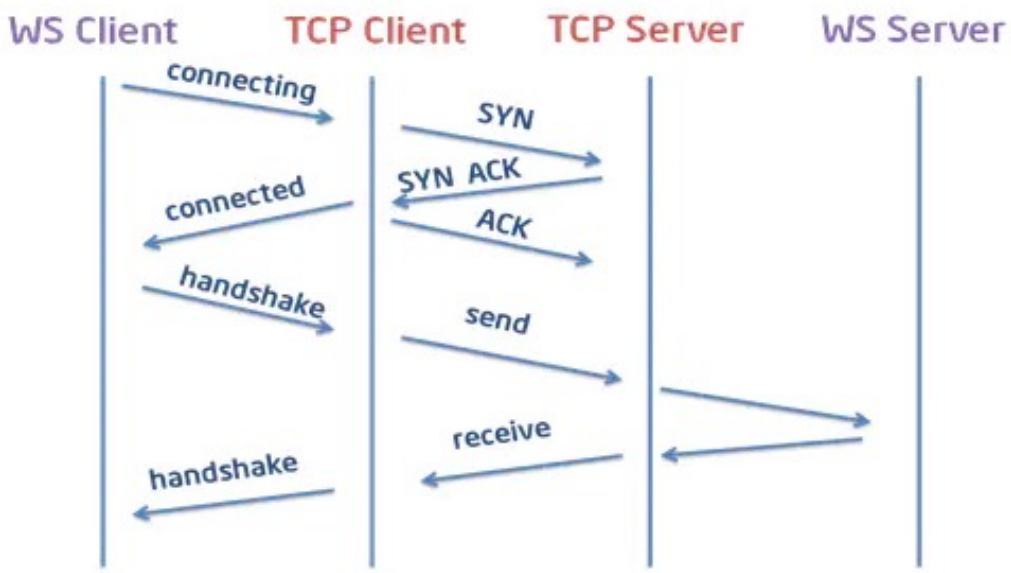
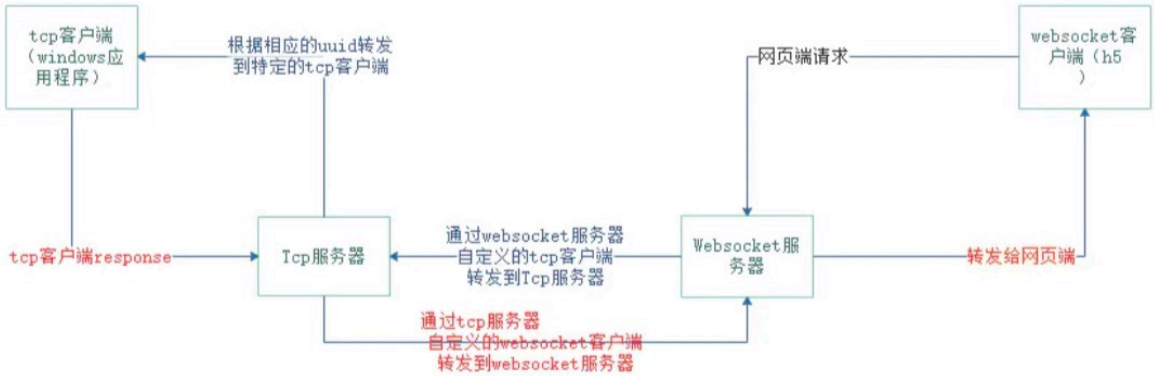
2020.05.07

websocket

websocket与TCPServer之间的沟通





本次选择在socket的server端判断并接收两种消息。

websocket_client 代码逻辑

WebSocket中包括两个主要的目标类型：endpoint和connection。

- endpoint负责建立连接，并返回连接的错误信息；处理分享的网络信息
- connection负责存储每个wss的信息。

> 一旦连接建立，endpoint和connection之间就没有了联系。所有的default设置都被endpoint复制到新的connection中。此时改变endpoint只会改变未来来到的连接。
>
> connection不会和endpoint有持久的联系（一旦连接成功，就detach了）

关于endpoint

endpoint role+ endpoint config

- role：client `<websocketpp/client.hpp>` ; server `<websocketpp/server.hpp>`
- config

  > 什么是 endpoint config?
  >
  > 通过 `config` template 类型，endpoints在编译的时候可以选择类型来创建。
  >
  > 比如：我们选择default配置：`<websocketpp/config/asio_no_tls_client.hpp>` ，此时配置使用的是boost::asio来进行网络传输，而不支持TLS。

  role和config组合起来作为endpoint，创建一个typedef：

  `typedef websocketpp::client<websocketpp::config::asio_client> client`

utility_client

1. 清除所有的的logging记录

```
m_endpoint.clear_access_channels(websocketpp::log::alevel::all);
m_endpoint.clear_error_channels(websocketpp::log::elevel::all);
```

2. 初始化传输系统，设置为perpetual状态（持续循环，即使没有连接进入）
   - 注：使用asio时需要，如果不用的话可以忽略这两行

```
m_endpoint.init_asio();
m_endpoint.start_perpetual();
```

3. 建立线程，启动client的`run` ，当endpoint在处理时，client处理到来的信息。

```
m_thread.reset(new websocketpp::lib::thread(&client::run, &m_endpoint));
```

4. client端需要一个安全随机秘钥生成器
   - 使用boost_random或者c++11中的
5. 建立新连接，为每一个连接分配id，用于查询该id下的msg。
6. 收到新连接时，更新websocket_endpoint
   - connect() 建立新连接
     - create：`endpoint::get_connection(uri)`

