继上篇 初心 已一星期，我们仍尚未谋面，我已迫不及待，是否还记得我曾经对你的承诺，我说，要让人人能编写高性能网络服务器，当然，这只是我一相情愿的告白，我不知道有没有被你看上，也不知道你是否还愿意在这条道上与我走一走，我们一起谈一谈，未来…

看起来这是一个梦啊，梦是当不得真的，但梦想还是可以做做的。

在这所有的所有的一切开始之前，我们还是不落俗套地见上一见吧，你说，自古套路得人心，而大家又总倾心于Hello World，要不，我们换一个，Ping Pong可好。

请你忘记你关于网络编程的想象，保留那么一点点CPP的印象，因为，好的忘记，是新的开始，而CPP正是我们的主角。

要懂的真的不多，稍微去了解一下，你甚至可能会怀疑以前的自己是不是走了很多弯路，虽然我们需要学习的概念并不多，但，还是有那么一点点，我们开始吧。

1. 网络中信息传递的基本问题

以ip:port为ID的机器存在于一张网上，我们把这些机器统称服务器，或网络节点。

机器与机器如果要传递信息，它们先要建立连接

已经建立连接的机器之间，可以互相传递消息

没错，网络信息传递，只有两个基本问题，即建立连接，然后传递消息。 而且这两项的工作，都已经被各操作系统实现好了，各家的技术细节并不一样，暴露出来的接口虽有相似之处，但仍然难以做到一份代码，到处运行。一些代码库，如libuv, libevent，在事件通知及系统兼容上面做得非常优秀，然，从实际应用开发者的角度来讲，它们遗留了很多需要开发者自己去解决的问题，如跨平台的一些细节，IO事件的处理，线程安全及性能调试，应用层协议的解析，而这些问题都涉及更多的知识点以及编程经验，这就是门槛啊，任何试图降低门槛的努力都是有价值的，也肯定是值得的。

Netplus试图往前走一步，将平台相关，IO事件处理，线程相关，性能相关，这些各APP里面都需要考虑的共性，进行抽象、封装。提供一个可扩展的机制，让各APP能通过为Channel添加自定义的Handler，在Handler里面做APP自己的协议解析，生成，以及业务处理，通过这种方式，APP开发者，只需要专注于自己的业务本身，就能轻松开发出基于网络通迅的应用程序。

同时，鉴于某些协议的流行程度，为了便于快速开发，Netplus也提供了直接的支持，如http/https,websocket，当然，也欢迎各位朋友为其添加其它的协议，让Netplu日渐丰满，我们的目标始终是，开箱即用，统统一把梭。

Netplus借鉴了netty的设计，那些熟悉 netty的朋友，就算不熟悉c++，应该也能快速上手。

2. Netplus里的基本概念

2.1 netp::ref\_base & netp::ref\_ptr

在Netplus里面，大部分的Class都继承于ref\_base，它提供引用计数的基本功能。

继承自ref\_base的class，直接new/delete 的时候，编译器会报错，这将能极大有利于内存管理。

继承netp::ref\_base的class，需要配合模板netp::ref\_ptr<T>进行使用。

netp::ref\_ptr<T>是一个基于引用计数的对象，通过netp::make\_ref<T> 进行创建，不需要的时候，通过置nullptr进行释放。此对象可以像普通类型那样进行赋值，也可以如指针一样用来访问其指向的对象。

更详细的知识，请参：https://link.zhihu.com/?target=https%3A//github.com/netplus/netplus/wiki/Smart-Pointer

下面是一个关于ref\_base, ref\_ptr如何使用example

class ref\_example: public netp::ref\_base

{

int number;

public:

ref\_example(int i):number(i){}

int get\_number() {return number;}

void set\_number(int i) {number=i;}

};

//创建，记住，不能直接new/delete, 只能借助 netp::make\_ref 函数;

netp::ref\_ptr<ref\_example> ref\_example\_1 = netp::make\_ref<ref\_example>(10);

//可像裸指针一样访问对象

ref\_example\_1 ->set\_number(10);

int number\_value = ref\_example\_1 ->get\_number();

//赋值，它是线程安全的

netp::ref\_ptr<ref\_example> ref\_example\_2 = ref\_example\_1;

{

//它是容器安全的,可在各容器里面自由存储

std::vector<netp::ref\_ptr<ref\_example>> ref\_example\_vector;

ref\_example\_vector.push\_back(ref\_example\_2);

}

//比较，注意，这里比较的是指向的地址

if( ref\_example\_2 == ref\_example\_1 ) {

//比较两个ref\_ptr对象是否指向同一个对象

}

//ref\_ptr对象的大小 == 机器的地址宽度，试想一下，将有何好处？

//sizeof(netp::ref\_ptr<ref\_example>) == sizeof(int\*);

