数 . 参考接下来的章节.

nocopy.hpp

```C++
#pragma once
#include <iostream>

class nocopy
{
public:
    nocopy(){}
    nocopy(const nocopy &) = delete;
    const nocopy& operator = (const nocopy &) = delete;
    ~nocopy(){}
};
```

UdpServer.hpp

```C++
#pragma once

#include <iostream>
#include <string>
#include <cerrno>
#include <cstring>
#include <unistd.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "nocopy.hpp"
```

```
#include "Log.hpp"
#include "Comm.hpp"
#include "InetAddr.hpp"

const static uint16_t defaultport = 8888;
const static int defaultfd = -1;
const static int defaultsize = 1024;

class UdpServer : public nocopy
{
public:
    UdpServer(uint16_t port = defaultport)
        : _port(port), _sockfd(defaultfd)
    {
    }
    void Init()
    {
        // 1. 创建 socket，就是创建了文件细节
        _sockfd = socket(AF_INET, SOCK_DGRAM, 0);
        if (_sockfd < 0)
        {
            lg.LogMessage(Fatal, "socket errr, %d : %s\n", errno,
strerror(errno));
            exit(Socket_Err);
        }

        lg.LogMessage(Info, "socket success, sockfd: %d\n",
_sockfd);

        // 2. 绑定，指定网络信息
        struct sockaddr_in local;
        bzero(&local, sizeof(local)); // memset
        local.sin_family = AF_INET;
        local.sin_port = htons(_port);
        local.sin_addr.s_addr = INADDR_ANY; // 0

        // local.sin_addr.s_addr = inet_addr(_ip.c_str()); // 1. 4
字节 IP 2. 变成网络序列

        // 结构体填完，设置到内核中了吗？？没有
        int n = ::bind(_sockfd, (struct sockaddr *)&local,
sizeof(local));
        if (n != 0)
        {
```

```
            lg.LogMessage(Fatal, "bind errr, %d : %s\n", errno,
strerror(errno));
            exit(Bind_Err);
        }
    }
    void Start()
    {
        // 服务器永远不退出
        char buffer[defaultsize];
        for (;;)
        {
            struct sockaddr_in peer;
            socklen_t len = sizeof(peer); // 不能乱写
            ssize_t n = recvfrom(_sockfd, buffer, sizeof(buffer) -
1, 0, (struct sockaddr *)&peer, &len);
            if (n > 0)
            {
                InetAddr addr(peer);
                buffer[n] = 0;
                std::cout << "[" << addr.PrintDebug() << "]# " <<
buffer << std::endl;
                sendto(_sockfd, buffer, strlen(buffer), 0, (struct
sockaddr *)&peer, len);
            }
        }
    }
    ~UdpServer()
    {
    }

private:
    // std::string _ip; // 后面要调整
    uint16_t _port;
    int _sockfd;
};
```

InetAddr.hpp

```
C++
#pragma once
#include <iostream>
#include <string>
#include <sys/types.h>
#include <sys/socket.h>
```

```
#include <netinet/in.h>
#include <arpa/inet.h>

class InetAddr
{
public:
    InetAddr(struct sockaddr_in &addr):_addr(addr)
    {
        _port = ntohs(_addr.sin_port);
        _ip = inet_ntoa(_addr.sin_addr);
    }
    std::string Ip() {return _ip;}
    uint16_t Port() {return _port;};
    std::string PrintDebug()
    {
        std::string info = _ip;
        info += ":";
        info += std::to_string(_port);  // "127.0.0.1:4444"
        return info;
    }
    ~InetAddr(){}
private:
    std::string _ip;
    uint16_t _port;
    struct sockaddr_in _addr;
};
```

Comm.hpp

```
C++
#pragma once

enum{
    Usage_Err = 1,
    Socket_Err,
    Bind_Err
};
```

- `Log.hpp` 已经有了，这里就不再复制粘贴了
- 云服务器不允许直接 bind 公有 IP，我们也不推荐编写服务器的时候，bind 明确的 IP，推荐直接写成 INADDR_ANY

```
C++
/* Address to accept any incoming messages.  */
```

```
#define INADDR_ANY    ((in_addr_t) 0x00000000)
```