        - configure：处理请求的配置
        - connect：`endpoint::connect()`将请求发给endpoint，加入处理队列
    - get_metadata() 给出ID
    - metadata内包含本次连接的id、hdl、status、uri、server
        - 在命令行内输入show id号 可以查看此id的连接状态
7. 接收信息
    - show id 可以看到所有过往发送的信息
    - 在已收到的连接ID中找到它的endpoint；将信息和Opcode（Ping-9，Pong-10，Close-8，文本帧-1，二进制帧-2）存入handle；保存消息到连接的metadata中，用于后续show（不保存了所有接收发送的信息）
8. 关闭连接

build

- 必须include WebSocket++ and Boost library
    - 本机信息`-I /Users/mac/Documents/websocketpp-master/ -I /usr/local/Cellar/boost/1.72.0_2/include/ -L /usr/local/Cellar/boost/1.72.0_2/lib/`
- 依赖于`boost_system`，必须以动态或静态连接linker
    - `-lboost_system -lboost_random -lboost_thread`
- 测试 /tutorial/utility_client

```
g++ -std=c++0x step6.cpp -D_WEBSOCKET_CPP11_STL -I
/Users/mac/Documents/websocketpp-master/ -I
/usr/local/Cellar/boost/1.72.0_2/include/ -L
/usr/local/Cellar/boost/1.72.0_2/lib/
```

bug汇总

1. client第一次连接本地服务器的一个端口时，client连接成功，在server端打印出报头信息，但handshke失败；第二次连接同一个端口时，握手失败。client没有收到来自server的握手包。（修改tcp客户端逻辑）

```
MacdeMac-mini-2:echo_client mac$ ./test ws://127.0.0.1:8888
[2020-05-07 09:17:19] [connect] Successful connection
[2020-05-07 09:17:24] [fail] WebSocket Connection 127.0.0.1:8888 -
"WebSocket++/0.8.2" / 0 websocketpp:22 The opening handshake timed out
MacdeMac-mini-2:echo_client mac$ ./test ws://127.0.0.1:8888`
```

2. 编译整个库时，出现boost库无法链接问题 (似乎不影响client和server的编译，待de)

```
MacdeMac-mini-2:websocketpp-master mac$ scons
scons: Reading SConscript files ...
<SCons.Script.SConscript.SConsEnvironment object at 0x1050fb280>
C++11 build environment enabled
scons: done reading SConscript files.
scons: Building targets ...
scons: *** [build/test/transport/asio/test_base_boost] Implicit dependency
`/usr/local/Cellar/boost/1.72.0_2/lib/libboost_thread.a' not found, needed by
target `build/test/transport/asio/test_base_boost'.
scons: building terminated because of errors.
```

3. c++11缺少库boost_chrono

```
Undefined symbols for architecture x86_64:
  "boost::chrono::steady_clock::now()", referenced from:
      boost::asio::detail::chrono_time_traits<boost::chrono::steady_clock,
boost::asio::wait_traits<boost::chrono::steady_clock> >::now() in step6-
6b1e71.o
  "boost::detail::thread_data_base::~thread_data_base()", referenced from:
      boost::detail::thread_data<boost::_bi::bind_t<void,
boost::_mfi::mf0<unsigned long,
websocketpp::transport::asio::endpoint<websocketpp::config::asio_client::t
ransport_config> >,
boost::_bi::list1<boost::_bi::value<websocketpp::client<websocketpp::confi
g::asio_client>*> > > >::~thread_data() in step6-6b1e71.o
  "boost::random::random_device::random_device()", referenced from:
      websocketpp::random::random_device::int_generator<unsigned int,
websocketpp::concurrency::basic>::int_generator() in step6-6b1e71.o
  "boost::random::random_device::~random_device()", referenced from:
      websocketpp::random::random_device::int_generator<unsigned int,
websocketpp::concurrency::basic>::int_generator() in step6-6b1e71.o
      websocketpp::random::random_device::int_generator<unsigned int,
websocketpp::concurrency::basic>::~int_generator() in step6-6b1e71.o
  "boost::random::random_device::operator()()", referenced from:
      unsigned int
boost::random::detail::generate_uniform_int<boost::random::random_device,
unsigned int>(boost::random::random_device&, unsigned int, unsigned int,
mpl_::bool_<true>) in step6-6b1e71.o
  "boost::thread::join_noexcept()", referenced from:
      boost::thread::join() in step6-6b1e71.o
  "boost::thread::native_handle()", referenced from:
      boost::thread::get_id() const in step6-6b1e71.o
  "boost::thread::start_thread_noexcept()", referenced from:
      boost::thread::start_thread() in step6-6b1e71.o
  "boost::thread::detach()", referenced from:
      boost::thread::~thread() in step6-6b1e71.o
  "typeinfo for boost::detail::thread_data_base", referenced from:
```

```
      typeinfo for boost::detail::thread_data<boost::_bi::bind_t<void,
boost::_mfi::mf0<unsigned long,
websocketpp::transport::asio::endpoint<websocketpp::config::asio_client::t
ransport_config> >,
boost::_bi::list1<boost::_bi::value<websocketpp::client<websocketpp::confi
g::asio_client>*> > > > in step6-6b1e71.o
  "vtable for boost::detail::thread_data_base", referenced from:
      boost::detail::thread_data_base::thread_data_base() in step6-
6b1e71.o
  NOTE: a missing vtable usually means the first non-inline virtual member
function has no definition.
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see
invocation)
```

在scons中手动添加chrono（还是没用）

修改tcpServer内部逻辑

收到消息时判断收到的是一般http请求还是websocket请求

- handle中有Upgrade: websocket
- 将socket和websocket的请求放在服务器的两个端口分别进行