2016 C++及系统软件技术大会
C++ and System Software Summit

C++ 和函数式编程

写紧凑、表意、可组合的代码

吴咏炜
wuyongwei@gmail.com

讲在前面

- 点到即止
  - 内容太多……
  - 有可工作的代码作参考（放在 GitHub 上）
- 充足的参考资料
- 主要模拟 Python 而非 Haskell

I. 基础篇

函数式编程的基本概念

- 函数是「一等公民」
  - 函数可以作参数用，也可以返回函数
- 函数引用透明；有返回值而没有副作用
  - 易于测试，易于重现错误
  - 易于并行执行

函数式编程的风格

- 强调数据的关系，而不是计算的步骤
- 使用递归
  - 传统函数式编程语言通常有尾递归优化
    - *Not* Python
- 使用序列和循环
  - Python!
- 使用高阶函数（函数的函数）
- 易于组合
- 易于惰性求值

最基本的函数式编程模式

- Map
  - map f [A, B, C, D, …] = [f A, f B, f C, f D, …]
- Reduce
  - reduce (+) 0 [A, B, C, D, …] = 0 + A + B + C + D …
- Compose
  - (f . g) A = f (g A)
  - h = f . g ➔ h A = f (g A)

C++的函数式语言特性

C++98 旧有

- 函数对象
- ~~std::bind1st/bind2nd~~
- ~~std::for_each~~
- std::transform
- std::accumulate

C++11/14 新增

- Lambda 表达式
- std::bind
- 基于区间的 for 循环
- std::function
- 自动类型推导
- 变参模板

函数对象

定义

```
class plus_n {
    int n_;
public:
    plus_n(int n) : n_(n) {}
    int operator()(int x) const { return x + n_; }
};
```

使用

```
plus_n plus_1(1);
cout << plus_1(41) << endl;
cout << plus_n(2)(40) << endl;
```

std::transform

```
int a[5] = {0, 1, 2, 3, 4};
transform(a, a + 5, a, plus_n(1));
// a = {1, 2, 3, 4, 5}

vector<int> v{0, 1, 2, 3, 4};
transform(v.begin(), v.end(), v.begin(), plus_n(1));
// v = {1, 2, 3, 4, 5}

transform(v.begin(), v.end(), a, plus_n(1));
// a = {2, 3, 4, 5, 6}
```

std::accumulate

```
vector<int> v{1, 2, 3, 4, 5};

cout << accumulate(v.begin(), v.end(), 0) << endl;
// 15

cout << accumulate(v.begin(), v.end(), 1,
                   std::multiplies<int>()) << endl;
// 120
```

自动类型推导

```
auto result = f(*begin(inputs));
decltype(auto) result = f(*begin(inputs));
decltype(f(*begin(inputs))) result;
decltype(f(declval<T>())) result;
```

- auto 和 decltype 有不同的推导规则
  - auto 抛弃引用和 cv 修饰
  - decltype 可保留引用（详见参考资料）
- decltype(auto) 使用 decltype 的推导规则
- std::declval 可宣称一个某类型的参数

Lambda 表达式和 std::function

```
function<int(int)> get_strange_func(int n)
{
    if (n > 0)
        return [n](int x) { return x + n; };
    else
        return [n](int x) { return x * n; };
}
…
    auto plus_1 = [](int x) { return x + 1; };  // 唯一类型，只能用 auto 接收
    cout << plus_1(41) << endl;                 // 42
    cout << get_strange_func(2)(40) << endl;    // 42
    cout << get_strange_func(-2)(-21) << endl;  // 42
```

- Lambda 表达式等价于构造了一个函数对象
- 捕捉表达式可以转换为构造函数和成员变量
- 泛型 lambda 表达式可以认为是带模板的 operator()
- function 可以保存符合调用规范的函数、函数对象和 lambda 表达式

std::bind

```
using namespace std::placeholders;

auto plus_1 = bind(plus<int>(), _1, 1);
cout << plus_1(41) << endl;
// 42
```

基于区间的 for 循环

```
for (auto x : r) {
    …
}
```

大致相当于

```
for (auto it = begin(r); it != end(r); ++it) {
    auto x = *it;
    …
}
```