在网络编程中，当一个进程需要绑定一个网络端口以进行通信时，可以使用 INADDR_ANY 作为 IP 地址参数。这样做意味着该端口可以接受来自任何 IP 地址的连接请求，无论是本地主机还是远程主机。例如，如果服务器有多个网卡（每个网卡上有不同的 IP 地址），使用 INADDR_ANY 可以省去确定数据是从服务器上具体哪个网卡/IP 地址上面获取的。

UdpClient.hpp

```
C++
#include <iostream>
#include <cerrno>
#include <cstring>
#include <string>
#include <unistd.h>
#include <sys/types.h> /* See NOTES */
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>

void Usage(const std::string &process)
{
    std::cout << "Usage: " << process << " server_ip server_port"
<< std::endl;
}

// ./udp_client server_ip server_port
int main(int argc, char *argv[])
{
    if (argc != 3)
    {
        Usage(argv[0]);
        return 1;
    }

    std::string serverip = argv[1];
    uint16_t serverport = std::stoi(argv[2]);

    // 1. 创建 socket
    int sock = socket(AF_INET, SOCK_DGRAM, 0);
    if (sock < 0)
    {
        std::cerr << "socket error: " << strerror(errno) <<
```

```
std::endl;
        return 2;
    }
    std::cout << "create socket success: " << sock << std::endl;

    // 2. client 要不要进行 bind? 一定要 bind 的！！
    // 但是，不需要显示 bind，client 会在首次发送数据的时候会自动进行 bind
    // 为什么？server 端的端口号，一定是众所周知，不可改变的，client 需要 port，bind 随机端口.
    // 为什么？client 会非常多.
    // client 需要 bind，但是不需要显示 bind，让本地 OS 自动随机 bind，选择随机端口号
    // 2.1 填充一下 server 信息
    struct sockaddr_in server;
    memset(&server, 0, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_port = htons(serverport);
    server.sin_addr.s_addr = inet_addr(serverip.c_str());

    while (true)
    {
        // 我们要发的数据
        std::string inbuffer;
        std::cout << "Please Enter# ";
        std::getline(std::cin, inbuffer);
        // 我们要发给谁呀？server
        ssize_t n = sendto(sock, inbuffer.c_str(), inbuffer.size(), 0, (struct sockaddr*)&server, sizeof(server));
        if(n > 0)
        {
            char buffer[1024];
            //收消息
            struct sockaddr_in temp;
            socklen_t len = sizeof(temp);
            ssize_t m = recvfrom(sock, buffer, sizeof(buffer)-1, 0, (struct sockaddr*)&temp, &len); // 一般建议都是要填的.
            if(m > 0)
            {
                buffer[m] = 0;
                std::cout << "server echo# " << buffer << std::endl;
            }
```

```
            else
                break;
        }
        else
            break;
    }

    close(sock);
    return 0;
}
```

- client 端要不要显示 bind 的问题

V2 版本 - DictServer

实现一个简单的英译汉的网络字典

dict.txt

```
C++
apple: 苹果
banana: 香蕉
cat: 猫
dog: 狗
book: 书
pen: 笔
happy: 快乐的
sad: 悲伤的
run: 跑
jump: 跳
teacher: 老师
student: 学生
car: 汽车
bus: 公交车
love: 爱
hate: 恨
hello: 你好
goodbye: 再见
summer: 夏天
winter: 冬天
```

Dict.hpp

```C++
#pragma once

#include <iostream>
#include <string>
#include <fstream>
#include <unordered_map>

const std::string sep = ": ";

class Dict
{
private:
    void LoadDict()
    {
        std::ifstream in(_confpath);
        if(!in.is_open())
        {
            std::cerr << "open file error" << std::endl; // 后面可以用日志替代打印
            return;
        }
        std::string line;
        while(std::getline(in, line))
        {
            if(line.empty()) continue;
            auto pos = line.find(sep);
            if(pos == std::string::npos) continue;
            std::string key = line.substr(0, pos);
            std::string value = line.substr(pos + sep.size());
            _dict.insert(std::make_pair(key, value));
        }
        in.close();
    }
public:
    Dict(const std::string &confpath):_confpath(confpath)
    {
        LoadDict();
    }
    std::string Translate(const std::string &key)
    {
        auto iter = _dict.find(key);
        if(iter == _dict.end()) return std::string("Unknown");
        else return iter->second;
```

```
    }
    ~Dict()
    {}
private:
    std::string _confpath;
    std::unordered_map<std::string, std::string> _dict;
};
```

