Лекция 11. Библиотека Boost.

Илья Макаров

ИТМО JB

16 ноября 2021

Санкт-Петербург

Boost

- Коллекция библиотек, расширяющих функциональность C++.
- Свободно распространяются по лицензии Boost Software License вместе с исходным кодом и документацией на `www.boost.org`.
- Лицензия позволяет использовать boost в коммерческих проектах.
- Библиотеки из boost являются кандидатами на включение в стандарт C++.
- Некоторые библиотеки boost были включены в стандарты C++ 2014/17 года (`std::variant`, `std::optional`, ...).
- При включении библиотеки в boost она проходит несколько этапов рецензирования.
- Библиотеки boost позволяют обеспечить переносимость.
- В текущей версии в boost более сотни библиотек.

Boost

- Существенно замедляют компиляцию ваших проектов.
- Достаточно часто встречаются баги (в отличие от STL).
- Существенно увеличивают размер библиотеки/исполняемого файла.

Категории библиотек Boost

- String and text processing
- Containers,
- Iterators
- Algorithms
- Function objects and higher-order programming
- Generic Programming
- Template Metaprogramming
- Concurrent Programming
- Math and numerics
- Correctness and testing
- Data structures
- Domain Specific
- System
- Input/Output
- Memory
- Image processing
- Inter-language support
- Language Features Emulation
- Parsing
- Patterns and Idioms
- Programming Interfaces
- State Machines
- Broken compiler workarounds
- Preprocessor Metaprogramming

program_options

```
#include <boost/program_options.hpp>

namespace po = boost::program_options;

int main() {
    po::options_description desc("Allowed options");
    desc.add_options()
        ("help", "produce help message")
        ("comp", po::value<int>(), "compression level");

    po::variables_map vm;
    po::store(po::parse_command_line(ac, av, desc), vm);
    po::notify(vm);

    ...
}
```

program_options

```
int main() {
    ...

    if (vm.count("help")) {
        cout << desc << endl;
        return 1;
    }

    if (vm.count("comp")) {
        const auto comp_level = vm["comp"].as<int>();
    } else {
        cout << "Compression level was not set." << endl;
    }

    ...
}
```

any

```
#include <any>

using namespace std;

int main() {
    any a = 1;
    // i:1
    cout << format("{}:{}", a.type().name(), any_cast<int>(a)) << endl;
    a = 3.14;
    // d:3.14
    cout << format("{}:{}", a.type().name(), any_cast<double>(a)) << endl;
    try {
        a = 1;
        cout << any_cast<float>(a) << '\n';
    }
    catch (const bad_any_cast & e) {
        cout << e.what() << endl;
    }
}
```

any

```
#include <any>
#include <list>

using namespace std;
using many = list<any>;

void append_int(many & values, int value) {
    any to_append = value;
    values.push_back(to_append);
}
void append_string(many & values, const string & value) {
    values.push_back(value);
}
void append_char_ptr(many & values, const char * value) {
    values.push_back(value);
}
void append_any(many & values, const any & value) {
    values.push_back(value);
}
...
```

function

```
std::function<float(int x, int y)> f;
struct int_div {
    float operator()(int x, int y) const { return ((float)x)/y; };
};
f = int_div();
std::cout << f(5, 3) << std::endl;

std::function<void(int values[], int n, int& sum, float& avg)> sum_avg;
void do_sum_avg(int values[], int n, int& sum, float& avg) { ... }
sum_avg = &do_sum_avg;

struct X { int foo(int); };
std::function<int(X*, int)> f;
f = &X::foo;
X x;
f(&x, 5);
```

bind

```
struct image;

struct animation {
    void advance(int ms);
    bool inactive() const;
    void render(image & target) const;
};

std::vector<animation> anims;

template<class C, class P>
void erase_if(C & c, P pred) {
    c.erase(
        std::remove_if(c.begin(), c.end(), pred),
        c.end());
}
```

bind

```
void update(int ms) {
    std::for_each(
        anims.begin(),
        anims.end(),
        std::bind(&animation::advance, _1, ms));
    erase_if(anims, std::mem_fn(&animation::inactive));
}

void render(image & target) {
    std::for_each(
        anims.begin(),
        anims.end(),
        std::bind(&animation::render, _1,
                  std::ref(target)));
}
```

