Vasilui 14 октября 2013 в 10:01

«Boost.Asio C++ Network Programming». Глава 7: Boost.Asio – дополнительные темы

Автор оригинала: Tim Packt Publishing

Программирование, C++, API

Перевод Tutorial

Всем привет!
Продолжаю перевод книги John Torjo «Boost.Asio C++ Network Programming».

Содержание:

В этой главе рассматриваются некоторые дополнительные темы Boost.Asio. Маловероятно, что вы будете использовать это каждый день, но, безусловно, будет не лишним это знать:

- Если отладка не удается, то вы увидите, что Boost.Asio поможет вам в этом
- Если вам придется работать с SSL, то посмотрите, что вам может предложить Boost.Asio
- Если вы пишите приложение под определенную ОС, то посмотрите, какие дополнительные функции есть в Boost.Asio для вас

Asio против Boost.Asio

Авторы Boost.Asio так же обеспечили поддержку Asio. Вы можете думать об этом как об Asio, так как эта библиотека поставляется в двух вариантах, Asio (не Boost) и Boost.Asio. Авторы утверждают, что обновления будут сначала появляться в Asio, и периодически будут добавляться в дистрибутив Boost-a.
Если в двух словах, то различия в следующем:

- Asio определено в нэймспейсе `asio::` , в то время как Boost.Asio определено в `boost::asio::`

- Заголовочный файл в Asio это `asio.hpp` и `boost/asio.hpp` в Boost.Asio
- В Asio есть свой класс для запуска потоков (эквивалент `boost::thread`)
- Asio предоставляет свои классы для кодов ошибок (`asio::error_code` вместо `boost::system::error_cod` и `asio::system_error` вместо `boost::system::system_error`)

Больше информации по Asio вы можете получить по ссылке.
Вы должны решить для себя, какой вариант вы предпочтете; лично я предпочитаю Boost.Asio. Вот несколько вещей, на которые стоит обратить внимание, делая свой выбор:

- Новые версии Asio выходят чаще, чем новые версии Boost.Asio (новые дистрибутивы Boost выходят довольно редко)
- Asio чисто заголовочная (в то время как часть Boost.Asio зависит от других библиотек Boost, которые могут понадобиться для компиляции)
- Обе, и Asio и Boost.Asio вполне развитые, поэтому если вы не горите желанием использовать функции из нового релиза Asio, то Boost.Asio довольно хорошая альтернатива, тем более что в вашем распоряжении будут и другие библиотеки Boost.

Вы можете использовать Asio и Boost.Asio в одном приложении, хотя, я не рекомендую вам делать этого. Это может получиться не специально, в этом случае все будет в порядке, например, если вы используете Asio, а некоторые сторонние библиотеки используют Boost.Asio и наоборот.

Отладка

Отладка синхронного приложения, как правило, проще, чем асинхронного приложения. Для синхронного приложения, если оно заблокируется, вы просто войдете в отладку и получите картину того, где произошло (синхронное означает последовательное). Однако, когда вы программируете асинхронно, события не происходят последовательно, поэтому, если произойдет ошибка, то ее действительно трудно будет отловить.
Чтобы избежать этого, во-первых, вы должны очень хорошо разбираться в сопрограммах. Если программа будет реализована правильно, то у вас, практически, не будет проблем вообще.
Только в том случае, когда дело доходит до асинхронного программирования, Boost.Asio протянет вам руку помощи; Boost.Asio позволяет отслеживать обработчики, если определен макрос `BOOST_ASIO_ENABLE_HANDLER_TRACKING`. Если это так, то Boost.Asio способствует выводу информации в стандартный поток вывода ошибок, записывая время, асинхронную операцию и, относящийся к ней, завершающий обработчик.

Информация отслеживания обработчиков

Информацию не так легко понять, но, тем не менее она очень полезна. На выходе Boost.Asio выдает следующее:

```
@asio|<timestamp>|<action>|<description>.
```

Первый тег всегда `@asio`, вы можете использовать его, чтобы легко фильтровать сообщения, приходящие от Boost.Asio в случае, если другие источники пишут в стандартный поток ошибок (эквивалент `std::cerr`). Экземпляр `timestamp` считается в секундах и микросекундах, начиная с 1 января 1970 UTC. Экземпляр `action` может быть чем-то из следующего:

- `>n` : используется, когда мы входим в обработчик с номером n. Экземпляр `description` содержит аргументы, передаваемые обработчику.
- `<n` : используется, когда обработчик номер n закрывается.
- `!n` : используется, когда мы вышли из обработчика n из-за исключения.
- `~n` : используется, когда обработчик с номером n разрушается без вызова; наверное, потому что экземпляр `io_service`

уничтожается слишком рано (до того, как n получит шанс вызваться).