UdpServer.hpp

```
C++
#pragma once

#include <iostream>
#include <string>
#include <cerrno>
#include <cstring>
#include <unistd.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unordered_map>
#include <functional>
#include "nocopy.hpp"
#include "Log.hpp"
#include "Comm.hpp"
#include "InetAddr.hpp"

const static uint16_t defaultport = 8888;
const static int defaultfd = -1;
const static int defaultsize = 1024;

using func_t = std::function<void(const std::string &req,
std::string *resp)>;

class UdpServer : public nocopy
{
public:
    UdpServer(func_t func, uint16_t port = defaultport)
        : _func(func),_port(port),_sockfd(defaultfd)
    {
    }
```

```
    void Init()
    {
        // 1. 创建 socket，就是创建了文件细节
        _sockfd = socket(AF_INET, SOCK_DGRAM, 0);
        if (_sockfd < 0)
        {
            lg.LogMessage(Fatal, "socket errr, %d : %s\n", errno,
strerror(errno));
            exit(Socket_Err);
        }

        lg.LogMessage(Info, "socket success, sockfd: %d\n",
_sockfd);

        // 2. 绑定，指定网络信息
        struct sockaddr_in local;
        bzero(&local, sizeof(local)); // memset
        local.sin_family = AF_INET;
        local.sin_port = htons(_port);
        local.sin_addr.s_addr = INADDR_ANY; // 0

        // local.sin_addr.s_addr = inet_addr(_ip.c_str()); // 1. 4
字节 IP 2. 变成网络序列

        // 结构体填完，设置到内核中了吗？？没有
        int n = ::bind(_sockfd, (struct sockaddr *)&local,
sizeof(local));
        if (n != 0)
        {
            lg.LogMessage(Fatal, "bind errr, %d : %s\n", errno,
strerror(errno));
            exit(Bind_Err);
        }
    }
    void Start()
    {
        // 服务器永远不退出
        char buffer[defaultsize];
        for (;;)
        {
            struct sockaddr_in peer;
            socklen_t len = sizeof(peer); // 不能乱写
            ssize_t n = recvfrom(_sockfd, buffer, sizeof(buffer) -
1, 0, (struct sockaddr *)&peer, &len);
```

```
            if (n > 0)
            {
                InetAddr addr(peer);
                buffer[n] = 0;
                std::cout << "[" << addr.PrintDebug() << "]# " <<
buffer << std::endl;
                std::string value;
                _func(buffer, &value); // 回调业务翻译方法
                sendto(_sockfd, value.c_str(), value.size(), 0,
(struct sockaddr *)&peer, len);
            }
        }
    }
    ~UdpServer()
    {
    }

private:
    // std::string _ip; // 后面要调整
    uint16_t _port;
    int _sockfd;
    func_t _func;
};
```

Main.cc

```
C++
#include "UdpServer.hpp"
#include "Comm.hpp"
#include "Dict.hpp"
#include <memory>

void Usage(std::string proc)
{
    std::cout << "Usage : \n\t" << proc << " local_port\n" <<
std::endl;
}

Dict gdict("./dict.txt");

void Execute(const std::string &req, std::string *resp)
{
    *resp = gdict.Translate(req);
}
```

```
// ./udp_server 8888
int main(int argc, char *argv[])
{
    if(argc != 2)
    {
        Usage(argv[0]);
        return Usage_Err;
    }

    // std::string ip = argv[1];
    uint16_t port = std::stoi(argv[1]);
    std::unique_ptr<UdpServer> usvr =
std::make_unique<UdpServer>(Execute, port);
    usvr->Init();
    usvr->Start();

    return 0;
}
```

```
whb@bite01:~/code/109/lesson46/1.udp_server_server$ ./udp_client 127.0.0.1 8888
create socket success: 3
Please Enter# apple
server echo# 苹果
Please Enter# goodbye
server echo# 再见
Please Enter# haaaa
server echo# Unknown
```

```
whb@bite01:~/code/109/lesson46/1.udp_server_server$ ./udp_server 8888
[Info][2024-6-12 16:48:41][2907106] socket success, sockfd: 3
[127.0.0.1:53965]# apple
[127.0.0.1:53965]# goodbye
[127.0.0.1:53965]# haaaa
^C
whb@bite01:~/code/109/lesson46/1.udp_server_server$
```