变参模板

```
template <typename T>
constexpr auto sum(T x)
{
    return x;
}

template <typename T1, typename T2, typename... Targ>
constexpr auto sum(T1 x, T2 y, Targ... args)
{
    return sum(x + y, args...);
}
```

- 编译期递归！

小串烧

```
int main()
{
    (void)sum<int, int, int, int, int>;  // GCC requires this instantiation

    auto sqr = [](auto x) { return x * x; };
    auto sqr_list = [sqr](auto x) { return fmap(sqr, x); };
    auto sum_list = [](auto x) { return reduce(plus<int>(), x); };
    vector<int> v{1, 2, 3, 4, 5};
    tuple<int, int, int, int, int> t{1, 2, 3, 4, 5};

    auto sum_sqr_list = compose(sum_list, sqr_list);
    cout << sum_sqr_list(v) << endl;
    // 55

    cout << pipeline(v, sqr_list, sum_list) << endl;
    // 55

    cout << apply(sum<int, int, int, int, int>, fmap(sqr, t)) << endl;
    // 55

    cout << reduce(plus<int>(), fmap(sqr, t), 0) << endl;
    // 55
}
```

II. 实战篇

A Tale of Two Languages (I)

```python
#!/usr/bin/env python
#coding: utf-8

import sys

def main():
    for line in sys.stdin:
        print(line.rstrip('\n'))

if __name__ == '__main__':
    main()
```

```cpp
#include <iostream>
#include <string>

using namespace std;

int main()
{
    string line;
    for (;;) {
        getline(cin, line);
        if (!cin) {
            break;
        }
        cout << line << '\n';
    }
}
```

istream_line_reader (C++98 兼容)

```
class istream_line_reader {
public:
    class iterator {  // implements InputIterator
    public:
        typedef const std::string& reference;
        typedef std::string value_type;

        iterator() : stream_(NULL) {}
        explicit iterator(std::istream& is) : stream_(&is)
        { ++*this; }
        reference operator*()
        {
            assert(stream_ != NULL);
            return line_;
        }
        value_type* operator->()
        {
            assert(stream_ != NULL);
            return &line_;
        }
        iterator& operator++()
        {
            getline(*stream_, line_);
            if (!*stream_) {
                stream_ = NULL;
            }
            return *this;
        }
        iterator operator++(int)
        {
            iterator temp(*this);
            ++*this;
            return temp;
        }
        bool operator==(const iterator& rhs) const
        { return stream_ == rhs.stream_; }
        bool operator!=(const iterator& rhs) const
        { return !operator==(rhs); }

    private:
        std::istream* stream_;
        std::string line_;
    };

    explicit istream_line_reader(std::istream& is)
        : stream_(is) {}
    iterator begin() { return iterator(stream_); }
    iterator end() const { return iterator(); }

private:
    std::istream& stream_;
};
```

```
reference operator*()
{
    assert(stream_ != nullptr);
    return line_;
}
```

```
iterator& operator++()
{
    getline(*stream_, line_);
    if (!*stream_) {
        stream_ = nullptr;
    }
    return *this;
}
```

```
#include <iostream>
#include <string>
#include "istream_line_reader.h"

using namespace std;

int main()
{
    istream_line_reader reader(cin);
    for (istream_line_reader::iterator
            it = reader.begin();
         it != reader.end(); ++it) {
        cout << *it << '\n';
    }
}
```

等效的 C++11 代码

C++98

```
#include <iostream>
#include <string>
#include "istream_line_reader.h"

using namespace std;

int main()
{
    istream_line_reader reader(cin);
    for (istream_line_reader::iterator
              it = reader.begin();
         it != reader.end(); ++it) {
        cout << *it << '\n';
    }
}
```

C++11

```
#include <iostream>
#include <string>
#include "istream_line_reader.h"

using namespace std;

int main()
{
    for (auto& line :
         istream_line_reader(cin)) {
        cout << line << '\n';
    }
}
```