//销毁， 不需要的时候，我们直接置nullptr即可

ref\_example\_1 = nullptr;

ref\_example\_2 = nullptr;

//借助于C++ RAII特性，局部变量，我们甚至都不用去置nullptr

{

netp::ref\_ptr<ref\_example> ref\_example\_3 = netp::make\_ref<ref\_example>(10);

//离开此作用域后， ref\_example\_3指向的内存将自动被释放

}

2.2 Packet

Packet是Netplus的一个重要Class, 继承自netp::ref\_base，它提供用于读写bytes buffer的接口，是我们用来操作bytes buffer的工具。

网络编程中，当处理Bytes Buffer，特别是处理protocol时，我们需要读或修改头部，或直接往头部前面添加一些数据。Packet被专门设计成应付这种场景，它既可以往buffer的左边写，也可以在buffer末端往前写，于是，处理起来将变得极为方便。

Channel将收到的bytes, 存储在Packet对象里面，然后再将此对象传递给它的第一个Handler。

往远端写bytes的时候，我们也是将bytes存储于一个Packet对象，然后最终传递给Channel。

Example:

//创建一个packet对象

netp::ref\_ptr<netp::packet> p = netp::make\_ref<netp::packet>();

//写入一个字符串

p->write("hello", netp::strlen("hello") );

//将字符串读入buf

//注：byte\_t 实为unsigned char

netp::byte\_t hello\_buf[10]={0};

netp::u32\_t read\_count = p->read(hello\_buf, 10);

//在packet buf的左边写入一个u32\_t大小的整数

p->write\_left<netp::u32\_t>(103);

//将刚刚写入的整数读入i103

netp::u32\_t i103 = p->read<u32\_t>();

//销毁packet对象

p = nullptr;

2.3 Channel

Channel是网络通信的主体，它即是当前的连接，它也是最终与操作系统进行交互，读写bytes，以及socket状态管理的实体。

每一个Channel将关联一个或多个Channel Handler，Channel Handler以单向链表的形式链接在一起，当与Channel相关的事件发生的时候，Channel负责将事件传递给第一个Channel Handler。

一个典型的channel read事件，按下面顺序在Handler中传递

socket read -> channel -> tail\_handler -> handler1 -> handler2 -> ...

一个典型的channel write事件，按下面顺序在Handler中传递

head\_handler -> ... -> handler2 -> handler1 -> tail\_handler -> channel -> socket write

每一个handler 都会自动添加 一个tail handler, 一个head handler，用于处理缺省行为

2.3 Channel Handler

Channel Handler是具体处理我们的消息的地方，处理完消息后，我们可以继续传递一个消息给下一个Handler，或直接终止当前消息的逻辑处理，甚至接关闭当前的Channel。

Channel Handler需要继承 netp::channelhandler\_abstract, 实现相应接口，用于处理网络事件，消息。

更多细节，请参: https://link.zhihu.com/?target=https%3A//github.com/netplus/netplus/wiki%23concept

在我们的例子中，我们将用到如下Channel handler接口

void read(netp::ref\_ptr<netp::channel\_handler\_context> const& ctx, netp::ref\_ptr<netp::packet> const& income);

void connected(netp::ref\_ptr<netp::channel\_handler\_context> const& ctx);

3. Netplus收发消息的基本流程

3.1 启动一个服务 (server端)：

实现自己的Channel Handler

在ip:port处监听服务

当Accept成功新的Channel后，为Channel添加Channel Handler

3.2 连接一个服务（client端）：

实现自己的Channel Handler

拨号至ip:port

当拨号成功时，为Channel添加Channel Handler

好了，就这些东西，没有更多了。

4. PINGPONG

4.1 PINGPONG服务器

监听在tcp://127.0.0.1:13103端口

收到到来自远端的的连接的时候，为Channel添加一个Pong Handler

Pong Handler: 此Handler只做一个事情，当收到来自客户端的消息后，回复PONG，代码如下：

class Pong :

public netp::channel\_handler\_abstract {

public:

Pong() :

channel\_handler\_abstract(netp::CH\_INBOUND\_READ)

{}

//for inbound

void read(netp::ref\_ptr<netp::channel\_handler\_context> const& ctx, netp::ref\_ptr<netp::packet> const& income) {

//reply with PONG

const std::string pong = "PONG";

netp::ref\_ptr<netp::packet> PONG = netp::make\_ref<netp::packet>(pong.c\_str(), pong.length());

netp::ref\_ptr<netp::promise<int>> write\_promise = ctx->write(PONG);