V2 版本 - DictServer 封装版

下面是一个封装版的，大家下来可以看一下

udp_socket.hpp

```
C
#pragma once
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <cassert>
#include <string>

#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
typedef struct sockaddr sockaddr;
typedef struct sockaddr_in sockaddr_in;
```

```
class UdpSocket {
public:
  UdpSocket() : fd_(-1) {
  }

  bool Socket() {
    fd_ = socket(AF_INET, SOCK_DGRAM, 0);
    if (fd_ < 0) {
      perror("socket");
      return false;
    }
    return true;
  }

  bool Close() {
    close(fd_);
    return true;
  }

  bool Bind(const std::string& ip, uint16_t port) {
    sockaddr_in addr;
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = inet_addr(ip.c_str());
    addr.sin_port = htons(port);
    int ret = bind(fd_, (sockaddr*)&addr, sizeof(addr));
    if (ret < 0) {
      perror("bind");
      return false;
    }
    return true;
  }

  bool RecvFrom(std::string* buf, std::string* ip = NULL,
uint16_t* port = NULL) {
    char tmp[1024 * 10] = {0};
    sockaddr_in peer;
    socklen_t len = sizeof(peer);
    ssize_t read_size = recvfrom(fd_, tmp,
                                 sizeof(tmp) - 1, 0,
(sockaddr*)&peer, &len);
    if (read_size < 0) {
      perror("recvfrom");
      return false;
    }
```

```
    // 将读到的缓冲区内容放到输出参数中
    buf->assign(tmp, read_size);
    if (ip != NULL) {
      *ip = inet_ntoa(peer.sin_addr);
    }
    if (port != NULL) {
      *port = ntohs(peer.sin_port);
    }
    return true;
  }

  bool SendTo(const std::string& buf, const std::string& ip,
uint16_t port) {
    sockaddr_in addr;
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = inet_addr(ip.c_str());
    addr.sin_port = htons(port);
    ssize_t write_size = sendto(fd_, buf.data(), buf.size(), 0,
(sockaddr*)&addr, sizeof(addr));
    if (write_size < 0) {
      perror("sendto");
      return false;
    }
    return true;
  }

private:
  int fd_;
};
```

UDP 通用服务器

udp_server.hpp

```
C
#pragma once
#include "udp_socket.hpp"

// C 式写法
// typedef void (*Handler)(const std::string& req, std::string*
resp);
// C++ 11 式写法，能够兼容函数指针，仿函数，和 lambda
#include <functional>
typedef std::function<void (const std::string&, std::string*
resp)> Handler;
```

```
class UdpServer {
public:
  UdpServer() {
    assert(sock_.Socket());
  }

  ~UdpServer() {
    sock_.Close();
  }

  bool Start(const std::string& ip, uint16_t port, Handler
handler) {
    // 1. 创建 socket
    // 2. 绑定端口号
    bool ret = sock_.Bind(ip, port);
    if (!ret) {
      return false;
    }
    // 3. 进入事件循环
    for (;;) {
      // 4. 尝试读取请求
      std::string req;
      std::string remote_ip;
      uint16_t remote_port = 0;
      bool ret = sock_.RecvFrom(&req, &remote_ip, &remote_port);
      if (!ret) {
        continue;
      }
      std::string resp;
      // 5. 根据请求计算响应
      handler(req, &resp);
      // 6. 返回响应给客户端
      sock_.SendTo(resp, remote_ip, remote_port);
      printf("[%s:%d] req: %s, resp: %s\n", remote_ip.c_str(),
remote_port,
             req.c_str(), resp.c_str());
    }
    sock_.Close();
    return true;
  }

private:
  UdpSocket sock_;
```

```
};
```

实现英译汉服务器

以上代码是对 udp 服务器进行通用接口的封装. 基于以上封装, 实现一个查字典的服务器就很容易了.