A Tale of Two Languages (II)

```python
#!/usr/bin/env python
#coding: utf-8

import sys

def backsort(lines):
    pairs = []
    for line in lines:
        pairs.append(('.'.join(
            reversed(line.split('.'))), line))
    return map(lambda item: item[1],
               sorted(pairs))

def main():
    result = backsort(map(
               lambda line: line.rstrip('\n'),
               sys.stdin))
    for line in result:
        print(line)

if __name__ == '__main__':
    main()
```

```cpp
#include …

using namespace std;
vector<string> split(const string& str, char delim);
template <typename C>
string join(const C& str_list, char delim);

template <typename C>
vector<string> backsort(C&& lines)
{
    vector<pair<string, string>> pairs;
    for (auto& line : lines) {
        auto split_line = split(line, '.');
        reverse(split_line.begin(),
                split_line.end());
        pairs.emplace_back(
            make_pair(join(split_line, '.'), line));
    }
    sort(pairs.begin(), pairs.end());
    vector<string> result(pairs.size());
    transform(pairs.begin(), pairs.end(),
              result.begin(),
              [](const auto& pr) { return pr.second; });
    return result;
}

int main(int argc, char* argv[])
{
    auto result = backsort(istream_line_reader(cin));
    for (auto& item : result) {
        cout << item << endl;
    }
}
```

简化后的 backsort

Old

```
template <typename C>
vector<string> backsort(C&& lines)
{
    vector<pair<string, string>> pairs;
    for (auto& line : lines) {
        auto split_line = split(line, '.');
        reverse(split_line.begin(),
                split_line.end());
        pairs.emplace_back(
            make_pair(join(split_line, '.'), line));
    }
    sort(pairs.begin(), pairs.end());
    vector<string> result(pairs.size());
    transform(pairs.begin(), pairs.end(),
              result.begin(),
              [](const auto& pr) { return pr.second; });
    return result;
}
```

New

```
template <typename C>
vector<string> backsort(C&& lines)
{
    vector<pair<string, string>> pairs;
    for (auto& line : lines) {
        auto split_line = split(line, '.');
        reverse(split_line.begin(),
                split_line.end());
        pairs.emplace_back(
            make_pair(join(split_line, '.'), line));
    }
    sort(pairs.begin(), pairs.end());
    return fmap([](const auto& pr) { return pr.second; },
                pairs);
}
```

A Tale of Two Languages (IIIa)

```python
#!/usr/bin/env python
#coding: utf-8

import sys

def cat(files):
    for fn in files:
        with open(fn) as f:
            for line in f:
                yield line.rstrip('\n')

def backsort(lines):
    result = {}
    for line in lines:
        result['.'.join(
            reversed(line.split('.')))] = line
    return map(lambda item: item[1],
               sorted(result.items()))

def main():
    if sys.argv[1:]:
        result = backsort(cat(sys.argv[1:]))
    else:
        result = backsort(map(
            lambda line: line.rstrip('\n'),
            sys.stdin))
    for line in result:
        print(line)

if __name__ == '__main__':
    main()
```

?

可能的解

- 像 istream_line_reader 一样实现个辅助类
  - 前者还算通用，这个……好复杂啊
- 实现类似于 cat 的函数，然后回调/调用 backsort
  - 哈，backsort 得改，这个接口没法用
- 把文件内容全读进来，再传给 backsort
  - 好吧，这次可以；如果函数做的事情不需要在内存保存所有数据，文件又超过内存大小呢？
- 实现 generator 模式！
  - 推荐 Boost.Coroutine2

简化 istream_line_reader (示意)

```
class istream_line_reader {
public:
    class iterator {  // implements InputIterator
    public:
        typedef const std::string& reference;
        typedef std::string value_type;

        iterator() : stream_(NULL) {}
        explicit iterator(std::istream& is) : stream_(&is)
        { ++*this; }
        reference operator*()
        {
            assert(stream_ != NULL);
            return line_;
        }
        value_type* operator->()
        {
            assert(stream_ != NULL);
            return &line_;
        }
        iterator& operator++()
        {
            getline(*stream_, line_);
            if (!*stream_) {
                stream_ = NULL;
            }
            return *this;
        }
        iterator operator++(int)
        {
            iterator temp(*this);
            ++*this;
            return temp;
        }
        bool operator==(const iterator& rhs) const
        { return stream_ == rhs.stream_; }
        bool operator!=(const iterator& rhs) const
        { return !operator==(rhs); }

    private:
        std::istream* stream_;
        std::string line_;
    };

    explicit istream_line_reader(std::istream& is)
        : stream_(is) {}
    iterator begin() { return iterator(stream_); }
    iterator end() const { return iterator(); }

private:
    std::istream& stream_;
};
```

```
typedef boost::coroutines2::coroutine<const
string&> coro_t;

void read_istream(coro_t::push_type& yield,
                  std::istream& is)
{
    for (;;) {
        string line;
        if (getline(is, line)) {
            yield(line);
        } else {
            break;
        }
    }
}

auto istream_line_reader(std::istream& is)
{
    return coro_t::pull_type(
        boost::coroutines2::fixedsize_stack(),
        bind(read_istream, _1, std::ref(is)));
}
```