- `n*m` : используется, когда обработчик n создает новую асинхронную операцию с завершающим обработчиком под номером m. После старта запущенная асинхронная операция отобразится в экземпляре `description` . Завершающий обработчик вызовется, когда вы увидите `>m(start)` и `<m(end)` .
- `n` : используется, когда обработчик с номером n выполняет операцию, которая отображается в `description` (которая может быть `close` или операцией `cancel`). Обычно, вы можете смело их игнорировать.

Всякий раз, когда n = 0, то снаружи выполняются все обработчики (асинхронно), обычно, вы видите, когда выполняется первая операция (операции) или в случае, если вы работаете с сигналами и срабатывает сигнал.
Вы должны обращать внимание на сообщения типа `!n` и `~n` , которые возникают, когда есть ошибки в коде. В первом случае, асинхронная функция не выбросила исключение, таким образом, исключение должно быть сгенерировано вами, вы не должны допускать исключений при выходе из вашего завершающего обработчика. В последнем случае вы, вероятно, уничтожили экземпляр `io_service` слишком рано, до завершения всех вызванных обработчиков.

Пример

Для того, чтобы показать вам пример вспомогательной информации, позвольте изменить пример из 6 главы. Все, что вам нужно сделать, это добавить дополнительный `#define` перед включением `boost/asio.hpp` :

```
#define BOOST_ASIO_ENABLE_HANDLER_TRACKING
#include <boost/asio.hpp>
...
```

Так же мы выведем дамп в консоль, когда пользователь войдет в систему и получит первый список клиентов. Вывод будет следующий:

```
@asio|1355603116.602867|0*1|socket@008D4EF8.async_connect
@asio|1355603116.604867|>1|ec=system:0
@asio|1355603116.604867|1*2|socket@008D4EF8.async_send
@asio|1355603116.604867|<1|
@asio|1355603116.604867|>2|ec=system:0,bytes_transferred=11
@asio|1355603116.604867|2*3|socket@008D4EF8.async_receive
@asio|1355603116.604867|<2|
@asio|1355603116.605867|>3|ec=system:0,bytes_transferred=9
@asio|1355603116.605867|3*4|io_service@008D4BC8.post
@asio|1355603116.605867|<3|
@asio|1355603116.605867|>4|
John logged in
@asio|1355603116.606867|4*5|io_service@008D4BC8.post
@asio|1355603116.606867|<4|
@asio|1355603116.606867|>5|
@asio|1355603116.606867|5*6|socket@008D4EF8.async_send
@asio|1355603116.606867|<5|
@asio|1355603116.606867|>6|ec=system:0,bytes_transferred=12
@asio|1355603116.606867|6*7|socket@008D4EF8.async_receive
@asio|1355603116.606867|<6|
@asio|1355603116.606867|>7|ec=system:0,bytes_transferred=14
@asio|1355603116.606867|7*8|io_service@008D4BC8.post
```

```
@asio|1355603116.607867|<7|
@asio|1355603116.607867|>8|
John, new client list: John
```

Позвольте проанализировать каждую строчку:

- Мы вводим `async_connect`, которая создает обработчик 1(в нашем случае все обрабатывают `talk_to_svr::step`)
- Вызывается обработчик 1 (после успешного подключения к серверу)
- Обработчик 1 вызывает `async_send`, которая создает обработчик 2 (здесь мы посылаем сообщение с логином на сервер)
- Обработчик 1 закрывается
- Вызывается обработчик 2 и посылает 11 байт (`login John`)
- Обработчик 2 вызывает `async_receive`, которая создает обработчик 3 (мы ждем, когда сервер ответит на наше сообщение с логином)
- Обработчик 2 закрывается
- Вызывается обработчик 3 и получает 9 байт (`login ok`)
- Обработчик 3 перенаправляет в `on_answer_from_server` (где создается обработчик 4)
- Обработчик 3 закрывается
- Вызывается обработчик 4, который потом запишет в дамп `John logged in`
- Обработчик 4 запускает еще один `step` (обработчик 5), который будет писать `ask_clients`
- Обработчик 4 закрывается
- Открывается обработчик 5
- Обработчик 5, `async_send` `ask_clients`, создает обработчик 6
- Обработчик 5 закрывается
- Вводится обработчик 6 (мы успешно отправили `ask_clients` серверу)
- Обработчик 6 вызывает `async_receive`, которая создает обработчик 7 (мы ждем, когда сервер отправит нам список существующих клиентов)
- Обработчик 6 закрывается
- Вызывается обработчик 7, и мы принимаем список клиентов
- Обработчик 7 запускает `on_answer_from_serve` (где создается обработчик 8)
- Обработчик 7 закрывается
- Открывается обработчик 8, и в дамп записывается список клиентов (`on_clients`)

Это займет некоторое время, чтобы привыкнуть, но, как только вы поймете это, вы сможете изолировать выходные данные, в которых содержится проблема и находить фактическую часть кода, которая должна быть исправлена.