dict_server.cc

```
C
#include "udp_server.hpp"
#include <unordered_map>
#include <iostream>

std::unordered_map<std::string, std::string> g_dict;

void Translate(const std::string& req, std::string* resp) {
  auto it = g_dict.find(req);
  if (it == g_dict.end()) {
    *resp = "未查到!";
    return;
  }
  *resp = it->second;
}

int main(int argc, char* argv[]) {
  if (argc != 3) {
    printf("Usage ./dict_server [ip] [port]\n");
    return 1;
  }
  // 1. 数据初始化
  g_dict.insert(std::make_pair("hello", "你好"));
  g_dict.insert(std::make_pair("world", "世界"));
  g_dict.insert(std::make_pair("c++", "最好的编程语言"));
  g_dict.insert(std::make_pair("bit", "特别 NB"));
  // 2. 启动服务器
  UdpServer server;
  server.Start(argv[1], atoi(argv[2]), Translate);
  return 0;
}
```

UDP 通用客户端

udp_client.hpp

```C
#pragma once
#include "udp_socket.hpp"

class UdpClient {
public:
  UdpClient(const std::string& ip, uint16_t port) : ip_(ip),
port_(port) {
    assert(sock_.Socket());
  }

  ~UdpClient() {
    sock_.Close();
  }

  bool RecvFrom(std::string* buf) {
    return sock_.RecvFrom(buf);
  }

  bool SendTo(const std::string& buf) {
    return sock_.SendTo(buf, ip_, port_);
  }
private:
  UdpSocket sock_;
  // 服务器端的 IP 和 端口号
  std::string ip_;
  uint16_t port_;
};
```

实现英译汉客户端

```C
#include "udp_client.hpp"
#include <iostream>

int main(int argc, char* argv[]) {
  if (argc != 3) {
    printf("Usage ./dict_client [ip] [port]\n");
    return 1;
  }
  UdpClient client(argv[1], atoi(argv[2]));
  for (;;) {
    std::string word;
    std::cout << "请输入您要查的单词: ";
```

```
    std::cin >> word;
    if (!std::cin) {
      std::cout << "Good Bye" << std::endl;
      break;
    }
    client.SendTo(word);
    std::string result;
    client.RecvFrom(&result);
    std::cout << word << " 意思是 " << result << std::endl;
  }
  return 0;
}
```