改造 backsort

```
typedef boost::coroutines2::coroutine<const string&> coro_t;

void cat(coro_t::push_type& yield, int files_cnt, char* file_names[])
{
    for (int i = 0; i < files_cnt; ++i) {
        ifstream ifs(file_names[i]);
        for (auto& line : istream_line_reader(ifs)) {
            yield(line);
        }
    }
}

int main(int argc, char* argv[])
{
    vector<string> result;
    if (argc > 1) {
        result = backsort(coro_t::pull_type(
            boost::coroutines2::fixedsize_stack(),
            bind(cat, _1, argc - 1, argv + 1)));
    }
    …
}
```

Pythagorean Triples (Haskell)

```
main = print (take 10 triples)

triples = [(x, y, z) | z <- [1..]
                     , x <- [1..z]
                     , y <- [x..z]
                     , x^2 + y^2 == z^2]
```

Pythagorean Triples (Python)

```python
#!/usr/bin/env python
#coding: utf-8

def gen_pythagorean_triple():
    z = 1
    while True:
        z += 1
        yield [(x, y, z) for x in range(1, z)
                         for y in range(x, z)
                         if x**2 + y**2 == z**2]

def main():
    generator = gen_pythagorean_triple()
    triples = []
    while len(triples) < 10:
        triples += next(generator);
    print(triples)

if __name__ == '__main__':
    main()
```

Pythagorean Triples (C++ Ranges)

```
#include <iostream>
#include <range/v3/all.hpp>

using namespace ranges;

int main()
{
    // Lazy ranges for generating integer sequences
    auto const intsFrom = view::iota;
    auto const ints = [=](int i, int j) { return view::take(intsFrom(i), j - i + 1); };

    // Define an infinite range of all the Pythagorean triples:
    auto triples = view::for_each(intsFrom(1), [=](int z)
        {
            return view::for_each(ints(1, z), [=](int x)
            {
                return view::for_each(ints(x, z), [=](int y)
                {
                    return yield_if(x*x + y*y == z*z, std::make_tuple(x, y, z));
                });
            });
        });

    // Display the first 10 triples
    for (auto triple : triples | view::take(10)) {
        std::cout << '('
            << std::get<0>(triple) << ','
            << std::get<1>(triple) << ','
            << std::get<2>(triple) << ')' << '\n';
    }
}
```

用 ranges 优化 backsort

Old

```
template <typename C>
vector<string> backsort(C&& lines)
{
    vector<pair<string, string>> pairs;
    for (auto& line : lines) {
        auto split_line = split(line, '.');
        reverse(split_line.begin(),
                split_line.end());
        pairs.emplace_back(
            make_pair(join(split_line, '.'),
                      line));
    }
    sort(pairs.begin(), pairs.end());
    return fmap([](const auto& pr)
                {
                    return pr.second;
                },
                pairs);
}
```

New

```
template <typename C>
vector<string> backsort(C&& lines)
{
    vector<pair<string, string>> pairs;
    for (auto& line : lines) {
        auto split_line = action::split(line, '.');
        pairs.emplace_back(make_pair(
            to_<string>(split_line |
                        view::reverse |
                        view::join('.')),
            line));
    }
    ranges::sort(pairs);
    return pairs | view::values;
}
```