Запись информации отслеживания обработчиков в файл

По умолчанию информация отслеживания обработчиков выводится в стандартный поток ошибок (эквивалент `std::cerr`). Очень вероятно, что вы захотите перенаправить этот вывод в другое место. С одной стороны, по умолчанию, для консольных приложений, вывод и сброс ошибок происходит в одно место, то есть в консоль. Но для Windows (не консольных) приложений, поток ошибок по умолчанию является пустым.
Вы можете перенаправить вывод ошибок с помощью командной строки, например, так:

```
some_application 2>err.txt
```

Если вы не поленитесь, то можете сделать это программно, как показано в следующем фрагменте кода:

```
// for Windows
HANDLE h = CreateFile("err.txt", GENERIC_WRITE, 0, 0, CREATE_ALWAYS,
        FILE_ATTRIBUTE_NORMAL , 0);
SetStdHandle(STD_ERROR_HANDLE, h);
// for Unix
int err_file = open("err.txt", O_WRONLY);
dup2(err_file, STDERR_FILENO);
```

SSL

Boost.Asio предоставляет классы для поддержки некоторых основных возможностей SSL. Внутри себя она использует `OpenSSL`. Так что, если вы хотите использовать SSL, то сначала загрузите и соберите OpenSSL. Следует отметить, что, как правило, построение `OpenSSL` задача не из легких, особенно, если у вас нет популярных компиляторов, таких как Visual Studio.
Если у вас есть успешно собранный `OpenSSL`, то Boost.Asio имеет некоторые классы надстройки над ним:

- `ssl::stream` : используется вместо класса `ip::::socket`
  `ssl::context` : это контекст для начала сеанса
  `ssl::rfc2818_verification` : этот класс является простым способом проверки сертификата по имени хоста в соответствии с правилами RFC 2818

В следующих нескольких статьях хочу предложить переводы первых глав нескольких книг (трех, возможно 4), какая больше придется вам по душе, ту и продолжу переводить первой.

Всем большое спасибо, до новых встреч!

Теги: c++, api, программирование
+31 123 25,1k 4 Поделиться
Хабы: Программирование, C++, API

68,5 Карма 0,0 Рейтинг

Подписаться

Кузьминых Василий @Vasilui
Пользователь

ПОХОЖИЕ ПУБЛИКАЦИИ

1 октября 2019 в 10:06

Make C++ great again!.. in Tula

55 19,3k 156 16 +16

19 сентября 2014 в 12:19

Изучаем C++ через программирование игр

26 79,2k 392 35 +35

11 июля 2014 в 04:16

О бедном C++ API замолвите словцо!

20 35,9k 174 71 +71

Комментарии 4

Отслеживать новые в ☐ почте ☐ трекере

DZhon 14 октября 2013 в 11:00 +1

Спасибо за верность нелёгкому делу переводчика!

Имеются следующие мысли:
1. В настоящий момент, насколько я знаю, чистая asio как раз отстаёт от boost-овой, да и вообще сайт выглядит заброшенным.
2. Вывод информации об отслеживании обработчиков таки можно визуализировать и представить в более удобном виде с точки зрения человека:

> The handler tracking output may be post-processed using the included handlerviz.pl tool to create a visual representation of the handlers (requires the GraphViz tool dot).

[1]

Ответить

RusMikle 14 октября 2013 в 11:33 +2

Спасибо что переводите и отдельное спасибо за то что делаете результат доступным.

Ответить

Vasilui 14 октября 2013 в 11:38 +2

Рад стараться, надеюсь, что будущие переводы будут не менее интересны для читателей!

Ответить

borisko 15 октября 2013 в 15:32 0

> Если вы не поленитесь, то можете сделать это программно, как показано в следующем фрагменте кода:

А можно кросплатформенно:

```
freopen("err.txt", "w", stderr);
```

Ответить

Написать комментарий

Предпросмотр Отправить Markdown (?)

САМОЕ ЧИТАЕМОЕ

Сутки Неделя Месяц

Собеседование в Яндекс: театр абсурда :/

+564 182k 449 1103 +1076

Дата-центр возле Амстердама называют «выгребной ямой интернета», но он продолжает работу

+42 23k 45 80 +54

Люди подозревают, что технологии — отстой, потому что они на самом деле отстой

+57 25k 40 257 +257

Листая старые подшивки. Взгляд изнутри на компьютерную прессу 90-х

+101 14,6k 55 92 +92

E-com-корзины как полуфабрикат. Кейсы вендоров

Мегапост

Если нашли опечатку в посте, выделите ее и нажмите Ctrl+Enter, чтобы сообщить автору.