[Asio实现浅析 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/55503053?from_voters_page=true)

# ASIO层次

* **namespace asio中：**

asio::io\_context :

1. io\_context类相当于应用程序与系统之间I/O操作的中介；
2. 所有“接收和发送”数据，都是通过将需求提交给io\_context,然后io\_context类在交给计算机来执行的；

asio::buffer函数:构造一个buffer结构

struct{

void\* buf;

s\_size len;

}

* **namespace asio - namespace ip 中**

tcp/udp其socket类:

ip::**tcp**::socket

ip::**udp**::socket

tcp/udp其端口类

ip::**tcp**::endpoint

ip::**udp**::endpoint

tcp/udp地址处理类：

ip::address

tcp接受客户端连接的类（acceptor）

ip::tcp::acceptor

# Tcp编程

## 同步示例

**服务器：**一个只接受一个客户端连接的回声服务器

#include<iostream>

#include"boost/asio.hpp"

using namespace std;

using namespace boost;

using asio::ip::tcp;

int main() {

std::cout << "server start ……" << std::endl;

asio::io\_context io;

//指定ipv4协议，绑定ip及端口

asio::ip::tcp::acceptor acptr(io, tcp::endpoint(tcp::v4(), 6688) );

tcp::socket sock(io);

acptr.accept(sock);

cout << "client:" << sock.remote\_endpoint().address() << endl;

try {

while (true) {

char buf[0xFF];

sock.receive(asio::buffer(buf));

sock.send(asio::buffer(buf));

}

}

catch(std::exception& e) {

cout << e.what();

}

sock.close();

::system("pause");

}

asio::ip::tcp::acceptor类：

封装了一个监听的socket，通过第二个参数绑定服务器IP及端口，用来监听同网段的IP连接；

asio::ip::tcp::endpoint类：

endpoint(tcp::v4(), 6688): 监听ipv4类型的本地所有地址（判断是否有连接请求）及端口

acceptor.accept(sock)函数：

接收客户端连接，参数：用来保存连接上来的客户端信息

sock.receive(asio::buffer)/sock.send(asio::buffer)函数：

进行同步接受与发送消息

**客户端**

#include<iostream>

#include"boost/asio.hpp"

using namespace std;

using namespace boost::asio;

int main()

{

io\_context io;

ip::tcp::socket sock(io);

sock.connect(ip::tcp::endpoint(ip::address::from\_string("127.0.0.1"),6688));

char buf[0xFF];

while (true) {

cin >> buf;

sock.send(buffer(buf));

memset(buf, 0, 0xFF);

sock.receive(buffer(buf));

cout << buf<<endl;

}

sock.close();

::system("pause");

}

sock.connect(ip::tcp::endpoint)函数：

连接服务器；

ip::address::from\_string("127.0.0.1")：

将ip地址转换成网络字节序

## 异步示例

#include<iostream>

#include"boost/asio.hpp"

#include"boost/bind.hpp"

using namespace std;

using namespace boost;

using asio::ip::tcp;

//异步连接的处理程序

void sock\_accept(tcp::socket\* sockCli);

//异步接受的处理程序

void sock\_Recv(char\* buf, tcp::socket \*sockCli);

//异步发送的处理程序

void sock\_Send(char\* buf, tcp::socket\* sockCli);

int main() {

cout << "server start ……" << endl;

asio::io\_context io;

tcp::acceptor acptr(io, tcp::endpoint(tcp::v4(), 6688));

tcp::socket \*sock=new tcp::socket(io);

acptr.async\_accept( \*sock , boost::bind(sock\_accept, sock) );

io.run(); //进入事件循环，等待执行，只要还有要给请求没有完成，就不会返回

}

void sock\_Send(char\* buf, tcp::socket\* sockCli)

{

try

{

sockCli->async\_receive( asio::buffer(buf, 0xFF), boost::bind(sock\_Recv, buf, sockCli) );

}

catch (std::exception& e)

{

cout << "";

cout << e.what();

delete sockCli;

delete[] buf;

}

}

void sock\_Recv(char\* buf, tcp::socket\* sockCli) {

try

{

sockCli->async\_send(asio::buffer(buf, 0xFF), boost::bind(sock\_Send, buf, sockCli));