III. 总结

C++ 里的函数式编程

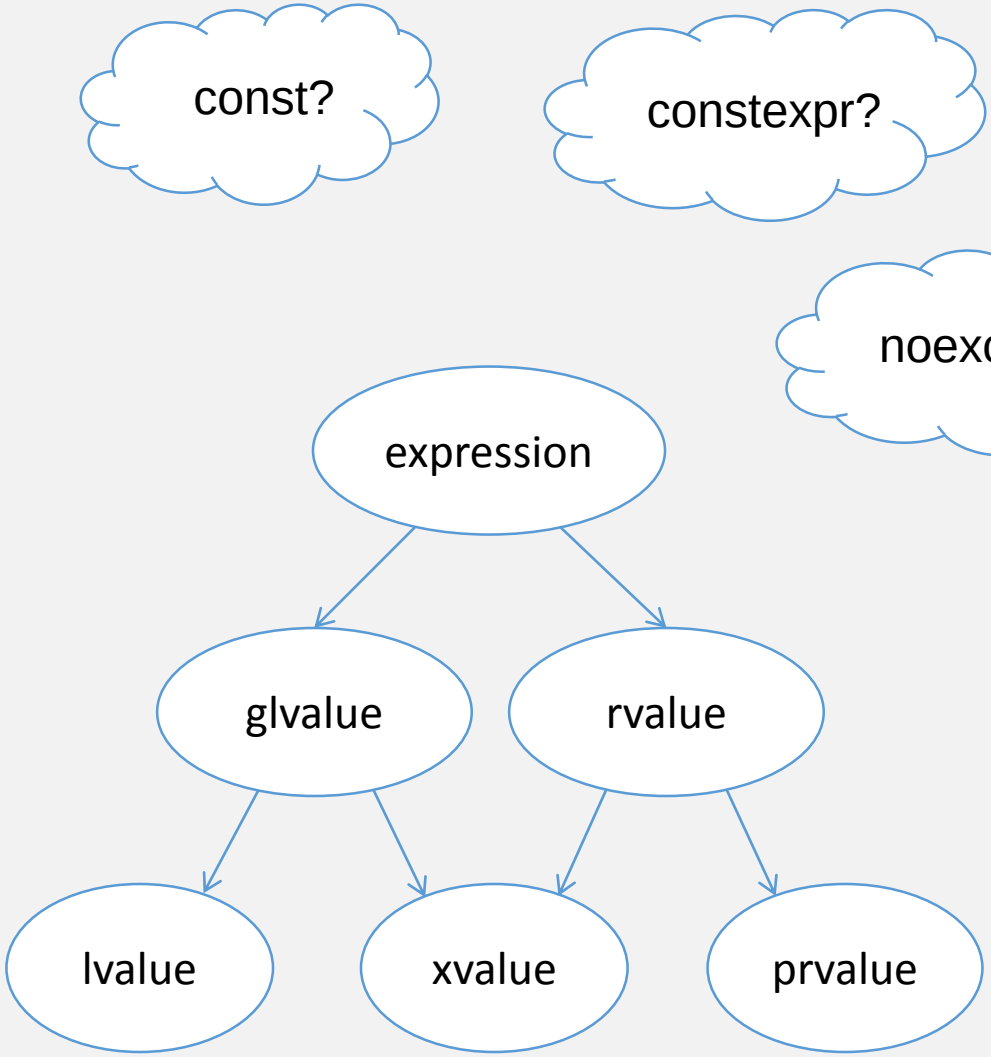
优点

- 提供更清晰、更好的抽象
- 代码更紧凑
- 代码更容易组合
- 惰性求值可导致更好的性能

缺点

- 造轮子比较辛苦
- 类型系统比大多数语言复杂
    - POD 值类型，非 POD 值类型，各种不同的引用
- 出错信息可能让人发疯

C++ 的坑




```
backsort1c.cpp: In instantiation of
'std::vector<std::__cxx11::basic_string<char> > backsort(C&&)
[with C = istream_line_reader]':
backsort1c.cpp:61:52:   required from here
backsort1c.cpp:55:16: error: no matching function for call to
'fmap(backsort(C&&) [with C =
istream_line_reader]::<lambda(auto:4&)>,
std::vector<std::pair<std::__cxx11::basic_string<char>,
std::__cxx11::basic_string<char> > >&)'
     return fmap([](auto& pr) { return pr.second; },
            ~~~~^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                 pairs);
                 ~~~~~~
In file included from backsort1c.cpp:6:0:
../nvwa/nvwa/functional.h:282:16: note: candidate:
template<class _Fn, class ... _Targs> constexpr auto
nvwa::fmap(_Fn, const nvwa::optional<_Targs>& ...)
 constexpr auto fmap(_Fn f, const optional<_Targs>&... args)
                ^~~~
../nvwa/nvwa/functional.h:282:16: note:   template argument
deduction/substitution failed:
backsort1c.cpp:55:16: note:
'std::vector<std::pair<std::__cxx11::basic_string<char>,
std::__cxx11::basic_string<char> > >' is not derived from
'const nvwa::optional<_Targs>'
     return fmap([](auto& pr) { return pr.second; },
            ~~~~^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                 pairs);
                 ~~~~~~
In file included from backsort1c.cpp:6:0:
../nvwa/nvwa/functional.h:303:16: note: candidate:
template<class _Fn, class ... _Targs> constexpr auto
nvwa::fmap(_Fn, nvwa::optional<_Targs>&& ...)
 constexpr auto fmap(_Fn f, optional<_Targs>&&... args)
                ^~~~
../nvwa/nvwa/functional.h:303:16: note:   template argument
deduction/substitution failed:
backsort1c.cpp:55:16: note:
'std::vector<std::pair<std::__cxx11::basic_string<char>,
std::__cxx11::basic_string<char> > >' is not derived from
'nvwa::optional<_Targs>'
     return fmap([](auto& pr) { return pr.second; },
            ~~~~^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                 pairs);
                 ~~~~~~
In file included from backsort1c.cpp:6:0:
../nvwa/nvwa/functional.h:329:16: note: candidate:
template<template<class, class> class _OutCont, template<class>
```

参考资料 I

C++ 编程

- cppreference.com, C++ Reference, http://en.cppreference.com/w/ (有中文版)
- Eric Niebler, “Range Comprehensions”, http://ericniebler.com/2014/04/27/range-comprehensions/
- Eric Niebler, “D4128: Ranges for the Standard Library: Revision 1”, https://ericniebler.github.io/std/wg21/D4128.html
- Eric Niebler, “Range-v3”, https://ericniebler.github.io/range-v3/
- Oliver Kowalke, “Boost.Coroutine2”, http://www.boost.org/doc/libs/release/libs/coroutine2/
- Thomas Becker, “C++ Rvalue References Explained”, http://thbecker.net/articles/rvalue_references/section_01.html
- Thomas Becker, “C++ `auto` and `decltype` Explained”, http://thbecker.net/articles/auto_and_decltype/section_01.html

参考资料 II

函数式编程

- 阮一峰，《函数式编程初探》，http://www.ruanyifeng.com/blog/2012/04/functional_programming.html
- 陈皓，《函数式编程》，http://coolshell.cn/articles/10822.html
- Slava Akhmechet, “Functional Programming For The Rest of Us”, http://www.defmacro.org/ramblings/fp.html (有中文版)

我的函数式参考代码

- fmap、reduce、compose、pipeline、apply、curry、optional 和 file_line_reader 的实现，http://nvwa.cvs.sourceforge.net/viewvc/nvwa/nvwa/ (functional.h, file_line_reader.h/cpp)
- 我的博客文章，http://wyw.dcweb.cn/index.htm#articles
- 演讲文件和代码，https://github.com/adah1972/cpp_conf_china_2016

其他

- Guido van Rossum, “Tail Recursion Elimination”, http://neopythonic.blogspot.com.au/2009/04/tail-recursion-elimination.html