V3 版本 - 简单聊天室

UdpServer.hpp

```
C++
#pragma once

#include <iostream>
#include <string>
#include <cerrno>
#include <cstring>
#include <unistd.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <functional>
#include <pthread.h>
// #include <mutex>
// #include <condition_variable>
#include "nocopy.hpp"
#include "Log.hpp"
#include "Comm.hpp"
#include "InetAddr.hpp"
#include "ThreadPool.hpp"

const static uint16_t defaultport = 8888;
const static int defaultfd = -1;
const static int defaultsize = 1024;
using task_t = std::function<void()>;
```

```
// using cb_t = std::function<std::string(std::string)>; // 定义了一个函数类型

// 聚焦在 IO 上
class UdpServer : public nocopy
{
public:
    // UdpServer(cb_t OnMessage, uint16_t port = defaultport)
    //     : _port(port), _sockfd(defaultfd), _OnMessage(OnMessage)
    UdpServer(uint16_t port = defaultport) : _port(port), _sockfd(defaultfd)
    {
        pthread_mutex_init(&_user_mutex, nullptr);
    }
    void Init()
    {
        // 1. 创建 socket，就是创建了文件细节
        _sockfd = socket(AF_INET, SOCK_DGRAM, 0);
        if (_sockfd < 0)
        {
            lg.LogMessage(Fatal, "socket errr, %d : %s\n", errno, strerror(errno));
            exit(Socket_Err);
        }

        lg.LogMessage(Info, "socket success, sockfd: %d\n", _sockfd);

        // 2. 绑定，指定网络信息
        struct sockaddr_in local;
        bzero(&local, sizeof(local)); // memset
        local.sin_family = AF_INET;
        local.sin_port = htons(_port);
        local.sin_addr.s_addr = INADDR_ANY; // 0

        // local.sin_addr.s_addr = inet_addr(_ip.c_str()); // 1. 4 字节 IP 2. 变成网络序列

        // 结构体填完，设置到内核中了吗？？没有
        int n = ::bind(_sockfd, (struct sockaddr *)&local, sizeof(local));
        if (n != 0)
```

```
        {
            lg.LogMessage(Fatal, "bind errr, %d : %s\n", errno,
strerror(errno));
            exit(Bind_Err);
        }

        ThreadPool<task_t>::GetInstance()->Start();
    }

    void AddOnlineUser(InetAddr addr)
    {
        LockGuard lockguard(&_user_mutex);
        for (auto &user : _online_user)
        {
            if (addr == user)
                return;
        }
        _online_user.push_back(addr);
        lg.LogMessage(Debug, "%s:%d is add to onlineuser
list...\n", addr.Ip().c_str(), addr.Port());
    }
    void Route(int sock, const std::string &message)
    {
        LockGuard lockguard(&_user_mutex);
        for (auto &user : _online_user)
        {
            sendto(sock, message.c_str(), message.size(), 0,
(struct sockaddr *)&user.GetAddr(), sizeof(user.GetAddr()));
            lg.LogMessage(Debug, "server send message to %s:%d,
message: %s\n", user.Ip().c_str(), user.Port(), message.c_str());
        }
    }
    void Start()
    {
        // 服务器永远不退出
        char buffer[defaultsize];
        for (;;)
        {
            struct sockaddr_in peer;
            socklen_t len = sizeof(peer); // 不能乱写
            ssize_t n = recvfrom(_sockfd, buffer, sizeof(buffer) -
1, 0, (struct sockaddr *)&peer, &len);
            if (n > 0)
            {
```

```
                InetAddr addr(peer);
                AddOnlineUser(addr);
                buffer[n] = 0;
                std::string message = "[";
                message += addr.Ip();
                message += ":";
                message += std::to_string(addr.Port());
                message += "]# ";
                message += buffer;
                task_t task = std::bind(&UdpServer::Route, this,
_sockfd, message);
                ThreadPool<task_t>::GetInstance()->Push(task);

                // 处理消息
                //  std::string response = _OnMessage(buffer);

                // std::cout << "[" << addr.PrintDebug() << "]# "
<< buffer << std::endl;
                // sendto(_sockfd, response.c_str(),
response.size(), 0, (struct sockaddr *)&peer, len);
            }
        }
    }
    ~UdpServer()
    {
        pthread_mutex_destroy(&_user_mutex);
    }

private:
    // std::string _ip; // 后面要调整
    uint16_t _port;
    int _sockfd;

    std::vector<InetAddr> _online_user; // 会被多个线程同时访问的
    pthread_mutex_t _user_mutex;
    // cb_t _OnMessage;   // 回调
};
```

- 引入线程池，这里就不重复贴代码了

InetAddr.hpp

```
C++
#pragma once
#include <iostream>
```

```
#include <string>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

class InetAddr
{
public:
    InetAddr(struct sockaddr_in &addr):_addr(addr)
    {
        _port = ntohs(_addr.sin_port);
        _ip = inet_ntoa(_addr.sin_addr);
    }
    std::string Ip() {return _ip;}
    uint16_t Port() {return _port;};
    std::string PrintDebug()
    {
        std::string info = _ip;
        info += ":";
        info += std::to_string(_port);  // "127.0.0.1:4444"
        return info;
    }
    const  struct sockaddr_in& GetAddr()
    {
        return _addr;
    }
    bool operator == (const InetAddr&addr)
    {
        //other code
        return this->_ip == addr._ip && this->_port == addr._port;
    }
    ~InetAddr(){}
private:
    std::string _ip;
    uint16_t _port;
    struct sockaddr_in _addr;
};
```