//check the reply status once the write operation is done

write\_promise->if\_done([](int reply\_rt) {

NETP\_INFO("[PONG]reply PONG, rt: %d", reply\_rt );

});

}

};

4.2 PINGPONG客户端

拨号到tcp://127.0.0.1:13103端口

当拨号成功之后，为Channel添加一个Ping Handler

Ping Handler: 当连接成功时，向服务器发送PING, 当成功收到回复的消息（PONG）后，继续发送PING，代码如下：

class Ping :

public netp::channel\_handler\_abstract {

public:

Ping():

channel\_handler\_abstract(netp::CH\_ACTIVITY\_CONNECTED|netp::CH\_INBOUND\_READ)

{}

void connected(netp::ref\_ptr<netp::channel\_handler\_context> const& ctx) {

NETP\_INFO("[PING]connected");

//initial PING

do\_ping(ctx);

}

void read(netp::ref\_ptr<netp::channel\_handler\_context> const& ctx, netp::ref\_ptr<netp::packet> const& income) {

NETP\_INFO("[PING]reply income");

do\_ping(ctx);

}

void do\_ping(netp::ref\_ptr<netp::channel\_handler\_context> const& ctx) {

const std::string ping = "PING";

netp::ref\_ptr<netp::packet> message\_ping = netp::make\_ref<netp::packet>();

message\_ping->write(ping.c\_str(), ping.length());

netp::ref\_ptr<netp::promise<int>> write\_p = ctx->write(message\_ping);

write\_p->if\_done([]( int rt ) {

NETP\_INFO("[PING]write PING, rt: %d", rt );

});

}

};

你看，服务器，客户端，都是三步曲

实现handler

监听/拨号

设置handler

So easy!

4.3 PING PONG的总体执行逻辑

服务器监听tcp://127.0.0.1:13103

当服务器有新的channel连接进来时，为新的channel添加handler

客户端拨号到tcp://127.0.0.1:13103

客户端拨号成功后，添加handler

4.4 main.cpp 完整代码如下：

int main(int argc, char\*\* argv) {

//initialize a netplus app instance

netp::app app;

std::string host = "tcp://127.0.0.1:13103";

netp::ref\_ptr<netp::channel\_listen\_promise> listenp = netp::socket::listen\_on(host, [](netp::ref\_ptr<netp::channel>const& ch) {

ch->pipeline()->add\_last( netp::make\_ref<netp::handler::hlen>());

ch->pipeline()->add\_last( netp::make\_ref<Pong>() );

});

int listenrt = std::get<0>(listenp->get());

if (listenrt != netp::OK) {

NETP\_INFO("listen on host: %s failed, fail code: %d", host.c\_str(), listenrt);

return listenrt;

}

netp::ref\_ptr<netp::channel\_dial\_promise> dialp = netp::socket::dial(host, [](netp::ref\_ptr<netp::channel> const& ch ) {

ch->pipeline()->add\_last( netp::make\_ref<netp::handler::hlen>() );

ch->pipeline()->add\_last( netp::make\_ref<Ping>() );

});

int dialrt = std::get<0>(dialp->get());

if (dialrt != netp::OK) {

//close listen channel and return

std::get<1>(listenp->get())->ch\_close();

return dialrt;

}

//wait for signal to exit

//Ctrl+C on windows

//kill -15 on linux

app.run();

//close listen channel

std::get<1>(listenp->get())->ch\_close();

//close dial channel

std::get<1>(dialp->get())->ch\_close();

return 0;

}

4.5 代码解读

4.5.1 netp::app

所有的netplus应用，netp::app app总是第一行代码，app实例代表着一个netplus对象，用处初始化netplus系统，设置信号处理。

app.run() 等待退出信号

4.5.2 Channel Handler hlen

这个handler用于处理格式为长度+内容的消息，内容长度占4bytes

发送时为消息添加消息长度，占4byte

接收时，先读4byte作为长度，然后按读到的长度继续读bytes, 直到读完给定的长度后，将read事件传递给下一个Handler

详细的描述，请参阅:https://link.zhihu.com/?target=https%3A//github.com/netplus/netplus/wiki/Handler%3A-hlen

（此篇文章待进一步整理）

完整的工程地址如下：https://link.zhihu.com/?target=https%3A//github.com/netplus/netplus/tree/main/test/pingpong

知识库：https://link.zhihu.com/?target=https%3A//github.com/netplus/netplus/wiki

如果你喜欢我的文章，请加个关注，点个赞，谢谢。

如果你有其它相关知识想要了解的，请直接，可以给我留言。

写代码的冰冰

姑苏城里平江路，入夜细雨扰我心，再会。

————————————————

版权声明：本文为CSDN博主「写代码的冰冰」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/ycopy2515/article/details/114217106