IV. 备用材料

Currying

- 看起来很美
  - plus_1 = (+) 1
  - plus_1 2 ➔ (+) 1 2 ➔ 3
- C++ 里可以手工用 bind 或 lambda 实现
  - auto plus_1 = bind(plus<int>(), _1, 1);
- 可以用库实现通用的方案（产品项目中不推荐）
  - auto curried_plus = nvwa::make_curry<int(int, int)>(plus<int>());
  - auto plus_1 = curried_plus(1);
  - curried_plus(1)(1) ➔ 2
  - plus_1(1) ➔ 2

fmap (simplified)

```
template <class T1, class T2>
struct can_reserve
{
    …;
    static const bool value = (sizeof(reserve<T1>(nullptr)) == sizeof(good) &&
                               sizeof(size<T2>   (nullptr)) == sizeof(good));
};
template <class T1, class T2>
void try_reserve(T1&, const T2&, false_type)
{
}
template <class T1, class T2>
void try_reserve(T1& dest, const T2& src, true_type)
{
    dest.reserve(src.size());
}

template <template <typename, typename> class OutCont = vector,
          template <typename> class Alloc = allocator,
          typename Fn, class Cont>
constexpr auto fmap(Fn&& f, const Cont& inputs)
{
    typedef decay_t<decltype(f(*begin(inputs)))> result_type;
    OutCont<result_type, Alloc<result_type>> result;
    try_reserve(result, inputs,
        integral_constant<bool, can_reserve<decltype(result), Cont>::value>());
    for (const auto& item : inputs)
        result.push_back(f(item));
    return result;
}
```

惰性的 fmap (示例)

```
template <typename Rs, typename Fn, typename Rng>
auto fmap_view_impl(
    typename boost::coroutines2::coroutine<Rs>::push_type& yield,
    Fn&& f, Rng&& inputs)
{
    for (auto& x : inputs) {
        yield(f(x));
    }
}

template <typename Fn, typename Rng>
auto fmap_view(Fn&& f, Rng&& inputs)
{
    typedef decltype(f(*begin(inputs))) result_type;
    return typename boost::coroutines2::coroutine<result_type>::pull_type(
        boost::coroutines2::fixedsize_stack(),
        bind(fmap_view_impl<result_type, Fn, Rng>,
             _1,
             forward<Fn>(f),
             forward<Rng>(inputs)));
}

…
cout << sum_list(fmap_view(sqr, v)) << endl;
```

不使用异常的错误返回方式

- optional 模板（将出现在 C++17 中）
- lift_optional 函数（nvwa）

```
 auto sqr_opt = lift_optional(sqr);
auto plus_opt = lift_optional(plus<int>());
auto invalid_value = optional<int>();
cout << sqr_opt(make_optional(2)) << endl;                    // 4
cout << sqr_opt(invalid_value) << endl;                       // invalid
cout << plus_opt(make_optional(2), make_optional(3)) << endl; // 5
cout << plus_opt(make_optional(2), invalid_value) << endl;    // invalid
```

用 file_line_reader 优化读行

Old

```
void cat(coro_t::push_type& yield,
         int files_cnt,
         char* file_names[])
{
    for (int i = 0; i < files_cnt; ++i) {
        ifstream ifs(file_names[i]);
        for (auto& line : istream_line_reader(ifs)) {
            yield(line);
        }
    }
}
```

```
template <typename C>
vector<string> backsort(C&& lines)
{
    vector<pair<string, string>> pairs;
    for (auto& line : lines) {
        auto split_line = action::split(line, '.');
        pairs.emplace_back(make_pair(
            to_<string>(split_line |
                        view::reverse |
                        view::join('.')),
            line));
    }
    ranges::sort(pairs);
    return pairs | view::transform([](const auto& pr)
                   {
                       return pr.second;
                   });
}
```

New

```
void cat(coro_t::push_type& yield,
         int files_cnt,
         char* file_names[])
{
    for (int i = 0; i < files_cnt; ++i) {
        FILE* fp = fopen(file_names[i], "r");
        if (!fp) continue;
        for (auto& line : file_line_reader(fp)) {
            yield(line);
        }
        fclose(fp);
    }
}
```

实测 file_line_reader 性能约为 istream_line_reader 的 10 倍

```
template <typename C>
vector<string> backsort(C&& lines)
{
    vector<pair<string, string>> pairs;
    for (const string& line : lines) {
        auto split_line = action::split(line, '.');
        pairs.emplace_back(make_pair(
            to_<string>(split_line |
                        view::reverse |
                        view::join('.')),
            line));
    }
    ranges::sort(pairs);
    return pairs | view::transform([](const auto& pr)
                   {
                       return pr.second;
                   });
}
```