- 在 InetAddr 中，重载一下==方便对用户是否是同一个进行比较

UdpClient.hpp

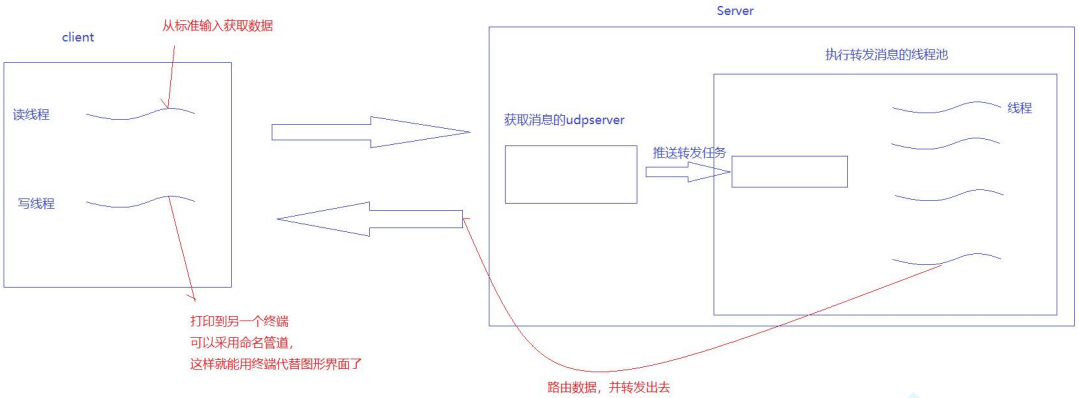
```
C++
#include <iostream>
```

```
#include <cerrno>
#include <cstring>
#include <string>
#include <unistd.h>
#include <sys/types.h> /* See NOTES */
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include "Thread.hpp"
#include "InetAddr.hpp"

void Usage(const std::string &process)
{
    std::cout << "Usage: " << process << " server_ip server_port"
<< std::endl;
}

class ThreadData
{
public:
    ThreadData(int sock, struct sockaddr_in &server) :
_sockfd(sock), _serveraddr(server)
    {
    }
    ~ThreadData()
    {
    }

public:
    int _sockfd;
    InetAddr _serveraddr;
};

void RecverRoutine(ThreadData &td)
{
    char buffer[4096];
    while (true)
    {
        struct sockaddr_in temp;
        socklen_t len = sizeof(temp);
        ssize_t n = recvfrom(td._sockfd, buffer, sizeof(buffer) -
1, 0, (struct sockaddr *)&temp, &len); // 一般建议都是要填的.
        if (n > 0)
        {
```

```
            buffer[n] = 0;
            std::cerr << buffer << std::endl; // 方便一会查看效果
        }
        else
            break;
    }
}

// 该线程只负责发消息
void SenderRoutine(ThreadData &td)
{
    while (true)
    {
        // 我们要发的数据
        std::string inbuffer;
        std::cout << "Please Enter# ";
        std::getline(std::cin, inbuffer);

        auto server = td._serveraddr.GetAddr();
        // 我们要发给谁呀？server
        ssize_t n = sendto(td._sockfd, inbuffer.c_str(),
inbuffer.size(), 0, (struct sockaddr *)&server, sizeof(server));
        if (n <= 0)
            std::cout << "send error" << std::endl;
    }
}

// ./udp_client server_ip server_port
int main(int argc, char *argv[])
{
    if (argc != 3)
    {
        Usage(argv[0]);
        return 1;
    }

    std::string serverip = argv[1];
    uint16_t serverport = std::stoi(argv[2]);

    // 1. 创建 socket
    // udp 是全双工的。既可以读，也可以写，可以同时读写，不会多线程读写
的问题
    int sock = socket(AF_INET, SOCK_DGRAM, 0);
    if (sock < 0)
```

```
    {
        std::cerr << "socket error: " << strerror(errno) <<
std::endl;
        return 2;
    }
    std::cout << "create socket success: " << sock << std::endl;

    // 2. client 要不要进行 bind? 一定要 bind 的！！但是，不需要显示
bind，client 会在首次发送数据的时候会自动进行 bind
    // 为什么？server 端的端口号，一定是众所周知，不可改变的，client 需
要 port，bind 随机端口.
    // 为什么？client 会非常多.
    // client 需要 bind，但是不需要显示 bind，让本地 OS 自动随机 bind，
选择随机端口号
    // 2.1 填充一下 server 信息
    struct sockaddr_in server;
    memset(&server, 0, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_port = htons(serverport);
    server.sin_addr.s_addr = inet_addr(serverip.c_str());

    ThreadData td(sock, server);
    Thread<ThreadData> recver("recver", RecverRoutine, td);
    Thread<ThreadData> sender("sender", SenderRoutine, td);
    recver.Start();
    sender.Start();

    recver.Join();
    sender.Join();

    close(sock);
    return 0;
}
```

- UDP 协议支持全双工，一个 sockfd，既可以读取，又可以写入，对于客户端和服务端同样如此
- 多线程客户端，同时读取和写入
- 测试的时候，使用管道进行演示



补充参考内容

地址转换函数

本节只介绍基于 IPv4 的 socket 网络编程,sockaddr_in 中的成员 struct in_addr sin_addr 表示 32 位 的 IP 地址