}

catch (std::exception& e)

{

cout << "";

cout << e.what();

delete sockCli;

delete[] buf;

}

}

void sock\_accept(tcp::socket\* sockCli)

{

char\* buf = new char[0xFF];

std::cout << "client ip:" << sockCli->remote\_endpoint().address() << std::endl;

sockCli->async\_receive(asio::buffer(buf, 0xFF), boost::bind(&sock\_Recv, buf, sockCli) );

}

* + 发送与接受数据函数形成互相调用，请求就永远不会终止，除非发生异常：如客户端断开了连接，跳出这个互相调用的循环；
  + 没了请求需要处理，io\_context的run函数就会返回，从而程序结束；

acceptor.async\_accept(tcp::socket , void (\*)(tcp::socket\* sockCli) ):

异步等待客户端连接（提交一个监听客户端连接的请求）;

sockCli->async\_receive(asio::buffer(buf,0xFF),void(\*)(char\* buf , tcp::socket \*sockCli) ) :

异步接受数据（提交一个接收数据客户端的请求）;

sockCli->async\_send(asio::buffer(buf, 0xFF),void(\*)(char\* buf, tcp::socket\* sockCli));

异步发送数据（提交一个发送数据客户端的请求）；

# UDP编程

## 同步示例

**服务器端**

#include<iostream>

#include"boost/asio.hpp"

#include"boost/bind.hpp"

using namespace std;

using namespace boost;

using asio::ip::udp;

int main() {

cout << "server start ……" << endl;

asio::io\_context io;

//指定ipv4协议，绑定本地IP及端口，用来监听同网段IP

udp::socket sock( io, udp::endpoint(udp::v4(), 6688) );

char buf[0xFF];

udp::endpoint cliPoint;

while (1) {

// cliPoint : 保存客端地址

sock.receive\_from(asio::buffer(buf), cliPoint);

sock.send\_to( asio::buffer(buf),cliPoint );

}

}

**客户端**

#include<iostream>

#include"boost/asio.hpp"

using namespace std;

using namespace boost;

using boost::asio::ip::udp;

int main()

{

asio::io\_context io;

udp::socket sock(io);

sock.open(asio::ip::udp::v4()); //使用Ipv4协议

udp::endpoint serPoint (asio::ip::address::from\_string("127.0.0.1"),6688);

while (1)

{

char buf[0xFF];

cin >> buf;

sock.send\_to(asio::buffer(buf), serPoint);

memset(buf, 0, 0xFF);

sock.receive\_from(asio::buffer(buf), serPoint);

cout << buf << endl;

}

::system("pause");

}

## 异步示例

**服务器端**

#include<iostream>

#include"boost/asio.hpp"

#include"boost/bind.hpp"

using namespace std;

using namespace boost;

using asio::ip::udp;

void sock\_recv(char\* buf, udp::socket\* sock, udp::endpoint\* cliPoint);

void sock\_send(char\* buf, udp::socket\* sock, udp::endpoint\* cliPoint);

int main()

{

cout << "server start ……" << endl;

asio::io\_context io;

udp::socket \*sock=new udp::socket(io, udp::endpoint(udp::v4(), 6688));

char \*buf=new char[0xFF];

udp::endpoint \*cliPoint=new udp::endpoint;

sock->async\_receive\_from( asio::buffer(buf,0xFF) ,\*cliPoint,boost::bind(sock\_recv,buf,sock,cliPoint) );

io.run();

}

void sock\_send(char\* buf, udp::socket\* sock, udp::endpoint\* cliPoint) {

try

{

sock->async\_receive\_from(asio::buffer(buf, 0xFF), \*cliPoint,boost::bind(sock\_recv, buf, sock, cliPoint));

}

catch (const std::exception& e)

{

cout << e.what();

}

}

void sock\_recv(char\* buf, udp::socket\* sock, udp::endpoint\* cliPoint) {

try

{

sock->async\_send\_to(asio::buffer(buf, 0xFF), \*cliPoint, boost::bind(sock\_send, buf, sock, cliPoint));

}

catch (const std::exception& e)

{

cout << e.what();

}

}