bind

```
void update(int ms) {
    for (auto && anim : anims) {
        anim.advance(ms);
    }
    erase_if(
        anims,
        [](auto & anim) { return anim.inactive(); });
}

void render(image & target) {
    for (auto && anim : anims) {
        anim.render(target);
    }
}
```

bind and function

```
struct button
{
    std::function<void()> onClick;
};

struct player
{
    void play();
    void stop();
};

button playButton, stopButton;
player thePlayer;

void connect()
{
    // equal to lambda wrapper
    playButton.onClick = std::bind(&player::play, &thePlayer);
    stopButton.onClick = std::bind(&player::stop, &thePlayer);
}
```

signal

```
#include <boost/signal.hpp>

using namespace std;

struct A {
    boost::signal<void()> SigA;
    boost::signal<void(int)> SigB;
};

struct B {
    void PrintFoo() { cout << "Foo" << endl; }
    void PrintBar(int i) { cout << "Bar: " << i << endl; }
};

int main() {
    A a; B b;

    // connect
    a.SigA.connect(bind(&B::PrintFoo, &b));
    a.SigB.connect(bind(&B::PrintBar, &b, _1));

    // post message
    a.SigA();
    a.SigB(4);
}
```

signal

```
#include <boost/signal.hpp>

using namespace std;

struct A {
    boost::signal<void()> SigA;
    boost::signal<void(int)> SigB;
};

struct B {
    void PrintFoo() { cout << "Foo" << endl; }
    void PrintBar(int i) { cout << "Bar: " << i << endl; }
};

int main() {
    A a; B b;

    // connect
    a.SigA.connect([&] { b.PrintFoo(); });
    a.SigB.connect([&](auto i) { b.PrintBar(i); });

    // post message
    a.SigA();
    a.SigB(4);
}
```

lexical_cast

```
#include <boost/lexical_cast.hpp>

int main(int, char ** argv) {
    std::vector<short> args;
    while (*++argv) {
        try {
            args.push_back(boost::lexical_cast<short>(*argv));
        }
        catch(const boost::bad_lexical_cast &) {
            args.push_back(0);
        }
    }
}

void log_message(const std::string &);

void log_errno(int yoko) {
    log_message(
        fmt::format("Error {}: {}",
        boost::lexical_cast<std::string>(yoko),
        strerror(yoko)));
}
```

optional

```
optional<char> get_async_input() {
    if ( !queue.empty() )
        return optional<char>(queue.top());
    else return optional<char>(); // uninitialized
}

void receive_async_message() {
    optional<char> rcv ;
    // The safe boolean conversion from 'rcv' is used here.
    while ( (rcv = get_async_input()) && !timeout() )
        output(*rcv);
}
```

String Algo

```
#include <boost/algorithm/string.hpp>
using namespace std;
using namespace boost;

string str1(" hello world! ");
to_upper(str1);  // str1 == "HELLO WORLD! "
trim(str1);      // str1 == "HELLO WORLD!"

string str2= to_lower_copy(
  ireplace_first_copy(str1,"hello","goodbye")); // str2 == "goodbye world"!

string str3("hello abc-*-ABC-*-aBc goodbye");
using find_vector_type = vector<iterator_range<string::iterator>>;

find_vector_type FindVec; // #1: Search for separators
ifind_all(FindVec, str3, "abc"); // { [abc],[ABC],[aBc] }

using split_vector_type = vector<string>;

split_vector_type SplitVec; // #2: Search for tokens
split(SplitVec, str3, is_any_of("-*-"), token_compress_on);
// { "hello abc","ABC","aBc goodbye" }
```

variant

```
#include "boost/variant.hpp"
#include <iostream>

struct my_visitor : public boost::static_visitor<int> {
    int operator()(int i) const {
        return i;
    }
    int operator()(const std::string & str) const {
        return str.length();
    }
};

int main() {
    boost::variant<int, std::string> u("hello world");
    std::cout << u; // output: hello world

    int result = boost::apply_visitor(my_visitor(), u);
    std::cout << result;
    // output: 11 (i.e., length of "hello world")
}
```

filesystem

```
// using namespace std::filesystem or boost::filesystem

int main(int argc, char **argv)
{
    path p(argv[1]);

    if (exists(p))
    {
        if (is_regular_file(p))
            cout << "size is " << file_size(p);
        else if (is_directory(p))
            cout << "is a directory";
        else
            cout << "exists, but is neither a regular file nor a directory";
    }
    else
        cout << p << "does not exist\n";
}
```