但是我们通常用点分十进制的字符串表示 IP 地址,以下函数可以在字符串表示 和 in_addr 表示之间转换;

字符串转 in_addr 的函数:

```
#include <arpa/inet.h>

int inet_aton(const char *strptr, struct in_addr *addrptr);
in_addr_t inet_addr(const char *strptr);
int inet_pton(int family, const char *strptr, void *addrptr);
```

in_addr 转字符串的函数:

```
char *inet_ntoa(struct in_addr inaddr);
const char *inet_ntop(int family, const void *addrptr, char *strptr,
size_t len);
```

其中 inet_pton 和 inet_ntop 不仅可以转换 IPv4 的 in_addr,还可以转换 IPv6 的 in6_addr,因此函数接口是 void *addrptr。

代码示例:

```c
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int main() {
  struct sockaddr_in addr;
  inet_aton("127.0.0.1", &addr.sin_addr);
  uint32_t* ptr = (uint32_t*)(&addr.sin_addr);
  printf("addr: %x\n", *ptr);
  printf("addr_str: %s\n", inet_ntoa(addr.sin_addr));
  return 0;
}
```

关于 inet_ntoa

inet_ntoa 这个函数返回了一个 char*, 很显然是这个函数自己在内部为我们申请了一块内存来保存 ip 的结果. 那么是否需要调用者手动释放呢?

```
The inet_ntoa() function converts the Internet host address in, given in network byte order, to a
string in IPv4 dotted-decimal notation. The string is returned in a statically allocated buffer, which
subsequent calls will overwrite.
```

man 手册上说, inet_ntoa 函数, 是把这个返回结果放到了静态存储区. 这个时候不需要我们手动进行释放.

那么问题来了, 如果我们调用多次这个函数, 会有什么样的效果呢? 参见如下代码:

```c
#include <stdio.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int main() {
  struct sockaddr_in addr1;
  struct sockaddr_in addr2;
  addr1.sin_addr.s_addr = 0;
  addr2.sin_addr.s_addr = 0xffffffff;
  char* ptr1 = inet_ntoa(addr1.sin_addr);
  char* ptr2 = inet_ntoa(addr2.sin_addr);
  printf("ptr1: %s, ptr2: %s\n", ptr1, ptr2);
  return 0;
}
```

运行结果如下:

```
[tangzhong@tz addr_convert]$ ./a.out
ptr1: 255.255.255.255, ptr2: 255.255.255.255
```

因为 inet_ntoa 把结果放到自己内部的一个静态存储区, 这样第二次调用时的结果会覆盖掉上一次的结果.

- 思考: 如果有多个线程调用 inet_ntoa, 是否会出现异常情况呢?
- 在 APUE 中, 明确提出 inet_ntoa 不是线程安全的函数;
- 但是在 centos7 上测试, 并没有出现问题, 可能内部的实现加了互斥锁;
- 同学们课后自己写程序验证一下在自己的机器上 inet_ntoa 是否会出现多线程的问题;
- 在多线程环境下, 推荐使用 inet_ntop, 这个函数由调用者提供一个缓冲区保存结果, 可以规避线程安全问题;

多线程调用 inet_ntoa 代码示例如下(同学们课后自己测试):

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <pthread.h>

void* Func1(void* p) {
  struct sockaddr_in* addr = (struct sockaddr_in*)p;
  while (1) {
    char* ptr = inet_ntoa(addr->sin_addr);
    printf("addr1: %s\n", ptr);
  }
  return NULL;
}

void* Func2(void* p) {
  struct sockaddr_in* addr = (struct sockaddr_in*)p;
  while (1) {
    char* ptr = inet_ntoa(addr->sin_addr);
    printf("addr2: %s\n", ptr);
  }
  return NULL;
}

int main() {
  pthread_t tid1 = 0;
  struct sockaddr_in addr1;
  struct sockaddr_in addr2;
  addr1.sin_addr.s_addr = 0;
  addr2.sin_addr.s_addr = 0xffffffff;
  pthread_create(&tid1, NULL, Func1, &addr1);
  pthread_t tid2 = 0;
  pthread_create(&tid2, NULL, Func2, &addr2);
  pthread_join(tid1, NULL);
  pthread_join(tid2, NULL);
  return 0;
}
```